INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

GuideMe - Service for Consulting, Publication and Recommendation of Touristic Locations

ARTEM UMANETS

(Licenciado em Engenharia Informática e de Computadores)

Trabalho de projeto realizado para obtenção do grau
de Mestre em Engenharia Informática e de Computadores

Orientadores:
Mestre Artur Jorge Ferreira
Mestre Nuno Miguel da Costa de Sousa Leite

Júri:
Presidente: Mestre Fernando Manuel Gomes de Sousa
Vogais:
Mestre Artur Jorge Ferreira
Doutor Luís Filipe Graça Morgado
Mestre Luís Manuel da Costa Assunção

Novembro de 2013

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

GuideMe - Service for Consulting, Publication and Recommendation of Touristic Locations

ARTEM UMANETS

(Licenciado em Engenharia Informática e de Computadores)

Trabalho de projeto realizado para obtenção do grau
de Mestre em Engenharia Informática e de Computadores

Orientadores:
Mestre Artur Jorge Ferreira
Mestre Nuno Miguel da Costa de Sousa Leite

Júri:
Presidente: Mestre Fernando Manuel Gomes de Sousa
Vogais:
Mestre Artur Jorge Ferreira
Doutor Luís Filipe Graça Morgado
Mestre Luís Manuel da Costa Assunção

Novembro de 2013

Acknowledgments

I would like to express my gratitude to my supervisors Artur Ferreira and Nuno Leite, whose understanding, advices and orientation considerably increased my knowledge and overall quality of the developed project. I sincerely appreciate their assistance during the writing of the document in cause.

I would also like to thank my mother and father, Tetyana and Valery, for supporting my decisions during my entire life, which have helped to become the person I am. A heartfelt thanks goes out to my girlfriend Carina, for all your love, support and motivation that you have given.

Last, but not least, I would like to thank the Instituto Superior de Engenharia de Lisboa, which have been one of my main occupations for the past 5 years, and had prepared me for the IT Engineering world.

Resumo

Nos últimos anos, com a proliferação de dispositivos móveis, tais como *smartphones*, as pessoas estão em constante troca de informação. Por exemplo, no contexto de visitas turísticas, as pessoas têm sempre o *smartphone* presente, o que lhes possibilita a consulta da informação sobre o ponto turístico escolhido. Quando se visita um determinado local, a aplicação de guia turístico recomenda informações úteis para o utilizador. Estas recomendações são baseadas na localização atual do utilizador, nas suas preferências e visitas anteriores, possibilitando ainda que seja fornecida a opinião do mesmo sobre cada visita.

Neste relatório, abordamos o desenvolvimento da aplicação móvel e a correspondente aplicação *Web*, denominada GuideMe, para consulta, divulgação e recomendação de locais turísticos. A aplicação móvel foi desenvolvida para a plataforma iOS, seguindo o conceito de *Universal Application*, suportando os dispositivos iPhone, iPad e iPod Touch. O serviço na sua totalidade, oferece um conjunto de funcionalidades que melhoram a experiência do utilizador durante as visitas dos locais turísticos. O foco principal da aplicação móvel, é o de apresentar os locais turísticos na vizinhança da posição atual do utilizador. A informação sobre a localização atual é representada sob a forma de coordenadas geográficas (latitude e longitude), obtidos pelo dispositivo móvel através do equipamento Global Positioning System (GPS) ou através da triangulação de antenas WiFi. Os utilizadores também têm acesso a opções de filtragem únicas aquando da pesquisa dos locais turísticos. A pesquisa pode-se basear na sua disponibilidade para a visita, condições meteorológicas do local e altura preferida do dia (dia ou noite). Cada utilizador pode consultar os pontos turísticos, marca-los como visitados ou por visitar. Posteriormente, o sistema irá usar esta informação para calcular as recomendações dos locais que não foram previamente visitados pelo utilizador.

A infra-estrutura que suporta o serviço GuideMe implementa o próprio sistema de recomendação, que tem como objetivo analisar as visitas de cada utilizador e construir listas de recomendação baseando-se nos gostos mútuos entre os utilizadores. As recomendações produzidas são adequadas ao gosto do utilizador em causa. Com o sucesso do sistema de recomendação da Amazon, este tema tem sido o alvo do estudo e contribuição constante nos últimos anos. Através da literatura analisada, teve-se a oportunidade de estudar as diversas metodologias e algoritmos que compõem os sistemas de recomendação atuais. Analisamos as vantagens e desvantagens das seguintes abordagens e metodologias: Filtragem Baseada em Conteúdo (FBC); Filtragem Colaborativa baseada em Itens (FCBI); Filtragem Colab-

orativa baseada em Utilizadores (FCBU); Sistemas Híbridos. A literatura justifica que os sistemas mais robustos e de maior qualidade são baseados em abordagens híbridas. Na sua vertente simples, um sistema híbrido é composto pela junção do FBC com FCBI ou FCBU, que minimiza os pontos fracos de cada um dos sistemas.

Apesar da eficiência e da qualidade dos sistemas híbridos, devido à sua elevada complexidade de implementação, a abordagem escolhida para este projeto é a FCBI. Realizamos o estudo e a avaliação estatística dos diversos fatores (desempenho, tempo de execução e a dimensão de dados) que nos possibilitaram concluir que o algoritmo Slope One é o mais adequado para o sistema de recomendação em questão. A computação das recomendações é realizada periodicamente, agendada pelo *Cron job* para as 3:00 da manhã, em cada dia. O serviço seleciona um conjunto de utilizadores com as novas visitas e atualiza a sua lista de recomendações.

O serviço GuideMe foi implementado com uma arquitetura de 3 camadas, composta pela camada de dados, camada lógica de negócio e camada de apresentação. A primeira é responsável por armazenar a informação num modelo relacional MySQL. Qualquer operação sobre o modelo de dados é realizada através da camada de acesso a dados, implementada com recurso à *framework* Hibernate. Na camada de negócio posiciona-se a componente de sistema de recomendação e a RESTful API. A API expõe um conjunto de *endpoints* a serem consumidos pelas aplicações clientes, é responsável pela sua segurança, autenticação, formato de dados, entre outros. O serviço de recomendação encontra-se implementado sob a forma de uma biblioteca, enquanto que a API RESTful é implementada com recurso à *Play Framework* como uma aplicação *Web*. Ambas as componentes acedem aos dados usando a mesma camada de acesso a dados.

A estabilidade do serviço foi avaliada através de testes de carga. Estes ajudaram a detectar os pontos fracos do sistema, implicando melhorias a nível de estabilidade e robustez do serviços desenvolvidos.

A aplicação móvel e a aplicação Web, utilizam os mesmos *endpoints* da API RESTful para consulta, publicação e edição dos dados. Suportam a autenticação através dos serviços sociais comuns para a atualidade, nomeadamente Facebook e Twitter. Ambas as aplicações suportam o conceito de internacionalização. Atualmente encontram-se em Inglês e Português, mas a infra-estrutura implementada possibilita uma fácil adição de novos idiomas. A internacionalização constituiu um desafio no desenho da arquitetura do sistema, é uma funcionalidade que teve de ser integrada desde o modelo relacional, até à camada de acesso a dados, API RESTful e as respetivas aplicações Web e móvel.

A aplicação Web implementa um subconjunto de funcionalidades da aplicação móvel, esta foi desenvolvida com o objetivo de provar o correto funcionamento da RESTful API demonstrando que várias aplicações clientes conseguem consumir a mesma fonte de informação. Atualmente a interface do *backoffice* é implementada pela aplicação móvel que pode ser apenas acedida por utilizadores com privilégios especiais. Em modo de administração é possível realizar operações tais como: editar a informação de um local; adicionar novos locais; consultar a informação reportada pelos utilizadores sobre os dados incorrectos de um local; e visualizar os erros das diferentes componentes do serviço. No futuro, pretende-se

com que a aplicação Web tenha o mesmo conjunto de funcionalidades da aplicação móvel. Foi o que diz respeito ao *backoffice*, de modo a facilitar a gestão do sistema em algumas situações.

As aplicações desenvolvidas providenciam uma interface agradável e de simples utilização. Foi conduzida uma avaliação de usabilidade da aplicação móvel, que nos possibilitou identificar o público alvo, as suas preferências e a opinião que obtiveram ao utilizar a aplicação GuideMe. Na análise realizada, a avaliação geral foi positiva e a maioria das pessoas mostrou interesse em utilizar a aplicação no futuro.

É expectável que a solução atual seja posta em prática e que contribua de forma positiva para o aumento de turismo nos locais menos conhecidos.

Palavras Chave

GuideMe; Guia Turístico; Aplicação Móvel; Aplicação Web; Sistema de Recomendação; iOS; iPhone; iPad; REST

Abstract

In the past few years, with the proliferation of mobile devices such as smartphones, people are experiencing frequent communication and information exchange. For instance, in the context of tourist visits, it is often the case that each person carries out a smartphone, to consult information about a chosen point of interest. When one visits a given location, a suitable tourist guide application will recommend useful information to the tourist, based on its current location, preferences, past visits, further allowing the user to provide feedback about each visit.

In this report, we address the development of a mobile application and the corresponding Web application, named GuideMe, for consultation, publication, and recommendation of touristic locations. Users are offered features that improve their experience when they visit some touristic location. Each user may consult places of touristic interest, receive suggestions of previously unseen touristic places according to past users recommendations, and to perform its own recommendations. The GuideMe service offers a set of search filters, based on several criteria (such as the distance between user and point of interest location and the user's available time, among others) to facilitate the exploration of new locations. The key novelties of GuideMe, as compared to previous approaches, lies in the use of a recommendation process and the social interaction between users.

The evaluation of the different recommendation algorithms has contributed to the quality of the implemented recommender system. The load tests have helped to detect some weak spots and improve overall the stability and robustness of the developed service.

Keywords

GuideMe; Tourist Guide; Mobile Application; Web Application; Recommender System; iOS; iPhone; iPad; REST

Contents

List of Figures

List of Tables

List of Code Listings

1

Introduction

In this world, most of the people think that Artificial Intelligence was created to simulate human behavior and the way we humans think. Even thought people are not wrong, Artificial Intelligence was also created to solve problems that humans are unable to solve, or to solve it in a shorter amount of time, with a better solution. For example, any exhaustive search requires pretty complex algorithms (heuristics, meta-heuristics etc) with complex mathematical functions in order to find a feasible solution. Problems humans may take days to find a feasible solution, or may not find a solution at all that fits its needs, algorithms like these may deliver a very good solution in minutes, hours or days, depending on how much time the human is willing to use in order to get a better solution and not just a feasible one.

A concrete example is the creation of timetables for scheduling. Timetables can be used for educational purposes, sports scheduling, transportation timetabling etc. It may look like it's simple to create a timetable, but the truth is, this process normally requires following some rules in order to fit its purpose. These rules are called constraints in the timetabling subject, in which the more it has the harder it gets to create a feasible solution for the problem. In some cases, there are so many constraints in which some can be contradictory to each other, that it is needed to "relax the constraints" in order to solve the problem.

The process of creating a timetable requires that the final solution follows a set of constraints. These can be divided in two groups: hard constraints and soft constraints. Hard constraints are a set of rules which must be followed in order to get a feasible solution. In the other hand, soft constraints may be called "optional" since there is no need to follow these to get a solution, because they are not mandatory. The quality of a solution is calculated by using these soft constraints. In another words: the more soft constraints a solution follows, the better it is. This quality is based on the points (weight) of the soft constraints that that solution didn't follow, so the more points, the worse the solution. The weight of each soft constraint is set by the one that created the constraints.

Now that I talked a little bit about timetabling, let me specify the aim of this project. Its main is to create an examination timetable generator using ITC-2007 specifications. The solutions will be validated using a validator also created by me which specifies the quality of the solution and may do some corrections on the timetable in order to get an even better solution. It is also required that the final product may work with two test seasons which

can be considered an extension to ITC-2007 formulation. In the end an (optional) Graphical User Interface will be created in order to allow the user to edit the current solution to fit the users needs and allow optimization to the edited solution. The generator will be tested using data from ITC-2007 and some actual data from six different programs presented in my university ISEL.

1.1 Tourist Guides: State of the Art

In this Section, we briefly review the state of the art regarding tourist guides. As of time of this writing (November 2013), there are a few services that provide some interesting functionalities regarding tourist assistance. In the following Subsections we describe the key features of these services.

1.1.1 TouristEye

One of the most relevant services is the TouristEye [**?**] and it is available as a Web application, with mobile clients for iPhone [**?**] and Android [**?**]. The TouristEye service offers a wide range of points of interest organized by categories (attractions, restaurants, entertainment, among others). Registered users have the possibility to mark touristic places as visited and leave a comment by indicating their degree of satisfaction (*I love it*, *I like it*, *It's ok* or *I didn't like it*). In addition, users have the possibility to write down their travel journal by taking notes and photos exclusively in the mobile application.

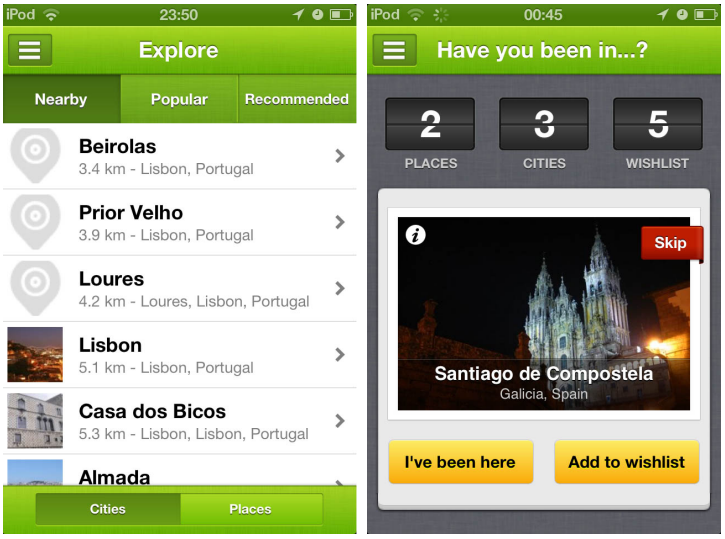
In order to visualize points of interest with accurate perspective, the Google Maps [**?**] service is integrated within the Web and mobile applications. Users can plan their trips, which are composed by various points of interest, whereas the map service is used to display routes between these locations.

The core functionalities of the TouristEye application (iPhone version) are shown on Figure 1.1.

With this application, users have the possibility to list nearby, popular, and recommended places as illustrated on Figure 1.1 (a). The screenshot on Figure 1.1 (b) illustrates the view where popular locations are suggested to the user. By choosing the **Add to wishlist** option, the location is added to the list of places that the user would like to visit. Later, this information is used to recommend new places. The recommendation engine is using a mix of Item-Based and User-Based Collaborative Filtering algorithms [**?**].

1.1.2 GuidePal Offline City Guides

Another tourist guide application with some interesting features is the GuidePal Offline City Guides [**?**][**?**]. It allows users to download varied content for different cities and to consult information regarding restaurants, coffee shops, places to visit, and other attractions. Two screenshots of this application are illustrated on Figure 1.2.

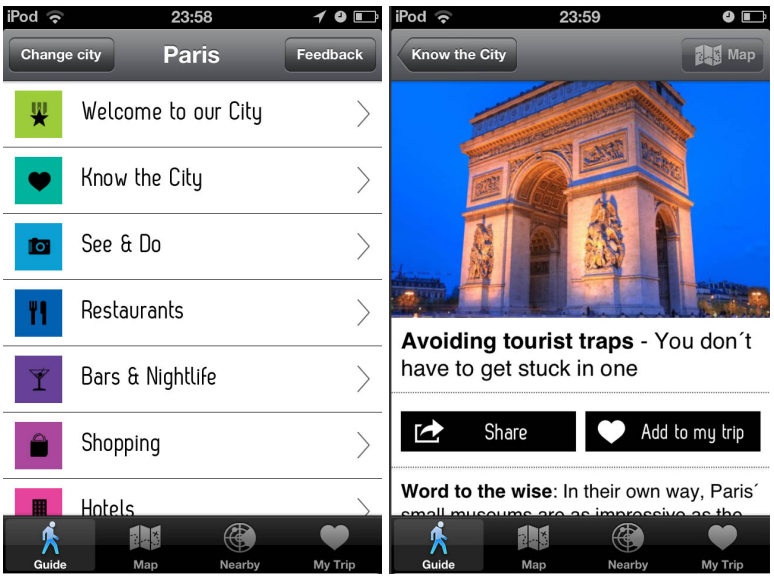


(a) Exploration of the places near user location.

(b) Preview of the suggested location.

Figure 1.1: TouristEye [**?**] iPhone application screenshots.

In order to list the existing points of interest, the user selects the desired city and category as illustrated by Figure 1.2 (a). After selecting the intended category, a description for the



(a) Category selection for filtering the locations.

(b) Detailed information about touristic location.

Figure 1.2: GuidePal [**?**][**?**] iPhone application screenshots.

point of interest is presented; Figure 1.2 (b) shows the selection of the "Know the City" category.

This application also have some augmented reality features, providing users with the possibility to map the real world (captured by the camera of the device) with the touristic attractions in real time. When the camera points to the direction where touristic locations are available near by, those are presented above the image. By selecting any of the presented locations, the user is redirected to the detail page similar to one illustrated on Figure 1.2 (b).

1.1.3 mTrip

mTrip [**?**] travel guide service is mainly used for big cities such as Paris, Berlin, Madrid, and others. It is available as a separate application for each one of the major cities (for both iPhone and Android) and allows people to consult information regarding points of interest without an Internet connection. Figure 1.3 presents some screenshots of this application.

Users can plan an itinerary or create guides for the cities by providing the detailed information on the touristic attractions they plan to visit. After specifying the dates for the trip, each user can manually compose his itinerary or allow mTrip to automatically generate an itinerary for the trip. When the automatic mode is chosen, one must indicate preferences for the attractions to be visited, as shown on Figure 1.3 (a). A list of points of interest is presented when the user selects a specific category, similar to those shown on Figure 1.3 (b).

(a) Customization of the itinerary.

(b) List of the touristic locations.

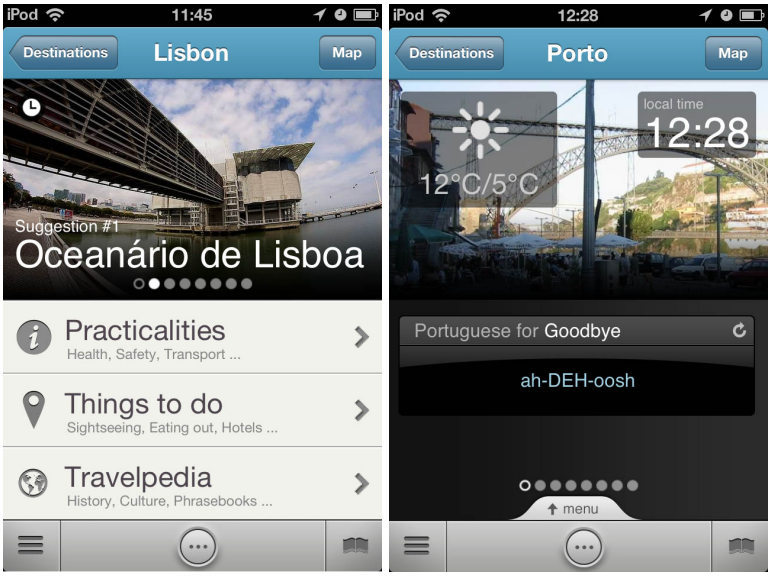
Figure 1.3: mTrip iPhone application screenshots.

Each point of interest is accompanied by a description, a photo, opening hour, prices, as well as the comments and ratings from other travellers. The application provides an augmented reality tool that allows to preview of the points of interest around the user's current location.

1.1.4 Triposo

Triposo [**?**] shares similar concepts with the mTrip application. However, it includes much more countries as well as smaller cities. When one picks the country to visit, the download of information regarding the points of interest for that country starts immediately, allowing to consult this information later in offline mode. Some major cities require additional downloads due to the large amount of the points of interest and touristic attractions. For big cities, users have access to special information regarding the city guide about all sights, a list of restaurants and extended nightlife options. The application also offers a travel dashboard with currency converter, weather, and useful native language phrases. Triposo uses freely available content from different sources, such as Wikipedia [**?**], Wikitravel [**?**], World66 [**?**], among others.

As it is shown by the screenshot on Figure 1.4 (a), a set of suggested points of interest for Lisbon is presented along with the main navigation items. Figure 1.4 (b) shows the current weather and current time for Porto as well as the spelling of some Portuguese words.



(a) Suggested points of interest.

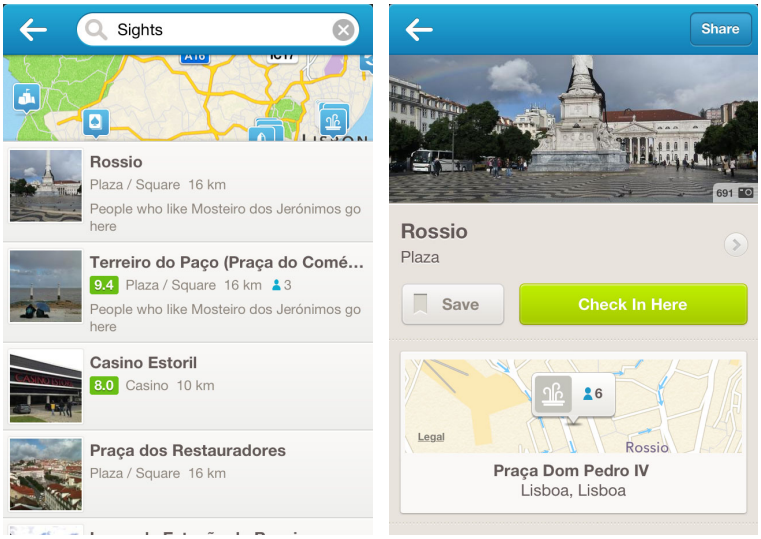
(b) Shortcuts and useful information.

Figure 1.4: Triposo iPhone application screenshots.

1.1.5 Foursquare

Foursquare is a service that allows registered users to "check-in[1]" at their current location. It is composed by both web and mobile applications for iPhone, Android and Blackberry. Users with special permission have the possibility to contribute with new locations, such as coffee shops, restaurants, sights, among others. The service was created in 2009 and in March of 2011 a new version with a recommendation engine [**?**] was developed, for suggesting places that users might like, based on their past actions.

In 2013 an updated version of the service was launched, where users can consult the sights nearby their current location. The map view with the recommended sights nearby users current location is shown on Figure 1.5 (a). An example of the touristic location details is shown on Figure 1.5 (b) where user can have the preview of the location, save it to favourites, perform the check in, among others.



(a) Map view with sights nearby user's location.

(b) Details of the Touristic location.

Figure 1.5: Foursquare iPhone application screenshots.

All these applications are modern solutions for tourist guides. The Foursquare service was created in 2009, the mTrip and TouristEye in 2010, whereas the Triposo and GuidePal have been published in 2011.

[1] Check-in refers to the user's action for posting his current location at a venue.

1.2 Proposed Solution

This Section introduces the simplified service architecture as well as the employed technologies used during the development of the system components. The system is mainly composed by the REpresentational State Transfer (REST) [**?**] service with a data access layer which exposes a set of endpoints[2] accessible by the client applications. Client applications are available as Web-based and Mobile (for iOS [**?**] platforms iPhone and iPad). Figure 1.6 shows the proposed service architecture.

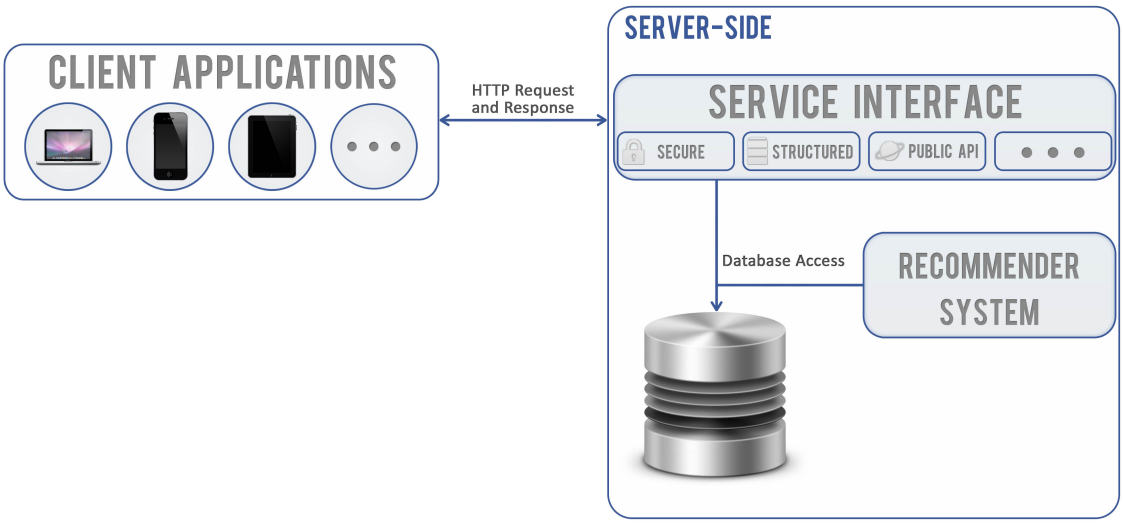


Figure 1.6: Generic service architecture.

The service is divided into four main layers, with the following roles:

- The Database layer stores information regarding points of interest, user data, among other types of data.
- The Service Interface is responsible for communicating with the Database layer. It makes all accesses transparent to the end client applications. This layer is mainly responsible for the data security and integrity, delivery of information in structured format, among others.
- The Recommender System is responsible for finding new points of interest based on the user's past actions, and to recommend these to the user. This service uses data from the database layer in order to compute recommendations.
- Client Applications make HyperText Transfer Protocol (HTTP) requests in order to access data from the public Application Programming Interface (API), provided by the Service Interface layer.

[2] In Service Oriented Architecture (SOA), an endpoint is the entry point to a service.

1.2.1 Tools and Technologies

This Subsection introduces the technologies and tools used to achieve the goals of this project. We begin by describing a set of technologies related to databases, data model, and the REST service. Then, the Web related technologies are discussed and finally we address the mobile applications.

For information storage, we use the MySQL [**?**] relational database, which uses the Structured Query Language (SQL) for accessing and manipulating databases. In order to facilitate the service development process, a pre-populated database was created with a few tens of points of interest.

The REST API is developed in order to provide a single data source for client applications, including those already developed within this project and others that may appear in the future. The API is implemented using the Java environment and Play Framework [**?**].

The Web application implementation is focused in technologies for user interface definition and structuring such as HyperText Markup Language version 5 (HTML5) [**?**], Cascading Style Sheets version 3 (CSS3) [**?**] and also the JavaScript [**?**] technology to enrich user's interactivity. The server side of the web application was developed using the Play Framework. Twitter Bootstrap [**?**] framework was used to generate the layout with desired dynamic content, thus providing an user friendly platform.

In the past few years, mobile devices have been used and widespread with great success, starting with Apple's iPhone launch and later with the iPad [**?**]. Both devices share the same operating system named iOS that offers a development platform for object-oriented applications. Those are written in Objective-C [**?**][3] using the iOS SDK (Software Development Kit). In the course of this project the targeted applications were designed taking into consideration advanced aspects of the iOS platform such as sharing code between iPhone and iPad, by means of an Universal Application [**?**].

The development of these applications was carried out using the Integrated Development Environment (IDE) Xcode [**?**].

1.2.2 Core Features

This Subsection contains a description of the core features provided by the service, which are presented in Table 1.1.

[3] The Objective-C is a computer programming language based on C and Smalltalk. It is designed to leverage the C language with full object-oriented programming capability, through the mechanism of message passing. When in Objective-C one sends a message, its target is resolved at runtime and then it interprets the message with the receiving object [**?**] [**?**].

Table 1.1: Core features of the GuideMe service.

Feature Name	**Description**
Internationalization support	The aim of the project is to support points of interest of Portugal, but its architecture is designed in order to easily be expanded and to support other countries. Multilingual support is implemented for the database, REST API and application levels.
Filtering options	The following features are supported: • lookup for points of interest nearby the user's current location; • filtering of the locations based on the following characteristics: – specific attraction; – user's current location, by using the geographic coordinates retrieved from the mobile device; – time that the user can spend; – time of day of the visit; – preferred weather conditions. • places that can be visited under given weather conditions such as rainy or sunny days, among others. It requires integration with the World Weather Online (WWO) meteorology API, in order to advise or not the visit to the chosen user location; • filtering by the maximum distance to the point of interest, based on the user's current location and mobility.
Wanted and already seen	Users have the possibility to create a list of locations that they intend to visit as well as access the history of already visited locations.
Social component	The following features are supported: • Integration with social networks in order to encourage users to share visits with their friends; • Based on the visits and mutual preferences between various users, the system recommends new places to visit that users may like. The recommendations are produced by the recommender system, using Collaborative Filtering (CF) [**?**][**?**][**?**] algorithms. The proposed solution is further explained in Section 2.3; • The service implements the social component by itself, providing the concept of the *Following* and *Follower* in order to establish friendship between users. Users are notified about certain actions performed by their friends.
Notification system	Users can receive push notifications on their iOS device when someone has started to follow them. These notifications are carried out using the Apple Push Notification Service (APNS) [**?**].
Login and sign up	Authentication and sign up through the most popular social network services, such as Facebook [**?**] and Twitter [**?**].

Back office	The iOS applications implements a back office system where privileged users can create or edit an existing location. They have permission to consult log information of the different service components, to analyse and correct information regarding the touristic location that were reported by other users.
Near by	Preview of the sights, near by user's geographic locations. The use-case of this feature is illustrated on Figure 1.7.
Recommended touristic locations	The service performs recommendation of the touristic locations to users, based on their past visits. The detailed illustration of this scenario is shown on Figure 1.8.

The use-case illustrated on Figure 1.7 consists in providing the user with information regarding the sights near by his/her current location.

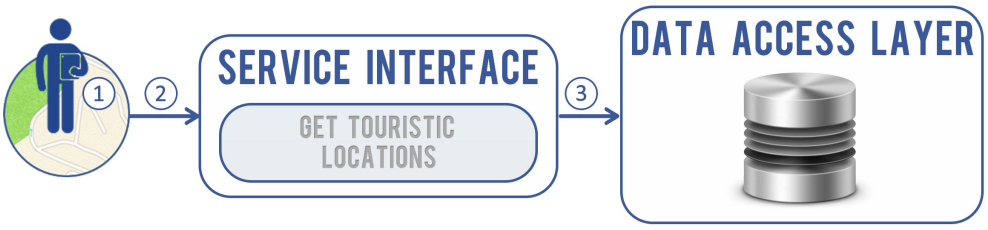


Figure 1.7: Use-case of the near by feature.

Each step is detailed as follows:

1. The geographic coordinates of user's current location, are obtained through the Global Positioning System (GPS) of the mobile device.
2. Through the service interface, users make the requests to obtain the touristic locations passing the previously obtained geographic coordinates.
3. The service interface interacts with the Data Access Layer (DAL) in order to list the touristic locations. The distance is calculated between each sight and the submitted geographic coordinates, providing the results sorted in ascending order of distance.

The recommender system is one of the most important features of the developed service. The summarised process of the recommendation system is illustrated on Figure 1.8.

The full process of the recommendation system is as follows:

1. Through the service interface, the user marks new locations as visited.
2. Using the implemented DAL, the information is stored into the database.
3. The Cron Job[4] **[?]** is triggered everyday at 3:00 A.M., performing the recommendation process.

[4] Cron is a time-based job scheduler in Unix-like computer operating systems.

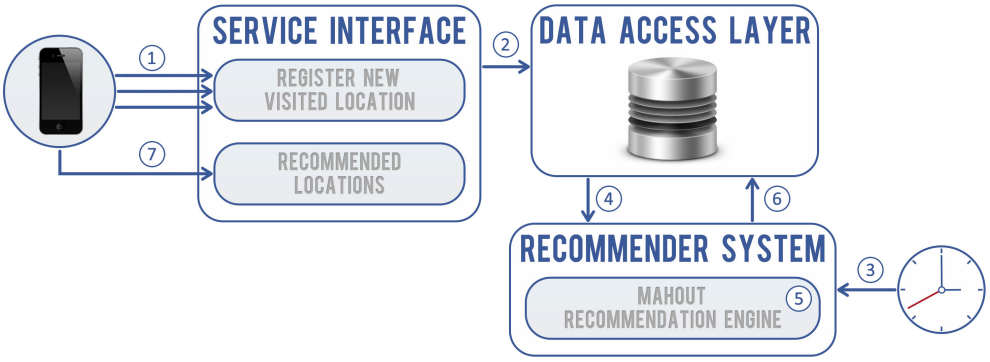


Figure 1.8: The recommendation process.

4. The recommender system obtains users eligible for the new recommendations.
5. For each user, the recommender system computes a set of locations previously unseen by the user.
6. Through the DAL, the recommender service removes previous recommendations and populates the database with the new information.
7. Later, the user obtains a list of recommendations through the service interface.

When comparing the GuideMe service with similar applications reviewed in Section 1.1, it offers:

- filters of unique kind that allow searching for points of interest;
- a friendly and usable interface for both Web and mobile applications;
- integration with algorithms and services that minimize human interaction during the approval process of newly inserted locations.

1.3 Report Organization

The remainder of this report is organized as follows. In Chapter 2, we introduce the background concepts and terminology of the recommender system and different collaborative filtering techniques. Chapter 3 approaches the proposed solution by detailing: the chosen service architecture, the integration with the APNS service, the analysis of different frameworks for the implementation of the Data Access Layer (DAL), and the REST API. Chapter 3 also includes prototypes that were initially designed before the development of the client applications.

In Chapter 4, we describe all the developments that were made for making this project a reality. In detail, this Chapter describes the implementation aspects of the database models for both Service and Log databases, structure of the the REST API endpoints, different characteristics of the implemented recommender system, and push notification provider. We introduce the main features, as also some implementation details of the web and iOS applications. Chapter 5 is devoted to the experimental results, which have guided the implementations in the correct direction. This Chapter reflects an analysis of the different recommendation system algorithms and load tests.

We end the document with the Chapter 6 by detailing the core aspects of the project and the fulfilled objectives. In order to make the service more usable and robust, as future work we present some points that should be improved in terms of security and functionalities.

1.4 Our Contribution

We've recently submitted an article [**?**] to the 2013 Conference on Electronics, Telecommunications and Computers (CETC 2013), where we describe the main parts of this work. We believe that the developed project contribute to the evolution of tourist guide services, and promotes the effective integration of recommendation systems as a tool to easily find new interesting places to visit.

2

Recommender Systems

Very often, users do not know the touristic points they would like to visit, the overwhelming amount of data requires mechanisms for efficient information filtering. The aim of the recommender system is to find sights that might interest users based on their past visits, and provide them with more relevant content.

In this Chapter, we introduce some background concept about Recommender Systems (RS) and their two different implementation approaches. In the following Sections, we describe the main goals of the recommender system and how it is used in our problem. We end up by justifying a chosen algorithm type for our recommender system implementation.

2.1 Introduction

Recommender System (RS) are used to generate meaningful information to a collection of users for items or products that might interest them. Nowadays, they are widely used, for example in electronic commerce company Amazon [**?**], which recommends additional items of what other shoppers bought along with the currently selected item. For instance, the Spotify [**?**] music service implements the RS which suggests new music to users based on their taste. There are many other services using this approach, because it makes easier the discovery of meaningful information without any additional effort by the user [**?**]. In our case, items are the points of interest to be recommended to users based on their past visits. Figure 2.1 depicts different approaches and algorithm types of RS.

Figure 2.1: Recommender systems hierarchy.

The recommendation process can be done through the analysis of the items characteristics,

named this as Content-Based Filtering (CBF). Another approach, designated as Collaborative Filtering (CF), consists in the collaboration between different users, and is divided into two main categories:

- User-Based Collaborative Filtering (UBCF) is used with memory-based type of algorithms. Consists in using user's rating in order to compute recommendations.
- Item-Based Collaborative Filtering (IBCF) uses a model-based methodology. These type of RS are developed using the data mining algorithms to find patterns based on training data.

In the following Sections, we describe each of these approaches, providing a comparative analysis.

Table 2.1 lists the symbols associated to CF and CBF terminology.

Table 2.1: List of frequently used symbols.

Symbol	Designation
I	A collection of n items $\{i_1, i_2, \ldots, i_n\}$
U	A set of m users $\{u_1, u_2, \ldots, u_m\}$
u_a	An active user
I_{u_a}	A set of items that belong to u_a
SIM	The similarity matrix of the items
$sim_{i,j}$	Similarity between two distinct items i and j
$P_{u,i}$	Prediction for user u and item i
$\bar{R}_u$	The average of the u-th user's ratings
$R_{u,i}$	The rating value for item i rated by user u

2.2 Overview of Content-based Filtering

CBF methods are based on the information about the characteristics of the items to be recommended. These systems can evolve and learn user preferences from user's actions, but they are limited to recommend content of the same type as that the user is already using. In order to have an accurate recommender system with this kind of algorithms, the item should have a large set of characteristics associated to it, in order to establish a degree of similarity between two different items. As presented in Figure 2.2, the input data of the algorithm is a vector of items $I = \{i_1, i_2, \ldots, i_n\}$. The similarity computation is made between distinct pairs of values of the vector which then produces a similarity matrix filled with values representing degrees of similarity $SIM = \{sim(i_1, i_2), sim(i_1, \ldots), \ldots, sim(i_2, i_3), \ldots, sim(i_{n-1}, i_n)\}$ between each pair. Finally, the recommendation process selects the top results as suggested items.

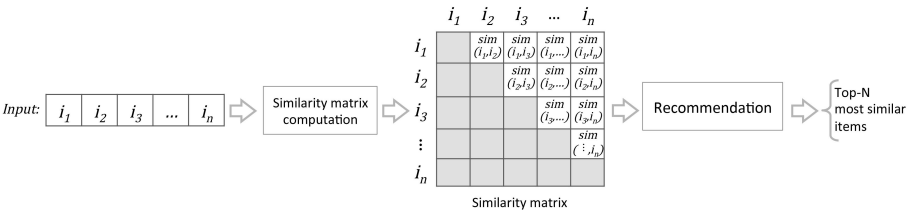


Figure 2.2: The Content-Based Filtering process. Creation of the similarity matrix and computation of the top most similar items.

For instance, the Pandora Radio [**?**] uses a subset of approximately 400 different song characteristics in order to recommend the "radio station" that plays music with similar properties to that preferred by the user. On the other hand, user's feedback is used to adjust the algorithm whenever one likes or dislikes some music [**?**].

For the current problem of a tourist guide development, in order to use, integrate, and adopt CBF algorithms in our work, we could use a large set of attributes to characterize points of interest, such as the landscape, urban, nature, mountain, monument, and so on. Figure 2.3 illustrates an example of a CBF recommender system with one user and three distinct locations.

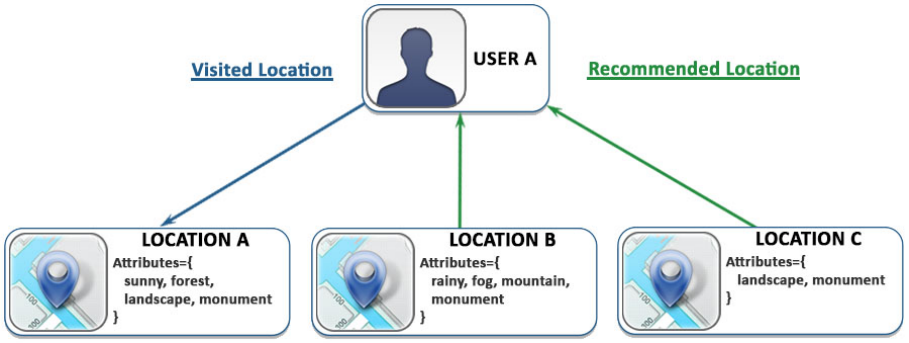


Figure 2.3: Example of a recommender system process using a CBF approach.

Assuming that `User A` visits the `Location A`, that same user will receive a recommendation to both `Location B` and `Location C` since it shares the same attribute `Landscape` and `Monument` with the visited location. For the current problem, it is possible to find a lot of attributes that will characterize the point of interest. A possible problem lies in the human factor. It is that users have the possibility to insert new locations. The recommender system may have a negative impact by suggesting wrong locations when users make a mistake on the attribute definition during the location creation phase.

Figure 2.4 shows an example of the CBF algorithm which refers to the information rep-

resented in Figure 2.3. As input we have an array of locations, which is used to compute

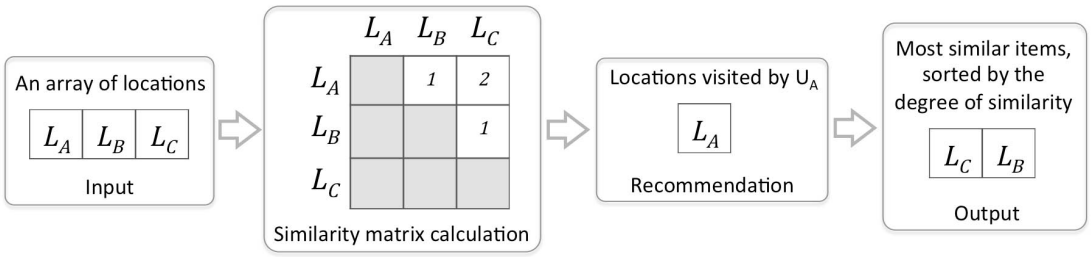


Figure 2.4: The CBF approach sequence. The computation of the similar items is illustrated for 3 items.

a similarity matrix in order to measure the degree of similarity between those items. We compute $sim(L_A, L_B) = 1$, $sim(L_A, L_C) = 2$ and $sim(L_B, L_C) = 1$. Later, during the recommendation phase, we exclude the locations visited by the user from the resulting set. The produced output is an array of locations sorted in decreasing order by their degree of similarity.

2.3 Overview of Collaborative Filtering

In this Section, we introduce CF algorithms, with an analysis of the recommendation process. The goal of a CF algorithm is to suggest new items for a particular user based on user's previous opinions and the opinions of other users with similar taste [**?**]. We first introduce a generic definition of a CF algorithm, what kind of data is used as input and output as well as a definition of the data transformation process.

As mentioned previously, the CF approach is divided in two types: Model-Based and Memory-Based. Memory-Based algorithm utilizes user rating data to compute similarity between users or items while the Model-Based uses data mining and machine learning algorithms to find patterns based on training data. As shown in Figure 2.5, the algorithm has as input an entire user-item matrix $m \times n$ of users $U = \{u_1, u_2, \ldots, u_j, \ldots u_m\}$ by items $I = \{i_1, i_2, \ldots, i_j, \ldots, i_n\}$. Each entry $a_{i,j}$ in the user-item matrix represents the preference score in a numerical scale of the ith user on the jth item. The *recommendation* part of the algorithm has as output a list of the most similar items for the active user u_a. The *prediction* part of the algorithm produces the prediction $P_{a,j}$ for the specific item i_j for the active user u_a, which expresses the predicted likeness of item $i_j \notin I_{u_a}$ for u_a, where I_{u_a} is a set of items that belong to u_a.
In order to integrate this type of algorithms into this project, each item will be a point of interest. The CF algorithm will be used to suggest locations for each user, providing an output containing the following characteristics:

- A list of locations that have not been visited by the user;
- The list depends on the user's past visits.

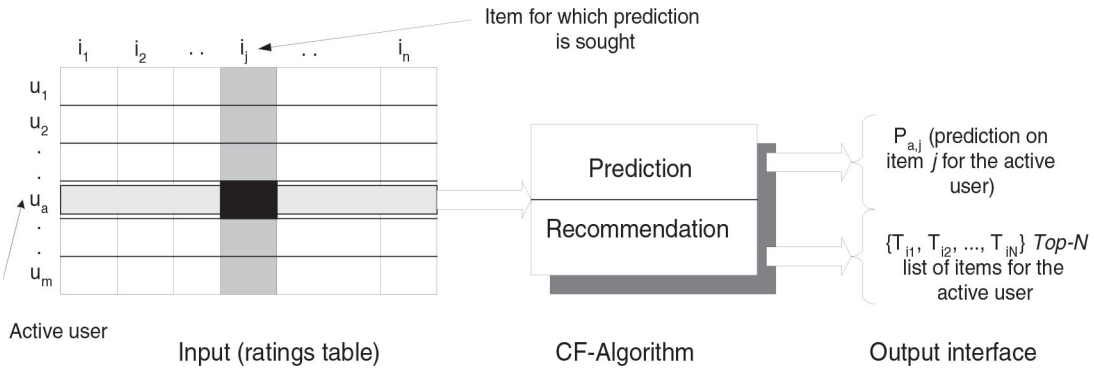


Figure 2.5: The Collaborative Filtering Process (adapted from [**?**]).

Figure 2.6 shows an example of a CF recommender system. As the recommendation process is based on the user's past actions, `User A` has visited `Location A` and `Location C` while `User B` has only visited `Location C`. The system will determine that `User A` and `User B` have similar preferences since they both have visited `Location C` and then it will recommend `Location A` to `User B`.

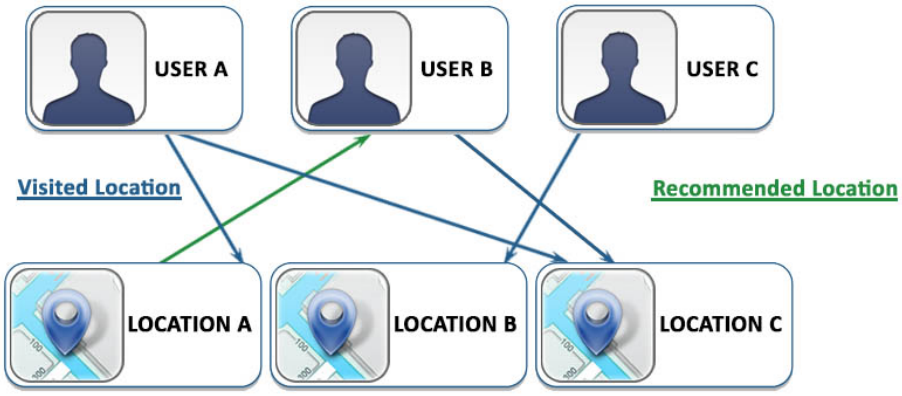


Figure 2.6: Example of a recommender system using a CF approach.

2.3.1 User-Based Approach

This Section describes the User-Based Collaborative Filtering (UBCF) methodology with Memory-based algorithms. The purpose of this category of algorithms is introduced along with their benefits and disadvantages.

The Memory-Based approach uses statistical analysis methods to find a set of users, designated by neighbors, that have similar preferences. It searches for neighbor users with a

large number of preferences in common. The most commonly used algorithms are the K-Nearest Neighbors (KNN) [**?**][**?**] and the Locality Sensitive Hashing (LSH) [**?**].

The KNN algorithm consists in finding the top K nearest neighbours for an item. Unknown ratings are predicted from a list of nearest neighbors which involve finding the user-to-user or item-to-item correlations. For large datasets, this algorithm imposes a significant computational burden.

In the presence of a wide range of users and items, the preferred algorithm would be LSH. It implements the nearest-neighbor mechanism in linear time and does not consider the content of the recommended items. The LSH algorithm is more effective on large data sets when an appropriate hash function is chosen [**?**], but its performance decreases when data gets sparse [**?**].

2.3.2 Item-Based Collaborative Filtering

This Section details the Item-Based Collaborative Filtering (IBCF) with Model-based methodology for producing predictions to users. This type of algorithms follows a different approach, as compared with user-based algorithms, from Subsection 2.3.1. Item-based approach looks into a set of items that a specific user has rated and computes a set of similar k items $\{i_1, i_2, \ldots, i_k\}$, as well as the similarities between themselves $\{s_{i1}, s_{i2}, \ldots, s_{ik}\}$.

The Model-Based method provides recommendation for items after defining a model based on user classifications. This type of algorithm uses data mining and machine learning techniques and offers some benefits regarding scalability as compared with the Memory-Based model. Usually the model building process is performed by algorithms such as Bayesian Networks [**?**] and rule-based approaches [**?**]. The Bayesian network model formulates a probabilistic model for the CF problems. The rule-based approach applies association rule discovery algorithms to find association between co-purchased[1] items and then generates item recommendations based on the strength of the association between items [**?**].

This type of algorithms handles data sparsity and scalability without negative implications on prediction. The disadvantage of this approach lies in the model building process. Large models should be considered in order to have good predictions but exhibit negative implications on both performance and scalability. By reducing the model size, loss of useful information may occur which worsens the quality of the predictions. One needs to have a tradeoff between scalability and prediction performance. Item-based CF generates predictions by using the user's own ratings for other items combined with those similarities to the target item, rather than other users' ratings and user similarities as in User-based CF. A RS needs a similarity function and a method to generate predictions from ratings and similarities [**?**].

In our system, each user will have the possibility to rate the visited locations in order to express the degree of enjoyment during the visit. Through collaboration of various users, the collected data will be used to compute the average rating for each location and later

[1] This example refers to e-commerce, but it can be applied in almost any scenario.

recommend it to users that haven't visited it yet. Figure 2.7 illustrates the first step of the recommendation process. It consists in the calculation of the item-to-item similarity matrix, which will compute the related pairs as $sim_{i,j}$.

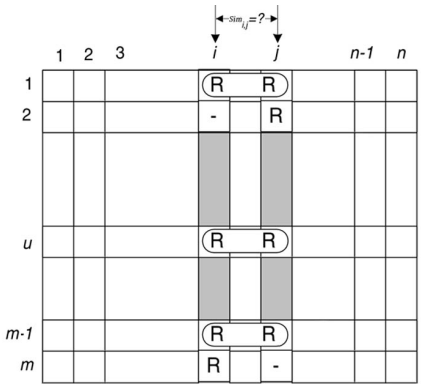


Figure 2.7: Isolation of the items with same rating value (Adapted from [**?**]).

The next step of the recommendation process, illustrated in Figure 2.8, consists in producing the list of the recommended items. The output of the algorithm is a set of items sorted by the similarity ranking to the i-th item. After isolating the most similar items based on the similarity measures, the next step is to look into the target users ratings and obtain a prediction of the most similar items [**?**].

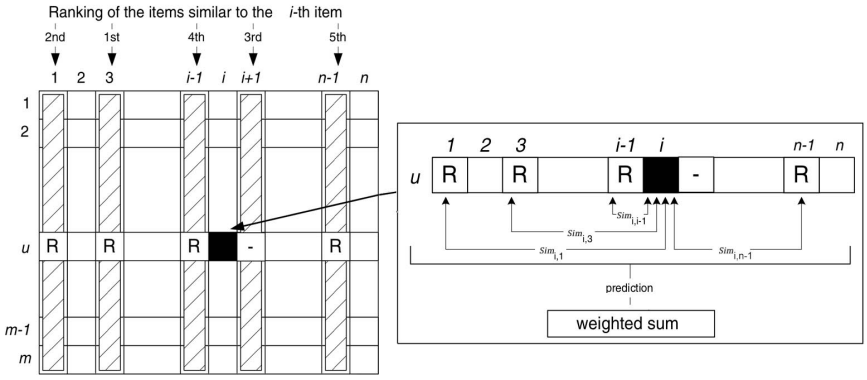


Figure 2.8: The Item-Based Collaborative Filtering Process with prediction generation process (Adapted from [**?**]).

There are many different ways to compute the similarity between items. The most commonly used methods are the Adjusted Cosine Similarity, Cosine based Similarity, and Correlation based Similarity [**?**]. After obtaining the similarities between items, the next step is to generate the output interface in terms of prediction. The prediction computation is obtained by looking into target user's ratings and use a weighted sum to obtain predictions. The aim of

this technique is to compute prediction on item i for user u by computing the sum of ratings given by user u on the items similar to i. Items i and j are weighted by corresponding similarity $sim_{i,j}$ [**?**].

An example illustrating how the recommendation process works with this type of algorithms is shown in Figure 2.9. We consider that the three users have visited `Location C` and have

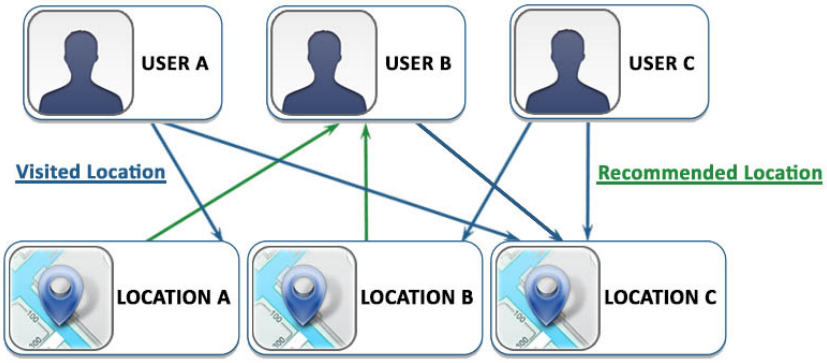


Figure 2.9: The Item-Based Collaborative Filtering example.

rated equally each one of the visited locations. The similarity is computed by first obtaining the list of users that have visited the location in cause. Then, for each one of those users, a list of locations that share the same rating as the targeted location is retrieved. The algorithm will recommend both `Location A` and `Location B` to `User B`, since those are considered similar by `User's B` actions.

An example of the IBCF algorithm is presented in Figure 2.10. In order to simplify the following examples, `User B` is denoted as U_B, `Location A` as L_A, `Location B` as L_B, and so on.

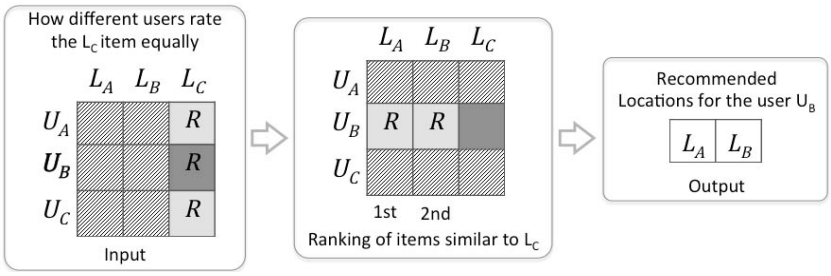


Figure 2.10: An algorithm example of the IBCF approach.

In this example, we compute locations similar to L_C for the active user U_B. Considering that U_B has rated L_C with the rating R, we search for other users that have rated the same

location equally. From these users, we retrieve a set of locations that share the same rating value R but weren't visited by U_B, in this case L_A and L_B. Then, we rank those by computing the weighted sum for the ratings. The algorithm produces a sorted array of locations, namely L_A and L_B.

2.4 Algorithm Comparison

In order to compare the approaches reviewed in this Section, the following set of characteristics was chosen:

- **Easy Implementation** - Refers to the complexity of the algorithm implementation.
- **Scalability** - Describes the algorithm ability to handle an increasing amount of data without noticeable performance implications.
- **Performance** - Refers to a reasonable algorithm performance without introducing a large computational burden in the system.
- **Data Sparsity** - Users tend to rate only a few items over the entire set, resulting in a very sparse matrix[2]. The computed matrix doesn't have useful rating information which degrades the algorithm performance, producing poor recommendations.
- **Cold Start** - Also known as the first rater problem. Caused by the data sparsity. Refers to the lack of information related to the new users which does not have past preferences. The system cannot draw any inferences for users or items about which it has not yet gathered sufficient information. New users will need to rate a sufficient number of items to ensure that the system provides reliable recommendations for them.
- **Grey-Sheep problem** - Refers to a set of users that have low correlation coefficients with other users, as they partially agree or disagree with other users. The presence of these users in a small community may result in inaccurate recommendations for those users. Larger communities of these users may negatively affect the overall recommendations of the system.

Table 2.2 lists a comparison of the advantages and disadvantages of the filtering approaches, discussed in this Section.

Table 2.2: Comparison of various filtering approaches.

Characteristic	CBF	IBCF	UBCF	Hybrid Approach
Easy Implementation	✓	✗	✓	✗
Scalability	✗	✓	✗	✓
Performance	✗	✓	✗	✓
Data Sparsity	✗	✓	✗	✓
Avoids Cold Start	✓	✗	✓	✓
Avoids Grey-Sheep problem	✓	✗	✗	✓

[2] A sparse matrix has many entries set to zero.

CBF recommender systems, computes correlations between the content of the items and the user's preferences as opposed to UBCF or IBCF systems that choose correlation between users or items with similar preferences [**?**]. The CBF algorithms are simple to implement and if the item attributes are correctly selected, they can improve their performance [**?**], but not on a large scale. Because of inaccurate predictions, this type of algorithms is not appropriate for the current problem of a tourist guide. The CF technique is a preferable choice when compared with the CBF, namely the UBCF and IBCF approaches, which have presented more benefits and are more accurate in terms of predictions, thus excluding CBF as the candidate for the implementation of the RS in this project. Although the CF technique is better than the CBF technique in some aspects, it is worse in other aspects, as pointed in Table 2.2. So, researchers have recently developed hybrid approaches that overcome deficiencies of both systems acting in isolation. Despite the fact that in the RS literature, the use of a hybrid approach is usually preferable [**?**] as compared to other approaches, we choose to use in our work the IBCF approach.

Through the research that was made, we conclude that IBCF approach presents sufficient amount of benefits for our scenario. While hybrid approaches are better, they were considered very complex to be implemented in the available time. The complexity of the IBCF RS is lower, it handles appropriately the information from our service while making accurate recommendations, thus being our first choice for the RS. The main disadvantages of the IBCF approach (described in Table 2.2) are its complexity, the cold start, and the grey-sheep problems. In the current project, the complexity problem is contoured by using the Mahout [**?**] framework, which provides a robust implementation of varied RS with support to different algorithms, including those described in this document. Neither the cold start nor gray-sheep problems were addressed in this project.

Moreover, the experimental results detailed in Subsection 5.1 helped us to choose the Slope One [**?**] algorithm, which fits best in our scenario. The Slope One algorithm has shown more benefits when compared with other solutions, namely in performance, variation of the model size, recommendation quality, among others.

The Slope One algorithm is a family of algorithms used for CF. It assumes that there's some linear regression between the rating value of one item and another. The estimated preference Y is given by the equation

$$Y = mX + b, \tag{2.1}$$

with $m = 1$, and X representing the item on which the preference is based. The average difference in the preference value for every pair of items is represented by b.

The algorithm handles computations quickly, providing a good performance. As all the information is stored in memory, the large amount of item-item preference value differences can cause memory issues [**?**]. Suppose that we have n items, m users. Computing the average rating differences for each pair of items requires a matrix with up to $n(n-1)/2$ entries, and up to mn^2 time steps [**?**].

2.5 Slope One - Hand Calculations

In this Subsection, we describe the pseudo-code of the Slope One algorithm, as also we manually compute a set of recommendations for a chosen example, to illustrate each step of the algorithm chosen for our recommender system [**?**].

The algorithm is divided in two stages: pre-processing and recommendation. Algorithm 1 describes the pseudo-code of the pre-processing stage of the Slope One deviation matrix [**?**].

Algorithm 1 Computation of Slope One deviation matrix.

Input: U: set of all users.
S: set of all items.
R: set of all rated items.
Output: dev: deviation matrix.
$freq$: co-rating frequency matrix.

```
   /* Step 1 */
1: for all u ∈ U do
2:    for all i ∈ Ru do
3:       for all j ∈ Ru do
4:          dev[i,j] ← dev[i,j] + (r[u,i] − r[u,j])
5:          freq[i,j] ← freq[i,j] + 1
6:       end for
7:    end for
8: end for
   /* Step 2 */
9: for all i ∈ S do
10:    for all j ∈ S do
11:       dev[i,j] ← dev[i,j] / freq[i,j]
12:    end for
13: end for
```

On Table 2.3 we present the input data that is used to perform the hand calculations. The input dataset contains 4 users and 4 touristic locations, where some of these locations were not rated by all users.

Table 2.3: The input dataset used by the hand calculations.

	Big Ben	Stonehenge	Taj Mahal	London eye
Steve	4	5	-	3
Jonathan	5	4	-	5
Tim	3	5	2	3
Eddy	5	-	4	-

On Table 2.4 we compute the entries for the deviation matrix through *Step 1* of the algorithm (Algorithm 1). For each rated pair of items, we add the difference between the items i and j to the already existing value for index $[i, j]$ in dev. In this phase, we also increment the value of $freq$ for each difference between i and j.

Table 2.4: Calculation of the entries for the deviation matrix.

	Big Ben	Stonehenge	Taj Mahal	London eye
Big Ben				
Stonehenge	4 – 5 = -1 5 – 4 = 1 3 – 5 = -2			
Taj Mahal	3 – 2 = 1 5 – 4 = 1	5 – 2 = 3		
London eye	4 – 3 = 1 5 – 5 = 0 3 – 3 = 0	5 – 3 = 2 4 – 5 = -1 5 –3 = 2	2 -3 = -1	

The result of the *Step 2* (Algorithm 1) is described on Table 2.5, which iterates through the dev matrix. For every index $[i, j]$ we calculate the average of the value stored in that index by dividing the sum of each component by the $freq$ value of the corresponding index.

Table 2.5: Deviation matrix with average difference between each combination of items.

	Big Ben	Stonehenge	Taj Mahal	London eye
Big Ben				
Stonehenge	$\frac{(-1)+1+(-2)}{3} = -\frac{2}{3}$			
Taj Mahal	$\frac{1+1}{2} = 1$	$\frac{3}{1} = 3$		
London eye	$\frac{1+0+0}{3} = \frac{1}{3}$	$\frac{2+(-1)+2}{3} = 1$	$\frac{-1}{1} = -1$	

The final dev matrix contains the average difference between each combination of items. For example, the difference value for the `Stonehenge` and `Taj Mahal` is 3.0, which means that, on average, the item `Stonehenge` is rated above the item `Taj Mahal` by 3.0.

Now we will apply the calculations required by the recommendation stage, which will have as output a list of recommendations computed for user `Eddy`. The following format is used to specify the involved components:
$R_{\{User\ that\ receives\ recommendation, Recommended\ location, Base\ location\ used\ for\ the\ rating\ computation\}}$.

$R_{\{Eddy,Stonehenge,BigBen\}} = 5 + (-\frac{1}{2}) = 4.5$
$R_{\{Eddy,Stonehenge,TajMahal\}} = 4 + 3 = 7$
$R_{\{Eddy,Stonehenge\}} = \frac{3*4.5+1*7}{3+1} = 5.125$

$R_{\{Eddy,LondonEye,BigBen\}} = 5 + \frac{1}{2} = 5.5$
$R_{\{Eddy,LondonEye,TajMahal\}} = 4 + (-1) = 3$
$R_{\{Eddy,LondonEye\}} = \frac{3*5.5+1*3}{3+1} = 4.875$

For user `Eddy`, the recommendation list will have two entries sorted in decreasing order by the estimated value: [5.125 - `Stonehenge`, 4.875 - `London Eye`].

2.6 RS Application in Tourist Guidance

As of time of this writing (November 2013), RS are being integrated in almost every scenario where these can help users to discover new useful information. In tourist guide scenario, with integration of the RS, users can become their own travel agents. The recommended touristic locations match user's preference to provide reliable recommendations which can be accessed at any time and anywhere.

Applications such as TouristEye and Foursquare already implement the RS with CF approach [**?**] [**?**]. These services provide users with the touristic locations that they might like to visit, based on their preferences. The developers of these solutions did not provided any detailed information regarding the RS that powers their service.

The chosen CF approach for the GuideMe project has demonstrated to provide accurate results. Combined with the Slope One algorithm, it provides fast and reliable recommendations of the touristic locations, and handles well the sparsity of users' ratings, improving the quality of the recommendation system [**?**].

There are some studies focused on the efficiency, complexity and reliability of the artificial intelligence-based architectures for the RS component. This architecture is based on several machine learning techniques, such as linear models, neural networks, classification and text mining [**?**]. The combination of these techniques contributes positively to the overall quality of the RS.

3

Proposed Solution

In this Chapter, we describe the implementation aspects of the project. A general overview and service architecture is introduced in Section 3.1. Section 3.2 describes technical aspects of the Apple Push Notification Service (APNS), detailing some integration points for the proposed architecture. In Section 3.3 we describe configurations used to improve the quality of the project's development process through the revision control system. Section 3.4 describes the developed prototypes for the mobile and Web applications.

3.1 Service Architecture

For the development of the project, we have used the following technologies:

- MySQL [**?**] - Service and Log Databases.
- Java Hibernate Framework [**?**] - Service and Log DALs.
- Java Play Framework [**?**] - REST API and Web application.
- Objective-C [**?**] - iOS Universal Application.

The presented technologies and frameworks were not studied during the curricular units of the Master's course. Regarding the MySQL technology, although it has not been studied, for used purposes it is very similar to Microsoft SQL Server [**?**], for the usage in this project.

The main reason for choosing Java as the principal language instead of other technologies is the licensing costs associated to the development. The hibernate framework is licensed under the LGPL 2.1 [**?**] and the Play Framework under the Apache License 2.0 [**?**], with both allowing commercial and non commercial use of the components. The choice in developing the iOS application is due to technology popularity, its beautiful design, and the dynamics that can be achieved within this technology.

The GuideMe service architecture is depicted in Figure 3.1.

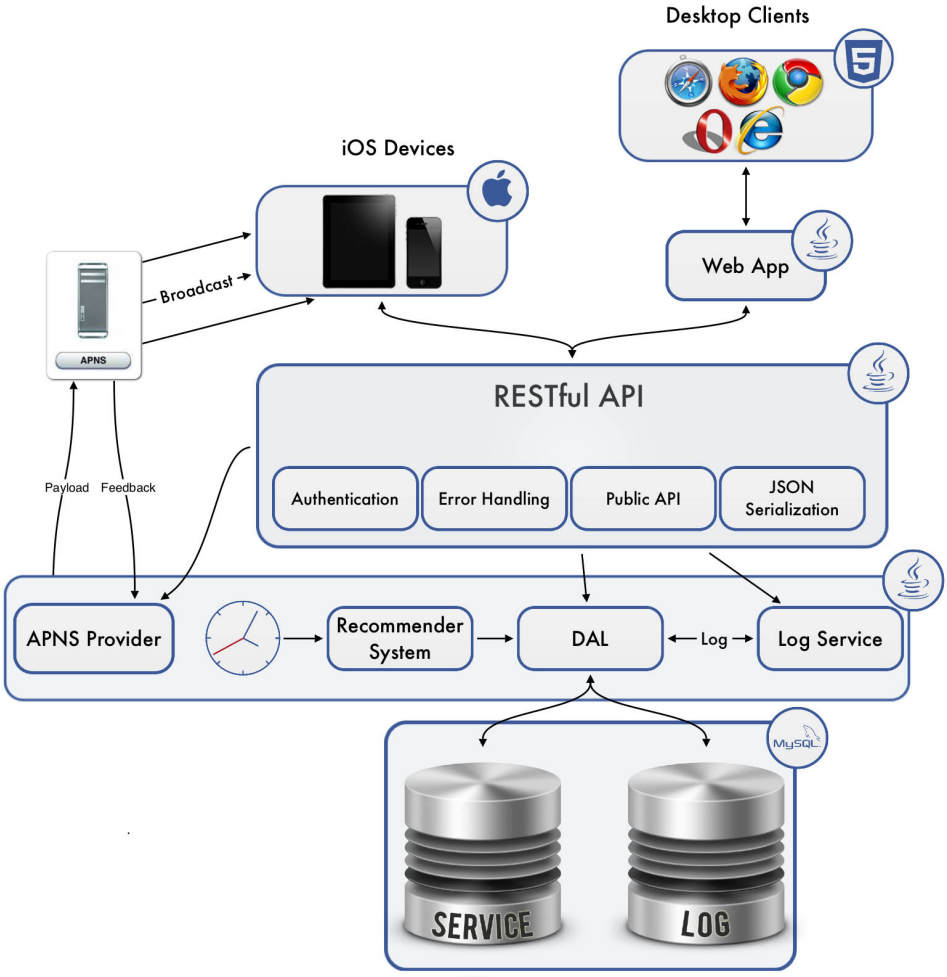


Figure 3.1: Detailed GuideMe service architecture.

The purpose of each service component is as follows:

- Service Database - the main database, responsible for storing data regarding users, locations, among others.
- Log Database - maintains log information about errors and warnings which may occur during the data retrieval or manipulation.

- Data Access Layer (DAL) - is responsible for communicating with the Service and Log database since each access to these storages is made through this component.
- Log Service - performs event logging by communicating with the DAL component.

- Recommender System - searches for new locations which can be recommended to users. This lookup process is triggered periodically by the configured Cron job.
- RESTful API - provides a set of endpoints that can be accessed by different client applications, namely by the Web Application and mobile iOS devices.
- APNS Provider - is responsible for sending push notifications to iOS clients in different occasions, for instance when new recommended locations are available.

To ensure the flexibility among the designed components, those were developed as a three tier architecture with the logical separation between the presentation, business logic, and database tiers. The adopted approach is shown on Figure 3.2.

Figure 3.2: Separation between service components as a three tier architecture.

The Data Tier (DT) is responsible for storing and retrieving data from the database services. Exposes a set of interfaces that are used by the Business Logic Tier (BLT). The BLT is responsible for controlling the service core functionalities by processing the client requests. The top most tier is the Presentation Tier (PT) which represents the aggregation of various client applications that interact with the BLT interface.

The main benefits and drawbacks of the implemented architecture are as follows [**?**]:

- Benefits
  - Scalability - The application servers for the BLT can be deployed on many machines where the database connection is only required for each application server.
  - Re-Use - The implementation aspects of the BLT are transparent to the PT, which have made this layer more reusable.

  - Security - When the BLT is located on separate application server from the DT, usually it is more secure and clients can't access the database directly.
- Drawbacks
  - Complexity - it's more difficult to build a three tier architecture than other more simplistic solution.

3.2 Integration with the APNS

APNS is responsible for propagating information to iOS devices such as iPhone, iPod touch, and iPad. When a device connects to the Internet, it establishes an authorized and encrypted Internet Protocol (IP) connection with the APNS and receives notifications over this persistent connection. If the application is not running, the user receives a notification on his devices [**?**].

Figure 3.3 illustrates the simplified push notification path from an APNS Provider to a client application, on the iOS device. The provider and the client application are the two parts which were developed in this project.

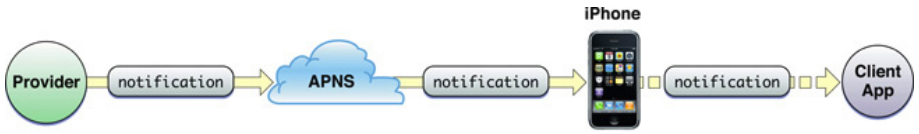


Figure 3.3: A push notification from a provider to a client application (adapted from [**?**]).

In the following paragraphs, we provide a description of the main features and some APNS service implementation details.

Feedback Service
When a user removes the application from the device, the Push Notification Service becomes unavailable for that user and the application in cause. The APNS provides the feedback mechanism that allows detecting which devices are not receiving notifications. By using this information it is possible to filter which users the push notification should not be sent, thus minimizing the computation time and bandwidth usage.

Quality of Service
The APNS guarantees that notifications are delivered to the user's device. When the device is offline or the user is unable to receive the notification, the APNS service stores the notification. Later, when the device reconnects, the APNS forwards the stored notification to the device. However, there are some limitations, only one notification per user is retained by the service and it remains for a limited time period before being removed.

Security
In order to establish communication between the provider and a device, the APNS requires connection and token to be trustworthy. A connection between the APNS and a provider is trusted if it is recognized by Apple, which has agreed to deliver notifications between the parties. After ensuring the connection trustworthiness, the APNS must guarantee that notifications are sent only to the intended device, and this is controlled by the device token which is requested during the client application startup.

The Notification Payload
Each push notification carries a payload (maximum of 256 bytes) which represents the data to be downloaded to the client application. Any notification that exceeds this limit will be refused by the APNS. Figure 3.4 represents the notification format.

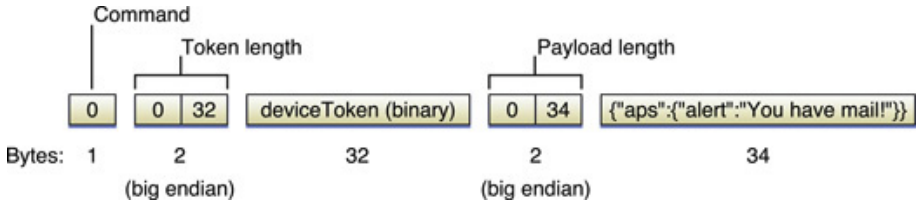


Figure 3.4: Notification Format (adapted from [**?**]).

For each notification, a provider should compose in JavaScript Object Notation (JSON) [**?**] format a dictionary named *aps*. This dictionary should contain properties that specify the following actions [**?**]:

- an alert message to be displayed;
- a badge number for the application icon;
- a sound to be played when a notification is received.

3.3 Environment Configuration

In this Section we describe the organization of the project and tools that have been used to improve the development process. Projects are hosted under the Git revision control [**?**] system using Bitbucket [**?**] web-based hosting service. Table 3.1 lists private repositories of the developed service components, and the base URL for all entries. The names under the *Repository URL* column are prefixed with `https://aumanets@bitbucket.org/aumanets`.

The purpose of these repositories is as follows:

- *GuideMeREST* repository stores the code of the REST service and the DAL for the Service and Log databases. There is also a Section for the documentation with a description of the database model and detailed information regarding the organization of the tables. Some SQL scripts to create and populate the database model are also under this version

Table 3.1: Git repositories maintained in the Bitbucket service.

Technology	Repository Name	Repository URL
	GuideMeREST	guidemerest.git
	GuideMeMobileAPI	guidememobileapi.git
	GuideMeMobileClient	guidememobileclient.git
	GuideMeWeb	guidemeweb.git
	GuideMeRecommenderEngine	recommenderengine.git
	GuideMeAPNS	guidemeapns.git
	GuideMeAPNSFeedback	recommenderengine.git

control repository. For each component of this project a set of unitary tests was developed, allowing the verification of the correctness of the designed software modules.

- *GuideMeMobileAPI* holds the source code of the developed framework which is used to access endpoints of the GuideMe REST API within iOS applications. It implements the wrappers for the JSON responses and the classes that allow to perform asynchronous requests in an organized way. There is a set of unitary tests for each remote operation responsible for validating the implementation of the REST API endpoints.
- *GuideMeMobileClient* repository stores the source code of the iPhone and iPad application and also contains their initial prototypes. This project has a dependency with the *GuideMeMobileAPI* repository.
- *GuideMeWeb* is a repository for the Web-based application. It contains prototypes for different pages and the source code for both the user interface design (Twitter Bootstrap framework [**?**]) and the server-side (Play framework [**?**]).
- *GuideMeRecommenderEngine* stores the source code of the implemented solution for the recommender system. The project contains the executable which is responsible for performing the recommendation computations and also a set of classes used to perform the evaluation of different RS implementations. It uses the GuideMeDAL in order to obtain users and locations for performing recommendations, as well as to store computed recommendations.
- *GuideMeAPNS* holds the source code of the Java client for the Apple Push Notification service. The library aims to provide a highly scalable interface to the Apple server, while still being simple and modular; it was originally obtained from [6].
- *GuideMeAPNSFeedback* repository with the Java project aimed to query the APNS Feedback service to consult iOS devices that are no longer able to receive Push Notifications. It uses the GuideMeDAL to update the database indicating the devices that should not receive push notifications.

3.3.1 An Analysis of the REST API Frameworks

Some REST frameworks for the implementation of the services with Java environment, were analyzed, more specifically the Play [**?**], Restlet [**?**] and (JAX-RS) with Jersey [**?**][**?**]. Table 3.2

shows a summarized comparison of these frameworks stating their key benefits and drawbacks.

Table 3.2: Comparison of three popular Java REST frameworks.

Framework	Scalable	Fast	Protocol Abstraction	Observations
(JAX-RS) Jersey	✓	✓	✗	• Base API in Java for the REST Web Service; • Requires everything to be implemented manually.
Restlet	✓	✓	✓	• Scarce and inconsistent documentation; • It is challenging to debug through the Restlet source.
Play	✓	✓	✓	• Excellent documentation; • No need to compile, deploy or restart the server.

Among these frameworks, the preferred choice is the Play framework which fulfills all requirements and has the highest number of benefits for the development process, as compared to the other two approaches.

3.3.2 Web Application Development Technologies

For the development of the web application, some programming languages and frameworks were studied. Initially, PHP [**?**] was the preferred language, but it has presented some problems and disadvantages, as compared to other approaches. As a script language, PHP is easy to learn but it lacks standards and conventions and it has earned a kind of bad reputation over the past years [**?**]. There is a lot of bad code and malformed examples that may lead to problematic solutions. The code is difficult to read and to maintain.

Web development with the Play Framework presents more benefits than PHP. It was designed to emphasize productivity. This framework comes with powerful Scala-based template engine [**?**]. The key advantages of the Play Framework, that are not easily achieved in PHP are as follows:

- Elegant URL design - flexible Uniform Resource Locator (URL) configuration without framework-specific limitations. For example `http://someurl/item?id=1` or `http://someurl/item/1`.
- Template system - extensible and designer-friendly template language that separates the design from content and Java code.
- Internationalization - full support for multi-language applications, which allows to specify translation strings and provides hooks for language-specific functionality.

3.3.3 Data Access Layer Frameworks

The DAL is implemented using the Hibernate [**?**] framework for Java. Hibernate is an Object-Relational Mapping (ORM) library that provides a framework for mapping an object-oriented domain model to a relational database, which makes the data access more abstract and portable, as compared to the traditional Java Database Connectivity (JDBC) approach. There are many other ORM frameworks (ActiveJDBC [**?**], jOOQ [**?**], among others), but the Hibernate has a large community and currently is the most widely used by developers. Greater tool support is an important aspect, because eases the task of finding solutions to software problems. These and the following advantages justify our decision in adopting Hibernate as the ORM framework.

The main advantages of the Hibernate framework are as follows [**?**]:

- Performance - generates efficient queries and employs first and second level caching strategy. The first level caching refers to the actions associated with the context of the current session, while the second level caching refers to the entire application. Both strategies are designed to minimize the number of SQL queries that are executed over the database.
- Effective Cross-Database Portability - supports portability across different relational databases. The framework allows to easily configure its dialect, being independent of the database concrete functionalities, it specifies which SQL language should be used with the database. With this feature, it is easy to migrate to another relational database engine, if needed.
- Developer's Productivity - it has a short learning curve. It creates a clear abstraction level between the Java objects and the database model.

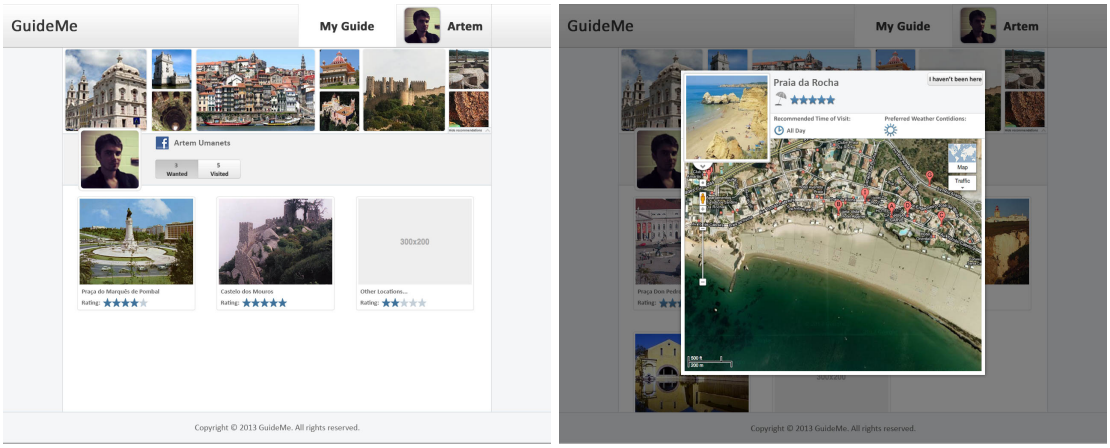
3.4 Prototyping

In this Section we describe the designed prototypes that have facilitated the developments of the Web and iOS applications. The complete set of prototypes can be found in the corresponding repository of the application in cause.

3.4.1 Web Application

Using the Adobe Photoshop [**?**] software, a set of prototypes was created for the Web application, representing its most important features. The prototypes can be found int the `GuideMeWeb` repository and they already implement the desired look and feel for this application. Each prototype provides the specification of the page URL and some notes regarding the specific actions. An example of the designed prototypes is shown on Figure 3.5. Figure 3.5 (a) shows the page with the touristic locations that the user wants to visit, with the recommended locations displayed under the navigation menu. The prototype on Figure 3.5 (b) illustrates the page with the location details, where the most important information regarding the chosen touristic locations is shown, such as: map with the location coordinates, average rating, preferred time of the visit, weather conditions, among others.

The hierarchy of the main operations of the Web application is shown on Figure 3.6. The



(a) Wanted locations. (b) Location details.

Figure 3.5: GuideMe Web application prototypes.

Admin users have access to the whole application while *Standard* users have the same permissions, except to consult the API Documentation.

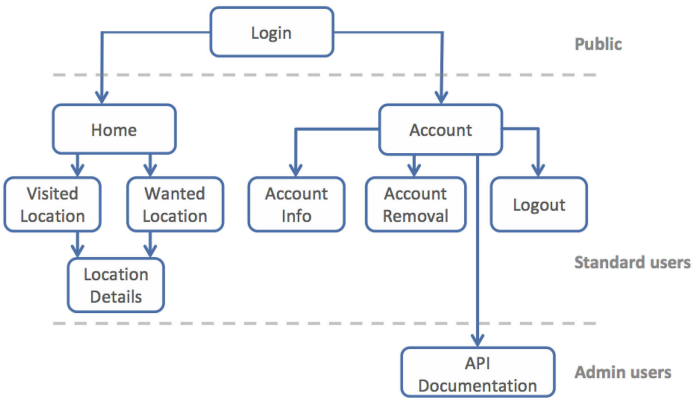


Figure 3.6: Hierarchy of the supported operations within the Web application.

3.4.2 iPhone Application

A set of prototypes for the iPhone application was designed using the Adobe Photoshop and the AppCooker [**?**] application for the iPad. The complete designs can be found in the `GuideMeMobile` repository. Some screenshots of the designed prototypes are shown on Figure 3.7.

Figure 3.7 (a) shows the authentication screen, where users are able to perform login by

(a) Login screen. (b) Menu options. (c) Nearby locations.

Figure 3.7: Screenshots of the GuideMe application for the iPhone.

using the preferred social service (e.g. Facebook, Twitter, or Google+). Figure 3.7 (b) illustrates the main screen with the expanded navigation menu, which allows the navigation through the different parts of the application. This kind of navigation could be implemented using the Toolbar (usually located at the bottom of the screen) component but the presented choice offers more benefits by providing more space to be filled with the information about the touristic locations. The recommended locations are presented above the users photography. Figure 3.7 (c) shows an example of the **Near Me** feature, which consists in showing the touristic locations near the user's current location. On the top part of the screen, it is possible to configure filtering options, by specifying the weather conditions and the preferred time of visit. Through the map, users can visualize touristic locations, places where they have been, and other locations visited by their friends.

Figure 3.8 illustrates the hierarchy of the operations that can be performed by standard users, where *Explore*, *My Sights*, and *Friends* represent the main sections of the application. Users with the *Admin* privileges have access to an additional section that is not visible to *Standard* users.

Each section is detailed as follows:

- Explore - presents a list of sights, filtered by the following criteria:
  - select the attraction, country, city or user's current location.
  - choose preferred weather conditions and time of day.
  - specify the user's available time.
  - search with custom description.
- My Sights - used to consult the visited, wanted, or recommended touristic locations.
- Friends - offers the possibility to consult the followed users and to search for new users.

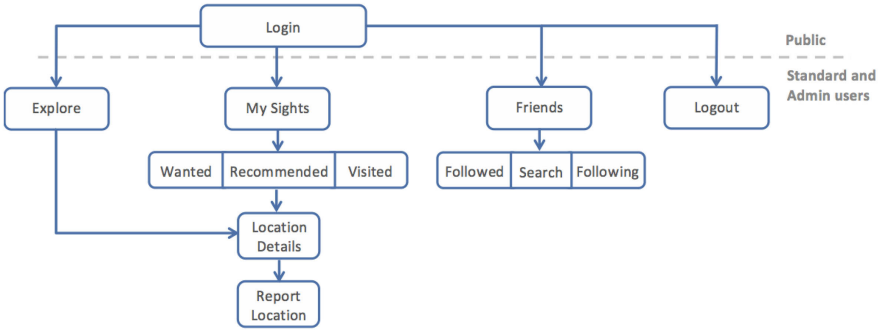


Figure 3.8: Operation hierarchy for the *Standard* user of the iOS application.

The *Admin* users can perform the same operations as the *Standard* users, as illustrated on Figure 3.8 and also some additional operations that are depicted on Figure 3.9. The main sections of the administrative area are:

- Manage Sights - searching for touristic locations, edit their information or create a new touristic location.
- Reported Sights - lists the locations that were reported by users, allowing to correct their information directly with the reported data.
- Service Logs - allows to consult the log information from different service components.

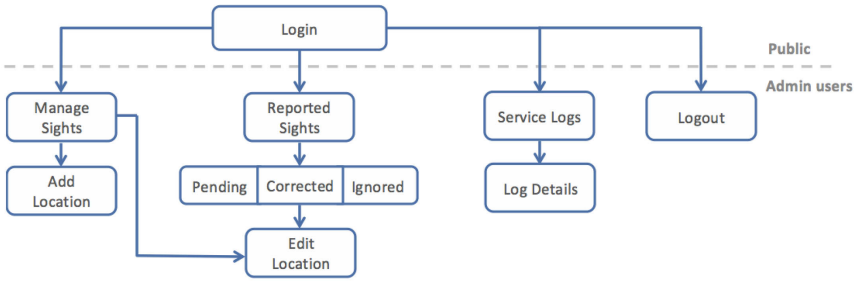


Figure 3.9: Operation hierarchy for the *Admin* user of the iOS application.

4

Implementation Details

In this Chapter we describe the implementation details of the project and employed technologies. In Section 4.1 we start by detailing the Service and Log databases. Section 4.2 approaches the Data Access Layer, and how it interacts with the database. In Section 4.3 we introduce the implementation aspects of the REST API and its communication with the DAL. Section 4.4 describes the technical aspects of the recommender system. Then in Section 4.5 we detail the development of the mobile application, followed by Section 4.6 with information regarding the implemented Apple Push Notification Service Provider. Section 4.7 is dedicated to the developments of the Web application. We finish the current Chapter with Section 4.8, describing additional developments that were made.

4.1 Database Model

In this Section, we introduce some implementation details of the databases, which were developed using the MySQL technology.

4.1.1 Service Database

The service database is designed to store and access, in a structured way, information regarding the touristic locations, users data, visited locations, among other fields. The Entity Relationship (ER) model for the Service Database is presented in Appendix A.1.

One of the objectives of the project is to support multilanguage. In order to provide this kind of support, some preparations were made at the database level. The `Locale` table was introduced, which is responsible for controlling the language of the different items. To demonstrate the localization functionality, the project was developed supporting two different languages: Portuguese [PT] and English [EN]. We use the language identifier as defined by the ISO 639-1 standard [**?**].

In order to provide the location with the descriptive information regarding weather conditions and time of visit, a standard approach would be to create a separate table for the attribute in cause and to have a many-to-many relationship between this table and the `Location` table (each attribute would generate two additional tables). By using the proposed solution, the complexity of the database model is minimized because the information

is organized in a way such that the number of existing tables is reduced, which minimizes the need for the additional *join* operations during the query execution. A mask of bits is used to characterize some attributes of the point of interest. Table 4.1 and Table 4.2 describe the proposed solutions to characterize the following components of the location:

- `visitWeatherMask` - represents the possible values for the weather condition.
- `visitTimeMask` - describes the time of day for the visit.

Table 4.1: Bitmask for the **visitWeatherMask** column.

Sun	Rain and Clouds	Snow
0x1	0x2	0x4

Table 4.2: Bitmask for the **visitTimeMask** column.

Day	Night
0x1	0x2

A similar approach is used to define the `profileMask` attribute for the `User` table which represents the user's role in the system. Table 4.3 shows the possible mask values for this attribute.

Table 4.3: Bitmask for the **profileMask** column.

Admin	Standard
0x1	0x2

Each role purpose is described below:

- `Admin` - User with the highest privilege access, being allowed to consult and edit any information regarding the point of interest.
- `Standard` - Has access to consult information regarding the touristic location.

These bit mask attributes are stored as integer values, where each power of two value represents different information that can be merged with other values through bitwise operations. An efficient approach is possible due to the well-defined domain values. Even if there is a need to insert new values, those can be easily added to the existing set. Assuming that the augmented set is composed by n elements, the newly added value will occupy the index 2^n, with $n \, \epsilon \, \{0;\ 31\}$ allowing a total of 32 different values.

Two different scripts were created for the database population. Each one performs insertions in the following tables, based on its `Locale` value:

- `Locale` - Representation of the English and Portuguese locales.
- `Country` - A set of 249 countries with English description and other 249 with Portuguese description.
- `Status` - Allowed values for the location attractions filled in English and Portuguese.
- `Attraction` - Allowed values to describe attractions of the location in English and Portuguese.
- `SocialService` - Contains information regarding the social services that are supported by the system, currently used for authentication purposes.

The GuideMe service supports the social interaction by allowing users to follow and to be followed. The Follower table was created in order to represent this relationship; its definition is presented in Appendix A.1.

4.1.2 Log Database

The log database is used to track the different kind of events that may occur in the system, such as errors, warnings, and other informative messages. The ER model for the Log database is presented in Appendix A.2.
The choice of performing the log into the database instead of a different method consists in providing a more interactive way to access the log data. Using this approach, it becomes easier to implement client applications with friendly user interfaces that can access, filter, and manipulate log information.

The logged information can be crucial to detect bugs. We have implemented the alternative logger solution using the Log4j [**?**] library, which ensures persistence of this information when the logging to the database fails due to some disruption causes. Each system component can optionally specify the configuration file, which is used to configure the logger properties as well as the destination of the file that should be used for logging. By default, the configuration file is located under the root directory of the corresponding project and is named as *LocalLocagger.properties*.

4.2 Data Access Layer

In this Section, we describe the key aspects of the DAL implementation, which was written in Java using the Hibernate framework.

4.2.1 Implementation Details

Some implementation details of the DAL for both the Service and Log databases are described in this Subsection. In order to not compromise the DAL usage with the Hibernate framework, a Data Access Object (DAO) [**?**] pattern is used to encapsulate the access to the

data with an interface. The layers above DAL are independent from the Hibernate framework, allowing this to be easily replaced, if needed. Different projects were currently configured in the Eclipse [**?**] IDE; the aim of each one of them is as follows:

- `DALCommon` - contains a set of classes that are common between the `GuideMeLogDAL` and `GuideMeServiceDAL`. It offers a generic implementation for the Entity Objects[1], DAO interfaces, as well as the exception hierarchy.
- `GuideMeLogDAL` - represents the implementation of the DAL that provides methods to manipulate the Log database. It allows to log events that occur in different applications. Contains the test cases that perform unitary testing against the DAO implementation.
- `GuideMeServiceDAL` - contains the implementation of the DAL for the service database. Provides different methods to query the data that represent the core information of the service. Implements unitary tests for every written operation.

`DALCommon` defines a generic set of classes and interfaces that generalise the implementation and access to specific database operations. Figure 4.1 shows the Unified Modeling Language (UML) [**?**] diagram with main components of the DAL.

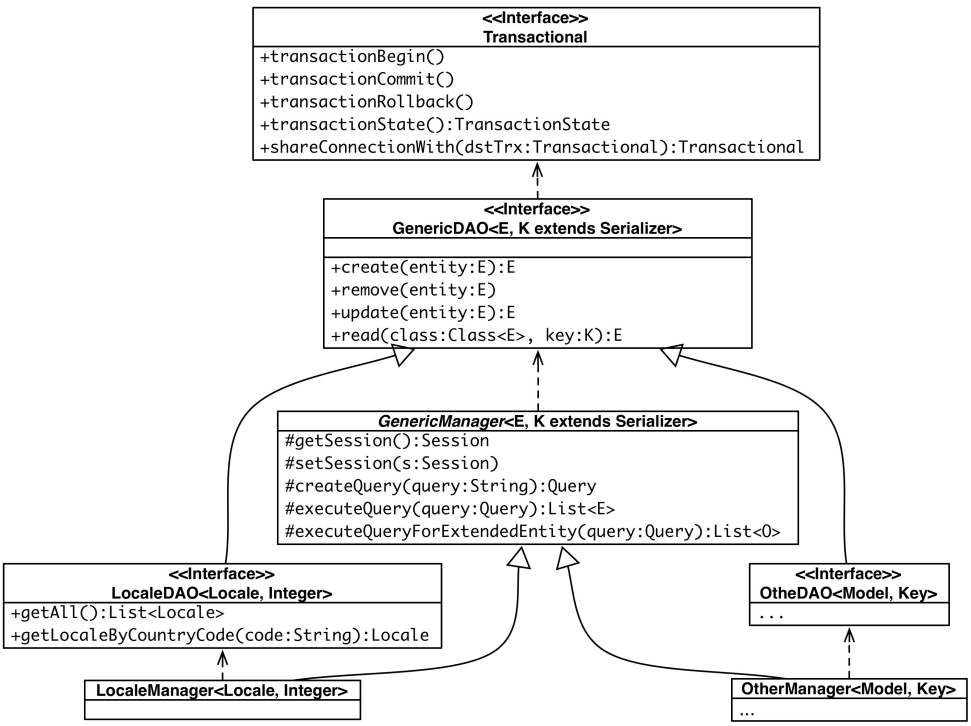


Figure 4.1: UML diagram of the DAL core classes.

On top, we have the `Transactional` interface which describes the contract for the ma-

[1] Entity Objects - are the components that encapsulate the data model. Any instance is mapped into a single object in the data source.

nipulation of the transaction. Then the `GenericDAO` describing the CRUD (Create, Read, Update and Delete) operations. The `GenericManager` class implements both interfaces, `Transactional` and `GenericDAO`, and it also offers additional helper methods that simplify the implementation of the operations for accessing data.

The `GenericDAO` interface and the `GenericManager` class are implemented by each hibernate entity that requires to be accessed within a query. In the concrete `DAO` interface, we describe specific methods that query the database to obtain data and the concrete `Manager` class provides an implementation for those methods as well as for the CRUD operations inherited from the `GenericManager`. The specific operations use the Hibernate Query Language (HQL) which is similar to SQL. The Hibernate framework is then responsible for converting the HQL into the corresponding SQL code.

The service database and log database DAL uses the structure illustrated in Figure 4.1 and both follow the same implementation flow, previously described.

4.2.2 Hibernate Framework Configuration

In this Subsection we describe the configuration aspects of the Hibernate framework along with the additional functionalities that were implemented. We also detail some obstacles that have emerged during the development of the DAL.

The Hibernate framework is configured with the *hibernate.cfg.xml* file which contains the information such as the database name, location, user name, password, among others fields. The configuration file also includes information regarding the entity mappings which are defined as a list of references to Hibernate Mapping (HBM) files. The Hibernate tool was installed for the Eclipse IDE, allowing to reverse engineer entities from the database. The auto-generated entity represents the Java object with the associated mapping HBM file which describes the attributes and relationship to other entities.

Another required configuration of the Hibernate framework is the dialect, which is responsible for generating appropriate SQL for the chosen database. A suitable dialect for our MySQL database is the `MySQL5InnoDBDialect`. We had a need to extend it in order to add the mapping for the bitwise operations, since those are not included in the original dialect and cannot be used in HQL. When the Hibernate framework is initialized, our custom dialect performs registry of the two custom functions (`bitwise_and` and `bitwise_or`) that can be used in a HQL query in order to be correctly converted into corresponding SQL query. The Code Listing 4.1 shows the usage example for the HQL with custom defined binary operation which is then mapped to SQL.

Code Listing 4.1: SQL mapped from HQL.

```
/* HQL */
SELECT e FROM Entity e WHERE bitwise_and(e.binary1, :someVal) = :someVal
/* SQL */
SELECT e.* FROM entity WHERE e.binary1 & ? = ?
```

4.2.3 Difficulties Found in the Implementation

Some difficulties were found during the implementation of the DAL, more specifically on the data mapping of the `Localized_Location` table. The first approach consisted in making the field `locale_id` as part of the composite Primary Key (PK) and also as the Foreign Key (FK) for the `city` and `attraction` tables. The ER model was valid and successfully generated, but during the creation of the new `Localized_Location` record through the Hibernate framework, the FK relationship for the `city` and `attraction` were not included in the generated SQL INSERT statement. This behaviour caused the insertion statement to fail, since `attraction_id` and `city_id` could not be `null`.

A suitable solution was found for this problem, which consisted in replicating the `locale_id` attribute. The following set of fields were created {`location_locale_id`, `city_locale_id`, `attraction_locale_id`} to replace the original `locale_id`. The implementation of the DAL guaranties that has the three fields have always the same value for the specific record.

Another problem has emerged when it became necessary to specify different databases in the configuration file. When the hibernate models are generated using the Hibernate Reverse Engineering tool, one of the attributes in the created files is the *catalog* which points to the database used during the code generation process. This binds the model to that database, which is not what we need. However, when the *catalog* attribute is not indicated, the hibernate engine automatically uses the database defined in the configuration file.

To circumvent this binding problem, the solution consisted in the creating the Ant [**?**] task that removes the *catalog* attribute, for each hibernate model file. After generating the hibernate model, it is required to execute the defined Ant task in order to re-process the HBM files.

4.3 REST API

This Section describes different aspects of the implemented REST API, and the technologies and methodologies used to design the API.

The REST API was implemented using the Play Framework. It integrates the components and API required for web application development. The framework is based on a lightweight, stateless, web-friendly architecture that optimizes the resource consumption for scalable applications [**?**].

Our API provides a set of endpoints whose purpose is to obtain data through DAL and serialize the response into JSON format. Every response is composed with header and body. The header contains the information that allows to identify if the request is successfully executed or not. An example of this scenario is illustrated on Code Listing 4.2. When the header contains the information regarding an error, the body is populated with the null value. The detailed version of the API documentation can be accessed through `Settings/API Documentation` on the Web application.

Code Listing 4.2: Example of the success and error headers.

```
/* Success */
{
    "header": {
        "operationSucceeded": true,
        "sessionState": "SessionStateValid" or "SessionStateNotRequired"
    },
    "body": ...
}

/* On error */
{
    "header": {
        "errorMessage": "Length of the locale identifier argument should be
            exactly 2",
        "operationSucceeded": false,
        "validArguments": false,
        "sessionState": "SessionStateInvalid",
        "friendlyErrorMessage": "Error communicating with the server. Please
            try again later."
    },
    "body": null
}
```

4.3.1 Internationalization Support

Internationalization is the technique for organizing localized resources so that an application can select the user-preferred set of resources at runtime. Localization is the translation of text displayed by an application. The importance of supporting multiple languages consists in reaching the maximum number of users, specifically in tourist guide applications where users of the application are from different countries. Currently, the API supports English and Portuguese languages, and it is possible to add other languages.

Every endpoint receives the `localeName` argument which should have the ISO 639-1 format and sends the response in the corresponding language. The English language is assumed by default when the `localeName` argument is not specified or refers to the unsupported language.

When an error occurs during the request, some properties of the header information are populated with the localized information regarding the error in cause, namely the *errorMessage* and *friendlyErrorMessage* properties. These messages are defined as a property list in the `conf/messages.pt` and `conf/messages.en` files. The Play framework offers an easy way to access the localized i18n [**?**] properties, as it is shown in Code Listing 4.3.

Code Listing 4.3: An example of the i18n definition and usage.

```
Message.UserName=Nome Completo      # messages.pt
Message.UserName=Full Name          # messages.en
```

```
// Depending on the current language, the framework will obtain
// the localized value for the  Message.UserName key
String example = Messages.get("Message.UserName") + ":" + user.getName();
```

4.3.2 Implementation Details

A flexible structure was developed to generalize and facilitate the implementation of every endpoint action. The UML diagram shown on Figure 4.2 illustrates the core classes involved in the operation. The `WebOperationResponse` class represents the response of the operation and it contains the header and body information.

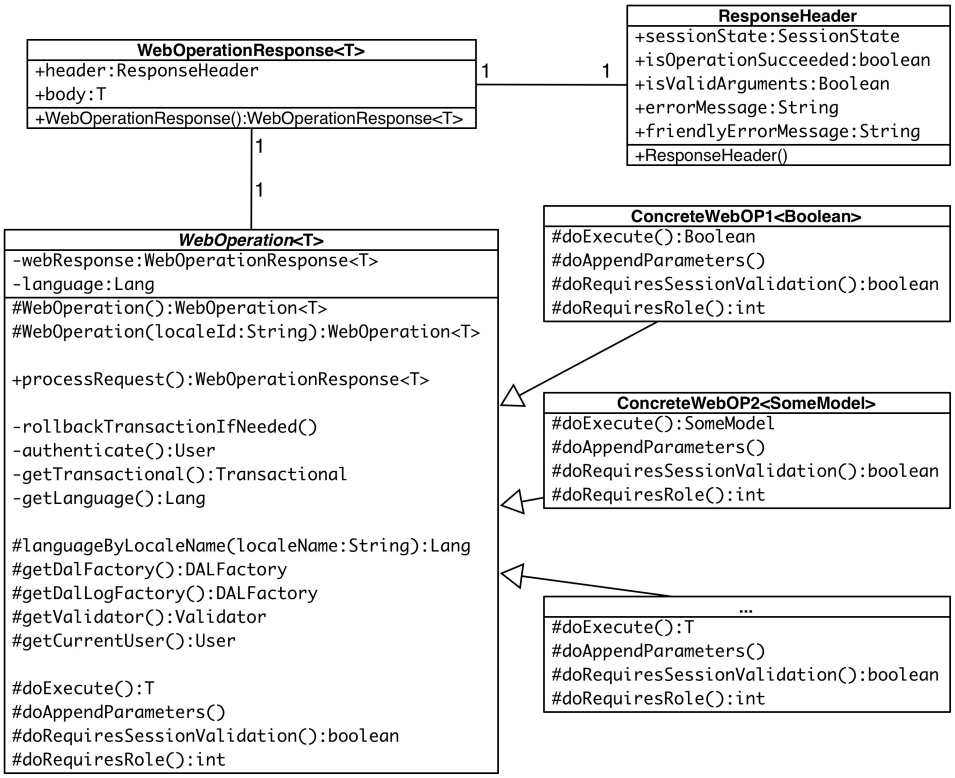


Figure 4.2: UML diagram representing the core structure of the API.

The processing flow of the web operation is shown in Figure 4.3, which describes the actions taken by the `processRequest()` method:

1. Concrete operation class adds the parameters to the validator.

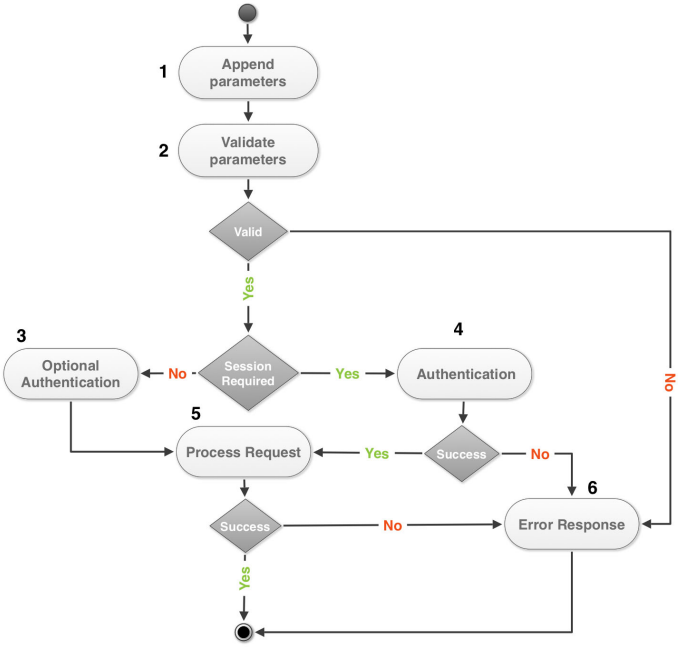


Figure 4.3: Operation's validation and processing.

2. Every parameter is validated to ensure that it is within the defined restrictions. If any of the parameters is invalid, the response header will contain the information explaining the error cause.
3. When the user authentication is optional, we verify if the sessionToken is passed, and if so, we fetch the user's information through the DAL. Any error that occurs during this phase is ignored.
4. When user authentication is required, same actions are performed to retrieve the user information, but if any of these actions fails, the error response is generated.
5. The `doExecute()` method from the child class is invoked, which is responsible for obtaining and processing the body of the response. If the operation succeeds, the response is populated with the result, otherwise it contains the information regarding the error content.

If some unexpected error occurs during the `processRequest()`, through the DAL, its cause is registered to the log database for later analysis.

Each web operation is implemented as a separate class which inherits the `WebOperation<T>` where `T` represents the type of the response upon the success. Every action delegates all the processing and validation to the respective instance of the `WebOperation` and its endpoints path are in the `conf/routes` configuration file. This file lists all the routes implemented by the API. Each route consists of an HTTP method and Uniform Resource Identifier (URI) pattern which are associated to an action method. The single route definition example is shown in Code Listing 4.4.

Code Listing 4.4: An example of the route file entry.

```
GET   /user/authenticate   controllers.User.authenticate(loc:String,tkn:String)
```

The responses of the web operation are serialised in JSON format. When sent to user, the HTTP Header representing the `content-type` is populated with `application/json` value. The serialisation is achieved by using the JSON Jackson [**?**] processor. During the development of the API we configured this framework to not serialize the attributes of the response when those are populated with `null`. The most direct example is the `ResponseHeader`, which excludes attributes presented in Code Listing 4.5 when the response is successfully executed. Entities that represent information regarding the `Location` and `User` were also configured in the same manner.

Code Listing 4.5: Optional properties of the ResponseHeader.

```
"errorMessage": null,
"validArguments": null,
"friendlyErrorMessage": null
```

This configuration allows to minimise the bandwidth occupied during the transmission of the response. With the illustrated example, we have spared 73 bytes, because that information would not be sent as part of the response, which equals to 1Mb for approximately each 15000 requests. For a large API usage, it will significantly minimise the amount of information to be sent from the server where the API is hosted and may increase the response time.

4.3.3 Authentication

Most of the implemented web operations retrieve data from the database through the DAL. But, the operations related to the authentication are different, since users can authenticate themselves on the service by using their Facebook [**?**] or Twitter [**?**] accounts. If the user exists, a session token is generated, allowing to execute API operations that require authentication. When the user authenticates himself by the first time, the service automatically creates a new account for that user. For the Facebook authentication, we use RestFB [**?**] framework that allows to obtain information regarding the users such as, name, email, among others fields. Then, the information is mapped directly to the Java classes. The framework provides an easy way to navigate through the Facebook Graph API [**?**].

Twitter's authentication service is managed in a similar way. To interact with the Twitter REST API v1.1 [**?**] we have used Twitter4j [**?**] library which already handles the authentication and provides wrappers to access user's information.

Both services implement the authentication using the OAuth 2.0 [**?**] protocol. From the application side, the user should perform authorization using the preferred service. Then, the authorization data (token, token secret, and expiry date) is sent to our authorization endpoint and used to obtain information regarding the user in question. Finally, this information is used to populate the database with user's information and consequently perform the authentication.

4.3.4 Location Creation

During the location creation, users have to pick the location positioning coordinates from the map. When those are submitted to the service, the information regarding the location such as city, country, and address, are obtained automatically through the Reverse Geocoding [**?**] process. The Google Geocoding (GG) API is used for this purpose by allowing to extract human-readable address from a map location.

The descriptive information regarding the address can be obtained in different languages. As of the time of this writing, the latest API version (v3) supports 55 different languages and dialects [**?**]. An example of a GG request (asking information about localization with latitude=38.76347 and longitude=-9.093783) is presented below:

```
http://maps.googleapis.com/maps/api/geocode/json?latlng=38.763470,-9.093783&
sensor=false&language=pt
```

Code Listing 4.6 shows an example of the GG response for the request presented above.

Code Listing 4.6: Google Geocoding JSON response example.

```
{
 "results" : [ {
     "address_components" : [
       {
         "long_name" : "Lisboa",
         "short_name" : "Lisboa",
         "types" : [ "locality", "political" ]
       },
       {
         "long_name" : "Portugal",
         "short_name" : "PT",
         "types" : [ "country", "political" ]
       },
       {...}
     ],
     "formatted_address" : "Passeio Ulisses 9, 2715-311 Lisboa, Portugal",
   },
   ...],
 "status" : "OK"
}
```

The `language` attribute is the union of the ISO 639-1 and RFC 3066 [**?**] standards, so its value is adapted according to the system language, which follows the ISO 639-1 format.

4.3.5 Location Filtering

The REST API provides a set of methods that allow to perform different operations, such as authentication, lookup for points of interest, among others. There is a specific set of

endpoints to perform the filtering of the touristic locations. Operations that obtain visited, recommended, and wanted locations are not subject to filtering criteria.

Touristic location filtering can be achieved with `/location/filter` endpoint where most of the arguments are optional. It allows to specify any combination of the arguments, easily adapting to user needs. It is possible to have any permutation of the filtering criteria such as city name, attraction, weather conditions, distance, among others.

When the geographic coordinates pointing to user's current location are retrieved from the device, the latitude and longitude are passed to the service during the queries related to location filtering. We use this information to compute the distance between the user and location, presenting the results sorted increasingly by distance. For this purpose, we have used the Spherical Law of Cosines [**?**], with the following set of equations:

$$\begin{aligned} \phi_1 &= userLat \times \frac{\pi}{180} \\ \phi_2 &= locationLat \times \frac{\pi}{180} \\ \Delta_\lambda &= (userLon - locationLon) \times \frac{\pi}{180} \\ R\bigotimes &= 6371 \end{aligned} \tag{4.1}$$

$$d = acos(sin(\phi_1) \times sin(\phi_2) + cos(\phi_1) \times cos(\phi_2) \times cos(\Delta_\lambda)) \times R\bigotimes$$

where the ϕ_1 represents user's latitude, ϕ_2 refers to the location's latitude, and the Δ_λ is the difference between the user's and location's longitude coordinate. These values are converted to radians. The $R\bigotimes$ is a constant representing the distance in kilometres from Earth's center to its surface. The result d represents the distance in kilometres between the user's specified coordinates and the touristic location at hand.

4.3.6 Integration with World Weather Online

Users can preview the temperature when they are consulting the details of the chosen touristic location. For this purpose, we have made integration with the World Weather Online (WWO) [**?**] public API which allows to access current weather conditions. Their API allows to consult detailed information regarding the weather and supports different response formats, namely the Extensible Markup Language (XML), Comma-Separated Values (CSV), and JSON. We are only consulting the information regarding the weather condition and temperature, with the response being serialized into JSON format.

The main reasons for choosing this service is that it doesn't require any commercial license, it can be used for personal and commercial purposes, and provides reliable information regarding current weather. The free version of this API is limited to 500 requests per hour.

The weather conditions and the temperature are obtained using the geographic coordinates of the location in cause. When the information is successfully obtained, it is stored into the database and considered valid for the next 3 hours. This implies that subsequent requests

for the location details will not trigger any call to the WWO API, improving significantly the response time of our service and still provide the user with the updated and accurate weather information.

Currently, the WWO API supports around 48 different statuses for classifying the weather conditions [**?**]. For this project, many of these statuses are not relevant, so we have shortened this set into 6 in order to represent the most common weather conditions. After querying the WWO service, the weather condition value is converted into our domain specified value and our users are provided with minimalistic information regarding this matter. As an example, the WWO API provides more than 15 statuses for classifying the rainy weather condition, when this information is received, it's mapped as *rainy*, thereby minimizing the amount of redundant information.

4.4 Recommender system

The Recommender system is one of the key features of the GuideMe service. In order to provide quality recommendations for our users, we have used the Apache Mahout Recommendation Engine library [**?**]. Mahout provides a varied set of the Collaborative Filtering algorithms, for user and item based recommendations. Nowadays, this library is widely used for the implementation of recommendation systems.

As we have described in Chapter 2, due to recommendation quality and algorithm performance benefits we have chosen the Item-Based Collaborative Filtering approach as our candidate for the recommender system. The implemented recommender system is implemented with the Slope One algorithm and scheduled with the Cron job to run every day at 3:00 AM. The chosen algorithm has proven to be fast, but overall performance has it's drawbacks when the datasource uses information from the database. To improve the performance of the recommendation system, we have imposed some restrictions which consist in selecting the users that should receive new recommendations. Through DAL, the service obtains a list of users who are eligible[2] for recommendations, and then for each user we apply the following equation:

$$PVu = VSR/TV, \tag{4.2}$$

where PVu is the percentage of the visited locations since last recommendation, for specific user u. TV represents the total number of visited locations and VSR refers to the number of the visited sights since last recommendation. The new recommendations are only computed if the user had an increase of 5% of the visited locations. This selection allows to improve algorithm performance by excluding users that don't have new visited locations. We consider that the change on the number of visits below 5% is meaningless regarding the recommendation results. Finally, we remove all previous recommendations and populate the database with new results.

To accelerate the recommendation process by taking the advantage of the processing power, users eligible for new recommendations are divided into equal sets. Each set is passed to a different thread where the recommendations are computed and stored to the database.

[2] By eligible users, we consider users that have visited at least one location.

4.5 Mobile Application

In this Section we describe the implementation aspects of the iOS mobile application, what kind of operations and features it offers and how those features were implemented. We start by detailing the communication layer between our service and iOS application. Then, we present the application overview, followed by aspects regarding the internationalization.

The iOS application is implemented following the concept of universal app. A universal app is a single application that is optimized for iPhone, iPod touch, and iPad devices. The final product is a representation of a single binary that adapts to the current device. When searched on the App Store [**?**] the same binary is shown for the iPhone/iPod and iPad devices instead of having different binaries for each device.

The development of the universal binary involved extra work. Because of the differences in device screen sizes, most of windows, views, and view controllers code for iPad is different from the code for iPhone and iPod touch. In addition, there are some differences between interface principles in iPhone and iPad platforms, which are described along in iOS Human Interface Guidelines [**?**].

Users usually prefer an universal app instead of a device-specific. They need to download it only once, and if an app is correctly designed, every additional feature that they activate is automatically enabled across all devices using the iCloud [**?**] service.

4.5.1 Communication Layer

A flexible interface for the communication with the service was designed to facilitate the developments of the iOS applications. Figure 4.4 shows the UML diagram describing the main classes responsible for the communication with the service infrastructure. Every concrete implementation of the model inherits the `BaseModel` and implements the `parserWithJson` method from the `ParserProtocol`. This method ensures that the model knows how to parse properties from the response serialized in JSON format.
The `GenericOperation` class is responsible for making the request to the endpoint defined by the child class. The request is executed as a background task using the Grand Central Dispatch (GCD)[3], which allows to execute the request asynchronously in a background thread. Figure 4.5 illustrates the most important steps required to perform the remote request, which are detailed as follows:

1. The class that inherits the `GenericOperation` specifies the endpoint and the HTTP method required to perform the operation.
2. The parameters are appended to the request.
3. The remote request is performed on a background thread.
4. When the response arrives, it is validated and parsed.

[3] GCD comprises language features, runtime libraries, and system enhancements that provide systemic, comprehensive improvements to the support for concurrent code execution on multicore hardware in iOS and OS X [**?**].

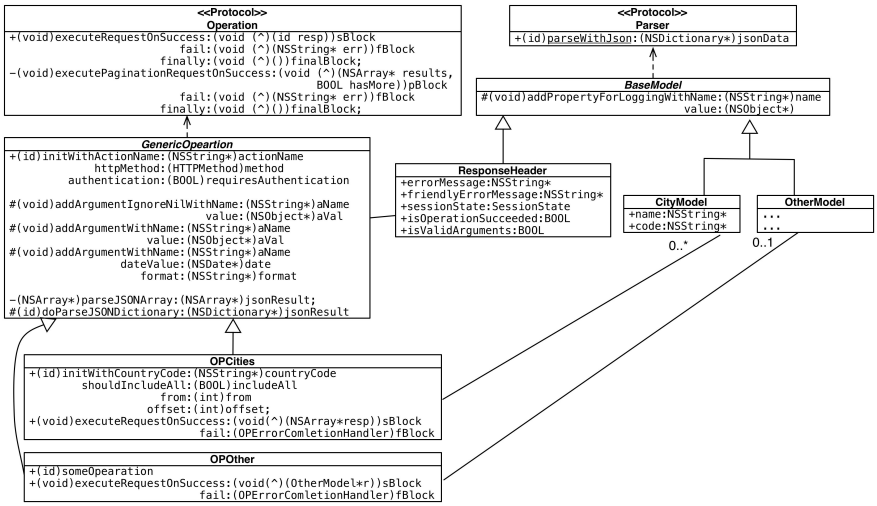


Figure 4.4: UML diagram representing the communication layer of the iOS application.

5. The failure block is invoked on the main thread when the response is invalid or the header is populated with information regarding an error.
6. The success block is called on the main thread if everything goes as expected.
7. If the user has specified the finally block, it will be executed on the main thread after the success or failure blocks.

The implementation with blocks has simplified the usage of the classes that perform the remote requests. The current approach performs the background task to handle the request and invokes the described blocks on a main thread, allowing to implement the User Interface (UI) operations directly inside the respective block. As a consequence, we minimise the amount of code to be written. Code Listing 4.7 illustrates an example of the common implementation to perform such tasks in a background thread and then update the UI on the main thread.

Code Listing 4.7: An example of the remote request using common GCD approach.

```
dispatch_queue_t getCitiesQueue = dispatch_queue_create("get_cities", NULL);
    dispatch_async(getCitiesQueue, ^(){
        NSError ** error;
        NSArray * cities = [CitiesOP performOperationWithCountryCode:@"PT"
                                shouldIncludeAll:NO from:0 count:10 error:&error];
        dispatch_async(dispatch_get_main_queue(), ^(){
            if(!cities){ /* Notify user about the error */ }
            else{        /* Handle the "cities" response here */ }
            // Perform some common operations
            });
});
```

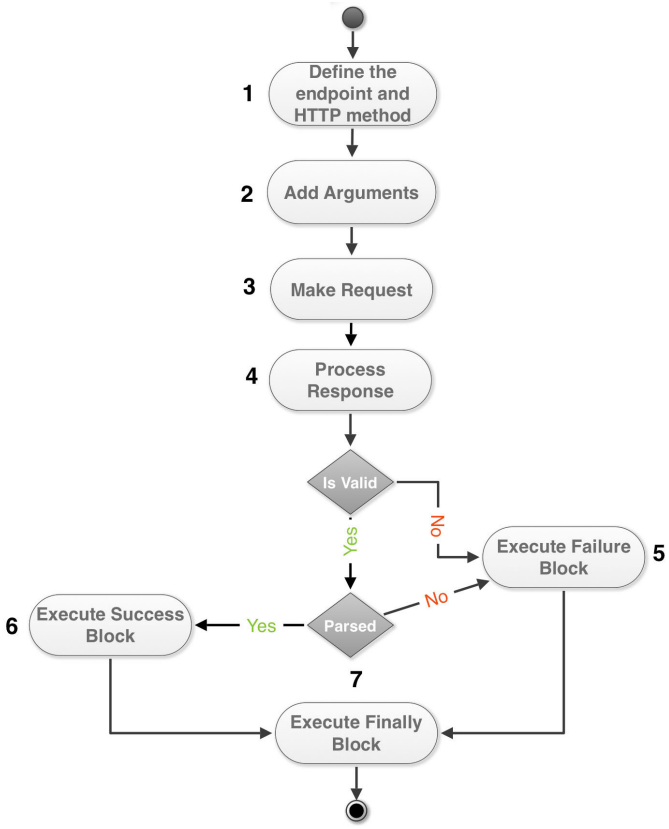


Figure 4.5: Processing of the remote request.

The adopted solution, shown in Code Listing 4.8 simplifies the readability and amount of code to be written. The scheduling to the background thread and the main thread are made internally by the implemented infrastructure.

Code Listing 4.8: An example of the remote request using the adopted solution.

```
OPCities *citiesOP = [[OPCities alloc] initWithCountryCode:@"PT"
                                                shouldIncludeAll:NO
                                                            from:0
                                                           count:100];
[citiesOP executeRequestOnSuccess:^(NSArray *cities){
    // Handle the "cities" response here
 } fail:^(NSString *errorMessage){
    // Notify the user about the occurred error
 } finally:^(){
    // Implementation of some common actions
}];
```

4.5.2 iOS Application Overview

The developed mobile application offers two interfaces. The common interface which can be accessed by any user and the interface for administration. All users can consult points of interest near their current location, apply filtering criteria (e.g. filtering by country, city, category, weather conditions, among others) to shorten the amount of results. Users also have the possibility to consult recommended locations, mark locations as visited or wanted, follow and unfollow other users.

Users with the administrative privileges have access to all of the described tasks. They can also insert new or update an existing touristic location, and consult information regarding the reported touristic locations. The main focus of the mobile applications is the users's current location, which is represented by the geographic coordinates obtained using the positioning service available on the user's device, such as GPS or Wi-Fi. By knowing the users's current location, the service can compute the distance between the touristic location and the user's location, as discussed before (Subsection 4.3.5). When the location services are not available, the service continues to allow consulting of the points of interest but without the indication of the distance between the user and the touristic location at hand.

Figure 4.6 illustrates some of the implemented features of the application for iPhone. In

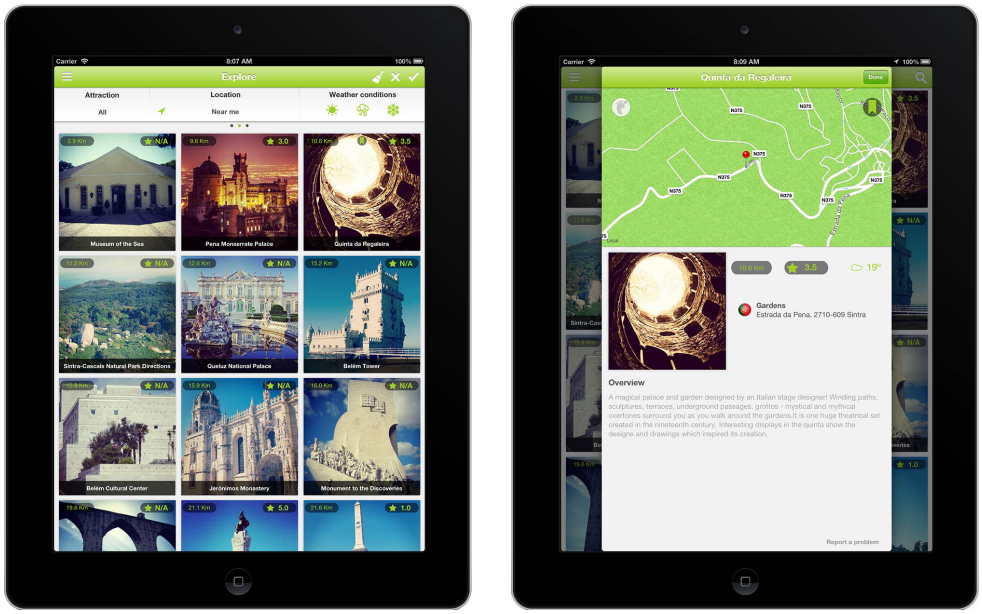
(a) Login screen. (b) Location search. (c) Location details.

Figure 4.6: Screenshots of the iPhone application

Figure 4.6 (a) we have the login screen, where the user may choose the preferred social service in order to perform login or sign up. Figure 4.6 (b) shows a list of the locations,

which have resulted from the applied filtering criteria. The detailed information regarding the chosen location is presented in a similar way as illustrated in Figure 4.6 (c).

As it was previously stated, the iPad device is also supported by the application. In Figure 4.7 (a) we have a screen with a list of the touristic locations and Figure 4.7 (b) depicts the screen with detailed information of the selected location. It is clearly visible that the content of each screen is adapted to the device resolution and differs from the one presented in Figure 4.6c (b) and Figure 4.6c (c). Both implementations share the same code with a few adjustments that distinguish the iPhone version from the iPad (universal application).



(a) Location search. (b) Location details.

Figure 4.7: Screenshots of the iPad application

The full set of features for each version of the application is documented and located in the *GuideMeMobileClient* repository.

4.5.3 Internationalization

The developed application provides translation strings for the English and Portuguese languages. If the user has configured any language other than English or Portuguese, the application assumes English by default. During the remote requests, the language identifier[4] used for loading localized resources is passed to our REST API, ensuring that the information is retrieved using the same locale as the one configured by the device.

[4] The language identifier in the iOS application is defined in ISO 639-1 format. The same type of locale identifiers is supported by the REST API.

4.6 Push Notifications

We use the APNS service to notify users about important events that occur regarding them. A use case of this concept, is when the user is followed by another, the push notification is sent detailing the occurred event as illustrated on Figure 4.8, otherwise this action could pass unnoticeable to the followed user.

Figure 4.8: An example of the push notification sent by GuideMe service.

To implement this feature, we have used the Java-APNS [**?**] implementation that already solves many of the difficulties, such as, communication with the APNS, establishing of a secure connection, among others. In order to make the provider communicate with the APNS and later the APNS with the device, it was required to correctly configure the certificate in Java-APNS framework and the Provisioning Profile [**?**] during the deployment of the mobile applications[5]. Each certificate is limited to a single application, identified by its bundle ID. It is also limited to one of two environments: development or production. These environments have their own assigned IP address and require their own certificates. It is also required to obtain provisioning profiles for each of these environments in order to receive the push notification when the application is deployed to the mobile device. The communication flow for sending the push notification, between the source and destination devices, is illustrated on Figure 4.9.

The steps regarding the push notification delivery are as follows:

1. The source device performs a remote request to execute an action (for example, follow some user).
2. After a successfully performed operation, the routine from the APNS provider is invoked.
3. A new notification is created and sent to the APNS server.
4. When possible, APNS delivers the notification to the destination device.

[5] To develop and deploy the provider side of an application for push notifications, it is required to obtain the SSL certificates from the appropriate Apple Developer Center [**?**].

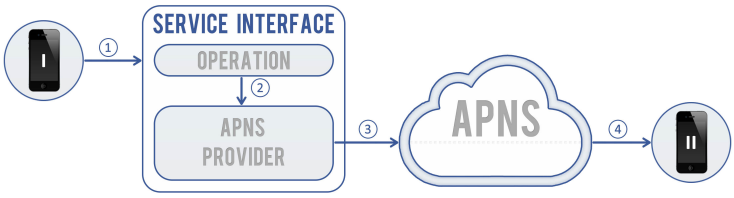


Figure 4.9: Delivery of the push notification to the destination device.

As it is stated by Apple, the provider is responsible for not propagating push notifications to devices that are not able to receive them. We have implemented a service, that is executed by Cron job every 10 minutes. This service connects to the APNS Feedback service and obtains the device tokens that have repeatedly reported failed-delivery attempts. Then, we update the database by deactivating the reported device tokens. This periodic service was implemented using the Java-APNS framework and uses the data access layers to update the status of the device token or report errors to the log database when it is necessary.

4.7 Web Application

In this Section, we describe the development aspects of the Web application. We detail the features and functionalities offered by this application.

4.7.1 Implementation Details

The Web application was developed using the same framework as the REST API. Play Framework offers powerful Scala[6] template engine which was inspired by ASP.NET Razor [**?**]. The template system was designed in a manner that facilitates the development of the web applications, where the '@' character indicates the Scala statement and it does not require to explicitly close the code-block. An example of the template engine is shown in Code Listing 4.9.

Code Listing 4.9: An example of the Scala template.

```
@(location: LocationModel)
@if(location == null){
   <h1>Location is invalid</h1>
}else{
   <h2>location.getTitle()</h2>
   <ul>
      @for(user <- location.visitedBy()) { <li>@user.name</li> }
   </ul>
}
```

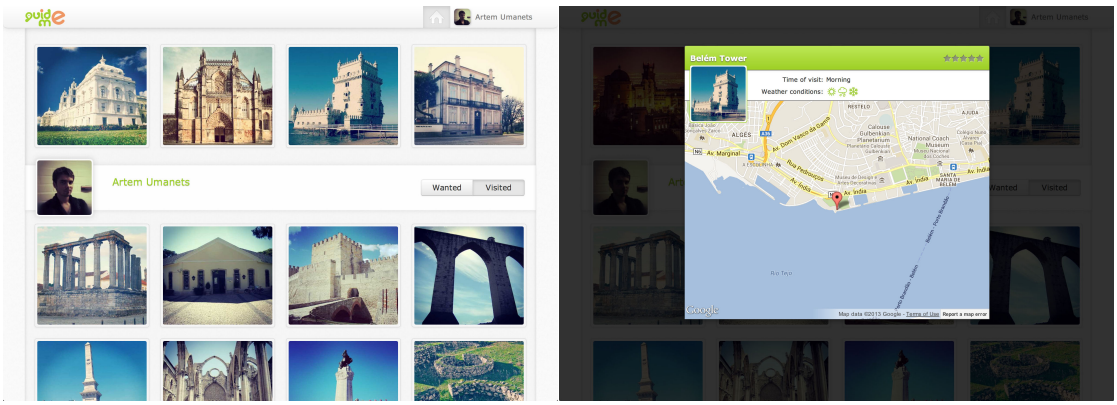
[6] Scala is an object-functional programming and scripting language for general software applications, statically typed, designed to concisely express solutions in an elegant, type-safe and lightweight manner.

The Play application follows the Model View Controller (MVC) architectural pattern applied to the Web architecture. This pattern splits the application into separate layers: the Presentation layer and the Model layer. The Presentation layer is further split into a View and a Controller layer. The Model holds the domain-specific representation of the information on which the operation executes. The View renders the model into a form suitable for interaction, while the Controller responds to the HTTP Requests and applies changes to the underlying model. The Views are designed using the Bootstrap [**?**] which is designed for faster and easier Web development. It defines a set of generic components and layouts which are used across the application and removes the compatibility barrier between different browsers.

4.7.2 Web Application Overview

The Web application implements a subset of functionalities of the mobile application. It allows to users authenticate using their Facebook account, consult visited, wanted and recommended locations, and the detailed information of each location. They can visualize their profile page with the information regarding the current account. On the details page of the touristic point, users can consult the exact location through the Google Maps service.

The Figure 4.10 (a) illustrates the page with a set of visited locations. By selecting any of the listed locations, its details are shown in a similar way as one from Figure 4.10 (b).



(a) Visited locations. (b) Location details.

Figure 4.10: Screenshots of the Web application

4.7.3 Internationalization Support

The Web application supports localized content using the same mechanism as the REST API. The HTTP cookie [**?**] with the language identifier is populated on the first request, and by default it refers to the English language. The action that changes the language overrides the language identifier with a new value which can be changed from either the login or profile page.

Some Web operations are implemented in asynchronous mode using the JavaScript (JS) and Asynchronous JavaScript and XML (AJAX) technologies in order to make the user's interaction more intuitive. We have used jQuery [**?**] library to make the development of the JS code easier and more readable. The Scala template engine is not able to parse the external JS files, so a problem with localized messages appeared when it became necessary to show messages to user through JS. The solution that was found consisted in using the extension for i18n properties for the jQuery library, consist in performing the AJAX request to the implemented endpoint `/i18n`, which returns the content of the appropriate messages file from the `/conf` directory. Then the localized messages can be accessed as follows: `$.i18n.prop("MessageKey")`.

4.7.4 REST API Documentation

Users with the administrative role can consult the documentation of the REST API on their profile page. Each endpoint is described as a separate text file located in the `public/api` directory. The template for the documentation file is shown in Code Listing 4.10:

Code Listing 4.10: Organization of the documentation information.

```
@->Title:<Title>
@->Description:<Endpoint Description>
@->Method:<HTTP Method>
@->Url:<Endpoint URL>
@->AuthenticationRequired:<Indicates when authentication is required>
@->Permission:<User Permissions>
@->RequiredParameters:
param1[Type] - Description of parameter1
...
@->OptionalParameters:
param2[Type] - Description of parameter2
...
@->CallExampleUrl:<Example of the invocation URL>
@->OutputExample:<Example of the response, usually in JSON format>
```

Each property is separated by `@->` and the first `:` separates the property name from its content. The endpoints are obtained by listing all the files from the `public/api` directory. Each of them is mapped into a model that is more easily integrated in the Scala template. An example illustrating the organization of the endpoint documentation is shown in Figure 4.11.

The approach for reading and parsing the documentation data is not very efficient. The usage of the database would be more appropriate but we did not want to introduce more databases and complexity to the current project. Improving this part of the project is one of the goals in the future.

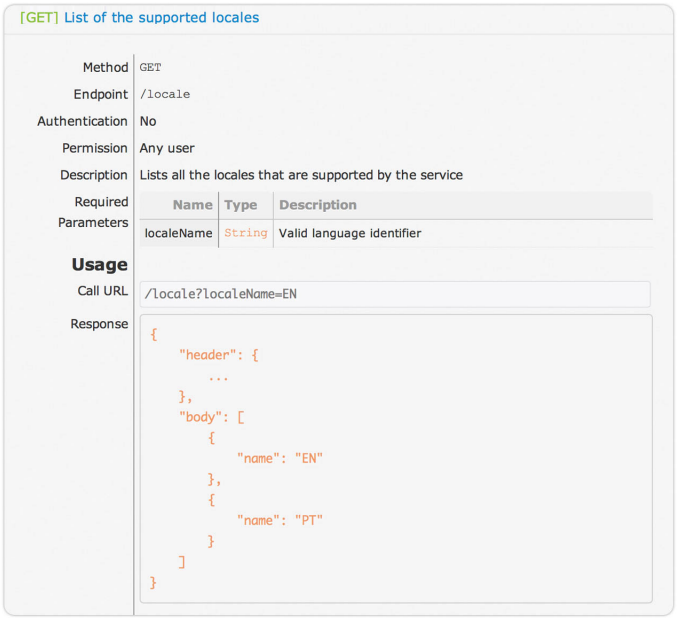


Figure 4.11: An example of the documentation entry.

4.8 Additional Developments

In this Section, we describe other developments that were made for this project.

4.8.1 Software Testing

Unitary test methodology is used to check the quality of the code and to facilitate the detection of bugs that may be discovered in the future. Currently, there are around 120 unit test for the REST API service and 190 unitary test for the Service and Log DAL. All unitary tests share the same pre-populated database and perform different kinds of operations and queries to ensure that the tested operations perform correct actions.

In order to analyse the code coverage, we have used the EclEmma [**?**] plugin for Eclipse IDE. This analysis helped us to improve coverage of the implemented unit tests, lowering the chance of bugs. The implemented unit tests are covering around 90% of the code related to the Service DAL and around 75% of the Log DAL. The uncovered code is mainly related to the entities generated by the Hibernate framework.

The iOS application implements 35 unit tests which aim is to perform the remote requests to the REST API. These tests help to detect any unpredictable changes that may emerge in the future.

4.8.2 Multi Environment Support

The DAL and REST API components were carefully designed in order to support multiple environments by mean of configuration. At the DAL layer, it is possible to specify different

configuration files for testing and production purposes. The same applies to the REST API, which allows to adjust the configurations that vary between test and production environments.

4.8.3 REST API Incoherence

The developed REST API invalidates some concepts of the REST architecture due to external causes. Usually, the `DELETE HTTP` method is used for resource removal and `UPDATE HTTP` method updates the resource. Some firewalls may block these HTTP verbs, therefore the API was developed in order to use the POST method for creating, updating and removing the resource and GET to consult its information. The type of the operation is indicated as part of the URL request (e.g. the operation to remove the user is implemented as follows: POST `http://host/user/delete?id=5` instead of DELETE `http://host/user/5`).

5

Experimental Evaluation

In this Chapter, we detail the experimental results obtained during different phases of the project. First, we present the information about the study that allowed us to pick the most suitable algorithm for the recommender system. Then, we describe the performance tests that were made in order to establish service usage limits.

5.1 Recommender System

In this Subsection, we present the experimental results regarding the evaluation of different recommender systems. To provide reliable evaluation results, we have implemented a few variations of the recommender systems using different algorithms, provided by the Mahout library. We analyse the performance and the quality aspects of the produced recommendations, which have helped to choose the best algorithm for our needs.

For the evaluation purpose we have used the MovieLens [**?**] datasets provided by GroupLens [**?**] research. MovieLens is a movie recommender site used to study recommendation engines. We have chosen this dataset due to the equality of the rating scale chosen for the GuideMe service.

For the recommender system evaluation, we have used the datasets illustrated on Table 5.1.

Table 5.1: MovieLens datasets used for evaluation.

N. Ratings	N. Users	N. Movies
100 000	1000	1700
1 000 000	6000	4000
10 000 000	72000	10000

In order to measure the accuracy of the different algorithms, we have used the Mean Absolute Error (MAE) metric. Absolute error value is computed for each rating-prediction pair

$< p_i, p_q >$, then is computed the average. Formally,

$$MAE = \frac{\sum_{i=1}^{n} |p_i - p_q|}{n} \tag{5.1}$$

which refers to the deviation (in a range $0 \leq MAE \leq 1$) of the recommendations from their user-specified values [**?**]. The lower the MAE, the more accurately the recommendation engine predicts user ratings. For most of the evaluations, we divided the datasets shown on Table 5.1 in two different sets, the training set which holds around 80% of data and the test set with the remaining 20%. In other evaluation scenarios, we indicate the distribution of data between the training and the test datasets.

To correctly measure the MAE value, the initial idea was to use the K-Fold Cross Validation[1] method [**?**]. It was not possible to apply this technique, since the latest version of the Mahout library (v0.7) does not provide any hooks for the implementation of this evaluation methodology. There is no way to divide the dataset into the training and the test sets and repeat the recommendation process alternating between K sub-samples. Instead of K-Fold Cross Validation, to ensure more accurate MAE results, each algorithm was executed 5 times, being the evaluation value the average of those executions.

For the following evaluations, we have chosen four different algorithms, Slope One, IB with Euclidean Distance, IB with Log Likelihood, and IB with uncentered cosine. The evaluation of the performance of these algorithms is illustrated on Figure 5.1 where the lowest MAE value is achieved by the Slope One algorithm. With the growing of the dataset, the presented algorithms become more accurate with the recommendations.

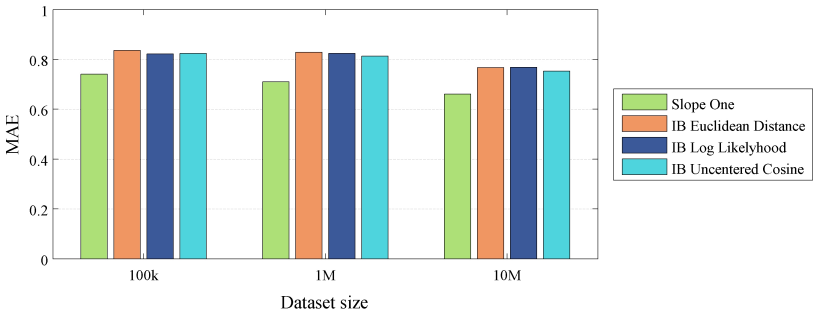


Figure 5.1: Performance of different recommender algorithms.

On Figure 5.2 we show the execution time of the analysed algorithms. The Slope One requires a lot of physical memory in order to compute the data model which then is used for

[1] K-Fold Cross Validation is a model validation technique. It is commonly used in environments where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. Consists in dividing the dataset into K sub-samples, where $k-1$ sub-samples are used as training data and a single sample is retained for validation. The validation process is repeated K times, with each of the K sub-samples used exactly once as the validation data.

recommendations, but it presents benefits in terms of performance when compared with other implementations.

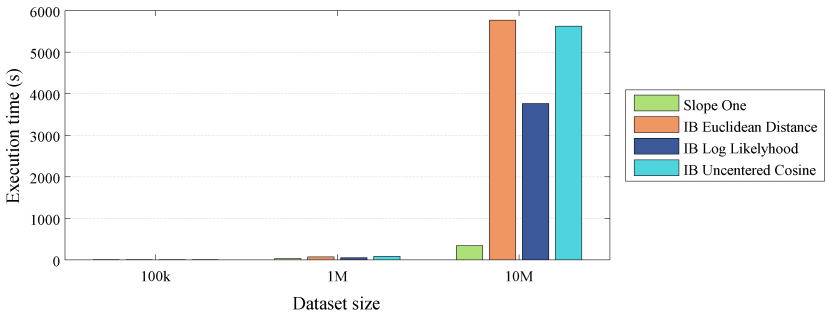


Figure 5.2: Evaluation of the execution time.

The presented analysis allowed us to make an appropriate choice of the algorithm for our recommender system. The forthcoming evaluations are based on variations of the Slope One algorithm. In order to minimize the memory consumption, we calibrate the algorithm with different sizes for the diff storage[2]. For the following evaluations we are considering the unlimited diff storage, diff storage with 500000, and 100000 entries. On Figure 5.3, we show that the MAE value is lower for the unlimited diff storage. The results are satisfying when the diff storage is limited to 500000 entries, thus minimizing memory consumption.

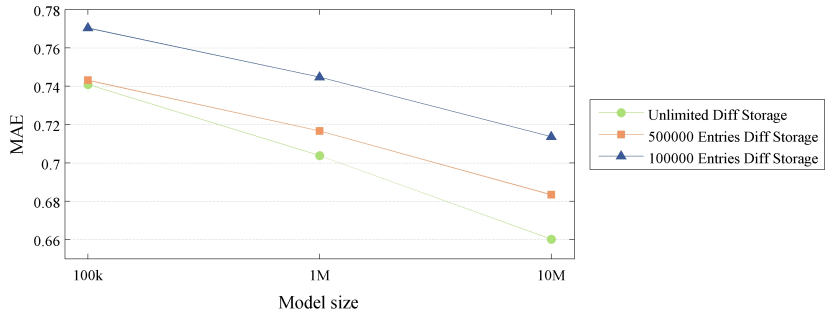


Figure 5.3: Sensitivity of the diff storage size for the Slope One algorithm.

Some evaluations were made by varying the percentage of the training and the test sets as illustrated on Figure 5.4. We have concluded that a larger dataset implies better recommendation results. By using the diff storage limited to 500000 entries, the MAE value for the

[2] Diff storage contains a precomputed average differences in preference values between all pairs of items. The number of diffs is configurable and the algorithm will try to keep the most useful diffs. Most useful here means those diffs between a pair of items that turn up most often together in the list of items associated with a user [**?**].

training set of 80% is around 0.7, better than the other algorithms reported on Figure 5.1.

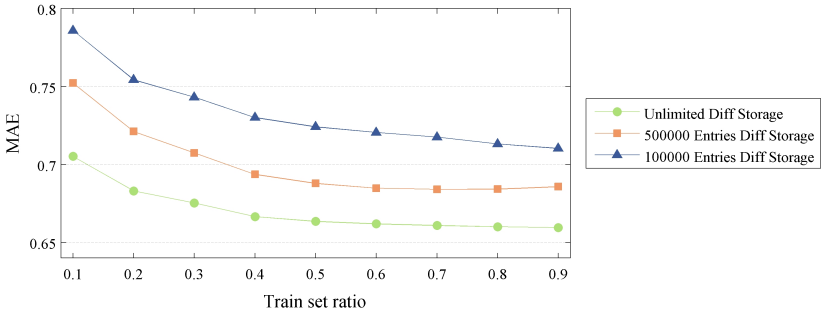


Figure 5.4: Variation of the dataset ratio for the Slope One algorithm.

These evaluations allowed us to choose the more suitable recommendation algorithm for our dataset, being our recommendation engine implemented with the Slope One algorithm configured with unlimited diff storage. In the future, with the grow of the dataset, we will limit the algorithm to 500000 diff storage entries.

5.2 Load Tests

In this Section, we detail the utility of the performed load tests, which allowed to evaluate the performance of the implemented service and detect issues that were affecting negatively its functionalities. All the tests are made to the REST API, deployed on Amazon EC2 [**?**] instance with specification described in Table 5.2.

Table 5.2: Amazon EC2 instance specification.

Type	Details
Operating System	Ubuntu Server 13.04.64
CPU	Intel® Xeon® CPU E5-2650 0 @ 2.00GHz, Single Core
RAM	589MiB

We have used the Gatling Stress Tool [**?**] to perform the load tests. The tests are named as simulations and are written in Scala. Each simulation is composed by a scenario, which represent users' behaviours, composed by one or multiple requests. To achieve reliable feedback, we have created a test scenario based on eight different requests, as detailed in Table 5.3.

Table 5.3: Requests for the testing test case.

Operation Sequence	Details
1. List Locations (Simple)	Obtain a list of locations without applying any filtering criteria.
2. List Locations (Complex)	Retrieve a set of locations with applied filtering criteria.
3. Location Details	Fetch complete information regarding a given location, simulates an action of consulting the location details.
4. Mark as Wanted	Add the location to the wanted list.
5. List Recommended	Obtain a list of the recommended locations.
6. List Wanted	Consult a list of all wanted locations that belong to some user.
7. List Following	Obtain a list of users that are followed by a given user.
8. List Followers	Consult a list of users that follow a given user.

The created scenario was executed several times varying the number of users within range $\{1, 2, 5, 10, 20, 50, 100, 200\}$ where each user performs the eight requests, described in Table 5.3. Two different metrics were extracted from the execution of the scenario, one representing the number of the requests per second and other referring to the average request execution time. Figure 5.5 illustrates the relation between the number of users and the number of requests executed per second. The performance of the current scenario becomes slower when it is executed by more than 200 users.

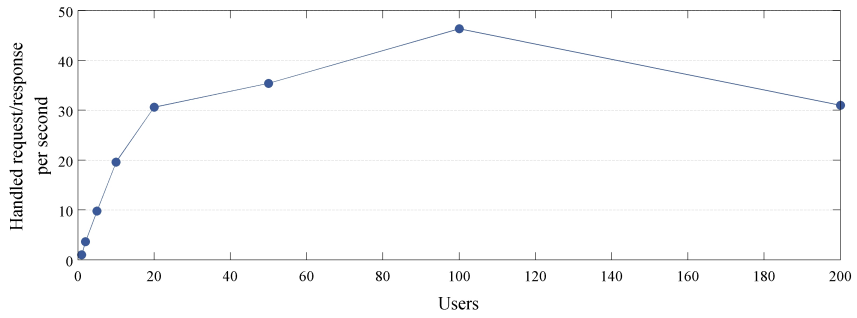


Figure 5.5: Evaluation of the maximum number of the requests per second.

Figure 5.6 shows the average request/response duration when the operations are performed by varied number of users. The system handles well around 100 users, but it becomes slower with increase of the users per request.

When the load tests were performed by the first time, those have helped to detect the database connection leaks. The life cycle of the connection was not correctly managed within DAO, which was surpassing periodically the maximum number of allowed database

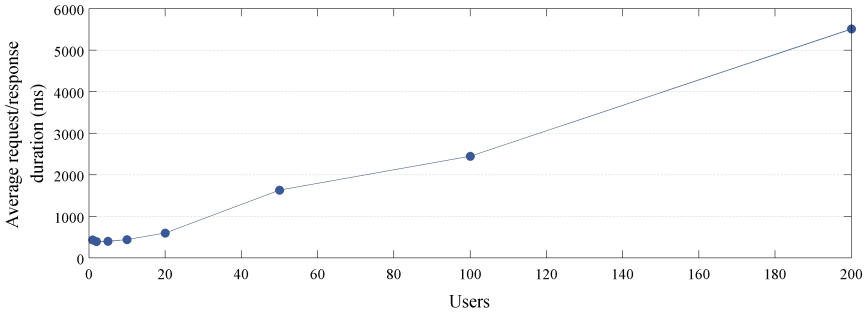


Figure 5.6: Evaluation of the average request time.

connections.

These created tests not only helped to evaluate the performance of the system, but also to correct the critical bugs that have emerged during this phase.

5.3 Usability Evaluation

In this Section, we describe the usability evaluation process based on a survey. We begin by introducing the organization of the questionnaire and then we describe the statistical data regarding some of the questions posed to the users.

In order to evaluate the usability of the developed application, we have collected the feedback from 30 users which have answered to a survey included in Appendix A.3. The survey is divided into four main categories, the general information where we collected the user's gender and age range, followed by a set of questions related to user's experience with mobile applications. The third section of the survey is composed by questions regarding the specific screens of the applications, where the user can rate the usefulness and attractiveness of the designed application. In the fourth section we ask the user if he/she would install the application in the future and optionally to leave some notes that may improve or enrich the application.

Users have started by fill in the first two sections of the questionnaire. After, they were provided with the iOS device running the application and a maximum of 5 minutes to interact with it. No help were given during that time. Finally, they were asked to fill the third and fourth sections of the questionnaire describing their opinion and impressions regarding the application.

Most of the users have liked the implemented set of features and the overall design of the application. Below, we present some statistical data that supports this statement. Figure 5.7 shows the distribution of the ratings that people have given to classify the appearance and content organization in the different screens. The used rating scale is between one (unattractive and difficult to use) and five (attractive and easy to use).

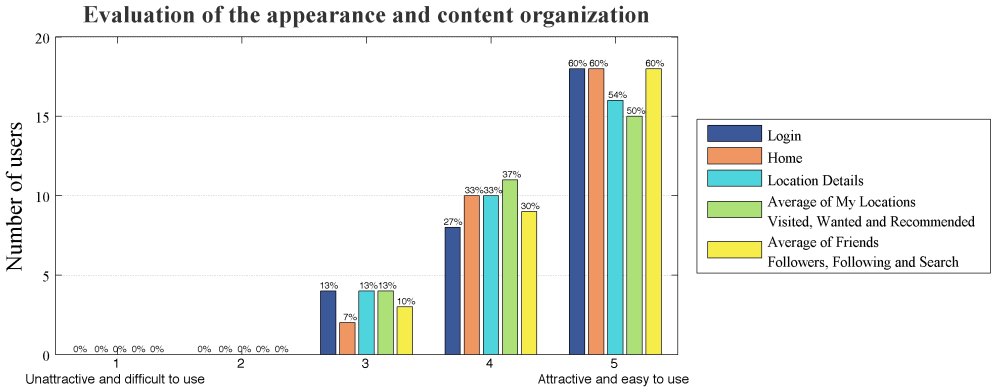


Figure 5.7: Evaluation of the appearance and content organization.

All screens share the similar rating distribution, where the majority of ratings are between 4 and 5, with a small group of users that have classified the screen with 3. The similar applies to the usefulness and importance of the available filters as well as the organization of the touristic location details screen, which evaluation is illustrated on Figure 5.8.

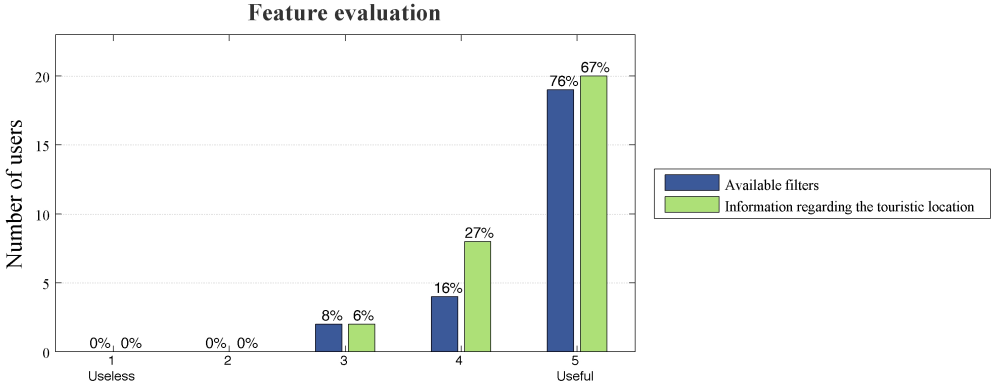


Figure 5.8: Evaluation of the implemented features.

This analysis helped us to conclude that with the developed application we can reach the majority of users and provide them with an attractive and easy to use application for touristic assistance. On Figure 5.9 is shown the distribution of the users that have answered *Yes* and *No* to the following question: *In the future, would you install this app?*

The majority have answered as *Yes*. But, the people who have answered *No*, have mostly answered to the questions 2.3 and 2.4 as follows:

- 2.3 How often are you using the external mobile applications?
  - I have my set of favourite apps that I use regularly

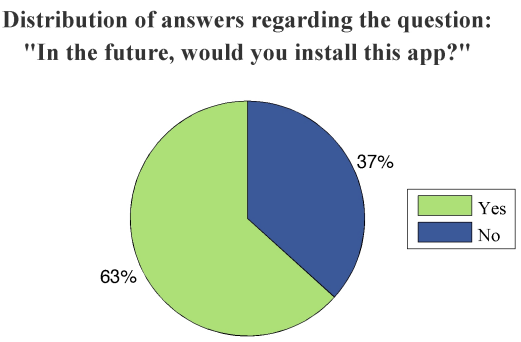


Figure 5.9: Distribution of users that would like to install the application in the future.

  - I rarely use external applications
- 2.4 Are you using any tourist guide application?
  - No
  - Yes, rarely

It is very difficult to achieve this type of users but they do not influence the evaluation of the application in a negative manner. The conducted evaluation has helped to understand our targeted audience and has ensured that the application is attractive and usable, with some interesting features to our users.

6

Conclusions

In this Chapter we start by taking the conclusions from the developed project and in Section 6.1 we present some technical and functional aspects that can be improved in the future.

Many people are stuck in their daily routine and when the time comes to travel, they choose to spent more time (and money) to visit well known touristic locations such as Eiffel Tower in Paris or the Big Ben in London. Sometimes they forget or ignore the fact that their home country also has great places to visit and to spend a great time. Many approaches for touristic guides have been proposed, but all of them are mainly focusing in the well known touristic locations.

We developed a service, and its client applications with web and mobile interfaces, which facilitate the discovery of the beautiful and previously unseen touristic points. Users can visualize places around their current geographic location. We have integrated a recommender engine which follows the collaborative filtering method. This service helps users to discover new sights without any effort from their part. The recommended information is based on locations previously rated by the user, when touristic locations are marked as visited. During the project development, in order to achieve better results, we have focused on the following issues:

- Study of the similar solutions for the problem in question.
- Analysis of different techniques and algorithms for the reliable recommender system.
- Careful design of the iOS and Web applications in order to make them look simplistic and to minimize the user's learning curve.

The chosen technologies are mostly open source, the combination of the MySQL, Hibernate and Play framework have made the server side of the GuideMe service functional and robust, while the iOS SDK allowed to develop a well designed and easy to use mobile application. Its usage will contribute to tourism, by promoting all kind of touristic locations, even the lesser known ones in the proximity of the users location. Its social component allows for users to interact between themselves.

6.1 Future Work

In this Section we approach directions for improvements and the future work that should be made in order to make the current implementation more robust for massive use.

When the user authenticates himself, the API provides the authentication token which is used to validate the user's session. Almost every request to the REST API requires that one passes the authentication token for identification. An alternative could be to pass one's id from the database, but this could compromise the user's private information. To overcome this security issue, the API will be deployed to a secure web server with Secure Socket Layer (SSL) certificate in order to keep the authentication token encrypted and to ensure information privacy. With use of the SSL certificate, the information between the sender and the server becomes unreadable to everyone except to the involved parties.

In the future we intend to implement the full set of functionalities in the Web application. By supporting most of the features through the Web interface, it would be possible to reach more users. We will use the Facebook and Twitter to expand our service with more users, by inviting friends of the registered users. It will be possible to leave descriptive feedback for the visited locations and exchange messages between users.

Currently, the recommendation system uses the information regarding all the visited locations, but the user's preference may change over time. We thought about the possibility to improve the quality of the recommender system by implementing the concept of the sliding window, which would include considerable number of the user's last visits, but not all. With this approach, the produced recommendations would be more similar to the user's last visits, avoiding the recommendation of the touristic locations in which user is not interested.

Another aspect to be improved is the social interaction between users. We intent to add support for user to leave his appreciation regarding the visited location, which can be read by other users. Current applications will also provide the feed page, where users will be able to consult what places their friends have been visiting recently.

The new version of the iOS (iOS7) was released on September 18 of 2013. It brings some new design concepts (more simplistic) to the iOS platform. As the development of the GuideMe application has started sooner, its UI was designed to be simple but it does not completely follow the design concepts of the new operating system. Our goal is to redesign some screens in order to perfectly fit the iOS7 UI.

A

Appendix

In this Chapter, we have included some implementation which is referenced along with the current report.

A.1 Service Database - Entity Relationship Model

Figure A.1 shows the ER model which describes the Service Database.

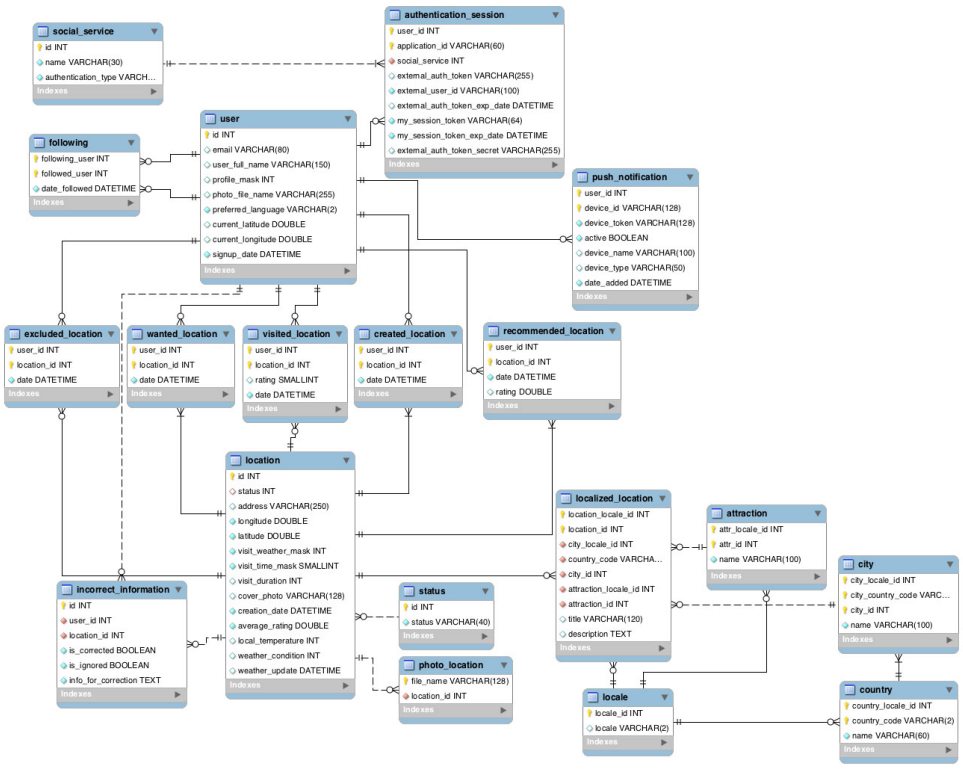


Figure A.1: Service Database ER model.

A.2 Log Database - Entity Relationship Model

Figure A.2 shows the ER Model which describes the Log Database. It allows tracking of the different events that may occur (errors, warnings, among others) for different service components.

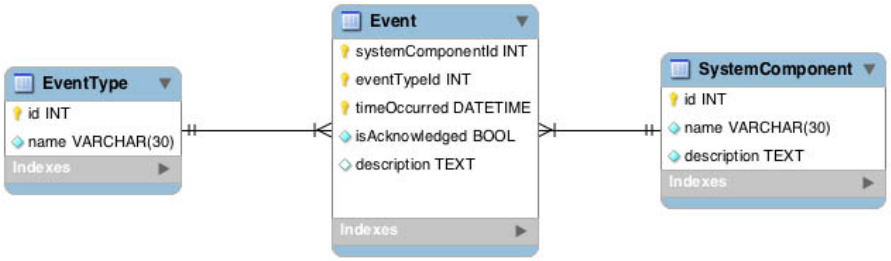
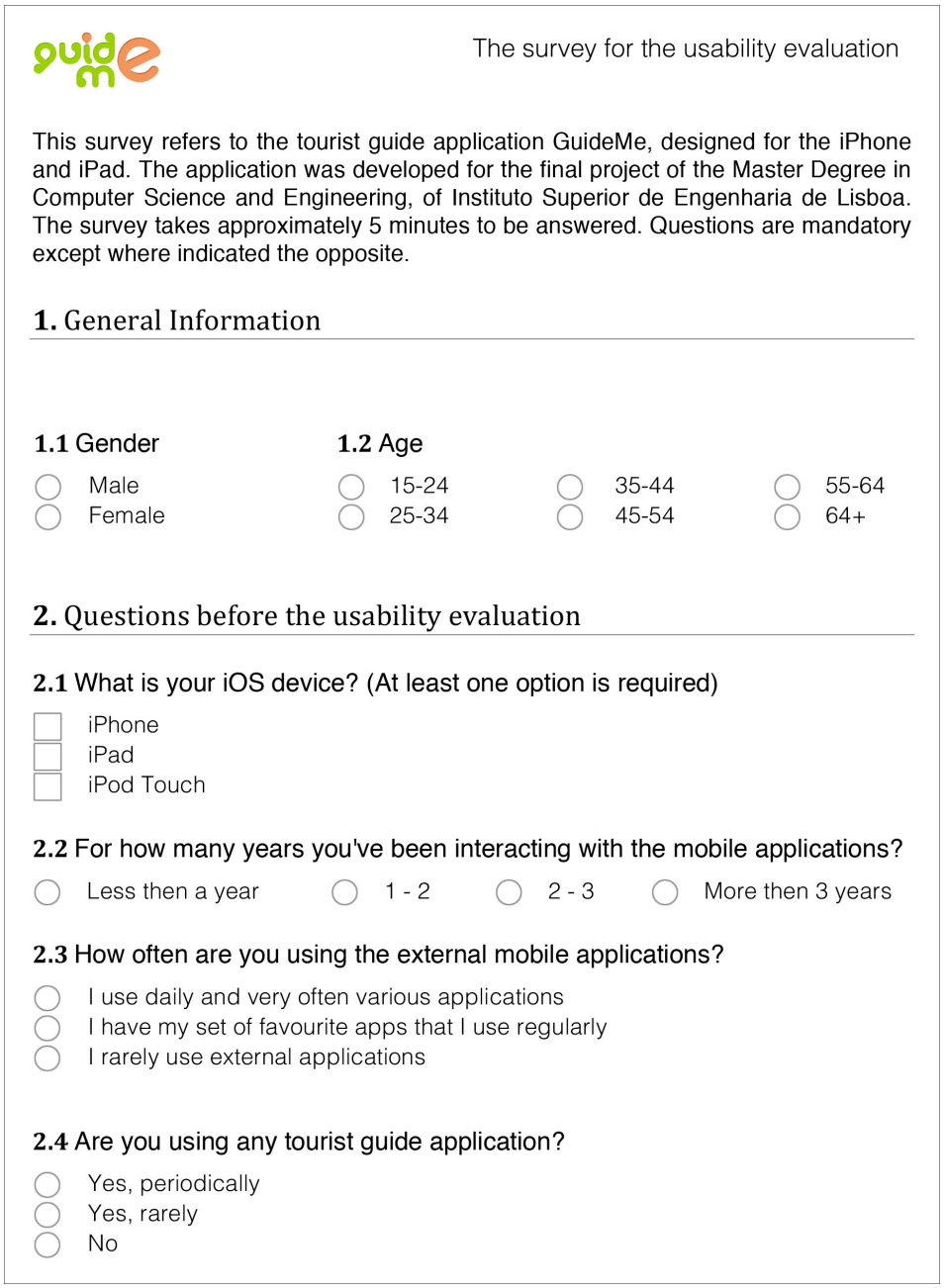


Figure A.2: Log Database ER model.

A.3 Usability Evaluation Survey

This Section contains a questionnaire that have been used during the usability evaluation process.



The survey for the usability evaluation

This survey refers to the tourist guide application GuideMe, designed for the iPhone and iPad. The application was developed for the final project of the Master Degree in Computer Science and Engineering, of Instituto Superior de Engenharia de Lisboa. The survey takes approximately 5 minutes to be answered. Questions are mandatory except where indicated the opposite.

1. General Information

1.1 Gender

- ◯ Male
- ◯ Female

1.2 Age

- ◯ 15-24
- ◯ 25-34
- ◯ 35-44
- ◯ 45-54
- ◯ 55-64
- ◯ 64+

2. Questions before the usability evaluation

2.1 What is your iOS device? (At least one option is required)

- ☐ iPhone
- ☐ iPad
- ☐ iPod Touch

2.2 For how many years you've been interacting with the mobile applications?

- ◯ Less then a year
- ◯ 1 - 2
- ◯ 2 - 3
- ◯ More then 3 years

2.3 How often are you using the external mobile applications?

- ◯ I use daily and very often various applications
- ◯ I have my set of favourite apps that I use regularly
- ◯ I rarely use external applications

2.4 Are you using any tourist guide application?

- ◯ Yes, periodically
- ◯ Yes, rarely
- ◯ No

Figure A.3: Questionnaire - Page 1 of 3

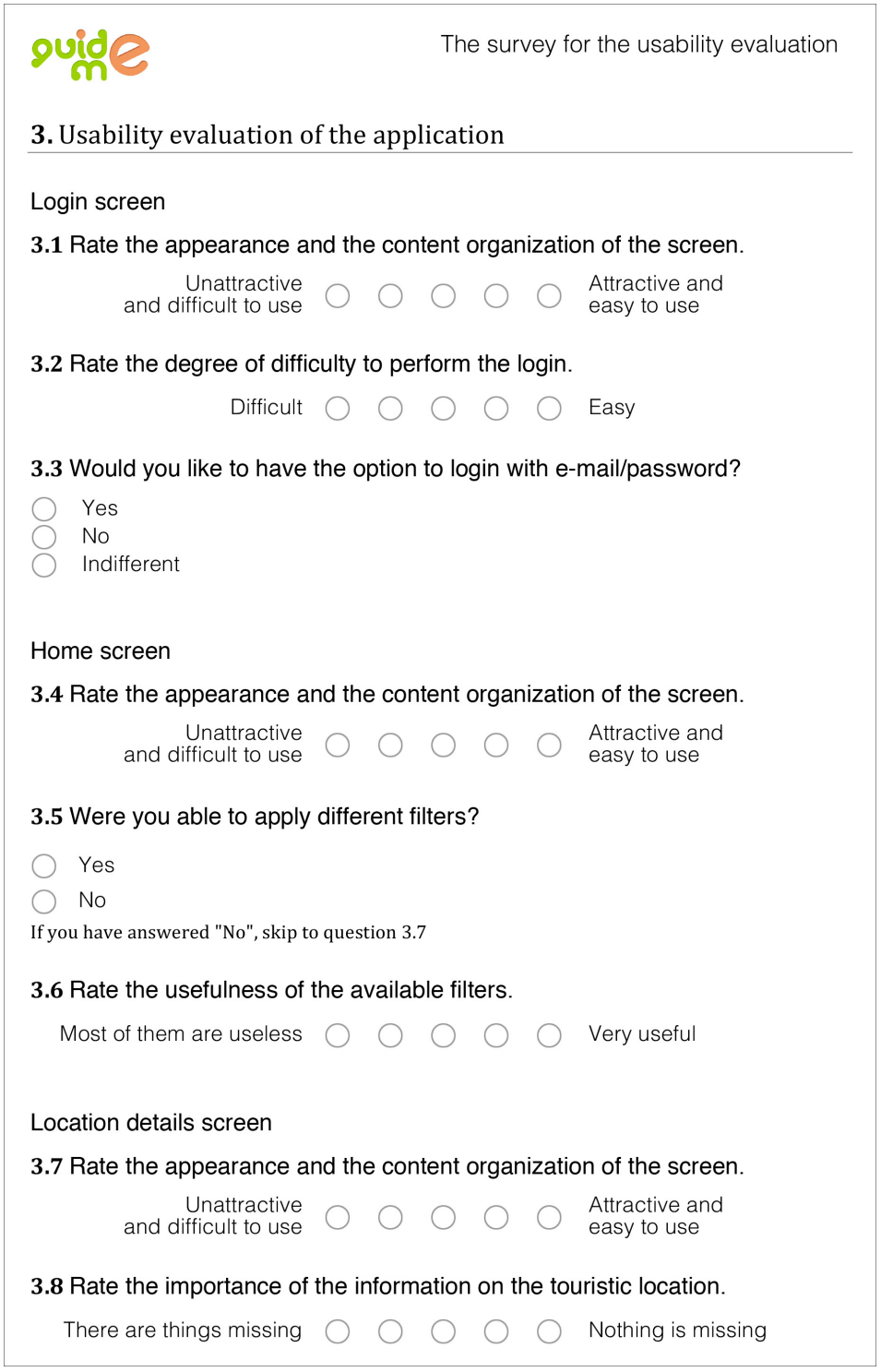


Figure A.4: Questionnaire - Page 2 of 3

Figure A.5: Questionnaire - Page 3 of 3