INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Recomendações Baseadas no Contexto dos Turistas num Sistema de Recomendação de Grupo Para o Turismo

FRANCISCO MANUEL PEREIRA SOARES LANDEIRO NEGRÃO
Outubro de 2023

POLITÉCNICO
DO PORTO

Recomendações Baseadas no Contexto dos Turistas num Sistema de Recomendação de Grupo Para o Turismo

Francisco Manuel Pereira Soares Landeiro Negrão

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Professora Ana Almeida

Co-orientador: Professora Patrícia Alves

Júri:

Vogais:

Porto, outubro 2023

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 14 de outubro de 2023

Agradecimentos

Gostaria de expressar o meu agradecimento a todos aqueles que me apoiaram e se mostraram sempre disponíveis para me ajudar nesta longa e complicada caminhada, mas enriquecedora. À minha família, nomeadamente aos meus pais pela paciência e encorajamento e à minha irmã pela sua amizade e alegria.

Aos meus amigos, em especial ao João Trindade por ter estado sempre ao meu lado nesta aventura.

À Professora Ana Almeida pela orientação e conselhos.

Não posso deixar de dar um agradecimento muito especial à Professora Patrícia Alves, por ter sido uma excelente tutora, estando sempre disponível para esclarecer dúvidas e ajudar no planeamento do projeto.

O meu obrigado a todos!

Resumo

Os sistemas de recomendação (SR) têm desempenhado um papel importante em várias áreas, permitindo aos utilizadores tomar decisões mais informadas e uma das áreas que mais tem beneficiado com o crescimento destes sistemas, é a área do turismo. No entanto, a grande parte destes sistemas não têm em consideração uma vertente importante: os interesses dos visitantes e o contexto envolvente, que são potencializadores de uma experiência mais enriquecedora e personalizada.

Neste contexto, é fundamental considerar aspetos como o clima, a personalidade do utilizador, bem como os seus medos e limitações. Estes elementos têm o potencial de melhorar a forma como as recomendações são feitas. Ao melhorar o motor de recomendações, é possível criar sugestões que tenham em conta o contexto meteorológico, adaptando-se às condições atuais, e que também considerem a personalidade e as limitações dos utilizadores.

No presente documento, é apresentada uma extensa revisão de literatura, uma análise e desenho da solução, detalhes sobre a implementação da solução e, por fim, uma avaliação e análise dos resultados obtidos.

A solução implementada, que se centrou na inclusão do contexto onde o utilizador se encontra inserido, permitiu que o SR a ser desenvolvido evoluísse, assegurando recomendações que estão mais de acordo com a personalidade, as preocupações e as limitações do utilizador.

Palavras-chave: Contexto, Sistemas de Recomendação, Sistemas de Recomendação para Grupos, Sistemas de Recomendação no Turismo, Virtual Pet, Meteorologia, Personalidade, Limitações

Abstract

Recommendation systems (RS) have played an important role in various areas, allowing users to make more informed decisions and one of the areas that has benefited most from the growth of these systems is tourism. However, most of these systems do not consider an important aspect: the interests of visitors and the surrounding context, which can enhance a more enriching and personalized experience.

In this context, it is essential to consider aspects such as the climate, the user's personality, as well as their fears and limitations. These elements have the potential to improve the way recommendations are made. By improving the recommendation engine, it is possible to create suggestions that consider the weather context, adapting to current conditions, and that also consider users' personalities and limitations.

This document presents an extensive literature review, an analysis and design of the solution, details on the implementation of the solution and, finally, an evaluation and analysis of the results obtained.

The solution implemented, which focused on including the context in which the user is located, allowed the SR being developed to evolve, ensuring recommendations that are more in line with the user's personality, concerns, and limitations.

Keywords: Context, Recommendation Systems, Recommendation Systems for Groups, Recommendation Systems in Tourism, Virtual Pet, Meteorology, Personality, Limitations

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
API	*Application Programming Interface*
CAP	*Consistency, Availability, Partition tolerance*
CARS	*Context-aware Recommender Systems*
CRUD	*Create, read, update and delete*
DTO	*Data Transfer Object*
FURPS	*Functionality, Usability, Reliability, Performance, Security*
GECAD	Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e o Desenvolvimento
GRASP	*General Responsibility Assignment Software Patterns*
GRS	*Group Recommender Systems*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IC	Índice de Consistência
IR	Índice aleatório
JDK	*Java Development Kit*
JDT	*Java Development Tools*
JPA	*Java Persistence API*
JRE	*Java Runtime Environment*
JVM	*Java Virtual Machine*
MAMS	*Multi-Agent Microservice*
MAS	*Multi-Agent Service*
MOSCOW	*Must-have,Should-have, Could-have, and Won't-have*
MTRS	*Mobile Tourism Recommendation System*

POI	*Point of Interest*
RC	Razão de Consistência
REST	*Representational State Transfer*
RS	*Recommender Systems*
SDK	*Software Development Kit*
SOLID	*Single Responsibility principle, Open-closed principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle*
SQL	*Structured Query Language*
UUID	*Universal Unique Identifier*

Lista de Símbolos

$\boldsymbol{\lambda max}$	Maior valor próprio da matriz A

1 Introdução

1.1 Contexto

Esta dissertação foi desenvolvida no âmbito da unidade curricular TMDEI, do Mestrado em Engenharia Informática, na especialidade Engenharia de software no Instituto Superior de Engenharia do Porto (ISEP).

O trabalho foi realizado no GECAD (Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development) que é uma Unidade de Investigação que tem como missão o desenvolvimento da investigação científica e inovação para a incorporação da Inteligência em Engenharia e Sistemas Complexos de Computação, situado no ISEP.

Esta dissertação foi desenvolvida no âmbito do projeto SmartTravel. Este é uma continuação do projeto que estava a ser desenvolvido anteriormente, o GroupPlanner. Esta iniciativa contou com a colaboração de um parceiro nacional (Sistrade) e de vários internacionais, sendo eles da Roménia, Espanha e Turquia.

O objetivo deste trabalho foi o de explorar e aperfeiçoar um sistema de recomendação para grupos na área de turismo usando o contexto, o protótipo Grouplanner. Este protótipo é um sistema de recomendação de pontos turísticos para grupos e individuais que está organizado em microsserviços e utiliza uma tecnologia baseada em agentes, com o intuito de fornecer recomendações personalizadas.

Para alcançar este objetivo, foram utilizadas metodologias de pesquisa e de análise rigorosas. O resultado deste trabalho é um conjunto de recomendações e orientações para a implementação de sistemas de recomendação eficazes no setor de turismo.

1.2 Problema

Os Sistemas de Recomendação (SR) são ferramentas essenciais em muitas áreas, ajudando os utilizadores a tomar decisões mais informadas. No entanto, quando se trata de grupos de pessoas, os desafios são mais complexos, uma vez que é necessário encontrar um equilíbrio das diferentes preferências de cada participante. Para ajudar neste tipo de problemas, Sistemas de Recomendação para Grupos (GRS) têm sido desenvolvidos no ramo do turismo, que integrados com dispositivos móveis que têm em conta o contexto do utilizador, são capazes de proporcionar uma experiência mais satisfatória ao utilizador (Alves, Carneiro, Marreiros, & Novais, 2019). Ainda assim, fazendo uma análise da literatura existente, chega-se à conclusão de que não existem muitos GRS que tenham em conta tanto os interesses dos visitantes, como o contexto em que eles se encontram. Por exemplo, se um grupo for visitar a Torre dos Clérigos e um dos membros possuir medo de alturas (acrofobia), o sistema deve avisá-lo para evitar a subida, prevenindo o desconforto, ou se estiver a chover e deixa de ser possível ir visitar um jardim zoológico, o GRS deve ser capaz de sugerir uma recomendação alternativa.

Com a realização desta dissertação, pretende-se ultrapassar as restrições ligadas ao contexto com o objetivo de melhorar a experiência dos turistas com GRS. A informação contextual será enviada para um *Virtual Pet* (projeto que está a ser desenvolvido em paralelo por outro colega) que será embutido na aplicação principal para notificar o turista, integrando assim os dois projetos. Para fornecer sugestões precisas e satisfatórias baseadas no contexto, vão ser feitas melhorias no motor de recomendações, no protótipo em desenvolvimento.

1.3 Objetivos

Com este projeto, pretendeu-se resolver as limitações de foco nos interesses pessoais dos turistas aliados ao seu contexto e melhorar a experiência dos turistas no GRS de turismo que se encontra em desenvolvimento, mostrando preocupação pelos seus interesses através do uso do contexto em que estão.

Foram esperados os seguintes resultados:

1. Contextualização sobre o estado da arte de Sistemas de Recomendação Individuais e para Grupos, contexto e preferências do utilizador, e *Virtual Pet*.

2. Melhoria do motor de recomendações do Grouplanner através do desenvolvimento da capacidade de fazer recomendações pessoais, ou para o grupo, com base no contexto recebido via HTTP REST;
3. Integração das informações contextuais com o *Virtual Pet*;
4. Testes de integração e funcionais do MAS com o protótipo do GRS e a aplicação fornecedora do contexto;
5. Teste final com cenários de casos de uso reais;
6. Análise dos resultados e escrita da dissertação.

1.4 Metodologia

Para o desenvolvimento do projeto, recorreu-se à *framework* ágil *Scrum*.

Scrum é uma das ferramentas mais conhecidas de gestão que as equipas utilizam para se organizarem e trabalharem para um objetivo comum. Esta *framework* define uma série de reuniões, recursos e tarefas para uma entrega eficaz do projeto (Russo, 2003a).

Para a gestão do projeto e resolução de problemas, as equipas *Scrum* utilizam alguns artefactos, sendo eles os seguintes (Russo, 2003a):

- ***Product Backlog*** – É a lista de recursos, requisitos e funcionalidades que devem ser realizados para que o produto final seja concluído com sucesso.
- ***Sprint Backlog*** – É a lista de itens que devem ser concluídos numa determinada iteração (*Sprint*). Para esta lista vão certos itens (definidos pela equipa) presentes no *Product Backlog* e estes são escolhidos quando se está a fazer o planeamento do *Sprint* (*Sprint Planning*).

Quando terminado um Sprint, é realizada uma reunião informal, com os membros da equipa, com o intuito de verificar e analisar os resultados obtidos. Esta ação denomina-se de *Sprint Review (Russo, 2003a)*.

No desenvolvimento deste projeto, cada *Sprint* teve uma duração de 15 dias. Para listar os itens e acompanhar o seu estado, utilizou-se o Excel como ferramenta de gestão.

2 Estado da Arte

Neste capítulo é fornecida uma visão geral do estado da arte em diversas áreas, nomeadamente o atual conhecimento dos tópicos mencionados, os trabalhos relacionados, as tecnologias existentes e a descrição de tomadas de decisão em certos campos.

O capítulo encontra-se dividido em várias secções, começando com sistemas de recomendação. Em seguida, é abordado o tema de sistemas de recomendação de grupos no turismo, que visam oferecer recomendações para grupos de pessoas que viajam juntas. Também é discutido o tema de sistemas de recomendação contextual, que levam em consideração o contexto em que as sugestões são fornecidas. Para além disso, esta seção apresenta uma análise dos trabalhos relacionados que têm sido desenvolvidos na área de sistemas de recomendação contextual e uma descrição das tecnologias existentes que são utilizadas no melhoramento desses sistemas.

2.1 Sistemas de Recomendação

Os sistemas de recomendação têm ganho cada vez mais importância como uma ferramenta que ajuda os utilizadores a receberem recomendações personalizadas (Alves, Gomes, Rodrigues, Carneiro, Novais & Marreiros, 2022). Estes sistemas têm sido investigados com o objetivo de ajudar os utilizadores a tomar decisões mais informadas, sugerindo opções que provavelmente mais satisfarão os seus gostos numa uma ampla variedade de áreas, cada uma com as suas particularidades e especificidades, como escolher um filme para assistir, uma música para ouvir, um local para visitar, um restaurante para almoçar, entre outros (Alves,

Carneiro, Marreiros, Novais, 2019). Além disso, esses sistemas mostraram ser úteis, ajudando os utilizadores a gerir o excesso de informações, selecionando apenas o que é mais importante (Gomes, 2021).

Os primeiros exemplos de utilização comercial de sistemas de recomendação surgiram na década de 90, como uma resposta aos desafios complexos da escolha decorrentes do crescimento acelerado da internet e da indústria do comércio eletrónico. Desde então, a sua presença tem se expandido noutras áreas, incluindo media, entretenimento, turismo, saúde e muito mais. Estes sistemas funcionam recolhendo dados sobre o comportamento dos utilizadores e usando técnicas de inteligência artificial para aferir tendências e fornecer recomendações (Gomes, 2022).

O Facebook, Amazon, Youtube e Netflix, são algumas das maiores empresas que utilizam sistemas de recomendação. Uma coisa que todos estes negócios de sucesso têm em comum é que eles colocam a satisfação do cliente acima de tudo. As pessoas gostam de passar tempo no Facebook, de fazer compras na Amazon, visualizar vídeos no Youtube e de assistir a documentários, séries e filmes na Netflix. O ciclo contínuo de recomendações personalizadas nestas plataformas é fundamental para o seu sucesso. Quando se entra no Facebook, por exemplo, são apresentados artigos, vídeos ou publicações de amigos considerados mais relevantes para as respetivas pessoas. A mesma lógica se aplica à Amazon e à Netflix, que já sabem de que se gosta, sugerindo opções relevantes que são difíceis de ignorar (Sharma & Duta, 2020).

2.1.1 Técnicas de Filtragem

As técnicas de filtragem são fundamentais em sistemas de recomendação que são usados na sugestão de itens (produtos, serviços, conteúdo, etc.) para utilizadores com base nas suas preferências, histórico de compras ou comportamento numa plataforma. A filtragem permite que o sistema selecione itens relevantes e os apresente ao utilizador, aumentando assim as probabilidades de satisfação do utente.

De forma a alcançar os melhores resultados em sistemas de recomendação, é necessário escolher a técnica de filtragem mais adequada para cada caso. A seguir, vão ser descritas algumas das técnicas mais comuns de filtragem, que podem ser usadas para encontrar padrões e sugerir itens relevantes aos utilizadores.

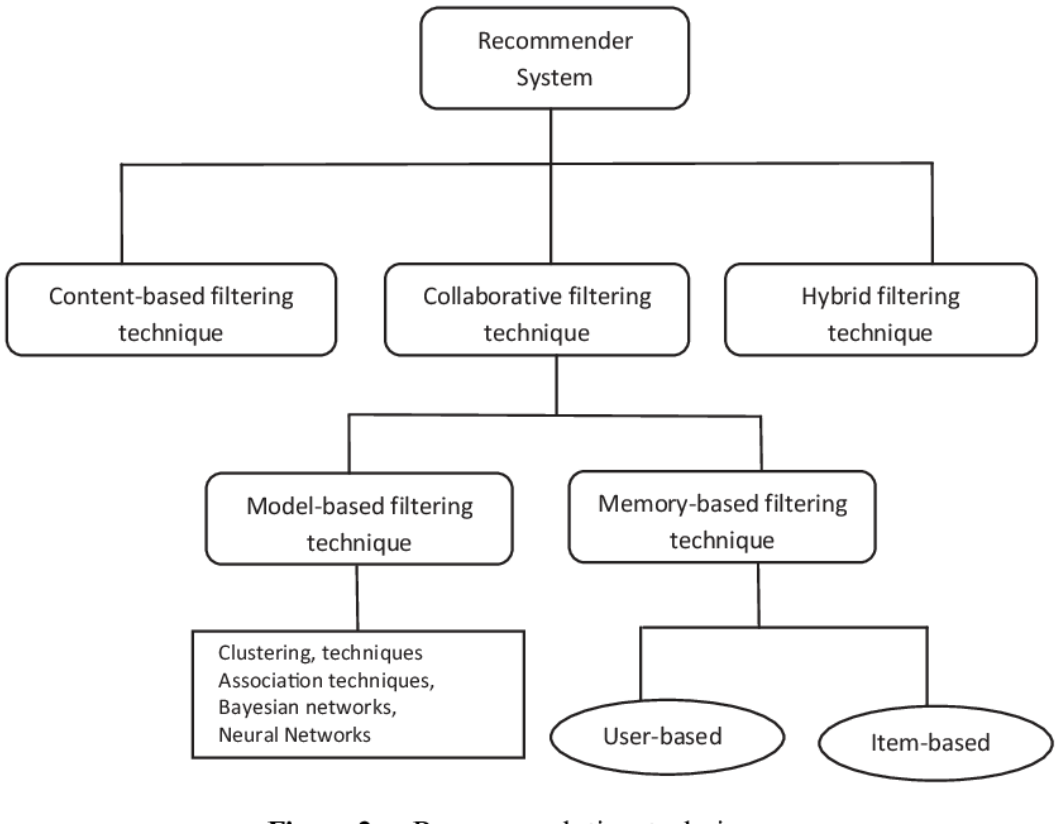


Figura 1- Técnicas de filtragem de sistemas de recomendação, retirado de Isinkaye, Folajimi and Ojokoh (2015)

2.1.1.1 Filtragem Colaborativa

O método mais frequente é baseado em dados de interação para determinar semelhanças entre utilizadores e itens (Pinelas, 2022). Esta abordagem supõe que os utilizadores possam ser classificados de acordo com as suas preferências, ou seja, irá apresentar conteúdo popular entre outros utilizadores com preferências similares às do utilizador em questão (Moreira, 2019).

Este tipo de filtragem é amplamente utilizado em muitos sistemas de recomendação em diferentes plataformas e serviços na internet, abrangendo áreas como comércio eletrónico, marketing, redes sociais, *streaming* de vídeo e gestão de relacionamento com o cliente (Pinelas, 2022).

Na generalidade, esta abordagem de recomendação costuma ter resultados positivos, uma vez que permite evitar o problema de recomendações redundantes (Abreu, 2019).

Este tipo de filtragem pode ser dividido em dois tipos, baseado em memória ou baseado em modelos (Pinelas, 2022; Moreira, 2019):

Baseada em memória: A técnica de Filtragem Colaborativa Baseada em Memória utiliza os dados no sistema para estabelecer uma relação entre os utilizadores ou itens através das suas semelhanças (Moreira, 2019). Esta abordagem é subdividida em duas categorias principais (Srifi et al., 2020):

- **Baseada em utilizadores**: A abordagem baseada em utilizadores avalia o interesse de um utilizador num item específico tendo em consideração as classificações desse item por outros utilizadores, que têm padrões comportamentais de classificação semelhantes ao do utilizador em causa (Pinelas, 2022; Moreira, 2019).

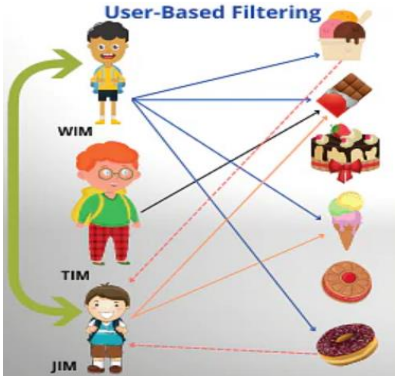


Figura 2- Filtragem baseada no utilizador, retirado de Ajitsaria (2022)

- **Baseada em itens**: A similaridade entre os itens é uma componente chave da filtragem colaborativa baseada em itens. Com base nas classificações do utilizador para produtos relacionados, ela prevê a classificação do utilizador para um determinado item. Ao usar essas estratégias, dois itens são considerados comparáveis se forem avaliados de forma semelhante por vários consumidores (Srifi, et al., 2020).

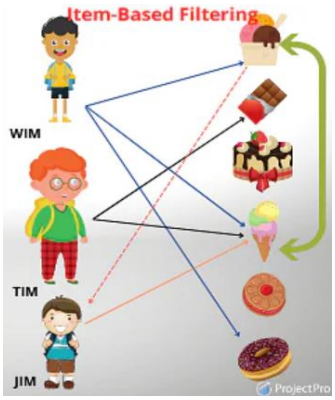


Figura 3- Filtragem baseada em itens, retirado de Ajitsaria (2022)

Sendo assim, uma etapa crucial nas abordagens de filtragem colaborativa baseadas em memória é a estimativa da similaridade utilizador/item (Moreira, 2019; Srifi, et al., 2020).

Baseada em modelos:

A ideia fundamental deste tipo de método é aplicar técnicas de aprendizagem automática para elaborar modelos de previsão (Srifi, et al., 2020). Este método usa avaliações geradas pelo utilizador para aprender um modelo preditivo que pode replicar a interação entre utilizadores e itens, levando em consideração alguns aspetos mais complexos, como preferências do utilizador e categorias de itens, em vez de usar classificações armazenadas numa base de dados para prever novas avaliações como na abordagem baseada no utilizador ou baseada em itens (Pinelas, 2022). Esta abordagem faz frente aos problemas causados pela escassez de dados e permite a geração rápida de sugestões com resultados comparáveis aos métodos baseados em memória, e geralmente gera previsões mais precisas (Moreira, 2019; Srifi, et al., 2020).

Para a tarefa de recomendação de itens, existem várias abordagens baseadas em modelos, sendo algumas das mais populares as seguintes: clustering e regras associativas (Moreira, 2019).

- **Clustering**: Particiona objetos em grupos de modo que os objetos que possuam semelhanças entre si fiquem no mesmo grupo, fazendo assim com que os objetos diferentes fiquem noutros grupos (Shen ,2008).

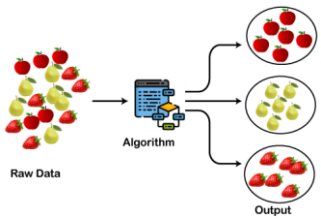


Figura 4- Filtragem baseada em Clustering, retirado de Javatpoint (n.d.)

- **Regras associativas**: Encontrar conexões ou padrões recorrentes entre conjuntos de dados é o objetivo da tarefa de associação, que procura componentes que sugiram a presença de elementos adicionais na mesma transação (Rodrigues, 2021).

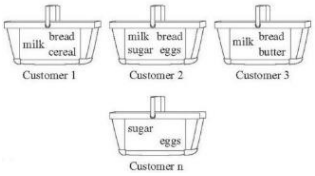


Figura 5- Filtragem baseada em regras associativas, retirado de Sarmah (2020)

2.1.1.2 Filtragem Baseada em Conteúdo

A seleção de itens baseada na técnica de filtragem baseada em conteúdo, é uma técnica amplamente utilizada. Um sistema de recomendação baseado no conteúdo, faz uso da informação que um utilizador forneceu direta ou implicitamente. É criado um perfil de utilizador utilizando essa informação, e são então dadas sugestões. O conteúdo dos itens é comparado com as preferências do utilizador ou itens que ele já gostou no passado. A ideia por trás desta técnica é que itens semelhantes serão avaliados de forma similar (Pinelas, 2022). Por exemplo, alguém que adquira o livro "The Java Programming Language" provavelmente estará interessado em livros sobre programação orientada a objetos ou na linguagem Java (Martins, 2016).

Ao contrário da filtragem colaborativa, a filtragem baseada em conteúdo não necessita de informações sobre outros utilizadores, uma vez que as sugestões são específicas para cada utilizador, facilitando assim a escalabilidade para um grande número de utilizadores (Pinelas, 2022). Ao reunir dados básicos dos utilizadores, os perfis começam a ser criados e a filtragem baseada em conteúdo pode fazer recomendações personalizadas logo no início.

2.1.1.3 Filtragem Híbrida

Esta técnica permite criar um sistema mais orientado às necessidades do utilizador sugerindo a fusão das técnicas acima descritas e dando uso às vantagens das mesmas (Martins, 2016). Usando esta técnica, a precisão das previsões de classificação melhora (Zagranovskaia & Mitura, 2021). Sendo assim, os sistemas de recomendação tornam-se mais fiáveis e os inconvenientes das metodologias anteriormente mencionadas são mitigados (Moreira, 2019).

2.2 Sistemas de Recomendação de Grupos no Turismo

As viagens de lazer tornaram-se num aspeto integral da nossa cultura. Tem-se demonstrado repetidamente que viajar é uma das atividades mais enriquecedoras que as pessoas podem fazer. Não só permite que se descubram novas culturas, pessoas e lugares, mas também tem muitos benefícios para a sua saúde mental e física, trazendo assim muitas vantagens para o bem-estar e o progresso pessoal (Brodeala, 2020).

A utilização da tecnologia móvel pela sociedade aumentou drasticamente durante os últimos quinze anos (Borrás & Valls, 2014). O setor do turismo reconheceu imediatamente os potenciais benefícios da tecnologia como os dispositivos móveis (computadores portáteis, tablets, smartphones). Com este tipo de tecnologia, é viável fornecer aos clientes serviços de Internet em qualquer altura e a partir de qualquer local, para os ajudar a planear as suas férias. Tais serviços de Internet são especialmente úteis para pessoas que visitam locais onde nunca estiveram (Khallouki, Abatal & Bahai, 2018). Como resultado, o setor das viagens tornou-se uma das áreas mais populares no meio do mundo das empresas de comércio eletrónico (Wang, Chan & Ngai, 2012).

Uma percentagem crescente de viajantes utiliza a Internet para pesquisar destinos e fazer planos de viagem. No entanto, como a Internet tem crescido em popularidade, os viajantes são por vezes sobrecarregados pela vasta quantidade de informação de viagem disponível, tornando o planeamento de viagens uma tarefa difícil. Para os visitantes, avaliar uma lista de possibilidades e escolher a que melhor satisfaz as suas exigências é um processo difícil e demorado (Borrás & Valls, 2014). Como resultado, os viajantes estão ansiosos por empregar ferramentas que os possam ajudar a tomar decisões enquanto organizam uma viagem, tais como selecionar um destino, selecionar pontos de interesse a visitar, criar um itinerário de vários dias, recomendar hotéis, e assim por diante (Moreno & Vansteenwegen, 2015). Os

sistemas de recomendação podem ajudar os visitantes a gerir a informação que lhes é acessível e a fazer melhores seleções de viagem. Por exemplo, estes sistemas podem propor lugares a visitar, atrações para ver, hotéis para ficar, e assim por sucessivamente (Wang, Chan & Ngai, 2012).

Mas, apesar disso, as táticas de recomendação não tiveram a mesma influência no turismo que noutros campos (cinema, literatura, redes sociais) (Wang, Chan & Ngai, 2012). Isto ocorre porque a criação de perfis de utilizador exatos para o turismo é uma tarefa bastante mais desafiante do que para outros campos de aplicação. Como o turismo é um passatempo consideravelmente menos comum do que ver filmes ou comprar livros, a quantidade de elementos turísticos disponíveis é também significativamente menor. Um produto turístico, por outro lado, tem uma estrutura muito mais sofisticada do que um livro ou um filme. Outra "questão" é que viajar é tipicamente feito em grupos de pessoas (cônjuge, família, amigos, colegas de trabalho) (Garcia & Onaindia, 2011). Esta última traz à superfície uma questão que é importante de ser mencionada, sistemas de recomendações de grupo, mais conhecidos por GRS.

Muitos sistemas de recomendação convencionais concentram-se apenas em modelos de utilizador único, mas, na realidade, à medida que os dispositivos móveis e as redes sociais se tornam mais predominantes, existem muitos contextos nos quais se interage em grupos, como assistir a um filme com amigos, ir ao jantar com colegas de trabalho, organizar uma viagem com amigos, etc. Sendo assim, a obtenção de recomendações para grupos é, portanto, uma preocupação que também deve ser abordada (Dara, Chowdary & Kumar, 2019).

Um sistema de recomendações de grupo recomenda itens para um grupo de utilizadores com base nas suas preferências. A qualidade da informação recomendada em diversos grupos é melhorada considerando características sociais e comportamentais dos membros do grupo, além das preferências do utilizador. Origem social, relacionamento interpessoal, confiança, interesses compartilhados e outros aspetos são alguns dos elementos que afetam os sistemas de recomendações de grupo (Dara, Chowdary & Kumar, 2019).

Embora seja difícil fazer sugestões individuais, sugerir recomendações apropriadas para um grupo é ainda mais difícil. Apoiar as sugestões do grupo é um processo altamente complexo, especialmente devido à variabilidade e interesses divergentes do grupo (Gomes, 2021).

Alguns dos fatores que aumentam a complexidade no processo de gerar recomendações em grupo, são as seguintes:

- Falta de dados sobre o grupo: os sistemas de recomendação precisam de informações sobre o grupo, como as preferências dos membros ou as relações entre eles, para fazer recomendações precisas (Gomes, 2022).
- Dificuldade de capturar as interações entre os membros do grupo: os sistemas de recomendação precisam ser capazes de entender as interações entre os membros do grupo, como as preferências compartilhadas ou as relações de influência (Gomes, 2022).
- No cenário individual, a apresentação de uma justificação de uma recomendação é um processo bastante mais simples, já que apenas existe uma pessoa que vai ser envolvida. Por outro lado, quando se fazem recomendações para grupos, a situação torna-se mais complicada, uma vez que existem vários motivos para se fazer uma dada recomendação. Essa variedade de motivos torna complicada a criação de uma explicação para um grupo de pessoas (Gomes, 2022).
- Independentemente do calibre e correção das sugestões feitas por um sistema de recomendação, estas podem ser ignoradas na mesma. Embora o acordo seja normalmente assumido no processo de tomada de decisão para sugestões individuais, as recomendações de grupo podem exigir um longo debate e negociação entre os membros para atingir o mesmo objetivo. Em sistemas onde não é possível aplicar simplificações como uma escolha parametrizada automática ou a definição de um responsável de escolha individual, é difícil obter resultados porque não é prático assumir a disponibilidade de todos os participantes em todas as circunstâncias (Gomes, 2022).

2.3 Sistemas de Recomendação Contextual

2.3.1 O que é o Contexto?

Antes de começar a falar sobre *Context Aware Reccomendation Systems* (CARS), é necessário perceber o verdadeiro significado do que significa o termo “contexto”. Segundo a definição que se encontra no dicionário Infopédia, o contexto é “conjunto de circunstâncias que

rodeiam um acontecimento". Todavia, o contexto é um conceito que tem sido estudado em diversos ramos, nomeadamente os de inteligência artificial, ciências cognitivas, linguística, filosofia, psicologia, e ciências organizacionais, por isso uma simples designação de contexto que se aplique a todas as áreas não existe (Adomavicius & Tuzhilin, 2015).

Como referido anteriormente, considerando que o contexto tem sido estudado em diversas áreas, é preciso perceber que cada uma dessas áreas vai possuir uma definição mais específica em comparação à que se encontra no dicionário. Ou seja, existem várias designações para os diferentes domínios, e mesmo dentro de subcampos específicos desses domínios (Adomavicius & Tuzhilin, 2015).

Tendo em conta que este trabalho se centra nos sistemas de recomendação e o significado de contexto é bastante alargado, o objetivo é fazer uma concentração apenas na área dos sistemas de recomendação.

Na área dos sistemas de recomendação, o contexto é definido como qualquer informação que possa ser utilizada para caracterizar a situação de qualquer pessoa, lugar, ou elemento que se pense ser significativa para a interação entre um utilizador e uma aplicação (Adomavicius & Tuzhilin, 2015).

O contexto foi inicialmente definido como a localização do utilizador, a identidade das pessoas próximas do utilizador, os objetos em redor e as alterações nestes elementos. Outros fatores foram posteriormente acrescentados a esta definição, como por exemplo a data, a estação, e a temperatura. Outras condições relevantes que acrescentam valor, são os estados físico e psicológico de um determinado utilizador. Tudo isto em conjunto cria uma condição que pode ajudar no desenvolvimento das preferências dos utilizadores considerando o contexto (Kulkarni & Rodd, 2020).

Abordagens que considerem o contexto podem oferecer sugestões eficazes, mediante o contexto em que o utilizador se encontra inserido. Por exemplo, pode ser necessário ajustar o itinerário de um turista que planeava visitar um museu que estava inicialmente programada para um certo dia, uma vez que o visitante pode estar demasiado distante e ter pouco tempo para lá chegar (Borrás & Valls, 2014).

Com este exemplo, dá-se início à introdução do tema de turismo e como é que CARS ajudam a facilitar a vida aos turistas.

2.3.2 Sistemas de recomendação contextual no turismo

Nos últimos quinze anos assistiu-se a um grande aumento na utilização de tecnologias móveis como sistemas de posicionamento (GPS) e telemóveis no setor do turismo (Borrás, Moreno & Valls, 2014; van Setten, Pokraev & Koolwaaij, 2004). Com isto, tornou-se possível oferecer serviços em tempo real às pessoas, quando e onde quer que elas estejam (Setten, Pokraev & Koolwaaij, 2004). Assim, os sistemas de e-Turismo proporcionam uma boa oportunidade para serviços móveis oferecerem recomendações baseadas nas preferências e no contexto atual de um utilizador (Borrás, Moreno & Valls, 2014).

Estes serviços de Internet são particularmente úteis para pessoas que visitam novas áreas. Sendo que uma grande parte dessas pessoas são turistas, estes frequentemente não fazem ideia quais os serviços públicos, lojas, ou museus que estão disponíveis para eles. Pode ser bastante aborrecido escolher onde ir, especialmente em áreas turísticas, devido à grande quantidade de informação que existe na internet. Para resolver esse problema, um viajante pode utilizar sistemas de recomendação para identificar locais que se adequem aos seus interesses (Setten, Pokraev & Koolwaaij, 2004).

Mas uma parte significativa das abordagens dos sistemas de recomendação concentra-se em propor as recomendações mais pertinentes sem ter em conta qualquer informação contextual, como por exemplo o tempo, lugar, e outros fatores. Por outras palavras, os sistemas convencionais de recomendação só funcionam com aplicações que têm dois tipos de entidades: pessoas e objetos, e não contextualizam estas entidades enquanto fazem sugestões (Adomavicius & Tuzhilin, 2015).

No entanto, para algumas aplicações, como as que permitem criar pacotes de férias, ter em conta apenas pessoas e itens não é suficiente. É crucial incluir informação contextual no processo de recomendação a fim de propor recomendações aos utilizadores em determinadas situações. Isto é particularmente benéfico quando os visitantes já estão no local e dispostos a utilizar os seus dispositivos móveis para personalizar as suas férias no momento (Borrás & Valls, 2014). Por exemplo, utilizando o contexto da localização do utilizador, um sistema de recomendação de viagens forneceria um ponto de interesse (Point of Interest, POI) que consideraria os horários de abertura e encerramento do mesmo, bem como o tempo de viagem entre POI e a localização (Adomavicius & Tuzhilin, 2015).

2.3.2.1 Definição e obtenção da informação contextual

Como já foi referido, os CARS têm em consideração diferentes atributos contextuais, mas nem toda a informação contextual é obrigatoriamente relevante ou útil para efeitos de recomendação (Kulkarni & Rodd, 2020; Adomavicius & Tuzhilin, 2015). A relevância de um tipo particular de informação contextual pode ser definida usando uma variedade de métodos, sendo os mais frequentes os seguintes (Adomavicius & Tuzhilin, 2015):

- Manualmente: utilizando os conhecimentos de domínio dos criadores do sistema de recomendação ou de um especialista de mercado num determinado domínio de aplicação.
- Automaticamente: utilizando numerosos procedimentos de seleção de características existentes da aprendizagem de máquinas, extração de dados, e estatísticas, com base nos dados de classificação existentes durante a fase de pré-processamento de dados.

A obtenção de informações contextuais consiste no processo de reunir informações relevantes sobre uma situação ou problema específico para ajudar a compreender melhor a situação e tomar decisões informadas. Existem várias maneiras de obter informações contextuais, e aqui estão algumas delas (Adomavicius & Tuzhilin, 2015):

- Expressamente: consiste em abordar diretamente indivíduos pertinentes e outras fontes de dados contextuais e obtendo explicitamente esta informação através de inquéritos ou outros métodos diretos. Por exemplo, uma aplicação pode solicitar que um utilizador preencha um formulário ou responda a uma série de perguntas para perceber a melhor a situação contextual do inquirido.
- Subentendidamente: consiste em detetar situações contextuais de forma implícita. A fonte da informação contextual implícita é acessível diretamente nestas situações, e os dados são recolhidos a partir dela sem a necessidade de se envolver com o utilizador ou outras fontes de informação contextual. Por exemplo, a alteração da localização de um utilizador pode ser detetada através de um sistema externo sem que o utilizador tenha alguma interação com a aplicação.

2.3.2.2 Paradigmas

Existem vários paradigmas para incorporar informação contextual nos CARS. Existem três abordagens principais, nomeadamente a pré-filtragem, pós-filtragem, e modelação contextual (Livne, et al., 2015).

- **Pré-filtragem**: Na pré-filtragem contextual, o contexto determina quais os dados que são escolhidos ou criados para um determinado contexto. Ou seja, a informação sobre a situação contextual é utilizada para escolher apenas o conjunto de dados pertinentes, e as classificações são então projetadas sobre os dados escolhidos utilizando qualquer sistema de recomendação (Adomavicius & Tuzhilin, 2015). Nesta abordagem, o contexto atua como um filtro para uma seleção adequada de dados de classificação. Um exemplo de um caso em que um sistema de recomendação utiliza uma pré-filtragem é o seguinte: se uma pessoa quiser visitar um POI num domingo, apenas os dados de classificação de domingo vão ser utilizados para fazer uma recomendação. Um dos grandes benefícios deste tipo de filtragem, é que este pode ser utilizado em várias técnicas de recomendação (Adomavicius & Tuzhilin, 2015).
- **Pós-filtragem**- No paradigma da pós-filtragem contextual, a informação do contexto é inicialmente desconsiderada, e as classificações são previstas no conjunto completo de dados utilizando qualquer sistema de recomendação. Depois disso, a informação contextual vai ser utilizada de forma que a lista de recomendações de cada utilizador seja ajustada. As recomendações que sejam inadequadas numa situação particular podem ser filtradas ou a classificação das sugestões da lista pode precisar de ser alterada. Um exemplo de um caso em que um sistema de recomendação utiliza uma pós-filtragem é o seguinte: se uma pessoa quiser visitar um POI num domingo, todos os POI que foram incluídos na recomendação que não podem ser visitados num domingo vão ser excluídos. À semelhança da pré-filtragem contextual, a estratégia de pós-filtragem tem a principal vantagem de permitir a utilização de várias técnicas de recomendação (Adomavicius & Tuzhilin, 2015).
- **Modelação Contextual**- Na modelação contextual, a técnica de modelagem inclui explicitamente dados contextuais como parte da estimativa de classificação. Enquanto as técnicas de pré filtragem e pós filtragem costumam usar funções de recomendação tradicionais, a técnica de modelação contextual produz funções de recomendação multidimensionais, que são basicamente modelos preditivos (Adomavicius & Tuzhilin, 2015).

2.4 Virtual Pet

A *gamification* de destinos turísticos é uma forma útil de aumentar o envolvimento dos utilizadores com uma aplicação. A utilização de animação 3D para promover destinos ajuda a atrair potenciais visitantes para um destino, ao melhorar as suas interações com o destino turístico. Daí, surge a ideia de implantar *virtual pets* como companhia de viagem utilizando uma aplicação móvel. Tendo em conta que pessoas de todas as idades jogam jogos e usufruem de aplicações, a noção de um *virtual pet* é agora especialmente relevante. *Os virtual pets* são criados digitalmente de maneira a que se assemelhem a animais de estimação que podem estabelecer ligações emocionais com os seus donos. Em geral, os animais de estimação virtuais através de uma aplicação móvel podem criar envolvimento com destinos turísticos, tornando-os uma ferramenta de marketing eficaz. As interações com um animal de estimação virtual através de uma aplicação móvel podem desencadear o desejo de alcançar experiências de viagem num destino específico, o que pode ser benéfico para destinos turísticos (Thirumaran, 2021).

2.5 Trabalhos Relacionados

Nesta subsecção são apresentados alguns dos trabalhos já realizados na área de sistemas de recomendação contextual. Esta é fundamental para perceber como é que a investigação nesta área tem evoluído ao longo do tempo, quais os principais desafios enfrentados e quais são as tendências atuais na área. Na lista dos trabalhos selecionados para análise estão o Compass (Setten, Pokraev & Koolwaaij, 2004), Intrigue (Ardissono, et al., 2003), Mobile Tourism Recommendation System (MTRS) (Gavalas & Kenteris, 2011), o ReRex (Baltrunas, et al., 2011) e POST-VIA 360 (Colomo-Palacios, et al., 2017)

2.5.1 Compass

Esta aplicação oferece um sistema de informação baseada em mapas aos turistas, de acordo com seu contexto e preferências específicas. Por exemplo, um turista interessado em história e arquitetura recebe informações sobre monumentos importantes nas proximidades, enquanto aqueles que procuram acomodações são fornecidos com uma lista de hotéis e parques de campismo na cidade e nos arredores que correspondem às suas preferências (Setten, Pokraev & Koolwaaij, 2004).

O Compass, após ser ligado, apresenta um mapa mostrando a localização atual do utilizador, que é determinada por meio da funcionalidade GPS. É importante salientar que o Compass é um sistema de informação que apenas tem em conta a localização do utilizador para fornecer serviços de informação *context-aware*. Dependendo do perfil e do objetivo do utilizador, uma seleção de edifícios próximos, amigos e outros objetos é mostrada no mapa e numa lista. O mapa e os objetos mostrados são atualizados quando o utilizador se desloca ou o seu perfil ou objetivos mudam (Setten, Pokraev & Koolwaaij, 2004).

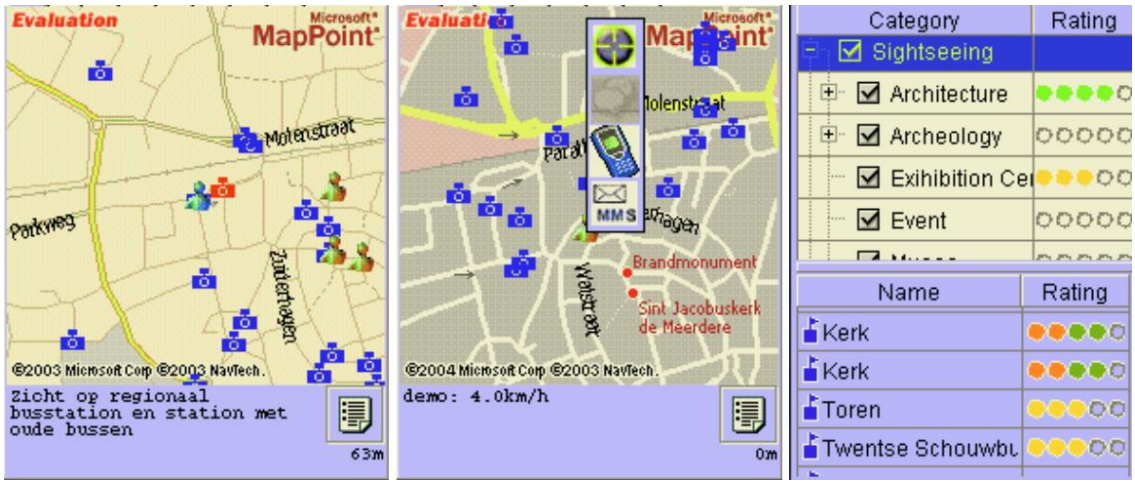


Figura 6- Capturas de ecrã da aplicação COMPASS: pontos turísticos próximos do utilizador no mapa, retirado de Setten, Pokraev & Koolwaaij (2004)

2.5.2 Intrigue

O Intrigue é um serviço que fornece informações sobre atrações e serviços turísticos, como alojamento e restaurantes, em uma determinada área geográfica. O seu principal objetivo é auxiliar o utilizador na organização de um passeio, considerando as possíveis preferências conflituosas de um grupo que viaja junta, como por exemplo, uma família com adultos e crianças. Concentra-se no planeamento e agendamento de uma visita personalizada. Para isso, são aplicadas estratégias de personalização que recomendam as atrações que melhor se adequam às preferências do grupo turístico, planeamento e agendamento personalizado (Ardissono, et al., 2003).

A aplicação, após ser ligada, apresenta uma lista de sugestões de pontos de interesse de acordo com a localização atual do utilizador e os seus interesses pessoais. Em relação aos serviços de informação *context-aware*, o Intrigue é um sistema que somente tem em conta a localização do utilizador (Ardissono, et al., 2003).

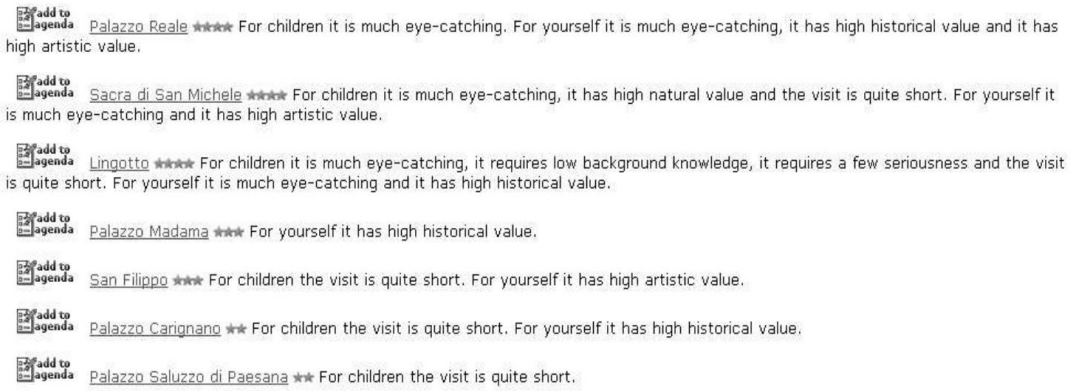


Figura 7- Recomendação de atrações no Intrigue, retirado de Ardissono, et al. (2003).

2.5.3 MTRS

O MTRS atribui pesos diferentes aos conteúdos fornecidos pelos turistas que visitam efetivamente um POI em comparação com as classificações submetidas pelos utilizadores da web. Desta forma, o MTRS capta avaliações e classificações de utilizadores sensíveis ao contexto e utiliza esses dados para fornecer recomendações a outros utilizadores com interesses semelhantes. Além disso, MTRS fornece vários serviços de recomendação personalizados a utilizadores, tendo em conta informação contextual como a localização do utilizador, a hora atual, as condições meteorológicas e o histórico de mobilidade do utilizador (Gavalas & Kenteris, 2011).

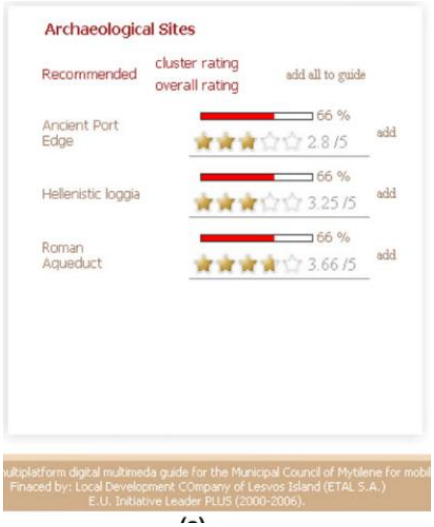


Figura 8- Recomendações personalizadas para o utilizador Web MTRS retirado de Gavalas & Kenteris (2011)

2.5.4 ReRex

ReRex é uma aplicação de planeamento de viagens que visa recomendar pontos de interesse com base no contexto e nas preferências dos utilizadores. As informações contextuais que o sistema tem em conta são diversas (distância em relação a um POI, dia da semana, hora do dia, orçamento, situação climatérica) e a influência que estas têm nas recomendações podem ser decididas pelo utilizador como demonstra a figura 9. A aplicação apresenta as recomendações geradas por um modelo preditivo e justifica as recomendações. Além de recomendar POI turísticos com base no contexto, a aplicação ajuda na criação de um itinerário completo e na modificação do itinerário de acordo com as circunstâncias e imprevistos que possam surgir durante a viagem. Por exemplo, o sistema pode justificar a recomendação de um determinado POI (como o Museu Oetzi) porque o utilizador está a viajar com crianças. Além disso, a aplicação pode notificar o utilizador de que uma condição contextual (como o clima) é propensa a mudar e, portanto, rever as recomendações para o utilizador (Baltrunas, et al., 2011).

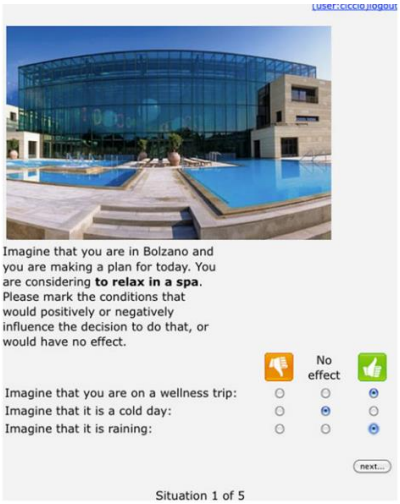


Figura 9- Ferramenta de inquérito baseado na Web para adquirir relevância de contexto, retirado de Baltrunas, et al. (2011).

2.5.5 POST-VIA 360

O POST-VIA 360 é uma plataforma capaz de fazer recomendações aos viajantes. Quando a aplicação é ativada pelo turista, o sistema recorre a inquéritos para recolher informações sobre as preferências do utilizador. Para além disso, faz uso da tecnologia GPS para detetar os movimentos e interações de um utilizador num determinado local. O POST-VIA 360 faz recomendações que são adaptadas aos gostos e à situação do viajante.

Por exemplo, um turista com preferência por gastronomia, pode obter recomendações de restaurantes, cafés e bares na zona que está a visitar, ao passo que um visitante interessado em cultura pode obter recomendações de museus, igrejas, etc. Mediante as movimentações do utilizador pela cidade, o sistema é capaz de detetar as mesmas e modificar as suas recomendações em tempo real.

2.6 Tecnologias Existentes

Neste capítulo é realizada uma análise às tecnologias que poderão ser potencialmente utilizadas e que foram solicitadas pelos responsáveis do projeto. Esta seção apresenta uma comparação entre elas, que ajuda a definir a melhor estratégia para a implementação e integração do contexto no sistema.

2.6.1 Frameworks e Ambientes de Desenvolvimento

2.6.1.1 Java

Java é uma linguagem de programação orientada a objetos e uma plataforma de software amplamente utilizada que funciona em diversos dispositivos, incluindo computadores portáteis, dispositivos móveis, consolas de jogos, dispositivos médicos, e muitos outros. Uma grande vantagem de desenvolver software com Java é a sua portabilidade, o que significa que uma vez que se tenha escrito o código para um programa Java num computador portátil, é muito fácil mover o código para um dispositivo móvel. Como resultado, Java é uma das linguagens mais populares para o desenvolvimento de software. Quando se trata de escolher uma linguagem de programação, existem várias razões técnicas para considerar Java, incluindo a sua interoperabilidade, escalabilidade e adaptabilidade (IBM, n.d.).

Para se poder desenvolver e correr o código Java, é necessário recorrer um pacote de ferramentas que o possa fazer. Esse pacote é denominado de *Java Development Kit* (JDK). O JDK é constituído pelo *Java Runtime Environment* (JRE) e pelas *Java Development tools* (JDT). O JRE é um pacote que fornece um ambiente para apenas correr o programa numa máquina. As JDT são compostas por várias ferramentas com, por exemplo um compilador, um *debugger* e outras. Uma das partes mais fundamentais do JDK e do JRE é a Java Virtual Machine (JVM), uma vez que esta é a responsável pela execução do código linha a linha (Lotfi, 2022).

Para o desenvolvimento de microsserviços, uma das *frameworks* mais utilizadas na área do Java, é o Spring. Esta oferece uma grande variedade de bibliotecas e recursos, permitindo que

os desenvolvedores escolham os recursos que melhor se adequam às necessidades do projeto. Desta forma, o processo de desenvolvimento de aplicações que recorrem à arquitetura de microsserviços fica personalizável. Por exemplo, uma das principais vantagens desta *framework* é que esta fornece modelos para as tecnologias *Java database connectivity* (JDBC), Hibernate, *Java Persistence API* (JPA), etc. Desta maneira, não há necessidade de escrever demasiado código (JavatPoint, n.d.; Spring, n.d.).

2.6.1.2 Node.js

Node.js é um ambiente de desenvolvimento multiplataforma *open-source* que permite executar aplicações web fora do navegador do cliente. Este ambiente permite criar aplicações web do lado do servidor e é muito útil para aplicações de dados intensivos, uma vez que utiliza um modelo assíncrono, orientado por eventos (Node.js, n.d.). Existem várias razões para optar por este ambiente, tais como a sua rapidez de execução, a extensa quantidade de pacotes disponíveis no Node Package Manager (NPM), a facilidade da sua importação, e a sua compatibilidade com várias plataformas. De realçar que o Node.js não precisa de um compilador, já que o código JavaScript é executado diretamente pelo interpretador do Node.js, sem a necessidade de compilar o código para um formato executável antes da execução (Sufiyan, 2023).

2.6.1.3 .NET

O .NET é uma plataforma de desenvolvimento *open-source* que permite a criação de aplicações *desktop*, *web* e móveis. Esta tolera a utilização de várias linguagens de programação, tais como C#, F#, ou Visual Basic. O sistema .NET inclui ferramentas, bibliotecas e linguagens que suportam desenvolvimento de software moderno, escalável e de alta performance. Independentemente do tipo de linguagem utilizada, o código é executado nativamente em qualquer sistema operativo compatível (Microsoft, n.d.).

Para desenvolver e compilar aplicações .NET, é necessário instalar o Software Development Kit (SDK) do .NET numa máquina. O SDK do .NET é um conjunto de ferramentas que inclui o compilador .NET, bibliotecas de classes e outras ferramentas que permitem desenvolver, construir e fazer debug de aplicações .NET (Gewarren, n.d.).

Quando se trata de arquiteturas de microsserviços, o .NET oferece várias vantagens significativas que tornam a sua utilização uma escolha popular. Algumas delas são: o facto desta plataforma incluir variadas ferramentas que facilitam a utilização da plataforma, como

por exemplo, utilizando o pacote do Visual Studio, os utilizadores podem testar o código de forma eficiente e rápida; as aplicações .NET serem bastante rápidas, exigindo um menor tempo de computação e executarem de forma eficiente tarefas no servidor, como o acesso a base de dados; a plataforma ser *open-source* o que significa que existe uma comunidade bastante ativa que mantém o software constantemente atualizado (Russo, 2003b).

2.6.2 Tecnologias de Armazenamento de Dados

SQL e NoSQL diferem quanto a serem relacionais (SQL) ou não relacionais (NoSQL), se os seus esquemas são pré-definidos ou dinâmicos, como são escalados e o tipo de dados que incluem. Decidir entre um tipo ou outro não é uma decisão fácil, sendo que é preciso fazer uma análise aprofundada do que cada tipo de base de dados oferece (IBM, n.d.).

2.6.2.1 SQL

O Structured Query Language (SQL) é uma linguagem de programação que tem sido usada ao longo de vários anos com o intuito de gerir informação em sistemas de gestão de bases de dados relacionais. Estas são capazes de gerir grandes quantidades de dados e lidar com operações de consultas bastante complexas (IBM, n.d.).

O esquema da base de dados SQL organiza os dados de forma relacional, utilizando tabelas com colunas ou atributos. Este esquema segue um formato rigoroso e, por isso mesmo, os dados necessitam de estar de acordo com esse formato requerendo assim, a organização e estruturação de dados antes de começar com a base de dados SQL (IBM, n.d.).

Os sistemas de gestão de bases de dados relacionais que dão uso a SQL, têm de respeitar as quatro propriedades mais conhecidas pelo acrónimo ACID (IBM, n.d.).

- Atomicidade: Todas as transações devem ter sucesso ou falhar completamente e não podem ser deixadas parcialmente completas, mesmo em caso de falha do sistema.
- Consistência: A base de dados deve seguir regras que validem e previnam a corrupção em cada passo.
- Isolamento: As transações simultâneas não se podem afetar umas às outras.
- Durabilidade: As transações são finais, e mesmo a falha do sistema não pode "fazer retroceder" uma transação completa.

Seguir as regras deste acrónimo garante a integridade dos dados nas operações realizadas e que uma base de dados relacional seja confiável (IBM, n.d.).

Algumas das bases de dados relacionais mais usadas incluem o MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL e SQLite. Cada uma destas bases de dados tem vantagens e desvantagens, e para escolher a mais adequada é necessário estudar as exigências da aplicação (IBM, n.d.).

2.6.2.2 NoSQL

O termo NoSQL refere-se às bases de dados que não são relacionais, ou seja, que permitem uma estruturação diferente das bases de dados SQL (tabelas e colunas). Ao contrário de SQL os sistemas NoSQL dão a possibilidade de trabalhar com vários tipos de estruturas. Uma vez que estes tipos de sistemas possibilitam um esquema dinâmico para dados não estruturados, geralmente não é necessário fazer um pré planeamento dos dados, permitindo uma maior facilidade na alteração dos mesmos. Normalmente, as bases de dados não relacionais são de um dos seguintes tipos (IBM, n.d.):

- **Orientada para colunas**, onde os dados são armazenados em células agrupadas num número praticamente ilimitado de colunas em vez de linhas.
- **Armazéns de valores-chave**, que utilizam um mapa como modelo de dados. Este modelo representa os dados como uma coleção de pares de valores-chave.
- **Armazéns de documentos**, que utilizam documentos para guardar os dados em formatos padrão, incluindo XML, YAML, JSON e BSON. Um benefício é que os documentos dentro de uma única base de dados podem ter diferentes tipos de dados.
- **Bases de dados em grafo**, que representam dados num grafo que mostra como diferentes conjuntos de dados se relacionam uns com os outros.

A maior parte dos sistemas de bases de dados não relacionais segue o teorema CAP. Este afirma que apenas duas das três propriedades do acrónimo CAP descritas em baixo são garantidas de cada vez (IBM, n.d.):

- ***Consistency*:** Cada pedido recebe ou o resultado mais recente ou um erro.
- ***Availability*:** Cada pedido tem um resultado não desacertado.
- ***Partition tolerance*:** Quaisquer atrasos ou perdas entre nós não interrompem o funcionamento do sistema.

As bases de dados não relacionais mais usadas incluem o MongoDB, Cassandra, Redis e Neo4j. Cada uma destas bases de dados tem benefícios e limitações, e para escolher a mais propícia é necessária uma análise dos requisitos da aplicação (IBM, n.d.).

2.6.3 Escolha das Tecnologias

Ao falar das tecnologias utilizadas no trabalho, é importante salientar que as decisões em relação a essas ferramentas foram determinadas pelo trabalho já existente, e, portanto, tomadas por outros elementos que trabalharam previamente no desenvolvimento do projeto. No entanto, essas escolhas foram feitas com base em diversos fatores, tais como os requisitos do projeto, as necessidades dos utilizadores e as melhores práticas do mercado. Além disso, foram levados em consideração aspetos como a segurança e a quantidade de informação e documentação das tecnologias escolhidas. De seguida, é explicado o motivo pelo qual certas tecnologias foram escolhidas.

2.6.3.1 Framework de Desenvolvimento

O .NET já existe no mercado há vários anos e é considerada uma plataforma fiável pela grande maioria da comunidade na construção de aplicações que fazem uso intensivo de recursos. O .NET fornece uma grande quantidade de ferramentas e bibliotecas para o desenvolvimento de aplicações e oferece múltiplas opções para alojamento de aplicações ou websites, como por exemplo Microsoft Azure, AWS, Google Cloud Platform, e Heroku. Para além disso, esta oferece uma grande comunidade de utilizadores e documentação verificada pela Microsoft tornando-se assim uma plataforma digna de confiança. Outra grande vantagem é que o .NET vem equipado com autenticação Windows integrada e características de configuração de aplicações que ajudam a construir aplicações seguras (Turing, n.d.).

O Node.js é uma plataforma relativamente recente, mas tem ganho muita popularidade nos anos recentes, com grandes empresas a optar por esta para a realização dos seus projetos. A plataforma beneficia de uma grande capacidade de lidar com várias tarefas de forma eficiente e rápida devido ao seu modelo assíncrono *single-thread* e um motor de alta velocidade Chrome V8. Node.js é composto por uma grande comunidade que está em constante crescimento onde ideias e problemas são discutidos. Um lado bastante negativo da plataforma é que tem opções limitadas para alojamento das aplicações (Turing, n.d.).

O Spring é baseado em java e por isso encontra-se implementado em várias aplicações, por causa da facilidade de implementar e adaptar em qualquer sistema operativo, tendo assim uma grande independência na utilização de plataformas. A nível de performance, o Spring encontra-se numa situação considerável, porém inferior tanto em relação ao .NET como ao Node.js (Gomes, 2022).

Pesando todos os a favores e os contras a decisão feita previamente, foi a de optar pelo .NET, principalmente pelas suas caraterísticas de segurança e pela questão das opções de alojamento (Gomes, 2022; Rodrigues, 2021).

2.6.3.2 Decisão da Tecnologia de Armazenamento de Dados

Nas bases de dados relacionais, os dados necessitam de estar altamente estruturados e precisam ser armazenados em tabelas. As bases de dados relacionais são geralmente a melhor opção para lidar com dados estruturados. Estas também se destacam na facilidade de lidar com a execução de consultas complexas que envolva várias tabelas, colunas e relações devido à sua linguagem bem documentada e convencionada. Se os dados precisam ser altamente consistentes e é necessário garantir que cada transação seja atómica, consistente, isolada e durável (ACID), uma base de dados relacional é geralmente a melhor opção (IBM, n.d.).

Por outro lado, nas bases de dados não relacionais os dados não precisam de estar estruturados, tendo assim uma maior capacidade de flexibilidade. Outra vantagem, é que uma base de dados deste tipo possui alta disponibilidade, ou seja, os dados estão sempre disponíveis para acesso (IBM, n.d.).

Ao investigar as duas possíveis soluções, o tipo de base de dados escolhido acabou por ser a base de dados com estrutura relacional. Isto aconteceu, uma vez que estas são consideradas mais fáceis de usar, possuem uma grande variedade de ferramentas, existe muita documentação e o projeto é composto por objetos com várias relações entre si, beneficiando assim de uma estrutura com tabelas e colunas (Rodrigues, 2021).

Para a escolha da base de dados relacional foram consideradas a PostgresSQL e o MySQL. PostgresSQL é uma base de dados relacional orientada a objetos, oferecendo uma maior gama e sofisticação nos tipos de dados, enquanto que MySQL é uma base de dados puramente relacional. Pelo lado negativo, é mais difícil trabalhar com PostgresSQL por causa

da sua maior complexidade Ambas são conhecidas por serem das bases de dados mais rápidas presentes no mercado. Enquanto a PostgresSQL é mais rápida com comandos de leitura, o MySQL lida melhor com operações de leitura-escrita e *queries* complexas. Em relação à utilização por parte da comunidade, ambas são das bases de dados mais utilizadas do mercado com um grande número de utilizadores, o que leva a um grande apoio por parte de todos os que a usam na resolução de dúvidas e problemas (Ravoof, 2023).

A escolha final entre PostgreSQL e MySQL acabou por pender para PostgresSQL, já que esta apresenta um sistema bastante avançado, tem maior sofisticação no tipo de dados e um tempo de resposta reduzido quando se fazem consultas complexas em bases de dados extensas (Rodrigues, 2021).

3 Análise de Valor

3.1 Oportunidade

Os sistemas de recomendação estão cada vez mais a aparecer como uma ferramenta que ajuda os utilizadores a receberem recomendações personalizadas (Alves, Gomes, Rodrigues, Carneiro, Novais & Marreiros, 2022). Estes sistemas vêm sendo investigados com o objetivo de ajudar os utilizadores a tomar decisões mais informadas, sugerindo opções que mais provavelmente satisfarão os seus gostos numa uma ampla variedade de áreas, cada uma com suas as próprias questões únicas, como escolher um filme para assistir, uma música para ouvir, um local para visitar, um restaurante para almoçar, entre outros (Alves, Carneiro, Marreiros, Novais, 2019). Além disso, esses sistemas mostraram ser úteis, ajudando os utilizadores a gerir o excesso de informações, selecionando apenas o que é mais importante (Gomes, 2021).

Na figura 10 é possível observar o crescimento da publicação de artigos sobre sistemas de recomendação.

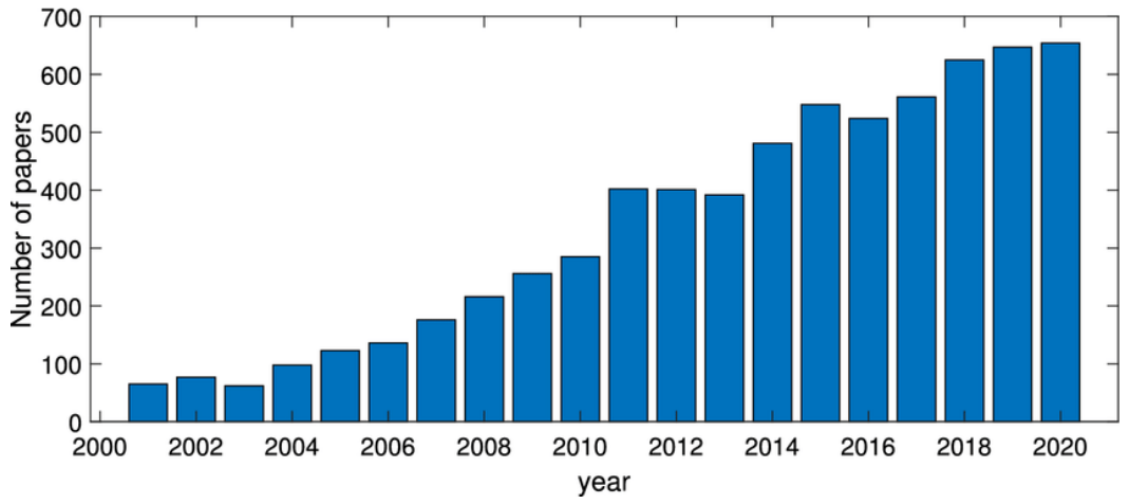


Figura 10- Número de artigos publicados sobre sistemas de recomendação (Disponível em: https://www.researchgate.net/journal/Knowledge-and-Information-Systems-0219-3116)

Porém, quando grupos de pessoas estão envolvidas, surgem conflitos de preferências e heterogeneidade. Para ajudar neste tipo de situações, em especial no turismo, estão a ser propostos Sistemas de Recomendação para Grupos (GRS), que se usarem as capacidades de um dispositivo móvel, como o contexto em que o utilizador se encontra, podem melhorar drasticamente as recomendações e a sua experiência.

No entanto, são poucos os GRS encontrados na literatura que se focam nos interesses pessoais dos turistas aliados ao seu contexto. Sendo assim, com um crescimento sustentado do turismo, e com um mundo cada vez mais exigente nas suas necessidades, os sistemas de recomendações de grupo contextuais representam uma grande oportunidade no mercado que vale a pena explorar.

Na figura 11 é possível observar que o número de artigos sobre sistemas de recomendação na área do turismo é menor que noutros ramos.

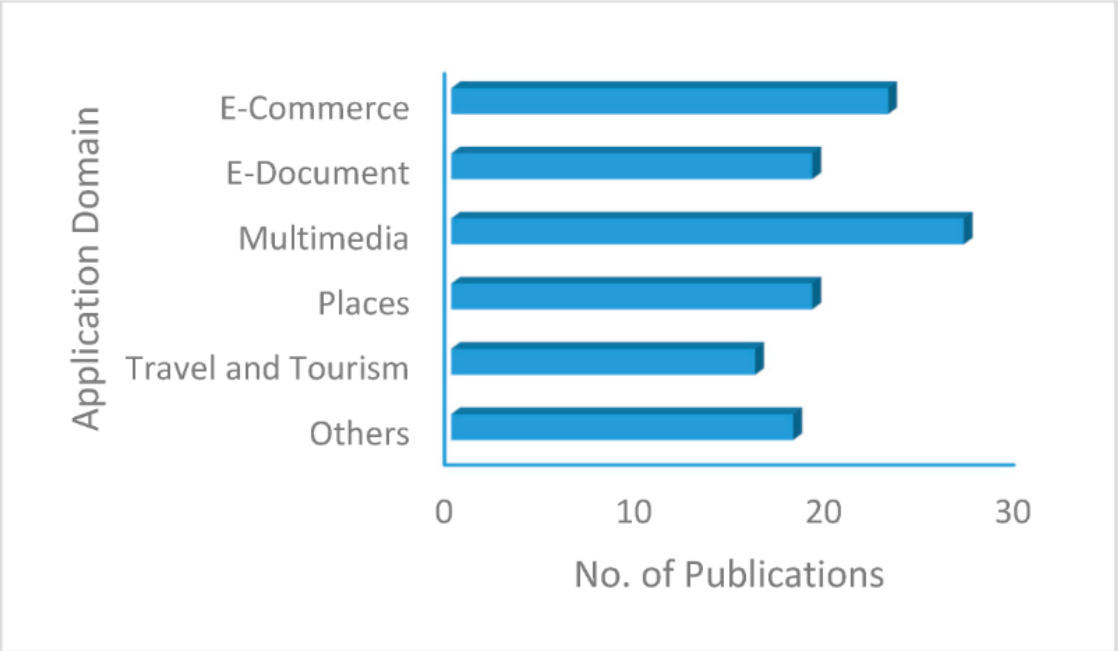


Figura 11- Classificação de artigos por domínio da aplicação (Disponível em: Imagem11- https://www.mdpi.com/2076-3417/7/12/1211)

Na figura 12 é possível perceber que o número de artigos sobre sistemas de recomendação contextuais está a crescer, mas continua a ser pouco explorado quando comparado com a totalidade de artigos publicados sobre sistemas de recomendação (figura 10)

.

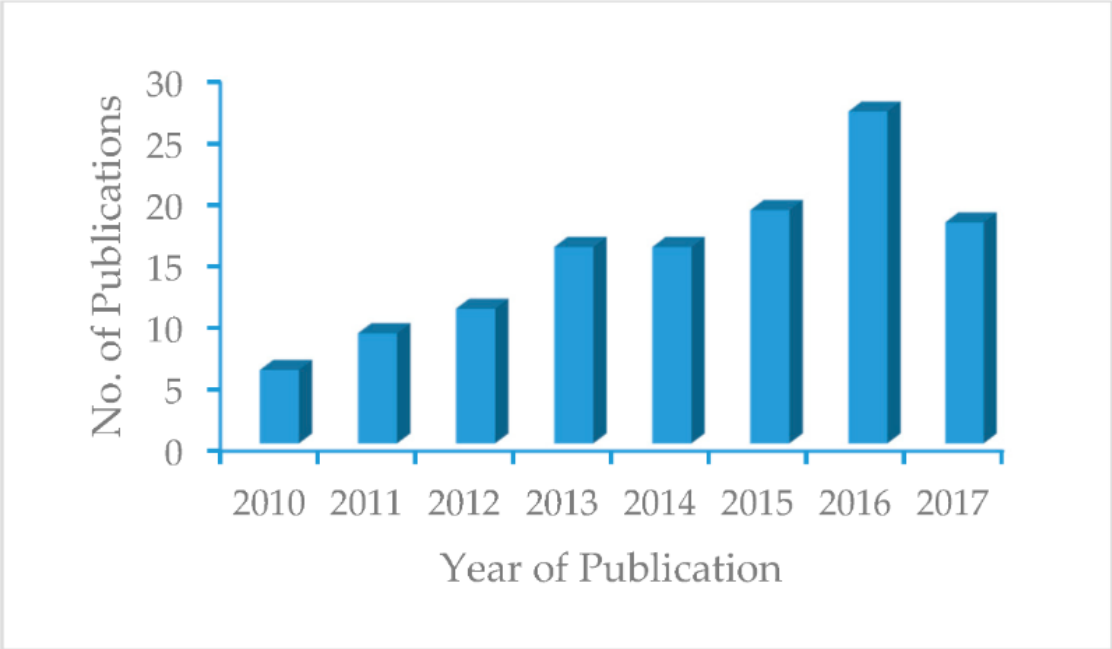


Figura 12- Número de artigos publicados sobre CARS (Disponível em: https://www.mdpi.com/2076-3417/7/12/1211)

Para analisar a oportunidade de implementar um sistema de recomendação contextual para grupos, recorreu-se a uma análise SWOT. Esta ferramenta permite identificar as forças, fraquezas, oportunidades e ameaças do desenvolvimento de um negócio. Desta maneira, é possível perceber se é viável construir este tipo de sistema de recomendação.

Na figura 13, encontra-se a análise SWOT para o sistema de recomendação contextual para grupos.

Figura 13- Análise SWOT

3.2 Proposta de Valor

Para fazer a proposta de valor para a solução apresentada, recorreu-se ao modelo criado por Osterwalder. Este diz que o *Value Proposition Canvas* é uma ferramenta de modelo de negócio que ajuda a assegurar que o produto ou serviço de uma empresa se posiciona em torno dos valores e necessidades dos clientes (Pereira, 2021).

O *Value Proposition Canvas* é composto por dois blocos - Proposta de Valor e Segmento do Cliente. São o núcleo do modelo de negócio porque se concentram em "O quê" e "A quem" (Pereira, 2021).

O Segmento do Cliente encontra-se subdividido em *Jobs-To-Be-Done*, *Pains* e *Gains* (Pereira, 2021). De seguida serão apresentados os termos mencionados:

Jobs-to-be-done

Isto é sobre o que o cliente está a tentar fazer. Tem de incluir todas as tarefas que os clientes estão a tentar realizar, os problemas que estão a tentar resolver, e as necessidades que querem satisfazer (Pereira, 2021).

Pains

Este engloba tudo o que aborrece o cliente enquanto este desempenha as suas funções, tais como experiências e emoções negativas, desafios, etc (Pereira, 2021).

Gains

São todos os benefícios que o cliente espera, sejam eles funcionais, emocionais, sociais ou financeiros (Pereira, 2021).

Já a Proposta de Valor encontra-se subdividida em *Products and Services, Gain Creators* e *Pain Relievers* (Pereira, 2021). Em seguida, serão expostos os termos citados:

Products & Services

Inclui todos os produtos e serviços que vão ser entregues ao cliente (Pereira, 2021).

Gain Creators

Consiste em identificar quais são os benefícios que o seu produto traz, e se os desejos e expectativas do seu cliente são alcançados (Pereira, 2021).

Pain Relievers

Descreve como o produto alivia as dores do cliente (Pereira, 2021).

Na figura 14 está representado o canvas que contém a Propopsta de Valor e o Segmento do Cliente de acordo com o projeto que está a ser desenvolvido.

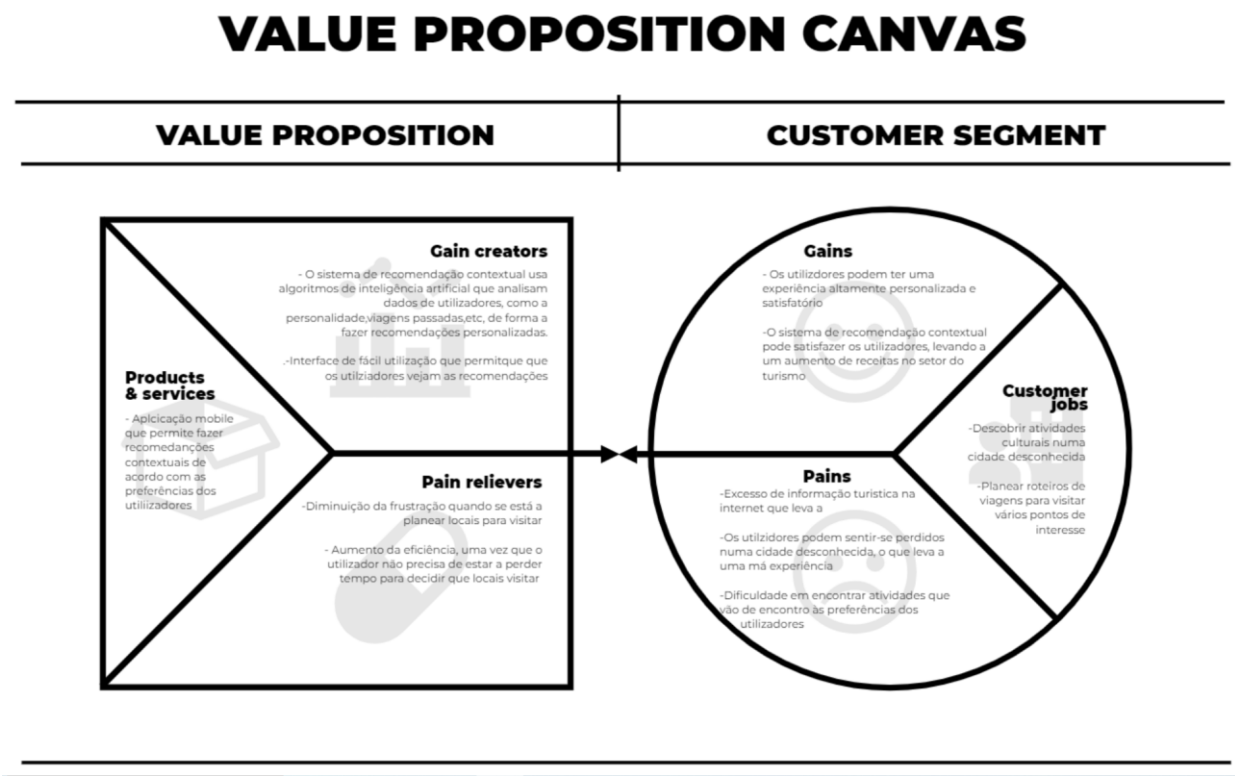


Figura 14- Canvas da proposta de valor

3.3 Seleção dos atributos contextuais

Um dos principais métodos desenvolvidos no ambiente das Decisões Multicritério Discretas é o Método de Análise Hierárquica (AHP) (Nicola, n.d.).

Selecionar os atributos relevantes para usar na contextualização é uma operação crucial. Desta forma, utilizar o método AHP é um passo relevante para escolher os elementos certos (Nicola, n.d.).

A ideia principal deste método é dividir o problema de decisão em níveis hierárquicos, facilitando, assim, sua compreensão e avaliação (Nicola, n.d.).

Este método está dividido em sete fases, sendo que essas vão ser descritas de seguida (Nicola, n.d.).

Fase 1 - Construção da árvore hierárquica de decisão

Nesta fase, o problema é definido e estruturado num diagrama de hierarquia. Enuncia-se o objetivo geral da decisão, definem-se os critérios associados ao problema de decisão e a lista de possíveis alternativas (Nicola, n.d.).

Para as alternativas, foram escolhidos um total de cinco possíveis elementos contextuais, sendo que desses cinco, apenas três são incluídos no projeto. Essas alternativas foram as seguintes:

- Lista de medos
- Lista de limitações físicas
- Clima
- Orçamento
- Estação atual

Os critérios escolhidos para influenciar a escolha das alternativas foram os seguintes:

- Simplicidade: Este critério mede o quão simples é implementar a funcionalidade.
- Relevância: Este critério mede o quão relevante é uma funcionalidade para a área em questão.
- Disponibilidade de recursos: Este critério refere-se à forma como o projeto está preparado para que estes atributos de contextualização sejam adicionados e o tempo que se despende para a adição dos atributos ao sistema.

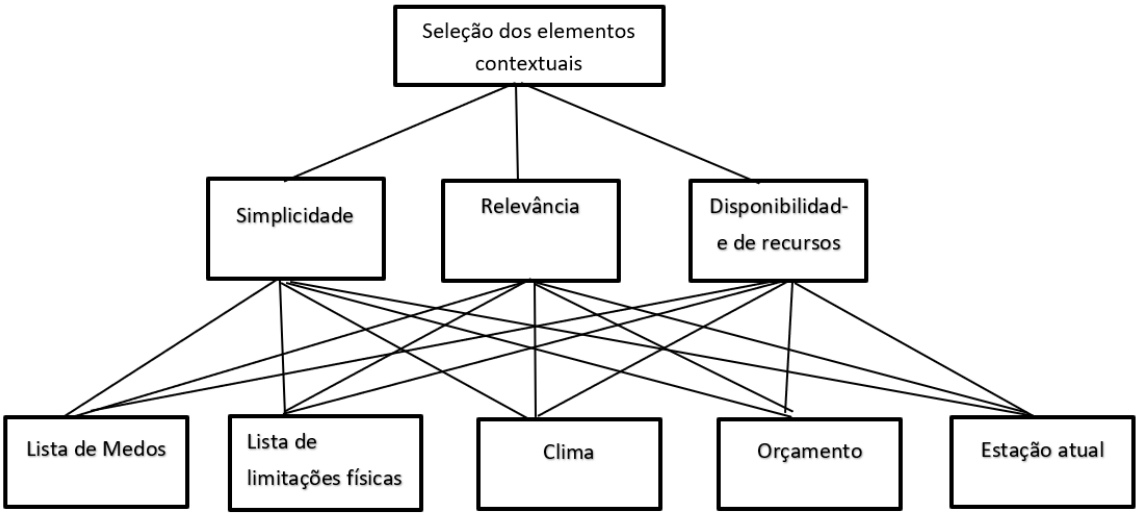


Figura 15- Árvore Hierárquica da Decisão

Fase 2: Comparação das alternativas e critérios.

Nesta fase estabelecem-se as prioridades entre os elementos para cada nível da hierarquia, através de uma matriz de comparação (Nicola, n.d.).

O primeiro passo a considerar é o de determinar que escala de valores vai ser usada na matriz de comparação. Esta não deve exceder um total de nove fatores. Para tal, é utilizada a escala fundamental do Saaty, exposta na figura 16 (Nicola, n.d.).

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 16- Escala fundamental de Saaty

De seguida, cria-se a matriz de comparação aplicando valores da escala de Saaty aos critérios.

Tabela 1- Matriz de comparação de critérios

	Complexidade	Relevância	Disponibilidade de Recursos
Complexidade	1	1/5	1/3
Relevância	5	1	3
Disponibilidade de recursos	3	1/3	1

Fase 3 - Prioridade relativa de cada critério

Nesta fase, começa-se pela normalização da matriz de comparações, com o intuito de igualar todos os critérios a uma unidade. Soma-se os elementos de cada coluna da matriz e divide-se cada valor da matriz pelo total da coluna correspondente (Nicola, n.d.).

De seguida, identifica-se a ordem de importância de cada critério. Para isto, a média aritmética dos valores de cada linha é calculada, obtendo-se assim a prioridade relativa.

Tabela 2- Matriz de comparação de critérios normalizada

	Complexidade	Relevância	Disponibilidade de Recursos	Prioridade Relativa
Complexidade	1/9	3/23	1/13	0.1062
Relevância	5/9	15/23	9/13	0.6333
Disponibilidade de Recursos	3/9	5/23	3/13	0.2605

Fase 4 - Avaliar a consistência das prioridades relativas

Nesta etapa calcula-se a Razão de Consistência (RC) para medir o quanto os julgamentos foram consistentes em relação a grandes amostras de juízos completamente aleatórios. O RC calcula-se, dividindo o Índice de Consistência (IC) pelo Índice Aleatório (IR) (Nicola, n.d.).

$$RC = \frac{IC}{IR}$$

O IR é obtido através de valores tabelados para matrizes de ordem n do Laboratório Nacional de Oak Ridge.

Tabela 3- Valores de RI para matrizes quadradas de ordem n

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IR	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

O valor de IR é equivalente a 0.58, uma vez que n é o número de critérios. Como o número de critérios considerados foi igual a 3 (n = 3), o valor associado a esse número é 0.58.

O IC é obtido através do seguinte cálculo:

$$IC = \frac{\lambda\max - \mathrm{n}}{\mathrm{n} - 1}$$

Nesta fórmula, o $\lambda\max$ que representa o maior valor próprio da matriz A e é obtido através da seguinte equação Ax = $\lambda\max$ x.

O valor de $\lambda\max$ é calculado através da seguinte forma:

$$\begin{bmatrix} 1 & 1/5 & 1/3 \\ 5 & 1 & 3 \\ 3 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} 0{,}1062 \\ 0{,}6333 \\ 0{,}2605 \end{bmatrix} = \lambda_{\max} \begin{bmatrix} 0{,}1062 \\ 0{,}6333 \\ 0{,}2605 \end{bmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{bmatrix} 0.3197 \\ 1{,}9458 \\ 0{,}7902 \end{bmatrix} = \lambda_{\max} \begin{bmatrix} 0{,}1062 \\ 0{,}6333 \\ 0{,}2605 \end{bmatrix} \Leftrightarrow$$

$$\Leftrightarrow \lambda_{\max} = \begin{bmatrix} 3.010 \\ 3.072 \\ 3.033 \end{bmatrix}$$

$$\textit{Valor próprio}\ \lambda\max = \frac{3.010+3.072+3.033}{3} = 3.038$$

Obtendo os valores necessários para calcular IC, o valor fica da seguinte maneira:

$$IC = \frac{3.038 - 3}{3 - 1} = 0.019$$

Por fim, tendo IC e RC, pode-se calcular RC, obtendo o seguinte resultado:

$$RC = \frac{0{,}019}{0.58} = 0{,}033$$

Se o RC é superior a 0,1 os julgamentos não são confiáveis porque estão demasiado perto para o conforto de aleatoriedade. Neste caso, como é possível observar, o valor de RC é de 0,033 logo os julgamentos são de confiança.

Fase 5: Construção da matriz de comparação paritária para cada critério, considerando cada uma das alternativas selecionadas.

A primeira matriz corresponde ao critério da Simplicidade. Este critério mede o quão simples é implementar a funcionalidade (Nicola, n.d.).

Tabela 4- Matriz de comparação da simplicidade da funcionalidade

Simplicidade	Lista de medos	Lista de limitações físicas	Clima	Orçamento	Estação do ano
Lista de medos	1	1	5	1	3
Lista de limitações físicas	1	1	5	1	3
Clima	1/5	1/5	1	1/5	1/2
Orçamento	1	1	5	1	3
Estação do ano	1/3	1/3	2	1/3	1

Tabela 5- Matriz de comparação da simplicidade da funcionalidade normalizada

Facilidade	Lista de medos	Lista de limitações físicas	Clima	Orçamento	Estação do ano	Prioridade
Lista de medos	40	40	5/18	40	2/7	24.113
Lista de limitações físicas	40	40	5/18	40	2/7	24.113
Clima	8	8	1/18	8	1/21	4.820
Orçamento	40	40	5/18	40	2/7	24.113
Estação do ano	40/3	40/3	1/9	40/3	2/21	8.041

A segunda matriz corresponde ao critério da relevância. Este critério mede o quão relevante é uma funcionalidade para a área em questão.

Tabela 6- Matriz de comparação da relevância da funcionalidade

Relevancia	Lista de medos	Lista de limitações físicas	Clima	Orçamento	Estação do ano

Lista de medos	1	1	1	7	9
Lista de limitações físicas	1	1	1	7	9
Clima	1	1	1	7	9
Orçamento	1/7	1/7	1/7	1	3
Estação do ano	1/9	1/9	1/9	1/3	1

Tabela 7- Matriz de comparação da relevância da funcionalidade normalizada

Relevancia	Lista de medos	Lista de limitaçõ es físicas	Clima	Orçament o	Estação do ano	Priorida de
Lista de medos	336	336	336	21/67	9/31	201.721
Lista de limitações físicas	336	336	336	21/67	9/31	201.721
Clima	336	336	336	21/67	9/31	201.721
Orçament-o	48	48	48	3/67	3/31	28.828
Estação do ano	37	37	37	3/201	1/31	22.209

A terceira matriz corresponde ao critério da disponibilidade de recursos. Este refere-se à forma como o projeto está preparado para que estes atributos de contextualização sejam adicionados, e ao tempo que se despende para a adição dos atributos ao sistema.

Tabela 8- Matriz de comparação da disponibilidade de recursos para implementar a funcionalidade

Disponibilidade de Recursos	Lista de medos	Lista de limitações físicas	Clima	Orçamento	Estação do ano
Lista de medos	1	1	2	4	5
Lista de limitações físicas	1	1	2	4	5
Clima	1/2	1/2	1	2	3

Orçamento	1/4	1/4	1/2	1	2
Estação do ano	1/5	1/5	1/3	1/2	1

Tabela 9- Matriz de comparação da disponibilidade para implementar a funcionalidade normalizada

Disponibilidade de Recursos	Lista de medos	Lista de limitações físicas	Clima	Orçamento	Estação do ano	Prioridade
Lista de medos	20/59	20/59	12/35	8/23	5/16	0.336
Lista de limitações físicas	20/59	20/59	12/35	8/23	5/16	0.336
Clima	10/59	10/59	6/35	4/23	3/16	0.174
Orçamento	5/59	5/59	3/35	2/23	1/8	0.093
Estação do ano	4/59	4/59	2/35	1/23	1/16	0.059

Fase 6: Obter a prioridade composta para as alternativas

Para calcular a prioridade composta para as alternativas, multiplica-se a matriz de prioridade relativa de cada critério com a matriz representativa dos pesos dos critérios (Nicola, n.d.).

$$\begin{bmatrix} 24.113 & 201.721 & 0.336 \\ 24.113 & 201.721 & 0.336 \\ 4.820 & 201.721 & 0.174 \\ 24.113 & 28.828 & 0.093 \\ 8.041 & 22.209 & 0.059 \end{bmatrix} \begin{bmatrix} 0{,}1062 \\ 0{,}6333 \\ 0{,}2605 \end{bmatrix} = \begin{bmatrix} 130.398 \\ 130.398 \\ 128.307 \\ 20.841 \\ 14.934 \end{bmatrix}$$

Fase 7: Escolha das alternativas

Observando os resultados, consegue-se perceber que os três atributos com as pontuações mais altas e escolhidos para integrar no projeto são os seguintes: lista de medos (130.398), lista de limitações físicas (130.398) e clima (128.307).

4 Análise e desenho

Neste capítulo são descritas as etapas da análise e do desenho da solução. São descritos os conceitos de domínio do problema, os requisitos funcionais e não funcionais identificados, bem como os vários diagramas que ajudam a visualizar o funcionamento do sistema e dos processos que lhe estão associados.

4.1 Domínio

Nesta secção é feita uma descrição dos conceitos de domínio identificados, que se encontram representados na figura 17.

O *User* é uma das componentes centrais do sistema. Este pode ser um Turista Normal ou um Turista Líder de Grupo. Os atributos mais importantes que o caraterizam são o nome, o seu papel, o seu género, a sua nacionalidade, entre outros.

Para a correta atribuição de recomendações, é necessário ter em conta várias entidades que estão diretamente ligadas ao *User* e que moldam o seu perfil. A personalidade (*Personality*), as limitações (*Limitation*), motivações (*Motivation*), preferências (*Preferences*) e categorias de turismo (*TourismCategories*) são alguns dos atributos que influenciam as sugestões de recomendações. Para cada utilizador está também associada uma localização (*Location*).

Os grupos (*Groups*) e subgrupos (*Subgroups*) também estão incluídos no modelo de domínio. Os utilizadores podem fazer parte de grupos, quando fazem uma excursão em conjunto. Estes grupos podem ser divididos em subgrupos pelo Sistema Multiagente (MAS). Os subgrupos são criados tendo em conta as personalidades e preferências dos utilizadores presentes no grupo principal.

O agregado do POI (*POI*) é um dos elementos mais importantes no contexto das recomendações. Este possui atributos como nome, localização, categoria, breve descrição, média de rating dos utilizadores, e horários de funcionamento. O conceito de POI está associado a *User*, *Group*, e *Subgroup*, sendo estes recomendados tanto na forma individual, como na forma coletiva (*POI_Recommendation*).

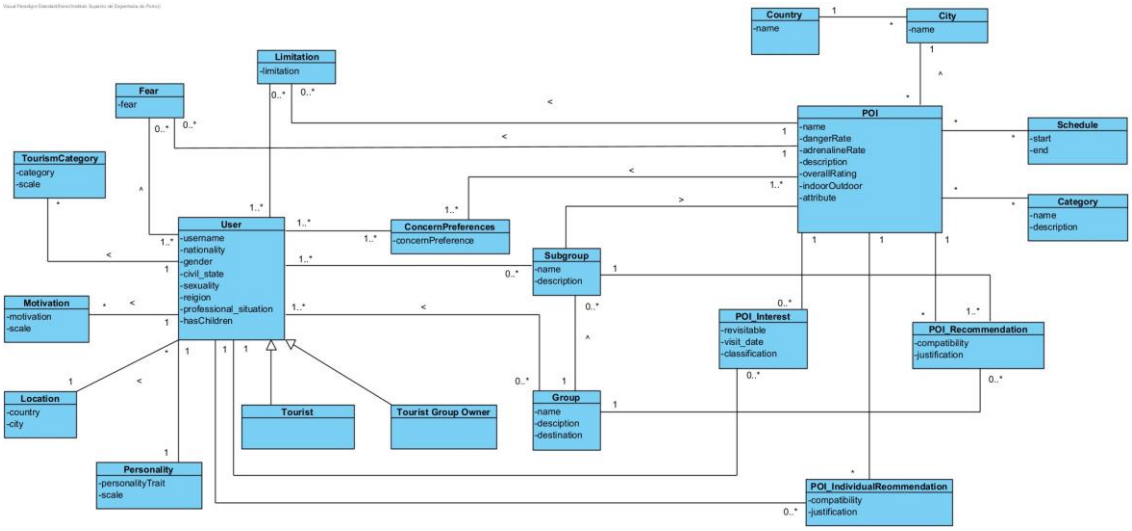


Figura 17 - Modelo Domínio Recommendation Engine Microservice

4.2 Requisitos

Na presente secção é feita uma exposição dos atores presentes no sistema, bem como o levantamento dos requisitos funcionais e não funcionais, finalizando com um diagrama de casos de uso que permite clarificar as relações entre os atores e as funcionalidades desenvolvidas.

4.2.1 Atores e as suas Características

Em engenharia de *software*, os "atores" referem-se a entidades que desenvolvem algum papel em relação ao sistema. Estes podem ser pessoas ou outros sistemas, como *hardware* ou *software* que interagem diretamente com o sistema. Os atores são quem desempenham os casos de uso, e um mesmo ator pode estar associado a um ou mais casos de uso (Nogueira, n.d.).

No sistema Grouplanner, foram identificados os seguintes atores:

Turista Geral:

O Turista Geral é um dos atores mais importantes no sistema. Este interage com as várias funcionalidades disponíveis na aplicação. Dentro das funcionalidades que ele pode executar, estas são as principais:

- Preencher um questionário de personalidade, permitindo assim que o sistema possa prever o tipo de atrações turísticas que gosta, assim como preferências e preocupações com a viagem (Alves et al., 2023), permitindo que sejam sugeridas recomendações que vão mais ao encontro dos gostos do mesmo.
- Solicitar recomendações individuais de pontos de interesse, que se alinham com as suas preferências.
- Dar *ratings* aos pontos de interesse. Esta interação permite que haja uma melhoria da qualidade das recomendações fornecidas pelo sistema ao longo do tempo a utilizadores com o mesmo tipo de personalidade.
- Ser inserido em grupos de excursão, sendo possível a sua entrada através de um convite. Dentro do grupo, este pode comunicar com outros participantes do mesmo grupo através de um chat.

Turista (Líder do Grupo):

O Turista que assume o papel de líder do grupo, para além de poder realizar as atividades que um turista participante interno pode fazer, tem também a possibilidade de criar e gerir grupos de excursão, servindo como um ponto fulcral na coordenação do grupo.

Além disso, o líder do grupo é o único com a capacidade de solicitar as recomendações de grupo, uma funcionalidade que leva em consideração as preferências e interesses de todos os membros do grupo, tendo por base as suas personalidades.

4.2.2 Levantamento dos Requisitos Funcionais

A análise de requisitos é um processo fundamental que permite definir as funcionalidades que são requisitadas. Os requisitos podem ser divididos em dois, sendo eles os requisitos funcionais e os requisitos não funcionais (Singla, 2022).

Os requisitos funcionais são as funcionalidades que o sistema oferece de forma que atinja as necessidades dos utilizadores. Todas essas funcionalidades devem ser obrigatoriamente incluídas no sistema, de acordo com o contrato (Singla, 2022).

O processo de levantamento de requisitos foi realizado no início do desenvolvimento do projeto, sendo uma etapa indispensável antes de se começar a implementação. Este processo foi feito em conjunto com pessoas responsáveis pela organização do projeto, nomeadamente investigadores e bolseiros do GECAD, mas também com a empresa Sistrade, uma vez que esta foi a responsável por fornecer a API do clima e era necessário integrar essa informação com o Grouplanner.

Estas pessoas possuem um conhecimento profundo dos objetivos do projeto, guiando os estudantes no processo de identificar as funcionalidades essenciais e as características que o sistema deve possuir.

Os requisitos funcionais foram definidos sob a forma de *User Stories* agrupadas por *Epic*. Para cada *User Story* definida são especificados os seguintes pontos:

- **Descrição** – Descrição sumária da *user story*;
- **Critério de aceitação** - Testes de aceitação para confirmar que o sistema efetua os requisitos propostos na *user story*;
- **Esforço** - Uma pontuação, na escala de Fibonacci, que quantifica o esforço necessário para desenvolver uma *user story*;
- **Prioridade** – Uma classificação MoSCoW, em que cada representa um nível de prioridade da implementação da *user story*. As letras representativas encontram-se na seguinte tabela:

Tabela 10 – Escala de MoSCoW

Utilizador	Detalhes
M (*Must*)	Estes são os requisitos cruciais que o projeto tem de atender. Sem esses requisitos, o projeto é considerado um falhanço.
S (*Should*)	Estes são os requisitos importantes para o sucesso do projeto, no entanto, não são críticos. O projeto pode ser lançado sem a sua execução.
C (*Could*)	Estes são os requisitos desejáveis. Frequentemente vistos como “era bom ter”.
W (*Won’t*)	Estes são os requisitos menos críticos, ou que não são apropriados nesta altura, mas poderão ser considerados no futuro.

Os requisitos funcionais identificados foram os seguintes:

4.2.2.1 REQ-01: Integração com o Serviço Meteorológico Externo para Recomendação de Pontos de Interesse

Tabela 11 – Epic-1

E-1 - Integração com o Serviço Meteorológico Externo para Recomendação de Pontos de Interesse

Descrição	Como *Product Owner*, pretendo otimizar a integração com um serviço meteorológico externo, para melhorar a precisão das recomendações de pontos de interesse, avaliando-os como disponíveis ou não disponíveis.

Tabela 12 – US-1.1

US-1.1 Obtenção de Condições Meteorológicas para um dado Local num dado Período	
Descrição	Como *Product Owner*, pretendo que o sistema obtenha as condições meteorológicas para um determinado local (país e cidade) e espaço temporal, com o intuito de analisar as condições climáticas num determinado sítio referente ao período da excursão.
Critério de aceitação	O Sistema deve ser capaz de fazer uma chamada com sucesso ao sistema meteorológico externo, enviando o local (cidade e país), e obter as previsões meteorológicas em formato JSON para o local enviado, de acordo com as datas de início e de fim inseridas da excursão.
Prioridade	S
Esforço	2

Tabela 13 - US-1.2

US-1.2 Identificação de Condições Meteorológicas Desfavoráveis	
Descrição	Como *Product Owner*, pretendo identificar as condições meteorológicas adversas, para determinar se um ponto de interesse deve ser recomendado ou não.
Critério de aceitação	O Sistema deve ser capaz de identificar as condições meteorológicas desfavoráveis nas previsões obtidas, com base nos códigos de tempo fornecidos pela Sistrade.
Prioridade	S
Esforço	3

Tabela 14 - US-1.3

US-1.3 Criação de Justificações para Exclusão de Pontos de Interesse	
Descrição	Como *Prodcut Owner*, pretendo que o sistema crie justificações quando um ponto de interesse não é recomendado por causa das condições meteorológicas desfavoráveis, para manter o utilizador informado sobre o porquê da exclusão do ponto de interesse.
Critério de aceitação	O Sistema deve criar uma mensagem explícita do porquê de o ponto de interesse não estar a ser recomendado, que posteriormente será apresentada no *VirtualPet.*
Prioridade	C
Esforço	1

4.2.2.2 REQ-02: Gerar Recomendações Individuais Baseadas em Fatores Contextuais

Tabela 15 - Epic-2

E-2 Gerar Recomendações Individuais Baseadas em Fatores Contextuais com Opção de Substituição de Pontos de Interesse sob Más Condições Meteorológicas	
Descrição	Como Turista, desejo receber recomendações de pontos de interesse que tenham em conta fatores contextuais, como as minhas preferências pessoais, limitações físicas, fobias, as condições meteorológicas atuais e a minha localização. Para além disso, pretendo ter a opção de ver os pontos alternativos sugeridos para locais onde as condições climáticas são desfavoráveis para que possa ter uma experiência de viagem mais personalizada e satisfatória.

Tabela 16 - US-2.1

US-2.1 Obtenção de Dados do Turista para Recomendações Personalizadas

Descrição	Como *Product Owner*, pretendo que o sistema obtenha os dados do turista, como fobias, limitações físicas, localização, personalidade, categorias de turismo, entre outros, para tornar as recomendações de pontos de interesse mais personalizadas e de encontro às preferências do turista.
Critério de aceitação	Os dados do turista, atualizados, devem ser obtidos do microsserviço *Multi-Agent Microservice* através de um pedido HTTP.
Prioridade	M
Esforço	2

Tabela 17 - US-2.2

US-2.2 Recomendações Considerando Fobias do Turista	
Descrição	Como Turista, pretendo receber recomendações de pontos de interesse que tenham em conta as minhas fobias, para proporcionar uma viagem mais segura e confortável.
Critério de aceitação	Quando o turista faz um pedido de recomendações individual, o sistema deve apresentar uma lista com várias recomendações de pontos de interesse que não possam causar desconforto ao utilizador devido aos seus medos, com uma justificação dessa recomendação e a categoria de turismo onde essa se insere.
Prioridade	M
Esforço	3

Tabela 18 - US-2.3

US-2.3 Recomendações Considerando Limitações Físicas do Turista

Descrição	Como Turista, pretendo receber recomendações de pontos de interesse que tenham em conta as minhas limitações físicas, para proporcionar uma viagem mais segura e confortável.
Critério de aceitação	Quando o turista faz um pedido de recomendações individual, o sistema deve apresentar uma lista com várias recomendações de pontos de interesse que não possam causar desconforto ao utilizador devido às suas limitações físicas, com uma justificação dessa recomendação e a categoria de turismo onde essa se insere.
Prioridade	M
Esforço	3

Tabela 19 - US-2.4

US-2.4 Identificação de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis	
Descrição	Como *Product Owner*, pretendo que o sistema identifique os pontos de interesse onde as condições meteorológicas são desfavoráveis, para permitir a substituição desses pontos por pontos alternativos.
Critério de aceitação	O Sistema deve ser capaz de verificar as condições meteorológicas para cada ponto de interesse e "assinalar" aqueles que devem ser passíveis de exclusão.
Prioridade	S
Esforço	2

Tabela 20 - US-2.5

US-2.5 Sugestão de Pontos de Interesse Alternativos

Descrição	Como *Product Owner*, pretendo que o sistema recomende pontos de interesse alternativos, para disponibilizar opções de substituição ao turista.
Critério de aceitação	O Sistema deve sugerir uma lista de pontos alternativos aos pontos de interesse onde as condições meteorológicas são desfavoráveis.
Prioridade	S
Esforço	3

Tabela 21 - US-2.6

US-2.6 Identificação de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis e a sua Eliminação Automática	
Descrição	Como *Product Owner*, pretendo que o sistema identifique e elimine automaticamente os pontos de interesse que estão sob condições meteorológicas desfavoráveis, para permitir um maior conforto ao turista sem necessidade da sua intervenção.
Critério de aceitação	O Sistema deve ser capaz de verificar as condições meteorológicas para cada ponto de interesse e eliminar aqueles que devem ser excluídos.
Prioridade	S
Esforço	2

4.2.2.3 REQ-003: Gerar Recomendações Para Grupos Baseadas em Fatores Contextuais

Tabela 22 – Epic-3

E-3 Geração de Recomendações para Grupos Baseadas em Fatores Contextuais com Opção de Substituição de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

Descrição	Como Turista Líder de Grupo, desejo receber recomendações de pontos de interesse para o grupo que lidero, que tenham em conta fatores contextuais, como as preferências pessoais dos elementos do grupo, limitações físicas e medos. Para além disso, também devem ter em conta as condições climáticas atuais e a localização do grupo, permitindo visualizar pontos alternativos sugeridos para locais onde estão condições meteorológicas desfavoráveis, de forma que o grupo possa ter uma experiência de viagem mais personalizada e satisfatória.

Tabela 23 - US-3.1

US-3.1 Obtenção de Dados de Subgrupos e Utilizadores para Recomendações Personalizadas	
Descrição	Como *Product Owner*, pretendo que o sistema obtenha os dados de cada subgrupo dentro de um grupo e de cada utilizador dentro de um subgrupo, como fobias, limitações físicas, localização, personalidade, categorias de turismo, entre outros, para tornar as recomendações de pontos de interesse mais personalizadas e de encontro às preferências do turista.
Critério de aceitação	Os dados de cada subgrupo e cada utilizador de um subgrupo, atualizados, devem ser obtidos do microsserviço *Multi-Agent Microservice* através de um pedido HTTP e utilizados para gerar as recomendações.
Prioridade	M
Esforço	2

Tabela 24 - US-3.2

US-3.2 Geração de Recomendações Considerando Fobias dos Elementos Presentes nos Subgrupos	
Descrição	Como Turista Líder de Grupo, pretendo obter as recomendações de pontos de interesse para cada subgrupo, que tenham em conta as fobias dos turistas, para proporcionar uma viagem mais segura e confortável.

Critério de aceitação	Quando o Turista Líder de Grupo faz um pedido de recomendações de grupo, o sistema deve apresentar uma lista com várias recomendações de pontos de interesse para cada subgrupo, que não inclua pontos que possam causar desconforto à maioria dos turistas de um subgrupo, com justificações para cada recomendação e a categoria de turismo onde cada ponto de interesse se insere.
Prioridade	M
Esforço	3

Tabela 25 - US-3.3

US-3.3 Geração de Recomendações Considerando as Limitações Físicas dos Elementos Presentes nos Subgrupos	
Descrição	Como Turista Líder de Grupo, pretendo obter as recomendações de pontos de interesse para cada subgrupo, que tenham em conta as limitações físicas dos turistas, para proporcionar uma viagem mais segura e confortável.
Critério de aceitação	Quando o Turista Líder de Grupo faz um pedido de recomendações de grupo, o sistema deve apresentar uma lista com várias recomendações de pontos de interesse para cada subgrupo, que não inclua pontos que possam causar desconforto a pelo menos um turista de um subgrupo, com justificações para cada recomendação e a categoria de turismo onde cada ponto de interesse se insere.
Prioridade	M
Esforço	3

Tabela 26 - US-3.4

US-3.4 Identificação de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis para cada Subgrupo	
Descrição	Como *Product Owner*, pretendo que o sistema identifique os pontos de interesse onde as condições meteorológicas são desfavoráveis, para permitir a substituição desses pontos por pontos alternativos.
Critério de aceitação	O Sistema deve ser capaz de verificar as condições meteorológicas para cada ponto de interesse e "assinalar" aqueles que devem ser passíveis de exclusão para cada subgrupo.
Prioridade	S
Esforço	2

Tabela 27 – US-3.5

US-3.5 Sugestão de Pontos de Interesse Alternativos para Subgrupos	
Descrição	Como *Product Owner*, pretendo que o sistema recomende pontos de interesse alternativos, para disponibilizar opções de substituição aos subgrupos.
Critério de aceitação	O Sistema deve sugerir uma lista de pontos alternativos aos pontos de interesse para cada subgrupo, onde as condições meteorológicas são desfavoráveis tendo em conta as restrições referidas nos requisitos anteriores.
Prioridade	S
Esforço	3

Tabela 28 - US-3.6

US-3.6 Identificação e Eliminação de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis para cada Subgrupo	
Descrição	Como *Product Owner*, pretendo que o sistema identifique e elimine automaticamente os pontos de interesse que estão sob condições meteorológicas desfavoráveis, para permitir um maior conforto aos utilizadores de cada subgrupo sem necessidade da sua intervenção.
Critério de aceitação	O Sistema deve ser capaz de verificar as condições meteorológicas para cada ponto de interesse e eliminar aqueles que devem ser excluídos.
Prioridade	S
Esforço	2

4.2.2.4 REQ-04 Atualização Diária de Recomendações e Integração com o *VirtualPet*

Tabela 29 – Epic-4

E-4 Atualização Diária de Recomendações e Integração com o *VirtualPet*	
Descrição	Como *Product Owner*, pretendo que haja uma atualização das recomendações de pontos de interesse de 24 em 24 horas, permitindo a substituição de pontos de interesse com condições meteorológicas desfavoráveis e integrar essa informação com o *VirtualPet* para uma experiência de utilizador mais imersiva e interativa.

Tabela 30 - US-4.1

US-4.1 Inicialização de um *Worker* para Atualizações Diárias	
Descrição	Como *Product Owner*, pretendo que o sistema inicialize um *worker* que seja capaz de executar tarefas de 24 em 24 horas, para atualizar diariamente as recomendações de pontos de interesse com base nas condições meteorológicas.

Critério de aceitação	Quando é feito um pedido de recomendações, um *worker* tem de ser inicializado e aplicar a lógica que em si está definida (obter recomendações atualizadas) de 24 em 24 horas.
Prioridade	S
Esforço	3

Tabela 31 - US-4.2

US-4.2 Obtenção Diária de Recomendações Atualizadas	
Descrição	Como *Product Owner*, pretendo que o sistema obtenha diariamente as recomendações de pontos de interesse, tanto as originais como as alternativas, para fornecer recomendações atualizadas.
Critério de aceitação	As recomendações devem ser obtidas e atualizadas de forma correta de 24 em 24 horas.
Prioridade	S
Esforço	3

Tabela 32 - US-4.3

US-4.3 Transmissão de Recomendações para o *VirtualPet*	
Descrição	Como Product Owner, pretendo que o sistema transmita as recomendações originais e as suas alternativas para o *VirtualPet*, para permitir a interação do utilizador com o mesmo de forma a aceitar ou rejeitar as alternativas sugeridas.
Critério de aceitação	Os dados dos pontos de interesse originais e alternativos enviados para o *VirtualPet* devem ser os corretos.
Prioridade	S

Esforço	3

4.2.3 Levantamento dos Requisitos Não Funcionais

Os requisitos não funcionais são as características de qualidade que o sistema deve ter, de forma a respeitar o contrato do projeto, focando-se mais em como o sistema realiza as suas funções, em vez do que faz especificamente (AltexSoft, 2019).

Para identificar os requisitos não funcionais, recorreu-se ao modelo FURPS+ permitindo assim uma maior facilidade na especificação dos requisitos.

De seguida encontram-se os requisitos não funcionais identificados para diferentes categorias presentes no modelo FURPS+ (Anzileiro, 2020):

4.2.3.1 Usabilidade

A categoria da usabilidade, refere-se aos requisitos que se relacionam com a experiência que o utilizador vai ter, por exemplo, com a interface.

Os requisitos não funcionais identificados para esta categoria foram os seguintes:

Tabela 33 - US-NF-1.1

US-NF-1.1 Inserção de Maneira Intuitiva de Datas	
Descrição	Como *Product Owner*, pretendo que, quando se faz um pedido de recomendações, a inserção das datas de início e de fim seja intuitiva, para evitar que o utilizador insira as datas erradas.
Critério de aceitação	O Sistema deve ter dois *DatePicker*, com a data de início predefinida para o dia em que está a ser feito o pedido, um para indicação da data de início e outro para a data de fim da viagem/excursão.
Prioridade	M
Esforço	2

Tabela 34 - US-NF-1.2

US-NF-1.2 Inserção de Maneira Intuitiva da Localização	
Descrição	Como *Product Owner*, pretendo que quando se faz um pedido de recomendações, a inserção da localização (país e cidade) seja intuitiva, para evitar que o utilizador insira o local errado.
Critério de aceitação	O Sistema deve sugerir o país e a cidade à medida que o utilizador digita essa informação.
Prioridade	S
Esforço	2

Tabela 35 - US-NF-1.3

US-NF-1.3 *Feedback* Imediato Após Pedido de Recomendações	
Descrição	Como *Product Owner*, pretendo que, logo após que o utilizador faça um pedido de recomendações, seja fornecido *feedback* sobre o sucesso ou insucesso do mesmo, para elucidar o utilizador sobre o estado do pedido.
Critério de aceitação	O *feedback* tem de ser direto e claro. No ecrã deve aparecer a mensagem “As recomendações foram geradas com sucesso!”, no caso de o pedido ser bem-sucedido, e a mensagem “Algo correu mal! Tente novamente.” no caso do pedido não ser bem-sucedido.
Prioridade	S
Esforço	1

4.2.3.2 Performance

A categoria da performance, refere-se à forma como o sistema se deve desempenhar a nível de velocidade, eficiência, eficácia, entre outros.

Os requisitos não funcionais identificados para esta categoria foram os seguintes:

Tabela 36 - US-NF-2.1

US-NF-2.1 Processamento Rápido de Pedidos de Recomendações	
Descrição	Como *Product Owner*, pretendo que o sistema processe os pedidos de recomendações num curto espaço de tempo, para que o utilizador tenha uma experiência mais agradável.
Critério de aceitação	O Sistema deve demorar no máximo dez segundos a processar os pedidos de recomendações.
Prioridade	S
Esforço	2

Tabela 37 - US-NF-2.2

US-NF-2.2 Suporte de Múltiplos Utilizadores ao Mesmo Tempo	
Descrição	Como *Product Owner*, pretendo que o sistema seja capaz de lidar com vários utilizadores em simultâneo, para garantir que o sistema continue com uma performance contínua.
Critério de aceitação	O Sistema tem de ser capaz de lidar com vários utilizadores a utilizarem a aplicação ao mesmo tempo sem perder performance. O Sistema tem de avisar os desenvolvedores quando houver uma sobrecarga de utilizadores.
Prioridade	S
Esforço	2

4.2.3.3 Suportabilidade

A categoria da suportabilidade, refere-se à forma como o sistema deve estar preparado para poder ser mantido e adaptado de forma a lidar com mudanças.

Os requisitos não funcionais identificados para esta categoria foram os seguintes:

Tabela 38 - US-NF-2.3

US-NF-2.3 Sistema Apto para Mudanças	
Descrição	Como *Product Owner*, pretendo que o sistema seja construído de uma forma modular, para permitir que sejam feitas alterações com facilidade e adicionadas novas funcionalidades.
Critério de aceitação	O Sistema tem de possuir uma arquitetura que facilite a adição de novas funcionalidades e alteração de funcionalidades já existentes. O código do Sistema deve estar bem organizado e comentado para facilitar a sua compreensão.
Prioridade	S
Esforço	2

Tabela 39 - US-NF-2.4

US-NF-2.4 Documentação Completa das Funcionalidades	
Descrição	Como *Product Owner*, pretendo que as funcionalidades do sistema estejam bem documentadas, para facilitar a compreensão por parte dos diferentes desenvolvedores.
Critério de aceitação	Cada funcionalidade desenvolvida tem de ter associada a si documentação que descreva com clareza o propósito e funcionamento dela.
Prioridade	S
Esforço	2

4.2.3.4 Confiabilidade

A categoria da confiabilidade, refere-se aos requisitos que se relacionam com a forma como o sistema é resiliente.

Os requisitos não funcionais identificados para esta categoria foram os seguintes:

Tabela 40 - US-NF-3.1

US-NF-3.1 Disponibilidade do Sistema	
Descrição	Como *Product Owner*, pretendo garantir que o sistema tenha uma alta disponibilidade, para que os utilizadores acedam às funcionalidades relacionadas com as recomendações em qualquer altura.
Critério de aceitação	O Sistema tem de ser capaz de lidar com picos de procura, para manter a sua disponibilidade e tem de alertar os desenvolvedores quando este se encontra em baixo.
Prioridade	S
Esforço	2

4.2.3.5 Segurança

A categoria da segurança, refere-se à forma como o sistema está preparado para proteger as informações sensíveis do mesmo.

Os requisitos não funcionais identificados para esta categoria foram os seguintes:

Tabela 41 - US-NF-4.1

US-NF-4.1 Autorização para Acesso a Dados de Recomendações	
Descrição	Como *Product Owner*, pretendo que as informações das recomendações só possam ser acedidas por pessoal autorizado, cuja autenticação é feita através de um par email/password, para proteger a privacidade dos utilizadores.
Critério de aceitação	O sistema deve garantir que apenas utilizadores com as permissões necessárias possam aceder às informações relacionadas com as recomendações.
Prioridade	M

Esforço	2

Tabela 42 - US-NF-4.2

US-NF-4.2 Comunicação Segura	
Descrição	Como *Product Owner*, pretendo que a aplicação utilize o protocolo HTTPS para a comunicação de dados, para garantir a segurança dos dados que são transmitidos.
Critério de aceitação	As comunicações de dados têm de ser efetuadas através de HTTPS.
Prioridade	S
Esforço	1

4.2.4 Diagrama de Casos de Uso

Na figura 18 é possível observar as interações entre os atores e o sistema. Verifica-se que existem dois atores: o ator Turista Geral e o ator Turista Líder de Grupo. O Turista Geral pode executar várias funcionalidades, mas as mais relevantes para este contexto são as seguintes: Gerar Recomendações Individuais Baseadas em Fatores Contextuais; Já o Turista Líder de grupo, para além de poder executar as funcionalidades que o Turista Participante executa, também pode executar as seguintes funcionalidades: Gerar Recomendações de Grupo Baseadas em Fatores Contextuais.

Os requisitos REQ02 E REQ03 são executados por turistas, já os requisitos REQ01 e REQ04 são executados pelo próprio sistema. O requisito REQ01 é executado no microserviço *Recommendation Engine Microservice*, enquanto o requisito REQ04 é executado no *Cliente* HTTP (Grouplanner app).

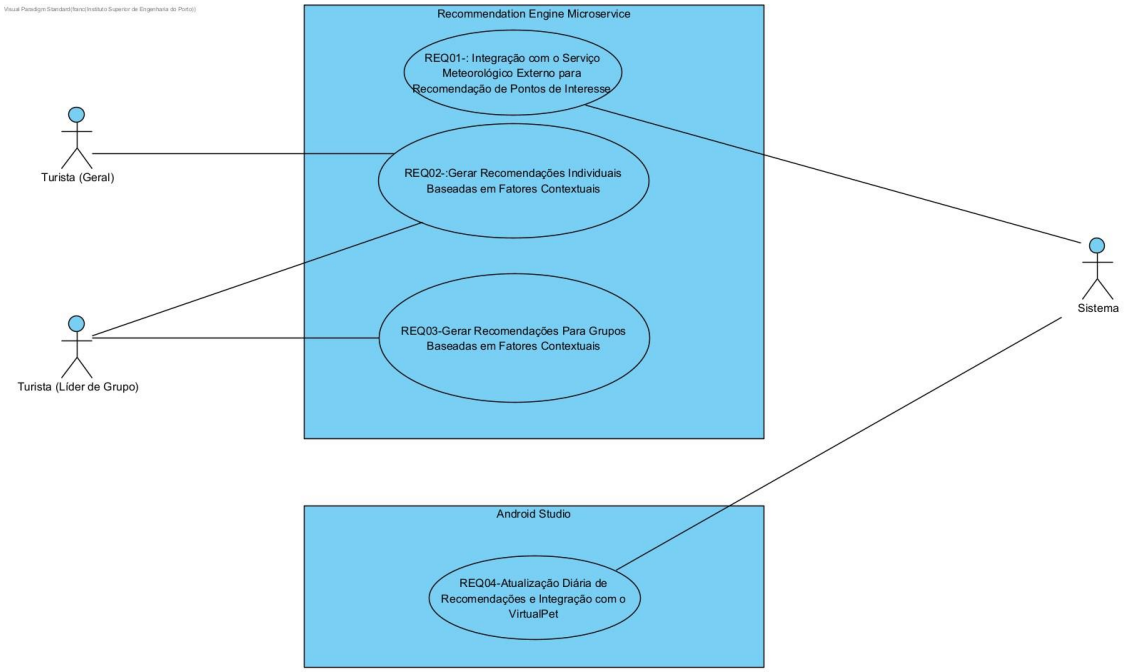


Figura 18 - Modelo de Casos de Uso

4.3 Desenho

Na presente secção, abordam-se os padrões de design adotados, a arquitetura do sistema e as suas componentes, exemplificadas através de diagramas de componentes e de implantação. Para além disso, são apresentados diagramas de sequência que permitem perceber o fluxo das funcionalidades realizadas pelo sistema.

4.3.1 Princípios e conceitos

Para a realização de um bom design, foi preciso adotar certos padrões e princípios. Os princípios abordados foram o SOLID e o GRASP.

Para além disso, de forma a construir uma aplicação robusta utilizaram-se algumas arquiteturas, nomeadamente a arquitetura de microserviços e a arquitetura de camadas.

Este capítulo visa apresentar as vantagens e no que consiste a utilização dos princípios e das arquiteturas referidas.

SOLID é um conjunto de princípios de design que são utilizados no desenvolvimento de *software* orientado a objetos. Este acrónimo representa cinco princípios de design fundamentais: *Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle e Dependency Inversion Principle*. Estes princípios são amplamente utilizados na indústria e trazem uma variedade de benefícios para os desenvolvedores (Watts, 2020).

O objetivo dos princípios SOLID é reduzir as dependências de maneira que seja possível alterar uma área do *software* sem afetar outras. Além disso, permitem tornar os projetos mais fáceis de compreender, manter e expandir. A utilização destes princípios facilita na prevenção de problemas e na criação de software adaptável, eficaz e ágil. Os princípios SOLID são os seguintes (Watts, 2020):

- ***Single Responsibility Principle***: Este princípio significa que cada classe deve ter uma responsabilidade única, focando-se numa parte da funcionalidade do *software*.
- ***Open-Closed Principle*****:** O princípio *Open-Closed* sugere que as entidades de software (classes, módulos, funções, etc.) devem estar abertas à extensão, mas fechadas à modificação, ou seja, em vez de alterar a classe, pretende-se simplesmente estendê-la evitando assim alterações diretas no código original.
- ***Liskov Substitution Principle***: Segundo o princípio *Liskov Substitution*, qualquer classe derivada deve ser capaz de substituir a classe base sem alterar as propriedades do programa.
- ***Interface Segregation Principle***: A ideia geral do princípio *Interface Segregation* é que é melhor a utilização de várias interfaces mais pequenas em vez de poucas interfaces maiores. Ou seja, não se deve começar com uma interface existente e adicionar novos métodos. Em vez disso, deve-se começar por criar uma interface e depois permitir que a classe implemente várias interfaces conforme necessário.
- ***Dependency Inversion Principle***: Este princípio afirma que módulos de alto nível não devem depender de entidades de baixo nível, ambos devem depender de abstrações.

GRASP é a sigla de *General Responsibility Assignment Software Patterns* (Padrões de Software de Atribuição de Responsabilidade Geral) que representa um conjunto de "regras" usadas no design orientado a objetos para atribuir responsabilidades a classes e objetos. O GRASP ajuda os desenvolvedores a decidir que responsabilidade deve ser atribuída a que objeto/classe (Rao n.d.).

A aplicação destes princípios auxilia os desenvolvedores na criação de sistemas mais escaláveis e fáceis de manter (Rao n.d.).

Existem nove padrões GRASP sendo eles os seguintes (Rao n.d.):

- ***Creator***: Este princípio é responsável por designar a responsabilidade de criar uma instância de uma classe A à classe B, ou seja, decide quem pode ser o criador com base na associação e interação dos objetos.
- ***Information Expert***: Este princípio é responsável por atribuir as responsabilidades às classes que dispõem das informações necessárias para as cumprir.
- ***Low Coupling*****:** Este princípio é responsável por atribuir responsabilidades de modo que o acoplamento permaneça baixo, minimizando as dependências entre as classes, tornando assim o sistema eficiente e o código reutilizável.

- ***Controller*:** Este padrão ajuda a determinar qual é o primeiro objeto que recebe a mensagem dos objetos da camada UI. Delega o trabalho para outras classes e coordena a atividade geral.
- ***High Cohesion***: Este princípio é responsável por manter as classes focadas e limitadas a uma responsabilidade específica, promovendo assim um código fácil de entender e de manter, o que facilita a reutilização do código.
- ***Polymorphism***: Este princípio permite que o código seja facilmente reutilizável, através da permissão de classes diferentes serem tratadas como se tratasse de uma classe comum.
- ***Pure Fabrication*:** Este princípio é responsável por permitir a criação de classes que não representam conceitos do domínio do problema, mas que ajudam a alcançar objetivos de design específicos como baixo acoplamento e alta coesão. Normalmente é implementado através de padrões de design como o *Adapter* e o *Strategy*.
- ***Indirection*:** Este princípio utiliza uma entidade intermediária para possibilitar a comunicação com as outras entidades, prevenindo assim que elas estejam diretamente acopladas.
- ***Protected Variation***: Este princípio é uma estratégia de design de *software* que visa minimizar o impacto das variações de certos elementos sobre outros elementos dentro de um sistema. Este princípio centra-se em torno da criação de uma interface bem definida, que serve como uma barreira, garantindo que as variações num componente não afetem negativamente outras unidades do sistema.

É importante destacar que os princípios SOLID que foram utilizados no projeto foram o *Dependency Inversion Principle, Interface Segregation Principle, Single Responsability Principle.* Em relação aos princípios GRASP, os mais utilizados foram o *Information Expert, Controller, Low Coupling e High Cohesion*.

Após explorar os princípios SOLID e GRASP, é necessário compreender o enquadramento teórico da arquitetura de microsserviços, utilizada para o desenvolvimento do protótipo do sistema de recomendação.

Com a evolução da tecnologia, a arquitetura de microsserviços emergiu como uma abordagem inovadora para o desenvolvimento de aplicações. Os microsserviços são uma abordagem arquitetónica e organizacional do desenvolvimento de software na qual a aplicação consiste em pequenos serviços independentes que comunicam entre si através de API bem definidas. Esta abordagem representa uma mudança significativa em relação aos sistemas monolíticos tradicionais (Russo, 2003c).

Utilizando uma arquitetura de microsserviços, uma aplicação é criada como componentes independentes que gerem cada processo da aplicação como um serviço. Os serviços são desenvolvidos para atender necessidades de negócio e cada serviço desempenha uma função específica. Como são executados de forma independente, cada serviço pode ser atualizado, implantado e escalado para satisfazer as necessidades de funções específicas de uma

aplicação. Além disso, este tipo de abordagem promove uma manutenção mais eficiente e uma resposta mais acelerada às alterações que acontecem no mercado (Russo, 2003c; Martinekuan, n.d.).

As arquiteturas baseadas em microsserviços facilitam a escalabilidade e agilizam o desenvolvimento de aplicações, permitindo uma maior inovação e redução do tempo de introdução de novas funcionalidades no mercado (Russo, 2003c; Martinekuan, n.d.).

Algumas das maiores vantagens desta arquitetura são as seguintes (Russo, 2003c; Martinekuan, n.d.):

- **Agilidade** – Dado que os microsserviços são implementados de forma independente, é possível gerir a resolução de falhas e o lançamento de novos recursos com mais facilidade. Esta estrutura permite a atualização de um serviço sem implementar novamente toda a aplicação.
- **Escalabilidade** – Os serviços podem ser escalados independentemente, o que permite a expansão dos subsistemas que necessitem de mais recursos sem que, para isso, seja necessária a expansão de toda a aplicação. Isto permite que as equipas responsáveis por cada serviço ajustem corretamente as necessidades de infraestrutura, avaliem com precisão o custo de um recurso e assegurem a disponibilidade quando um serviço experimenta um aumento de procura.
- **Fácil Implantação -** Os microsserviços permitem a integração e a entrega contínua, o que facilita a experimentação de novas ideias e o seu cancelamento caso algo não funcione corretamente.
- **Combinação de tecnologias** – Cada microsserviço pode ser desenvolvido com a tecnologia mais adequada ao respetivo serviço, permitindo assim a utilização de várias tecnologias adaptadas às exigências de cada serviço.
- **Resiliência** – Ao contrário do que acontece numa arquitetura monolítica (a falha de um único componente poderá causar a falha de todo o sistema), numa arquitetura de microsserviços, se um serviço individual ficar indisponível, não irá interromper toda a aplicação, desde que esses microsserviços estejam preparados para lidar com falhas, ou seja, a independência do serviço aumenta a resistência a falhas na aplicação.

Cada microsserviço segue uma arquitetura de camadas. Esta arquitetura é composta por quatro níveis, cada um com as suas próprias características (Walpita, 2020).

A arquitetura em camadas é o tipo de arquitetura mais "básico" de arquitetura de *software* e em comparação com outras, é uma das que tem um custo de implementação muito baixo. De seguida são apresentadas as camadas existentes nesta arquitetura (Walpita, 2020):

- **Camada de Apresentação:** Esta camada é constituída por todas as componentes associadas à camada de apresentação.
- **Camada de Negócio**: Esta camada é constituída por todas as componentes relacionadas com a lógica do negócio e responsável por executar as regras.

- **Camada de Persistência**: Esta camada é responsável por lidar com as operações de leitura e escrita de dados.
- **Camada de Base de Dados**: Esta camada é responsável armazenar os dados.

Para além de ter um baixo custo de implementação, como já foi referido, algumas das vantagens desta arquitetura são: a simplicidade e facilidade de implementação, a simplificação dos testes, entre outras (Walpita, 2020).

4.3.2 Arquitetura do Sistema

O Grouplanner é constituído por cinco microsserviços distintos. Esses microsserviços estão implantados no servidor *Microsoft Azure*, garantindo uma infraestrutura de alto desempenho. Cada microsserviço interage com os outros, graças à troca contínua de solicitações por meio de pontos HTTP REST. Cada um deles apresenta características únicas e essenciais que contribuem para o funcionamento global do sistema (Alves et al., 2022; Alves et al., 2023a).

Os microsserviços existentes no sistema são os seguintes (Alves et al., 2022; Alves et al., 2023a):

- ***User Management MS*** - Encarregue de gerir e guardar os dados dos utilizadores registados, abrangendo desde informações demográficas, traços de personalidade e limitações físicas, como deficiências auditivas, visuais e cardíacas, até receios/fobias, como o medo de alturas (acrofobia), espaços confinados (claustrofobia) e lugares lotados (agorafobia). Além disso, a plataforma contempla preferências por atrações turísticas, gostos e preocupações relacionadas com viagens, histórico de pontos de interesse visitados ou não, avaliações de pontos de interesse e outras informações pertinentes ao perfil de cada utilizador.
- ***Multi-Agent MS*** **(MAMS)** - Este microsserviço é constituído por agentes criados no ambiente do sistema multiagente sempre que um novo utilizador turista se regista na aplicação ou quando um novo grupo de excursão é criado. Cada utilizador é representado por um agente inteligente modelado de acordo com o respetivo perfil, incorporando as suas preferências e características individuais. O MAMS desempenha um papel fundamental ao criar agrupamentos de turistas com personalidades semelhantes, permitindo uma experiência turística mais personalizada e enriquecedora. No caso de grupos de excursão, o sistema divide o grupo principal em subgrupos com base nesses agrupamentos de perfis. Esta abordagem garante uma otimização da experiência do utilizador ao considerar as dinâmicas interpessoais e interesses individuais dos turistas.
- ***Recommendation Engine MS*** - Desempenha um papel crucial ao determinar os pontos de interesse a recomendar aos turistas, com base numa análise aprofundada das suas preferências por atrações turísticas, considerando também as suas escolhas e preocupações em relação às viagens. Além disso, o sistema leva em conta as

possíveis limitações físicas, receios/fobias individuais e quaisquer restrições previamente identificadas pelo MAMS. A singularidade deste sistema reside na inclusão do contexto climático, ajustando as recomendações de acordo com as condições climáticas predominantes, bem como no contexto da cidade onde o utilizador planeia viajar.
Ao combinar informações diversificadas e precisas, o sistema é capaz de oferecer sugestões com um elevado grau de personalização e relevância, permitindo que os turistas possam desfrutar de experiências que se alinham não apenas com os seus interesses, mas também com o ambiente específico da cidade de destino e com as condições climáticas previstas.

Isto resulta em viagens mais ajustadas aos gostos individuais do utilizador, proporcionando aos utilizadores uma conexão mais significativa com os destinos e atividades recomendados. Além disso, a consideração das condições climáticas e do contexto da cidade contribui para uma experiência de viagem mais imersiva.

- ***POI MS*** – Este microsserviço inclui todos os pontos de Interesse a serem sugeridos, obtidos por meio da *API* do *TripAdvisor* (atualmente, apenas pontos de interesse do norte de Portugal e de Istambul estão disponíveis na aplicação). Os pontos de interesse foram categorizados com base em onze categorias de turismo (Alves et al., 2023b), nas quatro preferências e preocupações relacionadas com viagens e ainda se eram apropriados para pessoas com as deficiências e/ou receios/fobias considerados. Estas classificações foram primeiramente realizadas pelos autores e posteriormente validadas por três especialistas da área do turismo, garantindo a qualidade das sugestões oferecidas (Alves et al., 2023b).
  Para além disso, o POI MS estabelece uma comunicação direta com a *API* do Google, permitindo que um utilizador registe um ponto de interesse já visitado ou receba informações atualizadas sobre os pontos de interesse nas listas de recomendação.
- ***Social Network MS*** – Este microsserviço desempenha um papel fundamental ao gerir as informações de grupos e subgrupos, bem como as conversas entre utilizadores. Essa gestão é essencial para futuras consultas dentro da aplicação e para a apresentação das informações aos utilizadores. Além disso, o microsserviço possibilita a postagem de comentários num *chat*, permitindo assim uma discussão sobre os pontos de interesse. Dessa maneira, este microsserviço permite criar um espaço colaborativo onde os utilizadores podem partilhar experiências e trocar ideias sobre os locais visitados e a visitar.

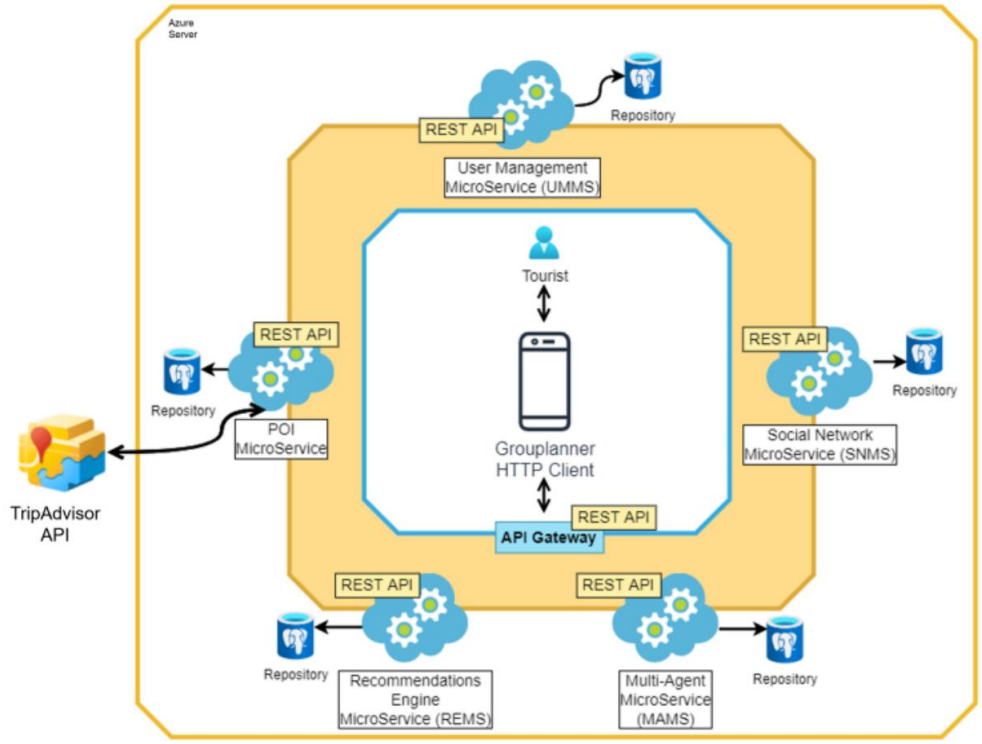


Figura 19 – Arquitetura de microsserviços do Grouplanner, retirada de Alves, et al., (2022)

4.3.2.1 Vistas Lógicas

Na figura 20, é possível observar o diagrama de vista lógica com as diferentes componentes que constituem o sistema na sua totalidade.

No diagrama, para além da componente da UI (Grouplanner App), estão presentes cinco microsserviços: *Multi-Agent Microservice, User Management Microservice, Social Network Microservice, POI Microservice e Recommendation Engine Microservice*. Cada microsserviço desempenha funções específicas e comunica com outros, trocando informações.

O *Recommendation Engine Microservice* consome dados do *POI Microservice* e da *API* da Sistrade (responsável por fornecer os dados meteorológicos) de forma a efetuar as suas funções. O MAMS depende dos dados fornecidos pelos *User Management Microservice, POI Microservice* e pelo *Recommendation Engine Microservice* de forma a realizar as suas tarefas.

Para além disso, o *Social Network Microservice* obtém dados do MAMS, e o *User Management Microservice* consome informações fornecidas pelo MAMS.

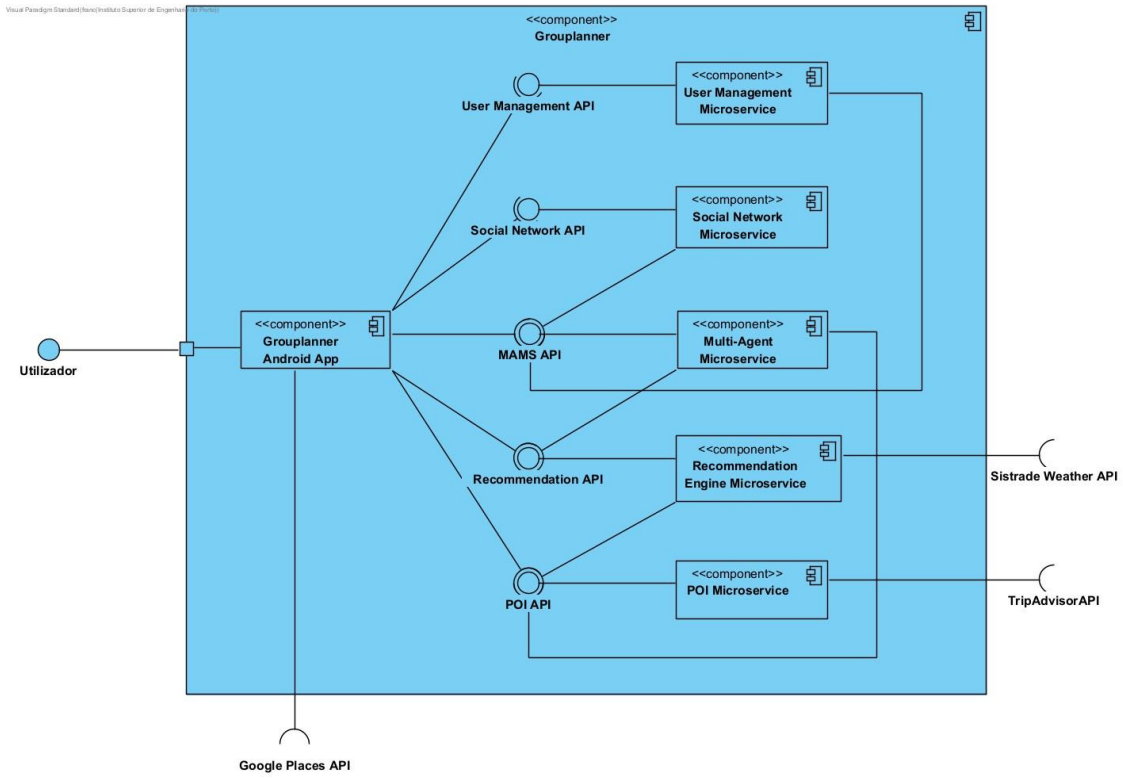


Figura 20 - Diagrama de Vista Lógica do Sistema

Na figura 21 é possível observar as componentes que constituem o microsserviço *Recommendation Engine Microservice*, que foi principalmente sobre o qual incidiram os desenvolvimentos relacionados com este projeto.

Os *Controllers* recebem pedidos e redirecionam as informações específicas para os *Services*.

Os *Services* contêm a lógica principal do negócio, interagem com a camada dos *Repositories* e possuem o conhecimento dos objetos que pertencem à componente dos *Models*.

A componente dos *Models*, possui entidades com as quais o sistema trabalha.

Já os DTO, são utilizados para a transferência de dados entre as diferentes componentes.

Os *Repositories* realizam operações CRUD, ou seja, de criação, leitura, atualização e exclusão de informação presente na base de dados.

Por fim, os *Mappers* fazem um mapeamento dos objetos *Model* para DTO e de DTO para *Model.*

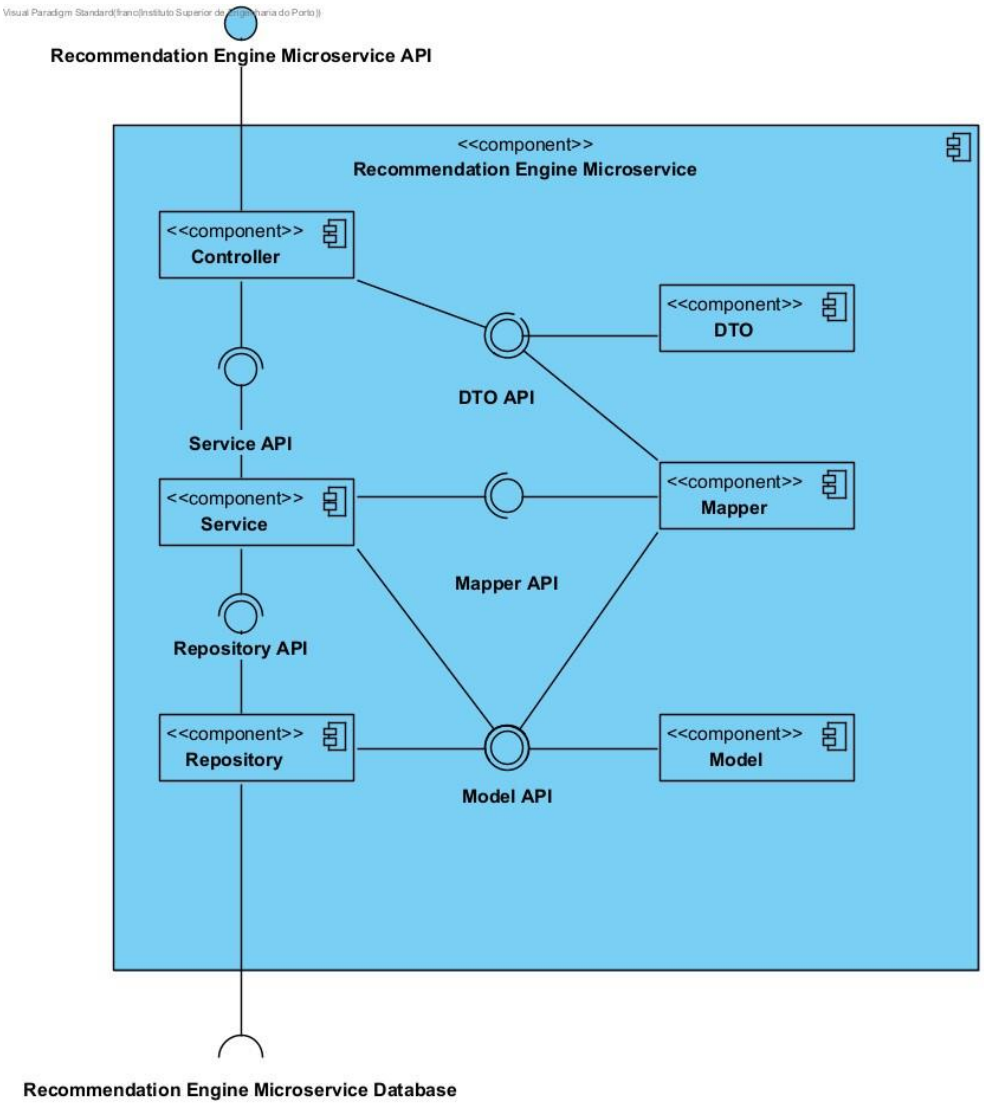


Figura 21 - Diagrama de Vista Lógica do *Recommendation Engine Microservice*

4.3.2.2 Vista Física

Na figura 22 é possível observar um diagrama de vista física que mostra a estrutura e a comunicação entre as diferentes partes do sistema.

No início, temos uma aplicação *mobile* que se encontra hospedada no dispositivo móvel do utilizador.

A aplicação móvel comunica diretamente com os cinco microsserviços distintos presentes no sistema. Estes microsserviços estão hospedados na plataforma *Microsoft Azure,* por decisão das pessoas responsáveis pelo projeto. A comunicação entre a aplicação e os microsserviços é feita através do protocolo HTTPS, permitindo que os dados transmitidos sejam seguros.

Existem três bases de dados para os cinco microsserviços, sendo que os microsserviços *POI Microservice*, *User Management Microservice* e *Social Network Microservice* partilham a mesma base de dados. Para a hospedagem da base de dados foi utilizado o *ElephantSQL*. A comunicação entre os microsserviços e as suas bases de dados é realizada através do protocolo TCP/IP, o que permite uma transmissão de dados fiável.

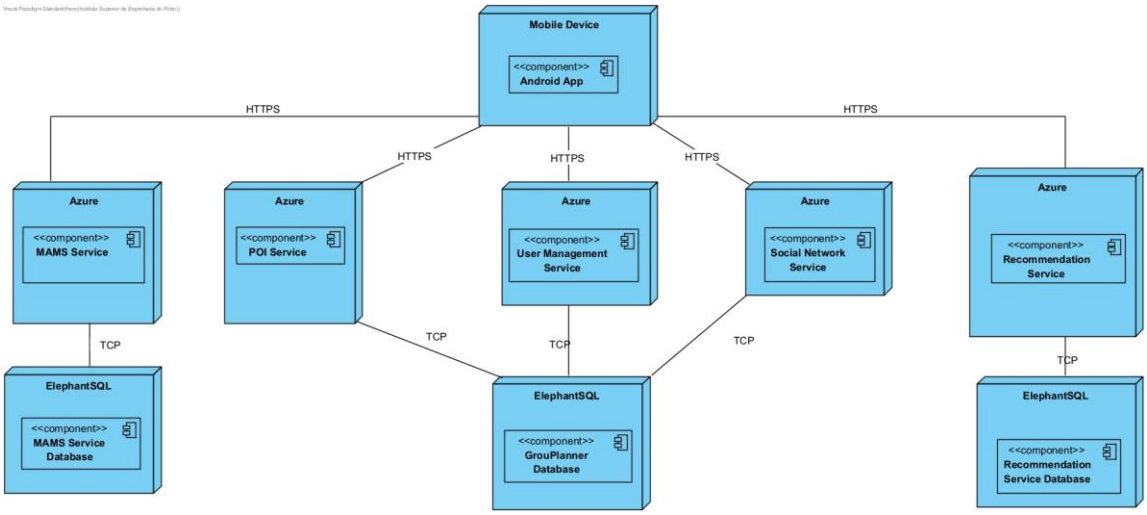


Figura 22 - Diagrama de Vista Física do Sistema

4.3.3 Diagramas de sequência

Nesta secção são apresentados os diagramas de sequência mais relevantes, que representam as funcionalidades desenvolvidas. Estes explicam os processos do sistema e como comunicam, e centram-se no comportamento do sistema em tempo de execução.

4.3.3.1 Integração com o Serviço Meteorológico Externo para Recomendação de Pontos de Interesse

Na figura 69, presente na secção de anexos, é possível observar o fluxo do caso de uso de integração do serviço meteorológico.

Quando o Sistema executa o método *IsAvailableDateString* presente no microsserviço *Recommendation Engine Microservice*, envia para ele o ponto de interesse que vai ser avaliado a nível de condições meteorológicas e as datas de início e de fim da excursão.

Se a diferença entre a data atual e a data de início da excursão for superior a cinco dias, então o ponto de interesse que está a ser avaliado é considerado como disponível. Isto acontece porque só é possível obter as previsões meteorológicas até cinco dias antes da data início da excursão.

Por outro lado, se a diferença for inferior a cinco dias, dá-se então a continuação do método. É feita uma chamada ao serviço externo *WeatherAPI*, que fornece os dados meteorológicos, e

verifica se as condições no ponto de interesse em questão são desfavoráveis no período das 8h às 18h. Se sim, e se esse ponto apenas suportar visitas ao ar livre, então considera-se que esse ponto de interesse não é recomendável a visitas.

4.3.3.2 Gerar Recomendações Individuais Baseadas em Fatores Contextuais com Opção de Substituição de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

Nas figuras 70 e 71 (ver Anexos) é possível observar o fluxo do caso de uso de pedido de recomendações individual contextual com opção de substituição de pontos de interesse sob condições meteorológicas desfavoráveis.

Quando o Turista faz um pedido de recomendações individual na aplicação, o seu primeiro contacto é feito com o microsserviço do MAMS, nomeadamente com a classe *MASService*. Nesta, o método *RequestIndividualRecommendationWeather* é executado. Este método vai fazer uma chamada a um recurso presente no *RecommendationController* com o nome *AssociateIndividualOpinionRecommendationsWeather* do microsserviço *Recommendations Service*, que é o microsserviço onde se desenvolveu a lógica das recomendações. O *RecommendationController* implementa o método *AssociateIndividualRecommendationsOpinionWeather*. Neste, é feita uma chamada ao método *AssociateIndividualOpinionRecommendationsWeather*. Esta função vai verificar cada ponto existente na base de dados e excluir aqueles que devem ser excluídos de acordo com os medos, limitações e preferências do utilizador.

Para além disso, dentro desta função, é invocada outra função chamada *POICategoryToOpinionRecommendationsWeather*. Nesta é aplicada a lógica da atribuição dos pontos de interesse relevantes e da sugestão de pontos alternativos em relação a pontos onde as condições meteorológicas são desfavoráveis.

4.3.3.3 Gerar Recomendações Para Grupos Baseadas em Fatores Contextuais Com Opção de Substituição de Pontos de Interesse Sob Condições Meteorológicas Desfavoráveis

Na figura 72 (ver Anexos) é possível observar o fluxo do caso de uso de pedido de recomendação de grupo contextual com opção de substituição de pontos de interesse sob condições meteorológicas desfavoráveis.

Quando o Turista Líder de Grupo faz um pedido de recomendações de grupo na aplicação, o seu primeiro contacto é feito com a classe *MASService*, presente no microsserviço do MAMS. Nesta, o método *RequestRecommendationWeather* é executado. Este método faz uma chamada a um recurso presente no *RecommendationController* com o nome *AssociateOpinionRecommendationsWeather* do *microsserviço Recommendation Engine Microservice*, que é o microsserviço onde se desenvolveu a lógica das recomendações. O *RecommendationController* implementa o método *AssociateOpinionRecommendationsWeather*. Neste, é feita uma chamada ao método *AssociateIndividualOpinionRecommendationsWeather*. Esta função verifica cada ponto de interesse existente na base de dados e exclui aqueles que devem ser excluídos de acordo com os medos, limitações e preferências dos utilizadores de cada subgrupo.

Para além disso, dentro desta função, é invocada outra função chamada *POICategoryToOpinionRecommendationsWeather*, a mesma utilizada no pedido de recomendações individual. Nesta é aplicada a lógica da atribuição dos pontos de interesse relevantes e da sugestão de pontos alternativos em relação a pontos onde as condições meteorológicas são desfavoráveis, para cada subgrupo.

4.3.3.4 Atualização Diária de Recomendações e Integração com o *VirtualPet*

Quando um Turista faz um pedido de recomendações, o sistema tem de executar um método chamado *startWorker* que dá início ao trabalho presente na classe *IndividualDailyRecommendationWorker*, que é executado de 24 em 24 horas. A lógica presente neste trabalho consiste em fazer uma atualização diária das recomendações de acordo com a situação meteorológica, até chegar ao dia do fim da excursão do turista individual ou de grupo e enviar essa informação atualizada para o *VirtualPet*, cuja lógica se encontra na *VirtualPetActivity*. Para isso, é feita uma chamada à API responsável por gerar as recomendações, transmitindo-as para o *VirtualPet*, dando assim hipótese ao utilizador de substituir os pontos de interesse onde as condições de tempo são desfavoráveis, por opções consideradas mais “confortáveis” para o turista ou grupo (ver figura 73, presente nos Anexos).

5 Implementação

Neste capítulo é descrito o processo de implementação das componentes e das funcionalidades desenvolvidas. Para desenvolver as funcionalidades requisitadas, foi necessário adaptar código já existente em certos microsserviços, nomeadamente nos microsserviços *Multi-Agent Microservice* e no *Recommendation Engine Microservice*. Foram criados métodos descritos de seguida, com o intuito de cumprir os requisitos especificados, utilizando a linguagem de programação C#. Para além disso, foram feitas adaptações e alterações no cliente HTTP, nomeadamente na aplicação Grouplanner, envolvendo principalmente as activities *IndividualRecommendationsActivity* e *GroupNavigationActivity,* utilizando a linguagem Java. O presente capítulo está dividido em várias secções, sendo que cada uma representa uma funcionalidade e detalha a maneira como esta foi desenvolvida.

5.1 Integração com o Serviço Meteorológico Externo para Recomendação de Pontos de Interesse

O método *IsAvailableDateString* é uma função assíncrona com uma grande importância para outras funcionalidades que façam uso deste método. Esta função é responsável por "averiguar" a disponibilidade de um ponto de interesse com base nas condições meteorológicas previstas. Este método foi construído de forma a explorar um certo número de critérios e regras que têm de ser cumpridos para poder fazer uma recomendação "acertada" de acordo com o contexto meteorológico.

Como se pode verificar na figura 23, este método recebe no seu cabeçalho um objeto do tipo POI, uma data de início e uma de fim no formato *string*. O objeto POI é utilizado para verificar de que tipo de ponto de interesse estamos a falar a nível de "infraestruturas", ou seja, se é um ponto de interesse que apenas pode ser visitado ao ar livre, ou se é capaz de ser visitado sem ser afetado por condições climatéricas adversas, o que pode ser verificado através do atributo indoorOutdoor. Já as datas, são utilizadas para verificar as condições climatéricas

para esse período. Este método retorna um tuplo com o par boolean *string*. Se o valor booleano assumir o valor *true*, então é porque o ponto de interesse em questão está disponível para ser visitado, enquanto se o valor for *false*, não se encontra disponível. A *string* pode assumir dois valores sendo eles uma *string* vazia (quando o ponto de interesse se encontra disponível) ou uma frase genérica que alerta sobre a possibilidade das más condições climatéricas (quando o ponto de interesse não se encontra disponível).

```
public async Task<Tuple<bool, string>> IsAvailableDateString(POI poi, string startDate, string endDate)
{
```

Figura 23 - Cabeçalho do Método *IsAvailableDateString*

No início do método, são inicializadas várias variáveis que ajudam a converter as datas recebidas em *string* para o formato data/hora Unix, que é uma representação do tempo medido em segundos desde a data 1 de janeiro de 1970 (ver Figura 24). Este formato facilita na comparação e no cálculo de datas.

```
var startDateEpoch = DateTimeOffset.ParseExact(startDate, "yyyy/MM/dd", CultureInfo.InvariantCulture).ToUnixTimeSeconds();
var endDateEpoch = DateTimeOffset.ParseExact(endDate, "yyyy/MM/dd", CultureInfo.InvariantCulture).ToUnixTimeSeconds() + (24 * 60 * 60)

var currentDate = DateTimeOffset.Now.ToUnixTimeSeconds();
var differenceInSeconds = startDateEpoch - currentDate;
var differenceInDays = TimeSpan.FromSeconds(differenceInSeconds).TotalDays;




var startDateEpoch2 = DateTimeOffset.ParseExact(startDate, "yyyy/MM/dd", CultureInfo.InvariantCulture).Date;
var endDateEpoch2 = DateTimeOffset.ParseExact(endDate, "yyyy/MM/dd", CultureInfo.InvariantCulture).Date;

var currentDate2 = DateTimeOffset.Now.Date;
var differenceInDays2 = (startDateEpoch2 - currentDate2).TotalDays;
```

Figura 24 - Conversão de Datas

De seguida, o método faz uma verificação para ver se a data de início da excursão ultrapassa por mais de cinco dias a data atual (instante em que se faz o pedido de recomendação) ou se a diferença entre a data de início e a data atual é inferior a zero (este caso só acontecerá se existir um erro no sistema). Se alguma destas condições for satisfeita, o método devolve o tuplo com o valor booleano a verdadeiro, e a *string* de justificação a vazio uma vez que não é preciso estar a justificar nada, já que o ponto se encontra disponível (ver Figura 25).

```
if (differenceInDays2 > 5 || differenceInDays2 < 0)
{
    return Tuple.Create(true, string.Empty);
}
```

Figura 25 - Verificação da Data de Início da Excursão

De seguida, constrói-se um URL que contém informações sobre a localização do ponto de interesse a partir do objeto POI (neste momento só se está a ter em conta pontos de interesse do norte de Portugal). Este URL é utilizado para fazer uma chamada à API externa da Sistrade, que fornece informações meteorológicas para os próximos cinco dias em relação à data em

que é feito o pedido (ver Figura 26). Na secção de anexos é possível observar a figura 74 que mostra os campos do JSON devolvido quando se faz a chamada à API externa.

```
var city = poi.city.Name;
var countryCode = "PT";
var location = city + "," + countryCode;

var url = $"https://smartravelapp.sistrade.com/WeatherAPI/fiveDaysForecast?location={location}";

try
{
    var response = await client.GetAsync(url);
    response.EnsureSuccessStatusCode();

    var json = await response.Content.ReadAsStringAsync();
    var weatherData = JsonConvert.DeserializeObject<WeatherForecast5DaysDTO>(json);

    string weatherJustification = "";

    var indoorOutdoor = poi.indoorOutdoor;

    bool isRecommended = true;
```

Figura 26 - Chamada à API da Sistrade

Por fim, após o resultado da chamada à API ser devolvido, um ciclo *for* percorre cada previsão dentro do intervalo da data de início e de fim da excursão. De referir que para cada dia são feitas oito previsões com um intervalo de três horas em relação à última previsão, ou seja, no total dos cinco dias, são feitas quarenta previsões meteorológicas.

Neste ciclo, são feitas várias verificações de forma a avaliar os códigos meteorológicos associados a cada previsão, fornecidos pela API da Sistrade. Estes códigos são identificadores de vários tipos de acontecimentos meteorológicos. Os códigos existentes são os seguintes:

- **Identificador 200 a 202**: Tempestades com chuva fraca a moderada
- **Identificador 211 a 232**: Tempestades de intensidade moderada a forte
- **Identificador 301 a 321**: Vários tipos de chuva ligeira
- **Identificador 500 a 531**: Vários tipos de chuva moderada a forte.
- **Identificador 602 a 622**: Queda de neve
- **Identificador 762 a 781**: Fenómenos meteorológicos extremos, como tornados, erupções, entre outros.

Se uma das previsões tiver um identificador igual aos que foram apresentados anteriormente e o ponto de interesse só oferecer condições de visita ao ar livre, o valor booleano do tuplo, que por omissão é verdadeiro, passa a false e uma justificação é criada para informar que existem previsões, no intervalo de datas da excursão, onde as condições meteorológicas não são favoráveis. Basta que haja uma previsão desfavorável dentro do período das 8h00 às 18h00 dos dias da excursão, para que o ponto de interesse seja "descartado" (ver Figura 27). Este descarte é feito noutras funcionalidades que invocam o método *IsAvailableDateString* e será explicado quando essas funcionalidades forem detalhadas nesta secção.

```
        foreach (var forecast in weatherData.weatherPrediction.List)
        {

            if ((forecast.dt >= startDateEpoch && forecast.dt <= endDateEpoch))
            {
                if ((forecast.Weather.Id >= 200 && forecast.Weather.Id <= 202) ||
                    (forecast.Weather.Id >= 211 && forecast.Weather.Id <= 232) ||
                    (forecast.Weather.Id >= 301 && forecast.Weather.Id <= 321) ||
                    (forecast.Weather.Id >= 500 && forecast.Weather.Id <= 531) ||
                    (forecast.Weather.Id >= 602 && forecast.Weather.Id <= 622) ||
                    (forecast.Weather.Id >= 762 && forecast.Weather.Id <= 781))
                {
                    if (indoorOutdoor == "outdoor")
                    {
                        var dtDateTime = DateTimeOffset.FromUnixTimeSeconds(forecast.dt).ToLocalTime();
                        if (dtDateTime.Hour >= (8 - 1) && dtDateTime.Hour < (18 - 1))
                        {
                            isRecommended = false;
                            weatherJustification = "Existem partes do dia onde a condição metereológica não é favorável";
                            break;
                        }
                    }
                }
            }
        }

        return Tuple.Create(isRecommended, weatherJustification.ToString());
    }
    catch (Exception)
    {
        return Tuple.Create(true, string.Empty);
    }
}
```

Figura 27 - Verificação das Condições Desfavoráveis

5.2 Gerar Recomendações Individuais Baseadas em Fatores Contextuais

5.2.1 Recomendações com Opção de Substituição de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

O processo de recomendações individual acontece quando um utilizador faz um pedido de recomendação através da aplicação Android. Esta ação desencadeia um pedido à API presente no microsserviço *MultiAgent Microservice*. Essa API é representada pelo recurso **mams/agents/tourists/{username}/requestRecommendationWeather**. No *UserController* é implementado o método que é executado quando é feito um pedido ao recurso. Este método é o *RequestRecommendationWeather*. No método do *Controller* é feita uma chamada à camada dos serviços, nomeadamente, ao método *RequestIndividualRecommendationWeather (ver Figura 28)*.

```
[AllowAnonymous]
[HttpPost("{username}/requestRecommendationWeather")]
0 references
public async Task<IActionResult> requestRecommendationWeather(AskIndividualRecommendationsDTO askIndividualRecommendationsDTO)
{
    try
    {
        //Request the recommendatiosn from the MAS
        var result = await masService.RequestIndividualRecommendationWeather(askIndividualRecommendationsDTO);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 28 - Método *RequestRecommendationWeather*

É importante referir que a lógica do método *RequestIndividualRecommendationWeather* presente no *MAS Service* não vai ser abordada, uma vez que essa funcionalidade foi desenvolvida por outro colega. Apenas se realça que dentro desse método é feita uma chamada ao recurso representado por **re/individualRecommendationsOpinionWeather** presente no *RecommendationController* do microsserviço *Recommendation Engine Microservice*, que é o microsserviço onde se desenvolveu a lógica das recomendações (ver Figura 29). O JSON que é passado quando se faz o pedido (demonstrado na figura 75, presente na secção de anexos), contem várias informações que são utilizadas ao longo do método, sendo as principais as seguintes: os traços de personalidade, as preferências de viagem, os medos, as limitações, localização para a qual o utilizador pretende receber recomendações, e a data de início e de fim da viagem.

```
myContent = JsonConvert.SerializeObject(opinionDTO);
buffer = Encoding.UTF8.GetBytes(myContent);
byteContent = new ByteArrayContent(buffer);
byteContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

result = await client.PostAsync("https://groupplanner-rems.azurewebsites.net/re/individualRecommendationsOpinionWeather", byteContent);
result.EnsureSuccessStatusCode();
json = await result.Content.ReadAsStringAsync();
ResponseIndividualRecommendationsWeatherMAMSDTO firstList2 = JsonConvert.DeserializeObject<ResponseIndividualRecommendationsWeatherMAMSDTO>(json);
```

Figura 29 - Chamada à API do *Recommendation Engine Microservice*

No microsserviço *Recommendation Service*, o *RecommendationController* implementa o método *AssociateIndividualRecommendationsOpinionWeather*. Neste, é feita uma chamada à camada dos serviços, nomeadamente ao método *AssociateIndividualOpinionRecommendationsWeather (ver Figura 30)*.

```
[HttpPost("individualRecommendationsOpinionWeather")]
[AllowAnonymous]
0 references
public async Task<IActionResult> AssociateIndividualRecommendationsOpinionWeather(IndividualRecommendationsAfterOpinionDTO dto)
{
    try
    {
        var result = await service.AssociateIndividualOpinionRecommendationsWeather(dto);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 30 - Método *AssociateIndividualRecommendationsOpinionWeather*

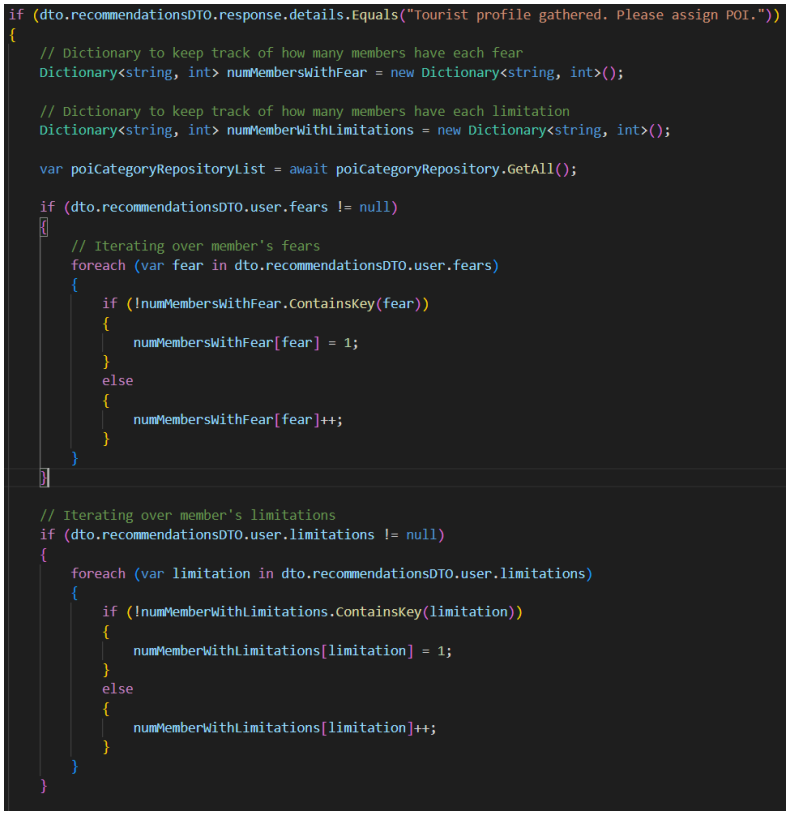
O método *AssociateIndividualOpinionRecommendationsWeather* do *RecommendationService* é uma função assíncrona que recebe no seu cabeçalho como parâmetro um DTO *IndividualRecommendationsAfterOpinionDTO* (ver Figura 31). Este DTO possui toda a informação que já foi referida anteriormente. O resultado devolvido por este método é um DTO do tipo *ResponseIndividualRecommendationsWeatherMAMSDTO*. Este DTO contém as recomendações de clima geradas, bem como sugestões alternativas semelhantes para os locais onde as condições climáticas não são favoráveis. Estas sugestões alternativas são fornecidas para garantir que o utilizador tenha opções viáveis, mesmo que o clima não seja ideal em algumas das recomendações originais.

```
public async Task<ResponseIndividualRecommendationsWeatherMAMSDTO> AssociateIndividualOpinionRecommendationsWeather(IndividualRecommendationsAfterOpinionDTO dto)
{
```

Figura 31 - Cabeçalho do Método *AssociateIndividualOpinionRecommnedationsWeather* da Classe *RecommendationService*

O método começa por verificar se a informação do perfil do turista foi recolhida e a partir daí dá-se o começo da atribuição de pontos de interesse (ver Figura 32).

Antes de avançar com a explicação, é importante referir que os medos e as limitações dos utilizadores são tratados como *strings*. Estes são recolhidos quando o utilizador cria um perfil, onde este preenche os seus dados pessoais. O método percorre os medos e as limitações do utilizador e, guarda em dois dicionários, um para cada tipo, a contagem de medos e limitações físicas (ver Figura 32).

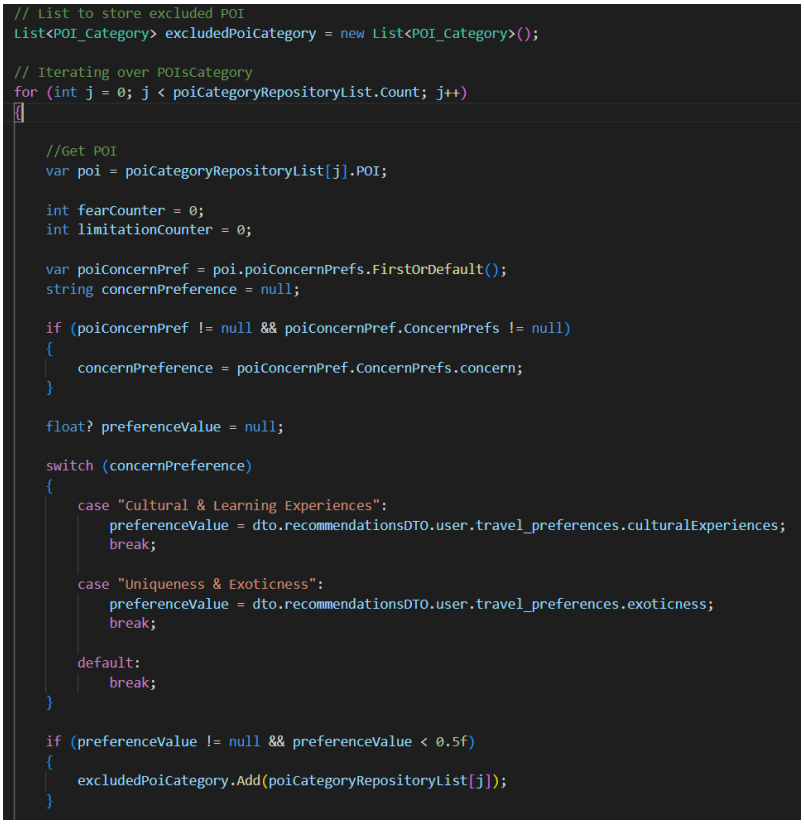


```
if (dto.recommendationsDTO.response.details.Equals("Tourist profile gathered. Please assign POI."))
{
    // Dictionary to keep track of how many members have each fear
    Dictionary<string, int> numMembersWithFear = new Dictionary<string, int>();

    // Dictionary to keep track of how many members have each limitation
    Dictionary<string, int> numMemberWithLimitations = new Dictionary<string, int>();

    var poiCategoryRepositoryList = await poiCategoryRepository.GetAll();

    if (dto.recommendationsDTO.user.fears != null)
    {
        // Iterating over member's fears
        foreach (var fear in dto.recommendationsDTO.user.fears)
        {
            if (!numMembersWithFear.ContainsKey(fear))
            {
                numMembersWithFear[fear] = 1;
            }
            else
            {
                numMembersWithFear[fear]++;
            }
        }
    }

    // Iterating over member's limitations
    if (dto.recommendationsDTO.user.limitations != null)
    {
        foreach (var limitation in dto.recommendationsDTO.user.limitations)
        {
            if (!numMemberWithLimitations.ContainsKey(limitation))
            {
                numMemberWithLimitations[limitation] = 1;
            }
            else
            {
                numMemberWithLimitations[limitation]++;
            }
        }
    }
```

Figura 32 - Verificação dos Medos e Limitações do Utilizador

De seguida, vai buscar todos os pontos de interesse ao repositório dos POI e percorre-os através de um ciclo *for*. Para cada ponto de interesse, as preferências de viagem do utilizador em relação a experiências culturais e exóticas são avaliadas. Se um ponto de interesse tiver associado a si as características de *Cultural & Learning Experiences* ou *Uniqueness & Exoticness*[1], o valor associado a essas características no utilizador é verificado (valor de 0 a 1). Se esse valor for inferior a 0.5, o ponto de interesse é adicionado a uma lista de pontos de interesse excluídos (ver Figura 33).

[1] Mais detalhes sobre as preferências de viagem podem ser consultados no artigo de Alves, et al. (2023b).



```
// List to store excluded POI
List<POI_Category> excludedPoiCategory = new List<POI_Category>();

// Iterating over POIsCategory
for (int j = 0; j < poiCategoryRepositoryList.Count; j++)
{

    //Get POI
    var poi = poiCategoryRepositoryList[j].POI;

    int fearCounter = 0;
    int limitationCounter = 0;

    var poiConcernPref = poi.poiConcernPrefs.FirstOrDefault();
    string concernPreference = null;

    if (poiConcernPref != null && poiConcernPref.ConcernPrefs != null)
    {
        concernPreference = poiConcernPref.ConcernPrefs.concern;
    }

    float? preferenceValue = null;

    switch (concernPreference)
    {
        case "Cultural & Learning Experiences":
            preferenceValue = dto.recommendationsDTO.user.travel_preferences.culturalExperiences;
            break;

        case "Uniqueness & Exoticness":
            preferenceValue = dto.recommendationsDTO.user.travel_preferences.exoticness;
            break;

        default:
            break;
    }

    if (preferenceValue != null && preferenceValue < 0.5f)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }
```

Figura 33 – Verificação das *Concern Preferences*

Para além disso, ainda durante esta fase, é verificado para cada ponto de interesse se as características desse ponto podem provocar medos ou limitações que estejam associados a esse utilizador[2]. Se sim, e se esse ponto ainda não foi adicionado à lista de pontos excluídos, então é adicionado (ver Figura 34).

[2] Mais detalhes em Alves et al. (2023b).

```
if (!excludedPoiCategory.Contains(poiCategoryRepositoryList[j]))
{

    // Get POI fears
    if (poi.fears != null)
    {
        List<string> poiFears = poi.fears;

        // Iterating over associated POI fears
        if (poiFears != null)
        {
            Console.WriteLine(poi.name);
            foreach (var fear in poiFears)
            {
                if (numMembersWithFear.ContainsKey(fear))
                {
                    fearCounter += numMembersWithFear[fear];
                }
            }
        }


    }

    if (poi.limitations != null)
    {

        // Get POI limitations
        List<string> poiAccessibility = poi.limitations;

        // Iterating over POI Accessibility

        foreach (var limitation in poiAccessibility)
        {
            if (numMemberWithLimitations.ContainsKey(limitation))
            {
                limitationCounter += numMemberWithLimitations[limitation];
            }
        }
    }

    // check if a member has that fear, if yes it exludes the poi
    if (fearCounter > 0)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }

    // check if a member has that limitation, if yes it exludes the poi
    if (limitationCounter > 0)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }
}
```

Figura 34 - Verificação dos Medos e Limitações dos Pontos de Interesse

Após estas verificações, o método identifica as categorias de turismo relevantes para o utilizador, através do método *FindHighestValueTourismCategories*, que não é relevante para esta explicação (ver Figura 35).

```
Dictionary<string, float> categories = Utils.Utils.FindHighestValueTourismCategories(dto.recommendationsDTO.user.tourismCategories);
```

Figura 35 - Método *FindHighestValueTourismCategories*

Depois disso, é invocado um método separado, com o nome *POICategoryToOpinionRecommendationsWeather* (Figura 36). Este método é responsável por gerar as recomendações propriamente ditas tendo em conta o clima, as categorias de turismo identificadas (Alves et al. 2022; Alves et al. 2023b), a lista de pontos de interesse excluídos, as datas de início e fim da excursão e a localização. Este método trata-se de uma função assíncrona que recebe no seu cabeçalho informação passada pelo método *AssociateIndividualOpinionRecommendationsWeather*. Da informação passada, destacam-se a lista de categorias de turismo relevantes, a lista dos pontos de interesse previamente

excluídos, a localização onde vai decorrer a excursão (país e cidade) e as datas de início e de fim da excursão. Esta função devolve um DTO chamado *WeatherRecommendationResultDTO* que contém as recomendações geradas, bem como as sugestões alternativas para as recomendações onde as condições climáticas não são favoráveis.

O método começa por definir várias listas que são responsáveis por armazenar as recomendações, as recomendações alternativas e os respetivos identificadores de cada uma destas recomendações. Para além disso, também é definido um dicionário que é uma cópia do dicionário das categorias associadas ao utilizador que foi passado por parâmetro. Esta cópia serve para determinar o número de pontos de interesse que vai ser atribuído a cada categoria (ver Figura 36).

Figura 36 - Cabeçalho do Método *POICategoryToOpinionRecommendationWeather*

De seguida, aplica-se uma lógica para definir quantos pontos de interesse devem ser atribuídos a cada categoria. Esta lógica seguiu uma estratégia dinâmica, onde as categorias com pontuações mais altas recebem mais pontos de interesse (apenas são consideradas as três primeiras categorias). Para isso, são somados os valores de todos os *scores* das categorias a uma única variável, e calculado o número de pontos de interesse para cada categoria com base na proporção em relação ao score total, sendo recomendados dez pontos de interesse, consoante o arredondamento da divisão inteira (ver Figura 37).

```
float scoresSum = 0;


// sums up all the scores of each category in the dictionary
for (int i = 0; i < MAX_CATEGORIES; i++)
{
    scoresSum = scoresSum + categories.ElementAt(i).Value;
    Console.WriteLine("scoresSum: " + scoresSum);
}


for (int i = 0; i < MAX_CATEGORIES; i++)
{
    // for each category, divide the score value by scoresSum, to get the % that this
    // score represents for the user, this way we know how many pois to assign to each category
    // e.g.: 0.8 / 2.1 = 0.38*10 = 4 poi, 0.8 / 2.1 = 0.38*10 = 4 poi, and 0.5 / 2.1 = 0.24*10 = 2 poi
    categoriesNumPOI[categoriesNumPOI.ElementAt(i).Key] =
        (float)Math.Ceiling(categoriesNumPOI[categoriesNumPOI.ElementAt(i).Key] / scoresSum * NUM_POI);

}
```

Figura 37 - Definição do Número de Pontos de Interesse a Recomendar para cada Categoria

Posteriormente, para cada categoria vai-se ao repositório *PoiCategoryRepository* recolher as recomendações. Para isto, é invocado um método, *GetCategoryByNameAndNumberOfResults*, que é no fundo uma *query* que vai buscar os pontos de interesse de acordo com vários parâmetros. De todos os parâmetros que são utilizados para filtrar os pontos de interesse que vão ser selecionados, estes são os mais relevantes: filtra os pontos de interesse com a categoria que é passada por parâmetro; filtra os pontos de interesse com o nome da cidade e nome do país que são passados por parâmetro; filtra os pontos de interesse de acordo se já estão na lista de recomendações para evitar repetições em futuras iterações (ver Figura 38).

```
for (int i = 0; i < numberOfCategories; i++)
{
    poiCount = 0;


    List<POI_Category> res = await poiCategoryRepository.GetCategoryByNameAndNumberOfResults(categoriesNumPOI.ElementAt(i).Key,
        (int)(categoriesNumPOI.ElementAt(i).Value + unwanted.Count + excludedPoiCategory.Count), listRecommendationsIds, locationCountry, locationCity);


    string justification = Utils.Utils.CreateDescriptionJustification(categories.ElementAt(i).Key, "recommendation");
```

Figura 38 - Obtenção dos Pontos de Interesse

Depois de recolher os pontos através do método auxiliar, a função tem em conta os pontos de interesse que já foram visitados no passado pelo utilizador. Se já tiver sido visitado, então passa-se esse ponto para o fim da lista de recomendações, garantindo assim, que só se não estiverem disponíveis mais pontos de interesse, pela razão que seja, que este seja sugerido. Isto é feito fazendo uma chamada a uma API do *POI Microservice* que vai devolver os pontos de interesse visitados por um determinado utilizador e colocando esses pontos no fim da lista de pontos de interesse recolhidos (ver Figura 39).

```
var nickname = usersIds.ElementAt(0);
// Call to the GetVisitedPoiByUser api
var url = $"https://localhost:5007/ps/poi/{nickname}";

var response = await client.GetAsync(url);
response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();
var visitedPoiData = JsonConvert.DeserializeObject<List<InfoVisitedPOIDTO>>(json);

List<POI_Category> visitedPOICategoriesInRes =
res.Where(poiCategory => visitedPoiData
    .Any(visitedPOI => Utils.Utils.CriarRegexParaTexto(visitedPOI.name)
            .IsMatch(Utils.Utils.RemoveAccents(poiCategory.POI.name))))
    .ToList();

res.RemoveAll(poiCategory => visitedPoiData.Any(visitedPOI => Utils.Utils.CriarRegexParaTexto(visitedPOI.name).
        IsMatch(Utils.Utils.RemoveAccents(poiCategory.POI.name))));

res.AddRange(visitedPOICategoriesInRes);
```

Figura 39 - Verificação dos Pontos de Interesse já Visitados

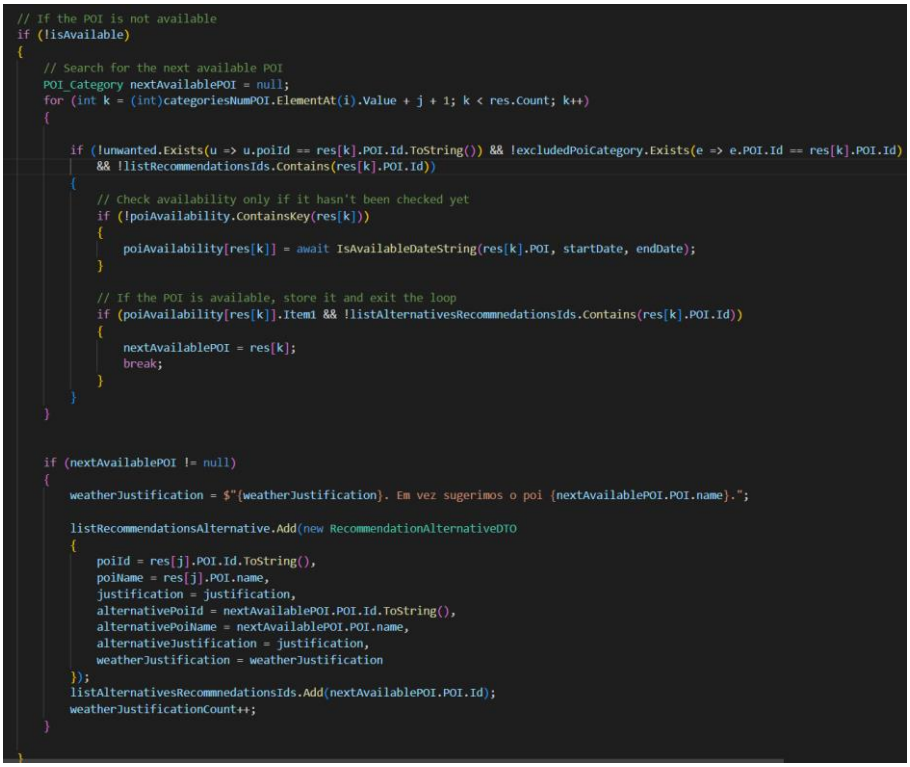
Ainda dentro da mesma iteração, é criado um dicionário com o nome *poiAvailability* que regista a disponibilidade de cada ponto de interesse. A lista de pontos de interesse obtida é percorrida e para cada ponto de interesse é verificado se este não se encontra na lista de pontos excluídos, na lista de pontos indesejados ou na lista de pontos que já foram

recomendados. Passando por estas verificações, examina-se se a disponibilidade desse ponto já foi verificada, verificando se esse ponto já se encontra presente no dicionário *poiAvailability*. Se não estiver presente, o método *IsAvailableDateString* (explicado anteriormente) vai ser invocado para se verificar a disponibilidade do ponto de interesse a nível de condições climáticas para as datas que o utilizador definiu, guardando o resultado no dicionário *poiAvailability* (ver Figura 40).

```
// Create a dictionary to hold the availability of each POI
Dictionary<POI_Category, Tuple<bool, string>> poiAvailability = new Dictionary<POI_Category, Tuple<bool, string>>();
int weatherJustificationCount = 0;
for (int j = 0; j < res.Count; j++)
{

    //If there is no need for more POIs, break
    if (0 == (int)categoriesNumPOI.ElementAt(i).Value)
    {
        break;
    }
    //Only adds to recommendation list if it is not an unwanted POI
    if (!unwanted.Exists(u => u.poiId == res[j].POI.Id.ToString()) &&
        !excludedPoiCategory.Exists(e => e.POI.Id == res[j].POI.Id) && !listRecommendationsIds.Contains(res[j].POI.Id))
    {

        // Check availability only if it hasn't been checked yet
        if (!poiAvailability.ContainsKey(res[j]))
        {
            poiAvailability[res[j]] = await IsAvailableDateString(res[j].POI, startDate, endDate);
        }

        var availabilityResult = poiAvailability[res[j]];
        bool isAvailable = availabilityResult.Item1;
        string weatherJustification = availabilityResult.Item2;
```

Figura 40 - Verificação da Disponibilidade de um Ponto de Interesse

Se o ponto estiver indisponível devido à situação climática, pontos de interesse alternativos são procurados e, quando um for encontrado, é criado um DTO chamado *RecommendationAlternativeDTO* que possui algumas informações, sendo as seguintes as mais relevantes: *poiId* (o identificador do ponto de interesse original que não está disponível); *poiName* (o nome do ponto de interesse original que não está disponível); *alternativePoiId* (o identificador do ponto de interesse alternativo ao original); *alternativePoiName* (o nome do ponto de interesse alternativo ao original); *weatherJustification* (justificação do ponto original ser substituído por outro, detalhando as condições climáticas). Este DTO criado é adicionado à lista de recomendações alternativas (ver Figura 41).



```
// If the POI is not available
if (!isAvailable)
{
    // Search for the next available POI
    POI_Category nextAvailablePOI = null;
    for (int k = (int)categoriesNumPOI.ElementAt(i).Value + j + 1; k < res.Count; k++)
    {

        if (!unwanted.Exists(u => u.poiId == res[k].POI.Id.ToString()) && !excludedPoiCategory.Exists(e => e.POI.Id == res[k].POI.Id)
            && !listRecommendationsIds.Contains(res[k].POI.Id))
        {
            // Check availability only if it hasn't been checked yet
            if (!poiAvailability.ContainsKey(res[k]))
            {
                poiAvailability[res[k]] = await IsAvailableDateString(res[k].POI, startDate, endDate);
            }

            // If the POI is available, store it and exit the loop
            if (poiAvailability[res[k]].Item1 && !listAlternativesRecommnedationsIds.Contains(res[k].POI.Id))
            {
                nextAvailablePOI = res[k];
                break;
            }
        }
    }


    if (nextAvailablePOI != null)
    {
        weatherJustification = $"{weatherJustification}. Em vez sugerimos o poi {nextAvailablePOI.POI.name}.";

        listRecommendationsAlternative.Add(new RecommendationAlternativeDTO
        {
            poiId = res[j].POI.Id.ToString(),
            poiName = res[j].POI.name,
            justification = justification,
            alternativePoiId = nextAvailablePOI.POI.Id.ToString(),
            alternativePoiName = nextAvailablePOI.POI.name,
            alternativeJustification = justification,
            weatherJustification = weatherJustification
        });
        listAlternativesRecommnedationsIds.Add(nextAvailablePOI.POI.Id);
        weatherJustificationCount++;
    }
}
```

Figura 41 - Obtenção de um Ponto de Interesse Alternativo e Adição desse Ponto a Uma Lista de Pontos Alternativos

Por outro lado, se o ponto de interesse estiver disponível e não estiver presente na lista de alternativas, então é adicionado à lista principal de recomendações *listRecommendations*. Este processo termina quando o número de pontos de interesse para uma determinada categoria é atingido (ver figura 42).

```
        if (!listAlternativesRecommnedationsIds.Contains(res[j].POI.Id))
        {

            listRecommendations.Add(new RecommendationListDTO
            {
                order = order,
                poiId = res[j].POI.Id.ToString(),
                poiName = res[j].POI.name,
                predictedCompatibility = compatibility - 1,
                justification = justification
            });

            listRecommendationsIds.Add(res[j].POI.Id);
            order++;
            compatibility--;
            poiCount++;
        }
    }

    if (poiCount == (int)categoriesNumPOI.ElementAt(i).Value)
    {
        poiCount = 0;
        break;
    }
}
```

Figura 42 - Adição de um Ponto de Interesse à Lista de Recomendações

Por fim é retornado o DTO *WeatherRecommendationResultDTO* que possui a lista de recomendações original e a lista de recomendações alternativas (ver Figura 43).

```
return new WeatherRecommendationResultDTO
{
    recommendations = listRecommendations,
    alternatives = listRecommendationsAlternative
};
```

Figura 43 - Tipo de Resposta Devolvida pelo Método *POICategoryToOpionionRecommendationsWeather*

Após a conclusão da chamada do método *POICategoryToOpinionRecommendationsWeather*, volta-se ao método inicial *AssociateIndividualOpinionRecommendationsWeather* e o DTO do tipo *WeatherRecommendationResultDTO* devolvido pelo método auxiliar *POICategoryToOpinionRecommendationsWeather* é enviado para um método presente na classe *RecommendationMapper*, que vai mapear a informação recebida de maneira a construir a resposta final que é devolvida pelo método. Na figura 76 presente na secção de anexos é possível observar a estrutura do DTO que é construído e devolvido pelo *mapper*.

5.2.2 Recomendações com Eliminação Automática de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

O processo de recomendações individuais sem a opção de substituição de pontos de interesse onde existem condições meteorológicas desfavoráveis segue praticamente a mesma lógica descrita no ponto 5.2.1. Um Turista solicita recomendações individuais através da aplicação, desencadeando um pedido à *API* no microsserviço *Multi-Agent Microservice*, especificamente ao recurso mams/agents/tourists/{username}/ requestRecommendation. O método associado ao recurso, está presente no *UserController* e tem o nome de *RequestRecommendation (ver Figura 44)*.

```
[AllowAnonymous]
[HttpPost("{username}/requestRecommendation")]
0 references
public async Task<IActionResult> requestRecommendation(AskIndividualRecommendationsDTO askIndividualRecommendationsDTO)
{
    try
    {
        var result = await masService.RequestIndividualRecommendation(askIndividualRecommendationsDTO);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 44 - Método *RequestRecommendation*

Este método chama o *RequestIndividualRecommendation* na camada de serviços e a chamada dentro desse método ao microsserviço *Recommendation Engine Microservice* é feita ao recurso re/individualRecommendationsOpinion (ver Figura 45).

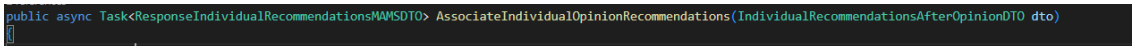


Figura 45 - Cabeçalho do Método *AssociateIndividualOpinionRecommendation* da Classe *RecommendationService*

A principal diferença entre a funcionalidade descrita no ponto 5.2.1 e esta, é que nesta é feita uma chamada ao método *POICategoryToOpinionRecommendations* em vez do *POICategoryToOpinionRecommendationsWeather*. No *POICategoryToOpinionRecommendations*, não existe a sugestão de pontos de interesse alternativos para substituir aqueles com condições meteorológicas desfavoráveis. Em vez disso, os pontos de interesse que não cumprem as condições climáticas ideais são excluídos da lista de recomendações finais (ver Figura 46).

```
if (isAvailable)
{
    listRecommendations.Add(new RecommendationListDTO
    {
        order = order,
        poiId = res[j].POI.Id.ToString(),
        poiName = res[j].POI.name,
        predictedCompatibility = compatibility - 1,
        justification = justification,
        city = res[j].POI.city.Name,
        country = res[j].POI.city.Country.Name
    });
    listRecommendationsIds.Add(res[j].POI.Id);
    order++;
    compatibility--;
    poiCount++;
}
```

Figura 46 - Verificação da Disponibilidade de um Ponto de Interesse

Após a conclusão da chamada ao método *POICategoryToOpinionRecommendations*, é devolvida uma lista com as recomendações obtidas. Esta é passada para o *mapper* *POIRecommendationsIndividual* que vai mapear a informação e devolver o resultado final, que é um DTO chamado *ResponseIndividualRecommendationsMAMSDTO*. Na figura 77 presente na secção de anexos, encontra-se a estrutura do DTO que é devolvido.

5.3 Gerar Recomendações Para Grupos Baseadas em Fatores Contextuais

5.3.1 Recomendações de Grupo com Opção de Substituição de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

O processo de recomendações de grupo acontece quando um líder de grupo faz um pedido de recomendação através da aplicação Android. Esta ação desencadeia um pedido à *API* presente no microsserviço *Multi-Agent Microservice*. Essa *API* é representada pelo recurso **mams/agents/groups/{group}/requestRecommendationWeather**. No *GroupController* é implementado o método que é executado quando é feito um pedido a esse recurso. Este método é o *RequestRecommendationWeather*. No método do *Controller* é feita uma chamada

à camada dos serviços sendo chamado o método *RequestIndividualRecommendationWeather (ver Figura 47)*.

```
[AllowAnonymous]
[HttpPost("{group}/requestRecommendationWeather")]
0 references
public async Task<IActionResult> RequestRecommendationWeather(string group, GroupDTO dto)
{
    try
    {
        // Thread.Sleep(6000);
        //Request the recommendatiosn from the MAS
        var result = await masService.RequestRecommendationWeather(dto);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 47 - Método *RequestRecommendationWeather* para Grupos

Dentro deste método é feita uma chamada ao recurso representado por **re/groupOpinionRecommendationsWeather** presente no *RecommendationController* do microsserviço *Recommendation Engine Microservice*. O JSON que é passado, quando se faz o pedido, contém várias informações que vão ser utilizadas ao longo do método, sendo as principais as seguintes: o nome do grupo; o identificador do grupo; o país e a cidade para onde o grupo pretende as recomendações; as datas de início e de fim da excursão; os subgrupos associados ao grupo em questão; informações de cada subgrupo tais como o identificador do subgrupo, o nome, lista de utilizadores, a personalidade, as categorias de turismo com mais relevo, as preferências de viagem, as motivações, os medos de cada membro e as limitações de cada membro.

No microsserviço *Recommendation Engine Microservice,* o *RecommendationController* implementa o método *AssociateOpinionRecommendationsWeather*. Neste é feita uma chamada à camada dos serviços, nomeadamente ao método *AssociateOpinionRecommendationsWeather (ver Figura 48)*.

Figura 48 - Método *AssociateOpinionRecommendationsWeather*

O método *AssociateOpinionRecommendationsWeather* do *RecommendationService* é uma função assíncrona que recebe no seu cabeçalho como parâmetro um DTO *GroupDivisionPOIMAMSAfterOpinionDTO (ver Figura 49)*. Este DTO possui toda a informação que já foi referida anteriormente. O resultado devolvido pelo método é um DTO do tipo *GroupDivisionPOIMAMSWeatherDTO*. Este DTO contém as recomendações de clima geradas, bem como sugestões alternativas para os locais onde as condições climáticas não são favoráveis. Estas sugestões alternativas são fornecidas para garantir que o utilizador tenha opções viáveis, mesmo que o clima não seja ideal em algumas das recomendações originais.

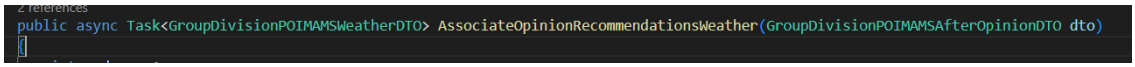
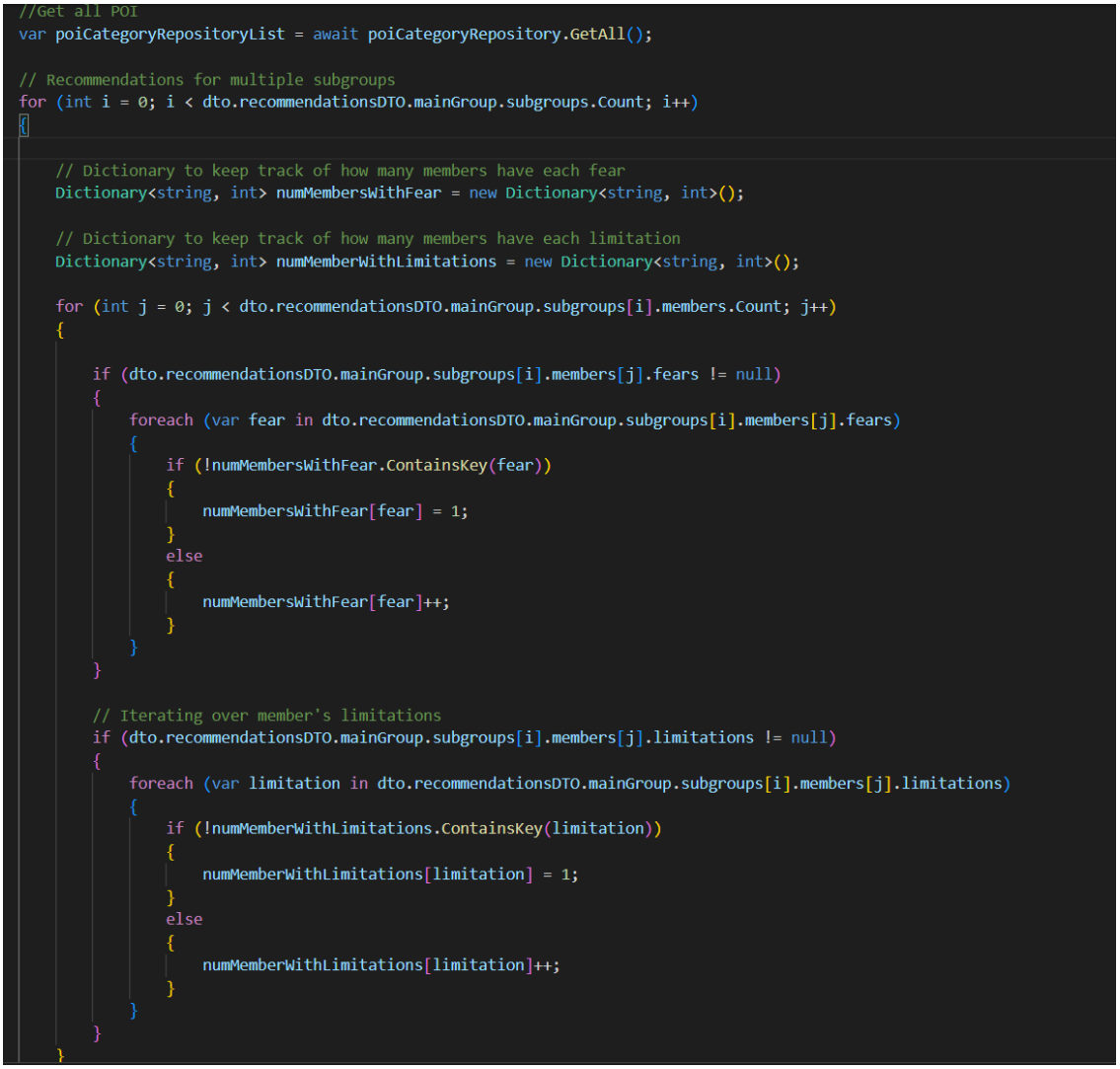


Figura 49 - Cabeçalho do Método *AssociateOpinionRecommendationWeather* da Classe *RecommendationService*

O método começa por verificar se os subgrupos foram formados e a partir daí dá-se o começo da atribuição de pontos de interesse a cada subgrupo.

O método percorre os medos e as limitações dos utilizadores de cada subgrupo e guarda em dois dicionários, um para cada tipo, a contagem de medos e limitações físicas (ver Figura 50).



```
//Get all POI
var poiCategoryRepositoryList = await poiCategoryRepository.GetAll();

// Recommendations for multiple subgroups
for (int i = 0; i < dto.recommendationsDTO.mainGroup.subgroups.Count; i++)
{

    // Dictionary to keep track of how many members have each fear
    Dictionary<string, int> numMembersWithFear = new Dictionary<string, int>();

    // Dictionary to keep track of how many members have each limitation
    Dictionary<string, int> numMemberWithLimitations = new Dictionary<string, int>();

    for (int j = 0; j < dto.recommendationsDTO.mainGroup.subgroups[i].members.Count; j++)
    {

        if (dto.recommendationsDTO.mainGroup.subgroups[i].members[j].fears != null)
        {
            foreach (var fear in dto.recommendationsDTO.mainGroup.subgroups[i].members[j].fears)
            {
                if (!numMembersWithFear.ContainsKey(fear))
                {
                    numMembersWithFear[fear] = 1;
                }
                else
                {
                    numMembersWithFear[fear]++;
                }
            }
        }

        // Iterating over member's limitations
        if (dto.recommendationsDTO.mainGroup.subgroups[i].members[j].limitations != null)
        {
            foreach (var limitation in dto.recommendationsDTO.mainGroup.subgroups[i].members[j].limitations)
            {
                if (!numMemberWithLimitations.ContainsKey(limitation))
                {
                    numMemberWithLimitations[limitation] = 1;
                }
                else
                {
                    numMemberWithLimitations[limitation]++;
                }
            }
        }
    }
```

Figura 50 - Verificação dos Medos e Limitações de Cada Utilizador de Um Subgrupo

De seguida, vai buscar todos os pontos de interesse ao repositório dos pontos de interesse e percorre-os através de um ciclo for. Para cada ponto de interesse, as preferências de viagem do grupo em relação a experiências culturais e exóticas são avaliadas. Se um ponto de interesse tiver associado a si as características de *Cultural & Learning Experiences* ou *Uniqueness & Exoticness*, o valor do subgrupo associado a essa característica é verificado (valor de 0 a 1). Se esse valor for inferior a 0.5, o ponto de interesse é adicionado a uma lista de pontos de interesse excluídos (ver Figura 51).

```
// Iterating over POIsCategory
for (int j = 0; j < poiCategoryRepositoryList.Count; j++)
{

    //Get POI
    var poi = poiCategoryRepositoryList[j].POI;

    var poiConcernPref = poi.poiConcernPrefs.FirstOrDefault();
    string concernPreference = null;

    if (poiConcernPref != null && poiConcernPref.ConcernPrefs != null)
    {
        concernPreference = poiConcernPref.ConcernPrefs.concern;
    }

    float? preferenceValue = null;

    switch (concernPreference)
    {
        case "Cultural & Learning Experiences":
            preferenceValue = dto.recommendationsDTO.mainGroup.subgroups[i].travelPreferences.culturalExperiences;
            break;

        case "Uniqueness & Exoticness":
            preferenceValue = dto.recommendationsDTO.mainGroup.subgroups[i].travelPreferences.exoticness;
            break;

        default:
            break;
    }

    if (preferenceValue != null && preferenceValue < 0.5f)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }
```

Figura 51 - Verificação das *Concern Preferences* de um Subgrupo

Para além disso, ainda durante esta fase, são consideradas as incapacidades e os medos/fobias dos turistas. Se um determinado membro de um subgrupo tiver uma incapacidade que impeça a visita a um determinado ponto de interesse, esse ponto de interesse não é recomendado, independentemente de os outros membros terem ou não a mesma incapacidade. O mesmo não se aplica aos medos/fobias. Por exemplo, só se a maioria dos membros de um subgrupo tiver medo de alturas, um ponto de interesse que tenha esse tipo de caraterísticas é adicionado à lista de pontos excluídos de forma que não seja recomendado (ver Figura 52). Estas decisões foram tomadas porque uma deficiência pode ser mais restritiva e incómoda para um turista do que um medo ou uma fobia. No entanto, a lógica pode ser facilmente alterada para ir ao encontro do *feedback* e das necessidades dos futuros utilizadores da aplicação.

```
if (!excludedPoiCategory.Contains(poiCategoryRepositoryList[j]))
{

    int fearCounter = 0;
    int limitationCounter = 0;

    // Get POI fears
    if (poi.fears != null)
    {
        List<string> poiFears = poi.fears;

        // Iterating over associated POI fears
        if (poiFears != null)
        {
            Console.WriteLine(poi.name);
            foreach (var fear in poiFears)
            {
                if (numMembersWithFear.ContainsKey(fear))
                {
                    fearCounter += numMembersWithFear[fear];
                }
            }
        }


    }

    if (poi.limitations != null)
    {

        // Get POI limitations
        List<string> poiAccessibility = poi.limitations;

        // Iterating over POI Accessibility

        foreach (var limitation in poiAccessibility)
        {
            if (numMemberWithLimitations.ContainsKey(limitation))
            {
                limitationCounter += numMemberWithLimitations[limitation];
            }
        }
    }


    // check if the number of members with that fear is equal or more than half of the subgroup members
    if (fearCounter >= dto.recommendationsDTO.mainGroup.subgroups[i].members.Count / 2)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }

    // check if a member has that limitation, if yes it exludes the poi
    if (limitationCounter > 0)
    {
        excludedPoiCategory.Add(poiCategoryRepositoryList[j]);
    }
```

Figura 52 - Verificação dos Medos e Fobias para cada Ponto de Interesse

Após estas verificações, o método identifica as categorias de turismo mais relevantes para o utilizador, através do método *FindHighestValueTourismCategories*.

Depois disso, é invocado um método separado, com o nome *POICategoryToOpinionRecommendationsWeather*. Este método é o mesmo que foi explicado anteriormente para as recomendações individuais e por essa razão não se vai voltar a falar do seu funcionamento. No entanto, é importante referir que existe uma diferença relevante quando se invoca este método em recomendações de grupo e em recomendações individuais. Na versão que estamos a considerar agora, existe uma condicional que verifica se o número de utilizadores é mais que um. Se sim, estamos a lidar com um grupo e a função adota uma metodologia diferente para lidar com a colocação dos pontos que já foram visitados no fim da lista dos pontos que estão a ser considerados para recomendação. Essa lógica é a seguinte: para cada utilizador de um subgrupo, os pontos de interesse visitados são obtidos através de

uma chamada a uma API. Depois de obter esses pontos, é verificado que pontos foram visitados pela maioria dos utilizadores e esses passam então para o fim da lista (Figura 53).

```
else
// if it's a group recommendation
{
    Dictionary<long, int> visitedCount = new Dictionary<long, int>();

    // For each user, get the visited POIs
    foreach (var nickname in usersIds)
    {
        var url = $"https://localhost:5007/ps/poi/{nickname}";
        var response = await client.GetAsync(url);
        response.EnsureSuccessStatusCode();

        var json = await response.Content.ReadAsStringAsync();
        var visitedPoiData = JsonConvert.DeserializeObject<List<InfoVisitedPOIDTO>>(json);

        // Increase count for visited POIs
        foreach (var visitedPoi in visitedPoiData)
        {
            if (visitedCount.ContainsKey(visitedPoi.Id))
                visitedCount[visitedPoi.Id]++;
            else
                visitedCount.Add(visitedPoi.Id, 1);
        }
    }

    // Determine the majority
    int majority = usersIds.Count / 2 + 1;

    // Find the POIs visited by majority of users
    List<POI_Category> majorityVisitedPOICategoriesInRes =
        res.Where(poiCategory => visitedCount.ContainsKey(poiCategory.POI_Id) && visitedCount[poiCategory.POI_Id] >= majority).ToList();

    // Remove them from the original position
    res.RemoveAll(poiCategory => majorityVisitedPOICategoriesInRes.Contains(poiCategory));

    // Add them to the end
    res.AddRange(majorityVisitedPOICategoriesInRes);

}
```

Figura 53 - Verificação dos Pontos de Interesse já visitados para cada Membro de um Subgrupo

Após a conclusão da chamada do método *POICategoryToOpinionRecommendationsWeather*, obtém-se o DTO do tipo *WeatherRecommendationResultDTO* e um novo objeto do tipo *SubgroupPOIMAMSWeather* é criado e adicionado à lista de recomendações de subgrupos *listSubgroupsRecomm* (ver Figura 54). Esse objeto possui informações importantes, tais como: o id do subgrupo atual, o nome do subgrupo, os membros do subgrupo, as recomendações originais para o subgrupo e as recomendações alternativas.

```
listSubgroupsRecomm.Add(new SubgroupPOIMAMSWeather
{
    id = dto.recommendationsDTO.mainGroup.subgroups[i].id,
    //name = char.ToUpper(categories.ElementAt(0).Key[0]) + categories.ElementAt(0).Key.Substring(1),
    name = char.ToUpper(categories.ElementAt(0).Key[0]) + categories.ElementAt(0).Key.Substring(1) +
        dto.recommendationsDTO.mainGroup.subgroups[i].id, //PAT, para trazer o nome do subgrupo+ID, pois como ha nomes repetidos nao estavam a aparecer bem na UI
    description = subgroupDescription,
    justification = subgroupDescription,
    numMembers = dto.recommendationsDTO.mainGroup.subgroups[i].numMembers,
    members = dto.recommendationsDTO.mainGroup.subgroups[i].members,
    recommendationsList = weatherRecommendationResultDTO.recommendations,
    alternatives = weatherRecommendationResultDTO.alternatives,
});
listRecommendations = new List<RecommendationListDTO>();
order = 1;
compatibility = 99;
```

Figura 54 - Criação de um Objeto do Tipo *SubgroupPOIMAMSWeather*

Depois de todos os subgrupos terem sido percorridos, é feita uma chamada ao método *POIRecommendationsSubgroupsWeather* do *RecommendationMapper*, passando a

listSubgroupsRecomm como parâmetro. Este *mapper* vai mapear a informação recebida de maneira a construir a resposta final que é devolvida pelo método. Na figura 78 presente na secção de anexos, é possível observar a estrutura do DTO do tipo *GroupDivisionPOIMAMSWeatherDTO* que é construído e devolvido pelo *mapper*.

5.3.2 Recomendações de Grupo com Eliminação Automática de Pontos de Interesse sob Condições Meteorológicas Desfavoráveis

O processo de recomendações de grupo sem a opção de substituição de pontos de interesse que estão sob condições meteorológicas desfavoráveis segue praticamente a mesma lógica descrita no ponto 5.3.1. Um Turista Líder de Grupo solicita recomendações de grupo através da aplicação, desencadeando um pedido à API no microsserviço *Multi-Agent Microservice*, especificamente ao recurso mams/agents/tourists/{group}/ requestRecommendation. O método associado ao recurso, está presente no *GroupController* e tem o nome de *RequestRecommendation* (ver Figura 55).

```
[AllowAnonymous]
[HttpPost("{group}/requestRecommendation")]
0 references
public async Task<IActionResult> RequestRecommendation(string group, GroupDTO dto)
{
    try
    {
        //Request the recommendatiosn from the MAS
        var result = await masService.RequestRecommendation(dto);
        return Ok(result);
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Figura 55 - Método *RequestRecommendation* para Grupos

Este método chama o *RequestRecommendation* na camada de serviços e a chamada dentro desse método ao microsserviço *Recommendation Engine Microservice* é feita ao recurso re/groupOpinionRecommendations (ver Figura 56).

```
public async Task<GroupDivisionPOIMAMSDTO> AssociateOpinionRecommendations(GroupDivisionPOIMAMSAfterOpinionDTO dto)
{
```

Figura 56 - Cabeçalho do Método *AssociateOpinionRecommendations* da Classe *RecommendationService*

A principal diferença entre o ponto 5.3.1 e o 5.3.2 é, novamente, a de que é feita uma chamada ao método *POICategoryToOpinionRecommendations* em vez do *POICategoryToOpinionRecommendationsWeather*. E que o resultado devolvido pelo método *POICategoryToOpinionRecommendationsWeather,* é enviado para o mapper *POIRecommendationsGroups*. Este mapper devolve um DTO do tipo *GroupDivisionPOIMAMSDTO*, sendo que a sua estrutura se encontra representada na figura 79 presente na secção de anexos.

5.4 Atualização Diária de Recomendações e Integração com o *VirtualPet*

Quando um utilizador faz um pedido de recomendações, são-lhe sugeridas várias recomendações. No entanto, existem situações em que um ponto de interesse sugerido está sob condições meteorológicas desfavoráveis. Quando isso acontece, de forma a melhorar a experiência do utilizador, surge a necessidade de fornecer recomendações alternativas onde a situação meteorológica não influencie negativamente a experiência do utilizador. Para além disso, tendo em conta que os utilizadores podem realizar excursões que duram mais que um dia, é essencial fazer pedidos de recomendações de forma contínua, de maneira a atualizar as sugestões.

Para isso, foi desenvolvida uma lógica em *frontend* que atua de forma a identificar pontos de interesse onde a situação meteorológica é desfavorável de forma contínua. Esta implementação garante que, mesmo em circunstâncias de más condições meteorológicas, os utilizadores podem ver não apenas as recomendações iniciais, mas também as alternativas. Esses dados são então passados para o *VirtualPet*, uma funcionalidade desenvolvida por outro colega. Aí, o utilizador pode então interagir de forma a substituir os pontos de interesse originais pelos alternativos sugeridos, se assim pretender. Desta maneira, o utilizador é facultado com uma experiência mais personalizada e divertida.

Uma vez que as condições meteorológicas variam com grande facilidade, para garantir que os utilizadores recebem recomendações que sejam apropriadas em termos meteorológicos, é necessário estar a fazer um *checkup* contínuo das condições climáticas até ao fim da excursão. Ou seja, é fundamental fazer pedidos contínuos de forma a obter as recomendações com as condições climáticas mais atualizadas. Tendo isto em conta, após alguma deliberação, foi concordado que a melhor estratégia seria fazer um pedido de 24 em 24 horas de maneira que não sobrecarregasse muito o sistema, mas mantendo uma atualização frequente das condições. Para a implementação deste trabalho contínuo, várias abordagens foram tidas em conta. Algumas das alternativas ponderadas foram a utilização de *frameworks* e bibliotecas externas capacitadas no agendamento e execução de tarefas em background, tal como a utilização de uma ferramenta fornecida pelo *AndroidStudio* chamada *WorkManager*. Após alguma deliberação e consulta com o colega responsável pelo desenvolvimento da funcionalidade do *VirtualPet*, decidiu-se que a melhor opção seria usar uma ferramenta já presente no *AndroidStudio*, já que o *VirtualPet* iria ser implementado dentro do ambiente Android. Sendo assim, a ferramenta *WorkManager* foi a escolhida para o desenvolvimento da funcionalidade a ser discutida.

O *WorkManager* é uma biblioteca que permite a criação de trabalhos em segundo plano. Esta ferramenta é a indicada para usar em ambientes Android. O *WorkManager* faz parte do *Android Jetpack*, um conjunto de bibliotecas de software que ajudam os desenvolvedores a seguir as melhores práticas. Quando é necessário que um trabalho seja executado mesmo se a aplicação estiver fechada ou depois do dispositivo ser reiniciado, o *WorkManager* é a solução ideal.

Para a realização da funcionalidade, foram desenvolvidos dois tipos de *workers*, sendo eles os seguintes: o *IndividualDailyRecommendationWorker* (*worker* utilizado para as recomendações individuais) e o *GroupDailyRecommendationWorker* (*worker* utilizado para as recomendações de grupo).

5.4.1 IndividualDailyRecommendationWorker

A classe *IndividualDailyRecommendationWorker* é uma classe que estende o *Worker*, e foi construída com o objetivo de realizar tarefas em segundo plano. Este *worker* em concreto, tem como principal objetivo pedir e atualizar as recomendações individuais de forma diária baseadas nas condições meteorológicas (ver Figura 57).

Nesta classe, está presente o método *doWork*. Este método é o responsável por aplicar a lógica da tarefa que vai ser executada em segundo plano. Este método é chamado quando o *worker* é executado e devolve um *Result*.

O método começa por verificar se a data atual em que o *worker* está a ser executado já ultrapassou a data de fim da excursão. Se sim, então o *worker* é cancelado e já não precisa de ser mais executado (ver Figura 57).

```
@NonNull
@Override
public Result doWork() {

    long currentDateInMillis = System.currentTimeMillis();

    SharedPreferences sharedPreferences = getApplicationContext().
            getSharedPreferences( s: "com.example.grouplannerapp", Context.MODE_PRIVATE);

    long endDateLong = convertDateStringToMillis(manager.getEndDate());

    if (currentDateInMillis >= endDateLong) {
        UUID workId = UUID.fromString(sharedPreferences.getString( s: "job_uuid_" + manager.getUsername(),  s1: ""));
        WorkManager.getInstance(getApplicationContext()).cancelWorkById(workId);
        return Result.success();
    }
```

Figura 57 - Verificação da Necessidade de Terminar o Trabalho

De seguida, é feito um pedido de recomendações, chamando a API responsável por gerar as mesmas. Para isso, são passadas algumas informações relevantes como o nome do utilizador, o destino (país e cidade) e a data de início e de fim da excursão (ver Figura 58).

```
String username = manager.getUsername();
api = ApiUtils.getMAMSService();

AskIndividualRecommendations askIndividualRecommendations = new AskIndividualRecommendations(username, manager.getDestinationCountry(),
                                                            manager.getDestinationCity(), manager.getStartDate(), manager.getEndDate());
Call<WeatherIndividualRecommendations> call = api.individualRecommendationsWeatherMAMS(username,askIndividualRecommendations);
Response<WeatherIndividualRecommendations> response = call.execute();
```

Figura 58 - Obtenção das Recomendações

Após a chamada à API, se a resposta devolvida for bem-sucedida, o método procede à substituição das recomendações originais onde a situação meteorológica é desfavorável pelas recomendações alternativas (ver Figura 59). É importante realçar que no código que está a ser apresentado, a substituição está a acontecer de forma automática, sem dar opção ao utilizador de escolher se quer que as recomendações originais sejam substituídas. No entanto, na versão final, o utilizador terá a opção de escolher se deseja que as recomendações originais sejam substituídas pelas alternativas. Esse trabalho foi feito pelo colega que trabalhou no desenvolvimento da funcionalidade do *VirtualPet* e para isso teve de adaptar e moldar minimamente o código fornecido aqui apresentado, de maneira a permitir que os utilizadores tenham a capacidade de fazer essa escolha.

```
if (response.isSuccessful() && response.body() != null && response.body() != null && response.body().recommendationResult.getRecommendations().size() != 0) {
    try {

        Log.d( tag: "DailyRecommend",  msg: "doWork() executado2");

        List<RecommendationList> recommendations = response.body().recommendationResult.getRecommendations();
        List<RecommendationAlternative> alternatives = response.body().recommendationResult.getAlternatives();

        for (int i = 0; i < recommendations.size(); i++) {
            for (RecommendationAlternative alternative : alternatives) {
                if (recommendations.get(i).getPoiId().equals(alternative.getPoiId())) {
                    // Replace the item in recommendations with the alternative
                    RecommendationList newRecommendation = new RecommendationList(
                            alternative.getAlternativePoiId(),
                            recommendations.get(i).getOrder(),
                            recommendations.get(i).getPredictedCompatibility(),
                            alternative.getAlternativeJustification()
                    );

                    recommendations.set(i, newRecommendation);
                    break;  // No need to check the rest of the alternatives
                }
            }
        }

        // Create the IndividualRecommendations object with the updated recommendations list
        IndividualRecommendations individualRecommendations = new IndividualRecommendations(
                response.body().resourceType,
                response.body().id,
                response.body().meta,
                response.body().source,
                response.body().destination,
                response.body().eventCoding,
                response.body().receiver,
                response.body().response,
                recommendations,  // the updated recommendations list
                response.body().tourismCategories,
                response.body().travelPreferences
        );
```

Figura 59 - Substituição das Recomendações Originais

Depois disso, é feita uma chamada à API responsável por associar as recomendações ao utilizador, associando assim as recomendações atualizadas (ver Figura 60). De referir novamente que na versão final, este método só será invocado se o utilizador decidir que pretende que as recomendações originais onde a situação meteorológica é desfavorável, sejam substituídas pelas alternativas.

Figura 60 - Associação das Recomendações ao Utilizador

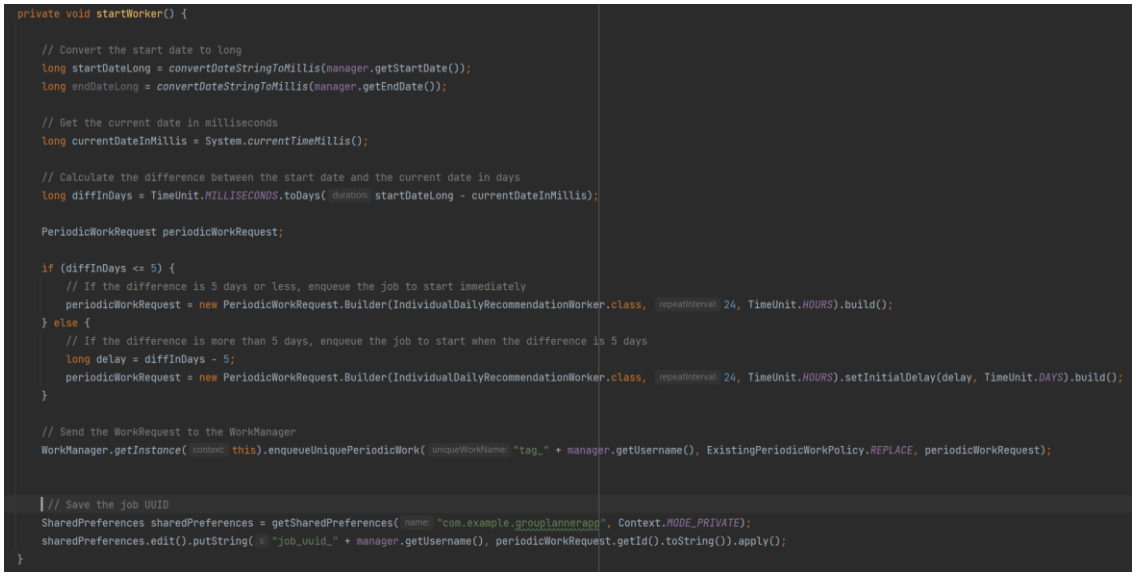
No final, é devolvido um *Result*, podendo este ser *Result.failure* se algo correu mal, ou *Result.success* se tudo correu dentro da normalidade.

É relevante referir que o *IndividualDailyRecommendationWorker* é inicializado e executado pela primeira vez quando o utilizador faz um pedido de recomendações na aplicação *Grouplanner*. Este pedido do utilizador é processado por um método presente no *frontend*, nomeadamente na classe *IndividualRecommendationsActivity* que vai comunicar com o método que gera as recomendações, que se encontra no *backend* (método explicado anteriormente). Dentro do método de *frontend*, além de outras operações que são realizadas para processar o pedido do utilizador, uma chamada é feita ao método *startWorker*. Este último, é o responsável então por inicializar a execução do *worker*.

Este método começa por calcular a diferença entre a data de início da excursão e a data atual, ou seja, o momento temporal em que o pedido de recomendações está a ser feito pelo utilizador (ver Figura 61).

Com base nessa diferença, se essa for de cinco dias ou menos, o então *IndividualDailyRecommendationWorker* é programado para começar imediatamente, executando o trabalho de 24 em 24 horas. Por outro lado, se a diferença for maior que cinco dias, o *IndividualDailyRecommendationWorker* é programado para ser inicializado e executado quando a diferença temporal for de cinco dias, sendo o trabalho igualmente executado de 24 em 24 horas (ver Figura 61). Isto acontece, uma vez que o sistema responsável por fornecer as informações meteorológicas só consegue no máximo ir buscar as informações até cinco dias depois do momento temporal em que é feito o pedido.

Por fim, o *WorkRequest* é então enviado para o *WorkManager* com um identificador único. A política *ExistingPeriodicWorkPolicy.REPLACE* é utilizada para garantir que um *worker* existente com o mesmo identificador seja substituído, de forma a permitir que quando um utilizador faz um novo pedido de recomendações, este último trabalho seja substituído pelo novo. Para além disso, o UUID do *WorkRequest* é salvo no *SharedPreferences* (API do AndroidStudio que permite guardar informação sobre a forma de par chave-valor) para ser usado no futuro (ver Figura 61).



```
private void startWorker() {

    // Convert the start date to long
    long startDateLong = convertDateStringToMillis(manager.getStartDate());
    long endDateLong = convertDateStringToMillis(manager.getEndDate());

    // Get the current date in milliseconds
    long currentDateInMillis = System.currentTimeMillis();

    // Calculate the difference between the start date and the current date in days
    long diffInDays = TimeUnit.MILLISECONDS.toDays( duration: startDateLong - currentDateInMillis);

    PeriodicWorkRequest periodicWorkRequest;

    if (diffInDays <= 5) {
        // If the difference is 5 days or less, enqueue the job to start immediately
        periodicWorkRequest = new PeriodicWorkRequest.Builder(IndividualDailyRecommendationWorker.class,  repeatInterval: 24, TimeUnit.HOURS).build();
    } else {
        // If the difference is more than 5 days, enqueue the job to start when the difference is 5 days
        long delay = diffInDays - 5;
        periodicWorkRequest = new PeriodicWorkRequest.Builder(IndividualDailyRecommendationWorker.class,  repeatInterval: 24, TimeUnit.HOURS).setInitialDelay(delay, TimeUnit.DAYS).build();
    }

    // Send the WorkRequest to the WorkManager
    WorkManager.getInstance( context: this).enqueueUniquePeriodicWork( uniqueWorkName: "tag_" + manager.getUsername(), ExistingPeriodicWorkPolicy.REPLACE, periodicWorkRequest);


    // Save the job UUID
    SharedPreferences sharedPreferences = getSharedPreferences( name: "com.example.grouplannerapp", Context.MODE_PRIVATE);
    sharedPreferences.edit().putString( s: "job_uuid_" + manager.getUsername(), periodicWorkRequest.getId().toString()).apply();
}
```

Figura 61 - Método Invocado para Inicializar o *IndividualDailyRecommendationWorker*

5.4.2 GroupDailyRecommendationWorker

Tal como a classe *IndividualDailyRecommendationWorker*, a classe *GroupDailyRecommendationWorker* é uma classe que estende o *Worker*, e foi construída com o objetivo de realizar tarefas em segundo plano. Este *worker* em concreto, tem como principal objetivo pedir e atualizar as recomendações de grupo de forma diária baseadas nas condições meteorológicas.

Nesta classe, está presente o método *doWork*. Este método é o responsável por aplicar a lógica da tarefa que vai ser executada em segundo plano. Este método é chamado quando o *worker* é executado e devolve um *Result*.

O método começa por ir buscar algumas informações importantes como o identificador do grupo e a data de fim de excursão. De seguida, verifica se a data atual em que o *worker* está a ser executado já ultrapassou a data de fim da excursão. Se sim, então o *worker* é cancelado e já não precisa mais de ser executado (ver Figura 62).

Figura 62 - Verificação da Necessidade de Terminar o Trabalho Existente

Se passar por essa verificação, são então feitos vários pedidos a diferentes API de forma a recolher dados pertinentes, como informações sobre o grupo e sobre os utilizadores. Também é feito um pedido à API responsável por gerar as recomendações para os subgrupos, recebendo esta algumas das informações recolhidas nas outras duas chamadas (ver Figura 63).

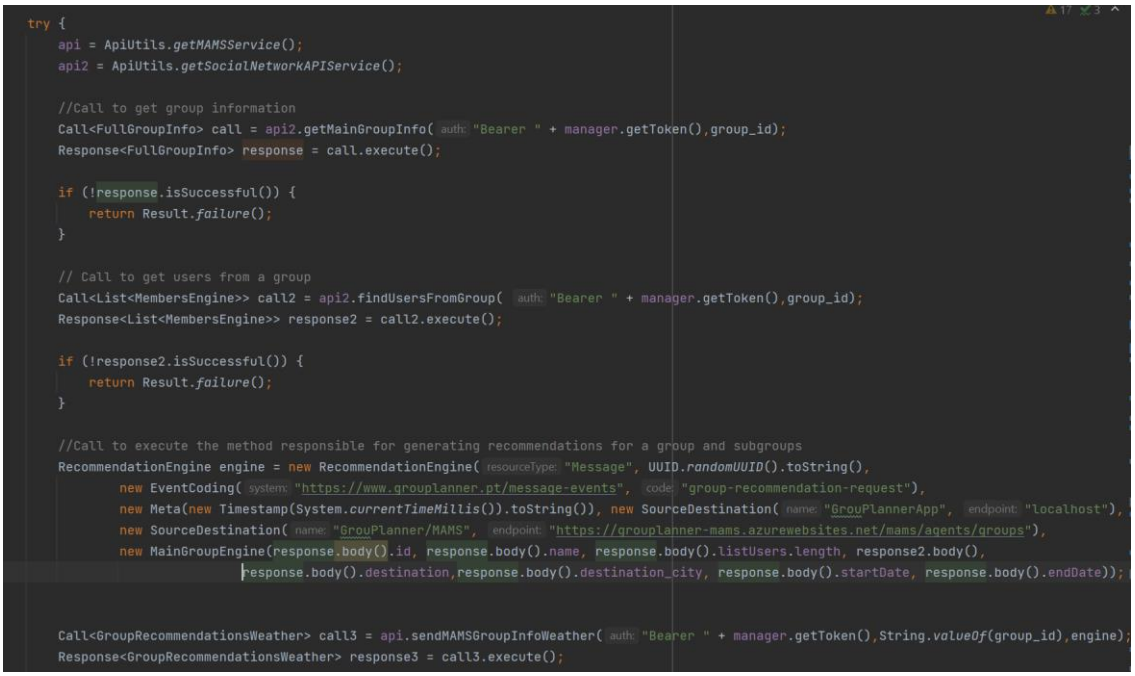
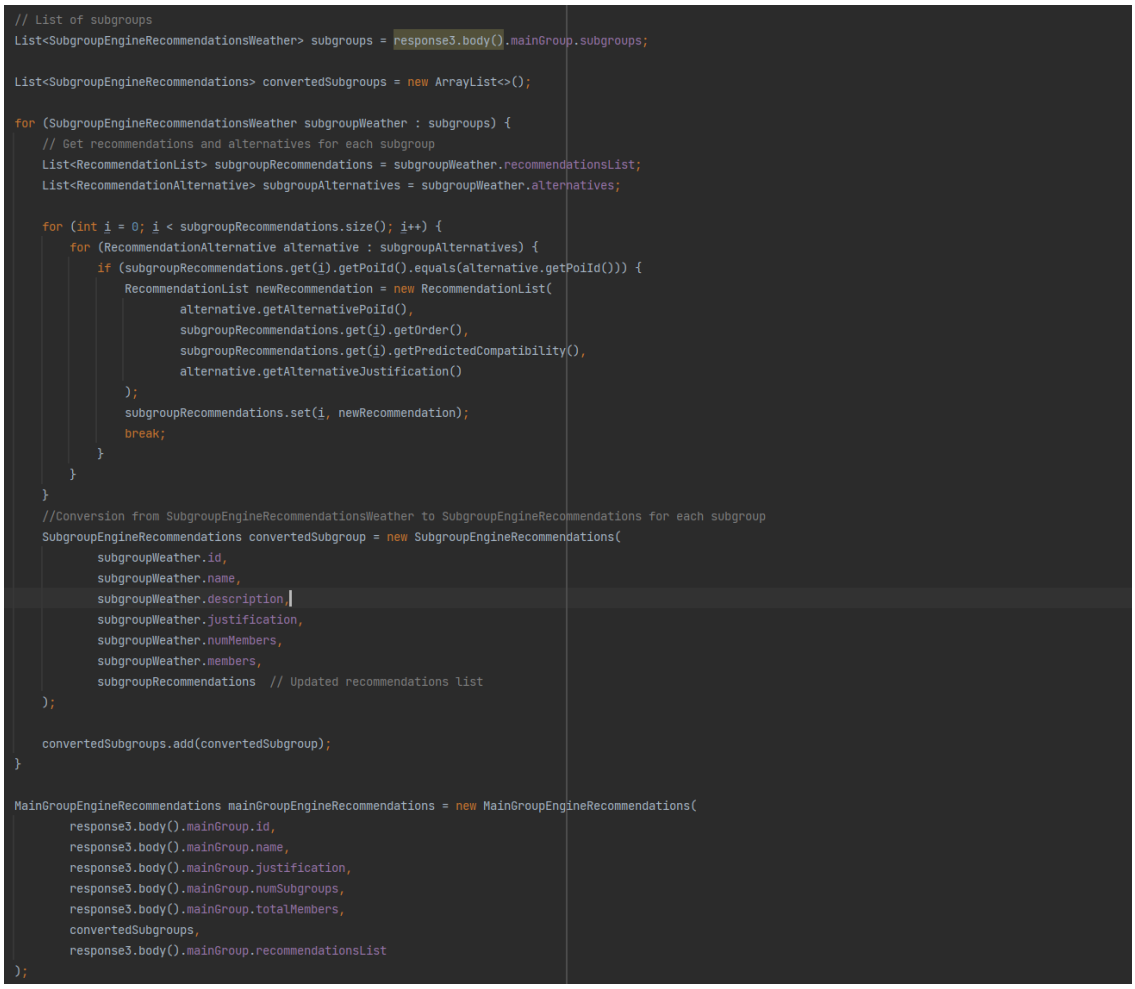


Figura 63 - Obtenção das Recomendações Atualizadas para os Subgrupos

Após a chamada à API de geração de recomendações, se a resposta devolvida for bem-sucedida, o método vai proceder à substituição das recomendações originais, onde a situação meteorológica é desfavorável, pelas recomendações alternativas, para cada subgrupo do grupo principal (ver Figura 64). É novamente importante realçar que no código que está a ser apresentado, a substituição está a acontecer de forma automática, sem dar opção ao líder de grupo de escolher se quer que as recomendações originais sejam substituídas. No entanto, na

versão final, o líder terá a opção de escolher se deseja que as recomendações originais sejam substituídas pelas alternativas.



```
// List of subgroups
List<SubgroupEngineRecommendationsWeather> subgroups = response3.body().mainGroup.subgroups;

List<SubgroupEngineRecommendations> convertedSubgroups = new ArrayList<>();

for (SubgroupEngineRecommendationsWeather subgroupWeather : subgroups) {
    // Get recommendations and alternatives for each subgroup
    List<RecommendationList> subgroupRecommendations = subgroupWeather.recommendationsList;
    List<RecommendationAlternative> subgroupAlternatives = subgroupWeather.alternatives;

    for (int i = 0; i < subgroupRecommendations.size(); i++) {
        for (RecommendationAlternative alternative : subgroupAlternatives) {
            if (subgroupRecommendations.get(i).getPoiId().equals(alternative.getPoiId())) {
                RecommendationList newRecommendation = new RecommendationList(
                        alternative.getAlternativePoiId(),
                        subgroupRecommendations.get(i).getOrder(),
                        subgroupRecommendations.get(i).getPredictedCompatibility(),
                        alternative.getAlternativeJustification()
                );
                subgroupRecommendations.set(i, newRecommendation);
                break;
            }
        }
    }
    //Conversion from SubgroupEngineRecommendationsWeather to SubgroupEngineRecommendations for each subgroup
    SubgroupEngineRecommendations convertedSubgroup = new SubgroupEngineRecommendations(
            subgroupWeather.id,
            subgroupWeather.name,
            subgroupWeather.description,
            subgroupWeather.justification,
            subgroupWeather.numMembers,
            subgroupWeather.members,
            subgroupRecommendations  // Updated recommendations list
    );

    convertedSubgroups.add(convertedSubgroup);
}

MainGroupEngineRecommendations mainGroupEngineRecommendations = new MainGroupEngineRecommendations(
        response3.body().mainGroup.id,
        response3.body().mainGroup.name,
        response3.body().mainGroup.justification,
        response3.body().mainGroup.numSubgroups,
        response3.body().mainGroup.totalMembers,
        convertedSubgroups,
        response3.body().mainGroup.recommendationsList
);
```

Figura 64 - Substituição das Recomendações Originais pelas Atualizadas

Depois disso, um objeto do tipo *GroupRecommendations* é criado e é feita uma chamada à API responsável por associar as recomendações ao grupo, associando assim as recomendações atualizadas (ver Figura 65). De referir novamente que, na versão final, este método só será invocado se o líder do grupo decidir que pretende que as recomendações originais onde a situação meteorológica é desfavorável, sejam substituídas pelas alternativas.

```
GroupRecommendations groupRecommendations = new GroupRecommendations(response3.body().resourceType,response3.body().id,
        response3.body().eventCoding,response3.body().response, response3.body().meta,
        response3.body().source, response3.body().destination,mainGroupEngineRecommendations);

//Call that creates the subgroups with the new recommendations
Call<Long> call4 = api2.createSubgroups( auth: "Bearer " + manager.getToken(),groupRecommendations);
Response<Long> response4 = call4.execute();
```

Figura 65 - Associação das Recomendações Atualizadas a cada Subgrupo

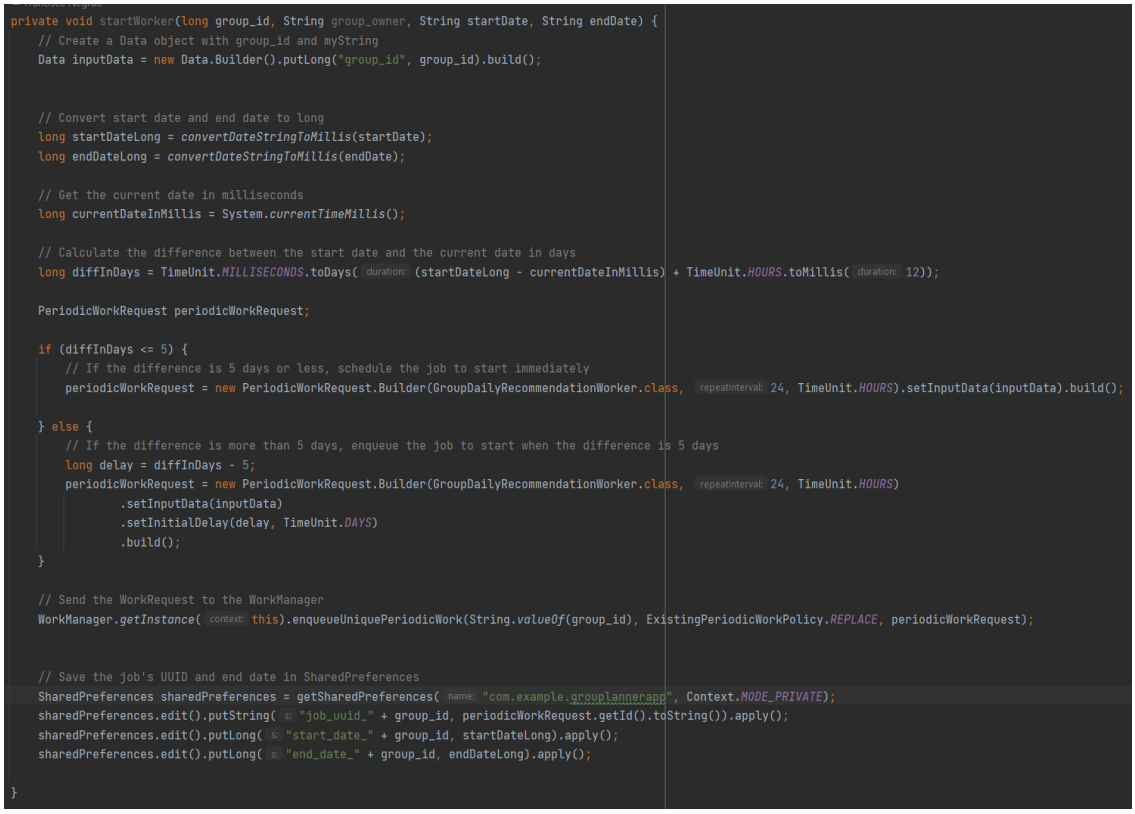
No final, é devolvido um *Result*, podendo este ser *Result.failure* se algo correu mal, ou *Result.success* se tudo correu dentro da normalidade.

É importante referir que o *GroupDailyRecommendationWorker* é inicializado e executado pela primeira vez quando o líder do grupo faz um pedido de recomendações na aplicação *Grouplanner*. Este pedido é processado por um método presente no *frontend*, nomeadamente na classe *GroupNavigationActivity* que vai comunicar com o método que gera as recomendações que se encontra no *backend* (método explicado anteriormente). Dentro do método de *frontend*, além de outras operações que são realizadas para processar o pedido do utilizador, é feita uma chamada ao método *startWorker*. Este último é então responsável por inicializar a execução do *worker*.

Tal como para as recomendações individuais, este método começa por calcular a diferença entre a data de início da excursão e a data atual, ou seja, o momento temporal em que o pedido de recomendações está a ser feito pelo utilizador (ver Figura 66).

Com base nessa diferença, se essa for de cinco dias ou menos, o então *GroupDailyRecommendationWorker* é programado para começar imediatamente, executando o trabalho de 24 em 24 horas. Por outro lado, se a diferença for maior que cinco dias, o *GroupDailyRecommendationWorker* é programado para ser e executado quando a diferença temporal for de cinco dias, sendo o trabalho igualmente executado de 24 em 24 horas (ver Figura 66).

Por fim, o *WorkRequest* é então enviado para o *WorkManager* com um identificador único. A política *ExistingPeriodicWorkPolicy.REPLACE* é utilizada para garantir que um *worker* existente com o mesmo identificador seja substituído, de forma a permitir que quando um turista líder de grupo faz um novo pedido de recomendações para um grupo em específico, este último trabalho seja substituído pelo novo. Para além disso, o UUID do *WorkRequest*, juntamente com a *startDate* e a *endDate*, são salvos no *SharedPreferences* para serem usados no futuro (ver Figura 66).



```
private void startWorker(long group_id, String group_owner, String startDate, String endDate) {
    // Create a Data object with group_id and myString
    Data inputData = new Data.Builder().putLong("group_id", group_id).build();


    // Convert start date and end date to long
    long startDateLong = convertDateStringToMillis(startDate);
    long endDateLong = convertDateStringToMillis(endDate);

    // Get the current date in milliseconds
    long currentDateInMillis = System.currentTimeMillis();

    // Calculate the difference between the start date and the current date in days
    long diffInDays = TimeUnit.MILLISECONDS.toDays( duration: (startDateLong - currentDateInMillis) + TimeUnit.HOURS.toMillis( duration: 12));

    PeriodicWorkRequest periodicWorkRequest;

    if (diffInDays <= 5) {
        // If the difference is 5 days or less, schedule the job to start immediately
        periodicWorkRequest = new PeriodicWorkRequest.Builder(GroupDailyRecommendationWorker.class,  repeatInterval: 24, TimeUnit.HOURS).setInputData(inputData).build();

    } else {
        // If the difference is more than 5 days, enqueue the job to start when the difference is 5 days
        long delay = diffInDays - 5;
        periodicWorkRequest = new PeriodicWorkRequest.Builder(GroupDailyRecommendationWorker.class,  repeatInterval: 24, TimeUnit.HOURS)
                .setInputData(inputData)
                .setInitialDelay(delay, TimeUnit.DAYS)
                .build();
    }

    // Send the WorkRequest to the WorkManager
    WorkManager.getInstance( context: this).enqueueUniquePeriodicWork(String.valueOf(group_id), ExistingPeriodicWorkPolicy.REPLACE, periodicWorkRequest);


    // Save the job's UUID and end date in SharedPreferences
    SharedPreferences sharedPreferences = getSharedPreferences( name: "com.example.grouplannerapp", Context.MODE_PRIVATE);
    sharedPreferences.edit().putString( s: "job_uuid_" + group_id, periodicWorkRequest.getId().toString()).apply();
    sharedPreferences.edit().putLong( s: "start_date_" + group_id, startDateLong).apply();
    sharedPreferences.edit().putLong( s: "end_date_" + group_id, endDateLong).apply();

}
```

Figura 66 - Método Invocado para Inicializar o *GroupDailyRecommendationWorker*

6 Testes

6.1 Testes de Integração

Com o intuito de verificar se os microsserviços desenvolvidos são capazes de trabalhar em conjunto de forma eficaz, realizaram-se vários testes de integração para as diferentes funcionalidades desenvolvidas. Para a criação desses testes, recorreu-se à plataforma Postman.

Os testes realizados nesta plataforma consistem no envio de um pedido às API, representantes de certas funcionalidades e a verificação do estado do código da resposta HTTP. Na figura 67, encontra-se um exemplo de um teste a um pedido de recomendações individual. Quando a solicitação é feita, para o teste ser bem-sucedido a resposta tem de ter o código HTTP 200, significando que o pedido teve sucesso.

```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});
```

Figura 67 - Teste de Integração Pedido de Recomendação Individual

6.2 Testes Funcionais

A realização de testes funcionais permite perceber se as funcionalidades desenvolvidas estão de acordo com os requisitos e as expectativas delineadas. Para a realização destes testes, recrutaram-se participantes para estes poderem interagir com a aplicação.

Os participantes foram guiados por desenvolvedores da aplicação a realizar e experimentar vários cenários de utilização da aplicação. Após a realização dessas tarefas, foram fornecidos

questionários aos utilizadores de forma a obter informações sobre a experiência com a aplicação, nomeadamente o grau de satisfação com as recomendações fornecidas. Estes testes permitiram obter informações importantes sobre o desempenho da aplicação e sobre aspetos a melhorar. Estes testes estão detalhados com mais pormenor na secção “Avaliação e Análise de Resultados”.

7 Experimentação e Avaliação

Neste capítulo, é descrito o processo de experimentação e avaliação de resultados da solução proposta. Para isso, definiram-se as hipóteses de investigação, a metodologia utilizada para fazer a avaliação e por fim a análise dos resultados.

7.1 Definição das Hipóteses

As hipóteses de investigação que são a base da análise devem ser estabelecidas antes de avaliar o sistema de recomendação. Com o objetivo de comprovar que a solução implementada é de qualidade e capaz de fornecer recomendações adequadas definiram-se as seguintes hipóteses:

- **Hipótese 1:** O sistema de recomendações funciona de maneira eficiente e sem erros clamorosos que influenciem negativamente a experiência do utilizador.
- **Hipótese Nula (H01)**: O sistema de recomendações não funciona de maneira eficiente e apresenta erros que influenciam a experiência do utilizador negativamente.
- **Hipótese 2:** O sistema de recomendações fornece sugestões de qualidade que satisfazem as pretensões dos utilizadores.
- **Hipótese Nula (H02)**: O sistema de recomendações fornece sugestões que não satisfazem as pretensões dos utilizadores.

7.2 Metodologia de Avaliação

Para avaliar se o sistema faz recomendações de forma eficiente e sem erros (H1) desenvolveram-se um conjunto de testes.

Dos testes desenvolvidos, os mais relevantes para este caso, a que o sistema foi submetido, foram os testes de integração. Estes testes foram criados para verificar se os vários componentes do sistema funcionam bem em conjunto.

Para avaliar a qualidade e a satisfação com as recomendações individuais e de grupo de pontos de interesse (H2), foram recrutados treze participantes para testarem as funcionalidades. O processo de recruta consistiu no envio de emails a docentes e não docentes do ISEP de variados departamentos. Para facilitar o processo, os participantes foram encorajados a trazer um smartphone Android, porém se não o tivessem, era-lhes emprestado um. Os testes das funcionalidades foram realizados numa sala de reuniões do GECAD, no ISEP.

No início, pediu-se aos participantes para preencher um pré-questionário online, desenvolvido no Microsoft Forms, seguindo os regulamentos da RGPD[3], com o objetivo de obter dados de caraterização demográfica, as preferências relativamente a onze categorias de turismo e as preocupações relacionadas com viagens de turismo.

De seguida, foi lhes dito para, separadamente e em silêncio, que tinham quatro minutos para encontrarem dez pontos de interesse, exclusivamente na cidade do Porto, Portugal, e registá-los num formulário.

Depois disso, foi mostrado um pequeno tutorial de como a aplicação funcionava e pediu-se aos participantes que se registassem nela, preenchessem as informações básicas de perfil e indicassem eventuais medos/fobias e incapacidades, preenchessem o questionário *Big Five Inventory*, que mede a personalidade de uma pessoa nas 5 dimensões da personalidade de Goldberg (1990), fizessem um pedido de recomendações individual e pontuassem os pontos de interesse sugeridos com uma classificação de 1 a 5 estrelas.

Posteriormente, foi apresentado aos participantes um questionário online para comparar as duas abordagens de obtenção de sugestões. Na figura 80, presente na secção de anexos, encontra-se o questionário que foi pedido para preencher após obter as recomendações individuais.

Para as recomendações de grupo, os participantes foram instruídos a formar grupos, selecionar um líder de grupo e, conjuntamente, procurar e acordar dez pontos de interesse para visitar na cidade do Porto. Os pontos de interesse deviam ser registados num formulário online pelo líder do grupo. Para realizar esta tarefa, os grupos dispuseram de oito minutos.

O passo seguinte, que foi instruído ao líder do grupo, foi o de formar um grupo na aplicação Grouplanner, incluir os outros membros e solicitar uma recomendação de grupo. Depois de receberem os pontos de interesse, participantes foram convidados a classificá-los numa escala de 1 a 5.

[3] Todos os participantes que aceitaram participar no estudo assinaram um consentimento informado. Os questionários utilizados eram anónimos e confidenciais e foram aprovados pelo Gabinete de Proteção de Dados do Politécnico do Porto.

Por fim, os participantes tiveram acesso a um questionário online para comparar as duas abordagens de recolha de recomendações do grupo. Na figura 81, presente na secção de anexos, encontra-se o questionário que foi pedido para preencher após obter as recomendações de grupo.

7.3 Análise de Resultados

Nesta secção são discutidos os resultados obtidos para as hipóteses previamente definidas.

7.3.1 Avaliação da eficiência do sistema de recomendações

Caso de Teste: Geração de Recomendações Considerando as Limitações Físicas de um Utilizador

Este teste tem o objetivo de garantir que os pontos de interesse recomendados são adequados às limitações físicas de um utilizador.

Tabela 43 - Resultado do Caso de Teste "Geração de Recomendações Considerando Limitações Físicas"

Cenário de Teste	Estimativa	Resultado
Geração de recomendações considerando que um utilizador está de cadeira de rodas	Sugestão de pontos de interesse acessíveis a utilizadores com cadeira de rodas	Sugestão de pontos de interesse acessíveis a utilizadores com cadeira de rodas. As recomendações não possuem atividades radicais que não permitem a participação do utilizador

Caso de Teste: Geração de Recomendações Considerando as Fobias de um Utilizador

Este teste tem o objetivo de garantir que os pontos de interesse recomendados são adequados às fobias de um utilizador.

Tabela 44 - Resultado do Caso de Teste "Geração de Recomendações Considerando Fobias"

Cenário de Teste	Estimativa	Resultado
Geração de recomendações considerando a fobia acrofobia de um utilizador	Sugestão de pontos de interesse acessíveis a utilizadores com acrofobia, pontes, por exemplo, não devem ser sugeridas	Sugestão de pontos de interesse acessíveis a utilizadores com fobias. As recomendações não possuem pontes ou outro tipo de ponto de interesse que provoque acrofobia

Caso de Teste: Geração de recomendações considerando condições meteorológicas adversas

Este teste tem o objetivo de garantir que o sistema sugere pontos de interesse que são "confortáveis" para visitar em condições meteorológicas desfavoráveis.

Tabela 45 – Resultado do Caso de Teste "Geração de recomendações considerando condições meteorológicas adversas"

Cenário de Teste	Estimativa	Resultado
Geração de recomendações considerando condições meteorológicas adversas, num dia que está a chover	Sugestão de pontos de interesse adequados para o clima desfavorável, como pontos que sejam abrigados da chuva	Sugestão de pontos de interesse adequados para o clima desfavorável. As recomendações não possuem praias, que seriam desconfortáveis de visitar com tempo desfavorável

Concluindo, é possível perceber que os testes realizados, demonstraram que o processo de geração de recomendações é eficiente e sem falhas.

7.3.2 Avaliação da eficiência do sistema de recomendações

A amostra recolhida era constituída por 13 pessoas, sendo a maioria homens (69,2%) e as restantes mulheres (30,8%). A idade média dos participantes é de 22 anos, sendo 18 anos a idade mínima e 40 anos a idade máxima. A nível de situação de estado civil, a maioria dos participantes (69,2%) encontra-se solteiro, enquanto 30,8% estão numa relação.

O nível de educação dos participantes tem algumas variações. O grau de licenciatura é o mais comum (46,2%), seguido do ensino secundário (38,5%) e do mestrado (15,4%). Em relação à área em que os membros trabalham ou estudam, todos pertencem ao ramo da engenharia e da tecnologia.

Em relação à situação profissional que os participantes se encontram, é possível verificar que a maioria das pessoas (76,9%) é estudante, seguidos de dois trabalhadores que trabalham por conta própria (15,4%) e de uma pessoa que trabalha por conta própria (7,7%).

No que se refere à saúde, a esmagadora maioria dos participantes (92,3%) são pessoas saudáveis, sendo que apenas um participante (7,7%) tem uma doença não especificada.

Por fim, analisando as fobias dos participantes, é possível observar que mais de metade dos participantes (53,8%) não tem fobias, seguidos de três (23,1%) que referiram ter medo de alturas e três (23,1%) que relataram ter medo de espaços confinados e, por fim, um (7,7%) que relatou ter outros tipos de medos não definidos.

Tabela 46 - Dados da Amostra

		n	%
Estado Civil	Numa relação	4	31
	Solteiro	9	69
Nível de Ensino.	Ensino secundário	5	39
	Licenciatura (3-5 anos)	6	46
	Mestrado	2	15
Situação Profissional	Estudante	10	77
	Trabalhador-estudante	1	8
	Trabalhador por conta de outrem	2	15

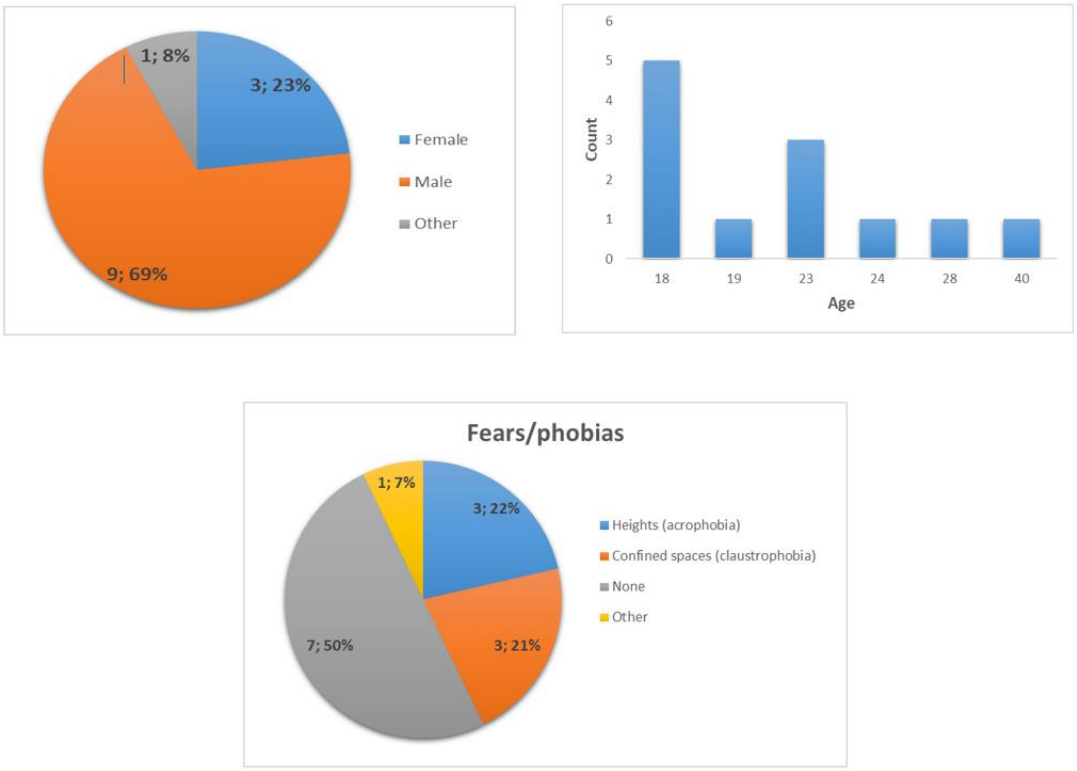


Figura 68 - Caraterísticas dos participantes: Canto superior esquerdo - género dos participantes; Canto superior direito - idade dos participantes; No centro- medos dos participantes.

Observando os dados expostos na tabela 47, é possível observar que para as recomendações individuais, a maioria dos participantes 61,5% (8 em 13) concorda que é mais rápido utilizar a aplicação para obter pontos de interesse do que procurar manualmente por eles, enquanto 30,8% tem uma opinião neutra, permitindo perceber que a aplicação oferece uma experiência de pesquisa eficaz.

Para além disso, 84,6% (11 em 13) dos participantes referiram preferir utilizar a aplicação em vez de fazerem uma pesquisa manual.

Em relação à satisfação dos participantes com a qualidade das sugestões da aplicação, 69,2% (9 em 13) dos utilizadores mostraram-se satisfeitos com os pontos de interesse sugeridos pela aplicação, atribuindo-lhe uma pontuação de 4 ou 5. Este é indicativo importante de que as recomendações da aplicação estão de acordo com os gostos e as expectativas dos participantes.

Passando para as recomendações de grupo, os dados mostram que 61,5% (8 em 13) dos participantes concordam que a procura de pontos de interesse seria mais rápida nas sugestões de grupo, mostrando que a aplicação é vista como uma ferramenta útil também em situações de grupo.

No que se refere em utilizar a aplicação para obter recomendações, 61,5% (8 em 13) dos inquiridos concordaram que preferiam utilizar a aplicação para obter recomendações em vez de procurar os pontos de interesse de forma manual.

Por fim, o grau de satisfação com as recomendações de grupo da aplicação é ainda maior, com 76,9% (10 em 13) dos utilizadores a indicarem satisfação. Isto mostra que a aplicação é útil tanto para utilizadores individuais como para grupos de utilizadores.

Tabela 47 - Comparação da pesquisa manual de pontos de interesse com as recomendações automáticas feitas pelos participantes. As respostas estão numa escala de Likert, de 1 - Discordo totalmente a 5 - Concordo totalmente.

Recomendações Individuais	**Scores**				
Pergunta	**1**	**2**	**3**	**4**	**5**
Foi mais rápido procurar manualmente os POI do que utilizar a aplicação para o efeito	2	6	4	1	0
Preferi utilizar a aplicação do que procurar manualmente os POI	0	0	2	7	4
As recomendações sugeridas pela aplicação foram do meu agrado	0	1	3	5	4
Group recommendations	**Scores**				
Question	**1**	**2**	**3**	**4**	**5**
Foi mais rápido procurar manualmente os POI do que utilizar a aplicação para o efeito	1	7	3	1	1
Preferi utilizar a aplicação para obter recomendações do que procurar manualmente POI	1	1	3	4	4
As recomendações sugeridas pela aplicação foram do meu agrado	0	1	2	8	2

Analisando os dados, pode-se concluir que a aplicação fornecedora de recomendações de pontos de interesse é capaz de satisfazer os utilizadores, facilitando a descoberta de pontos relevantes para os gostos dos utilizadores.

8 Conclusão

Neste capítulo são descritos os objetivos concretizados mediante os objetivos propostos, os problemas e limitações que surgiram durante o desenvolvimento do sistema, as melhorias que podem ser aplicadas ao que foi desenvolvido e, por fim, algumas considerações finais sobre o projeto.

8.1 Objetivos Concretizados

A fase inicial do trabalho começou com uma contextualização do estado da arte de sistemas de recomendação, individuais e de grupo, na área do turismo. Para além disso, este processo também consistiu na pesquisa sobre como o contexto do utilizador e as suas preferências podem ser inseridos no sistema de maneira a proporcionar recomendações que elevem o grau de satisfação do utilizador.

O segundo passo consistiu na contextualização do projeto em si, que já estava em desenvolvimento anteriormente. Esta fase foi uma das mais complicadas, mas fundamental para perceber o trabalho que tinha sido feito, identificar componentes e funcionalidades que necessitavam de melhorias e estabelecer um plano para a continuação do trabalho.

De seguida, deu-se uma fase de levantamento de requisitos, crucial para compreender as necessidades que eram preciso de implementar de modo a que os utilizadores tivessem as suas expetativas correspondidas. Nesta fase estabeleceram-se as funcionalidades chave do sistema, garantindo que este oferecesse recomendações personalizadas e contextualizadas.

Em relação à implementação, pode-se dizer que foi uma operação bem-sucedida, já que o sistema é capaz de fazer recomendações personalizadas e contextualizadas de pontos de interesse, tendo em conta a personalidade, as restrições físicas, as fobias e o contexto meteorológico, contribuindo assim para auxiliar no processo de decisão sobre quais pontos de

interesse a visitar, sendo uma ferramenta válida tanto para planeamentos individuais, como para grupos.

Por fim, foram utilizados participantes reais para avaliar o funcionamento e a qualidade do sistema. Esta fase permitiu observar a forma como os utilizadores interagiam com o sistema e como é que este respondia às expectativas dos participantes. Esta fase foi de extrema importância para verificar na prática a eficácia e qualidade do sistema desenvolvido.

8.2 Limitações e Trabalho Futuro

Durante o trabalho desenvolvido, surgiram algumas dificuldades, como a necessidade de compreender o sistema complexo já existente, que tinha sido iniciado anteriormente por outros colegas. Este foi um dos desafios mais complicados durante o desenvolvimento do projeto. Este sistema possuía uma dificuldade elevada de compreensão devido às suas inúmeras funcionalidades, e por muitas delas estarem pouco documentadas, exigindo um trabalho extra para compreender a lógica das mesmas.

Para além disso, a adição das novas funcionalidades e lógica às já existentes, nomeadamente no motor de recomendações, foi uma tarefa complicada, já que era fundamental garantir que estes funcionavam em conjunto com os componentes e funcionalidades já implementados no sistema.

Outra dificuldade que vale a pena referir, foi o facto de ter de utilizar a ferramenta *Android Studio*, com a qual não tinha qualquer experiência prévia, para realizar algum trabalho na área de *frontend.*

Apesar das dificuldades referidas, este foi um processo de aprendizagem importante que me permitiu adaptar a certas adversidades e a melhorar como desenvolvedor.

Em relação ao trabalho futuro, foram identificadas algumas áreas que podem ser melhoradas, com o intuito de aprimorar a precisão do sistema de recomendações.

Primeiramente, no futuro seria importante avolumar os atributos ligados aos pontos de interesse. Desta forma, o sistema será capaz fazer recomendações ainda mais pormenorizadas e mais ao encontro das preferências e contextos dos utilizadores, acrescentando mais características a cada ponto de interesse, sendo uma dessas, por exemplo, informação sobre os horários de funcionamento dos pontos de interesse. É extremamente importante considerar os horários em que os pontos estão disponíveis para serem visitados.

Outra área a ser melhorada, seria a de encontrar uma tecnologia menos consumidora de recursos, responsável pelas atualizações contínuas de recomendações. A utilização do *WorkManager* provoca uma maior lentidão na aplicação e leva a um consumo excessivo de bateria dos dispositivos móveis dos utilizadores. Tendo em conta o que foi dito, seria importante identificar e implementar, se possível, uma solução mais otimizada a nível de consumo de recursos.

Para além dessa, uma área que também necessita de ser revista é a otimização da funcionalidade de recomendações de grupo. Após vários testes, foi possível observar que em algumas ocasiões, a execução desta funcionalidade consome uma grande quantidade de recursos, excedendo em alguns casos, a quota de memória fornecida pelo serviço da *Microsoft Azure* (local em que o sistema se encontra hospedado).

Resumindo, as aplicações das melhorias sugestões mencionadas podem melhorar bastante a qualidade do sistema, contribuindo para uma experiência de utilizador mais satisfatória.

8.3 Considerações Finais

O processo de desenvolvimento deste projeto no centro de investigação GECAD foi uma experiência única e esclarecedora. Este trabalho permitiu-me um desenvolvimento enquanto engenheiro, enquanto me proporcionou um local de trabalho agradável e exigente. Alguns dos problemas propostos do projeto foram em áreas em que eu tinha pouca, ou nenhuma experiência. Passar por estes desafios e dificuldades foi uma jornada trabalhosa, mas bastante satisfatória.

Para além disso, é com uma grande satisfação saber que o desenvolvimento deste projeto não só culminou na realização de um sistema eficaz de recomendações, como também contribuiu para a escrita de um do artigo científico "Grouplanner, a Group Recommender System for Tourism: are Heterogeneity and Conflicting Preferences no Longer a Problem", que se encontra em estado de revisão na revista Expert Systems with Applications (JCR2022 IF 8.5, Q1).

Referências

Abreu, D. (2019*) Aplicação móvel para aconselhamento de treino de jovens atletas.* thesis.

Adomavicius, G. and Tuzhilin, A. (2010) "Context-aware Recommender Systems," Recommender Systems Handbook, pp. 217–253. Available at: https://doi.org/10.1007/978-0-387-85820-3_7.

Adomavicius, G. and Tuzhilin, A. (2015) "Context-aware Recommender Systems" Recommender Systems Handbook, pp. 191–226. Available at: https://doi.org/10.1007/978-1-4899-7637-6_6.

Alves, P., Carneiro, J., Marreiros, G., & Novais, P. (2019, September). Modeling a Mobile Group Recommender System for Tourism with Intelligent Agents and Gamification. In International Conference on Hybrid Artificial Intelligence Systems (pp. 577-588). Springer, Cham.

Alves, P., Gomes, D., Rodrigues, C., Carneiro, J., Novais, P., & Marreiros, G. (2022). Grouplanner: a Group Recommender System for Tourism with Multi-Agent MicroServices. In Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation - The PAAMS Collection. Springer, Cham.

Alves, P., Martins, A., Novais, P., & Marreiros, G. (2023a). Improving Group Recommendations using Personality, Dynamic Clustering and Multi-Agent MicroServices. In Seventeenth ACM Conference on Recommender Systems (RecSys '23), September 18–22, 2023, Singapore, Singapore. ACM, New York, NY, USA, 3 pages. Available at https://doi.org/10.1145/3604915.3610653

Alves, P., Martins, H., Saraiva, P., Carneiro, J., Novais, P., & Marreiros, G. (2023b). Group recommender systems for tourism: how does personality predict preferences for attractions, travel motivations, preferences and concerns?. User Modeling and User-Adapted Interaction, 1-70.

Anzileiro, A. (2020). Definindo e Revisando Seu Produto utilizando FURPS, Medium. Available at: https://anzileiro.medium.com/definindo-e-revisando-seu-produto-utilizando-furps-fbe6aa41c53b (Accessed: 12 October 2023).

Ardissono, L. et al. (2002) "Ubiquitous user assistance in a tourist information server," Lecture Notes in Computer Science, pp. 14–23. Available at: https://doi.org/10.1007/3-540-47952-x_4.

Ardissono, L. et al. (2003) "Intrigue: Personalized recommendation of tourist attractions for desktop and hand held devices," Applied Artificial Intelligence, 17(8-9), pp. 687–714. Available at: https://doi.org/10.1080/713827254.

Baltrunas, L. et al. (2011) "Context relevance assessment and exploitation in Mobile Recommender Systems," Personal and Ubiquitous Computing, 16(5), pp. 507–526. Available at: https://doi.org/10.1007/s00779-011-0417-x.

Borràs, J., Moreno, A. and Valls, A. (2014) “Intelligent Tourism Recommender Systems: A survey,” Expert Systems with Applications, 41(16), pp. 7370–7389. Available at: https://doi.org/10.1016/j.eswa.2014.06.007.

C. Brodeala, L. (2020) “Online recommender system for accessible tourism destinations,” Fourteenth ACM Conference on Recommender Systems [Preprint]. Available at: https://doi.org/10.1145/3383313.3411450.

Dara, S., Chowdary, C.R. and Kumar, C. (2019) “A survey on group Recommender Systems,” Journal of Intelligent Information Systems, 54(2), pp. 271–295. Available at: https://doi.org/10.1007/s10844-018-0542-3.

Functional and nonfunctional requirements: Specification and types (2019) AltexSoft. Available at: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/ (Accessed: 14 October 2023).

Garcia, I., Sebastia, L. and Onaindia, E. (2011) “On the design of individual and group Recommender Systems for tourism,” Expert Systems with Applications, 38(6), pp. 7683–7692. Available at: https://doi.org/10.1016/j.eswa.2010.12.143.

Gavalas, D. and Kenteris, M. (2011) “A web-based pervasive recommendation system for mobile tourist guides,” Personal and Ubiquitous Computing, 15(7), pp. 759–770. Available at: https://doi.org/10.1007/s00779-011-0389-x.

Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alex Tuzhilin

Gewarren (no date) .NET SDK overview, .NET SDK overview | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/dotnet/core/sdk (Accessed: February 18, 2023).

Goldberg, L.R., 1990. An alternative” description of personality”: the big-five factor structure. *Journal of personality and social psychology 59*, 6, 1216.

Gomes, Á. (2021) *Desenvolvimento de um Sistema de Recomendação para Grupos de Turismo*. thesis.

Gomes, D. (2022) *Desenvolvimento de um Sistema Multi-Agente usando Microserviços num Sistema de Recomendação para Grupos de Turismo*. thesis.

Khallouki, H., Abatal, A. and Bahaj, M. (2018) “An ontology-based context awareness for Smart Tourism Recommendation System,” Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications [Preprint]. Available at: https://doi.org/10.1145/3230905.3230935.

Kulkarni, S. and Rodd, S.F. (2020) “Context aware recommendation systems: A review of the state of the art techniques,” Computer Science Review, 37, p. 100255. Available at: https://doi.org/10.1016/j.cosrev.2020.100255.

Livne, A. et al. (2022) “Evolving context-aware recommender systems with users in mind,” Expert Systems with Applications, 189, p. 116042. Available at: https://doi.org/10.1016/j.eswa.2021.116042.

Lotfi, A. (2022) How java code is compiled and run?, Medium. Javarevisited. Available at: https://medium.com/javarevisited/how-java-code-compiled-and-run-e4702fb83ffa (Accessed: February 18, 2023).

Martinekuan (no date) Estilo de Arquitetura de microsserviços, Azure Architecture Center | Microsoft Learn. Available at: https://learn.microsoft.com/pt-pt/azure/architecture/guide/architecture-styles/microservices (Accessed: 12 October 2023).

Martins, R. (2016) *Sistema de Recomendação de Tutoriais*. thesis.

Microservices (no date). Available at: https://spring.io/microservices (Accessed: February 18, 2023).

Moreira, A. (2019) Sistema de recomendação para uma plataforma de comércio eletrónico. thesis.

Moreno, A., Sebastiá, L. and Vansteenwegen, P. (2015) "Tours'15," Proceedings of the 9th ACM Conference on Recommender Systems. Available at: https://doi.org/10.1145/2792838.2798713.

Nicola, S. (no date) "Análise de Valor." ISEP - Instituto Superior de Engenharia do Porto

Node.js (no date) About, Node.js. Available at: https://nodejs.org/en/about/ (Accessed: February 18, 2023).

Node.js vs. ASP.NET - which is the best for app development? (2022) Node.js vs. ASP.NET - Which is the Best for App Development? Turing Enterprises Inc. Available at: https://www.turing.com/kb/node-js-vs-asp-net (Accessed: February 18, 2023).

Nogueira, A. UML - Unified Modeling Language - Atores, atividades e componentes. (no date) Linha de Código. Available at: http://www.linhadecodigo.com.br/artigo/853/uml-unified-modeling-language-atores-atividades-e-componentes.aspx (Accessed: 12 October 2023).

Pereira, D. (2021) What is the value proposition canvas?, Business Model Analyst. Available at: https://businessmodelanalyst.com/value-proposition-canvas/ (Accessed: February 23, 2023).

Pinelas, F. (2022) *Sistema de recomendação para alívio de stress*. thesis.

Ravoof, S. (2023) PostgreSQL vs MySQL: Explore their 12 critical differences Kinsta®. Available at: https://kinsta.com/blog/postgresql-vs-mysql/ (Accessed: February 18, 2023).

Rao, D. (no date) Grasp design principles - University of Colorado Boulder Computer. Available at: https://home.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf (Accessed: 12 October 2023).

Rodrigues, C. (2021) *Desenvolvimento de um Sistema de Recomendação para Grupos de Turismo (Aplicação Móvel)*. thesis.

Rodrigues, D. (2021) *Aprimoramento do Modelo de Seleção dos Padrões Associativos: Uma abordagem de Mineração de Dados*. thesis.

Russo, A.J. (2003a) O que é o Scrum?, Amazon. Available at: https://aws.amazon.com/pt/what-is/scrum/ (Accessed: 14 October 2023).

Russo, A.J. (2003b) O que é o .NET?, Amazon. Filbert Pub. Available at: https://aws.amazon.com/pt/what-is/net/ (Accessed: February 18, 2023).

Russo, A.J. (2003c) O que são microsserviços?, Amazon. Available at: https://aws.amazon.com/pt/microservices/ (Accessed: 12 October 2023).

Sharma, N. and Dutta, M. (2020) "Movie Recommendation Systems," Proceedings of the 8th International Conference on Computer and Communications Management [Preprint]. Available at: https://doi.org/10.1145/3411174.3411194.

Shen, J.J. (2008) *Using cluster analysis, cluster validation, and consensus clustering to identify subtypes of pervasive developmental disorders.* thesis. Library and Archives Canada = Bibliothèque et Archives Canada.

Singla, C. Functional vs Non Functional Requirements (2022) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/ (Accessed: 12 October 2023).

Srifi, M. et al. (2020) "Recommender systems based on collaborative filtering using review texts—A survey," Information, 11(6), p. 317. Available at: https://doi.org/10.3390/info11060317.

Sufiyan, T. (2023) What is node.js: A comprehensive guide, Simplilearn.com. Simplilearn. Available at: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs#what_is_nodejs (Accessed: February 18, 2023).

Tarus, J.K., Niu, Z. & Mustafa, G. "Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning". Artif Intell Rev 50, 21–48 (2018). https://doi.org/10.1007/s10462-017-9539-5

Thirumaran, K. et al. (2021) "Virtual pets want to travel: Engaging visitors, creating excitement," Tourism Management Perspectives, 39, p. 100859. Available at: https://doi.org/10.1016/j.tmp.2021.100859.

van Setten, M., Pokraev, S. and Koolwaaij, J. (2004) "Context-aware recommendations in the Mobile Tourist Application Compass," Lecture Notes in Computer Science, pp. 235–244. Available at: https://doi.org/10.1007/978-3-540-27780-4_27.

Walpita, P. (2020) Software architecture patterns - layered architecture, Medium. Available at: https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057 (Accessed: 12 October 2023).

Wang, Y., Chan, S.C.-F. and Ngai, G. (2012) “Applicability of demographic recommender system to tourist attractions: A case study on trip advisor,” 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology [Preprint]. Available at: https://doi.org/10.1109/wi-iat.2012.133.

Watts, S. (2020) The importance of solid design principles, BMC Blogs. Available at: https://www.bmc.com/blogs/solid-design-principles/ (Accessed: 12 October 2023).

What is .net? an open-source developer platform. (no date) Microsoft. Available at: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (Accessed: February 18, 2023).

What is java? (no date) IBM. Available at: https://www.ibm.com/topics/java (Accessed: February 18, 2023).

Zagranovskaia, A. and Mitura, D. (2021) “Designing hybrid recommender systems,” IV International Scientific and Practical Conference [Preprint]. Available at: https://doi.org/10.1145/3487757.3490921.

Anexos

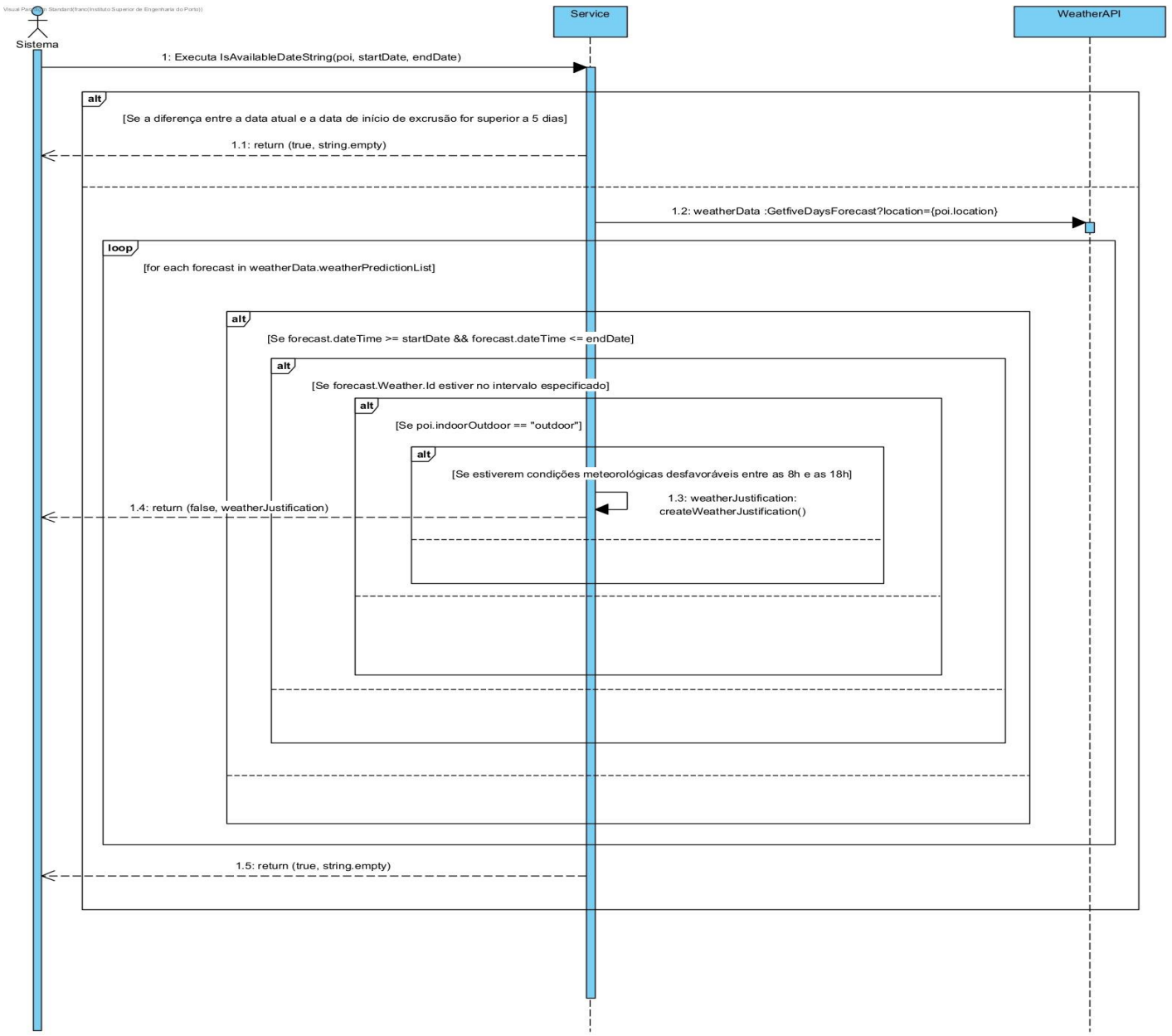

Sistema
Service
WeatherAPI
1: Executa IsAvailableDateString(poi, startDate, endDate)
alt
[Se a diferença entre a data atual e a data de início de excrusão for superior a 5 dias]
1.1: return (true, string.empty)
1.2: weatherData :GetfiveDaysForecast?location={poi.location}
loop
[for each forecast in weatherData.weatherPredictionList]
alt
[Se forecast.dateTime >= startDate && forecast.dateTime <= endDate]
alt
[Se forecast.Weather.Id estiver no intervalo especificado]
alt
[Se poi.indoorOutdoor == "outdoor"]
alt
[Se estiverem condições meteorológicas desfavoráveis entre as 8h e as 18h]
1.3: weatherJustification:
createWeatherJustification()
1.4: return (false, weatherJustification)
1.5: return (true, string.empty)

Figura 69 - Diagrama de Sequência da funcionalidade Integração com o Serviço Meteorológico Externo para Recomendação de Pontos de Interesse

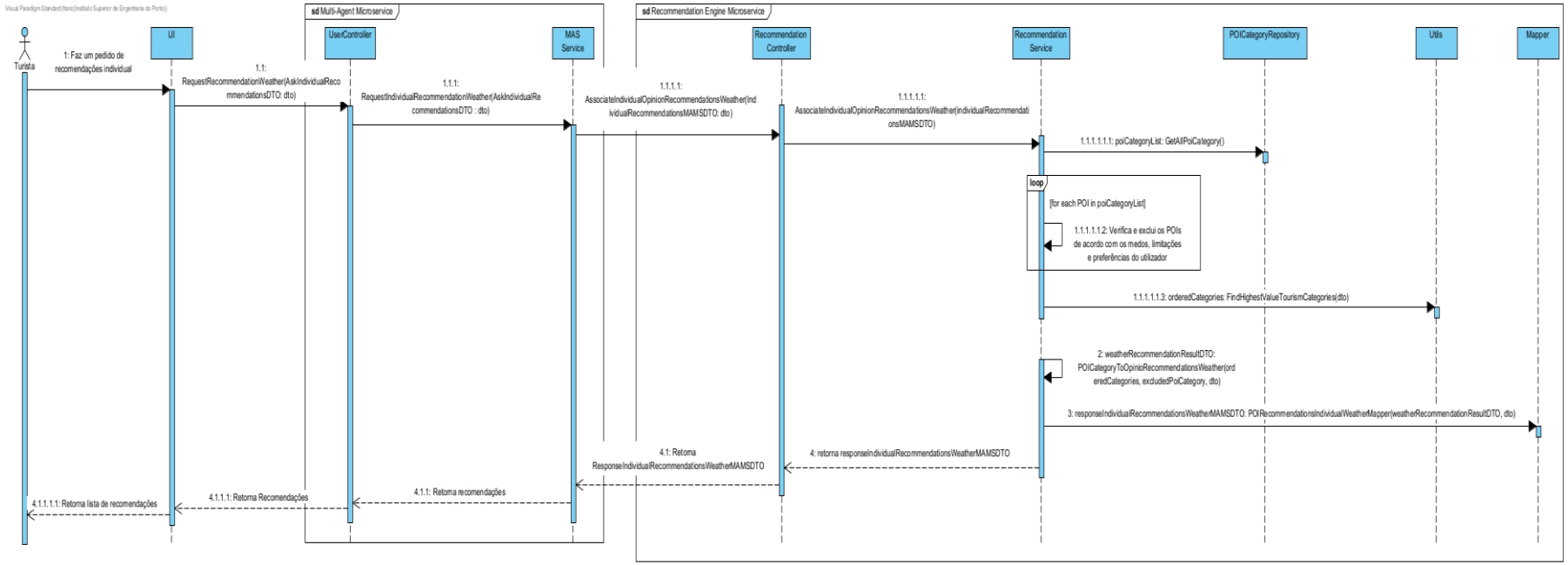


Figura 70 - Diagrama de Sequência da funcionalidade Recomendações Individuais Baseadas em Fatores Contextuais com Opção de Substituição de Pontos de Interesse sob Más Condições Meteorológica

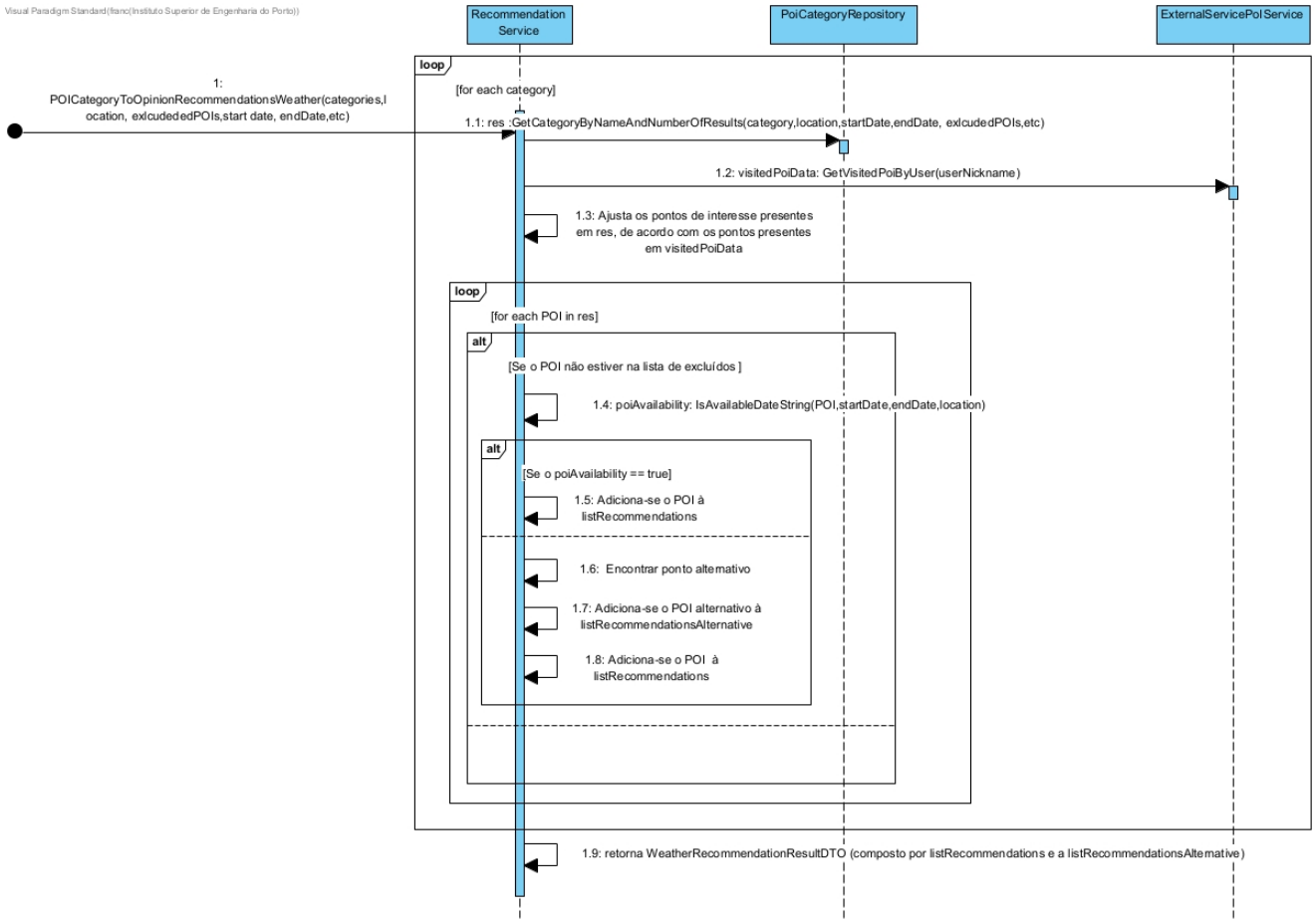


Figura 71 - Diagrama de Sequência do Método POICategoryToOpinionRecommendationsWeather

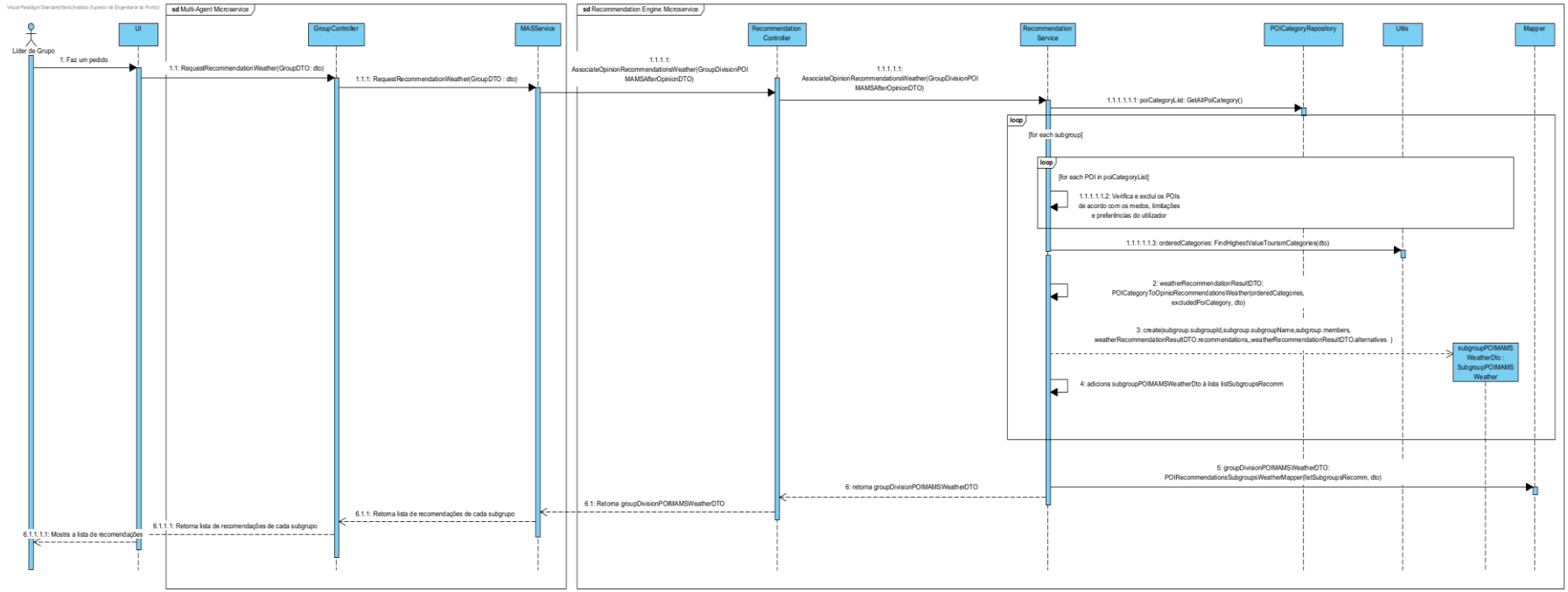


Figura 72 - Diagrama de Sequência da Funcionalidade Gerar Recomendações Para Grupos Baseadas em Fatores Contextuais Com Opção de Substituição de Pontos de Interesse Sob Condições Meteorológicas Desfavoráveis

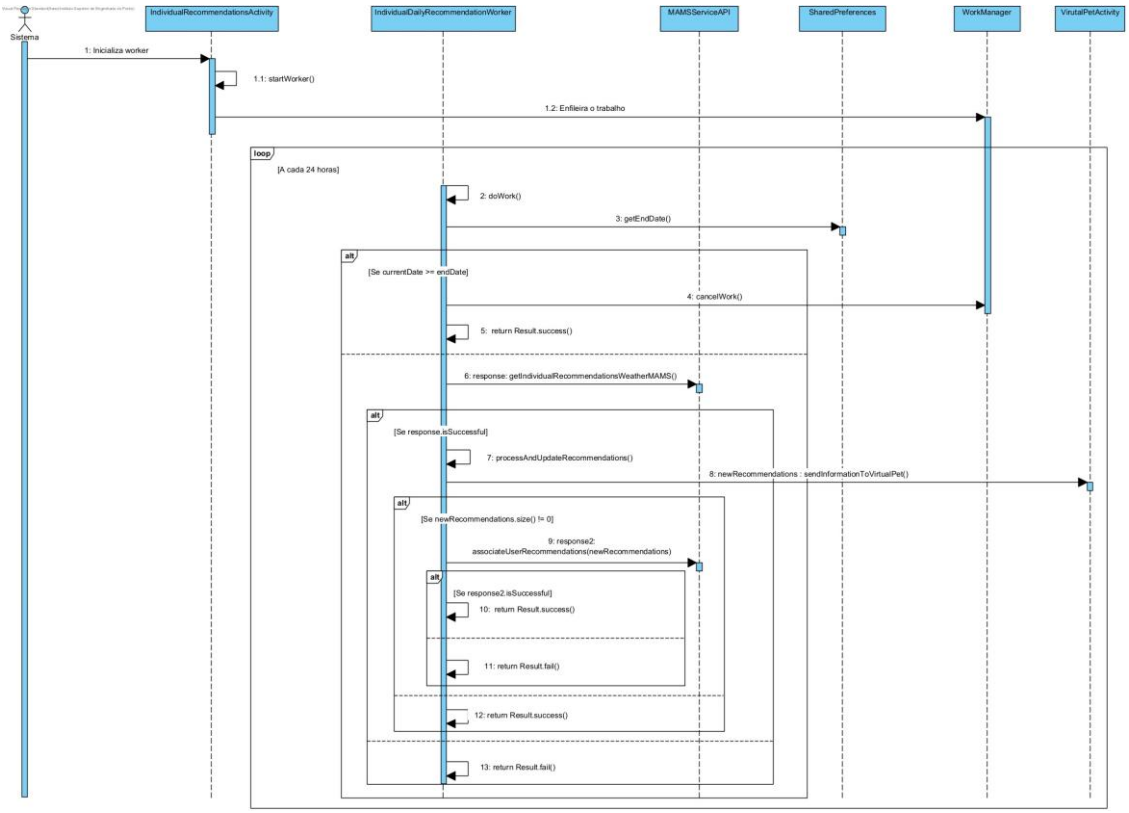


Figura 73 - Diagrama de Sequência do IndividualDailyRecommendationWorker

```
{
  "cnt": 40,
  "list": [
    {
      "dt": 1672941600,
      “weather”: {
        "id": 800,
        "main": "Clear",
        "description": "clear sky",
        "temp": 13.43,
        "feels_like": 12.47,
        "temp_min": 11.53,
        "temp_max": 13.43,
        "pressure": 1027,
        "sea_level": 1027,
        "grnd_level": 1022,
        "humidity": 63,
      },
      "clouds": {
        "all": 2
      },
      "wind": {
        "speed": 0.75,
        "deg": 280,
        "gust": 0.85
      },
      "visibility": 10000,
      “precipitation”: 0,
      “part_day”:  "n“,
      "dt_txt": "2023-01-05 18:00:00"
    },
    ...
  ],
  "city": {
      "id": 2735943,
      "name": "Porto",
      "coord": {
        "lat": 41.14,
        "lon": -8.61
      },
      "country": "PT",
      "timezone": 0,
      "sunrise": 1672905591,
      "sunset": 1672939146
    }
}
```


Figura 74 - Estrutura do JSON devolvido pela API Meteorológica

```
4 references
public string nickname { get; set; }
0 references
public string id { get; set; }
0 references
public string type { get; set; }
0 references
public string birthday { get; set; }
0 references
public string country { get; set; }
0 references
public string city { get; set; }
0 references
public string gender { get; set; }
3 references
public string locationCity { get; set; }
3 references
public string locationCountry { get; set; }
2 references
public string startDate { get; set; }
2 references
public string endDate { get; set; }
4 references
public List<string> limitations { get; set; }
4 references
public List<string> fears { get; set; }
0 references
public PersonalityDTO personality { get; set; }
4 references
public TravelPreferencesDTO travel_preferences { get; set; }
0 references
public MotivationsDTO motivations { get; set; }
0 references
public List<InfoVisitedPOIDTO> visitedPOI { get; set; }
5 references
public TourismCategoriesDTO tourismCategories { get; set; }
```

Figura 75 - Estrutura do JSON passado do *Multi-Agent Microservice* para o *Recommendation Engine Microservice*, quando se faz um pedido de Recomendação individual

```
using System.Collections.Generic;

namespace recommendationengine.DTO.MAMS
{
    8 references
    public class WeatherRecommendationResultDTO
    {
        3 references
        public List<RecommendationListDTO> recommendations { get; set; }
        3 references
        public List<RecommendationAlternativeDTO> alternatives { get; set; }
    }
}
```

Figura 76 – Estrutura do DTO *WeatherRecommendationResultDTO*

Figura 77 – Estrutura do DTO *ResponseIndividualRecommendationsMAMSDTO*

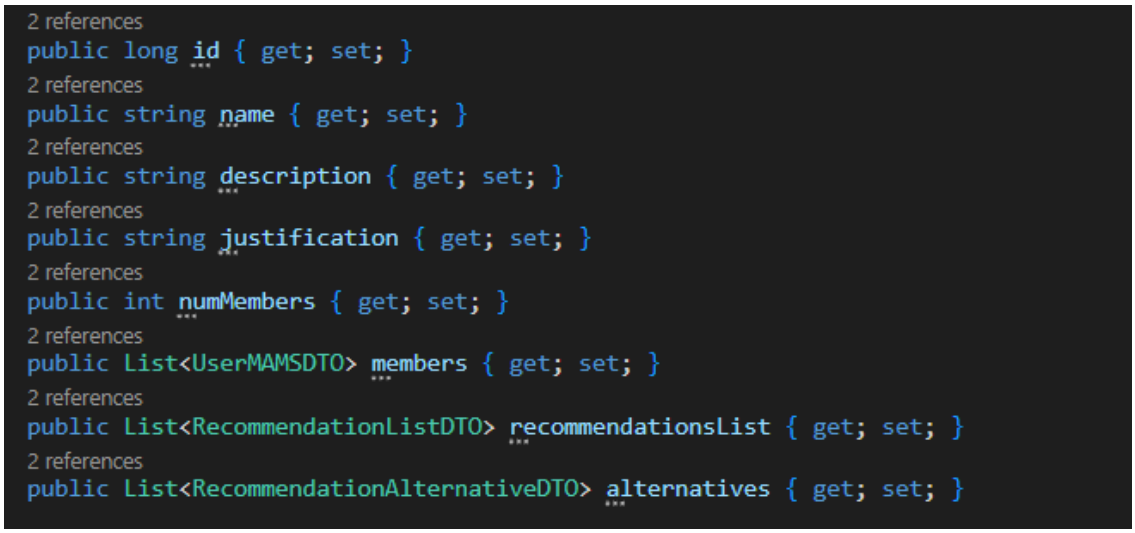


Figura 78 – Estrutura do DTO *GroupDivisionPOIMAMSWeatherDTO*

Figura 79 – Estrutura do DTO *GroupPOIMAMSDTO*

32. Considerando o método de pesquisa manual de atrações a visitar e a aplicação para obter recomendações, classifique as seguintes afirmações, numa escala de 1 (discordo totalmente) a 5 (concordo totalmente). *

	Discordo totalmente	Discordo	Não concordo nem discordo	Condordo	Concordo totalmente
Foi mais fácil procurar manualmente os locais a visitar do que usar a aplicação para isso	○	○	○	○	○
Foi mais rápido procurar manualmente os locais a visitar do que usar a aplicação para isso	○	○	○	○	○
Gostei mais de usar a aplicação para obter recomendações do que procurar manualmente os locais	○	○	○	○	○
As recomendações sugeridas pela aplicação foram do meu agrado	○	○	○	○	○
Consegui encontrar os 10 locais dentro do tempo permitido	○	○	○	○	○

Figura 80 - Questionário que foi pedido para preencher após obter as recomendações individuais

34. Considerando o método de pesquisa manual de atrações a visitar e a aplicação para obter recomendações para o grupo, classifique as seguintes afirmações, numa escala de 1 (discordo totalmente) a 5 (concordo totalmente). *

	Discordo totalmente	Discordo	Não concordo nem discordo	Condordo	Concordo totalmente
Foi mais fácil procurar manualmente os locais a visitar do que usar a aplicação para isso	○	○	○	○	○
Foi mais rápido procurar man ualmente os locais a visitar do que usar a aplicação para isso	○	○	○	○	○
Gostei mais de usar a aplicação para obter recomendaçõ es de grupo do que procurar manualmente os locais	○	○	○	○	○
As recomendaçõ es sugeridas pela aplicação foram do meu agrado	○	○	○	○	○
Conseguimos encontrar os 10 locais dentro do tempo permitido	○	○	○	○	○

Figura 81 - Questionário que foi pedido para preencher após obter as recomendações de Grupo