INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema de Suporte à Decisão Espacial Baseado na Web para Instalação de Equipamentos Públicos Considerando Acessibilidade Urbana e Indicadores Socioec

VICENTE LUCAS ESCÓRCIO DE OLIVEIRA
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Web-based Spatial Decision Support System for Public Equipment Installation Considering Urban Accessibility and Socioeconomic Indicators

Vicente Lucas Escórcio de Oliveira

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Computer Systems

Supervisor: Dr. Paulo Oliveira

Evaluation Committee:
President:
Dr. José Marinho, Professor, DEI/ISEP

Members:
Dra. Fátima Rodrigues, Professor, DEI/ISEP
Dra. Isabel Azevedo, Professor, DEI/ISEP
Dr. Paulo Oliveira, Professor, DEI/ISEP

Porto, September 14, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, September 14, 2024

Abstract

The rapid expansion of urban areas presents significant challenges for sustainable infrastructure development and maintenance. Addressing these challenges requires intelligent **urban planning** solutions that ensure equitable access to public facilities. This thesis proposes a **web-based spatial decision support system (WebSDSS)** designed to enhance **decision-making** processes regarding the installation of public equipment. The system integrates spatial, **socioeconomic**, and **demographic** data, providing a visualization interface that overlays this information on interactive maps. The system allows users to create criteria based on sociodemographic parameters, with the value of each criterion calculated using an **accessibility model** of accumulative opportunities. A **multi-criteria decision analysis (MCDA)** method, such as TOPSIS, is applied to select the best location based on these criteria. This approach aims to facilitate the equitable distribution of urban opportunities by considering both **accessibility** and demographic indicators. The **WebSDSS** supports decision-makers in locating public facilities, thereby addressing disparities in the distribution of urban opportunities. The system also presents itself as a tool for development **smart cities**, providing a data-driven approach to urban planning. This study evaluates the system through software testing, usability assessment using the System Usability Scale (SUS), and a Quantitative Evaluation Framework (QEF). The results demonstrate that while the QEF shows positive compliance with the functional requirements, certain non-functional aspects, such as scalability and real-world deployment, remain to be improved. The SUS evaluation provided an excellent score of 80.625, affirming a high degree of user satisfaction and effective interaction. Despite these successes, limitations in scalability, supportability, and full validation of the research hypotheses highlight areas for future work. Additionally, the influence of political and social factors in real-world decision-making scenarios complicates the straightforward application of purely **data-driven** solutions. However, the **WebSDSS** effectively addresses its core goal of aiding urban planners in making informed, accessible, and equitable infrastructure decisions. Future improvements will focus on system scalability, spatial database deployment, and broader real-world testing.

Keywords: Urban Planning, Decision-Making, WebSDSS, Accessibility Model, Socioeconomic data, Demographic data, MCDA, Smart Cities, Data-Driven

Resumo

A rápida expansão das áreas urbanas apresenta desafios significativos para o desenvolvimento e manutenção sustentável da infraestrutura. Abordar esses desafios requer soluções inteligentes de **planejamento urbano** que garantam o acesso equitativo às instalações públicas. Esta tese propõe um **sistema de suporte à decisão espacial baseado na web (WebSDSS)** projetado para aprimorar os processos de **tomada de decisão** relacionados à instalação de equipamentos públicos. O sistema integra dados espaciais, **socioeconômicos** e **demográficos**, fornecendo uma interface de visualização que sobrepõe essas informações em mapas interativos. O sistema permite que os usuários criem critérios baseados em parâmetros sociodemográficos, com o valor de cada critério calculado usando um **modelo de acessibilidade** de oportunidades acumulativas. Um método de **análise de decisão multicritério (MCDA)**, como o TOPSIS, é aplicado para selecionar o melhor local com base nesses critérios. Essa abordagem visa facilitar a distribuição equitativa de oportunidades urbanas, considerando tanto a **acessibilidade** quanto os indicadores demográficos. O **WebSDSS** apoia os tomadores de decisão na localização de instalações públicas, abordando disparidades na distribuição de oportunidades urbanas. O sistema também se apresenta como uma ferramenta para o desenvolvimento de **cidades inteligentes**, proporcionando uma abordagem orientada por dados ao planejamento urbano. Este estudo avalia o sistema por meio de testes de software, avaliação de usabilidade usando a Escala de Usabilidade do Sistema (SUS) e um **Quadro de Avaliação Quantitativa (QEF)**. Os resultados demonstram que, embora o QEF mostre conformidade positiva com os requisitos funcionais, certos aspectos não funcionais, como escalabilidade e implantação no mundo real, ainda precisam ser melhorados. A avaliação do SUS obteve uma excelente pontuação de 80.625, confirmando um alto grau de satisfação do usuário e interação eficaz. Apesar desses sucessos, limitações em escalabilidade, suporte e validação completa das hipóteses de pesquisa destacam áreas para trabalhos futuros. Além disso, a influência de fatores políticos e sociais em cenários de tomada de decisão no mundo real complica a aplicação direta de soluções puramente **baseadas em dados**. No entanto, o **WebSDSS** aborda de maneira eficaz seu objetivo principal de auxiliar os planejadores urbanos na tomada de decisões informadas, acessíveis e equitativas sobre infraestrutura. Melhorias futuras focarão na escalabilidade do sistema, na implementação de um banco de dados espacial e em testes mais amplos no mundo real.

Acknowledgement

First of all, I would like to thank my dear wife Nicole, who said yes when I asked her to marry me. But also because you support me unconditionally and inspire me to be a better human being every day. To my parents and my siblings, who have been with me since my birth and shaped who I am today. To my friends, who have always supported me and helped me overcome challenges. To all the teachers who contributed to my academic education, from preschool to university: you are the engine of the world, and no technology can replace the knowledge you share!

And to everyone who, in some way, contributed to my reaching this point. Thank you very much.

I conclude my acknowledgments with a phrase that guides my life: 'With great power comes great responsibility.' - Uncle Ben

Contents

List of Figures

List of Tables

List of Source Code

List of Abbreviations

AHP	**A**nalytic **H**ierarchy **P**rocess
API	**A**pplication **P**rogramming **I**nterface
AWS	**A**mazon **W**eb **S**ervices
CI	**C**ontinuous **I**ntegration
CD	**C**ontinuous **D**eployment
CDE	**C**ontinuous **D**elivery
DDM	**D**ata **D**ialog **M**odel
DSS	**D**ecision **S**upport **S**ystem
ECS	**E**lastic **C**ontainer **S**ervice
ELB	**E**lastic **L**oad **B**alancer
ETUFOR	**U**rban **T**ransportation **C**ompany of **F**ortaleza
FAST	**F**unction **A**nalysis **S**ystem **T**echnique
GIS	**G**eographic **I**nformation **S**ystem
GTFS	**G**eneral **T**ransit **F**eed **S**pecification
IBGE	**B**razilian **I**nstitute of **G**eography and **S**tatistics
IS	**I**nformation **S**ystem
MCDMA	**M**ulti-**C**riteria **D**ecision-**M**aking **A**nalysis
MCDM	**M**ulti-**C**riteria **D**ecision-**M**aking
MC-WebSDSS	**M**ulti-**C**riteria **W**eb-based **S**patial **D**ecision **S**upport **S**ystem
MVC	**M**odel-**V**iew-**C**ontroller
MVT	**M**odel-**V**iew-**T**emplate
NCGIA	**N**ational **C**enter for **G**eographic **I**nformation and **A**nalysis
OSM	**O**pen**S**treet**M**ap
QEF	**Q**uantitative **E**valuation **F**ramework
QFD	**Q**uality **F**unction **D**eployment
RDS	**R**elational **D**atabase **S**ervice
SDSS	**S**patial **D**ecision **S**upport **S**ystem
SEINFRA	**S**tate **I**nfrastructure **S**ecretariat
SPA	**S**ingle **P**age **A**pplication
SUS	**S**ystem **U**sability **S**cale
TBD	**T**runk **B**ased **D**evelopment
TDD	**T**est-**D**riven **D**evelopment
UML	**U**nified **M**odeling **L**anguage
UN-Habitat	**T**he **U**nited **N**ations **H**uman **S**ettlements **P**rogramme
WebSDSS	**W**eb-based **S**patial **D**ecision **S**upport **S**ystems
WKB	**W**ell-**K**nown **B**inary
WKT	**W**ell-**K**nown **T**ext

Chapter 1

Introduction

1.1 Context

Urban areas are experiencing rapid expansion. In 2020, they already accounted for 55% of the world's population, and it is predicted that they will reach a notable mark of 68% by 2050 (UN-Habitat 2020). This growth, often disordered, poses challenges in sustainability and maintenance of infrastructure services. Sectors related to urbanization, such as energy supply, transportation, housing, sewage, etc., require intelligent solutions in planning and urban design.

The United Nations Human Settlements Programme (UN-Habitat) summarizes urban planning as a decision-making process aimed at achieving economic, social, cultural, and environmental goals through the development of visions, strategies, and territorial plans (U. N. H. S. P. UN-Habitat 2015). It is essential for decision-making related to the location of urban infrastructure, which can be defined as a group of either social (health, education, security, leisure, etc.), economic or institutional administrative functions (Zmitrowicz and Neto 1997), to focus on achieving accessibility levels that allow the population to make use of it.

There is a misconception about what it means to have a city capable of providing an infrastructure of services that truly work. Having a physical infrastructure (e.g., hospitals, schools, etc.) spread throughout the city doesn't guarantee that the population has access to it, even if transit networks are efficient. At this point, policymakers and urban planners often focus on the wrong problems, addressing issues related to mobility rather than accessibility. It is more than just reducing travel time; it is necessary to consider what the population can reach within a given time and the associated costs.

Urban planners also struggle to incorporate social, demographic, and economic factors that affect accessibility. For instance, researchers found that the low-income population in Vancouver, despite being within the catchment area of health facilities, has less access to those facilities compared to their higher-income counterparts (Mayaud, Tran, and Nuttall 2019). This problem is exacerbated by the fact that transportation costs are proportionally higher for vulnerable households that are excluded, compared to the rest of the population. Another good example illustrating the relationship between spatial and non-spatial data comes from Fortaleza, Brazil. A research (Pereira et al. 2021) found that regions outside the expanded center of the city have access to less than 10 of job opportunities by bike. This is problematic because in Brazil, the low-income population depends heavily on active transportation to reach their jobs, and this population is concentrated in the peripheries of the city.

The decision-making process applied to the development of cities typically requires the modeling of spatial and non-spatial data, mainly in terms of accessibility measures. It considers different stakeholders, such as public transportation, distribution of facilities, and social aspects of the

population. Therefore, the development of platforms that assist in the decision-making process and deal with spatial data is necessary, whether in modeling or visualizing data. Spatial decision support systems can meet these requirements.

1.2 Problem Description

In terms of accessibility, various activities and locations—like public transportation, job agency services, or hospital clinical care—are viewed as opportunities. However, these opportunities often concentrate and circulate among those social profiles that already have access, resulting in an inequitable distribution (Rolnik 2023).

Given these circumstances, the need for a better distribution of opportunities that meets the demands of different sectors of society becomes possible through an infrastructure that takes into account urban accessibility and macroeconomic and demographic indicators of the population in the vicinity. From this problem, public managers have to decide where to locate these facilities, always in a place that can meet not only political but also social demands.

Public sector decision-makers operate within a political context that doesn't necessarily favor decisions aligned with the truth, as Lamm noted (cited in Henderson and Schilling 1990). Instead, decisions often thrive due to a mix of other political and relational factors. It becomes clear that the decision-making process is not only complex but also influenced by a variety of factors, including the political context, the decision-maker's experience, and the data available.

The lack of decision support systems, which are based on modeled population and spatial data and tailored for this type of decision-making, makes the task more challenging. Furthermore, an environment that is easy to learn, share, and access is necessary for users to understand the value delivered by the system, which can drive decisions more aligned with the data. Additionally, the ability to visualize urban data on interactive maps can help increase system acceptance.

1.3 Objectives

The objective of this project is to create a Web-based multi-criteria spatial decision support system that addresses the challenges of equitable distribution of public services and infrastructure. The system will integrate spatial, socioeconomic, and demographic data to provide a comprehensive understanding of the supply and demand of opportunities such as healthcare facilities, educational institutions, and public transportation. It will enable planners to evaluate the situation at a small-scale spatial level and make informed infrastructure decisions, reducing the workload necessary to evaluate the impact of new facilities and increasing the quality and transparency of the analysis.

The system will also include a visualization interface that overlays the data on interactive maps, allowing users to control different accessibility components and measure potential accessibility to opportunities in specific areas. The socioeconomic aspects of the surrounding region will be taken into account to ensure a more intelligent distribution of opportunities throughout the city.

From a more scientific perspective, the work's objectives are expressed through the formulation of research questions, originating from a main question "How can a Data-Driven Decision Support System use accessibility and socioeconomic indicators to improve public facilities usage?" in a systematic mapping (Kitchenham and Charters 2007):

- **RQ1:** How do accessibility and socioeconomic indicators affect the public facilities distribution and urban infrastructure?
- **RQ2:** What impact can a Decision Support Systems have on urban planning?

- **RQ3:** What are the uses of a Decision Support Systems in urban planning?

1.4 Contributions

One of the expected contributions is the improvement of the urban planning process through the use of the proposed web-based Spatial Decision Support System (SDSS). It is expected that the distribution of opportunities throughout the city will be done in a more intelligent manner, taking into account the socioeconomic aspects of the population to provide better accessibility. Additionally, the project will produce the following contributions:

- A state-of-the-art report containing a summary of the concepts inherent to spatial decision support systems and accessibility measures, and how these concepts are related to urban planning.
- Identification and definition of functional and non-functional requirements, as well as the design of a web-based SDSS architecture.
- Delivery of a WebSDSS capable of helping public sector decision maker to choose public facilities location.
- Application of this system in a real scenario, modeling and analyzing data extracted from selected sources.

1.5 Ethical Considerations

The accessibility model behind the Web-based Spatial Decision Support Systems (WebSDSS) proposed will utilize socioeconomic and demographic data, geolocation data, and transit feed data (e.g., bus arrival and departure times).

The demographic and socioeconomic data will be obtained from the Brazilian census conducted by the Brazilian Institute of Geography and Statistics (IBGE). These data are collected and manipulated following privacy, fairness, protection, and integrity policies and are made available to the public on the institution's platform, as stipulated by Law nº 5.534. Therefore, all the moral considerations regarding the use of these data have already been addressed by the institution itself.

Regarding geolocation data and transit feed, the sources will be, respectively, the open-source geo-referenced map data from the collaborative project OpenStreetMap and the company Google. Since these data are not sensitive or personal, they do not imply any moral considerations in this work.

1.6 Methodology

To ensure an understanding of what is effectively being done in this project, it is important to present the methodology employed. Naturally, there is an implicit methodology that encompasses the execution of all this work, including documentation and the built system. Throughout each stage, specific methodologies are used to assist in the production of artifacts, as they meet the specific objectives of each of these stages.

More concretely, the principles of the Agile Manifesto (Beck et al. 2001) are being used, which involve software development. These principles mainly consist of iterative development of the software, focused on continuous delivery and flexibility in changing requirements. Agile processes also prioritize communication with the customer, high responsiveness, and the generation of value at each interaction (Project Management Institute 2017). This model fits into the development of

web systems such as that proposed by this work, which tends to undergo constant changes, as it is essential to meet requirements aimed at better interaction with the decision-maker who uses it.

The stages described below illustrate the methodology used in the construction of the proposed system, producing the documentation that will be reflected in this dissertation.

1.6.1 Analysis

The initial stage of the project involves contextualizing the problem through a study of the state of the art, which includes reviewing main concepts, technologies, and related works. The studies were selected and classified using an adapted systematic mapping approach (Kitchenham and Charters 2007).

In a second stage, an interview with a public sector decision-maker will take place to understand the clients' needs regarding decision-making processes involving public facilities location and support systems usability. The interview will be followed by the development and analysis of software requirements.

Finally, a study and selection of spatial and socioeconomic data sources, as well as the selection of the accessibility model that will be used in the proposed support decision system, will occur.

1.6.2 Design

Once the system requirements have been identified and the entire concept behind the system is understood, the design phase can be initiated. The focus is on developing an architecture that meets the requirements, enabling the solution to the presented problem.

1.6.3 Implementation

With the defined architecture, the next step is to implement the application using the technologies analyzed in the first stage, taking into account the established architecture. This phase can be preceded by the creation of a prototype.

Implementation also involves choosing a scenario for the software application. In the context of this work, due to the dependence on the availability of data sources that were analyzed in the Analysis stage, the implementation scenario must already be chosen.

1.6.4 Evaluation and Testing

The penultimate stage involves conducting tests and evaluations that seek to assess compliance with the established requirements and whether the application is stable and, above all, presents a solution to the problem. Among the tests are unit, integration, and end-to-end. The evaluation part is done with end-users.

1.6.5 Deployment

Once the system is validated, it is made available on a web server so that it can be accessed via tools such as a web browser.

In addition to the system, the documentation for each stage will be published on the appropriate academic platform in the form of a dissertation.

1.6.6 Schedule of Activities Planning

It is essential to emphasize that the agile methodology is based on an iterative development process, which means that each stage described will be revisited multiple times, and at the end of each interaction, there will be value generation. The state of the art and data analysis will be the only steps that will not be iterative.

The tasks are separated between sprints and tasks. The sprint represents the iterative part of the development, covering mostly the web platform development. Each sprint contains 3 weeks of work passing through Analysis, Design, Implementation, Evaluation, and Deployment. At the end of each sprint, a release of the web platform that fulfills part of the requirements but comprises a usable system will be delivered. Normal tasks are related to the documentation production and data sources analysis

Figure 1.1 shows how tasks and sprints are distributed over time, and Figure 1.2 illustrates the due date for each one.

Each activity is described below:

- **Task 1:** This is the main task and it will take place throughout the activities cycle. As the development of the platform advances, the documentation will be updated. Not only will new content be added, but fixes and improvements might be required.
- **Task 2:** This task's goal is to analyze the sources for georeferenced and socioeconomic and demographic data.
- **Task 3:** This task is dedicated to choosing the model of accessibility measure that will be implemented by the system.
- **Task 4:** In order to develop the requirements, an interview with public sector decision makers is planned to be conducted.
- **Sprint 1:** This sprint comprises the development of the requirements for the system and the start of the first design for the architecture based on the developed requirements.
- **Sprint 2:** The second sprint will focus on developing a first version of the web platform, focusing on basic structural functionalities such as the interaction between client and server and simple data rendering.
- **Sprint 3:** The goal of this sprint is to create data visualization with interactive maps, besides creating forms to get user input.
- **Sprint 4:** In this sprint, the idea is to implement the accessibility measure model algorithm on the server side and present the results to the user.
- **Sprint 5:** The objective of this sprint is to implement a multi-criteria analysis algorithm.
- **Sprint 6:** This sprint is dedicated to conducting the real case scenario testing and taking note of the results.

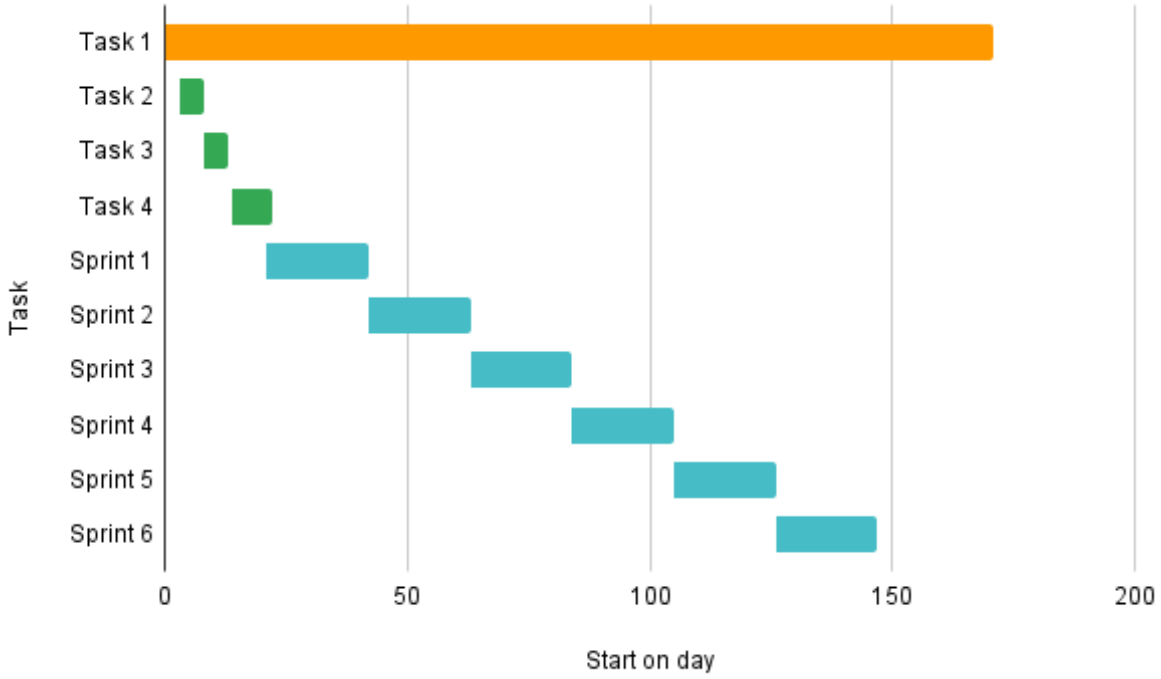

Figure 1.1: Distribution of Activities

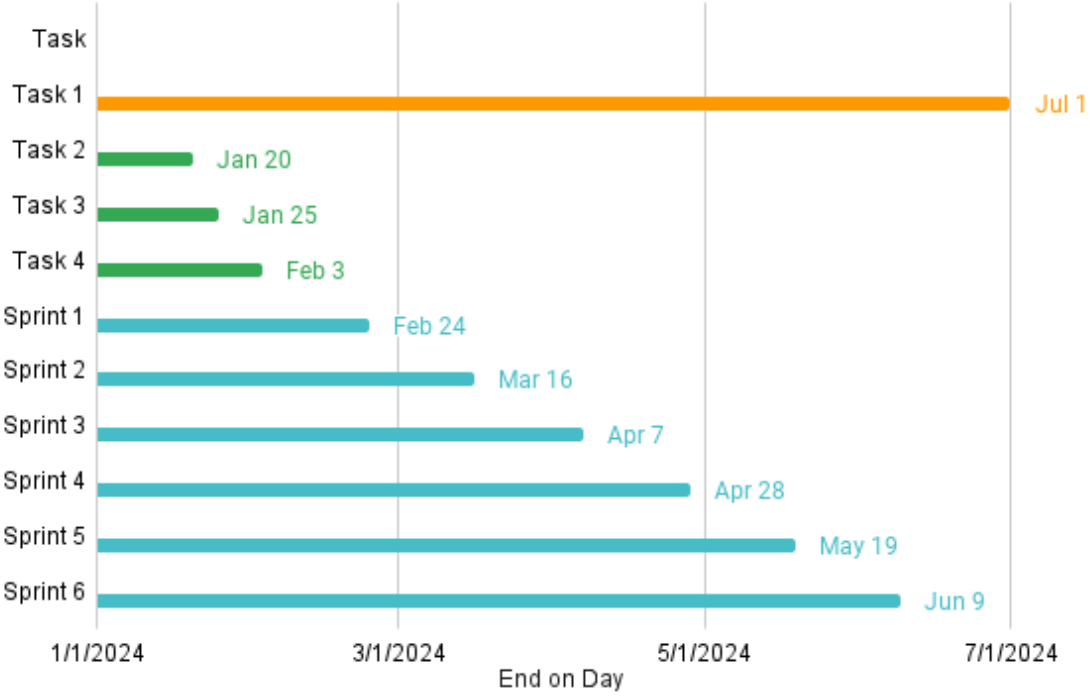

Figure 1.2: Schedule of Activities

1.7 Structure

This dissertation is described in the following chapters:

1. **Introduction:** This chapter introduces the problematic and provides an overview of the methodology used in the study.
2. **State of the Art:** This chapter addresses the explanation of the methodology used to select the references and the key concepts for the development of the project, focusing on explaining accessibility measures in the context of urban planning and how DSS involves into WebSDSS, reviewing their definitions, architectures, and principles. Besides, it presents technologies and current implementations that make use of the presented concepts.
3. **Analysis:** This chapter dives into the analysis of the problem and sets use case scenario for the proposed solution, including the model domain and requirements for the solution.
4. **Value Analysis:** This chapter presents the value analysis of the solution, which includes the identification of the main stakeholders, the value proposition, and the value stream.
5. **Design:** This chapter illustrates the artifacts developed, taking into account the solution selected in the previous chapter.
6. **Implementation:** This chapter is dedicated to the technical details of the implementation applied to the project.
7. **Evaluation of the Solution:** This chapter presents the test methodologies used for the solution. Also, the tests and their behavior against the same are detailed.
8. **Conclusion:** This chapter exposes the conclusions obtained from the project as a whole. The concretized objectives and possible improvements for future work are also emphasized.
9. **References:** Lists all references used.

Chapter 2

State of Art

The goal of this chapter is to present the research process and the main concepts underlying the context of this dissertation, highlighting the information valuable to it. Furthermore, it covers related works, comparing and analyzing their aspects, and positioning the current work among them. Finally, a description of some existing technologies that can be used as tools to apply the presented concepts and help achieve the objectives of this dissertation.

2.1 Research Process

An adapted systematic mapping study (see Figure 2.1) was undertaken for this project, with a focus on comprehending the general aspects of the topics introduced by the research questions.

A Systematic Mapping is a comprehensive review of existing primary studies in a specific research topic, aiming to identify the available evidence on that topic. It is a secondary study that seeks to identify and classify research related to a broad research topic (Kitchenham and Charters 2007).

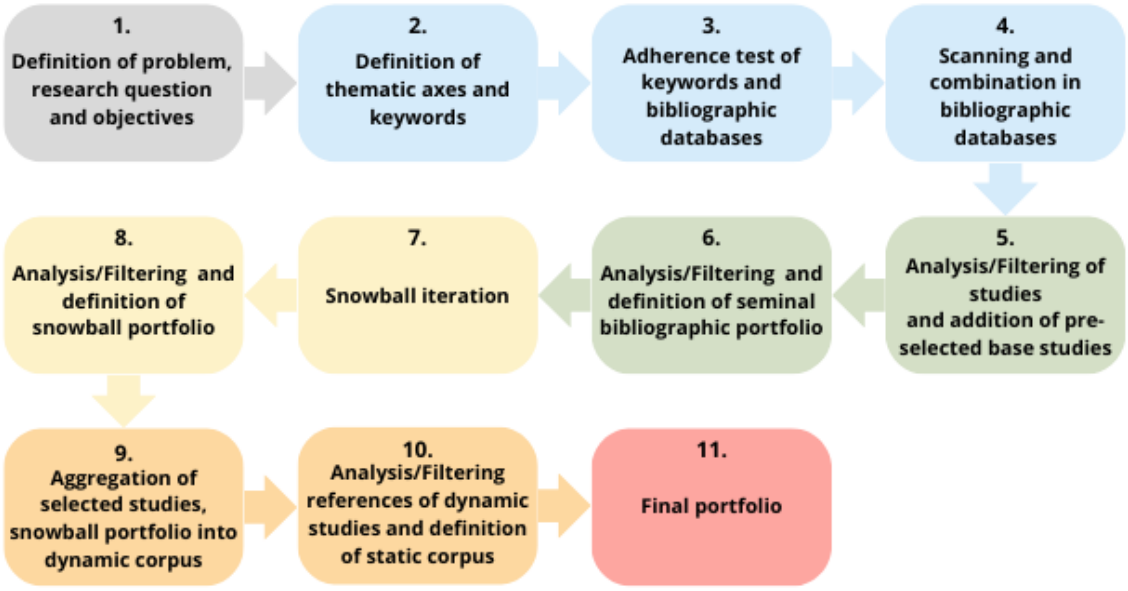

Figure 2.1: Adapted systematic mapping process workflow

2.1.1 Data Sources

As seen in Table 2.1, a small number of data sources were chosen for this systematic mapping, and the main goal is to work with sources that predominantly have publications in the same areas of study presented by the questions.

Search engine syntax might vary between the data sources, but as long as the logical relationship of the search terms remains the same, the search won't be affected.

Table 2.1: Data sources

ID	Name	URL
DS1	ACM digital Library	https://dl.acm.org/
DS2	IEEE Explorer	https://ieeexplore.ieee.org/Xplore
DS3	B-on (Biblioteca de Conhecimento Online)	https://www.b-on.pt/

2.1.2 Research Questions

This work, as a systematic mapping, aims to address the main question: "How can a Decision Support System use accessibility and socioeconomic indicators to improve public facilities usage?" With the objective of formulating an adequate answer, which covers all the different domains involving the question, three questions were derived from the main one:

- **RQ1**: How do accessibility and socioeconomic indicators affect the public facilities distribution and urban infrastructure?
- **RQ2**: What impact can a Decision Support Systems have on urban planning?
- **RQ3**: What are the uses of a Decision Support Systems in urban planning?

2.1.3 Search Terms

Based on the questions, search terms were chosen to reflect the thematic axis(see Table 2.2).

Table 2.2: Thematic Axis divided by domain and keywords

Domain	Keywords
Decision Support Systems	"Data analysis tools for decision making" OR "Decision support systems" OR "spatial decision support systems" OR "geographic information systems"
Urban Accessibility	"Accessibility" OR "Urban Accessibility"
Urban Planning and public facilities	"Urban infrastructure" OR "Public services" OR "Facilities" OR "Urban Planning"
Socioeconomic and demographic indicators	"Socioeconomic index" OR "Socioeconomic data" OR "Population data" OR "Demographic data"

The final query and the results obtained from the data sources are described in Table 2.3:

2.1.4 Inclusion and Exclusion Criteria

The inclusion and exclusion criteria are listed below in Tables 2.4 and 2.5:

Table 2.3: Final query and the results obtained per data source

Query	Results			
	DS1	DS2	DS3	Total
("Accessibility" OR "Urban Accessibility") AND ("Data analysis tools for decision making" OR "Decision support systems" OR "spatial decision support systems" OR "geographic information systems") AND ("Urban infrastructure" OR "Public services" OR "facilities" OR "Urban Planning") AND ("socioeconomic index" OR "socioeconomic data" OR "Population data" OR "Demographic data")	55	4	12	71

Table 2.4: Inclusion Criteria

ID	Criteria
IC1	Publications in English or Portuguese.
IC2	Strong adherence to the theme of urban accessibility.
IC3	Documents that cover, at least, the urban accessibility thematic and one of the others.
IC4	Studies on decision-making support systems.

2.1.5 Results

In the adapted systematic mapping process employed in this work (see Figure 2.1), a study collection method called 'Snowballing' was incorporated. This method involves forward and backward interactions, pulling citations and references from articles, and applying inclusion and exclusion criteria to select them.

For all analyses and filtering, all inclusion criteria were considered, and exclusion criteria EC1, EC2, and EC3 were applied. EC4 was taken into consideration when filtering the studies to define the seminal portfolio, and EC5 was used to define the static corpus.

To enhance understanding, the filtering process that uses all inclusion criteria and exclusion criteria EC1, EC2, and EC3 will be referred to as "base filtering."

The initial phase of the process involved formulating search terms from the research questions and initiating the search in the selected bibliographic databases (stages 1, 2, 3, and 4).

After searching for studies in the databases, a base filtering process is carried out. These articles are then combined with other pre-selected studies from parallel studies or academic recommendations that are considered foundational texts for the subject of the work. From these studies, a seminal portfolio is selected by applying EC4, so it can be used as the input set for snowballing (stages 5 and 6).

Snowballing was applied using the SnowGlobe tool (*SnowGlobe Project* 2023), which, from a list of input articles, performs the interactions and exports a new list with all the results. These results then undergo the base filtering again, resulting in a snowball portfolio (stages 6, 7, and 8).

Next, the studies selected for the snowball portfolio are combined with the selected studies from stage 5, resulting in a dynamic corpus. From this corpus, all the references undergo the base filtering and the exclusion criteria EC5 to find studies considered foundational, which then become part of the static corpus (stages 9 and 10).

Table 2.5: Exclusion Criteria

ID	Criteria
EC1	Works that do not consider mobility or land use components for the accessibility analysis.
EC2	Documents that don't cover the use of data science tools or theory.
EC3	Works that do not use or only partially use a GIS tool.
EC4	Works with less than 10 citations.
EC5	Studies referenced in at least 10% of the other selected studies.

In the end, the sum of seminal, dynamic, and static corpora in each stage makes up the final portfolio. Figure 2.2 shows the number of studies obtained at the end of each section.

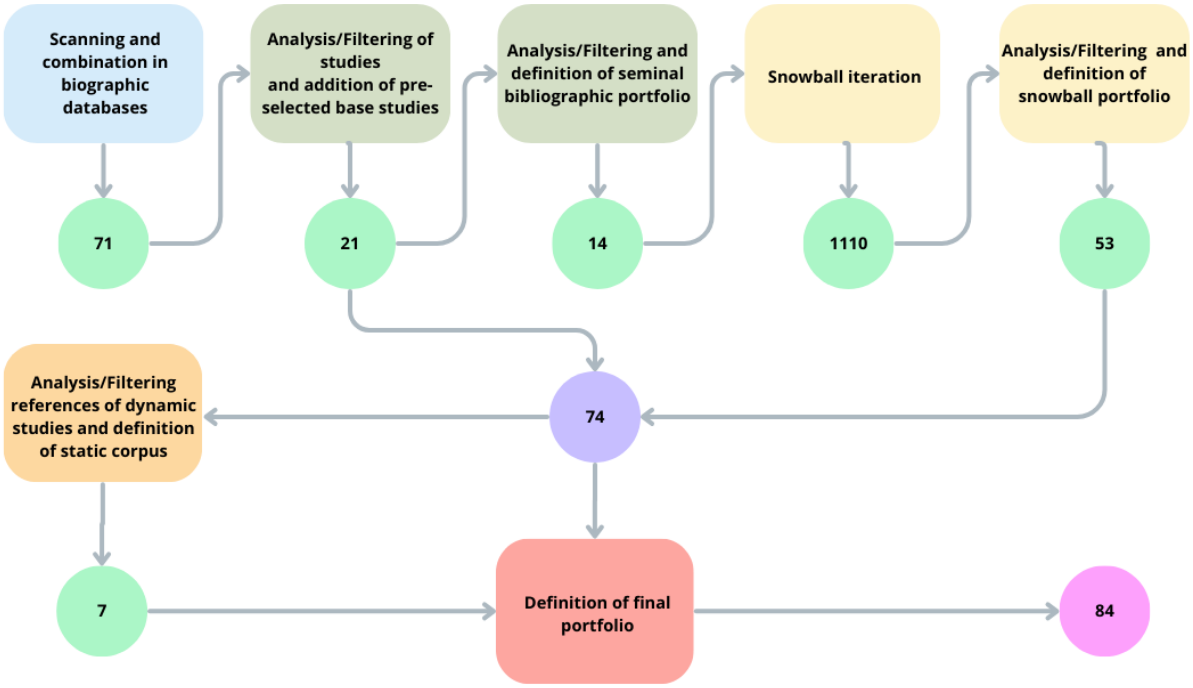

Figure 2.2: Systematic Mapping process results

2.1.6 Studies Mapping

The studies were categorized based on three facets: research type, accessibility model, and target opportunity.

The research type reflects the specific goal of the study, whether it aimed to conduct a use case analysis, an evaluation, propose a solution, build a platform, or provide a literature review on the current subject. The category is described below:

- Literature Review (R1)
- Study Evaluation (R2)
- Solution Proposal (R3)
- System Development (R4)
- Validation Research (R5)

The accessibility model serves as a categorization to comprehend which components of the accessibility model were employed for measurement. These models include land-use, transportation, individual, temporal, or others. 'Others' signifies that the study did not directly utilize accessibility models but instead employed practical analyses through GIS tools or statistical models. The values described below:

- Land-use (C1)
- Transport (C3)
- Temporal (C4)
- Individual (C5)
- Other (C6) - multi criteria, statistical, and interview, or using GIS features.

The third facet is the target opportunity, describing the specific opportunity under study. For instance, it could be related to public transportation, healthcare, employment, leisure, or other areas. The category is described below:

- HealthCare (O1)
- Employee (O2)
- Leisure (O3)
- Transport (O4)
- Others (O5) - Does not fit into only one category or is discussed in a general manner

Figure 2.3 shows how the studies are distributed in each category.

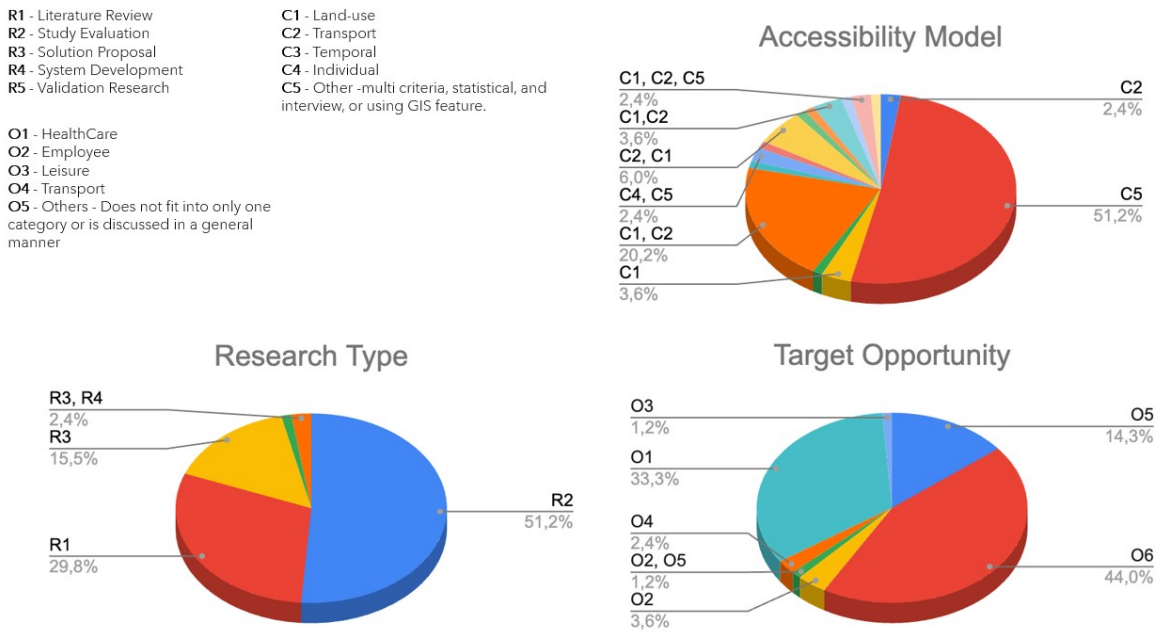

Figure 2.3: Studies mapping by category

2.2 Urban Planning and Accessibility Measures

Gaining an understanding of how accessibility measures relate to urban planning is paramount for this work. These indicators exhibit sensitivity to various aspects of urban infrastructure, including the transportation network, distribution of services, and the structure of public transportation. Clarifying these connections is essential for a comprehensive analysis of the subject matter.

2.2.1 Urban Planning

UN-Habitat defines urban planning as a decision-making process aimed at achieving social, economic, cultural, and environmental goals through strategies and territorial plans (U. N. H. S. P. UN-Habitat 2015). Souza 2003 emphasizes the need to separate management from planning, defining it as an attempt to simulate processes to anticipate problems and maximize benefits.

Housing production and urban planning reflect the growth of large centers, respectively based on architecture and urbanism. Urban design lies between the two, being an intersection and a subset of the disciplines. It concerns the functionality of cities and the use of public space (Barnett and Jones 1982). Cowan 2005 attributes to urban design the responsibility of formulating physical scenarios for urban living in municipalities and villages, as well as involving the conception of spaces, landscapes, constructions, and processes that provide success in the development of these regions.

As cities grow, different aspects of their design change, whether infrastructural, land-use, socioeconomic, etc. The need for measures that encompass all of these domains increases. The perspective of urban planning as a decision-making process presents the challenge to decision-makers to consider all aspects during the evaluation.

Understanding the formation of urban space is essential for better management. Zmitrowicz 2002 highlights the importance of urban infrastructure as a technical system responsible for providing services to society and needing to be operated and related to the population. Public managers must comprehend, among several other things, the aspects influencing accessibility to these services and how to measure it. To this end, various accessibility measurement approaches have been developed.

2.2.2 Accessibility

Accessibility is often defined by the ease of reaching destinations or activities, known as opportunities, but authors add more constraints to this definition to avoid misconceptions. K. T. Geurs and Ritsema van Eck 2001 highlights the role of land-use and transport systems in enabling individuals and goods to reach opportunities. On the other hand, Bhat (2000, as cited in C. Curtis and Scheurer 2010) narrows the definition by specifying accessibility as the ease of pursuing a desired activity given type, location, mode, and interval of time. Furthermore, Bertolini, LeClercq, and Kapoen (2005, as cited in C. Curtis and Scheurer 2010) define accessibility as "the amount and diversity of places that can be reached within a given travel time and/or cost." This definition summarizes accessibility didactically when it implicitly considers the urban systems role but at the same time addresses explicitly the constraints.

By definition, accessibility encompasses various factors that influence the ability to reach desired goods, services, activities, and destinations. These factors belong to wider systems working under different measures and criteria. For instance, transport system performance is significantly impacted by mobility and traffic planning, but these plans primarily focus on individuals and goods and vehicles displacements (C. Curtis and Scheurer 2010). However, accessibility goes beyond traffic and mobility concerns, as it also considers land purpose, management, and human interactions to conduct a more in-depth analysis (Litman 2003, as cited in C. Curtis and Scheurer 2010). This can be illustrated in a scenario where roads are designed to facilitate traffic flow and provide a faster

way to travel, but nearby destinations and activities are not easily accessible by walking or biking. Alternatively, prioritizing land use focused on biking and walking might lead to problems such as congestion and a scarcity of parking spots.

Finally, accessibility is a measure defined by the interplay of many others from different fields. It is comprised by four main components (K. T. Geurs and Ritsema van Eck 2001): transport, temporal, land-use, and individual. While transport and temporal components comprise measures such as the cost of movement in space and the availability of activities in time, land-use and individual consider the spatial distribution of opportunities and how they compete with each other, as well as the influence of socioeconomic and demographic factors. There are also other ways of componentization (Burns 1979; Koenig 1980, as cited in C. Curtis and Scheurer 2010) consisting of two parts: transportation and activity. The first comprises transport and temporal, and the second one individual and land-use of the previously described components.

Therefore, measuring accessibility is a systematic task that should address different evaluations.

K. Geurs and Wee 2004 in Figure 2.4 illustrate how these components and their measures interplay with each other in accessibility measurement.

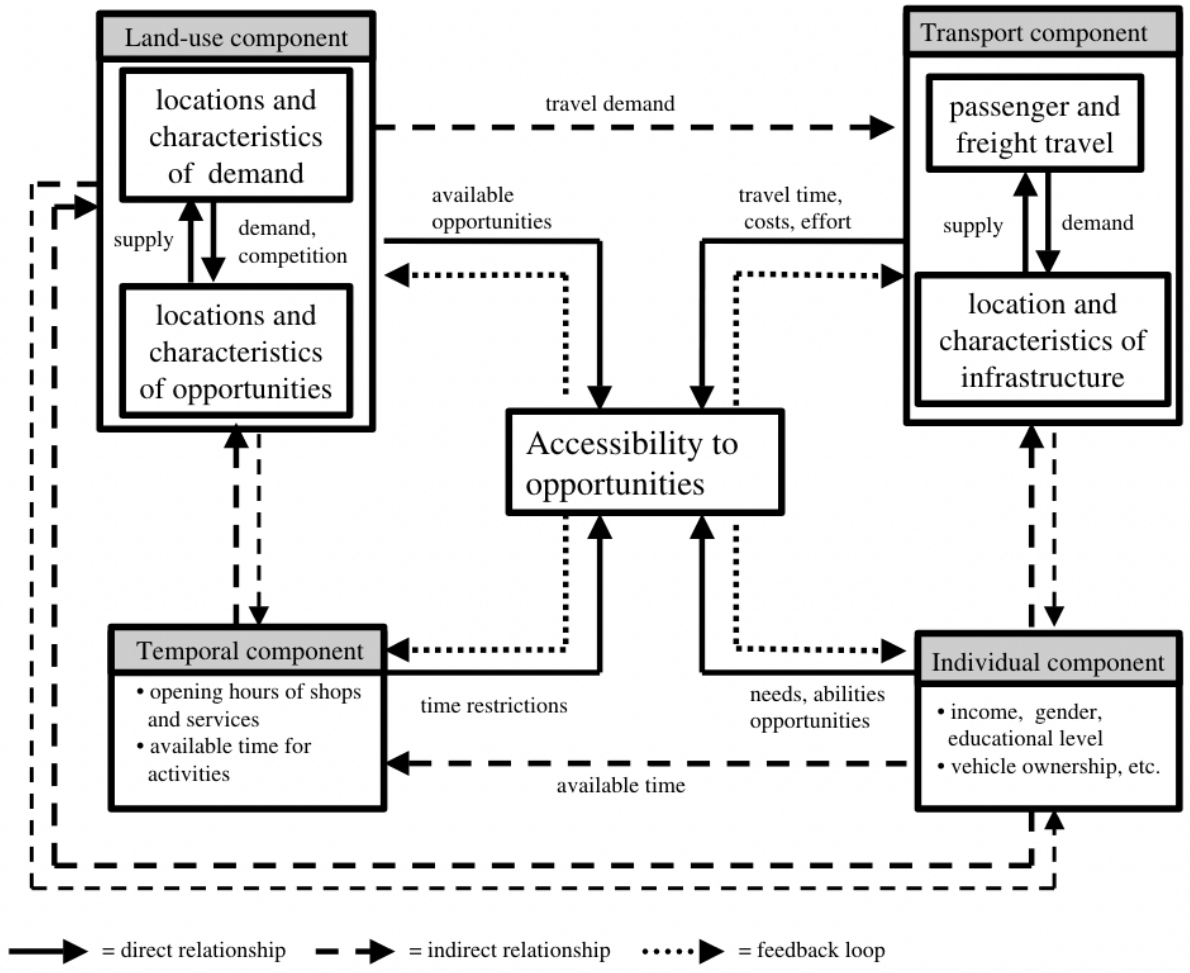

Figure 2.4: Relationships between components of accessibility (K. Geurs and Wee 2004)

2.2.3 Measuring Accessibility

Measuring accessibility is a wide task, once ideally should take into consideration all its components, but in real world, depending on the perspective, the measurement might focus on a sub-group of it (K. Geurs and Wee 2004). These perspectives caused accessibility indicators to be categorized based on them, being described through distinguish approaches by different authors.

For Baradaran and Ramjerdi (2001), there are five types of accessibility indicators approaches: gravity or opportunities, travel cost, constraints-based, utility-based surplus and composite. Similarly, Handy, S L; Niemeier, D A (1997 as cited in K. Geurs and Wee 2004) classified accessibility measures among three types: cumulative opportunities, gravity-based and utility-based. K. Geurs and Wee 2004, on the other hand, categorize then into four basic persperctives: Infrastructure-based, Location-based, Person-based and Utility-based. These categories are described below in terms of Geurs and Van:

- Infrastructure-based measures, which analyze the observed or simulated performance or service level of transport infrastructure, such as the level of congestion and average travel speed on the road network. This measure type is typically used in transport planning.
- Location-based, also known as gravity-based, measures analyze accessibility at specific locations, typically on a macro-level. The measures describe the level of accessibility to spatially distributed activities, such as the number of jobs within a 30-minute travel time from origin locations. More complex location-based measures explicitly incorporate capacity restrictions of supplied activity characteristics to include competition effects. Location-based measures are typically used in urban planning and geographical studies.
- Person-based measures, which analyze accessibility at the individual level, such as the activities in which an individual can participate at a given time. This type measures limitations on an individual‘s freedom of action in the environment, i.e. the location and duration of mandatory activities, the time budgets for flexible activities, and travel speed allowed by the transport system.
- Utility-based measures, which analyze the economic benefits that people derive from access to spatially distributed activities. This type of measure has its origin in economic studies.

Given the different types of accessibility indicators, there are several applications for them, mostly applied to the urban context. For instance, measuring employment access (Hess 2005), future analysis of transit networks (Guthrie, Fan, and Das 2017) (Machado et al. 2018), medical care coverage and health system resources distribution (Cooke et al. 2010) (Liu et al. 2020) and planning (Gil Solá and Vilhelmson 2019) (Carey Curtis 2011).

This dissertation will focus on the relationship between urban planning and accessibility, employing a location-based approach to measure accessibility with a strong emphasis on land-use and individual components.

2.2.4 Use Case: Location-based potential accessibility measure

Potential accessibility measures, also known as gravity-based measures, have been commonly used in urban and geographical studies since the late 1940s. These measures estimate the accessibility of opportunities in a given zone (i) to all other zones (n), where smaller and/or more distant opportunities provide diminishing influences. The development of potential accessibility can be attributed to changes in (K. Geurs and Wee 2004):

- Land use: The amount, characteristics, and spatial distribution of opportunities.

- Transport: The travel impedance between origin and destination locations, such as travel time, costs, etc.
- Travel preferences.

The function based on gravity is explained below, where *Ai* represents the level of accessibility in zone *i* to all opportunities *D* in zone j, *cij* denotes the travel costs between *i* and *j*, and β indicates the cost sensitivity parameter. The choice of cost sensitivity function has a significant impact on the accessibility measurement results. To ensure credible outcomes, the form of the function must be selected carefully, and the function's parameters should be estimated using recent empirical data of spatial travel behaviour in the study area. The potential measure integrates assumptions regarding a person's transport perceptions by adopting a distance decay function as a cost. These measures are appropriate social indicators for evaluating the level of access to economic and social opportunities for various socio-economic groups (K. Geurs and Wee 2004).

$$A_i = \sum_{j=1}^{n} D_j e^{-\beta_{cij}} \tag{2.1}$$

This decay function can be simplifyed to a binary function where it considers only the presence of opportunities in a given area defined by a treshold of travel time. This approach can be define as 'catchment profile analysis', or it could be a 'cumulative-opportunities measure' (Liu et al. 2020). The equation would be as follows:

$$A_i = \sum_{j=1}^{n} D_j F(cij) \tag{2.2}$$

Where F is a binary function that returns 1 if the travel time is less than a treshold and 0 otherwise.

Karst and Eck 2003 in "Evaluation of accessibility impacts of land-use scenarios: the implications of job competition, land-use" analyzes the contribution of land-use and transport changes to the development of job accessibility by car, assuming travel preferences remain constant:

'*Assume a contour around a residential location, enclosing the area that can be reached from that location within, for example, 45 minutes in 1995. The number of jobs in this area in 1995 is the contour measure for 1995. The change in the number of jobs in this area between 1995 and 2020 is the land-use component of the accessibility change. In 2020, the 45-minute contour will have moved, due to changes in congestion levels, new infrastructure, etc. The area between the two contours, for 1995 and 2020, can be visualised as a ring. The number of jobs in this ring in 1995 (positive where the 2020 contour is wider than the 1995 contour, negative where it is narrower) is the travel-time component. The change in the number of jobs in this ring between 1995 and 2020 is the combination component. Note that visualization is more difficult for other types of potential accessibility measures because of the absence of a maximum travel distance or time. The general principle of the three components of accessibility change is, however, the same for all potential accessibility measures.*'

Figure 2.5 has a graphical representation of the opportunities based on variation of each component and the contour around given each year.

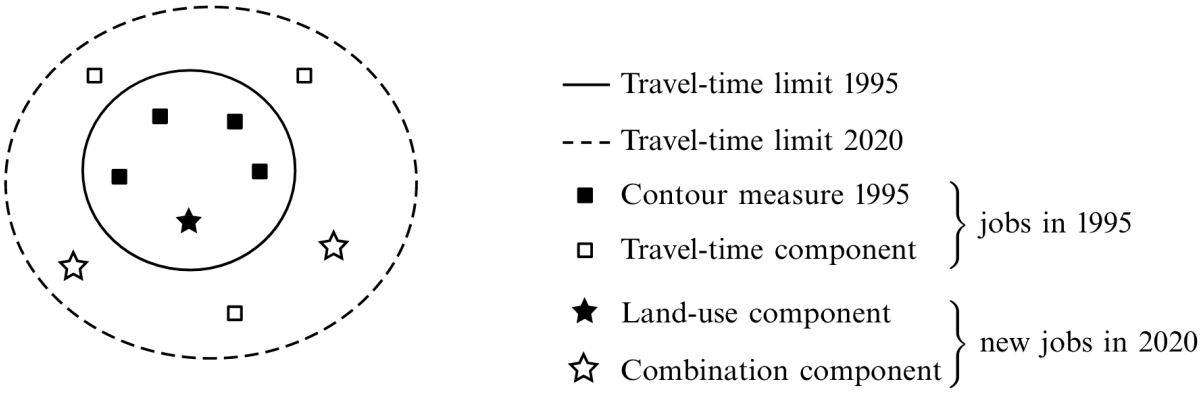

Figure 2.5: Land-use, travel-time, and combination components of accessibility change (Karst and Eck 2003)

Given the simplicity and ease of use of the cumulative-opportunities measure, it is a good candidate for evaluating the accessibility of public facilities in the context of decision-making in urban planning.

2.3 Decision Support Systems

This work proposes to develop a decision support system and, for that, requires an understanding of the definition of these systems and how they are structured. This section provides a review of the conceptualization of these platforms over time and how their architecture has evolved, in addition to understanding their applications. Finally, it describes how these systems have been utilized in decision-making contexts that require the visualization and manipulation of spatial data, highlighting how the growth of geographic information tools facilitated this transformation.

2.3.1 Theorical Concept

The definition of Decision Support System (DSS) dates back more than 40 years and has evolved in line with technological developments and improvements. In the late 1970s, P.G.W Keen and Scott Morton (as cited in Averweg 2012) defined DSS as computerized systems that influence decision-making processes by providing analytical tools and aids, while still requiring the manager's judgment as a crucial component (Averweg 2012, p. 15). However, as stated, this is a concept that constantly changes over the years and may embody different interpretations.

According to Van Schaik (1988, as cited in Averweg 2012), the 1970s marked the emergence of the DSS concept, introducing a new approach to using computers for managerial decision-making. The origin of the notion of what would be DSS could be based on three elements: decision, support, and system. Decision emphasizes the primary focus on decision-making in a problem situation, while support clarifies the computer's role in aiding rather than replacing the decision-maker. Finally, system highlights the integrated nature of the overall approach, suggesting the wider context of machine, user, and decision environment (Averweg 2012)..

In "Decision Support Systems: A Research Perspective," P. G. Keen 1980 reviewed more than 30 articles to understand the different interpretations of what a DSS is. He encountered common concepts of definition such as the type of structured tasks they address, design strategy, support to decision-makers' cognitive process, and implementation strategies to make computers useful through humanized interfaces. Keen emphasized that not all definitions rely on the idea that DSS are computer-based systems but on terms of context and use. Keen revisited this discussion a few years later (P. Keen 1987) and observed the problems of the absence of an established concept of

what DSSs are, once the current concepts were too broad, attracting contributions from different areas. It became a tough task to define what is not a DSS. Even with different concepts, the final understanding is that those systems were made to help "people in organizations do their jobs better.".

As DSS can be defined by their context and use, it is consequential, with the development of computation as a tool within organizations and people's day-to-day, to establish itself as a computer-based system. This can be noticed throughout the years. Bidgoli (1989, as cited in Averweg 2012) stated DSS as "a computer-based information system consisting of hardware/software and the human element designed to assist any decision-maker at any level". Sauter (1997, as cited in Averweg 2012) later understood DSS as computer-based systems that can assist organizations in analyzing and evaluating information from different sources with the use of specific models. Aligned with Bidgoli, Turban (2005, as cited in Averweg 2012) relies on the idea that DSSs are computer-based information systems that, with the use of models and data, attempt to solve structured or unstructured problems, complying also with the first idea brought by Morton and Keen and emphasizes the user involvement in the process.

2.3.2 Structural Concepts

As DSS evolved into a field or study (Sprague and Watsin 1996, as cited in Averweg 2012), several principles also were developed and, at some point, became a definition of a framework for these systems. The most important ones to describe for this work will be covered by this topic.

DDM Paradigm

The Data Dialog Model (DDM) paradigm is a set of three capabilities in the areas of dialog, data, and modeling that are necessary for a good decision support system (Sprague and Carlson 1982):

- Non-technical decision-makers should be able to have easy interaction
- Wide variety of data should be accessible
- Different ways of analysis and modeling should be available

Figure 2.6 shows how the components are related to each other. The data and modeling components are responsible for managing the database and model base, respectively. On the other hand, the dialog is responsible for managing the interface between the user and the system (Averweg 2012).

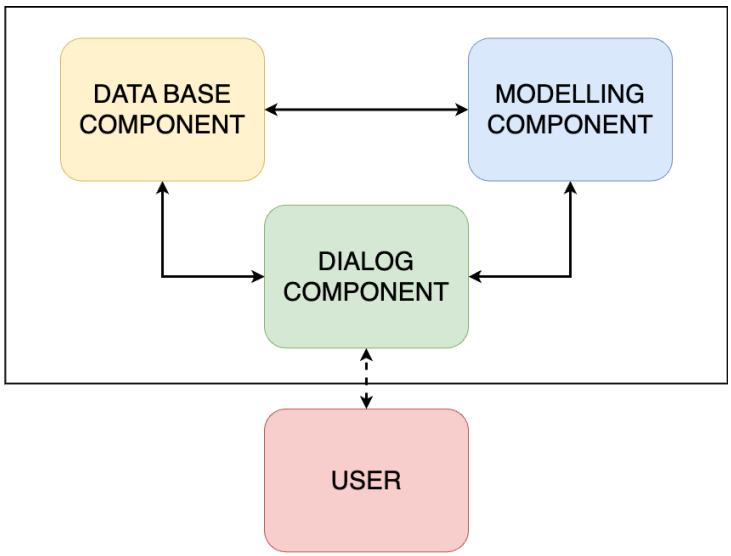

Figure 2.6: The Components of DSS (adapted from Averweg 2012)

Levels of Technology

Three levels of technology are instrumental in developing DSS (Averweg 2012). This concept emphasizes the usefulness of configuring DSS tools into a DSS generator, which can be used to swiftly and easily create various specific DSS to assist decision-makers. The system that actually performs the work is known as the specific DSS, being the first level. It comprises the software/hardware that enables a specific decision-maker to address a set of related problems. The second level of technology is the DSS generator, a package of related hardware and software that provides capabilities to rapidly and easily build a specific DSS. The third level of technology is DSS tools, which facilitate the development of either a DSS generator or a specific DSS.

Iteractive Design and Organizational Environment

DSS require an iterative development that permits quick and easy changes as the problem or decision also changes, with user feedback as a constant (Averweg 2012). Additionally, those systems require an environment where they can thrive and evolve, including an interactive group of people, software and hardware technologies, data sources, and analysis models (Averweg 2012).

2.3.3 DSS to SDSS

Decision-makers can use different applications to identify and solve problems, complete decision process tasks, and make decisions (Dessì, Garau, and Pes 2012). When more sophisticated data analysis is needed, decision-makers define a model for solving the problem by specifying the relevant attributes and behaviors. They then instantiate the related parameters with appropriate data and select a solver for the execution of the model instance. This approach, namely model-driven, emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model and is not necessarily data-intensive. When spatial and non-spatial aspects coexist and/or the spatial information plays a major role in decisions, it is difficult to consider both aspects in a single model. A common solution is to model one aspect at a time and then devise mechanisms for integrating them together (Dessì, Garau, and Pes 2012).

Computation became a powerful tool, allowing DSS applications to evolved at a point to make use of spatial data and interfaces such maps. Although the improvements of DSS was focused on Information Systems (IS), Geographic Information Systems (GIS) has its own community

concentrating on developing geographic data processing applications (Keenan 2003). At some point, DSSs community start to make use of spatial techniques, exploring public available data in fields as location and routing analysis (Keenan and Jankowski 2018). In parallel, GIS community found interesting adding decision tools improving the decision modeling, for example forests and land management (Keenan and Jankowski 2018).

The branching of DSS through its context and use reflects idea of fragmentation. The mixing between spatial data and modeling techniques with tools converged into a new subfield of Spatial Decision Support Systems (SDSS) as a common link between DSSs and GIS applications.

2.3.4 Geographic Information Systems

Geographic Information System (GIS) has its first implementations in the 1960s in North America for the automated production of maps (Wikipedia 2023). It encompasses a technology that not only creates, manages, analyzes, and maps diverse data types but also establishes connections between data and maps. This involves integrating location data seamlessly with various descriptive information (Gimond 2023). Those systems are inserted in a complex environment with high-resolution visualization capabilities, large storage devices, specialized algorithms and specialized query languages that allow GIS users to interact with geo referenced data to extract especially locational subsets (Griffith 2015). Due to this characteristics, the widespread of their implementation was limited by the computational power evolution, since spatial data demands large volume of data compared to other business cases (Keenan 2003).

GIS has been used in various fields of study such as cartography, logistics, urban planning, etc (Sánchez-Lozano et al. 2013) and can address different questions, for instance 'what is present in a particular location?' and 'what map pattern or space-time pattern emerges across a region' (Griffith 2015). It also applies to the network, land-use and accessibility analysis (Malczewski 2004).

Due to the capabilities of GIS to handle and analyze spatial data and provide an interpretive map interface to the users, this systems start do being used to address decision-making problems regarding location in integration with techniques of Multi-Criteria Decision-Making (MCDM) (Erdin and Akbaş 2019).

Two popular GIS software options are ArcGIS (ArcGIS 2023) and QGIS (see Figure 2.7) (QGIS 2023). ArcGIS is a commercial software with a wide range of functionalities and a technical support system. QGIS is an open-source software with an active and strong community, although it may have fewer functionalities than the first option. Both platforms deliver important GIS tasks.

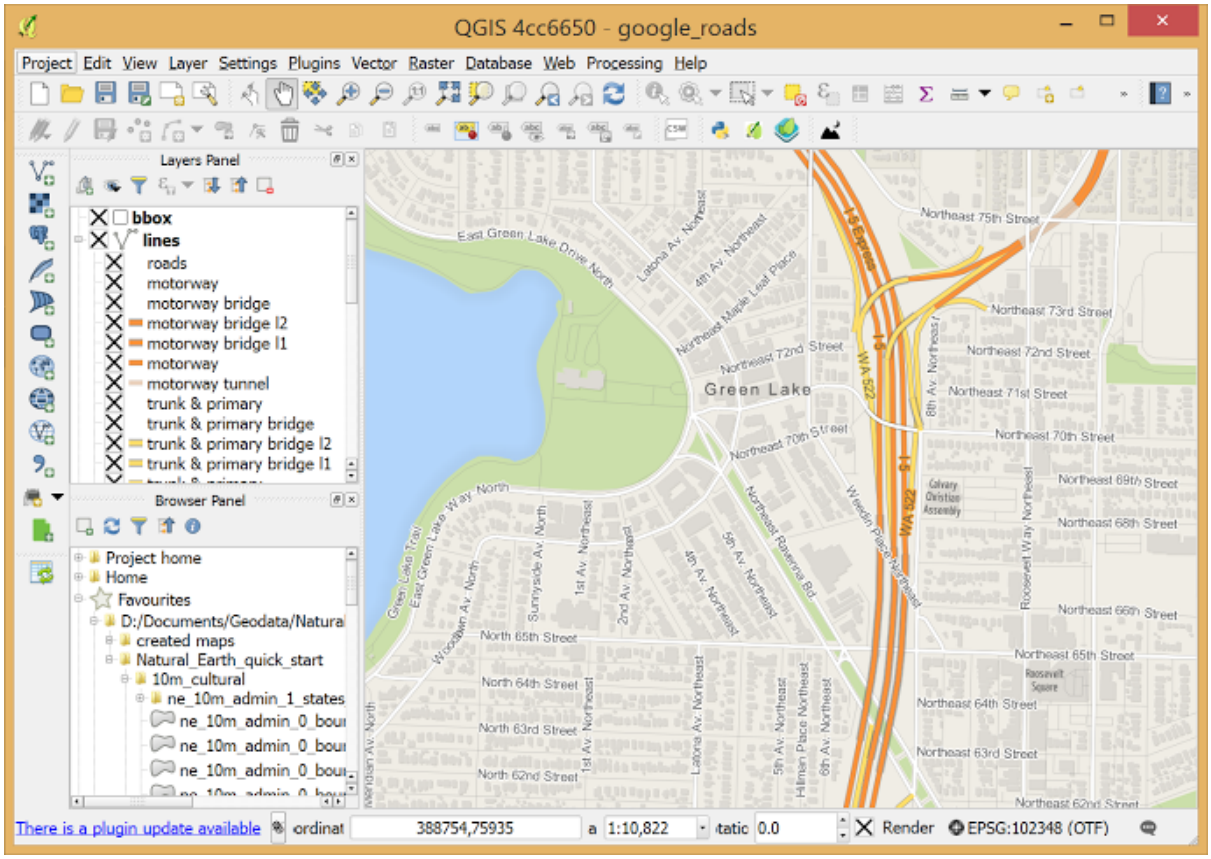

Figure 2.7: Screeshot of QGIS platform (QGIS 2023)

2.4 Spatial Decision Support Systems

The concept of SDSS has strong roots in the mid-1980s when Armstrong, Densham, and Rushton proposed designing a system to address the need for locational planning (Armstrong, Densham, and Rushton 1986). In the 1990s, SDSS established itself as a study field, as evidenced by its inclusion in an authoritative review of the GIS field by Densham (1991, as cited in Keenan 2003) and the foundation of research initiatives such as the National Center for Geographic Information and Analysis (NCGIA) focused on GIS and related technologies (*National Center for Geographic Information and Analysis* 2023).

SDSS integrate both spatial and non-spatial data, as well as the analysis and visualization capabilities of GIS, with decision models specific to certain domains (Keenan and Jankowski 2018). Spatial characteristics refer to a location's geographical coordinates and spatial relationships such as proximity, overlap, containment, and distribution pattern.

It incorporates a variety of data types and analytical modeling capabilities underlying on graphical interfaces, being able to adapt depending on the style of problem-solving and change its capabilities when required. Additionally, SDSSs embody the main principles of a DSS when they combine database, interface, and model components directed at a specific problem (Sprague and Carlson 1982). This structure for what will be an SDSS is proposed in 'Architecture for a microcomputer-based spatial decision support system' by Armstrong, Densham, and Rushton (see Figure 2.8) and is aligned with the capabilities described in the DDM diagram proposed by Sprague and Carlson.

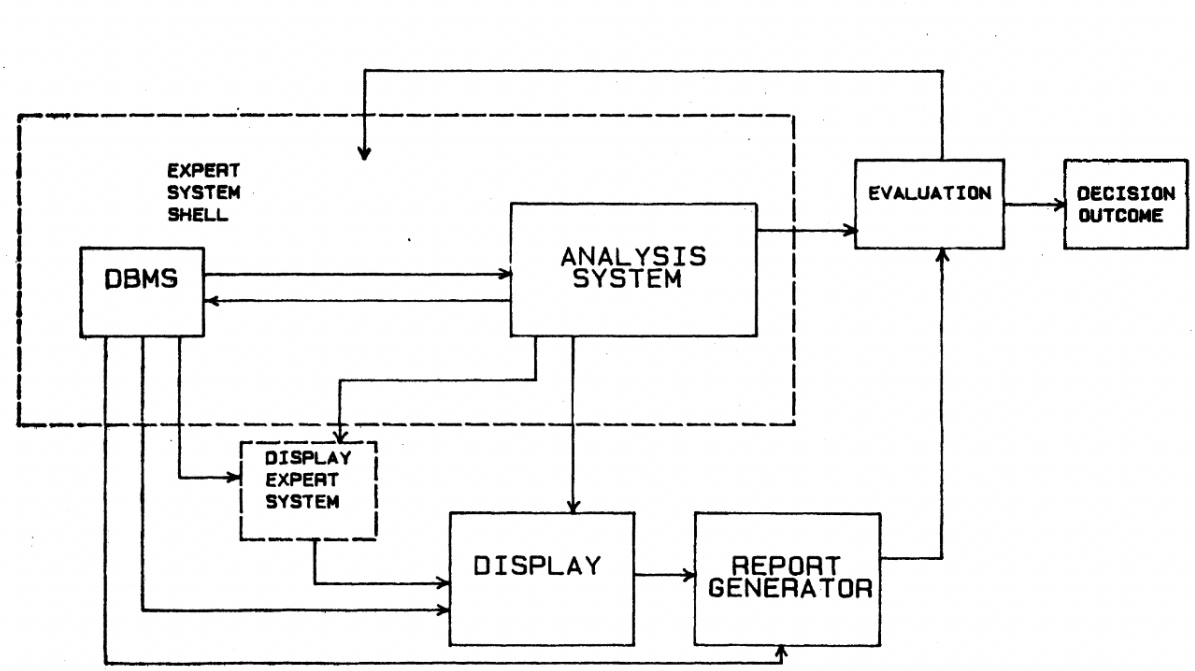

Figure 2.8: SDSS software components (Armstrong, Densham, and Rushton 1986)

Although DSSs and SDSSs are very intrinsically related, their concept is often applied to GIS, mostly because this system is frequently defined as a DSS that facilitates the integration of spatial data into problem-solving context (Sikder and Yasmin 1997). GIS lacks the modeling component necessary to qualify as a DSS, therefore Spatial Decision Support Systems can be viewed as a superset that results from the intersection of GIS and other techniques (Keenan 2003). This perspective considers GIS as a type of DSS generator that can be enhanced with models to create a specific DSS (Keenan 2003).

SDSS enables the integration of spatial data and models into decision-making processes, providing a more comprehensive understanding of the problem and its context. This integration is particularly useful in urban planning, where the spatial distribution of opportunities and the transportation network are crucial to the decision-making process. The structural components of a DSS are aligned with the idea of providing an interface for learning, specifically within the dialog component. This helps the acceptability of the policy-makers and urban planners of an urban accessibility model for driving their decisions.

2.4.1 Spatial Data in Transit networks

Spatial data became widely available as systems capable of analyzing and handling this type of data developed themselves into powerful tools. System Data Infrastructure (SDI) initiatives have cataloged a wider range of data sources, but integrating data from different sources remains problematic (Keenan and Jankowski 2018). Open source crowdsourced data, such as OpenStreetMap, a global project and community that aims to create and maintain a free and editable database and map of the world through the contributions of volunteer mappers shows up as a solution. As of March 2022, the OSM database includes almost 7.5 billion data points (nodes), contributed by approximately 1.8 million users, making it one of the most significant examples of a crowdsourced geo information project and volunteered geographic information concept (Grinberger et al. 2022).

'Big Data' and analytics are fields developed from the high volume of data sources, and spatial data is part of this. In a world where any mobile device is a potential data source, especially with the

prevalence of social media (Keenan and Jankowski 2018), more of this data is passively acquired, and location plays a crucial role in most application features. Therefore the modern analytics englobes use of the spatial data such as spatial movement of people across urban space and time (Keenan and Jankowski 2018), considering not just the routing analyses in terms of distances but the transit network surrounding the locations.

A transit system is not fully captured by the lines on a map or stop locations. The links in a transit network, such as bus routes and rail lines, effectively exist only when a bus or train arrives at specific moments in time (Guthrie, Fan, and Das 2017). Standard GIS data do not contain this temporal information. To address this lack of information, General Transit Feed Specification (GTFS) provide public transportation planners associated with spatial data. It is mostly known for the use with Google Maps transit directions and routing analysis features (Guthrie, Fan, and Das 2017).

The use of GTFS data alongside with geo referenced data present on geographic databases as OpenStreetMap can address the analysis, for instance of routing and travel time.

2.5 Web-based Spatial Decision Support

In an increasingly integrated and shared virtual environment, especially with the development of the web as a technology and cloud services, along with the growing availability of spatial data generated mainly in the context of big data, there is a demand for decision support systems capable of dealing with this integration. This demand is driven not only by the web technology's ability to increase the number of users (Rinner 2003), but also by the sharing and integration of data, flexible support, and the provision of these decision support systems on a larger scale, promoting them as a service (Rinner, Keßler, and Andrulis 2012). Bhargava and Power 2001 in 'Decision Support Systems report' conclude 'Decision Support Systems can benefit in many ways from the availability of Web technologies' through a distributed computation, platform independent and exchange of complex multimedia information.

WebSDSS provides decision support information and/or tools to the user through a thin-client web browser (Power 1998, as cited in Bhargava and Power 2001). This means it relies on the technologies supported by the web browser to deliver the interface to the user, even though the server side can make use of other technologies not supported by the client. In the case of a WebSDSS, the informations and features delivered are related to spatial data, considering a GIS as part of the system.

GIS-based systems on internet were described as client-server systems, dividing different functionalities among the two fronts as presentation, program logic and database (Peng and Tsou 2003, as cited in Rinner 2003), therefore WebSDSS will incorporate those characteristics at some point. Rinner 2003 distinguished WebSDSS by their program logic:

- Client-side WebDSS: The program logic and analysis are performed on the user's computer or device. These implementations are related to applications with interactive cartographic visualization methods and manipulation tools, as well as multi-criteria evaluation techniques on the client-side.
- Server-side WebDSS: In this type, the processing and analysis of data, are performed on the server-side. The client is responsible for setting the constraints and factors through interactive forms or even maps and sending them in a request to the server, then receiving the results that can be visualized through cartographic representations. This type is chosen when large amounts of data or complex processing are required.

- Mixed WebDSS: As the name suggests, data analysis and decision-making processes are executed on both the server and client sides. Most of these applications are used for collaborative decision-making processes involving input from various users.

Dessì, Garau, and Pes 2012 proposed an architecture (see Figure 2.9) for a WebSDSS relying on the data and features integration through APIs and reusable services. The architecture comprise on Data Collection Layer, an Application Layer, and a Presentation Layer. The main differential is the application layer, responsible for providing the core functionalities for the system through a modular service organization made of task-oriented, knowledge-oriented, data access/management services and a mashup-up modules. Those modules combined are able to extract, modeling and share data from and with external services, besides make use of their functionalities in the decision making process.

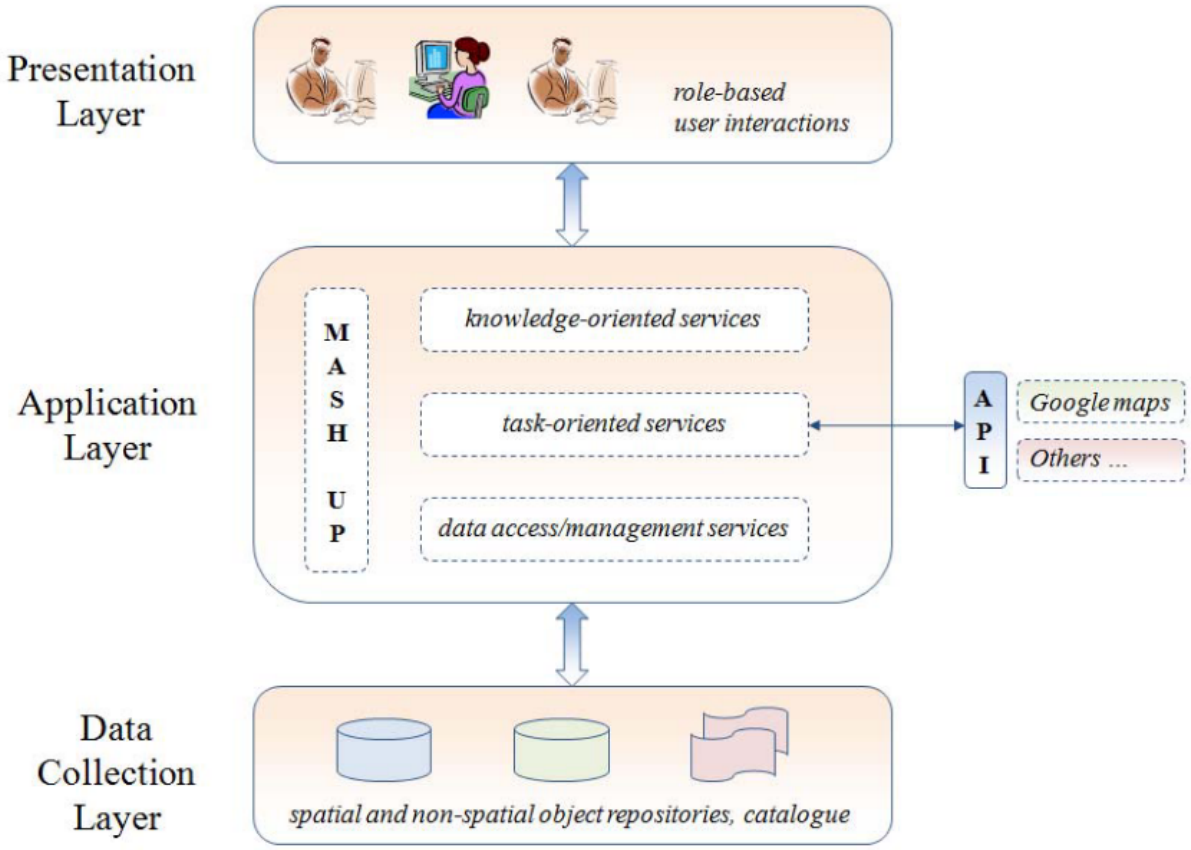

Figure 2.9: Architecture of a WebSDSS (Dessì, Garau, and Pes 2012)

Rinner 2003 also identifies that the availability of geographic data on the web itself make the environment itself a tool for support the decision making process, therefore some Web based systems that are limited to be a data retrieval or/and reporter, but has no decision support techniques implemented.

The choice of developing a WebSDSS is grounded in the principles of iterative development and the organizational environment. Web systems are more flexible and can be easily changed, which is important for a system that needs to adapt to changing problems or decisions and incorporate user feedback. Additionally, the web environment allows for the integration of different technologies, such as GIS, and the sharing of data and analysis models.

2.6 Related Works

While many studies have used GIS-based accessibility measures to evaluate domain-specific scenarios, the integration of non-spatial and spatial data in a single model needs to be conducted outside of GIS tools. Consequently, these studies have utilized various components to address this modeling, falling short of achieving a DSS structural concept as discussed in Section 2.3.2.

With the development of web technologies, the capacity for the integration of GIS tools such as ArcGIS (ArcGIS 2023) has increased. It presents a variaty of tools for spatial analysis that are spread along different products, including interactive mapping, spatial analytics, real-time data integration, and support for field operations through mobile applications. The platform also features advanced functionalities such as 3D visualization and indoor GIS, and it integrates with the Internet of Things (IoT) for real-time monitoring. Key products include ArcGIS Pro for desktop analysis, ArcGIS Online for cloud-based mapping, and ArcGIS Enterprise for on-premises deployment, along with various extensions like Network Analyst and Geostatistical Analyst that enhance analytical capabilities. The user can make use of those tools to support decision-making process applied to different contexts.

However, this general-purpose tool might not fit in a very specific-domain context where the user has no advanced knowledge of such tools, which affects usability and, as mentioned in Section 1.2, the onboarding of lay users. One of the characteristics of a decision support system, by definition, is the capacity of being usable by different users across contexts that intersect the domain in study. The author seeks to develop a system that fits well in the context of use, facilitating the use of the system by policymakers in the public decision-making context, achieving a better balance between simplicity and effectiveness. For this reason, the author opted not to include ArcGIS in the comparison.

2.6.1 Pereira's Work

Pereira's research (Pereira et al. 2021) focuses on the concept of accessibility, based on the idea that it is critical determinant of social inclusion, economic opportunity, and quality of life, particularly in urban areas where the concentration of resources and opportunities varies widely across different neighborhoods.

To accomplish this, Pereira developed an 'R package to conduct rapid realistic routing on multimodal transport networks. This tool, called r5r, is a powerful platform for conducting accessibility analysis, as it allows users to calculate travel times and distances between different locations, considering various modes of transportation and network conditions.

As an output of this reasearch a web tool was created to visualize on a map measures of potential accessibility, depending on transport, opportunity type and city. The tool aims to provide policy-makers, urban planners, and researchers with valuable insights into spatial disparities in access to opportunities within Brazilian cities (see Figure 2.10).

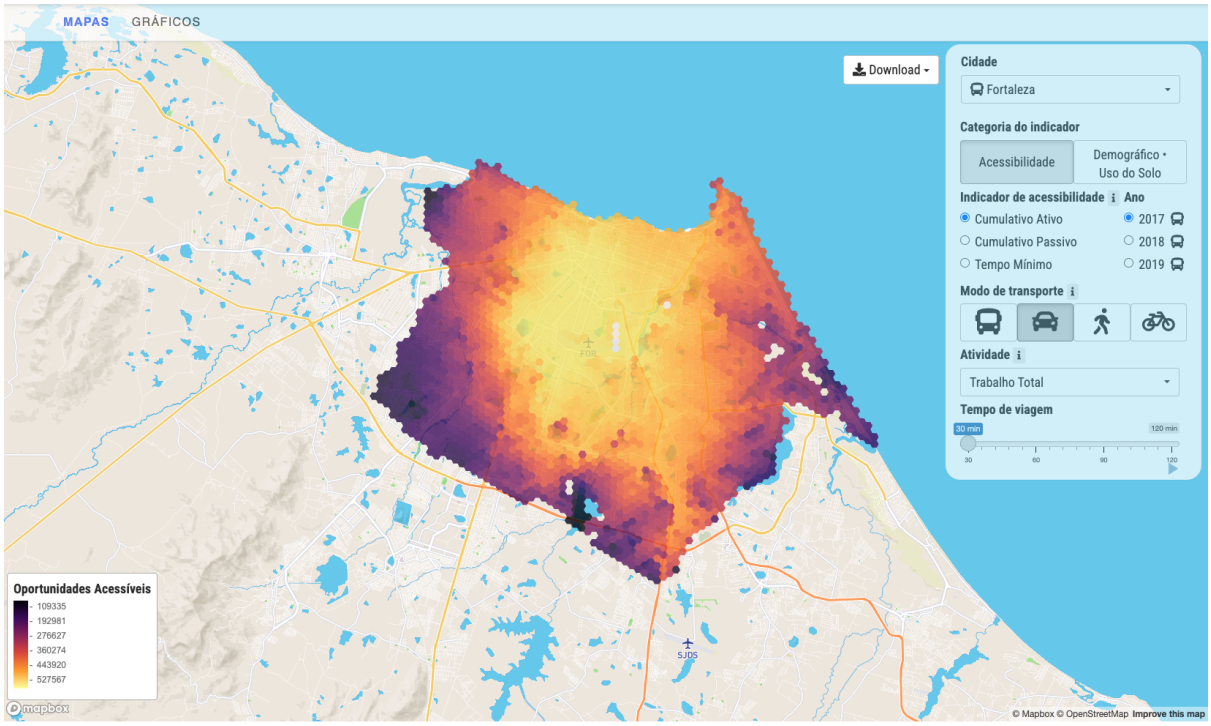

Figure 2.10: Screenshot of r5r explore web tool (Pereira et al. 2021)

The tool utilizes spatial analysis techniques and geographic information systems (GIS) to quantify and visualize accessibility patterns across Brazilian cities. By integrating various data sources, including transportation networks, land-use data, and the location of amenities and services, it generates detailed accessibility maps that highlight areas of opportunity and areas with limited access.

Figure 2.11 shows a screenshot of the r5r explore web tool, plotting the results of a potential accessibility analysis for the city of São Paulo, Brazil. The map displays the number of jobs accessible within a travel time of 30 minutes from different origins represented by centroids of hexagons that segment the city.

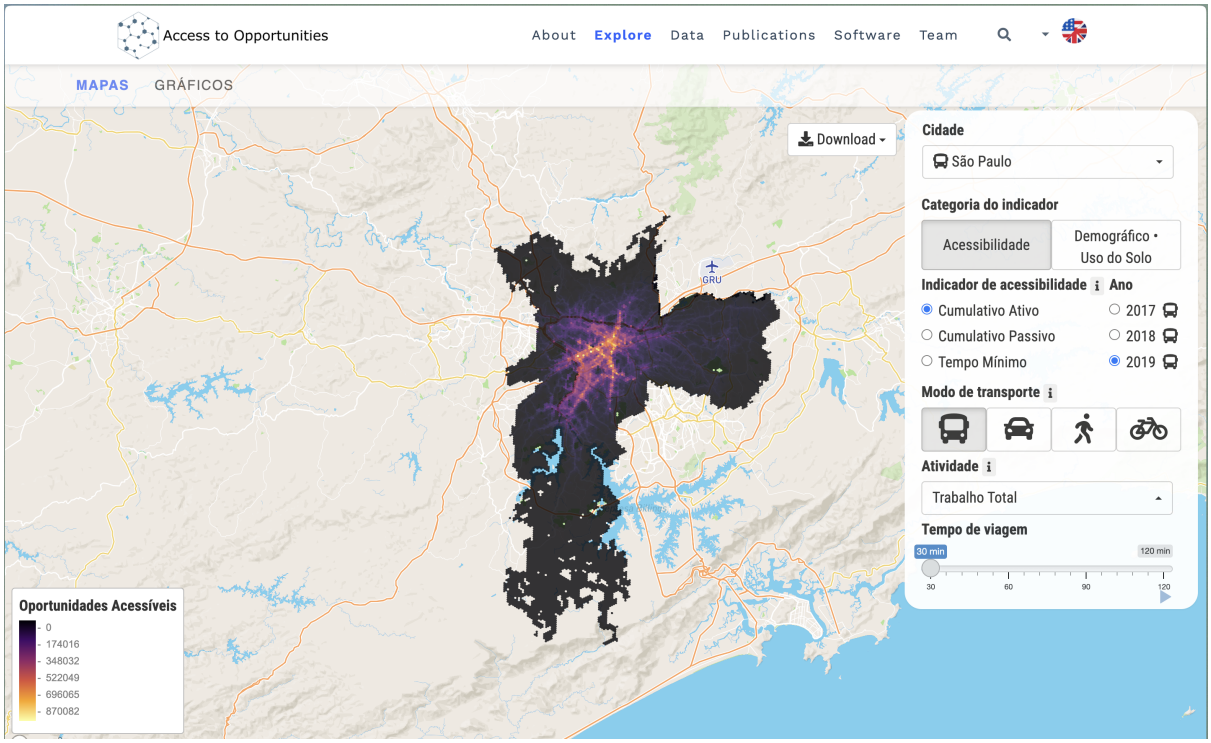

Figure 2.11: Number of total jobs accessible within 30 minutes by public transport by the population of State of São Paulo, Brazil (Pereira et al. 2021)

These features support efforts to evaluate the access to opportunities in big cities. Furthermore, the tool is open source, allowing for transparency, collaboration, and customization.

Although the tool enables different analyses, it was not designed to be a decision support system. Firstly, it doesn't present itself as an end-to-end system with a dedicated website. This limits its ability to provide a seamless user experience and hinder the implementation of a dialog component, as discussed in Section 2.3.2.

Finally, it does not include socioeconomic and demographic data, which are important in order to understand the disparities in access to opportunities among different socio-economic groups and individuals. Furthermore, a multi-criteria evaluation component is also necessary for comparing different scenarios and identifying the most effective interventions to improve accessibility.

2.6.2 Conveyal

Conveyal's web application (Conveyal 2023a) (see Figure 2.12), which relies on R5 routing engine, expedite the calculation of robust accessibility indicators, enabling the visualization of underlying patterns and network effects. It provides control of different aspects of accessibility's transport and temporal components, enabling users to compare potential scenarios and future plans. It enables the change of transport mode, destination and departure point, travel time, speed, and the area of analysis.

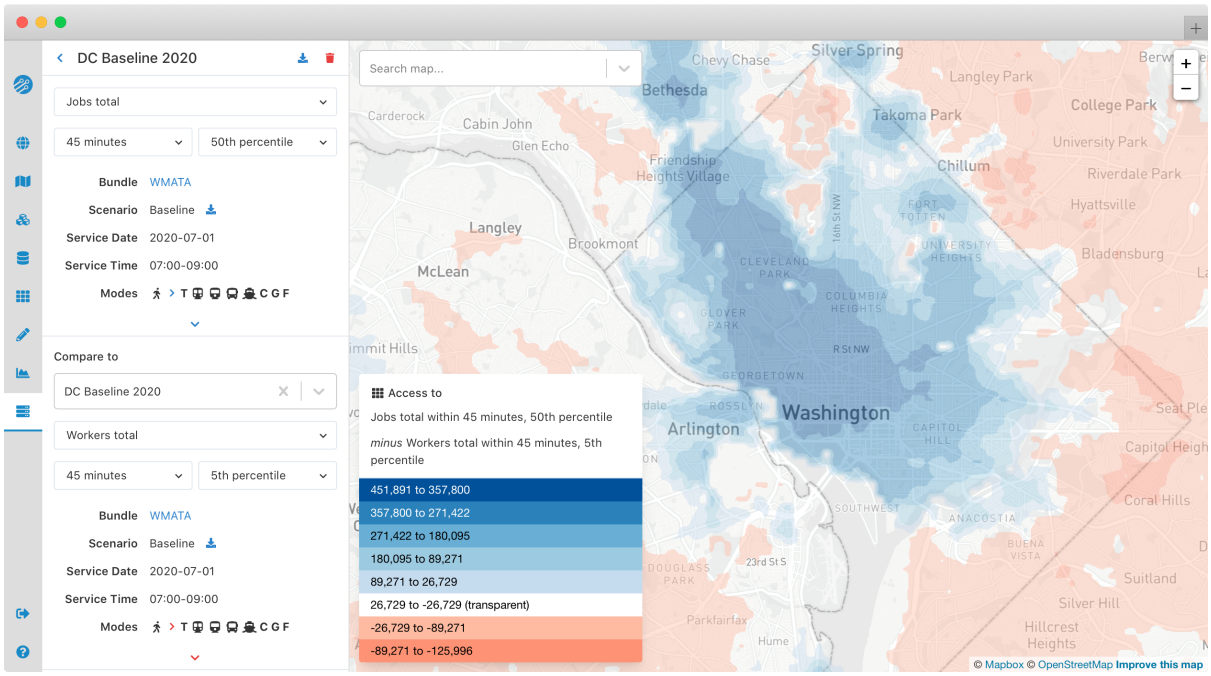

Figure 2.12: Conveyal platform. Output of the analysis regarding number of jobs accessible within 45 minutes in Washigton DC (Conveyal 2023a).

At the heart of Conveyal's capabilities lies its scenario planning feature. Planners can create and assess various land-use and transportation scenarios, considering factors such as population growth projections, zoning regulations, and transit options. By simulating different scenarios, decision-makers gain valuable insights into how different policy choices and infrastructure investments shape land-use patterns over time.

Conveyal also offers a range of accessibility indicators, including the number of jobs, schools, and other amenities accessible within a given travel time. These indicators are essential for understanding the spatial disparities in access to opportunities and identifying areas with limited access. Planners can pinpoint neighborhoods or regions where residents face significant barriers in reaching healthcare providers, schools, or job opportunities.

One of the features present in Conveyal is the capacity to upload your own data. It's possible to upload various types of data, including transportation networks (such as road networks, transit routes, and bike lanes) (see Figure 2.13), land-use information (such as zoning maps and parcel data), demographic data (including population counts and socio-economic characteristics), as well as point-of-interest data (such as the location of amenities, services, and employment centers). This functionality allows users to incorporate locally relevant data into their analyses, ensuring that the modeling outcomes accurately reflect the specific conditions and context of the study area. Additionally, Conveyal supports the integration of data in standard formats such as shapefiles, GeoJSON, CSV, and OpenStreetMap data, facilitating seamless data interoperability and analysis workflows.

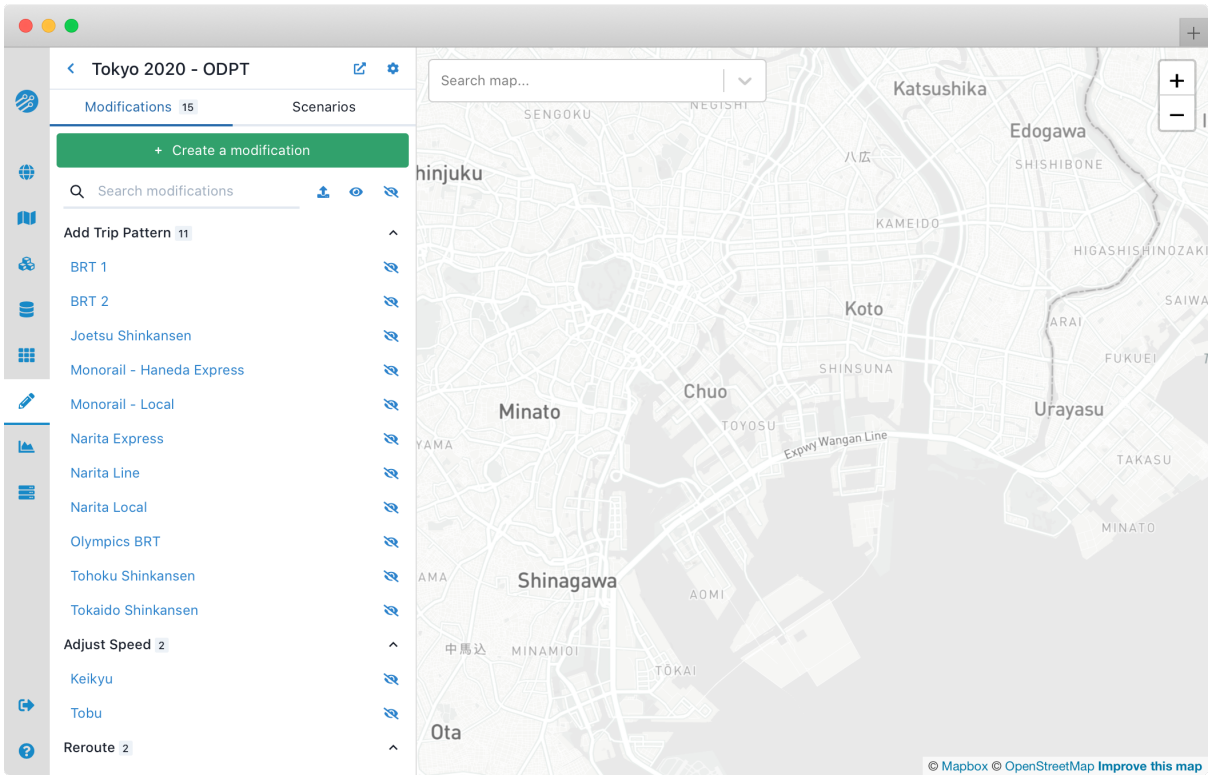

Figure 2.13: Edit page of Transport Network data (Conveyal 2023a)

Conveyal's platform is designed to be user-friendly, with an intuitive interface that allows users to interact with the data and models, visualize the results, and share their findings with others. The platform's visualization capabilities enable users to create compelling maps, charts, and graphs that effectively communicate the results of their analyses.

It offers both free and paid versions of its software platform. The free version provides basic functionality and access to certain features, while the paid version offers additional capabilities, advanced tools, and support services.

Among the related works, Conveyal is one of the most powerfull and complete web application regarding scenario analysis, accessibility assessment, transit and land-use planning, equity analysis, visualization, and resilience planning. But as the others, do not rely on a module to implement multi-criteria evaluation. It does allow the comparison of different scenarios but doesn't suggest the best option based on given criterias.

2.6.3 Online Multicriteria WebSDSS

The work presented in Krügel et al. 2024 is a WebSDSS that supports planning decisions in the context of public service infrastructures (e.g., schools, pharmacies, supermarkets). The application combines multi-criteria decision analysis and geographical information systems to provide planners with a tool for analyzing the spatial distribution of public services and identifying areas with limited access.

To conduct an analysis of supply and demand, the system requires the user to define the public services that will be included in the analysis (supply) and the population group that requires those services (demand). The user must then assign weights to both the supply and demand aspects, indicating the relative importance of different types of infrastructure and the decay of supply

accessibility with increasing distance from the population. This means that the accessibility of a service decreases as the distance between the service and the population increases.

The system then calculates the accessibility of each service for the population group, taking into account the distance between the service and the population, the service's capacity, and the population's demand. The results are visualized on a map, with different colors representing different levels of accessibility. The system also generates a report summarizing the analysis results and highlighting areas with limited access to public services as seen in the prototype in Figure 2.14.

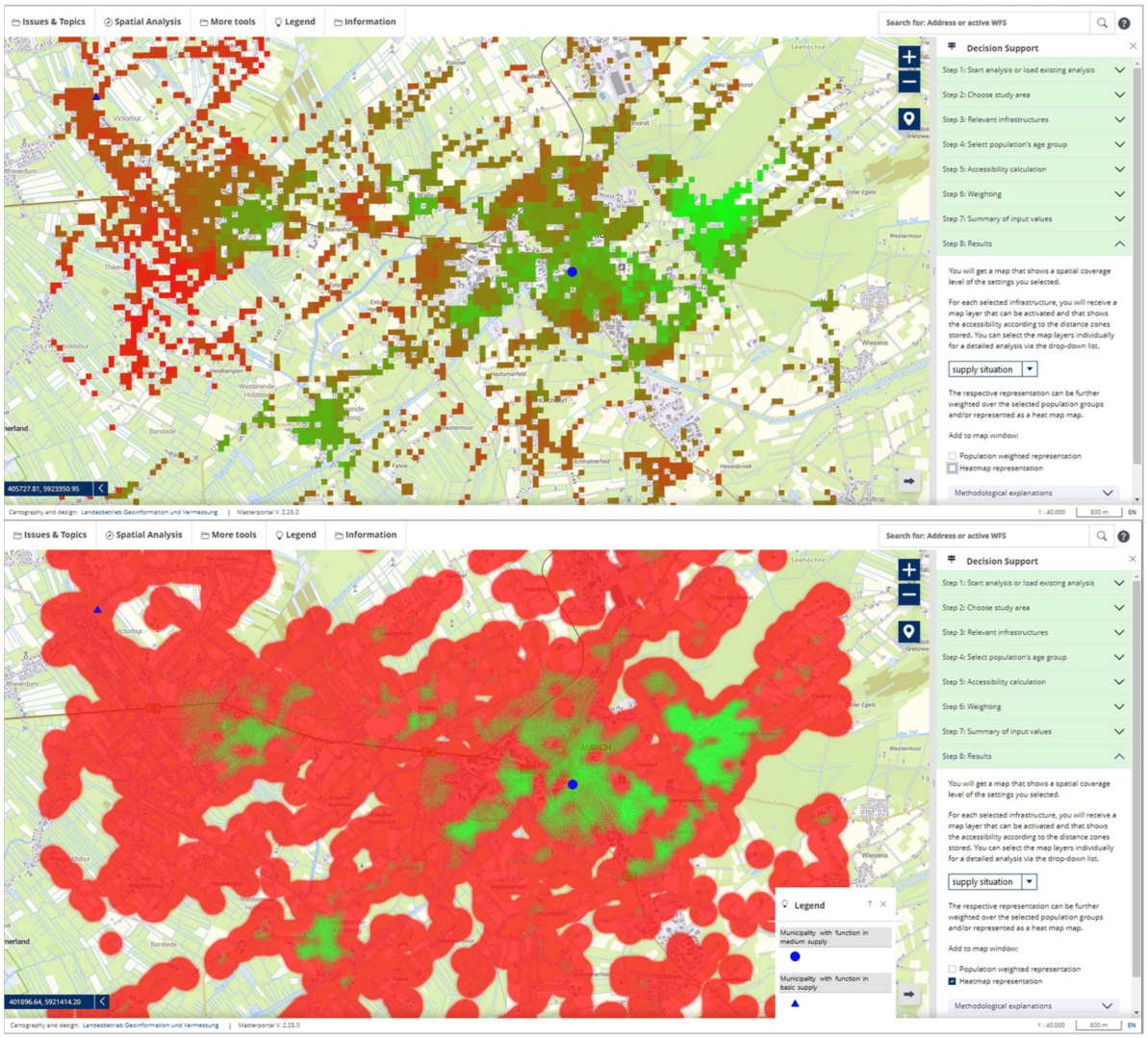

Figure 2.14: Flow diagram of the proposed system (Krügel et al. 2024)

This work is a good example of a WebSDSS that incorporates multi-criteria evaluation techniques to support planning decisions. It allows users to define their own criteria and weights, providing a flexible and customizable tool for analyzing the spatial distribution of public services. It considers socio-economic and demographic data, which are essential for understanding the disparities in access to public services among different population groups. Besides of including a visualization component that enables users to explore the results of the analysis on a map and identify areas with limited access to public services. Finally, it is aligned with the idea of having a user-friendly interface that can be easily accessed through a web browser, making it accessible to a wide range of users.

This related work closely aligns with the goals presented in this dissertation. However, its focus is primarily on enabling multicriteria analysis within an accessibility model, using a general model as the mechanism for generating specific DSS. In contrast, the system proposed in this dissertation aims not only to allow the creation of different SDSS through its multicriteria accessibility model, but also through its components, leveraging various technologies and data sources. This approach enables a more flexible and customizable tool by integrating different components, external services, and web-accessible data available in the web environment.

Aditionally, the system proposed in this dissertation is set to compare different points of interest to install the public services, while the related work presents an analysis of a given area and the services that are already present in it.

2.6.4 Conclusion

As seen most of the related works presented are focused on the analysis of accessibility, but just a few of them are capable of providing a decision support system alongside a multi-criteria evaluation. This is mostly related to the complexity of building a system that can address most of decision-making making scenario, mainly when it involves accessibility analysis withing urban planning.

Among the works, Pereira's work and Conveyal are powerful tools for accessibility analysis, but they do not include a module for implementing multi-criteria evaluation. On the other hand, the Online Multicriteria WebSDSS is a good example of a WebSDSS that incorporates multi-criteria evaluation techniques to support planning decisions, but it primarily focuses on the model itself rather than the system implementation. Specifically for Pereira's work and the Online Multicriteria WebSDSS, their implementation is still in a very exploratory phase, so they are considered non-available tools for public sector use, but they are open-source. Additionally, all of these tools allow for the comparison of different scenarios but do not provide the best option based on given criteria and alternatives.

The Table 2.6 shows a comparison between the three major works presented in terms of the criteria discussed: integration of non-spatial and spatial data, web technology utilization, accessibility indicator calculation, visualization capabilities, multi-criteria evaluation support, and open-source availability.

Table 2.6: Comparison of Decision Support Systems

Criteria	**Conveyal**	**Pereira's Work**	**Online Multicriteria SDSS**
Integration of non-spatial and spatial data	Yes	No	Yes
Web technology utilization	Yes	Limited	Limited
Accessibility indicator calculation	Yes	Yes	Yes
Visualization capabilities	Yes	Yes	Yes
Multi-criteria evaluation support	Limited	No	Yes
Open-source	No	Limited	Limited

In conclusion, the proposed system aims to address the limitations of existing tools by providing a comprehensive decision support system that fullfill most of the criteria and offers a user-friendly interface accessible through a web browser. By leveraging various technologies and data sources,

the system will enable users to conduct comprehensive accessibility studies, compare different scenarios, and identify the most effective interventions to improve accessibility in urban areas.

2.7 Technologies

This section explores various technologies aimed at solving the problem. These technologies are associated with building components for a Client-side WebSDSS and are compared to select the one that best meets the project's requirements. Among the technologies discussed are those for modeling, storing, analyzing, and visualizing spatial and non-spatial data.

2.7.1 Spatial Data Modeling

The accessibility model, essential for the development of the decision-making platform, necessitates the manipulation of geolocation and geometry data—either through transformation or visualization plotting. Moreover, there is a requirement to model this data alongside socioeconomic and demographic indicators, creating a comprehensive model that establishes relationships between this information.

This particular need calls for a programming language capable of handling various geographic data formats and facilitating the creation of representations that seamlessly incorporate both spatial and non-spatial information. To fulfill this purpose, object-oriented languages have been chosen for their modeling ease, a characteristic inherent in this paradigm.

Python

Python, renowned for its versatility and user-friendly nature, offers a range of features such as being interpreted, dynamically typed, and supporting multiple paradigms (Python Core Team 2023). It finds applicability across diverse domains.

GeoPandas (GeoPandas 2023) is one of the prominent libraries in Python designed for working with geospatial data. Extending Pandas, GeoPandas facilitates spatial operations on geometric types. Leveraging the capabilities of Pandas, it provides tools for reading, writing, and manipulating spatial data in various formats, including shapefiles, GeoJSON, and PostGIS (PostGIS Development Group 2022) (PostGIS, an extension for PostgreSQL, enables the management of geospatial types and functions).

In the realm of data visualization, Python boasts the Matplotlib (Hunter 2007) library. Matplotlib offers an extensive array of plotting functions and customization options, making it suitable for generating high-quality visualizations across different data types. It includes the Basemap toolkit, allowing the creation of maps with geospatial data plotting.

For routing and transit network analysis, Python features libraries such as R5py and PyOTP (*PyOTP - The Python One-Time Password Library* 2023). R5py (Conveyal 2023b) facilitates rapid, realistic routing on multimodal transport networks (walk, bike, public transport, and car), enabling developers to calculate travel time matrices and assess accessibility across various travel modes.

By combining GeoPandas, Matplotlib, R5py, and PyOTP, users can seamlessly read, manipulate, and visualize spatial data, gaining insights and creating compelling visualizations.

Ruby

Ruby is strongly rooted in Object-Oriented principles but embraces support for multiple paradigms. As an open-source language (Ruby Core Team 2023), Ruby is renowned for its dynamism, simplicity,

and productivity. It is a popular choice for web app development and finds applications in automation, data analysis, and DevOps.

Ruby boasts a variety of libraries catering to different needs, including geospatial data modeling and analysis. One notable library is RGeo (rgeo Development Team 2023), enabling developers to create location-aware applications in Ruby. RGeo offers features such as the representation of spatial and geolocation data objects (points, lines, and polygons), spatial analysis, geometry operations, and the ability to read and write locations in Well-Known Text (WKT) and Well-Known Binary (WKB) formats.

JavaScript

JavaScript (International 2023) is a versatile programming language known for its clear syntax and interpretation capabilities. Extensively used in web development, hybrid mobile applications, and server-side programming, it benefits from a large and active community. Its advantages include: interpreted nature, cross-platform compatibility, and a rich library ecosystem that facilitates project development and integration with other tools.

In terms of geographic data handling, JavaScript stands out with its abundant and mature libraries and tools. Examples include Leaflet (Vladimir Agafonkin and Leaflet Contributors 2022) and OpenLayers (OpenLayers Contributors 2024), which provide robust capabilities for the manipulation and analysis of geographic data.

Comparison

Table 2.7 compares the languages based on criteria important for this work.

Table 2.7: Comparison of Programming Languages

Criterion	Python	Ruby	JavaScript
Ease of learning	Simple and easy-to-learn syntax	Simple and easy-to-learn syntax	Simple and easy-to-learn syntax
Language type	Multi Paradigm	Object-oriented	Multi Paradigm
Community and support	Has an innovative and data science mostly focused community with a wide range of libraries and frameworks	Has an innovative and web-focused community with a wide range of libraries and frameworks	Web-focused, large and constantly evolving community with a wide range of libraries and frameworks
Applications	More suitable for academic applications, artificial intelligence, machine learning, and scientific programming	Best used for web applications and functional programming, such as the Rails framework	Extensively used in web development, hybrid mobile applications, and server-side programming
Performance and efficiency	May be slower in terms of performance and resource consumption	Faster and more memory-efficient, especially in web applications	Generally slower than compiled languages but optimized for web applications
Ability to handle geographic data	Has the GeoPandas library and the Matplotlib library for manipulation and visualization of geographic data	Has the RGeo library and the Ruby on Rails framework with the Geocoder extension for manipulation and geocoding of geographic addresses	Has abundant and mature libraries and tools, such as Leaflet and OpenLayers, for manipulation and analysis of geographic data

2.7.2 Geospatial Databases

As mentioned earlier, the utilization of geo-referenced data is essential, and to achieve this, a database containing this information will be necessary. These geo-referenced data must correspond to the locations of streets, avenues, bus stops, terminals, and roads. In the actual application scenario, these data will be employed to construct travel time matrices and calculate the potential accessibility.

OpenStreetMap

OpenStreetMap (OSM) (OpenStreetMap contributors 2023) is an open-source geographic database maintained through collaboration among over 1 million volunteers in an active and extensive community. The database was created to tackle the issue arising from the limited availability of open data maps. It encompasses various types of data alongside georeferenced information, including buildings, addresses, shops, points of interest, railways, land use, and more.

Google Maps

Google Maps is a free, high-level mapping service developed and operated by the North American information technology company, Google. It offers street maps, satellite imagery, and additional features such as directions, location search, and marker sharing. Renowned for its user-friendly interface and map accuracy.

The Google Maps API (Application Programming Interface) exposes Google Maps data and features to developers, providing access to static, interactive, and customizable maps, as well as information on places, points of interest, routes, and locations. It is commonly used to integrate various types of maps into web or mobile applications (Google 2023). Note that some functionalities may require payment.

Bing Maps

Bing Maps, developed by Microsoft, is a comprehensive mapping service offering users an intuitive and user-friendly interface to navigate the world. While primarily proprietary, Bing Maps provides detailed and accurate geographic information, encompassing various features such as points of interest, businesses, and transportation networks. Its user-friendly design ensures easy navigation, making it a valuable tool for both commercial and personal use. However, it should be noted that Bing Maps is not as openly collaborative as OpenStreetMap, as its data sources are predominantly proprietary. Nonetheless, the service continues to be a reliable resource for high-quality mapping and location-based information.

Comparison

Table 2.8 compares the databases based on criteria important for this work.

2.7.3 Web-App Framework: User Client Interface and Application Server

This work proposes the creation of a web platform for decision-making, and for its development, the choice was made to utilize a web framework due to the advantages it offers. Web system frameworks provide a set of tools, components, and pre-structured conventions for the rapid, scalable, and secure development of a web system. The primary purpose of a framework is to enable developers to efficiently structure their applications.

Table 2.8: Comparison of Geospatial Databases

Feature	Google Maps	OpenStreetMap	Bing Maps
Geographic database	Proprietary and closed	Open and collaborative	Proprietary and closed
Service type	Commercial and free	Free and collaborative	Commercial and free
User interface	Easy to use and intuitive	Easy to use and intuitive	Easy to use and intuitive
Map accuracy	High	High	High
Offline data availability	No	Yes(partially)	No
Map editing	No	Yes	No

Rails

Ruby on Rails is a framework based on the Model-View-Controller (MVC) pattern for building web applications written in the Ruby programming language (Ruby on Rails Team 2012). It is designed to expedite web application development by providing default structures for a database, a web service, and web pages. Rails includes everything needed to create database-backed web applications, such as rendering HTML templates, updating databases, and providing security protections. It also facilitates both front-end and back-end development, along with the easy implementation of business logic.

The Ruby on Rails framework has an extension called Geocoder (Alex Reisner 2015), which enables geocoding and geographic address searches. Additionally, the framework supports PostGIS.

Ruby on Rails is renowned for its active community and support, facilitating problem-solving and resource discovery.

Django

Django is a framework with a pragmatic design that facilitates easy web application development, following the Model-View-Template (MVT) pattern (Django Software Foundation 2010). In this pattern, the template represents the user interface. The template and view together constitute the view in the MVC pattern of other web frameworks, such as Rails. Templates are static files that Django fills in with data, allowing for data use, logic handling, built-in tags and filters, and customization of templates with code extensions. Templates are the primary tool for building user interfaces within the framework.

Django has a vast library of tools for visualizing geographic data, including GeoDjango, an extension of Django that enables the creation of geospatial applications and provides support for PostGIS, an extension of PostgreSQL for storing and manipulating geospatial data.

Next.js

Next.js (Vercel, Inc. 2024) is a JavaScript web development framework built on React, designed for creating high-performance web applications with minimal configuration (React Contributors 2024). It supports server-side rendering and static site generation, making it suitable for both small and large-scale projects.

Following the JAMstack (JavaScript, APIs, and Markup) architecture. JavaScript handles dynamic functionalities on the client-side, allowing developers to build rich, interactive user interfaces. APIs serve as reusable services that provide data and functionalities via HTTP, enabling the frontend to interact with various backend services and databases. Markup refers to prebuilt HTML that is rendered at build time, ensuring fast load times and improved SEO. The idea behind this architecture is emphasizing the decoupling of the frontend and backend, enhancing performance and security

Finally, it integrates well with technologies like TypeScript (TypeScript Contributors 2024) and CSS-in-JS, and can be combined with GIS tools such as Leaflet and OpenLayers for geographic data manipulation and visualization.

Comparison

Rails, Django, and Spring have the capability to handle geographic data for visualization. All three support the implementation of JavaScript libraries such as Leaflet.js (Vladimir Agafonkin and Leaflet Contributors 2022) and Mapbox.js (Mapbox 2022), which are popular for creating interactive maps and visualizing geospatial data.

Table 2.9 below shows a comparison with the most relevant criteria for this work.

Table 2.9: Comparison of Web Frameworks

Framework	**Django**	**Rails**	**Next.js**
Programming Language	Python	Ruby	JavaScript
Paradigm	MVT	MVC	JAMstack
Development	Complex	Simple	Medium
Scalability	Yes	Yes	Yes
Documentation	Yes	Yes	Yes
Geospatial Data Visualization	GeoDjango, Leaflet.js, Mapbox.js	Leaflet.js, Mapbox.js, GeoCoder	Leaflet.js, OpenLayers, Mapbox.js

Conclusion

In the initial phase, the focus can be on the frontend capabilities of the system, so Next.js is the best choice for this task, primarily to facilitate building a system with interactive maps. The backend can be developed in Python or Ruby, but initially, having a solid spatial database with the data already prepared and modeled for the specific task allows the backend to be simplified to a simple API that will query the database.

For the process of data preparation, such as applying accessibility models, routing analysis, and generating geographically structured data objects, Python is the best choice. This is mainly due to the libraries available for this purpose, such as Pandas, GeoPandas, and R5py. The community and support for Python are also more focused on data science and geographic data manipulation, making it the best choice for this task.

Chapter 3

Analysis

The Analysis chapter delves into three key aspects of the project development. Initially, it describes a real-life case study of Fortaleza city in Brazil, serving as the application setting for the proposed solution. This section includes an overview of accessibility levels and the socioeconomic and demographic makeup of its populace. Subsequently, drawing from existing literature and prior projects, it outlines both the functional and non-functional requirements of the solution. Additionally, it includes requirements modeling utilizing Unified Modeling Language (UML) diagrams.

3.1 Case Study of Fortaleza City

It is important to note that the data used in this analysis is based on the 2010 census, which may not reflect the current state of Fortaleza. The census is typically conducted every 10 years, but due to delays in data processing, the most recent information is not yet available. However, the data still provides valuable insights into the demographic and socioeconomic characteristics of the city. The data was obtained from the Brazilian Institute of Geography and Statistics (IBGE) (Brazilian Institute of Geography and Statistics 2024) and the Fortaleza City Hall (Fortaleza City Hall 2024), covering aspects such as population, education, income, and more. This description provides a brief overview of the education, health, and transportation network areas, aiming to gain a comprehensive understanding of the city's past situation and identify potential areas for improvement.

Fortaleza, the capital of the state of Ceará, is the 4th largest city in terms of population and the 8th largest in terms of territory size in Brazil. The city has a population of approximately 2.5 million people, with a population density of almost 8,000 inhabitants per km2. The city is divided into 119 neighborhoods and 3,043 census tracts, each with its own characteristics and challenges (Instituto Brasileiro de Geografia e Estatística (IBGE) 2024).

The city's population is relatively young, with 30% of residents under the age of 20. Fortaleza has a high level of income inequality, with the wealthiest 10% of the population earning 40% of the total income, while the poorest 10% earn only 1%. Figure 3.1 show income distribution based on neighborhoods of Fortaleza. In terms of education, the city has a literacy rate of 6.9% for people who can't read or write, with 6.5% between the ages of 24 and 59 and 2.1% between 15 and 24. Education facilities are spread over 937 kindergarten schools, 1,109 elementary schools, and 298 high schools. The city has 531 healthcare facilities, with 426 administered by the private sector and 105 by the public sector, with 187 covered by the SUS (Sistema Único de Saúde) public healthcare insurance. The number of facilities with hospitalization services is 56, totaling 6,704 beds, with 2,803 being public and 3,901 private (Instituto Brasileiro de Geografia e Estatística (IBGE) 2024).

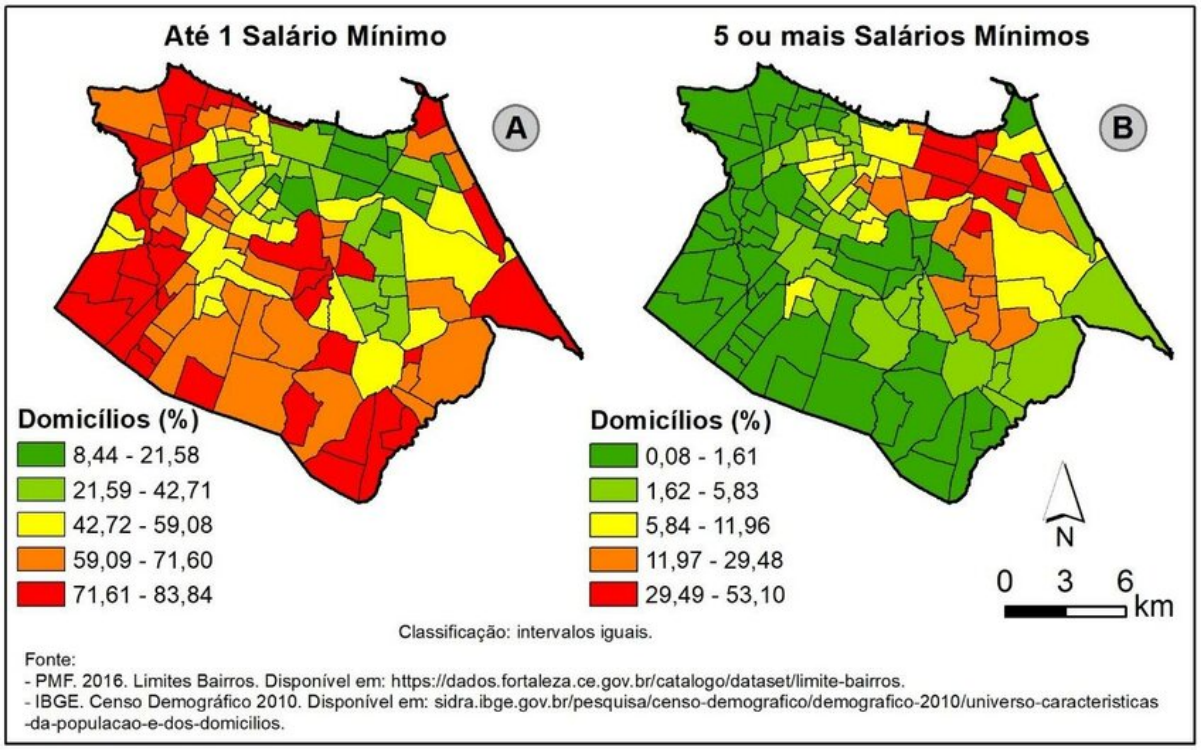

Figure 3.1: Distribution of population according monhtly income per individual by neighborhoods (Olímpio et al. 2020)

The city heavily relies on buses as the primary mode of public transportation. It has approximately 350 bus routes covering a total area of 314 km2 (PINTO, 2020). The Integrated Transportation System of Fortaleza (SIT-FOR) operates the public transportation system, which follows a trunk-feeder structure. The city also has two railway lines, Linha Oeste and Linha Sul, connecting the central region to other areas. Despite extensive bus coverage, connectivity is limited due to the reliance on terminals as integration points. The metro-rail network faces challenges in fully integrating into the transportation system. Recent interventions include BRT bus lines, dedicated lanes, a shared bicycle system, and the construction of the East metro line (Sousa 2019). Figure 3.2 shows the transport network of Fortaleza, highlighting the bus routes and metro lines.

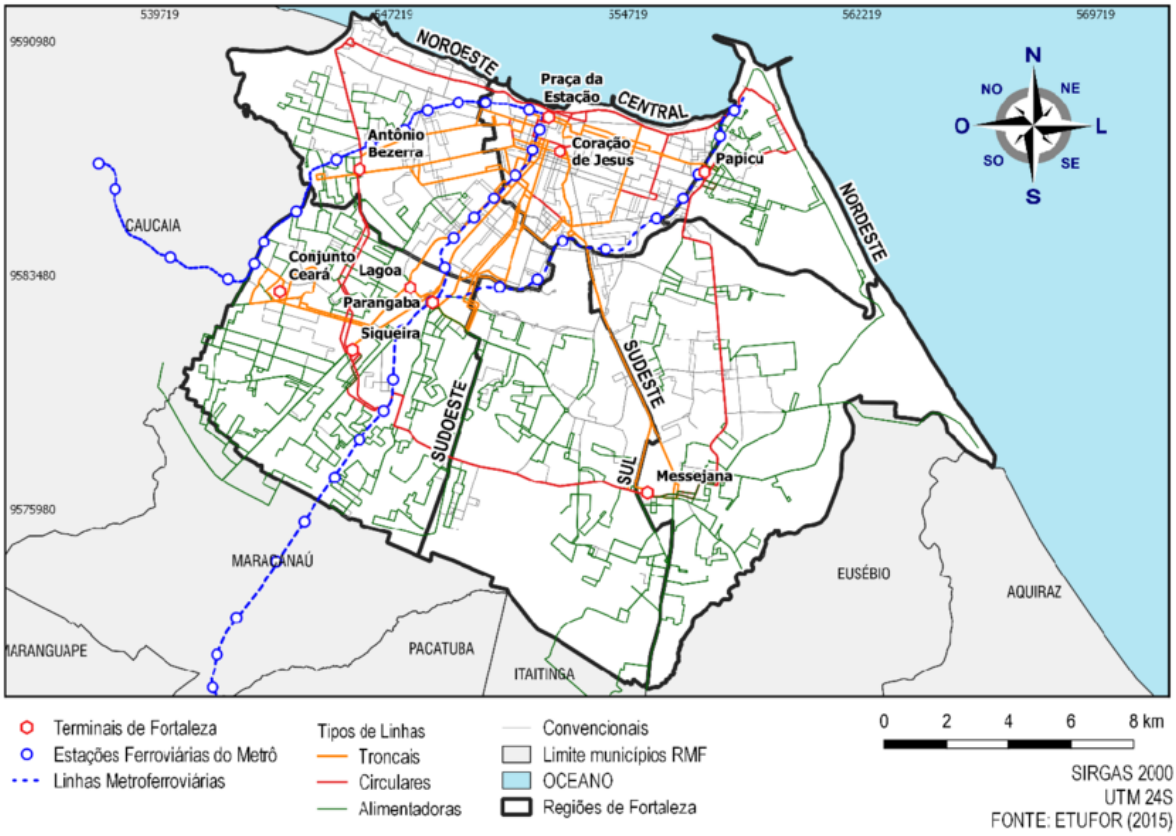

Figure 3.2: Fortaleza's transport network (Sousa 2019)

3.2 The project

As stated at Section 1.3, this dissertation focused on the development of a Web-based Spatial Decision Support Systems (WebSDSS) that uses data science to assist public managers in decision-making regarding the installation of public facilities, taking into account urban accessibility, macroeconomic and demographic indicators of the population in the vicinity.

The application should make use of a Multi-Criteria Decision-Making (MCDM) method to rank the best location for the installation of a public facility, considering the criteria provided by the policymaker regarding characteristics of the population in the area. The value of each criteria is calculated based on an accessibility model that count the amount of people that can reach certain point of interest based on some given parameters of accessibility such as transport mean, time threshold, etc. The criteria to be used in the multicriteria method are different segmentation of people such as people by age, per capita income, etc. Finally, the application should be designed to be user-friendly, allowing the user to interact with an interative map.

3.3 Domain Analysis and Requirements

In this section, the domain analysis and requirements of the project are detailed. This section aims to provide an overview of the domain and requirements that underpin the solution. UML diagrams are used to illustrate the business entities and user interaction scenarios with the solution. Lastly, both functional and non-functional requirements of the solution are addressed. These requirements include concerns such as technological constraints, response times, among other factors that influence the design and implementation of the system.

3.3.1 Domain Model

The Domain Model is a structured representation of the problem domain, incorporating vocabulary, key concepts, behavior, and relationships of its entities. It serves as a conceptual framework for understanding the domain and its entities, helping to eliminate gaps in understanding and interpretation of requirements. The Domain Model should accurately depict the problem being solved and the proposed solution (Culttt 2014).

The model created that represents in the context of this work is shown in Figure 3.3. It is a model that considers the entities:

- **Policy-maker**: Represents the system users, who are individuals from the public sector involved in the decision-making process.
- **Instructions**: Represents the guidelines provided to the user for completing the form.
- **Interactive Map**: Represents the map that the user interacts with to visualize spatial and sociodemographic data, pinpoint points of interest, and view accessibility analysis results.
- **Point of Interest**: Represents the locations marked by the user on the map. These points mainly correspond to potential sites for the installation of public facilities.
- **Analysis Input Form**: Represents the form that the user fills out to submit the analysis. This form consists of the Address Form, Input Object, Modeling Component, MCDA Component, and Socioeconomic Demographic Criteria Input Form.
- **Accessibility Model Input Form**: Represents a part of the Analysis Input Form where the user inputs parameters for the accessibility model. Some of the parameters include travel time threshold, mode of transportation, and time.
- **Socioeconomic Demographic Criteria Input Form**: Represents a part of the Analysis Input Form where the user inputs criteria for the multicriteria analysis. The values for these criteria are based on the results of the accessibility model calculations. Some of the criteria include age groups, per capita income, and family composition.
- **MCDA Input Form**: Represents a part of the Analysis Input Form where the user inputs weights for the multicriteria analysis.
- **Address Form**: Represents a part of the Analysis Input Form where the user inputs the addresses of the locations of interest for the analysis. These addresses can be used to mark points on the map.
- **Input Object**: Represents the object created from the user's input in the Analysis Input Form. This object is used as input for calculating the best location for the installation of a public facility.
- **Modeling Component**: This component is responsible for analyzing the best location for the installation of a public facility based on the input object provided by the user. It consists of the Accessibility Model Component and the MCDA Component.
- **MCDA Component**: Represents the multicriteria analysis model used to calculate the best location for the installation of a public facility based on the criteria and weights provided by the user.
- **Accessibility Model Component**: This component is responsible for calculating the values of the selected criteria based on the user-configured accessibility model. The value represents the number of people who meet the socioeconomic or demographic criteria and can reach the point of interest according to the accessibility model.

- **Spatial Socioeconomic Database**: Represents the database that contains the spatial and socioeconomic data used in the analysis.
- **Results**: Represents the results of the analysis, which are displayed on the map and in a detailed report table.
- **Ranked List of Locations**: Represents the list of locations ranked by the multicriteria analysis model from best to worst.
- **Report with Detailed Information**: Represents the report table that displays the detailed results of the analysis.
- **Map Layer with Detailed Information**: Represents the layer on the map that displays the detailed results of the analysis.

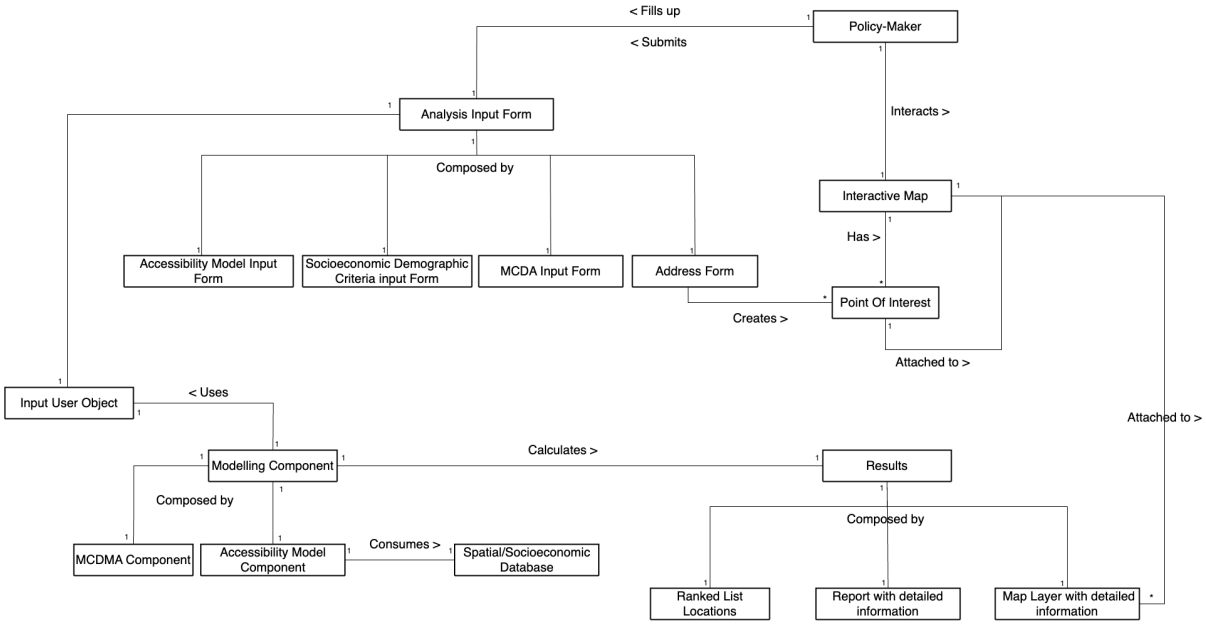

Figure 3.3: Domain model of the solution

3.3.2 Analysis of Requirements

The requirements analysis is crucial when developing a software solution. It helps to identify the needs of the stakeholders and the constraints that the system must meet. A software requirement can be defined as the capability or condition that must be met in order for the software to solve a real-world problem (IEEE Computer Society 2024). These problems can be related to the need to automate parts of a system, correct deficiencies, or implement new features.

The requirements can be divided into two categories: functional and non-functional requirements. Functional requirements describe the system's behavior and the interactions between the system and its users. Non-functional requirements describe the system's quality attributes, such as performance, security, scalability, and usability. Some functional requirements can also fulfill non-functional requirements, such as the requirement to provide a user-friendly interface.

The following sections describe the functional and non-functional requirements of the proposed solution, as well as provide modeling of these requirements using UML diagrams.

Functional Requirements

Functional requirements define the specific features and behaviors that the system should exhibit in order to fulfill the needs and expectations of the users. These requirements outline the actions and functionalities that the system should provide to the users. They are directly visible and perceptible by the users, and they play a crucial role in determining the system's overall behavior and functionality (V. Solutions 2024).

The functional requirements of the proposed solution are as follows:

- **R1:** The system should allow the policymaker to choose different points of interest on an interactive map through a graphical interface.
- **R2:** The system should allow the policymaker to interact with a map through a graphical interface.
- **R3:** The system should allow the policymaker to input parameters for an accessibility model, such as travel times threshold, mode of transportation and time through a form in a graphical interface.
- **R4:** The system should allow the policymaker to choose the criteria and weights for a multicriteria analysis model, such as people by age group, per capita income, or family composition, through a form in a graphical interface.
- **R5:** The system should be able to calculate the values of the selected criteria based on accessibility model configured. The value must be the amount of people that fits in the socioeconomic or demographic criteria that can reach the point of interest accordingly the accessibility model.
- **R6:** The system should be able to calculate the result of a multicriteria analysis of the best point of interest, considering the criterias and weights entered by the policy-maker.
- **R7:** The system should be able to render the results of the multicriteria analysis on an interactive map, applying a layer containing detailed information.
- **R8:** The system should be able to display the results through a table in a simple graphical interface.
- **R9:** The system should provide instructions for filling out the form through a graphical interface.
- **R10:** The system should provide a brief explanation of the results obtained through a graphical interface.

Non-Functional Requirements

Differently from functional requirements, non-functional requirements are not directly related to the system's behavior or features. Instead, they describe the system's quality. Since they are related to software quality, non-functional requirements play an important role to estabilish some criterias to evaluate the system and what technologies or architectures should be used (Chung 2012).

Non-functional requirements are essencial to enable the system to achieve functional requirements, since a poor performance, for example, can make the system not to work as expected.

The non-functional requirements identified for the proposed solution are as follows:

- **R11:** The system should able to validate all data inputs provided by the user, avoiding inconsistencies and showing the errors commited.

- **R12:** The system should calculate and display the results in a timely manner of less than 20 seconds.
- **R13:** The system should provide an interactive map that is easy to use and renders results quickly.
- **R14:** The system should provide user-friendly input forms for policy-makers with a good user experience.
- **R15:** The system should be developed to facilitate changes.
- **R16:** The system should be developed following good software development practices.
- **R17:** The system should support various web browsers (Chrome, Safari, Edge, Firefox, Opera).
- **R18:** The system should be responsive to different devices with varying dimensions.
- **R19:** The system should allow scalability without major difficulties.
- **R20:** The system should provide an interactive map where street and road names can be seen.
- **R21:** The system should be able to, through an interactive map, display all selected types of public facilities.
- **R22:** The system should make use of data with free access and open source.

Based on the aforementioned requirements, the next step is to identify and categorize the non-functional requirements. To accomplish this, the Functionality, Usability, Reliability, Performance, Supportability, and other qualities (FURPS+) model will be used. This model is a classification system for requirements, where the acronym represents categories that are used to define requirements and also represents software quality attributes (Pearce 2024). FURPS+ stands for Functionality, Usability, Reliability, Performance, Supportability, and other requirements.

Functionality covers the system's features and interoperability. Usability measures ease of use, interface design, and accessibility. Reliability ensures consistent performance. Performance assesses efficiency and scalability. Supportability focuses on maintenance and modularity. The "+" includes design constraints, implementation requirements, and regulatory needs for a comprehensive quality evaluation.

The requirements were categorize by the FURPS+ model as follows:

- **Usability:** R11, R20.
- **Supportability:** R15, R16, R17.
- **Performance and Reliability:** R12, R13, R14, R18, R19.

All functional requirements presented in Section 3.3.2 belong to the Functionality category.

3.3.3 Requirements Modeling

The requirements modeling helps to understand the system's behavior and the interactions between the system and its users. This modeling can be done using UML diagrams, such as use case diagrams, system sequence diagrams, and activity diagrams.

UML Diagram

Use-case diagrams in UML are used to model the behavior of a system and capture its requirements. They describe the high-level functions and scope of the system, as well as the interactions between the system and its actors. However, use-case diagrams do not depict the internal operations of the system (IBM 2024).

Based on the requirements described in the previous section, the use-case diagram was created, as shown in Figure 3.4. The diagram shows the main actors and use cases of the system, as well as the interactions between them, as requirement related.

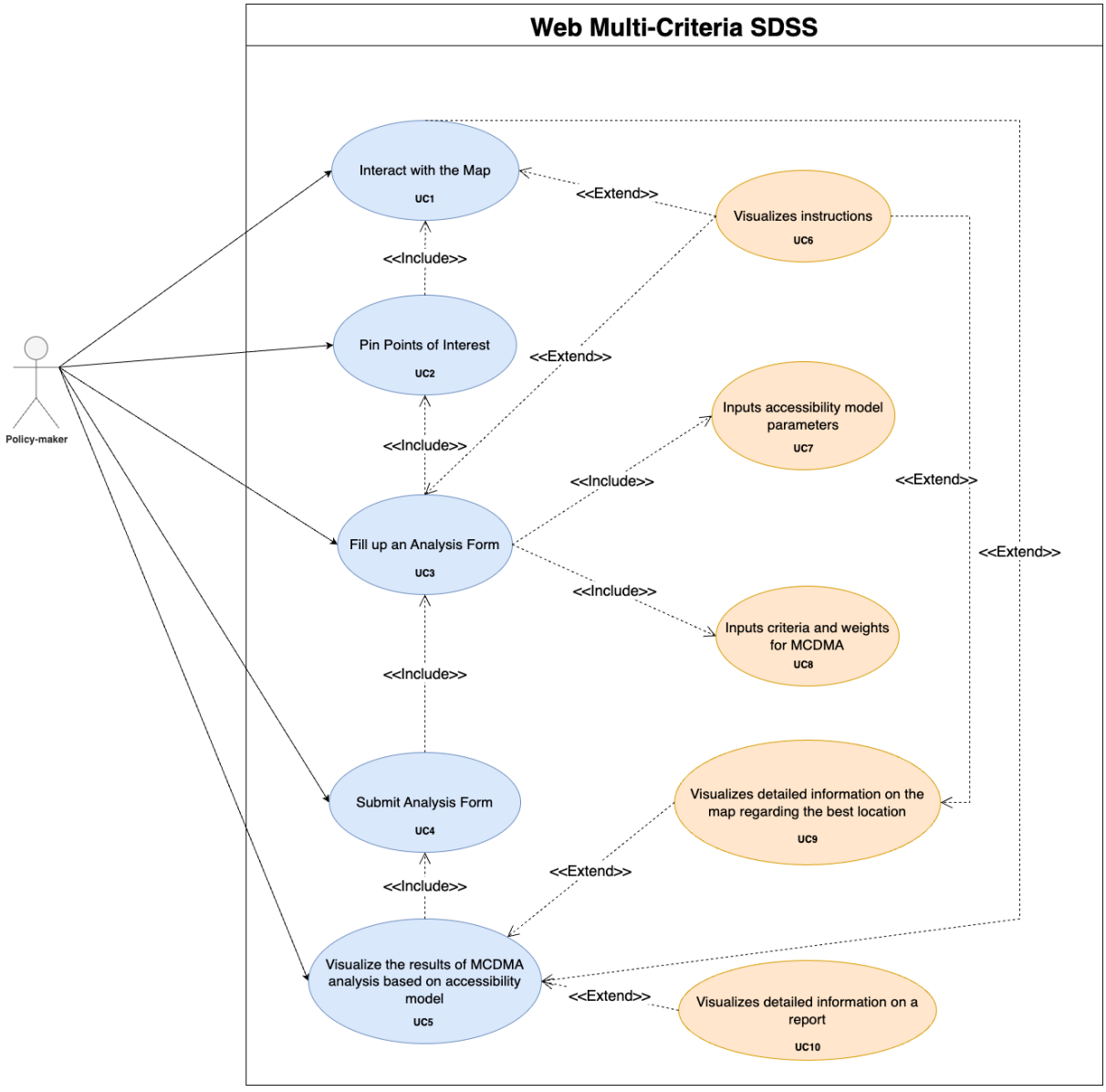

Figure 3.4: Use case diagram of the solution

The use-case diagram illustrates the main interactions between the actors and the system. These interactions involve interacting with the map (UC1), either pinning points of interest (UC2) or visualizing analysis results (UC5) on it, besides filling out (UC3) and submitting an analysis form (UC4). These use cases serve as the foundation and can be included or extended by others as shown in the diagram.

Before dive into the details of the use cases, it is important to understand the difference between include and extend relationships. The extend relationship means the extending use case is optional and conditionally triggered, adding extra behavior to the base use case. In contrast, the include relationship means the included use case is always a part of the base use case, primarily to reuse common actions and simplify complex behaviors. The base use case is incomplete without the included use case, making it mandatory.

For the main flow of the use cases, it can be observed that the use cases UC2, UC3, UC4, and UC5 are directly triggered by the policy-maker. However, there is a chain of dependency between them, as illustrated by the 'include' relationship. For example, it is not possible to pin points of interest on the map without interacting with the map, just as it is not possible to submit the analysis form without filling it up.

The use cases UC6, UC7, UC8, UC9, and UC10 are not directly triggered by the policy-maker, but are either included in the scope or extension paths of the main use cases. Among them, it is worth mentioning that both UC7 and UC8 are related to system inputs, and it is not possible to input specific data without initializing the analysis form fulfillment. UC9 and UC10 are related to the results of the analysis, which are optional and provide different ways to visualize the results.

The author opted for describing the main use cases UC2, UC3, UC4 and UC5, in more details below:

UC2: Pins points of interest on the map

- **Actor:** Policy-maker.
- **Description:** Policy-maker interacts with the map and pins points of interest on it.
- **Trigger:** Policy-maker wants to define points of interests to know the best option to install a public facility.
- **Related Requirements:** R1, R2.
- **Preconditions:** User interacts with the Map.
- **Main Flow:**
  1. Policy-maker pins points of interest on the map.
  2. System receives the pinned points.
  3. System displays the pinned points on the map.
- **Postconditions:** Points of interest are successfully pinned on the map.

UC3: Fills Up Analysis Form

- **Actor:** Policy-maker.
- **Description:** Policy-maker inputs parameters for the analysis.
- **Trigger:** Policy-maker wants to input parameters for the analysis.
- **Related Requirements:** R3, R4.
- **Preconditions:** Form interface for parameter input is accessible.

- **Main Flow:**
  1. Policy-maker accesses the form interface.
  2. Policy-maker inputs parameters for either accessibility model and multi-criteria analysis.
  3. System receives and validates the input parameters.
- **Postconditions:** Inputs parameters are successfully recorded and validated, or input errors are displayed.

UC4: Submits Analysis Form

- **Actor:** Policy-maker.
- **Description:** Policy-maker submits the analysis form, triggering the calculation process.
- **Trigger:** policymaker wants to submit the analysis form and initiate the calculation.
- **Related Requirements:** R5, R6.
- **Preconditions:** UC2, UC3 and UC4 are done.
- **Extend Use Cases:** Policy-maker.
- **Main Flow:**
  1. Policy-maker fills out the analysis form with required parameters.
  2. System receives the submitted form.
  3. System return errors if data is inconsistent, otherwise it triggers the analysis of the best location based on submitted form.
- **Postconditions:** Analysis is successfully calculated and results are displayed on the map and in a report table, or input errors are displayed.

UC5: Visualizes the results of MCDMA analysis based on accessibility model

- **Actor:** Policy-maker.
- **Description:** Policy-maker visualizes the results of the analysis on the map.
- **Trigger:** Policy-maker wants to visualize the results of the analysis.
- **Related Requirements:** R1, R2.
- **Preconditions:** Analysis form is submitted and results are calculated.
- **Main Flow:**
  1. Policy-maker accesses the map.
  2. System displays the results of the analysis on the map.
  3. System displays the results of the analysis in a report table.
- **Postconditions:** Results are successfully displayed on the map and in a report table.

Chapter 4

Value Analysis

This Section focus on define the main concepts around value analysis of a solution and what parts of it will be applied to measure the value delivered to the interested parts, such as the policy-maker and the collaborators. The chapter explores a theoretical dimension, conducting a value analysis of the proposed solution project. This analysis employs an integrated approach, combining the Quality Function Deployment (QFD) and Function Analysis System Technique (FAST) methods.

Before start to analyze the value of the solution, it is important to understand what is a solution value. This concept has different meanings since is related to the perception of who is consuming that solution. It can be related to performance, style, apeal and other many factors. But an interesting aspect is cost. The value of a solution can be represented by the simple equation (N. Solutions 2024):

$$Value = (Performance + Capability)/Cost = Function/Cost \tag{4.1}$$

Value is not solely determined by minimizing cost. In certain cases, the value of a product can be enhanced by increasing its functionality (performance or capability) and cost, as long as the additional function outweighs the additional cost (N. Solutions 2024). In the case of this work, the study of its cost is not the main focus due to the limited scope of implementation. The main focus is to ensure that the capabilities of the solution fulfill most of the functional requirements and maximize the value delivered to the final customer, the policy-maker. In conclusion, the objective of the analysis in this Section is to evaluate the capabilities of the Web-based Spatial Decision Support Systems (WebSDSS) and determine if they are worth the value provided.

Value Analysis (VA) is a systematic and structured approach that aims to enhance the value of products, services, or processes by examining their functions. It involves analyzing each function and determining whether it adds value to the end user or customer (N. Solutions 2024). The goal of Value Analysis is to identify opportunities for cost reduction, performance improvement, or quality enhancement. In the context of this work, Value Analysis can be applied to evaluate the functions of the WebSDSS and determine if they effectively deliver value to the policy-maker and other stakeholders. To accomplish this, two methods will be used: FAST and QFD.

Finally, this chapter will conduct an analysis to evaluate which accessibility model will be used by the module that will calculate the values of the socioeconomic demographic criteria, as stated in Section 1.3 and in more detail in Section 3.2. The choice criteria are quantitative values such as the number of people per age, per capita income, etc., that are defined based on accessibility models that can reach the points of interest. Choosing the right accessibility model can illustrate how the value of a function can be improved when its scope is well defined. Thus, we transition from answering "what is being done?" to "how it is being done?". To accomplish this, the Analytic Hierarchy Process (AHP) method will be used to evaluate the best model to be used in the solution.

4.1 Function Analysis System Technique (FAST)

In this section, the functional analysis of the system will be conducted using the FAST methodology. FAST is a technique that generates a diagram representing the logical relationships between the functions of a project, product, process, or service. It helps answer questions such as "How?", "Why?", and "When?". This diagram serves as a reference to evaluate the proposed solution and ensure it meets the required needs (Bouchereau and Rowlands 2000). In resume, it helps breaking down the operation of the system/product by function, identifying and understanding their roles in the system. This technique is useful as it reminds the development team which functions need to be prioritized and what the relationships between them are.

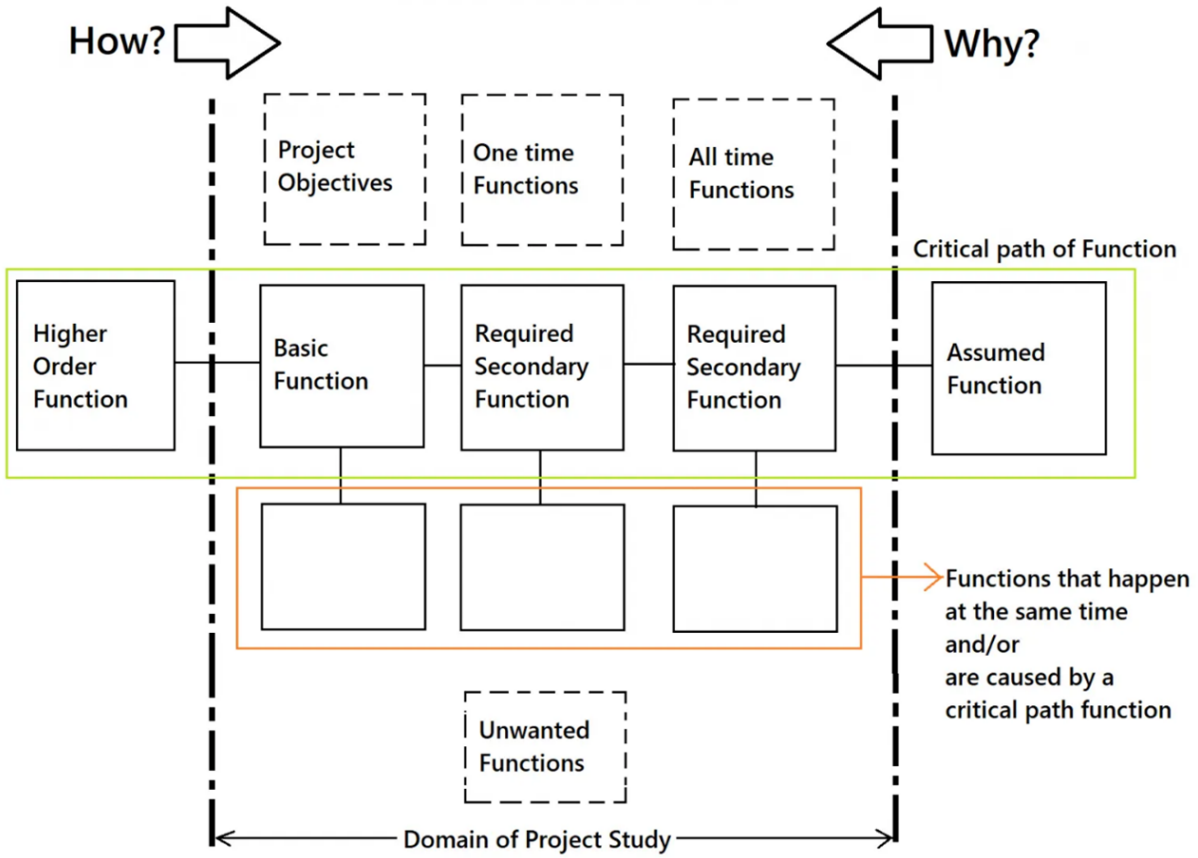

Figure 4.1: FAST diagram template (Journey 2022)

The diagram of the FAST analysis follows a certain structure that helps to visualize the dependency between functions and their roles in the system. Figure 4.1 is a template of a diagram. Finally, in order to understand the step by step to build the diagram, it is necessary to understand the types of functions that can be represented in the diagram. The types of functions are:

- **High Order/Basic Function:** Represents the main function of the system.
- **Secondary/Essencial Functions:** Represents the secondary functions that are derived from the main function.
- **One-time Functions:** Represents the functions that are performed only once.
- **All-time Functions:** Represents the functions that are performed continuously.

- **Unwanted Functions:** Represents the functions that are not desired in the system.
- **Simultaneous Functions:** Represents the functions that are performed simultaneously to the main and secondary functions.
- **Consequential Functions:** Represents the functions that are performed as a consequence of the main and secondary functions.
- **Assumed/External Function:** Represents the functions that are assumed to be performed by the system but are not part of the system or are external to it.

More than simply identifying all functions, it is necessary to understand the semantics of the FAST diagram in order to build it. There are basically two axes through which the diagram can be interpreted: the "How?" and the "Why?". The "How?" represents the reading from left to right, where each function further to the right corresponds to a secondary function that answers the previous one. The "Why?" represents the reading from right to left, where the function further to the left answers the "Why?" question of the immediately previous function. These functions belong to a critical path since they are originated from the high-order function or basic function, which is the most basic function of the system from which all other functions in this main path are derived, the secondary functions. Finally, each of those functions can give rise to other functions that are represented vertically (top to bottom) according to the types listed previously.

Following sections show the steps to build the FAST diagram and their application to the project.

1. Identifying and Categorizing All Functions

The first step in building the FAST diagram is to identify all the functions that are part of the system. This step involves brainstorming and listing all the functions that are necessary for the system to operate.

To determine the functions, an approach of answering questions like "What actions should the system perform?" using verbs and nouns can be followed. Alternatively, the functions can be derived from the system's requirements. By combining both approaches, the following functions and their types have been identified:

- **High Order/Basic Function:** Choose the best location option for a public facility installation.
- **Secondary/Essential Functions:**
  - Classify locations based on socioeconomic and demographic indicators using an accessibility model.
  - Apply a Multi-Criteria Decision-Making Analysis.
  - Define criteria, weights, options, and option criteria values.
  - Submit a form with input parameters for the accessibility model, criteria, weights, and locations.
- **One-time Functions:** Tag points of interests on the map.
- **All-time Functions:**
  - Render an interactive map.
  - Render input forms.
- **Unwanted Functions:**
  - Create confusion by making the results difficult to understand.

  - Create inconsistencies from user's input.
  - Create a confusing form.
- **Simultaneous Functions:**
  - Display Results on an Interactive Map.
  - Visualize the results using an interactive map layer.
  - Calculate the optimal location options based on specified criteria using the AHP-TOPSIS method.
  - Calculate criteria values with a accessibility model.
- **Consequential Functions:**
  - Generate a comprehensive report with clear graphics and explanations.
  - Validate user's inputs.
  - Show form errors to user.
  - Show instructions of how to fill up the form.
- **Assumed/External Function:** Policy-makers provide input parameters.

2. Building the Critical Path

The critical path is the path that connects the high-order function to the secondary functions. It represents the main path of the system and is the most important path to be analyzed. The critical path of the system is as follows:

1. Calculate and display the results of the analysis in a timely manner.
2. Interact with the map.
3. Choose points of interest on the map.
4. Submit the analysis form and trigger the calculation.
5. Input parameters for the accessibility model.
6. Choose criteria and weights for the multicriteria analysis model.
7. Display the results through a table in a simple graphical interface.

3. Placing the Other Functions

Other functions are placed in the FAST diagram according to their types and relationships with the critical path.

4. Checking "How?" and "Why?"

In this step, the diagram is analyzed to check if it answers the "How?" and "Why?" questions.

5. Considering the Results

After completing the initial steps, the FAST diagram model is constructed and analyzed to ensure the accurate placement and categorization of all functions.

After the steps, it can be seen in Figure 4.2 the FAST analysis of the project associated with this document.

5. Review and Refine

As an iterative process, the FAST diagram is reviewed and refined, allowing for the creation of a new diagram that covers a deeper analysis of a specific function, or refactor the current one. This enables a more detailed examination of specific and complex components.

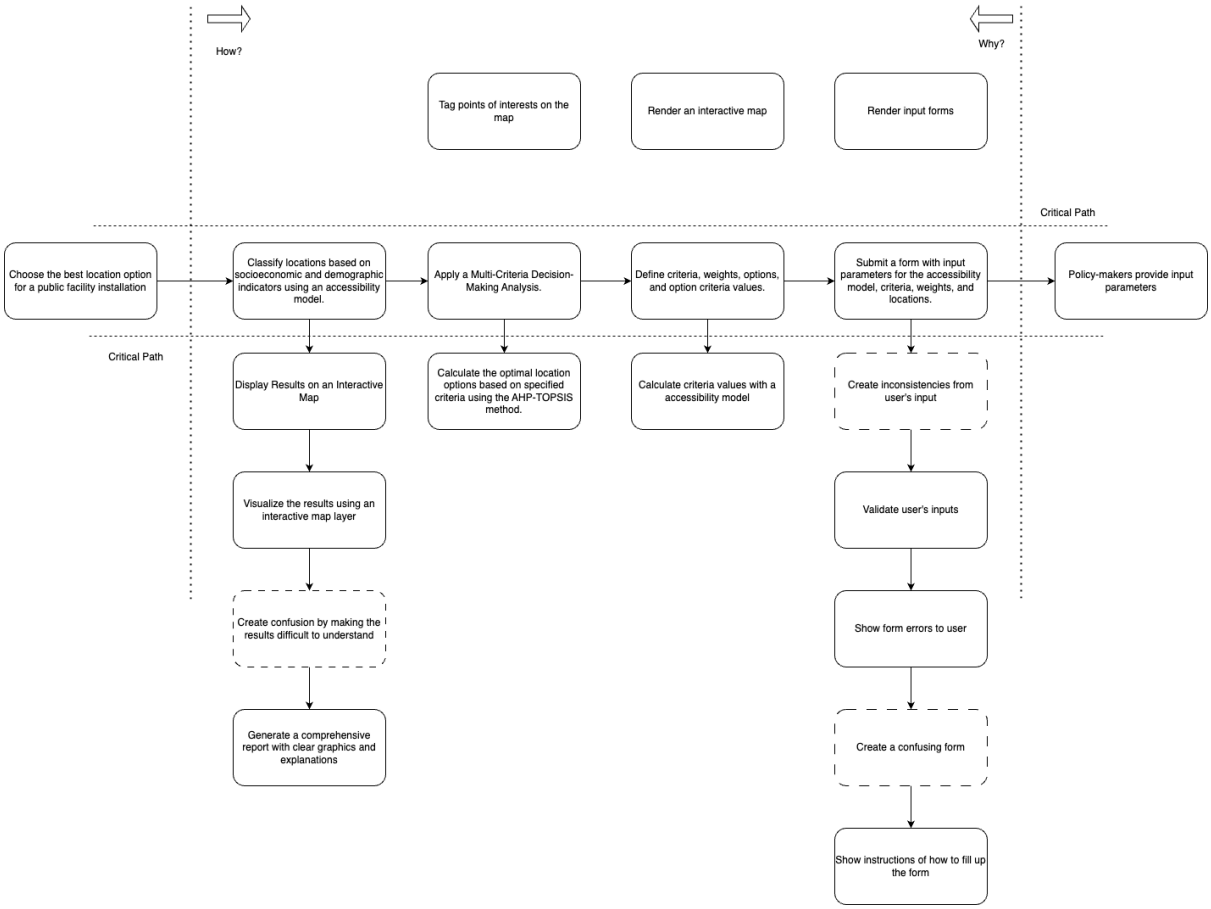

Figure 4.2: FAST diagram of the solution

4.1.1 Conclusions

The following conclusions came from the FAST analysis:

- Critical path reflects the importance of user interactions with the system, such as choosing points of interest on the map, submitting the analysis form, and inputting parameters for the accessibility model and multicriteria analysis model.
- The secondary functions are basically system responses to the user's actions, such as rendering an interactive map, displaying results and input forms, and calculating both criteria values and the optimal location options.
- Since users inputs are crucial for the system to work properly, the consequential functions are important to validate the inputs and show errors and instructions to the user. These functions are direct answers to the unwanted functions, which are mainly related to user confusion and inconsistencies.

- Finally, it's important to note the all-time functions which the main goal is delivery an interactive interface through map and form inputs rendering.

4.2 Quality Function Deployment (QFD)

Once the functions that compose the system have been identified, the next step is to evaluate system's functionalities value from the perspective of the client that in this work are policy-makers. This evaluation can be done using the Quality Function Deployment (QFD) methodology created by Yoji Akao in 1960s. QFD is a structured approach that helps to translate customer needs into specific product or service requirements. It is a tool that can be used to ensure that the functions of a system are aligned with the needs of the users and that the system delivers value to the customers (ReVelle and Moran 1998).

There are 4 key steps to apply the QFD methodology: product planning, part planning, process planning, and production planning. In this work, only the first step, product planning, will be applied. This step involves translating the customer needs into specific product requirements, benchmarking the product's performance against competitors, setting targets for improvements, and identifying the key steps in the development process (GeeksforGeeks 2024). The following steps are more focused on product design, which is not object of study of this section.

The QFD methodology was applied to the project following the steps below:

1. Client Needs

To carry out this work, the author had an undocumented contact through an interview with a professional in the public management field who helped clarify some important points in the context of decision-making in public management, especially regarding the installation of public facilities. Given the lack of documentation and an applied methodology, the information collected will be considered authored by the author of this document, not necessarily reflecting the reality of a real client. In addition to the unofficial information collected from the interview with the client, the author also conducted a literature review to understand the needs that a decision support system should meet as described in Section 2.3.2, as well as analyze academic papers and solutions in the area of decision-making involving accessibility models, as seen respectively in Section 2.6.

The needs of the client were identified and classified according to the following criteria:

- **Easy-to-use and Fluid Interactive Map Interface:** The system should provide an interactive map interface that allows the policy-maker to select points of interest and view the analysis results.
- **Prescriptiveness and Adjustability:** The system should provide a prescriptive tool that allows the policy-maker to input parameters and adjust the analysis for different scenarios.
- **Use of Open-source Data:** The system should utilize open-source data sources in conjunction with the user's parameters for conducting the analysis.
- **User-friendly and Accessible Approach:** The system should be user-friendly and accessible, providing clear instructions and explanations to the policy-maker.
- **Consider Aspects of Accessibility:** The system should employ an accessibility model to calculate the values of the socioeconomic demographic criteria.

2. Importance of Client Needs

Once the needs have been defined, the next step is to classify them by assigning an importance index on a scale of 1 to 5, where 5 corresponds to the highest degree of importance. The classification of the needs is illustrated in the table 4.1

Client Needs	**Importance Rating**	**Relative Importance (%)**
Easy-to-use and Fluid Interactive Map Interface	4	20%
Prescriptiveness and Adjustability	5	25%
Use of Open-source Data	3	15%
User-friendly and Accessible Approach	5	25%
Consider aspects of Accessibility	4	15%

Table 4.1: Client Needs and Importance Ratings

3. Engineering Characteristics

The next step is to define the engineering characteristics that will be used to evaluate the system's functions. These characteristics are associated with the system's ability to meet the client's needs and are used to evaluate the system's performance. The engineering characteristics identified for this work are as follows:

- **Interactiveness:** This characteristic is associated with the system's ability to interact with the user, allowing them to choose points of interest and view the analysis results in a smooth manner with an interface that is suitable for the application context, in this case, spatial data.
- **User-friendly and Accessible Approach:** This characteristic is associated with the ease of use of the platform, ensuring that the user understands what is being done and knows how to interact with it.
- **Prescriptiveness and Adjustability:** This characteristic is associated with the system's ability to be prescriptive and adjustable, allowing the user to input parameters and adjust the analysis for different scenarios.
- **Open-source Data:** This characteristic is associated with the use of open-source data sources for spatial and socioeconomic data.
- **Use of an Accessibility Model:** This characteristic is associated with the use of an accessibility model to calculate the values of the socioeconomic demographic criteria.
- **Use of a MCDMA:** This characteristic is associated with the use of a Multi-Criteria Decision-Making Analysis to classify locations based on socioeconomic and demographic indicators.

4. Correlation Scales

Lastly, in order to relate these characteristics to the previously established client needs, it is necessary to define two scales. One of these scales represents the relationships between the engineering characteristics and the client needs, while the other scale represents the correlation between the different engineering characteristics. These scales are represented in the Figure 4.3

5. Competitors

Regarding the analysis of competing platforms as seen in Section 2.6, only platforms that met the client's needs to some extent were considered, namely the Conveyal, Pereira and Falko Spatial

Decision Support System (SDSS). They were rated on a scale of 1 to 5, with 5 being the best. It is important to note that these ratings are based on the author's own experience using each platform.

6. Relationship Matrix

As a product of previous steps, the matrix is generated and it is represented by the Figure 4.3. This matrix shows the relationship between the client's needs, the engineering characteristics, and the competitors.

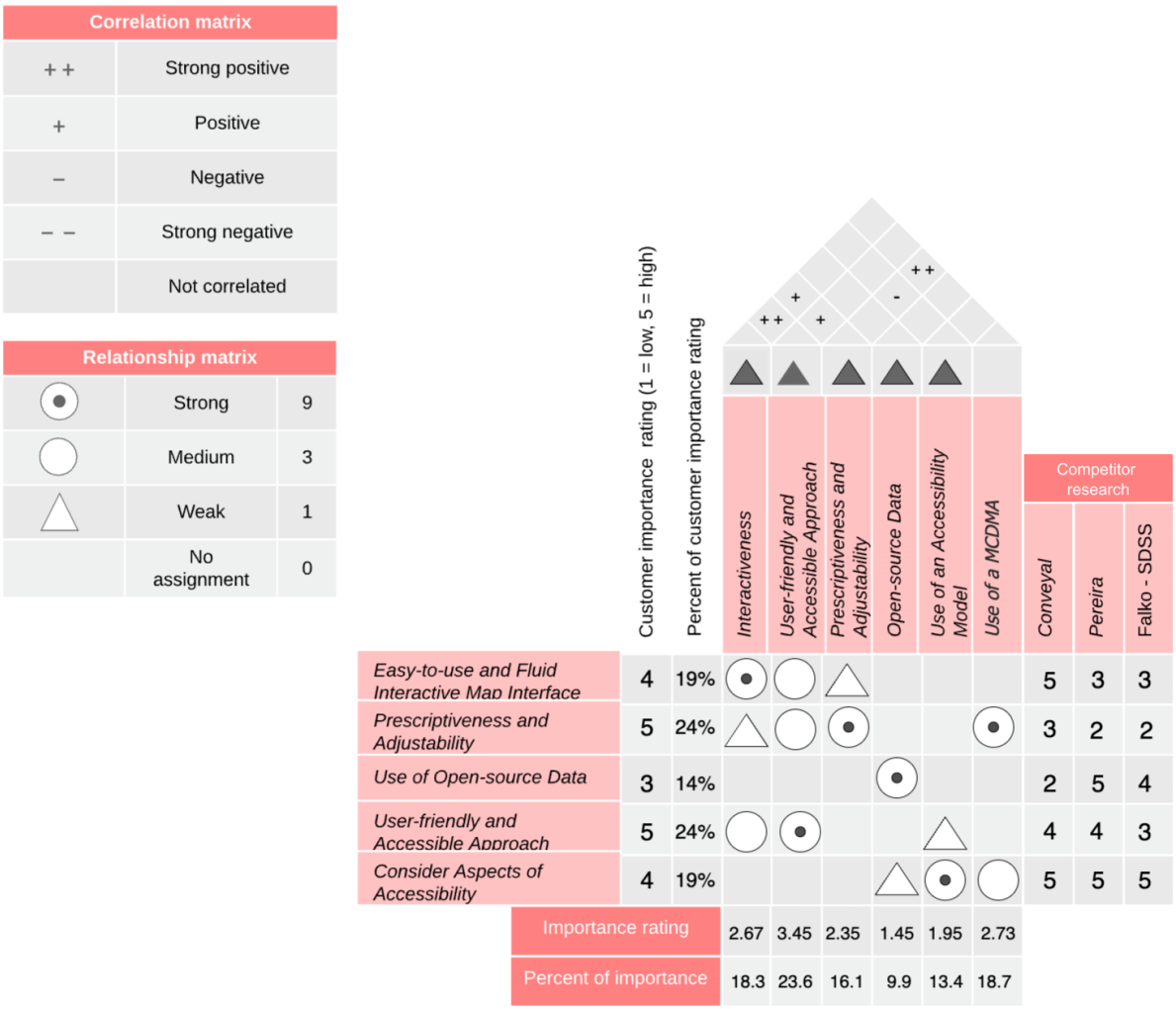

Figure 4.3: QFD matrix of the solution

4.2.1 Conclusions

- As expected, QFD analysis is well aligned to FAST's, once "User-friendly and Accessible Approach' stood out as the most important engineering characteristic which is highly correlated with the critical path seen in the FAST diagram that shows the importance of user inputs and interactions with the system.
- In the second place of importance for a engineering characteristic, there's is "Use of a MCDMA". This is aligned with the main goal of the system which is provide choose the best location option for a public facility installation.

- Other important conclusion is that client need "Prescriptiveness and Adjustability' depend of almost all engineering characteristics. This enphasizes the problematic described on Section1.2 regarding political decision-making context, so an approach less impositive and more prescritive contributes to system acceptance.
- Although "Use of an Accessibility Model' and "Open-source Data' have a lower importance rating than engineering characteristics related to user experience, they are still important to the system's performance and value delivery.

4.3 AHP best accessibility measure

Finalizing the value analysis, the last step is to evaluate which accessibility model will be used by the module that will calculate the values of the socioeconomic demographic criteria. As perceveid from the FAST diagram and QFD model, calculate accessibility of the point of interest is essencial to deliver value the basic function of the system.

The selection of the accessibility model will be done using the AHP method. AHP is a structured technique developed by Thomas L. Saaty in the 1970s, which helps in organizing and analyzing complex decisions. It is based on mathematics and psychology and has been widely studied and refined over the years. It allows decision-makers to prioritize criteria and alternatives when faced with multiple options, breaking down complex decisions into a hierarchy of criteria and alternatives (Saaty 2008).

Following sections will resume the steps and how they will be applied to choose the best accessibility model for the solution.

Hierarchy Creation

The first step is to define the main objective of the analysis, alternatives, and criteria (see Figure 4.4).

It is important to note that the use of an accessibility model in the context of this work is to show how the system can be used to calculate accessibility based on the data sources and the model chosen. Therefore it's not object of study to chose the best model to calculate accessibility, but the one that can easily showcase system capabilities.

The criteria that will be considered in the evaluation are data availability, computational complexity, interpretability, and relevance to stakeholder objectives. The alternatives are the accessibility models such as the gravity model, the cumulative opportunity model, and the floating catchment area model.

Alternatives and criteria are explained below:

- **A1- Gravity Model:** The Gravity Model is a spatial analysis technique used to estimate the flow of people, goods, or information between locations based on their relative attractiveness and the distance between them (K. Geurs and Wee 2004). It is derived from the principle of gravitational attraction, where larger and closer destinations exert greater attraction. This model is exemplify in Section 2.2.4.
- **A2- Cumulative Opportunity Model:** The Cumulative Opportunity Model assesses accessibility by aggregating the cumulative availability of opportunities within a given catchment area around each location (K. Geurs and Wee 2004). It emphasizes the total accessibility to opportunities, such as jobs, services, or amenities, within a specified distance threshold. This model assumes that individuals or goods are more likely to travel to locations with higher

cumulative opportunities, regardless of administrative boundaries, and accounts for the spatial distribution of demand and supply.

- **A3- Floating Catchment Area Model:** The Floating Catchment Area (FCA) model is an extension of the Cumulative Opportunity Model that addresses the spatial distribution of demand and supply (Delamater, Shortridge, and Kilcoyne 2019). It defines catchment areas around each location based on travel time or distance thresholds, rather than administrative boundaries. The FCA Model calculates accessibility by aggregating the supply (e.g., number of facilities) and demand (e.g., population) within each catchment area, considering the accessibility decay with distance. This model is commonly used in analyzing access to healthcare services, retail amenities, and other spatially distributed facilities.
- **C1- Data Availability:** Consider the availability and reliability of data required to calculate each accessibility measure, including socioeconomic indicators, transportation networks, and demographic data.
- **C2- Computational Complexity:** Evaluate the computational complexity of implementing each accessibility measure, considering factors such as the number of variables, spatial scale, and computational resources required.
- **C3- Interpretability:** Assess how easily stakeholders can interpret and understand the results provided by each accessibility measure, considering the clarity of visualization and presentation.
- **C4- Relevance to Stakeholder Objectives:** Consider the extent to which each accessibility measure aligns with the goals and objectives of decision-makers, policymakers, and other stakeholders involved in the decision-making process.

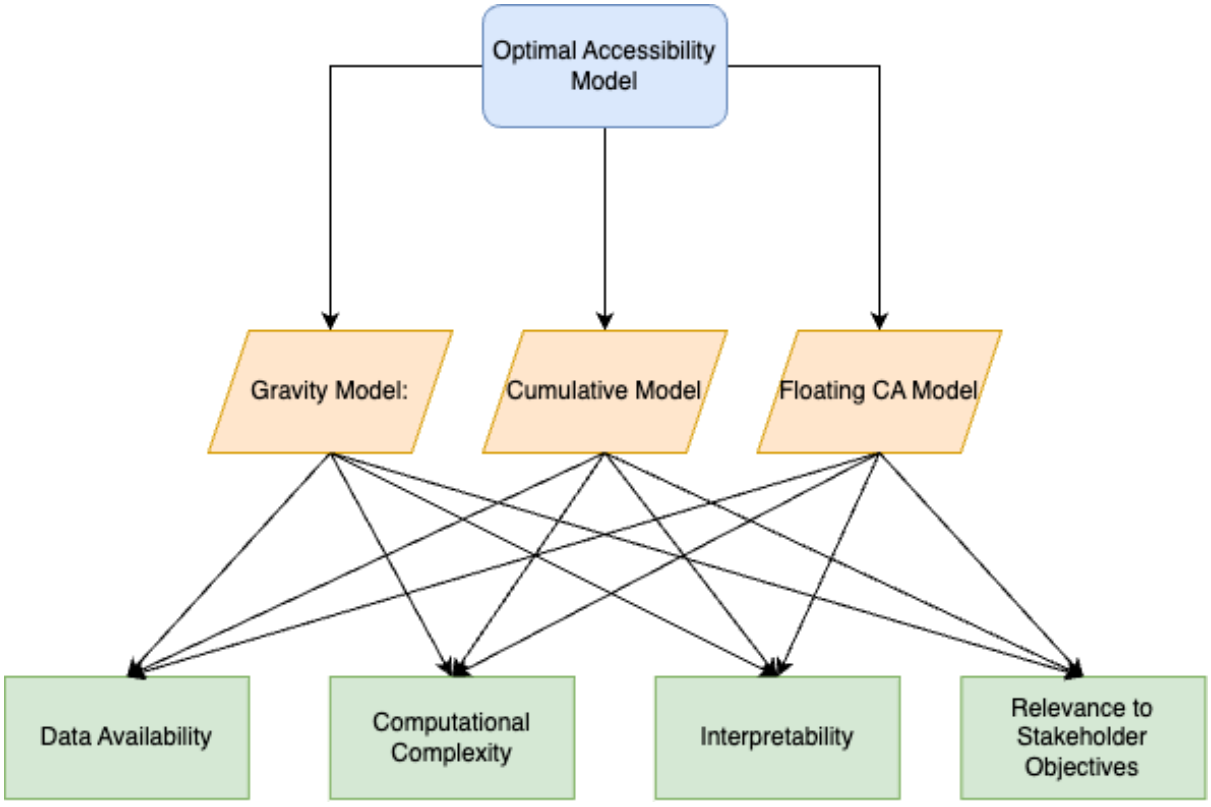

Figure 4.4: AHP Hierarchy of the solution

Pairwise Comparisons

In this step, pairwise comparisons are conducted to determine the relative importance of criteria and alternatives. This involves comparing each criterion or alternative with every other one and assigning a numerical value to represent their relative importance. The Saaty scale (Saaty 2008) ranges from 1 to 9, with 1 indicating equal importance and 9 indicating extreme importance.

To normalize the pairwise comparisons, each value in a column is divided by the sum of all values in that column. The normalized values are then used to calculate the relative priority of each criterion and alternative, which is the average of the normalized values in each row.

The pairwise comparisons of criteria and alternatives and its normalized version are shown in Tables 4.2 and 4.11, respectively.

Table 4.2: Pairwise Comparisons of Criteria

Criteria	C1	C2	C3	C4
C1	1	5	3	1/2
C2	1/5	1	1/2	1/7
C3	1/3	2	1	1/3
C4	2	7	3	1

Table 4.3: Normalized Pairwise Comparisons of Criteria with Relative Priority

Criteria	C1	C2	C3	C4	Relative Priority %
C1	0.2835	0.3333	0.4000	0.2538	31.77
C2	0.0567	0.0667	0.0667	0.0711	6.53
C3	0.0935	0.1333	0.1333	0.1675	13.19
C4	0.5663	0.4667	0.4000	0.5076	48.52

Applying the same normalization process to the pairwise comparisons of alternatives based on a specific criteria, the following results were obtained:

For the criteria Data Availability (C1): all three alternatives are almost equal since the data needed as regarding population and space are the same. It can be discussed that the Floating Catchment Area Model has a slight disavantage over the other two alternatives, since the decay function needs to be determined.

Table 4.4: Pairwise Comparisons of Alternatives for Criteria Data Availability

Alternatives	A1	A2	A3
A1	1	1	2
A2	1	1	2
A3	1/2	1/2	1

Table 4.5: Normalized Pairwise Comparisons of Alternatives for Criteria Data Availability

Criteria	A1	A2	A3	Relative Priority %
A1	0.4	0.4	0.8	40
A2	0.4	0.4	0.8	40
A3	0.2	0.2	0.4	20

For the criteria Computational Complexity (C2): the Cumulative Opportunity Model has a slight advantage over the other two alternatives for the use case of this work, which is to define a travel

cost and determine which points are reachable based on that. The Gravity Model has a disadvantage due to the need to define all the points that can be reached arbitrarily, and the Floating Catchment Area Model has a disadvantage due to the need to determine the decay function. The points were given following a logic where as the complexity increases, the value decreases.

Table 4.6: Pairwise Comparisons of Alternatives for Criteria Computational Complexity

Alternatives	A1	A2	A3
A1	1	1/2	2
A2	2	1	3
A3	1/2	1/3	1

Table 4.7: Normalized Pairwise Comparisons of Alternatives for Criteria Computational Complexity

Criteria	A1	A2	A3	Relative Priority %
A1	0.286	0.272	0.333	29.70
A2	0.571	0.546	0.5	53.90
A3	0.143	0.182	0.166	16.30

For the criteria Interpretability (C3): Interpretability is a key factor in decision-making, as stakeholders need to understand the results provided by the accessibility model. The Cumulative Opportunity Model and the Gravity Model have a slight advantage over the Floating Catchment Area Model in this criteria.

Table 4.8: Pairwise Comparisons of Alternatives for Criteria Interpretability

Alternatives	A1	A2	A3
A1	1	1	4
A2	1	1	4
A3	1/4	1/4	1

Table 4.9: Normalized Pairwise Comparisons of Alternatives for Interpretability

Criteria	A1	A2	A3	Relative Priority %
A1	0.444	0.444	0.444	44.45
A2	0.444	0.444	0.444	44.45
A3	0.111	0.111	0.111	11.11

For the criteria Relevance to Stakeholder Objectives (C4): To assign the points for this criteria, the author considered that the Cumulative Opportunity Model is the most relevant to the stakeholder objectives. This model is widely used for the use case of this work as it focuses on the distribution of supply and demand, rather than determining the impact of distances on opportunities or considering all points that can be reached arbitrarily.

Consistency Check

After completing pairwise comparisons, consistency checks are performed to ensure the reliability of the comparisons. If inconsistencies are detected, adjustments may be made to the comparisons until consistency is achieved. The evaluation of a AHP method is based on the axiom of transivity and that the decision maker is racional, consistent and logical. For example, if A is more important than criteria B and B is more than C, then A is more important than C. The consistency can

Table 4.10: Pairwise Comparisons of Alternatives for Criteria Relevance to Stakeholder Objectives

Alternatives	A1	A2	A3
A1	1	1/2	3
A2	2	1	4
A3	1/3	1/4	1

Table 4.11: Normalized Pairwise Comparisons of Alternatives for Relevance to Stakeholder Objectives

Criteria	A1	A2	A3	Relative Priority %
A1	0.3	0.286	0.375	32.00
A2	0.6	0.571	0.5	55.80
A3	0.1	0.143	0.125	12.20

be determined by the formula, where the Consistency Ratio (CR) should be less than 0.1 to the analysis be considered consistent:

$$[CR = \frac{CI}{RI}]$$

Where:

- CI is the Consistency Index and RI calculated by the formula:

$$CI = \frac{\lambda_{\max} - n}{n - 1}$$

  Which $\lambda_{\max}$ is the Principal Eigenvalue and n is the order of the matrix.

- RI Random Index is a measure of the expected consistency for a given number of criteria or alternatives (see Table 4.12). For example, for 4 criteria or alternatives, the Random Index is 0.9.

To calculate CI, it is necessary to find the principal eigenvalue λ, determined by the formula:

$$\mathrm{Aw} = \lambda \times \mathrm{w}$$

Where:

- A is the pairwise comparison matrix.
- λ is the principal eigenvalue of matrix A.
- w is the eigenvector associated with λ, representing the relative weights of the criteria or alternatives.

By performing matrix multiplication and then comparing it to the given vector w, the results are:

$$\mathrm{Aw} = \begin{bmatrix} 1 & 5 & 3 & \frac{1}{2} \\ \frac{1}{5} & 1 & \frac{1}{2} & \frac{1}{7} \\ \frac{1}{3} & 2 & 1 & \frac{1}{3} \\ 2 & 7 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 31.77 \\ 6.53 \\ 13.19 \\ 48.52 \end{bmatrix}$$

$$= \begin{bmatrix} 31.77 + 32.65 + 39.57 + 24.26 \\ 6.35 + 6.53 + 6.60 + 6.93 \\ 10.59 + 13.06 + 13.19 + 16.17 \\ 63.54 + 45.71 + 48.52 + 48.52 \end{bmatrix}$$

$$= \begin{bmatrix} 127.25 \\ 26.41 \\ 53.01 \\ 206.29 \end{bmatrix}$$

Finally, the principal eigenvalue λ is given by:

$$\lambda = \frac{\frac{127.25}{31.77} + \frac{26.41}{6.53} + \frac{53.01}{13.19} + \frac{206.29}{48.52}}{4} = 4.08$$

Applying this value to CI formula:

$$CI = \frac{\lambda_{max} - n}{n - 1} = \frac{4.08 - 4}{4 - 1} = 0.0267$$

Now, calculate the Random Index (RI) using the table 4.12:

Table 4.12: Random Index for AHP

n	Random Index (RI)
1	0.00
2	0.00
3	0.58
4	0.90
5	1.12
6	1.24
7	1.32
8	1.41
9	1.45
10	1.49

For $n = 4$, the Random Index is 0.90. Therefore, the Consistency Ratio (CR) is calculated as:

$$CR = \frac{CI}{RI} = \frac{0.0267}{0.90} = 0.0297$$

Since the Consistency Ratio (CR) is less than 0.1, the pairwise comparisons are considered consistent.

Priority Calculation

Once pairwise comparisons are completed and consistency is ensured, priority weights are calculated for each criterion or alternative. These weights reflect their relative importance in achieving the overall goal. The formula for calculating the priority weights is as follows:

$$S \times w = Sw$$

Where:

$$\begin{bmatrix} 0.40 & 0.2970 & 0.4445 & 0.32 \\ 0.40 & 0.5390 & 0.4445 & 0.5580 \\ 0.20 & 0.1630 & 0.1111 & 0.1220 \end{bmatrix} \times \begin{bmatrix} 0.3177 \\ 0.0653 \\ 0.1319 \\ 0.4852 \end{bmatrix}$$

After performing the multiplication, the results are:

$$\begin{bmatrix} (0.40 \times 0.3177) + (0.2970 \times 0.0653) + (0.4445 \times 0.1319) + (0.32 \times 0.4852) = 0.476 \\ (0.40 \times 0.3177) + (0.5390 \times 0.0653) + (0.4445 \times 0.1319) + (0.5580 \times 0.4852) = 0.492 \\ (0.20 \times 0.3177) + (0.1630 \times 0.0653) + (0.1111 \times 0.1319) + (0.1220 \times 0.4852) = 0.147 \end{bmatrix}$$

The matrix shown in Table 4.13 relates all alternatives to all criteria and the overall priority of each alternative.

Table 4.13: Pairwise Comparisons of Alternatives for Criteria Relevance to Stakeholder Objectives

Alternatives	C1	C2	C3	C4	Relative Priority %
A1	0.40	0.2970	0.4445	0.32	47.60
A2	0.40	0.5390	0.4445	0.5580	49.20
A3	0.20	0.1630	0.1111	0.1220	14.70
Relative Priority	0.3177	0.653	0.1319	48.52	

Based on the results obtained, it can be inferred that the best alternative is A2, the Cumulative Opportunity Model. This alternative is associated with the use of a model that is easy to interpret, has low complexity, and is highly relevant to the platform user. This statement is supported by the higher relative priority value of 49.20%.

Chapter 5

Design

Previous chapters have conducted requirements elicitation and analysis, allowing for the definition and categorization of all solution requirements. Additionally, a value analysis was performed to assess whether the proposed solution delivers value to the stakeholders. The next step is to design the solution, ensuring that project development proceeds efficiently and effectively, meeting the requirements and enabling the generation of the expected value.

In this chapter, the solution design will focus on defining the software architecture and creating artifacts capable of describing different components of the system at different levels of abstraction, as well as how they interact with each other. The chapter is structured into three main sections: Decision Support System (DSS) design overview, software architecture and spatial data design.

Firstly, the DSS design overview section revisits the DSS architectural concepts, describing how they serve as a baseline for the solution design. Secondly, the software architecture section is divided into five subsections, each representing a view of the 4+1 architectural model. Finally, the spatial data design section details how the spatial data is modeled. As a result, this chapter aims to provide a detailed description of the solution design, which will serve as a reference for the development and implementation phases.

5.1 DSS Design Overview

In Section 2.3.2, the structural concepts of Spatial Decision Support System (SDSS) and how these systems should be organized to meet the requirements for effective decision making were discussed. The Data Dialog Model (DDM) model presented will be used as a guideline for this system design, encompassing there referred three main capabilities: dialog, data, and modeling. Aditionally, it will be also use the concept of the three levels of technology: specifc DSS, DSS generator and DSS tools. These two concepts serve as the foundations for designing the proposed solution.

This project aims to create a two-level technology system, meaning a system that integrates DSS and web tools to generate Multi-Criteria Web-based Spatial Decision Support System (MC-WebSDSS). This generator employs various web technologies and components, which may be either built-in or sourced from external services (refer to Figure 5.1). These tools include programming languages, code libraries, and APIs. One of the goals of the proposed system design is to achieve the two levels of technology as an SDSS.

Structuring the proposed solution within the paradigm of an SDSS, the proposed system is built upon the fundamental principles of the DDM model. Below is a description of how each capability is designed within the proposed solution:

- **Dialog Component:** For the proposed system, the dialog component is a web interface that allows the user to input parameters, view results and interact with an interactive map.

- **Data Component:** The data component manages the data used by the system. This includes data from the census records, public transportation, and other sources that are used to calculate accessibility and socioeconomic indicators. This data is modelled and then stored in a database to be accessed by the system as needed.
- **Modelling Component:** The modelling component performs the calculations and analysis required by the system. This involves calculating accessibility values, running multicriteria analysis, and generating results based on the user's input. The modelling component uses the data from the data component to perform these calculations.

The Figure 5.1 shows an overview of how the project is structured within the composition of an SDSS, including the tools and modules that comprise it.

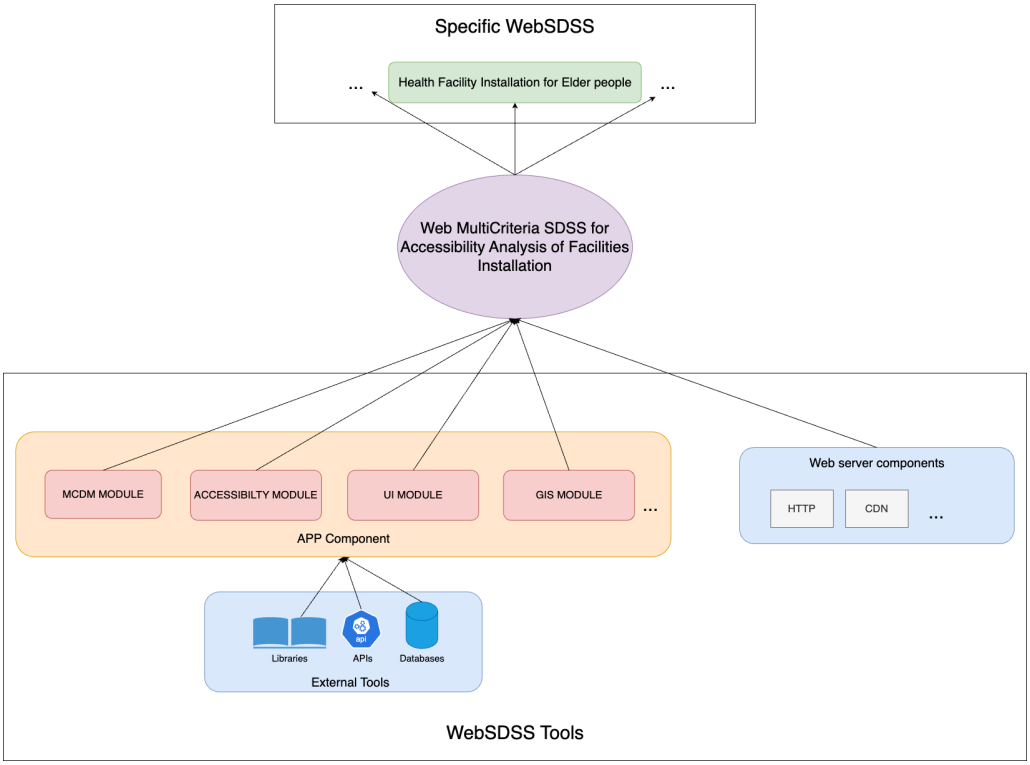

Figure 5.1: Extension of Sprague's three-level framework applied to WebSDSS development of the proposed solution. Adapted from Rinner 2003

This architecture enables the system to behavior as an Web-based Spatial Decision Support Systems (WebSDSS) generator. The idea is that they can be rearranged to generate different specific WebSDSS applications. For example, different multicriteria methods or accessibility models can be combined with socioeconomic indicators to create a specific WebSDSS for different public facilities.

By offering an online interface with an interactive map, policy-makers can autonomously adjust parameters and obtain diverse analysis results with minimal software dependencies, thanks to the web environment. The application does not require a deep level of understanding of the mathematical aspects of the models used. The user can focus on the results and the decision-making process, rather than the technical details of the models. This approach enhances transparency and the quality of analysis, while also reducing the burden on planners to manage various data sources and develop models to accommodate all types of data in the accessibility and socioeconomic context. Therefore, it is crucial to describe the data sources for the project and the kind of data they provide.

5.2 Software Architecture

Software architecture deals with the design and implementation of the high-level structure of the software, including elements, forms, and rationale/constraints. It addresses functionality, performance, and non-functional requirements such as reliability, scalability, portability, and availability (Kruchten 1995). The architecture of the proposed system can be described using a model called 4+1, which consists of five main views (Kruchten 1995):

- **Logical View:** Represents the object model of the design, focusing on the functionalities that the system provides to end-users. UML diagrams such as class and state diagrams can be used to represent this view.
- **Development View:** This view, also known as the implementation view, details the system from the developer's point of view, focusing on software management. UML diagrams representing this view include component and package diagrams.
- **Physical View:** Also known as the deployment view, describes the system from the perspective of a systems engineer. It focuses on the topology of software components in the physical layer, including physical connections and protocols used between them. The deployment diagram is used to represent this view.
- **Process View:** Deals with the dynamic aspects of the system, explaining its processes and focusing on the system's behavior at runtime. UML diagrams used to represent this view are sequence, communication, and activity diagrams.
- **Scenarios:** Although redundant considering the previous ones, serves as a driver to discover architectural elements during the architecture design and plays the role of validation and illustration after the project's completion.

The Figure 5.2 illustrates how those views are related to each other and what they represent.

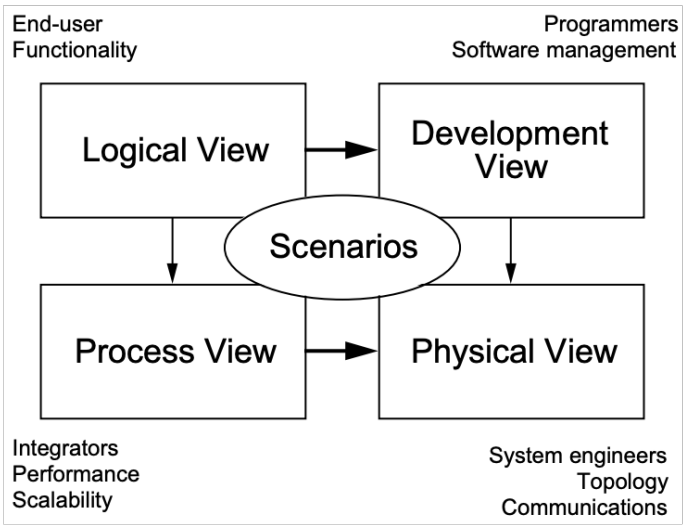

Figure 5.2: 4+1 Architectural View Model (Kruchten 1995)

The following sections will describe the software architecture of the proposed solution, focusing on some of the views of the 4+1 architectural model. However, the logical and scenarios views have already been addressed in Sections 3.3.1 and 3.3.3, where the domain model and the use

case diagram were defined, respectively. Additionally, the author has chosen to present possible variations of the architectural views that were considered during the design phase but were not implemented in the final solution. This approach aims to provide a broader understanding of the design process and the decisions made during this phase.

5.2.1 Development View

This view is crucial for understanding the system's structure and the interactions between components. Its main goal is to provide a comprehensive description of the components and their relationships, serving as a reference for the development phase.

It is important to note, for this view, the decision was made to represent the components without specifying the technologies used. Instead, the focus is on the features and architecture that these components should provide in order to fulfill the proposed design. In the implementation phase, the technologies will be defined, and the components will be further detailed.

A final consideration is that the author considered two phases of development for the proposed solution: an initial phase and an advanced phase. The initial phase represents the first version of the system, while the advanced phase represents a more complex and scalable version of the system. The development view will be presented for both phases, highlighting the differences between them.

Initial Phase

The initial phase of the design was intended to follow a monolithic architecture, where the backend and frontend are combined in a single component. This approach was chosen to simplify the development process and reduce system complexity. However, it also has some limitations, such as reduced scalability and flexibility, since the backend and frontend applications would be tightly coupled.

The diagram in Figure 5.3 illustrates a single component that constitutes the application—the WebSDSS app—along with an HTTP server that manages web access, a spatial GeoJSON files component containing the spatial data, and, finally, an external component, which is the user's web browser.

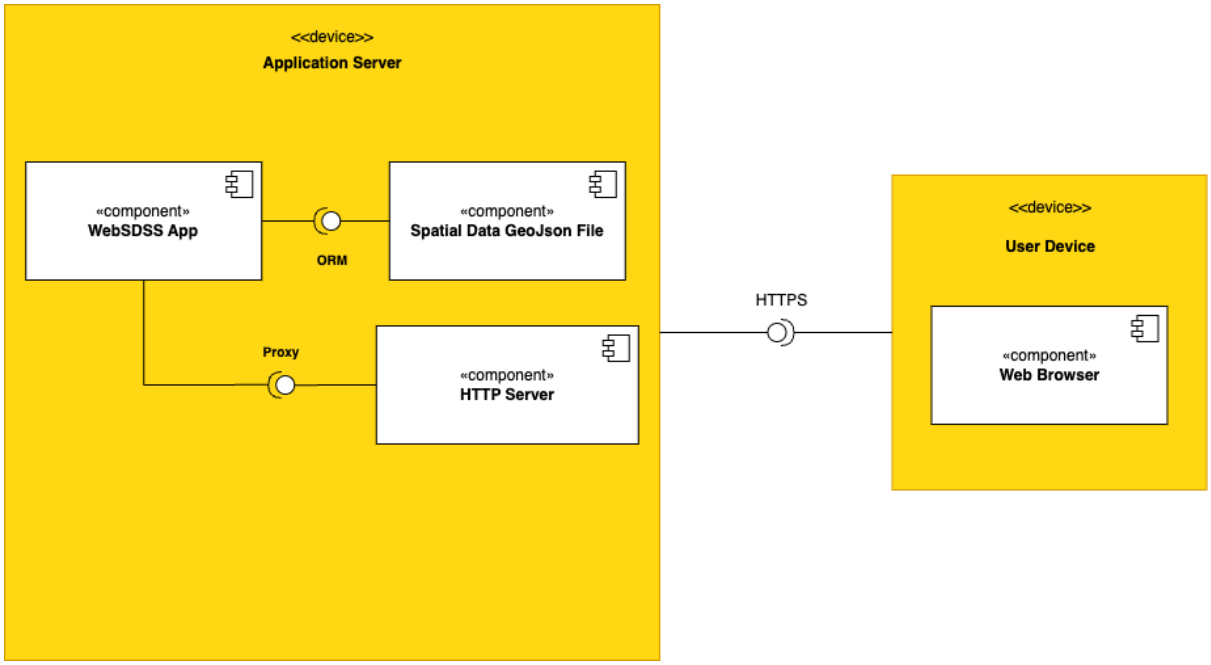

Figure 5.3: Development view of Initial Phase

The disposition of the internal packages within the WebSDSS app component is shown in Figure 5.4. It illustrates the structure of the main packages that comprise the web application. This organization is based on the structure of a Next JS application (Vercel, Inc. 2024).

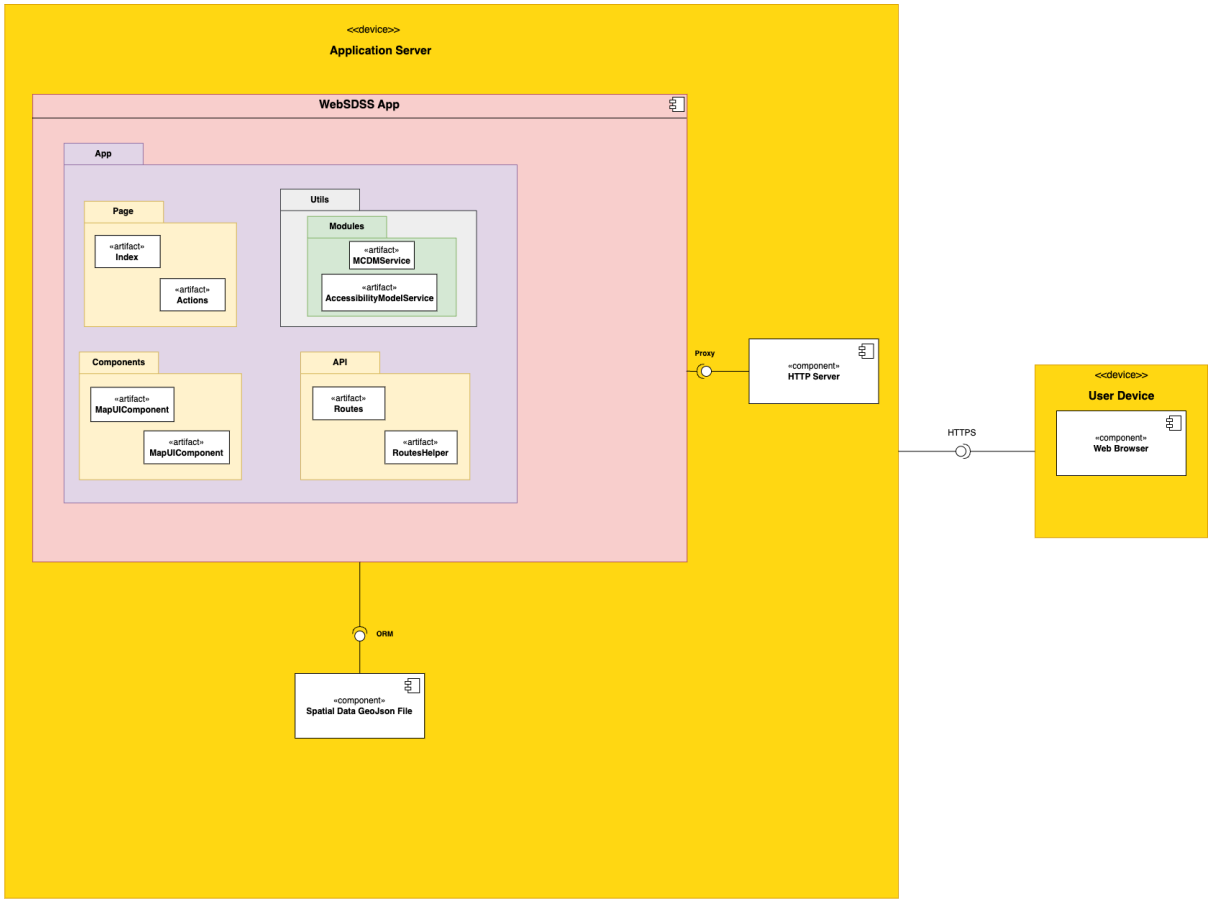

Figure 5.4: Development view of Initial Phase packages

The main packages are the 'pages', 'components', and 'utils/modules', which are responsible for rendering the web pages, defining reusable react ui components, and managing the business logic of the application, respectively.

Advanced Phase

In an advanced stage of the project, the author considered implementing the system based on the diagram shown in Figure 5.5. The diagram illustrates four components that make up the application, along with an external component, which is the user's web browser. The internal components within the application server represent the elements of the system that are developed or configured within the scope of the project. These components are as follows:

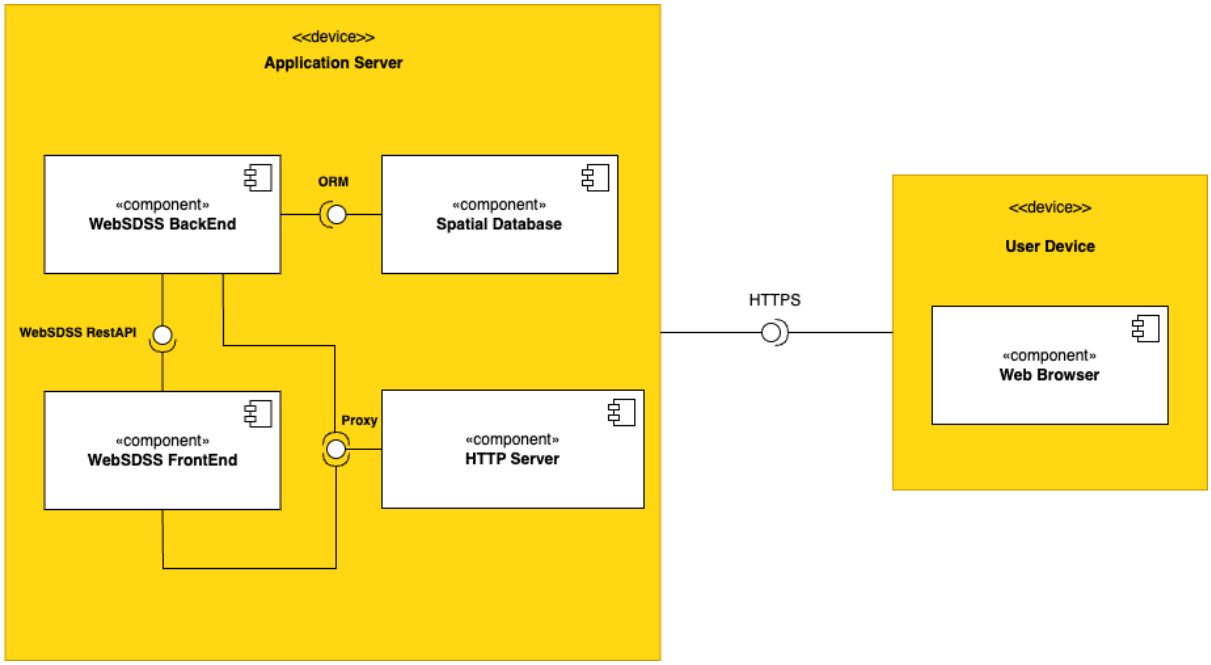

Figure 5.5: Development view of Advanced Phase

- **WebSDSS Backend:** This component is responsible for handling the business logic of the application, which includes calculating accessibility indicators and applying multi-criteria decision analysis. It also manages the spatial data stored in the spatial database. As shown in Figure 5.1, this component is part of the WebSDSS tools. However, in the implementation of this work, it serves as the main component of the system, implementing most of the functionalities through built-in services. In a different analysis, it can also work as the interface that integrates external APIs as services. It exposes a RESTful API to the frontend application and other clients using web technologies such as the HTTP communication protocol and JSON content.
- **WebSDSS Frontend:** This component provides a user-friendly interface for the system, allowing users to interact with the application. It is developed using JavaScript frameworks and libraries to render an interactive map, input forms and display the results of the analysis. The frontend application consumes the services provided by the backend application through the RESTful Application Programming Interface (API).
- **Spatial Database:** This component is responsible for storing and managing spatial data used by the system. It provides a structured and efficient way to store geographic information, such as census data, public transportation routes, and road networks. The spatial database is accessed by the backend application to perform accessibility calculations and analysis.
- **HTTP Server:** This component is responsible for managing user requests and serving the frontend application to the user's web browser. It acts as an intermediary between the frontend and backend applications, handling communication between the two components. The HTTP server is responsible for routing requests to the appropriate endpoints and serving static files to the user.

Drilling down into the components, the diagram of the initial phase of the project is shown in Figure 5.6. It illustrates the structure of the main components that comprise the web application and their internal packages.

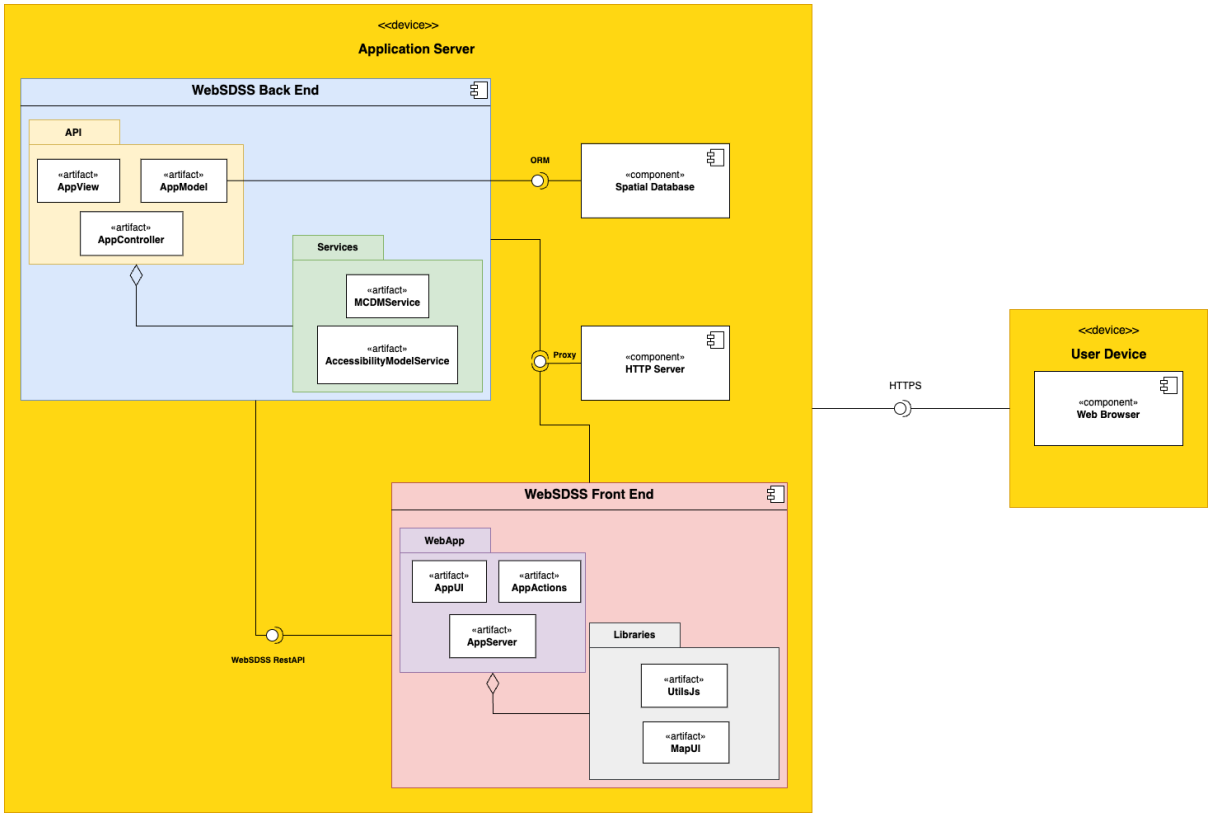

Figure 5.6: Development view of Advanced Phase packages

Specifically for the internal organization of WebSDSS Backend component, the author opted for a framework web application architecture, which is based on the Model-View-Controller (MVC) design pattern. This choice was strongly based on the following characteristics:

- **Separation of Concerns:** MVC divides an application into three interconnected components, allowing for better organization and management of code. The model handles data and business logic, the view handles user interface elements, and the controller acts as an intermediary, processing user requests and updating the model or view accordingly.
- **Modularity:** MVC promotes modularity by breaking down an application into smaller, more manageable components. This makes it easier to develop, test, and maintain code, as changes made to one component are less likely to impact other parts of the application.
- **Scalability:** MVC architecture provides a scalable foundation for web applications. As the application grows in complexity, developers can extend or modify individual components (models, views, or controllers) without having to overhaul the entire system.
- **Support for Frontend-Backend Separation:** MVC inherently supports the separation of frontend (view) and backend (model and controller) components. This allows frontend developers to focus on designing user interfaces, while backend developers work on implementing business logic and data handling.

These characteristics make the MVC design pattern a suitable choice for the proposed solution. It provides a structured and organized way to develop web applications and allows for the rearrangement of components as needed. This enables the system to evolve and adapt to changing requirements, as well as attach new functionalities either as new components or as extensions of existing ones.

In the context of this project, the main services are the 'MCDM' and 'AccessibilityModel'. However, it is possible to create new services in the future that provide different urban planning models for generating new analyses. Therefore, the initial monolithic architecture can be extended to a microservices architecture, as described in Figure 5.7.

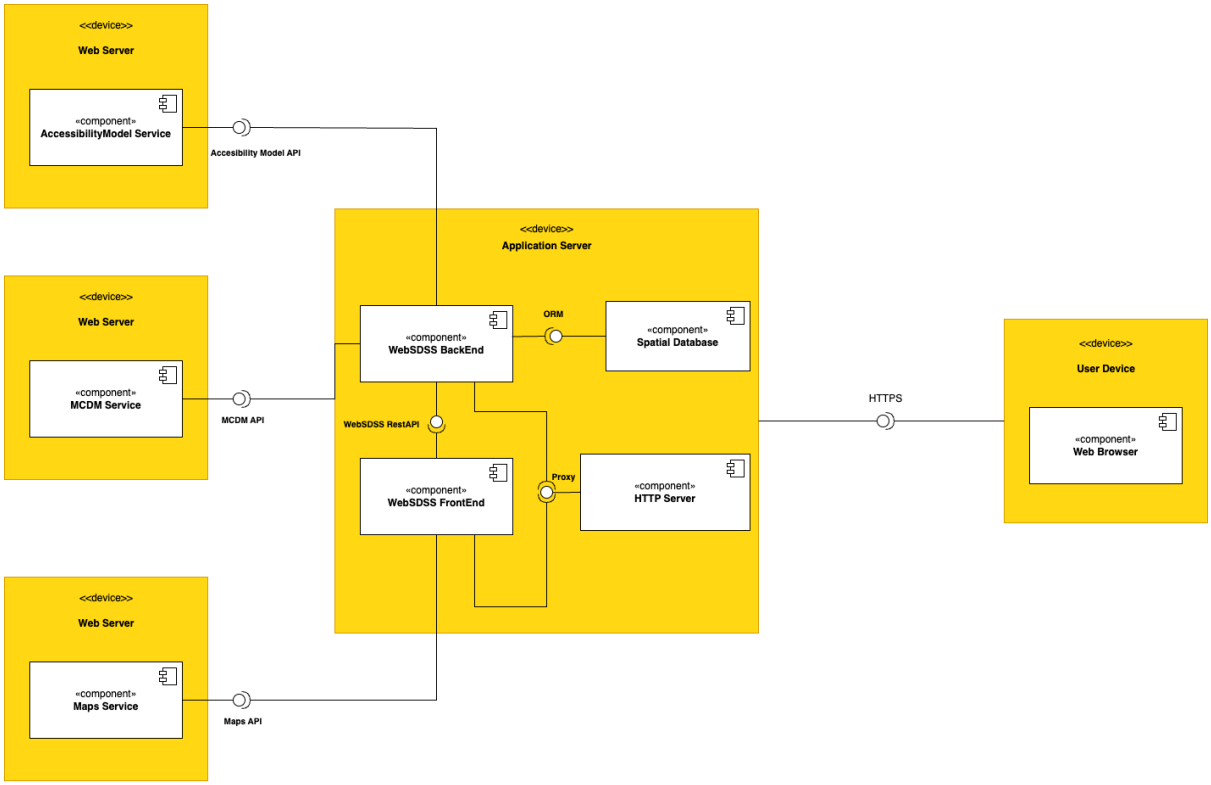

Figure 5.7: Development view of microservices architecture

The microservices architecture proposed is based on the concept of breaking down an application into smaller, loosely coupled services, each responsible for a specific business function. This approach aligns with the concept of a three-level technology SDSS, enabling the creation of different services that can be combined to generate a specific DSS. It also allows for the reusability of services and the creation of new ones as needed.

In the context of the WebSDSS frontend, the author chose to create a frontend client using a JavaScript framework. As demonstrated in the previous chapter, both Function Analysis System Technique (FAST) and Quality Function Deployment (QFD) analysis emphasize user interaction with the system, focusing on usability. The use of JavaScript frameworks for the frontend application ensures that the system can provide a user-friendly interface that meets the stakeholders' needs, thereby contributing to the generation of value. This choice is primarily justified by the following reasons:

- **Enhanced User Experience:** JavaScript frameworks enable the creation of interactive and dynamic user interfaces, resulting in a more engaging user experience. Features like virtual DOM manipulation, reactive updates, and client-side routing contribute to smoother and faster interactions.
- **Cross-browser Compatibility:** Frameworks often include polyfills and tools for handling browser inconsistencies, ensuring that applications work consistently across different browsers

and devices. This saves developers from having to write browser-specific code and enhances the application's accessibility.

- **Scalability:** JavaScript frameworks provide architecture and design patterns that support scalability, allowing applications to grow and evolve over time. They offer features like state management, code splitting, and server-side rendering, which are essential for building large-scale and maintainable applications.
- **Performance:** JavaScript frameworks are optimized for performance, providing features like lazy loading, tree shaking, and server-side rendering to improve the application's speed and responsiveness. This ensures a smooth and seamless user experience, even for complex and data-intensive applications.
- **Wide use in Web:** JavaScrip frameworks are widely used in web development, which means that there is a large community of developers, resources, and tools available to support the development process. This makes it easier to find solutions to common problems, share knowledge, and collaborate with other developers.

Another important aspect is that the frontend application is decoupled from the backend application, which allows for greater flexibility and scalability. This decoupling enables the frontend and backend to evolve independently, making it easier to update, modify, or replace one component without affecting the other.

In conclusion, the frontend client, along with the spatial database, constitutes the GIS component of an SDSS tool. Additionally, the backend app is responsible for providing the frontend client with relevant spatial data and offering various tools for data analysis.

5.2.2 Physical View

This view allows for a detailed examination of the environment in which the system is executed, as well as describing the structure of the hardware components that run the software. The design for this project aims to achieve scalability, an easy deployment process, and a well-configured environment to run the applications. To achieve this, the author decided to use specific framework hosting plataform and later Docker (Docker, Inc. 2024) containers, which provide a lightweight and portable way to run applications in different environments. This approach allows the system to be easily deployed in various environments, including local development, testing, and production.

Aditionally, in order to address the physical aspects of the system, the execution environments need to be defined. Rather than providing detailed information about each machine's hardware and specifications, the author opted to utilize specific framework application hosting services.

In the scenario of a Next js app in a monolithic architecture as seen in the initial phase, the author considered use Vercel (Vercel 2024) as a hosting service. Vercel is a cloud platform that enables developers to deploy web applications with ease, providing features such as automatic deployments, serverless functions, and global content delivery network (CDN) support. This service allows the author to deploy the frontend application and serve it to users through a secure and scalable environment.

Vercel uses Docker containers to run the applications, which provides a consistent and isolated environment for each deployment. But the image build is managed by the platform, which simplifies the deployment process and ensures that the application runs smoothly in different environments, although it doesn't provide the same level of control and customization as managing the containers manually.

Figure 5.8 illustrates the physical view of the system in the initial phase, showing how the different components are deployed in the Vercel environment. The diagram represents physical machines

with the '«device»' notation, execution environments with the '«executionEnvironment»', and the components with '«component»'.

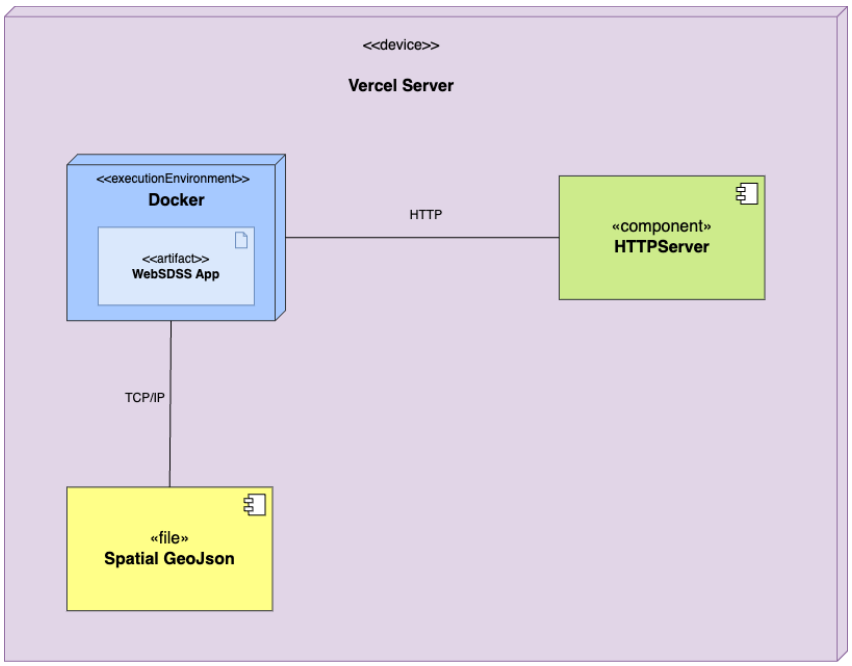

Figure 5.8: Physical view of Initial Phase

For advanced phase, the author considered using Amazon Web Services (AWS) services to host the backend application and spatial database (Amazon Web Services, Inc. 2024). AWS provides a wide range of on-demand cloud services that can be used to deploy and manage web applications, databases, and other resources. The Elastic Container Service (ECS) service was chosen to deploy the backend application in a Docker container, the Relational Database Service (RDS) service to host the spatial database, and the Elastic Load Balancer (ELB) service to manage incoming traffic and distribute it across multiple instances of the backend application.

The diagram in Figure 5.9 illustrates the physical view of the advanced phase of the system, showing how the different components are deployed in the AWS ECS environment. The diagram represents physical machines with the '«device»' notation, execution environments with the '«executionEnvironment»', and their internal components with '«artifact»'.

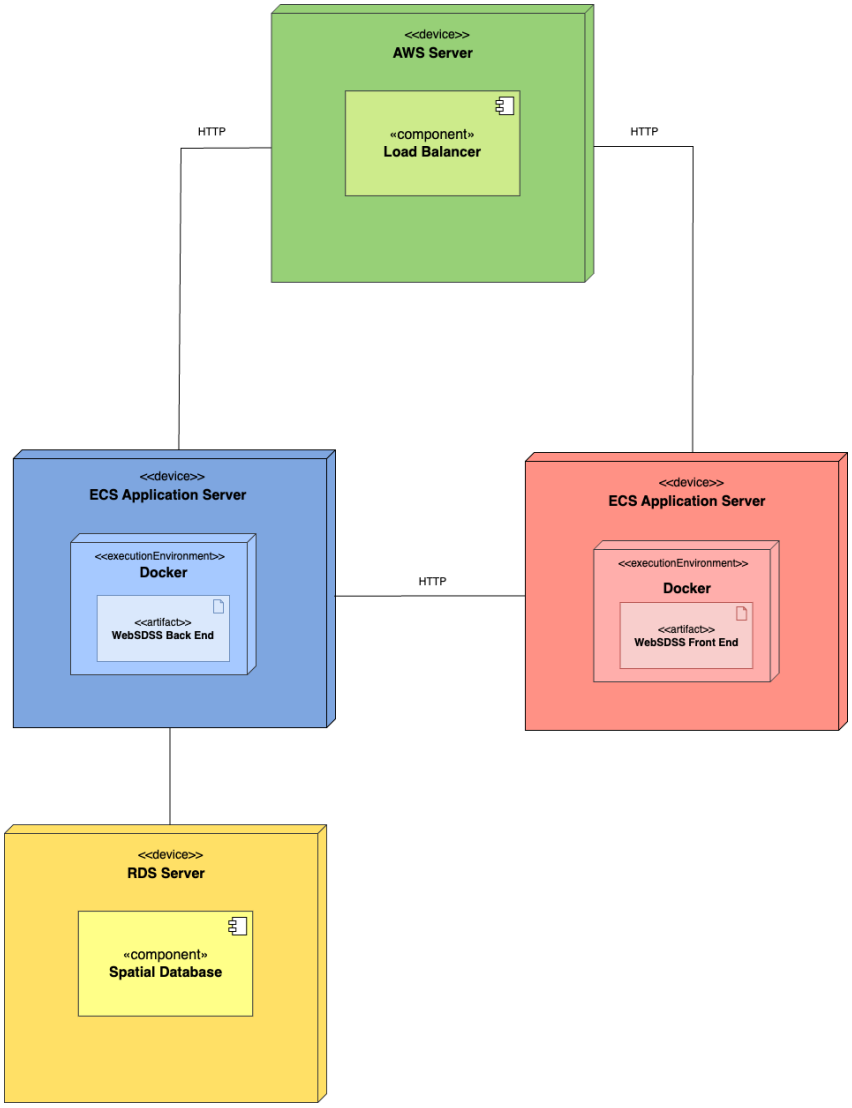

Figure 5.9: Physical view of multi-application servers

One of the possible variations considered by the author for the advanced phase physical view is having a single application server running the backend and frontend applications, as shown in Figure 5.10. This approach would simplify the deployment process and reduce the number of running environments, making it easier to manage and maintain the system. However, it would also introduce some limitations, such as reduced scalability and flexibility, as the backend and frontend applications would be tightly coupled.

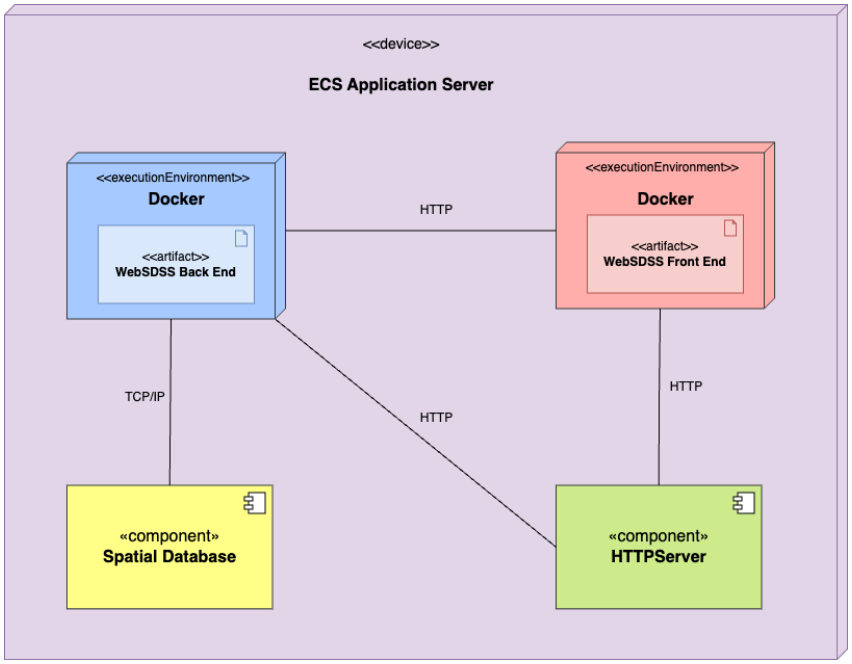

Figure 5.10: Physical view of a single application server

As shown in the development view, the microservices architecture could also be implemented in the physical view, if some backend functionalities are provided by other services. This approach would involve deploying each microservice as a separate Docker container in the AWS ECS environment or consuming external ones, allowing for independent scaling and management of each service. Figure 5.11 illustrates how the microservices architecture would look like if the 'AccessibilityModel' was provided by an external service.

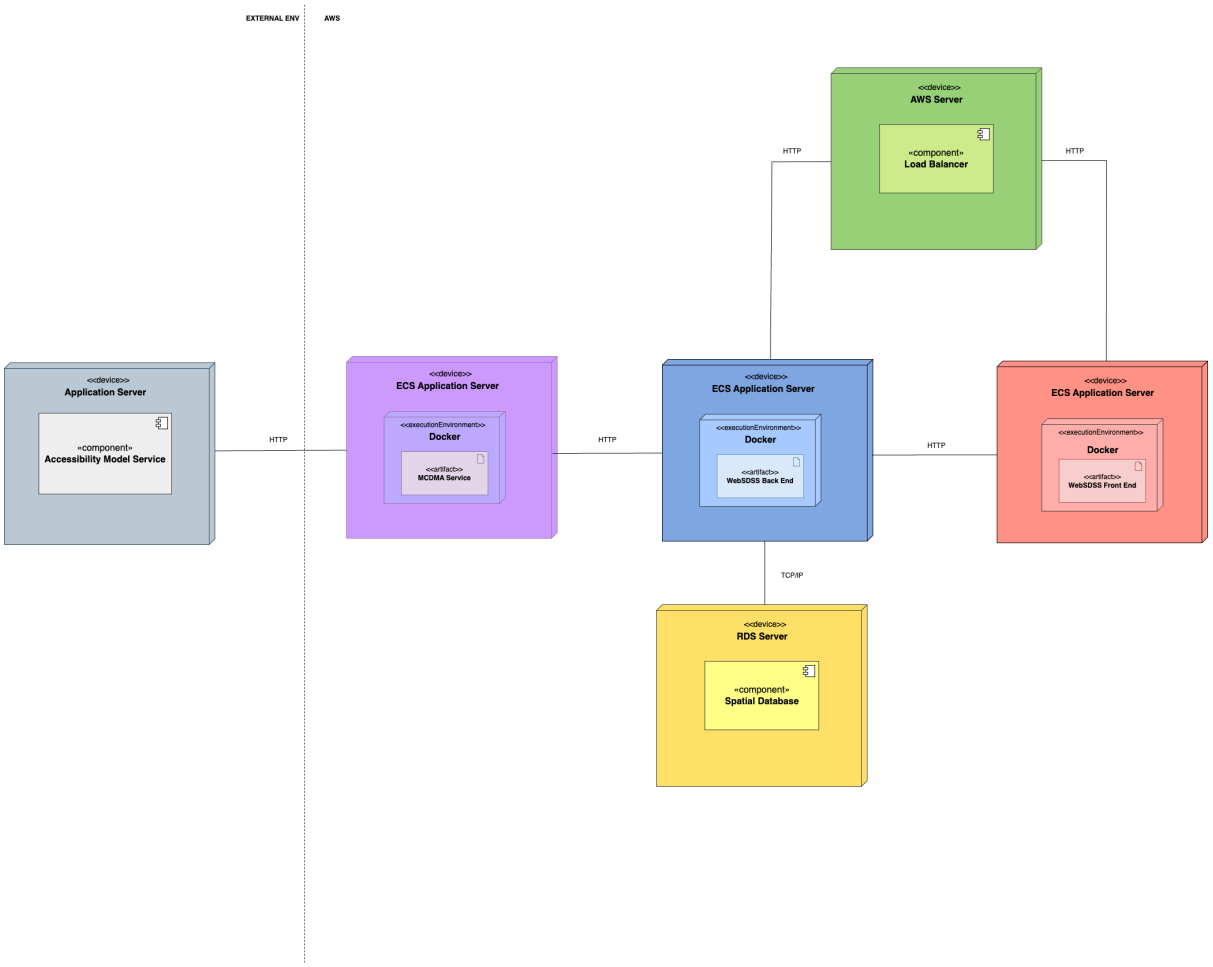

Figure 5.11: Physical view of microsservices architecture

Physical view design is crucial for ensuring that the system can be deployed and run effectively in different environments. Starting from a simplifying monolithic and centralized physical disposition and evolving to using AWS services, the system can benefit from the scalability, reliability, and security features provided by the platforms, enabling the system to meet the requirements mainly related to modularity and flexibility focused on the three levels of technology of an SDSS.

5.2.3 Proccess View

The main objective of this section is to describe the dynamic aspects of the system, highlighting the processes in its two logical parts: the front end and the back end. Activity and sequence diagrams will be presented to demonstrate the interactions between the policy-maker and the system's components during the submission of the analysis form.

Activity Diagram

To provide a clear understanding of the submission process for analysis forms, as well as the accessibility and multicriteria calculation and result visualization, an activity diagram was created. This diagram depicts the flow of control within the system, highlighting the sequence and conditions of the steps involved.

Figure 5.12 illustrates the process of submitting the analysis form and visualizing the results from both the user's and the system's perspectives. The user's steps are represented through interactions

with frontend interfaces, while the system performs analysis and data validation processes in the backend, as well as rendering some UI elements in the frontend.

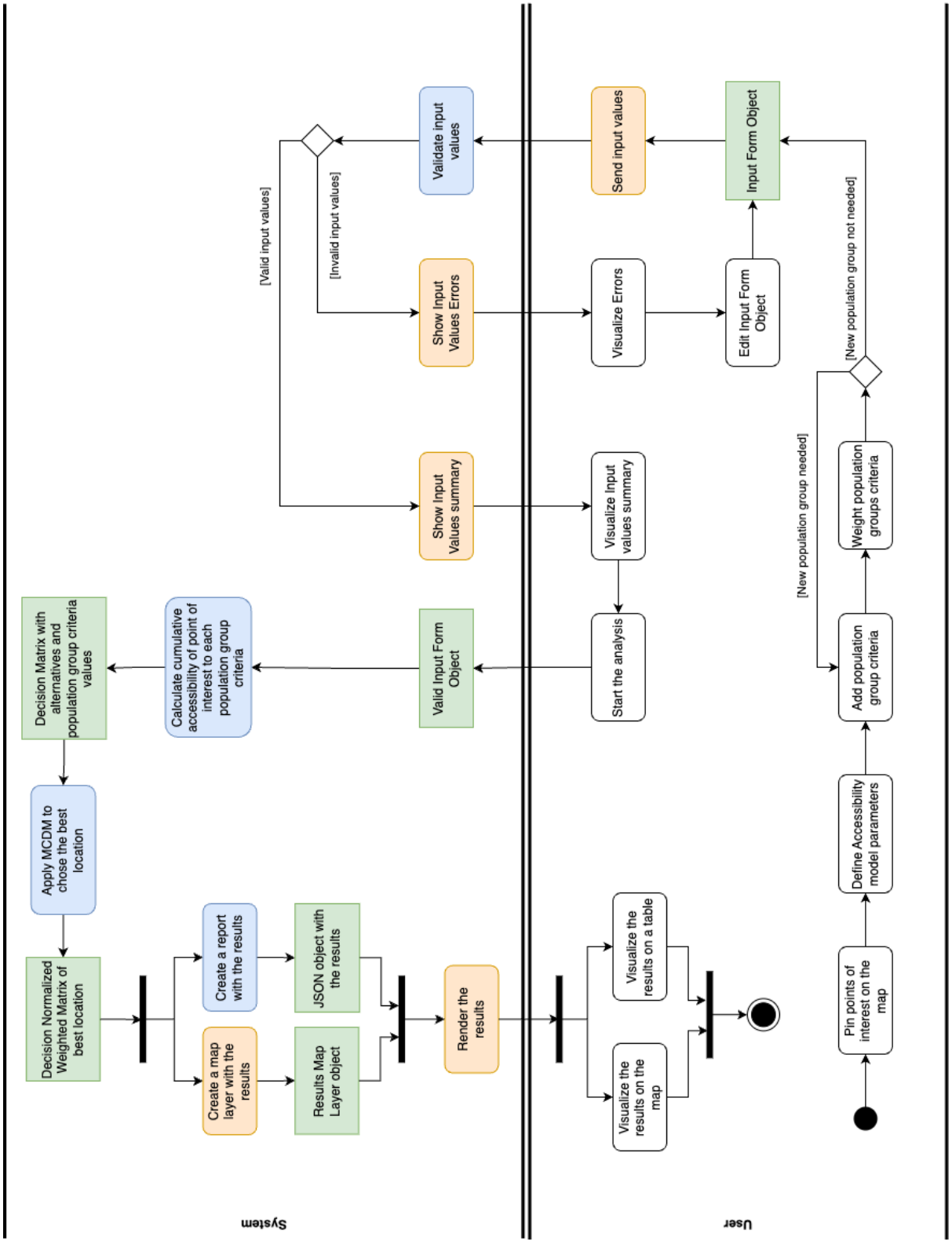

Figure 5.12: Activity Diagram

All activities performed by the frontend component are represented in orange, while the backend

activities are shown in blue. The remaining activities depict the interaction between the user and the frontend application. It is worth mentioning that the objects generated as output from each activity are displayed in green.

The flow described by the activity diagram involves the user interacting with the map by selecting points of interest, inputting parameters for the accessibility analysis and multicriteria analysis, submitting the analysis form, and visualizing the results. The system receives the input data, validates it, and if the user's input is valid, calculates the accessibility and multicriteria analysis, returning the results to the frontend application. If the input data is invalid, an error message is returned to the user, who can then correct the input and resubmit the form.

The objects generated at the end of certain activities illustrate how the system transforms the user's data into the results of the analysis. It is interesting to note how an output object is used as an input for the next activity, demonstrating the flow of data and control between the components. As an example, the input form data is used to calculate the accessibility indexes by population groups, which are then used as input for the multicriteria analysis in form of a decision matrix. After the analysis is completed, the results are displayed to the user in the form of a map layer and a table, providing a visual representation of the analysis output.

Sequence Diagram

In order to meet the user requirements and enable flexible parameterization, it is crucial to establish a clear and standardized process in the user interface, as well as a systematic and analytical workflow in the backend. Therefore sequence diagrams are a great option to illustrate the flow of events and messages between the actors and the system, as well as the control and conditional structures that govern the process.

Sequence diagrams were constructed to reflect the architecture of the WebSDSS and its components, as described in the development and physical view diagrams depicted in Figures 5.5 and 5.9, respectively. All of the sequence diagrams mentioned in this section are in Appendix Z.

The author chose to highlight the functionalities described by the functional requirements related to both the frontend and backend logical parts of the system. These requirements are R3 and R4, which are related to obtaining user inputs through a form on the frontend, mainly described by use case UC3. Additionally, requirements R5, R6, and R7 pertain to the backend and involve form submission, analysis calculation, and delivery of results.

The first diagrams in Figures Z.1, Z.2, and Z.3 show the interaction between the user and the system when filling out the analysis form, as represented by requirements R3 and R4. These diagrams illustrate the flow of events between the components that make up the frontend logical part of the proposed solution, including the WebBrowser, the WebSDSS frontend application, and the main actor, the policy-maker.

The WebBrowser participant is included in the diagram because this is a web-based solution, and web browsers are the applications used to access web systems based on HTML, JS, and CSS technologies for rendering web graphical interfaces. In this solution, a JavaScript framework is also chosen to build the frontend app, and some UI components are rendered by the browser itself. However, in more recent frameworks like Next JS, some components can also be rendered on the server side.

Since the frontend interface is designed as a Single Page Application (SPA), the first three sequence diagrams depict the initial request made by the browser to the frontend app to obtain the HTML resources and JS scripts. After that, all the interactions between the user and the interface are conducted through the execution of the downloaded JS scripts. The frontend app may also

dynamically fetch required data and JavaScript to update or load specific components, avoiding the need to reload the entire page.

In the diagram of Figure Z.1, the user interacts with the interactive map by pinning points of interest when clicking on the map. The frontend app then stores the selected points in the form so that they can be sent along with the form data as input for accessibility and multicriteria analysis.

For the sequence diagram that represents filling in the MCDMA parameters in the analysis form (see Figure Z.2), the user interacts with the form to input the desired parameters for the accessibility analysis. This includes selecting the time threshold (groups reached within a specific travel time), the transport mode, and the hour of the day for performing the analysis.

Finally, in Figure Z.3, the user interacts with the form to input the desired parameters for the multicriteria analysis. This can be done by adding new population groups, which serve as criteria for the MCDMA analysis, and assigning weights to them. The user can add up to five groups, as represented by the loop in the diagram.

Regarding the backend logical part of the system and requirements R5, R6, and R7, the sequence diagrams in Figures Z.4 and Z.5 illustrates the interactions between the frontend and backend components.

This first diagram (see Figure Z.4) represents the flow of events that occur when the user submits the analysis form for validation. The backend application validates both the accessibility and Multi-Criteria Decision-Making Analysis (MCDMA) input data and then returns the results to the frontend application, which will render a summary containing any errors or warnings, if applicable.

The second diagram illustrates the submission of the analysis form after validation. The user initiates the submission through the frontend application, and then the backend calculates the accessibility and multicriteria analysis based on the input data. The results are then returned to the frontend application and displayed to the user, as shown in Figure Z.5.

The WebSDSS API flow shows potential for evolving into a microservices architecture through modularity and separation of concerns. Services like Accessibility and MCDMA can be decoupled and either become external services or form new components, enabling a comprehensive decision support system.

5.3 Spatial Database Design

Following the DDM paradigm presented in Subsection 2.3.2, one of the main components of a DSS is a spatial database. This component allows access to spatial data and facilitates different analyses. The design of the spatial database is crucial to ensure that the system can efficiently store, query, and analyze spatial data, as well as provide the necessary functionalities to support the decision-making process.

This section will first present the spatial data sources used to compose the final data records, followed by the spatial data structure, which defines how spatial data is organized based on the application of the accessibility model and the use of data sources.

5.3.1 Data Sources

The data sources provide data from three different areas: public transportation, socio-spatial data, and road and urban infrastructure layout.

IBGE Demographic Census Data

The IBGE provides various geospatial and statistical data on its website, resulting from its mappings, studies, and Censuses (Instituto Brasileiro de Geografia e Estatística (IBGE) 2024). Two files from the 2010 Census were downloaded: the geospatial data and the statistical data. The file with statistical data contains information collected by the 2010 Census for each Census Tract in the state of Ceará (population, households, income, water supply, etc.). The other one with geospatial data is in shapefile format and contains the geographic representation of the Census Tracts. Each feature in the shapefile has an identification code for the tract, which will be used to link both files information.

Public Transportation Routes and Schedules

The public transportation routes and schedules data in Fortaleza were obtained from the Urban Transportation Company of Fortaleza (ETUFOR) and the State Infrastructure Secretariat (SEINFRA) through the 'Dados Abertos Fortaleza' portal and the Metrofor portal, respectively (Fortaleza City Hall 2024). The data was provided in the form of a General Transit Feed Specification (GTFS) file, which is a widely adopted open standard for transportation system data communication.

Although the static GTFS data does not account for traffic congestion, it is crucial for accurately mapping the functionality of the transportation system and integrating it into urban accessibility modeling. It provides a reliable representation of public transportation operations.

Road Layout and Urban Infrastructure

To obtain the spatial layout of road networks and pedestrian infrastructure in Fortaleza, the OpenStreetMap (OSM) API was used to retrieve a '.pbf' file (OpenStreetMap contributors 2023). The OSM data is organized using a topological data structure, consisting of nodes (geographical points defined by latitude and longitude coordinates), ways (ordered lists of nodes representing polylines or polygons), relations (lists of nodes, ways, and other relations expressing connections between them), and tags (key-value pairs providing metadata about the map objects).

The data regarding the facilities in Fortaleza, such as schools, hospitals, and police stations, administered by the city government, is also provided by the Fortaleza City Hall open platform. However, this data is not necessary for the purpose of this work. It includes information about the number of vacancies, location of the facilities, among others. Integrating this data with the demographic census and public transportation data is essential for future works that aim to analyze the demand and supply of public services in the city.

5.3.2 Spatial Data Structure

The spatial data structure defines how spatial data is organized and stored within the database. The system utilizes spatial data from various sources as listed in the previous Section.

In the context of this work, as seen in Section 4.3, the author decided to use the Cumulative Opportunities model to calculate accessibility. This model is described in Section 2.2.4, but for this work, the passive perspective of accessibility was chosen based on multi-criteria analysis of Section 4.3. This means that the accessibility calculation will be based on how many people can reach a specific location (opportunity) using public transportation within a certain time threshold, instead of how many locations can be reached from a specific point within the same time threshold.

This database should be able to store the travel-time matrix between two points in the city, adding socioeconomic and demographic indicators of the population at the origin point. This way, it is possible to group the rows that are within the determined travel time cost, aggregating the

socioeconomic and demographic indicators. This makes it easy to determine how many people, based on socioeconomic and demographic aspects, can reach that point. Figure 5.13 shows how this matrix is calculated using the data sources.

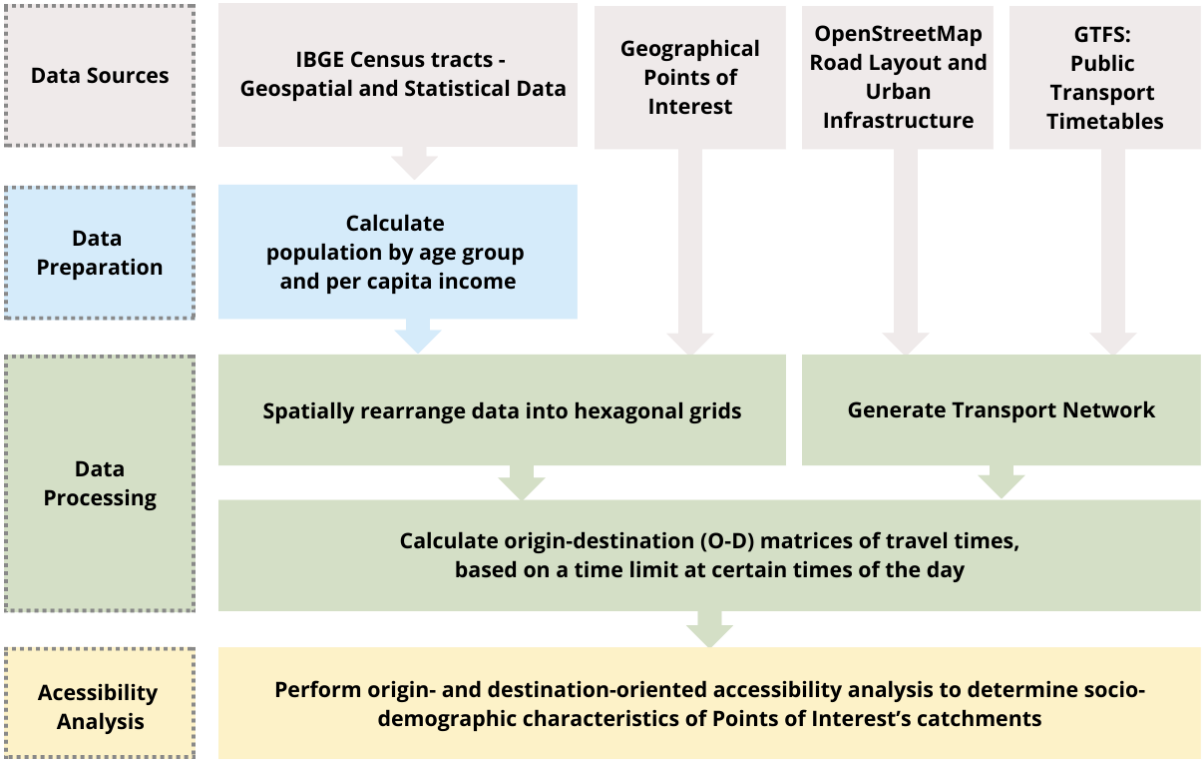

Figure 5.13: Travel-time accessibility matrix calculation process

All following steps were implemented using python and the libraries used are described in each section. The script is availale in the appendix A.

Organize data into hexagon grid

Most of the socioeconomic and demographic data of the city of Fortaleza is separated into Census Tracts, which are irregular polygons that represent specific areas of the city. In order to normalize the data and facilitate the calculation of accessibility, the author decided to organize the data into a hexagon grid as seen in Figure 5.14. The hexagon grid is used to discretize the city into manageable units, making it easier to store, query, and analyze spatial data.

Hexagons are particularly valuable in representing geographic data in accessibility studies as they distribute areas evenly and minimize edge bias (Pereira et al. 2021). This approach provides an efficient and equitable geographic structure for discretizing urban areas into analysis units. Not only does it maintain data consistency, but it also facilitates the calculation of accessibility metrics such as travel time or distance, allowing for a fair assessment of access to urban services and resources.

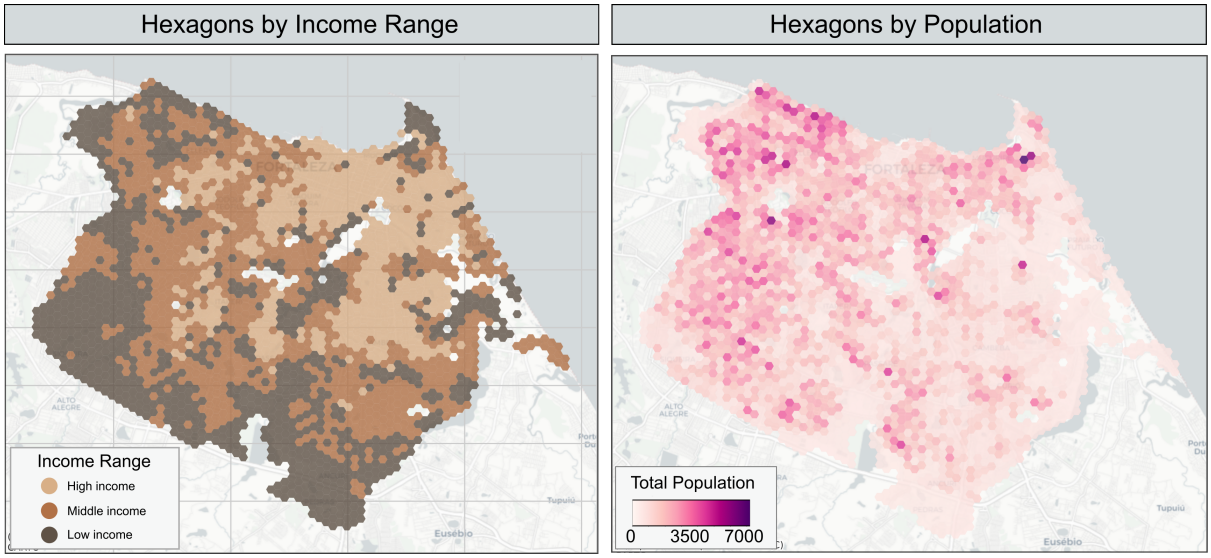

Figure 5.14: Fortaleza Data Distribution in Hexagons

Hexagons with a 400-meter diameter and an average area of 0.105 km^2 were used, closely matching the average area of the census tracts in Fortaleza (0.102 km^2). Each hexagon has a unique code and fixed geographic coordinates, ensuring data uniformity and accuracy.

In summary, the procedure to organize the data into a hexagon grid involves the following steps and is illustrated in Figure 5.15:

- **Hexagon Generation:** Generate a hexagon grid covering the entire city of Fortaleza, with each hexagon having a unique code and fixed geographic coordinates.
- **Data Aggregation:** Aggregate the socioeconomic and demographic data from the Census Tracts to the corresponding hexagons whose centroids fall within the Census Tract polygons, ensuring that each hexagon contains the relevant information. Additionally, for Census Tracts that do not contain any hexagon's centroid, a revaluation procedure is performed, aggregating their information to the nearest hexagon. In cases where a Census Tract contains more than one hexagon, the information is normalized and distributed among them.

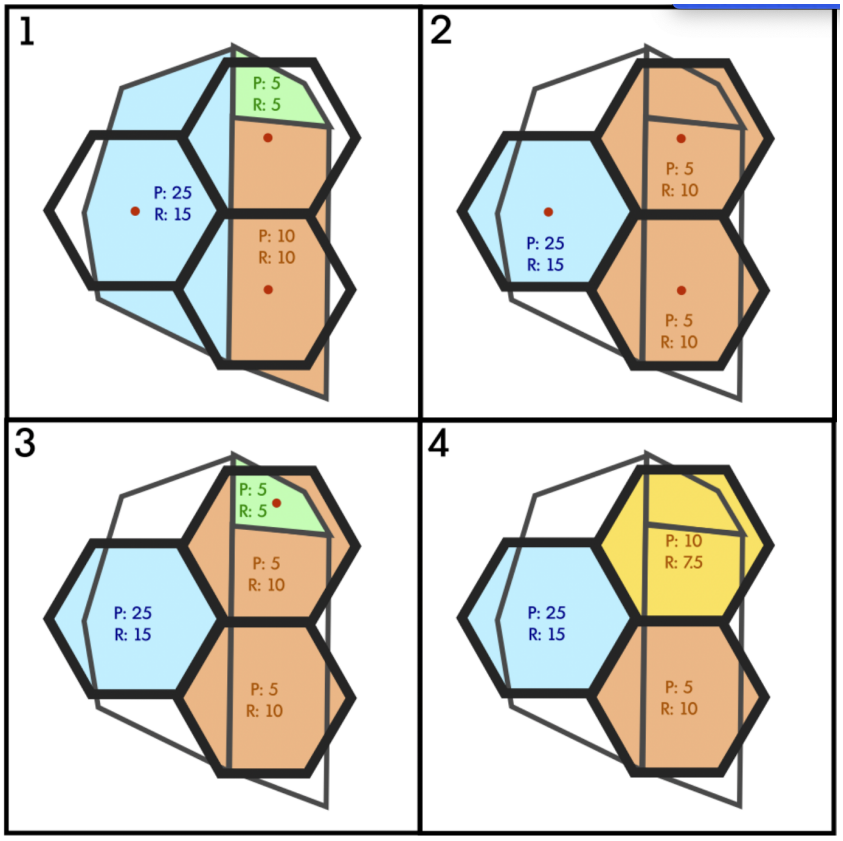

Figure 5.15: Steps of geographic data distribution in hexagons

To calculate this, the H3Pandas (Butler 2020) tool is used, which is based on the open-source H3 API developed by Uber.

Create Transport Network This step consists of building the transportation network, which involves combining spatial and temporal information. To perform this task, the Python library R5py is used, which specializes in interacting with GeoDataframes from GeoPandas and is capable of performing fast and realistic routing on multimodal transportation networks, including modes such as walking, cycling, public transportation, and driving.

The input parameters for the calculation are the '.pbf' file, which contains the road network and urban infrastructure data extracted from OpenStreetMap, and the files containing the GTFS data for the bus network.

Based on these data, a transportation network object is constructed, modeled as a directed graph. In this graph, the nodes represent points of interest in the transportation system, such as bus stops and metro stations, while the edges represent the connecting routes, such as roads, bus routes, and pedestrian walkways. It is important to note that each edge has a travel-time attribute, which indicates the time required to travel between the connected nodes. With the defined transportation network structure, it is possible to proceed with the calculation of travel times.

Generate Matrix of Travel Times

The next step in the analysis involves calculating the origin-destination matrices, which are critical for understanding travel times between the hexagons on the map. To perform this calculation, the Python library R5py (Conveyal 2023b) is used.

Several input parameters are considered, such as the mode of transportation, the time of day, the date, and the duration of the trip. For this work, the mode of transportation is set to bus, and the time of day is limited to the first and last hour of business hours. The date is set to a typical weekday.

The result of this process is the creation of a detailed matrix that represents travel times between the hexagons. The matrix contains the IDs of the origin and destination hexagons, along with the corresponding travel times in minutes. Additionally, the matrix includes aggregated socioeconomic and demographic data corresponding to the origin hexagon. Tables 5.1 and 5.2 shows an example of how these matrices are organized in a tabular data structure.

It is important to note that the data produced may not represent the optimal application of the model, as the input parameters were not thoroughly studied to obtain the best results. However, the goal is to demonstrate how the model is applied and how the data is organized to calculate accessibility.

from_id	to_id	travel_time	h3_origin	h3_destination	hex_polygon
0	0	2.0	89801040323ffff	89801040323ffff	POLYGON ((-38.50232 -3.88586, ...)
1	0	21.0	89801040327ffff	89801040323ffff	POLYGON ((-38.50527 -3.88410, ...)
2	0	38.0	8980104032bffff	89801040323ffff	POLYGON ((-38.49932 -3.88419, ...)
3	0	14.0	8980104032fffff	89801040323ffff	POLYGON ((-38.50227 -3.88243, ...)
4	0	17.0	89801040337ffff	89801040323ffff	POLYGON ((-38.50532 -3.88754, ...)

Table 5.1: First lines of Travel Time Matrix- part 1

per_capita_income	pop_total	pop_0_to_5	pop_6_to_10	pop_11_to_14
435.35	120.250000	11.500000	11.250000	7.500000
435.35	120.250000	11.500000	11.500000	7.500000
339.74	108.600000	8.800000	9.700000	9.500000
420.35	27.774194	2.774194	2.129032	2.16129
435.35	120.250000	11.500000	11.250000	7.500000

Table 5.2: First lines of Travel Time Matrix - part 2

Chapter 6

Implementation

With the system design in hand and a concrete architecture defined for the project, the next step is the implementation of this architecture. This chapter begins with preliminary considerations regarding the implementation of the solutions, highlighting their limitations and the aspects taken into account given the implementation context. Following this, the development methodology applied to the project is presented. Next, the integration and continuous delivery processes employed throughout the development are described. The chapter then delves into the development of the solution itself, detailing all internal components of the system and their interactions, and emphasizing how they were constructed at the code level to align with the designed architecture. Finally, the chapter concludes by discussing the most relevant peculiarities associated with the solution's implementation. Throughout the chapter, code snippets are included to illustrate both the implementation of the solution and its adherence to the previously designed architecture.

6.1 Considerations

The proposed solution encompasses a diverse array of accessibility model options and demographic data. However, certain reductions and simplifications were necessary to make the implementation feasible within the context of this thesis, primarily due to constraints related to time, data availability, and infrastructure.

The following considerations were made to simplify the implementation while ensuring that the requirements are still met:

6.1.1 Reduced Available Demographic Data

In Section 5.3.2 of the previous chapter, a spatial data structure for the accessibility analysis matrix was presented. This structure included results from the displacement between hexagons, into which the city was segmented based on the transport network, along with merged demographic data from the origin hexagons. Due to limited data, the analysis focuses on a reduced demographic dataset, including only the population of individuals aged 0 to 16 years and the total population, along with income data. Ideally, the scenario would incorporate different tiers of age, gender, and education levels. However, for this implementation, only the demographic data presented in Section 5.3.2 will be utilized.

6.1.2 Simplified Accessibility Analysis Matrix

To perform various accessibility analyses based on transportation mode, time of day, and travel time thresholds, multiple accessibility matrices need to be calculated according to these parameters, covering different analytical scenarios. To simplify the process, an accessibility matrix based on a 30-minute travel time threshold, public transportation mode, and the time of day from 8:00 to 10:00 AM will be used.

6.1.3 Simplified Source Data Format

The accessibility matrix will be stored in a file rather than in a database, primarily to reduce infrastructure costs. While file-based storage generally has lower performance compared to a database and offers less flexibility in terms of openness and access, the reduced size of the accessibility analysis matrix due to simplified accessibility analysis described in Section 6.1.2 makes file storage feasible.

6.1.4 Monolithic Architecture

The author opter for a monolithic architecture presented in Section 5.2.1 to implement the system. The focus was on implementing the core features, delivering a working prototype within the scope of this thesis. Future iterations of the system may benefit from a more modular and scalable architecture, such as a microservices architecture, which allows for greater flexibility and scalability as the system grows.

6.1.5 Vercel as Hosting Platform

Vercel is a cloud platform for static sites and serverless functions that enables developers to deploy web applications with ease (Vercel 2024). It provides features such as automated preview and production deployments, which are essential for implementing continuous delivery and deployment practices. It was chosen as the hosting platform for this project due to its ease of use, scalability, and integration with GitHub, the remote code repository platform used.

Besides the simplified deployment process, Vercel also provides a free tier that allows developers to deploy their applications without incurring any costs. This free tier is ideal for small projects or personal projects that do not require a large amount of resources. This factor was of great importance as it allowed the author to deploy the application without incurring any costs, making it an ideal choice for this project.

Finally, Vercel is well-suited for Next.js applications, the framework used for the frontend and backend of this project implementation. It provides built-in support for Next.js applications, making it easy to deploy and scale on the platform.

6.2 Development Methodology

A development methodology serves as a framework for managing software projects. It encompasses various aspects, including the selection of features to be included in the current software release, the scheduling of its launch, and the determination of required tests. Choosing the right development methodology can significantly influence the project's success by affecting adherence to deadlines, managing costs, and ensuring the software's robustness (Young 2013).

For this project, Git (Torvalds et al. 2005) was chosen as the version control system, alongside GitHub (GitHub 2024) as the Git hosting platform. Given that a monolithic architecture was selected for the implementation, the entire project is contained in a single repository, following the trunk-based development strategy.

The adopted development methodology for this project is called Trunk Based Development (TBD). TBD is a software development methodology that promotes collaboration and efficiency by having developers work directly on a single main branch, or trunk, rather than using long-lived development branches.

TBD encourages integration and frequent code merges, reducing complexity and minimizing merge conflicts. TBD supports a culture of continuous delivery with frequent, small updates, allowing

teams to deliver value quickly and respond to feedback promptly. It also emphasizes automated testing and deployment, contributing to a more stable and reliable codebase. While TBD may not suit every project, it enhances collaboration, accelerates feedback, and increases overall development agility (Authors 2024). Figure 6.1 demonstrates this strategy concisely.

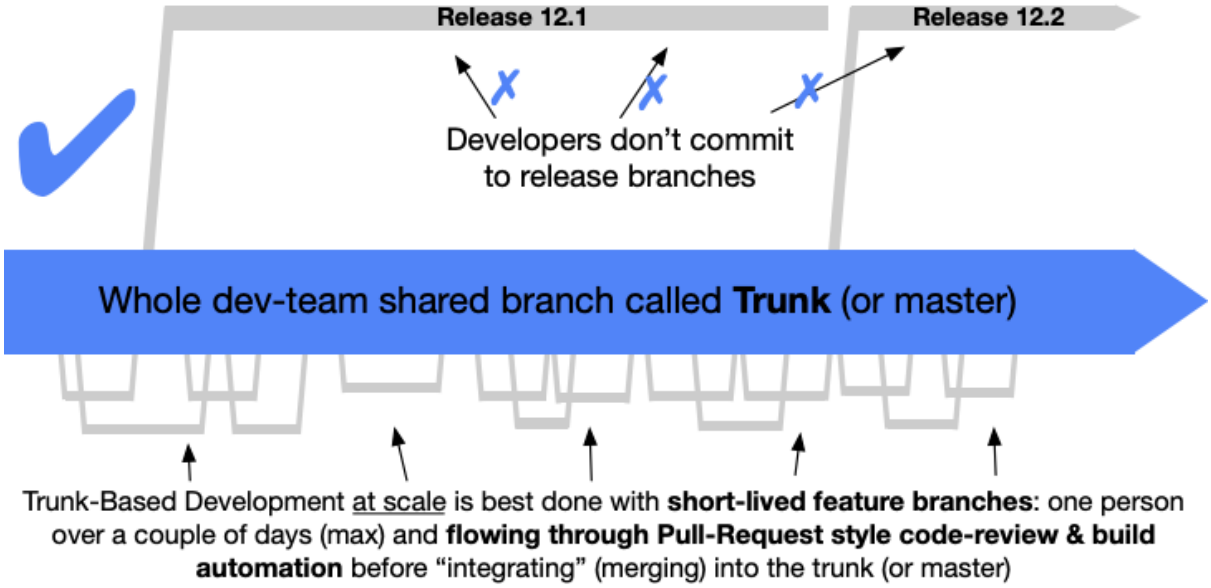

Figure 6.1: Trunk Based Development Strategy (Authors 2024)

This approach involves developing system features in isolated, short-lived branches derived from the main branch. There are no separate development or release branches; instead, the code from feature branches is merged directly into the main branch, which is then deployed. To ensure that the branch code is deployable and maintains the integrity of the main branch, each new branch undergoes a continuous integration pipeline.

Finally, there are two environments for deployment: staging and production. Commits to branches other than the main branch result in the system components being published, deployed, and tested in the preview environment of Vercel host platform. Conversely, the production environment of Vercel is used for the main branch.

6.3 Continuous Integration and Delivery

Continuous practices, including Continuous Integration (CI), Continuous Delivery (CDE), and Continuous Deployment (CD), are vital in the software development industry, enabling frequent and reliable releases of new features and products by automating the building, testing, and deployment of software. These practices reduce the time and effort needed to deliver new features, improve software quality, and increase team productivity (Shahin, Ali Babar, and Zhu 2017).

CI involves team members frequently integrating their work, leading to shorter release cycles and enhanced productivity through automated building and testing. CDE ensures applications are always production-ready by passing automated tests and quality checks, reducing risks, costs, and speeding up user feedback. CD extends CDE by automatically deploying applications to production environments without manual steps (Shahin, Ali Babar, and Zhu 2017).

For implementing CI and CD, this project utilizes features of GitHub Actions and Vercel. Vercel is a cloud platform for static sites and serverless functions, enabling developers to deploy web applications

with ease. Vercel provides features such as automated deploys, preview deployments, and production deployments, which are essential for implementing continuous delivery and deployment practices.

The following sections describe how these practices are implemented in this project.

6.3.1 Github CI

As mentioned previously, the development methodology for this project is based on the trunk-based development strategy, which requires a solid continuous integration and delivery process to ensure that the main branch remains deployable at all times. Each GitHub repository contains workflows that implement automated tasks, such as running tests and deploying components to the cloud, defined in files under a folder named **.github**. These files are written in YAML format, a data serialization language commonly used for configuration files (YAML 2021).

The author defined three workflows triggered by the pull request event: the Linting Check, Code Changes Check, and Test Build Check. These workflows are mandatory checks for pull requests, meaning code cannot be merged until all checks are passed. Figure 6.2 shows these checks in the pull request.

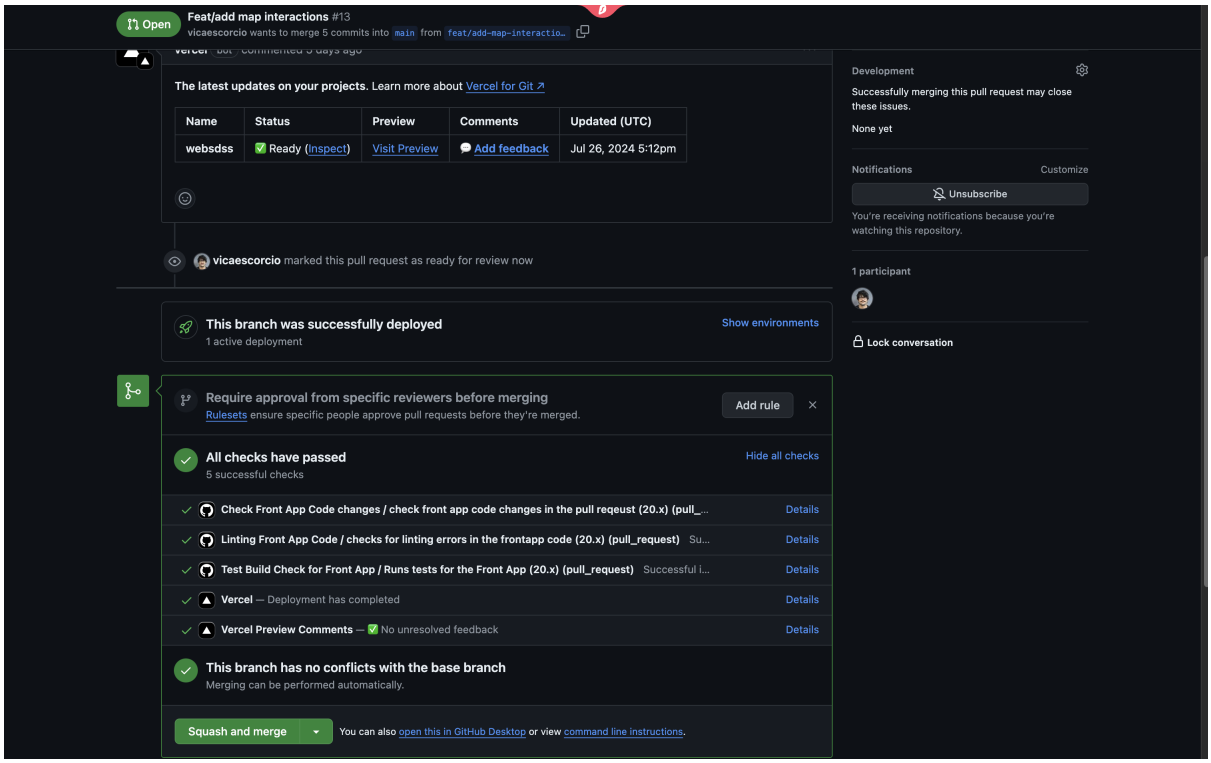

Figure 6.2: Screenshot of Github Pull Request workflow checks

Workflows are defined in the '.github/workflows' folder in the repository and they have jobs that are executed in parallel. Each workflow has jobs, and each job has steps that are executed sequentially. The author defined common steps that are included in all workflows. The common steps are as follows:

- Checkout the code from the repository.
- Set up the Node.js environment.
- Cache the node modules to speed up the build process.

- Install the dependencies.

The Appendix I shows these common steps for all workflow files described as following:

Code Changes Check

"Code Changes Check', in the source code I.1, essentially runs the build script of the project, which is written in TypeScript. The build command transpiles the TypeScript code into JavaScript code. Specifically, the build commands used here are **npm ci** to install the dependencies and npm run build to transpile the code.

In addition to the build step, there is a final step that builds a Docker image. This Docker is defined in a DockerFile as seen in the Appendix F.

This Dockerfile builds a Node.js application in multiple stages for efficiency. It starts with a lightweight Node.js base image (node:20-alpine). The deps stage installs necessary dependencies and copies the package.json files to install project dependencies. The builder stage copies dependencies and source code, then builds the application. In the runner stage, it sets up the environment for production, creates a system user for security, copies the built files, sets file ownership, and specifies the user to run the application. Finally, it exposes port 3000 and sets the command to start the server using node server.js.

This step anticipates potential deployment problems, as the Vercel host platform compiles a Docker image to deploy the application, ensuring that the code compiles successfully and that there are no syntax or type errors.

Linting Check

Linting involves analyzing code to identify potential errors, bugs, stylistic inconsistencies, and suspicious constructs. **Linters** are static code analysis tools that identify problematic patterns or code that does not adhere to certain style guidelines. They are used to enforce coding standards, improve code quality, and catch bugs early in the development process (Software n.d.).

In this project, the author uses **ESLint**, a popular JavaScript linter, to identify and fix problems in the code (Zakas 2013). The author has created a workflow that runs the **ESLint** script on the codebase to ensure adherence to defined style guidelines and best practices. This script is defined in the **package.json** file and is executed by the linting step in the workflow. The Appendix I shows the linting step in the workflow file.

To configure **ESLint**, the author installs the **ESLint** package and creates an **ESLint** configuration file called **.eslintrc.js** in the project's root directory. This file contains the **ESLint** configuration, including the rules that the linter should enforce. It extends the **Prettier** configuration file, which is a code formatter that enforces a consistent code style across the codebase (Prettier n.d.).

The code with the rules definition for Prettier and ESLint configuration are in the Appendix E.

Test Build Check

The **test build check** runs the test script of the project, executing both unit and integration tests. The specific test script used in this project is `npm run test`. This check ensures that the code passes all unit tests and that the tests adequately cover the codebase. The code snippet is shown in Appendix I.

The **Jest** package, a popular JavaScript testing framework, provides a simple and easy-to-use interface for writing and running tests (Facebook n.d.). The author creates a test script in the

`package.json` file to run the Jest tests. This script is executed by the test build check in the workflow file. All files following the naming pattern ***.test.tsx** are considered test files.

To configure the test build, the author creates a Jest configuration file called `jest.config.js` in the project's root directory. This file (see code in the Appendix E) contains the Jest configuration, including the test environment, test match patterns, and coverage thresholds.

6.3.2 Vercel CD/CDE

In addition to these custom checks, the author enabled the default automated deployments provided by Vercel, the hosting platform used for this project, as mentioned in Section 6.1.5.

Deploy Preview

This feature, called Preview Deploy, automatically deploys every pull request to a unique URL, allowing the author to preview changes before merging them into the main branch. This feature is enabled by default for all Vercel projects and can be disabled if necessary.

Deploy Production

The author also enabled the Production Deploy feature, which automatically deploys the main branch to the production environment. This feature is also enabled by default for all Vercel projects and can be disabled if necessary.

6.4 Solution Development

This section demonstrates the implementation of the solution's components, detailing how they were coded and organized to align with the architecture and meet the requirements from the previous chapter. The solution comprises three main components: the frontend, backend, and spatial database, each described in terms of structure, code organization, and interactions. The section is divided into three subsections, focusing on the main use cases: filling out the analysis form, validating and submitting the form, and displaying the analysis results, with each subsection covering the implementation of the frontend, backend, and spatial database components for each use case.

The solution's architecture is monolithic, as depicted in the development view's initial phase. Both the frontend and backend are built using Next.js, a React framework. The frontend functions as a SPA and interacts with the backend via API calls. Additionally, the author uses a CSS framework called Material UI, a popular React UI framework that provides a set of components and styles for building modern web applications. Material UI is used to style the frontend components and provide a consistent look and feel across the application.

The backend processes these calls, manages data, and returns results. Finally, spatial database, implemented as a file (see Section 6.1.3), contains the accessibility analysis matrix (see Section 5.3.2) used by the backend for accessibility analysis.

An important consideration before diving into the subsections is the aspect of Next.js applications adopting server-side rendering. This means that some files are first rendered on the server and then sent to the client, which is useful for SEO purposes and improving the application's performance. By default, all React components are rendered on the server, but you can explicitly define where the files will be rendered by using use client or use server at the beginning of a file.

A quick look at the project folder structure is shown in Figure in the Appendix G.1, which illustrates how the components are organized in the project and the hierarchy of the components.

6.4.1 Fill Up Analysis Form

This use case is managed on the frontend by two main React components: 'AnalysisForm' and 'Map'. The 'AnalysisForm' component, which is built around a child component called 'Stepper', organizes the form into a navigable wizard. Each form component within 'Stepper' has its own server-side actions for handling data processing tasks like validation and submission. The 'Map' component renders an interactive map, enabling users to select locations and visualize results.

The code in the Appendix C illustrates how the child forms are listed in the 'steps' variable and passed as parameters to the 'Stepper' component. Each object in this list contains a label, the corresponding component, and an 'onValidate' action. The server-side files process and validate the form data before forwarding it to the backend. User input is validated at each step; upon clicking the 'next' button, the data is validated. If correct, the user progresses to the next step; otherwise, an error message is displayed.

The 'AnalysisForm' component also manages state objects and functions related to form data. These state objects store the form data, which is then passed as props to child components for access and modification. Functions are similarly passed as props, allowing child components to update state objects and handle form submission.

Figure 6.3 displays the graphical interface of the 'AnalysisForm' component, showing the location, accessibility, and multicriteria forms divided into steps.

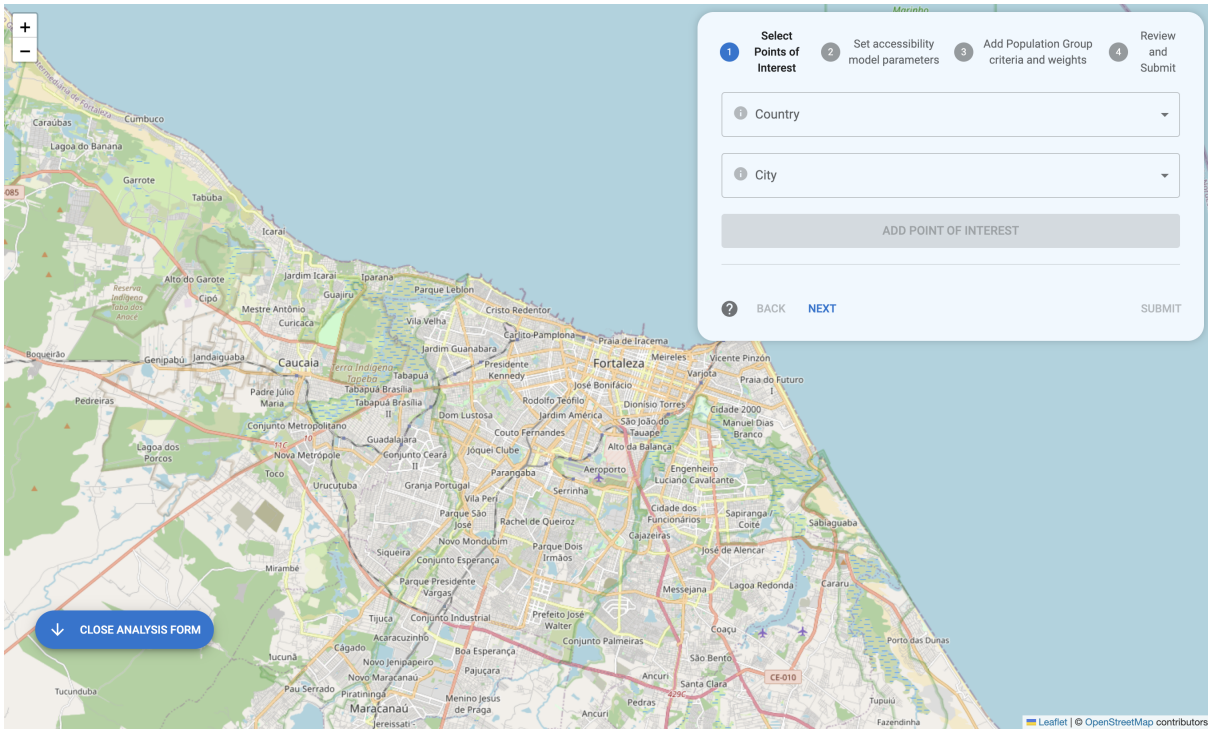

Figure 6.3: Screenshot of the proposed WebSDSS AnalysisForm component

The 'AnalysisForm' includes a helper button that opens a 'HelperCard', offering explanations and guidance on how to fill out the form. This 'HelperCard' is essential for users, meeting the functional requirement of providing instructions (requirement R9) and a brief explanation of results (requirement R10). Moreover, the 'HelperCard' supports a user-friendly and accessible experience, in line with non-functional requirements for good user experience R14, device responsiveness R18, and browser compatibility R17.

The design and implementation of the 'AnalysisForm' and 'HelperCard' components comprehensively fulfill the system's functional needs, such as selecting points of interest, interacting with the map, inputting accessibility model parameters, and choosing multicriteria analysis criteria and weights (requirements R1, R2, R3, R4). Additionally, the implementation supports the system's scalability, ensuring it can handle increased usage without major difficulties (requirement R19) and adheres to good software development practices, facilitating maintenance and future modifications (requirement R16).

The 'HelperCard' component provides general information about the accessibility model and instructions on using the tool (see Figures 6.4 and 6.5), making it a vital element for ensuring the system's functionality and user-friendliness.

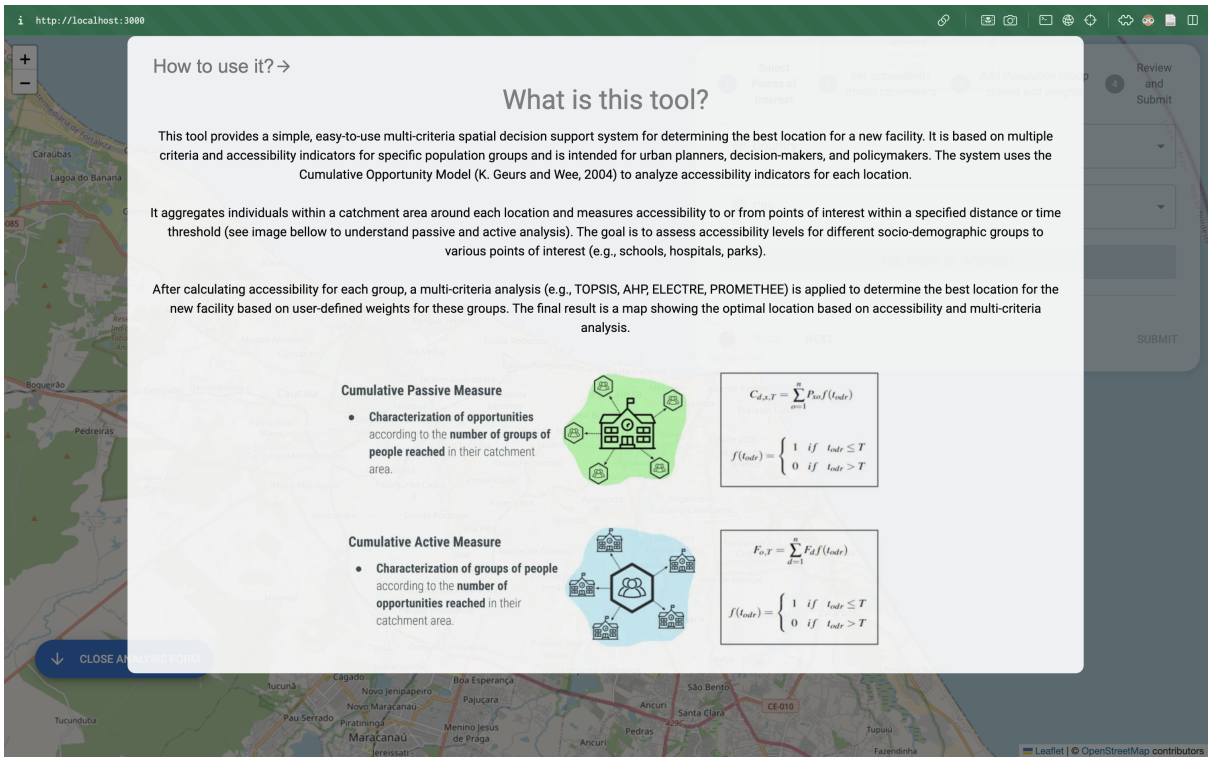

Figure 6.4: Screenshot of HelperCard page "What is this tool?" 1 of 2

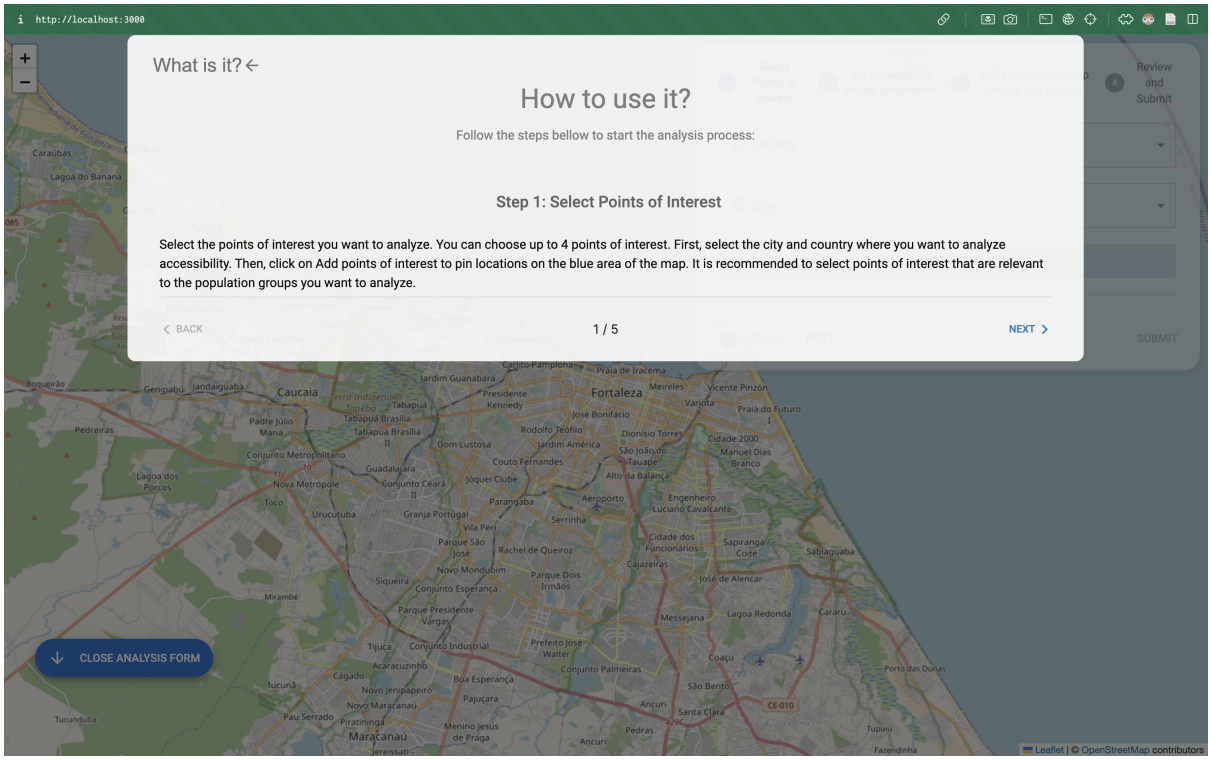

Figure 6.5: Screenshot of HelperCard page "How to use it?" 2 of 2

The 'Map' component, which renders an interactive map, enables user interaction and serves as a platform for visualizing results through map layers. This component will be explored in greater detail in the subsequent sections.

Rendering the Map

The interactive map is rendered using React Leaflet, a React library that provides a set of components for building interactive maps (Vladimir Agafonkin and Leaflet Contributors 2022). The Map component is a functional component that utilizes the 'MapContainer' component from React Leaflet to display the map, which uses the OpenStreetMap spatial database to render all geographical data. The code in Appendix K shows the Map component definition.

This component enables interactions such as pinpoints and hexagon selection, fulfilling the functional requirements of allowing the policymaker to choose different points of interest and interact with the map, as specified in requirements R1 and R2.

The Appendix O shows how the Map component is imported into the main Page component. The use of the dynamic import feature in Next.js ensures that the map component is only loaded on the client side, which addresses the limitation of the map components relying on the window object, available exclusively on the client side. This approach supports the non-functional requirement of compatibility with various web browsers, corresponding to requirement R17.

Furthermore, the useMemo hook from React is employed to memorize the Map component, preventing unnecessary re-renders and thereby enhancing the system's performance and responsiveness, aligning with the non-functional requirements of good user experience and responsiveness across different devices, as outlined in requirements R14 and R18.

Since all components need to trigger map events, they use a hook called 'useMapEvents', which handles all map events, such as click, zoom, and move events. This implementation ensures that

the system can efficiently manage user interactions with the map, supporting the non-functional requirement of scalability, as detailed in requirement R19.

To utilize this hook, components must be wrapped in the MapContainer component from React Leaflet, providing the map context to the components. To prevent mouse events from bubbling up to the map, the stopPropagation method from the event object is used. The form components are wrapped in a React element called PreventLeafletControl (see Appendix P), which calls stopPropagation whenever the mouse is over the 'AnalysisForm' component. This method ensures that the interactive map remains functional and user-friendly, adhering to the non-functional requirement of providing a smooth and intuitive user experience, as per requirement R14.

Other map layers will be discussed in the next subsections.

Fill up Location Form

The 'LocationForm' is an integral part of the 'AnalysisForm' component, allowing users to select the location for analysis. It includes two select boxes for the country and city, and once both are selected, a button is enabled to render the 'HexGrid Map' layer. This layer allows users to click on a hexagon, selecting a point that is then added to the location form. These interactions fulfill functional requirements R1 and R2, which require the system to allow policymakers to choose different points of interest and interact with the map through a graphical interface.

Figure 6.6 illustrates the hexgrid layer with selected points on the map, while Figure 6.7 shows the popup displaying demographic data for the selected hexagon.

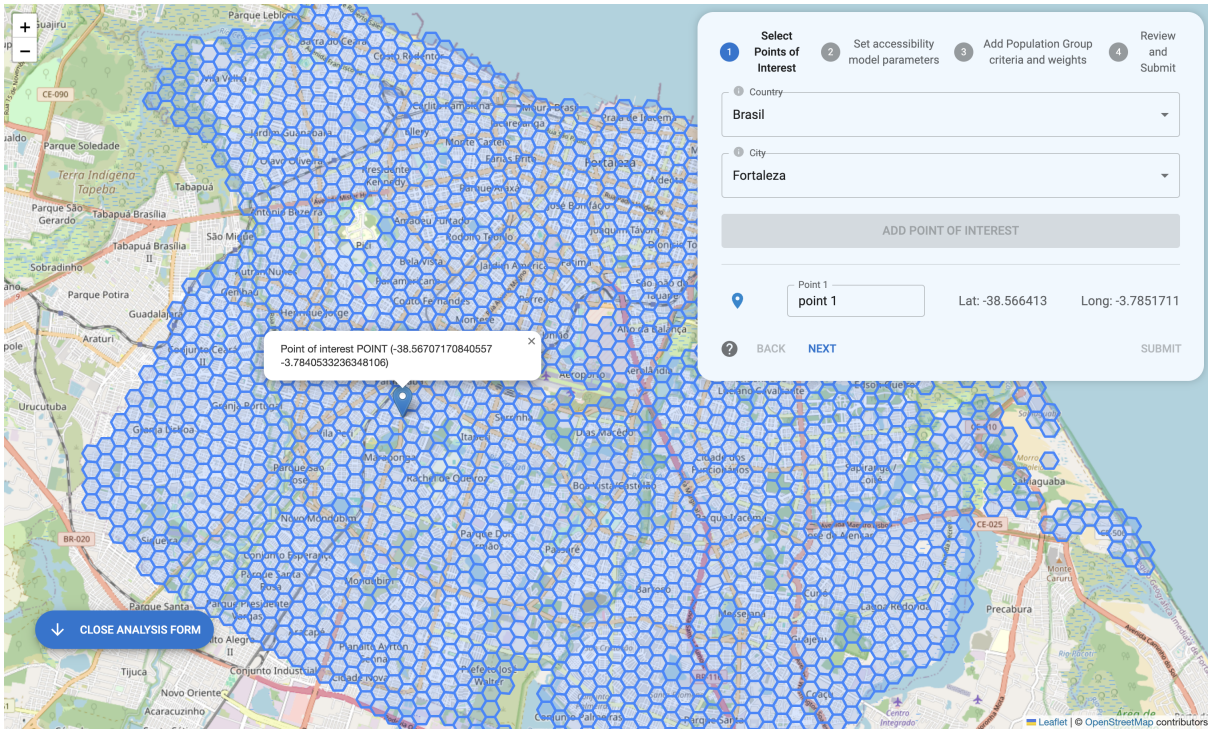

Figure 6.6: Screenshot of the proposed WebSDSS Point of Interest selected on the map

The hexagon grid is based on the segmentation discussed in Section 5.3.2. Along with the GeoJSON file used for accessibility analysis, a sectors demographic GeoJSON was generated for the city's boundaries using the script shown in the Appendix U. The GeoJSON file (see Appendix T) contains all hexagon geometries that ensures user-selected locations are within the city boundaries and align

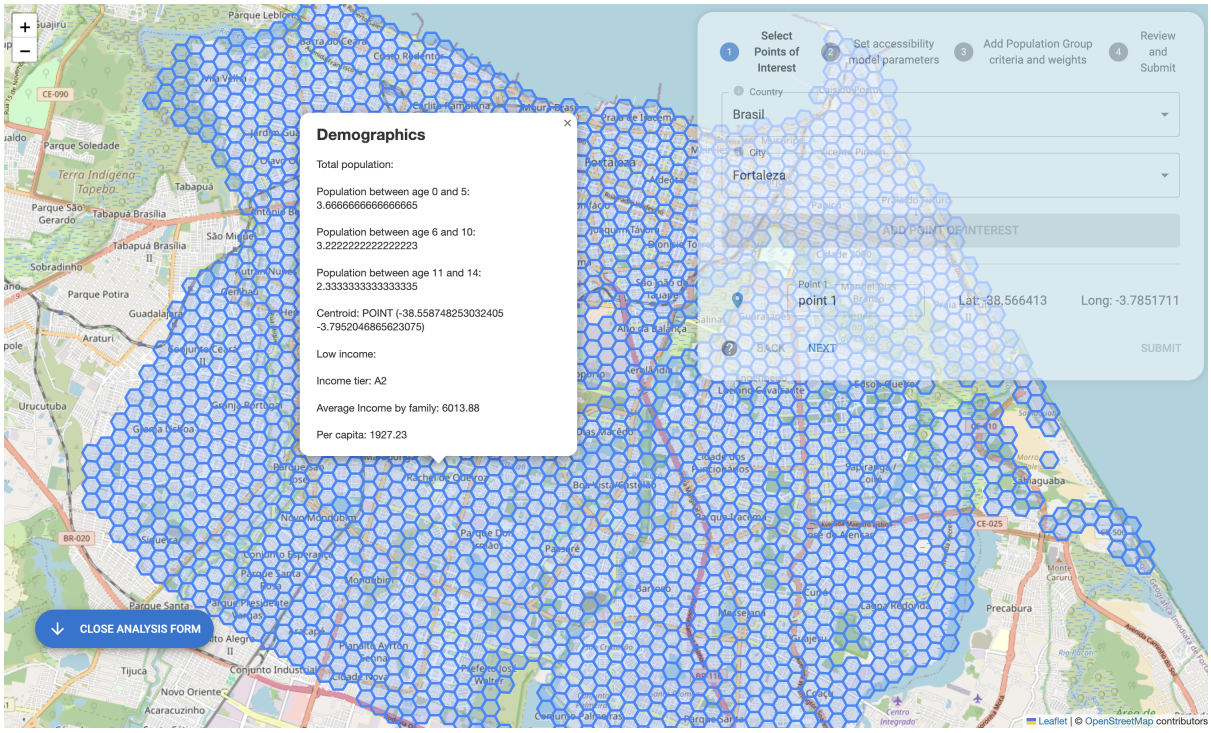

Figure 6.7: Screenshot of the proposed WebSDSS PopulationHexPopup component

with the hexagon grid defined in the accessibility analysis, supporting the non-functional requirement of accurate geographic data management as part of the system's overall robustness and reliability.

The hexagon grid is rendered using the HexGrid component (see code in the Appendix J), which implements the 'react-hexgrid' library. This library takes the GeoJSON file as input and renders the grid on the map as an overlay. This overlay allows users to interact with the map and select a point for analysis as defined in the 'onFeature' function, as depicted in Figure 6.6. Each hexagon contains demographic data, displayed when the user hovers over it, providing a user-friendly way to visualize complementary information.

After selecting a point on the map, it is added to the location form. Users have the flexibility to change the name or remove the location if desired, and they can proceed to the next step once at least two points are added. The server action file validates this data before advancing to the accessibility form, ensuring data consistency and preventing errors, thereby addressing non-functional requirements related to data validation and consistency (requirement R11). Users can also click on a location icon to centralize the map on the selected point, enhancing the overall user experience.

This implementation of the 'LocationForm' not only meets the functional requirements for selecting and interacting with locations on the map (requirements R1 and R2) but also ensures the system is user-friendly, responsive, and robust, adhering to non-functional requirements such as good user experience (requirement R14) and system reliability (requirement R16).

Fill up Accessibility Form

The accessibility form fulfills requirement R3 by providing users with the ability to select the accessibility parameters for the analysis. As part of the 'AnalysisForm' component, it offers a user interface divided into four key sections: a radio button group for choosing the accessibility model type, a radio group for selecting the year of the demographic data, a radio group for choosing the

transport mode, and a slide bar for setting the travel time threshold. Figure 6.8 illustrates the graphical interface of the accessibility form.

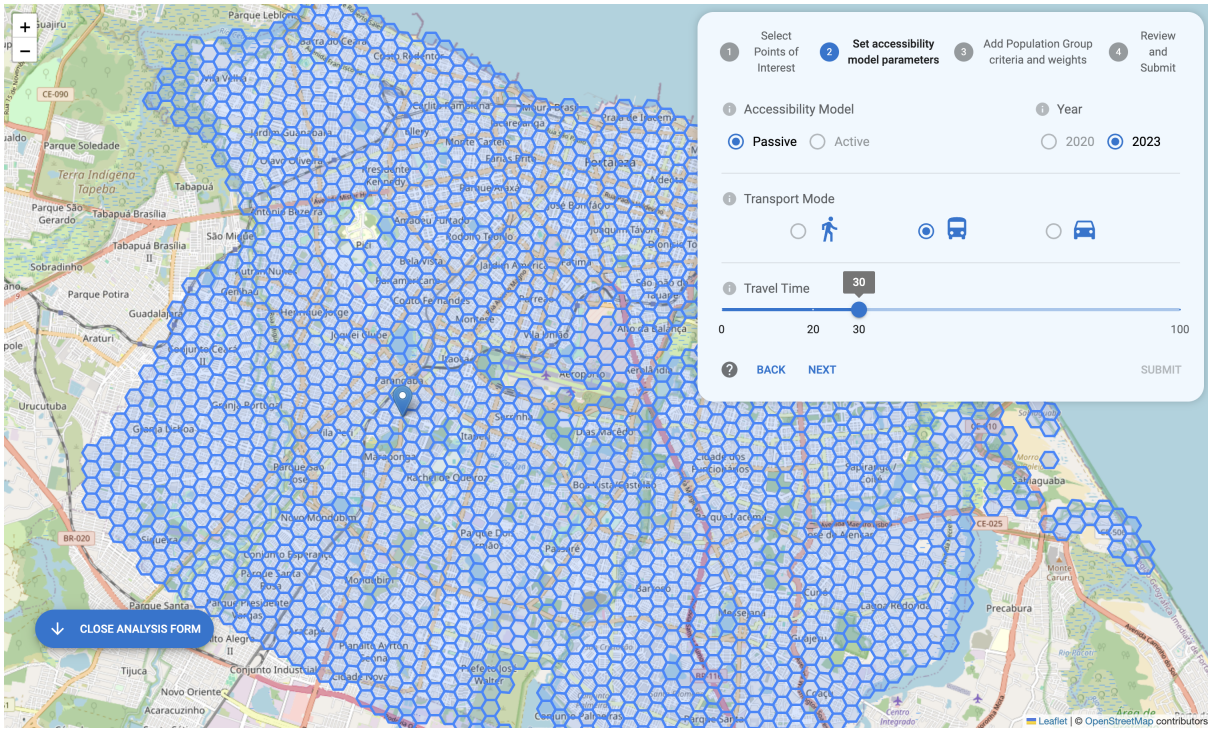

Figure 6.8: Screenshot of the proposed WebSDSS AccessibilityForm component

This component is designed to facilitate user interaction by presenting options in an intuitive manner, which directly supports the non-functional requirement of delivering a user-friendly experience, as specified in requirement R14. The use of clearly defined sections with radio buttons and sliders makes it easy for users to input their preferences, enhancing the system's accessibility and responsiveness across different devices, thereby fulfilling requirements R18 and R17 for device responsiveness and browser compatibility.

The accessibility form is composed of the frontend component and a server actions file that handles input data validation. This validation process ensures that the selected parameters meet the system's criteria before being submitted for analysis, aligning with non-functional requirements for data accuracy and consistency (requirement R11).

The form includes three select boxes for transport mode, time of day, and travel time threshold. The transport mode select box allows users to choose between options such as walking, cycling, or public transportation, addressing various accessibility scenarios. The time of day select box offers choices like morning, afternoon, or evening, enabling users to tailor the analysis to specific periods. The travel time threshold select box provides preset options such as 15, 30, or 45 minutes, allowing users to define the scope of the analysis. These features collectively contribute to a comprehensive and flexible user experience, ensuring the system can accommodate a wide range of user needs, supporting requirement R13 for ease of use.

The code for the accessibility form component is detailed in the Appendix (see Appendix A), where the implementation specifics are outlined, further demonstrating how this component is integral to the system's overall functionality and user experience.

Fill up Multicriteria Form

The multicriteria form follows the same structure as the other forms, consisting of a React component for the user interface and a server action that acts as a micro-backend for validating user input. This component fulfills requirement R4 by allowing users to capture all groups of criteria they wish to use in the analysis. Figure 6.9 illustrates the rendered multicriteria form.

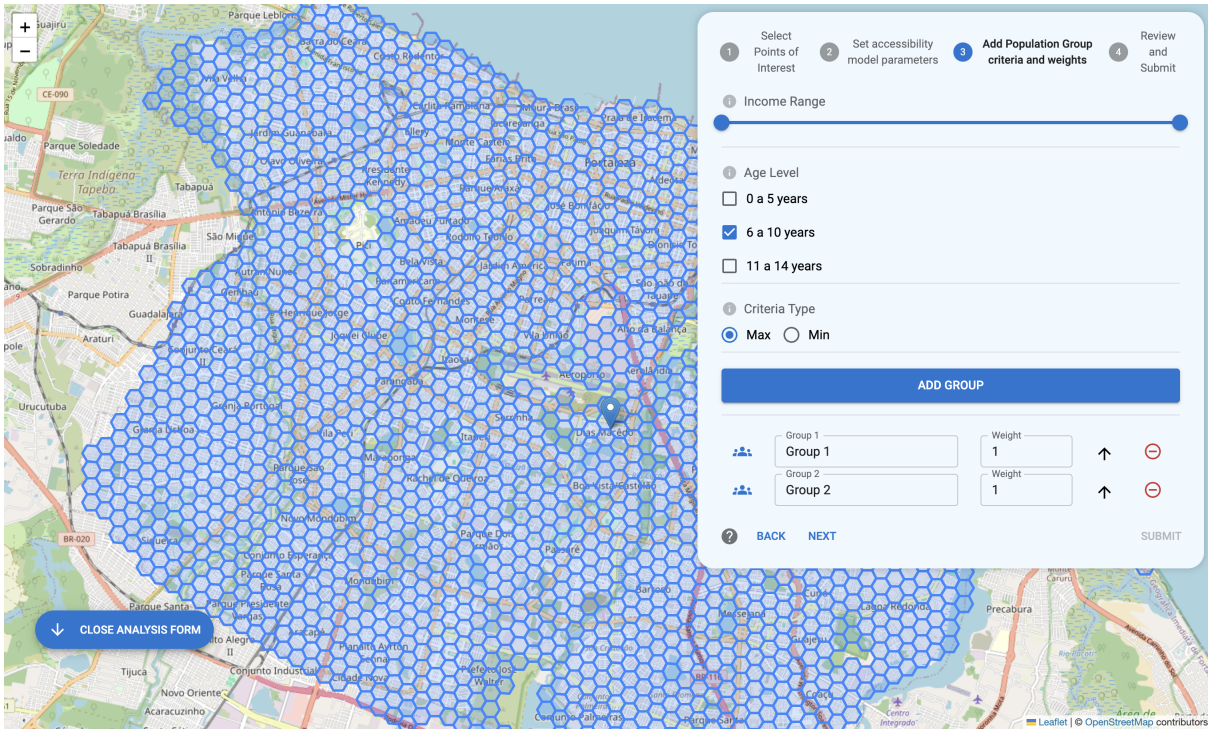

Figure 6.9: Screenshot of the proposed WebSDSS MultiCriteriaForm component

The form enables users to select multiple criteria groups based on population age level, income range, and criteria type. After specifying the parameters of a group, the user can add it by clicking the 'add group criteria' button. This interaction allows for flexible customization, addressing the functional requirement for criteria and weight selection in a multicriteria analysis, as outlined in requirement R4. Additionally, the ability to change the group's name and weight directly influences how the criteria are considered in the multicriteria algorithm, ensuring that the system remains adaptable to various analysis scenarios.

The multicriteria form also supports non-functional requirements related to user experience and data validation. By providing a clear and intuitive interface for selecting and managing criteria, the system ensures a positive user experience, meeting the non-functional requirement for good user experience (requirement R14). The form's structure, which allows users to easily modify or remove criteria groups, contributes to the system's overall usability and flexibility, supporting requirements R13 and R18 for ease of use and responsiveness across different devices.

Furthermore, the server action associated with this form validates the user input before the data is processed in the multicriteria analysis. This validation ensures that the data meets the necessary standards, supporting the non-functional requirement of data accuracy and consistency (requirement R11).

The code for the MultiCriteriaForm component is detailed in Appendix M, providing a comprehensive overview of how this component is implemented to fulfill both functional and non-functional requirements, contributing to the system's effectiveness and reliability.

6.4.2 Submit Analysis Form

The submission of the analysis form covers requirements R5, R6, and R10. This process involves both frontend and backend components, with the frontend responsible for sending form data to the backend, and the backend processing this data to return results. This section details the ReviewForm component and the API endpoint, both critical in calculating accessibility values and conducting multicriteria analysis, thus fulfilling key system requirements.

Review Form

The 'ReviewForm' component (see code in Appendix S) is designed to display all the data entered by the user in previous steps, allowing for a thorough review before submission. Users can review location, accessibility, and multicriteria data within a structured interface. This component plays a crucial role in fulfilling requirement R10, which mandates that the system provide a brief explanation of the results obtained and ensure that users have the opportunity to confirm their inputs before submission.

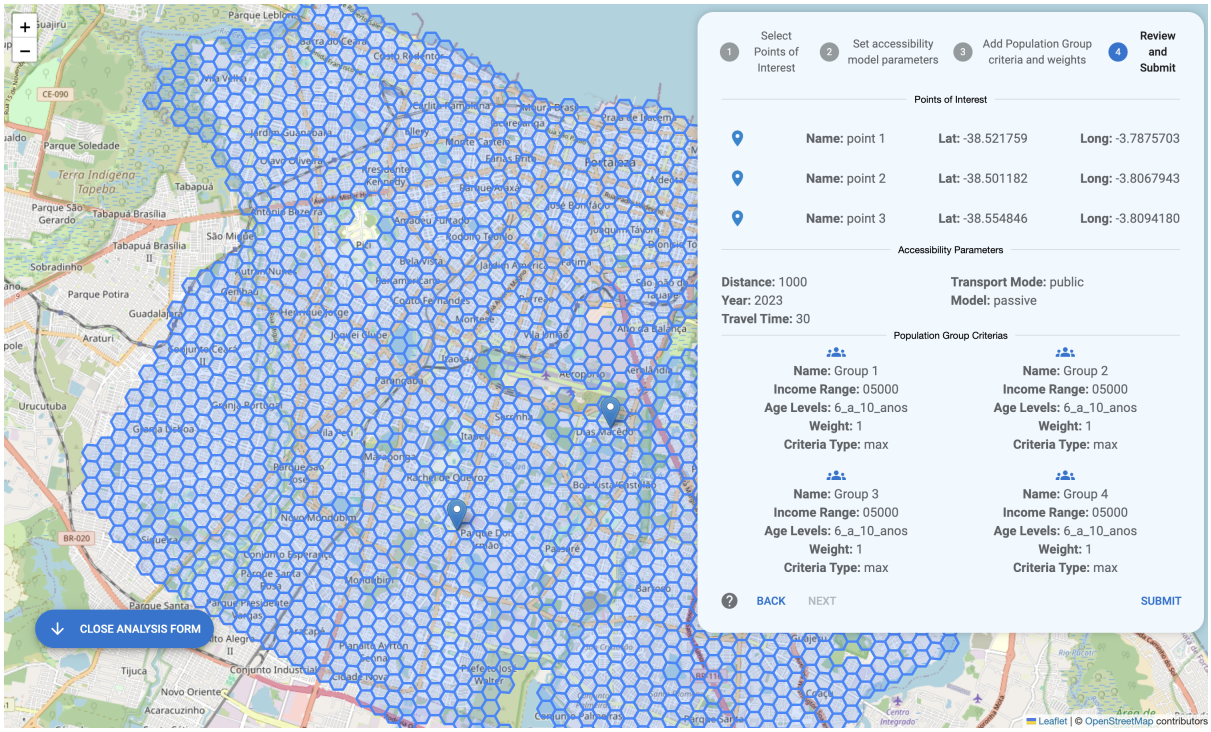

Figure 6.10: Screenshot of the proposed WebSDSS ReviewForm component

After reviewing the form data, users can click the submit button to finalize their submission. This action triggers the 'onSubmit' function, which sends the form data to the backend for processing via a POST request. The 'onSubmit' function (see code in Appendix N) utilizes the Fetch API, facilitating HTTP requests directly from the browser. This setup not only handles the form submission but also ensures that data is transmitted securely and efficiently, supporting non-functional requirements related to data transmission and system performance.

API

An API route, following Next JS conventions, was implemented under the 'api/accessibility' directory to manage form submissions, generating the endpoint '/accessibility'. This API route (see code in Appendix D) is responsible for receiving form data, processing it to calculate accessibility values, and performing multicriteria analysis. The results are then returned to the client as an 'AnalysisResult' object, parsed as JSONṪhis setup is essential for fulfilling requirement R5 and R6, which focus on the system's ability to calculate values and conduct multicriteria analysis effectively.

The API employs two main methods that interact with backend modules: one for accessibility analysis and another for multicriteria analysis. This modular approach ensures that the system remains scalable and maintainable, aligning with non-functional requirements like system scalability (requirement R19) and adherence to good software development practices (requirement R16).

Calculate Accessibility Values

The accessibility analysis module calculates the accessibility values based on the form data, directly addressing requirement R5. This calculation involves summing up the population of hexagons within the travel time threshold, depending on the transport mode and time of day selected by the user. The relevant hexagons are chosen from the accessibility analysis matrix (see Section 5.3.2) and their populations are summed to produce the final accessibility values. These values are then used as input for the multicriteria analysis module.

The code in Appendix B shows the main functions responsible for this process:

- **readData:** Reads and processes a travel-time matrix created from the process described in 5.3.2. This matrix is represented by a GeoJSON file (see Appendix H) asynchronously, taking 'hexLocations', 'groupCriteria', and 'accessibilityOptions' as parameters, and returns an 'AccessibilityAnalysisResult'. It processes each data entry individually and groups the results accordingly.
- **processData:** Evaluates and transforms each data entry based on the provided criteria, updating the 'criteriaMatrix' and returning it.
- **groupResults:** Consolidates the processed results into an 'AccessibilityAnalysisResult' by aggregating data and summarizing the outcomes.

These functions ensure that the system accurately calculates accessibility values, meeting the necessary data accuracy and consistency standards (requirement R11).

Apply Multicriteria Analysis Algorithm

The multicriteria analysis module applies the TOPSIS algorithm (Madanchian and Taherdoost 2023) to the accessibility values result matrix, fulfilling requirement R6. This method evaluates alternatives based on multiple criteria by calculating the geometric distance between each alternative and the ideal and negative ideal solutions, ranking them accordingly (Madanchian and Taherdoost 2023). This process ensures that the system can provide a ranked list of optimal points of interest based on user-defined criteria.

The code in Appendix L shows how the 'apply' function performs this analysis. It receives a matrix of accessibility values and criteria weights, normalizes the matrix, calculates the weighted normalized matrix, and ranks the alternatives based on the calculated distances. The results are returned as an object that effectively communicates the ranked alternatives, supporting the system's requirement for data accuracy and effective decision-making.

6.4.3 Display Analysis Results

The analysis results are displayed in two primary ways: through a map layer and a report. The map layer visually represents results on the map, showing which hexagons were reachable and pinpointing points of interest. Additionally, the same container that renders the analysis form displays a review of the results, including information regarding the locations, ranked from best to worst based on the criteria. This implementation fulfills requirements R7 and R8, which pertain to rendering the results on a map and displaying them in a tabular format.

Render Results Map Layer

The map result layer is implemented in the 'ResultGrid' component (see code in Appendix Q). This component receives the results from the accessibility and multicriteria analyses and renders all reachable hexagons on the map. Figure 6.11 demonstrates how the results are displayed.

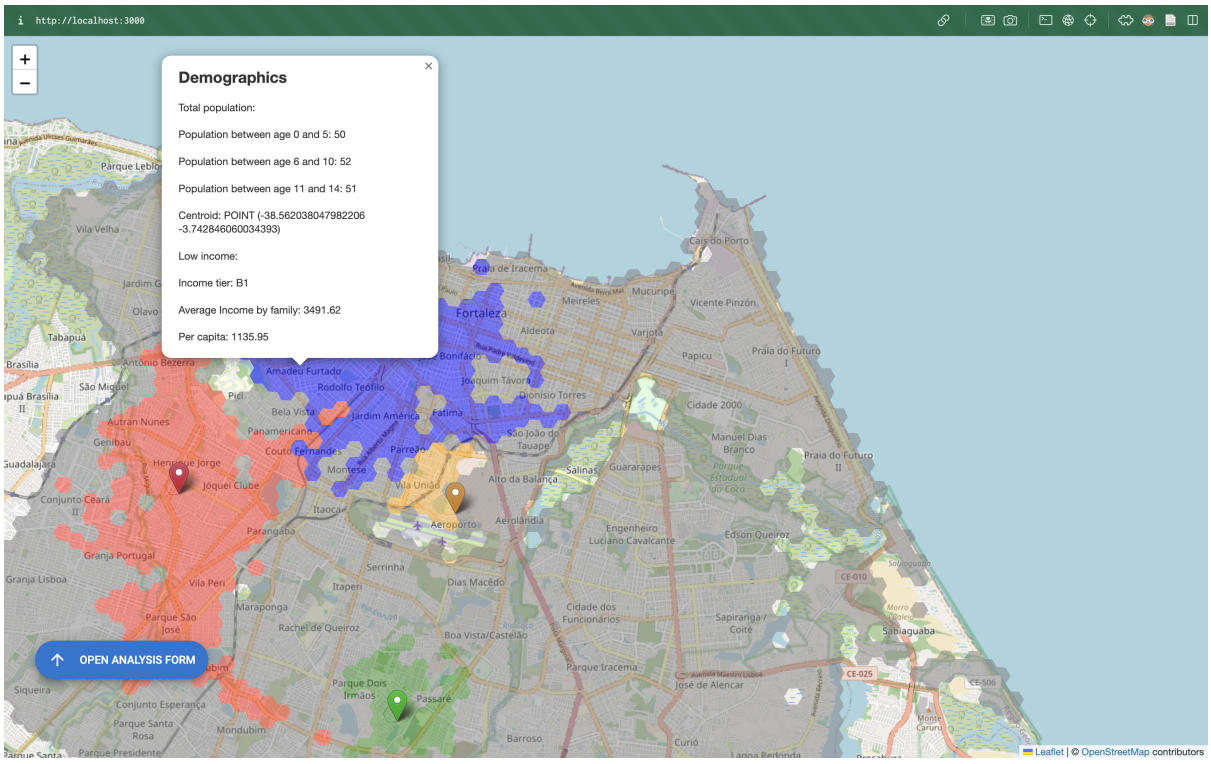

Figure 6.11: Screenshot of the proposed WebSDSS ResultGrid map layer component

The results are color-coded to distinguish between points of interest and the hexagons reached by them. Each hexagon shows demographic data when moused over, fulfilling requirement R7 by providing a detailed and interactive map-based visualization of the analysis results.

Render Results Report

The results report is rendered in the same container where the forms are displayed, using the 'ResultsForm' component (see code in Appendix R). This component presents the results from the accessibility and multicriteria analyses in a table format. Figure 6.12 shows the graphical interface for the results, with locations and analysis results ordered from best to worst based on the multicriteria analysis values.

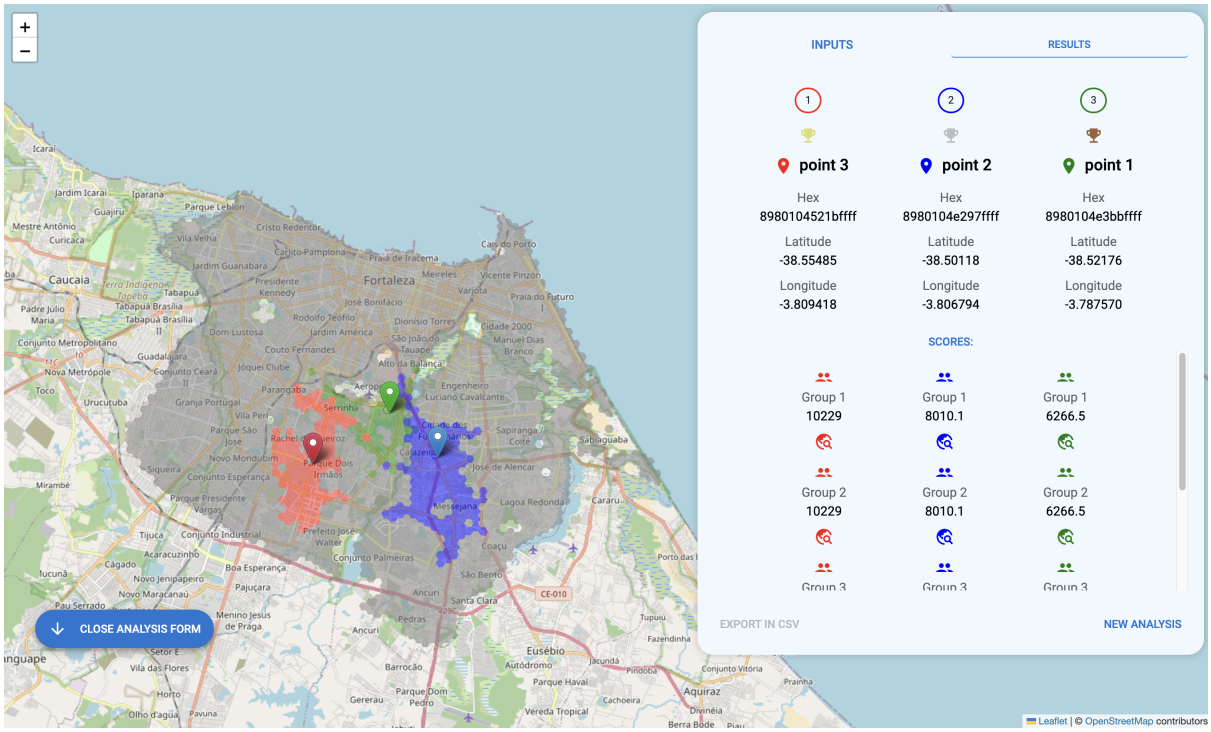

Figure 6.12: Screenshot of the proposed WebSDSS ResultForm component

Users can click on a location name to view it on the map, with each result identified by a color representing the selected location. The 'ResultsReport' also utilizes the 'ReviewForm', allowing users to switch between viewing results and input data, thereby fulfilling requirements R8 and R10. This feature helps users understand the relationship between the input data and the analysis outcomes, contributing to the overall user experience and supporting the system's transparency and comprehensibility.

Chapter 7

Evaluation of the Solution

After implementing and deploying the solution, it is crucial to evaluate its effectiveness. This evaluation process aims to verify whether the predefined requirements have been fully met and to assess the quality of the proposed solution. To achieve this, research hypotheses related to the problem are formulated, along with corresponding indicators and sources of information. This chapter outlines the evaluation methodology and concludes with an analysis of the results obtained from various evaluation methods.

7.1 Research Hypotheses

A hypothesis is a potential answer to a research question. It is a guess or presumption upon which a study is conducted. This hypothesis is tested for possible approval or rejection. If accepted, it demonstrates that the presumption was correct ReadingCraze 2015. In this context, the hypotheses are formulated based on the research questions and the objectives of the study. The hypotheses and their corresponding research questions are presented below:

- **RQ1:** How do accessibility and socioeconomic indicators affect the distribution of public facilities and urban infrastructure?
  - **H1:** The proposed solution can incorporate accessibility and socioeconomic indicators as valid criteria for urban planning decisions.
- **RQ2:** What impact can a Decision Support System have on urban planning?
  - **H2:** The proposed solution can address gaps in the currently available spatial decision support system platforms within the urban planning domain.
- **RQ3:** What are the uses of a Decision Support System in urban planning?
  - **H3:** The proposed solution serves as a valuable decision support tool for selecting facility locations.

The purpose of these hypotheses is to demonstrate that the proposed solution is robust, free from errors or failures, and performs as expected, aligning with the initially defined objectives. It is important to note that the validity and completeness of these hypotheses will be assessed through various methodologies, including the Quantitative Evaluation Framework (QEF) and the System Usability Scale (SUS), which will be discussed in greater detail later in this chapter.

7.2 Indicators and Sources of Information

In the previous section and the value analysis section, it is possible to identify two characteristics that can be evaluated. One of them is related to the usability of the final product, an aspect that is very important for the system to deliver value, as the QFD model suggests. The evaluation

process consists of creating a questionnaire. This questionnaire, based on the SUS evaluation method, should be applied to individuals working in the field of public urban planning, mainly policymakers. Based on the questionnaire responses, it is possible to evaluate how complete the usability requirements are.

Another aspect that should also be subject to evaluation concerns the degree of quality of the final solution. This characteristic is evaluated by a policymaker with experience in urban planning and also with digital tools using the QEF assessment model. The chosen indicators are deliberately selected because they are directly related to the previously formulated research hypotheses.

The quality indicator evaluates both the functional and non-functional aspects of the solution, aiming to validate the statements made by hypotheses H1 and H2 in previous Section, as they focus on quality aspects such as scalability, performance, and provided functionalities. On the other hand, the usability indicator, which addresses concerns of navigability, accessibility, and comprehension, is more suitable to address research hypothesis 7.1, as it focuses on the usability of the system as a SDSS in the context of urban planning and the public sector, which is the main target audience of the solution.

7.3 Evaluation Methodology

This section presents the tools and processes used to evaluate usability, quality, and performance metrics. In this regard, the evaluation processes imposed on the final solution are contextualized and detailed.

7.3.1 Software Testing

Software testing is a process that aims to verify and validate the software, ensuring that it meets the requirements and specifications defined during the development phase. The testing process is essential to ensure that the software is free from errors and that it performs as expected. It is commonly divided into three categories: unit testing, integration testing, and end-to-end testing (IBM 2023).

The testing pyramid, Figure 7.1, as described by Fowler (2012), emphasizes having many unit tests, fewer integration tests, and the fewest end-to-end tests. Unit tests, at the base of the pyramid, focus on testing individual components or functions in isolation. Integration tests, in the middle layer, verify that different components work together correctly. At the top are end-to-end tests, which simulate real user scenarios to ensure the entire system functions as intended.

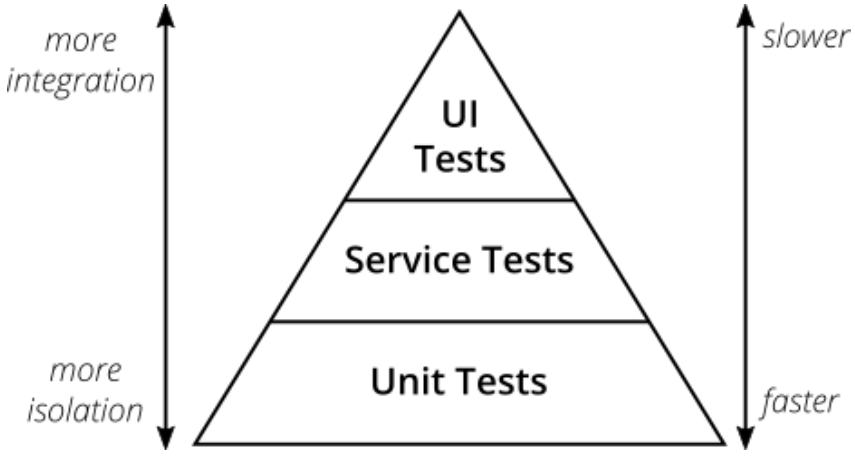

Figure 7.1: Testing Pyramid (Fowler 2012).

The unit and integration tests run alongside the script presented in Section 6.3.1 and are written using the Jest framework. The end-to-end tests are done manually after the deploy preview build is generated.

Unit Testing

As mentioned, unit tests focus on the smallest units of the software, such as functions and methods.

The unit tests were written to verify the behavior of the main UI components, ensuring that they perform as expected. They were also written to verify some functions used by the accessibility and multicriteria modules. These unit tests are focused on fullfilling the following requirements: R11, R13, R5, R6, specically for AnalysisComponent, its functionality impacts the requirements R1, R2, R3, R4.

The author opted to mock the API calls and other child React component in the case of user interface files, as the focus of the tests is on the frontend components. Following, what is tested in each main component:

- **Analysis Component:** The unit tests for the analysis component verify that the form is rendered correctly and that the form fulfillment, validation, and submission are handled correctly (see Code X.2).
- **Map Component:** The map component is responsible for rendering the map and the facilities on the map. The unit tests for the map component verify that the map is rendered correctly and that the facilities are displayed on the map (see Code X.3).
- **Accessibility Module:** The accessibility module is responsible for calculating the accessibility indicators for each facility. The unit tests for the accessibility module verify that the accessibility indicators are calculated correctly (see Code X.1).
- **Multicriteria Module:** The multicriteria module is responsible for calculating the multicriteria indicators for each facility. The unit tests for the multicriteria module verify that the multicriteria indicators are calculated correctly (see Code X.4).

Integration Testing

Integration tests focus on the interaction between different units of the software. In this case, they are represented by tests of the API endpoint that is responsible for receiving the data from the frontend and calculating accessibility and multicriteria results to return to the backend. The tests verify that the API endpoint is working correctly and that the data is being sent and received as expected. The requirements verified by the integration tests are R19, R15 and R12.

The following scenarios were tested and can be seen in the Appendix Y:

- With valid user input data;
- With invalid user input data;
- With empty input data.

End-to-End Testing

End-to-end tests focus on the entire system, simulating the user's interaction with the software. The author opted not to implement an automated end-to-end test solution; these tests are performed manually after the deploy preview build is generated (see Section 6.3.2).

These manual tests verify the fulfillment of most of the non-functional requirements defined in Section 3.3.2, such as the system's performance, scalability, and security.

The end-to-end test scenarios are as follows:

- Test map interactions such as pin-points, drag and zoom;
- Fill up the analysis form;
- Submit the analysis form;
- Verify the results of the analysis.

7.3.2 Quantitative Evaluation Framework (QEF)

The QEF, developed by Paula Escudeiro and José Birrada (Escudeiro and Bidarra 2008), compares the real system with an ideal system. The real system refers to the current state of the solution, while the ideal system represents the ideal state that the final solution should achieve. This comparison involves placing both solutions in a three-dimensional space to calculate the Euclidean distance between them.

According to this model, the quality of the system is inversely proportional to the distance between the ideal system and the real system (Escudeiro and Bidarra 2008). This means that the greater the calculated distance, the further the real solution is from the ideal solution. The QEF is based on three essential concepts: Dimension, Factor, and Requirement. These concepts follow a hierarchy: a dimension is composed of factors, which in turn are composed of requirements. The dimensions are interconnected with the final quality of the solution, as the quality is directly related to the performance level of each dimension.

The factors in the evaluation are characterized by two aspects:

- Importance: Measures the significance of the factor by dividing the sum of the relevance degrees of the associated requirements by the sum of the relevance degrees of all requirements in the dimension where the factor is located.
- Compliance: Assesses the fulfillment of the factor by dividing the sum of the compliance evaluations of the requirements by the total compliance value of the requirements within the factor.

Similarly, the requirements in the evaluation are characterized by two aspects:

- Relevance: Indicates the level of importance of the requirement and is classified on a scale of 0, 2, 4, 6, 8, and 10, with 10 representing the highest level of relevance.
- Evaluation: Represents the degree of fulfillment of a requirement and is classified on a percentage scale of 0, 25, 50, 75, and 100, with 100 being the highest level of compliance.

For this project, based on the requirements defined in Section 6.3.2, three dimensions were defined: Functionality, Usability, and Maintainability. Each dimension is composed of factors and requirements, which are evaluated according to the relevance and compliance criteria as seen in Table 7.1 and explained in the following sections.

Functionality

Functionality is a crucial dimension that ensures the software system performs the tasks and operations required by its users. It encompasses the following factors:

Dimension	Factors	Weight	Score	Requirements
Functionality	Functional Completeness	0.6	10	R1, R2, R3, R4, R5, R6, R7, R8, R20, R21
	Functional Correctness	0.2	6	R11
	Functional Appropriateness	0.2	8	R10
Usability	Appropriateness Recognizability	0.3	8	R9, R10, R14
	Learnability	0.2	6	R9, R10
	Operability	0.2	10	R13
	User Error Protection	0.2	6	R11
	Accessibility	0.1	4	R18
Maintainability	Modularity	0.3	10	R15
	Analyzability	0.3	8	R16
	Modifiability	0.2	6	R15
	Testability	0.2	6	R16

Table 7.1: Evaluation Metrics and Corresponding Requirements

- **Functional Completeness**: This factor measures the extent to which the software provides all the necessary functions as specified by the requirements.
- **Functional Correctness**: This factor evaluates the software's ability to provide correct results and outputs as per its specifications.
- **Functional Appropriateness**: This factor assesses the suitability of the functions provided by the software for their intended purposes.

Usability

Usability focuses on the ease with which users can learn to use and effectively interact with the software. It includes the following factors:

- **Appropriateness Recognizability**: This factor considers how easily users can recognize the software's capabilities and how well it meets their needs.
- **Learnability**: This factor measures how easy it is for users to learn to use the software.
- **Operability**: This factor evaluates the ease of use and control of the software's operations.
- **User Error Protection**: This factor assesses the software's ability to prevent and correct user errors.
- **Accessibility**: This factor considers the software's accessibility across different devices and assistive technologies.

Maintainability

Maintainability ensures that the software can be easily modified, improved, and adapted. It covers the following factors:

- **Modularity**: This factor measures the extent to which the software's components can be separated and recombined.
- **Analyzability**: This factor evaluates how easily defects and issues can be diagnosed and fixed.

- **Modifiability**: This factor assesses the effort required to make specified modifications to the software.
- **Testability**: This factor considers how easily the software can be tested to ensure it functions correctly.

7.3.3 System Usability Scale (SUS)

To measure the usability level of the solution, SUS evaluation method was chosen. Developed by Bangor, Kortum, and Miller 2008, this method involves a questionnaire that allows users of a particular product or service to evaluate it quickly and easily.

The reasons for choosing this method include its suitability for computer interfaces (e.g., websites), its relatively quick and easy usage, and the provision of a single score on a scale that is easy to understand.

The scale comprises five ratings:

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

The questionnaire itself consists of 10 questions, which can be seen in Table 7.2. The target audience for this questionnaire includes individuals involved in urban planning and decision-making in this field. Any unanswered questions should be assigned a value of three when interpreting the results.

#	Statement
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

Table 7.2: System Usability Scale (SUS) Statements

Once the questionnaire is completed, the next step is to interpret the results. For positively connoted questions (1, 3, 5, 7, and 9), subtract 1 from their respective values. Conversely, for negatively connoted questions (2, 4, 6, 8, and 10), calculate 5 minus the value of the respective response. Sum all the adjusted values and multiply the result by 2.5 to obtain the overall SUS score. This score helps draw conclusions regarding the system's usability.

If the obtained score is below 50, it indicates severe usability issues. The best possible scenario would include a score very close to 100, although this is unlikely. A more reasonable score would range between 7585 points.

Lastly, the following steps should be taken by participants before completing the questionnaire:

- Navigate to the main page of the final solution;
- Fill out the facility location analysis form;
- Interact with the map, filling in the required fields;
- Review the provided inputs and submit the form;
- Review the analysis results through the generated report;
- Review the results on the map.

The scenario consisting of the steps above focuses primarily on interacting with the interactive map, filling out the facility location analysis form, and analyzing the results. These are the three most important aspects of the final solution and should therefore be evaluated by the participants.

7.4 Results Analysis

This section aims to determine whether the methods applied to the previously discussed evaluation methodology have produced results capable of validating the initially set objectives.

7.4.1 Software Testing Results

From the tests conducted on the final solution, whether they were integration tests, load tests, security tests, or end-to-end tests, most only validated success scenarios with few failure scenarios. This indicates that there are several other scenarios with alternative flows that were not tested, leading to the conclusion that, although the tests validated the 'happy path', they were not as exhaustive as initially intended by the author.

The absence of a testing methodology like Test-Driven Development (TDD) (Tosun et al. 2018) should also be noted, which could have been implemented to ensure that all scenarios were tested. This would not only ensure a reliable software development flow as proposed by requirement R16 but also guarantee that the software was developed more efficiently and with fewer errors.

It is also important to mention the lack of automation in the end-to-end tests that could have been conducted using tools such as Cypress (Cypress.io 2023) or Selenium (Selenium 2023). Automating end-to-end tests is a common practice in software development projects and is a way to ensure that the software behaves as expected in a wide variety of scenarios.

Despite this, the results obtained during the testing phase were positive, as they ensured that the system behaved as expected in the majority of circumstances.

7.4.2 QEF Results

Based on the level of fulfillment of the requirements, Table 7.3 shows the results of the evaluation metrics of QEF analysis.

Some factors scored less than 100% due to the fact that some requirements were not fully implemented. In the Usability dimension, the Accessibility factor scored 50% because the solution does not provide an interface adaptable for different devices. The User Error Protection factor scored 70% due to insufficient error handling mechanisms. The Learnability, Operability, and Appropriateness Recognizability factors scored 80% because the solution is very specific to the urban planning domain and uses different analysis models that can be difficult to understand. The Analyzability and Testability factors in the Maintainability dimension scored 75% because the

Dimension	Factors	Weight	Score	Max Score	Compliance (%)
Functionality	Functional Completeness	0.6	75	100	75
	Functional Correctness	0.2	80	100	80
	Functional Appropriateness	0.2	80	100	80
Usability	Appropriateness Recognizability	0.3	80	100	80
	Learnability	0.2	80	100	80
	Operability	0.2	80	100	80
	User Error Protection	0.2	70	100	70
	Accessibility	0.1	50	100	50
Maintainability	Modularity	0.3	85	100	85
	Analyzability	0.3	75	100	75
	Modifiability	0.2	85	100	85
	Testability	0.2	75	100	75

Table 7.3: Evaluation Metrics results for QEF analysis

solution could be more easily analyzed and tested, and the interactive map complicates the UI testing process.

Compliance Calculations

Compliance calculations are performed by multiplying the weight of each factor by the score obtained for that factor. The results are then summed to obtain the compliance percentage for each dimension. The formula for calculating the compliance percentage is described below:

$$\text{Dimension Compliance} = \sum_{i=1}^{n} (w_i \times c_i) \tag{7.1}$$

The compliance percentage for each dimension is calculated as follows:

Functionality Dimension

$$\begin{aligned} \text{Functionality Compliance} = (0.6 \times 75) + (0.2 \times 80) + (0.2 \times 80) \quad = \\ 45 + 16 + 16 = 77\% \end{aligned} \tag{7.2}$$

Usability Dimension

$$\begin{aligned} \text{Usability Compliance} = (0.3 \times 80) + (0.2 \times 80) + (0.2 \times 80) + (0.2 \times 70) + (0.1 \times 50) \\ = 24 + 16 + 16 + 14 + 5 = 75\% \end{aligned} \tag{7.3}$$

Maintainability Dimension

$$\begin{aligned}\text{Maintainability Compliance} &= (0.3 \times 85) + (0.3 \times 75) + (0.2 \times 85) + (0.2 \times 75) \\ &= 25.5 + 22.5 + 17 + 15 = 80\%\end{aligned} \tag{7.4}$$

Overall Real Quality Percentage

$$\begin{aligned}\text{Real Quality Percentage} &= \frac{77 + 75 + 80}{3} \quad = \\ &\frac{232}{3} \approx 77.33\%\end{aligned} \tag{7.5}$$

Euclidean Distance to Ideal System

The Euclidean distance to the ideal system is calculated by the formula below:

$$d = \sqrt{\sum_{i=1}^{n} w_i \times (I_i - R_i)^2} \tag{7.6}$$

It is calculated by summing the squared differences between the real quality percentage and the ideal quality percentage for each dimension. The square root of the sum is then taken to obtain the final distance.

$$\begin{aligned}d = &\sqrt{0.6 \times (100-75)^2 + 0.2 \times (100-80)^2 + 0.2 \times (100-80)^2 + 0.3 \times (100-80)^2} \\ &+ 0.2 \times (100-80)^2 + 0.2 \times (100-80)^2 \\ &+ 0.2 \times (100-70)^2 + 0.1 \times (100-50)^2 + 0.3 \times (100-85)^2 \\ &+ 0.3 \times (100-75)^2 + 0.2 \times (100-85)^2 + 0.2 \times (100-75)^2 \\ = &\sqrt{0.6 \times 625 + 0.2 \times 400 + 0.2 \times 400 + 0.3 \times 400} \\ &+ 0.2 \times 400 + 0.2 \times 400 + 0.2 \times 900 + 0.1 \times 2500 + 0.3 \times 225 \\ &+ 0.3 \times 625 + 0.2 \times 225 + 0.2 \times 625 \\ = &\sqrt{375 + 80 + 80 + 120} \\ &+ 80 + 80 + 180 + 250 + 67.5 \\ &+ 187.5 + 45 + 125 \\ = &\sqrt{1570} \approx 39.62\end{aligned} \tag{7.7}$$

Summary:

- **Functionality Compliance**: 77%
- **Usability Compliance**: 75%
- **Maintainability Compliance**: 80%
- **Real Quality Percentage**: 77.33%
- **Euclidean Distance to Ideal System**: 39.62

Although the solution has an acceptable percentage of compliance, the Euclidean distance to the ideal system is high, indicating that the solution is not yet close to the ideal system. This is due to the fact that some requirements were not implemented or did not achieve the expected quality level.

Nevertheless, the author considers that the results obtained in the QEF analysis are positive, as they demonstrate that the solution meets the main requirements and needs of end users.

7.4.3 SUS Results

The usability evaluation of the solution is conducted through the responses obtained in the system usability questionnaire as seen in Section 7.3.3. The questionnaire involved the participation of 10 individuals who predominantly work in the field of urban planning and decision-making, which involves city development. The ratings obtained for each question are illustrated in the bar chart in Figure 7.2.

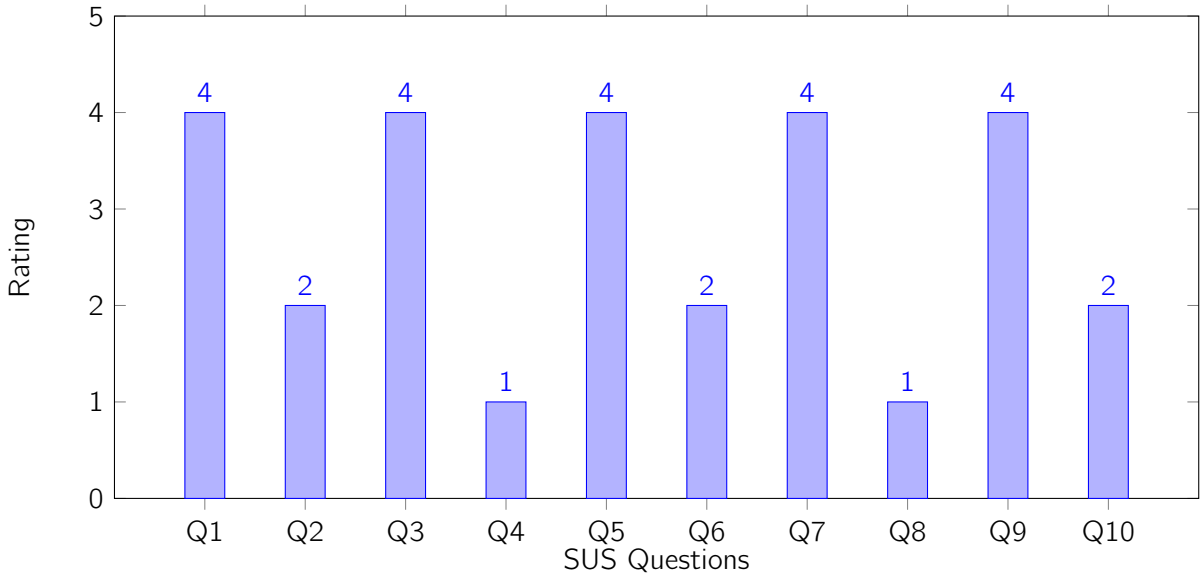

Figure 7.2: SUS Questionnaire Results

It should be noted that for odd-numbered questions, a higher score indicates a better result, with a score as close to 5 as possible. Conversely, for even-numbered questions, a better result involves a score as close to 1 as possible.

With this in mind, and generally speaking, the author considered that based on the responses obtained, it is possible to conclude that these represent positive results concerning the system's usability. However, to finalize the SUS evaluation, it is necessary to calculate the overall system score. This calculation can be performed using the expression below:

$$P = \frac{A + B}{0.4} \tag{7.8}$$

Where A is the sum of the average scores for the odd-numbered questions, subtracting 1 from each average, and B is the sum of the scores for the even-numbered questions, with the value of each average being 5 minus the score. The obtained score is **80.625**, which indicates that the system is considered 'excellent' according to the SUS rating scale.

7.4.4 Investigation Hypotheses Results

At the beginning of this chapter, specifically in Section 7.1, research hypotheses were formulated with the intent of demonstrating that the final solution behaves as expected according to the initially set objectives. Thus, through the indicators and sources of information expressed in Section 7.2, a relationship was established between the research hypotheses and the said indicators.

Based on the results analysis conducted so far, which involves the quality and usability indicators, it is now necessary to verify the results obtained concerning the research hypotheses. These results are explicitly presented in the following list:

- H.1: As the final solution provides a set of functionalities that allow the use of sociodemographic indicators to define evaluation criteria, the statement proposed by the hypothesis is confirmed.
- H.2 and H.3: In an evaluation based solely on the results of the presented indicators, as they show a positive aspect, these hypotheses are shown to be true. However, the author considers that the presented evaluation is insufficient to confirm these hypotheses, given that there is a broader dimensionality to be considered, such as how the political and social context influences decision-making in urban planning.

Finally, based on the results obtained in this subsection, it is concluded that the scenarios proposed by the research hypotheses are partially achieved in the final solution.

Chapter 8

Conclusion

This chapter summarizes the study's findings, highlighting the strengths and limitations of the proposed Web-based Spatial Decision Support Systems (WebSDSS), and suggests future research directions. It reflects on the project's work and results, lists achieved objectives, discusses limitations and potential improvements, and provides a personal assessment of the development process.

8.1 Objectives Achieved

The initial project proposal outlined objectives for developing the WebSDSS solution to ensure equitable distribution of public services and infrastructure. The system integrates spatial, socioeconomic, and demographic data, providing a comprehensive understanding of service supply and demand. It helps planners evaluate small-scale spatial situations, make informed infrastructure decisions, and reduce the workload for assessing new facilities. Assessing these objectives is crucial for determining the project's success.

To guide the study, these objectives were rewritten as research questions, all intended to be answered by the end of the study. Section 1.3 derives these questions from the major question, "How can a Data-Driven Decision Support System use accessibility and socioeconomic indicators to improve public facilities usage?", to specify the analysis. The following aspects are important in the pursuit of answering the main question:

- This work explores how a Decision Support System (DSS) is built, its architecture, its development, and its incorporation by other areas. It examines how these areas use their tools to enhance these systems' capacities, culminating in a WebSDSS capable of using GIS tools, web technologies, and spatial data to assist urban planners in making better decisions.
- During the analysis and design phases, the author applied the main concepts of Spatial Decision Support System (SDSS) to ensure the solution aligned with its definition and was suitable for the political urban-planning decision environment. The goal was to design a powerful and distributed web-based software architecture. Additionally, the author sought to elucidate the main sources of sociodemographic and spatial data that could be used to feed the system, as well as to identify the best format for presenting the data to the end user.
- This work conducts a thorough value analysis to identify the key functionalities that meet the target audience's demands and consequently fill the gaps that decision support tools have in the context of urban planning.
- Finally, the project implementation prioritizes the construction of a prototype capable of delivering the main functionalities and an interactive interface that uses geographic information for better data visualization.

In conclusion, although some aspects of the project implementation did not fully meet the requirements, the author believes this project directly addresses the primary research question. It delivers

the basic concepts and tools related to a DSS and demonstrates how they can be structurally and conceptually organized to provide a solution that urban planners can use to make better decisions.

8.2 Limitations

Even though the objectives outlined in the previous section were generally achieved, it is essential to critically evaluate the work completed throughout this dissertation.

The system has several limitations that prevent it from meeting some non-functional requirements, particularly those related to scalability and supportability. The proposed solution is limited in terms of scalability, as it was not designed for deployment in a high-demand production environment. The current architecture does not support running multiple containers on different physical machines, restricting the system's expansion capacity. Additionally, the solution was not tested in a real production environment, which could result in performance and reliability issues. The solution operates in a development environment, limiting the ability to assess its performance in a real production setting, largely due to financial and hardware constraints. Besides the limited deployment infrastructure, the automation process lacks extensive coverage of unit and integration tests for different scenarios, as mentioned in the previous chapter. Regarding non-functional requirements, the author acknowledges that some best practices in software development were not followed. For example, adopting practices like Test-Driven Development (TDD) could have resulted in broader scenario coverage.

Many limitations related to functional requirements, or at least impacting their full fulfillment, are described in Section 6.1. The most notable issue is the author's scope reductions to complete the project within the established timeframe. Perhaps the most significant reduction was implementing a database for spatial data, as using files impacts system performance and risks failing to meet some requirements. Besides the available sociodemographic data does not enable many analysis scenarios for some population groups.

Finally, the author believes the developed system meets the established functional requirements but recognizes that there is room for future improvements and expansions.

8.3 Future Work

Future works should initially focus on improvements addressing the limitations listed in the previous section, which have an immediate impact on meeting the requirements. These improvements may include:

- Implementing a spatial database to store geographic data, particularly using tools suitable for this purpose, such as PostGIS.
- Expanding unit and integration test scenarios to ensure the system functions correctly in various situations.
- Deploying a real production environment to evaluate the system's performance and reliability.
- Implementing a monitoring and auditing system to track the system's performance and identify potential issues.
- Enhancing continuous integration to automate the deployment process and ensure correct system deployment.
- Updating the database with more recent and accurate data to ensure the system provides up-to-date and precise information.

In a later phase, the system can be expanded to include new functionalities and improvements, such as:

- Implementing a microservices architecture to increase the system's scalability and flexibility, particularly by detaching the backend API from the frontend.
- Utilizing artificial intelligence to predict future demand for public services and infrastructure based on historical data and trends.
- Incorporating accessibility analysis with an ownership component, considering factors like ease of negotiation, availability, and existing infrastructure.
- Improving the accessibility model to consider aspects such as competition between services and more complex decay functions that account for population density and distance between services.

In addition to immediate actions on the system, the author believes that the theoretical foundation can be further connected to the concept of smart cities. Tools that use data to improve people's quality of life are fundamental to smart cities.

The developed project is understood to be a starting point for future research and improvements. The system can be expanded and enhanced to meet different needs and scenarios, becoming a valuable tool to support urban planning and decision-making.

8.4 Final Considerations

During the author's academic journey, studies on urban design and city development have always been of great interest. This dissertation provided an opportunity to explore these topics in depth, particularly by considering aspects often neglected in urban planning decisions, such as accessibility. This motivated the project, focusing on how accessibility can improve the distribution of public services and infrastructure in cities.

The author believes that merely applying accessibility methodologies would not constitute a significant study, as these analytical tools are already extensively used in other fields, as discussed in Chapter 2. Therefore, integrating these analyses into the context of computer engineering through the development of a system capable of incorporating them and providing useful information for urban planners was seen as a valuable approach.

Regarding the analysis models for both accessibility and decision-making, the author found the application and implementation of these models in languages like Python and JavaScript to be particularly interesting. This included studying the mathematical foundations of these models and their practical applications. In terms of the technologies used, the author had limited experience with frameworks like Next.js, which motivated learning new technologies and applying modern web development concepts. This was crucial in the project's development, serving as a significant source of motivation and encouragement.

Additionally, the author appreciated learning and applying new evaluation methodologies such as QEF and SUS, and value analysis methods like Quality Function Deployment (QFD). As well as considers this dissertation work a valuable advancement in its academic journey. More importantly, the knowledge gained will undoubtedly be applied in future endeavors.

Ultimately, the project meets the initially established requirements, as the final solution provides functionalities that assist public managers in the urban planning decision-making process. These

features include an interactive graphical interface with maps for contextualized spatial data visualization, the application of multi-criteria analysis models to aid decision-making, and the use of sociodemographic data as criteria in calculating the accessibility of points of interest.

References

Alex Reisner Joshua Clifford, Michael Kruger Peter Baczek (2015). *Geocoder*. Version 1.5.3. url: https://github.com/alexreisner/geocoder.

Amazon Web Services, Inc. (2024). *Amazon Web Services (AWS)*. Accessed: 2024-05-01. url: https://aws.amazon.com.

ArcGIS (2023). *ArcGIS Online Overview*. https://www.esri.com/en-us/arcgis/products/arcgis-online/overview. Accessed: 2023-12-26.

Armstrong, Marc P, Paul J Densham, and Gerard Rushton (1986). "Architecture for a microcomputer based spatial decision support system". eng. In: pp. 120–131.

Authors, Various (2024). "Trunk-Based Development". In: *Trunk-Based Development*. Accessed: 2024-07-25. url: https://trunkbaseddevelopment.com.

Averweg, U. R. F. (2012). *Decision-making support systems: Theory and practice*. Udo Richard Franz Averweg. isbn: 978-740-301762.

Bangor, Aaron, Philip Kortum, and James Miller (Aug. 2008). "The System Usability Scale (SUS): An Empirical Evaluation". In: *International Journal of Human-Computer Interaction* 24, pp. 574–594. doi: 10.1080/10447310802205776.

Barnett, Jonathan and BL Jones (1982). *An introduction to urban design*. Harper & Row New York. isbn: 9780064303767.

Beck, K. et al. (2001). "Manifesto for Agile Software Development". In: url: http://agilemanifesto.org.

Bhargava, Hemant K. and Daniel J. Power (2001). "Decision Support Systems and Web Technologies: A Status Report". In: url: https://api.semanticscholar.org/CorpusID:55270766.

Bouchereau, Vivianne and Hefin Rowlands (2000). "Methods and techniques to help quality function deployment (QFD)". In: *Benchmarking: An International Journal* 7, pp. 8–20. url: https://api.semanticscholar.org/CorpusID:1648427.

Brazilian Institute of Geography and Statistics (2024). *Brazilian Institute of Geography and Statistics*. https://www.ibge.gov.br/en/. [Online; accessed 1-May-2024].

Butler, Kevin J. (2020). *H3Pandas: H3 Indexing for Pandas*. Accessed: 2024-05-01. url: https://github.com/kevintw/h3pandas.

Chung, Lawrence (2012). *Non-functional requirements in software engineering*. Springer.

Conveyal (2023a). *Conveyal - Learn more about Conveyal Analysis*. https://conveyal.com/learn. Accessed: December 22, 2023.

– (2023b). *R5py: Rapid Realistic Routing with R5 in Python*. https://r5py.readthedocs.io/en/stable/. Accessed on December 26, 2023.

Cooke, Graham S et al. (2010). "Population uptake of antiretroviral treatment through primary care in rural South Africa". In: *BMC public health* 10.1, p. 585.

Cowan, Rob (2005). *Essential Urban Design*. RIBA Publishing. isbn: 9781000401042.

Culttt (2014). *Domain Model - Domain Driven Design*. url: https://culttt.com/2014/11/12/domain-model-domain-driven-design/ (visited on 05/01/2024).

Curtis, C. and J. Scheurer (2010). "Planning for sustainable accessibility: Developing tools to aid discussion and decision-making". In: *Progress in Planning* 74.2, pp. 53–106.

Curtis, Carey (2011). "Integrating Land Use with Public Transport: The Use of a Discursive Accessibility Tool to Inform Metropolitan Spatial Planning in Perth". In: *Transport Reviews* 31.2,

pp. 179–197. doi: 10.1080/01441647.2010.525330. eprint: https://doi.org/10.1080/01441647.2010.525330. url: https://doi.org/10.1080/01441647.2010.525330.

Cypress.io (2023). *Cypress: JavaScript End to End Testing Framework*. Accessed: 2024-08-04. url: https://www.cypress.io.

Delamater, Paul L, Ashton M Shortridge, and Ryan C Kilcoyne (2019). "Using floating catchment area (FCA) metrics to predict health care utilization patterns". In: *BMC health services research* 19.1, p. 144. doi: 10.1186/s12913-019-3969-5.

Dessì, Nicoletta, Gianfranco Garau, and Barbara Pes (2012). "Web 2.0 Technologies Empowering Spatial Decision Making". In: *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 462–467. doi: 10.1109/WETICE.2012.31.

Django Software Foundation (2010). *Django: A Web Framework for the Python Programming Language*. url: http://djangoproject.com.

Docker, Inc. (2024). *Docker: Enterprise Container Platform*. Accessed: 2024-05-01. url: https://www.docker.com.

Erdin, Ceren and Halil Emre Akbaş (2019). "A Comparative Analysis of Fuzzy TOPSIS and Geographic Information Systems (GIS) for the Location Selection of Shopping Malls: A Case Study from Turkey". In: *Sustainability* 11.14. issn: 2071-1050. doi: 10.3390/su11143837. url: https://www.mdpi.com/2071-1050/11/14/3837.

Escudeiro, Paula and José Bidarra (Jan. 2008). "Quantitative Evaluation Framework (QEF)". In: *Conselho Editorial/Consejo Editorial*, p. 16.

Facebook, Inc. (n.d.). *Jest: Delightful JavaScript Testing*. https://jestjs.io/. Accessed: 2024-08-01.

Fortaleza City Hall (2024). *Fortaleza City Hall*. https://www.fortaleza.ce.gov.br/. [Online; accessed 1-May-2024].

Fowler, Martin (2012). "The Practical Test Pyramid". In: url: https://martinfowler.com/articles/practical-test-pyramid.html.

GeeksforGeeks (2024). *Quality Function Deployment (QFD) in Software Quality*. https://www.geeksforgeeks.org/quality-function-deployment-qfd-in-software-quality/. [Online; accessed on 7-March-2024].

GeoPandas (2023). *Ecosystem*. https://geopandas.org/en/latest/community/ecosystem.html. Accessed: December 26, 2023.

Geurs, K.T. and B. van Wee (2004). "Accessibility evaluation of land-use and transport strategies: review and research directions". In: *Journal of Transport Geography* 12, pp. 127–140.

Geurs, Karst T and Jan R Ritsema van Eck (2001). "Accessibility Measures: Review and Applications". In: *Transportation Research Record* 1780.1, pp. 33–41.

Gil Solá, Ana and Bertil Vilhelmson (2019). "Negotiating Proximity in Sustainable Urban Planning: A Swedish Case". In: *Sustainability* 11.1. issn: 2071-1050. doi: 10.3390/su11010031. url: https://www.mdpi.com/2071-1050/11/1/31.

Gimond, Manuel (2023). *Chapter 1 Introduction to GIS | Intro to GIS and Spatial Analysis*. url: https://mgimond.github.io/Spatial/introGIS.html.

GitHub, Inc. (2024). *GitHub*. Accessed: 2024-07-25. url: https://github.com.

Google (2023). *Google Maps Platform | Google for Developers*. Accessed on insert date. url: https://developers.google.com/maps.

Griffith, Daniel A. (2015). "Geographic Information Systems". In: *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Ltd, pp. 1–8. isbn: 9781118445112. doi: https://doi.org/10.1002/9781118445112.stat03430.pub2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat03430.pub2. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat03430.pub2.

Grinberger, A. Yair et al. (2022). "OSM Science;The Academic Study of the OpenStreetMap Project, Data, Contributors, Community, and Applications". In: *ISPRS International Journal of*

Geo-Information 11.4. issn: 2220-9964. doi: 10.3390/ijgi11040230. url: https://www.mdpi.com/2220-9964/11/4/230.

Guthrie, Andrew, Yingling Fan, and Kirti Vardhan Das (2017). "Accessibility Scenario Analysis of a Hypothetical Future Transit Network: Social Equity Implications of a General Transit Feed Specification–Based Sketch Planning Tool". In: *Transportation Research Record* 2671.1, pp. 1–9. doi: 10.3141/2671-01. eprint: https://doi.org/10.3141/2671-01. url: https://doi.org/10.3141/2671-01.

UN-Habitat (2020). *World Cities Report 2020: The Value of Sustainable Urbanization*. eng. Nairobi, pp. 352–377. doi: 10.52817/9789211328721. url: https://unhabitat.org/sites/default/files/2020/11/world_cities_report_2020_abridged_version.pdf.

UN-Habitat, United Nations Human Settlements Programme (2015). "International Guidelines on Urban and Territorial Planning". In: Handbook available at https://unhabitat.org/international-guidelines-on-urban-and-territorial-planning-ig-utp-handbook. url: https://unhabitat.org/international-guidelines-on-urban-and-territorial-planning.

Henderson, JC and DA Schilling (1990). "Design and Implementation of Decision Support Systems in the Public Sector". In: *College of Administrative Science*. url: https://www.jstor.org/stable/249116.

Hess, Daniel Baldwin (2005). "Access to employment for adults in poverty in the Buffalo-Niagara region". In: *Urban Geography* 26.5, pp. 407–432.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95.

IBM (2023). "What Is Software Testing?" In: *IBM*. Accessed: 2024-08-04. url: https://www.ibm.com/topics/software-testing.

– (2024). *Use Case Diagrams*. IBM. url: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case (visited on 03/08/2024).

IEEE Computer Society (2024). *Software Requirements Specifications*. IEEE Computer Society. url: https://www.computer.org/resources/software-requirements-specifications (visited on 03/08/2024).

Instituto Brasileiro de Geografia e Estatística (IBGE) (2024). *IBGE Cidades - Fortaleza*. Accessed: Date. url: https://cidades.ibge.gov.br/brasil/ce/fortaleza/pesquisa/23/27652?detalhes=true.

International, Ecma (2023). *ECMAScript Language Specification*. https://tc39.es/ecma262/. Accessed: 2024-08-06.

Journey, A Lean (2022). *FAST Diagram - Function Analysis System*. url: http://www.aleanjourney.com/2022/05/fast-diagram-function-analysis-system.html (visited on 03/07/2024).

Karst, T and Jan R Ritsema van Eck (2003). "Evaluation of Accessibility Impacts of Land-Use Scenarios: The Implications of Job Competition, Land-Use, and Infrastructure Developments for the Netherlands". In: *Environment and Planning B: Planning and Design* 30.1, pp. 69–87. doi: 10.1068/b12940. eprint: https://doi.org/10.1068/b12940. url: https://doi.org/10.1068/b12940.

Keen, P. (1987). "Decision support systems: The next decade". In: *Decision Support Systems* 3.3, pp. 253–265. doi: 10.1016/0167-9236(87)90180-1.

Keen, Peter GW (1980). *Decision support systems: A research perspective*. Report 54. Cambridge, Mass.: Center for Information Systems Research, Alfred P. Sloan School of Management. url: http://hdl.handle.net/1721.1/47172.

Keenan, P. B. (2003). "Spatial Decision Support Systems". In: Location: Publisher. isbn: 9780367864392.

Keenan, P. B. and P. Jankowski (2018). "Spatial Decision Support Systems: Three decades on". In: *International Journal of Geographical Information Science* 32.5, pp. 983–1010. doi: 10.1080/13658816.2018.1447549.

Kitchenham, Barbara and Stuart Charters (2007). *Guidelines for performing systematic literature reviews in software engineering*. Joint Report EBSE 2007-001. Keele University and Durham University.

Kruchten, Philippe (1995). "4+1 View Model of Software Architecture". In: *IEEE International Conference on Software Engineering*, pp. 25–25. url: https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

Krügel, Falko et al. (2024). "An Online Multicriteria—Spatial Decision Support System for Public Services Planning". In: *Applied Sciences* 14.4. issn: 2076-3417. doi: 10.3390/app14041526. url: https://www.mdpi.com/2076-3417/14/4/1526.

Liu, Yizhou et al. (2020). "Rapidly measuring spatial accessibility of COVID-19 healthcare resources: a case study of Illinois, USA". In: *International Journal of Health Geographics* 19.1, p. 36.

Machado, M. A. T. et al. (2018). "Future accessibility impacts of transport policy scenarios: Equity and sensitivity to travel time thresholds for Bus Rapid Transit expansion in Rio de Janeiro". In: *Transportation Research Record: Journal of the Transportation Research Society* 27 (3), pp. 321–332. doi: 10.1007/s1360700321-332. url: https://ideas.repec.org/a/eee/jotrge/v74y2019icp321-332.html.

Madanchian, Mitra and Hamed Taherdoost (Aug. 2023). "A comprehensive guide to the TOPSIS method for multi-criteria decision making". In: *Sustainable Social Development* 1. doi: 10.54517/ssd.v1i1.2220.

Malczewski, Jacek (2004). "GIS-based land-use suitability analysis: a critical overview". In: *Progress in Planning* 62.1, pp. 3–65. issn: 0305-9006. doi: https://doi.org/10.1016/j.progress.2003.09.002. url: https://www.sciencedirect.com/science/article/pii/S0305900603000801.

Mapbox (2022). *Mapbox*. Mapbox. url: https://www.mapbox.com/.

Mayaud, Jerome R., Martino Tran, and Rohan Nuttall (2019). "An urban data framework for assessing equity in cities: Comparing accessibility to healthcare facilities in Cascadia". In: *Computers, Environment and Urban Systems* 78, p. 101401. issn: 0198-9715. doi: https://doi.org/10.1016/j.compenvurbsys.2019.101401. url: https://www.sciencedirect.com/science/article/pii/S0198971519303813.

National Center for Geographic Information and Analysis (2023). https://en.wikipedia.org/wiki/National_Center_for_Geographic_Information_and_Analysis. Accessed: 2023-12-26.

Olímpio, João et al. (Sept. 2020). "DESIGUALDADE SOCIOAMBIENTAL E A CAPACIDADE DE LIDAR COM A PANDEMIA DE COVID-19: AVALIAÇÃO DA GEOESPACIALIDADE DA VULNERABILIDADE EM FORTALEZA-CE". In: *Revista da Casa da Geografia de Sobral (RCGS)* 22, pp. 70–89. doi: 10.35701/rcgs.v22n2.695.

OpenLayers Contributors (2024). *OpenLayers: Free Maps for the Web*. https://openlayers.org/. Accessed: 2024-08-06.

OpenStreetMap contributors (2023). *OpenStreetMap: A free, editable map of the world*. Accessed on insert date. url: https://welcome.openstreetmap.org/what-is-openstreetmap/.

Pearce, David (2024). *Identifying User Requirements with Use Cases*. San Jose State University. url: https://www.cs.sjsu.edu/~pearce/modules/lectures/ooa/requirements/IdentifyingURPS.htm (visited on 03/08/2024).

Pereira, Rafael H. M. et al. (Mar. 2021). "r5r: Rapid Realistic Routing on Multimodal Transport Networks with R^5 in R". en. In: *Transport Findings*. Publisher: Network Design Lab. doi: 10.32866/001c.21262. url: https://findingspress.org/article/21262-r5r-rapid-realistic-routing-on-multimodal-transport-networks-with-r-5-in-r (visited on 03/04/2021).

PostGIS Development Group (2022). *PostGIS*. PostgreSQL Global Development Group. url: https://postgis.net/docs/reference.html.

Prettier (n.d.). *Prettier: An opinionated code formatter*. https://prettier.io/. Accessed: 2024-08-01.

Project Management Institute (2017). *Agile Practice Guide*. 1st ed. Newtown Square, PA: Project Management Institute, p. 167. isbn: 978-1-62825-199-9. url: https://www.pmi.org/pmbok-guide-standards/practice-guides/agile.

PyOTP - The Python One-Time Password Library (2023). https://pyauth.github.io/pyotp/. Accessed in December 2023.

Python Core Team (2023). *Python: A dynamic, open source programming language*. Accessed on insert date. Python Software Foundation. url: https://www.python.org/.

QGIS (2023). *QGIS - The Leading Open Source Desktop GIS*. https://qgis.org/en/site/about/index.html. Accessed: 2023-12-26.

React Contributors (2024). *React - A JavaScript library for building user interfaces*. https://reactjs.org/. Accessed: 2024-08-06.

ReadingCraze (Feb. 2015). *Hypothesis Formulation in Research*. Accessed: 2022-02-14. url: http://readingcraze.com/index.php/hypothesis-formulation-research/.

ReVelle, Jack B and John W Moran (1998). *QFD Handbook*. John Wiley & Sons.

rgeo Development Team (2023). *rgeo: Geospatial data library for Ruby*. https://github.com/rgeo/rgeo.

Rinner, Claus (2003). "Web-based Spatial Decision Support: Status and Research Directions". In: url: https://api.semanticscholar.org/CorpusID:16229899.

Rinner, Claus, Carsten Keßler, and Stephen Andrulis (2012). "Web 2.0 Technologies Empowering Spatial Decision Making". In: *IEEE Xplore*. doi: 10.1109/MCSE.2012.63.

Rolnik, Raquel (2023). *"Porque é tão caro comprar a casa própria"*. "Accessed on 22/12/2023". url: https://youtu.be/gVhjorTQWRw.

Ruby Core Team (2023). *Ruby Programming Language*. https://www.ruby-lang.org/pt/.

Ruby on Rails Team (2012). *Ruby on Rails: A Web Framework for the Ruby Programming Language*. url: http://rubyonrails.org.

Saaty, Thomas L (2008). *Decision Making with the Analytic Hierarchy Process*. Springer.

Sánchez-Lozano, Juan M. et al. (2013). "Geographical Information Systems (GIS) and Multi-Criteria Decision Making (MCDM) methods for the evaluation of solar farms locations: Case study in south-eastern Spain". In: *Renewable and Sustainable Energy Reviews* 24, pp. 544–556. issn: 1364-0321. doi: https://doi.org/10.1016/j.rser.2013.03.019. url: https://www.sciencedirect.com/science/article/pii/S1364032113001780.

Selenium (2023). *Selenium: Web Browser Automation*. Accessed: 2024-08-04. url: https://www.selenium.dev.

Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu (2017). "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5, pp. 3909–3943. doi: 10.1109/ACCESS.2017.2685629.

Sikder, Iftikhar Uddin and Nagma Yasmin (1997). "Spatial Decision Support System for Location Planning". In: *International Journal of Aerospace*.

SnowGlobe Project (2023). https://snowglobe.soc.northwestern.edu/. Accessed: January 6, 2024.

Software, Perforce (n.d.). *What Is Linting?* https://www.perforce.com/blog/qac/what-is-linting. Accessed: 2024-08-01.

Solutions, NPD (2024). *Value Analysis*. https://www.npd-solutions.com/va.html.

Solutions, Visure (2024). *Functional Requirements: Definição, Exemplos e Template [Guia Completo]*. Visure Solutions. url: https://visuresolutions.com/pt/blog/functional-requirements/ (visited on 03/08/2024).

Sousa, Francelino (Dec. 2019). "DIAGNÓSTICO ESTRATÉGICO DAS DESIGUALDADES SOCIOESPACIAIS NA ACESSIBILIDADE AO TRABALHO EM FORTALEZA FORTALEZA". PhD thesis. doi: 10.13140/RG.2.2.35180.23680.

Souza, ML de (2003). "Mudar a Cidade: uma Introdução Crítica ao Planejamento e à Gestão Urbanos. 2ª". In: *Ed., Rio de Janeiro: Bertrand Brasil*.

Sprague, Ralph H. and Eric D. Carlson (1982). "Building effective decision support systems". In: *Computers and Standards* 1.2. Hardback; index, illustrations, problems, p. 329. issn: 0167-8051. doi: https://doi.org/10.1016/0167-8051(82)90033-X. url: https://www.sciencedirect.com/science/article/pii/016780518290033X.

Tang, Wenwu et al. (2022). "A Web-based Spatial Decision Support System of Wastewater Surveillance for COVID-19 Monitoring: A Case Study of a University Campus". In: *medRxiv*. doi: 10.1101/2021.12.29.21268516. eprint: https://www.medrxiv.org/content/early/2022/01/01/2021.12.29.21268516.full.pdf. url: https://www.medrxiv.org/content/early/2022/01/01/2021.12.29.21268516.

Torvalds, Linus et al. (2005). *Git*. Version 2.40.0. Accessed: 2024-07-25. url: https://git-scm.com/.

Tosun, Ayse et al. (2018). "On the Effectiveness of Unit Tests in Test-Driven Development". In: *Proceedings of the 2018 International Conference on Software and System Process*. ICSSP '18. Gothenburg, Sweden: Association for Computing Machinery, pp. 113–122. isbn: 9781450364591. doi: 10.1145/3202710.3203153. url: https://doi.org/10.1145/3202710.3203153.

TypeScript Contributors (2024). *TypeScript: Typed JavaScript at Any Scale*. https://www.typescriptlang.org/. Accessed: 2024-08-06.

Vercel (2024). *Vercel: Develop. Preview. Ship. For the best frontend teams.* https://vercel.com/. Accessed: 2024-07-30.

Vercel, Inc. (2024). *Next.js: The React Framework for Production*. https://nextjs.org/. Accessed: 2024-08-06.

Vladimir Agafonkin and Leaflet Contributors (2022). *Leaflet*. Leaflet. url: https://leafletjs.com/.

Wikipedia (2023). *Canada Geographic Information System*. https://en.wikipedia.org/wiki/Canada_Geographic_Information_System.

YAML (2021). *YAML Ain't Markup Language (YAML) Revision 1.2.2*. Accessed: 2024-07-31. url: https://yaml.org/.

Young, David (Aug. 2013). "Software Development Methodologies". In: *White paper*.

Zakas, Nicholas C. (2013). *ESLint: The pluggable linting utility for JavaScript and JSX*. https://eslint.org/. Accessed: 2024-08-01.

Zmitrowicz, Witold (2002). "Planejamento territorial urbano". In: *Escola Politécnica da*.

Zmitrowicz, Witold and Gisis Neto (1997). "Infra-estrutura urbana". In: *São Paulo: EPUSP*.

Appendix A

Sequence Diagrams

Provides sequence diagrams for key use cases in the application, illustrating the interaction between components during the execution of processes.

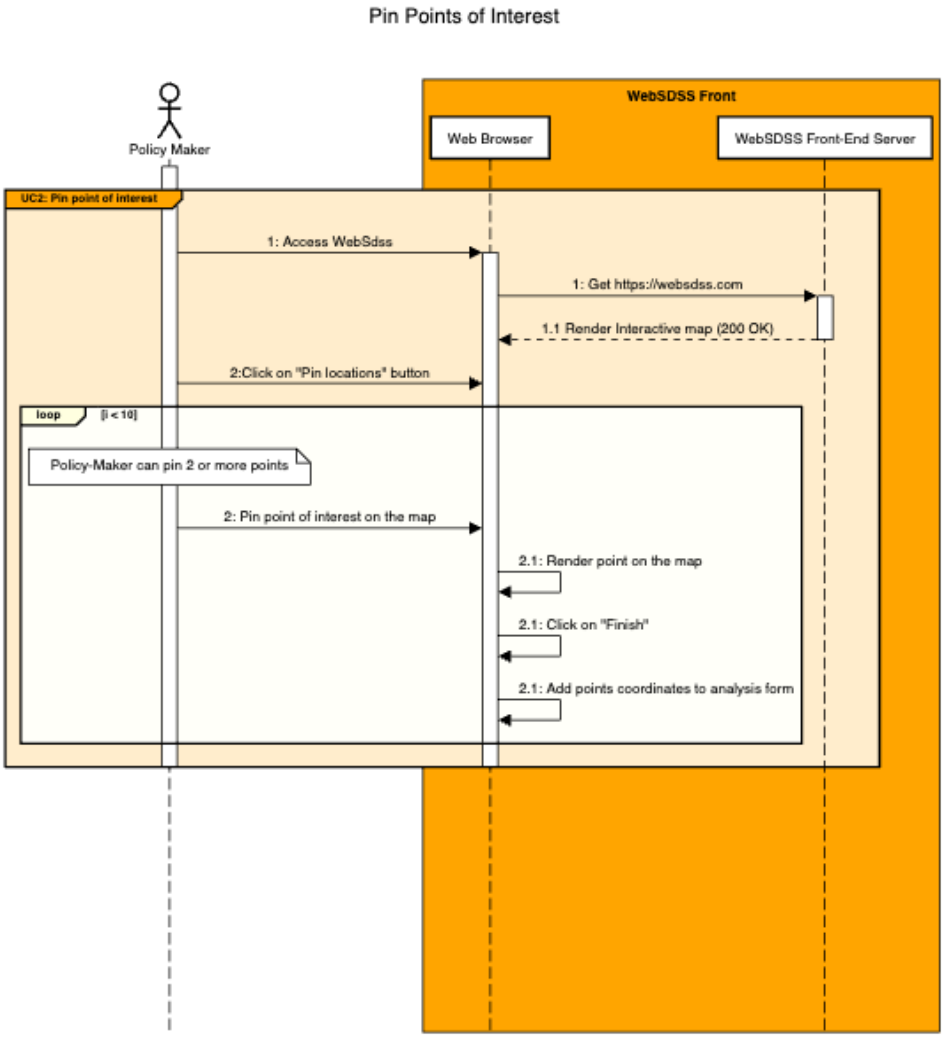

Figure A.1: Sequence diagram 'Pin Points of Interest'

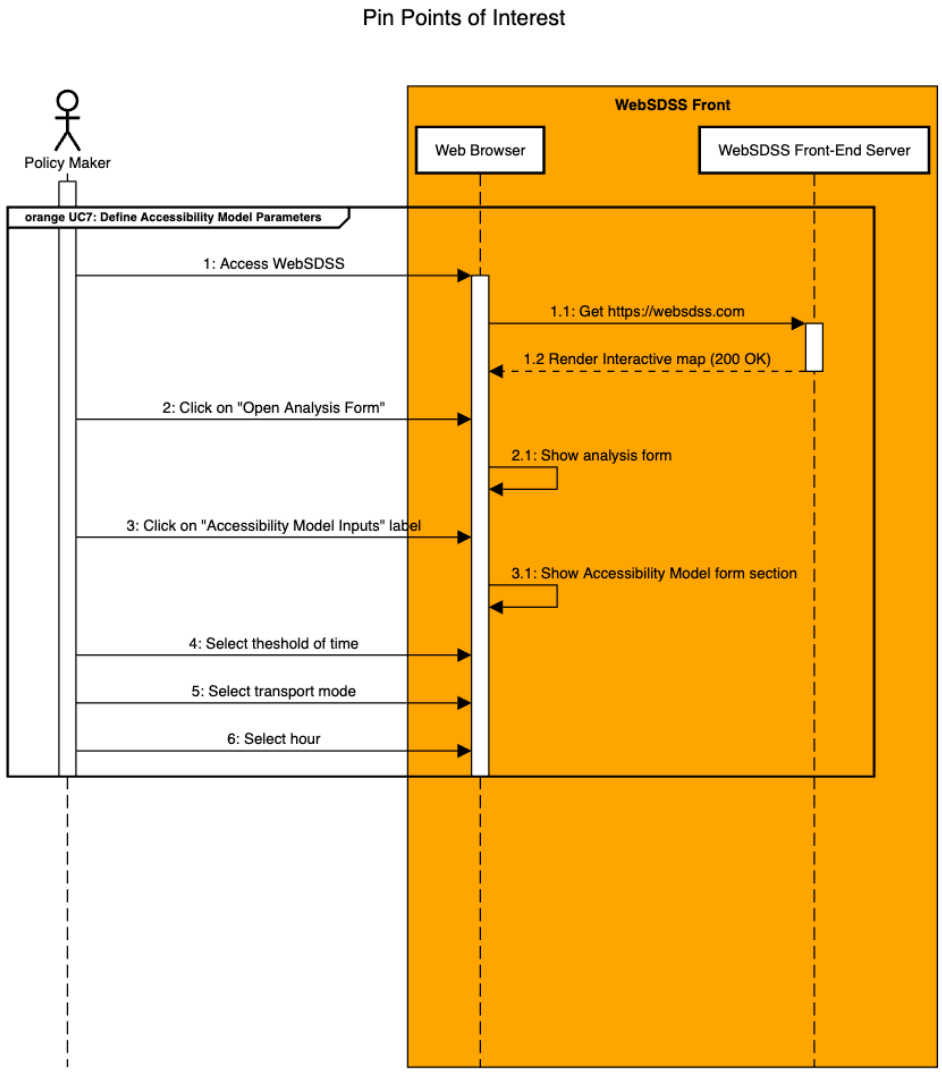

Figure A.2: Sequence diagram 'Input Accessibility Model Parameters'

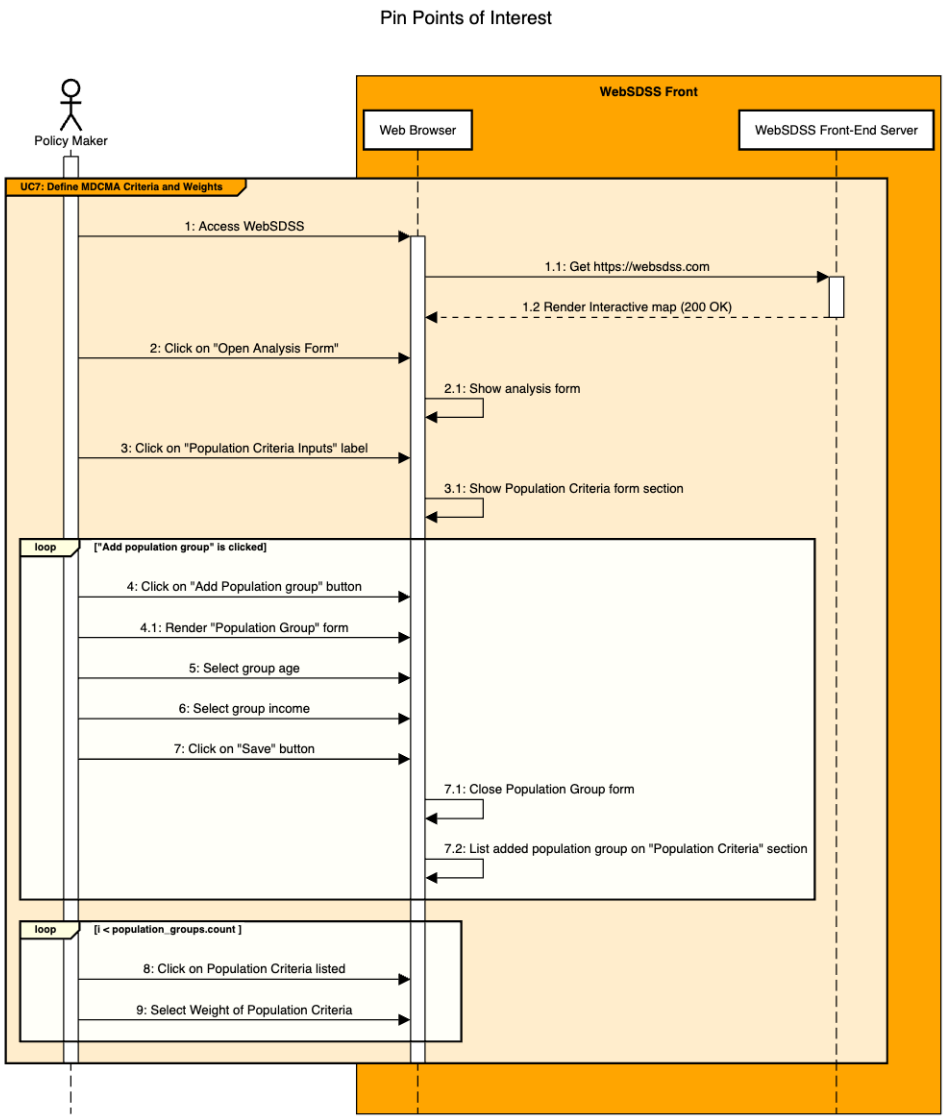

Figure A.3: Sequence diagram 'Input MCDMA Parameters'

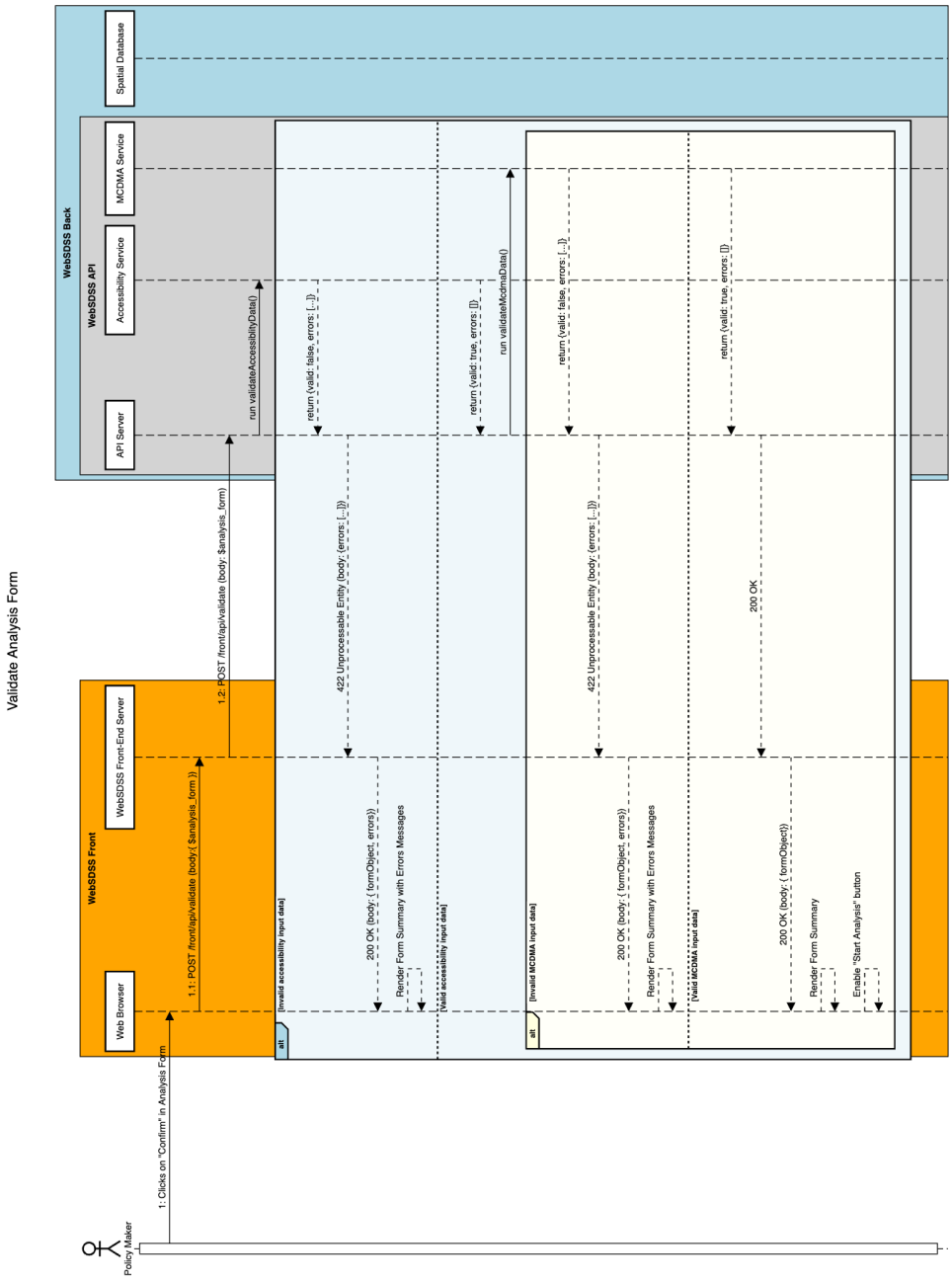

Figure A.4: Sequence Diagram 'Submit Analysis Form'

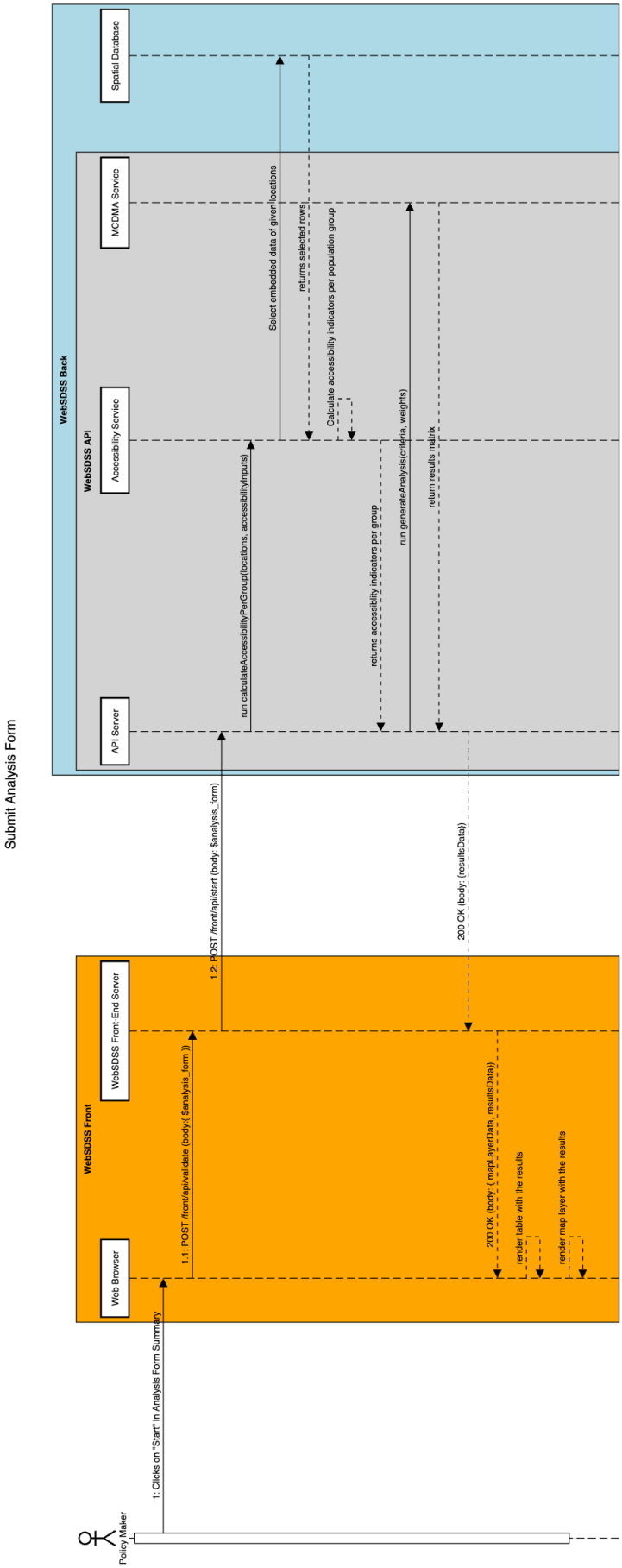

Figure A.5: Sequence Diagram 'Submit Analysis Form'

Appendix B

AccessibilityForm Source Code

Provides the source code for the AccessibilityForm component, responsible for handling input forms related to accessibility analysis.

```
import { SetStateAction, useState } from 'react';
import {
  FormControl,
  FormLabel,
  Slider,
  Box,
  Divider,
  Tooltip,
  RadioGroup,
  Radio,
  FormControlLabel,
} from '@mui/material';
import DirectionsCarIcon from '@mui/icons-material/DirectionsCar';
import DirectionsBusIcon from '@mui/icons-material/DirectionsBus';
import DirectionsWalkIcon from '@mui/icons-material/DirectionsWalk';
import Info from '@mui/icons-material/Info';

import { AnalysisForm } from '../types';
import { AccessibilityForm as AccessibilityFormType } from './types';

import style from './style.module.css';

const marks = [
  {
    value: 0,
    label: '0',
  },
  {
    value: 20,
    label: '20',
  },
  {
    value: 30,
    label: '30',
  },
  {
    value: 100,
    label: '100',
  },
];
const transportModes = [
  {
    value: 'walking',
    label: 'WALKING',
    icon: <DirectionsWalkIcon fontSize='large' color='primary' />,
```

```
    },
    {
      value: 'public',
      label: 'PUBLIC',
      icon: <DirectionsBusIcon fontSize='large' color='primary' />,
    },
    {
      value: 'private',
      label: 'PRIVATE',
      icon: <DirectionsCarIcon fontSize='large' color='primary' />,
    },
  ];

  const years = [
    {
      value: '2020',
      label: '2020',
    },
    {
      value: '2023',
      label: '2023',
    },
  ];

  const models = [
    {
      value: 'passive',
      label: 'Passive',
    },
    {
      value: 'active',
      label: 'Active',
    },
  ];

  const AccessibilityForm = ({ formData }: { formData: AnalysisForm }) => {
    const [accessibilityForm, setAccessibilityForm] =
      useState<AccessibilityFormType>(formData.accessibilityForm);

    const handleChange = (e: any) => {
      setAccessibilityForm((previous: AccessibilityFormType) => {
        const newAccessibilityData = {
          ...previous,
          [e.target.name]: e.target.value,
        };

        formData.accessibilityForm = newAccessibilityData;

        return newAccessibilityData;
      });
    };

    function valuetext(value: number) {
      return `${value} min`;
    }
    return (
      <Box className={style.accessibilityFormContainer}>
        <Box
          className={style.formControl}
          sx={{ display: 'flex', mt: '10px', justifyContent: 'space-between' }}
        >
          <FormControl>
```

```
108             <FormLabel
109               id='radio-model-label'
110               sx={{ display: 'flex', alignItems: 'center' }}
111             >
112               <Tooltip title='Choose accessibility analysis type (see helper card for more
    details).'>
113                 <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
114               </Tooltip>
115               Accessibility Model
116             </FormLabel>
117             <RadioGroup
118               row
119               aria-labelledby='radio-model-label'
120               name='model'
121               onChange={handleChange}
122               sx={{
123                 color: 'black',
124                 display: 'flex',
125                 flexDirection: 'row',
126                 m: 1,
127               }}
128               value={accessibilityForm.model}
129             >
130               {models.map((model) => (
131                 <FormControlLabel
132                   key={model.value}
133                   value={model.value}
134                   control={<Radio />}
135                   label={model.label}
136                   disabled={model.value === 'active'}
137                 />
138               ))}
139             </RadioGroup>
140           </FormControl>
141
142           <FormControl>
143             <FormLabel
144               id='radio-year-label'
145               sx={{ display: 'flex', alignItems: 'center' }}
146             >
147               <Tooltip title='Base year of the data used for the analysis.'>
148                 <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
149               </Tooltip>
150               Year
151             </FormLabel>
152             <RadioGroup
153               row
154               aria-labelledby='radio-year-label'
155               name='year'
156               onChange={handleChange}
157               sx={{
158                 color: 'black',
159                 display: 'flex',
160                 flexDirection: 'row',
161                 m: 1,
162               }}
163               value={accessibilityForm.year}
164             >
165               {years.map((year) => (
166                 <FormControlLabel
167                   key={year.value}
168                   value={year.value}
```

```
                control={<Radio disabled={year.value === '2020'} />}
                label={year.label}
              />
            ))}
          </RadioGroup>
        </FormControl>
      </Box>
      <Divider sx={{ mb: '20px' }} />
      <FormControl className={style.formControl}>
        <FormLabel
          id='radio-transport-mode-label'
          sx={{ display: 'flex', alignItems: 'center' }}
        >
          <Tooltip title='Type of transportation used for displacement in the analysis'>
            <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
          </Tooltip>
          Transport Mode
        </FormLabel>
        <RadioGroup
          row
          aria-labelledby='radio-transport-mode-label'
          name='transportMode'
          onChange={handleChange}
          sx={{
            color: 'black',
            display: 'flex',
            flexDirection: 'row',
            justifyContent: 'space-evenly',
            m: 1,
          }}
          value={accessibilityForm.transportMode}
        >
          {transportModes.map((mode) => (
            <FormControlLabel
              key={mode.value}
              value={mode.value}
              control={<Radio disabled={mode.value != 'public'} />}
              label={<Tooltip title={mode.label}>{mode.icon}</Tooltip>}
            />
          ))}
        </RadioGroup>
      </FormControl>
      <Divider sx={{ mb: '20px' }} />
      <FormControl className={style.formControl}>
        <FormLabel sx={{ display: 'flex', alignItems: 'center' }}>
          <Tooltip title='Distance threshold of the displacement based on travel time
    (defines catchment area).'>
            <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
          </Tooltip>
          Travel Time
        </FormLabel>
        <Slider
          aria-label='Always visible'
          getAriaValueText={valuetext}
          step={10}
          marks={marks}
          valueLabelDisplay='on'
          value={accessibilityForm.travelTime}
          name='travelTime'
          onChange={handleChange}
        />
      </FormControl>
```

```
230         </Box>
231       );
232     };
233     export default AccessibilityForm;
```

Listing B.1: AccessibilitForm component source code

Appendix C

Accessibility Module Source Code

Contains the source code for the Accessibility module handling accessibility calculations and processing data related to demographic and geographic parameters.

```
import { GroupCriteria } from '@/components/Forms/AnalysisForm/MultiCriteriaForm/types';
import fs from 'fs';
import { parser } from 'stream-json';
import { streamArray } from 'stream-json/streamers/StreamArray';
import { AccessibilityOptions, AccessibilityAnalysisResult } from './types';

function readData(
  hexLocations: string[],
  groupCriteria: GroupCriteria[],
  accessibilityOptions: AccessibilityOptions
): Promise<AccessibilityAnalysisResult> {
  return new Promise((resolve, reject) => {
    const results: any[] = [];
    const stream = fs.createReadStream(
      process.cwd() + '/src/utils/modules/accessibility/geojson.json'
    );

    stream
      .pipe(parser())
      .pipe(streamArray())
      .on('data', (d: any) => {
        const result = processData(
          d.value,
          accessibilityOptions,
          hexLocations,
          groupCriteria
        );
        if (result) results.push(result);
      })
      .on('end', () => {
        resolve(groupResults(results));
      })
      .on('error', (err: any) => {
        console.error('Error reading stream:', err);
        reject(err);
      });
  });
}

function processData(
  data: any,
  accessibilityOptions: AccessibilityOptions,
  hexLocations: string[],
  groupCriteria: GroupCriteria[]
) {
```

```
    const criteriaMatrix: any = {};
    if (
      hexLocations.includes(data.properties.h3_polyfill_destino) &&
      data.properties.travel_time <= accessibilityOptions.travelTime
    ) {
      groupCriteria.forEach((criteria) => {
        if (
          !(
            criteriaMatrix[data.properties.h3_polyfill_destino] &&
            criteriaMatrix[data.properties.h3_polyfill_destino][criteria.name]
          )
        ) {
          criteriaMatrix[data.properties.h3_polyfill_destino] = {
            ...criteriaMatrix[data.properties.h3_polyfill_destino],
            [criteria.name]: { total: 0, hex: '' },
          };
        }
        if (
          data.properties.renda_per_capita >= criteria.incomeRange[0] &&
          data.properties.renda_per_capita < criteria.incomeRange[1]
        ) {
          criteria.ageLevel.forEach((ageLevel: string) => {
            criteriaMatrix[data.properties.h3_polyfill_destino][
              criteria.name
            ].total += data.properties[ageLevel];
          });
          if (
            criteriaMatrix[data.properties.h3_polyfill_destino][criteria.name]
              .total > 10
          )
            criteriaMatrix[data.properties.h3_polyfill_destino][
              criteria.name
            ].hex = data.properties.h3_polyfill_origem;
        }
      });
      return criteriaMatrix;
    }
  }

  export const groupResults = (results: any[]): AccessibilityAnalysisResult => {
    const groupedResults: AccessibilityAnalysisResult = {};
    results.forEach((result) => {
      Object.keys(result).forEach((key) => {
        if (!groupedResults[key]) {
          groupedResults[key] = {};
        }
        Object.keys(result[key]).forEach((group) => {
          if (!groupedResults[key][group]) {
            groupedResults[key][group] = { total: 0, hex: [] };
          }
          groupedResults[key][group].total += result[key][group].total;
          groupedResults[key][group].hex.push(result[key][group].hex);
        });
      });
    });
    return groupedResults;
  };

  export { readData, processData };
```

Listing C.1: Accessibility module source code

Appendix D

AnalysisForm Source Code

Displays the source code for the AnalysisForm component, which deals with input forms in the spatial decision-making process.

```
'use client';
import { useState, useRef, Fragment, MutableRefObject, useEffect } from 'react';
import {
  Fab,
  Slide,
  Box,
  Typography,
  Modal,
  LinearProgress,
  Container,
  Alert,
  Button,
} from '@mui/material';
import LocationForm from './LocationsForm';
import AccessibilityForm from './AccessibilityForm';
import MultiCriteriaForm from './MultiCriteriaForm';
import Stepper from './Stepper';
import { ArrowUpward, ArrowDownward } from '@mui/icons-material';

import { AnalysisForm as AnalysisFormType } from './types';
import * as LocationFormTypes from './LocationsForm/types';

import style from './style.module.css';
import HelperCard from '@/components/HelperCard';
import ReviewForm from '../ReviewForm';
import ResultForm from './ResultForm';
import { AnalysisResult } from '@/app/api/accessibility/types';
import { ResultsFilter } from './ResultForm/types';

const AnalysisForm = ({
  locationFormData,
  setLocationFormData,
  analysisFormData,
  setCityGeoJson,
  handleSubmit,
  submitting,
  results,
  resultsFilter,
  setResultsFilter,
  resetAnalysis,
}: {
  locationFormData: LocationFormTypes.LocationForm;
  setLocationFormData: React.Dispatch<
    React.SetStateAction<LocationFormTypes.LocationForm>
  >;
```

```
    analysisFormData: MutableRefObject<AnalysisFormType>;
    setCityGeoJson: (value: any) => void;
    handleSubmit: (params: any) => void;
    submitting: boolean;
    results: AnalysisResult | null;
    resultsFilter: ResultsFilter | null;
    setResultsFilter: (value: any) => void;
    resetAnalysis: () => void;
  }) => {
    const [checked, setChecked] = useState(false);
    const containerRef = useRef<HTMLElement>(null);
    const [openHelperModal, setOpenHelperModal] = useState(false);
    const [errors, setErrors] = useState<string[]>();
    const OpenFormButtonContent = checked ? (
      <Fragment>
        <ArrowDownward sx={{ mr: 2 }} />
        Close Analysis Form
      </Fragment>
    ) : (
      <Fragment>
        <ArrowUpward sx={{ mr: 2 }} />
        Open Analysis Form
      </Fragment>
    );

    const steps = [
      {
        label: 'Select Points of Interest',
        component: (
          <LocationForm
            locationFormData={locationFormData}
            setLocationFormData={setLocationFormData}
            formData={analysisFormData.current}
            setCityGeoJson={setCityGeoJson}
          />
        ),
        onValidate: (params: any) => {
          if (analysisFormData.current.locationForm.points.length < 1) {
            setErrors(['Please select at least one location']);
            setTimeout(() => {
              setErrors([]);
            }, 2000);
            return false;
          } else {
            return true;
          }
        },
      },
      {
        label: 'Set accessibility model parameters',
        component: <AccessibilityForm formData={analysisFormData.current} />,
        onValidate: (params: any) => {
          if (!analysisFormData.current.accessibilityForm.model) {
            setErrors(['Please select an accessibility model']);
            setTimeout(() => {
              setErrors([]);
            }, 4000);
            return false;
          } else {
            return true;
          }
        },
```

```
      },
      {
        label: 'Add Population Group criteria and weights',
        component: <MultiCriteriaForm formData={analysisFormData.current} />,
        onValidate: (params: any) => {
          if (analysisFormData.current.multiCriteriaForm.groups.length < 1) {
            setErrors(['Please add at least one group']);
            setTimeout(() => {
              setErrors([]);
            }, 4000);
            return false;
          } else {
            return true;
          }
        },
      },
      {
        label: 'Review and Submit',
        component: <ReviewForm formData={analysisFormData.current} />,
        onValidate: (params: any) => {
          setTimeout(() => {
            setErrors([]);
          }, 4000);
          return true;
        },
      },
    ];

    const openForm = () => {
      setChecked((prev) => !prev);
    };

    return (
      <Fragment>
        <Box className={style.analysisFormContainer}>
          <Slide in={checked} container={containerRef.current}>
            <Box component='form' className={style.analysisForm}>
              {errors && errors.length > 0 && (
                <Alert severity='error'>{errors.join(', ')}</Alert>
              )}
              {submitting ? (
                <Container>
                  <Box
                    sx={{
                      display: 'flex',
                      justifyContent: 'space-around',
                    }}
                  >
                    <Typography variant='body2'>Calculating...</Typography>
                  </Box>
                  <LinearProgress />
                </Container>
              ) : results ? (
                <ResultForm
                  results={results}
                  analysisFormData={analysisFormData.current}
                  resultsFilter={resultsFilter}
                  setResultsFilter={setResultsFilter}
                  resetAnalysis={resetAnalysis}
                />
              ) : (
                <Stepper
```

```
                steps={steps}
                onHelperClick={() => {
                  setOpenHelperModal(true);
                }}
                onSubmit={handleSubmit}
                initialStep={0}
              ></Stepper>
            )}
          </Box>
        </Slide>
      </Box>

      <Fab
        onClick={openForm}
        className={style.analysisOpenButton}
        variant='extended'
        color='primary'
      >
        {OpenFormButtonContent}
      </Fab>

      <Modal
        open={openHelperModal}
        onClose={() => {
          setOpenHelperModal(false);
        }}
        aria-labelledby='modal-modal-title'
        aria-describedby='modal-modal-description'
      >
        <HelperCard />
      </Modal>
    </Fragment>
  );
};

export default AnalysisForm;
```

Listing D.1: AccessibilitForm component source code

Appendix E

API Route Source Code

Provides the source code for a POST API endpoint that processes accessibility analysis requests. The endpoint accepts JSON data containing points of interest, group criteria (such as income and age range), and accessibility options (such as travel time and transport mode).

```
  import { NextRequest, NextResponse } from 'next/server';
import { readData } from '@/utils/modules/accessibility';
import * as MCDMA from '@/utils/modules/mcdma';
import { AccessibilityPayload, AnalysisResult, RankedResult } from './types';
export async function POST(request: NextRequest, response: NextResponse) {
  try {
    const { pointsOfInterest, groupCriteria, accessibilityOptions } =
      (await request.json()) as AccessibilityPayload;

    if (!groupCriteria || !accessibilityOptions || !pointsOfInterest) {
      return new Response(
        JSON.stringify({
          error: 'Invalid request. Use the template.',
          templatePayload: {
            groupCriteria: [
              {
                name: 'Group 1',
                avgIncomePerCapita: [0, 800],
                ageRange: [0, 100],
              },
            ],
            accessibilityOptions: {
              travelTime: 10,
              transportMode: 'car | bus | train | walk | bike',
              accessibilityType: 'active | passive',
            },
          },
        }),
        {
          status: 400,
          headers: {
            'Content-Type': 'application/json',
          },
        }
      );
    }

    const accessibilityAnalysisResult = await readData(
      pointsOfInterest,
      groupCriteria,
      accessibilityOptions
    );
    console.log(
      'params',
```

```
      pointsOfInterest,
      groupCriteria,
      accessibilityOptions
    );
    console.log(
      'results',
      Object.values(accessibilityAnalysisResult).map((group: any) =>
        Object.values(group).map((value: any) => value.hex)
      )
    );

    const decisionMatrix: number[][] = Object.values(
      accessibilityAnalysisResult
    ).map((group: any) =>
      Object.values(group).map((value: any) => value.total)
    );

    const weights = groupCriteria.map((criteria) => criteria.weight);
    const criteria = groupCriteria.map((criteria) => criteria.criteriaType);
    const multiCriteriaAnalysisResult = MCDMA.topsis.apply(
      decisionMatrix,
      weights,
      criteria
    );

    const hexLocationsRanked = Object.keys(accessibilityAnalysisResult);
    const rankedResults: RankedResult[] = multiCriteriaAnalysisResult.map(
      (result) => {
        return [
          hexLocationsRanked[result],
          accessibilityAnalysisResult[hexLocationsRanked[result]],
        ];
      }
    );

    const analysiResult = {
      message: 'Data received',
      rankedResults,
    } as AnalysisResult;

    // Example response to the client
    return new Response(JSON.stringify(analysiResult), {
      status: 200,
      headers: {
        'Content-Type': 'application/json',
      },
    });
  } catch (error) {
    return new Response(JSON.stringify({ error: 'Something went wrong' }), {
      status: 500,
      headers: {
        'Content-Type': 'application/json',
      },
    });
  }
}
```

Listing E.1: API Route source code

Appendix F

Application Configuration files

Lists the application configuration files, essential for setting up the environment for running the application.

```
{
  "extends": ["next/core-web-vitals", "prettier"]
}
```

Listing F.1: ESLint configuration file

```
{
  "trailingComma": "es5",
  "semi": true,
  "tabWidth": 2,
  "singleQuote": true,
  "jsxSingleQuote": true,
  "plugins": []
}
```

Listing F.2: Prettier configuration file

```
// jest.config.js
const nextJest = require('next/jest');
const createJestConfig = nextJest({
  // Path to Next.js app to load next.config.js
  dir: './',
});

/** @type {import('@jest/types').Config.InitialOptions} */
const customJestConfig = {
  preset: 'ts-jest',
  testEnvironment: 'jsdom',
  moduleNameMapper: {
    '^@/(.*)$': '<rootDir>/src/$1',
  },
  /**
   * Custom config goes here, I am not adding it to keep this example simple.
   * See next/jest examples for more information.
   */
};

module.exports = async () => ({
  ...(await createJestConfig(customJestConfig)()),
  transformIgnorePatterns: [
    // The regex below is just a guess, you might tweak it
    '/node_modules/(?!(react-leaflet/lib|@react-leaflet/core/lib)/)',
  ],
});
```

Listing F.3: Jest configuration file source code

Appendix G

Application Dockerfile

Provides the Dockerfile for the Next.js application, detailing steps to build and run the project in a containerized environment.

```
FROM node:20-alpine AS base

FROM base AS deps

RUN apk add --no-cache libc6-compat
WORKDIR /app

COPY package*.json ./

RUN npm update && npm ci

FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

RUN npm run build

FROM base AS runner
WORKDIR /app

ENV NODE_ENV production
RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public

RUN mkdir .next
RUN chown nextjs:nodejs .next

COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000

CMD ["node", "server.js"]
```

Listing G.1: Application Dockerfile

Appendix H

Project Folder Structure

This Figure G.1 shows the project folder structure, offering a visual representation of how the project's files and directories are organized.

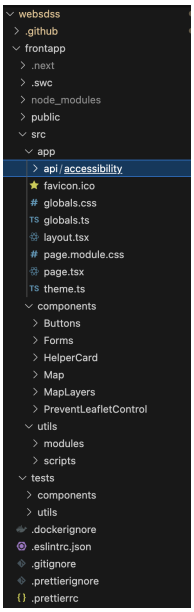

Figure H.1: Project Folder Structure

Appendix I

GeoJSON FeatureCollection with travel time between all hexagons with demographic data of Fortaleza

Contains the GeoJSON FeatureCollection with travel time and demographic data for all hexagons in Fortaleza, used in spatial analysis.


```
{
  "type": "FeatureCollection",
  "features": [
```

```
4     { "type": "Feature", "properties": { "from_id": 0, "to_id": 0, "
      travel_time": 2.0, "h3_polyfill_origem": "89801040323ffff", "input_polygon
      ": "POLYGON ((-38.5015148998841994 -3.8864281756697800,
      -38.5016614561913997 -3.8865957009236900, -38.5016617849552958
      -3.8865958990529097, -38.5028502773181955 -3.8873121427948396,
      -38.5029014517375998 -3.8873278887700300, -38.5029526261570041
      -3.8873436347452199, -38.5030518619307003 -3.8873082105022099,
      -38.5040162484640973 -3.8865575048631200, -38.5040430349388956
      -3.8865366535167301, -38.5055861405075035 -3.8852769755015197,
      -38.5058546520776019 -3.8862310485270601, -38.5058636633201985
      -3.8862630671978002, -38.5060762339852971 -3.8869362076371798,
      -38.5074616623717105 -3.8857987394449838, -38.5074618798021007
      -3.8857985609296897, -38.5074620477898932 -3.8857990918912502,
      -38.5076823234546950 -3.8864953203318500, -38.5077157836520030
      -3.8866508118368599, -38.5085188283866984 -3.8860130998416595,
      -38.5094806651596997 -3.8853649643597699, -38.5097322080608961
      -3.8852117849323200, -38.5099612200874972 -3.8850722514704898,
      -38.5099612382292946 -3.8850722398535296, -38.5112671015442984
      -3.8842360394435702, -38.5106004901362979 -3.8831141101191800,
      -38.5098289199999968 -3.8819690100000002, -38.5092421200000032
      -3.8825729999999998, -38.5090988400000001 -3.8821699800000000,
      -38.5083399600000007 -3.8828030399999998, -38.5079968800000003
      -3.8804730300000001, -38.5010701200000014 -3.8814650099999999,
      -38.5015148998841994 -3.8864281756697800))", "renda_per_capita": 435.35, "
      total": 120.25, "0_a_5_anos": 11.5, "6_a_10_anos": 11.25, "11_a_14_anos":
      7.5, "h3_polyfill_destino": "89801040323ffff" }, "geometry": { "type": "
      Polygon", "coordinates": [ [ [ -38.502319622293442, -3.885863230380012 ], [
       -38.501301977908447, -3.884159277595567 ], [ -38.502270156369853,
      -3.882425394132555 ], [ -38.504255980460599, -3.882395455037078 ], [
      -38.505273635747592, -3.884099406534471 ], [ -38.504305456043042,
      -3.885833298414388 ], [ -38.502319622293442, -3.885863230380012 ] ] ] } },]
5 ...     }
```

Listing I.1: All Types and Interfaces defined for the project

Appendix J

GitHub Actions Workflows Definition Files

Includes the GitHub Actions workflows for CI/CD, detailing how the front-end application code is tested and built.

```yaml
name: Check Front App Code changes

on:
  pull_request:
    paths:
      - "frontapp/**"
      - ".github/workflows/**"
jobs:
  frontapp-pull-request-check:
    name: check front app code changes in the pull reqeust
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node: [20.x]
    timeout-minutes: 10
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node }}
      - name: Cache NPM dependencies
        uses: actions/cache@v4
        id: frontapp-npm-cache
        with:
          path: node_modules
          key: ${{ runner.OS }}-npm-cache-${{ hashFiles('package-lock.json') }}
          restore-keys: |
            ${{ runner.OS }}-npm-cache-
      - name: Install dependencies
        working-directory: frontapp
        run: npm ci
        if: steps.frontapp-npm-cache.outputs.cache-hit != 'true'
      - name: Build the workers code
        working-directory: frontapp
        run: |
          npm run build
        env:
          NODE_OPTIONS: "--max_old_space_size=4096"
      - name: Build Docker image
        working-directory: frontapp
```

```
          run: |
            docker build -t my-docker-image .

  # These are legacy default permissions that were added automatically.
  # https://github.com/tadodotcom/github-management/issues/7
  # Please do reconsider this list when you work on this file and remove
  # permissions you do not actually need! If only pure read-only access to
  # check out and build the source is needed, you can remove this section.
  permissions:
    actions: write
    checks: write
    contents: write
    deployments: write
    issues: write
    packages: write
    pages: write
    pull-requests: write
    repository-projects: write
    security-events: write
    statuses: write
```

Listing J.1: Code changes check workflow definition

```
name: Test Build Check for Front App

on:
  pull_request:
    paths:
      - "frontapp/**"
      - ".github/workflows/**"
jobs:
  frontapp-tests-check:
    name: Runs tests for the Front App
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node: [20.x]
    timeout-minutes: 10
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node }}
      - name: Cache NPM dependencies
        uses: actions/cache@v4
        id: frontapp-npm-cache
        with:
          path: node_modules
          key: ${{ runner.OS }}-npm-cache-${{ hashFiles('package-lock.json') }}
          restore-keys: |
            ${{ runner.OS }}-npm-cache-
      - name: Install dependencies
        working-directory: frontapp
        run: |
          npm ci
        if: steps.frontapp-npm-cache.outputs.cache-hit != 'true'
      - name: Run tests
        working-directory: frontapp
        run: |
          npm run test
```

```
40
41  # These are legacy default permissions that were added automatically.
42  # https://github.com/tadodotcom/github-management/issues/7
43  # Please do reconsider this list when you work on this file and remove
44  # permissions you do not actually need! If only pure read-only access to
45  # check out and build the source is needed, you can remove this section.
46  permissions:
47    actions: write
48    checks: write
49    contents: write
50    deployments: write
51    issues: write
52    packages: write
53    pages: write
54    pull-requests: write
55    repository-projects: write
56    security-events: write
57    statuses: write
```

Listing J.2: Test check workflow definition

```
1   name: Linting Front App Code
2
3   on:
4     pull_request:
5       paths:
6         - "frontapp/**"
7         - ".github/workflows/**"
8   jobs:
9     frontapp-pull-request-check:
10      name: checks for linting errors in the frontapp code
11      runs-on: ubuntu-latest
12      strategy:
13        matrix:
14          node: [20.x]
15      timeout-minutes: 10
16      steps:
17        - name: Checkout
18          uses: actions/checkout@v4
19        - name: Setup Node
20          uses: actions/setup-node@v4
21          with:
22            node-version: ${{ matrix.node }}
23        - name: Cache NPM dependencies
24          uses: actions/cache@v4
25          id: frontapp-npm-cache
26          with:
27            path: node_modules
28            key: ${{ runner.OS }}-npm-cache-${{ hashFiles('package-lock.json') }}
29            restore-keys: |
30              ${{ runner.OS }}-npm-cache-
31        - name: Install dependencies
32          working-directory: frontapp
33          run: |
34            npm ci
35          if: steps.frontapp-npm-cache.outputs.cache-hit != 'true'
36        - name: Code Linting
37          working-directory: frontapp
38          run: |
39            npm run lint
40
41  # These are legacy default permissions that were added automatically.
```

```
42  # https://github.com/tadodotcom/github-management/issues/7
43  # Please do reconsider this list when you work on this file and remove
44  # permissions you do not actually need! If only pure read-only access to
45  # check out and build the source is needed, you can remove this section.
46  permissions:
47    actions: write
48    checks: write
49    contents: write
50    deployments: write
51    issues: write
52    packages: write
53    pages: write
54    pull-requests: write
55    repository-projects: write
56    security-events: write
57    statuses: write
```

Listing J.3: Lint check workflow definition

Appendix K

HexGrid Source Code

Displays the source code for the HexGrid component, used for generating hexagonal grids in spatial analysis.

```
import { GeoJSON } from 'react-leaflet/GeoJSON';
import { useMapEvents } from 'react-leaflet/hooks';
import { HexProperties } from './types';
import {
  LocationForm,
  LocationPoint,
} from '@/components/Forms/AnalysisForm/LocationsForm/types';
import L, { LatLngTuple, Layer, Map } from 'leaflet';
import PopulationHexPopup from '../PopulationHexpPopup';
import React from 'react';
import * as ReactDOMServer from 'react-dom/server';

const HexGrid = ({
  data,
  setLocationFormData,
  analysisFormData,
}: {
  data: GeoJSON.FeatureCollection;
  setLocationFormData: any;
  analysisFormData: any;
}) => {
  const map = useMapEvents({});
  const onClick = (
    feature: GeoJSON.Feature<any, HexProperties>,
    layer: Layer,
    map: Map
  ) => {
    const point = feature.geometry.coordinates[0][0];

    const marker = L.marker([point[1], point[0]]).bindPopup(
      `Point of interest ${feature.properties?.id as string}`
    );

    marker.on({
      click: (e) => {
        setLocationFormData((previous: LocationForm) => {
          marker.remove();
          const newPoints = previous.points.filter(
            (el, i) =>
              el.hexId !=
              (feature.properties?.h3_polyfill || `${point[0]}-${point[1]}`)
          );
          const newLocationData = { ...previous, points: newPoints };
          analysisFormData.current.locationForm = newLocationData;

```

```
            return newLocationData;
          });
        },
        mouseover: (e) => {
          marker
            .bindPopup(
              `Point of interest ${feature.properties?.centroid as string}`
            )
            .openPopup();
        },
      });
      setLocationFormData((previous: LocationForm) => {
        if (previous.points.length > 3) return previous;
        map.addLayer(marker);
        const newLocationData = {
          ...previous,
          points: [
            ...previous.points,
            {
              hexId: feature.properties?.h3_polyfill || `${point[0]}-${point[1]}`,
              latitude: point[0],
              longitude: point[1],
              name: `point ${previous.points.length + 1}`,
            } as LocationPoint,
          ],
        };

        analysisFormData.current.locationForm = newLocationData;

        return newLocationData;
      });
    };

    const onEachFeature = (feature: any, layer: Layer) => {
      let popupContent: any;
      const properties = feature.properties;
      if (properties) {
        popupContent = ReactDOMServer.renderToString(
          <PopulationHexPopup properties={properties} />
        );
      }
      let timeout: any;
      layer.on({
        click: () => {
          onClick(feature, layer, map);
        },
        mouseover: (e: any) => {
          timeout = setTimeout(() => {
            layer.bindPopup(popupContent).openPopup();
          }, 500);
        },
        mouseout: () => {
          clearTimeout(timeout);
          layer.closePopup();
        },
      });
    };

    return (
      <GeoJSON
        key={JSON.stringify(data)}
        data={data}
```

```
        onEachFeature={onEachFeature}
      />
    );
  };

  export default HexGrid;
```

Listing K.1: HexGrid component source code

Appendix L

Map Source Code

Contains the source code for the Map component, crucial for rendering geographic maps in the web application.

```
'use client';

import {
  MapContainer,
  TileLayer,
  Marker,
  Popup,
  useMapEvents,
} from 'react-leaflet';
import { LatLngExpression, LatLngTuple } from 'leaflet';

import 'leaflet/dist/leaflet.css';
import 'leaflet-defaulticon-compatibility/dist/leaflet-defaulticon-compatibility.css';
import 'leaflet-defaulticon-compatibility';
import { useRef, useState, Fragment } from 'react';

import { MapProps } from './types';

const defaults = {
  zoom: 12,
};

const data: GeoJSON.Feature = {
  type: 'Feature',
  properties: {
    name: 'Coors Field',
    amenity: 'Baseball Stadium',
    popupContent: 'This is where the Rockies play!',
  },
  geometry: {
    type: 'Point',
    coordinates: [-104.99404, 39.75621],
  },
};

const Map = (Map: MapProps) => {
  const { zoom = defaults.zoom, posix } = Map;

  return (
    <Fragment>
      <MapContainer
        center={posix}
        zoom={zoom}
        scrollWheelZoom={true}
        style={{ height: '100vh', width: 'auto' }}
```

```
46          >
47            <TileLayer
48              attribution='&copy; <a
    href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
49              url='https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'
50            />
51            {Map.children}
52          </MapContainer>
53        </Fragment>
54      );
55    };
56    export default Map;
```

Listing L.1: Map component source code

Appendix M

MCDMA TOPSIS Module Source Code

Lists the source code for the Multi-Criteria Decision-Making Analysis (MCDMA) TOPSIS Module, responsible for performing multi-criteria decision-making analysis in the system.

```
// Step-by-Step Implementation

// Step 1: Normalize the Decision Matrix
// Method: Normalize each element of the decision matrix by dividing it by the square root of the sum of squares of its column.

// Step 2: Calculate the Weighted Normalized Decision Matrix
// Method: Multiply each normalized value by the corresponding weight.

// Step 3: Determine the Positive Ideal and Negative Ideal Solutions
// Method: For each criterion, determine the maximum and minimum values in the weighted normalized decision matrix.
// The positive ideal solution consists of the maximum values for benefit criteria and the minimum values for cost criteria.
// The negative ideal solution consists of the opposite.

// Step 4: Calculate the Separation Measures
// Method: Calculate the Euclidean distance from each alternative to the positive and negative ideal solutions

// Step 5: Calculate the Relative Closeness to the Ideal Solution
// Method: Calculate the relative closeness of each alternative to the ideal solution.

// Step 6: Rank the Alternatives
// Method: Rank the alternatives based on their relative closeness to the ideal solution, in descending order.

//
  ----------------------------------------------------------------------------------------------------------------------------


// Step 1: Normalize the Decision Matrix
function normalizeMatrix(matrix: number[][]): number[][] {
  let normalizedMatrix: number[][] = [];
  let numRows = matrix.length;
  let numCols = matrix[0].length;
  const epsilon = 0.0001; // Small constant to avoid division by zero

  for (let j = 0; j < numCols; j++) {
    let sumSquares = 0;
    for (let i = 0; i < numRows; i++) {
      sumSquares += (matrix[i][j] === 0 ? epsilon : matrix[i][j]) ** 2;
    }
    let normFactor = Math.sqrt(sumSquares);
    for (let i = 0; i < numRows; i++) {
```

```
        if (!normalizedMatrix[i]) {
          normalizedMatrix[i] = [];
        }
        normalizedMatrix[i][j] =
          (matrix[i][j] === 0 ? epsilon : matrix[i][j]) / normFactor;
      }
    }

    return normalizedMatrix;
  }

  // Step 2: Calculate the Weighted Normalized Decision Matrix
  function weightedNormalizedMatrix(
    normalizedMatrix: number[][],
    weights: number[]
  ): number[][] {
    let weightedMatrix: number[][] = normalizedMatrix.map((row) =>
      row.map((value, index) => value * weights[index])
    );
    return weightedMatrix;
  }
  // Step 3: Determine the Positive Ideal and Negative Ideal Solutions
  function idealSolutions(
    weightedMatrix: number[][],
    criteria: string[]
  ): { positiveIdeal: number[]; negativeIdeal: number[] } {
    let numCols = weightedMatrix[0].length;
    let positiveIdeal: number[] = Array(numCols).fill(-Infinity);
    let negativeIdeal: number[] = Array(numCols).fill(Infinity);

    for (let j = 0; j < numCols; j++) {
      for (let i = 0; i < weightedMatrix.length; i++) {
        if (criteria[j] === 'max') {
          if (weightedMatrix[i][j] > positiveIdeal[j])
            positiveIdeal[j] = weightedMatrix[i][j];
          if (weightedMatrix[i][j] < negativeIdeal[j])
            negativeIdeal[j] = weightedMatrix[i][j];
        } else {
          if (weightedMatrix[i][j] < positiveIdeal[j])
            positiveIdeal[j] = weightedMatrix[i][j];
          if (weightedMatrix[i][j] > negativeIdeal[j])
            negativeIdeal[j] = weightedMatrix[i][j];
        }
      }
    }

    return { positiveIdeal, negativeIdeal };
  }
  // Step 4: Calculate the Separation Measures
  function separationMeasures(
    weightedMatrix: number[][],
    idealSolutions: { positiveIdeal: number[]; negativeIdeal: number[] }
  ): { positiveSeparation: number; negativeSeparation: number }[] {
    let positiveIdeal = idealSolutions.positiveIdeal;
    let negativeIdeal = idealSolutions.negativeIdeal;
    let separations = weightedMatrix.map((row) => {
      let positiveSeparation = Math.sqrt(
        row.reduce(
          (sum, value, index) => sum + (value - positiveIdeal[index]) ** 2,
          0
        )
      );
```

```
      let negativeSeparation = Math.sqrt(
        row.reduce(
          (sum, value, index) => sum + (value - negativeIdeal[index]) ** 2,
          0
        )
      );
      return { positiveSeparation, negativeSeparation };
    });

    return separations;
  }
  // Step 5: Calculate the Relative Closeness to the Ideal Solution
  function relativeCloseness(
    separations: { positiveSeparation: number; negativeSeparation: number }[]
  ): number[] {
    return separations.map(
      (separation) =>
        separation.negativeSeparation /
        (separation.positiveSeparation + separation.negativeSeparation)
    );
  }
  // Step 6: Rank the Alternatives
  function rankAlternatives(closeness: number[]): number[] {
    let sortedIndices = closeness
      .map((value, index) => [value, index] as [number, number])
      .sort((a, b) => b[0] - a[0])
      .map((pair) => pair[1]);
    return sortedIndices;
  }

  const apply = (matrix: number[][], weights: number[], criteria: string[]) => {
    let normalizedMatrix = normalizeMatrix(matrix);
    console.log('normalized', normalizedMatrix);
    let weightedMatrix = weightedNormalizedMatrix(normalizedMatrix, weights);
    console.log('weighted', weightedMatrix);
    let ideals = idealSolutions(weightedMatrix, criteria);
    console.log('ideals', ideals);
    let separations = separationMeasures(weightedMatrix, ideals);
    console.log('separations', separations);
    let closeness = relativeCloseness(separations);
    console.log('close', closeness);
    let ranking = rankAlternatives(closeness);

    console.log('Ranking (0-indexed):', ranking);
    return ranking;
  };

  export {
    apply,
    normalizeMatrix,
    weightedNormalizedMatrix,
    idealSolutions,
    separationMeasures,
    relativeCloseness,
    rankAlternatives,
  };

  // Example usage:
  // let decisionMatrix: number[][] = [
  //   [250, 16, 12, 5],
  //   [200, 16, 8, 3],
  //   [300, 32, 16, 4],
```

```
163   //   [275, 32, 8, 4],
164   //   [225, 16, 16, 2],
165   // ];
166
167   // let weights: number[] = [0.25, 0.25, 0.25, 0.25];
168   // let criteria: string[] = ['max', 'max', 'max', 'max'];
169
170   // apply(decisionMatrix, weights, criteria);
```

Listing M.1: MCDMA TOPSIS module source code

Appendix N

MultiCriteriaForm Source Code

Provides the source code for the MultiCriteriaForm component, enabling users to input and adjust parameters for multi-criteria analysis.

```
import { SyntheticEvent, useState } from 'react';
import {
  FormControl,
  FormLabel,
  Slider,
  FormControlLabel,
  FormGroup,
  Box,
  RadioGroup,
  Button,
  Radio,
  IconButton,
  InputLabel,
  OutlinedInput,
  Divider,
  Tooltip,
  TextField,
  Checkbox,
  Icon,
} from '@mui/material';
import {
  ArrowDownward,
  ArrowUpward,
  Groups,
  Info,
  RemoveCircleOutline,
} from '@mui/icons-material';
import style from './style.module.css';
import { AnalysisForm } from '../types';
import {
  GroupCriteria,
  MultiCriteriaForm as MultiCriteriaFormType,
} from './types.d';

const MAX_INCOME_VALUE = 5000;

const MultiCriteriaForm = ({ formData }: { formData: AnalysisForm }) => {
  const [multiCriteriaForm, setMultiCriteriaForm] =
    useState<MultiCriteriaFormType>(formData.multiCriteriaForm);

  const isChecked = (value: string) => {
    return multiCriteriaForm.ageLevel.filter((age) => age == value).length > 0
      ? true
      : false;
  };
```

```
46      const handleCheckBoxChange = (event: any) => {
47        const value = event.target.value;
48
49        setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
50          if (event.target.checked) {
51            const newMultiCriteriaData = {
52              ...previous,
53              ageLevel: [event.target.value, ...previous.ageLevel],
54            };
55            formData.multiCriteriaForm = newMultiCriteriaData;
56            return newMultiCriteriaData;
57          } else {
58            const newMultiCriteriaData = {
59              ...previous,
60              ageLevel: previous.ageLevel.filter((el) => el !== event.target.value),
61            };
62            formData.multiCriteriaForm = newMultiCriteriaData;
63            return newMultiCriteriaData;
64          }
65        });
66      };
67
68      const handleChange = (e: any) => {
69        setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
70          const newMultiCriteriaData = {
71            ...previous,
72            [e.target.name]: e.target.value,
73          };
74
75          formData.multiCriteriaForm = newMultiCriteriaData;
76
77          return newMultiCriteriaData;
78        });
79      };
80      return (
81        <Box className={style.multiCriteriaFormContainer}>
82          <FormControl className={style.formControl}>
83            <FormLabel sx={{ display: 'flex', alignItems: 'center' }}>
84              <Tooltip title='Range of per capita income of individuals of the group.'>
85                <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
86              </Tooltip>
87              Income Range
88            </FormLabel>
89            <Slider
90              getAriaLabel={() => 'Renda'}
91              value={multiCriteriaForm.incomeRange}
92              max={MAX_INCOME_VALUE}
93              onChange={handleChange}
94              name='incomeRange'
95              valueLabelDisplay='auto'
96              getAriaValueText={function valuetext(value: number) {
97                return `R$${value}`;
98              }}
99            />
100         </FormControl>
101         <Divider sx={{ mb: '20px' }} />
102         <FormControl>
103           <FormLabel
104             id='radio-ageLevel-label'
105             sx={{ display: 'flex', alignItems: 'center' }}
106           >
107             <Tooltip title='Age range of inviduals of the group.'>
```

```
            <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
          </Tooltip>
          Age Level
        </FormLabel>

        <FormGroup>
          <FormControlLabel
            control={
              <Checkbox
                checked={isChecked('0_a_5_anos')}
                value={'0_a_5_anos'}
                onChange={handleCheckBoxChange}
              />
            }
            label='0 a 5 years'
          />
          <FormControlLabel
            control={
              <Checkbox
                value={'6_a_10_anos'}
                checked={isChecked('6_a_10_anos')}
                onChange={handleCheckBoxChange}
              />
            }
            label='6 a 10 years'
          />
          <FormControlLabel
            control={
              <Checkbox
                value={'11_a_14_anos'}
                checked={isChecked('11_a_14_anos')}
                onChange={handleCheckBoxChange}
              />
            }
            label='11 a 14 years'
          />
        </FormGroup>
      </FormControl>
      <Divider sx={{ mb: '20px' }} />
      <FormControl>
        <FormLabel
          id='radio-criteriaType-label'
          sx={{ display: 'flex', alignItems: 'center' }}
        >
          <Tooltip title='Determine if criteria is max or min'>
            <Info fontSize='small' color='disabled' sx={{ mr: 1 }}></Info>
          </Tooltip>
          Criteria Type
        </FormLabel>
        <RadioGroup
          row
          aria-labelledby='radio-criteria-type-label'
          name='criteriaType'
          onChange={handleChange}
          sx={{ color: 'black' }}
          value={multiCriteriaForm.criteriaType}
        >
          <FormControlLabel value='max' control={<Radio />} label='Max' />
          <FormControlLabel value='min' control={<Radio />} label='Min' />
        </RadioGroup>
      </FormControl>
      <Divider sx={{ mb: '20px' }} />
```

```
170
171         <FormGroup className={style.formControl}>
172           <Button
173             onClick={(e) => {
174               setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
175                 const newMultiCriteriaData = {
176                   ...previous,
177                   groups: [
178                     ...previous.groups,
179                     {
180                       name: `Group ${previous.groups.length + 1}`,
181                       weight: multiCriteriaForm.weight,
182                       incomeRange: multiCriteriaForm.incomeRange,
183                       ageLevel: multiCriteriaForm.ageLevel,
184                       criteriaType: multiCriteriaForm.criteriaType,
185                     } as GroupCriteria,
186                   ],
187                 };
188
189                 formData.multiCriteriaForm = newMultiCriteriaData;
190
191                 return newMultiCriteriaData;
192               });
193             }}
194             disabled={multiCriteriaForm.groups.length > 3}
195             className={style.searchButton}
196             size='large'
197             variant='contained'
198           >
199             ADD GROUP
200           </Button>
201         </FormGroup>
202
203         <Divider sx={{ mb: '20px' }} />
204
205         <Box
206           sx={{
207             display: 'flex',
208             mt: '5px',
209             flexWrap: 'wrap',
210           }}
211         >
212           {multiCriteriaForm.groups.map((group, index) => {
213             return (
214               <Box
215                 key={index}
216                 sx={{
217                   display: 'flex',
218                   alignItems: 'center',
219                   width: '100%',
220                   mb: '8px',
221                   justifyContent: 'space-around',
222                 }}
223               >
224                 <Tooltip title={`Groups details: ${JSON.stringify(group)}`}>
225                   <Groups color={'primary'} />
226                 </Tooltip>
227                 <TextField
228                   label={`Group ${index + 1}`}
229                   sx={{ width: '40%' }}
230                   value={
231                     multiCriteriaForm.groups.find((el) => group.name == el.name)
```

```
                    ?.name
                }
                size='small'
                onChange={(e) => {
                  formData.multiCriteriaForm.groups =
                    formData.multiCriteriaForm.groups.map((el) => {
                      if (el.name == group.name) el.name = e.target.value;
                      return el;
                    });

                  setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
                    const newMultiCriteriaData = {
                      ...previous,
                      groups: formData.multiCriteriaForm.groups,
                    };
                    return newMultiCriteriaData;
                  });
                }}
              ></TextField>
              <FormControl size='small' sx={{ width: '20%' }}>
                <InputLabel htmlFor='outlined-adornment-latitude'>
                  Weight
                </InputLabel>
                <OutlinedInput
                  type='number'
                  size='small'
                  id='outlined-adornment-name'
                  label='weight'
                  value={
                    multiCriteriaForm.groups.find((el) => group.name == el.name)
                      ?.weight
                  }
                  onChange={(e) => {
                    formData.multiCriteriaForm.groups =
                      formData.multiCriteriaForm.groups.map((el) => {
                        if (el.name == group.name) el.weight = +e.target.value;
                        return el;
                      });

                    setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
                      const newMultiCriteriaData = {
                        ...previous,
                        groups: formData.multiCriteriaForm.groups,
                      };
                      return newMultiCriteriaData;
                    });
                  }}
                />
              </FormControl>
              <FormControl>
                <Icon>
                  <Tooltip title={`Criteria type: ${group.criteriaType} `}>
                    {group.criteriaType == 'max' ? (
                      <ArrowUpward />
                    ) : (
                      <ArrowDownward />
                    )}
                  </Tooltip>
                </Icon>
              </FormControl>
              <IconButton
                onClick={(e) => {
```

```
                    setMultiCriteriaForm((previous: MultiCriteriaFormType) => {
                      const newGroups = previous.groups.filter(
                        (el, i) => i !== index
                      );
                      const newMultiCriteriaData = {
                        ...previous,
                        groups: newGroups,
                      };
                      formData.multiCriteriaForm = newMultiCriteriaData;
                      return newMultiCriteriaData;
                    });
                  }}
                  color='error'
                >
                  <Tooltip title={`Remove group`}>
                    <RemoveCircleOutline />
                  </Tooltip>
                </IconButton>
              </Box>
            );
          })}
        </Box>
      </Box>
    );
  };
  export default MultiCriteriaForm;
```

Listing N.1: MultiCriteriaForm component source code

Appendix O

onSubmitFunction Source Code

Contains the onSubmitFunction source code, which processes form submissions in the accessibility analysis.

```

const handleSubmit = async (event: any) => {
  event.preventDefault();
  setSubmitting(true);

  const accessibilityAnalysisParams: AccessibilityPayload = {
    pointsOfInterest: locationFormData.points.map(
      (point: LocationPoint) => point.hexId
    ),
    groupCriteria: analysisFormData.current.multiCriteriaForm.groups,
    accessibilityOptions: {
      travelTime: analysisFormData.current.accessibilityForm.travelTime,
      transportMode: analysisFormData.current.accessibilityForm.transportMode,
      accessibilityModel: analysisFormData.current.accessibilityForm.model,
    },
  };

  const response = await fetch('/api/accessibility', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(accessibilityAnalysisParams),
  });

  setResults((await response.json()) as AnalysisResult);

  setSubmitting(false);
  return {};
};
```

Listing O.1: 'onSubmitFunction' code snippet

Appendix P

Main Page Source Code

Shows the source code for the main Page component, which is the central interface of the application.

```
'use client';
import AnalysisForm from '@/components/Forms/AnalysisForm';
import { AnalysisForm as AnalysisFormType } from '@/components/Forms/AnalysisForm/types';

import dynamic from 'next/dynamic';
import { useMemo, useRef, useState } from 'react';
import {
  LocationForm,
  LocationPoint,
} from '@/components/Forms/AnalysisForm/LocationsForm/types';
import { Container, LinearProgress } from '@mui/material';
import {
  AccessibilityPayload,
  AnalysisResult,
} from './api/accessibility/types';
import { ResultsFilter } from '@/components/Forms/AnalysisForm/ResultForm/types';
import MapFiltersForm from '@/components/Forms/MapFiltersForm';

const initialAnalysisForm: AnalysisFormType = {
  locationForm: {
    country: '',
    city: '',
    points: [],
  },
  accessibilityForm: {
    distance: 1000,
    transportMode: 'public',
    year: '2023',
    model: 'passive',
    travelTime: 30,
  },

  multiCriteriaForm: {
    groups: [],
    gender: 'male',
    weight: 1,
    incomeRange: [0, 5000],
    criteriaType: 'max',
    ageLevel: ['6_a_10_anos'],
  },
};

export default function Page() {
  const analysisFormData = useRef<AnalysisFormType>({ ...initialAnalysisForm });
  const [cityGeoJson, setCityGeoJson] =
```

```
46        useState<GeoJSON.FeatureCollection | null>(null);
47
48      const [locationFormData, setLocationFormData] = useState<LocationForm>({
49        ...analysisFormData.current.locationForm,
50      });
51
52      const [submitting, setSubmitting] = useState(false);
53      const [results, setResults] = useState<AnalysisResult | null>(null);
54      const [resultsFilter, setResultsFilter] = useState<ResultsFilter | null>(
55        null
56      );
57
58      const handleSubmit = async (event: any) => {
59        event.preventDefault();
60        setSubmitting(true);
61
62        const accessibilityAnalysisParams: AccessibilityPayload = {
63          pointsOfInterest: locationFormData.points.map(
64            (point: LocationPoint) => point.hexId
65          ),
66          groupCriteria: analysisFormData.current.multiCriteriaForm.groups,
67          accessibilityOptions: {
68            travelTime: analysisFormData.current.accessibilityForm.travelTime,
69            transportMode: analysisFormData.current.accessibilityForm.transportMode,
70            accessibilityModel: analysisFormData.current.accessibilityForm.model,
71          },
72        };
73
74        const response = await fetch('/api/accessibility', {
75          method: 'POST',
76          headers: {
77            'Content-Type': 'application/json',
78          },
79          body: JSON.stringify(accessibilityAnalysisParams),
80        });
81
82        setResults((await response.json()) as AnalysisResult);
83
84        setSubmitting(false);
85        return {};
86      };
87
88      const Map = useMemo(
89        () =>
90          dynamic(() => import('@/components/Map/'), {
91            loading: () => <LinearProgress />,
92            ssr: false,
93          }),
94        []
95      );
96
97      const HexGrid = useMemo(
98        () =>
99          dynamic(() => import('@/components/MapLayers/HexGrid'), {
100           ssr: false,
101         }),
102       []
103     );
104
105     const ResultGrid = useMemo(
106       () =>
107         dynamic(() => import('@/components/MapLayers/ResultGrid'), {
```

```
        ssr: false,
      }),
    []
  );

  const PreventLeafletControl = useMemo(
    () =>
      dynamic(() => import('@/components/PreventLeafletControl'), {
        ssr: false,
      }),
    []
  );

  const resetAnalysis = () => {
    window?.location.reload();
  };

  const [centerPoint, setCenterPoint] = useState([-3.731862, -38.526669]);
  return (
    <div>
      <Map posix={centerPoint}>
        {!results ? (
          <HexGrid
            data={cityGeoJson as GeoJSON.FeatureCollection}
            setLocationFormData={setLocationFormData}
            analysisFormData={analysisFormData}
          />
        ) : (
          <ResultGrid
            data={cityGeoJson as GeoJSON.FeatureCollection}
            results={results}
            mainHex={resultsFilter?.hex}
            mainGroup={resultsFilter?.group}
          ></ResultGrid>
        )}
        <PreventLeafletControl>
          <AnalysisForm
            locationFormData={locationFormData}
            setLocationFormData={setLocationFormData}
            analysisFormData={analysisFormData}
            setCityGeoJson={setCityGeoJson}
            handleSubmit={handleSubmit}
            submitting={submitting}
            results={results}
            resultsFilter={resultsFilter}
            setResultsFilter={setResultsFilter}
            resetAnalysis={resetAnalysis}
          />
          {resultsFilter && (
            <MapFiltersForm
              filters={resultsFilter}
              setFilters={setResultsFilter}
            />
          )}
        </PreventLeafletControl>
      </Map>
    </div>
  );
}
```

Listing P.1: Page component source code

Appendix Q

PreventLeafletControl Source Code

Displays the source code for the PreventLeafletControl component, used for controlling map interactions events that can propagating to other components in the web application.

```
import {
  Box,
  Divider,
  FormLabel,
  IconButton,
  TextField,
  Tooltip,
} from '@mui/material';
import style from './style.module.css';

import { AnalysisForm as AnalysisFormType } from '../AnalysisForm/types';
import { LocationPoint } from '../AnalysisForm/LocationsForm/types';
import { Groups } from '@mui/icons-material';
import { GroupCriteria } from '../AnalysisForm/MultiCriteriaForm/types';
import { useMapEvents } from 'react-leaflet/hooks';
import LocationMapIconButton from '@/components/Buttons/LocationMapIconButton';

const ReviewForm = ({ formData }: { formData: AnalysisFormType }) => {
  const {
    locationForm: locationFormData,
    accessibilityForm: accessibilityFormData,
    multiCriteriaForm: multiCriteriaFormData,
  } = formData;
  const map = useMapEvents({});
  return (
    <Box className={style.reviewFormContainer}>
      <Divider>Points of Interest</Divider>
      <Box className={style.dividerContainer}>
        {locationFormData.points.map((point: LocationPoint, index) => {
          return (
            <Box key={index} className={style.dividerLocationContainer}>
              <LocationMapIconButton
                map={map}
                coords={[point.longitude, point.latitude]}
              />
              <FormLabel>
                <b>Name:</b> {point.name}
              </FormLabel>
              <FormLabel>
                <b>Lat:</b> {point.latitude.toPrecision(8)}
              </FormLabel>
              <FormLabel>
                <b>Long:</b> {point.longitude.toPrecision(8)}
              </FormLabel>
            </Box>
```

```
            );
          })}
        </Box>
        <Divider>Accessibility Parameters</Divider>
        <Box className={style.dividerContainer}>
          <FormLabel sx={{ width: '50%' }}>
            <b>Distance:</b> {accessibilityFormData.distance}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <FormLabel sx={{ width: '50%' }}>
              <b>Transport Mode:</b> {accessibilityFormData.transportMode}
            </FormLabel>
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Year:</b> {accessibilityFormData.year}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Model:</b> {accessibilityFormData.model}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Travel Time:</b> {accessibilityFormData.travelTime}
          </FormLabel>
        </Box>
        <Divider>Population Group Criterias</Divider>
        <Box
          sx={{
            display: 'flex',
            flexDirection: 'row',
            justifyContent: 'space-between',
            flexWrap: 'wrap',
            rowGap: '15px',
          }}
        >
          {multiCriteriaFormData.groups.map((group: GroupCriteria, index) => {
            return (
              <Box
                sx={{
                  display: 'flex',
                  flexDirection: 'column',
                  alignItems: 'center',
                  width: '50%',
                }}
                key={index}
              >
                <Groups color={'primary'} />
                <FormLabel>
                  <b>Name:</b> {group.name}
                </FormLabel>
                <FormLabel>
                  <b>Income Range:</b> {group.incomeRange}
                </FormLabel>
                <FormLabel>
                  <b>Age Levels:</b> {group.ageLevel.join(', ')}
                </FormLabel>
                <FormLabel>
                  <b>Weight:</b> {group.weight}
                </FormLabel>
                <FormLabel>
                  <b>Criteria Type:</b> {group.criteriaType}
                </FormLabel>
              </Box>
            );
```

```
108           })}
109         </Box>
110       </Box>
111     );
112   };
113
114   export default ReviewForm;
```

Listing Q.1: PreventLeafletControl component source code

Appendix R

ResultGrid Source Code

Lists the source code for the ResultGrid component, responsible for rendering the grid of analysis results.

```
import { GeoJSON } from 'react-leaflet/GeoJSON';
import { useMapEvents } from 'react-leaflet/hooks';
import L, { Layer, Map, icon } from 'leaflet';
import { AnalysisResult } from '@/app/api/accessibility/types';
import { POINTS_COLOR } from '@/app/globals';
import PopulationHexPopup from '../PopulationHexpPopup';
import ReactDOMServer from 'react-dom/server';

const ResultGrid = ({
  data,
  results,
  mainHex,
  mainGroup,
}: {
  data: GeoJSON.FeatureCollection;
  results: AnalysisResult;
  mainHex: string | undefined;
  mainGroup: string | undefined;
}) => {
  const getIcon = (color: string) => {
    return new L.Icon({
      iconUrl: `https://raw.githubusercontent.com/pointhi/
      leaflet-color-markers/master/img/marker-icon-2x-${color}.png`,
      shadowUrl:
        'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-shadow.png',
      iconSize: [25, 41],
      iconAnchor: [12, 41],
      popupAnchor: [1, -34],
      shadowSize: [41, 41],
    });
  };
  const map = useMapEvents({});
  const { rankedResults } = results || {};
  const hexPoints = rankedResults.map((r) => r[0]);

  const colorMap = hexPoints.reduce(
    (acc, hex, index) => {
      acc[hex] = POINTS_COLOR[index];
      return acc;
    },
    {} as Record<string, string>
  );
  const onEachFeature = (feature: any, layer: Layer) => {
    let popupContent: any;
    const properties = feature.properties;
```

```
      if (properties) {
        popupContent = ReactDOMServer.renderToString(
          <PopulationHexPopup properties={properties} />
        );
      }

      let timeout: any;
      layer.on({
        mouseover: (e) => {
          timeout = setTimeout(() => {
            layer.bindPopup(popupContent).openPopup();
          }, 500);
        },
        mouseout: () => {
          clearTimeout(timeout);
          layer.closePopup();
        },
      });

      if (hexPoints.includes(feature.properties.h3_polyfill)) {
        const hexPoint = rankedResults.find(
          (r) => r[0] === feature.properties.h3_polyfill
        );
        const point = feature.geometry.coordinates[0][0];
        const marker = L.marker([point[1], point[0]], {
          icon: getIcon(colorMap[feature.properties.h3_polyfill]),
        }).bindPopup(`Point of interest ${hexPoint && hexPoint[0]}`);
        map.addLayer(marker);
      }
    };

    return (
      <GeoJSON
        key={JSON.stringify(data)}
        data={data}
        style={(feature) => {
          const featureHex = results.rankedResults.find((r) =>
            Object.values(r[1]).find((v) =>
              v.hex.includes(feature?.properties.h3_polyfill)
            )
          );

          const featureGroups = featureHex
            ? Object.keys(featureHex[1])
                .map((k) => {
                  if (
                    featureHex &&
                    featureHex[1][k].hex.includes(feature?.properties.h3_polyfill)
                  ) {
                    return k;
                  }
                })
                .filter(Boolean)
            : [];

          let fillColor = 'grey';

          if (featureHex) {
            if (mainHex && featureHex[0] !== mainHex) {
              fillColor = 'grey';
            } else {
              fillColor = colorMap[featureHex[0]];
```

```
            }
          }
          if (mainGroup && !featureGroups.includes(mainGroup)) {
            fillColor = 'grey';
          }

          return {
            color: colorMap[feature?.properties.h3_polyfill],
            weight: 0.5,
            fillColor: fillColor,
            fillOpacity: 0.5,
          };
        }}
        onEachFeature={onEachFeature}
      />
    );
  };

  export default ResultGrid;
```

Listing R.1: ResultGrid component source code

Appendix S

ResultForm Source Code

Contains the source code for the ResultForm component, which manages the form to display analysis results.

```
import {
  Box,
  Button,
  Chip,
  FormLabel,
  Icon,
  Typography,
  IconButton,
  Tooltip,
} from '@mui/material';
import style from './style.module.css';
import { AnalysisResult, RankedResult } from '@/app/api/accessibility/types';
import { ResultsFilter } from './types';
import { AnalysisForm } from '../types';
import { EmojiEvents, Group, TravelExplore } from '@mui/icons-material';
import { Fragment, useState } from 'react';
import ReviewForm from '../../ReviewForm';
import LocationMapIconButton from '@/components/Buttons/LocationMapIconButton';
import { useMapEvents } from 'react-leaflet/hooks';
import { POINTS_COLOR } from '@/app/globals';

const ResultForm = ({
  results,
  analysisFormData,
  resultsFilter,
  setResultsFilter,
  resetAnalysis,
}: {
  results: AnalysisResult | null;
  analysisFormData: AnalysisForm;
  resultsFilter: ResultsFilter | null;
  setResultsFilter: (value: any) => void;
  resetAnalysis: () => void;
}) => {
  const map = useMapEvents({});
  const { rankedResults } = results || {};
  const [showResult, setShowResult] = useState(true);
  const {
    locationForm: { points },
    multiCriteriaForm: { groups },
  } = analysisFormData;

  return (
    <Fragment>
      <Box
```

```
          sx={{
            display: 'flex',
            flexDirection: 'row',
            justifyContent: 'space-between',
            mb: 2,
          }}
        >
          <Button
            color='primary'
            size={showResult ? 'large' : 'small'}
            onClick={() => setShowResult(false)}
            sx={{
              textAlign: 'center',
              width: '50%',
              mb: 2,
              alignSelf: 'center',
              borderBottom: showResult ? 'none' : '1px solid',
            }}
          >
            INPUTS
          </Button>
          <Button
            color='primary'
            size={showResult ? 'small' : 'large'}
            onClick={() => setShowResult(true)}
            sx={{
              textAlign: 'center',
              width: '50%',
              mb: 2,
              alignSelf: 'center',
              borderBottom: showResult ? '1px solid' : 'none',
            }}
          >
            RESULTS
          </Button>
        </Box>
        {showResult ? (
          <Box className={style.resultFormContainer}>
            {rankedResults?.map((r: RankedResult, index) => {
              const point = points.find((p) => p.hexId === r[0]);

              const ResultContainer = (
                <Box key={index} className={style.resultsContainer}>
                  <Chip
                    label={`${index + 1}`}
                    sx={{ borderColor: POINTS_COLOR[index], borderWidth: 2 }}
                    variant='outlined'
                  />
                  <FormLabel sx={{ mt: 2 }} component='legend'>
                    <Icon>
                      <EmojiEvents
                        sx={{
                          color:
                            ['#dddd6e', '#bdbdbd', '#986634'][index] || '#986634',
                        }}
                      />
                    </Icon>
                  </FormLabel>
                  <Typography variant='h6'>
                    <LocationMapIconButton
                      map={map}
                      color={POINTS_COLOR[index]}
```

```
                    coords={[point?.longitude || 1, point?.latitude || 1]}
                  />
                  {point?.name}
                </Typography>
                <FormLabel sx={{ mt: 1 }} component='legend'>
                  Hex
                </FormLabel>
                <Typography variant='body1'> {point?.hexId}</Typography>
                <FormLabel sx={{ mt: 1 }} component='legend'>
                  Latitude
                </FormLabel>
                <Typography variant='body1'>
                  {point?.latitude.toPrecision(7)}
                </Typography>
                <FormLabel sx={{ mt: 1 }} component='legend'>
                  Longitude
                </FormLabel>
                <Typography variant='body1'>
                  {point?.longitude.toPrecision(7)}
                </Typography>
              </Box>
            );
            return ResultContainer;
          })}
          <Typography
            sx={{
              textAlign: 'center',
              width: '100%',
              mt: 3,
              alignSelf: 'center',
            }}
            variant='subtitle2'
            color='primary'
          >
            SCORES:
          </Typography>
          <Box
            sx={{ padding: '15px', height: '300px', overflowY: 'scroll' }}
            className={style.resultFormContainer}
          >
            {rankedResults?.map((r: RankedResult, index) => {
              const ResultContainer = (
                <Box key={index} className={style.resultsContainer}>
                  {Object.keys(r[1]).map((key) => {
                    const point = points.find((p) => p.hexId === r[0]);
                    return (
                      <Fragment key={key}>
                        <Icon>
                          <Group
                            sx={{
                              color: POINTS_COLOR[index],
                              mt: 1,
                            }}
                          />
                        </Icon>
                        <FormLabel sx={{ mt: 1 }} component='legend'>
                          {key}
                        </FormLabel>
                        <Typography key={key} variant='body1'>
                          {r[1][key].total.toPrecision(5)}
                        </Typography>
                        <LocationMapIconButton
```

```
                            title={'Visualize this group criteria on the map'}
                            customIcon={
                              <TravelExplore
                                sx={{ color: POINTS_COLOR[index] }}
                              />
                            }
                            callback={() =>
                              setResultsFilter({ hex: r[0], group: key })
                            }
                            map={map}
                            coords={[point?.longitude || 1, point?.latitude || 1]}
                          />
                        </Fragment>
                      );
                    })}
                  </Box>
                );
                return ResultContainer;
              })}
            </Box>
            <Box
              sx={{
                width: '100%',
                mt: 3,
                display: 'flex',
                flexDirection: 'row',
                justifyContent: 'space-between',
              }}
            >
              <Button disabled>Export in CSV</Button>
              <Button onClick={resetAnalysis}>New Analysis</Button>
            </Box>
          </Box>
        ) : (
          <ReviewForm formData={analysisFormData}></ReviewForm>
        )}
      </Fragment>
    );
  };

  export default ResultForm;
```

Listing S.1: ResultForm component source code

Appendix T

ReviewForm Source Code

Displays the source code for the ReviewForm component, used for reviewing user input during the decision-making process.

```
import {
  Box,
  Divider,
  FormLabel,
  IconButton,
  TextField,
  Tooltip,
} from '@mui/material';
import style from './style.module.css';

import { AnalysisForm as AnalysisFormType } from '../AnalysisForm/types';
import { LocationPoint } from '../AnalysisForm/LocationsForm/types';
import { Groups } from '@mui/icons-material';
import { GroupCriteria } from '../AnalysisForm/MultiCriteriaForm/types';
import { useMapEvents } from 'react-leaflet/hooks';
import LocationMapIconButton from '@/components/Buttons/LocationMapIconButton';

const ReviewForm = ({ formData }: { formData: AnalysisFormType }) => {
  const {
    locationForm: locationFormData,
    accessibilityForm: accessibilityFormData,
    multiCriteriaForm: multiCriteriaFormData,
  } = formData;
  const map = useMapEvents({});
  return (
    <Box className={style.reviewFormContainer}>
      <Divider>Points of Interest</Divider>
      <Box className={style.dividerContainer}>
        {locationFormData.points.map((point: LocationPoint, index) => {
          return (
            <Box key={index} className={style.dividerLocationContainer}>
              <LocationMapIconButton
                map={map}
                coords={[point.longitude, point.latitude]}
              />
              <FormLabel>
                <b>Name:</b> {point.name}
              </FormLabel>
              <FormLabel>
                <b>Lat:</b> {point.latitude.toPrecision(8)}
              </FormLabel>
              <FormLabel>
                <b>Long:</b> {point.longitude.toPrecision(8)}
              </FormLabel>
            </Box>
```

```
            );
          })}
        </Box>
        <Divider>Accessibility Parameters</Divider>
        <Box className={style.dividerContainer}>
          <FormLabel sx={{ width: '50%' }}>
            <b>Distance:</b> {accessibilityFormData.distance}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <FormLabel sx={{ width: '50%' }}>
              <b>Transport Mode:</b> {accessibilityFormData.transportMode}
            </FormLabel>
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Year:</b> {accessibilityFormData.year}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Model:</b> {accessibilityFormData.model}
          </FormLabel>
          <FormLabel sx={{ width: '50%' }}>
            <b>Travel Time:</b> {accessibilityFormData.travelTime}
          </FormLabel>
        </Box>
        <Divider>Population Group Criterias</Divider>
        <Box
          sx={{
            display: 'flex',
            flexDirection: 'row',
            justifyContent: 'space-between',
            flexWrap: 'wrap',
            rowGap: '15px',
          }}
        >
          {multiCriteriaFormData.groups.map((group: GroupCriteria, index) => {
            return (
              <Box
                sx={{
                  display: 'flex',
                  flexDirection: 'column',
                  alignItems: 'center',
                  width: '50%',
                }}
                key={index}
              >
                <Groups color={'primary'} />
                <FormLabel>
                  <b>Name:</b> {group.name}
                </FormLabel>
                <FormLabel>
                  <b>Income Range:</b> {group.incomeRange}
                </FormLabel>
                <FormLabel>
                  <b>Age Levels:</b> {group.ageLevel.join(', ')}
                </FormLabel>
                <FormLabel>
                  <b>Weight:</b> {group.weight}
                </FormLabel>
                <FormLabel>
                  <b>Criteria Type:</b> {group.criteriaType}
                </FormLabel>
              </Box>
            );
```

```
          })}
        </Box>
      </Box>
    );
  };

  export default ReviewForm;
```

Listing T.1: ReviewForm component source code

Appendix U

GeoJSON FeatureCollection with travel time between all hexagons with demographic data of Fortaleza

Lists another GeoJSON FeatureCollection file, containing travel time data between hexagons in Fortaleza.


```
{
  "type": "FeatureCollection",
  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
  "features": [
```

```
{ "type": "Feature", "properties": { "h3_polyfill": "89801040323ffff", "
input_polygon": "POLYGON ((-38.5015148998841994 -3.8864281756697800,
-38.5016614561913997 -3.8865957009236900, -38.5016617849552958
-3.8865958990529097, -38.5028502773181955 -3.8873121427948396,
-38.5029014517375998 -3.8873278887700300, -38.5029526261570041
-3.8873436347452199, -38.5030518619307003 -3.8873082105022099,
-38.5040162484640973 -3.8865575048631200, -38.5040430349388956
-3.8865366535167301, -38.5055861405075035 -3.8852769755015197,
-38.5058546520776019 -3.8862310485270601, -38.5058636633201985
-3.8862630671978002, -38.5060762339852971 -3.8869362076371798,
-38.5074616623717105 -3.8857987394449838, -38.5074618798021007
-3.8857985609296897, -38.5074620477898932 -3.8857990918912502,
-38.5076823234546950 -3.8864953203318500, -38.5077157836520030
-3.8866508118368599, -38.5085188283866984 -3.8860130998416595,
-38.5094806651596997 -3.8853649643597699, -38.5097322080608961
-3.8852117849323200, -38.5099612200874972 -3.8850722514704898,
-38.5099612382292946 -3.8850722398535296, -38.5112671015442984
-3.8842360394435702, -38.5106004901362979 -3.8831141101191800,
-38.5098289199999968 -3.8819690100000002, -38.5092421200000032
-3.8825729999999998, -38.5090988400000001 -3.8821699800000000,
-38.5083399600000007 -3.8828030399999998, -38.5079968800000003
-3.8804730300000001, -38.5010701200000014 -3.8814650099999999,
-38.5015148998841994 -3.8864281756697800))", "id": 1759, "centroid": "POINT
 (-38.50553026596492 -3.8837462322161924)", "baixa_renda": false, "
faixa_de_renda": "C1", "renda_media_familiar": 1259.4, "renda_per_capita":
435.35, "total": 120.25, "0_a_5_anos": 11.5, "6_a_10_anos": 11.25, "11
_a_14_anos": 7.5 }, "geometry": { "type": "Polygon", "coordinates": [ [ [
-38.502319622293442, -3.885863230380012 ], [ -38.501301977908447,
-3.884159277595567 ], [ -38.502270156369853, -3.882425394132555 ], [
-38.504255980460599, -3.882395455037078 ], [ -38.505273635747592,
-3.884099406534471 ], [ -38.504305456043042, -3.885833298414388 ], [
-38.502319622293442, -3.885863230380012 ] ] ] } },
```

```
6     { "type": "Feature", "properties": { "h3_polyfill": "89801040327ffff", "
    input_polygon": "POLYGON ((-38.5015148998841994 -3.8864281756697800,
    -38.5016614561913997 -3.8865957009236900, -38.5016617849552958
    -3.8865958990529097, -38.5028502773181955 -3.8873121427948396,
    -38.5029014517375998 -3.8873278887700300, -38.5029526261570041
    -3.8873436347452199, -38.5030518619307003 -3.8873082105022099,
    -38.5040162484640973 -3.8865575048631200, -38.5040430349388956
    -3.8865366535167301, -38.5055861405075035 -3.8852769755015197,
    -38.5058546520776019 -3.8862310485270601, -38.5058636633201985
    -3.8862630671978002, -38.5060762339852971 -3.8869362076371798,
    -38.5074616623717105 -3.8857987394449838, -38.5074618798021007
    -3.8857985609296897, -38.5074620477898932 -3.8857990918912502,
    -38.5076823234546950 -3.8864953203318500, -38.5077157836520030
    -3.8866508118368599, -38.5085188283866984 -3.8860130998416595,
    -38.5094806651596997 -3.8853649643597699, -38.5097322080608961
    -3.8852117849323200, -38.5099612200874972 -3.8850722514704898,
    -38.5099612382292946 -3.8850722398535296, -38.5112671015442984
    -3.8842360394435702, -38.5106004901362979 -3.8831141101191800,
    -38.5098289199999968 -3.8819690100000002, -38.5092421200000032
    -3.8825729999999998, -38.5090988400000001 -3.8821699800000000,
    -38.5083399600000007 -3.8828030399999998, -38.5079968800000003
    -3.8804730300000001, -38.5010701200000014 -3.8814650099999999,
    -38.5015148998841994 -3.8864281756697800))", "id": 1759, "centroid": "POINT
     (-38.50553026596492 -3.8837462322161924)", "baixa_renda": false, "
    faixa_de_renda": "C1", "renda_media_familiar": 1259.4, "renda_per_capita":
    435.35, "total": 120.25, "0_a_5_anos": 11.5, "6_a_10_anos": 11.25, "11
    _a_14_anos": 7.5 }, "geometry": { "type": "Polygon", "coordinates": [ [ [
    -38.505273635747592, -3.884099406534471 ], [ -38.504255980460599,
    -3.882395455037078 ], [ -38.505224155573849, -3.880661562133611 ], [
    -38.507209987213159, -3.880631612312204 ], [ -38.508227653396773,
    -3.882335562517943 ], [ -38.507259477045658, -3.884069463836739 ], [
    -38.505273635747592, -3.884099406534471 ] ] ] } },]}
```


Listing U.1: Fortaleza City Sectors Hexagons as a GeoJSON file

Appendix V

Script to generate all Hexagons in a Grid given city boundaries

Provides a script to generate all hexagons in a grid based on the city boundaries of Fortaleza, written in JavaScript.

```
const { polygonToCells, cellToBoundary } = require('h3-js');
const fs = require('fs');

function run() {
  const polygon = [
    [-3.679618, -38.464783],
    [-3.730144, -38.45686],
    [-3.792613, -38.424522],
    [-3.822952, -38.456755],
    [-3.828744, -38.489289],
    [-3.8292, -38.523823],
    [-3.772887, -38.599274],
    [-3.710264, -38.567265],
    [-3.728039, -38.501904],
    [-3.685435, -38.47396],
  ];

  const hexagons = polygonToCells(polygon, 9);

  const featureCollectionHexagons: GeoJSON.FeatureCollection = {
    type: 'FeatureCollection',
    features: [],
  };

  const coordinatesHexagons = hexagons.map((hexagon: any) => {
    const coords = cellToBoundary(hexagon, true);
    return {
      type: 'Feature',
      properties: {},
      geometry: {
        type: 'Polygon',
        coordinates: [coords] as GeoJSON.Position[][],
      },
    } as GeoJSON.Feature;
  });

  featureCollectionHexagons.features = coordinatesHexagons;

  const data = JSON.stringify(featureCollectionHexagons);

  fs.writeFile('fortaleza-hexgrid.json', data, (error: any) => {
    if (error) {
```

```
        console.error(error);
        throw error;
      }
      console.log('fortaleza-hexgrid.json written correctly');
    });
  }
  run();
```

Listing V.1: Script to generate hexagons within a city

Appendix W

All Custom Objects Types and Interfaces

Contains all custom object types and interfaces used in the application.

```

  export type AccessibilityPayload = {
  pointsOfInterest: string[];
  groupCriteria: GroupCriteria[];
  accessibilityOptions: AccessibilityOptions;
};

export type RankedResult = [
  string,
  {
    [key: string]: {
      total: number;
      hex: string[];
    };
  },
];

export type AnalysisResult = {
  message: string;
  rankedResults: RankedResult[];
};
export type AccessibilityOptions = {
  travelTime: number;
  transportMode: string;
  accessibilityModel: 'active' | 'passive';
};

export type AccessibilityAnalysisResult = {
  [key: string]: {
    [key: string]: {
      total: number;
      hex: string[]; // Replace `any` with the appropriate type for the `hex` property
    };
  };
};

export interface HexProperties {
  id: string;
  h3_polyfill: string;
  input_polygon: string;
  centroid: string;
  baixa_renda: boolean;
  faixa_de_renda: string;
  renda_media_familiar: number;
  renda_per_capita: number;
  total: number;
```

```
  '0_a_5_anos': number;
  '6_a_10_anos': number;
  '11_a_14_anos': number;
}

// data are from the "to" hexagon
export interface AnalysisHexProperties {
  from_id: number;
  to_id: number;
  travel_time: number;
  h3_polyfill_origem: string;
  input_polygon: string;
  renda_per_capita: number;
  total: number;
  '0_a_5_anos': number;
  '6_a_10_anos': number;
  '11_a_14_anos': number;
  h3_polyfill_destino: string;
}

export type AnalysisForm = {
  locationForm: LocationForm;
  accessibilityForm: AccessibilityForm;
  multiCriteriaForm: MultiCriteriaForm;
};
export type LocationPoint = {
  hexId: string;
  latitude: number;
  longitude: number;
  name: string;
};

export type LocationForm = {
  country: string;
  city: string;
  points: LocationPoint[];
};
export type AccessibilityForm = {
  distance?: number;
  transportMode: 'walking' | 'public' | 'private';
  year: '2020' | '2023';
  model: 'passive' | 'active';
  travelTime: number;
};
export type GroupCriteria = {
  name: string;
  weight: number;
  incomeRange: number[];
  criteriaType: 'max' | 'min';
  ageLevel: string[];
};
export type MultiCriteriaForm = {
  groups: GroupCriteria[];
  incomeRange: number[];
  gender: string;
  weight: number;
  criteriaType: 'max' | 'min';
  ageLevel: string[];
};
export interface MapProps {
  posix: LatLngExpression | LatLngTuple;
  zoom?: number;
```

```
109   children?: React.ReactNode;
110 }
111 export type ResultsFilter = {
112   hex: string;
113   group: string;
114 };
```

Listing W.1: All Types and Interfaces defined for the project

Appendix X

Travel time matrix computation

Describes the travel time matrix computation process, detailing how travel time is calculated between hexagonal areas, and how the spatial and demographic data is processed and stored for further analysis.

X.0.1 Import Libraries

[1]:
```
import geopandas as gpd
import pandas as pd
import h3pandas
from shapely import wkt
from shapely.wkt import loads
import r5py
from r5py import TransportNetwork, TravelTimeMatrixComputer, TransportMode
from datetime import datetime, timedelta

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

X.0.2 Organize Data into Hexagon Grid

Import IBGE Demographic Census Data

[2]:
```
# Read CSV file with geometry polygon column to a GeoDataframe with hexagons␣
 ↪that centroids falls in
csv_file_path = '/ibge-census-data-fortaleza.csv'
df = pd.read_csv(csv_file_path)

# Make sure geometry column is a geometry object
df['geometry'] = df['geometry'].apply(wkt.loads)

# Convert dataframe(Pandas) into a Geodataframe (geopandas)
gdf = gpd.GeoDataFrame(df, crs='epsg:4326') # coordinate reference system␣
 ↪(CRS) format

# Generate an id column with the default index values (important for future␣
 ↪operations: grouping, merging, filtering, etc.)
gdf["id"] = gdf.index
```

Step 1: Generate the hexagon grid. The hexagon centroid will aggregate the information of the Census tract where it 'falls'.

[3]:
```
# 1.1 - Setup hexagon resolution (see table cell statistics -> https://h3geo.
 ↪org/docs/core-library/restable/ )
resolution = 9 # (hexagon edge: 0.200786148km)

# 1.2 - Generate Geodataframe containing all hexagons belonging to polygons␣
 ↪defined in df
gdf_h3 = gdf.h3.polyfill(resolution)
# Depending on the resolution, some polygons won't be considered to have␣
 ↪hexagons belonging to it,
#once a hexagon centroid didn't cut througt its boundaries
```

Step 2: Normalize total population. Larger Census Sectors will have more than 1 hexagon assigned to them. To avoid population multiplication, divide the total population by the number of hexagons that were assigned to each Census Sector.

[4]:
```
# 2.1 - Divide population values by number of hexs within the polygon to␣
 ↪drill down
gdf_h3['total'] = gdf_h3['total']/gdf_h3.h3_polyfill.map(lambda n: {True:␣
 ↪len(n), False: 1} [len(n) > 0])
gdf_h3['0_a_5_anos'] = gdf_h3['0_a_5_anos']/gdf_h3.h3_polyfill.map(lambda n:␣
 ↪{True: len(n), False: 1} [len(n) > 0])
gdf_h3['6_a_10_anos'] = gdf_h3['6_a_10_anos']/gdf_h3.h3_polyfill.map(lambda␣
 ↪n: {True: len(n), False: 1} [len(n) > 0])
gdf_h3['11_a_14_anos'] = gdf_h3['11_a_14_anos']/gdf_h3.h3_polyfill.
 ↪map(lambda n: {True: len(n), False: 1} [len(n) > 0])

# 2.2 drop h3_polyfill column to regenerate hexs but with new population␣
 ↪normalized data
gdf_h3 = gdf_h3.drop(columns=['h3_polyfill'])

# 2.3 regenerate hexs but now with exploded rows(hexs now have their own␣
 ↪rows and are not aggregated in a columns as a list)
gdf_h3 = gdf_h3.h3.polyfill(resolution, explode=True)

# 2.4 - Guarantee crs format after data manipulation
gdf_h3 = gdf_h3.set_crs(epsg=4326, inplace=True)
```

Step 3: Recovery of Sectors that were not included in any Hexagon centroid. We perform a reverse operation and assign the information of the 'excluded' Sectors to the hexagon whose Census Sector centroid is overlapped.

[5]:
```
# 3 - Bring on data from not considered polygons (the ones that don't have␣
 ↪hex centroids falling in)
# This step is important once valuable data would be lost

# 3.1 Change geometry column to represent centroid of input polygon, keeping␣
 ↪previous geometry value(input polygon)
```

```
gdf_h3['input_polygon'] = gdf_h3['geometry']
gdf_h3['geometry'] = gdf_h3['centroid'].apply(wkt.loads)

# 3.2 Generate a hexagon for all input polygon with no hexagon with centroid␣
 ↪falls in
gdf_h3.loc[gdf_h3['h3_polyfill'].isnull(),'h3_polyfill'] = gdf_h3.
 ↪loc[gdf_h3['h3_polyfill'].isnull()].h3.geo_to_h3(resolution).index.values

# 3.3 Groupby h3_polyfill to get rid of hex duplicates, suming population␣
 ↪values
gdf_h3 = gdf_h3.groupby(["h3_polyfill"]).agg({'input_polygon': 'first', 'id':
 ↪ 'first', 'centroid': 'first','baixa_renda': 'first',
                                                'faixa_de_renda':␣
 ↪'first','geometry': 'first','renda_media_familiar': 'first',
                                                'renda_per_capita': 'first',␣
 ↪'CD_GEOCODI': list,'total': sum,'0_a_5_anos': sum,
                                                '6_a_10_anos':␣
 ↪sum,'11_a_14_anos': sum})
```

Step 4: Generate hex polygons

[6]:
```
# 4 - Generate hexs polygons
gdf_h3 = gdf_h3.set_geometry('geometry')
gdf_h3.set_crs(epsg=4326, inplace=True)
gdf_h3 = gdf_h3.h3.h3_to_geo_boundary()
```

X.0.3 Create Transport Network

[7]:
```
# Generate transport network with OSM and GTFS files

# 1 - OSM file of Fortaleza City
# In order to get this file, with the aid of Osmium-tools (https://docs.
 ↪osmcode.org/osmium/latest/osmium-extract.html),
# city was extracted from OSM file of northeast region downloaded from http:/
 ↪/download.geofabrik.de/south-america/brazil/nordeste.html
osm_fp = '/fortaleza.osm.pbf'

# 2 - Folderpath to GTFS data
#bus
gtfs_fp = '/fortaleza_gtfs_bus_20220401.zip'
#metro
gtfs_metrofor = '/fortaleza_gtfs_metrofor_2020.zip'

# 3 - Generate transport network with r5py (see -> https://r5py.readthedocs.
 ↪io/en/stable/ )
transport_network = TransportNetwork(
    osm_fp,
    [
        gtfs_fp,
```

```
        gtfs_metrofor
    ]
)
```

X.0.4 Generate Travel-time Matrix

[8]:
```
# 1 - Prepare hexagons data
gdf_h3 = gdf_h3.reset_index()
gdf_h3['hex_polygon'] = gdf_h3['geometry']
gdf_h3['geometry'] = gdf_h3['geometry'].centroid
gdf_h3['sector_id'] = gdf_h3['id']
gdf_h3['id'] = gdf_h3.index
```

```
/tmp/ipykernel_13682/1761180264.py:4: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to
re-project geometries to a projected CRS before this operation.

  gdf_h3['geometry'] = gdf_h3['geometry'].centroid
```

[9]:
```
# 2 Setup params of control

# 2.1 Departure time
departure_date = datetime(2022,4,1,7,00)

# 2.2 origins
origins = gdf_h3

# 2.3 destinations
destinations = gdf_h3

# 2.4 transport modes
#transport_modes = [TransitMode.TRANSIT, LegMode.WALK]
transport_modes = [
    TransportMode.TRANSIT,
    TransportMode.WALK
]
```

[10]:
```
# 3 - Calculate Travel-time Matrix

# 3.1 - Calculate matrix according to control parameters
travel_time_matrix_computer = TravelTimeMatrixComputer(
    transport_network,
    origins=origins,
    destinations=destinations,
    departure=departure_date,
    transport_modes=transport_modes,
    breakdown=True
)
# 3.2 - Generate Dataframe
travel_time_matrix = travel_time_matrix_computer.compute_travel_times()
```

[12]:
```
travel_time_matrix_1 = travel_time_matrix.merge(origins[['h3_polyfill',␣
↪'input_polygon', 'renda_per_capita', 'total', '0_a_5_anos',
                                                                    ␣
↪   '6_a_10_anos', '11_a_14_anos','hex_polygon', 'id']],
                                                                    ␣
↪left_on="from_id", right_on="id")
# Renomeando código do hexágono de origem
travel_time_matrix_1.rename(columns={'h3_polyfill': 'h3_polyfill_origem'},␣
↪inplace=True)
# Remover coluna id
travel_time_matrix_1 = travel_time_matrix_1.drop('id', axis=1)
```

[13]:
```
travel_time_matrix_2 = travel_time_matrix_1.
↪merge(destinations[['h3_polyfill', 'id']],
                                                                    ␣
↪left_on="to_id", right_on="id")
# Renomeando código do hexágono de origem
travel_time_matrix_2.rename(columns={'h3_polyfill': 'h3_polyfill_destino'},␣
↪inplace=True)
# Remover coluna id
travel_time_matrix_2 = travel_time_matrix_2.drop('id', axis=1)
```

[17]:
```
travel_time_matrix_2.columns
```

[17]:
```
Index(['from_id', 'to_id', 'travel_time', 'h3_polyfill_origem',
       'input_polygon', 'renda_per_capita', 'total', '0_a_5_anos',
       '6_a_10_anos', '11_a_14_anos', 'hex_polygon', 'h3_polyfill_destino'],
      dtype='object')
```

X.0.5 Data output

[15]:
```
#travel_time_matrix_2.to_csv('/travel-time-matrix.csv', index=False)
```

Appendix Y

Unit Tests Source Code

Displays the source code for unit tests of various components and modules in the application.

```
  import { NextRequest, NextResponse } from 'next/server';
import { POST } from './route'; // Adjust the import path accordingly
import { readData } from '@/utils/modules/accessibility';
import * as MCDMA from '@/utils/modules/mcdma';

jest.mock('@/utils/modules/accessibility', () => ({
  readData: jest.fn(),
}));

jest.mock('@/utils/modules/mcdma', () => ({
  topsis: {
    apply: jest.fn(),
  },
}));

describe('API Route POST /api/yourApiRoute', () => {
  const mockRequestPayload = {
    pointsOfInterest: ['hex1', 'hex2'],
    groupCriteria: [
      {
        name: 'Group 1',
        weight: 0.5,
        incomeRange: [0, 800],
        criteriaType: 'max',
        ageLevel: ['0_a_5_anos'],
      },
    ],
    accessibilityOptions: {
      travelTime: 10,
      transportMode: 'car',
      accessibilityModel: 'passive',
    },
  };

  const mockAccessibilityAnalysisResult = {
    hex1: {
      'Group 1': {
        total: 10,
        hex: ['origin_hex1'],
      },
    },
    hex2: {
      'Group 1': {
        total: 20,
```

```
45          hex: ['origin_hex2'],
46        },
47      },
48    };
49
50    const mockMultiCriteriaAnalysisResult = [0, 1]; // Assuming two hex locations
51
52    it('returns 400 when the request is invalid', async () => {
53      const invalidRequest = {
54        pointsOfInterest: null,
55        groupCriteria: null,
56        accessibilityOptions: null,
57      };
58
59      const req = new NextRequest(JSON.stringify(invalidRequest));
60      const res = await POST(req, new NextResponse());
61
62      const resJson = await res.json();
63
64      expect(res.status).toBe(400);
65      expect(resJson.error).toBe('Invalid request. Use the template.');
66      expect(resJson.templatePayload).toBeDefined();
67    });
68
69    it('returns 200 and processes data correctly', async () => {
70      // Mock the return value of readData and MCDMA.topsis.apply
71      (readData as jest.Mock).mockResolvedValue(mockAccessibilityAnalysisResult);
72      (MCDMA.topsis.apply as jest.Mock).mockReturnValue(
73        mockMultiCriteriaAnalysisResult
74      );
75
76      const req = new NextRequest(JSON.stringify(mockRequestPayload));
77      const res = await POST(req, new NextResponse());
78
79      const resJson = await res.json();
80
81      expect(res.status).toBe(200);
82      expect(readData).toHaveBeenCalledWith(
83        mockRequestPayload.pointsOfInterest,
84        mockRequestPayload.groupCriteria,
85        mockRequestPayload.accessibilityOptions
86      );
87      expect(MCDMA.topsis.apply).toHaveBeenCalledWith(
88        [[10], [20]],
89        [0.5],
90        ['max']
91      );
92
93      expect(resJson.message).toBe('Data received');
94      expect(resJson.rankedResults).toEqual([
95        ['hex1', mockAccessibilityAnalysisResult.hex1],
96        ['hex2', mockAccessibilityAnalysisResult.hex2],
97      ]);
98    });
99
100   it('returns 500 when an error occurs', async () => {
101     (readData as jest.Mock).mockRejectedValue(new Error('Test Error'));
102
103     const req = new NextRequest(JSON.stringify(mockRequestPayload));
104     const res = await POST(req, new NextResponse());
105
106     const resJson = await res.json();
```

```
107
108     expect(res.status).toBe(500);
109     expect(resJson.error).toBe('Something went wrong');
110   });
111 });
```

Listing Y.1: Accessibility module test source code

```
1   import React from 'react';
2   import { render, fireEvent, screen } from '@testing-library/react';
3   import '@testing-library/jest-dom/extend-expect';
4   import AccessibilityForm from './index';
5   import { AccessibilityForm as AccessibilityFormType } from './types';
6   import { MultiCriteriaForm } from '../MultiCriteriaForm/types';
7
8   describe('AccessibilityForm Component', () => {
9     const mockFormData = {
10      accessibilityForm: {
11        model: 'passive',
12        year: '2023',
13        transportMode: 'public',
14        travelTime: 30,
15      } as AccessibilityFormType,
16      locationForm: {
17        country: 'Country',
18        city: 'City',
19        points: [],
20      },
21      multiCriteriaForm: {
22        groups: [
23          {
24            name: 'Group 1',
25            criteriaType: 'max',
26            ageLevel: ['0_a_5_anos'],
27            incomeRange: [0, 800],
28            weight: 1,
29            // Add any other required properties here
30          },
31        ],
32        incomeRange: [],
33        gender: '',
34        weight: 1,
35        criteriaType: 'max' as 'max',
36        ageLevel: [],
37        // Add any other required properties here
38      } as MultiCriteriaForm,
39    };
40
41    it('renders the component with correct initial values', () => {
42      render(<AccessibilityForm formData={mockFormData} />);
43
44      // Check that the radio buttons and slider have the correct initial values
45      expect(screen.getByLabelText('Passive')).toBeChecked();
46      expect(screen.getByLabelText('2023')).toBeChecked();
47      expect(screen.getByLabelText('WALKING')).not.toBeChecked();
48      expect(screen.getByLabelText('PUBLIC')).toBeChecked();
49      expect(screen.getByLabelText('PRIVATE')).not.toBeChecked();
50      expect(screen.getByRole('slider')).toHaveValue(30);
51    });
52
53    it('allows the user to change the accessibility model', () => {
54      render(<AccessibilityForm formData={mockFormData} />);
```

```
55
56      const passiveRadio = screen.getByLabelText('Passive');
57      const activeRadio = screen.getByLabelText('Active');
58
59      fireEvent.click(passiveRadio);
60      expect(passiveRadio).toBeChecked();
61      expect(mockFormData.accessibilityForm.model).toBe('passive');
62
63      // Note: The 'Active' model is disabled, so it shouldn't change on click
64      fireEvent.click(activeRadio);
65      expect(activeRadio).not.toBeChecked();
66      expect(passiveRadio).toBeChecked();
67    });
68
69    it('allows the user to change the year', () => {
70      render(<AccessibilityForm formData={mockFormData} />);
71
72      const year2020Radio = screen.getByLabelText('2020');
73      const year2023Radio = screen.getByLabelText('2023');
74
75      // The year 2020 radio is disabled, so it should not be checked
76      fireEvent.click(year2020Radio);
77      expect(year2020Radio).not.toBeChecked();
78      expect(year2023Radio).toBeChecked();
79
80      // Changing to the already selected year should work normally
81      fireEvent.click(year2023Radio);
82      expect(year2023Radio).toBeChecked();
83    });
84
85    it('allows the user to change the transport mode', () => {
86      render(<AccessibilityForm formData={mockFormData} />);
87
88      const walkingRadio = screen.getByLabelText('WALKING');
89      const publicRadio = screen.getByLabelText('PUBLIC');
90      const privateRadio = screen.getByLabelText('PRIVATE');
91
92      // Only the 'PUBLIC' option should be enabled
93      fireEvent.click(walkingRadio);
94      expect(walkingRadio).not.toBeChecked();
95      expect(publicRadio).toBeChecked();
96
97      fireEvent.click(privateRadio);
98      expect(privateRadio).not.toBeChecked();
99      expect(publicRadio).toBeChecked();
100
101     fireEvent.click(publicRadio);
102     expect(publicRadio).toBeChecked();
103   });
104
105   it('allows the user to change the travel time', () => {
106     render(<AccessibilityForm formData={mockFormData} />);
107
108     const slider = screen.getByRole('slider');
109     expect(slider).toHaveValue(30);
110
111     fireEvent.change(slider, { target: { value: '20' } });
112     expect(slider).toHaveValue(20);
113     expect(mockFormData.accessibilityForm.travelTime).toBe(20);
114   });
115 });
```

Listing Y.2: AccessibilitForm component test source code

```
import React from 'react';
import { render } from '@testing-library/react';
import '@testing-library/jest-dom';
import Map from '.';
import { MapContainer, TileLayer } from 'react-leaflet';

// Mocking Leaflet's MapContainer and TileLayer as they rely on browser APIs
jest.mock('react-leaflet', () => {
  const MapContainer = ({ children }: any) => <div>{children}</div>;
  const TileLayer = () => <div>TileLayer</div>;
  return { MapContainer, TileLayer };
});

describe('Map Component', () => {
  const defaultProps = {
    posix: [39.75621, -104.99404] as [number, number], // Using the coordinates from your GeoJSON data as default
    zoom: 12,
    children: <div>Marker</div>, // You can mock children to see if they render properly
  };

  it('should render the map container with correct props', () => {
    const { getByText } = render(<Map {...defaultProps} />);

    // Check if the map container is rendered with the TileLayer component
    expect(getByText('TileLayer')).toBeInTheDocument();

    // Check if the children passed are rendered correctly
    expect(getByText('Marker')).toBeInTheDocument();
  });

  it('should render with default zoom if none is provided', () => {
    const { getByText } = render(<Map posix={defaultProps.posix} />);

    // Again, we're checking for the TileLayer mock to confirm render
    expect(getByText('TileLayer')).toBeInTheDocument();
  });
});
```

Listing Y.3: Map component test source code

```
import {
  apply,
  normalizeMatrix,
  weightedNormalizedMatrix,
  idealSolutions,
  separationMeasures,
  relativeCloseness,
  rankAlternatives,
} from './index'; // Adjust the import path accordingly

describe('TOPSIS Implementation', () => {
  let decisionMatrix: number[][];
  let weights: number[];
  let criteria: string[];

  beforeEach(() => {
    decisionMatrix = [
```

```
        [250, 16, 12, 5],
        [200, 16, 8, 3],
        [300, 32, 16, 4],
        [275, 32, 8, 4],
        [225, 16, 16, 2],
      ];
      weights = [0.25, 0.25, 0.25, 0.25];
      criteria = ['max', 'max', 'max', 'max'];
    });

    it('should normalize the decision matrix correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      expect(normalized).toHaveLength(decisionMatrix.length);
      expect(normalized[0]).toHaveLength(decisionMatrix[0].length);

      // Further tests can include specific value checks if needed
    });

    it('should calculate the weighted normalized matrix correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      const weighted = weightedNormalizedMatrix(normalized, weights);

      expect(weighted).toHaveLength(normalized.length);
      expect(weighted[0]).toHaveLength(normalized[0].length);

      // Further tests can include specific value checks if needed
    });

    it('should determine the ideal solutions correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      const weighted = weightedNormalizedMatrix(normalized, weights);
      const ideals = idealSolutions(weighted, criteria);

      expect(ideals.positiveIdeal).toHaveLength(decisionMatrix[0].length);
      expect(ideals.negativeIdeal).toHaveLength(decisionMatrix[0].length);

      // Further tests can include specific value checks if needed
    });

    it('should calculate the separation measures correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      const weighted = weightedNormalizedMatrix(normalized, weights);
      const ideals = idealSolutions(weighted, criteria);
      const separations = separationMeasures(weighted, ideals);

      expect(separations).toHaveLength(decisionMatrix.length);

      // Further tests can include specific value checks if needed
    });

    it('should calculate the relative closeness to the ideal solution correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      const weighted = weightedNormalizedMatrix(normalized, weights);
      const ideals = idealSolutions(weighted, criteria);
      const separations = separationMeasures(weighted, ideals);
      const closeness = relativeCloseness(separations);

      expect(closeness).toHaveLength(decisionMatrix.length);

      // Further tests can include specific value checks if needed
    });

```

```
    it('should rank the alternatives correctly', () => {
      const normalized = normalizeMatrix(decisionMatrix);
      const weighted = weightedNormalizedMatrix(normalized, weights);
      const ideals = idealSolutions(weighted, criteria);
      const separations = separationMeasures(weighted, ideals);
      const closeness = relativeCloseness(separations);
      const ranking = rankAlternatives(closeness);

      expect(ranking).toHaveLength(decisionMatrix.length);

      // Further tests can include specific value checks if needed
    });

    it('should execute the full TOPSIS process correctly', () => {
      const ranking = apply(decisionMatrix, weights, criteria);

      expect(ranking).toHaveLength(decisionMatrix.length);

      // Depending on the expected result, you can test for specific rankings
      // Example: expect(ranking).toEqual([2, 3, 0, 1, 4]);
    });
  });
```

Listing Y.4: MCDMA module test source code

```
/* eslint-disable react/display-name */
import React, { MutableRefObject } from 'react';
import { render, fireEvent, screen, waitFor } from '@testing-library/react';
import '@testing-library/jest-dom';
import AnalysisForm from '.';
import { LocationForm as LocationFormType } from './LocationsForm/types';
import { AccessibilityForm as AccessibilityFormType } from './AccessibilityForm/types';
import { MultiCriteriaForm as MultiCriteriaFormType } from './MultiCriteriaForm/types';

// Mock subcomponents that are used in AnalysisForm
// eslint-disable-next-line react/display-name
jest.mock('./LocationsForm', () => () => <div>LocationsForm Component</div>);
// eslint-disable-next-line react/display-name
jest.mock('./AccessibilityForm', () => () => (
  <div>AccessibilityForm Component</div>
));
// eslint-disable-next-line react/display-name
jest.mock('./MultiCriteriaForm', () => () => (
  <div>MultiCriteriaForm Component</div>
));
// eslint-disable-next-line react/display-name
// jest.mock(
//   './Stepper',
//   () =>
//     ({ steps, onSubmit }: { steps: any; onSubmit: any }) => (
//       <div>
//         <div>Stepper Component</div>
//         <button onClick={onSubmit}>Submit</button>
//       </div>
//     )
// );
// eslint-disable-next-line react/display-name
jest.mock('./ResultForm', () => () => <div>ResultForm Component</div>);
// eslint-disable-next-line react/display-name
jest.mock('../ReviewForm', () => () => <div>ReviewForm Component</div>);
// eslint-disable-next-line react/display-name
jest.mock('@/components/HelperCard', () => () => (
```

```
38      <div>HelperCard Component</div>
39    ));
40
41    describe('AnalysisForm Component', () => {
42      const defaultProps = {
43        locationFormData: {
44          country: '',
45          city: '',
46          points: [],
47        },
48        setLocationFormData: jest.fn(),
49        analysisFormData: {
50          current: {
51            locationForm: {
52              country: '',
53              city: '',
54              points: [],
55            } as LocationFormType,
56            accessibilityForm: {
57              transportMode: 'walking',
58              year: '2020',
59              model: 'passive',
60              travelTime: 1,
61            } as AccessibilityFormType,
62            multiCriteriaForm: {} as MultiCriteriaFormType,
63          },
64        } as MutableRefObject<any>,
65        setCityGeoJson: jest.fn(),
66        handleSubmit: jest.fn(),
67        submitting: false,
68        results: null,
69        resultsFilter: null,
70        setResultsFilter: jest.fn(),
71        resetAnalysis: jest.fn(),
72      };
73
74      it('renders AnalysisForm component', () => {
75        render(<AnalysisForm {...defaultProps} />);
76
77        expect(screen.getByText('Open Analysis Form')).toBeInTheDocument();
78        expect(screen.queryByText('LocationsForm Component')).toBeInTheDocument();
79      });
80
81      it('opens and closes the form when the Fab is clicked', () => {
82        render(<AnalysisForm {...defaultProps} />);
83
84        const openButton = screen.getByText('Open Analysis Form');
85        fireEvent.click(openButton);
86
87        expect(screen.getByText('Close Analysis Form')).toBeInTheDocument();
88
89        fireEvent.click(screen.getByText('Close Analysis Form'));
90
91        // expect(
92        //   screen.queryByText('LocationsForm Component')
93        // ).not.toBeInTheDocument();
94      });
95
96      it('displays an error if validation fails', async () => {
97        const customProps = {
98          ...defaultProps,
99          analysisFormData: {
```

```
          current: {
            locationForm: { country: '', city: '', points: [] },
            accessibilityForm: {
              transportMode: 'walking',
              year: '2020',
              model: 'passive',
              travelTime: 30,
            } as AccessibilityFormType,
            multiCriteriaForm: {
              criteriaType: 'max' as 'max',
              ageLevel: [],
              groups: [],
              incomeRange: [],
              gender: '',
              weight: 1,
              // Add any other missing properties here
            } as MultiCriteriaFormType,
          },
        },
      };

      render(<AnalysisForm {...customProps} />);

      fireEvent.click(screen.getByText('Open Analysis Form'));
      fireEvent.click(screen.getByText('Next'));

      await waitFor(() => {
        expect(
          screen.getByText('Please select at least one location')
        ).toBeInTheDocument();
      });
    });

    it('shows the loading state when submitting', () => {
      const customProps = {
        ...defaultProps,
        submitting: true,
      };

      render(<AnalysisForm {...customProps} />);

      expect(screen.getByText('Calculating...')).toBeInTheDocument();
    });

    it('renders ResultForm when results are available', () => {
      const customProps = {
        ...defaultProps,
        results: { message: 'result', rankedResults: [] },
      };

      render(<AnalysisForm {...customProps} />);

      expect(screen.getByText('ResultForm Component')).toBeInTheDocument();
    });

    it('opens and closes the helper modal', () => {
      render(<AnalysisForm {...defaultProps} />);

      fireEvent.click(screen.getByText('Open Analysis Form'));

      fireEvent.click(screen.getByTestId('helpModal')); // Assuming the modal role

```

```
162       expect(screen.getByText('HelperCard Component')).toBeInTheDocument();
163     });
164   });
```

Listing Y.5: AnalysisForm component test source code

Appendix Z

Route API Test Source Code

Contains source code for API route tests, ensuring the backend API routes function correctly.

```
import { NextRequest, NextResponse } from 'next/server';
import { POST } from './route'; // Adjust the import path accordingly
import { readData } from '@/utils/modules/accessibility';
import * as MCDMA from '@/utils/modules/mcdma';

jest.mock('@/utils/modules/accessibility', () => ({
  readData: jest.fn(),
}));

jest.mock('@/utils/modules/mcdma', () => ({
  topsis: {
    apply: jest.fn(),
  },
}));

describe('API Route POST /api/yourApiRoute', () => {
  const mockRequestPayload = {
    pointsOfInterest: ['hex1', 'hex2'],
    groupCriteria: [
      {
        name: 'Group 1',
        weight: 0.5,
        incomeRange: [0, 800],
        criteriaType: 'max',
        ageLevel: ['0_a_5_anos'],
      },
    ],
    accessibilityOptions: {
      travelTime: 10,
      transportMode: 'car',
      accessibilityModel: 'passive',
    },
  };

  const mockAccessibilityAnalysisResult = {
    hex1: {
      'Group 1': {
        total: 10,
        hex: ['origin_hex1'],
      },
    },
    hex2: {
      'Group 1': {
        total: 20,
```

```
        hex: ['origin_hex2'],
      },
    },
  };

  const mockMultiCriteriaAnalysisResult = [0, 1]; // Assuming two hex locations

  it('returns 400 when the request is invalid', async () => {
    const invalidRequest = {
      pointsOfInterest: null,
      groupCriteria: null,
      accessibilityOptions: null,
    };

    const req = new NextRequest(JSON.stringify(invalidRequest));
    const res = await POST(req, new NextResponse());

    const resJson = await res.json();

    expect(res.status).toBe(400);
    expect(resJson.error).toBe('Invalid request. Use the template.');
    expect(resJson.templatePayload).toBeDefined();
  });

  it('returns 200 and processes data correctly', async () => {
    // Mock the return value of readData and MCDMA.topsis.apply
    (readData as jest.Mock).mockResolvedValue(mockAccessibilityAnalysisResult);
    (MCDMA.topsis.apply as jest.Mock).mockReturnValue(
      mockMultiCriteriaAnalysisResult
    );

    const req = new NextRequest(JSON.stringify(mockRequestPayload));
    const res = await POST(req, new NextResponse());

    const resJson = await res.json();

    expect(res.status).toBe(200);
    expect(readData).toHaveBeenCalledWith(
      mockRequestPayload.pointsOfInterest,
      mockRequestPayload.groupCriteria,
      mockRequestPayload.accessibilityOptions
    );
    expect(MCDMA.topsis.apply).toHaveBeenCalledWith(
      [[10], [20]],
      [0.5],
      ['max']
    );

    expect(resJson.message).toBe('Data received');
    expect(resJson.rankedResults).toEqual([
      ['hex1', mockAccessibilityAnalysisResult.hex1],
      ['hex2', mockAccessibilityAnalysisResult.hex2],
    ]);
  });

  it('returns 500 when an error occurs', async () => {
    (readData as jest.Mock).mockRejectedValue(new Error('Test Error'));

    const req = new NextRequest(JSON.stringify(mockRequestPayload));
    const res = await POST(req, new NextResponse());

    const resJson = await res.json();
```

```
107
108       expect(res.status).toBe(500);
109       expect(resJson.error).toBe('Something went wrong');
110     });
111   });
```

Listing Z.1: API Route source code