iscte

UNIVERSITY
INSTITUTE
OF LISBON

A mobile tour guide app for sustainable tourism

Filipe Eduardo da Silva Vida Larga

Master in Computer Science and Business Management

Supervisor:
Doctor Fernando Brito e Abreu, Associate Professor,
Iscte

Co-supervisor:
Doctor José Luís Cardoso da Silva, Assistant Professor,
Iscte

November, 2020

[This page has been intentionally left blank]

TECHNOLOGY
AND ARCHITECTURE

A mobile tour guide app for sustainable tourism

Filipe Eduardo da Silva Vida Larga

Master in Computer Science and Business Management

Supervisor:
Doctor Fernando Brito e Abreu, Associate Professor,
Iscte

Co-supervisor:
Doctor José Luís Cardoso da Silva, Assistant Professor,
Iscte

November, 2020

[This page has been intentionally left blank]

A mobile tour guide app for sustainable tourism

[This page has been intentionally left blank]

Acknowledgements

I would like to thank Professor Fernando Brito e Abreu for inviting me to this research project and for his guidance throughout the dissertation.
I would also like to thank my co-supervisor, Professor José Luís Silva, for his invaluable guidance and feedback.
I express my gratitude to Professor João Caldeira for his contribution to the template that made this document possible, and wish him good luck with his doctoral thesis.
I am particularly grateful for my colleagues in this research project, Duarte Almeida and Rúben Beirão, who showed great cooperation and comradeship, and without whom I would not have been able to complete this dissertation.
I would also like to express my gratitude to all my colleagues that accompanied me throughout this five year journey, for all the help and all the wonderful moments we shared together.
I would like to thank my dear friend Michael, for his valuable support and advice, which helped me improve the work done in this dissertation.
And finally, I am thankful for all the support and encouragement my family has provided me during this period.

Lisboa, November 2020

Filipe Eduardo da Silva Vida Larga

[This page has been intentionally left blank]

Abstract

Portugal has had a flourishing tourism sector for the past few years. In fact, Portugal's tourism boom has made the industry one of the biggest contributors to the national economy and the largest employer. In the year 2019, Portugal had a total of 27 million tourists, surpassing once again the record established in the previous year. However, tourism also brings a series of unintended negative side effects, such as overcrowding. The *Santa Maria Maior* historic district in Lisbon is being particularly affected by this problem.

The work undertaken in this dissertation is part of the *Sustainable Tourism Crowding* project, that aims to mitigate the overcrowding phenomenon in this district, by fostering a balanced distribution of visitors while promoting the visitation of sustainable points of interest. This dissertation focuses on developing a mobile app prototype targeted at tourists, through which these sustainable walking tour recommendations can be delivered.

To validate the functional requirements of the prototype, more specifically the trip creation process, a series of unit tests, integration tests, and manual tests were developed. To evaluate the usability of the prototype, a user-centered approach was adopted during the design stage, in which two usability techniques were conducted with members of ISCTE's research center *ISTAR* and partners from the *Junta de Freguesia de Santa Maria Maior*, that guided and validated the decisions made.

The achieved prototype contains mechanisms for measuring tourists' adherence to the recommended tours using the *Dynamic Time Warping* algorithm, which raises new research opportunities on tourists' behaviour.

Keywords: Tourism, Sustainability, Mobile Application, Mobile Tour Guide

[This page has been intentionally left blank]

Resumo

O desenvolvimento próspero do setor turístico em Portugal nos últimos anos fez da indústria um dos maiores contribuintes para a economia nacional e o maior empregador do país. No ano de 2019, Portugal recebeu um total de 27 milhões de turistas, ultrapassando uma vez mais uma vez o recorde estabelecido no ano anterior. No entanto, o turismo traz também uma série de efeitos secundários negativos não intencionais, tais como *overcrowding*. A freguesia histórica de Santa Maria Maior em Lisboa está a ser particularmente afetada por este problema.

O trabalho desenvolvido nesta dissertação faz parte do projeto de pesquisa *Sustainable Tourism Crowding*, que visa mitigar o fenómeno de *overcrowding* nesta freguesia, promovendo uma distribuição equilibrada dos visitantes e incentivando a visita de pontos de interesse sustentáveis. Esta dissertação foca-se no desenvolvimento de uma aplicação móvel protótipo destinada a turistas, através do qual recebem recomendações de visitas sustentáveis.

Para validar os requisitos funcionais do protótipo, mais especificamente o processo de criação de visitas, foram desenvolvidos testes unitários, testes de integração, e testes manuais. Para avaliar a usabilidade do protótipo, foi adotada uma abordagem centrada no utilizador durante a fase de conceção, em que foram utilizadas duas técnicas de usabilidade em parceria com o *ISTAR* (centro de investigação do ISCTE) e com a Junta de Freguesia de Santa Maria Maior, cujos resultados guiaram e validaram as decisões tomadas.

O protótipo desenvolvido contém mecanismos para medir a aderência dos turistas às recomendações sugeridas através do algoritmo *Dynamic Time Warping*, proporcionando novas oportunidades de pesquisa nesta área.

Palavras-chave: Turismo, Sustentabilidade, Aplicação Móvel, Guia Turístico Móvel

[This page has been intentionally left blank]

Contents

[This page has been intentionally left blank]

List of Figures

[This page has been intentionally left blank]

List of Tables

[This page has been intentionally left blank]

Listings

[This page has been intentionally left blank]

Acronyms

API	Application Program Interface.
AR	Augmented Reality.
BLE	Bluetooth Low Energy.
BLoC	Business Logic Component.
DTW	Dynamic Time Warping.
GIS	Geographic Information System.
IA	Information Architecture.
IDE	Integrated development environment.
MTG	Mobile Tour Guides.
MVC	Model-View-Controller.
MVP	Model-View-Presenter.
MVVM	Model-View-ViewModel.
PDA	Personal digital assistant.
POI	Point of Interest.
PWA	Progressive Web App.
RR	Rapid Review.
RS	Recommender System.
SDK	Software Development Kit.
STC	Sustainable Tourism Crowding.
UCD	User-centered design.
UI	User Interface.

[This page has been intentionally left blank]

Chapter 1. Introduction

Contents

This chapter introduces the motivation and scope of this work, describes the main research questions that it aims to answer, along side the goals and steps taken to achieve them, and the contributions achieved. Finally, it explains how this dissertation is outlined.

[This page has been intentionally left blank]

Chapter 1
Introduction

1.1 Motivation and scope

Tourism plays a crucial role in the economic development of Portugal [28]. In the year 2017, tourism contributed with 13,7% (or 26 844,7 million euros) to the total GPD of Portugal. As for the number of tourists, in the year 2019 Portugal had a total of 27 million tourists, surpassing the record established in the previous year (7,2% increase). These tourists stayed an average of 2.6 nights in accommodation establishments (totaling 69.8 million overnight stays), which converted into a revenue of 3.2 million euros.

Portugal is regarded as one of the best tourist destinations and has been respectively awarded as so in multiple categories. The *World Travel Awards* organization has recognized Portugal as the *World's Leading Destination* three times in a row (2017 to 2019)[1]. The city of Lisbon in particular has won several awards, such as being the *Europe's Leading City Destination* in 2018[2] and having the *Europe's Leading Cruise Port* from 2016 to 2020[3].

However, tourism also brings a series of unintended negative side effects, such as overcrowding. Overcrowding has many negative impacts [64], such as increasing the costs of living and of real estate, causing a decline in the quality of life of the locals, increasing the noise pollution of the city and more. Another aspect of overcrowding that affects the city of Lisbon is the loss of authenticity and of cultural and natural heritage. What once were local stores and small businesses, are now being replaced by *"tourist traps"* and large multinational franchises (e.g. *Starbucks* and *McDonalds*).

The Sustainable Tourism Crowding (STC) research project, an initiative by ISCTE's research center *ISTAR*, is an ongoing project in the *Santa Maria Maior* district in Lisbon, to mitigate overcrowding by fostering a balanced distribution of visitors, while promoting the visitation of sustainable Point of Interest (POI)s. The work presented in this thesis is part of this research project.

At the same time that tourism is growing to unsustainable levels, smartphones have been gaining significance both in general and in the subject of tourism specifically. Computers have been dethroned from the role of people's primary device by smartphones [89]. As of late 2019, mobile phones have 52% of the market share worldwide, while desktops decreased to 45%[4]. A study by eMarketer[5] shows that about 90% of the time spent on smartphones is using apps.

These applications enable consumers to access a variety of services and resources without the need for a web browser and establish a new channel for purchases [86]. For businesses, apps serve as a way to interact closer with mobile shopper from any location at any time, enhancing the effectiveness of promotions and increasing customer loyalty [13].

[1]https://www.worldtravelawards.com/award-worlds-leading-destination-2019
[2]https://www.worldtravelawards.com/award-europes-leading-city-destination-2018
[3]https://www.worldtravelawards.com/award-europes-leading-cruise-port-2020
[4]https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide
[5]https://www.emarketer.com/content/us-time-spent-with-mobile-2019

Smartphone's versatile features, such as GPS and Bluetooth, have already allowed for the development of some very interesting apps that help travelers in tourism-related contexts, such as museum tours [14], hiking [3], and skiing [39]. As smartphone capabilities continue to evolve, so does the tourism industry. New features such as Augmented Reality can offer unique and interesting ways to provide services to tourists [50].

In 2017, Google and PhocusWright conducted a study that demonstrates the predominance of smartphones in tourism [38]. Over 70% of U.S. travelers report that they always use their smartphones when traveling, up from 41% in 2015, showing an increasing interest in these devices for travelling. This includes researching attractions, restaurants and looking up directions. Figure 1.1 illustrates the adoption of smartphones for the planning process in seven countries around the world. Furthermore, this study highlights the importance of presenting information that is personalized and easy to navigate, as having friction in mobile experiences can lead to travelers trying different ones.

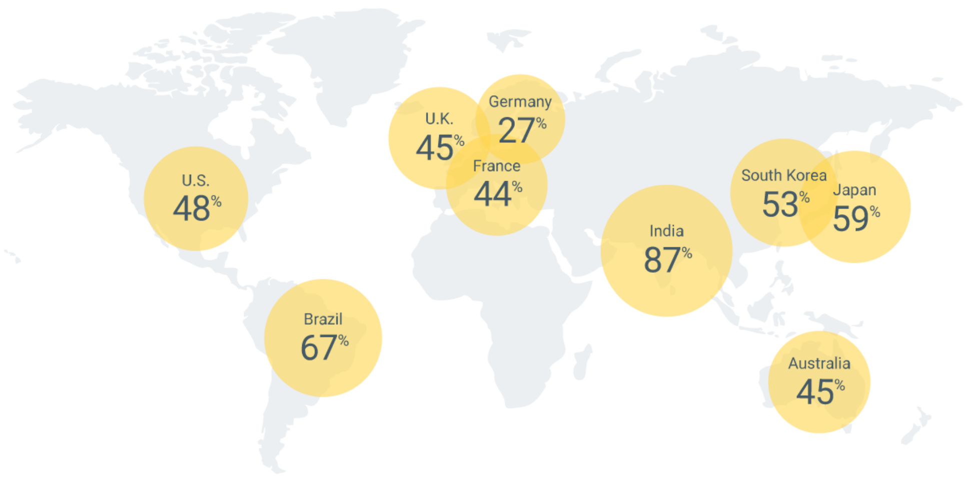


Figure 1.1: Percentage of smartphone users that are comfortable researching, planning and booking their entire trip using only mobile devices. [38]

The ability to support the traveler in every stage of the trip, especially on the move and within the destination, where other devices fall short, is also one of the reasons why smartphones have become so popular in tourism. [41]. This opportunity makes mobile apps the ideal platform to target tourists directly and align their behaviour with the goals of the STC project, hopefully changing the landscape of tourism in Lisbon.

The STC project started in 2019 and has already led to interesting case studies and prototypes. The work of Silva et al. [78] contributed to the detection of crowding using multiple radio techniques, while the work of Peixoto et al. [65] contributed with the classification of POIs in terms of sustainability, and with research on generating sustainable walking tours. This year, two new student members will continue to develop the project, one focusing on advancing the creation of sustainable walking tours, and the other focusing on developing the app through which tourists can obtain them. This dissertation corresponds to the latter, describing the work undertaken to design and develop an app prototype and integrating with the rest of the remaining components of the STC project.

1.2 Research questions and goals

The main research question the present dissertation aims to answer is:

- Is it possible to create a mobile tour guide app that promotes sustainable behaviours in tourists and potentially be used to reduce the overcrowding phenomenon in pressured cities.

The research question can be broken down into the following research goals, which this dissertation aims to achieve:

- Design of a suitable information architecture and user interface for a mobile tour guide that aligns with the requirements of the STC project;
- Development of a functional prototype cross-platform mobile app according to the resulting design specification and integration with the remaining components of the STC project;
- Plan the quality of service for a mobile tour guide in the scope of an intermittently available network.

1.3 Research steps

The steps to be taken for answering the first research goal are:

- Review other mobile tour guides by doing a systematic rapid review;
- Review the literature for the topic of usability in mobile applications and specifically mobile tour guides;
- Develop a user-centered design process for the prototype where relevant user groups are involved in the design stages;
- Use the results from said process to design a final version of the information architecture and user interface, using a specialized design tool such as *Figma*.

The steps to be taken for answering the second research goal are:

- Research what cross-platform mobile software development kits are available and determine which one is more suited for the project;
- Determine the functional requirements for the APP component of the STC project, together with the advisers, and co-student of the trip generator element;
- Implement the user interface and the functional requirements into a prototype mobile application;
- Integrate the developed prototype with the other components of the STC architecture.

- Develop a series of unit tests, integration tests, and manual tests in order to perform a functional validation.

The steps to be taken for answering the third research question are:

- Determine which network dependant features are considered crucial and which ones can be relinquished or downgraded without disrupting the activity flow of the user;
- Determine which of these features can be with modified the usage of local storage as an alternative;
- Design and implement the model classes that will store the data pertaining to these features;
- Investigate what are the different local storage options available for the chosen software development kit;
- Design and implement the caching mechanisms using one of the local storage options above.

1.4 Main contributions

This section presents a summary of the main results and contributions achieved with the development of this dissertation:

Contribution 1 - Development of a functional prototype mobile application for creating trips and guiding users through them, according to the needs and requirements of the STC project. The source code is available on the STC repository on GitHub so that everyone can use it as a starting point to develop and investigate their own research questions.

Contribution 2 - Integration of the developed prototype with the other components of the STC architecture. This includes the design and documentation of multiple APIs (and respective JSON schemas) using a specialized tool that easily allows for future improvements. This is also open-source and included in the STC repository on GitHub. See section 3.2 for more details.

Contribution 3 - A literature review on the topic of mobile tour guides, with a taxonomy focused on the features and characteristics of the apps. See section 2.3.3 for more details.

Contribution 4 - A brief introduction and review of the Flutter software development kit which, due to its novelty, has not seen much discussion in the scientific world. See section 4.1 for more details.

1.5 Dissertation outline

This dissertation is organized in six chapters, which are briefly summarized as follows:

Chapter 1 - Introduction. This chapter provides context for work undertaken in this dissertation, introduces the research questions and subsequent steps to answer them, as well as the main contributions.

Chapter 2 - State of the Art. It gives an overview of the related work, using the systematic literature review approach for the topic of mobile tour guides.

Chapter 3 - Sustainable Tourism Crowding Project. This chapter presents the STC technical architecture, explains how the work undertaken in this dissertation fits into the main project and how it interacts with the other systems.

Chapter 4 - Development Approach. In this chapter we cover the technical decisions taken to develop the prototype. This includes the architecture of the application and the software development kit used for development. Additionally, this chapter contains the tests carried out for the functional validation of the prototype.

Chapter 5 - Application Design. This chapter presents the high level functionalities of the prototype and the design decisions taken. Furthermore, it contains the process of validating the user interface.

Chapter 6 - Conclusion. It concludes and summarizes the achievements of this dissertation. It also opens the discussion for challenges and future work in this area.

[This page has been intentionally left blank]

Chapter 2.

State of the Art

Contents

This chapter describes the systematic rapid review undertaken for the topic of mobile tour guides. It includes the review protocol and the discussion of the results.

[This page has been intentionally left blank]

Chapter 2
State of the Art

2.1 Introduction

In order to gain a better understanding of the scientific work done in the area of Mobile Tour Guides (MTG), and to attain a clearer definition of the problem being researched, a systematic Rapid Review (RR) was performed on this topic. A RR is a methodology for identifying, evaluating and interpreting relevant published research relevant to a particular topic of interest. It is a more streamlined version of a traditional systematic literature review with a smaller time frame [48]. Using this technique, we can guarantee that our review is thorough and fair, since otherwise it would be of little scientific value.

This RR follows guidelines proposed by [17] for systematic rapid reviews adapted to reflect the specific problems of software engineering research and follows the proposed review protocol. Based on this reference, this RR is partition into three main phases planning, performing, and reporting. The planning phase of a systematic RR implies the creation of a protocol to define all the decisions and procedures demanded to conduct the RR. Overall, this systematic RR had a duration of 5 weeks from January to February.

This systematic rapid review will focus on MTGs that were covered in primary studies. Be aware that many MTG are "just" commercial applications, therefore not subject to research scrutiny.

2.2 Review protocol

The purpose of the review protocol is to specify the plan which the review will follow to identify, appraise and collect primary studies [17]. A pre-defined protocol is necessary to reduce the possibility of researcher bias [49]. Taken from these previously mentioned references, our review protocol contains research objectives that the review is intended to accomplish; a strategy that will be used to search for primary studies, including search terms and databases and a selection criteria for including or excluding studies from the systematic RR. Figure 2.1 summarizes the review protocol.

2.2.1 Research objectives

The objectives of this review are to:

- Undertake a systematic rapid review of research on the benefits and limitations of mobile tour guides.
- Select a sub-set of studies to review in-depth.
- Identify any gaps in current research in order to suggest areas for further investigation.

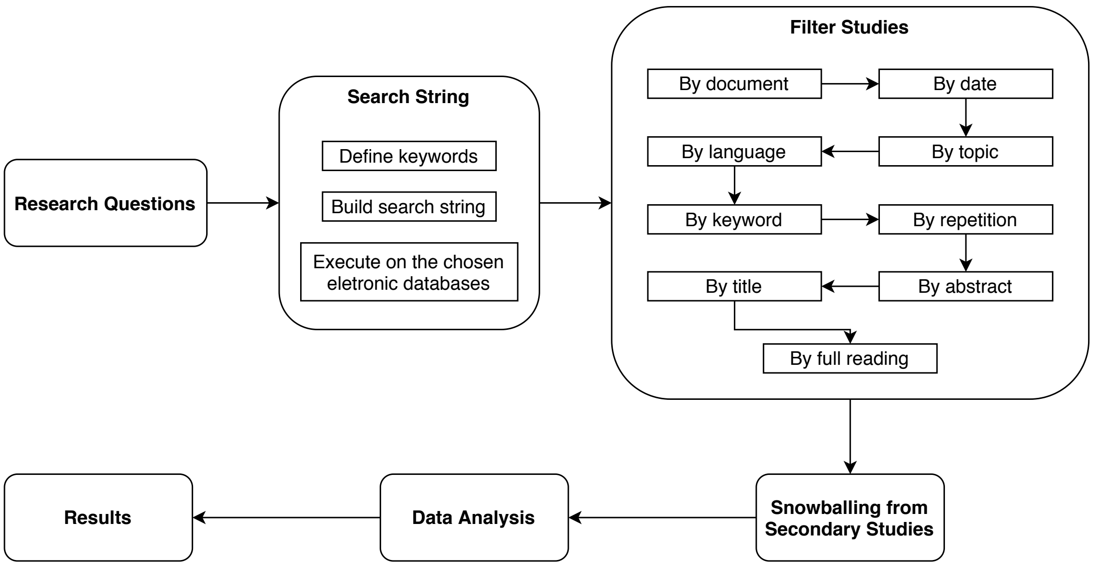


Figure 2.1: Review protocol

2.2.2 Search strategy

In order to identify studies that directly relate to the systematic rapid review questions and objectives defined above, the implemented search strategy consists of searching established bibliographic databases in the area of software engineering with specific search strings, reviewing secondary studies and going through the references quoted in these articles, also known as *snowballing* [91].

Search Bibliographic Databases:

The following bibliographic databases have a vast collection of computer science related content that is suitable for the topic of MTGs.

- Scopus (https://www.scopus.com)
- Web of Science (https://www.webofknowledge.com/)
- Research Gate https://www.researchgate.net/

Search String:

Although we adopted the term Mobile Tour Guide (MTG), it is not the only term used in the literature. Therefore, we also included other terms in the search string:

("mobile application"OR "mobile tour guide"OR "mobile tourist guide"OR "personalised electronic tourist guide"OR "mobile recommender system"OR "mobile travel guide") AND ("travel"OR "trip"OR "attraction"OR "visit"OR "point of interest")

2.2.3 Selection criteria

The selection criteria is fairly simple (Tables 2.1 and 2.2). We include every study relevant to mobile tour guides written in English that is peer-reviewed and published in a conference or journal or book. Like previously mentioned, we exclude commercial publications on MTGs that have little or no scientific value.

Table 2.1: Inclusion criteria.

Criterion	**Description**
IC1	Written in English.
IC2	Published in conferences or journals or books.
IC3	Title relevance regarding mobile tour guides.

Table 2.2: Exclusion criteria.

Criterion	**Description**
EC1	Studies published before the year 2000.
EC2	Only researched-based articles are considered.
EC3	Articles classified with irrelevant topics, keywords or titles.
EC4	Articles with similarly relevant titles but irrelevant content.

2.2.4 Results

By following the protocol 14 primary studies have been found. Ten come from executing the search string (Table 2.3) and another four have been added from snowballing from the secondary studies. Note that, as part of the revision process for the final submission of this dissertation, two additional studies were added so that the RR considers a few more recent approaches to MTGs as well.

2.2.5 Validity threats

Just as described in [16], this RR is not without its validity threats. This RR includes a total of 16 primary studies, which is not representative of the number MTGs in the literature and might call into question the reliability of the results. Some of the criteria included in the proposed taxonomy is not objectively defined (i.e. the *Personalization* and *Usability* criteria) and they are not always clearly described by the authors, which might have led to an incorrect classification of the extracted data.

Table 2.3: Search string execution

	SCOPUS	Web of Science	Research Gate
Results from search string	1522	333	100
After removing by document type	1415	326	100
After removing by date	1271	309	100
After removing by topic	744	87	100
After removing by language	731	83	100
After removing by keyword	138	83	100
After removing duplicates	138	74	82
After removing by title	46	20	24
After removing by abstract	12	10	15
After removing by full reading	4	3	3

2.3 Related work

In this section we conduct a short review of some secondary studies conducted in the area of MTGs, which will serve as guidance for our own proposed taxonomy. Then we review the primary studies resulting from the review protocol and classify them according to the taxonomy. Finally, we present a short discussion about the studies reviewed and how we will explore areas not touched by these studies.

2.3.1 Secondary studies

Secondary studies review primary studies (individual publication or a study) related to some research questions to integrate/synthesize evidence related to those questions [49], much like our RR. These studies can not only help us to find good primary studies of the same topic but also to understand how these primary studies can be evaluated and categorized. This is particularly useful for the definition of our own taxonomy. In the subject area of mobile tour guides there are a few secondary studies that our taxonomy benefited from (Table 2.4).

Kenteris et al. [45] is a paper that focuses on the evaluation of research and commercial applications which are used by tourists (and not only) to get information, navigation, guidance or just cultural information using a mobile device. It classifies these applications *"using a detailed set of evaluation criteria to extract design principles"*. This includes parameters such as the architecture/network infrastructure, input/output modalities and position/map technologies.

Karanasio et al. [43] is another secondary study used as reference. In this paper the authors propose a frame of reference for the categorisation of mobile tourism applications (not exclusively tour guides) and a framework for evaluating mobile tourism applications. This framework includes parameters such as customization and the type of service delivered.

Borràs et al. [11] review a number of MTGs (named as intelligent tourism recommender systems) according to their interfaces and functionalities, but also according to the type of recommendation techniques and artificial intelligence techniques used by these systems. Additionally, this article also provides some guidelines for the development of future MTGs.

Finally, Gavalas et al. [37] is an article that reviews mobile recommender systems in the subject of tourism. In this article we find classification propositions for these systems according to the services offered, based on the architectural style of the system, based on the degree of user involvement and based on the criteria taken into account for deriving recommendations.

Table 2.4: Secondary studies

Authors	Year	Title
Kenteris et al.	2008	Evaluation of Mobile Tourist Guides
Karanasio et al.	2011	A Classification of Mobile Tourism Applications
Borràs et al.	2014	Intelligent tourism recommender systems: A survey
Gavalas et al.	2014	Mobile recommender systems in tourism

2.3.2 Taxonomy proposal

In order to better understand the characteristics and to compare the strengths and restrictions of MTG apps, there is a need to define concrete criteria by which they will be classified. The proposed taxonomy is based on all of the previously mentioned secondary studies, but takes a perspective that is more concerned with the aspects and functionalities of the apps that bring value to the end-user, disregarding some of the more technological and behind-the-scenes aspects of the routing system. In the following subsections we describe each of the criteria composing the proposed taxonomy.

2.3.2.1 Development approach

Today, the number of development options for mobile applications is high. Choosing a Software Development Kit (SDK) is perhaps the biggest decision developers have to make that will highly impact the development process. Different options have different advantages and drawbacks so there is no definitive correct answer. Evaluating these primary studies according to their development approach is important because it is a decision that will have to be made for our own work, so we must understand what reasons and considerations underly the authors' decisions/choices.

We consider three categories for this criteria:

- Native Applications - Separate apps made with/for each operating system's native SDK;
- Hybrid Applications - Apps that combine both native development and web technology;
- Web Applications - Mobile web apps or progressive web apps that run on the device's browser.

2.3.2.2 Availability

This criterion is concerned with the number of end-users (i.e. tourists) that have access to the MTG during their travels, since targeting more device types enlarges the potential size of the target audience. Accordingly, these are the categories that describe the portion of mobile devices supported, following an ordinal scale:

1. **Limited Audience (Specific Device Type)** - These apps were developed with a specific target device in mind and are not distributed publicly (e.g. e-book reader app for the *Kindle* device).

2. **Moderate Audience (Single Device Type)** - Only a subset of devices are targeted (e.g. supporting only iOS or Android).

3. **Mass Audience (Multiple Device Types)** - These apps were intended for mass distribution and support numerous types of devices.

2.3.2.3 Personalization

Mobile tourism leverages many opportunities to provide highly accurate and effective tourist recommendations that respect personal preferences and capture usage, personal and environmental contextual parameters [37]. MTGs that provide more freedom and customization, appeal not only to those who feel comfortable with guided tours, but also to users who enjoy a more dynamic and unrestricted travel experience [51].

Inspired by the reviewed secondary studies [11, 37], personalization is defined by the extent that the app customizes it's experience according to users' preferences and external factors. This criterion is classified following an ordinal scale:

1. **Basic Personalization** - These MTGs allow for some user input, although in a very limited manner. Personalization consists only of adapting to the user's location, time constraints and which POIs they want to visit.

2. **Intermediate Personalization** - The recommendation system takes the user's preferences and constraints into consideration in order to give a tailored experience. There are many ways in which the user can interact with the recommendation system in order to get a custom experience.

3. **Advanced Personalization** - Along with taking direct user interaction to mold the details of the trip, these MTGs also collect information from other sources, such as weather forecasts or monitoring user's behaviour to readjust recommendations automatically.

2.3.2.4 Usability

Usability is a particularly interesting topic for MTGs, given their particular usage context. It is important to evaluate the usability criterion on the primary studies to understand what special considerations the authors had when developing their apps. The selected primary studies will be categorized according to the level of concern for usability found in the papers. For this criterion, we propose the following ordinal scale :

1. **No Concern** - There is a general lack of concern for the usability of the app. Few or even no mentions on usability can be found in the study.

2. **Average Concern** - Some consideration for usability was addressed by the authors that ensure the app provides a satisfactory experience when interacting with its features.

3. **High Concern** - The authors designed effective solutions to improve the user experience resulting in a very intuitive and usable app.

2.3.2.5 Social features

The taxonomy proposed by Kennedy et al. [44], classifying the services provided by tourism apps categorizes apps with social features as either social networks apps or communication apps. This includes apps with a sharing, collaboration, communication or social component. This taxonomy does not seem to be the most appropriate for our purposes, as MTGs can have social features without being outright social networks or communication based apps.

Social features are classified into two categories by Zhao et al. [99], differentiating interactions among contents from interactions among users. The following classification is adapted from the one proposed by this author:

1. **No Social Features** - No social features are offered by these apps.

2. a) **Interactions among contents** - User-generated content (UGC), content personalization, content rating and commenting, and content sharing are some of the features provided by this type of apps.

   b) **Interactions among users** - These apps contain features like follow/unfollow people, online chat (peer-to-peer conversation among people), and invite contacts from existing social networks.

3. **Hybrid Interactions** - These apps contain features from both types of interaction.

2.3.2.6 Evaluation method

This final criterion is concerned with how the authors evaluate and validate their apps. There are multiple aspects that can be evaluated in a MTG, such as the routing and recommendation algorithms. In our case, we are more concerned about evaluating the user experience and usability of the app. Accordingly, these are the categories that describe the different methods:

1. **Evaluation by experts** - Refers to the evaluation of the app by consulting with experts in the area (e.g. user-centered design).

2. **Empirical evaluation** - Refers to the testing of the app by real users (e.g. a field trial). Usually followed by a usability survey.

2.3.3 Primary studies

2.3.3.1 Cheverst et al., *"Developing a context-aware electronic tourist guide: Some issues and experiences"*, 2000

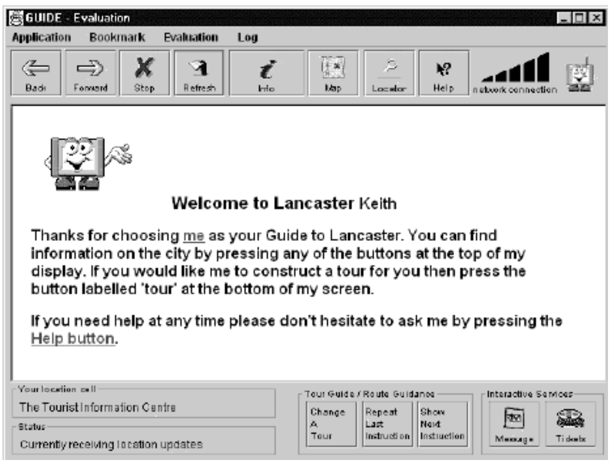
GUIDE was one of the first context-aware MTG prototypes to be developed. It was built for the city of Lancaster in the year 2000, with the goal of "*overcoming many of the limitations of the* *traditional information and navigation tools available to city visitors"*. The system couples personal mobile devices with wireless communications and context awareness to satisfy the information and navigation needs of tourists.

The system tries to accommodate different types of tourists: the ones that prefer guided tours and the others that prefer to explore on their own. GUIDE is flexible enough to work as an intelligent tour guide or as a richly featured guidebook, enabling visitors to explore Lancaster in their own way. Additionally, travellers never have to feel pressured to adhere to the plan, as the system adjusts to unexpected attraction visit times or in-between breaks.

Another set of features this app includes is accommodation bookings and an electronic messaging service with other visitors or the Lancaster's Tourist Information Centre. The authors noted that tourists make repeat visits to the Tourist Information Centre, often during the course of a single day, to request some sort of help/information or to book accommodations.

The app was developed for a single device, namely, the *Fujitsu TeamPad 7600* Personal digital assistant (PDA). Unlike most of the reviewed MTGs, this one does not utilize GPS in order to obtain positioning information. Instead, GUIDE devices receive location messages that are transmitted from strategically positioned base stations. Admittedly, this approach results in a lower resolution of positioning information. Wireless communication is also done differently than most MTGs. A number of WaveLAN cells were deployed in the city, each one supported by a GUIDE server. These cells disseminate data to the GUIDE devices using a broadcast approach.

The user interface (Fig. 2.2) is based on web browsers, as the authors believed that the *"growing acceptance of the web and the increasing familiarity of the browser metaphor"* would make the system easy to use and to learn. From this interface the user can: (1) retrieve information about attractions, weather, news and general information about the city, (2) navigate using a map, (3) create and follow a tour of the city, (4) communicate with other visitors or the Tourist Information Centre and (5) book accommodations. Some of these features are network dependent, making them unusable during periods of disconnection. This is represented in the user interface by "greying-out" said actions and buttons (direct manipulation paradigm), helping the user understand the state of the system and ultimately avoid seeming unpredictable and unreliable.



Welcome to Lancaster Keith

Thanks for choosing me as your Guide to Lancaster. You can find information on the city by pressing any of the buttons at the top of my display. If you would like me to construct a tour for you then press the button labelled 'tour' at the bottom of my screen.

If you need help at any time please don't hesitate to ask me by pressing the Help button.

The Tourist Information Centre

Currently receiving location updates

Figure 2.2: User interface [21]

In order to determine the initial requirements of the app, the authors conducted *"a series of semi-structured, one-to-one interviews with members of staff at Lancaster's Tourist Information Centre"*. Furthermore, they observed the information needs of tourists in the Tourist Information Centre for several days. Later evaluations were made using two different approaches. The first one used four experts in the area of user-centered design and computer supported learning that tested the full range of the app, resulting in valuable feedback, some used to improve GUIDE for the following evaluation. This next evaluation consisted of a field trial. Visitors were asked to use the prototype freely instead of having to perform a predefined set of tasks. Their interactions with the system were logged, they were encouraged to audio record their thoughts and, in the end, a semi-structured interview was conducted in order to obtain the visitor's subjective opinion of the system.

The system is capable of adapting to the user's context (e.g., visitor's interests, current location), environmental context (e.g., the time of day and the opening times of attractions) and it supports dynamic information notifications (e.g., notifying the user that an interesting attraction that was previously closed is now open).

Although GUIDE is network dependent, it tries its best to function offline by locally caching parts of the information model. Unfortunately, during long periods of disconnection the cached information can easily become outdated.

Table 2.5: Cheverst et al. (2000) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Web	Limited Audience (Specific Device Type)	Intermediate Personalization	Average Concern	Interactions Among Users	Empirical Evaluation

2.3.3.2 Dunlop et al., *"Design and development of Taeneb City Guide: From Paper Maps and Guidebooks to Electronic Guides"*, 2004

This MTG is called the Taeneb City Guide and was developed by researchers from the University of Strathclyde, UK. It proposes to improve on the available literature by focusing on these three main features: (1) query-able dynamic map interface, (2) dynamic information content and (3) community review systems and users' forum. Granted, these features can be found in many of the more recent MTGs reviewed, but perhaps not so much in the year 2004.

The article does not provide much detail about the system architecture that complements the MTG, focusing rather on the three main features previously mentioned.

The user interface for the Taeneb City Guide was designed with two main types of views: a "Map View Display" (Fig. 2.3) and a "List View Display" (Fig. 2.4). These two views represent maps and guidebooks, the two types of publications most often used by tourists. *"Using maps with guidebooks solves the problem of where attractions are and what attractions are in a given location"*.

The map view is very interactive, supporting features such as zooming and panning, pointing and tapping on symbols of attractions for more information and filtering of the attraction

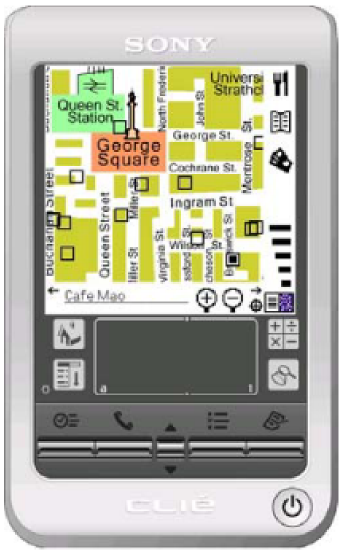


Figure 2.3: Map View display [26]

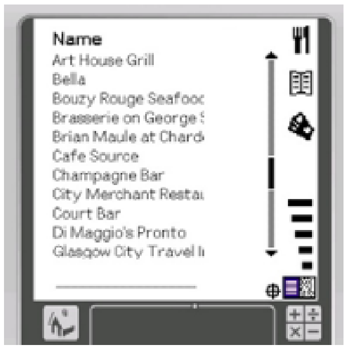

Name
Art House Grill
Bella
Bouzy Rouge Seafooc
Brasserie on George S
Brian Maule at Chard
Cafe Source
Champagne Bar
City Merchant Restau
Court Bar
Di Maggio's Pronto
Glasgow City Travel I

Figure 2.4: List View display [26]

categories. While the authors claim that *"maps were specially drawn for easy reading and use by a tourist while walking"*, it appears that the fact that it is not using Google Map's SDK (perhaps not available at the time), but instead this custom solution (Fig. 2.3), really harms the readability and usability of the interface, specially on small devices.

The list view contains a list of attractions, much like an index at the end of a typical guidebook, with the advantage of providing different sorting criteria to the user. The authors believe that *"simply switching between the displays allows making a quick connection between where attractions are, brief descriptions and reviews"*.

Tourist attractions can be very volatile. In order to provide current and up to date information to tourists *"Taeneb City Guide uses a central database server to store its content including the data about all the attractions and associated events"*. This central database can easily be updated via a web-based interface. Mobile devices synchronise their data directly using Wi-Fi, GSM and GPRS mobile phone networks or through a PC via a Hot Synch process.

Alternatively, for users that prefer to arrange their trips in advance and then synchronise the information into their mobile devices, a web site was created (www.citygogo.com, no longer available) with the same functionalities as the MTG.

Although the development approach of this app is unknown, it seems that only PDAs with the mobile operating system PalmOS were capable of using the it.

One of the main features of this MTG is a review system where tourists can review attractions privately or publicly. They could rate attractions out of a scale and make an additional text review. These reviews were filtered so that travelers only find reviews of other tourists with similar demographic profiles.

An initial usability trial was conducted in the city of Glasgow for two days. Twenty participants were recruited from an international conference happening at the same time. They used the application mostly for searching for restaurants and found the review system to be very useful.

Table 2.6: Dunlop et al. (2004) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Unknown	Moderate Audience (Single Device Type)	Basic Personalization	Average Concern	Interactions Among Users	Empirical Evaluation

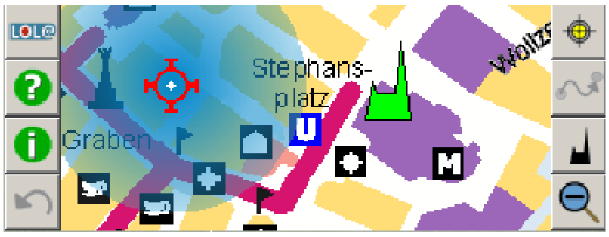


Figure 2.5: User Interface [69]

2.3.3.3 G. Pospischi et al., *"Designing LoL@, a Mobile Tourist Guide for UMTS"*, 2002; G. Pospischi et al., *"LOL@, A MOBILE TOURIST GUIDE FOR UMTS"*, 2002

LoL@ is a research prototype of a MTG designed for tourists visiting the city center of Vienna. It provides predefined tours and point of interest information. It is meant to be an *"interactive version of the classic printed guidebook"*. It was designed with these three usage scenarios in mind: (1) a tourist walks through the city, (2) a tourist plans his/her sightseeing tour in the hotel room and (3) a user accesses tour information and personal tour data from a desktop PC.

The system architecture is very complex and composed of many different technologies. The most relevant ones are the Java applets that power the app and the Java servlets that power the server. The app is implemented on top of a web-browser. Multimedia, speech recognition, voice transmission and route calculation all use third party solutions. For location-based functionality the system utilizes GPS, but expected to support other sources (e.g. cell ID and radio signal propagation) to complement it in the future.

The user interface used a browser metaphor where users interact by clicking on icons, hypertext links, and buttons, by selecting from menus, and through spoken commands. A set of predefined voice commands can be used as shortcuts to perform specific actions.

Using something different than the widely accepted map SDKs (e.g. Google Maps, Mapbox) can be a risky way to compromise the usability of an app. However, since none of this options were available at the time, a custom solution was implemented (Fig. 2.5). Two variants of the city of Vienna map were available: an "overview map" and a "detail map". In the overview map, POIs are grouped into clusters to avoid clutter. In the detail map, individual POI and street names are shown. Important POIs have unique symbols and more common POIs have category symbols (e.g. museum). Included, as well, is the user's location and a distinction between already visited and remaining parts of the tour.

The tour diary allows users to upload information (including pictures) about their trip. All visited POIs are automatically included. This data is stored remotely on a server, making it accessible from other devices and locations.

The authors showed great care when designing the user interface. Things like the color scheme were considered to accommodate for color blind people and the elderly. The only negative remark in usability is the constant need of network connectivity.

Provisions were made to evaluate usability issues and technical aspects of the system were made: students attending a user interface design and usability course would conduct a usability study in the summer of 2002.

Table 2.7: Pospischil et al. (2002) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Web	Moderate Audience (Single Device Type)	Basic Personalization	High Concern	No Social Features	N/A

2.3.3.4 Kramer et al., *"Behavioural Impacts of Mobile Tour Guides"*, 2007; Kramer et al., *"Development and evaluation of a context-driven, mobile tourist guide"*, 2008

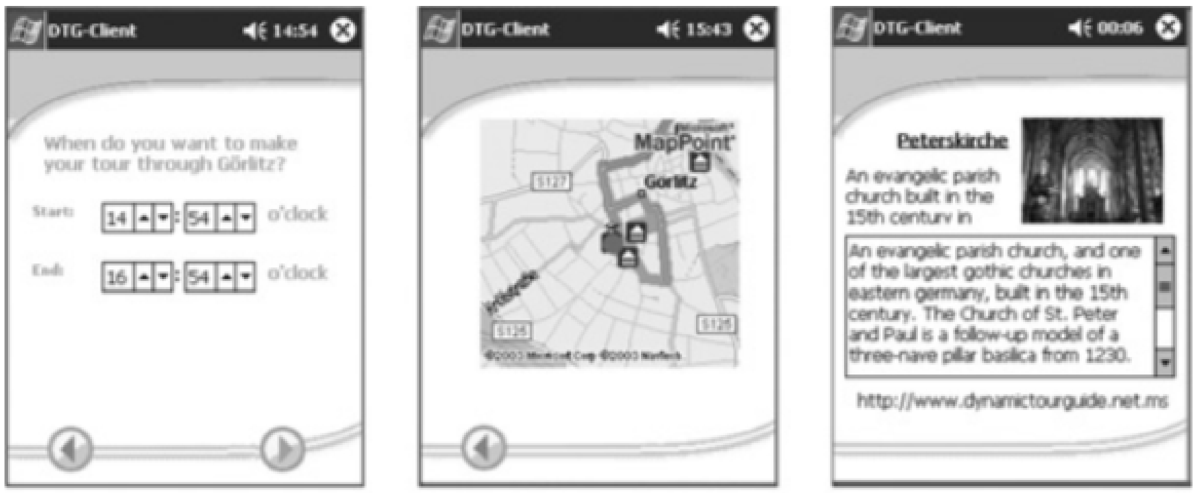
The Dynamic Tour Guide (DTG) [52] has been extensively reviewed by many authors and is one of the most discussed research prototypes in the field of MTGs. It was one of the first MTGs that calculated tours in real time by considering available context information like personal interests and location based services.

The authors found an identical problem of overcrowding in the city of Goerlitz, Germany. Popular attractions were becoming overcrowded and others had rarely any visitors. They suspect that tourists ended up following the most popular paths simply due to lack of information, and not because it was their preference. The objective of the DTG is to have a mobile guide that understands the individual interests and timeframe, knows the local situation and suggests a personal tour to each tourist.

The Dynamic Tour Guide (DTG) is a mobile application which, in its "Planner mode", is able to create personalised tours based on tourists' generic preferences and context based constraints such as tour duration, starting time and location. The DTG also offers an "Explorer mode" that computes no tour plan but provides context based information whenever requested. Tourists can follow their own position on a map and see a list of available attractions in their close vicinity. Either manually selecting or staying at a certain sight will trigger context-driven interpretation. Points of interest have to be manually added by a content provider. These POIs contains address, interest coverage, descriptive text modules, various pictures, audio files and general information.

Screenshots from the article show that although it may not have the most modern-looking user interface, it is certainly functional. The authors have shown concern for usability issues when designing the interface, mentioning small screen real-estate and issues when elicit the preferences of a tourist regarding the selection of attractions for a tour. Menus, navigation and layout are well thought out (Figure 2.6).

In August 2006 a field trial was conducted in Görlitz, Germany to examine the behaviour of tourists resulting from the use of the mobile Recommender System (RS) DTG. The main objective was to study the differences of the Planner and Explorer modes in terms of their impact



When do you want to make your tour through Görlitz?

Start: 14 : 54 o'clock

End: 16 : 54 o'clock

Peterskirche

An evangelic parish church built in the 15th century in

An evangelic parish church, and one of the largest gothic churches in eastern germany, built in the 15th century. The Church of St. Peter and Paul is a follow-up model of a three-nave pillar basilica from 1230.

http://www.dynamictourguide.net.ms

Figure 2.6: Application layout [51]

on tourist behaviours. They used a limited amount of specific mobile devices during this study, some operating on explorer and others on planner mode. The software was equipped with an instrumentation mechanism in the background to monitor the usage patterns. The authors identify some limitations in this methodology, namely that the data obtained from monitoring users gave no insight regarding their intentions or reasons, or whether their behaviour was influenced by external causes.

Table 2.8: Kramer et al. (2007) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Unknown	Moderate Audience (Single Device Type)	Intermediate Personalization	Average Concern	Interactions Among Contents	Empirical Evaluation

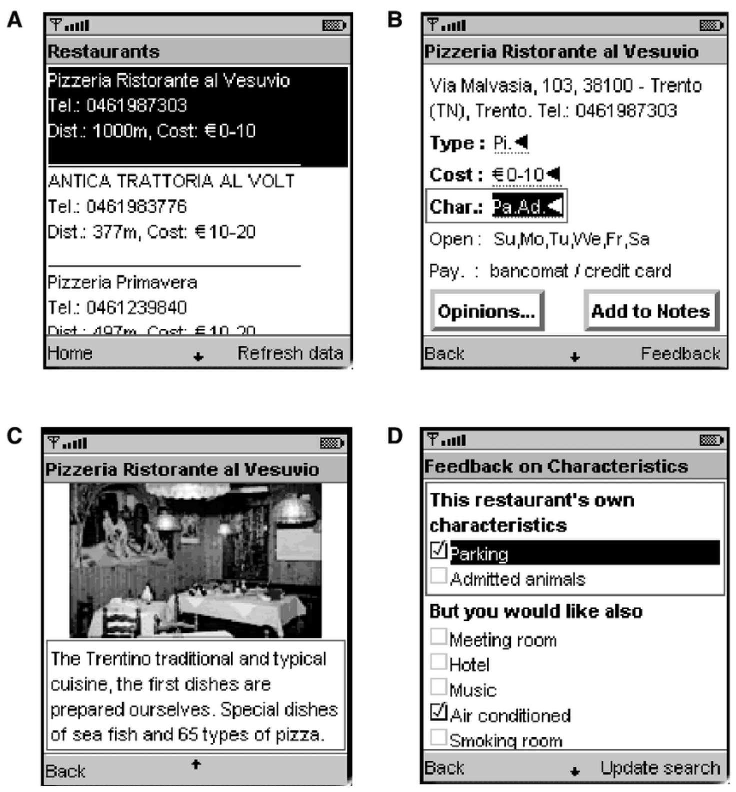
2.3.3.5 Ricci et al., *"A conversational recommender system for on-the-move travellers"*, 2006

MobyRek was a MTG prototype developed specifically for the on-tour stage of travelling, aiming that, at this stage of travel, tourists would use the system *"to search for desired travel products, or to complement their pre-travel plan"*.

For the pre-travel and post-travel stages, there is a second system named NutKing. It is a web recommender system that *"helps users in selecting one or more destinations to visit and then adding additional products related to the selected destinations (accommodations, activities, events)"*. Even though NutKing had already been successfully validated, it is not suitable for tourists on the move, where the time span and cognitive effort needs to be minimized.

MobyRek and NutKing were built to work together. MobyRek makes use of NutKing's pre-travel plan to extract a set of preferences from the product choices made by users. This means that it works under the constraint that the complementary products recommended should conform to those that have already been selected in the pre-travel stage. The way the system tunes these recommendations is quite different from other MTGs. In an attempt to minimize user effort, it involves the user in a dialogue metaphor by allowing them to criticize each option (e.g. too expensive). The system automatically updates the user's preferences model with either mandatory restrictions or optional preference (e.g. preference for paying with credit card).

This article focuses on how the system makes recommendations (which is considered out of the scope of this rapid review), instead of expanding on the details of the overall system architecture or the app itself. The user interface is composed of four screens (Fig. 2.7): (1) a screen consisting of a list of recommendations, (2) a screen with the attributes of the POI, (3) a screen with the description and customers' opinions of the POI and (4) a "travel notes" screen consisting of all the saved points of interest.


A

Restaurants

Pizzeria Ristorante al Vesuvio
Tel.: 0461987303
Dist.: 1000m, Cost: €0-10

ANTICA TRATTORIA AL VOLT
Tel.: 0461983776
Dist.: 377m, Cost: €10-20

Pizzeria Primavera
Tel.: 0461239840
Dist.: 497m, Cost: €10-20

Home Refresh data

B

Pizzeria Ristorante al Vesuvio

Via Malvasia, 103, 38100 - Trento (TN), Trento. Tel.: 0461987303
Type : Pi.
Cost : €0-10
Char.: Pa.Ad.
Open : Su,Mo,Tu,We,Fr,Sa
Pay. : bancomat / credit card

Opinions... Add to Notes

Back Feedback

C

Pizzeria Ristorante al Vesuvio

The Trentino traditional and typical cuisine, the first dishes are prepared ourselves. Special dishes of sea fish and 65 types of pizza.

Back

D

Feedback on Characteristics

This restaurant's own characteristics
☑ Parking
☐ Admitted animals

But you would like also
☐ Meeting room
☐ Hotel
☐ Music
☑ Air conditioned
☐ Smoking room

Back Update search

Figure 2.7: User interface [73]

The user interface does not seem to provide a map that tourists can use to guide themselves, which is an important aspect of MTGs, resulting in a bad user experience. However, the authors did mention efforts (e.g. reducing cognitive efforts, designing for smaller screens) to improve the usability of the app.

The evaluation process involved six test users. The participants were first trained by being introduced to the user interface and the system functionality. Then, the users tested the system by using it in a real life scenario. Finally, the test users were asked to complete a usability survey.

Table 2.9: Ricci et al. (2006) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Unknown	Unknown	Intermediate Personalization	Average Concern	No Social Features	Empirical Evaluation

2.3.3.6 Cena et al., *"Integrating Heterogeneous Adaptation Techniques to Build a Flexible and Usable Mobile Tourist Guide"*, 2006

Developed by the university of Turin, UbiquiTO was a MTG prototype that adapts to the user, the device and the context of interaction. While this prototype focuses on travelling workers, the authors envision a system that will serve any type of tourist. In this very thorough article, the authors detail their system architecture and the strategies adopted to make the system adaptive.

The system architecture (Fig. 2.8) of this prototype is fairly complex. The core of the system are the two personalization agents: the Recommender and the Presentation Adapter. Additionally, there are three support modules: the Interaction Manager, the UI Generator and the Watcher. Finally, there are three knowledge bases (KB) and two databases (DB) that support the other components. All of these components are located on the server side, in order to *"support flexibility and device independence"*, meaning that the system can function regardless of what device the user is using it from.

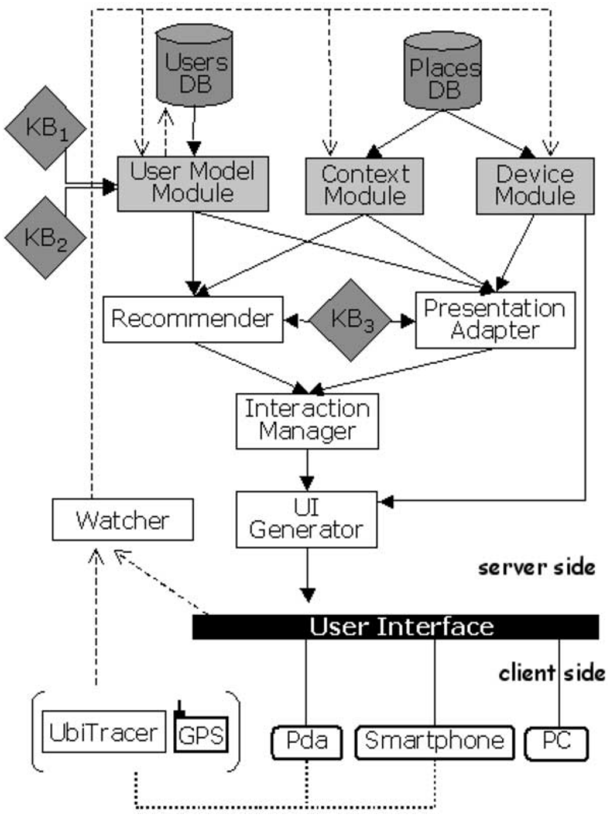


Figure 2.8: System architecture [18]

From the two personalization agents, the *Recommender* tailors the suggested items (points of interest) to the users preferences and to the context (location). The *Presentation Adapter* molds the presentation (e.g., interface layout) according to user preferences, the context and the device's characteristics.

Between the support modules, the *Interaction Manager* is responsible for creating XML objects that correspond to the (personalized) content of the interface page. The *UI Generator* serves to apply styling (XSL stylesheets) to the XML objects into the final (X)HTML pages that produce the UI, according to the device and context features. The *Watcher* tracks information from the user either implicitly, by observing user's interaction with the system (e.g., GPS, page views), and explicitly (e.g., from the initial registration form). Additionally, there is an optional forth module - the *UbiTracer* - that is a client side location agent, which can be downloaded on GPS-enabled devices, that sends the user's location to the Watcher agent. On devices without GPS, users alternatively provide their current location by selecting it on a map.

Two separate versions of the user interface were developed, one for desktop and another for smartphones/PDAs. The mobile version (Fig. 2.9) has been simplified in order to account for the smaller screen real estate. These adaptive interfaces are created using Java Servlet technology that generates HTML pages dynamically.

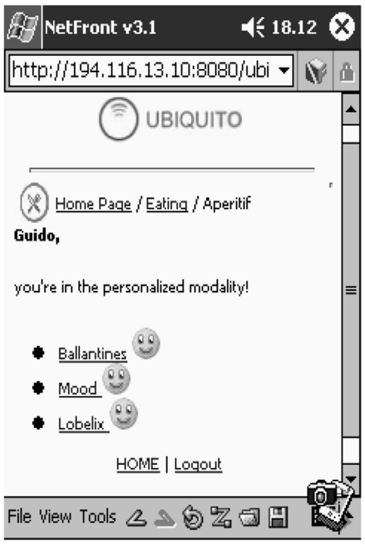

NetFront v3.1
18.12
http://194.116.13.10:8080/ubi
UBIQUITO
Home Page / Eating / Aperitif
Guido,
you're in the personalized modality!
Ballantines
Mood
Lobelix
HOME | Logout
File View Tools

Figure 2.9: App interface [18]

Due to the initial design decision to make a system for every type of tourist, the authors managed to create a system that truly adapts to the end user and his choice of device. The system uses explicit information (e.g., demographic data, categories of interest, visit constraints) and implicit information (e.g., propensity to spend) to create the best possible recommendations.

The user interface not only adapts to the device's characteristics, but also to context conditions (e.g., bigger font size when the user is on the move or according to the time of day), leading to improved usability.

The authors conducted multiple tests to different parts of the system. The usability of the user interface was evaluated in a set of heuristics evaluations performed by three usability/domain experts.

Table 2.10: Cena et al. (2006) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Web	Mass Audience (Multiple Device Type)	Advanced Personalization	High Concern	No Social Features	Evaluation by Experts

2.3.3.7 Cunningham et al., *"A Context-Aware e-Tourism Application Enabling Collaboration and Knowledge Exchange among Tourists"*, 2006

Kamer eTourism Application is a MTG prototype that is part of the Italian project *Kamer (Knowledge Management in Ambient Intelligence: technologies enabling the innovative development of Emilia Romagna)* that focuses on Ambient Intelligence (AmI) and context-awareness supporting knowledge exchange and collaboration among individuals in mobility.

The authors see *"tourist communities as groups of tourists with common interests, where communication should be enabled and knowledge should be formalised and exchanged for the benefit of all members"*. They applied this philosophy into the app, treating tourists as a *"source of valuable information, which can be useful for other tourists and for tourist service providers as well"* by designing two social driven features: (1) an instant messaging service where tourists can communicate between them and (2) a service provider reputation manager that lets tourists rate and see other people's ratings of services.

This app was designed to run on a specific PDA via a Web browser. Besides the aforementioned services it also provides users with location-based content delivery, which means that users will proactively receive notifications with descriptions and multimedia content related to the nearby POIs. The app does a very good job of being context aware, adapting to *"user preferences, current activity, closeness to points of interest and physical proximity to other tourists, especially those with similar preferences and itineraries"*. This means that, for instance, if it's around noon, the system will notify the tourist of nearby places to eat.

The system architecture is divided into three layers. The *Context Consumer* layer (in this case, the app) receives processed context data to dynamically serve the user. The *Context Management Middleware* receives raw context data from multiple providers, processes it into higher-level context data and delivers it to the consumers. Finally, the *Context Provider* layer which is composed of individual distributed sensors that gather context data (e.g., temperature, light).

The *Context Management Middleware* serves as a separation of the app logic and the task of context management, improving reusability and extensibility of context management modules, and simplifying the design and development of the app itself. This layer is comprised of said *Context Management* modules, which are J2EE programs dealing with *"context data acquisition from context providers, data representation, reasoning and delivery to context-aware applications, managing queries from applications or notifying them when context has changed"*.

This prototype specializes in capitalizing the social aspect of tourism to enrich the contents of the app, while also allowing users to interact with each other to improve their experiences.

The authors have shared some of their issues and concerns about usability. They choose a specific PDA due to it's higher screen resolution. They allow users to configure some of the information flow to their preference and they made some design decisions that better fit the mobile experience.

By the time of publication, evaluation tests had not yet been conducted, but there were two planned stages. The first stage was meant to test the usability of the app in group sessions of Italian students in the centre of Piacenza. The second staged would involve older people from different backgrounds, with varying technological skills.

Table 2.11: Cunningham et al. (2006) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Web	Limited Audience (Specific Device Type)	Advanced Personalization	Average Concern	Hybrid Interactions	N/A

2.3.3.8 Hopken et al., *"Context-based Adaptation of Mobile Applications in Tourism"*, 2010

This article presents a framework for mobile applications in tourism that enables flexible implementations of adaptive, context-aware tourism applications. The authors believe that user acceptance of apps strongly depends on the application adaptation to concrete usage context.

The proposed framework focuses on user interface adaptation, content adaptation (recommendation), and interaction modality adaptation. It intends to increase information quality, the efficiency of the human–computer interaction, and the overall usability.

The framework has been instantiated and evaluated in the form of two app prototypes for two different application scenarios, a city guide for the city of Innsbruck *(Innsbruck.mobile)* and a skiing guide for the ski resort DolomitiSuperski *(DolomitiSuperski.mobi)*.

Both the *Innsbruck.mobile* and the *DolomitiSuperski.mobi* apps were created according to the authors' proposed framework. They follow the architectural structured represented by Figure 2.10.

The *User Model Service* constitutes a central user modeling component that manages users' profiles and their interaction histories. The data for the user model is derived from explicit user input, as well as user interactions within the search and browse, recommendation, or PUSH service. In the *Innsbruck.mobile* version the user also interacts directly with the app in order to input his time restrictions and personal tastes.

There is a strict separation of the application content and its concrete representation and application behavior (based on XML and XSLT). The content and general structure of the application is provided in a neutral format, independent of the concrete user interface, and the presentation and behavior of the application is dynamically generated based on transformation instructions. This enables a comprehensive and flexible adaptation of an application's user interface. Due to this mechanism, the apps are compatible with a multitude of platforms and devices from smartphones to PC's.

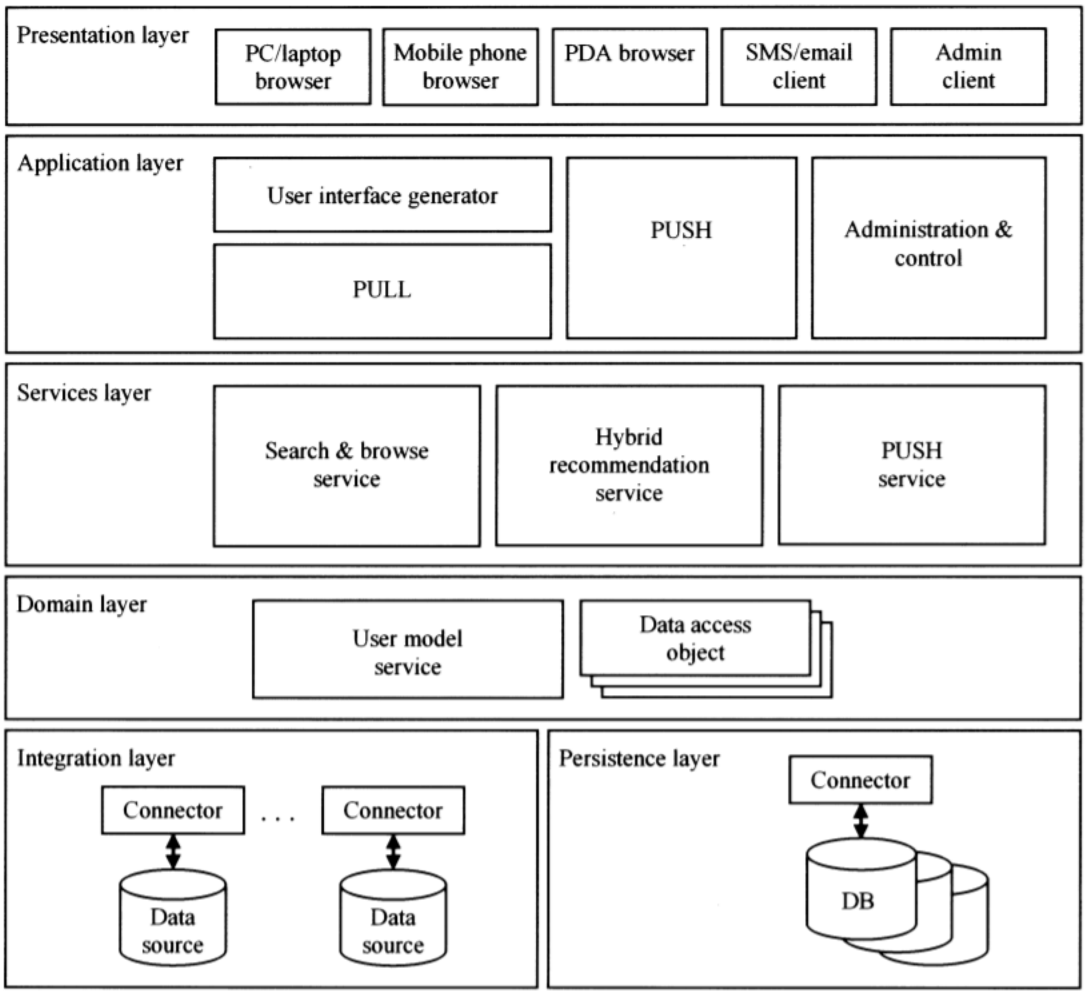


Figure 2.10: Architectural structure [41]

DolomitiSuperski.mobi was implemented during the winter season 2007/08 and 2008/09. During the winter season 2008/09 a total number of 20,240 unique visits were registered. The authors monitored the usage patterns of their users. Additionally a laboratory-based evaluation took place in April, 2008 were they assessed the service quality of the app.

Innsbruck.mobile was evaluated in June, 2008 during the European Football Championship EURO2008™. It was the official mobile guide of the host city Innsbruck. Much like *DolomitiSuperski.mobi*, usage patterns were also monitored.

Table 2.12: Höpken et al. (2010) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Web	Mass Audience (Multiple Device Type)	Intermediate Personalization	Average Concern	No Social Features	Empirical Evaluation

2.3.3.9 Kenteris et al., *"An innovative mobile electronic tourist guide application"*, 2009

This paper presents the design and implementation issues of a MTG research prototype. Namely, it enables the creation of portable tourist applications with rich content that matches user preferences. The prototype implemented by the authors is called *myMytileneCity* and it was made for tourists planning to visit the city of Mytilene (Lesvos Island, Greece). The main design and implementation objectives were the following:

- The system should allow the generation of tourist guide applications over the Internet, which can later be executed on non-networked mobile terminals;
- The personalized tourist applications should execute on any mobile device (mobile phone, smart phone, PDA, palmtop, etc.) independently of their hardware platform or operating system;
- The applications should enable the dynamic delivery of tourist content with minimal user intervention, given that the user has expressed interest on the correspond- ing content type.

The app design was heavily influenced by these previously mentioned objectives. It includes a database-enabled tourist web site wherein, on a first stage, users choose the content that interests them (lodging, sightseeing, entertainment, etc). They may download these personalized applications (optimized for their specific device's model) either directly to their mobile device, or first to a PC and then to a mobile terminal (through infrared or Bluetooth). Constant internet access is not required, as the applications execute in standalone mode and may be updated when the user returns online. New tourism content can be added (or updated) by administrators, and it will be pushed to the mobile device. This prototype is particularly careful with only updating content that the user is interested, in order to avoid network costs.

In order to execute on any mobile device, the app was developed using the Java 2 Micro Edition[1]. J2ME configurations define the minimum features of a Java Virtual Machine and a minimum set of libraries for a "horizontal" family of devices. Profiles are implemented on the top of configurations (a configuration is the base for one or more profiles). Typically, a profile includes libraries specialized in the unique characteristics of a particular class of devices. Java applications (like *myMytileneCity*) developed for the only pair of configuration/profile available for mobile phones at the time, were called MIDlets. MIDlets were ideal for MTGs that wanted platform independence and the power of the Java language.

The user can interact with the web site in order to input his interests into the system and get a more personalized experience. However, once the application is created and downloaded, user profile is locked and the system will no longer adapt to any changes in user preferences. The web site offers advanced search facilities for tourist content that match specific criteria and allow users (through a specialized forum) to share their opinions and experiences with respect to destinations they have already visited, make recommendations to other users, etc. Post-visit reports and voting for best-to-see attractions features are also available.

[1]https://objectcomputing.com/resources/publications/sett/august-2002-the-next-generation-of-java-2-micro-edition

Although the application design looks outdated (it is an old app, after all), usability was taken into consideration according to mobile device's specific properties: small screens, limited input and mobility.

Usability tests have been performed both in experimental environment and with field studies. During experimental studies, participants have been asked to accomplish specific tasks using the mobile application. For the field studies the participants' personal mobile phones have been utilized as test tools (after downloading and installing the *myMytileneCity* guide application on them). During usability tests, the evaluators recorded several quantitative usability attributes measurements for each individual participant (i.e. effectiveness, efficiency and learnability). Additionally, qualitative usability attributes have been evaluated through interviewing the participants (i.e user satisfaction, simplicity, comprehensibility, perceived usefulness and system adaptability).

Table 2.13: Kenteris et al. (2009) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Hybrid	Moderate Audience (Single Device Type)	Basic Personalization	High Concern	No Social Features	Empirical Evaluation

2.3.3.10 Lim et al., *"Feel the difference: A guide with attitude!"*, 2007; Lim et al., *"Narrative construction in a mobile tour guide"*, 2007

The *Affective Guide* presents a different take on what a MTG should provide. Instead of a traditional guide that displays facts and descriptions of attractions, this MTG emulates a real guide's behaviour by presenting stories based on the user's interests. It has its own ideological perspectives and is capable of expressing its viewpoint about events, through improvisational storytelling.

A prototype was built and used in the Heriot-Watt Edinburgh campus. The "Los Alamos" site of the Manhattan Project was chosen as the narrative domain. *"The Manhattan Project was chosen because it contained many characters with different personalities and ideologies that can be used as Affective Guides"*. While this MTG may not be very similar to the other ones reviewed, which makes comparing it unreasonable, it is included due to it being a interesting take on the MTG paradigm.

This MTG was built for PDAs that support text-to-speech and GPS. All of the processing and information storage is done remotely on a server and accessed on demand through wireless communication.

The first interaction with the system starts with the guide introducing himself and engaging in a conversation with the user, which the authors call an "ice-breaking session", where the guide extracts information about the user's name and interests. From there the system chooses the POIs and the quickest route to them. Upon arriving at a POI, the system starts the storytelling process. During this interaction, the user is able to express the level of interest in

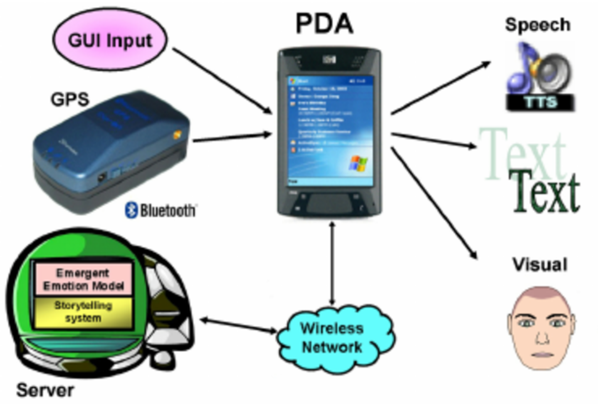


Figure 2.11: System architecture [55]

the guide's stories, as well as the level of agreement with the guide's argument, all through a rating bar metaphor. The system takes this input and reacts by adjusting the emotional state and the extensiveness of the stories.

The authors tested different versions of the *Affective Guide*: an emotional variant (with an animated character), a non-emotional variant, and a random emotions variant. These versions were tested with real users, that interacted with the system and were then asked to answer two sets of questionnaires, one about their subjective opinion on the system and another with questions based on the stories the participants listened to during the tour. The authors wanted to measure the effectiveness of the system.

The development approach was not mentioned in the articles. Also it seems that there are no social features in this MTG. Unfortunately, no descriptions or figures of the user interface were given.

Table 2.14: Lim et al. (2007) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Unknown	Moderate Audience (Single Device Type)	Intermediate Personalization	Unknown	No Social Features	Empirical Evaluation

2.3.3.11 Garcia et al., *"Intelligent Routing System for a Personalised Electronic Tourist Guide"*, 2009

The main focus of this paper is actually not a MTG, but rather an intelligent routing system that includes public transportation, adapts to real-life circumstances, such as rush-hours and updates in real-time when necessary. Nonetheless, a functional prototype MTG that takes advantage of said routing capabilities has also been developed and described here.

It was created for tourists in the city of San Sebastian, Spain. A city *"best visited by combining public transportation with short walks"*, making it ideal for this intelligent routing system. Inspired by the *Explorer Mode* of the DTG [51, 52], this system also allows users to free browse the city by accessing information about attractions and public transportation. Users can create their "tourist profile" with restrictions, attractions to be visited, and the location of their accommodation. This was admittedly kept simple, as it was not the main focus of the authors. As previously mentioned, tourists create routes that will be updated when new events are detected, after confirmation by the tourist. All of this is complemented by a map-based interface (Fig. 2.12) that helps with the visualization and exploration of the route and attractions.

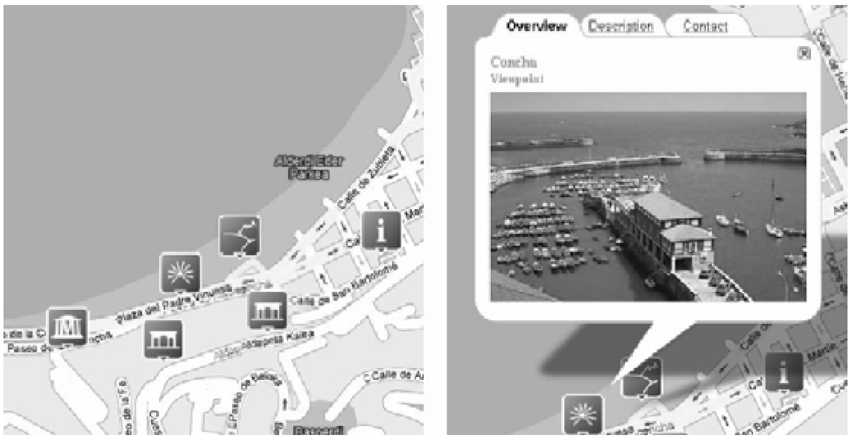


Figure 2.12: App interface [36]

The system architecture is composed of three main elements: the client, the agents and the server. The client is a web application that runs on browsers and is developed using *Google Web Toolkit* that transforms native Java code into HTML and JavaScript, while also guaranteeing compatibility with new browser versions. *Google Gears* is also used to access the location data of the client. Agents monitor changes in relevant context areas, such as weather or traffic, and report them to the system. Each agent is responsible for monitoring one specific element. The server consists of an *Apache* web server and a *Mysql* database manager. It receives information from the client and the agents to provide personalised tourist routes in real time. Information about attractions is stored in said database and relies on local tourist organizations to keep it updated, "*which is a crucial factor to create high quality routes*".

With the main focus of this paper being the system's routing algorithm and not the MTG itself, mentions of usability are hard to find. Nonetheless, from descriptions of the system, one can assume that the app is very dependent on internet connection to communicate with the server. One positive aspect is the fact that the app is built to be very aware of the context surrounding travelling, supported by the agents, resulting in real-time updates of the route.

While a real test of the prototype for tourists in the city of San Sebastian was planned for the summer, at the time of writing the authors had only carried out laboratory tests validating the routing algorithm.

Table 2.15: Garcia et al. (2009) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Web	Mass Audience (Multiple Device Types)	Advanced Personalization	Bad Usability	No Social Features	N/A

2.3.3.12 Noguera et al., *"A mobile 3D-GIS hybrid Recommender System for tourism"*, 2012

In this comprehensive article the authors present their design and implementation of a MTG prototype for restaurants in the province of Jaén in southern Spain, as an extension of a previously built RS named *REJA*.

The proposed app wants to overcome the challenges of MTGs by upgrading *REJA* in the following aspects: (1) moving from a computer web platform to mobile, so that *"users may use the system wherever they like"*; (2) becoming aware of the location of the user to provide better recommendations; (3) using a 3D-interface with innovative features like 3D geovisualization to improve the usability of the app.

The *3D Geographic Information System (GIS)* interface gives the user an immersive experience, using the device's GPS and compass to mimic the user's perspective into a virtual overview of the city with an accurate landscape (Fig. 2.13). The map is also interactive so that users can control the view to locate points of interest using the touchscreen. The main objectives of this design decision are to make the app easy to use and driver-friendly, and also to *"provide a spatial and intuitive representation of the user's surroundings together with the location of the recommended items"*. Unfortunately, retaining good performance requires a trade-off in the form of depending on internet access. Only a small subset of the map is stored in memory, and the rest is downloaded from a specific GIS server when necessary, as the tourist moves across the environment.

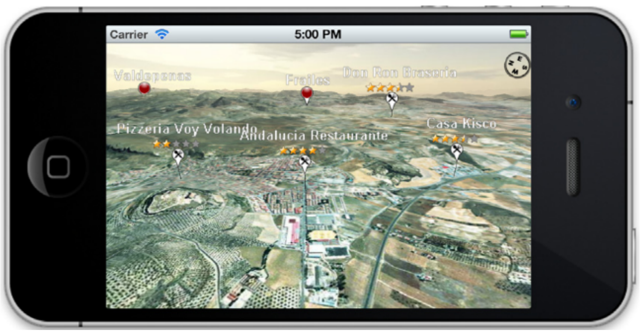


Figure 2.13: 3D interface [61]

The app architecture is fairly clean (Fig. 2.14). The app communicates exclusively with the GIS server with requests for recommendations or map data on demand. The GIS server is responsible for storing the entire terrain data set and delivering chunks of it to the app, as well as recommendations retrieved from the recommender server. The recommender server simply provides real time recommendations of restaurants according to the user's location, and some optional basic settings.

The app was written as a plain C++ native program, so it supports a broad range of devices: iOS (iPhone, iPad), Symbian OS, Windows Mobile, Win32 and GNU/Linux.

The level of personalization is fairly limited on some aspects and advanced on others. Initially, users have to create a profile over the web, where they specify their preferences, which the RS use to improve the recommendation quality. Additionally, the RS will keep fine tuning the recommendations based on the ratings the user gives. However, the only known setting the user can change dynamically over the app is the distance he is willing to travel to the POI.

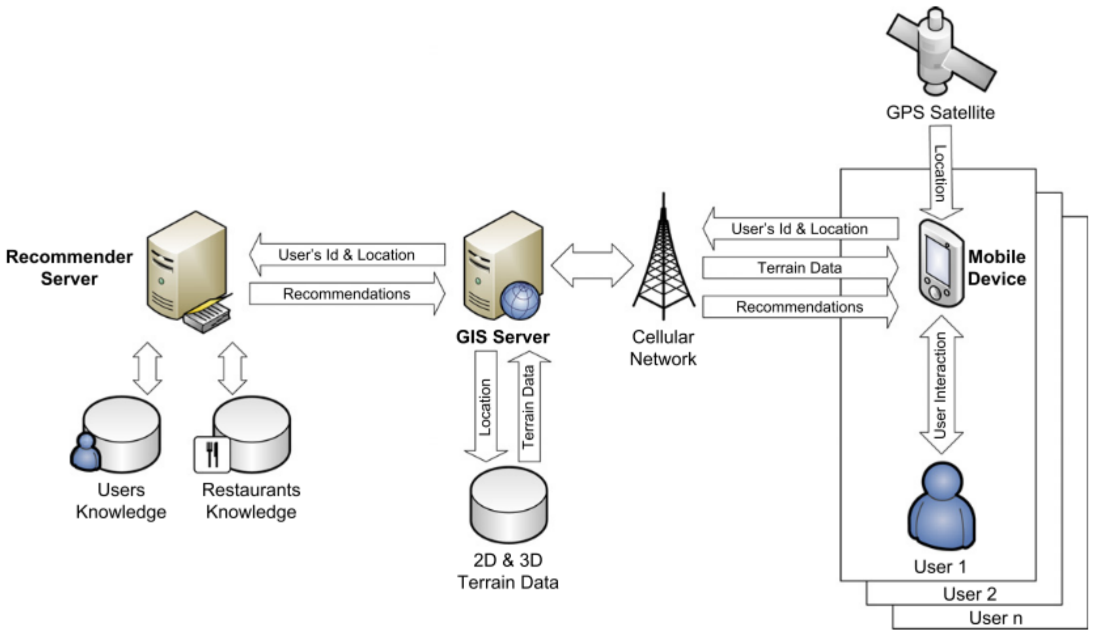


Figure 2.14: System architecture diagram [61]

To evaluate the system, the authors conducted user studies with the main goal of answering two questions: *"Are 3D maps a good way to provide location-aware recommendations on mobile devices?"* and *"Does location-awareness increase the quality and usefulness of the system?"*. Consequently, they also built 2 variants of the prototype. One with a 2D *Google Maps* alternative and another without location aware recommendations.

The first user study evaluated the usability of the app by comparing the 3D interface with the 2D variant. Users were asked to complete a set of tasks and then fill out a questionnaire involving typical usability questions. For the second study, users were asked to complete the same tasks with both variants of the recommender system and then fill out a questionnaire about the quality of the recommendations. The validity of the 3D map was confirmed as users state that both map variants were easy to use, clear to understand and have a good practical use. Overall, this article presents one of the most matured MTG prototypes.

Table 2.16: Noguera et al. (2012) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Native	Mass Audience (Multiple Device Types)	Intermediate Personalization	Good Usability	Interactions Among Contents	Empirical Evaluation

2.3.3.13 Wijesuriya et al., *"Interactive Mobile Based Tour Guide"*, 2013

This MTG was developed for the Royal Botanical Gardens in Peradeniya, Sri Lanka, by the Sri Lanka Institute of Information Technology (SLIIT). Much like Portugal, Sri Lanka has a thriving tourism sector that is a leading driver of growth in the economy. The authors remark that the local tourism industry has a low adoption for new technologies, and that can serve as an obstacle for tourists.

To improve this issue, the authors created an app to: *"help the tourists to travel on their own and take full advantage of the visit without missing the main attractions"*, thereby replacing the traditional tour guide booklet.

The app is supported by a web server that stores tour guide information and delivers it when needed. The app uses the user's location and selections to obtain relevant tour guide information from the web server (Fig. 2.15). This architecture has a single point of update that makes it easy to serve up to date information to the user, while also keeping the app size low.

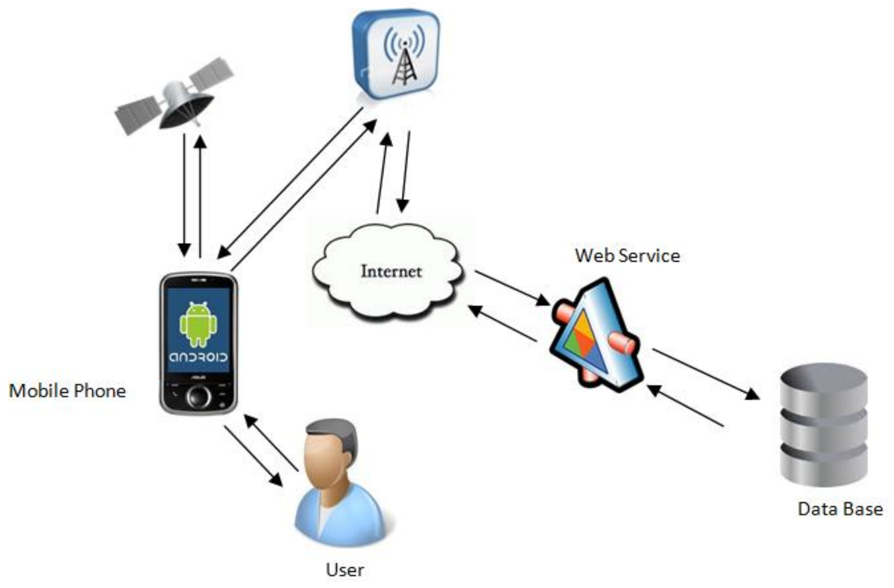


Figure 2.15: System architecture diagram [90]

The app itself is available for Android devices only, and it has two main features: *Map View* and *Camera View*.

Map View makes use of *Google Maps'* API to suggest paths given a start and end location, to provide information about points of interest with audio descriptions (stored as text in the web server and converted to audio in real time), and to offer virtual tours of different locations capable of navigating through processed images that give a realistic three dimensional view of the location.

Camera View showcases an interesting application of *Augmented Reality (AR)* in assisting the traveler at the tourist sight. It utilizes both the device's GPS and built-in compass to detect when a user is near a POI and guides them with on-screen arrows indicating the correct way.

The main focus of this project was exploring location based augmented reality to enhance the tour guide system of Sri Lanka. Accordingly, a native Android application was built as an experiment. Unfortunately, other important aspects like social features, personalization and usability were not given much care. Additionally, no evaluation methods were used to validate the MTG.

Table 2.17: Wijesuriya et al. (2013) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Native	Moderate Audience (Single Device Types)	Basic Personalization	Bad Usability	No Social Features	N/A

2.3.3.14 Yang et al., *"iTravel: A recommender system in mobile peer-to-peer environment"*, 2013

iTravel is a recommender system and MTG prototype that relies on mobile peer-to-peer technology and other tourists' ratings to recommend attractions. Mobile applications that interact with a remote server can be a struggle for tourists on the move, due to intermittent internet connection and high roaming charges. The proposed MTG intends to: *"provide mobile users effortless and inexpensive means for exchanging ratings about attractions and makings effective attraction recommendation."*. The result is a completely decentralized system that requires no main servers for the app to communicate with

Additionally, the authors want to *"exploit other tourists' ratings on their visited attractions for recommending attractions to a mobile user"*, as tourists share common interests and usually rely on each other for help and suggestions. This system was expected to improve as the user is on the move, trading information with other tourists making the recommendations more accurate.

The app was implemented using Java 2 Micro Edition and its architecture (Fig. 2.16) consists of 5 main components: (1) the interface, (2) the *Location Manager*, (3) the *Communication Manager*, (4) the *Rating Data Manager* and (5) the *Recommendation Manager*.

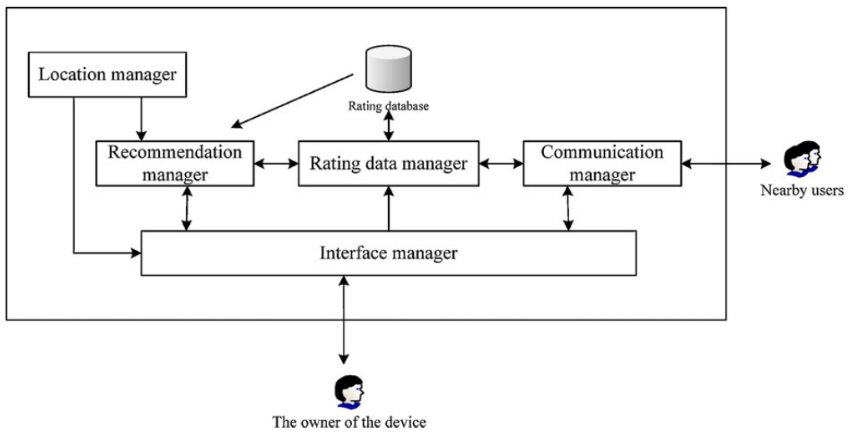


Figure 2.16: iTravel system architecture [96]

The interface (Fig 2.17) shows users where they are on the map, where nearby tourists are (green markers) and recommended attractions (blue markers). Additionally, users can also see a text description of the attractions and rate them on a scale from one to ten.

The *Location Manager* is simply responsible for detecting the position of users. The *Communication Manager* handles the information exchange between users. The most notable peer-to-peer communication technologies widely available in mobile devices are Bluetooth and Wi-Fi.

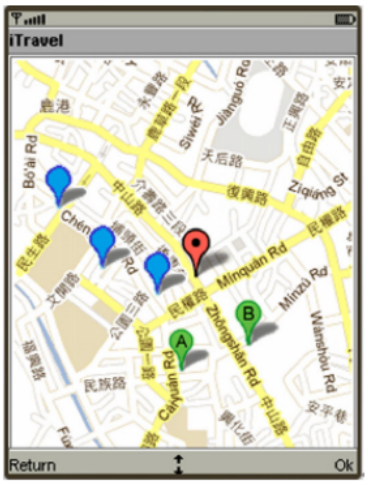


Figure 2.17: iTravel interface (home screen) [96]

Both technologies allow for short-range wireless broadband communication. The authors compared both options and decided on Bluetooth due to less power consumption, accepting the downside of a lower throughput and lesser range.

The *Rating Data Manager* contains a rating database that stores the user's personal ratings but also the ratings of other users. It determines the sets of ratings to send and receive when exchanging information with another iTravel system. The article contains a lot of detail about different data exchange approaches that were purposefully omitted in this review, as they are not considered relevant. Finally, the recommendation manager recommends unvisited attractions to the user, based on the rating lists stored in the rating database.

The proposed system provides an interesting solution to the "intermittent connectivity" problem. It focuses on the social aspect of tourism to improve the recommendation accuracy. User personalization was not the focus of this MTG so the only way users can directly adapt the recommendation system is by rating attractions.

The performance of the system was evaluated with simulations and the usability of the app was evaluated using empirical user studies. The prototype was deployed in Lukang, Central Taiwan, which has a notable historical site and cultural scene. 95 volunteers travelled to Lukang and used the iTravel system to explore the city. The participants were asked to fill a survey form regarding the usability at the end. Their rating, browsing and visiting behaviours were also logged.

Table 2.18: Yang et al. (2013) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Hybrid	Moderate Audience (Single Device Types)	Basic Personalization	Good Usability	No Social Features	Empirical Evaluation

2.3.3.15 Zhao et al., *"A Mobile Service Platform for Xinjiang Tourism"*, 2015

In this article the authors constructed a mobile service platform for Xinjiang tourism. According to the authors, Xinjiang has a rich cultural heritage with the right conditions to be a tourism triumph, but it is still behind when compared with other domestic historical and cultural cities. The proposed MTG intends to elevate the tourism services of Xinjiang to the standards of international and domestic tourism, by meeting the requirements of mobility, timeliness and interactivity of the travel information for different tourists.

Below (Fig. 2.18) are presented some of the screens of the application prototype. The app's main screen shows a map of the city with the users location, which is obtained through GPS. Instead of using the more traditional map provider *Google Maps*, which has a reputation for having poor fidelity in China, the prototype uses *Baidu* map information, to *"improve the accuracy of the data, so that the provided map information is more reliable and available"*.

Through this map interface, the users can search nearby hotels, attractions and get the corresponding routes either by foot or car. By selecting a point of interest, the user navigates to a screen showing the complete details of said POI. Additionally, if the user is logged in, he will also have the option to make a reservation and to comment or review the POI. According to the authors, during tours the application will play relevant media (video and audio) and, depending on the POI, the app can also dynamically display the relevant artifacts in 3D, and the historical and cultural allusions, so that visitors can see them from different angles without interference with each other in the process.

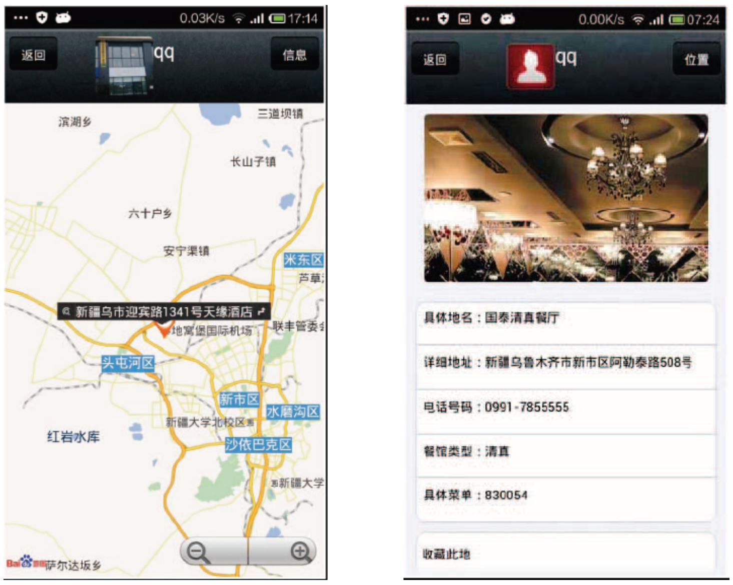

具体地名：国泰清真餐厅
详细地址：新疆乌鲁木齐市新市区阿勒泰路508号
电话号码：0991-7855555
餐馆类型：清真
具体菜单：830054

Figure 2.18: User interface [98]

The system's architecture that supports application is not clearly presented in the article. Nonetheless, the authors mention a *"User System"* and an *"Attractions System"*. The *"User System"* allows users to create a profile, insert and manage their personal information, which can then be used in order to find other tourists, friend them, and chat with them through the app. The *"Attractions System"* contains an point of interest database of the city, with their location and details, which is then shown to the user on the app. Additionally, through this system, the user can make reservations to these points of interest, when suited, and also comment and review them. The communication between the client and the server is done by using the JSON data format.

The application was developed for *Android* devices only, using the native *Android* SDK and the *Eclipse* development environment. The authors make no mention of usability and they mention that the app is very network dependent. Finally, no evaluation tests were conducted.

Table 2.19: Zhao et al. (2015) classification

Development Approach	Availability	Personalization	Usability	Social Features	Evaluation
Native	Moderate Audience (Single Device Type)	Basic Personalization	No Concern	Hybrid Interactions	N/A

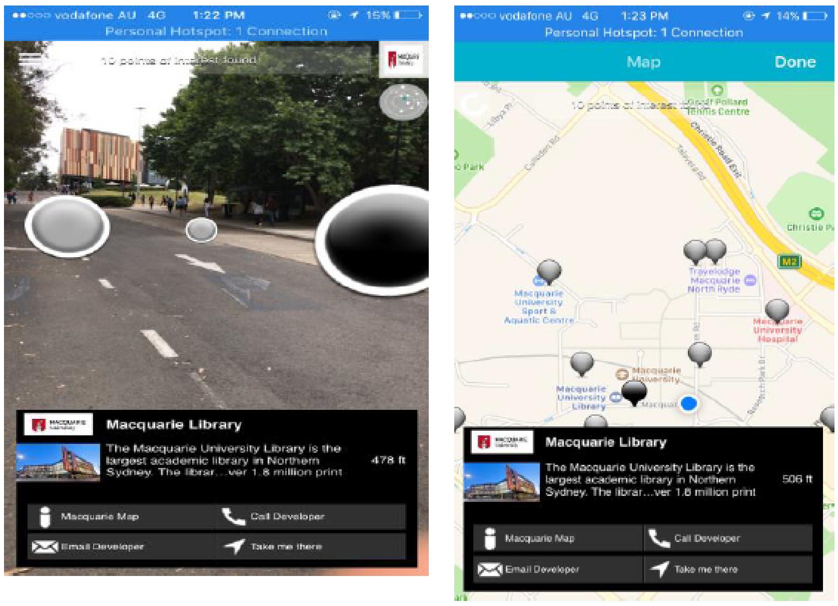
2.3.3.16 Alqahtani et al., *"iMAP-CampUS: Developing an Intelligent Mobile Augmented Reality Program on Campus as a Ubiquitous System"*, 2017

This article defers from others included in this RR in the sense that it is not made for the tourism environment of a specific city, but instead for the Macquarie University based in Sydney, Australia. In this article, the authors share the details of their app, *iMAP-campUS*, developed to help students navigate through the university, which has a particularly long and complex layout.

This application adopted an innovative approach of integrating augmented reality technology aspects into the app. Using this technology it is possible to help students navigate by *"locating places of interest close to them by moving the camera of the device in all probable directions to overlay information of places around them"*.

The application's graphical user interface (Fig. 2.19) is divided into multiple screens that have distinct functionalities. The *Reality View* is the main view of the application, and the one that makes use of the augmented reality technology. In here, the application uses the device's camera to capture the users surroundings and superimposes icons that identify the nearby buildings. Similarly, the app also has a *Map View* screen which uses a more traditional 2D map interface, using the Google Maps API, in which the POIs are also shown. By selecting any of these POIs, the app display more information about it as well as directions on how to get to it using the smartphones' maps application. Additionally, depending on the POI, the app allows the user to make a call, play audio and video, and open the POI's web page.

The system architecture of the iMAP-CampUS (Fig. 2.20) was built using the *Layar* platform, an SDK for building augmented reality applications, and is made up of five components: (1) the client app that runs on the mobile device of the user, through which GPS coordinates of the current location are gathered; (2) the Layar server, which is the core component of the system and functions as the intermediary between the remaining components; (3) the *Layar* publishing website, used for administrative and management purposes; (4) the content sources that stores the content to be viewed in the app, this includes a database with the POI details; (5) and the service providers, responsible for providing these contents when requested.



Personal Hotspot: 1 Connection
10 points of interest found
Map
Done
Macquarie Library
The Macquarie University Library is the largest academic library in Northern Sydney. The librar....ver 1.8 million print
478 ft
506 ft
Macquarie Map
Call Developer
Email Developer
Take me there

Figure 2.19: iMAP-CampUS user interface [5]

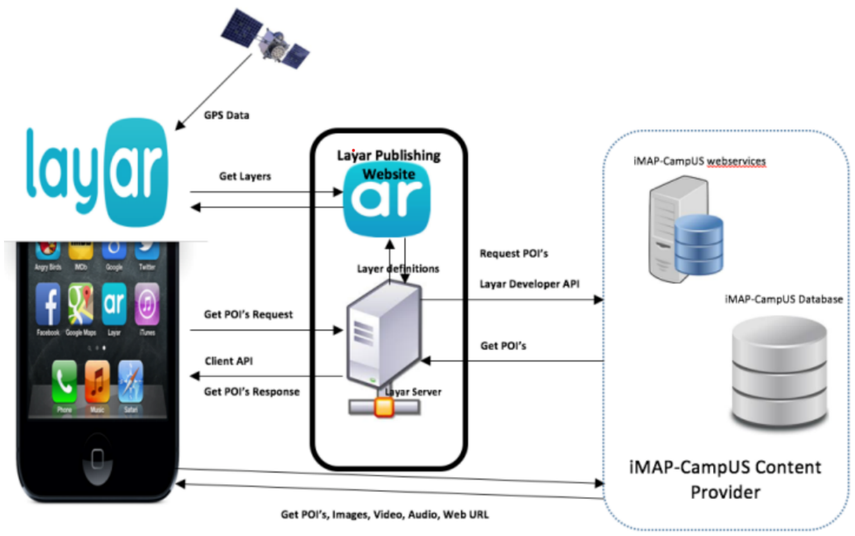


Figure 2.20: iMAP-CampUS system architecture [5]

The iMAP-CampUS has been developed for both iOS and Android platforms and run on smartphones and tablets with different screen sizes. The app also has a "share" feature through which users can share the app with friends through various channels such as *Facebook*, *WhatsApp* and *Twitter*. Finally, the authors mention plans to evaluate the acceptance of the app in the future.

Table 2.20: Alqahtani et al. (2017) classification

Development Approach	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Hybrid	Mass Audience (Multiple Device Types)	Basic Personalization	Good Usability	Interactions among contents	N/A

2.3.4 Summary

The landscape of MTGs employed in tourism is extremely diverse in terms of their architectural, technological and functional aspects. They have evolved from the earliest tour guides like *GUIDE* [21] to more automated and personalized apps that adapt to the user's needs with real-time data. All of them try to enhance the travel experience, making use of mobile phones' particular set of characteristics to answer travelers' expectations of instant access to relevant and personalized information anywhere, anytime. A summary of the classifications can be found in Table 2.21.

Most of the apps developed in these studies are noticeably older and have become outdated when it comes to the graphical interfaces and user experience offered. Successful commercial implementations of MTGs such as *Google Maps* or *TripAdvisor* have raised the bar on what to expect from this type of app. Additionally, the devices of the time were more limited and had smaller screen sizes which also increased the difficulty of building a user interface. Our prototype will attempt to provide a refreshed user interface and experience that can compete with these commercial apps for the role of the traveler's primary MTG.

In the same vain, the age of these studies means that the development approach used in most cases has become outdated and sub-optimal. In our work we intend to investigate and detail the process of developing an app using a modern SDK. Another direct consequence of the older development approaches is that the availability of the app is in most cases inferior that what is achievable today by the same "cost". Since mobile devices have become much more "standardized" (i.e. Android or iOS devices), it is easier to target a larger audience without any extra effort. In our work, we make it a priority to have mass availability.

One noticeable pattern in most studies is that they make use of the Internet (or other alternatives) with the expectation that the network infrastructures will evolve to the point where tourists having easy access to Internet connection won't be a problem. While it is certainly true that Internet is much more readily available for a tourist now that it was before, we believe that increasing the quality of service with the trade-off of more internet dependency is not the path to take for MTGs. Therefore, we will explore how to diminish MTGs dependency on Internet connection.

Another issue found in these studies is that while most claim to have done some sort of validation of the usability of the apps, either with field studies, user surveys, or conducting reviews with experts, only a few documented in what ways did this feedback affect the design process. In our work, we intend to take a user-centered design approach in which we employ exploratory techniques, in collaboration with potential users, in order to guide the design process.

Finally, we found very interesting uses of social features, such as having user generated content like ratings and comments, to improve the overall quality of the recommendations. One path that was not fully explored is how to use social features, such as sharing trips to friends, can increase user engagement and bring new users to the app.

Table 2.21: Primary studies classification

Authors	**Development Approach**	**Availability**	**Personalization**	**Usability**	**Social Features**	**Evaluation**
Cheverst et al. (2000)	Web	Limited Audience (Specific Device Type)	Intermediate Personalization	Average Concern	Interactions Among Users	Empirical Evaluation
Dunlop et al. (2004)	Unknown	Moderate Audience (Single Device Type)	Basic Personalization	Average Concern	Interactions Among Users	Empirical Evaluation
Ricci et al. (2006)	Unknown	Unknown	Intermediate Personalization	Average Concern	No Social Features	Empirical Evaluation
Cena et al. (2006)	Web	Mass Audience (Multiple Device Type)	Advanced Personalization	High Concern	No Social Features	Evaluation by Experts
Cunningham et al. (2006)	Web	Limited Audience (Specific Device Type)	Advanced Personalization	Average Concern	Hybrid Interactions	N/A
Pospischil et al. (2007)	Web	Moderate Audience (Single Device Type)	Basic Personalization	High Concern	No Social Features	N/A
Kramer et al. (2007)	Unknown	Moderate Audience (Single Device Type)	Intermediate Personalization	Average Concern	Interactions Among Contents	Empirical Evaluation
Lim et al. (2007)	Unknown	Moderate Audience (Single Device Type)	Intermediate Personalization	Unknown	No Social Features	Empirical Evaluation
Kenteris et al. (2009)	Hybrid	Moderate Audience (Single Device Type)	Basic Personalization	High Concern	No Social Features	Empirical Evaluation
Garcia et al. (2009)	Web	Mass Audience (Multiple Device Type)	Advanced Personalization	No Concern	No Social Features	N/A
Höpken et al. (2010)	Web	Mass Audience (Multiple Device Type)	Intermediate Personalization	Average Concern	No Social Features	Empirical Evaluation
Noguera et al. (2012)	Native	Mass Audience (Multiple Device Type)	Intermediate Personalization	High Concern	Interactions Among Contents	Empirical Evaluation
Wijesuriya et al. (2013)	Native	Moderate Audience (Single Device Type)	Basic Personalization	No Concern	No Social Features	N/A
Yang et al. (2013)	Hybrid	Moderate Audience (Single Device Type)	Basic Personalization	High Concern	No Social Features	Empirical Evaluation
Zhao et al. (2015)	Native	Moderate Audience (Single Device Type)	Basic Personalization	No Concern	Hybrid Interactions	N/A
Alqahtani et al. (2017)	Hybrid	Mass Audience (Multiple Device Types)	Basic Personalization	Good Usability	Interactions among contents	N/A

[This page has been intentionally left blank]

Chapter 3.

Sustainable Tourism Crowding Project

Contents

This chapter is dedicated to the *STC* research project, which this dissertation is part of. In the first section, an overview of the project is given, including its mission and goals, and insight into its technical components. Section 3.2 explains how the work undertaken in this dissertation fits into the main project and how it interacts with the other components.

[This page has been intentionally left blank]

Chapter 3
Sustainable Tourism Crowding Project

3.1 Project overview

The STC research project, introduced in chapter 1.1, aims to locally mitigate tourism overcrowding and promote tourism sustainability by distributing tourists over less crowded and more sustainable POIs. To accomplish this goal, the STC project developed a strategy that consists of three key initiatives:

1. Detecting and predicting crowding and their movement;
2. Generating walking trip recommendations that avoid places with a higher concentration of population and promote the visitation of more sustainable POIs;
3. Providing these trips to tourists through a mobile app that guarantees user engagement and retention.

The main efforts of this project target the *Santa Maria Maior* district, the historical center of Lisbon, which has already been the subject of initial case studies and prototypes [65, 78], due to the large number of visitors it welcomes every year. Nonetheless, the ultimate goal of this project is to prove that a system can be built to help mitigate the unintended side effects of tourism in any specific location.

This project makes use of a distributed microservices architecture, represented using AjiL, a graphical modeling language for microservice architectures [82] in Figure 3.1, that consists of the following components:

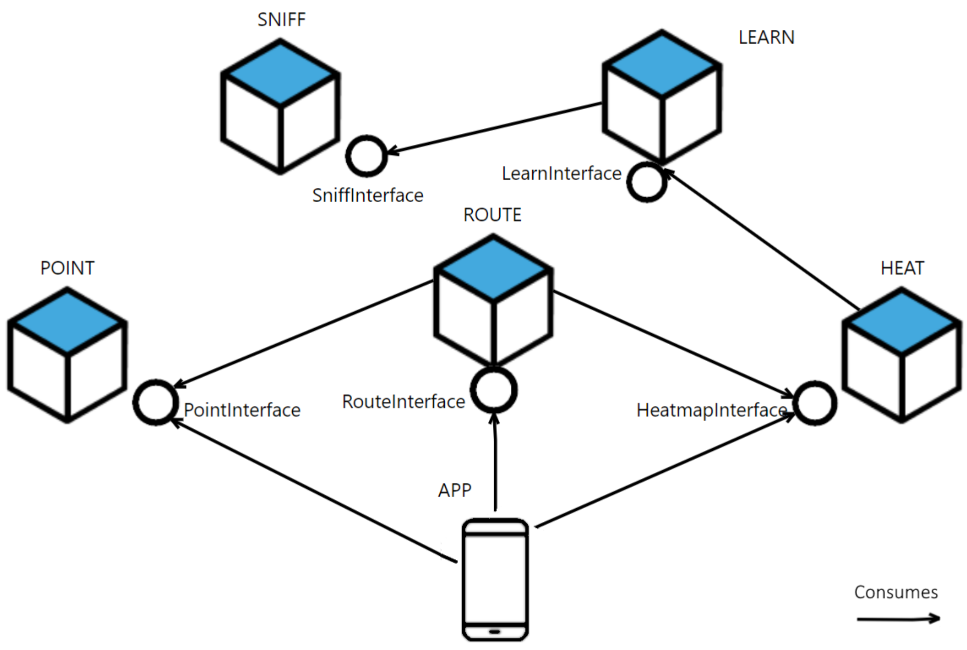


Figure 3.1: Project overview diagram

- APP - The interface between the tourist and the system. It is a mobile app that listens to tourists' preferences and constraints and provides them with trip recommendations generated by the system. Also functions as a monitoring tool that feeds useful information (e.g. user's location) back to the system;

- SNIFF – Edge computing device that detects the number of mobile devices active in its vicinity, considered to be a good surrogate for the number of people there, and feeds this information into the system [78]. The device is able to detect nearby mobile devices by listening to the trace elements of the usage of *Wi-Fi*, *Bluetooth*, and mobile network;

- ROUTE – Microservice that provides walking tour recommendations for APP instances. Uses POI and crowding information from the system to generate sustainable recommendations according to the tourist's preferences and constraints.

- POINT – A map-based web interface meant for local territory administrators (e.g. *Santa Maria Maior* parish council workers) to easily insert, remove and edit POIs into the system. The system relies on territory administrators' local knowledge to input sustainability data to the POIs [65];

- LEARN – Microservice responsible for producing crowding forecasts by studying historical crowding data and extracting patterns from it. It includes a web interface for the visualization of said projections over different periods of time;

- HEAT – Consists of a cartogram database of the current population's location and it's respective representation via a heatmap. It listens for updates from other sources to update its data, at regular points in time;

Most of these microservices are still in development and being iterated upon, so their requirements can be expected to change over time. Additionally, most of them communicate or depend on others as reported by the dependencies in Figure 3.1. Some details have been purposefully omitted (with the exception of the APP, explained in the next section), as they would add complexity to this overview and are considered out of scope for this dissertation.

3.2 Mobile APP

This distributed architecture means that there is a good separation of concerns, which also helps the process of breaking down the work effort into individual tasks. Each team member of the STC, comprised mostly of master's thesis students, is assigned responsibility for a component. This dissertation is responsible for the Mobile APP component and its integration with the system.

In order for the APP to work properly, it relies on other microservices. Likewise, other microservices depend on the APP to function. The following use case diagram (Fig. 3.2) summarizes the scenarios where the app communicates with other microservices in the STC project.

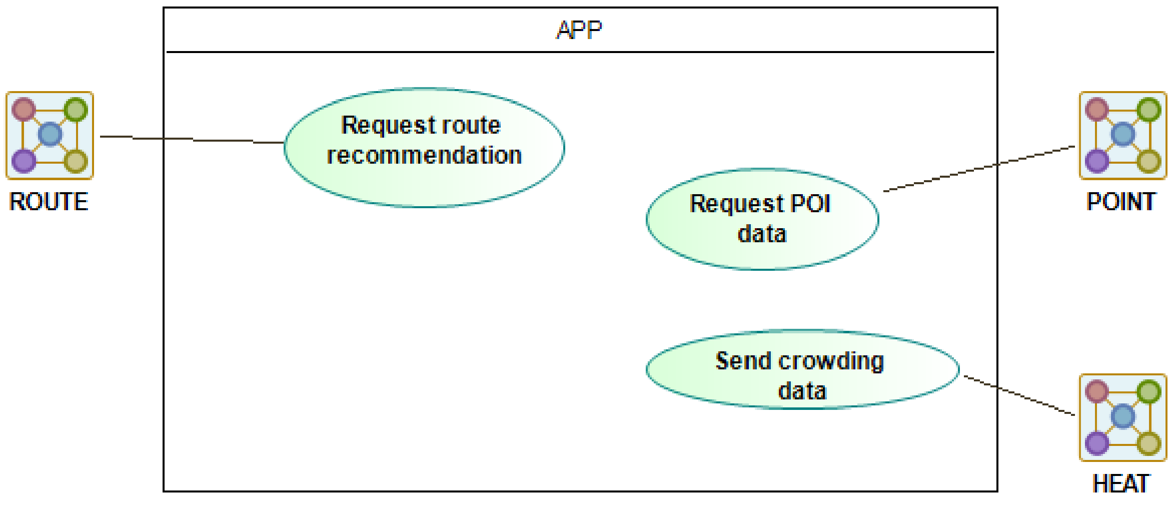


Figure 3.2: APP - Project communication use case diagram

Since the APP runs on the end user's mobile phone, it is outside of the system's internal network. This means that all required communications have to be made through REST Application Program Interface (API)s exposed by the internal microservices.

The process of designing the APIs was a collaborative effort that made extensive use of *Stoplight*[1]. *Stoplight* is an online tool for designing APIs following the OpenAPI[2] format: a standard, language-agnostic interface. It offers an intuitive user interface, called *Stoplight Studio*, where we define operations and the respective JSON schema for the messages. Furthermore, *Stoplight* also provides server mocking, documentation generation, and support for version control (i.e. git), which also improved the development process.

3.2.1 APP - POINT communication

It is important for APP instances to have an updated list of the city's POIs. This information is used in multiple ways, such as displaying them through the mobile app on a map-based interface. The only way to ensure that the APP has the newest version of this list is to periodically ask for updates (i.e. at startup). This process consists of sending a GET request to the POINT microservice, more precisely, to the /points path (Fig. 3.3).

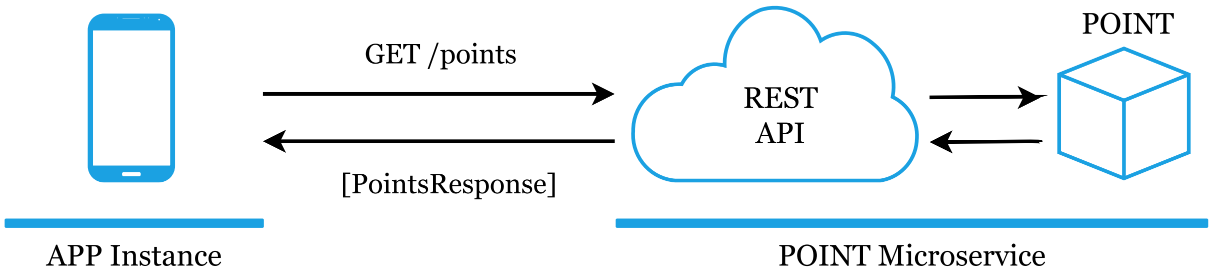


Figure 3.3: APP - POINT communication

[1]https://stoplight.io/
[2]https://www.openapis.org/

The response message body consists of a *PointsResponse* that simply contains an array with multiple *PointOfInterest* objects. An example representation of the *PointOfInterest* JSON object can be found in Appendix A.

3.2.2 APP - ROUTE communication

APP instances will often need to provide users with personalized trip recommendations. These trips are not generated by the APP, but instead by the dedicated ROUTE microservice, which has access to all the crucial information in order to generate the best trips for the user, on demand.

APP instances gather all the necessary information on the user side, such as their preferences (e.g. POI categories to visit) and constraints (e.g. budget) and sends these to the ROUTE microservice. ROUTE uses these constraints, along with external factors, to generate a selection of trips, and sends them back to the APP instance.

In technical terms, this translates into the APP instance sending a POST method to the /routes path (Fig. 3.4). Typically, this would be done over a GET method. GET methods are used to retrieve a resource, while POST methods are used to create, update or delete resources [59]. However, due to the need to pass a high number of parameters, GET becomes a less appealing option over the simpler solution of sending these parameters in the body of a POST method. This solution can be seen in similar use cases, such as the *Graphhopper* API[3], where a POST method is used to get routes.

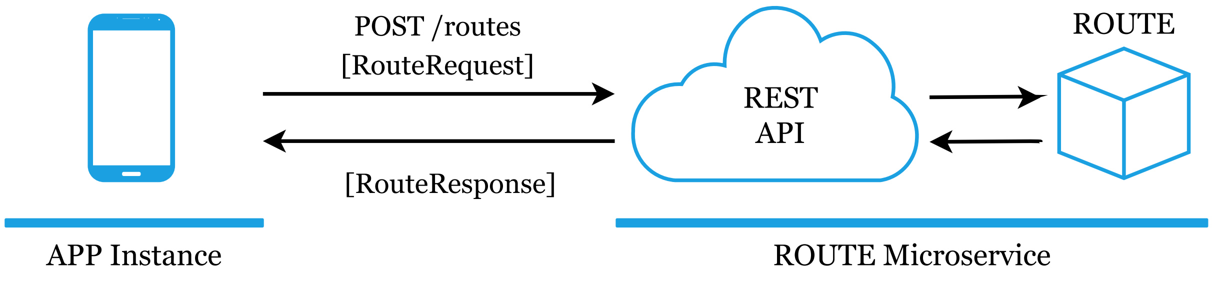


Figure 3.4: APP - ROUTE communication

Appendix B contains a series of scenarios used to test APP - ROUTE communication, each one demonstrating the JSON representations of both the *RouteRequest* JSON object and the *RouteRequest* JSON object.

3.2.3 APP - HEAT communication

Unlike the previous cases, this line of communication is used exclusively to feed information back to the system. The HEAT microservice requires constant periodic updates to prevent its heatmap data from becoming outdated, and therefore ineffective. Aside from the SNIFF microservice, which exists for the sole purpose of feeding this data to HEAT, the APP can also serve as a provider of crowding data. The scale of the provision depends on the number of APP instances.

[3]https://docs.graphhopper.com/#operation/postRoute

APP instances run on mobile phones equipped with GPS and Bluetooth capabilities. These capabilities allow an APP instance to keep track of the user's location and scan nearby devices to get an estimation of nearby people. As tourists travel the city, running APP instances will be able to obtain data from disperse locations that are not targeted by SNIFF devices.

This data is periodically sent to the HEAT microservice in the form of a POST method to the /tracking path (Fig. 3.5).

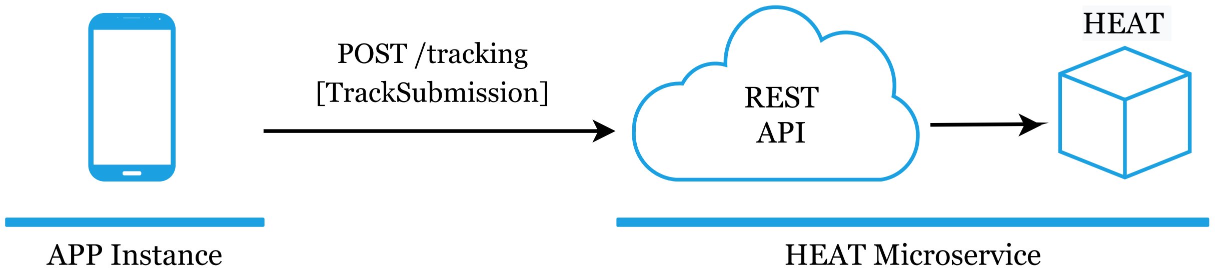


Figure 3.5: APP - HEAT communication

The message body consists of a *TrackSubmission* that simply contains a location, a timestamp and a device count. An example of its JSON representation can also be found in Appendix A.

[This page has been intentionally left blank]

Chapter 4.

Development Approach

Contents

This chapter covers the development decisions made to develop the prototype. We discuss the *SDK* used (Sec 4.1), the application architecture chosen (Sec. 4.2) and finally how we implemented the Bluetooth detection service (Sec. 4.4).

[This page has been intentionally left blank]

Chapter 4
Development Approach

4.1 *Flutter* SDK

Our prototype mobile application was developed using *Flutter*[1], a fairly new *SDK* by Google for creating cross-platform applications. It is based on the *Dart* programming language, also built by Google. In the next section (Sec. 4.1.1), we present the criteria used to determine which *SDK* to use for the development of our app. Afterwards, we give a high-level overview of the architecture of *Flutter* (Sec. 4.1.2) and explain some fundamental concepts that define the way we develop applications using it (Sec. 4.1.3). Lastly, we go over the experience of using such *SDK* and the tools that *Flutter* provides to help the developers work fast and easily (Sec. 4.1.4).

4.1.1 Decision

Traditionally, developers were forced to build their mobile apps for each operating system using their respective *SDK*s [62]. *Android's SDK* is based on *Java* [12] (*Kotlin* is also supported nowadays), while *iOS's SDK* is based on *Objective-C* [4] (*Swift* is also supported nowadays).

Today, the number of development options is much larger. Besides the previously mentioned (1) native approach, there is the opportunity to use (2) mobile web apps that run on the device's browser and don't need to be downloaded [8], (3) Progressive Web App (PWA)s that take advantage of modern browser capabilities to improve on the features of traditional web apps, and can be installed just like a normal app [8], and finally, (4) hybrid apps that combine both native development and web technology [67].

All four development approaches have advantages and drawbacks (Table. 4.1) [8, 19, 24, 62, 67, 76, 80, 94]. Additionally, each of these categories has its own set of tools and *SDK*'s that work differently and impact the software development process [24] [42].

Table 4.1: Advantages and drawbacks of different development approaches [67]

Features	**Native App**	**Hybrid App**	**Web App**	**PWA**
Development Language	Native Only	Native and Web or Web Only	Web Only	Web Only
Device Access	Full	Full	Partial	Partial (Better than Web App)
Speed	Very Fast	Native Speed	Fast	Fast
Development Cost	Expensive	Reasonable	Reasonable	Reasonable
App Store	Yes	Yes	No	Both

In order to better understand which approach is more suitable for our application, we need to define a set of requirements. Requirements are descriptions of how the system should behave, or of a system property or attribute [81]. More specifically, we need to define requirements that

[1] `https://flutter.dev/`

act as constraints on the development process of the app so that we can rule out development approaches and reach a conclusion.

Ultimately, we decided to use the *Flutter* *SDK*, which fits into the hybrid development category. Listed below are the requirements used to evaluate different options and an explanation on how *Flutter* qualifies in each one:

4.1.1.1 Cross-Platform support

Data from multiple sources[2,3] show that *Android* and *iOS* combined have approximately 99% of the mobile operating system market share worldwide in 2019. Targeting both platforms ensures that almost all tourists have easy access to the app from their smartphones.

To do so, we can build two native applications for *Android* and *iOS*, or opt for a mobile *SDK* that supports cross-platform applications. The first option can easily become a waste of development time and effort, and imply an increased maintenance cost [94]. Due to time constraints, we can immediately rule out this alternative. *Flutter* belongs to the second category, which according to their website[4]: *"Flutter is for developers that want a faster way to build beautiful apps, or a way to reach more users with a single investment"*. By using *Flutter* we can speed up the development process and reduce the cost and complexity of app production across platforms. This is because we only need a single codebase (both for the app logic and for building the UI) that can compile and run natively on both operating systems.

4.1.1.2 Distribution

Typically, apps are distributed in online marketplaces that handle all the complexity of distribution for developers and businesses, making them easily available to everyone. The most common marketplaces are *Google Play* and *Apple App Store*, the official app stores for *Android* and *iOS* respectively [42]. For iPhone users, *Apple App Store* is the only legitimate way to get apps. For *Android* users, apps can also be downloaded from other marketplaces like, *Amazon Appstore* or *Aptoid*.

We definitely want to deploy our app into the main marketplaces of each platform. *Flutter*'s documentation provides detailed guides[5,6] with step-by-step instructions on how to release an app into these stores.

4.1.1.3 Offline map support

Later, in section 5.3.1, we discuss the importance of using offline tiles for our map, instead of the regular approach of using the internet to fetch the latest tiles on demand from a tile provider (e.g. *Google Maps*, *Mapbox*).

Most mobile *SDK*s have some sort of map support that is mostly a wrapper for a Google Maps native view. This means that there is no support for offline mode. This is also the

[2]https://gs.statcounter.com/os-market-share/mobile/worldwide
[3]https://deviceatlas.com/blog/android-v-ios-market-share
[4]https://flutter.dev/docs/resources/faq
[5]`https://flutter.dev/docs/deployment/android`
[6]`https://flutter.dev/docs/deployment/ios`

case with official map package in *Flutter*. However, there is also a community made package, *flutter_map*[7], that supports drawing markers and polylines, the normal map controlls such as panning and zooming, and finally using *png* images (transformed from map tiles) under the format of */z/x/y.png* where *x*, *y* and *z* are the coordinates of the image to display. This way, it is possible to use map data from any source, such as OpenStreetMap data in our case, and convert it into the map images to include in the application, and have access to them offline. This alternative solution has proven to be suitable for our project.

4.1.1.4 Hardware access

The application is designed to take advantage of the device's many hardware features, such as GPS and Bluetooth sensors, or the disk for local storage. This means that we need to use a development approach that supports easy access to the underlying hardware of the device. A Web app lacks that support [94], which rejected its choice for building this app.

Flutter uses a flexible system that allows you to call platform-specific APIs whether available in *Kotlin* or *Java* code on *Android*, or in *Swift* or *Objective-C* code on *iOS* (explained in section 4.1.2). Using this technique, we can write native code that communicates with the operating system's APIs or better yet, make use of already existing *Flutter* packages (produced either by the *Flutter* team or by the community) that abstract and simplify this task.

4.1.1.5 PWA vs Hybrid development

From the previous requirements both native and traditional web approaches to mobile development have been excluded. The final decision is to chose between PWAs or a hybrid development approach. PWAs are a new generation of web applications that load just like regular websites but take advantage of features supported by modern browsers, including service workers and web app manifests [9]. Through these features, PWAs can make use of devices APIs (e.g. Bluetooth, GPS, storage) and they can work completely offline (and also be installed via APP stores).

However, support for these features is still improving gradually. Apple's *Safari* browser is the slowest browser to adopt these new technologies, so the iOS platform is yet to fully realise and leverage the potential of the technological advancement. For instances, as of 2020 *Safari* does not yet properly support push notifications. For the usage context of our app, which foresees background location tracking for long periods of time, this approach is not ideal. Ultimately, it was decided to adopt a hybrid development approach in which both *iOS* and *Android* are supported as traditional apps, from a single codebase.

4.1.1.6 Hybrid development SDK

The hybrid-app approach is the right one for our project, where we benefit from full device access with a lower development cost than native. In this category, the most appealing options are the *React Native*[8] *SDK* and the *Flutter* *SDK*. Both are strong candidates that are perfectly able to meet our requirements and more, while still providing a good developer experience. The final decision was made simply according to the previous experience with the *Flutter* *SDK*,

[7] https://github.com/fleaflet/flutter_map
[8] https://reactnative.dev/

in hopes of speeding up the development process by avoiding any unnecessary learning curves that come with every new SDK.

4.1.2 Architectural overview

Flutter is designed as an extensible, layered system (Fig. 4.1). It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable [30].

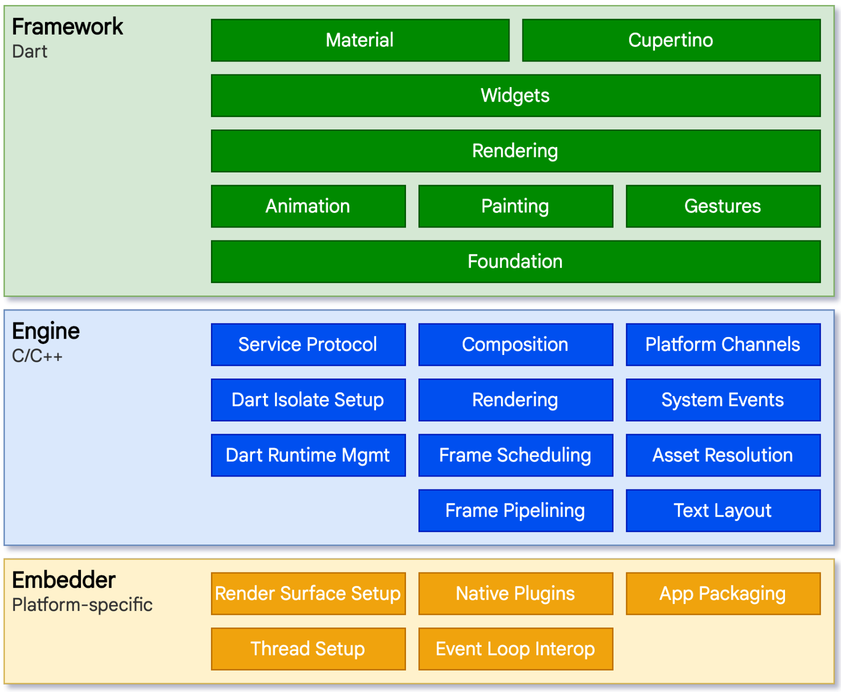


Figure 4.1: *Flutter* architecture [30]

To the underlying operating system, *Flutter* applications are packaged in the same way as any other native application. *Flutter* apps are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if targeting the web. Using the embedder, *Flutter* code can be integrated into an existing application as a module, or the code may be the entire content of the application.

At its core, the *Flutter* engine, which is mostly written in *C++*, supports the primitives required to support all *Flutter* applications. The engine is responsible for rasterizing composite scenes whenever a new frame needs to be painted. It provides the low-level implementation of *Flutter*'s core API, including graphics (through *Skia*, a *C++* graphics engine), text layout, file and network I/O, accessibility support, plugin architecture, and a *Dart* (the programming language of *Flutter*) runtime and compile toolchain.

The engine is exposed to the *Flutter* framework through *dart:ui*, which wraps the underlying *C++* code in *Dart* classes. This library exposes the lowest-level primitives, such as classes for driving input, graphics, and text rendering subsystems. Most of the time developers interact only with *Flutter* framework, which provides all the high level APIs needed to build the UI. This layer is relatively small and can be added upon with packages that function like plugins, such as platform plugins (e.g. camera, webview), platform-agnostic features (e.g. animations, http) and more.

4.1.3 Core concepts

In order to learn how to make apps with *Flutter*, we must first understand a few core concepts that define the development process. The *Flutter* documentation does a great job of introducing new developers to the language, covering a variety of topics from networking to internationalization. In this section, we will only cover the most essential topics relevant to the project.

4.1.3.1 Widgets

Widgets are the basic building blocks of a *Flutter* UI. In fact, almost everything in *Flutter* is a widget. A widget is an immutable object that describes a specific part of a UI. They are composable, meaning that you can combine existing widgets to make more sophisticated widgets. A widget's main job is to implement a *build* function, which describes the widget in terms of other, lower-level widgets [31].

The current version (1.20[9]) of *Flutter* contains 395 out of the box widgets. These widgets can be divide into three main categories: basic widgets, layout widgets, and customization widgets [31]. Basic widgets are the fundamental pieces of the UI. The best examples of this category are the *Text* and *Image* widgets. Much like a raw HTML page (with no styling), these widgets allow for creating a functional app, albeit one that lacks the qualities that make an app desirable. Layout widgets are used to determine how other widgets are organized. The most commonly used layout widgets are the *Column*, *Row* and *Stack* widgets, which are used to create the majority of layouts, by arranging children (other widgets) vertically, horizontally, or overlapping each other. Finally, customization widgets are used for styling and personalizing other widgets, like adding a padding around them (with the *Padding* widget).

All UI screens of a *Flutter* app are composed by a combination of these three widget categories. Basic widgets represent the content to be displayed, the layout widgets determine how to arrange them and the customization widgets change their appearance. If somehow a combination of these three is not enough to create the desired UI element, there is also the *CustomPaint* widget that provides a canvas on which the developer can paint using a low-level painter to draw shapes.

4.1.3.2 Widget tree

Since everything in *Flutter* is a widget, and widgets are combined to create more complex widgets/screens, this means that there has to be some sort of structure to organize them. *Flutter* uses a widget tree structure, where widgets become "parents" and "children" to other widgets. Take, for instances, the *TextButton* widget, which displays a borderless button that can be pressed. This widget takes another widget as its child, usually a *Text* widget, that consists of the text inside the button. Widgets can also take multiple children, like the *Row* and *Column*, that display multiple widgets in their respective axis.

[9] https://medium.com/flutter/announcing-flutter-1-20-2aaf68c89c75

This idea of creating more complex widgets or full screens (which are themselves widgets) by combining multiple simpler widgets is called *Composition*, and ultimately forms the widget tree hierarchy. Using the example counter app that comes with every new *Flutter* project, we can see how a screen translates into a widget tree (Fig. 4.2).

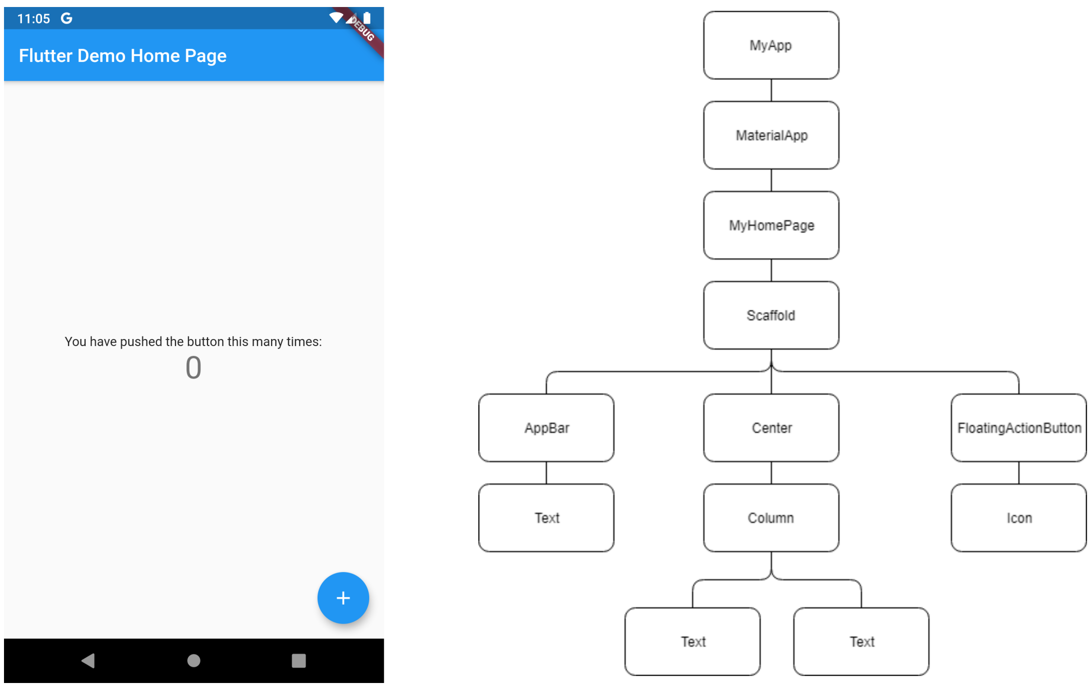


Figure 4.2: Counter app and its corresponding widget tree [32]

This tree based approach to building UIs does have some disadvantages. For starters, the nesting of widgets can hurt the code legibility and increase the file size. But perhaps the more important issue is with performance. In certain situations, when a parent widget needs to rebuild itself it will also cause a rebuild on all descendent widgets. If this parent widget is high on the tree hierarchy it will force a tremendous amount of rebuilds on descendent widgets, which can cause an on performance.

4.1.3.3 Declarative UI

Most UI frameworks use an imperative style of UI programming, where UI entities are manually constructed, and then later mutated using methods and setters. This means that if the state of the app changes, it is necessary to obtain the instances of the views and manually update them to reflect the current state. This is the style used in traditional *Android* and *iOS* programming.

Flutter uses a declarative style of UI programming. That means that instead of manipulating views every time the state changes, we simply change the state and that triggers a redraw of the parts of the UI that need to reflect the new state. Widgets are immutable [32], which means that they cannot be mutated, only rebuilt.

One benefit of this approach is that there is only one code path for any state of the UI. *"We describe what the UI should look like for any given state, once—and that is it"* [32]. There is also no separation of code and layout. In *Android*, it is common to construct the layout using a

XML layout file and then set up UI behaviour with *Java* code, while in *Flutter* both are done by widgets, using only *Dart* code, so there is no change in programming context.

For the sake of comparison, let us imagine a simple screen with a text label displaying "*Click the button*" and a button that, when clicked, changes said label to display "Button clicked!" (Fig. 4.3). This text represents the state of the application. It has an initial value and after user input this value changes and the UI needs to reflect this change.

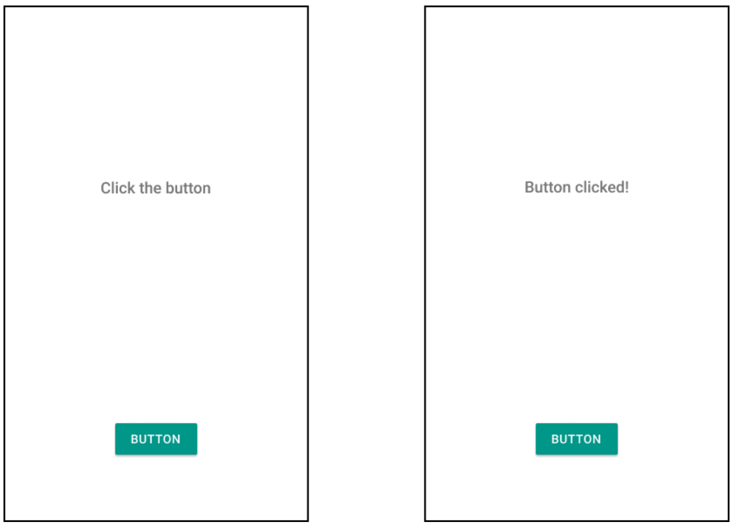


Figure 4.3: Comparison screens

In a traditional *Android* application, there are a couple of common operations developers may wish to perform on a view: set properties, set focus, set up listeners and set the visibility [6]. For this example, we want to set up a listener for catching the button click and then set the property of the text to the new value "*Button clicked!*". To do this, we would first have to obtain the instances of the views from the XML layout file. The code in Listing 4.1 demonstrates how this is typically done in *Android*.

In *Flutter*, there is no separation from code and layout, so we do not need to worry about binding code to views and retrieving instances of views from the XML layout file using IDs. To achieve the behaviour in the example, we define a stateful widget (extends the *State* class) that holds the string of text, and builds the text widget with the string message. When the string message is changed from the button click event, we force the screen to rebuild and the UI will reflect the changes made. Stateful widgets are widgets that hold some form of state and can trigger a rebuild on themselves when this state changes, constructing a new widget subtree. The code in Listing 4.2 demonstrates how this is achieved in *Flutter*.

This new UI programming paradigm is also being adopted by both *Android* and *iOS* native *SDK*s. In 2019, Apple released *SwitfUI*[10], a declarative programming framework for constructing user interfaces on *iOS* and other Apple platforms. As of 2020, the *Android* team is working on its own version of declarative UI called *Compose*[11], a "*modern toolkit for building native UI*". In the world of web apps, frameworks like *Vue* and *React* also adopted the declarative way of writing UI.

[10]https://developer.apple.com/xcode/swiftui/

[11]https://developer.android.com/jetpack/compose

4.1.3.4 Asynchronous programming

Dart is a single-threaded language [22], which might not seem ideal considering *Flutter* applications need to operate with asynchronous and reactive code. *Dart* code runs in an *isolate*, which is a space with its own private chunk of memory and single thread running an event loop. If there is a need to perform a very time consuming computation, it is possible to create a separate isolate to complete the computation, leaving the main isolate free to render the UI in the meantime. The new isolate gets its own event loop and its own memory, which the original isolate, even though it is the parent of this new one, is not allowed to access.

In most cases, computations are not heavy enough that we need to move them to another *isolate*, so other ways of achieving asynchronous coding are preferred, specifically *futures*, *streams* and *async/await*, all built as APIs to manage the *event loop*. The *event loop*, in which the single thread runs on, simply takes the oldest event (e.g. button pressed) in the event queue, processes it, goes back for the next one, processes it, and so on. All of the high-level APIs and language features that *Dart* has for asynchronous programming are built on and around this simple loop.

One of the most basic APIs that *Dart* has for asynchronous programming is "futures", similar to the *promise* API found in *Javascript*. A future represents the result of an asynchronous operation, and can have three states: uncompleted, completed with a value or completed with an error. Take for example an http request (Lst. 4.3). Instead of blocking the thread awaiting for the network result, the *get* method immediately returns a *Future* and the thread moves on to the next piece of code. To register a callback for when the network request is completed we use the *then* method. This method takes a function as a parameter that is called when the future completes successfully and another function as an optional parameter for when the future completes with an error. There is a dedicated widget *FutureBuilder* to work with futures, wich takes a future and a builder method, and it automatically rebuilds its children when the future completes. In the builder method we can access the state and value of the future and build our widget accordingly.

"Streams" work similarly to futures, but they can deliver zero or more values and errors over time. To subscribe to a stream we use the *listen* method where again we must provide a function that is called every time a value is emitted by the steam, and an optional *onError* function to catch and process any errors. There is also a dedicated widget to work with streams. It is called *StreamBuilder* and it takes a stream and a builder method. When a new value is emitted by the stream it automatically rebuilds the child widget. In the builder method we can access the state of the stream (e.g. completed, waiting) and build our child accordingly. This way we truly achieve a reactive way of building widgets.

Finally, there is the *async/await* API. This API is simply an alternate syntax for using futures and streams that can help writing cleaner, more readable code. Imagine a scenario where we want to load an id value from disk, then fetch some network data according to that id and finally store that data into a class (Lst. 4.4). Using the future method, the code becomes hard to

Listing 4.1: Android example

```
//XML Layout omitted for brevity.

//Obtain the instance of the textview from the XML layout file
TextView textview = (TextView) findViewById(R.id.textview);
//Obtain the instance of the button from the XML layout file
Button button = (Button) findViewById(R.id.button);

//Create the listener function that will be given to the button
public onClickListener listener = new View.OnclickListener{
    onclick(View v){
      // Set the text property of the textview to display the new message
      textview.setText("Button␣clicked!");
    }
}

button.setOnClickListener(listener); //Add the listener to the button
```

Listing 4.2: *Flutter* example

```
class _ExampleScreenState extends State<ExampleScreen> {
  String _message = 'Click␣the␣button'; //The initial state of the UI

  @override
  Widget build(BuildContext context) {
    //Describes the part of the user interface represented by this widget.
    return Scaffold(
      body: Column(
        children: [
          Text(_message), //Text widget that displays the message
          MaterialButton(
            onPressed:
                _updateMessage, //Method called when the button is clicked
            child: Text('BUTTON'),
          )
        ],
      ),
    );
  }

  //Method to update the message
  void _updateMessage() {
    //Calling [setState] notifies the framework that the internal state
    //of this object has changed in a way that might impact the
    //user interface in this subtree, which causes the framework to
    //schedule a [build] for this [State] object.
    setState(() => _message = 'Button␣clicked!');
  }
}
```

Listing 4.3: Future example

```
void _onSuccess(http.Response response) {
  print('Success!␣$response');
}

void _onError() {
  print('Error!');
}

void _networkRequest(String url) {
  //Make the method call which returns a Future
  Future<http.Response> networkFuture = http.get(url);
  //Register the callback to when the future is completed
  //Provide a success and an error function
  networkFuture.then(_onSuccess, onError: _onError);
}
```

read because of all the chaining of methods. The point of *async/await* is to make asynchronous code look like normal code. First we add the *async* keyword before the function body, then we place the *await* keyword in front of each future the function needs to wait for. Note that while it might seem that the thread is locked awaiting for the results of the two methods, this is not the case. The event loop will continue to process other events until the futures from these methods complete. The underlying behavior of the event loop is completely identical in both cases.

Listing 4.4: Async / Await example

```
class DataHolder {
  final Placeholder data;
  DataHolder(this.data);
}

Future<int> _loadFromDisk() {
  //Simulate a 1 second operation and return the value 1
  Future.delayed(const Duration(seconds: 1)).then((value) => 1);
}

Future<Placeholder> _fetchNetworkData(int id) {
  //Simulate a 1 second operation and return fake data
  Future.delayed(const Duration(seconds: 1)).then((value) => Placeholder());
}

Future<DataHolder> createDataWithFuture() {  //Using the Future API
  return _loadFromDisk().then((id) {
    return _fetchNetworkData(id);
  }).then((data) {
    return DataHolder(data);
  });
}

Future<DataHolder> createDataWithAsync() async {  //Using the Async/Await API
  final id = await _loadFromDisk();
  final data = await _fetchNetworkData(id);
  return DataHolder(data);
}
```

These methods of asynchronous programming were very utilized throughout this project. Whenever there might be an event that will block the thread, specially ones that depend on hardware (e.g. GPS) or on network (e.g. communication with other microservices), we rely on futures and async / await to guarantee that the performance of the app is not affected. Furthermore, streams are heavily utilized in our state management approach, mentioned further below in section 4.2.3.

4.1.4 Developer experience

Flutter promises a fast and painless development experience. In this subsection we will go over the features and tools that contribute to it do be like that and also report on our personal experience.

Perhaps the most advertised feature in *Flutter* is the *Hot Reload* feature. Every time code changes are saved in a supported Integrated development environment (IDE), they are quickly reflected in the app. This is particularly useful when working on UI changes in a screen that is very "far" from the entry point of the app. Normally, to see the visual modifications introduced by new code, it is necessary to wait for the app to compile, load, and then navigate to the modified screen every single time. *Hot Reload* makes it easy to experiment with different UIs, new features and bug fixes by keeping the app running, maintaining its state, and updating only the changes introduced by new code.

When working on native app development, developers are forced to use either the *Android* Studio IDE ot the iOS Xcode IDE. Since IDE choice comes down to personal preference, the more IDEs an *SDK* supports the better. *Flutter* supports *Android* Studio, IntelliJ, Visual Studio Code and Emacs through extensions/plugins for *Flutter* that offer a lot of benefits to the developers, namely: code completion, formatting and highlighting; refactoring assists; quick code snippets and finally the *DevTools*, a suite of performance and debugging tools. It includes a widget inspector for visualizing and exploring widget trees to understand existing layouts and diagnosing layout issues. This inspector also allows for changing the actual the UI in a more "drag and drop" manner, and visualizing these changes live just like hot reload. Additionally, *DevTools* provide performance profilers for cpu, memory and network.

Flutter's documentation[12] is very extensive and welcoming of new developers. Its team has written specific articles for developers that come from other platforms, such as native mobile and web. Furthermore, there are available tutorials, codelabs (guided, hands-on coding practice) and cookbooks that demonstrate how to solve common problems. Moreover, the *Flutter* team created an open-source gallery[13] of sample *Flutter* apps and also a *Youtube* channel that discusses a lot of topics in video format (they even have a "show" with one-hour long episodes of the *Flutter* team live coding apps). Finally, by being open-source, developers can look into how certain widgets are built and apply these same coding practices into their own code. The community is also very supportive. A lot of help and discussion can be found on websites like *Stack Overflow*, *Medium*, *Reddit*, and *Youtube*. Due to the community, *Flutter* has a large *package* ecosystem that most likely will cover any use case a developer might have. If not,

[12] `https://flutter.dev/docs`

[13] `https://gallery.flutter.dev/#/`

these packages are also open-source and very often the maintainers are welcoming of new issues and pull requests, so while adjusting these packages for personal purposes, one can afterwards apply these improvements to the package itself. Our prototype makes use of a large number of packages, from networking to UI related, and even contributed to one with a pull request[14].

Being a fairly new *SDK*, *Flutter* sees a lot of updates over short periods of time. These updates often bring new or updated widgets, performance optimizations or development tool improvements. While this is a good sign that the *SDK* is under active and healthy development, it also has its downsides. Sometimes these updates are "breaking changes" and force developers to update their existing codebase, and this is also the case with packages. This situation happened frequently during the development of our prototype, which caused some minor delays. Nonetheless, the *Flutter* teams tries to minimize these breaking changes and when it does happen, they provide migration guides and documentation on how to migrate an older app into the new version.

4.2 Application architecture

There is no definitive way to design and build an app. Defining the app architecture is perhaps the biggest decision a developer has to make, as it will highly influence the complexity and maintainability of the codebase [84]. In this section we discuss the challenges involved in building a mobile application (Sec. 4.2.1), we review some of the most established design patterns used for applications (Sec. 4.2.2) and adapt them to the world of *Flutter* applications (Sec. 4.2.3). Finally we present an overview of the complete architecture of our prototype (Sec. 4.2.4).

4.2.1 Mobile development challenges

Besides sharing many of the challenges of traditional software development, such as traditional desktop and web applications, mobile app development has a unique set of issues that need to be addressed. In order to design an architecture, we must first understand what these issues are and how they are currently being solved.

While traditional applications often have clear and unique workflows, in the mobile world users often interact with multiple apps for different periods of time [20], resulting in apps that need to adapt to different kinds of user-driven workflows and tasks. In our particular case, it is not expected that tourists will have the app turned on for the whole duration of a trip, but instead will open and close it many times. Another extra consideration to keep in mind with mobile apps is the limitation of the devices. In some cases, the underlying operating systems can terminate idle or background apps if it decides it requires additional computational resources [54].

Another important concern comes from the usage context of mobile apps, particularly the environmental context [27]. Mobile apps need to be prepared for unexpected circumstances so that they can tolerate, or at least inform users, of external conditions that make it impossible for the app to function. An MTG such as ours needs to gracefully handle lapses in GPS and

[14] https://github.com/RaviKavaiya/sliding_panel/pull/20

internet connection for undetermined periods of time while maintaining the parts that don't depend on these external factors completely functional, resulting in a more reliable and robust experience.

Mobile apps are also constantly evolving. Every week, 29% of the top one thousand mobile apps are updated on Google Play Store. If we change the time period to every month the number increases to 65% and if we look at the number of updates yearly, 91% of these apps get updated at least once [1]. These large numbers demonstrate how important maintainability is in mobile development. The ISO/IEC 25010 standard defines maintainability as follows: *"Maintainability is the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers"*. In other words, we want our software to be easy to understand, enhance, or correct [29]. In [29, 34, 58] we find a number of common mistakes that make software harder to maintain. Generally, these mistakes come from large and complex classes, badly managed dependencies, and increased coupling in the system, all of which can be prevented by known software practices like separation of concerns and code reusability.

For our project, it is certain that there will be a need to revisit the codebase many times, as new features get requested and issues get discovered. Furthermore, as new members join the project to collaborate, we want to make the integration process as effortless as possible. A good way to diminish the entry barrier of our project is to adhere to the conventions and guidelines of the technology stack. Google provides guidelines for the *Dart* programming language under the name *Effective Dart*[15], which covers code style, documentation, usage and design. *Flutter*'s extensive set of resources and examples also serve as a reference to create consistent and familiar code.

4.2.2 Presentation design patterns

It is essential to design an architecture that addresses all the challenges and complexities of mobile app development. Such decisions are known to be important for the understand-ability and maintainability of the code [35, 84]. Furthermore, making a wrong decision may lead to increased complexity and maintenance overhead [85].

A good app architecture should decompose its components into at least three main layers [33]: (1) the presentation layer, responsible for displaying information to the user and for interpreting commands from the user into actions; (2) the domain layer, also known as business logic or simply the application's state, which represents the work that the application needs to do; (3) the data layer, responsible for communicating with other systems that carry out tasks on behalf of the application, usually databases or external APIs [33].

Perhaps the most important and complex part of an app architecture is the integration of the presentation layer with the domain layer. Fundamentally they are about different concerns. When planning or developing a visual element we think about the mechanisms of UI and how to layout a good user interface. When working with a model we think about the application logic. With this separation, we can build different presentations that rely on the same model, making it easy to change the visual aspect of an application without having to modify the application logic as well [97].

[15] https://dart.dev/guides/language/effective-dart

In the eighties, the first design pattern that attempted to solve this problem, called Model-View-Controller (MVC) was created by Trygve Reenskaug for the Smalltalk platform [53]. It is the pioneering pattern for synchronizing user interfaces with domain data citeSyromiatnikov2014. Over time, a few similar patterns emerged and became well established variants of the MVC, namely the Model-View-Presenter (MVP) and the Model-View-ViewModel (MVVM).

4.2.2.1 MVC Pattern

As expected, the MVC pattern is divided into three main components (Fig. 4.4): the view, the controller, and the model. The fundamental idea is the separation of the representation of the application domain (the model) from the display of the application's state (the view) and the user interaction processing (the controller) [85]. A view represents the display of the model in the UI, like an HTML page. A model is an object that represents some information about the domain. It is a non-visual object and contains all the data and behavior other than that used for the UI. A controller takes user input and manipulates the model and causes the view to update appropriately. The UI is therefore a combination of the view and the controller [33]. While the model is not aware of the other components, the view and controller maintain direct links to the Model in order to observe, read, and modify it [85]. Therefore, the only dependency comes from the presentation layer depending on the model and not the other way around.

MVC provides no explicit means to deal with the presentation of state that is not part of the model but that makes a user interface more convenient for usage [85]. For example, in an information dialog, the text could change color between red or green according to the success status of the action. Since the text color data is purely a user interface property it is not part of the model. This architectural pattern becomes less beneficial for modern development frameworks, like *Flutter*, where user interface widgets have some state of their own to properly render themselves and also come with basic input handling so that they don't require controllers.

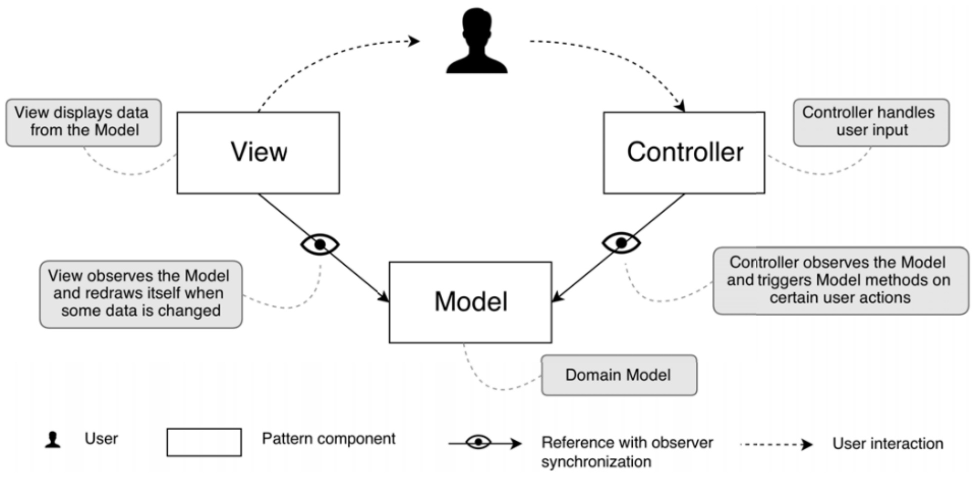

View displays data from the Model
View
Controller handles user input
Controller
View observes the Model and redraws itself when some data is changed
Model
Controller observes the Model and triggers Model methods on certain user actions
Domain Model
User
Pattern component
Reference with observer synchronization
User interaction

Figure 4.4: Model-View-Controller pattern [85]

4.2.2.2 MVP Pattern

The MVP pattern originated in the early nineties at Taligent [70]. Much like the MVC, there are also three main components, but instead of a controller there is a presenter (Fig. 4.4). Likewise, the view is the user interface (now with the added controller's responsibilities) and the model is the component that preserves data, state, and business logic. It exposes a group of service interfaces to the presenter and hides the internal details. The presenter sits in between the view and the model. It oversees the view, handles user events, and modifies the view via direct calls. This means that the view exposes a contract through which the Presenter accesses the portion of the view that is dependent on the rest of the system [85, 97]. It is worth noting that the MVP architecture has many variants within itself that give different responsibilities to each component, changing the structure presented in Figure 4.4.

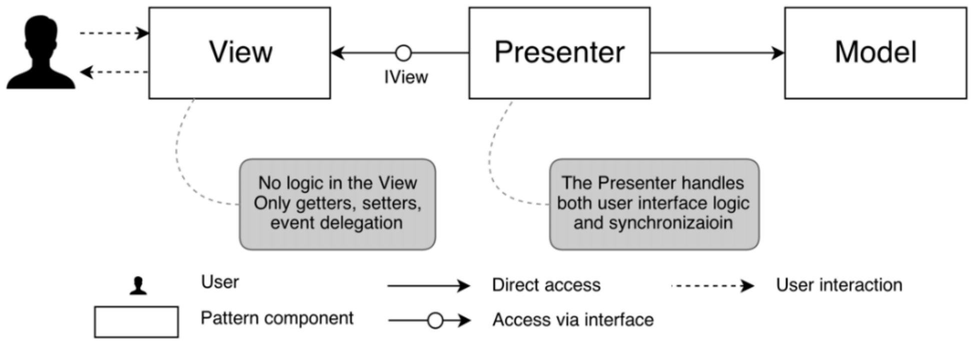

View
IView
Presenter
Model
No logic in the View Only getters, setters, event delegation
The Presenter handles both user interface logic and synchronizaioin
User
Pattern component
Direct access
Access via interface
User interaction

Figure 4.5: Model-View-Presenter pattern [85]

4.2.2.3 MVVM Pattern

The MVVM design pattern was presented in 2005 by John Gossman, one of the Windows Presentation Foundation and Silverlight Architects at Microsoft [79]. Microsoft designed these frameworks so that it is easy to build applications using the MVVM pattern, so it quickly became the lingua franca of the developers working with the frameworks [79]. In 2018, the *Android* team released *Jetpack* [16], a collection of *Android* libraries that incorporate best practices, eliminate boilerplate code, and provide backwards compatibility in your *Android* apps. With this announcement, the *Android* team started officially advocating for the MVVM pattern and supporting it with its own ViewModel library.

In MVVM, the View-Controller pair (as in MVC) is not considered as two distinct components, but merged in a single View component, so some presentation logic and basic user input handling are directly in user interface widgets. The domain model maintains domain state, much like in the other patterns. The viewmodel sits between the view and model and it is responsible for handling view state. It has access to the domain model, so it could work with domain data and invoke business logic [85].

Jetpack defines the ViewModel[17] as follows: *"A ViewModel object provides the data for a specific UI component, such as a fragment or activity, and contains data-handling business logic to communicate with the model. For example, the ViewModel can call other components to load the data, and it can forward user requests to modify the data. The ViewModel doesn't know about UI components, so it isn't affected by configuration changes, such as recreating an activity when rotating the device."*

[16] https://developer.android.com/jetpack
[17] https://developer.android.com/topic/libraries/architecture/viewmodel

The model is unaware of the viewmodel and the viewmodel is unaware of the view. This approach allows for the creation of several different views for the same data, and observer synchronization makes these views work simultaneously [85].

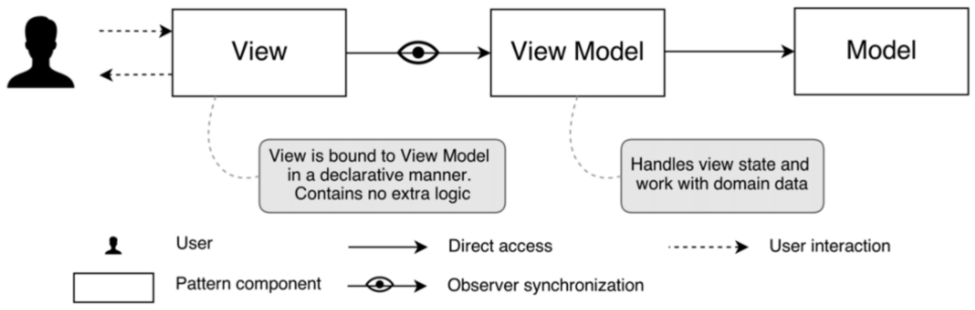

View
View Model
Model
View is bound to View Model in a declarative manner. Contains no extra logic
Handles view state and work with domain data
User
Direct access
User interaction
Pattern component
Observer synchronization

Figure 4.6: Model-View-ViewModel pattern [85]

All of these approaches have unique advantages and disadvantages, making it hard to reach a conclusion on which to adopt. Each pattern has a different purpose, so their value greatly depends on the context of the application. Furthermore, there is little to no empirical evidence that demonstrates which is better [57]. A good procedure is to also understand what solutions are being adopted for the specific technology stack in use.

4.2.3 *Flutter* state management

Within the *Flutter* community, the design pattern discussion is better known as state management, and it is one of the most discussed topics between developers, as it completely shapes the way the app is structured. In a similar fashion, there is also the need to separate the presentation layer from the domain layer. This presentation layer has ephemeral state and the domain layer has the application state.

In *Flutter*, the first type of state corresponds to *"state you can neatly contain in a single widget"* [32]. Take for example the index of the currently selected tab in a BottomNavigationBar. No other part of the application needs to access this kind of state, so there is no need for complex state management solutions. *Flutter*'s out of the box solution of stateful widgets discussed in 4.1.3.3 is ideal for this situation.

The second type of state is the state that *"you want to share across many parts of your app and that you want to keep between user sessions"* [32]. This could be the authentication state of the user or the shopping cart in an e-commerce app. It's easy to think of a situation where the shopping cart data is relevant in multiple parts of the app. In the product list screen, we may want to omit products that are already added to the cart. In the product details screen, we want to allow users to add the product to the cart, and may also want to highlight the fact that said product is already in the cart. Lastly, we also want to show this information in the shopping cart screen just before checkout, where users might want to remove products from the cart. The fact that this data needs to be accessed, modified, and synchronized from a lot of different classes and widgets increases the complexity of the application.

It is possible to manage this state with only stateful widgets, with techniques such as "lifting state up", where a stateful widget higher in the tree holds this state and passes it down to all the parts that need it. However, in large or medium-sized applications this can get out of hand quickly and become very hard to maintain, therefore it is recommended to not overuse this technique [32].

As a general rule, state that belongs only to a widget is ephemeral state and should be handled with a stateful widget. State that is shared between multiple widgets is app state that requires a more advanced state management technique (Fig. 4.7).

There are a growing number of community-made state management solutions that use a variety of different methods (e.g. streams, observers) to solve this issue, some even mimicking the design patterns mentioned above like MVC[18]. Perhaps the most relevant and established solutions are the Provider and the BLoC solutions.

Both solutions are open source community-driven projects[19,20] and highly appreciated by the community. They come in the form of packages that are added to a *Flutter* project like any other library. They work very similarly to the viewmodel in the MVVM pattern, sitting between the presentation and domain layers.

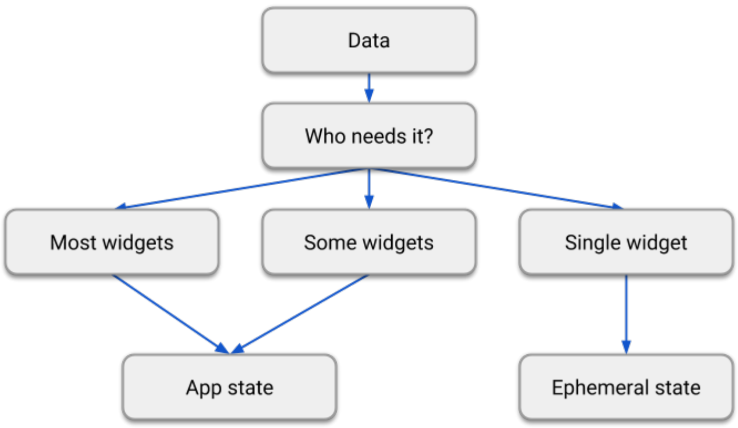


Figure 4.7: Types of state [32]

4.2.3.1 Provider

Provider is perhaps the simplest of the two because it involves less boilerplate code, making it a lower barrier of entry to new developers. For this reason, as well as the popularity of the package, Google started officially endorsing it as the go-to solution for state management.

With Provider, we start by creating a model class representing the application state that we want to share and synchronize between multiple widgets. Then, we provide this model to the widgets that need it. Since everything in *Flutter* is a widget, we use a widget (the ChangeNotifierProvider widget) to provide the model and we also use a widget (the Consumer widget) to consume said provider. Since we want to provide the same model to all the parts of the app that need it, we must insert it in the widget tree at a level that is higher and common to all the widgets that will consume it (Fig. 4.8). Consumers are widgets responsible for observing

[18] https://github.com/brianegan/flutter_architecture_samples/tree/master/mvc
[19] https://github.com/rrousselGit/provider
[20] https://github.com/felangel/bloc/tree/master/packages/flutter_bloc

the model and rebuilding the widgets beneath them. Anytime the model from the provider changes, the consumers will be notified and the widget beneath will be rebuilt with synchronized data. Widgets under the consumer have direct access to the model and can modify it via direct method calls.

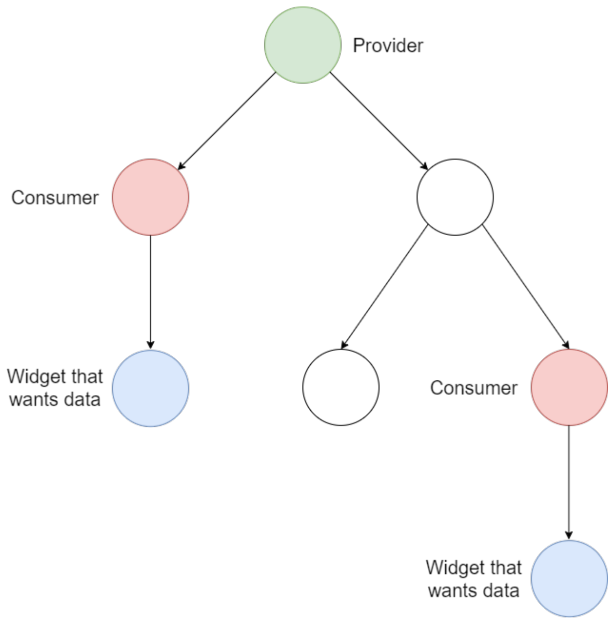


Figure 4.8: Provider widget tree example

Following the shopping cart example, the subsequent pseudo-code (Listing 4.5) demonstrates what the Provider implementation would look like. The shopping cart model contains a list of products and some utility methods. The ChangeNotifierProvider provides this model to anyone below it in the widget tree. The two consumers listen to changes in the model and update the Product Details and Shopping Cart screens. In the Product Details screen it is shown how to add a product to the shopping cart. In the Shopping Cart screen it is shown how to obtain the list of products to display.

```dart
class ShoppingCartModel with ChangeNotifier {
  final List<Product> _products = [];

  List<Product> get products => List.from(_products);

  void addProduct(Product product) {
    _products.add(product); //Could add some business logic here
    notifyListeners(); //Notify the UI so it reflects the changes made
  }

  void removeProduct(Product product) {
    _products.remove(product);
    notifyListeners();
  }

  bool contains(Product product) {
    return _products.contains(product);
  }

  bool isEmpty() {
    return _products.isEmpty;
  }
}

//Provider
ChangeNotifierProvider<ShoppingCartModel>(
    create: (context) => ShoppingCartModel(),
    child: ... //the rest of the widget tree
);

//Product Details
Consumer<ShoppingCartModel>(
  builder: (model) {
    ... //Rest of the Product Details UI...
    RaisedButton(     //Button to add product to cart
      child: Text('Add to cart'),
      onPressed: () => model.addProduct(product),
    );
  },
);

//Shopping Cart
Consumer<ShoppingCartModel>(
  builder: (model) {
    //Obtaining the list of products to display
    final productList = model.getProducts();
    ...  //Rest of the Shopping Cart UI...
  },
);
```

Listing 4.5: Provider example

4.2.3.2 Business Logic Component

The BLoC plays the same role as the Provider package, sitting between the widgets and the data layer. However, it differs from Provider in a few key aspects. Instead of having a unique model class that represents all variants of state, it has multiple state classes (e.g. loading state, error state, complete state), each of them representing one distinct variation. This means that instead of calling direct methods of the model, whenever the widgets want to change the state of the app, they do it by sending events (e.g. AddProduct, LoadProducts). Events are *"the input to a Bloc. They are commonly added in response to user interactions such as button presses or lifecycle events like page loads"* [10]. Finally, there is the actual bloc class, responsible for processing the events and outputting states. This is where the actual business logic and the communication with the data layer happens (Fig. 4.9).

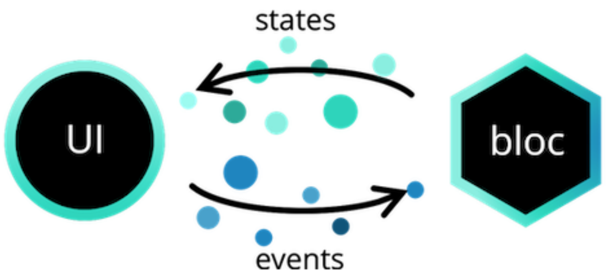


Figure 4.9: BLoC diagram [10]

Instead of relying on observer mechanisms, the BLoC uses streams to introduce these new states to the widgets. For the developer, this does not differ a lot from the Provider alternative as the BLoC package does a good job of abstracting the underlying mechanisms. It looks very similar to the solution from Figure 4.8, where a BlocProvider widget is used to provide the bloc to the widget tree, and then BlocBuilders (the equivalent of Consumers) update the widgets below them when the state changes.

Following the shopping cart example, the subsequent pseudo-code (Lst. 4.6) demonstrates what the BLoC implementation would look like. The shopping cart can have two different states: empty and loaded. There are also two different events that can happen to the shopping cart: adding a product or removing a product. The ShoppingCartBloc holds the actual list of products and modifies it after receiving events. Afterwards, it outputs the new state to the widgets. The BlocProvider and BlocBuilder widgets were purposefully omitted as they work very similarly to the Provider solution.

As demonstrated, this option does have a bigger amount of boilerplate code before it becomes functional. Nonetheless, this was the state management option chosen for our project. Having clear and distinct state classes makes adapting the UI to the states easier and becomes more comprehensible. It also forces us to properly plan ahead what states can the app be in, lowering the chance of unpredictable app states during runtime, increasing the robustness of the overall app.

```
@immutable
abstract class ShoppingCartState {
  const ShoppingCartState();
}

class ShoppingCartEmpty extends ShoppingCartState {
  const ShoppingCartEmpty();
}

class ShoppingCartLoaded extends ShoppingCartState {
  final List<Product> products;
  const ShoppingCartLoaded(this.products);

  bool contains(Product product) {
    return products.contains(product);
  }
}

@immutable
abstract class ShoppingCartEvent {
  const ShoppingCartEvent();
}

class AddProduct extends ShoppingCartEvent {
  final Product product;
  const AddProduct(this.product);
}

class RemoveProduct extends ShoppingCartEvent {
  final Product product;
  const RemoveProduct(this.product);
}

class ShoppingCartBloc extends Bloc<ShoppingCartEvent, ShoppingCartState> {
  final List<Product> _products = [];
  ShoppingCartBloc() : super(const ShoppingCartEmpty()); //Initial state

  @override
  Stream<ShoppingCartState> mapEventToState(  //Map the incoming events to states
    ShoppingCartEvent event,
  ) async* {
    if (event is AddProduct)
      yield* _addProduct(event.product);
    else if (event is RemoveProduct) yield* _removeProduct(event.product);
  }

  Stream<ShoppingCartState> _addProduct(Product product) async* {
    _products.add(product);  //We could add more business logic here, such as duplicate
     validation...
    yield ShoppingCartLoaded(List.from(_products));
  }

  Stream<ShoppingCartState> _removeProduct(Product product) async* {
    _products.remove(product);
    if (_products.isEmpty)
      yield ShoppingCartEmpty();
    else
      yield ShoppingCartLoaded(List.from(_products));
  }
}
```

Listing 4.6: BLoC example

4.2.4 Complete architecture

Deciding on a state management solution does not give us a complete app structure, we still need to determine how the data source layer will be organized and how to integrate it with the BLoC. Afterall, the app needs to handle data coming from multiple sources to show to the user. A good example of a complete flow of information would be the creation of a trip. Preferences and constraints come from user input from the presentation layer, if needed, some business logic happens in the BLoC and then goes to the data layer to be sent to the ROUTE microservice. Afterwards, the data layer receives the trip from the microservice, sends it to the BLoC layer that subsequently sends it to the presentation layer to show the user.

In *Jetpack*, the *Android* team also has an app architecture guide where they recommend dividing the data source layer into repositories and data providers (Fig. 4.10). Within the *Flutter* community, a prominent architecture guide called *Clean Architecture* [72] also uses this repository pattern (Fig. 4.11).

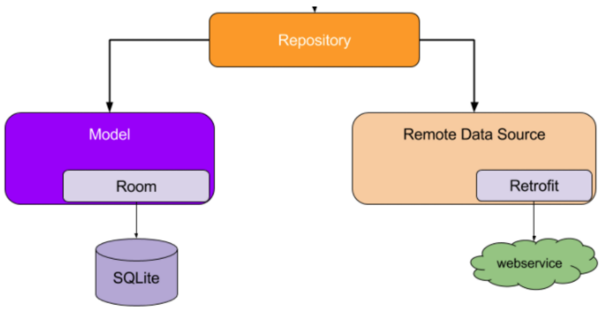


Figure 4.10: Android data layer

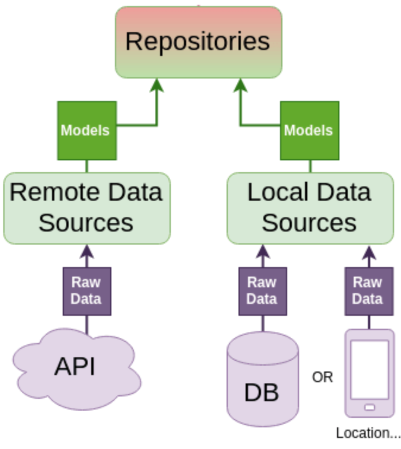


Figure 4.11: Clean architecture data layer

This is the approach we are also taking. In a real life app, the same type of data can come from different sources. We might want to fetch data from the internet (e.g. REST API) but if that is not available, we need to fetch cached data from a local source (e.g. local database). Having a central repository for multiple data providers allows for handling of any necessary logic such as this. Furthermore, the data from providers can come in many formats, so the repository can standardize this data and send it to the BLoC.

Since most of the data sources are external, they are more likely to change over time without our control, which translates into providers being very prone to implementation changes. Any change made to the way a data provider is implemented becomes invisible to the BLoC, since it has an additional repository layer abstracting such details.

This solution implies a lot of dependencies. The BLoCs needs to have instances of the repositories, and the repositories needs to have instances of their data providers. To deal with this issue, both the *Jetpack* and *Clean Architecture* guides recommend using a dependency injection design pattern [71]. Dependency injection is a pattern where an object receives other objects (dependencies) that it depends on. The code that passes the service to the client can be many kinds of things and is called the injector. The "injection"refers to the passing of a dependency (a service) into the object (a client) that would use it. In our case, we will use the *Flutter* package *get_it*[21] to act as a service locator that provides the instances of data providers to the repositories that need it and provides the repositories to the BLoCs that need it.

To conclude, the architecture chosen for our application (Fig. 4.12) has four main components: (1) the UI composed of widgets; (2) the BLoCs responsible for listening to UI events and outputting UI states; (3) the repositories responsible for managing and providing data to the BLoCs; (4) the data providers that fetch (and store) data from local or external sources. The BLoC and repository components have some overlap between the domain layer and their respective presentation and data layers. The dependency flow and call flow are from top to bottom. Each component depends and calls methods from only the component directly below it, becoming completely immune to changes from components on other levels. At the bottom level, the data providers have no dependencies and are oblivious to what is going on in the rest of the app.

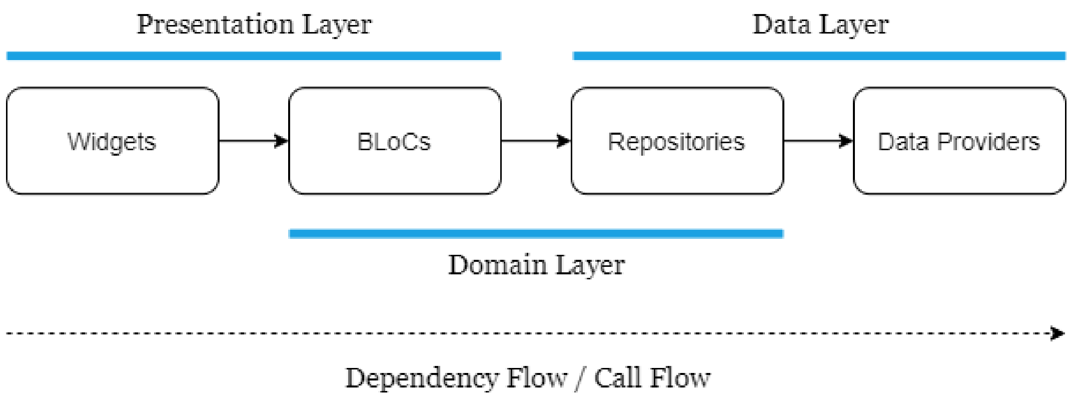


Figure 4.12: Complete architecture

4.3 Functional testing

Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and, conversely, that it does not do anything unintended. *"Software should be predictable and consistent, presenting no surprises to users."* [60]

Testing techniques can vary depending on the the objectives and target metrics, which in turn will result in different levels of specificity (i.e. unit, integration or system) [68]. In our case, the main objective is to perform a functional validation of the app using functional testing, also known as correctness or conformance testing, where test cases are designed to check that the functional specifications of the app are correctly implemented. In order to do so, we developed a series of unit tests and integration tests.

A *unit test* consists in testing a single function, method, or class. The goal of a *unit test* is to verify the correctness of a unit of logic under a variety of conditions[22]. For this reason, external dependencies of the unit under test are generally mocked out. An *integration test* consists in testing a large part or functionality of an app. The goal of a *integration test* is to

[21]`https://pub.dev/packages/get_it`
[22]`https://flutter.dev/docs/testing#unit-tests`

test how individual pieces work together as a whole[23]. *Integration tests* verify the interactions among software components that can not be tested by *unit tests*. In both cases, the general idea is to call parts of code with certain input values and *assert* that the resulting values are what we expect.

Testing is only valuable if the test cases are adequate and extensive. At the same time, the app is big enough that covering all the code and possible scenarios, states and data flows is simply inconceivable. Therefore we have to be selective about what to test. Since the trip creation process is one of the most crucial and code-heavy parts of our app, we decided to focus our tests towards this process and the business logic behind it. Overall, the following unit and integration tests have a code coverage of 40% or 330 lines out of 817 total. The implementation of these tests is available in the *github* repository.

4.3.1 Unit and integration testing

Another great advantage of using a state management solution such as BLoC is that it makes testing much easier. In fact, easier testing is a benefit of most state management solutions. Specifically with BLoC, we know what state our application is in at any point in time, which means we can easily test every case to make sure our app is responding appropriately[24].

By separating the domain layer from the presentation layer, we do not have to depend on the latter to test out the former. This means that we do not resort to subpar and gimmicky techniques, such as having to manually detect the UI components (e.g. buttons) and virtually perform the necessary actions (e.g. tapping or swiping) to emulate a real user, in order to test the business logic of the app. Instead, the business logic is invulnerable to UI changes, so we only need to write the appropriate business logic tests once, regardless of any UI changes that might happen in the future.

The same developers that created the *flutter_bloc* package also created the *bloc_test*[25] package, made specifically for the testing of BLoCs, so this is the solution we adopted for testing our app. Similar to other unit testing frameworks, like jUnit for Java, this package provides a simple mechanism, called *blocTest*, to test blocs against specific test cases. This mechanisms has a few key parameters that we use to carry our test cases: *build* is a function used for BLoC initialization and preparation and must return the BLoC under test; *act* is a callback function used to add events to the bloc; *expect* (also known as *assert* in other frameworks) must contain the states which the bloc is expected to emit after the *act* is executed; *verify* is another callback function for additional verification/assertions. Below (Lst. 4.7) is a simple test case for a counter BLoC.

Note that the following test cases have also been extensively tested out manually, over a real device (OnePlus 7 Pro, *Android* 10), to make sure that there are no bugs that come from the UI or from using the device's GPS adapter, both not covered in these tests.

[23] https://flutter.dev/docs/testing#integration-tests
[24] https://bloclibrary.dev/#/whybloc
[25] https://pub.dev/packages/bloc_test

```
blocTest(
    'emits [1] when CounterEvent.increment is added', //Description of the test case
    build: () => CounterBloc(), //Initialize bloc (state is 0)
    act: (bloc) => bloc.add(CounterEvent.increment), //Add increment event to bloc
    expect: [1], //Assert that bloc state is 1
);
```

Listing 4.7: Bloc test example

Unit tests carried out to test the business logic from the individual steps that comprise the trip creation process:

- Initial state is StartingTrip;
- Test the process of adding an origin location;
- Test the process of adding destination location;
- Test advancing from the Start Trip panel to the Interests panel;
- Test going back to the Start Trip panel from the Interests panel;
- Test adding POIs to the list of interests;
- Test the calculation of the total price of the POIs;
- Test the calculation of the total visit time of the POIs;
- Test the calculation of the average sustainability of the POIs;
- Test advancing from the Interests panel to the Restrictions panel;
- Test going back to the Interests panel from the Restrictions panel;
- Test changing budget restriction;
- Test changing visitation time restriction;
- Test changing effort restriction;

Integration tests carried out that combine and test the previous test cases as a group. The tests simulate the business logic that would occur when a user creates a trip. The resulting JSON structure (see Sec. 3.2.2) is evaluated against the expected result that would be sent to the ROUTE microservice.

- Test the scenario where a user creates a trip from different origin and destination locations, with two POIs added as interests, with a high effort level, 100 euros as budget and 2 hours as visitation time;
- Test the scenario where a user creates a round-trip from a specific location, with no POIs added as interests, with a medium effort level, 50 euros as budget and 1 hour as visitation time;
- Test the scenario where a user creates a trip from different origin and destination locations, with one POI added as interests, but without changing the default restrictions;

Finally, there are the unit tests carried out to test the communication with the ROUTE microservice. Testing with the live ROUTE microservice would cause a few issues, such as slowing down test execution and increasing the difficulty of testing for all outcomes (because it would require configuring the microservice to produce the needed outputs), but most importantly, it would make it so that the results of these tests also depend on the ROUTE microservice rather than depending entirely on the app's code. Instead, we use the *Mockito*[26] package to create a mock version of the ROUTE microservice and have full control over its output, so we can easily test varying scenarios. The idea of these tests is to assert that the app is correctly handling the different scenarios that can happen when communicating with the ROUTE microservice.

- Test the scenario where the communication is successful and ROUTE returns a valid trip;
- Test the scenario where the communication is successful but ROUTE cannot return a valid trip (due to the preferences and constraints of the user);
- Test the scenario where the communication is unsuccessful (e.g. no internet connection);

4.3.2 Functional testing

Besides developing these automatic test cases with the purpose of programmatically testing the business logic that occurs during the process of creating a trip, we have also developed a methodical process for testing the integration between the app and the ROUTE microservice. The idea is to create a set of scenarios that cover both the main use case and edge use cases of the app and ROUTE microservice, therefore simulating a real user. These scenarios were handcrafted along side the student in charge of the ROUTE microservice, so both components of the STC architecture are being tested. The next step is to run each scenario individually as if they are being executed by a real user. For the app, this means manually creating a trip with the right constraints and preferences, sending them to the ROUTE microservice and receiving back the trip, and displaying said trip to the user.

Since the ROUTE microservice does not currently have a live API, the communication between these two components was simulated by exchanging JSON strings *"by hand"* and inputting them into the code where the communication would otherwise happen. Note that on the side of the app, which plays a much easier role in the communication (consisting simply of http requests), this communication was indeed tested with the previously mentioned unit tests.

Overall we created a total of 5 scenarios that we believe test all the important use cases of both the APP and the ROUTE microservice. Below we present these scenarios and explain what they are meant to test. In the Appendix B we include screenshots from the APP executing each scenario and the respective JSON messages exchanged between them.

The first scenario is fairly simple (Tab. 4.2). The user has just arrived at the city by train and wants to go on a trip that ends at the hostel he will be staying at. During this trip he only wants to visit churches and monuments. The second scenario (Tab. 4.3) emulates a user that has just finished dining at a restaurant and wants to go on a short, free trip that ends at the hostel he is staying at. This way, we are testing a trip that spans over two days that has no budget.

The third scenario (Tab. 4.4) is meant to simulate a trip that starts and ends at his hostel, during the rain (relevant to ROUTE), where he wants to specifically visit the *Museu Nacional do Desporto* and a few local stores to buy some souvenirs for his family. This way we are testing a round-trip during bad weather, with a low number of selected POIs. The fourth scenario (Tab. 4.5) is actually design to fail. The user wants to start a trip at 6:30 PM and visit the *Igreja de*

[26] `https://pub.dev/packages/mockito`

Table 4.2: Test scenario #1

Parameters	**Description**
Origin	Train Station (R. 1º de Dezembro)
Destination	Hostel (R. dos Douradores)
Departure Date	2:30 PM
Selected POIs	None
Selected Categories	Churches, Monuments
Budget	50 €
Visitation Time	5 hours
Effort	Medium

Table 4.3: Test scenario #2

Parameters	**Description**
Origin	Largo Picadeiro (Restaurant)
Destination	Hostel (R. dos Douradores)
Departure Date	10:00 PM
Selected POIs	None
Selected Categories	Viewpoints, Squares
Budget	0 €
Visitation Time	3 hours
Effort	Easy

Santo António and *Miradouro do Recolhimento*. While both POIs are open at that time, meaning the app allows the user to select them, it is impossible for the user to visit both of them before they close. With this scenario we test a failure case due to bad constraints, and setting the origin as the current location of the user.

Table 4.4: Test scenario #3

Parameters	**Description**
Origin	Hostel (R. dos Douradores)
Destination	Hostel (R. dos Douradores)
Departure Date	10:00 AM
Selected POIs	Museu Nacional do Desporto
Selected Categories	Local Stores, Museums
Budget	50 €
Visitation Time	6 hours
Effort	Medium

Table 4.5: Test scenario #4

Parameters	**Description**
Origin	Current location (R. Garret)
Destination	Appointment Local (R. Vítor Cordon)
Departure Date	6:30 PM
Selected POIs	Igreja de Santo António, Miradouro do Recolhimento
Selected Categories	None
Budget	10 €
Visitation Time	1:30 hours
Effort	Hard

Finally, the fifth scenario (Tab. 4.6) is meant to simulate a gastronomic trip where the user goes from restaurant to restaurant trying out the Portuguese cuisine. This test case is more useful for the ROUTE microservice, which is being tested against a large set of POIs and a tight set of constraints.

Table 4.6: Test scenario #5

Parameters	**Description**
Origin	Campo das Cebolas
Destination	Hostel (R. dos Douradores)
Departure Date	12:00 PM
Selected POIs	Nicola Café, Casa Portuguesa do Pastel de Bacalhau, Martinho da Arcada, As Bifanas do Afonso, A Brasileira
Selected Categories	All
Budget	70 €
Visitation Time	5 hours
Effort	Medium

4.4 Bluetooth detection service

As mentioned in section 3.2.3, the app is also responsible for providing crowding updates to the HEAT microservice. It does that by detecting nearby Bluetooth devices and combining the results with a location and timestamp reference. This functionality comes in the form of a service that runs within the app, making use of the device's Bluetooth capabilities.

4.4.1 Bluetooth in smartphones

In 2010, a new type of Bluetooth technology called Bluetooth Low Energy (BLE) was announced [23]. This technology, also called Bluetooth Smart or Bluetooth 4.0 (now Bluetooth 5), has been adopted by smartphone manufacturers at different rates. This also caused fragmentation in the Bluetooth frameworks provided by mobile operating systems. In fact, *iOS* native Bluetooth APIs do not properly support classic Bluetooth. Core Bluetooth[27] is the primary Bluetooth framework offered by *iOS* and it is solely for BLE devices. External Accessory[28] is another *iOS* framework for classic devices, but only for specific MFi compliant devices, i.e., devices specifically designed to interface with an *iOS* application. To deal with this fragmentation, the service makes use of two *Flutter* packages, *flutter_blue* and *flutter_bluetooth_serial*, that abstract the interaction with the operating systems' native Bluetooth APIs. This way, we can make use of both Bluetooth technologies, classic and BLE, in a uniform and simple manner.

4.4.2 Design and implementation

The service is designed to be as independent as possible so that it does not rely on or interfere with the natural flow of the app. In fact, since it requires no user input, we decided to make this service completely invisible to the user. Its only dependencies are the necessary data providers (i.e. GPS and Bluetooth) and the app lifecycle itself. As a user navigates through, out of, and back to the app, it transitions through different states in its lifecycle. The service needs to observe these states (i.e., inactive, paused, suspending and resumed) so that it can stop and resume appropriately.

It is also built with failure in mind. When thinking about the usage context of the app it is likely that all requirements that make the service work are not always met. Internet connection is not always present, GPS and Bluetooth functionalities are not always on and their permissions may not be granted. To counteract these issues, the service uses a timer-based approach, such that in case of failure, it will not crash or stop, but simply try again after a fixed time amount. Additionally, if no internet connection is available the results are stored locally and sent on the attempt, and if any permissions are missing the app will display a prompt asking for permissions.

This timer-based approach was also adopted for a different reason. Bluetooth scanning has a significant impact on the device's battery, especially on devices that are not BLE [23]. Since trips can last a couple of hours, we have to be particularly mindful of the device batteries. It is not viable to be constantly scanning for devices, so a fixed duration of fifteen minutes was

[27] `https://developer.apple.com/documentation/corebluetooth`
[28] `https://developer.apple.com/documentation/externalaccessory`

decided for the timer, as it replicates the time between updates from the SNIFF microservice to the HEAT microservice.

The service workflow, represented by an UML activity diagram in Figure 4.13, can be described as follows. When the service starts (by app startup or a resumed lifecycle event) it checks the timestamp of the last scan. If the time passed since the last timestamp is equal or bigger to fifteen minutes, it immediately starts a new scan. Otherwise, it awaits the time difference before starting. Once the scan is finished, the service checks if there are any additional results pending, and sends those as well to the HEAT microservice. If no internet connection is available, the service stores the results locally. Finally, the service schedules a new scan in fifteen minutes. If during this process an error occurred due to previously mentioned circumstances, the service will simply skip to the scheduling part.

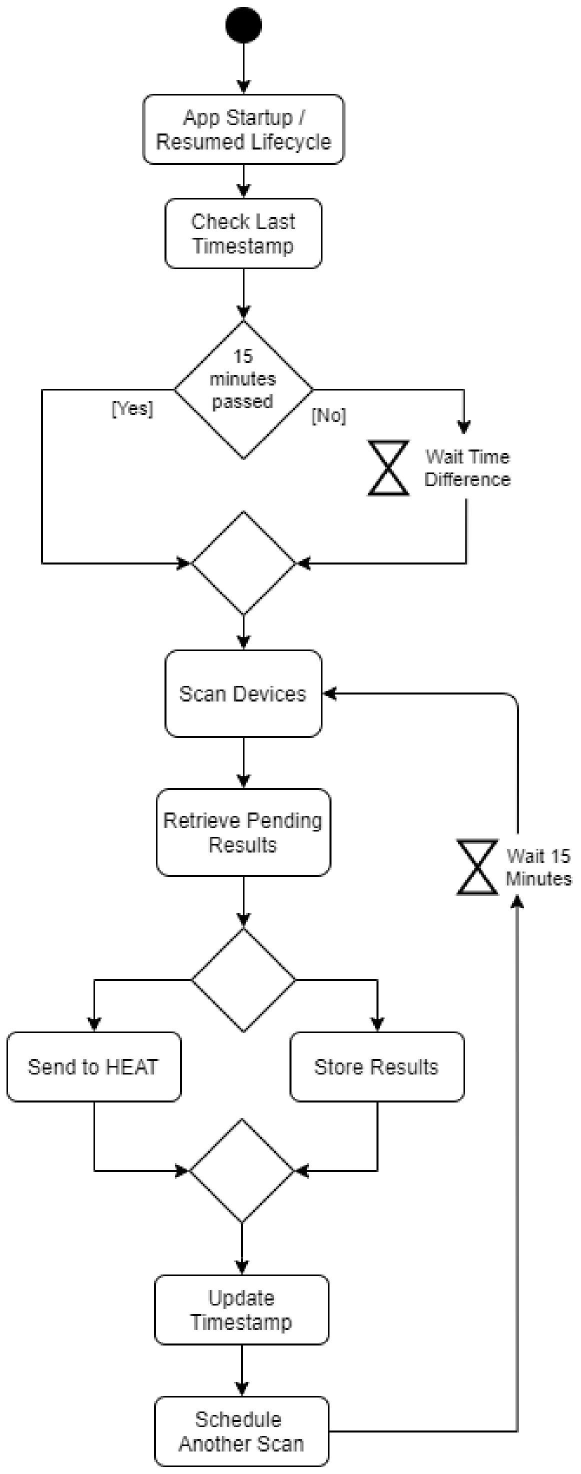


Figure 4.13: Bluetooth detection service diagram

[This page has been intentionally left blank]

Chapter 5.

Application Design

Contents

In section 5.1 we describe the main features of the app prototype, along side a usage scenario that shows the way we envision how tourists, our main target audience, will use the app. Section 5.2 goes over the design process, the different iterations of the UI and its validation. Additionally, we present the final UI mockups with detailed explanations for each of screen/feature. Finally, in section 5.3 we discuss further design considerations taken for the app prototype.

[This page has been intentionally left blank]

Chapter 5
Application Design

5.1 Application description

To better describe the functional requirements and to demonstrate the main use cases of the app, we have created a scenario that shows the way we envision how tourists, our main target audience, would use the app, while justifying the reasons behind some of the main features in the app.

Dinis, the tourist, is staying in a hotel at the *Santa Maria Maior* perish for a week. While riding a *Tuk Tuk*, a popular tourist activity in Lisbon, he encounters an advertisement with a QR code, which is part of a promotional campaign for our app. Dinis becomes intrigued and decides to scan this QR code which redirects him to the app's app store page on his device's default app marketplace. From there, he decides to download the app.

After his ride, Dinis decides to launch the app so that he can explore the the *Santa Maria Maior* perish. The app uses Dinis' current location to center him on a map of the city, which also contains a list of markers representing the city's POIs. Dinis can pan and drag the map freely to explore the city as he pleases, and he look up more information on any POI or discover the name of any street. Furthermore, Dinis can manually search for locations (e.g. the hotel he is staying) by typing an address, which the app will Geocode and then display its location on the map.

The following day Dinis decides that he wants to spend his afternoon visiting the city. To do so, he has approximately 5 hours and 100 euros available. Through the mobile application, Dinis plans a trip to depart immediately from his current location and conclude at his hotel room. He has heard of an interesting exhibition taking place at the Museu Nacional do Desporto, so he explicitly tells the application that he wants to visit this point of interest. Additionally, he specifies which categories of POIs he is interested in: museums, churches and monuments. Optionally, Dinis can change the effort level of his trip and the system will adjust the physical effort of the route accordingly. The system will use these constraints and preferences to create a trip that maximizes sustainability and reduces crowding by recommending the right POIs. This trip is shown to Dinis both in the map and in the form of a timeline, which he can use to confirm or reject it. After the trip starts Dinis can consult the app for guidance and information about the route and POIs he is visiting. If Dinis ever ends up far away from the planned route, the app will provide him the option of recalculating a new route, now starting from his current position.

After a few days using the app, Dinis decides he wants to check his activity. The app keeps track of a few statistics (i.e. POIs visited, distance travelled and average sustainability of his trips) and a list of all the completed trips and planned future trips. While looking at his past trips, Dinis finds a particularly interesting one that he wants to share with his friends. Knowing that one of his friends is also visiting Lisbon the next month, he uses the *sharing* functionality as a form of introduction and recommendation of the app to his friend, so that he can also use it in his stay in Lisbon.

In addition to this scenario, we include a use case diagram (Fig. 5.1) with the purpose of highlighting and summarizing the main functionalities of our app and showing how the user interacts with them.

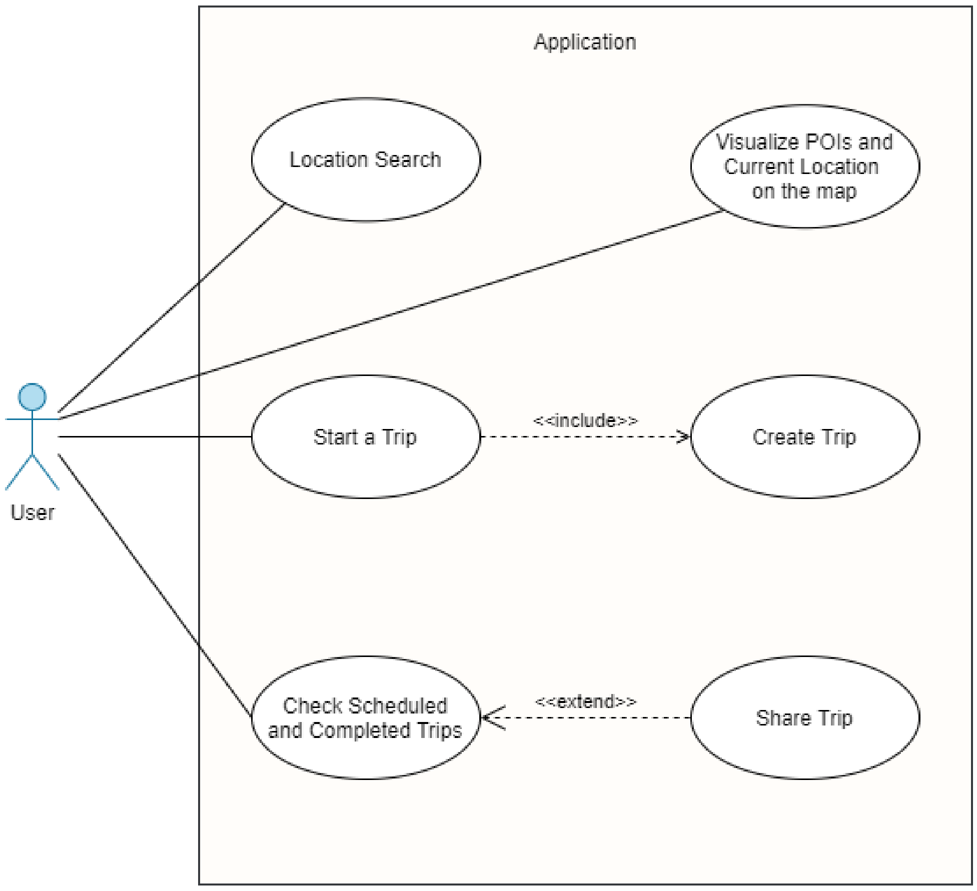


Figure 5.1: Application use case diagram

5.2 User centered design and UI validation

User-centered design (UCD) is a broad term to describe design processes in which end-users influence how a design takes shape. It is both a broad philosophy and variety of concrete methods and techniques [2]. UCD provides a top-down approach for creating websites and apps according to the users' needs, by placing the user at the center of the design. The role of the designer is to facilitate the task for the user and to make sure that the user is able to make use of the product as intended and with a minimum effort to learn how to use it. It not only focuses on understanding the users but also requires an understanding of the tasks that users will perform with the system and of the environment (organizational, social, and physical) in which they will use the system.

There are many different ways to involve the user in the design [2]. Each technique has a stage of the design cycle in which it is most appropriate. For example, background interviews and questionnaires, for the purpose of collecting data related to the needs and expectations of users, is more appropriate at the beginning of the design process. In our case, we carried out two techniques, one named *Card Sorting* and another named *Tree testing*, during the early stages of the design process. The focus of these techniques is on developing and subsequently evaluating the Information Architecture (IA) of our app.

Information architecture is the art/science of structural designing a labeled hierarchy for the information in an information system [74], just like our app. Establishing an IA is an integral part of the design process, since it *forms a skeleton of the product. Visual elements, functionality, interaction, and navigation are built according to the information architecture principles*[1]. Consequently, we decided that it is best to employ a user-centered process for the design of the IA of our app.

To create and carry out these usability techniques we used Optimal Workshop[2]. It is an online platform that offers solutions within the area of user research, UX design and information architecture. This tool contains support for creating, sharing and completing both the card sorting and tree testing techniques. Additionally, it also includes a powerful data analysis tool specialized for each technique.

Altogether, we conducted two instances of these techniques. The first one, supposed to target a smaller audience, was conducted with colleagues and members of the Software Systems Engineering group at ISCTE. The second one was planned to be conducted in collaboration with APECATE, an association of tourist related businesses and partner of the STC project, due to their experience and expertise in the subject of tourism. APECATE has a large number of members, so we expected to get a high number of participants. Unfortunately, due to the Covid-19 situation, APECATE was unavailable to collaborate, so we resorted to another partner of the STC project, the *Junta de Freguesia de Santa Maria Maior*, which also has some knowledge in the subject of tourism. The idea is to develop an iterative process where we use the feedback from the previous variation to create a new and improved version, in order to obtain better results.

5.2.1 Card sorting

Card sorting is a widely known and accepted technique that UI/UX designers, and more specifically information architects, employ throughout the information design phase. *"It is a great, reliable, inexpensive method for finding patterns in how users would expect to find content or functionality."* [83].

It is a user-centered design method that actively involves the target user groups by asking them to sort a series of cards, each labeled with a piece of content or functionality, into groups that make sense to them. It takes in real user input, not the gut feeling or strong opinions of a designer. There are a few variants of this technique, like the *open* card sort, where participants are asked not only to sort cards into groups but to also name them appropriately.

The goal of this technique is to *"gain insight into users' mental models, illuminating the way that they often tacitly group, sort and label tasks and content within their own heads"* [83]. Card sorting is, therefore, a *generative* and *formative* technique, and not a evaluation technique to

[1] https://www.optimalworkshop.com/solutions/information-architecture/
[2] https://www.optimalworkshop.com/

determine what is wrong with an already designed website or app. It is best used as an input to designing or re-designing a website/app or a part of it. Accordingly, we used this technique during the early stages of designing the information architecture of our app.

The Optimal Workshop's card sorting platform includes an analytics tool that allows us to better interpret the results from the participants. From the several different analysis segments provided by this tool, perhaps the most important ones are the simpler *cards* and *categories* segments and the more complex *dendograms* segment.

Using the *cards* analysis we can observe how each card is categorized and identify where the participants agree, where they have different ideas and potentially if there were card labels that they did not understand. The *categories* analysis shows how the participants created and named categories in order to understand the different language that the participants used and whether these categories contain similar cards.

The *dendrograms* analysis (Fig. 5.2) is useful for understanding which card groupings have the strongest agreement. Cards are listed down the left-hand side of the dendrogram, while the axis along the top measures the level of agreement across participants. Clusters closer to the left indicate that more participants agreed with this grouping.

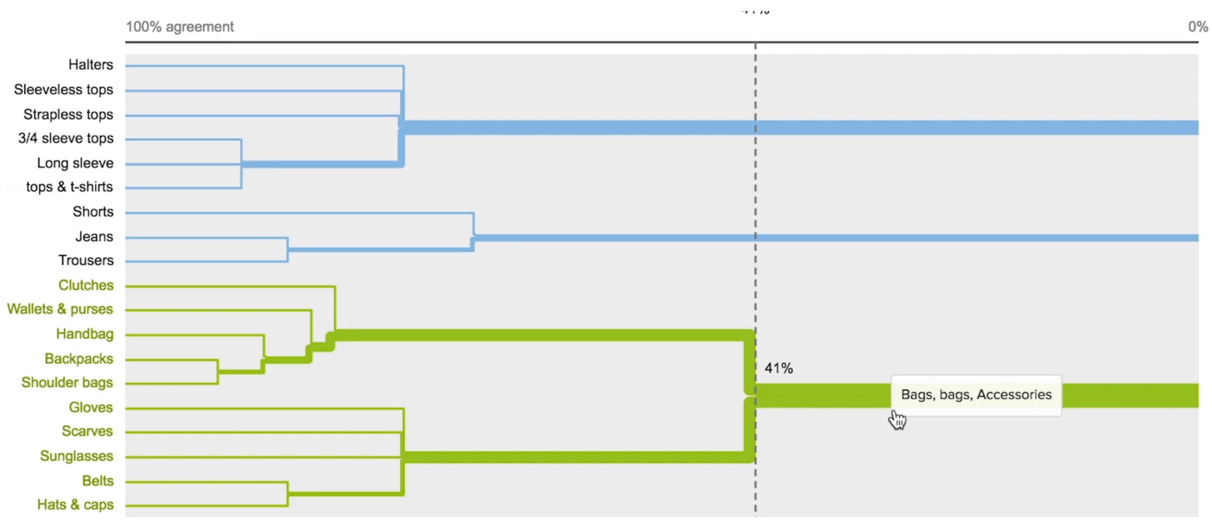


Figure 5.2: Dendrogram example for a clothing website card sort

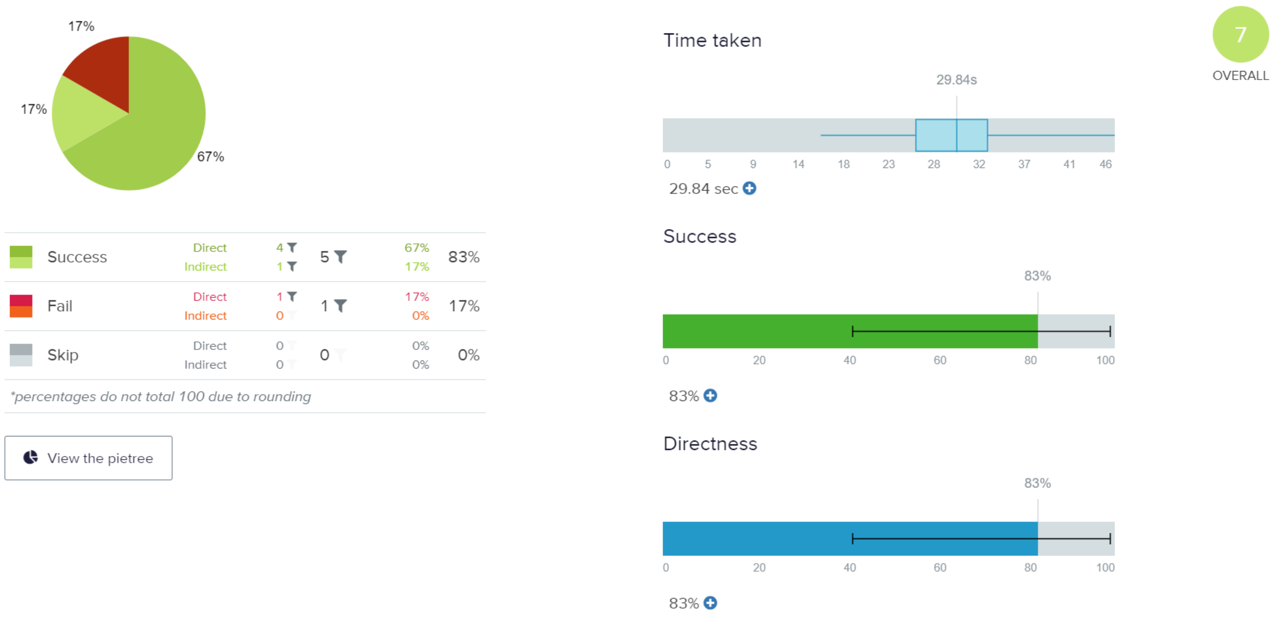
5.2.2 Tree testing

Tree testing is another UI/UX technique pioneered by the same expert that created card sorts, Donna Spencer. It is usually coupled with card sorting techniques to further gain an understanding of how to design an appropriate information architecture for the target users. More specifically, it is useful for *"understanding where people are currently getting lost and how they expect to look for key information on your website"* [93].

Tree testing has two main elements: the tree, and the tasks. The tree is a text-only version of the website/app structure. The tasks consist of finding certain items of information, which are the *"child nodes"* below category items (*"parent nodes"*), inside that tree. Participants are asked to complete these tasks by navigating through the tree and nominating the information they think is correct. It is another user-centered design method that actively involves the target user groups, which allows designers to get real, objective evaluation of the information architecture they designs.

The goal of this technique is to *"find out if the labels and structure of your information on your website, intranet, or product is easy to understand"* [93]. It is a *validative* technique that produces valuable insights for the validation of the IA of an app or website. It is also very flexible, since it can be used on websites or apps with structures ranging from small to large and it can be executed as one large test or split into multiple smaller tests.

The Optimal Workshop's tree testing platform also includes an analytics tool specialized for this type of technique. Using this tool look at each task one by one (Fig. 5.3) and understand how many people got it right, the paths people took before they selected an answer, and how long it took people to complete the task. It attributes an overall score to each task, from a scale of 1 to 10, using the *Success*, *Directness* and *Time taken* measurements.



Success	Direct Indirect	4 1	5	67% 17%	83%
Fail	Direct Indirect	1 0	1	17% 0%	17%
Skip	Direct Indirect	0 0	0	0% 0%	0%

*percentages do not total 100 due to rounding

Figure 5.3: Tree test task analysis example

5.2.3 First iteration results

The card sorting technique had a total of 10 participants, members of the Software Systems Engineering group at ISCTE. The majority of these participants are in the 45-54 year range (82%), have professional experience in the IT area (91%) and a masters degree (64%). The technique took an average of 10 minutes to complete.

In this technique, we asked participants to group a total of 16 cards, and then label said groups. The cards, listed below and translated from Portuguese into English, contain pieces of content and functionality mentioned in section 5.1:

- Origin and destination points;
- Date and time of departure;
- Difficulty and route effort (e.g. calories);
- Budget;

- Categories of interest (e.g. churches, parks, etc.);
- Timeline of the POIs to visit;
- Interactive city map;
- Current Location;
- List of city's POIs;
- Share previous trips;
- Statistics of completed trips;
- Point of interest name and address;
- Attraction image;
- Place details (e.g. visit time, entry price, sustainability rating);
- Location Search;
- Reuse shared trips;

Almost all the participants (90%) decided to group the cards into four categories. Looking at the dendrogram (Fig. 5.4) it is possible to discover which cards the participants most often grouped together. At 80% agreement level, four category clusters take shape. Note that the analysis tool offers two different dendrograms, one that depicts only absolutely factual relationships (recommended for 30 answers and higher), and another that makes assumptions about larger clusters (recommended for fewer participants). The one we used to interpret our results is the latter.

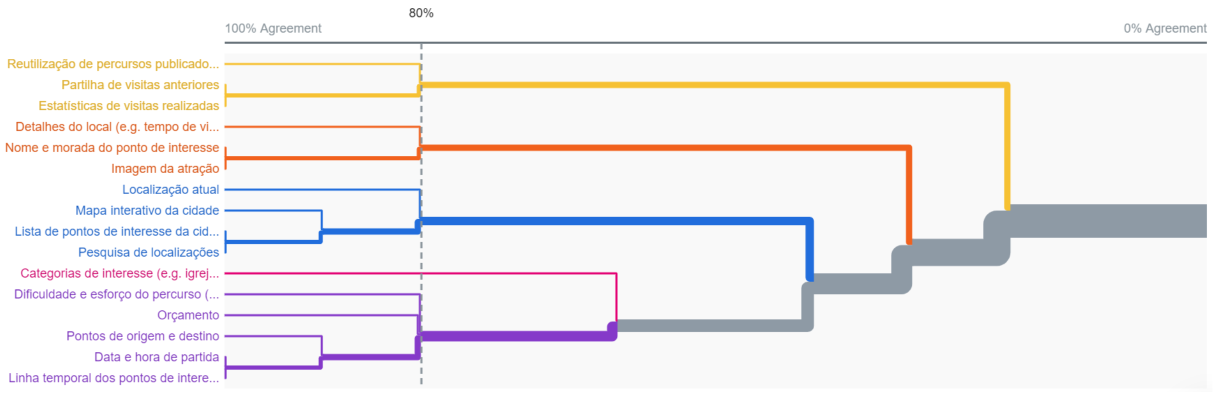


Figure 5.4: Dendrogram for the first card sort

The first group has been labelled with names such as *Trip History*, *History and Sharing*, and *Share*. For the second group, common labels are *Point of Interest* and *Attraction Detail*. In the third group the most common labels are some variation of *Map*. The final group has been labelled with names such as *Create Trip* and *Itinerary*.

Direct feedback from the participants made us realize that perhaps the context and goal of this technique was not clear enough to the participants. This is something that was improved on the next iteration.

The tree testing technique also had a total of 10 participants from the Software Systems Engineering group. In this technique, we asked participants to solve 3 tasks, by navigating through an information tree (Fig. 5.5), which is based on the results from the card sort.

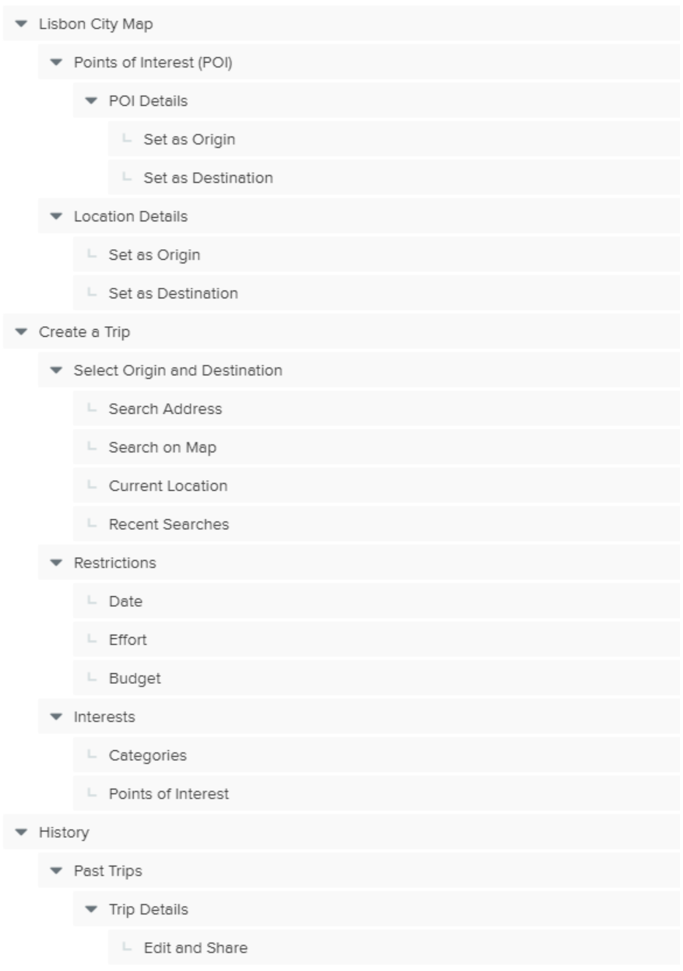


Figure 5.5: Tree of the first tree test

The tasks, listed below and translated from Portuguese into English, were written as hypothetical scenarios and cautiously worded as to not include he same language that was used in the tree, to prevent the participants from matching the phrases rather than actually deciding if the information is correct [93]:

- Suppose you only have 50€ to travel. Where would you add this limitation?
- Imagine you want to start a trip departing from the nearest POI from your location. Demonstrate how you would do this.
- Imagine you recently finished a trip and want to share it with your friends on social network. Go to the spot where you will find this functionality.

The results from this test are fairly positive. The first task has an overall score of 9 (out of 10). The second task is rated at a 7. The third and final task has a rating of 8. In the Appendix D we present a more detailed version of these results, that includes the exact number of direct and indirect successes, the number of direct and indirect failures, as well as the *directness* metric of each task and the time taken to solve them. Regarding this particular tree test, no feedback was received from the participants.

5.2.4 Second iteration results

The second version of the card sorting test shared with the *Junta de Freguesia de Santa Maria Maior* resulted in 11 participants. The majority of these participants claim to have a bachelors degree (64%) and previous professional experience in the IT area (73%), with ages fairly evenly distributed over the 18-24 year range to the 55-64 year range. The technique took an average of around 5 minutes to complete.

Due to the previous feedback from the first iteration of the card sort, the introductory message was improved to include a brief summary of the STC project and to better explain the goal of a card sort. Furthermore, a few card were altered to better reflect the content of the app and to make sure the participants understand the concepts behind them clearly. This card sort had a total of 17 cards, listed below:

- Attraction details (e.g. visit time, price);
- Budget Restrictions;
- Completed trips statistics (e.g. attractions visited, total distance);
- Current location;
- Images of the tourist spot;
- Interactive city map;
- List of planned, current and completed trips;
- Location of city's points of interest;
- Location search;
- Origin and Destination Selection;
- Point of interest name and address;
- Reuse shared trips from friends;
- Select Departure Time;
- Share completed trips;
- Travel interests (e.g. churches, parks);
- Trip difficulty and effort (e.g. slope, calories);
- Trip timeline;

Every participant decided to group the cards into three categories. Looking at the dendrogram (Fig. 5.6) it is possible to discover which cards the participants most often grouped together. At 73% agreement level, three category clusters take shape. Again, we used the *Best Merge* method that makes assumptions about larger clusters based on individual pair relationships.

The first group, represented in green, is similar to the first group of the previous iteration, also labelled with variations of either *History* and *Sharing*. The second group, represented in red, is also equivalent to the last group of the first iteration, and is also labelled with variations of *Trip Creation*. Finally, the third group, highlighted in blue, contains the cards from both the *Point of Interest* and *Map* group, that were present in the first iteration. In this version, this group is labelled with variations of *City Map*.

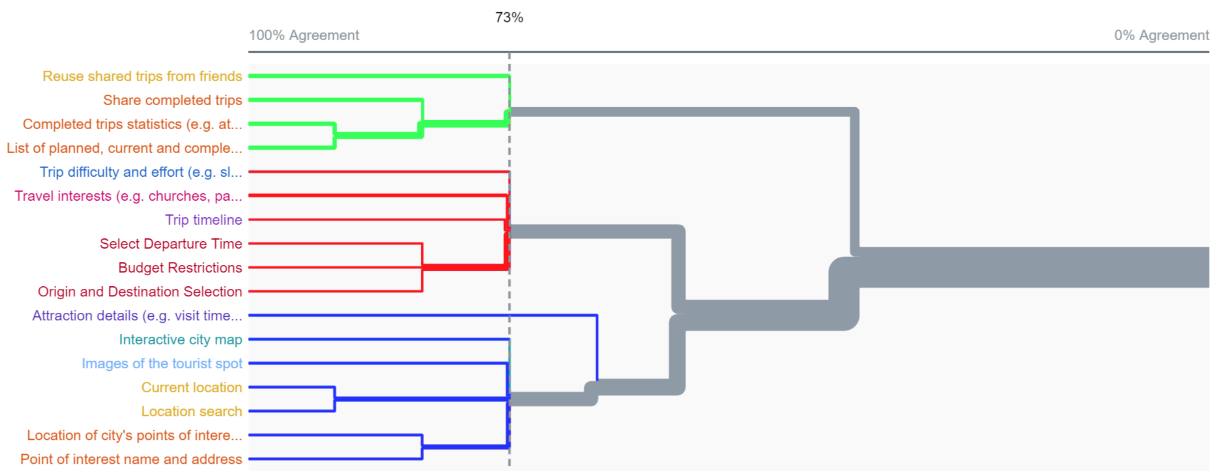


Figure 5.6: Dendrogram of the final card sort

The second iteration of the tree testing technique also had a total of 11 participants from the *Junta de Freguesia de Santa Maria Maior*, sharing the same demographic characteristics as shown for the card sort technique.

In this iteration, the information tree has been simplified (Fig. 5.7). The old tree had a lot of similar child nodes that were not interesting to test (e.g. the distinction between an *Effort* constraint and a *Budget* constraint), which were removed in this iteration. Additionally, the number of tasks was increased to 6, in order to increase the path coverage of the tree, so that each important node is at least targeted once. The tasks used in this tree test are the following:

- Imagine you recently finished a trip and want to share it with your friends on social network. Go to the spot where you will find this functionality,
- Suppose you want to check your planned trip for tomorrow. Navigate to the place you would expect to find this information.
- You are a tourist that wants to go on a trip that ends at the place you are currently accommodated. Where would you add this restriction?
- Imagine you want to start a new trip that visits only churches. Where would you add this limitation.
- Suppose you want to check the price and visitation time of the *Museu do Fado* POI. Navigate to the place you would expect to find this information.
- Imagine that you want to check the total number of attractions you have visited using the app during your stay in Lisbon. Navigate to the place you would expect to find this information.

Once again, the results from the tree test are positive, validating the information architecture prototype. Overall, the tasks have an average score of 8 out of 10. In fact, every task has a score of 8, with the exception of task #4, which has a score of 7 out of 10. In the Appendix D we present a more detailed version of these results, that includes the exact number of direct and indirect successes, the number of direct and indirect failures, as well as the *directness* metric of each task and the time taken to solve them.

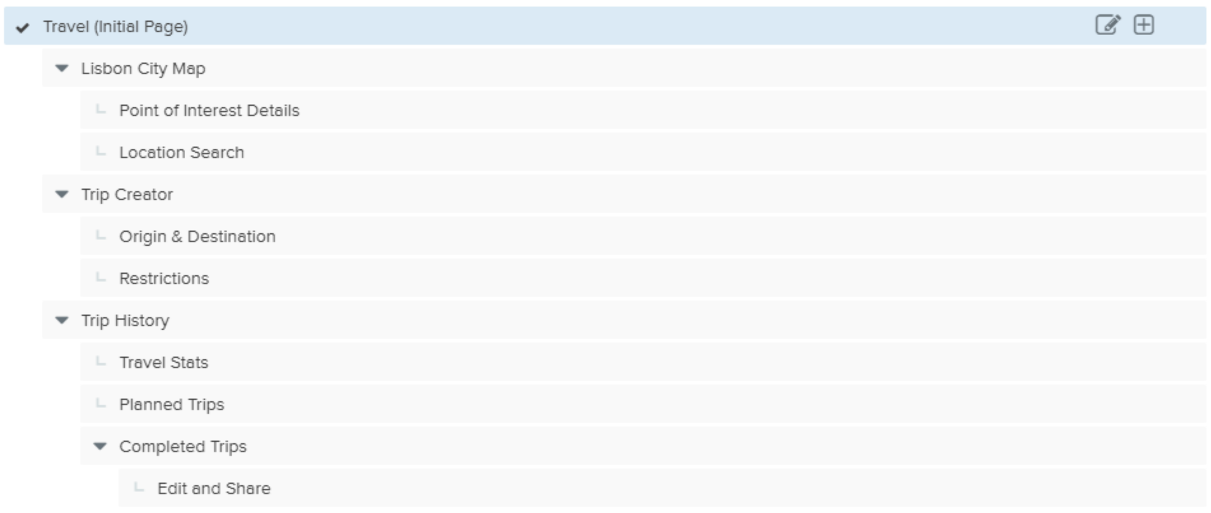


Figure 5.7: Tree of the second tree test

5.2.5 Validity threats

The techniques employed are subject to a number of external validity threats [92], which condition the ability to generalize results to the overall tourist population. While the subject population (i.e. the participants) have some experience in the subject of tourism, they are not representative of the population we want to generalize to (i.e. tourists). Additionally, both iterations had a small population size of 10 participants and 11 participants respectively, which again makes it so that the results may not reflect the population correctly. Finally, the results also suffer from a construct validity threat [92], since the techniques use a conceptual representations of the UI that is different than what end users will see (i.e. cards in the card sorting technique and an information tree in the tree testing technique), therefore making it harder to generalize these results to the actual developed UI prototype. Asking participants to perform the sames tasks from the tree testing technique on the actual user interface, specially in a touristic setting, might have led to different tasks scores.

5.2.6 UI prototype

In both iterations of the card sort, participants grouped cards into a *History/Sharing* group. As the name suggests, this group contains the functionalities and content pertaining the travel activity of the user, such as displaying travel statistics and the list of past and planned trips, and the sharing of said trips. These elements were grouped in the UI and labelled as *Trip History*.

Another group that is found in both iterations of the card sort is the *Trip Creation* group, as labelled by participants. In this group, the functionalities and content of the trip creation process can be found, such as selecting the origin and destination, as well as the content of guiding the user through the trip, such as displaying the trip timeline.

Finally, there are the cards pertaining to the exploration of the city, such as location search, display the city's POIs as markers on the map, and consulting information about POIs. In the first iteration of the card sort, there is a specific group for the POI related cards, while in the second iteration this cards are grouped with the rest. For the development of the prototype UI, it was considered that these cards are closely related and should be grouped together (e.g. the location search functionality can be used to search for a POI, to obtain more information about it). The previously presented use case diagram (Fig. 5.1) can now be organized in these three categories, as presented in Figure 5.8.

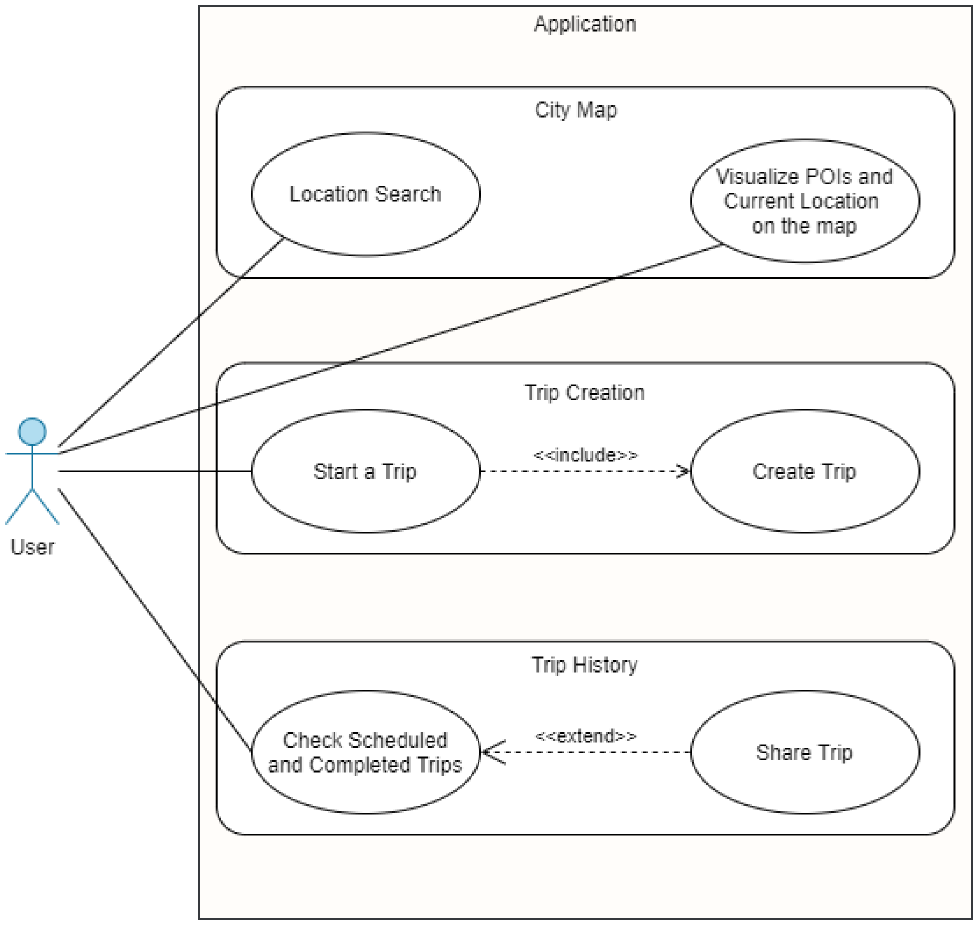


Figure 5.8: Application organized use case diagram

To develop the UI prototype we used *Figma*[3]. Figma is a collaborative interface design tool that allows for the creation of designs and even interactive mobile prototypes. Through Figma, it was possible to quickly iterate between various versions of the UI, before beginning the development stage. Below, we explain in more detail how each of these groups were designed and include the respective UI mockups.

5.2.6.1 City Map

The *City Map* (Fig. 5.9) is heavily inspired by the *Explorer Mode* found in the *Dynamic Tour Guide* [51]. In this mode, which is the default and initial state of the app, the application takes a step back from being a typical "tour guide" in the sense that it does not provide any type of recommendations. Instead, its purpose is to simple function as an enhanced digital map, by displaying the user's current location along with the location of the city's POIs, on a map. The

[3]https://www.figma.com/

map is fully interactive, supporting all the modern gestures one would expect, such as zooming, dragging and panning.

Additionally, this mode also provides *Location Search* functionality, in which users can manually type a street address and the app will show the corresponding location on the map (i.e. Geocoding), or they can select any location on the map (including POIs) and the app will show the respective street address to the user (i.e. Reverse-Geocoding). This is particularly useful if the user wants to search for a specific location, such as the hotel he is staying in or a POI he is interested in.

Search functionality is known to be a usability challenge [88]. Particularly on mobile apps, requiring the user to type on the virtual keyboards is not ideal. It is a time-consuming task that demands a lot of effort from the user, and the small "keys" often lead to typos and errors. To help alleviate this burden, in addition to a simple textfield where the user can type addresses, we also provide a few extra elements: a set of shortcut actions (*Current Location* and *Choose on Map*) and a list of recent searches. The app saves the most recent searches made by the user so that he does not have to type these addresses again. This is especially helpful if the address is a very long string or if the user is repeatedly searching for a specific location (e.g. his hotel).

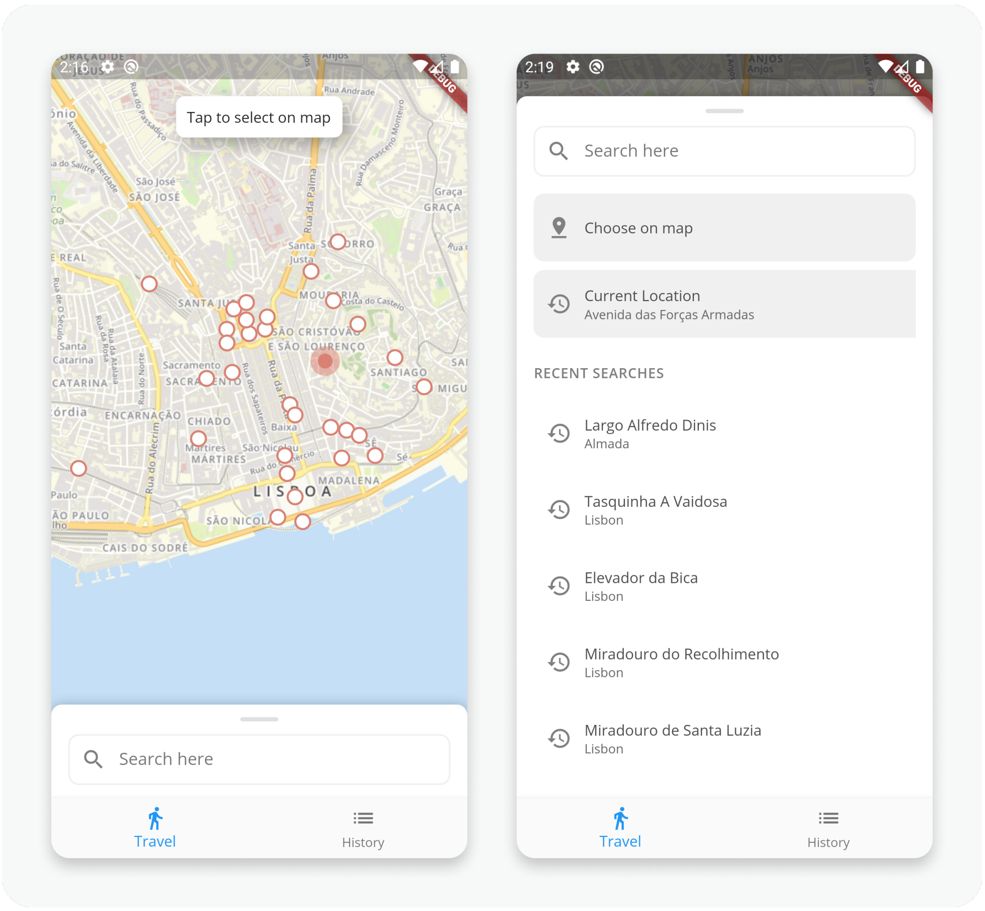


Figure 5.9: City Map UI

With the *City Map* we hope to also appeal to the tourists that prefer to explore a city on their own over being told what to visit. Furthermore, other well-known commercial mapping apps such as Google Maps also provide this type of functionality, which makes it important to include in our app, so that our users don't have the need to alternate between a mapping app and our MTG to fulfill their travelling needs. We took the decision of making this the initial and default mode of the app because we do not expect that users will commit to using the app to create and follow a trip right after they download and try it for the first time. When there is no trip being created or followed, we use the *City Map* to fill the gap until the user decides to do so.

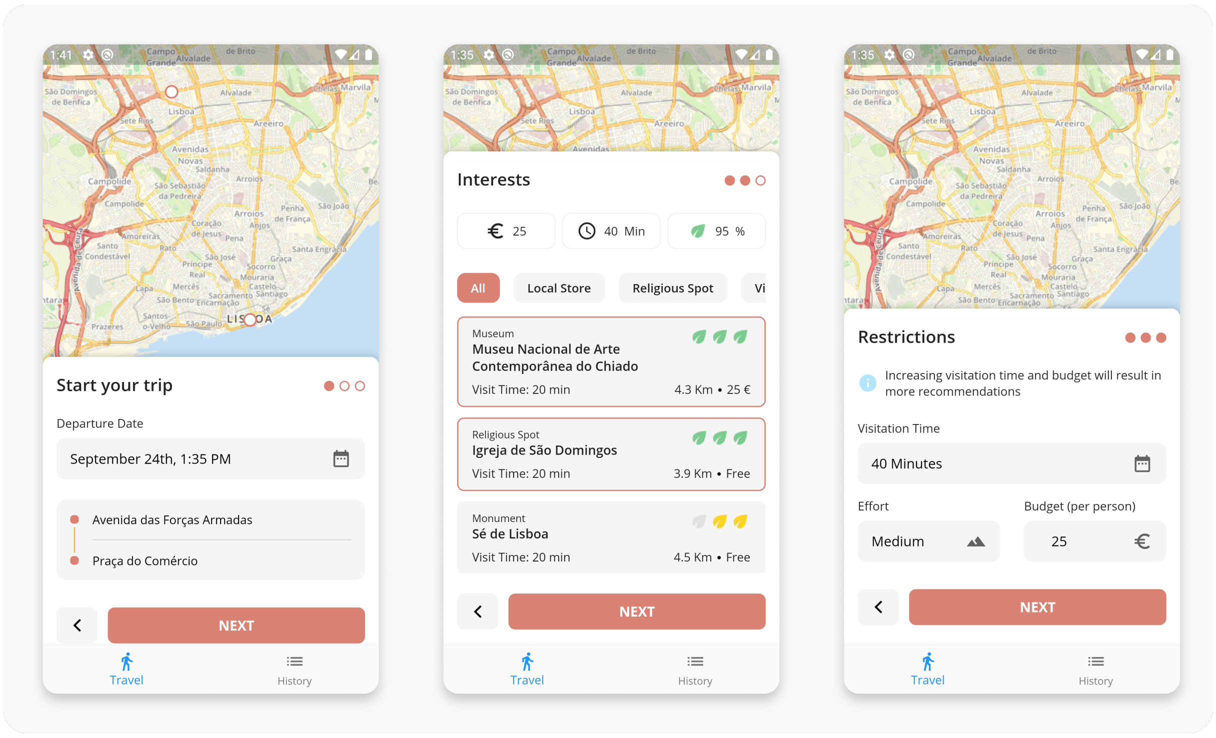
5.2.6.2 Trip Creation

Trip Creation contains the core functionalities of the app. The main goal of the STC project is to improve the sustainability of tourism and reduce crowding in Lisbon. We achieve that by guiding tourists through dynamically generated routes and POIs that suit the city's interest the best. This is the philosophy that the app was designed around.

Trip Creation can be summarized into two main use cases. The first being the process of creating a trip and the second one being the process of guiding the user through the trip. In order for the ROUTE microservice to generate a trip, we must first gather the necessary information from the user, that is, the preferences and constraints of the tourist.

A common best-practice when designing user interfaces is to decompose large tasks into smaller action steps [77]. This way, the user has a more accurate perception of the progress he has made so far and how much is left to do. Additionally, by splitting up the task into a sequence of chunks, each of which can be dealt with in a discrete "mental space" by the user, the task becomes effectively simpler [87]. The trip creation process is a prime candidate for applying this practice, so we decided to divide it into three smaller steps (Fig. 5.10). In the initial stage, the user inputs the departure date and the origin (defaults to the user's current location) and destination locations. Since the process of choosing a location is similar to the *Location Search* feature from *City Map*, we present the user with the same UI components he is already familiar with. The next step is to gather the interest of the users. We present a list of possible POIs and categories from which he can select. The final step is to collect the constraints of the user, i.e. the budget, available time and effort level. Concretely, this process is more complex than what is described in this summary, therefore, we also provide a more detailed use case description in Appendix C that includes some alternative cases.

The process of guiding the user through the trip takes a very different approach. Instead of demanding the user for inputs and attention, this time, the application plays a more supportive and passive role. The route of the trip, the location of the POIs and the user's current location are all displayed on the map. Furthermore, if the user wants more information he can expand the bottom panel, to visualize a more detailed trip overview (Fig. 5.11). This trip overview is presented in the style of a timeline, listing all of the POIs to visit in order and with a timestamp of the expected arrival time. If the user deviates from the intended route, the app will present the user with an option to recalculate a new route starting from his current location, so that he does not feel lost in the city.


Start your trip

Departure Date

September 24th, 1:35 PM

Avenida das Forças Armadas

Praça do Comércio

NEXT

Interests

25 | 40 Min | 95 %

All | Local Store | Religious Spot | Vi

Museum
Museu Nacional de Arte Contemporânea do Chiado
Visit Time: 20 min — 4.3 Km • 25 €

Religious Spot
Igreja de São Domingos
Visit Time: 20 min — 3.9 Km • Free

Monument
Sé de Lisboa
Visit Time: 20 min — 4.5 Km • Free

NEXT

Restrictions

Increasing visitation time and budget will result in more recommendations

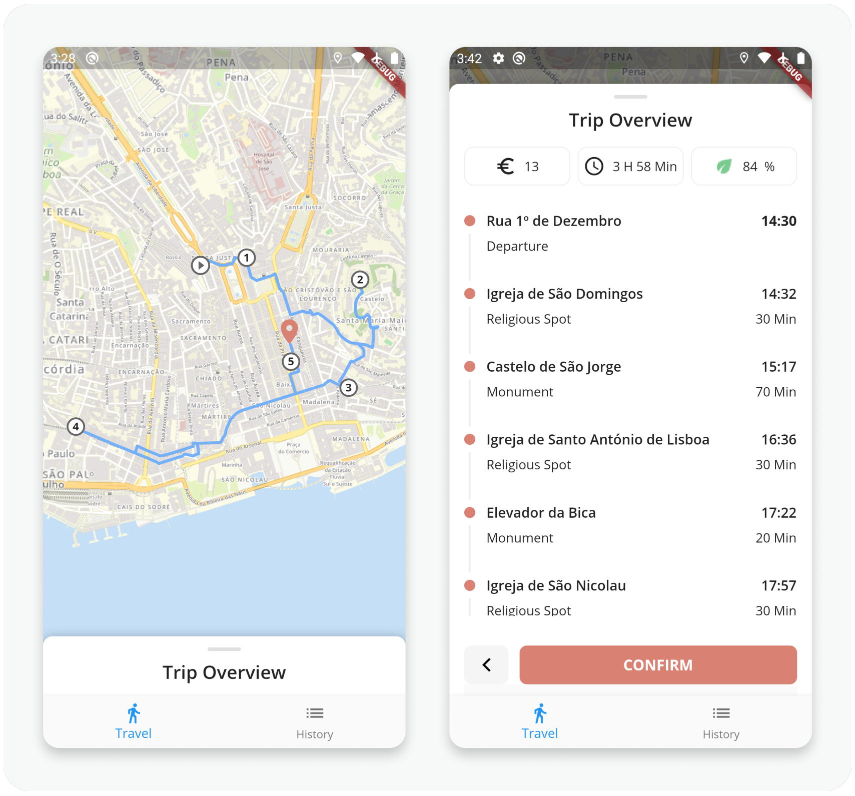
Visitation Time: 40 Minutes

Effort: Medium

Budget (per person): 25 €

NEXT

Figure 5.10: Trip Creation UI


Trip Overview

13 | 3 H 58 Min | 84 %

Rua 1º de Dezembro	14:30
Departure	
Igreja de São Domingos	14:32
Religious Spot	30 Min
Castelo de São Jorge	15:17
Monument	70 Min
Igreja de Santo António de Lisboa	16:36
Religious Spot	30 Min
Elevador da Bica	17:22
Monument	20 Min
Igreja de São Nicolau	17:57
Religious Spot	30 Min

CONFIRM

Figure 5.11: Trip Overview UI

5.2.6.3 Trip History

The *Trip History* feature is purposefully separated from the rest of the functionalities of the app. Since *Trip Creation* and *City map* both take advantage of a map-based interface, it makes sense that their respective UIs are similar and that the transition from one to the other is seamless. *Trip History* on the other hand does not benefit from a map-based interface, so it is more beneficial to move this feature to another screen entirely. Users can navigate to this screen (and return to the previous) using the *Bottom Navigation Bar* situated at the bottom of the app. Both Material Design and Human Interface Guidelines recommend the usage of *Bottom Navigation Bars* (called *Tab Bars* by the latter) for navigation between up to five top level destinations.

In this screen (Fig. 5.12), we present the user with two main UI elements. One is a top header containing statistics about his travels, more precisely, the amount of POIs visited, the distance he has travelled and the average sustainability score of his trips. It is a small instance of gamification that might give the tourist a sense of progress and make him want to come back to the app and "do better" (e.g. increase the sustainability score). The other UI element is a list of all the planned or completed trips the user has made, which he can toggle between. From this list, users can see a preview of the trips that contains the date, level of sustainability and the origin and destination locations, so that they become easily recognizable. If the user wants to see the full details of a specific trip, he can tap on such trip to transition to another screen that contains such information. This screen is very similar to the *Trip Overview* screen that the user should already be familiar with. It displays the route and POIs location on the map and the trip information in the style of a timeline.

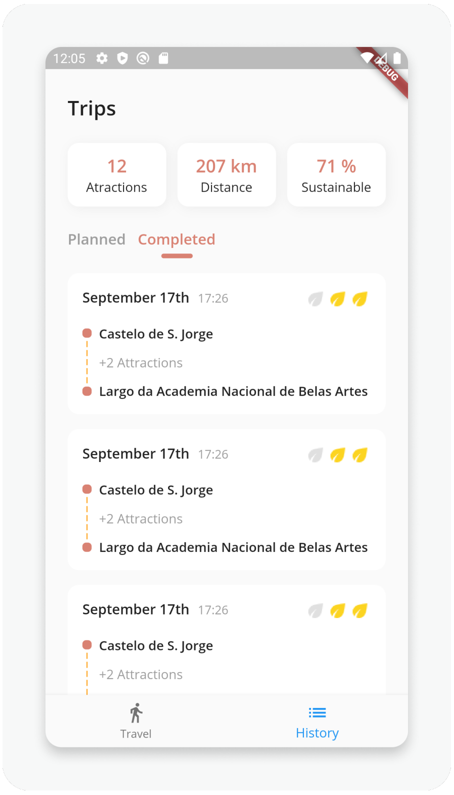


Figure 5.12: Trip History UI

Additionally, from this screen, tourists should be able to share their trips with their family and friends. The idea behind this feature is that the STC project, and therefore our app as well, is very dependant on having a large enough user base. The more users we have the better crowding information the system can gather, which in turn will result in better trip recommendations in the app. We hope that this feature can function as a sort of self-advertisement tool, in which tourists will share their trips on social media or other mediums, associated with the branding of our app. Unfortunately, due to time constraints, we were not able to develop this feature. In the Future Work section we dive into how an implementation of this feature could be done.

5.3 Design considerations

The ISO 9241-11 standard defines usability as the *"extent to which a system, product or service can be used by specified users to achieve specific goals with effectiveness, efficiency and satisfaction in a specified context of use"*. Any interactive system should consider usability as a major critical success factor [15].

Most studies on MTGs, usability and other mobile related areas [40, 42, 46, 47, 56, 95] claim that some variation of: (1) small screen sizes and different resolutions, (2) intermittent connectivity, (3) context of use, (4) input methods and (5) limited processing capabilities are the main usability concerns to address when developing these systems.

We believe that small screen sizes and limited processing capabilities are now less of a problem than they were at the time the authors wrote such articles. Mobile devices are on a trend to become larger each year and converging to a small set of aspect ratios. Similarly, mobile devices' specifications (i.e. hardware) have become very powerful, resulting in very capable phones. Granted, not everyone has a modern smartphone, specially considering that the prices have also gone up. Nonetheless, for our app, we assume that tourists, our target audience, have a smartphone from the last 5 years, which means that it has GPS and Bluetooth functionalities and should be able to run the app without problems.

On the other hand, we believe that the usage context of an MTG, together with the reality of intermittent connectivity, have a tremendous impact in the usability of our app. In this section, we present our solution to minimize these problems, along with other design concerns we took to improve the user experience and usability of our app.

5.3.1 Intermittent network

Due to the nature of tourism and travel, it is expected that our users will interact with the app whilst on the move. In fact, this is one of the major purposes of our app: to guide the user on a trip through the city's most sustainable attractions. In this travelling context, it is inevitable that our users lose Internet connectivity for variable periods of time. Either due to entering Internet dead zones, by moving between free Wi-Fi hotspots from local shops and *cafés*, or simply to prevent expensive roaming charges, Internet connection will become unavailable at some point during the apps intended usage context.

A study by [25] shows us that developers should not assume the tourist is "always connected" and that tourists themselves have a desire for disconnection. It is easy to imagine a scenario where a tourist decides to go offline while starting his trip in Lisbon, so that he does not get disturbed by incoming email notifications. Furthermore, a study on the impact of technology dead zones [63] observed that tourists' dependence on mobile connectivity results in several "tensions" when they travel to destinations where Internet connectivity is unavailable.

Abandoning the user while he is on the move, after entrusting us with the task of guiding them through the city would cause an extremely bad experience. It is essential that our app can survive these dead zones and keep providing support to the user at every stage of the travelling process.

Be that as it may, some loss of functionality is inevitable. Without Internet connection we cannot communicate with the other microservices or use crucial location services such as *geocoding*. This means that some compromises have to be made. Creating a trip is simply not possible without Internet connection, while keeping track of the user's position and displaying it on the map can and should be done, since it requires only GPS and not an Internet connection. In order to reduce the amount of compromises to a minimum, we made some design decisions that make the app less dependent on Internet connection and more functional overall.

One of the first big decisions we made that shaped the prototype is to support offline maps. While most map APIs obtain their map tiles by streaming them on demand over the Internet, there is a second option of including the tiles as *PNG* assets within the app. This way, there is no need for any type of download, stream or communication with a tile provider over the Internet. There are no loading times and the full area of the map (in our case, Lisbon) is instantly available.

There are some downsides to this approach as well. Working with offline tiles means that they don't automatically reflect the latest map data available (e.g. new roads or buildings) and require a manual update every time something changes (i.e. manually extracting tiles from a tile provider and updating the app through the marketplace). Since tiles do not change very often (the latest Lisbon tiles from *OpenMapTiles*[4] are from 2017), this problem is somewhat irrelevant. Another more important issue is the cost this approach has on the app size. The map tiles used for our prototype include most of the city of Lisbon and around and have a file size of 265MB, but this size shrinks when building the app to a release version (the total app size on a real device is only 171MB). However, when comparing to other apps such as *Google Maps*, that has a total size of 210MB, this downside does not look that problematic. Perhaps when applying this prototype to a larger city, this solution will become noticeably less scalable and show its real disadvantage.

Related to the offline map is the issue of using and displaying POI information. While previously (Sec. 3.2.1) we mentioned that the app retrieves the list of the city's POIs from the POINT microservice, therefore requiring Internet connection, this is not necessarily always the case. Just like with map tiles, the app comes preloaded with a list of the city's POIs so that it can display them on the map even when offline. However, since POIs are more likely to change than map tiles, a different approach has to be taken when it comes to updating them.

In fact, every time the app starts, it attempts to retrieve the latest POI data from the POINT microservice. If it is unsuccessful, it uses the latest available data stored locally. If it is successful, it stores the data locally so that it can be used the next time in case of lack of Internet connection (Fig 5.13). This way, users can have access to the city's latest POI information when an Internet connection is available and have a local (note that this does not mean it is outdated) as a fallback. For the user, this process is completely invisible and he does not even realise

[4] https://openmaptiles.com/downloads/tileset/osm/europe/portugal/lisbon/

which version he is using.

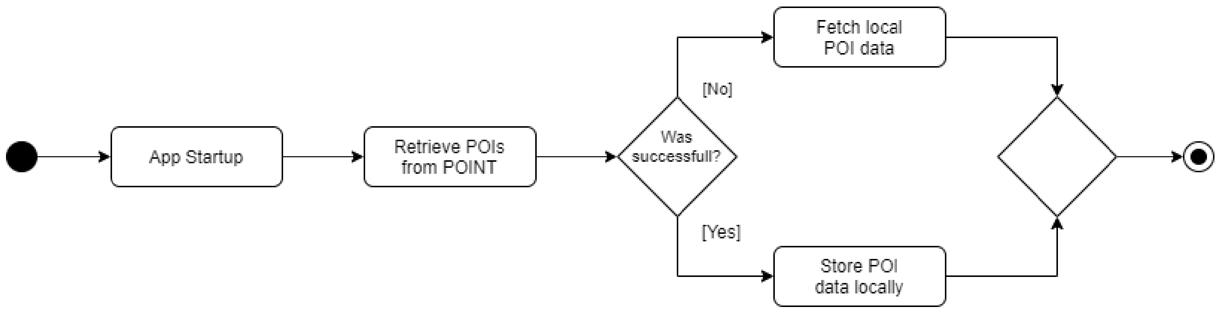


Figure 5.13: POIs caching mechanism

Location search is another functionality that is particularly affected by the lack of Internet. Without an Internet connection it is impossible to perform the necessary geocoding (and reverse geocoding) to locate and show the places the user is searching for. To circumvent this, we cache the successful results from searches done with Internet connection into a local database and present these results to the user on a *Recent Searches* list (Fig. 5.14). While we inevitably lose a significant part of the functionality one would expect from location search, using this technique we can guarantee a minimal functional version, increasing our support to the user and lowering our dependence on Internet connectivity.

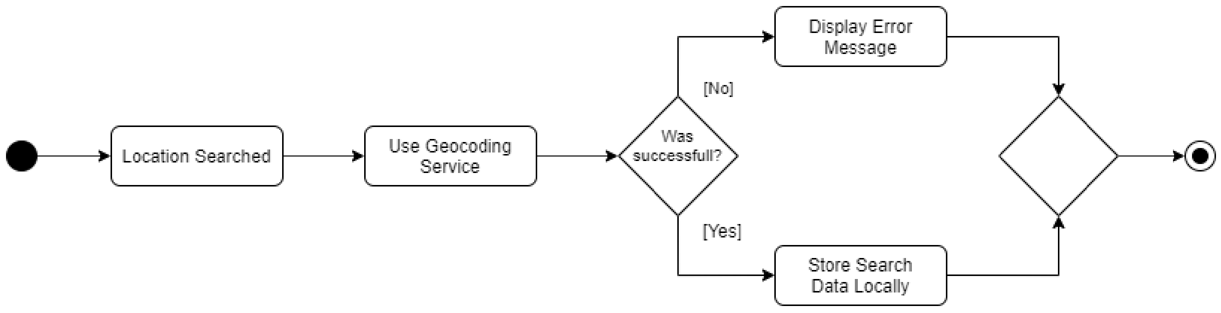


Figure 5.14: Search caching mechanism

5.3.2 Handling errors and alternative states

We have previously discussed how the usage context of our app makes it inevitable for the user to encounter errors at some point while using the app. When any of these circumstances occur, what we must do is inform the user properly about the situation and provide useful information to solve what went wrong. We do this by presenting the user with an error message.

The phrasing and contents of error messages can significantly affect user performance and satisfaction [77]. On top of that, *"improving the error messages is one of the easiest and most effective ways to improve an existing interface."* In our app we created a design system for our error messages that consists of an illustration (from unDraw[5], an open-source illustration library), a text explanation and if appropriate a call-to-action button. We apply this configuration consistently on our error messages, following a few established guidelines [66, 77], such as being specific and humane, using proportional language, and providing a solution if possible.

[5]https://undraw.co/

One example of such error messages occurs when the user tries to create a trip but has no Internet connection (Fig. 5.15). In this scenario, we must inform the user that this task is impossible without an Internet connection, advise him to connect to the Internet and provide him with two options: cancel the trip creation process or try again.

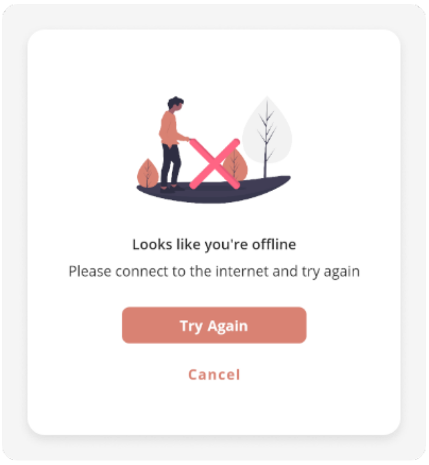

Looks like you're offline
Please connect to the internet and try again
Try Again
Cancel

Figure 5.15: Trip creation error

Errors are not the only alternative state of a UI component that we need to account for. A common mistake designers make is to overlook empty states [75]. Empty states happen when a user navigates to a certain UI component for information, but there is no data or content to show them (e.g. an empty friends list). If the component depends on user-generated content, it is likely that at some point the UI element will have no meaningful data to present to the user (e.g. no friends added yet).

This state can be treated similarly to an error state. When suitable, it is recommended to incorporate an image or illustration to grab the user's attention, and emphasizing the call-to-action to encourage them to take the next step [75]. In our app, we encounter this empty state in a few of our UI components, specifically on lists. When using the location search feature for the first time, there are obviously no previous searches to display on the *recent search* list. Instead of an empty list, we present a message, similar to an error message, that instructs the user that his recent searches will be saved and shown "here" the next time the user uses this feature (Fig. 5.16). Likewise, if the user navigates to the trip history screen before planning or completing any trip there will be nothing to show them. In this scenario, we alert the user that there are no planned or completed trips and provide a call to action for the user to plan one (Fig. 5.17).

Finally, UI components can also be in an intermediate state of loading, before displaying the respective content, empty state or error message. This loading state can be aggravating to users. According to 2018 research by Google[6], 53% of mobile users leave a site that takes longer than three seconds to load. In the context of our app, where users can be expected to have slow Internet such as a shared Wi-Fi hotspots, loading times can become painstakingly slow if the content being shown depends on a communication with other microservices (e.g. receiving a trip from the ROUTE microservice). It would be a bad user experience if the UI froze while

[6] https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/

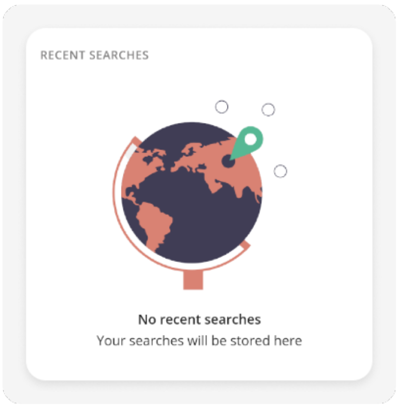


Figure 5.16: Trip history empty state

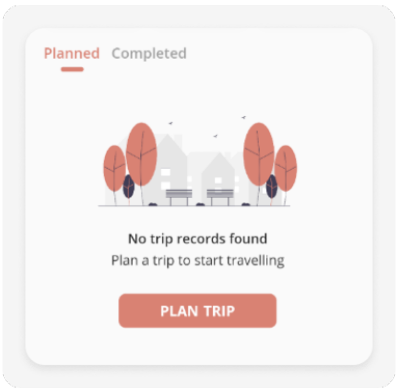


Figure 5.17: Recent searches empty state

waiting for the loading to finish, without giving any explanation to the user. To avoid this, UI components that are loading should explicitly show this to the user. One option is to use the shimmer loading effect, which is a technique we used throughout our app. One example of this effect is when the user navigates to the search panel and the *recent searches* list is still loading (Fig. 5.18).

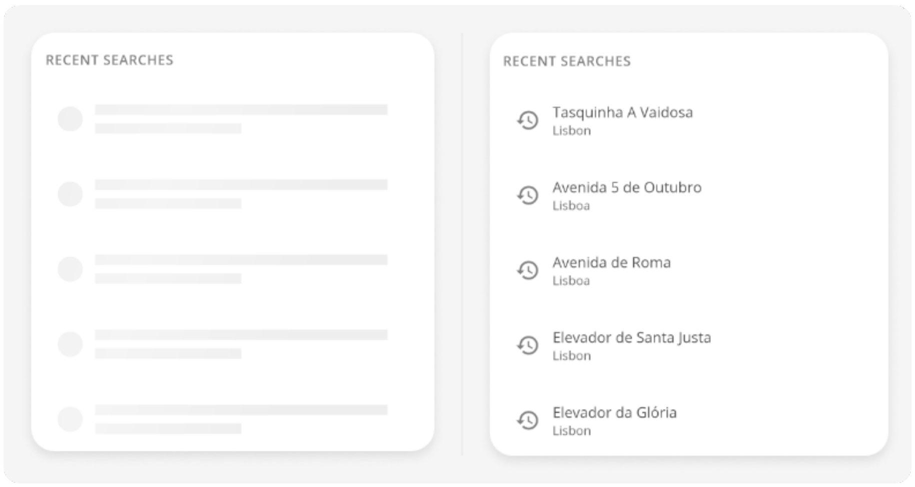


Figure 5.18: Shimmer loading effect

5.3.3 Adaptive UI components

Adaptive User Interfaces might mean different things to different people, depending on the context. In the world of web apps, an adaptive UI implies the creation of multiple, distinct versions of a web page (with distinct HTML structures) to better fit a user's device. This approach involves a designer creating 6 to 7 layouts that can be quite different from one another in terms of elements, spacing, and flow [7]. In the world of mobile apps, where devices aren't too different but the operating systems are, adaptive UI means building distinct UIs according to the device's operating system.

[7] https://www2.architech.ca/responsive-adaptive-web-design-for-ux-ui

Android apps usually follow the Material Design guidelines recommended by Google. Material Design[8] is a design language intended to be applied universally across all platforms and devices, based on a foundation of best practices and is informed by user experience research and cognitive science. More specifically, it provides guidance on core aspects of a UI like layout, color, typography and more. Apple's design language counterpart is called Human Interface Guidelines[9], made to help designers and developers *"create more compelling, intuitive, and beautiful experiences and design better apps."* It provides the same purpose as Material Design but is intended for Apple's devices: iOS, macOS, watchOS and tvOS.

When confronted with the task of building for these two operating systems, designers face the challenge of choosing between creating a universal design system, separate designs for each platform, or somewhere in between. In the Material Design documents Google includes a section[10] about when the differences in platform convention can affect the user's ability to understand the UI or complete certain tasks, so it becomes necessary to adapt it to the device's platform. Usually, these scenarios occur in UI components that require a higher level of interaction and effort from the user material, such as input/control components or action dialogues.

For our app, we decided to go with a single design system that works on both platforms, to save both on design and development time and effort, while also achieving a consistent branding across all devices. With that said, we took advice from Material's adaptation guidelines and included some adaptive UI components when appropriate to give users native look and feel on iOS and Android. Below (Fig. 5.19) is an example of two adaptive UI components (i.e. an modal action sheet and a slider) in our app that change their aspect according to the device's operating system.

[8]https://material.io/design

[9]https://developer.apple.com/design/human-interface-guidelines/

[10]https://material.io/design/platform-guidance/cross-platform-adaptation.html#when-to-adapt

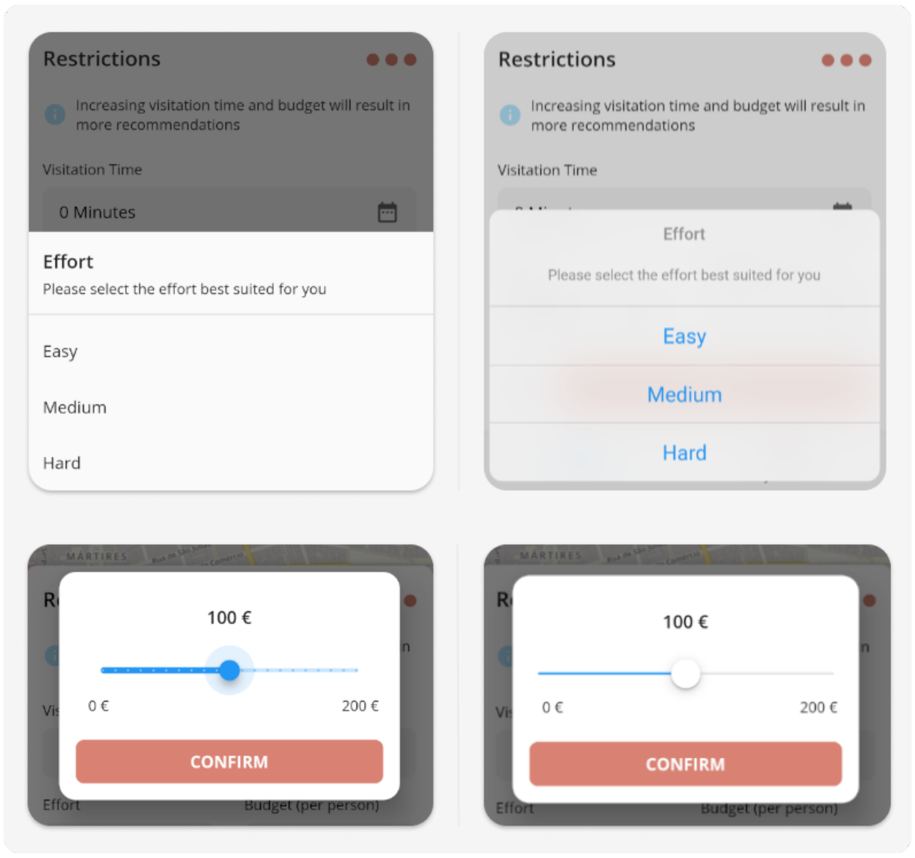

Restrictions
Increasing visitation time and budget will result in more recommendations
Visitation Time
0 Minutes
Effort
Please select the effort best suited for you
Easy
Medium
Hard
100 €
0 €
200 €
CONFIRM
Effort
Budget (per person)

Figure 5.19: Adaptive UI example. Android on the left and iOS on the right

Chapter 6.

Conclusion

Contents

This chapter concludes and summarizes the achievements of this dissertation. It also opens the discussion for challenges and future work in the area of MTGs

[This page has been intentionally left blank]

Chapter 6
Conclusion

6.1 Conclusions

This dissertation was developed as part of the STC project that intends to mitigate the problem of tourism overcrowding that is currently affecting the *Santa Maria Maior* district in Lisbon, by creating a mobile app designed for tourists (an MTG) that provides walking trip recommendations that avoid crowded areas and target sustainable POIs.

The first research goal of this dissertation pertains to the definition of the IA and UI. In order to do so, we adopted a user-centered approach to the design process, in which potential end-users influence the design, by including them through a series of usability techniques, namely a *Card Sort* technique and a *Tree Test* technique. An initial iteration of these techniques was done with members of the *Software Systems Engineering* group at ISCTE, and then a second iteration was done in collaboration with the *Junta de Freguesia de Santa Maria Maior* partners. The card sorting techniques were used as a generative technique to define a prototype IA, which was subsequently validated by the positive results of the tree testing technique (see Section 5.2). Mobile devices, together with the expected usage context of the app, bring a particular set of usability concerns that must also be addressed in the design stage of the prototype. These design considerations are discussed in Section 5.3.

The second research goal of this dissertation is directed towards the development of the MTG prototype and its integration with the rest of the remaining components of the STC project. The prototype contains the main features that are essential to an MTG, presented in Section 5.2, and to the STC project, such as the Bluetooth Detection Service discussed in Section 4.4. The prototype was developed using the *Flutter SDK*, which enabled us to target both Android and iOS easily with a single code base due to its cross-platform capabilities, greatly reducing the development time and effort. The functional aspects of the app prototype were validated with a series of unit tests, integration tests and manual tests, discussed in Chapter 4.3.

The third research question of this dissertation pertains to minimising the negative effects of losing Internet connectivity, by defining a minimum level of quality of service for when the app is disconnected from the Internet, that still guarantees a good user experience. This is particularly challenging not only due to the usage context of the app, but also because the app relies on communication with other components of the STC microservice architecture. In order to achieve this goal we created caching mechanisms that reduce the app's dependency on Internet (Sec. 5.3.1). Model classes were created to hold the relevant data (e.g. POI, location search), and stored in persistent databases. While it is not possible to completely incorporate a full-featured offline MTG into an app, it is certainly possible to attenuate the negative effects of losing Internet connection and still provide a satisfactory experience to the user.

With the work undertaken in this dissertation, we hope to have contributed with a significant advancement for the STC project, by developing the APP component and integrating it with the remaining components, and contributed to the MTG literature, by creating an MTG that diverges from the ones reviewed in the RR in the aspects discussed in Section 2.3.4.

6.2 Future work and limitations

Many different advancements, tests, and research questions have been left for the future due to lack of time. In this section, we present a few of those, that we consider most relevant, and a proposal of how they could be achieved.

6.2.1 Further validation

The work undertaken in this dissertation has been validated using two different techniques, a functional validation was employed to validate the functional requirements of the app, using a series of unit tests, integration tests and manual tests, and the usability of the app was validated with two usability techniques involving potential users. However, both validation techniques have limitations.

The unit test and integration tests developed for the functional validation only target the trip creation process as of right now, so it is necessary to expand them to include other functional requirements of the app and increase the code coverage/path coverage. Additionally, the communication with the ROUTE microservice has been tested manually, but once the ROUTE and the remaining microservices are live and properly deployed, this communication should also be tested with unit tests and integration tests.

The validation of the app's usability is another area that can be improved upon. Besides increasing the number of participants so that the results have a better statistical validity, it would also be interesting to conduct these techniques with actual tourists, to see if the results would differ. Additionally, there are other techniques that can be done to further evaluate the usability, such as the *First-click Testing*, which helps us understand where on an interface people would click first to complete a task. Finally, to obtain a more thorough evaluation the usability of the app, it is also necessary to conduct a user study with potential users. The limitations of the employed techniques are also discussed in Section 5.2.5

Finally, once the components of the STC project are more mature, it is important to deploy the app into a real-life scenario with actual tourists, in order to evaluate if the app is successful with its recommendations, by measuring the user's adherence to the trips. In fact, a mechanism to measure users' adherence to the trip recommendations is already developed in the app, using the Dynamic Time Warping (DTW) algorithm [7]. To validate that this mechanism works correctly at identifying deviations from the intended route, a test case was developed with a simpler, linear trip recommendation and four hypothetical travel paths (A to D), where route A is the closest to the original recommendation and route D is the furthest one. The simulated paths and the subsequent results of applying the DTW algorithm are presented in Appendix E, which validate that this mechanism is indeed appropriate at measuring user's adherence to the recommendations.

The RR undertaken for the *State of the Art* chapter is also subject to validity threats that are discussed in Section 2.2.5.

6.2.2 Prototype enhancements

The prototype can be improved in a number of ways, from small quality-of-life improvements to bigger features. One of the ways the app prototype can be improved on is in the process of guiding the user through the trip. Right now, trip information is static, which means that it does not get updated throughout the trip. However, by analysing the rate at which the user is moving, and by comparing the time he spends at POIs with the expected visitation time, it is possible to create a new prediction of the arrival times for the next POIs. Furthermore, if the crowding data changes significantly, to the point where the current route now crosses a crowded path, it should either automatically readjust or prompt the user to do so. One step further would be giving the user ability to reconfigure his trip while in the middle of it, by adding/removing POIs or readjusting their visitation order.

Another aspect of the app prototype that can be further improved is the profiling of the user. Right now, the prototype's only initiative towards user profiles is storing user's completed and planned trips and displaying them to the user, along side some statistics. With this information alone, we could be doing more. For instances, we could send the history of previously visited POIs to the ROUTE microservice so it would filter these out. There is also the opportunity to explore more advanced types of profiling, such as making sentiment analysis based on the way the user is travelling, the time spent on POIs, and more.

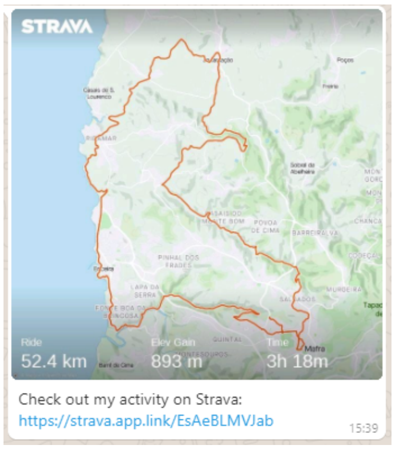
6.2.3 Research opportunities

One important research question that we would like to have explored is: *How to reuse and share trip experiences in social networks for the purpose of increasing user attraction?* A good example of trip sharing can be found in the commercial application *Strava*[1]. Strava is a mobile app made for runners and cyclists that tracks their runs or rides using GPS. It allows users to record and analyse statistics, explore and compete with others, and share and connect with friends and family. Strava's social features are very developed, to the point of being a *"social network for athletes"*. The most relevant social feature Strava has is the sharing of rides. Ride sharing generates a link to a page in Strava's website containing detailed information about the ride, including the route, distance, time, calories, and more, that can be sent to anyone. From there, Strava tries to "convince" users into downloading the app and trying it for themselves. This link also includes an image preview that can be displayed by most social apps, such as WhatsApp (Fig. 6.1).

We believe that this is a great way to increase user attraction and get more people to install our app, and get them to reuse these sustainable trips, which in turn will increase the effectiveness of the STC project.

Another innovative idea that we would like to explore, that is also capturing attention on other MTG and mapping apps, is related to the current Covid-19 pandemic situation. Now, more than ever, crowding has become an extremely important problem. While the goals of the STC project and the app already try to prevent crowding in an automatic manner, now that the end users are more aware of the risks and negative effects of it, it can be a good time

[1] `https://www.strava.com/mobile`



Check out my activity on Strava:
https://strava.app.link/EsAeBLMVJab

Figure 6.1: Strava WhatsApp ride sharing

to also count on them to further diminish this negative issue. The idea we want to explore is to add and display real time crowding data on the app using some sort of map appropriate format, such as polygons or curves (e.g. topographic maps). Users will be able to read this information and make their itinerary decision based on crowding. Knowing which streets are more or less crowded can perhaps increase the safety and tranquility of tourists (or even the local population) when on a trip.

Bibliography

[1] 42matters. *Google Play Store ASO with App Update Frequency Statistics 2020*. Aug. 2020.

[2] C. Abras, D. Maloney-Krichmar, J. Preece, and Others. "User-centered design." In: *Bainbridge, William. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications* 37.4 (2004), pp. 445–456.

[3] D. Ahlers, S. Boll, and D. Wichmann. *Virtual signposts for location-based story-telling*. Tech. rep. OFFIS, 2008.

[4] A. Allan. *Learning iOS Programming, Third Edition*. 2013. ISBN: 9781449359348.

[5] H. Alqahtani and M. Kavakli. "IMAP-CampUS: Developing an Intelligent Mobile Augmented Reality Program on Campus as a Ubiquitous System." In: *Proceedings of the 9th International Conference on Computer and Automation Engineering*. ICCAE '17. Sydney, Australia: Association for Computing Machinery, 2017, 1–5. ISBN: 9781450348096. DOI: 10.1145/3057039.3057062.

[6] Android Team. *View | Android Developers*. 2020.

[7] D. J. Berndt and J. Clifford. "Using dynamic time warping to find patterns in time series." In: *KDD workshop*. Vol. 10. 16. Seattle, WA, USA: 1994, pp. 359–370.

[8] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli. "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development." In: *International Conference on Web Information Systems and Technologies*. 2017. ISBN: 9789897582462. DOI: 10.5220/0006353703440351.

[9] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli. "Progressive web apps: The possible web-native unifier for mobile development." In: *International Conference on Web Information Systems and Technologies*. Vol. 2. SCITEPRESS. 2017, pp. 344–351.

[10] Bloc Team. *Bloc State Management*. 2020.

[11] J. Borràs, A. Moreno, and A. Valls. "Intelligent tourism recommender systems: A survey." In: *Expert Systems with Applications* 41.16 (2014), pp. 7370 –7389. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.06.007.

[12] E. Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. Vol. 1163. 1-2. 2010, pp. 145–56. ISBN: 9781934356562.

[13] D. Cameron, C. Gregory, and D. Battaglia. "Nielsen personalizes the mobile shopping app: If you build the technology, they will come." In: *Journal of Advertising Research* 52.3 (2012). ISSN: 00218499. DOI: 10.2501/JAR-52-3-333-338.

[14] J.-C. Cano, P. Manzoni, and C.-K. Toh. "UbiqMuseum: A Bluetooth and Java Based Context-Aware System for Ubiquitous Computing." In: *Wireless Personal Communications* 38.2 (July 2006), pp. 187–202. ISSN: 0929-6212. DOI: 10.1007/s11277-005-9001-x.

[15] R. Capilla, L. Carvajal, and H. Lin. "Addressing Usability Requirements in Mobile Software Development." In: *Relating System Quality and Software Architecture*. Elsevier Inc., July 2014, pp. 303–324. ISBN: 9780124171688. DOI: 10.1016/B978-0-12-417009-4.00012-0.

[16] B. Cartaxo, G. Pinto, and S. Soares. "The Role of Rapid Reviews in Supporting Decision-Making in Software Engineering Practice." In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. EASE'18. Christchurch, New Zealand: Association for Computing Machinery, 2018, 24–34. ISBN: 9781450364034. DOI: 10.1145/3210459.3210462.

[17] B. Cartaxo, G. Pinto, and S. Soares. "Rapid Reviews in Software Engineering." In: *Contemporary Empirical Methods in Software Engineering*. Ed. by M. Felderer and G. H. Travassos. Cham: Springer International Publishing, 2020, pp. 357–384. ISBN: 978-3-030-32489-6. DOI: 10.1007/978-3-030-32489-6_13.

[18] F. Cena, L. Console, C. Gena, A. Goy, G. Levi, S. Modeo, and I. Torre. "Integrating Heterogeneous Adaptation Techniques to Build a Flexible and Usable Mobile Tourist Guide." In: *AI Communications* 19.December 2006 (2006), pp. 369–384.

[19] A. Charl and B. LeRoux. "Mobile Application Development: Web vs. Native: Web apps are cheaper to develop and deploy than native apps, but can they match the native user experience?" In: *Queue* 9.4 (Apr. 2011), pp. 20–28. ISSN: 15427749. DOI: 10.1145/1966989.1968203.

[20] Q. Chen, M. Zhang, and X. Zhao. "Analysing customer behaviour in mobile app usage." In: *Industrial Management and Data Systems* 117.2 (2017), pp. 425–438. ISSN: 02635577. DOI: 10.1108/IMDS-04-2016-0141.

[21] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. "Developing a context-aware electronic tourist guide: Some issues and experiences." In: *Conference on Human Factors in Computing Systems - Proceedings*. New York, New York, USA: ACM Press, 2000, pp. 17–24. ISBN: 1581132166. DOI: 10.1145/332040.332047.

[22] Dart Team. *Dart documentation | Dart*. 2020.

[23] J. Decuir. *Introducing bluetooth smart: Part 1: A look at both classic and new technologies*. Jan. 2014. DOI: 10.1109/MCE.2013.2284932.

[24] L. Delia, N. Galdamez, P. Thomas, L. Corbalan, and P. Pesado. "Multi-platform mobile application development analysis." In: *Proceedings - International Conference on Research Challenges in Information Science*. Vol. 2015-June. June. IEEE Computer Society, June 2015, pp. 181–186. DOI: 10.1109/RCIS.2015.7128878.

[25] J. E. Dickinson, J. F. Hibbert, and V. Filimonau. "Mobile technology and the tourist experience: (Dis)connection at the campsite." In: *Tourism Management* 57 (Dec. 2016), pp. 193–201. ISSN: 02615177. DOI: 10.1016/j.tourman.2016.06.005.

[26] M. D. Dunlop, P. Ptasinski, A. Morrison, S. McCallum, C. Risbey, and F. Stewart. "Design and development of Taeneb City Guide: From Paper Maps and Guidebooks to Electronic Guides." In: *Information and Communication Technologies in Tourism 2004*. Springer Vienna, 2004, pp. 58–64. DOI: 10.1007/978-3-7091-0594-8_6.

[27] C. Emmanouilidis, R. A. Koutsiamanis, and A. Tasidou. "Mobile guides: Taxonomy of architectures, context awareness, technologies and applications." In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 103–125. ISSN: 10848045. DOI: 10.1016/j.jnca.2012.04.007.

[28] I. N. de Estatística. *Statistical Yearbook Portugal 2019*. Tech. rep. Instituto Nacional de Estatística, 2019.

[29] N. Fenton and J. Bieman. *Software metrics: a rigorous and practical approach*. CRC press, 2014. ISBN: 9781439838228.

[30] Flutter Team. *Flutter architectural overview*. 2020.

[31] Flutter Team. *Introduction to widgets - Flutter*. 2020.

[32] Flutter Team. *State management - Flutter*. 2020.

[33] M. Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 9780321127426.

[34] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018. ISBN: 9780134757599.

[35] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995. ISBN: 9780201633610.

[36] A. Garcia, M. T. Linaza, O. Arbelaitz, and P. Vansteenwegen. "Intelligent Routing System for a Personalised Electronic Tourist Guide." In: *Information and Communication Technologies in Tourism 2009*. Springer Vienna, 2009, pp. 185–197. DOI: 10.1007/978-3-211-93971-0_16.

[37] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou. "Mobile recommender systems in tourism." In: *Journal of Network and Computer Applications* 39.1 (2014), pp. 319–333. ISSN: 10848045. DOI: 10.1016/j.jnca.2013.04.006.

[38] Google and PhocusWright. *How smartphone usage is shaping travel decisions - Think with Google*. Feb. 2018.

[39] E. Haid, G. Kiechle, N. Göll, and M. Soutschek. "Evaluation of a Web-based and Mobile Ski Touring Application for GPS-enabled Smartphones." In: *Information and Communication Technologies in Tourism 2008*. Springer Vienna, Feb. 2008, pp. 313–323. DOI: 10.1007/978-3-211-77280-5_28.

[40] R. Harrison, D. Flood, and D. Duce. "Usability of mobile applications: literature review and rationale for a new usability model." In: *Journal of Interaction Science* (2013). DOI: 10.1186/2194-0827-1-1.

[41] W. Hopken, M. Fuchs, M. Zanker, and T. Beer. "Context-Based Adaptation of Mobile Applications in Tourism." In: *Information Technology & Tourism* 12 (2010), pp. 175–195. DOI: 10.3727/109830510X12887971002783.

[42] R. Jabangwe, H. Edison, and A. N. Duc. "Software engineering process models for mobile app development: A systematic literature review." In: *Journal of Systems and Software* 145.May (2018), pp. 98–111. ISSN: 01641212. DOI: 10.1016/j.jss.2018.08.028.

[43] S. Karanasios, S. Burgess, and C. Sellitto. "A classification of mobile tourism applications." In: *Global Hospitality and Tourism Management Technologies*. IGI Global, 2011, pp. 165–177. ISBN: 9781613500415. DOI: 10.4018/978-1-61350-041-5.ch011.

[44] H. Kennedy-Eden and U. Gretzel. "A taxonomy of mobile applications in tourism." In: *University of Wollongong* (2012).

[45] M. Kenteris, D. Gavalas, and D. Economou. "Evaluation of mobile tourist guides." In: *Communications in Computer and Information Science*. Vol. 19. Springer Verlag, 2008, pp. 603–610. ISBN: 9783540877820. DOI: 10.1007/978-3-540-87783-7_77.

[46] M. Kenteris, D. Gavalas, and D. Economou. "An innovative mobile electronic tourist guide application." In: *Personal and Ubiquitous Computing* 13.2 (2009), pp. 103–118. ISSN: 16174909. DOI: 10.1007/s00779-007-0191-y.

[47] M. Kenteris, D. Gavalas, and D. Economou. *Electronic mobile guides: A survey*. Jan. 2011. DOI: 10.1007/s00779-010-0295-7.

[48] S Khangura, K Konnyu, R Cushman, et al. "Evidence summaries: a rapid review method." In: *Syst Rev* 1 (2012), pp. 20146–24053. DOI: 10.1186/2046-4053-1-10.

[49] B. Kitchenham. "Procedures for Performing Systematic Reviews." In: *Keele, UK, Keele University* (2004).

[50] C. D. Kounavis, A. E. Kasimati, and E. D. Zamani. "Enhancing the Tourism Experience through Mobile Augmented Reality: Challenges and Prospects." In: *International Journal of Engineering Business Management* 4 (Jan. 2012), p. 10. ISSN: 1847-9790. DOI: 10.5772/51644.

[51] R. Kramer, M. Modsching, K. ten Hagen, and U. Gretzel. "Behavioural Impacts of Mobile Tour Guides." In: *Information and Communication Technologies in Tourism 2007*. Springer Vienna, 2007, pp. 109–118. DOI: 10.1007/978-3-211-69566-1_11.

[52] R. Kramer, M. Modsching, and K. ten Hagen. "Development and evaluation of a context-driven, mobile tourist guide." In: *International Journal of Pervasive Computing and Communications* 3.4 (Apr. 2008), pp. 378–399. ISSN: 17427371. DOI: 10.1108/17427370710863121.

[53] G. E. Krasner and S. T. Pope. "A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80." In: *Journal of Object-oriented Programming - JOOP* 1.3 (1988), pp. 26–49. ISSN: 0896-8438. DOI: dl.acm.org/doi/10.5555/50757.50759.

[54] N. Lebeck, A. Krishnamurthy, H. M. Levy, and I. Zhang. “End the Senseless Killing: Improving Memory Management for Mobile Operating Systems.” In: *2020 {USENIX} Annual Technical Conference ({USENIX} {ATC} 20)*. {USENIX} Association, 2020, pp. 873–887. ISBN: 978-1-939133-14-4.

[55] M. Y. Lim and R. Aylett. “Feel the difference: A guide with attitude!” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 4722 LNCS. Springer, Berlin, Heidelberg, 2007, pp. 317–330. ISBN: 9783540749967. DOI: 10.1007/978-3-540-74997-4_29.

[56] Q. Liu, H. Ma, E. Chen, and H. Xiong. *A survey of context-aware mobile recommendations*. 2013. DOI: 10.1142/S0219622013500077.

[57] T. Lou and Others. “A comparison of android native app architecture–mvc, mvp and mvvm.” In: *Eindhoven University of Technology* (2016).

[58] U. A. Mannan, D. Dig, I. Ahmed, C. Jensen, R. Abdullah, and M Almurshed. “Understanding Code Smells in Android Applications.” In: *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2016. ISBN: 9781450341783. DOI: 10.1145/12345.67890.

[59] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O’Reilly Media, Inc.", 2011. ISBN: 9781449319908.

[60] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler. *The art of software testing*. Vol. 2. Wiley Online Library, 2004. DOI: 10.1002/stvr.322.

[61] J. M. Noguera, M. J. Barranco, R. J. Segura, and L. Martínez. “A mobile 3D-GIS hybrid recommender system for tourism.” In: *Information Sciences* 215 (Dec. 2012), pp. 37–52. ISSN: 00200255. DOI: 10.1016/j.ins.2012.05.010.

[62] M. Palmieri, I. Singh, and A. Cicchetti. “Comparison of cross-platform mobile development tools.” In: *2012 16th International Conference on Intelligence in Next Generation Networks, ICIN 2012*. IEEE, Oct. 2012, pp. 179–186. ISBN: 9781467315265. DOI: 10.1109/ICIN.2012.6376023.

[63] P. Pearce and U. Gretzel. “Tourism in technology dead zones: documenting experiential dimensions.” In: *Faculty of Business - Papers (Archive)* (Jan. 2012). DOI: 10.1080/15980634.2012.11434656.

[64] P. Peeters, S. Gössling, J. Klijs, C. Milano, M. Novelli, C. Dijkmans, E. Eijgelaar, S. Hartman, J. Heslinga, R. Isaac, O. Mitas, S. Moretti, J. Nawijn, B. Papp, and A. Postma. “Research for TRAN Committee - Overtourism: impact and possible policy responses.” In: *European Parliament, Directorate General for Internal Policies, Policy* (2018).

[65] A. R. H. Peixoto. “A graph-based approach for sustainable walking tour recommendations: the case of Lisbon overcrowding.” Master’s thesis. Iscte – Instituto Universitário de Lisboa, 2019. DOI: http://hdl.handle.net/10071/20234.

[66] P. Perea and P. Giner. *UX Design for Mobile*. Packt Publishing Ltd, 2017. ISBN: 9781787283428.

[67] M. T. Phyo. “Choosing a Mobile Application Development Approach.” In: *ASEAN Journal of Management and Innovation* (2014), p. 69. DOI: 10.14456/AJMI.2014.4.

[68] R. E. F. Pierre Bourque. *Guide to the Software Engineering Body of Knowledge (SWEBOK(r)): Version 3.0*. 3rd. IEEE Computer Society Press, 2014. ISBN: 0769551661,9780769551661.

[69] G. Pospischil, M. Umlauft, and E. Michlmayr. “Designing LoL@, a Mobile Tourist Guide for UMTS.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2411. 2002, pp. 140–154. DOI: 10.1007/3-540-45756-9_12.

[70] M. Potel. “MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java.” In: *Taligent* (1996).

[71] D. R. Prasanna. “Dependency injection.” In: *Manning Publications* (2009).

[72] Reso Coder. *Flutter Clean Architecture Course*. 2020.

[73] F. Ricci and Q. N. Nguyen. “MobyRek: A conversational recommender system for on-the-move travellers.” In: *Destination Recommendation Systems: Behavioural Foundations and Applications*. CABI Publishing, July 2006, pp. 281–294. ISBN: 0851990231. DOI: 10.1079/9780851990231.0281.

[74] L. Rosenfeld and P. Morville. *Information architecture for the world wide web*. "O’Reilly Media, Inc.", 2002.

[75] Schoger Steve and Wathan Adam. *Refactoring UI*. 1st. Vol. 1. 2018.

[76] N. Serrano, J. Hernantes, and G. Gallardo. “Mobile web apps.” In: *IEEE Software* 30.5 (July 2013), pp. 22–27. ISSN: 07407459. DOI: 10.1109/MS.2013.111.

[77] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[78] R. D. da Silva. “A tourism overcrowding sensor using multiple radio techniques detection.” Master’s thesis. Iscte – Instituto Universitário de Lisboa, 2019. DOI: http://hdl.handle.net/10071/20212.

[79] J. Smith. “Patterns-wpf apps with the model-view-viewmodel design pattern.” In: *MSDN magazine* 72 (2009).

[80] P. Smutný. “Mobile development tools and cross-platform solutions.” In: *Proceedings of the 2012 13th International Carpathian Control Conference, ICCC 2012*. IEEE, May 2012, pp. 653–656. ISBN: 9781457718687. DOI: 10.1109/CarpathianCC.2012.6228727.

[81] I. Sommerville and P. Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[82] J. Sorgalla. “AjiL: A Graphical Modeling Language for the Development of Microservice Architectures.” In: *Extended Abstracts of the Microservices 2017 Conference*. 2017.

[83] D. Spencer and T. Warfel. “Card sorting: a definitive guide.” In: *Boxes and arrows* 2 (2004), pp. 1–23.

[84] P. N. Sukaviriya, J. D. Foley, and T. W. Griffith. “A Second Generation User Interface Design Environment: The Model and the Runtime Architecture.” In: *Proceedings of the INTERACT’93 and CHI’93 Conference on Human Factors in Computing Systems*. Georgia Institute of Technology, 1993.

[85] A. Syromiatnikov and D. Weyns. “A journey through the land of model-view-design patterns.” In: *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*. IEEE Computer Society, 2014, pp. 21–30. ISBN: 9781479934126. DOI: 10.1109/WICSA.2014.13.

[86] G. W. H. Tan, V. H. Lee, B. Lin, and K. B. Ooi. “Mobile applications in tourism: The future of the tourism industry?” In: *Industrial Management and Data Systems*. Vol. 117. 3. Emerald Group Publishing Ltd., 2017, pp. 560–581. DOI: 10.1108/IMDS-12-2015-0490.

[87] J. Tidwell. *Designing interfaces: Patterns for effective interaction design*. "O’Reilly Media, Inc.", 2010. ISBN: 9781449379704.

[88] V. Uren, Y. Lei, V. Lopez, H. Liu, E. Motta, and M. Giordanino. “The usability of semantic search tools: a review.” In: *The Knowledge Engineering Review* 22.4 (2007), pp. 361–377.

[89] R. Want. *When cell phones become computers*. Apr. 2009. DOI: 10.1109/MPRV.2009.40.

[90] M Wijesuriya, S Mendis, B. Bandara, K Mahawattage, N. Walgampaya, and D. de Silva. *(INTERACTIVE MOBILE BASED TOUR GUIDE*. 2013.

[91] C. Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering.” In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10.

[92] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[93] O. Workshop. *Tree testing 101 – Optimal Workshop*. 2020.

[94] S. Xanthopoulos and S. Xinogalos. “A comparative analysis of cross-platform development approaches for mobile applications.” In: *ACM International Conference Proceeding Series*. New York, New York, USA: ACM Press, 2013, pp. 213–220. ISBN: 9781450318518. DOI: 10.1145/2490257.2490292.

[95] R. Yáñez Gómez, D. Cascado Caballero, and J.-L. Sevillano. “Heuristic Evaluation on Mobile Interfaces: A New Checklist.” In: *The Scientific World Journal* 2014 (2014). Ed. by M. S. Obaidat, p. 434326. ISSN: 2356-6140. DOI: 10.1155/2014/434326.

[96] W. S. Yang and S. Y. Hwang. “ITravel: A recommender system in mobile peer-to-peer environment.” In: *Journal of Systems and Software* 86.1 (2013), pp. 12–20. ISSN: 01641212. DOI: 10.1016/j.jss.2012.06.041.

[97] Y. Zhang and Y. Luo. “An architecture and implement model for Model-View-Presenter pattern.” In: *2010 3rd international conference on computer science and information technology*. Vol. 8. IEEE. 2010, pp. 532–536.

[98] H. Zhao, H. Peng, and C. Sun. “A Mobile Service Platform for Xinjiang Tourism.” In: *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation*. 2015, pp. 26–30. DOI: 10.1109/ICMTMA.2015.15.

[99] Z. Zhao and C. Balagué. “Designing branded mobile apps: Fundamentals and recommendations.” In: *Business Horizons* 58.3 (May 2015), pp. 305–315. ISSN: 00076813. DOI: 10.1016/j.bushor.2015.01.004.

Appendix A. API JSON schemas

In this annex we present the JSON schema designed so that the APP instances can communicate with the multiple microservices of the STC architecture, developed using *Stoplight* in collaboration with the remaining members of the STC project. Each JSON schema is presented with an example for demonstration purposes.

A.1 Track submission JSON schema

The following listing (Lst. A.1) is an example of a track submission JSON representation, where 10 devices were found on July 21, at 17:38:07 in the *Igreja de Santa Maria Maior* POI.

```
{
    "count": 10,
    "timestamp": 1595353087,
    "location": {
        "latitude": 38.7097222,
        "longitude": -9.13305555,
        "altitude": 3.20932
    }
}
```

Listing A.1: TrackSubmission example

A.2 Point of interest JSON schema

The following listing (Lst. A.2) is an example of a *PointOfInterest* JSON representation, namely the *Martinho da Arcada* restaurant.

```
{
    "name":"Martinho da Arcada",
    "pointID":36,
    "visitTime":60,
    "openHour":7,
    "closeHour":17,
    "price":35,
    "coordinates":{
        "latitude":38.708675,
        "longitude":-9.135841
    },
    "sustainability":50,
    "categoryID":4
}
```

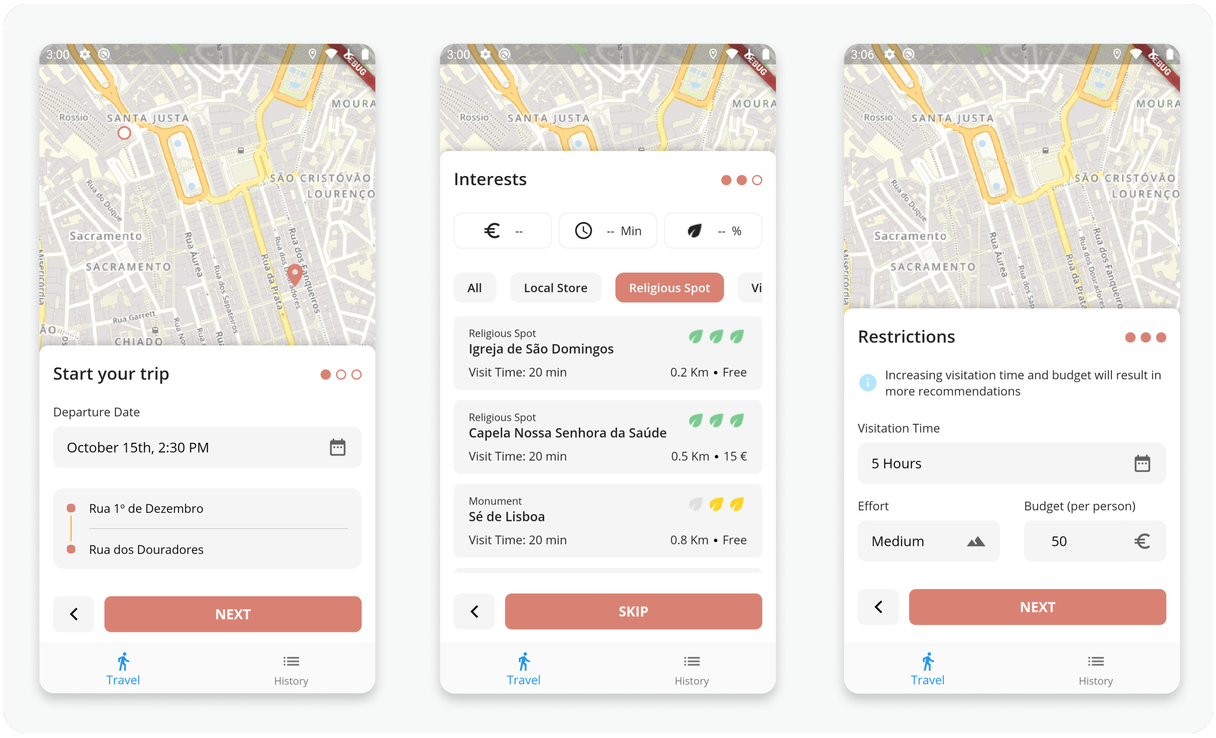
Listing A.2: PointOfInterest example

Appendix B. Test Scenarios

In this annex we present the results of testing the different scenarios. This includes screenshots of the app during the multiple stages of executing each scenario and the JSON messages that are exchanged between the app and the ROUTE microservice.

B.1 Scenario #1

Below are the results from scenario #1, described in Section 4.3.2 and Table 4.2.



Start your trip

Departure Date

October 15th, 2:30 PM

Rua 1º de Dezembro

Rua dos Douradores

NEXT

Interests

€ -- | -- Min | -- %

All | Local Store | Religious Spot | Vi

Religious Spot
Igreja de São Domingos
Visit Time: 20 min — 0.2 Km • Free

Religious Spot
Capela Nossa Senhora da Saúde
Visit Time: 20 min — 0.5 Km • 15 €

Monument
Sé de Lisboa
Visit Time: 20 min — 0.8 Km • Free

SKIP

Restrictions

Increasing visitation time and budget will result in more recommendations

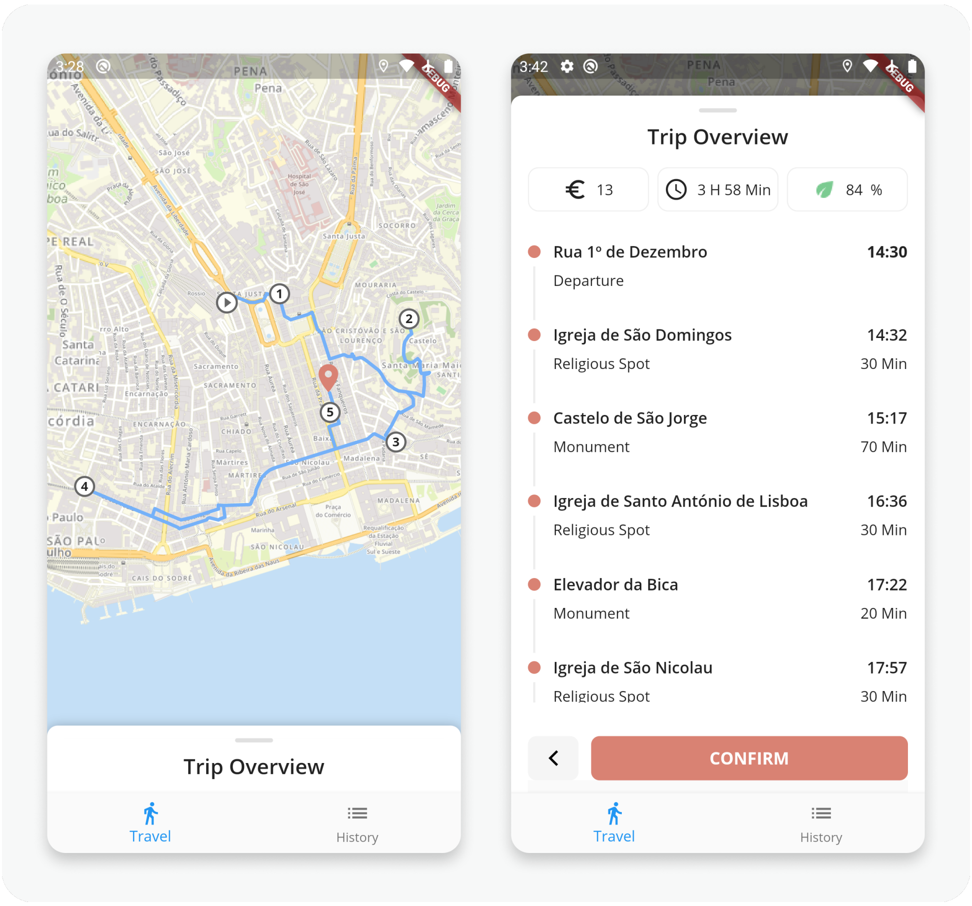
Visitation Time: 5 Hours

Effort: Medium

Budget (per person): 50 €

NEXT

Figure B.1: Scenario #1 Trip Creation UI



Trip Overview

€ 13 | 3 H 58 Min | 84 %

Rua 1º de Dezembro 14:30
Departure

Igreja de São Domingos 14:32
Religious Spot 30 Min

Castelo de São Jorge 15:17
Monument 70 Min

Igreja de Santo António de Lisboa 16:36
Religious Spot 30 Min

Elevador da Bica 17:22
Monument 20 Min

Igreja de São Nicolau 17:57
Religious Spot 30 Min

CONFIRM

Figure B.2: Scenario #1 Trip Overview UI

```
//Sent to ROUTE
"origin":{
    "latitude":38.7144118,
    "longitude":-9.1408772
},
"destination":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
"departureDate":1602772200000,
"visitationTime":300,
"budget":50,
"effortLevel":2,
"selectedPoints":[],
"selectedCategories":[2,7]
```

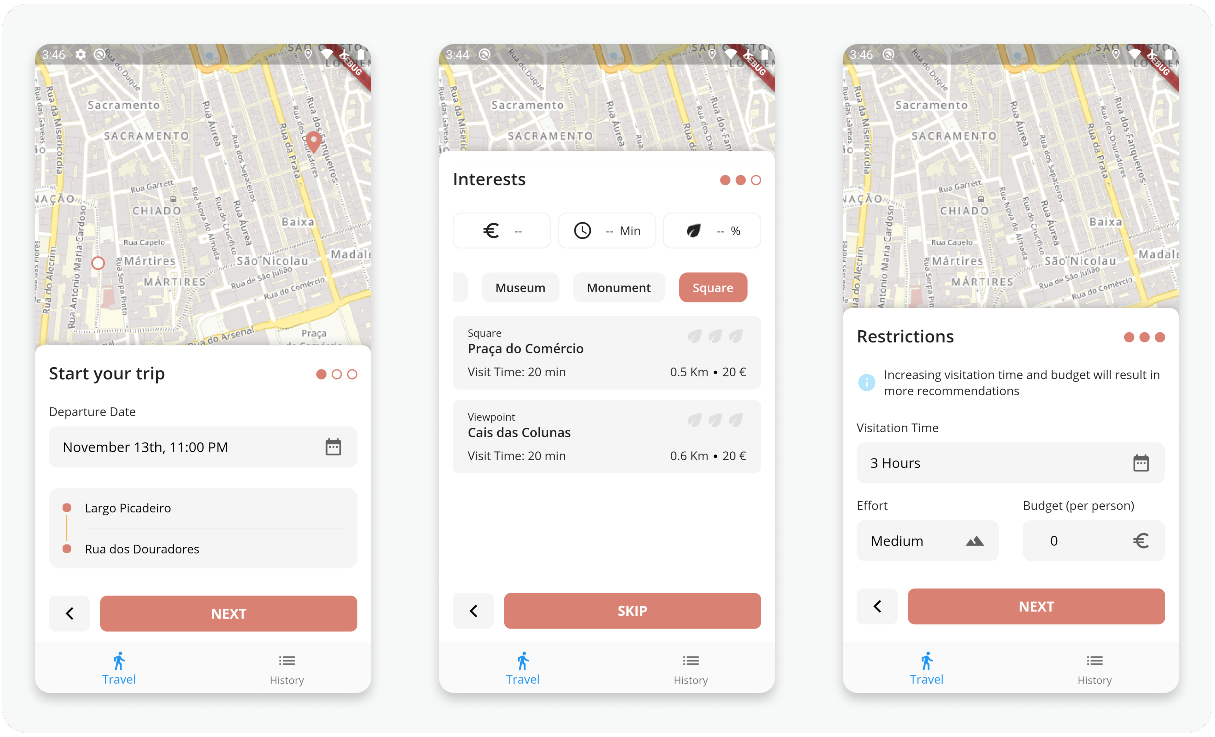
Listing B.1: Scenario #1 Route Request JSON

```
//Received from ROUTE
"origin":{
      "latitude":38.7144118,
      "longitude":-9.1408772
},
 "destination":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
"time":{
      "startTime":1602772200000,
      "endTime":1602786480000
},
"durationTime":14305687,
"distance":4869.105330647069,
"price":13,
"calories":229,
"sustainability":84,
"line":[
    {
        "latitude":38.71449467646691,
        "longitude":-9.14075525419612
    },
    ..., //Omitted for brevity
    {
        "latitude":38.7115624504062,
        "longitude":-9.136716261610735
    }
],
"pois":[
    {
        "timestamp":1602772367000,
        "poi": {
            "name":"Igreja de Sao Domingos",
            ..., //Omitted for brevity
        }
    },
    {
        "timestamp":1602775033000,
        "poi": {
            "name":"Castelo de Sao Jorge",
            ..., //Omitted for brevity
        }
    },
    {
        "timestamp":1602779775000,
        "poi": {
            "name":"Igreja de Santo Antonio de Lisboa",
            ..., //Omitted for brevity
        }
    }
    ... //Omitted for brevity
]
```

Listing B.2: Scenario #1 Route Response JSON

B.2 Scenario #2

Below are the results from scenario #2, described in Section 4.3.2 and Table 4.3.



Start your trip

Departure Date

November 13th, 11:00 PM

Largo Picadeiro

Rua dos Douradores

NEXT

Interests

€ -- | -- Min | -- %

Museum | Monument | Square

Square
Praça do Comércio
Visit Time: 20 min — 0.5 Km • 20 €

Viewpoint
Cais das Colunas
Visit Time: 20 min — 0.6 Km • 20 €

SKIP

Restrictions

Increasing visitation time and budget will result in more recommendations

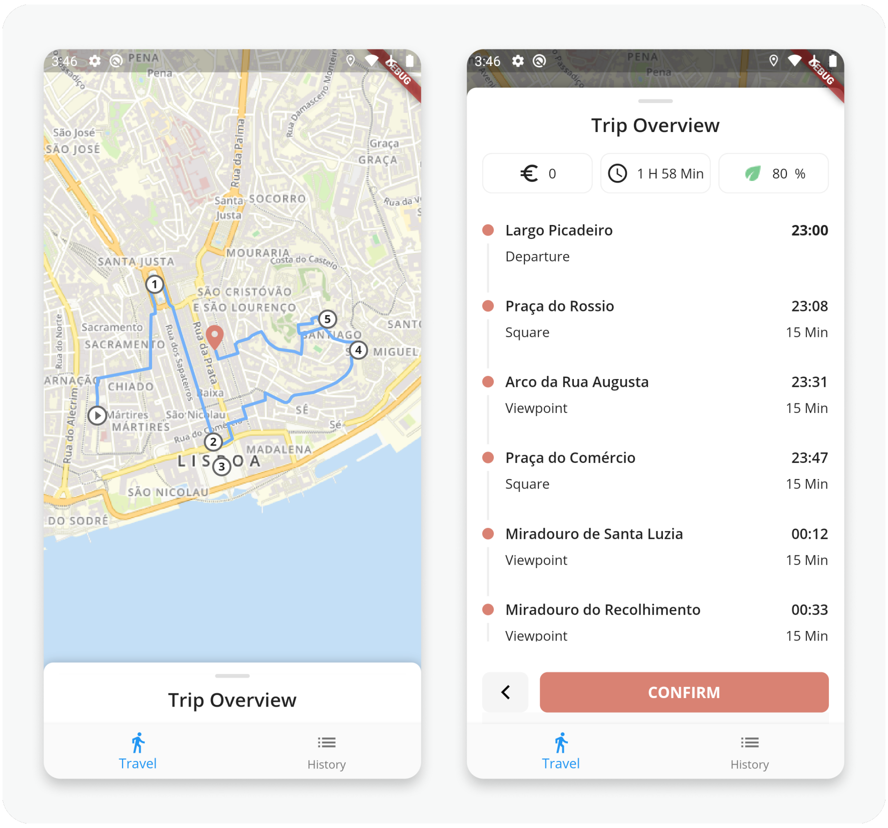
Visitation Time: 3 Hours

Effort: Medium

Budget (per person): 0 €

NEXT

Figure B.3: Scenario #2 Trip Creation UI



Largo Picadeiro 23:00
Departure

Praça do Rossio 23:08
Square 15 Min

Arco da Rua Augusta 23:31
Viewpoint 15 Min

Praça do Comércio 23:47
Square 15 Min

Miradouro de Santa Luzia 00:12
Viewpoint 15 Min

Miradouro do Recolhimento 00:33
Viewpoint 15 Min

Figure B.4: Scenario #2 Trip Overview UI

```
//Sent to ROUTE
"origin":{
  "latitude":38.7093008,
  "longitude":-9.141986
},
"destination":{
  "latitude":38.7115605,
  "longitude":-9.1367243
},
"departureDate":1602802800000,
"visitationTime":180,
"budget":0,
"effortLevel":2,
"selectedPoints":[],
"selectedCategories":[3,8]
```

Listing B.3: Scenario #2 Route Request JSON

```
//Received from ROUTE
"origin":{
      "latitude":38.7093008,
      "longitude":-9.141986
},
 "destination":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
"time":{
      "startTime":1602802800000,
      "endTime":1602809871000
},
"durationTime":7096643,
"distance":3606.5402985991464,
"price":0,
"calories":170,
"sustainability":80,
"line":[
    {
        "latitude":38.709302325206714,
        "longitude":-9.141963953758477
    },
    ..., // Omitted for brevity
    {
        "latitude":38.7115624504062,
        "longitude":-9.136716261610735
    }
],
"pois":[
    {
        "timestamp":1602803309000,
        "poi": {
            "name":"Praça do Rossio",
            ..., // Omitted for brevity
        }
    },
    {
        "timestamp":1602804699000,
        "poi": {
            "name":"Arco da Rua Augusta",
            ..., // Omitted for brevity
        }
    },
    {
        "timestamp":1602805621000,
        "poi": {
            "name":"Praça do Comércio",
            ..., // Omitted for brevity
        }
    }
    ... // Omitted for brevity
]
```

Listing B.4: Scenario #2 Route Response JSON

B.3 Scenario #3

Below are the results from scenario #3, described in Section 4.3.2 and Table 4.4.

```
//Sent to ROUTE

"origin":{
  "latitude":38.7115605,
  "longitude":-9.1367243
},
"destination":{
  "latitude":38.7115605,
  "longitude":-9.1367243
},
"departureDate":1602756000000,
"visitationTime":180,
"budget":50,
"effortLevel":2,
"selectedPoints":[32],
"selectedCategories":[1,6],
```

Listing B.5: Scenario #3 Route Request JSON

```
//Received from ROUTE
"origin":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
 "destination":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
"time":{
      "startTime":1602756000000,
      "endTime":1602767565000
},
"durationTime":11577370,
"distance":2329.7214642941553,
"price":23,
"calories":117,
"sustainability":85,
"line":[
    {
        "latitude":38.7115624504062,
        "longitude":-9.136716261610735
    },
    ..., // Omitted for brevity
    {
        "latitude":38.7115624504062,
        "longitude":-9.136716261610735
    }
],
"pois":[
    {
        "timestamp":1602756273000,
        "poi": {
            "name":"Manteigaria Silva",
            ..., // Omitted for brevity
        }
    },
    {
        "timestamp":1602758185000,
        "poi": {
            "name":"Museu Nacional do Desporto",
            ..., // Omitted for brevity
        }
    },
    {
        "timestamp":1602763107000,
        "poi": {
            "name":"Museu Nacional de Arte Contemporânea do Chiado",
            ..., // Omitted for brevity
        }
    }
]
```

Listing B.6: Scenario #3 Route Response JSON

B.4 Scenario #4

Below are the results from scenario #4, described in Section 4.3.2 and Table 4.5. Note that this is meant to test a situation where it is impossible to create a trip due to the selected preferences and constraints, so the ROUTE microservice does not reply with a trip.

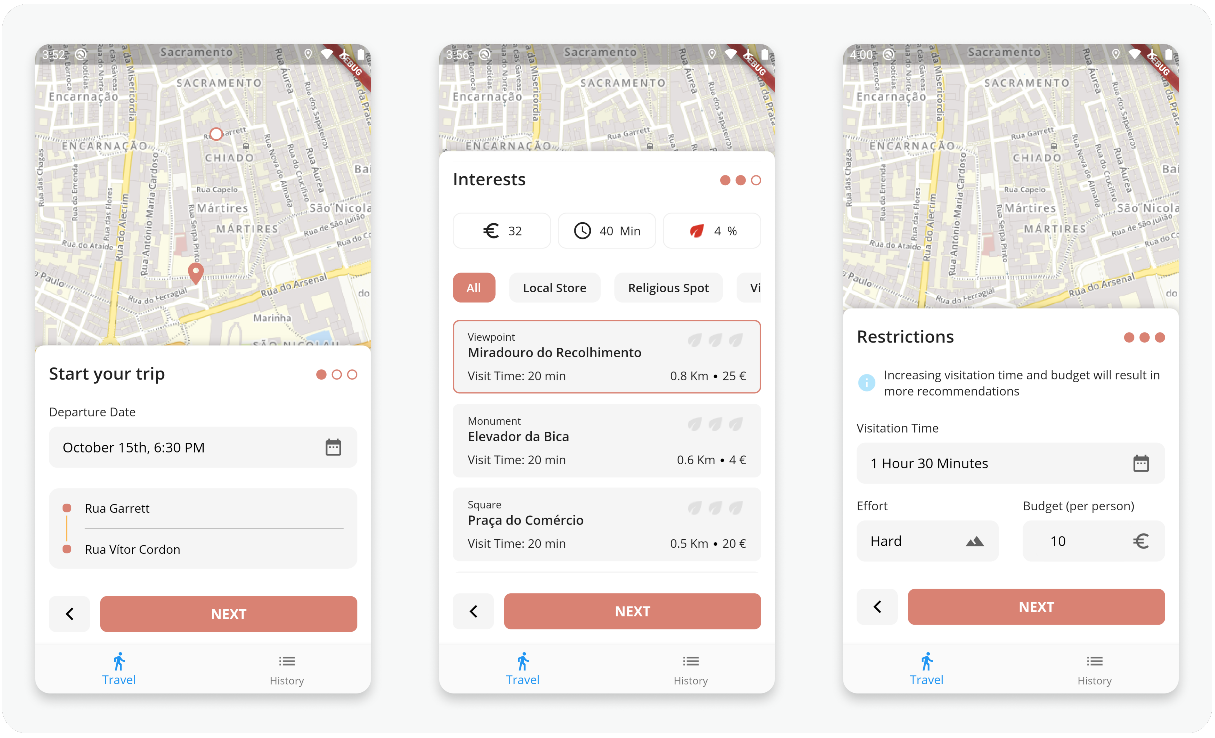


Figure B.5: Scenario #4 Trip Creation UI

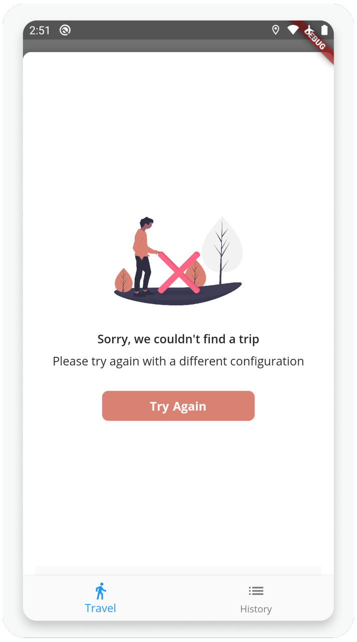

Sorry, we couldn't find a trip
Please try again with a different configuration
Try Again
Travel
History

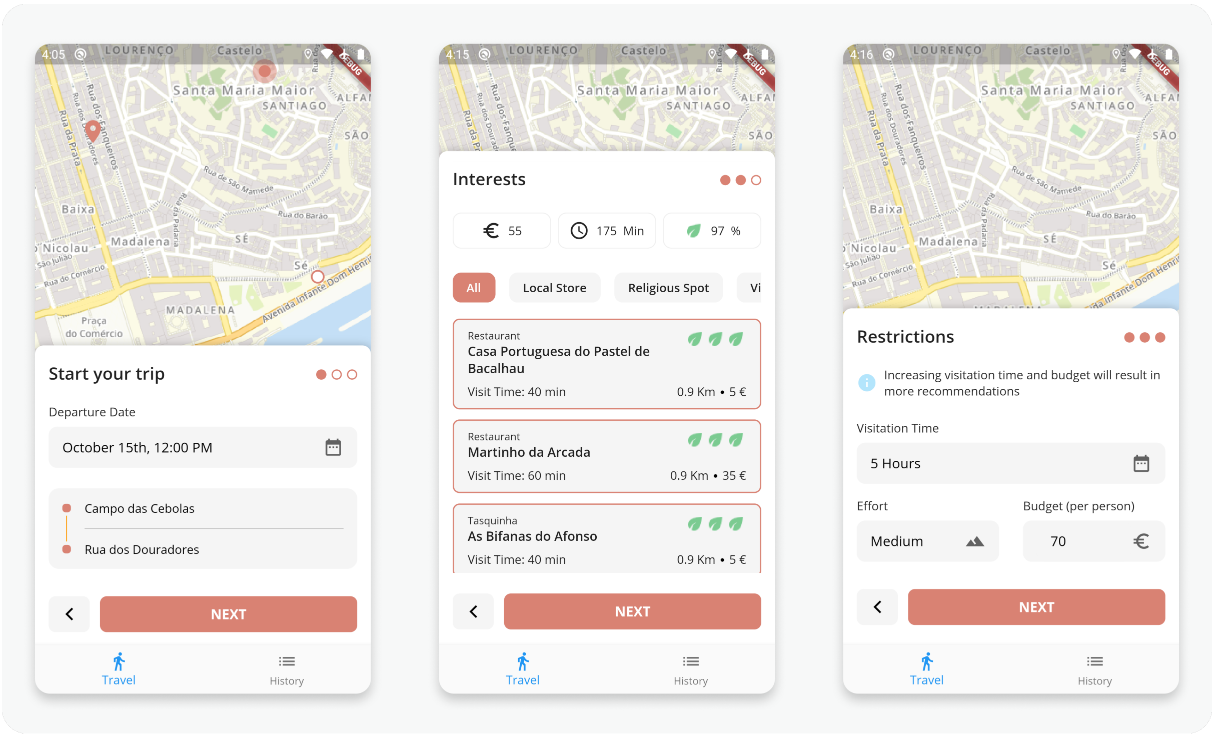
Figure B.6: Scenario #4 Error Trip UI

```
//Sent to ROUTE

"origin":{
  "latitude":38.710793100000004,
  "longitude":-9.140844399999999
},
"destination":{
  "latitude":38.7079863,
  "longitude":-9.141485
},
"departureDate":1602786600000,
"visitationTime":90,
"budget":10,
"effortLevel":3,
"selectedPoints":[19,38],
"selectedCategories":[1,2,3,4,5,6,7,8]

//Received from ROUTE

"codeJustiffication": "There is any possible recommendation regarding the chosen POIs,
    their schedule and the user available time"
```

Listing B.7: Scenario #4 JSON

B.5 Scenario #5

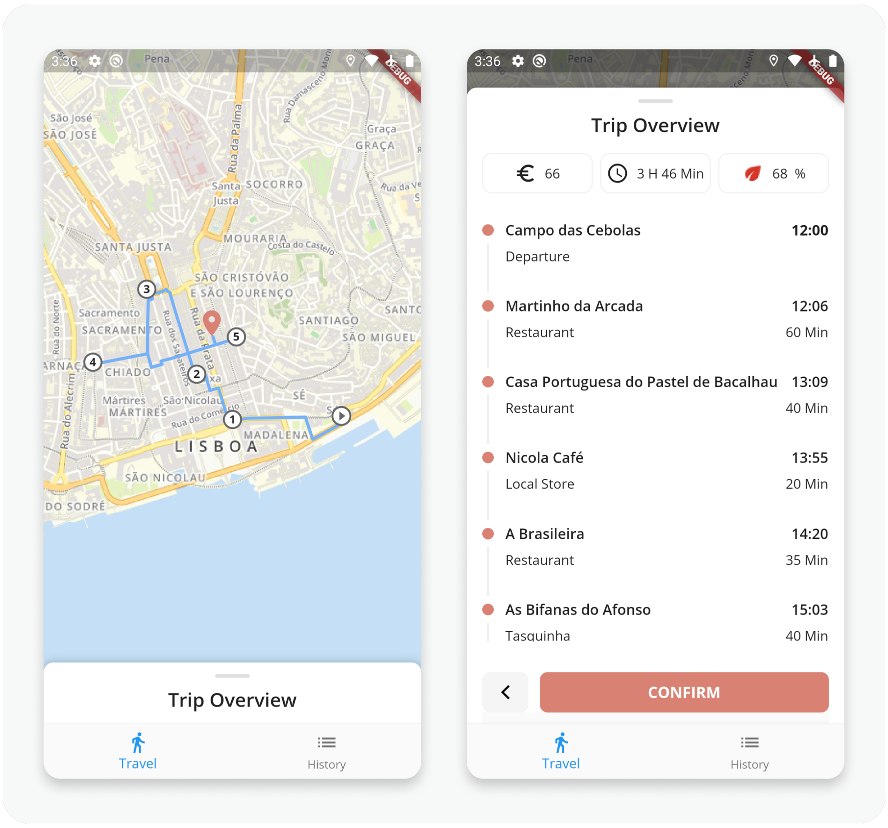
Below are the results from scenario #5, described in Section 4.3.2 and Table 4.6.


Start your trip
Departure Date
October 15th, 12:00 PM
Campo das Cebolas
Rua dos Douradores

Interests
55 | 175 Min | 97 %
All | Local Store | Religious Spot
Restaurant
Casa Portuguesa do Pastel de Bacalhau
Visit Time: 40 min 0.9 Km • 5 €
Restaurant
Martinho da Arcada
Visit Time: 60 min 0.9 Km • 35 €
Tasquinha
As Bifanas do Afonso
Visit Time: 40 min 0.9 Km • 5 €

Restrictions
Increasing visitation time and budget will result in more recommendations
Visitation Time
5 Hours
Effort
Medium
Budget (per person)
70

Figure B.7: Scenario #5 Trip Creation UI


3:36

Trip Overview

Travel

History

3:36

Trip Overview

66 | 3 H 46 Min | 68 %

Campo das Cebolas	12:00
Departure	
Martinho da Arcada	12:06
Restaurant	60 Min
Casa Portuguesa do Pastel de Bacalhau	13:09
Restaurant	40 Min
Nicola Café	13:55
Local Store	20 Min
A Brasileira	14:20
Restaurant	35 Min
As Bifanas do Afonso	15:03
Tasquinha	40 Min

CONFIRM

Travel

History

Figure B.8: Scenario #5 Trip Overview UI

```
//Sent to ROUTE

"origin":{
  "latitude":38.7087856,
  "longitude":-9.1309565
},
"destination":{
  "latitude":38.7115605,
  "longitude":-9.1367243
},
"departureDate":1602763200000,
"visitationTime":300,
"budget":70,
"effortLevel":2,
"selectedPoints":[22,38,34,36,33],
"selectedCategories":[1,2,3,4,5,6,7,8]
```

Listing B.8: Scenario #5 Route Request JSON

```
"origin":{
      "latitude":38.7087856,
      "longitude":-9.1309565
},
 "destination":{
      "latitude":38.7115605,
      "longitude":-9.1367243
},
"time":{
      "startTime":1602763200000,
      "endTime":1602776720000
},
"durationTime":13533855,
"distance":2547.0722801598895,
"price":66,
"calories":125,
"sustainability":68,
"line":[
    {
        "latitude":38.70860329900734,
        "longitude":-9.130819564102051
    },
    ..., //Omitted for brevity
    {
        "latitude":38.7115624504062,
        "longitude":-9.136716261610735
    }
],
"pois":[
    {
        "timestamp":1602763570000,
        "poi": {
            "name":"Martinho da Arcada",
            ..., //Omitted for brevity
        }
    },
    {
        "timestamp":1602767394000,
        "poi": {
            "name":"Casa Portuguesa do Pastel de Bacalhau",
            ..., //Omitted for brevity
        }
    },
    {
        "timestamp":1602770103000,
        "poi": {
            "name":"Nicola Café",
            ..., //Omitted for brevity
        }
    }
    ... //Omitted for brevity
]
```

Listing B.9: Scenario #5 Route Response JSON

Appendix C. Trip Creation Process

This appendix contains a detailed use case descriptions of the trip creation process, as discussed in Section 5.2.6.2.

SET PREFERENCES AND CONSTRAINTS
PRECONDITIONS:

- -

POSTCONDITIONS:

- User has filled the preferences and constraints;

MAIN SCENARIO:

1. APP presents a form with origin and destination and departure time;
2. User fills the form and the APP uses the departure time to filter out POIs according to their schedule;
3. APP presents the list of filtered POIs and categories which the user then selects to visit in his trip;
4. APP defines the minimum values for the budget and arrival time constraints according to the selection the user made;
5. APP requests the user for the final constraints: budget, arrival time, effort and if he the weather to be checked;
6. [**REQUEST ROUTE RECOMMENDATION**];

REQUEST ROUTE RECOMMENDATION
PRECONDITIONS:

- User has internet connection;
- User has filled the preferences and constraints;

POSTCONDITIONS:

- A route recommendation is presented to the user;

MAIN SCENARIO:

1. User asks the route recommendation;
2. APP sends the information to the ROUTE service;
3. *{User has no internet connection}*[**No Internet connection**];
4. [**PROVIDE ROUTE RECOMMENDATION**];
5. APP shows the route to the user on a map-based interface; which the user can accept or decline;
6. User accepts or declines the trip recommendation;

ALTERNATIVE SCENARIO [No Internet connection]

i. APP displays an error message informing that it requires Internet connection;
ii. Suggests the user to try again when Internet connection is available;
iii. Goes to step 1;

PROVIDE ROUTE RECOMMENDATION
PRECONDITIONS:

- *ROUTE* receives a route request;

POSTCONDITIONS:

- A route is created and sent to the mobile app;

MAIN SCENARIO:

1. *{The user selected the maximum number of POIs possible}* Go to step 11;
2. *{The user requests that the weather need to be checked}*[**Wants weather to be checked**];
3. ROUTE filters out POIs according to the user's categories of interest;
4. ROUTE checks the available time and budget;
5. *{No more time available left after visiting mandatory POIs}* Do not add new POIs to the route and goes to step 9;
6. *{Budget is totally spent on visiting the mandatory POIs}* [**Maximum budget is reached**];
7. Route filters out POIs that are not open during the available time period;
8. *{No POIs left to suggest}* Do not add new POIs to the route and goes to step 9;
9. Order POIs by sustainability level;
10. Add as many POIs to the trip as the constraints allow, from the top of the list;
11. ROUTE service creates a route that complies the requirements and sends it to APP;
12. Sends route to the mobile app;

ALTERNATIVE SCENARIO [Wants weather to be checked]

i. ROUTE checks the probability of raining during the available duration for the trip;

ii. *{Rain probability > 50%}* ROUTE filters out POIs not suitable to visit when it is raining;

ALTERNATIVE SCENARIO [Maximum budget is reached]

i. Filter out non-free POIs;

ii. *{number of suitable POIs = 0}* Goes to step 9;

iii. *{number of suitable POIs > 0}* Goes to step 11;

[This page has been intentionally left blank]

Appendix D. Tree Test Results

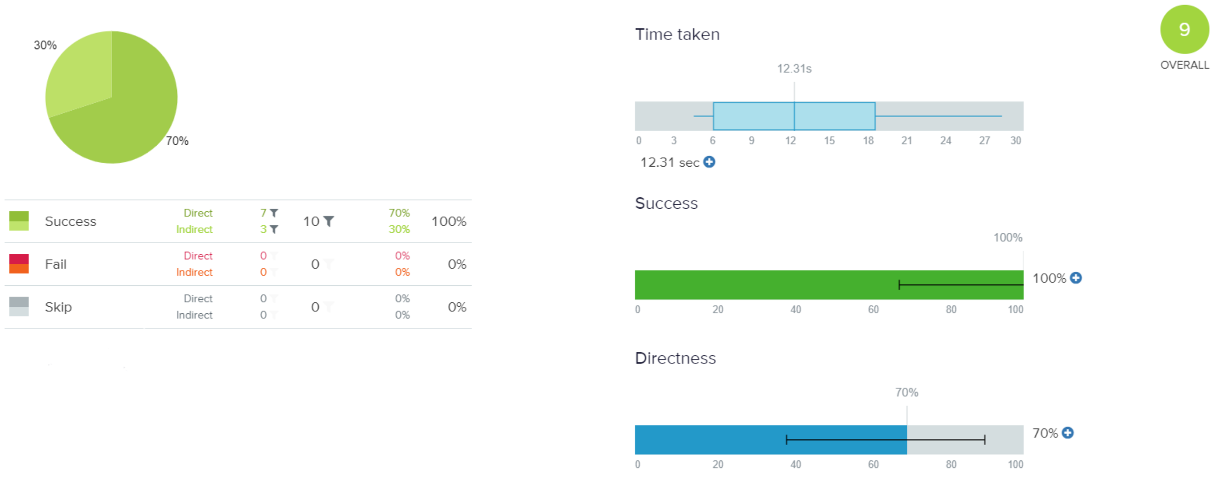
In this appendix we present the results of the tree testing techniques. These results include the overall rating of each task, as well as more detailed information such as: number of direct or indirect success and number of direct or indirect failure, the *directness* of the participants and the time taken to complete each task.

D.1 First iteration results

The first iteration of the tree test had a total of 3 tasks:

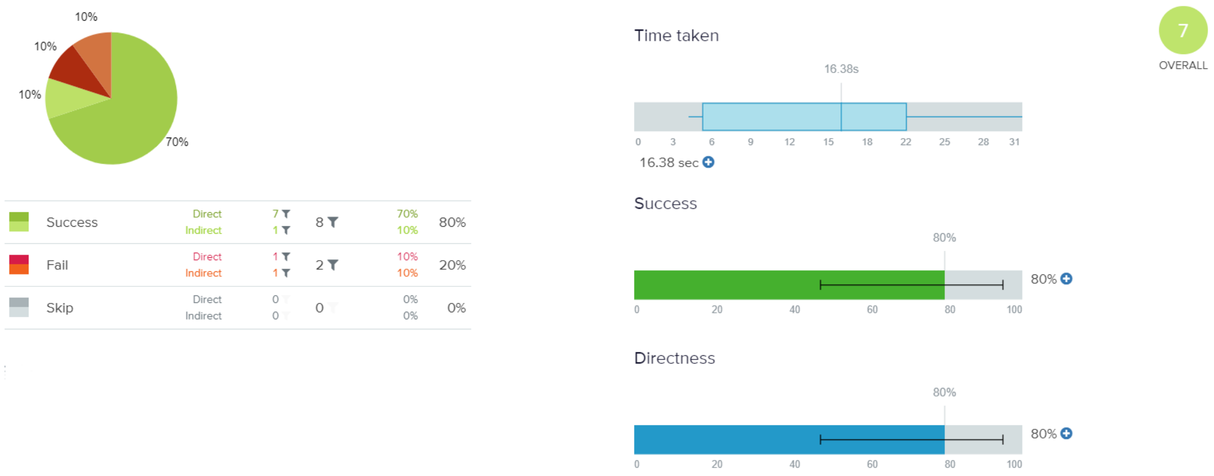
- Suppose you only have 50€ to travel. Where would you add this limitation?
- Imagine you want to start a trip departing from the nearest POI from your location. Demonstrate how you would do this.
- Imagine you recently finished a trip and want to share it with your friends on social network. Go to the spot where you will find this functionality.

The results of these tasks are presented below, in the same order they were introduced:



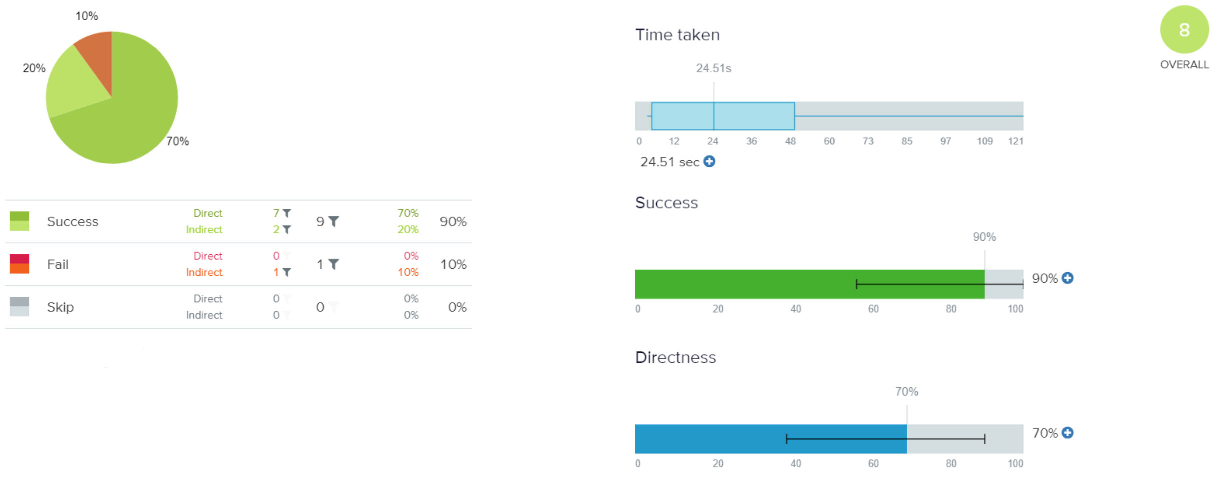
🟩	Success	Direct	7	10	70%	100%
		Indirect	3		30%	
🟥	Fail	Direct	0	0	0%	0%
		Indirect	0		0%	
⬜	Skip	Direct	0	0	0%	0%
		Indirect	0		0%	

Figure D.1: First iteration, task #1 results



🟩	Success	Direct	7	8	70%	80%
		Indirect	1		10%	
🟥	Fail	Direct	1	2	10%	20%
		Indirect	1		10%	
⬜	Skip	Direct	0	0	0%	0%
		Indirect	0		0%	

Figure D.2: First iteration, task #2 results



🟩	Success	Direct	7	9	70%	90%
		Indirect	2		20%	
🟥	Fail	Direct	0	1	0%	10%
		Indirect	1		10%	
⬜	Skip	Direct	0	0	0%	0%
		Indirect	0		0%	

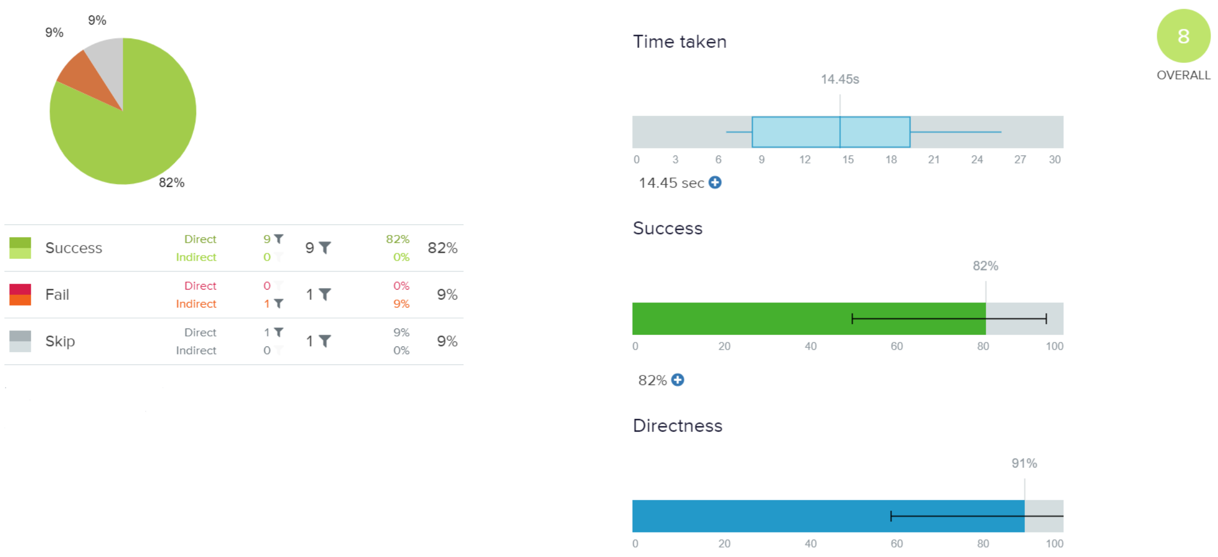
Figure D.3: First iteration, task #3 results

D.2 Second iteration results

The second iteration of the tree test had a total of 6 tasks:

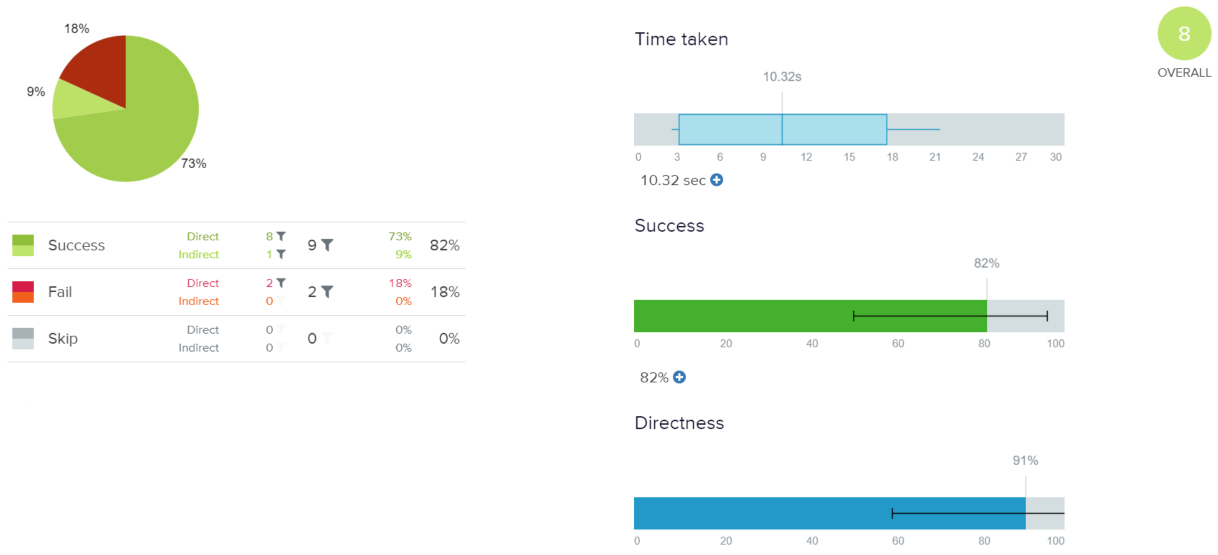
- Imagine you recently finished a trip and want to share it with your friends on social network. Go to the spot where you will find this functionality,
- Suppose you want to check your planned trip for tomorrow. Navigate to the place you would expect to find this information.
- You are a tourist that wants to go on a trip that ends at the place you are currently accommodated. Where would you add this restriction?
- Imagine you want to start a new trip that visits only churches. Where would you add this limitation.
- Suppose you want to check the price and visitation time of the *Museu do Fado* POI. Navigate to the place you would expect to find this information.
- Imagine that you want to check the total number of attractions you have visited using the app during your stay in Lisbon. Navigate to the place you would expect to find this information.

The results of these tasks are presented below, in the same order they were introduced:



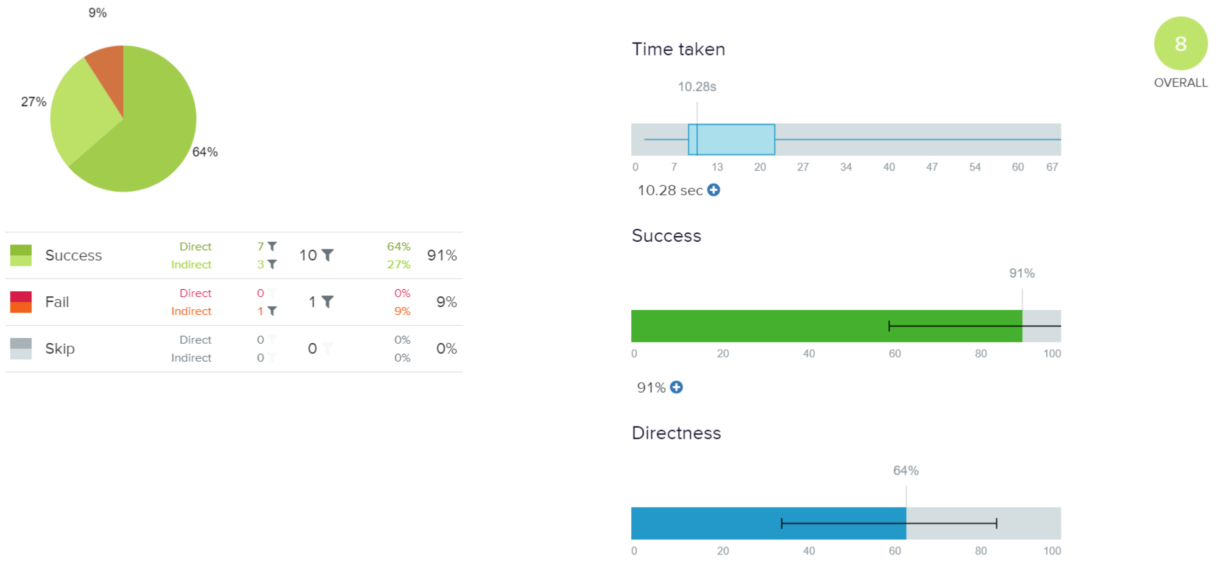
🟩 Success	Direct	9	9	82%	82%
	Indirect	0		0%	
🟥 Fail	Direct	0	1	0%	9%
	Indirect	1		9%	
▨ Skip	Direct	1	1	9%	9%
	Indirect	0		0%	

Figure D.4: Second iteration, task #1 results



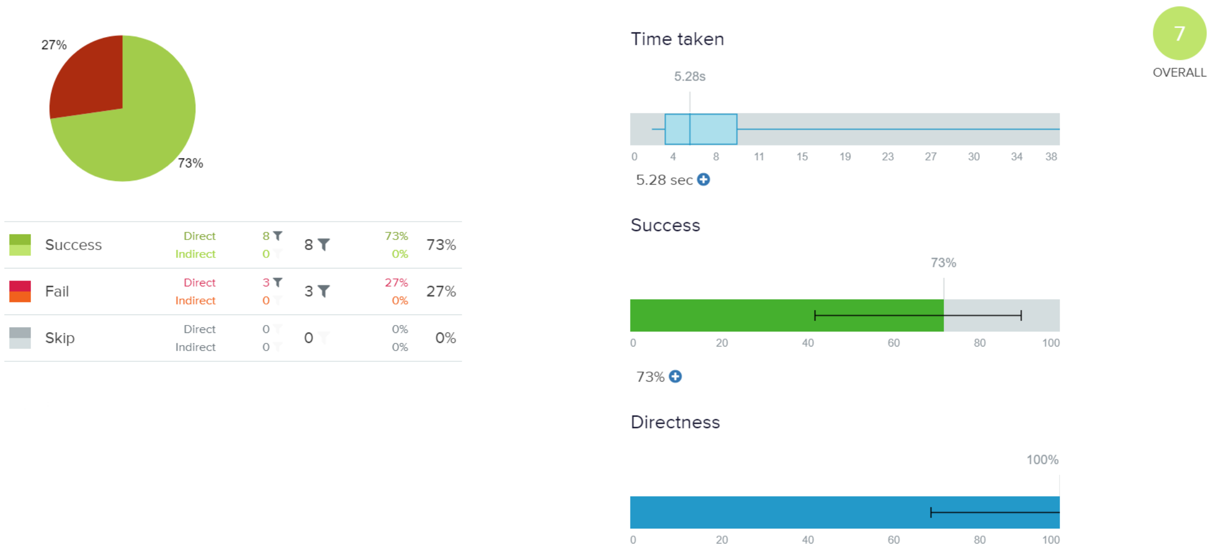
🟩 Success	Direct	8	9	73%	82%
	Indirect	1		9%	
🟥 Fail	Direct	2	2	18%	18%
	Indirect	0		0%	
▨ Skip	Direct	0	0	0%	0%
	Indirect	0		0%	

Figure D.5: Second iteration, task #2 results



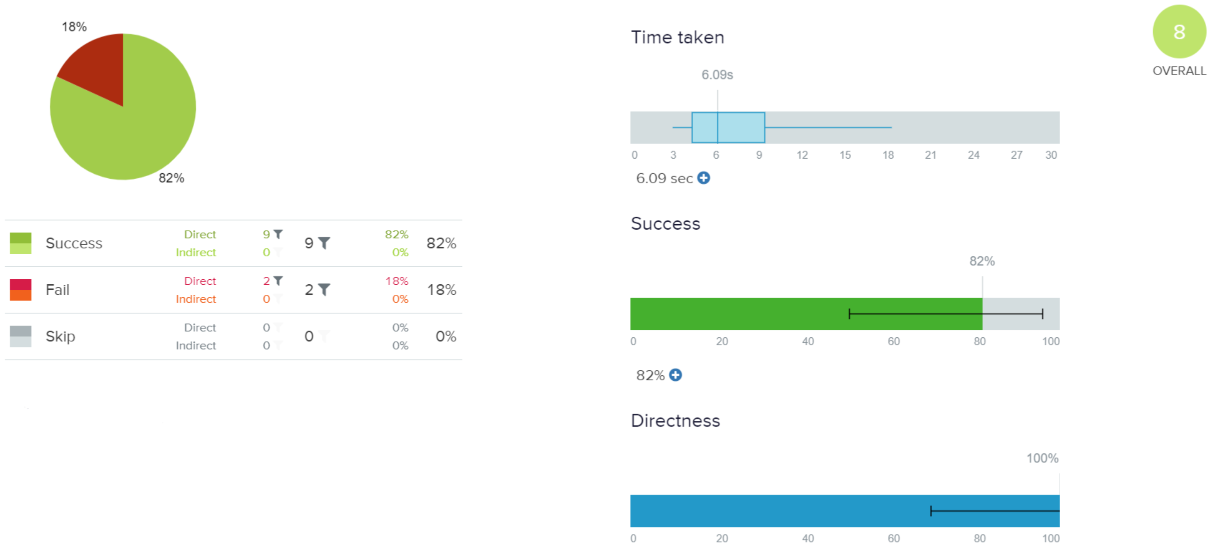
🟩 Success	Direct	7	10	64%	91%
	Indirect	3		27%	
🟥 Fail	Direct	0	1	0%	9%
	Indirect	1		9%	
▨ Skip	Direct	0	0	0%	0%
	Indirect	0		0%	

Figure D.6: Second iteration, task #3 results



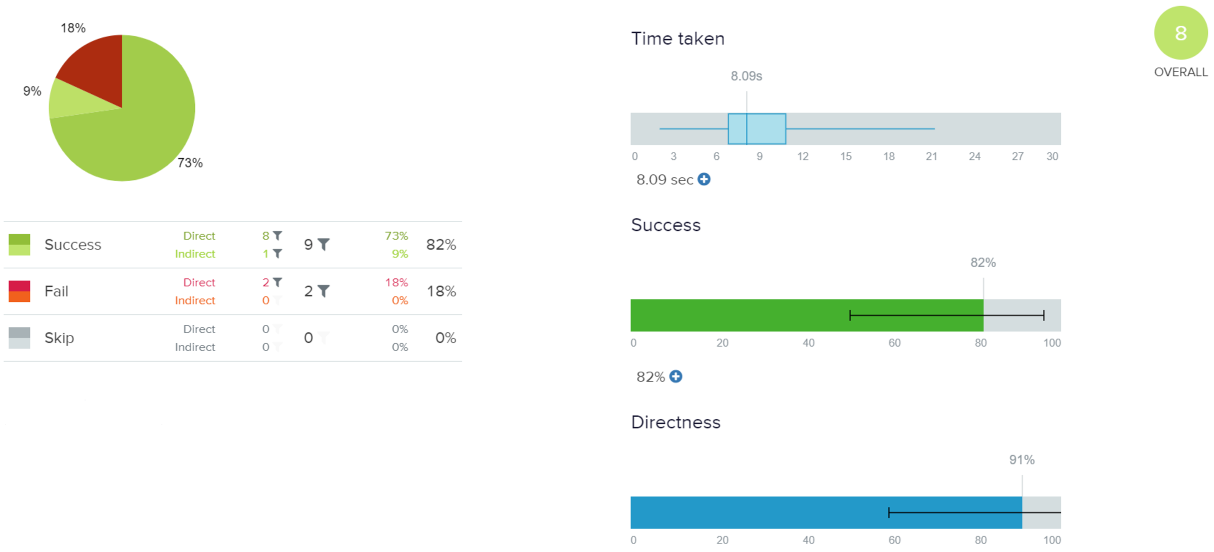
Success	Direct	8	8	73%	73%	
	Indirect	0		0%		
Fail	Direct	3	3	27%	27%	
	Indirect	0		0%		
Skip	Direct	0	0	0%	0%	
	Indirect	0		0%		

Figure D.7: Second iteration, task #4 results



Success	Direct	9	9	82%	82%	
	Indirect	0		0%		
Fail	Direct	2	2	18%	18%	
	Indirect	0		0%		
Skip	Direct	0	0	0%	0%	
	Indirect	0		0%		

Figure D.8: Second iteration, task #5 results



Success	Direct	8	9	73%	82%	
	Indirect	1		9%		
Fail	Direct	2	2	18%	18%	
	Indirect	0		0%		
Skip	Direct	0	0	0%	0%	
	Indirect	0		0%		

Figure D.9: Second iteration, task #6 results

[This page has been intentionally left blank]

Appendix E. Dynamic Time Warping

In this Appendix the results of the DTW algorithm validation mentioned in Section 6.2.1 are presented. The simulated routes are presented below in Figure E.1, which contains screenshots of the app, where the original recommendation route is represented by the continuous line in blue, and the tourist's path represented by the dotted line in red.

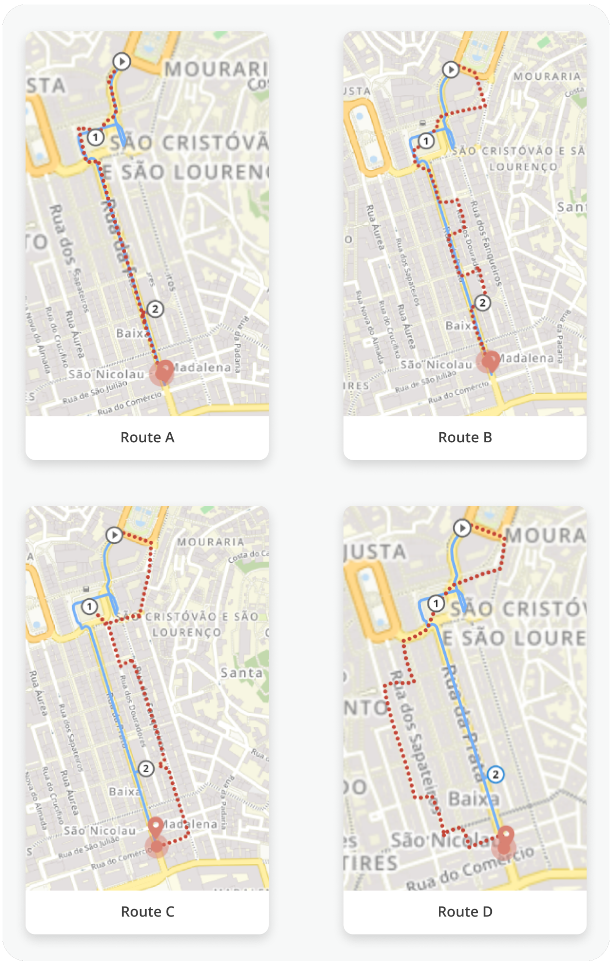


Figure E.1: DTW simulated paths

Route A is the closest to the original recommendation and route D is the furthest one. These travel paths were simulated using an *Android* emulator by sequentially changing the location of the virtual device to create a route. It is expected that the outcome of applying the DTW algorithm to these paths would result in a low score for route A and gradually increasing scores for the subsequent routes, with a high DTW score in route D. The real results of the DTW algorithm measured in the app, presented in Figure E.2, are aligned with the expected outcome, therefore validating that the app is indeed capable of measuring users' adherence to the trip recommendations.

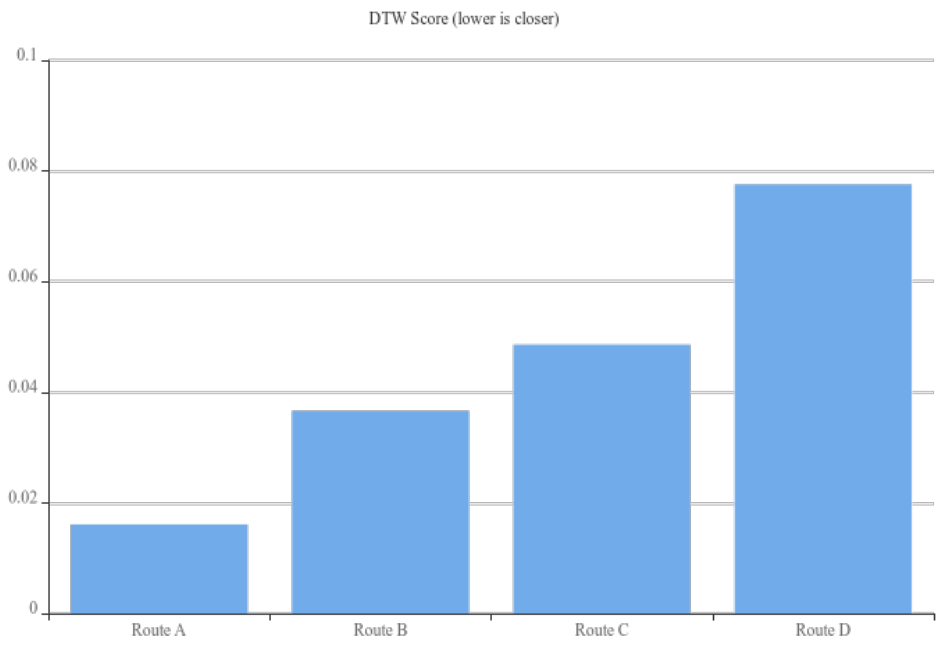


Figure E.2: DTW simulated paths

iscte
UNIVERSITY
INSTITUTE
OF LISBON

A mobile tour guide app for sustainable tourism

Filipe Larga