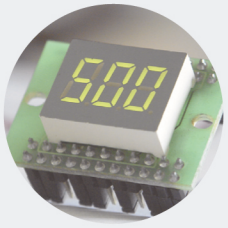
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Solução de Mobilidade numa Cidade Inteligente: Um Sistema de Informação ao Público em Tempo-real

RODRIGO TEIXEIRA GUILHERME AGUIAR RODRIGUES
julho de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Smart City Mobility Solution: A Real-time Passenger Information System

Rodrigo Guilherme Rodrigues

Master in Electrical and Computer Engineering
Specialization Area of Autonomous Systems

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

July, 2023

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Autonomous Systems.

Candidate: Rodrigo Guilherme Rodrigues, No. 1180659,
1180659@isep.ipp.pt

Scientific Guidance: Prof. Paula Viana, pmv@isep.ipp.pt

Scientific Co-Guidance: Prof. Luís Lima, lul@isep.ipp.pt

Company: Associação Porto Digital

Advisor: Eng. Gonçalo Fonseca, goncalo.fonseca@portodigital.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

July, 2023

Acknowledgements

A realização desta dissertação deve agradecimentos a algumas pessoas que, cada uma à sua maneira, me ajudaram a manter os níveis de ânimo elevados e de ansiedade baixos e que sempre se demonstraram disponíveis para me apoiar.

Devo um agradecimento especial à Engenheira Paula Viana por me ter dado a honra de a ter como orientadora da minha tese de mestrado. Com a sua vasta experiência e conhecimento, orientou a escrita desta dissertação sem nunca limitar a minha criatividade individual, cooperando no desenvolvimento de um documento do qual tenho total orgulho. Adicionalmente, agradeço ao Engenheiro Luís Lima por, nos momentos essenciais de análise do documento, ter dado a sua opinião especializada e de enorme relevo.

Ao Engenheiro Paulo Calçada agradeço o voto de confiança no desenvolvimento do projeto proposto nesta dissertação. A ele agradeço a oportunidade de poder desenvolver este projeto na Porto Digital, tendo todos os recursos e apoio à minha disposição.

Agradeço a todos os colegas da Porto Digital por terem, direta ou indiretamente, contribuído para um excelente ambiente no local de trabalho, para o qual me dirigia diariamente com total motivação. Individualmente, agradeço ao Engenheiro André Miranda por todo o acompanhamento, apoio e conhecimento que me transmitiu durante todo o meu percurso e ao João Figueiredo pela sua experiência e criatividade em momentos de minha maior desinspiração. Finalmente, agradeço ao Engenheiro Gonçalo Fonseca pelo seu, sempre que necessário, apoio técnico e fornecimento das ferramentas digitais imprescindíveis para o desenvolvimento e implementação do sistema apresentado neste documento.

Incondicionalmente e eternamente agradecido à minha família pelo apoio diário e estabilidade imprescindível que me proporcionaram durante todo o meu percurso académico.

Finalmente, e não menos importante, aos meus amigos, Ric, Johnny, Loureiro, Luís, Nando, Carol, Kika, Tiago e Tomás, agradeço pelo constante bem-estar, bom humor e companhia que proporcionam diariamente. É um prazer partilhar as minhas conquistas convosco e um orgulho ver-vos a vencer na vida.

A todos, e mais alguns, obrigado!

Abstract

In alignment with Porto Digital's mission and values, the Smart City vision aims to establish a robust digital and technological infrastructure that facilitates open innovation and co-creation. By centralising useful public data, this perspective enables the development and implementation of innovative technologies that enhance the quality of life for citizens.

The mobility and public transport sector have a significant impact on the flow of people in a city. Therefore, raises the need to create innovative changes and technologies that provide helpful information to improve and benefit the users of these public services. This will enable them to access data that was previously only available to private entities.

The project proposed in this thesis aims to comply with the mindset and visions mentioned above, developing and creating a Passenger Information System, with a device and server application that can be optimised and adapted to other application scenarios. Creating a system with two major components, the device(s) and the server side, requires software development that ensures full compatibility between both agents while ensuring a plug-and-play hardware solution with minimal manual configuration and setup required.

The prototyping process of the device system, complying with formal requirements and budget limitations, raises major complexities that are tackled with the design of simple but smart solutions minimising the complexity of the embedded system and interface between the hardware components of this device, the microcomputer and an E-paper display. The choice of designing a server architecture that can be fully containerised and compatible with any system using Docker containers makes the idea of a large-scale implementation project possible.

The results of the final functional device and server prototypes show that the system proposed in this project is capable of adapting to multiple application scenarios while being compliant with the open-source mindset and green digital transition vision and mission, benefiting from the digital infrastructure and tools made available by Porto Digital and the city of Porto.

Keywords: Smart City, Open Data, Real-time Passenger Information Systems, E-paper Displays, Embedded Systems

Resumo

A missão e os valores da empresa Porto Digital alinham-se com uma visão de *Smart City* que se apresenta com o objetivo de criar uma infraestrutura digital e tecnológica robusta com vista a facilitar a inovação aberta e a cocriação. A centralização de dados públicos de elevada utilidade permitirá o desenvolvimento e a implementação de tecnologias inovadoras que irão melhorar a qualidade de vida dos cidadãos.

O sector da mobilidade e dos transportes públicos tem um impacto significativo no fluxo de pessoas numa cidade. Torna-se fundamental o desenvolvimento de tecnologias inovadoras que forneçam informação útil aos seus utilizadores. A melhoria destes serviços é o principal objetivo, permitindo ao cidadão o acesso a dados que são tratados somente por entidades privadas.

O projeto proposto nesta dissertação pretende respeitar a missão e visão supramencionada, apresentando o desenvolvimento e implementação de um sistema de informação ao público em tempo-real. Adicionalmente, este sistema inclui um aplicativo WEB otimizado e adaptável a outros possíveis cenários de aplicação. A criação de um sistema com dois componentes, o dispositivo e o servidor, requer o desenvolvimento de *software* que garanta a compatibilidade total entre ambos e que assegure uma solução de *hardware plug-and-play* que exija o mínimo de configuração manual.

O processo de prototipagem do dispositivo, respeitando os requisitos formais e limites orçamentais, gera dificuldades que são resolvidas com a conceção de soluções simples minimizando a complexidade do sistema e da interface entre os componentes de hardware. A opção de uma arquitetura de servidor que possa ser totalmente compartimentada e compatível com qualquer sistema que utilize contentores Docker torna viável uma implementação em grande escala.

Os resultados do protótipo funcional final do dispositivo e do servidor demonstram uma capacidade de adaptação a múltiplos cenários de aplicação, estando em conformidade com a mentalidade *open-source* e uma visão de transição digital e verde, beneficiando da infraestrutura digital e ferramentas disponibilizadas pela empresa Porto Digital e a cidade do Porto.

Palavras-Chave: Cidade Inteligente, Dados Abertos, Sistemas de Informação ao Público em Tempo-real, Ecrãs E-Paper, Sistemas Embebidos

Contents

List of Figures

List of Tables

Listings

List of Acronyms

3GPP	3rd Generation Partnership Project
ADC	Analog to Digital-converter
API	Application Programming Interface
CAN	Controller Area Network
CI/CD	continuous integration and deployment
CS	Chip-Select
DNS	Domain Name System
EPD	E-Paper Display
ETA	Estimated Time of Arrival
FI-PPP	Future Internet Public Private Partnership
FIWARE	Future of Internet Ware
GPIO	General-purpose input/output
GTFS	General Transit Feed Specification
HAT	hardware attached on top
I2C	Inter-Integrated Circuit
ICT	Information and Communication Technology
IoT	Internet of Things
IRQ	Interrupt-Request
LoRa	Long Range Radio
LTE-M	Long Term Evolution for Machines
MaaS	Mobility as a Service
MAC	Media Access Control

MDS	Mobility Data Specification
MISO	Master-Input Slave-Output
MOSI	Master-Output Slave-Input
MSC	Message Sequence Chart
NB-IoT	Narrowband IoT
OS	Operating System
OTP	OpenTripPlanner
PCB	Printed Circuit Board
PIS	Passenger Information Systems
PWM	Pulse-Width Modulation
RDBMS	Relational Database Management Systems
RTPIS	Real-time Passenger Information Systems
SCLK	Serial Clock signal
SMEs	Small and medium-sized enterprises
SoC	System on a Chip
SPI	Serial Peripheral Interface
SS	Slave Select
UART	Universal Asynchronous Receiver/Transmitter

Chapter 1

Introduction

The first chapter of this thesis aims to present the project that underlined the developed and implemented work. Also, this chapter aims to clarify and introduce the context from which this project benefits. Finally, it also serves the purpose of guiding the reader through the document, providing a general explanation of how it is organised and the main contents of each chapter.

To introduce the reader to this project, it is relevant to answer the question, "How was this project born, and what is the need for such implementation?". Accordingly, section 1.1 outlines the motivation and benefits of creating systems that improve the information available to commuters, optimising urban mobility.

The Municipality of Porto has a past and present in smart city projects, transforming the city into a common living laboratory. Accordingly, in section 1.2, we will present the city of Porto and the company as a matter of context exposition.

Passenger Information Systems (PIS) present several requirements and objectives to be fully developed and implemented, as they are intended to be used publicly to display useful information. General project objectives for a successful implementation, hardware specifications, budget limitations and product requirements will be described in section 1.3.

Finally, for the reader's complete understanding and recognition of this thesis's organisation and structure, in section 1.4, we have included a comprehensive and detailed description of the contents presented in each chapter. This section aims to assist the reader in comprehending the flow and organisation of the thesis, generally introducing the topics addressed in each chapter.

1.1 Motivation

Mobility as a Service (MaaS) is a relevant topic when discussing the smart cities paradigm. It calls for new actions and services to improve the general experience of commuters [1]. Accordingly, any new project should provide new answers to optimise public services' performance and provide citizens with as much information as possible.

The latest Resilience and Recuperation Plan from the Portuguese Government is divided into several distinct sectors of activity that aim to implement reforms and investments with the objective of national sustainable economic growth. One of the Agendas created under this program intends to develop and implement a "Customer Journey" concept with disruptive solutions to improve tourists' experience along their complete journey inside our country [2]. This Agenda is divided into seven areas of activity, identified as Work Packages, and Porto Digital is one of the stakeholders in the second Work Package, focused on Territory Digitalisation.

Work Package Number 2 aims to enhance the tourist experience by introducing infrastructure, services, and training improvements. While Porto Digital acts on all three sectors, this thesis project proposal focuses on improving the services sector while adhering to its vision of positively impacting citizens' quality of life and promoting sustainable development in Porto. To this end, the digital platform Explore Porto [3] has been developed, centralising all public transport, points of interest, private mobility services, and path-planning information. The aim is to provide tourists with easy access to reliable and relevant data for their daily travels, ensuring a hassle-free and seamless journey.

Finally, the program aims to enhance the accessibility of pertinent information for tourists. In the past, Porto Digital developed and implemented the "Beacons", an Explore Porto ally combining a QR Code and NFC technology, to facilitate access to the platform's information. This underscores the relevance of developing a companion to the Beacons to display relevant information visually. This environmentally friendly and cost-effective device will present location-specific information and public transportation details, using state-of-the-art technologies to create a compact outdoor display. This display will use an adaptable and optimised web service to generate the images to be displayed.

1.2 The City of Porto and Project Context

In section 1.1, we introduced the motivation behind the project proposed in this thesis and the general aims for its implementation. To better comprehend the involvement of both the Municipality and Porto Digital in this type of project, it is important to understand both stakeholders' vision and mission from which this

project will take advantage. In the next subsections, we will introduce the company where this project was developed and present Porto from an innovation and technological point-of-view, explaining why it is entitled to be called a Smart City.

1.2.1 The Company

Porto Digital is a private non-profit association, created in 2004, which currently has as associates the Municipality of Porto, the University of Porto (UP) and the company Metro do Porto.

The company's main goal is to promote projects in the area of digital technologies in the context of the city of Porto and its metropolitan area, as well as the development of projects in the areas of innovation and experimentation.

Their mission is to use Innovation and Digital Transition as accelerators of the city's transformation, contributing to better and more efficient public services with the guarantee of a significant impact on citizens' quality of life and sustainable development.

1.2.2 Porto as a Smart City

There has been a serious and solid investment in Porto's Smart City vision and strategy, leveraged mainly by the Municipality and its partners. Combining innovation and efficient management, the Municipality's vision supports a technological and digital infrastructure and provides the necessary tools for open innovation and co-creation. Open data management and use with a sight of promoting citizens' digital rights embrace a mindset of transparency to all the Municipality stakeholders.

The Municipality and its partners endorse initiatives that converge with the city's broad strategy and vision, developed in a common living laboratory - the city of Porto itself – open to experimentation and use of open data made available by the Municipality. Most of these initiatives are sustained in local and international networks of stakeholders, underlining the importance of engaging the ecosystem of different entities, organisations, private companies and citizens in the objectives and ultimate success of all projects developed on Porto. It also corroborates Porto Digital's experience managing and mentoring different partnerships in this subject matter.

Confirming this positioning, Porto is already part of the following European networks: EuroCities, Living_In.EU, OASC, ICC, Cities Coalition for Digital Rights, ENoll, UserCentriCities, DS4SSCC. In addition, Porto leverages projects that promote collaboration between cities, such as the CityCatalyst project.

The involvement mentioned above, and the influence of Porto's Municipality in smart city initiatives is possible by considering the city's digital and communication infrastructure already implemented. This infrastructure combines a network of

more than 4000km of Optical Fiber, providing connection to over 220 Municipality Buildings, Schools, Universities, Health Care Centers and other Institutions and a free Wi-Fi network with high capacity connectivity in an immersive way in the main areas of the city surpassing 10 million sessions in the last year (2022).

Finally, the project proposed in the Master's thesis will take clear advantage of being developed in a city with this Innovation mindset and technological infrastructure with all the needed tools for its implementation, complying with all the system's requirements.

1.3 Objectives

This thesis project proposal aims to create a device that can display useful information and is supported by a flexible web application that manages its content. The aim is to develop a system that can be implemented in various scenarios to improve public access to reliable and relevant data. The primary scenario application is to implement the devices in the new Porto bus stops, displaying real-time arrival information to passengers.

To address this main objective, mission and vision, the project has to comply with a set of requirements from both hardware and software perspectives. To achieve a successful and functional prototype that contributes to this area of public information systems, the development of this project, including the device and web application, has to follow the subsequent list of specific objectives:

1. Formal definition of device-specific requirements: the project mandates using E-paper displays and sets a strict budget limit of 400€ per device. This objective aims to define the limits for the system's development and rules to comply with the mission of creating an eco-friendly and cost-effective device for a large-scale implementation scenario.

2. Study and analyse the device hardware components. This objective raises the need for research about the technology behind E-paper displays, their hardware and software requirements such as driver boards and interface-specific needs. It also requires an analysis of the market availability of microcomputers, considering the hardware requirements to interface with the E-paper displays. This will require useful knowledge of embedded systems, general interfacing communication protocols, and adapting the project to the low market availability and flexibility of options.

3. Development and design of the hardware solution and the driving software to interface with the E-paper displays. This objective requires the correct design and production of a PCB that enables the connection of a pair of displays

without requiring complex wiring connections. Also, we will need to develop the driver software to enable the correct interface with multiple displays.

4. Research and design of the complete system architecture, defining the responsibilities of both device and server side. This objective is useful for understanding the workflow and requirements of each system component and defining the most relevant software functions on each side of this system. This architecture should comply with the requirement of developing a server system with complete isolation and replication capability.

5. Software development of all the server components. This objective requires programming a web application and configuring a database, ideally optimised and adaptable for any device application scenario. Additionally, the device software should comply with the system architecture and require minimal manual configuration at setup, thus involving developing a standard Operating System (OS) image.

By achieving these objectives, the resulting system will comply with all the defined requirements and be available for public implementation. Adhering to an open-source and standardisation mindset, the final product aims to be completely adaptable, thus requiring minimal configuration to build the same system in another city with their specific information. While this project will benefit from national and regional Agendas, developing a system respecting these flags and mission is important to contribute to other cities' and regions' sustainable growth and digital transition.

1.4 Dissertation Structure

This dissertation is divided into five chapters guiding the reader through every relevant phase of the study and development of the proposed project. Chapter 1 has the objective of clarifying the motivations that contributed to the birth of this project. Also, this chapter aims to clarify and introduce the context from which this project benefits, presenting the city of Porto and the company where it was developed. Finally, it describes all the objectives and requirements for successful development and implementation.

Chapter 2 has high relevancy as it theoretically introduces all the key topics around Passenger Information Systems. Organised in an "inverted pyramid" scheme, all the knowledge regarding Smart Cities, areas of action of a Smart City, Mobility, Public transport and Real-time Passenger Information systems will be provided and explained. Also, all the technologies used in developing this system will be presented so that everyone can understand the implementation process and the justifications behind each option regarding hardware and software.

Chapter 3 describes all work done to accomplish a functional prototype. Divided into three parts, the first section 3.1 has the intention of providing a general overview and introduction of the complete system architecture and main components, clearing the way for the remaining two sections. These two sections, 3.2 and 3.3, divide the implementation chapter of this project into Hardware and Software development, respectively. In the hardware section, we present the justifications behind each hardware component and the work done to surpass any constraint. Finally, the software-focused section aims to clarify all the software development done on the device or server sides. Both sections have code listings and useful diagrams included for aiding the comprehension of more complex algorithms, with the least relevant ones being included in the Appendix A, B and C.

Chapter 4 aims to present the main results of both hardware and software implementations. It displays the images of the prototype device, including the case designed for the first fully functional deliverable. Also, focusing on the server side, we discuss the relevancy of high-load testing of the web app and its components. Accordingly, this chapter also includes the tests and results made to the web application and server during high load, demonstrating a large-scale implementation analysis.

Finally, in the last chapter 5, we will summarise the main conclusions of the research and development of the proposed project. Also, we will discuss the performance of the implementation complying with the defined objectives, highlighting the main accomplishments and work developed. The chapter also identifies areas for future work, including potential features and advancements for further development that resulted from findings in research about this project.

Chapter 2

Literature Review

The literature review of this dissertation aims to address current knowledge about key concepts for this project by examining previously published studies and analysing the state-of-the-art concerning the topics where the project is included.

This chapter will be organised in an inverted-pyramid descending path, starting with a broader and general topic review in section 2.1, introducing the concept of a *smart city* and all the areas of action within this definition. Getting further and deeper, the focus of section 2.2 will be the specific area of action of mobility and the technology enablers that take part in creating the digital infrastructure needed for this project. With a more specific and technical analysis of the state-of-the-art, in section 2.3, the key aspects of similar projects already implemented will be analysed, considering the impact of its implementation and the technological point-of-view of the systems. Finally, section 2.4 provides essential information to understand the project's implementation. It breaks down each critical element of the proposed project and introduces them theoretically to aid the reader's comprehension of the complete system development.

2.1 Smart City

The first time that the term *smart cities* appeared in scientific publications was in 1999, in a study developed by Margaret Tan where the author describes the city of Singapore as entitled to be defined as a *Smart City* as a result of its advancements

in *Information Highways* [4]. However, studies using the term *smart city* and thesis about this topic only became more relevant and frequent in the late 2010s [5]. The evolution and incremental implementation of Information and Communication Technology (ICT) in urban areas justifies this growth, which is only possible by creating infrastructures capable of transporting and monitoring large amounts of information [6].

Nowadays, smart cities are widely discussed and a matter of public interest, with stakeholders in areas like technological, social, economic, political, etc., all grouping and getting actively involved in the development and growth of these so-called smart cities, being the citizen's quality of life improvement the main goal [6]. The European Commission describes a smart city as one where traditional networks and services are positively influenced by using ICT to benefit its citizens. It also reinforces the idea of getting all the stakeholders acting together towards the same goal by creating the term *Smart Cities Marketplace* where cities, industries, Small and medium-sized enterprises (SMEs), investors, banks and researchers will have their contribution [7].

The Mckinsey Global Institute, in a report published in 2018 [8], presents the contemporary shift in the definition and conception of *smart cities*. The report highlights the evolution from the notion that a city can become intelligent through the sheer implementation of infrastructures and technologies, resulting in greater efficiency in all city operations, to the present understanding that digital technology and data are employed with a strong focus on improving the quality of life and decision-making for citizens. The report also recognises that smartphones have become crucial tools that enable access to information regarding transit, traffic, health services, safety alerts, and community news, thus serving as *keys to the city* for millions of individuals.

When all stakeholders, including citizens, collaborate, there are certain critical areas where their efforts and contributions can have a more significant impact based on their relevance to social, economic, and climate-related concerns. The European Commission's *marketplace* ideology [7] categorises the main areas of action for smart cities based on their operational levels. These areas are:

- sustainable urban mobility
- sustainable districts and built environment
- integrated infrastructures and processes in energy, information and communication technologies and transport
- citizen focus
- policy and regulation
- integrated planning and management

- knowledge sharing
- baselines, performance indicators and metrics
- open data governance
- standards
- business models, procurement and funding

The Smart Cities Marketplace employs an integrated *Explore-Shape-Deal* Matchmaking process that aids in knowledge exchange and developing Smart City solutions. This process includes *Explore*, *Shape*, and *Deal* phases, which build on each other to enable Smart City projects' development, implementation, replication, and upscaling. The *Explore* phase provides access to Smart Cities knowledge and promotes idea generation, while the *Shape* phase transforms ideas into bankable projects and facilitates dialogue among key stakeholders. The final *Deal* phase enables project promoters to connect with the financing community to secure funding [7]. Sustaining this approach, Deloitte presents smart cities as having six well-defined domains: economy, environment and energy, government and education, living and health, mobility, and safety and security [9]. These are the main areas of action where all the technological developments should have a more significant impact and focus.

Upon comparison of both approaches, it can be ascertained that although they differ in their methods of defining domains and areas of focus, they share fundamental principles centred on leveraging technology to promote the well-being of citizens while ensuring the sustainable and efficient growth of the urban regions. These principles reflect the evolving definition of smart cities, emphasising using digital technology and data-driven insights to enable evidence-based decision-making, enhance public services, and encourage innovation. By prioritising citizen-centric solutions that drive sustainable and inclusive growth, these approaches aim to foster resilient and adaptive cities capable of meeting the needs of present and future generations.

In the modern era, the sustainable growth of a smart city relies on the fast advancements of technology and the efficiency of how they are implemented in our society. According to the World Bank [10], 56% of the world's population lives in urban areas, but the trend is to increment. By 2050 we will see this value reach 70%, considering that 80% of global gross domestic value will be generated in urban areas; this raises the challenge of scaling the existing services and sustainable growth of the metropolitan regions, with the assistance of technology implementations.

Smart cities are emerging due to technological progress and can achieve sustainable growth if properly managed. The previously cited McKinsey Global Institute study [8] reveals that by adopting smart transportation and energy systems, smart cities can potentially lower greenhouse gas emissions by 10-15% by 2030 by adopting smart transportation and energy systems. To attain sustainable growth, smart

cities must prioritise using renewable energy sources, like wind or solar power, and energy-efficient infrastructure, such as intelligent street lighting and smart grids. Furthermore, smart cities should promote public transportation and shared mobility solutions, such as bike-sharing programs, to reduce private vehicles on the road and their related carbon emissions.

Another critical aspect of the sustainable growth of smart cities is the adoption of *circular economy* principles. The European Commission defines a *circular economy* as an economic system that minimises waste, reuses or recycles products and materials, and conserves natural resources [11]. By applying *circular economy* principles, smart cities can decrease their reliance on finite resources and reduce waste generation. For instance, Amsterdam Smart City launched a *circular energy* project, which employs waste heat from data centers to heat homes and offices, reducing the city's carbon footprint [12]. Smart cities can promote sustainable growth by implementing *circular economy* initiatives, benefiting their communities and the environment.

2.2 Mobility and Technology Enablers in Smart Cities

2.2.1 Mobility Sector

In section 2.1, the main areas of action in a smart city were described, defined and compared, but the focus of the present section will reside on the specific topic of mobility as this study relies on projects implemented in this area of action in smart cities.

Mobility in smart cities can be divided into two project rating systems, the first generation having the primary focus on the design and construction of road projects, and the newer generation systems, defined by moving towards a new life-cycle approach where communities evaluate a broader range of projects, plans, and programs, focusing mainly on public transportation [13].

According to the literature review presented in [14], the author approaches the public transport system as a three key players system, where all of them have relevant participation in all the processes regarding public transportation. The stakeholders are the passengers, agency/operator and community/municipality/government, and their operations are illustrated in figure 2.1.

With the evolution of cities to the concept of *smart city*, mobility also follows the same path, and a new core concept emerges with this evolution, Mobility as a Service (MaaS). According to MaaS Alliance, MaaS is "the integration of various forms of transport services into a single mobility service accessible on demand" [1]. The new concept changes the view on mobility from a possession perspective to a philosophy of a service provided by the municipalities and operators to the citizens. MaaS Alliance also introduces MaaS as "... the integration of various forms

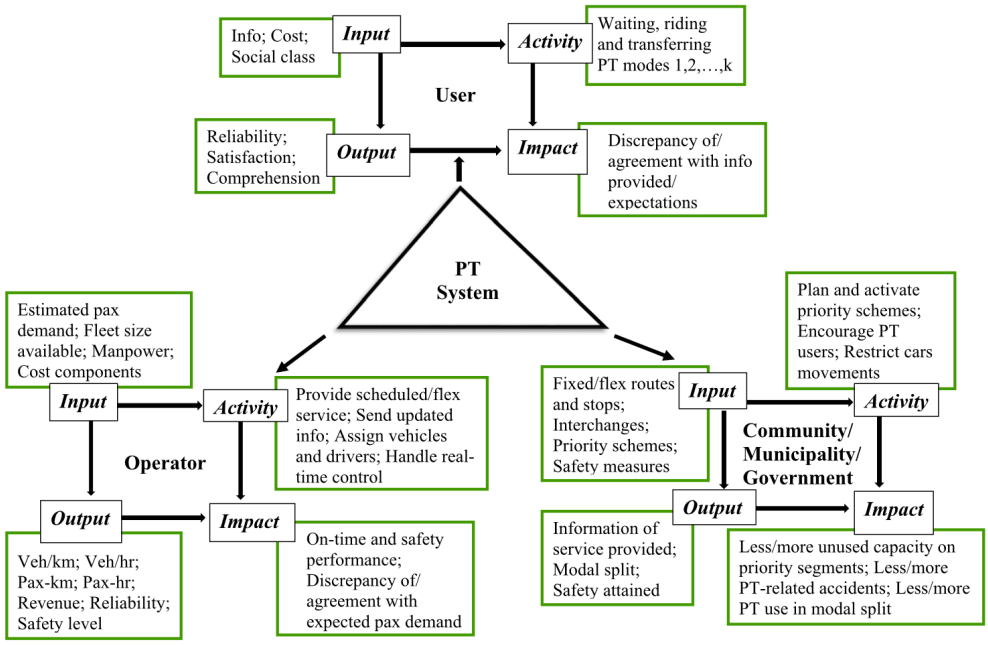


Figure 2.1: The three key players of the PT system and their processes. [14]

of transport services into a single mobility service accessible on demand by MaaS Alliance. To meet a customer's request, a MaaS operator facilitates a diverse menu of transport options, be they public transport, ride-, car- or bike-sharing, taxi or car rental/lease, or a combination thereof. For the user, MaaS can offer added value through a single application to provide access to mobility with a single payment channel instead of multiple ticketing and payment operations. For its users, MaaS should be the best value proposition by helping them meet their mobility needs and solve the inconvenient parts of individual journeys and the entire system of mobility services.".

2.2.2 Technolgy Enablers in Smart Cities

Mobility as a Service brought to cities the need to implement new technological systems in public transport services. Implementing new systems in big cities will raise problems such as the deprecation of older systems, the difficulties in interoperability between them (new and old systems) and the lack of standardisation. To mitigate these specific issues, the Future of Internet Ware (FIWARE) platform was created, being this platform the result of a project funded by the European Commission under its Future Internet Public Private Partnership (FI-PPP) program, where cooperation with private companies in the technology sector for the development of Internet-based technologies occurs [15].

FIWARE stands in the open-source market as a platform that enables the interoperation of smart solutions, building bridges between applications and other platform components. These components are also described as generic enablers. FIWARE generic enablers provide general-purpose functions through a group of Application Programming Interface (API), which allows developers to create applications which could be used in multiple sectors.

As described in [15], FIWARE platform generic enablers are organised in five blocks where components from different layers can communicate. *Context Broker* is the only mandatory block of the presented architecture model responsible for managing context information. *Context information* consists of context entities and context properties. Context entities represent real-life objects such as parking, sensors etc. *Context attributes* are properties of context entities such as parking capacity or the number of free spots.

Using the FIWARE platform, and its main principles, the authors in [16] describe a possible Internet of Things (IoT) architecture for smart cities. Keeping in mind the essential components of the FIWARE IoT platform, the main features proposed by the authors are divided into three domains: *Data Collection*, where the main components are sensors, real-time data consumers, and IoT agents responsible for unifying all the data gathered; *Persistence of Big Data*, a layer responsible of persisting measurements and generating history based in new changes; and *Visualisation*, the analysing layer, processing the data to be used in decision-making processes.

The authors in [15] and [16] find common ground in the same principle regarding the benefits of FIWARE, presenting the platform as a solution to creating open standards in smart city technological projects, disabling the sense of *vendor-locking* and enabling the interoperability between general-purpose components.

Going even further on what are the technology enablers of mobility projects in smart cities, and with the same objectives and flags of the FIWARE platform, the Open Mobility Foundation creates the open-source tool Mobility Data Specification (MDS). The Foundation defines MDS as "a digital tool that helps cities to better manage transportation in the public right of way. MDS standardises communication and data-sharing between cities and private mobility providers, such as e-scooter and bike-share companies. This allows cities to share and validate policy digitally, enabling vehicle management and better outcomes for residents. Plus, it provides mobility service providers with a framework they can reuse in new markets, allowing for seamless collaboration that saves time and money." [17].

MDS works as a set of APIs, like protocols, that allow data to flow securely between cities and providers. The three primary APIs allow cities and providers to communicate differently: Provider, Policy and Agency, as defined in [17]. This tool overview is described in figure 2.2.

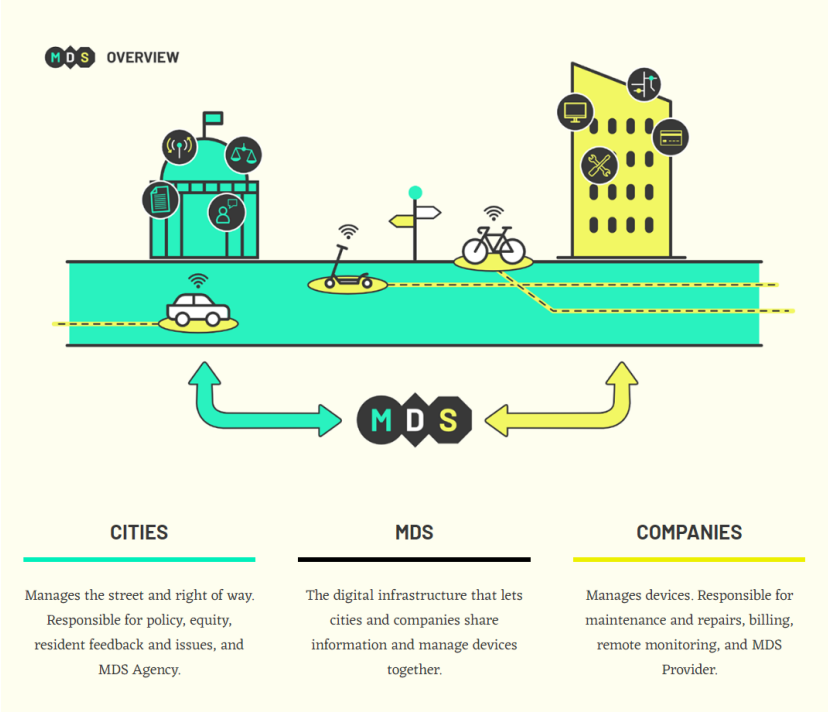


Figure 2.2: Mobility Data Specification Overview [17]

Nowadays, almost all digital infrastructure used in mobility in smart cities worldwide shares the same data protocol, the General Transit Feed Specification (GTFS) [18]. The GTFS protocol is a widely adopted open data format that has transformed how public transit data is shared, accessed, and analysed. Developed by Google in 2005 and nowadays maintained by MobilityData, GTFS has become a de facto standard for publishing transit data and has been adopted by thousands of transit agencies worldwide. The GTFS protocol provides a structured way to represent transit schedules, stops, routes, and trip data in a machine-readable format that can be quickly processed and analysed. The protocol defines a set of data fields and values that describe transit information, such as stop locations, arrival and departure times, route information, and trip schedules. These data elements are organised into text files that can be easily shared and updated through various channels, including transit agency websites, data portals, and third-party apps.

The city of Porto, where the main project described in this dissertation is implemented, has its own GTFS dataset accessible via this link[1]. The public availability of this dataset results from a European ruling (DR EU2017/1926) that makes it mandatory for every city's transport operators to provide their data in formats that can be integrated with different platforms, for example, the GTFS format.

[1]https://opendata.porto.digital/dataset/horarios-paragens-e-rotas-em-formato-gtfs-stcp

The above-mentioned European ruling opened doors to an area of software development that aims to implement the data provided by transport operators, and Digitransit (https://digitransit.fi/) is the best example and already a standard in the flagship European smart cities. Developed in Helsinky by HSL and Traficom, Digitransit is an open-source *journey planner* platform with several components to provide features such as mobile routing and real-time information support.

Combining both the Digitransit platform and MDS, Porto Digital developed Explore Porto [3], a platform that combines both project principles resulting in a web application that aggregates all the transport and mobility providers' information, such as Buses, metro, taxis and urban private mobility companies, real-time information of all these means of transportation, and also displays all of Porto's points of interest. In figures 2.3 and 2.4, we can observe two examples of this platform. On the first one, we can see the arrivals of a chosen bus stop, and in the second one, we have a list of all the public transport options for a route previously specified.

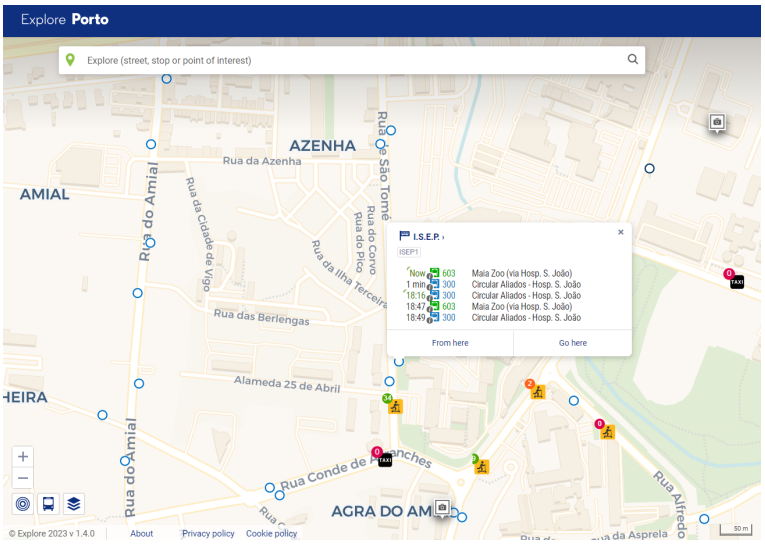


Figure 2.3: ExplorePorto Screenshot with Bus Stop Information

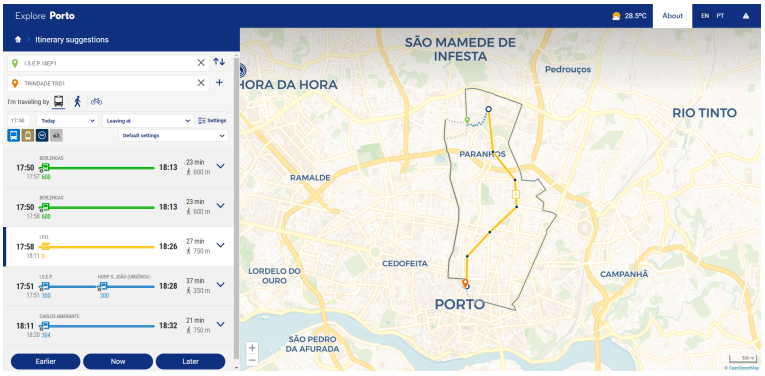


Figure 2.4: ExplorePorto Screenshot with Route Planning Options

FIWARE, MDS and GTFS are relevant technology enablers in smart cities, helping to create an open digital infrastructure, destroying vendor-locking situations, enabling cross-platform operation and creating the possibility to develop and implement various projects in all the domains of smart cities, mainly on the mobility aspect; this being the primary focus of this section.

2.3 Real-Time Public Information Systems

The three projects presented in the section 2.2, FIWARE, MDS and GTFS, are introduced as technology enablers, as they create the needed digital infrastructure to implement projects based on embedded, sensoring, data gathering, security and monitoring systems. The present section will focus on the literature review and state-of-the-art analysis of systems that directly or indirectly benefit from the digital infrastructure presented before. The main focus will be the real-time public information systems applied to public transportation and all the components, hardware and software, involved in their conception and implementation.

Real-time information systems have transformed the way passengers use public transport. The ability to access real-time data on the arrival and departure times of vehicles, delays, and alternative routes has improved the passenger experience and reduced uncertainty. The development of real-time public information systems has been a research focus in recent years as transport providers strive to optimise the transport system and reduce congestion.

2.3.1 Real-time Data

The data used in Real-time Passenger Information Systems is the result of several components that all act together to provide an *Endpoint* for developers to access it. In section 2.2 we mentioned FIWARE, Digitransit and the GTFS dataset format, all being key agents in handling real-time bus transit data in the city of Porto.

Before delving into describing the data architecture and flow for bus transit data, it is relevant to introduce OpenTripPlanner (OTP) [19]. OTP is a family of open-source software projects that provide passenger information and transportation network analysis services. They rely on data in the GTFS format and combine it with the real-time position of the vehicles, resulting in updated scheduling and delay information.

The bus fleet in our city is almost fully equipped with an onboard unit that sends its GPS location wirelessly. This information is then made available by STCP, the city's bus transit agency, through its own API using the MQTT protocol. Porto Digital stores this data in a *real-time data consumer*, using FIWARE, to be easily accessible by other services. All the transit information related to timing is then calculated using the already presented OpenTripPlanner in combination with the static GTFS dataset, resulting in updated information fully compatible with Digitransit software packages. Finally, Digitransit organises this information creating a real-time GTFS dataset accessible through a GraphQL Endpoint. Further in this dissertation, this Endpoint will be useful for the proposed system access to real-time bus arrival information, as one of the possible application scenarios. The described architecture built to handle bus transit real-time data is illustrated in figure 2.5.

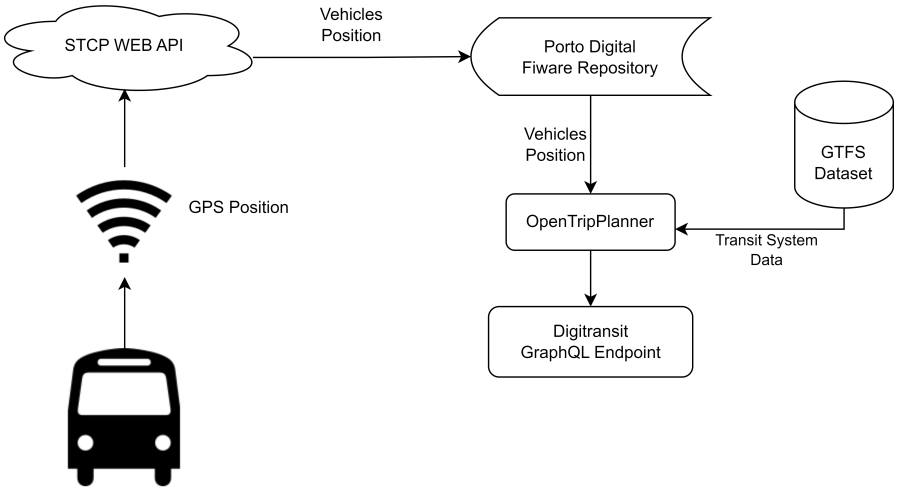


Figure 2.5: Real-time Bus Data Architecture

2.3.2 Passenger Information Systems using E-paper displays

Papercast and E-Ink are two companies that have been at the forefront of developing passenger information systems using E-paper displays. Papercast is the market leader in Real-time Passenger Information Systems (RTPIS) using e-paper displays technology [20]. They develop various products using this type of display used in different appliances, bus stops being their main market focus. Characterised as

an ultra-low power system with wireless connectivity, solar-powered and sunlight-readable. When a company holds a significant position in the market and gains immense popularity, it becomes a role model within the system, with e-paper displays playing a primary role in this regard. In figure 2.6, it's possible to see three examples of products designed and implemented by Papercast, displayed in three different configurations.

Figure 2.6: Papercast E-paper Totem Examples [21]

On the other hand, E-Ink is a company that specialises in developing E-paper technology. Their E-paper displays are widely used in e-readers and other consumer electronics and have also found applications in transportation systems. E-Ink's displays offer high contrast and low power consumption, making them ideal for outdoor displays in all lighting conditions. E-Ink's display solutions have been used in bus stop displays, train station displays, and other transportation applications.

In the article by Boshita et al. [22], the authors present a solution for a smart bus stop using a specific communication protocol, LoRaWAN, and displays with e-paper technology. The authors divide the architecture into two parts: the server side with a cloud computing architecture that manages all the BusData and sends it to the devices located in the bus stops; the device side that receives the BusData, with the help of a LoRaWAN transceiver, and generates the image to be displayed in the e-paper. This proposal opens the debate on how the system architecture should be organised, the communication protocols, and what micro-computer to choose.

Regarding PIS architecture, in the article [22], the author presents the architecture visible in figure 2.7. Using Microsoft Azure as the cloud computing server platform, the bus delay information is provided using real-time positioning using a GPS module on each vehicle. The data is then analysed, and the Estimated Time of Arrival (ETA) is calculated and distributed in an App Service or the Smart Bus Stop devices.

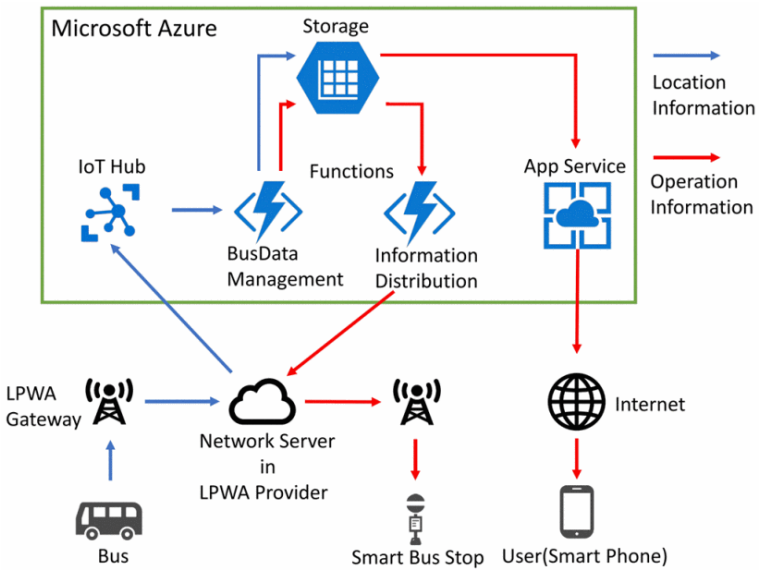


Figure 2.7: Passenger Information System Architecture Proposal [22]

This architecture proposal by Boshita et al. clears the road to the debate and analysis on how the service's system should be organised, the components of this system and the way the data should flow. It also introduces the topic of cloud computing. In the article, the proposal for the cloud computing solution is the well-known Microsoft Azure platform. However, the project presented later in this dissertation has a different context. This is justified by Porto Digital's vision and principle of doing everything *on-premise* without relying on Azure or other cloud computing service providers.

Porto Digital's infrastructure enables cloud computing on every municipality service and gives access to tools that help to build systems with advanced, adaptable and optimised architectures. The project described later in this dissertation has a system architecture designed and based on the *on-premise* flags of the organisation while also adapting and using other software tools to add needed features and capabilities.

The system architecture proposed by Boshita et al. is designed so that the device receives information about the next arrivals and then generates the image based on a template. This way, a relevant principle in IoT combined with cloud computing is lost – the objective of creating a *dumb* device with the simplest configuration possible and the closest to a *plug-and-play* type of setup. In a large-scale implementation, when connected to the network, the device should automatically receive the minimum software required for its functioning and have the minimal manual setup required. The architecture and all the system's components will be described in depth, presenting all technologies used and with software and framework description, as well as the device-embedded system-specific software.

In a study presented in Marcela Macedo's dissertation [23], the author concludes that, in the city of Porto, a Passenger Information Systems (PIS) using an e-paper display is desired by the users of the city's bus agency STCP, with almost 98% of the enquiries answering positively to the survey made by the author. The dissertation lacks expertise and knowledge on how the city of Porto's digital infrastructure is organised and who the main stakeholder is in work done in this area, Associação Porto Digital [24]. In this dissertation, the author also makes a proposal of a PIS very similar to the products developed by the industry-leading Papercast, leaving place to a vendor-lock situation and anti-open-source mindset, flags that are not in line with the European smart city standards, respected by the municipality of Porto. Nonetheless, this dissertation has high relevancy for this study as it does an organised analysis of the implications, mainly social, of implementing such types of systems.

2.4 Relevant Technologies

The project proposed in this thesis includes technologies, either hardware or software, that need to be introduced for a better understanding and comprehension of the developed work. The next subsections will cover relevant knowledge on E-paper displays and microcomputer-specific characteristics such as wireless communication and hardware interfacing protocols. Also, it will present a theoretical introduction to the topic of web frameworks as it presents a major role in the server side of the proposed system.

2.4.1 E-paper vs LCD Displays

The advancement of display technology has significantly changed how we interact with electronic devices. One of the most noticeable changes is the introduction of two different display technologies: e-paper and LCD. E-paper displays are known for their ability to mimic the appearance of ink on paper, while LCDs provide a richer and more colourful viewing experience. The choice between these technologies depends on the intended use and personal preference.

In table 2.1, we can compare e-paper and LCD technologies by analysing their features, advantages and disadvantages. This will help us better understand which technology suits specific applications. The information about dimensions and the cost of implementation is with the reference of Waveshare Displays [25] and [26].

This comparison helps to conclude that the e-paper displays, in the context of outdoor 24/7 appliances, are a more viable solution because of their ultra-low power efficiency and visibility under sunlight conditions. Their paper-like appearance also makes them less likely to be vandalised or robbed.

Table 2.1: Comparison between E-paper displays and LCDs

	E-Paper	**LCD**
Technology	Reflective	Emissive
Visibility Day/Night	Total/Low	Medium/High
Dimensions/Weight	Thin/Light	Thick/Heavy
Energy Efficiency	High	Low
View Angle	180º	Variable(centered)
Cost	300€-400€	€100-€200
Ease of Implementation	Hard	Easy

2.4.2 E-Paper Technical Analysis

E-paper displays, also known as e-ink displays or electrophoretic displays in a more technical way, have received much attention during the last two decades. Amazon Kindle is one of the most popular devices using these types of displays. Electrophoretic Displays work based on the principle of the movement of suspended charged particles in a suspension fluid under the influence of a DC electric field [27]. Depending on the signal of the electric charge, the particles, either white or black, change place and move to a visible surface. In figure 2.8, it's possible to visualise this phenomenon.

With a distance between each cell of about a micron, e-paper displays present a high contrast ratio and wide viewing angle solution. It's also a relevant feature that the particles tend to rest in the state they are left in if the display enters sleep mode or disconnects from its power source, being this feature the primary focus when the requirement is minimum power consumption. Concluding the technical analysis, the main factor in choosing an e-paper display in an outdoor bus stop is its perfect readability under sunlight and low power consumption.

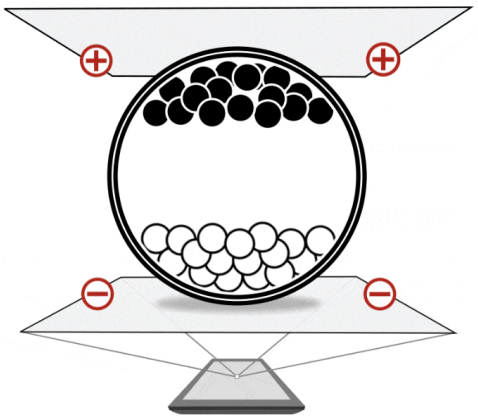


Figure 2.8: E-Ink Technology [28]

2.4.3 Microcomputer

The concept of microcomputers is not based on any particular creation. Still, it is the result of the evolution of computer systems to the degree that the size of their components can all fit on a limited-in-size printed circuit board. In a journal from 1975, the author introduces the microcomputer as the "evolutionary successor of the minicomputer, (...) a set of microelectronic *chips* serving the various computer functions." [29]. Microcomputers are computers of small proportions capable of doing the same functions as a generic computer. They have less computational power but are built specifically for each purpose they are supposed to serve, in a low-power setup and focusing the hardware on the needs of the applications.

Microcomputers differ from microcontrollers by having a CPU, GPU, and all the needed components to become a functional PC on the same chip, also known as System on a Chip (SoC). It can also support the interface for various kinds of hardware, either sensors, actuators, displays or others. The following list of interfaces is generic and present in almost every single-board computer available in the market; the choice between each is solely dependent on the hardware to be interfaced [30].

- Inter-Integrated Circuit (I2C) is a serial bus that hosts multiple master and slave devices using just two connections.
- Serial Peripheral Interface (SPI) is a serial bus capable of hosting a single master with multiple slave devices per bus and has a lower power consumption than I2C.
- Pulse-Width Modulation (PWM) is not an interface but is rather a technique to encode a message by varying the power supplied by a digital pin and is often implemented in hardware.
- Universal Asynchronous Receiver/Transmitter (UART) refers to the hardware that converts parallel to serial communications and is often simply called serial; it is the communication standards over UART to which people generally refer; the most common are RS-232 and RS-485.
- Controller Area Network (CAN) is a message-based protocol defined by ISO 11898–1 that allows multiple master device communication and was designed over the last 30 years for in-vehicle electronic networking.
- An Analog to Digital-converter (ADC) is a device that converts a continuous analog voltage to a digital number. Ensuring the analog signal is sampled at a high enough resolution is important to avoid losing analog variations.
- General-purpose input/output (GPIO) is a general-purpose pin whose behaviour can be controlled by the user at the run time. Some GPIO pins can

be configured to function as Interrupt-Request (IRQ) inputs—rising, falling or both edges of the incoming signal.

Microcomputers have become increasingly popular in developing real-time passenger information systems due to their low cost and high flexibility. These devices can display real-time passenger information to users, interfacing with a display board at a bus stop or train station, thus providing accurate and up-to-date information about their travel options.

Raspberry Pi is a great example of a microcomputer and the most popular platform in the single-board computer market. What started as a small project from a UK-based charity to create a small computer for educational purposes [31] quickly gained traction and raised a lot of attention between the hobbyists and IoT project developers. Nowadays, Raspberry Pi models can be found in all industrial sectors with various applications and use cases [32]. In its industry-standard 40-pin GPIO header, the Raspberry Pi has all the interfaces mentioned above. Nonetheless, it presents other useful hardware interfaces, as we can see in figure 2.9. The platform in the image is known as the B format. Further in this thesis, we will present a Raspberry Pi 3 with the format A+, a smaller, more compact and cheaper model compared to the B format.

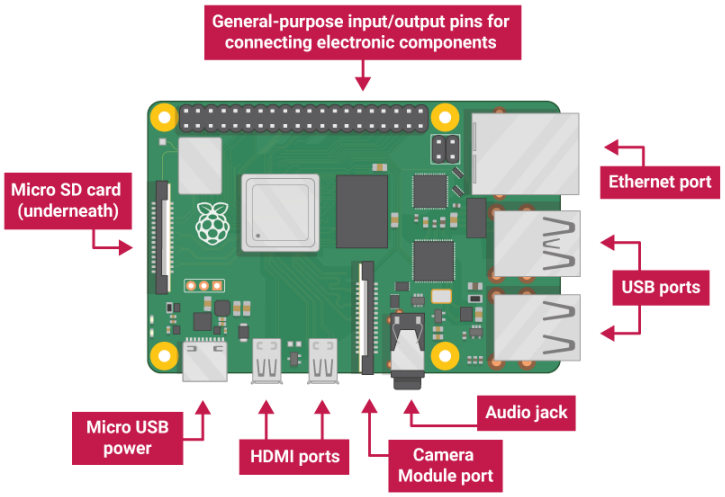


Figure 2.9: Raspberry Pi Interfaces Diagram [31]

In figure 2.10, we can see a more detailed description of the pinout of the 40-pin GPIO of the Raspberry Pi. The highlighted pins are used by the SPI bus SPI0 and SPI1, with high relevancy in this project as this is the communication protocol to interface with the e-paper displays driver-board.

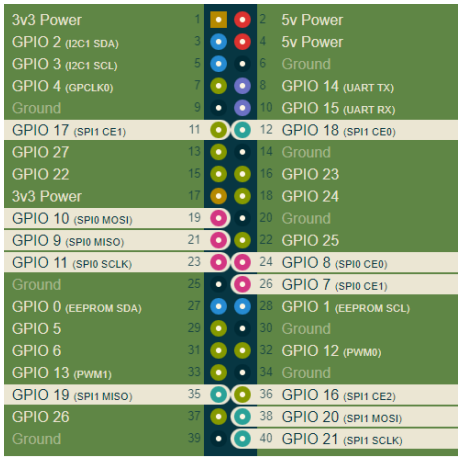


Figure 2.10: Raspberry Pi GPIO Pinout [31]

As another example of a microcomputer board but for other use cases, in figure 2.11, we can see the Developer Kit Jetson Nano by NVIDIA. It's a developer board designed for *Artificial Intelligence* appliances, so it would be considered an *overkill* option for this project scenario. Still, it has all the technical requirements above-mentioned.

Figure 2.11: NVIDIA Jetson Nano for AI [33]

Another important consideration is the availability of software and development tools for the chosen microcomputer. It is important to choose a device with a large and active developer community, as this can provide access to a wide range of software libraries and tools to support the development of the system.

Finally, the cost is important when choosing a microcomputer for a real-time passenger information system. While microcomputers are generally low-cost, the cost of sensors and other hardware components can add up quickly. It is important to evaluate the overall cost of the system when choosing a microcomputer, to ensure that it is cost-effective and within budget.

Microcomputers offer a flexible and low-cost platform for developing real-time passenger information systems. By considering key factors such as processing power,

connectivity, software availability, and cost, it is possible to choose a microcomputer that is well-suited to the needs of the system and the project budget.

2.4.4 Wireless Communication Protocols

In 2020, Internet of Things (IoT) devices connections surpassed, for the first time, the number of non-IoT devices [34]. Wireless communication protocols can make these connections; cellular, Wi-Fi or LoRaWAN protocols are the most common.

Wi-Fi [35] is the usual solution for systems that need high bandwidth or low latency and is also the best choice in places with ample coverage without the need to switch access points. Nonetheless, it presents disadvantages relevant to IoT: sensitive data shouldn't be sent freely over the internet without any VPN that needs to be configured for that specific purpose; the power consumption of typical Wi-Fi modules is an issue in low power/no power scenarios.

Cellular, either Long Term Evolution for Machines (LTE-M) and Narrowband IoT (NB-IoT) [36], both 3rd Generation Partnership Project (3GPP) standards, is the most reliable option when a 24/7 uptime is essential and is the best option where data security is critical, when global coverage is required, in low/power no/power scenarios and when the location of deployment of the device is unpredictable. The main disadvantage of cellular is when the services require high bandwidth or low latency connection.

Long Range Radio (LoRa) [37], supported by LoRa Alliance, aims to solve the classic IoT dilemma of balancing low power with long-range communication requirements. It is the best solution where low power is critical and the implementation requires long distances to the transceivers. Security-wise, it's also a good option as the communication is encrypted by default using AES CCM encryption. The disadvantages are the impossibility of implementing it outside the EU and when medium bandwidth is required (data is sent in small portions of 51-241 bytes and limited by each country's regulations).

Concluding, the three options are all possible to be used in this type of system, and the choice relies on the existing infrastructure of the city, the energy consumption concerns and the availability of Wi-Fi coverage because of its ease of implementation and availability on the majority of microcomputers.

2.4.5 SPI Communication Protocol

The SPI communication protocol, like the I2C, is one of the most used hardware embedded systems' protocols. For a better understanding of some options made in this project and described in chapter 3 it is relevant to explain the basics and working principle of SPI communication protocol. In figure 2.12, we can see a basic

diagram of how the common wiring and connections are made in an interface using this communication protocol.

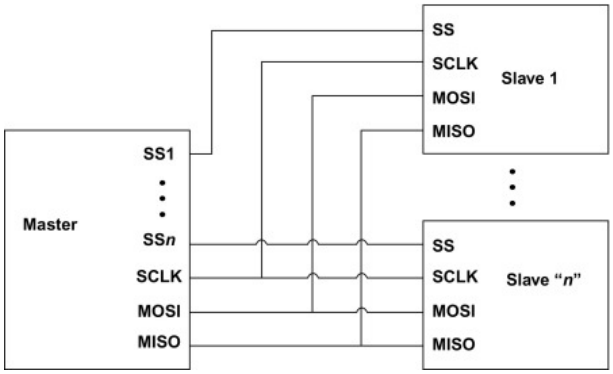


Figure 2.12: Serial Peripheral Interface (SPI) Diagram [38]

The Slave Select (SS) line, also known as Chip-Select (CS), is used by the master to select the slave it wishes to communicate. One SS line is required per slave. So, if n slaves are connected to a master in an SPI bus configuration, n SS pins are required on the master. The Serial Clock signal (SCLK), Master-Output Slave-Input (MOSI), and Master-Input Slave-Output (MISO) lines are shared between all slaves. For a bus configuration where n slaves are connected to a master, (3 + n) pins will be required on the master [38].

Each transaction begins with the master selecting a slave using the SS line, without the possibility of simultaneous communication between the master and two slaves. The master then proceeds to operate the SCLK at a frequency less than or equal to the maximum frequency supported by the slave (typical speeds are in the MHz range). In each clock cycle, the master sends 1 bit to the slave, and the slave sends 1 bit to the master. In case either the master or the slave has no data to send, they can send 0 s.

2.4.6 Web Application Frameworks

A Web Application Framework, or simply a Web Framework, represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on. Most web frameworks are used in Python and Javascript programming ecosystems, the two most popular and used programming languages for this type of application. In this project, apart from Docker-related files, all code is written in Python, justified by Python's simplicity and flexibility and to create a seamless bridge with the display's driver board software also written in Python.

Comparing Flask [39] and Django [40] as the most popular web frameworks on Python, the author in [41] analyses both frameworks and presents a rich comparison

testing them through different problems. The author concludes: "Based on the study, it is evident that Django can be the best fit for large-scale projects with the cost of the learning curve. Flask is the best fit for prototyping and small-scale projects, but not limited to it. Flask can be learned and set up quickly, but when it comes to managing and maintaining, it requires more work than the former.". This project, from a web application point-of-view, doesn't require a framework capable of large-scale implementations as it should handle only one type of request and consecutively answer them with an image, concluding that Flask is the ideal option for this API.

2.4.7 Nginx Reverse-Proxy

To better comprehend the presence of this type of component in a cloud computing infrastructure, it's relevant to compare it to a proxy server. A proxy server is a server application that acts as an intermediary between a client requesting a resource and the servers on the internet providing that resource. A reverse proxy server is a kind of proxy server that is usually positioned behind the firewall in a private network. Its purpose is to intercept traffic that requests access to resources on backend servers in a private network. By adding an extra level of abstraction and control, a reverse proxy ensures the seamless flow of network traffic between clients and private servers [42].

This important component is seen in various applications; in [43], the authors propose an industrial application of reverse proxy to manage the access to the analytics services inside an enterprise in the industrial sector, adding a layer of security, certification and hardware registry to their systems. In reverse proxy configurations, Nginx [44] is the most popular option, as it is open-source and widely adaptable for various applications and customer needs. The authors in [45] present a description of all the technologies related to the topic of reverse proxy and a relevant analysis of the algorithms of load balancing used by Nginx, the main feature of this technology and the most relevant in a large-scale implementation.

2.4.8 Docker Containers and Docker-Compose

With cloud computing exponentially growing and being present in most IT companies, operating system virtualisation also keeps up with this growth, with Docker [46] as the main agent in this technology.

Firstly, it's important to understand the principles of OS virtualisation relevant to this project, introducing concepts such as containers, the steps to create a container and the docker-compose tool.

In a cloud computing infrastructure with various applications, it is important to organise all the components considering their dependencies and interactions. A

full application service should have all its components in the same virtual machine. Still, all these components should also be containerised for easy development, build and deployment, using APIs to help with their interaction. A container is a portable software unit with all the necessary code and dependencies to run an application. It packages everything together, such as runtime, tools, libraries, and settings, enabling reliable and efficient execution across different computing environments. Containers operate at the app layer, allowing multiple isolated processes to share the same OS kernel. Unlike Virtual Machines, containers are smaller, more efficient, and can run more applications using fewer resources [47].

A Compose file, executed with the Docker-Compose tool, is a YAML file defining services, networks, and volumes for a Docker application. When building services that depend on each other and will require network communication, the Compose file enables a fast and automated start procedure for all the components required.

In the Compose file, each service has its image defined or a build point where the Dockerfile to create the image is located. Also, it includes the definition of a name for the correspondent container and the network port mappings. This port mapping refers to the pair of the Virtual Machine's port and the Docker Engine Container's port. Additionally, to prevent errors on startup, it's a good practice to ensure that the mandatory containers start successfully, and only after that can we run the other containers; this procedure is specified in the *depends_on* parameter on the less important services configurations [48].

2.5 Discussion

This literature review helps present the base knowledge needed to comprehend the project and study of this dissertation, focusing on the topics of smart cities, mobility and real-time passenger information systems. This review organisation intends to direct the reader through an "inverted pyramid path" relative to how comprehensive each topic is. The review starts with a wider and more general explanation in section 2.1. In this section, we should ask ourselves, "What is a Smart City?" and "What are the main sectors of activity in a city?". European Commission [7] and Deloitte's [9] visions on answering both questions should be highlighted for their relevancy and objective approach.

The review then descends, focusing on the mobility sector, and answers the question, "What does a city need to be smart in the mobility sector?" introducing technology enablers and relevant projects. It's relevant to highlight the definition of Mobility as a Service as clarified in [1] and analyse all the key players on the public transport system as seen in figure 2.1. Also, in section 2.2, FIWARE and MDS are presented as two relevant technology enablers for projects in smart cities because of their main goal of disrupting vendor-locking actions and giving way for

cooperation between the various stakeholders of technology projects implemented in smart cities. This starts to centralise the attention on the main area of this project when implemented in bus arrivals displaying scenarios.

At the end of the pyramid, section 2.3 presents the state-of-the-art of Passenger Information Systems, including the analysis and study of similar systems by the industry leader PaperCast and other academic projects. It's also relevant to highlight the work done by two authors on how they contributed to the development and comparisons to the project proposed in this thesis. The system architecture and hardware choices seen in [22] helped us acknowledge the general requirements of a total system architecture and a possible solution to develop the interaction between the devices and the server. Additionally, the thesis by Marcela Macedo [23] is relevant in evaluating and analysing the implementation impact on the usefulness and social aspects. This section lacks scientific extension as the development of this type of system isn't popular in the academic and investigation community. All the major work done in this area presents a major vendor lock by Papercast, being the market leader and the only viable solution at present.

Finally, the technologies introduced in section 2.4 provide the reader with a useful theoretical introduction. This section had the intention of facilitating the analysis of the development that will be described in chapter 3. Divided into hardware and software topics, the reader should now be technically aware of all the key topics mentioned in the remainder of this dissertation.

Chapter 3

Project Development and Implementation

In this chapter, we present the work done in the proposed project, describing the decision-making process behind each option made to get to the final functional prototype. After introducing the project requirements and motivations, this chapter will be divided into distinct sections where we will divide the description of the system between the work done in the device, its possible configurations and the server side with its whole system architecture.

The E-paper display analysis and the microcomputer requirements will be the main focus of section 3.2. The communication between the displays and the microcomputer has a lot of development that needs to be clearly described and explained, justifying the need to adapt the software to drive the displays and the design and production of a Printed Circuit Board (PCB). The thought process and implementation of an architecture where the device is as simple and *plug-and-play* as possible will also be discussed further in this chapter. The software development for driving the displays will be described in section 3.3 followed by the server architecture and components.

In both sections, there will be a theoretical introduction and discussion phase justifying the options for each component, followed by a description of all the development and work done in the various areas of the system.

3.1 Project Architecture Introduction

As presented before, this project aims to create a functional prototype of an E-Paper display connected to a microcomputer and to develop the software system to provide the images displayed in the devices. Without entering into technical descriptions and analysis, figure 3.1 shows the high-level architecture of the system divided into two distinctive parts, the Device and the Server. Also, it is relevant to note that the GraphQL Endpoint illustrated in this architecture is, in fact, the same Endpoint at the end of the diagram of figure 2.5, used for real-time bus arrivals data retrieval.

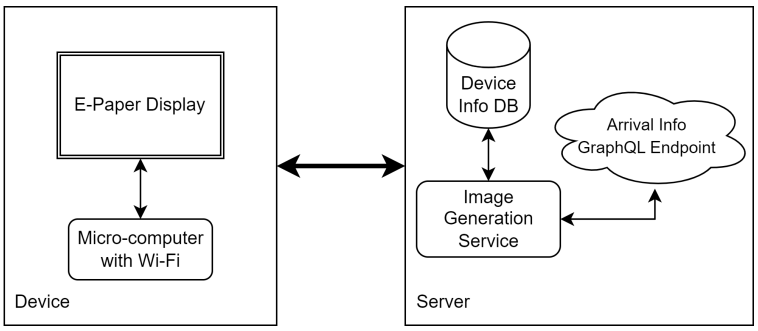


Figure 3.1: High-level System Architecture

Observing this architecture, the objectives and requirements of this implementation become clear, and in the next sections, we will describe the work done to meet them: micro-computer preparation; communication between the micro-computer and the display; communication between the device and the server; server architecture; API development and database creation.

3.2 Hardware Analysis and Description

Every embedded systems project is divided into three distinctive phases: state-of-the-art study, market analysis and general requirements definition; components choice and high-level system architecture design; development, testing and implementation.

In the next subsections, we will present the thought process behind each option made and analyse the methods of its implementation, dividing this explanation into each hardware component, the display and the microcomputer, while granting the system complies with the defined objectives and requirements mentioned in subsection 1.3.

3.2.1 E-Paper Display

Over this subsection, the used approach will explain why the selected display is the best choice for this specific project and the working principle behind it and its driver board.

Market Analysis and Project Option

After analysing the technology behind e-paper displays presented in section 2.4, and being this type of display a formal requirement of the final prototype, we had to analyse the market for the best option. Table 3.1 presents the resume of the market analysis.

Table 3.1: E-paper Displays Options Analysis

	Driver Board	**Price(€)**	**Refresh Time**
Waveshare 13.3inch	✓	410	<1s
Waveshare 9.7inch	✓	222	<1s
Waveshare 7.5inch	✓	55	∼5s
Shineworld Innov. 12inch	X	∼200	?
Zhuhay Suny 13.3inch	X	∼300	?

As mentioned before, the project described in this Master's thesis aims to develop and present a functional prototype, so the cost and the availability of the components are a priority when analysing the market. Considering this, Waveshare presents itself as the only company with e-paper kits (display + driver board), fast delivery times, and fair prices for purchases of small quantities. The other two options resulted from a search on international markets for a better understanding of the availability and cost of importing large quantities of these types of products conducted by the author of this dissertation. They resulted in the conclusion that the display sold without a driver board could be less expensive than other kit suppliers but would raise the need to create a complex individual driver board, an effort not justified when developing a functional prototype in a company without sufficient resources to produce such type of components.

Focusing on Waveshare solutions, it is clear that the increment of price is not linear between the three options but is justified by the clear difference in refresh time between the 7.5-inch option and the other two. The refresh time technical aspect and the cost of each solution raise the idea of implementing different configurations. These configurations should be divided into a standard and large-scale implementation, with two to four 7.5-inch display setups, and another premium and feature reach implementation with one or two 9.7-inch displays or one 13-inch display. The

options that resulted from this market analysis and the respective driver boards are demonstrated in Appendix A figure A.1.

Driver Board and Working Principle

The driver board is mandatory for the SPI communication between the display and the microcomputer. It aims to convert the image data received via SPI to a *waveform*. A *waveform* in eink displays is a predefined voltage impulse sequence the controller uses to manipulate the particles mentioned before. The voltage sequence and temperature range are stored in a *lookup table*. They are precisely paired to the display by production lots to enable precise placement of pigment to achieve an accurate grey level.

The driver board of the 9.7-inch display is more complex than the one present in the other display; the distinctive working principle and capabilities of both displays justify that difference. The 7.5-inch display is only capable of full-update routines with some black and white flickering between the content refresh, which takes approximately five seconds to complete. The 9.7-inch display has fast-update and partial-update capabilities because of the IT8951 micro-controller in the driver board, an industry standard for driving E-paper displays. It has an internal cache memory and works so that every time a new image is received, it updates a buffer comparing the changes between the actual and new content and sends only the needed updates to the display.

The richer features of the 9.7-inch displays and its fast update routine raise no constraints on its implementation, with only its drivers being more complex. When implementing a solution with 7.5-inch displays, we must consider that the image update will take five seconds to finish, resulting in the need to make the update of multiple displays simultaneous so that it doesn't look unprofessional to the general public.

3.2.2 Microcomputer Choice

The microcomputer, or Single-Board Computer, is the main component in every IoT device. Depending on the requirements of the system and project limitations, there are various options in the market to meet the defined objectives.

For context, in the past three years, the market for microcomputers and other technologies has seen a lot of difficulties with the lack of semiconductors due to the 2020 crisis of this component [49].

Over this subsection, the author will describe the requirements for this component, the market availability to meet them and the limitations encountered during the development of the device prototype.

The project proposed in this dissertation has a clear definition of the minimum interface requirements needed in the microcomputer used. The wireless connectivity available at almost every bus stop defines the device's requirement for a Wi-Fi transceiver. Additionally, the E-paper display's driver board and possible configurations require two SPI buses to enable the synchronised refresh of multiple displays.

After analysing the market for these requirements considering the solution's total cost and market availability to answer a large-scale implementation, table 3.2 shows the results of this analysis with the most appropriate options considering the requirements defined.

Table 3.2: Microcomputer Options Analysis

	Stock	**Price(€)**	**2 SPI Buses**
Raspberry Pi 4 B	X	70	✓
Raspberry Pi 3 A	✓	32	✓
Odroid-C4	✓	55	X
A64-OLinuXino	✓	65	X
Orange Pi Zero 2	✓	30	X

Analysing the pros and cons of each option in this table, the three microcomputers that don't meet the requirement of having two SPI buses available aren't viable solutions for this project. These three microcomputers share a common characteristic in most devices of this kind; the existence of SPI flash memory in this board occupies one of two SPI buses available for the hardware interface. This isn't an issue on Raspberry Pi boards as they don't have onboard storage, only raising the disadvantage of requiring an additional SD card.

Resting only two possible options, the Raspberry Pi 3 A complies with all the defined requirements, is highly available in the market, and its price meets the budget limits of the overall system. In image 3.2, we see the model chosen for this project; being available in the company for practical implementation and prototyping.

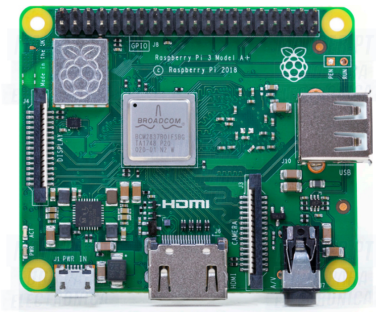


Figure 3.2: Raspberry Pi 3 A+ [50]

3.2.3 Microcomputer-Display Interface

The needed knowledge to comprehend the functioning of the display driver-board and SPI communication protocol was already introduced in sections 2.4 and 3.2.1, coming together to cooperate and establish the interface between the microcomputer and E-paper displays. This interface results from a thought process where we had to meet the requirements of each device configuration while keeping the connections simple, and software drivers optimised and adaptable. This interface's hardware definition and analysis will be described in this sub-section, while the software development will be treated in section 3.3.

In sub-section 3.2.1, the idea of different configurations of the final device raised attention to the possibility of interfacing multiple displays with only one microcomputer. To meet this requirement and to minimise the wiring needed to connect two displays and respective driver boards, we had to define the best solution for each configuration.

Both driver boards, relative to 7.5-inch and 9.7-inch displays, are designed to be mounted on the Raspberry, also known as hardware attached on top (HAT) design, abling the interface to only one display per microcomputer. With configurations using multiple displays, we can increase screen space, consecutively resulting in bigger text size and better readability.

Both hardware and software must adapt to a configuration using more than a single display. In section 3.3, we will describe the changes made to the display drivers to solve the adaptability issue when using multiple SPI buses. Acknowledging the working principle description of the e-paper displays and using the knowledge introduced in the SPI short presentation, the technical limitation of five second refresh time of the 7.5-inch solution raises the need to implement a hardware solution that enables simultaneous refresh of the displays, thus requiring two separate SPI buses, instead of using one SPI bus and two SS lines for each device.

The hardware solution proposed, designed and produced is a PCB that ables the connection of two E-paper displays using the same connector as the one present in the vendor's driver board, a JST PH 2.0 connector and a 40-pin female header compatible with the Raspberry Pi GPIO header. This PCB was designed using Kicad open-source Electronics Design Automation Suite [51]. To conclude this description, the objective of this PCB is to decrease the wiring of the final hardware setup and simplify the connection of two displays to the same microcomputer, only requiring the cable compatible with the specified connector.

3.3 Software Description

One of the most important principles in large-scale IoT projects is the ease of setup and initial configuration required, ensuring that the device is fully functional and capable of performing the required tasks with minimal manual intervention. This approach is crucial to streamline the deployment process and ensure the project's smooth operation. To adhere to this mindset, the software developed for such projects should strive to have the fewest dependencies possible while fully optimising for quick installation and setup. In this specific case, there are only two key components, the display driver software package responsible for the communication between the E-paper and the microcomputer and a small script that makes an HTTP request to the server to receive the image to be displayed.

Before delving into the specifics of the device software, it is worth mentioning that these components are implemented within a larger system based on a cloud service. This cloud service is the project's backbone, generating and providing the requested images for each smart bus stop. Through seamless integration with the cloud service, the IoT devices can stay up-to-date with the latest information, ensuring that the displayed content remains relevant and useful to the passengers.

To provide a comprehensive understanding of the overall system architecture from a software perspective, the present section will be divided into two components and include a high-level system architecture analysis. This architectural overview will individually delve into the device and server-side software, highlighting their functionalities and interactions. By breaking down the system into manageable subsections, we can gain a deeper insight into how each software component contributes to the overall functionality and performance of the IoT project.

3.3.1 High-Level System Architecture

As this system is built on top of the already integrated company's cloud-computing infrastructure, some system architecture elements are already pre-defined and can't be an object of options analysis. Nonetheless, in subsection 3.3.3, with the knowledge previously introduced in section 2.4, the work done in all the components used in this cloud-computing system will be rigorously described.

Defining the microcomputer as the starting point, in figure 3.3, we can see the full high-level system architecture with most technologies specified and with the numeric reference on how the data flows through the complete process.

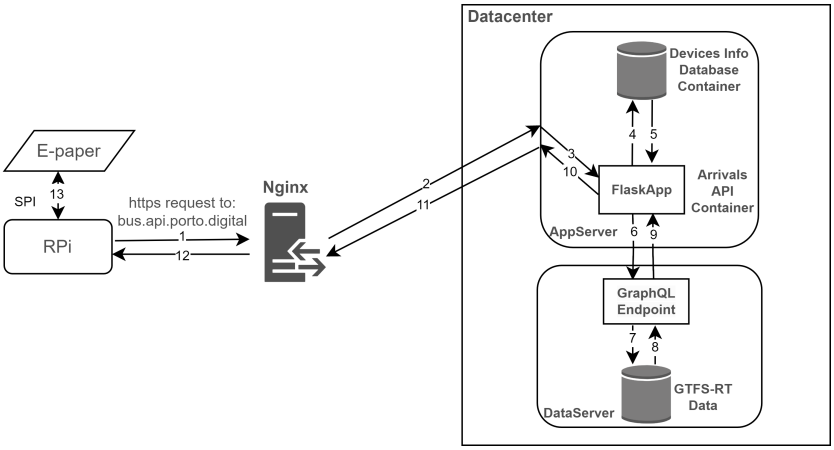


Figure 3.3: Implemented System Architecture and Flow

Before entering the distinctive description of the device and server software, it is also relevant to translate the diagram seen in figure 3.3 using a Message Sequence Chart (MSC) and a flowchart for a better understanding of how the flow of data occurs and the algorithm behind the whole system. Accordingly, diagrams in figures 3.4 and 3.5 describe the algorithm behind the proposed solution and how and when the data requests and queries occur.

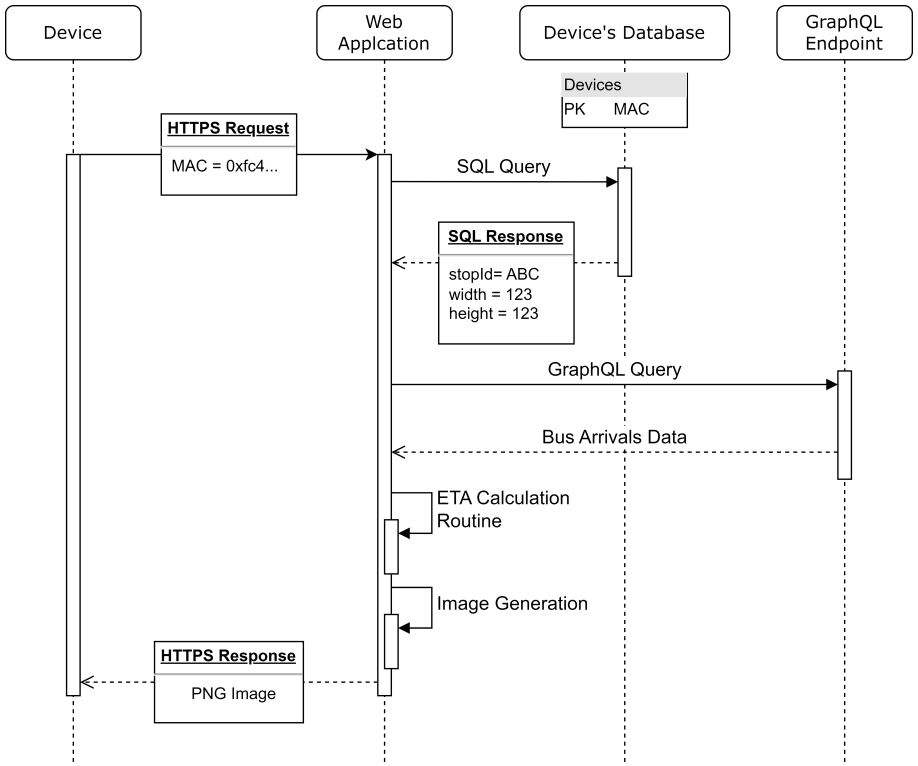


Figure 3.4: Software Message Sequence Chart

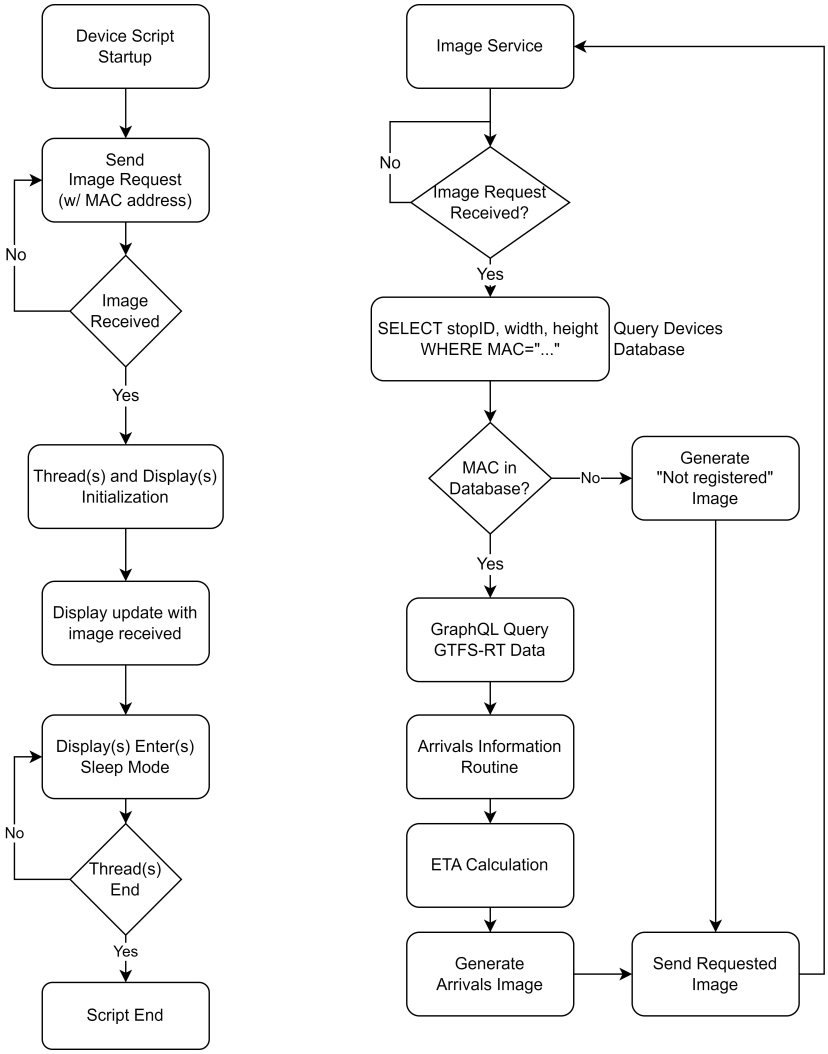


Figure 3.5: Total System Flowchart

On the device part of this system, the algorithm consists of a simple script, started with a cronjob previously installed in the OS, responsible for making an HTTP request, sending the device MAC address as an argument, and reacting to the response accordingly. After a successful reception of an image, the threads needed, equal to the number of displays implemented, are initialised, and the respective image is sent as an argument. The display proceeds to update and enter sleep mode. The cronjob automatically executes the program every minute. Additionally, to prevent display damage, it is advised to do a full clean refresh of the display every 24h. Hence, another cronjob is also responsible for executing this function daily at midnight.

On the server side of the system, also referred to as "Image Service", the web application waits for requests, and when one is received, it checks if a MAC address was sent as an argument. It then queries a database with all the devices registered

using the MAC address to retrieve the needed information. If the device is registered, the application will query a GraphQL Endpoint to receive the next bus arrivals for that stop, analyse the data, calculate the ETA for each bus, and generate an arrivals table image. If the device is not registered, the application generates a default image with a message and indication to register the device. After having the required image, it is sent back to the device that made the request.

3.3.2 Device Software

In the context of a large-scale implementation of an IoT project, the work done on the device, from a software point-of-view, plays a pivotal role in ensuring compliance with the requirements and objectives. To effectively achieve a simple and *plug-and-play* configuration for each device, creating an Operating System (OS) image that is fully configured and includes all the necessary applications and dependencies is crucial. This *plug-and-play* approach streamlines the deployment process, ensuring the devices run quickly and efficiently.

Moreover, this section will encompass a detailed analysis and description of the driving software and the main program development. These components are essential for the overall functionality of the device. The driving software, in particular, facilitates communication between the microcomputer and the display. It ensures smooth and efficient data transfer using the SPI buses. Finally, the main program function requests the needed image and updates the e-paper display.

OS Image Creation

The decision to create a custom OS image stems from several factors. Firstly, the chosen image Raspberry OS Lite, the official operating system provided by the Raspberry Pi Foundation, offers a lightweight and flexible foundation for this project. However, it lacks the software and configurations for our specific use case. We can address these limitations by developing a custom OS image and tailoring the operating system to our precise needs.

Secondly, a custom OS image allows us to optimise the performance of the Raspberry Pi by removing unnecessary software components and packages that are irrelevant to our project. By streamlining the operating system, we can allocate more system resources to the essential functionalities, thereby enhancing the overall performance and responsiveness of the Raspberry Pi.

Furthermore, creating a custom OS image provides the opportunity to simplify the initial setup process of the device. By pre-configuring and including specific software, drivers, and settings in the custom image, we can achieve the *plug-and-play* objective mentioned before. By eliminating the need for manual installation

and configuration of individual components, we aim to enhance the optimisation of the device and ensure a smoother and more straightforward setup process.

The first phase of creating the custom OS image is to flash the Raspberry Pi OS Lite in a device and manually configure it with all the software and configurations needed. The list of mandatory steps to install and configure all requirements and dependencies is as follows:

1. WLAN Configuration for automatic Wi-Fi connection on boot
2. Change Raspberry Pi GPIO configurations to enable two SPI busses operation
3. Install BCM2835 GPIO access libraries
4. Pull Git repository with Display Drivers and Main Program into a standard directory (Ex: /opt)
5. Run the setup.py script inside the downloaded repository to install all program's dependencies
6. Create the Cronjobs responsible for scheduling the program's start and e-paper daily cleaning

Upon accomplishing all the requisite procedures, the SD card used on the device should then be cloned, thereby generating an OS image file. This image file constitutes a perfect duplicate of the complete filesystem, encompassing both utilised and unutilised space, necessitating the implementation of an image reduction technique. To undertake this operation, we leveraged a bash script called PiShrink [52], which analyses the filesystem residing within the image file and eliminates any free space, yielding an image solely composed of the utilised space.

The final step involves flashing the resulting image file. Flashing essentially means transferring or writing the image file onto another storage device. By doing so, we are duplicating the entire filesystem from the original device onto a new medium. This allows the cloned image file to be used independently, either as a replacement for the original device or for data recovery. Once the image file is successfully flashed onto a suitable SD card, it is ready to be used as the boot device of a Raspberry Pi, fully prepared and functional without any manual startup.

E-paper Display Driving Software

The driving software of both display types exhibits similar characteristics, with the primary distinction in the sequence of commands transmitted to the respective devices before communication, as well as the image refreshment procedure. As elaborated in subsection 3.2.1, the refresh process in the 9.7-inch display's driver

board involves a comparative analysis of disparities between the displayed image and the new one, followed by a selective update of the display.

To illustrate the configuration and communication process of the devices, we will employ the 7.5-inch display driving software as a demonstrative example.

The driving software is divided into the initial configuration phase and interface functions. The initialisation requires sending the SPI Bus number, chip-select pin and the E-paper module-specific pins as an argument, represented in the following step list:

1. Create EPD Object sending as argument: SPI_BUS, CS and EPD specific Pins
2. EPD Configuration routine using the previously specified pins
3. GPIO initialisation
4. SPI Bus Initialisation

As it was introduced in the multiple display configuration discussion, the driving software wasn't originally able to drive multiple displays, thus not accepting the arguments mentioned above. The changes made and implemented are highlighted with the colour green in the two block codes 3.1 and 3.2. With this adaptation, we enable the rest of the communication functions to use the SPI bus and display specific pins relative to the initialised "EPD" object. The original software is accessible via this link[1].

```
1
2 class EPD:
3     def __init__(self, SPI_BUS, CS_PIN, BUSY_PIN, RST_PIN, DC_PIN):
4         self.reset_pin = RST_PIN
5         self.dc_pin = DC_PIN
6         self.busy_pin = BUSY_PIN
7         self.cs_pin = CS_PIN
8         self.spi_bus = SPI_BUS
9         self.width = EPD_WIDTH
10        self.height = EPD_HEIGHT
11        self.rasp = epdconfig.RaspberryPi(SPI_BUS,
              CS_PIN, BUSY_PIN, RST_PIN, DC_PIN)
```

Listing 3.1: EPD Object Init Function With Modifications Higlighted

[1] https://github.com/waveshareteam/e-Paper

```
1
2 class RaspberryPi:
3     def __init__(self, SPI_BUS, CS_PIN, BUSY_PIN, RST_PIN,DC_PIN):
4         import spidev
5         import RPi.GPIO
6         self.GPIO = RPi.GPIO
7         self.SPI = spidev.SpiDev()
8         #GPIO pins
9         self.CS_PIN = CS_PIN
10        self.BUSY_PIN = BUSY_PIN
11        self.RST_PIN = RST_PIN
12        self.DC_PIN = DC_PIN
13
14        #GPIO init
15        self.GPIO.setmode(self.GPIO.BCM)
16        self.GPIO.setwarnings(False)
17        self.GPIO.setup(self.RST_PIN, self.GPIO.OUT)
18        self.GPIO.setup(self.DC_PIN, self.GPIO.OUT)
19        self.GPIO.setup(self.CS_PIN, self.GPIO.OUT)
20        self.GPIO.setup(self.BUSY_PIN, self.GPIO.IN)
21        #spi init
22        self.SPI_BUS = SPI_BUS
23        self.SPI.open(self.SPI_BUS, 0)
24        self.SPI.max_speed_hz = 4000000
25        self.SPI.mode = 0b00
```

Listing 3.2: EPD Config Init Function With Modifications Higlighted

Upon successfully initialising the e-paper object, adhering to a predetermined sequence of steps for communication and image updates on the display becomes imperative. These functions are illustrated in the flowchart in figure B.1. From a software point-of-view and analysing the driver software package, we designed the flowchart seen in figure 3.6 so it is easier to understand how the driving software is organised. Additionally, all the blocks in the flowchart have their functions represented in the code listing C.1.

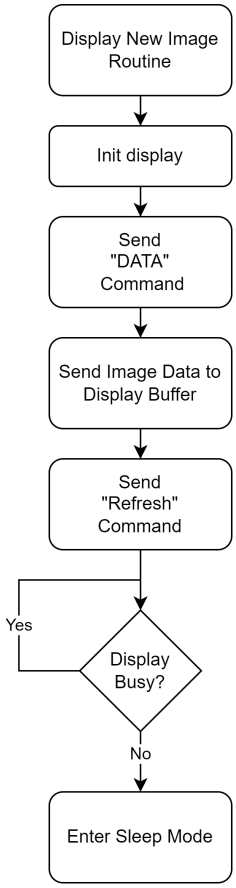


Figure 3.6: EPD Update Specific Routine

All the GPIO and SPI specific initialisation and communication functions are part of both Python modules *RPi.GPIO* [53] and *spidev* [54].

Main Program

7.5-inch and 9.7-inch displays share the same algorithm behind the architecture of the main program. In the device side of the flowchart seen in figure 3.5, we can understand that the main device's script is simple and requires no advanced computational power.

The program starts with an HTTP request using the device's MAC address and conversion to an image object of the response that resulted from it. The image is then processed to be divided in half and rotated to be later sent to the respective e-paper display. The program enters a routine of thread creation, starting and waiting for their conclusion. Threads enable the device to communicate and refresh both displays simultaneously when using separate SPI buses. The code regarding this program is detailed in Appendix C section C.1.

3.3.3 Server Software

The server software is a critical component of the cloud computing infrastructure used in this project. It forms the backbone of the entire system, facilitating communication, data storage and interaction between different components. The server software stack includes several key elements, such as Nginx, a high-performance reverse proxy server; a Flask WebApp, a small and versatile web framework for responding to received requests; MariaDB database for efficient data management of equipment information; and a GraphQL endpoint for flexible and efficient querying of bus arrival data. Also relevant to mention that both the WebApp and Database are encapsulated within Docker containers using docker-compose.

In this chapter, we will delve into the specifics of each server software component, exploring their roles, functionalities, and interactions within the system. We will discuss the architecture usefulness of Nginx, which acts as a frontend web server, handling incoming requests and distributing them to the appropriate backend components. The Flask web framework's server application will be examined in detail, highlighting its capabilities in building RESTful APIs and serving dynamic arrivals images. Furthermore, we will explore the MariaDB database, focusing on its role in efficiently storing and retrieving the registered device's data while ensuring data integrity and security. By understanding the intricacies of each server software component, we can gain a comprehensive understanding of the entire cloud-computing infrastructure and its potential for supporting the large-scale implementation of E-paper display devices. The whole system architecture and the components abovementioned can be seen in Figure 3.3

Nginx Reverse Proxy

Following the natural flow of data in this system, the Nginx reverse proxy is the first agent to influence the path of information coming from the devices and the last before arriving at them.

After registering the domain "bus.api.porto.digital" in the company's Domain Name System (DNS), the Nginx server configurations must be modified to redirect all requests received using that domain to the machine with the API responsible for handling the requests. This configuration is made by mapping the domain to a machine IP address and to which port in that IP it is supposed to redirect the HTTP requests. This port is chosen when configuring the docker container containing the API. It is impossible to show the real configuration file used on this project for security purposes as it displays sensitive IP addresses.

Device's Information Database

As presented before, this database stores the device's information, such as their MAC address, bus stop identification code, and the image resolution to be generated.

Presented as an evolution and replacement of MySQL with more functionalities, resulting in better performance and security, MariaDB is one of the most popular choices when choosing a RDBMS. In a report by the company that owns MariaDB, SkySQL, they present it as a solution with "extra features, performance improvements, better testing, and fewer bugs" [55]. In Porto Digital, the RDBMS used in most systems is also MariaDB, concluding this is the best option for this project.

The MAC address is a unique identification method of the device, so it is set as the table's primary key, meaning there can't be two rows using the same MAC. In figure 3.7, it is possible to observe the table created with two devices already registered. The first is in a 9.7-inch display configuration, and the second is in a two 7.5-inch display configuration.

	MAC	stopID	width	height
1	0x28cdc1028a1c	1AL2	1,200	825
2	0xb827eb4dd36c	TRD1	960	800

Figure 3.7: Device's Database Table

In our application, to retrieve the "stop_id", and the "width" and "height" to generate the image of a specific device, the needed query is as follows:

```
1 USE devices_db;
2 SELECT stopID, width, height FROM devices WHERE MAC='{MyMAC}}';
```

Listing 3.3: Example SQL Query

Docker Containers and Docker-Compose

The container's image used to host the web application in this system is defined with the Dockerfile detailed in code listing 3.4. The Dockerfile consists of instructions readable by the Docker Engine to build the images used by the containers.

```
1  FROM python:3.9
2
3  WORKDIR /dir
4
5  RUN mkdir ./resources
6  RUN mkdir ./src
7
8  RUN apt install --upgrade libmariadb3 libmariadb-dev -q -y
9
10 COPY requirements.txt ./
11 COPY /src ./src
12 COPY /resources ./resources
13
14 ENV FLASK_APP=./src/pngGenHttp.py
15 ENV FLASK_RUN_HOST=0.0.0.0
16 ENV TZ Europe/Lisbon
17
18 RUN pip install --upgrade pip
19
20 RUN pip install -r requirements.txt
21
22 EXPOSE 5000
23
24 CMD [ "flask", "run"]
```

Listing 3.4: Web App Dockerfile

The "FROM" instruction specifies the Parent Image to base this system. The "WORKDIR" instruction sets the working directory for any "RUN", "CMD", "ENTRYPOINT", "COPY", and "ADD" instructions that follow it in the Dockerfile [56].

After defining the WORKDIR, we create two directories and install useful software dependencies for communicating with the database. The contents of the local directory are then copied to their corresponding locations on the image. The "ENV" command ables the user to specify environment variables relevant to the project and application runtime. Finally, after installing the required dependencies listed in the "requirements.txt" file, the port for requests is "EXPOSED", and the application is started using the command in the last line.

In code listing 3.5, we can see the Compose file used in this system, with two *services*: "db", the database based on a MariaDB docker image, and "web", the web application based on the image that results from the Dockerfile described before; and the volume where the data of the database is stored locally to ensure its integrity.

```
version: "3"
volumes:
  data:
services:
  db:
    image: mariadb
    container_name: devices_mariadb
    environment:
      MYSQL_ROOT_PASSWORD: ***
      MYSQL_DATABASE: devices_db
      MYSQL_USER: ***
      MYSQL_PASSWORD: ***
    volumes:
      - data:/var/lib/mysql
    ports:
      - "3306:3306"
    restart: always
  web:
    container_name: image_api
    build: .
    ports:
      - "5000:5000"
    restart: always
    depends_on:
      - "db"
```

Listing 3.5: Docker-compose file

Finally, with the command *"docker-compose up –build -d"*, we start the *services* defined in the Compose file, building the image ("–build" argument) of the web application and prompt it to be run in system's background ("-d" argument).

Web Application

The main agent on the server side of this system is the Web Application, also mentioned as an API or web service. In this project, we propose an application responsible for handling requests by IoT devices. Consequently, the application should generate a response using information from a database and a GraphQL Endpoint, thus granting these three components seamless integration and interaction.

It is possible to analyse and comprehend the architecture of this application while acknowledging the request and response procedures with the aid of the diagram in figure 3.8. As it was described in sub-section 3.3.2, the device executes an HTTP request to a pre-defined URL, *bus.api.porto.digital/arrivalsX*, and the web application using the *app.route()* function handles the request and the arguments that come with it. Note that the "X" in this image can be either "75" or "97", depending on

which type of device is making the request, simplifying the default response in case it isn't registered in the database yet.

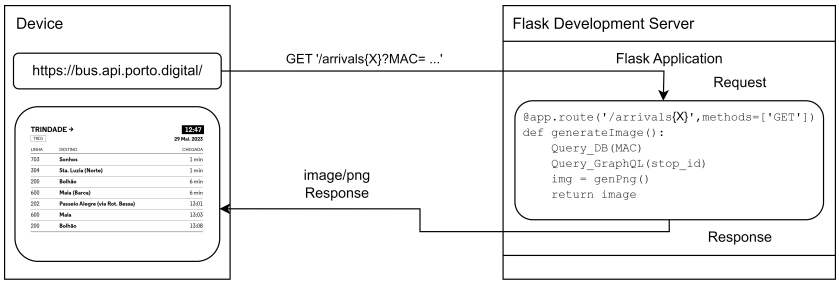


Figure 3.8: API Architecture Overview

The flowchart in figure B.2 details the full application algorithm behind the architecture demonstrated in figure 3.8.

The code listing 3.6 is the starting point whenever a proper request is received, immediately extracting the MAC address from the request arguments. After having the device's MAC address, the program opens a connection with the database using the MariaDB connector module [57] and queries the database with the respective query string.

```
1  @app.route('/arrivals75', methods=['GET'])
2  def generateImage75():
3      try:
4        myMAC = request.args['MAC']
5        conn = connectDB()
6        cur = conn.cursor()
7        cur.execute(f"SELECT stopID, width, height FROM devices_db.
             devices WHERE MAC='{myMAC}';")
8        params = cur.fetchone()
9        conn.close()
10       stop_id = "\"2:" + params[0] + "\""
11       arrivals = query_execute(stop_id, client)
12       return genPng(arrivals, stop_id, int(params[1]), int(params
             [2]))
13     except:
14       return genPngMAC(myMAC, 960, 800)
```

Listing 3.6: Main Function to handle Image Requests

If the device is registered in the database, the identification code of the bus stop where the device is located, "stop_id", is extracted, and the program executes the GraphQL query, demonstrated in the code listing C.4. This GraphQL query, executed with the python module *gql* [58], receives as a response all the information needed for generating an arrivals list:

- name - Name of the Bus Stop
- code - Code name of the Bus Stop
- arrival Information:
  - headsign - Destination of the Bus
  - realtimeState - If the arrival information is in real-time or scheduled
  - realtimeArrival - Time passed, in seconds, since 00:00 for real-time bus arrival
  - scheduledArrival - Time passed, in seconds, since 00:00 for scheduled bus arrival
  - routeShortName - Bus Code (usually a number)
  - serviceDay - Date in Unix Timestamp format when this bus service started

All this information is then grouped, creating an array of "Arrival_info" objects with all the information gathered for each arrival in the above query. In the moment of initialisation of each arrival "object", a script function is executed to calculate the Estimated Time of Arrival (ETA) for each bus arrival. The respective flowchart for better comprehension of its algorithm is described in figure 3.9,

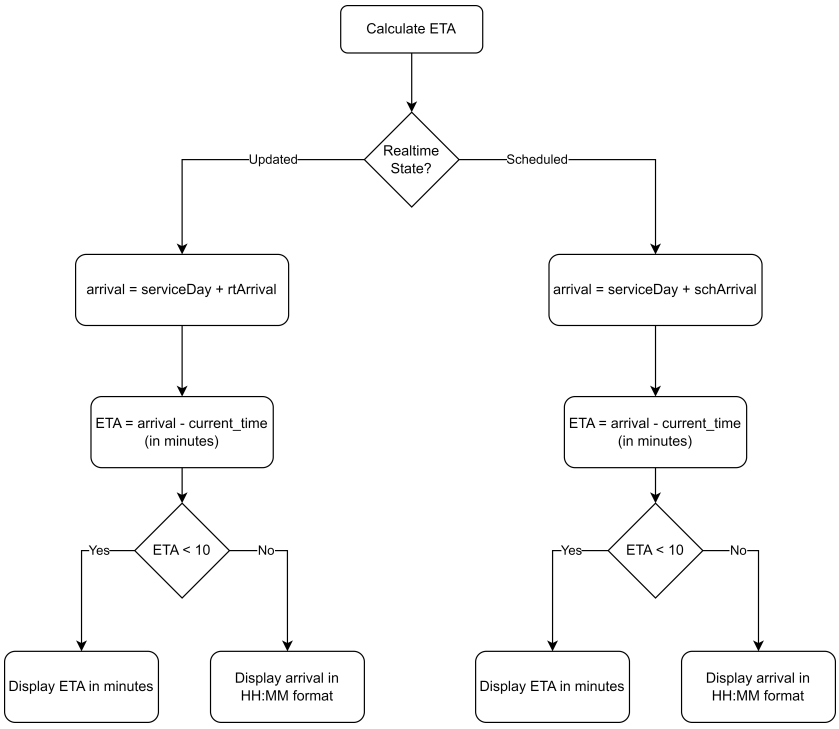


Figure 3.9: ETA Calculation Script Flowchart

Using the timestamps received in the above query, the function starts by checking if the value of the variable "realtimeState" is "SCHEDULED", meaning the timestamps received are congruent with the pre-defined scheduling or "UPDATED", meaning the timestamps received are updated in real-time. The next step will be calculating the bus's ETA. This ETA results from the difference between the actual time and the arrival information received, with the last value being the result of the sum of *serviceDay* and *rtArrival*/*schArrival* variables. (All code listings regarding ETA calculation are present in Appendix C section C.2)

If this ETA is less than 10 minutes, the time until the arrival should be displayed in minutes; if not, the time of arrival should be displayed in the HH:MM format. This format is used for better user experience and comprehension and is in concordance with the Explore Porto [3] arrivals information format.

The final step of this algorithm is the generation of an image. This process is done with the Python Image Library module, also known as Pillow. The fonts used in all text segments are the Municipality's official fonts, and the template design and formatting result from a thought process and analysis of Porto Digital's Graphic Designer.

In case the query to the database fails, meaning the device's MAC address is not registered, an image is generated displaying the MAC address that was received to be seen by the technician responsible for registering the device in the database.

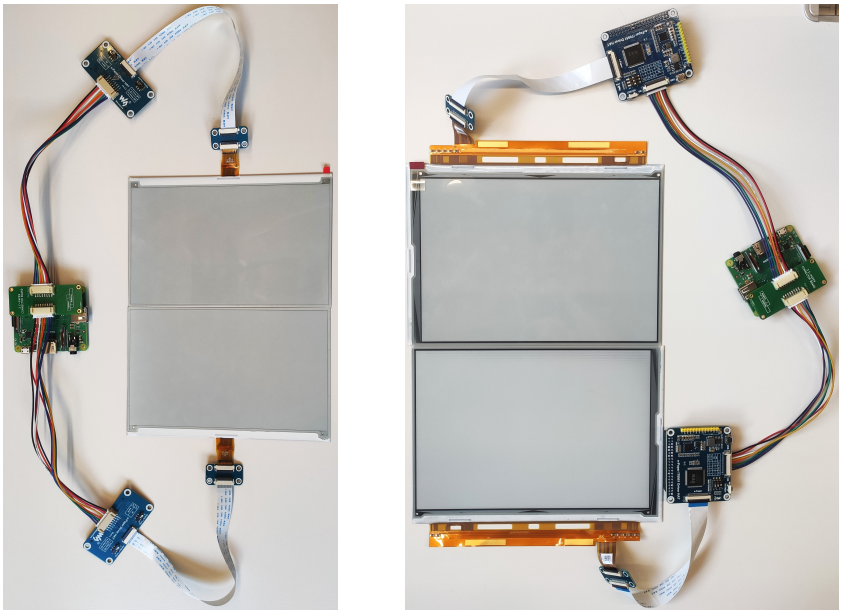
Chapter 4

Results

In a project implicating both hardware and software development, the results tend to be narrowed to show the physical implementation of the embedded system. In this project, the device is also combined with a web application that needs validation to analyse if it is optimised for a large-scale implementation. In brief, in the next sections, we will present relevant hardware implementation results and analyse key performance indicators on the server architecture built, also presenting the visible results such as the images from the web application resultant from the web application requests.

4.1 Hardware Results

The hardware results can be divided into two moments: the successful interface between the microcomputer and the displays and a functional real-world implementation. The PCB, as presented in sub-section 3.2.3, had a relevant influence on reducing the wiring needed to interface with a pair of displays, combined with an adapted and optimised driving software. It was possible to implement a solution with less wiring and ease of connection, as seen in figure 4.1. (Produced PCB image in Appendix A)



(a) 7.5-inch Configuration (b) 9.7-inch Configuration

Figure 4.1: Multiple Display Configuration

After developing a functional prototype of the device, and with the minimum required server-side deployed, with the help of a partner company Everythink [59], based on UPTEC, we designed and produced a 3D printed case. The main objective of this trial is to experiment with the implementation of a real-world use case of the device for better comprehension and analysis of the display's readability and physical dimensions. In figure 4.2, we can see the device inside the case with a built-in light strip for better nighttime visibility.

(a) LED Lighting ON (b) LED Lighting OFF

Figure 4.2: Device inside 3D Printed Case

4.2 Server Performance and Results

On the server side, the performance of the web application during high-load operations, meaning multiple devices are requesting their image simultaneously, is considered a key performance indicator. Also, evaluating how fast the connection and image retrieval happens in different types of wireless communication protocols is relevant.

Before presenting the performance tests done to the web application, it is relevant to present the designs and resulting images of this service, as seen in figure 4.3. In case the query to the database fails, meaning the device's MAC address is not registered, an example of the resulting image is demonstrated in figure 4.4.

TRINDADE → **12:47**

TRD1 **29 Mai. 2023**

LINHA	DESTINO	CHEGADA
703	**Sonhos**	1 min
304	**Sta. Luzia (Norte)**	1 min
200	**Bolhão**	6 min
600	**Maia (Barca)**	6 min
202	**Passeio Alegre (via Rot. Bessa)**	13:01
600	**Maia**	13:03
200	**Bolhão**	13:08

Figure 4.3: Arrivals Example Image

Figure 4.4: Device not Registered Example Image

To test the web application performance, we used Chrome's Web Browser DevTools to analyse the timings of the request and response procedures. On the Network tab, we can observe the different steps of this HTTP communication, evaluate the timings of each one, and simulate the communication speeds of different wireless communication protocols, such as 3G Cellular communication. In figures 4.5 and 4.6, we can visualise and compare both speed tests and relative communication timings, concluding that, as expected, the fast Wi-Fi connection provides a faster total communication procedure than the one made with a 3G Mobile connection.

Figure 4.5: 3G Communication Timings

Figure 4.6: Fast Wi-Fi Communication Timings

A relevant test to be made to evaluate the capacity and performance of a web application is doing a high-load test directed to the specified URL. The results of executing sequentially 20 curl [60] operations to the web service are listed in table 4.1. Highlighting the average time variable, the results help to conclude an increment of time compared to the test made using a single Wi-Fi request operation.

Table 4.1: Server Requests Performance

#	**Time(s)**	**Size(B)**
19	0,2088	28475
18	0,2281	28475
17	0,2412	28475
16	0,2083	28475
15	0,2178	28475
14	0,2318	28475
13	0,2364	28475
12	0,2317	28475
11	0,2183	28475
10	0,2376	28475
9	0,2262	28475
8	0,2209	28475
7	0,2243	28475
6	0,2315	28475
5	0,2268	28475
4	0,2392	28475
3	0,2251	28475
2	0,2162	28475
1	0,2310	28475
0	0,2132	28403
Average:	0,2257	

Chapter 5

Conclusions and Future Work

This research proposed the development of a new device that can display useful information to citizens. This information is generated by a flexible and optimised web service that uses a database in which the information is stored. The overall goal was to create a system, transversal to different use scenarios, to provide the community with reliable and relevant data. This holistic objective was sustained by the Tourism Agenda, included in the Resilience and Recuperation Plan from the Portuguese government.

The system has been fully developed and implemented as described in chapter 3, demonstrating its potential to contribute to sustainable growth and digital transition in cities and regions. While the system's ability to provide real-time bus arrival information is just one practical application, its versatility allows deployment in different contexts, as intended in the context and motivation presented in section 1.1. The prototype has achieved the initial goals, yet major developments and accomplishments occurred in specific areas. Adapting the software and hardware to interface multiple displays with only one microcomputer improved the system's scalability and flexibility. This enables broader deployment across various environments and application scenarios with multiple configurations possible. It is also important to highlight the work done on the server side, as it required the study of technologies not included in our program, such as DevOps, cloud computing and databases. This resulted in a server architecture fully capable and optimised for a large-scale implementation and flexible to include any future additions.

Considering future developments, these should focus on implementing additional

features on either the device or the server. On the one hand, additional layers of security will be needed to ensure robust protection against potential threats on the device. On the other hand, it would be crucial to endorse a software development that would be bulletproof against device and server errors. This would be achieved by handling all possible errors that can occur to guarantee reliable and uninterrupted operation. Specifically, developing a management and monitoring software tool for the device fleet would also facilitate efficient oversight and maintenance. Also, future work should implement continuous integration and deployment (CI/CD) processes and automatic device software updates. This would streamline the deployment of new features, bug fixes, and security patches, ensuring the system remains up-to-date and optimised without requiring constant manual setup.

In brief, this thesis has successfully developed and implemented a prototype system that provides reliable and relevant information through devices supported by a flexible web service. The project proposal contributes to cities' and regions' digital transition and sustainable growth. However, further research and development are needed to address challenges and enhance the system's capabilities. The system can be improved by incorporating additional security layers, implementing management tools, and adopting CI/CD processes for safe and resilient large-scale implementation. The outcomes of this research provide a strong foundation for future studies in information display systems and their applications. Hopefully, this work will inspire further exploration and innovation, improving public access to data and enhancing user experiences.

References

[1] MaaS-alliance, "What is maas?" 2018. [Online]. Available: https://maas-alliance.eu/homepage/what-is-maas/ [Cited on pages 2, 10, and 27]

[2] AMA-Agência para a Modernização, "Agenda acelerar e transformar o turismo," 2022. [Online]. Available: https://transparencia.gov.pt/ (Accessed 2023-06-13). [Cited on page 2]

[3] Porto Digital, "Explore porto," 2023. [Online]. Available: https://explore.porto.pt/ (Accessed 2022-12-20). [Cited on pages 2, 14, and 49]

[4] M. Tan, "Creating the Digital Economy: Strategies and Perspectives from Singapore," *International Journal of Electronic Commerce*, vol. 3, no. 3, pp. 105–122, Apr. 1999. [Online]. Available: https://doi.org/10.1080/10864415.1999.11518344 (Accessed 2023-03-07). [Cited on page 8]

[5] E. Ismagilova, L. Hughes, Y. K. Dwivedi, and K. R. Raman, "Smart cities: Advances in research—An information systems perspective," *International Journal of Information Management*, vol. 47, pp. 88–100, Aug. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401218312738 (Accessed 2023-03-07). [Cited on page 8]

[6] S. Hasija, Z.-J. M. Shen, and C.-P. Teo, "Smart City Operations: Modeling Challenges and Opportunities," *Manufacturing & Service Operations Management*, vol. 22, no. 1, pp. 203–213, Jan. 2020, publisher: INFORMS. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/msom.2019.0823 (Accessed 2023-03-02). [Cited on page 8]

[7] European Commission, "Smart cities Definition by the European Comission." [Online]. Available: https://commission.europa.eu/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en (Accessed 2023-02-16). [Cited on pages 8, 9, and 27]

[8] McKinsey, "Smart city technology for a more liveable future | McKinsey," 2019. [Online]. Available: https://www.mckinsey.com/capabilities/operations/our-insights/

smart-cities-digital-solutions-for-a-more-livable-future#part1 (Accessed 2023-03-02). [Cited on pages 8 and 9]

[9] Deloitte, "Smart Cities Strategies." [Online]. Available: https://www2.deloitte.com/us/en/pages/consulting/solutions/smart-cities-strategies.html (Accessed 2023-03-02). [Cited on pages 9 and 27]

[10] W. Bank, "Urban Development," 2019. [Online]. Available: https://www.worldbank.org/en/topic/urbandevelopment/overview (Accessed 2023-03-07). [Cited on page 9]

[11] E. Commission, "Circular economy action plan." [Online]. Available: https://environment.ec.europa.eu/strategy/circular-economy-action-plan_en (Accessed 2023-03-07). [Cited on page 10]

[12] A. S. City, "The challenges in the circular energy transition," Apr. 2022. [Online]. Available: https://amsterdamsmartcity.com/updates/news/the-challenges-in-the-circular-energy-transition (Accessed 2023-03-07). [Cited on page 10]

[13] E. Barrella, A. Amekudzi-Kennedy, M. Meyer, C. Ross, and D. Turchetta, "Best Practices and Common Approaches for Considering Sustainability at US State Transportation Agencies," *Transportation Research Record Journal of the Transportation Research Board*, vol. 2174, p. 10, Dec. 2010. [Cited on page 10]

[14] A. A. Ceder, "Urban mobility and public transport: future perspectives and review," *International Journal of Urban Sciences*, vol. 25, no. 4, pp. 455–479, Oct. 2021, publisher: Routledge _eprint: https://doi.org/10.1080/12265934.2020.1799846. [Online]. Available: https://doi.org/10.1080/12265934.2020.1799846 (Accessed 2023-03-02). [Cited on pages vii, 10, and 11]

[15] A. Sinanovic, E. Meskovic, A. Mujcic, and N. Suljanovic, "Smart city use case development based on FIWARE technology," in *2022 30th Telecommunications Forum (FOR)*, Nov. 2022, pp. 1–4. [Cited on pages 11 and 12]

[16] D. M. Berbes Villalón, L. Sánchez Jiménez, M. de la Iglesia Campos, M. E. Díaz Aguirre, and T. Delgado Fernández, "An IoT architecture for smart cities based on the FIWARE platform," *Revista de Ciencia y Tecnología: RECyT*, vol. 38, no. 1, pp. 20–27, 2022. [Online]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=8765906 (Accessed 2023-03-15). [Cited on page 12]

[17] OMF, "About MDS | Open Mobility Foundation," Sep. 2020. [Online]. Available: https://www.openmobilityfoundation.org/about-mds/ (Accessed 2023-03-15). [Cited on pages vii, 12, and 13]

[18] MobilityData, "General Transit Feed Specification," 2022. [Online]. Available: https://gtfs.org/ (Accessed 2023-03-10). [Cited on page 13]

[19] OTP, "Opentripplanner," 2023. [Online]. Available: https://www.opentripplanner.org/ (Accessed 2023-02-10). [Cited on page 16]

[20] Papercast, "Papercast bus Stops solution," 2022. [Online]. Available: https://www.papercast.com/e-paper-display-applications/bus-stops/ (Accessed 2023-03-31). [Cited on page 16]

[21] K. Marchbank, "Papercast joins IMMERSION+ 2020 hosted by NEC," Jul. 2020. [Online]. Available: https://www.papercast.com/event/papercast-joins-immersion-2020-hosted-by-nec/ (Accessed 2023-06-26). [Cited on pages vii and 17]

[22] T. Boshita, H. Suzuki, and Y. Matsumoto, "Smart Bus Stop using LoRaWAN and e-Paper," in *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*. Osaka, Japan: IEEE, Oct. 2019, pp. 278–282. [Online]. Available: https://ieeexplore.ieee.org/document/9015448/ (Accessed 2022-12-18). [Cited on pages vii, 17, 18, and 28]

[23] M. M. C. Macedo, "The Implications of Interactive E-Paper Technology on Public Transportation," Jul. 2021. [Online]. Available: https://repositorio-aberto.up.pt/handle/10216/135082 [Cited on pages 19 and 28]

[24] Porto Digital, "Associação Porto Digital," 2023. [Online]. Available: https://www.portodigital.pt/associacao-porto-digital/ (Accessed 2023-03-31). [Cited on page 19]

[25] Waveshare, "Waveshare 10.1 lcd display," 2020. [Online]. Available: https://www.waveshare.com/product/displays/lcd-oled/lcd-oled-1/10.1inch-hdmi-lcd-e.htm?____SID=U (Accessed 2023-03-31). [Cited on page 19]

[26] Waveshare, "Waveshare 10.3 e-paper display," 2022. [Online]. Available: https://www.waveshare.com/10.3inch-e-paper.htm (Accessed 2023-03-31). [Cited on page 19]

[27] S. Kholghi Eshkalak, M. Khatibzadeh, E. Kowsari, A. Chinnappan, W. A. D. M. Jayathilaka, and S. Ramakrishna, "Overview of electronic ink and methods of production for use in electronic displays," *Optics & Laser Technology*, vol. 117, pp. 38–51, Sep. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030399217315487 (Accessed 2023-05-24). [Cited on page 20]

[28] E Ink, "Electronic ink, how it works," 2022. [Online]. Available: https://www.eink.com/tech/detail/How_it_works (Accessed 2023-02-10). [Cited on pages vii and 20]

[29] A. G. Vacroux, "Microcomputers," *Scientific American*, vol. 232, no. 5, pp. 32–41, 1975. [Online]. Available: https://www.jstor.org/stable/24949796 (Accessed 2023-04-05). [Cited on page 21]

[30] S. J. Johnston, M. Apetroaie-Cristea, M. Scott, and S. J. Cox, "Applicability of commodity, low cost, single board computers for Internet of Things devices," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec. 2016, pp. 141–146. [Cited on page 21]

[31] R. P. Foundation, "Teach, learn, and make with the Raspberry Pi Foundation," May 2023. [Online]. Available: https://www.raspberrypi.org/ (Accessed 2023-05-25). [Cited on pages vii, 22, and 23]

[32] S. J. Johnston and S. J. Cox, "The raspberry pi: A technology disrupter, and the enabler of dreams," *Electronics*, vol. 6, no. 3, 2017. [Online]. Available: https://www.mdpi.com/2079-9292/6/3/51 [Cited on page 22]

[33] NVIDIA, "Developer kit jetson nano," 2023. [Online]. Available: https://www.nvidia.com/pt-pt/autonomous-machines/jetsonstore/ (Accessed 2023-04-10). [Cited on pages vii and 23]

[34] IoT Analytics, "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time," Nov. 2020. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ (Accessed 2023-03-31). [Cited on page 24]

[35] IEEE, "IEEE 802.11-2020/Cor 1-2022," Dec. 2022. [Online]. Available: https://standards.ieee.org (Accessed 2023-04-28). [Cited on page 24]

[36] S. Tabbane, *IoT StandardsPart II: Part II: 3GPP Standards*, Bandung, Indonesia, Sep. 2018. [Online]. Available: https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/Documents/Events/2018/IoT-BDG/7.IoTStandardsPartII-SamiTabbane.pdf [Cited on page 24]

[37] L. Alliance, "Lorawan® specification," 2023. [Online]. Available: https://lora-alliance.org/about-lorawan/ (Accessed 2023-04-28). [Cited on page 24]

[38] P. Barry and P. Crowley, "Modern embedded computing," in *Modern Embedded Computing*. Boston: Morgan Kaufmann, 2012, p. iv. [Online]. Available: https:

//www.sciencedirect.com/science/article/pii/B9780123914903020014 [Cited on pages vii and 25]

[39] Flask, "Flask python." [Online]. Available: https://flask.palletsprojects.com/ (Accessed 2023-05-30). [Cited on page 25]

[40] "Django." [Online]. Available: https://www.djangoproject.com/ (Accessed 2023-06-26). [Cited on page 25]

[41] D. Ghimire, "Comparative study on Python web frameworks: Flask and Django," 2020. [Online]. Available: http://www.theseus.fi/handle/10024/339796 (Accessed 2023-06-06). [Cited on page 25]

[42] "What is a Reverse Proxy Server?" [Online]. Available: https://www.nginx.com/resources/glossary/reverse-proxy-server/ (Accessed 2023-06-25). [Cited on page 26]

[43] N. Nikolakis, A. Marguglio, G. Veneziano, P. Greco, S. Panicucci, T. Cerquitelli, E. Macii, S. Andolina, and K. Alexopoulos, "A microservice architecture for predictive analytics in manufacturing," *Procedia Manufacturing*, vol. 51, pp. 1091–1097, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978920320102 (Accessed 2023-05-16). [Cited on page 26]

[44] "Advanced Load Balancer, Web Server, & Reverse Proxy." [Online]. Available: https://www.nginx.com/ (Accessed 2023-05-16). [Cited on page 26]

[45] C. Ma and Y. Chi, "Evaluation Test and Improvement of Load Balancing Algorithms of Nginx," *IEEE Access*, vol. 10, pp. 14 311–14 324, 2022. [Cited on page 26]

[46] Docker, "Docker." [Online]. Available: https://www.docker.com/ (Accessed 2023-05-30). [Cited on page 26]

[47] Docker, "Docker basics," Jun. 2023. [Online]. Available: https://docs.docker.com/get-started/ (Accessed 2023-06-12). [Cited on page 27]

[48] Docker, "Docker compose overview," Jun. 2023. [Online]. Available: https://docs.docker.com/compose/compose-file/ (Accessed 2023-06-12). [Cited on page 27]

[49] H. Jin, D. Busvine, and D. Kirton, "Analysis: Global chip shortage threatens production of laptops, smartphones and more," *Reuters*, Dec. 2020. [Online]. Available: https://www.reuters.com/business/autos-transportation/global-chip-shortage-threatens-production-laptops-smartphones-more-2020-12-17/ (Accessed 2023-05-25). [Cited on page 32]

[50] Raspberry Pi, "Raspberry Pi 3 Model A+." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-a-plus/ (Accessed 2023-06-14). [Cited on pages vii and 33]

[51] Kicad, "KiCad EDA." [Online]. Available: https://www.kicad.org/ (Accessed 2023-05-29). [Cited on page 34]

[52] Drewsif, "Pishrink." [Online]. Available: https://github.com/Drewsif/PiShrink (Accessed 2023-05-29). [Cited on page 39]

[53] PyPi, "Raspberry pi gpio python module." [Online]. Available: https://pypi.org/project/RPi.GPIO/ (Accessed 2023-05-29). [Cited on page 42]

[54] PyPi, "Linux kernel spi dev python module." [Online]. Available: https://pypi.org/project/spidev/ (Accessed 2023-05-29). [Cited on page 42]

[55] D. Bartholomew, "Mariadb vs. mysql," *Dostopano*, vol. 7, no. 10, p. 2014, 2012. [Cited on page 44]

[56] Docker, "Dockerfile reference," Jun. 2023. [Online]. Available: https://docs.docker.com/engine/reference/builder/ (Accessed 2023-06-12). [Cited on page 45]

[57] MariaDB, "Mariadb python connector." [Online]. Available: https://pypi.org/project/mariadb/ (Accessed 2023-05-30). [Cited on page 47]

[58] GraphQL, "Graphql python module." [Online]. Available: https://pypi.org/project/gql/ (Accessed 2023-03-12). [Cited on page 47]

[59] Everythink, "Everythink - design for change." [Online]. Available: https://everythink.com/ (Accessed 2023-05-30). [Cited on page 52]

[60] Curl, "command line tool and library for transferring data with urls," 2023. [Online]. Available: https://curl.se/ (Accessed 2023-04-13). [Cited on page 55]

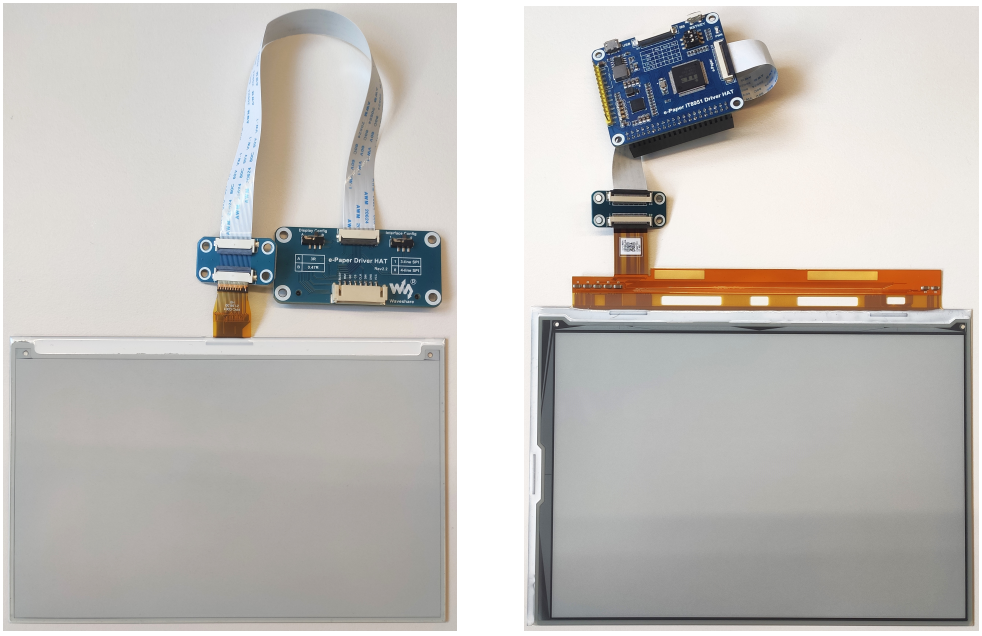
[61] Waveshare, "7.5-inch hat manual." [Online]. Available: https://www.waveshare.com/wiki/7.5inch_e-Paper_HAT_Manual#Overview (Accessed 2023-05-29). [Cited on pages vii and 68]

Appendix A

Hardware Development

A.1 Device Options

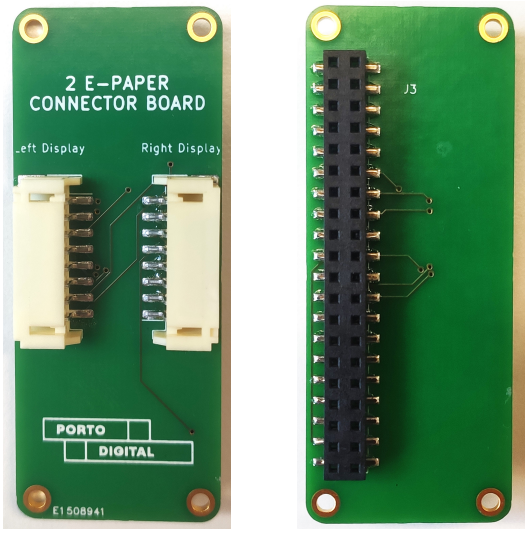
A.1.1 Displays and Driver Boards



(a) 7.5-inch (b) 9.7-inch

Figure A.1: Two Waveshare E-paper Displays and Driver Board

A.1.2 Two-display Connector PCB



(a) Frente (b) Verso

Figure A.2: Two-display Connector PCB

Appendix B

Software Diagrams

B.1 E-Paper Display Functioning Flowchart

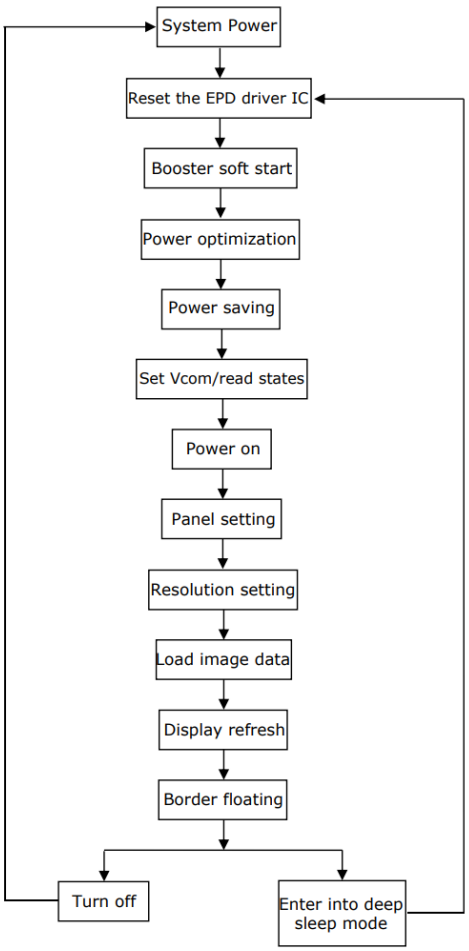


Figure B.1: EPD Full Initialisation and Functioning Routine [61]

B.2 Total Web App Flowchart

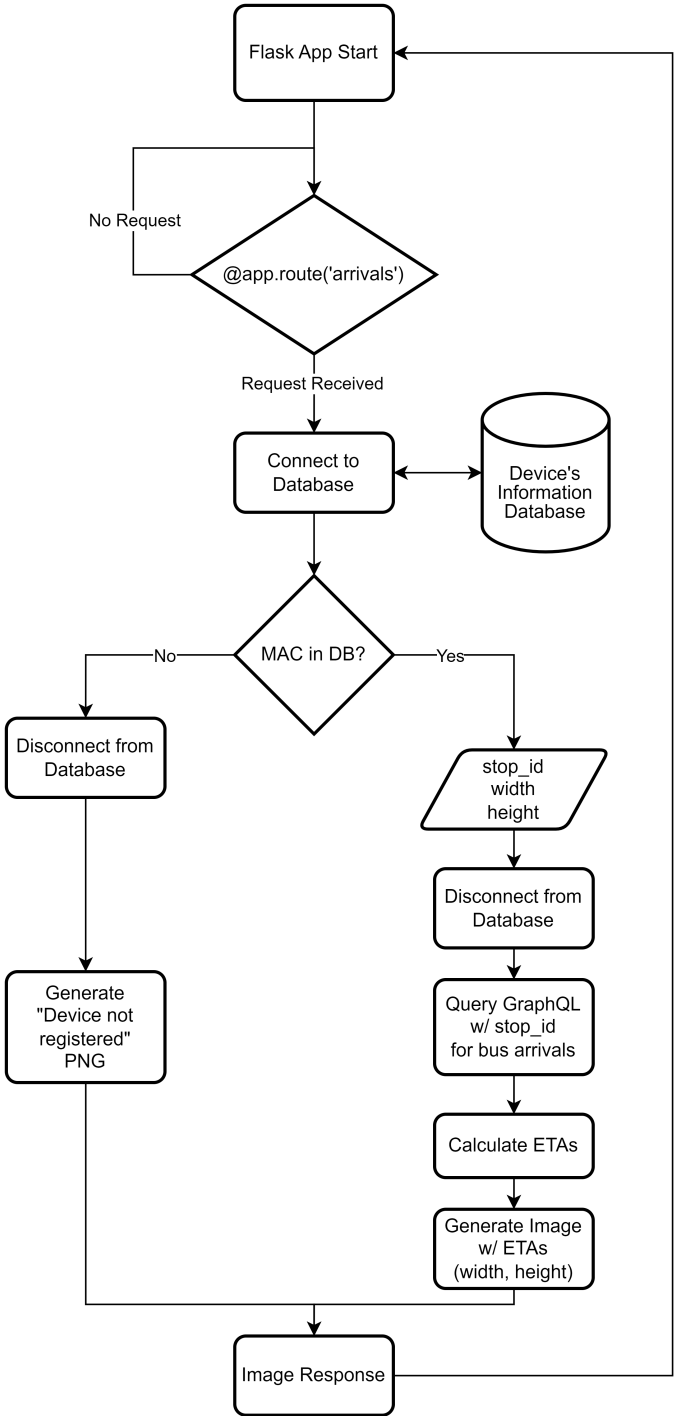


Figure B.2: Server Application Flowchart

Appendix C

Code Listings

C.1 Device Software

C.1.1 E-Paper Display Communication Functions

```
1
2  def send_command(self, command):
3      self.rasp.digital_write(self.dc_pin, 0)
4      self.rasp.digital_write(self.cs_pin, 0)
5      self.rasp.spi_writebyte([command])
6      self.rasp.digital_write(self.cs_pin, 1)
7
8  def send_data2(self, data):
9      self.rasp.digital_write(self.dc_pin, 1)
10     self.rasp.digital_write(self.cs_pin, 0)
11     self.rasp.SPI.writebytes2(data)
12     self.rasp.digital_write(self.cs_pin, 1)
13
14 def ReadBusy(self):
15     logger.debug("e-Paper busy")
16     iter = 0
17     while self.rasp.digital_read(self.busy_pin) == 0:
18         self.rasp.delay_ms(100)
19         iter += 1
20         if iter > 150:
21             logger.info("Forced e-paper busy release")
```

```
22              break
23          logger.debug("e-Paper busy release")
24
25  def display(self, image):
26      self.send_command(0x13)
27      self.send_data2(image)
28
29      self.send_command(0x12)
30      self.rasp.delay_ms(100)
31      self.ReadBusy()
```

Listing C.1: Communication-specific Functions

C.1.2 Device's Main Program

```
1
2  if __name__ == "__main__":
3
4      #Get MAC address
5      myMAC = (hex(uuid.getnode()))
6
7      #Request for image
8      response = requests.get(f"https://bus.api.porto.digital/
           arrivals75?MAC={myMAC}")
9
10     #convert byte array to image object
11     image = Image.open(BytesIO(response.content))
12
13     image1 = image.convert(mode="1")
14     width, height = image1.size
15
16     #Numbrer of pictures = Number of displays
17     pictures = 2
18     box = []
19
20     #Divide image into two parts and rotate them
21     edges = np.linspace(0, width, pictures+1)
22     for start, end in zip(edges[:-1], edges[1:]):
23         box.append((start, 0, end, height))
24     halfLeft = image1.crop(box[0])
25     halfRight = image1.crop(box[1])
26     halfLeft1 = halfLeft.rotate(90, expand=True)
27     halfRight1 = halfRight.rotate(-90, expand=True)
28
29     try:
30         #Initialisation of both threads and respective arguments
```

```
31          display1Thread = threading.Thread(target=send_display1,
                args=(halfLeft1,), daemon = True)
32          display2Thread = threading.Thread(target=send_display2,
                args=(halfRight1,), daemon = True)
33
34          logging.info("starting threads")
35          display1Thread.start()
36          display2Thread.start()
37
38          display1Thread.join()
39          display2Thread.join()
40          logging.info("threads ended")
41
42          time.sleep(1)
43
44      except IOError as e:
45          logging.info(e)
46
47      except KeyboardInterrupt:
48          logging.info("ctrl + c:")
49
50          exit()
```

Listing C.2: Main Device Program

```
1
2  def send_display1(image):
3      #EPD init
4      # epd pin init (SPI_BUS, CE, BUSY, RST, DC)
5      epd = epd7in5_V2.EPD(0, 8, 27 , 22, 17)
6      epd.init()
7      logging.info("displaying on display 1")
8      epd.display(epd.getbuffer(image))
9      time.sleep(2)
10     logging.info("ended displaying on display 1")
11     epd.sleep()
```

Listing C.3: Thread Example of Device's Program

C.2 Server Software - Web Application

```
def query_execute(stop_id, client):
  actual_time = int(time.time())
  query_string = f"""
              query Query{{
                  stop(id: {stop_id}) {{
                    name
                    code
                    stoptimesWithoutPatterns(startTime: {
                        actual_time} , numberOfDepartures: 7,
                        timeRange: 86400) {{
                      headsign
                      realtimeState
                      realtimeArrival
                      scheduledArrival
                      trip {{
                        routeShortName
                      }}
                      serviceDay
                    }}
                }}
  }}
  """
  query = gql(query_string)
  #Query to the Explore Porto GraphQL
  result = client.execute(query)

  #List of the next 6 arrivals
  arrivals_list = []
  curr_time_raw = datetime.now().strftime("%H:%M:%S")
  print(f"Evaluating at this time: {curr_time_raw}")
  #Extraction of the info of each arrival and append to
      arrivals_list
  for n in (result['stop']['stoptimesWithoutPatterns']):
    x = Arrival_info(result['stop']['name'], n['headsign'], n['
        realtimeState'], n['realtimeArrival'], n['scheduledArrival'
        ], n['serviceDay'], n['trip']['routeShortName'], 0)
    arrivals_list.append(x)
    print(f"{x.routeNumber} - {x.headsign} - {x.ETA}")
  return arrivals_list
```

Listing C.4: GraphQL Query for Arrival Data

```
1
2  def calculateETA(rtState, current_time, rtArrival, schArrival,
       serviceDay):
3    if(rtState == 'UPDATED'):
4      arrival = serviceDay + rtArrival
5      ETA = math.ceil((int(arrival) - current_time)/60)
6
7      if ETA < 10:
8        ETA = str(ETA) + " min"
9
10     else:
11       ETA = datetime.fromtimestamp(arrival).strftime('%H:%M')
12
13   elif (rtState == 'SCHEDULED'):
14     arrival = serviceDay + schArrival
15     ETA = math.ceil((int(arrival) - current_time)/60)
16
17     if ETA< 10:
18       ETA = str(ETA) + " min"
19
20     else:
21       ETA = datetime.fromtimestamp(arrival).strftime('%H:%M')
22
23   return ETA
```

Listing C.5: ETA calculation Function

```
1
2  def genPng(arrivals, stop_id, width, height):
3
4      now = datetime.now()
5      time_string = now.strftime("%H:%M")
6      current_month = months[now.month]
7      date_string = now.strftime("%d ") + current_month + now.
           strftime(" %Y")
8
9      img = Image.new(mode = "L", size=(width, height), color="white
           ")
10     drw = ImageDraw.Draw(img)
11
12
13     stop_id = stop_id[3:]
14     stop_id = stop_id[:-1]
15
16     ########  Import Fonts  ############
17     regularB45 = ImageFont.truetype(os.path.join(pngFilesPath, "
           Regular-Bold.otf"), size = int(5.95*(height/100)))
```

```
18      regularB40 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Bold.otf"), size = int(5.35*(height/100)))
19      regularB32 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Bold.otf"), size = int(4.70*(height/100)))
20      regularB30 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Bold.otf"), size = int(4.14*(height/100)))
21      regularB22 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Bold.otf"), size = int(2.5*(height/100)))
22
23      regularM32 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Medium.otf"), size = int(4.70*(height/100)))
24      regularM25 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Medium.otf"), size = int(3.53*(height/100)))
25      regularM24 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Medium.otf"), size = int(3.41*(height/100)))
26      regularM23 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Medium.otf"), size = int(3.29*(height/100)))
27      regularM22 = ImageFont.truetype(os.path.join(pngFilesPath, "
            Regular-Medium.otf"), size = int(3.17*(height/100)))
28
29      ########  Header zone (clock, stop name and code)  ############
30      time_size= drw.textlength(time_string, font=regularB40)
31      date_size= drw.textlength(date_string, font=regularB30)
32      time_box = drw.rounded_rectangle([(85.42*(width/100), 4*(
            height/100)),(97.42*(width/100), 11*(height/100))], fill='
            black', radius = 2)
33      time = drw.text((91.67*(width/100),9.38*(height/100)),
            time_string, anchor="ms", fill='white', font=regularB40)
34      time_date = drw.text((96.88*(width/100),16.25*(height/100)),
            date_string, anchor="rs" , fill='black', font = regularB30)
35      stop_name = drw.text((3.13*(width/100),9.38*(height/100)),
            arrivals[0].stopName + " >", anchor="ls", fill='black',
            font = regularB45)
36      stop_code = drw.text((7.29*(width/100),15.63*(height/100)),
            stop_id, anchor='ms', fill='black', font = regularM24)
37      stop_code_box = drw.rounded_rectangle([(3.13*(width/100), 12*(
            height/100)),(11.46*(width/100),16.88*(height/100))],
            outline='black', width=1, radius= 2)
38
39      ######## Create table header  ########
40      row_font = regularM32
41      row_font_destino = regularB32
42      collumn_font = regularM25
43      drw.text((3.13*(width/100), 24.38*(height/100)), 'LINHA',
            anchor='ls', fill='black', font=collumn_font)
44      drw.text((18.75*(width/100),24.38*(height/100)), 'DESTINO',
            anchor='ls', fill='black', font=collumn_font)
45      drw.text((96.88*(width/100),24.38*(height/100)), 'CHEGADA',
            anchor='rs', fill='black', font=collumn_font)
```

```
46
47      ########  Number of arrivals counter  ########
48      arrivalsLen = len(arrivals)
49      count = 0
50
51      ########  Create table static horizontal line  ########
52      line_static_start = drw.line([(3.13*(width/100),26.25*(height
            /100)), (96.88*(width/100),26.25*(height/100))], fill='
            black', width=2)
53
54      ########  Add rows and cells to the body of the table
            ########
55      for (arrival, y) in zip(arrivals, range (1, arrivalsLen + 1)):
56
57        count += 1
58        y_insert =  (8.75 * (y + 1) + 15)*(height/100)
59
60        if count != arrivalsLen:
61          line_static_ms = drw.line([(3.13*(width/100),y_insert +
                3.13*(height/100)), (96.88*(width/100),y_insert +
                3.13*(height/100))], fill='black', width=1)
62        else:
63          line_static_end = drw.line([(3.13*(width/100),y_insert +
                3.13*(height/100)), (96.88*(width/100),y_insert +
                3.13*(height/100))], fill='black', width=2)
64
65        for x in range (1, 4):
66          if x == 1:
67            x_insert = 3.13*(width/100)
68            drw.text((x_insert, y_insert), arrival.routeNumber,
                  anchor='ls', font=row_font, fill='black')
69
70          elif x == 2:
71            x_insert = 18.75*(width/100)
72            drw.text((x_insert, y_insert), arrival.headsign, anchor=
                  'ls', font=row_font_destino, fill='black')
73
74          else:
75            x_insert = 96.88*(width/100)
76            drw.text((x_insert, y_insert), arrival.ETA, font=
                  row_font, anchor='rs', fill='black')
77
78      # Save the file
79      img_io = BytesIO()
80      img.save(img_io, 'PNG', quality=100)
81      img_io.seek(0)
82      return send_file(img_io, mimetype='image/png')
```

Listing C.6: Arrivals PNG Generation Function

```
def genPngMAC(MAC, width, height):
    now = datetime.now()
    time_string = now.strftime("%H:%M")
    current_month = months[now.month]
    date_string = now.strftime("%d ") + current_month + now.
        strftime(" %Y")

    img = Image.new(mode = "L", size=(width, height), color="white
        ")
    drw = ImageDraw.Draw(img)

    ########  Import Fonts  ############
    regularB45 = ImageFont.truetype(os.path.join(pngFilesPath, "
        Regular-Bold.otf"), size = int(5.95*(height/100)))
    regularB40 = ImageFont.truetype(os.path.join(pngFilesPath, "
        Regular-Bold.otf"), size = int(5.35*(height/100)))
    regularB30 = ImageFont.truetype(os.path.join(pngFilesPath, "
        Regular-Bold.otf"), size = int(4.14*(height/100)))

    ########  Header zone (clock, stop name and code)  ############
    time_size= drw.textlength(time_string, font=regularB40)
    date_size= drw.textlength(date_string, font=regularB30)
    time_box = drw.rounded_rectangle([(85.42*(width/100), 4*(
        height/100)),(97.42*(width/100), 11*(height/100))], fill='
        black', radius = 2)
    time = drw.text((91.67*(width/100),9.38*(height/100)),
        time_string, anchor="ms", fill='white', font=regularB40)
    time_date = drw.text((96.88*(width/100),16.25*(height/100)),
        date_string, anchor="rs" , fill='black', font = regularB30)

    message = drw.text((50*(width/100), 40*(height/100)), "Device
        not registered", anchor="ms", fill='black', font=regularB45
        )
    mac = drw.text((50*(width/100), 60*(height/100)), "MAC: " +
        MAC, anchor="ms", fill='black', font=regularB40)

    # Save the file
    img_io = BytesIO()
    img.save(img_io, 'PNG', quality=100)
    img_io.seek(0)
    return send_file(img_io, mimetype='image/png')
```

Listing C.7: Device not Registered PNG Generation