UNIVERSITAT POLITÉCNICA DE CATALUNYA
(UPC) Barcelona Tech

FACULTAT D'INFORMÁTICA DE BARCELONA (FIB)

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

Development of a web application for touristic data categorization and trip planning.

Advisor:
Josep Liuís Larriba-Pey
Data Management Group (DAMA)

Author:
Marina Ciavarra

22 October 2019

Contents

List of Tables III

List of Figures V

1 Introduction 1
- 1.1 Data retrieval and validation . . . 1
- 1.2 Data categorization . . . 1
- 1.3 Tour creation . . . 2
- 1.4 Thesis Structure . . . 3

2 State of the art 5
- 2.1 Existing Data Set . . . 5
- 2.2 Data collection . . . 6
- 2.3 Map creation tools . . . 6
- 2.4 Trip planning . . . 8

3 Methodologies 11
- 3.1 Agile . . . 11
- 3.2 Scrum . . . 13

4 Technologies 15
- 4.1 React . . . 15
- 4.2 React Redux . . . 16
- 4.3 React Router . . . 17
- 4.4 Node.js . . . 17
- 4.5 Express . . . 18
- 4.6 PostgrSQL . . . 19

5 Requirement 21
- 5.1 Functional Requirement . . . 21
  - 5.1.1 Use Case UC01: Register . . . 21
  - 5.1.2 Use Case UC02: Login . . . 23
  - 5.1.3 Use Case UC03: Add new category . . . 24
  - 5.1.4 Use Case UC04: Add new subcategory . . . 25
  - 5.1.5 Use Case UC05: Add new point of interest . . . 26

5.1.6 Use Case UC06: Generate a touristic tour . . . 27
5.1.7 Use Case UC07: Review and validation of categories and subcategories 29
5.1.8 Use Case UC08: Review valid categories and subcategories . . . 30
5.1.9 Use Case UC09: Review invalid categories and subcategories . . . 31
5.1.10 Use Case UC10: Review and validate points of interest . . . 32
5.1.11 Use Case UC11: Review valid points of interest . . . 33
5.1.12 Use Case UC12: Review invalid points of interest . . . 34
5.2 Non Functional Requirement . . . 35
5.2.1 Usability . . . 35
5.2.2 Maintainability . . . 35
5.2.3 Performance . . . 35
5.2.4 Platform compatibility . . . 36
5.2.5 Reliability . . . 36
5.2.6 Robustness . . . 36
5.2.7 Operability . . . 36
5.2.8 Security . . . 37

6 Architecture and Design 39
6.1 System architecture . . . 39
6.1.1 Back-end architecture . . . 40
6.1.2 Front-end architecture . . . 40
6.2 Data representation . . . 41

7 Development 43
7.1 Homepage . . . 43
7.2 User login . . . 44
7.3 User Registration . . . 46
7.4 Token Validation . . . 46
7.5 Standard user features . . . 48
7.5.1 Insertion of a new Category . . . 48
7.5.2 Insertion of a new point of interest . . . 49
7.5.3 Tour Creation . . . 51
7.6 Authorized user features . . . 53
7.6.1 Validation of a category . . . 54
7.6.2 Validation of a point of interest . . . 55

8 Conclusion and future works 59

Bibliography 61

List of Tables

5.1 Describes the main actor, the precondition, and the intentions of the Registration process. In the output section are described the steps to achieve the registration. 22

5.2 This table describes the login process in all of its parts. First, define the main actor and the preconditions. Then it explains what the user wants to do. In the last section are reported all the subsequent actions performed by the user and the system with the possible outcome. 23

5.3 Describes the main actor, the precondition, and the intentions when the user wants to add a new category. In the output section are described the steps to achieve the action and the possible outcome. 24

5.4 Describes the main actor, the precondition, and the intentions when the user wants to add a new subcategory. In the output section are described the steps to achieve the action and the possible outcome. 25

5.5 Describes the main actor, the precondition, and the intentions when the user wants to add a new point of interest. In the output section are described the steps to achieve the insertion and the possible outcome. 26

5.6 In the tour generation process are described the main actor, the precondition, and the intentions of the user. In the output section are described the steps to achieve the action and the possible outcome. 27

5.7 Describes the main actor, the precondition, and the intentions when the user wants to review a category. In this scenario it can validate or invalidate a category, as well as do nothing. In the output section are described the steps to achieve the validation of a category. The process to invalidate a category is exactly the same, the only difference is the button clicked in the fifth step. The same process can be executed as well for the subcategories. 29

5.8 Describes the main actor, the precondition, and the intentions when the user wants to review an already validated category. In this scenario it can invalidate a category, as well as do nothing. In the output section are described the steps to achieve the invalidation. The same process can be executed as well for the subcategories. 30

5.9 Describes the main actor, the precondition, and the intentions when the user wants to review an already invalidated category. In this scenario it can validate a category, as well as do nothing. In the output section are described the steps to achieve the validation. The same process can be executed as well for the subcategories. 31
5.10 Describes the main actor, the precondition, and the intentions when the user wants to review a point of interest. In this scenario it can validate or invalidate the point, as well as do nothing. In the output section are described the steps to achieve the validation of a point of interest. The process of invalidation is exactly the same, the only difference is the button clicked in the fifth step. 32
5.11 Describes the main actor, the precondition, and the intentions when the user wants to review an already validated point of interest. In this scenario it can invalidate it, as well as do nothing. In the output section are described the steps to achieve the invalidation. 33
5.12 Describes the main actor, the precondition, and the intentions when the user wants to review an already invalidated point of interest. In this scenario it can validate it, as well as do nothing. In the output section are described the steps to achieve the validation. 34
5.13 Describes the usability requirement of the application. 35
5.14 Describes the maintainability requirement of the application. 35
5.15 Describes the performance requirement of the application. 35
5.16 Describes the platform compatibility requirement that also satisfies the portability one. 36
5.17 Describes the reliability requirement of the application. 36
5.18 Describes the robustness requirement of the application. 36
5.19 Describes the operability requirement of the application. 36
5.20 Describes the security requirement of the application. 37

List of Figures

2.1 Examples of detail level comparison between OpenStreetMap and Google maps in two different locations. Figure (a) and (b) is a capture of Montjuic and the Port Vell in Barcelona. Figure (c) and (d) opposes the detail level in a not touristic area in the south of Italy. 8

3.1 Agile methodology. 12
3.2 Scrum methodology. 13

4.1 Model-View-Controller design pattern. 16
4.2 Interaction between store, action, and reducer in React Redux. 17

5.1 The use case describes the access process to the system, both for authorized and standard users. The detailed description is reported in the table 7.3 and 5.2 22
5.2 The use case describes all the actions that a standard user can execute in the system. All the actions require that the user is logged in and is of type standard. The detailed description is reported in the table 5.3, 5.4, 5.5 and 5.6 28
5.3 The use case describes all the actions that an authorized user can execute in the system. All the actions require that the user is logged in and is of type authorized.The detailed description is reported in the table 5.7, 5.8, 5.9, 5.10, 5.11 and , 5.12 34

6.1 Client-server architecture of the project. It shows the connection between the components and also includes the server connection with the PostgreSQL database and the OpenTripPlanner service. Each client has its instance of the OpenStreetMap. 39
6.2 React+Redux architecture details with highlights on the components and the relations between them. 41
6.3 Schema of the project database. 42

7.1 App registration page. The user has pointed his mouse on the Plaça d'España marker and the label is shown 44
7.2 Sequence diagram for points of interest retrieval. 44

7.3 Sequence diagram of the login process. The first lifeline on the left it the one related to the client. The others are all calls within the server. 45
7.4 App login page 45
7.5 App registration page 46
7.6 Sequence diagram of the registration process. 47
7.7 Sequence diagram of the authentication process. 47
7.8 Add new Category tab of the application. The user has already selected the subcategory field, and the system has rendered the possible parent categories. 48
7.9 Sequence diagram of the category insertion process. 49
7.10 Sequence diagram of the process for get the categories already validated by the authorized user. 49
7.11 Add new point of interest tab of the application. The user has already selected the point on the map and the *Outdoor* category. The subcategory rendered by the system are the one related to the *Outdoor* category already selected. The user has also inserted the name and the description and selected the *Fountain* subcategory. 50
7.12 Sequence diagram of the point of interest insertion process. 50
7.13 Sequence diagram of subcategory request process. 51
7.14 Creates point of interest tab of the application. The user has already selected the *Business* category and the system renders the subcategory related to it. The user selects *Shopping* and *Restaurant* subcategories . . . 51
7.15 Sequence diagram of the tour creation process. 52
7.16 Creates point of interest tab of the application after the tour creation. The user navigates on the map and points on the bus line *59*. 53
7.17 Application tab for the category reviews. 54
7.18 Sequence diagram of the validation process of a category. 54
7.19 Sequence diagram of the category review process. 55
7.20 Application tab for the points of interest reviews. 56
7.21 Sequence diagram of the point validation process. 56
7.22 Sequence diagram of the points of interest review process. 57

Abstract

Nowadays, everyone travels and uses the internet to discover the place to visit. The goal of the project is to retrieve and categorize points of touristic interest. Based on these points, the application can generate a customized journey.

The points are inserted and categorized by the user to make sure that the tour generated afterward is consistent with the user's point of view. This ensures that when a user wants to create his journey, he receives points with high quality that fit well to his expectations. A set of reviewers guarantees the data quality by validating the data inserted by the previous user in terms of correct place information and categorization.

The project report the design and implementation of a client-server web application in chapter 6 and 7. It is developed using ReactJs and NodeJs, two JavaScript frameworks very popular in web application development along with other technologies described in chapter 4. State of the art, methodologies, and requirement specification are reported respectively in chapters 2, 3, and 5.

Chapter 1

Introduction

In modern society, people travel all around the world for many different reasons, but knowing new cultures and places is the leading element into a new trip choice. Nowadays, the internet is the primary source of information both for the journey organization and the place to visit. The average tourist would like to visit both well-known places, which made the city history that he is going to visit, and some little-known hidden places.

Having in mind the context above, producing the thesis idea was straightforward. The goals are mainly two: retrieve and categorize points of touristic interest, and develop a web application capable of generating a specific touristic journey based on that information.

1.1 Data retrieval and validation

People are the most significant data source ever, and they own estimable information that can be retrieved and used to create a new services. To do so, the project has to achieve a supplementary goal: to give value to the information entered and include the user in the creation of the data collection, making him aware of the value he owes. The user can decide to participate by inserting new categories or insert and categorize points of interest as well as utilize the app only for the tour creation. The majority of the people that are going to use the application are named in the project standard users, and they have access to all the functionality explained earlier.

There is another important role covered by a selected number of users that have to ensure data quality. These users in the project are called authorized users to differentiate them from the users inserting the data. The authorized user can review the categories and subcategories inserted by the others and, if these are considered valid, they are going to be available to all the users. Besides, the authorized user has to validate the point of interest inserted checking the name, the coordinates, and the description and that the categorization done by the user is coherent with the point of interest inserted.

1.2 Data categorization

For the data classification, the first step is to choose the categorizing criteria in terms of the class number, the existence of subcategories, and accordingly, the subcategories'

number. After researching different datasets, six focused categories were chosen as default for the application:

Buildings category represents construction and includes as subcategories the purpose of use that people made of it. Subcategories can be military, education, government, residential, commercial, transport, religious, and historical.

People links a point of interest with a leading figure that has designed or modified the corresponding places or was born and lived there. The subcategories are related to the person's profession as artist, philosopher, writer, architect, and historical figure.

Artistic Movement contains the most famous artistic currents of all time. The subcategories are: Modernism, Gothic, Surrealism, Cubism, Romanesque, Renaissance, Pop-Art, and Street-Art.

Museum is the most straightforward category and contains several subcategories: biographical, archaeology, art, maritime, history, science, design, military, and zoological park.

Business includes all the activities related to tourism in terms of accommodation, food, and shopping. The subcategories contained are restaurant, hotel, bar-pub, shopping, grocery, and ATM.

Outdoor is related to the city structure and its components, including the following subcategories: park, garden, square, street, fountain, beach, and viewpoint.

Although the default categories cover entirely the data needed, there is also the possibility to insert a new category by the user.

1.3 Tour creation

Over the internet, there are several journey planning that, most of the time, incorporates flight planning, accommodations, and other material that made the user less available to use it just for plan the tour. The goal of the project is to give the user the possibility to choose what he wants to visit and give back to him a journey using public transportation.

The filter selection is made using the categories and subcategories explained in section 1.2, and the user can select them with any limitation. To make the user more comfortable with a UI already known, the system shows the points that fit the selected filter and the itinerary on a map.The user can interact with it and see the points of interest name, how to reach them by bus, metro, or by walk. A written description to help the user to understand the journey is displayed on the screen. By design choices, the charged touristic tours are not inserted to give everyone the possibility to discover a city without any restriction.

1.4 Thesis Structure

The document is divided into eight chapters:

Introduction explains the motivations and the project goals, as well as the main actors and the default categories used in the application.

State of the art describes the scientific and technological progress in data collection, map creation, and trip planning. Besides, it reports the existing touristic dataset and the website for trip planning existing nowadays.

Methodologies used in the development with a particular focus on the Agile and Scrum methodologies.

Technologies adopted in the development of the project divided into front-end, back-end, and database technologies. There is an overview of each technology with a particular focus on their main aspects used in the project.

Requirements specifies the detailed definition of the project in terms of functionality to develop and non-functional requirements to be achieved. The requirements are reported using tables and use case diagrams.

Architecture and design describe the system architecture in general and the ones used in front-end and back-end, followed by the data model used. First is explained the client-server architecture, then the REST architecture of the back-end, and at the end the React+Redux front-end architecture.

Development details the implementation of the project with the technical resource used and the development environment. Moreover, it outlines the user interaction with the application and the process executed using the UML sequence diagrams.

Conclusion and future work states all the goals achieved and explain the possible subsequent upgrades.

Chapter 2

State of the art

This project affects many specialists and large areas of the IT world. The primary concern is if there are existing datasets that fit the project needs or, in the contrary case, how to collect data from the users.

Section 2.1 defines what a Dataset is and describes different existing datasets, which can be used. There is also a discussion on why they don't fit the project goal. In Section 2.2, there is an overview of the data collection in the current century. The techniques used are reported.

When the problem of data is solved, the development of the touristic tour turned out Section 2.3 analyze the existing tools for map creation with their advantage and disadvantage. Section 2.4, instead analyzes the existing software for trip planning development and the existing applications for trip planning.

2.1 Existing Data Set

Before list the dataset related to touristic information found during the researches, it is essential to give the dataset definition. Dataset is a collection of information composed of separate elements organized in block structures that can be manipulated as a unit. It exists many different types of dataset, differentiated on the data storage and structure.

Different website maintains various dataset that classify data from all kinds. One need only think about government entities as the European Union, dates September 2019, it provides 13879 up-to-date datasets [5]. Google created in 2008 a public data explorer allowing the user to search and examine large third-part datasets in the form of graphs, plots, or on map [3]. The Spanish government maintains 22877 datasets subdivided into 22 categories: public sector, environment, society and welfare, economy, demography, culture and leisure, education, employment, healthcare,treasury, tourism, transport, science and technology, town planning and infrastructures, rural environment, housing, commerce, legislation and justice, energy, industry, security, and sport [4].

The research for existing datasets that fit in the project scope was focused precisely on this last one. The selected categories were tourism, culture and leisure, and environment.

Environment mainly contains geological data, forecaster measurement, and waste collection. In this category, it can be found some information about fountains, beaches

and green areas located in a significant part of Spain but nothing about Barcelona.

Culture and leisure include data regarding cultural heritage, historic sites, monuments, museums, and libraries. For the city of Barcelona, the only one relevant was the "Archeological map of Barcelona" where the remains were classified from the prehistory to the Spanish Civil War.

Tourism covers information about hotels, statistics, and points of interest around Spain. Only in this category, was found an interesting dataset on the points of interest of Barcelona maintained by the Ajuntament. The dataset collects name, geographical position, and a series of labels that structurally categorize the data. Specifically, the recurring labels are museum, park and garden, urban space, architecture, transport, sports facility, theaters but nothing about artistic movement, or famous people linked with these places.

Given that, the only useful dataset was not exhaustive, to have the data needed for the scope, the next step was to collect the categorized data from the most potent source of information: the users.

2.2 Data collection

The frequency and magnitude of data produced using the internet increases every day enough to become the 21st-century leading resource for business. The tech giants capture significant values from the data inserted by the users and the devices. The core of their business is processing and selling data as well as use them for advertising purposes [1].

Facebook is the top scorer in terms of Billions generated from advertising. Amazon is the leader in data collection and processing, and use that data to improve its data-centric algorithm. These two are just simple examples of companies that made data their primary asset. This trend is moving fast, but the consumer knows little about all the data he is generating; Some time data are delivered consciously, but other times the user might not know they're giving up anything at all. In this last case, the user receives something in return for his data.

Information is primarily collected through screens when people use computers and smartphones. With the adoption of new data-guzzling devices, like smart speakers, facial recognition into smartphone and wearable devices, the amount and the variety of data retrieved is going to be huge. Whether data is created by real people or by computers, the biggest concern is how it is analyzed. Tech companies are also beginning to acknowledge that personal data collection needs to be regulated, but it is not enough to only protect personal data. Consumers need to own their information and receive compensation when their data are used [2].

2.3 Map creation tools

Software technologies are always under evolution, although the tech giant are the driving force of this industry, they are not the undisputed market leaders. When a developer has

to insert a map with some functionality, the first API considered is the Google Maps one.

Google Maps is a project born in 2005 that allows map visualization, but the services built on top of it is its strength. It allows to research restaurant, monument, bus station, airport and also to calculate the road routs between two or more points. Street view and traffic information are two additional services integrated with it [11]. The API renders four basic map types:

Roadmap is the default road view.

Satellite includes the Google Earth images.

Hybrid is a combination of the previous two.

Terrain displays a physical map based on terrain information.

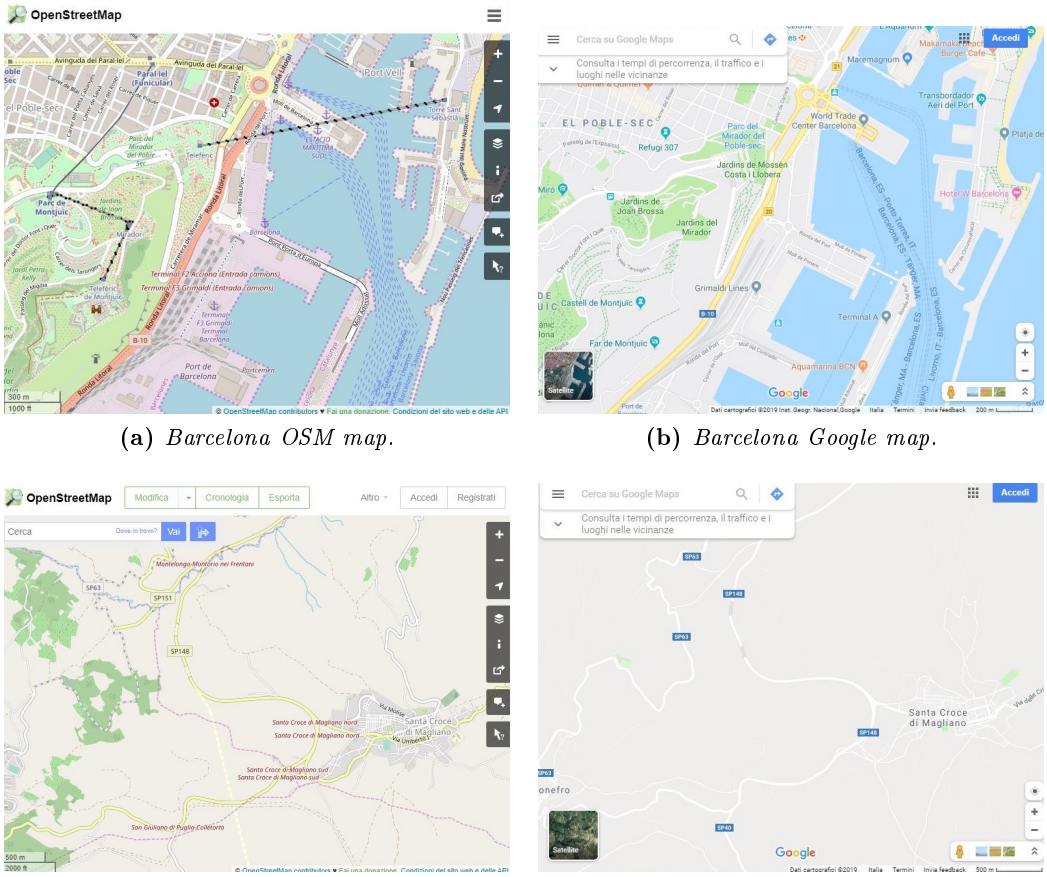
Both the element and the style of this basic map can be customized. The API supports the overlay map type which is designed to work on top of an existing map type, and it adds a new layer showing additional information to the user. There is also the possibility of creating a custom map type.

At its first release, Google Maps was a free service but over the years have been applied overly stringent rules. Nowadays, to use the API, the developer must include an API key, enable billing on all of its projects, and it has a maximum of 500 requests per second [12]. Moreover, the developer has no control over the policy applied by Google. On the market, there are various alternatives. Apple MapKit is an Apple library that has higher limits for map views and service calls, but it is still a beta version. Bing Maps: is similar to Google one, but Microsoft owns it. It has 125000 free calls per year. These three API are not for free, and they have a closed approach to data collection and distribution.

OpenStreetMap (OSM for conciseness) instead adopts an open philosophy: it is a free and editable map released under the open-content license. Their goal is to create a database which can be used by third parties. Nowadays, OSM maps are used both for humanitarian aid and scientific research. The OSM project collects data using a collaborative model similar to the Wikipedia one. It creates a free editable map of the world entirely updated by the volunteers. The updates are immediately visible to all the users, and the user and the OSM community own them. OSM also uses satellite images and government data for map production[6]. Quality and granularity, as well as equal coverage, is guaranteed by the OSM community. Google instead puts more effort and resources into areas that are most profitable to sell. The figure 2.1 shows well this concept comparing the Google and OMS maps in two different places: Barcelona and a small town in the south of Italy.

Even if the Google map in Barcelona is detailed, this map is still less updated that the OSM one. For example, in figure 2.1 (a) we can see the representation of the Telefèric de Montjuic, the Telefèrico del Puerto, and the Funicular de Montjuic that are not represented in the Google map.

OSM maps can be used both directly for their server or through an open-source Javascript library. All the features of Google Maps are supported in OpenStreetMap and in some cases, they are even better.



(a) *Barcelona OSM map.* **(b)** *Barcelona Google map.*

(c) *South of Italy OSM map.* **(d)** *South of Italy Google map.*

Figure 2.1: Examples of detail level comparison between OpenStreetMap and Google maps in two different locations. Figure (a) and (b) is a capture of Montjuic and the Port Vell in Barcelona. Figure (c) and (d) opposes the detail level in a not touristic area in the south of Italy.

2.4 Trip planning

Over the last decades, various Trip Planner systems have been developed, and researches are still ongoing. A transit trip planner is a particular engine that assists users in planning their trip inserting origin and destination, departure or arrival time, and the transport means. Most systems are based on static schedules that don't take into consideration the traffic congestion and so the possible delay. There are also real-time transit trip planners includes Vehicle Mounted Unit (VMU), on the bus. This system, based on the GPS mounted in the vehicle, provides real-time location information which is used to track the movement and, accordingly, the arrival time.

In 1999, the first multimodal itinerary planner was developed to provide the tourist with schedules of public transports. The Itinerary Planner was developed to help travelers find a suitable itinerary by generating alternative travel plans for a single Origin-Destination pair with time and mode of travel constraints. It didn't consider the real-time scenario but the unique one to start the Multimodal Trip Planner with constraints. In

2010 was introduced the Park-n-Ride mode support for multimodal trip planner. It considers parking lots near public transportation access points. During the years, different optimization algorithms have been created by inserting data caching and applying more strict search criteria. For transit route planners to work, transit schedule data must always be kept up to date. To facilitate data exchange and interoperability, in 2006 was developed a standard data formats have emerged called General Transit Feed Specification or GTFS [8]. The most advanced trip planner until today is Google Transit [7].

There are another two competitors with similar performance and free: OpenTripPlanner and Navitia. OpenTripPlanner or OTP is an open-source and multi-platform multimodal trip planner. It has a monolithic architecture that makes it easy to configure and run, and It uses the GTFS standard. Real-data informations are available as a continuous stream from the GTFS-RT. Navitia is an open-source framework provided as a hosted open service preloaded with open data from several regions. It has a modular system more suitable for high-throughput service and uses an extended transit data similar to GTFS but incompatible with it. The integration with real-time data is slow due to the protocol applied [9].

Journey planners use an in-memory representation of the transportation network and timetable to enable an efficient and rapid search and routing algorithms to search the transport network graph. The routing can use Dijkstra's algorithm when it is independent of time. Users can create a custom itinerary by using different websites that have a prebuilt database of points of interest.

Google travel put together flights, and hotels deal with information on the trip destination. In the explore section, the user gets an overview of the main activities in the selected city. Then it sees the popular things to do and some predefined Day plans based on actual travelers' visits.

Itineree helps the trip planning. It has a section called Do & See This Day, where the user can see the top ten places for TripAdvisor or the most recommended attractions. It can also search for places inserting a keyword, but for this feature, the website uses an external resource like Google and TripAdvisor. Besides, it also includes flight, hotel, and restaurant facilities.

Sygic Travel Maps shows attractions, hotels, restaurants and shops directly on the map using a proprietary database of touristic attractions. There are two other sections where the main attraction and the exiting touristic tours are presented.

TripHobo is an itinerary building website that based on seven categories, and the number and typology of travelers create a customized tour. The categories available are adventure, arts and culture, entertainment, historical, leisure, outdoors, and museum. It also includes transportation and hotel facilities.

Roadtrippers has two main functionality: explore places or plan trip. The explore place section allows the user to select some feature as accommodation, attraction and culture, food and drink, outdoors and recreation, point of interest, entertainment and nightlife and some others more. It uses a map visualization where the user selects one or more points, and the system renders the road trip between them.

Inspirock is like TripHobo, but it is limited to selected countries, and it doesn't allow the user to edit the trip after its creation. The filter mechanism enables the user to select the degree of point of interest popularity and one or more categories between culture, romantic, museum, outdoors, beaches, shopping, relaxing, wildlife, and historical sites. The daily trips proposed are based on existing touristic tours.

Ixigo uses the same style of Google travel. When the user inserts the city, it gives him some tips about the best period of the year to visit the city with the weather forecast and which fabric type is more suitable for the selected period.

There are also countless other web sites that allows the user to insert the itinerary manually and to keep track of his trip without give any suggestion on the journey creation.

Chapter 3

Methodologies

Software development methodology is a series of processes like design, develop and test phases, used to define the software structure and to codify the communication. The latter is regulated by a set of norms stating how and when clients and developers have to communicate.

In the literature, there are many different ways to organize software development. From the ancestor *Waterfall* model to the newest *Scrum* methodologies, it is impossible to choose the best because each one has its benefits. The only two mandatory elements are discipline and organization in the development. Nowadays, most of the software companies follow the *Agile* methodologies, specifically its subset *Scrum*. This project follows the *Scrum* methodology.

3.1 Agile

The Agile methodology was developed as a response to the inflexibility of the Waterfall methodology. This approach helps the developers to respond to the unpredictability of software production. The underlying logic is to produce the software as quickly as possible, and then make adjustments and bug fixing in subsequent improvement cycles. The Manifesto for Agile Software Development explains the Agile philosophy [33]:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over processes and tools.*
> *Working software over comprehensive documentation.*
> *Customer collaboration over contract negotiation.*
> *Responding to change over following a plan.*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

Twelve principles are the basis of the Manifesto [34]:

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

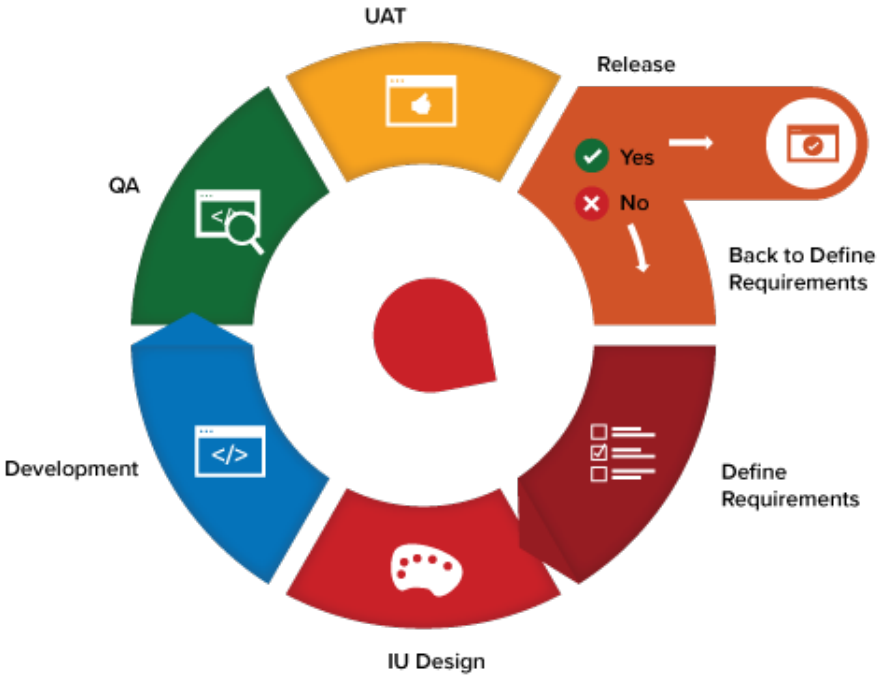


Figure 3.1: Agile methodology.

Based on these principles, the Agile workflow can be divided into six phases repeated over time. Brainstorm is the first one, where the team produces the system requirements

based on customer and stakeholder feedback, product, and sprint backlog. In the design phase, high-level UML diagrams are created to show the functionalities overview and to establish the timeline. The third phase is the sprint development, followed by the testing phase divided into internal, external, and Quality Assurance tests. The delivery phase integrates and deliveries the working system into production. The last phase is composed of feedbacks from customers and stakeholders that are then inserted in the requirement of the next iteration.

3.2 Scrum

Scrum is a flexible and straightforward subset of Agile, it is not a methodology but a framework used to address complex and adaptative problems while delivering products of the highest possible value. Scrum is based on five principles:

1. Courage to do the right thing and work on severe problems.
2. Focus on the work and to reach the Sprint goals.
3. Personal commitment to achieve the Scrum Team goal.
4. Respect for the other members in terms of capabilities and independence.
5. Openness to work and the challenges associated with it.

Scrum is strongly collaborative, the developers, the Scrum Master, and the product owner belong to the Scrum Team indeed. The product owner represents both the business and the customers. He prioritizes the tasks in the Product Backlog an he is available to answer questions and clarify the requirement. The developers are self-organized and cross-functional. In other terms, they decide what is achievable in one sprint, and they have the skills to do all the work. The Scrum Master helps everyone to understand the scrum theory and make sure that everyone has everything he needs.

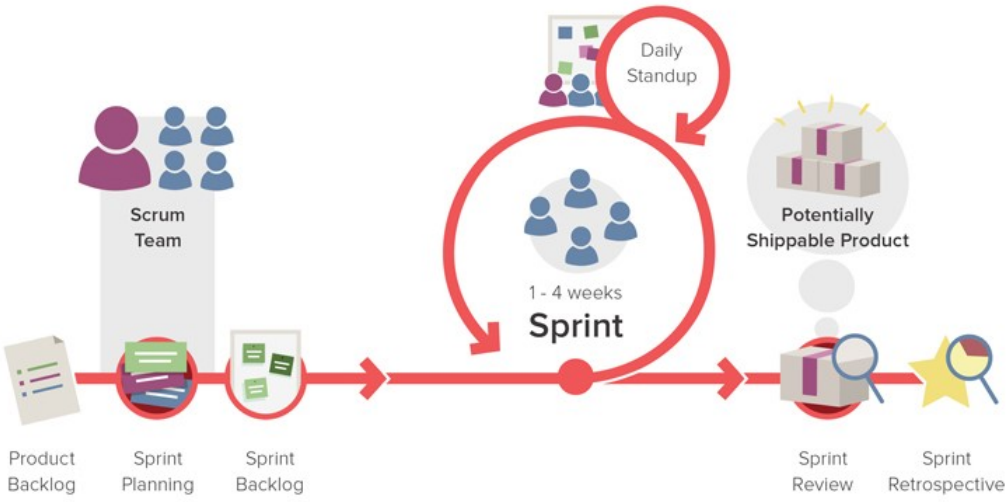


Figure 3.2: Scrum methodology.

Scrum has three artifacts to maximize transparency in communication:

Product Backlog is an ordered list of requirements prioritized by the Product Owner.

Sprint Backlog contains a set of items of the Product Backlog specially selected for a Sprint. It is the team to-do list.

Increment is the result of the Sprint added to the previous Sprint completed. It defines the project progress.

Scrum contains five events designed to maximize the benefits of face-to-face communication and maintain transparency. Sprint Planning is the first one where the team decides what requirement insert into a specific Sprint.The main event is the Sprint: a work loop where the developing takes place. It has a fixed length between 1 to 4 weeks. During the Sprint the team participates in a Daily Scrum where the work for the next 24 hours is planned. At the end of each Sprint, there is a Sprint Review to inspect the product, set the Increments, and adapt the Product Backlog. The last event is the Sprint Retrospective, where the team can adjust the goals for the next Sprint. Figure 3.2 summarizes the Scrum methodology.

For the development of this project, the Sprint length had two different lengths: monthly Sprint for the first period, followed by weekly Sprint at the end of the development.

Chapter 4

Technologies

Before discussing about the requirements and the developing process, the technologies used must be introduced. The ones employed in the frontend will be presented first, followed by the backend technologies and, in the end, the database.

4.1 React

React is a javascript library purposely designed for the UI creation. The goal after its development is to have an intuitive language capable of building dynamic and increasing complex UIs. It is a declarative and stateless language made of immutable and reusable elements nested together. React's components can receive inputs through props or can store data in the state. Every time the state change, it updates the UI parts which depend on it. React supports several frameworks and external plugin to manage routing, API interaction, and more sophisticated features [20] [21].

Due to is nature, React well fits into the Model-View-Controller software design pattern. This paradigm decouples the presentation logic and the data representation, enhancing code reusability, and the usage of any backend language without restrictions.

Although there are several libraries for web application developing, the choice fell on React because:

- Unlike JQuery it excludes a direct operation on the DOM. Instead, it provides a virtual DOM to the developers to guarantee the best performance
- It is less complicated compared to AjaxJS
- It applies the Separation of Concerns. React isolate state and code based on their scope of use to make the components more reusable at the expense of the inherent separation of the MVC pattern.
- React's component uses the JSX syntax allowing the developer to include HTML in React.

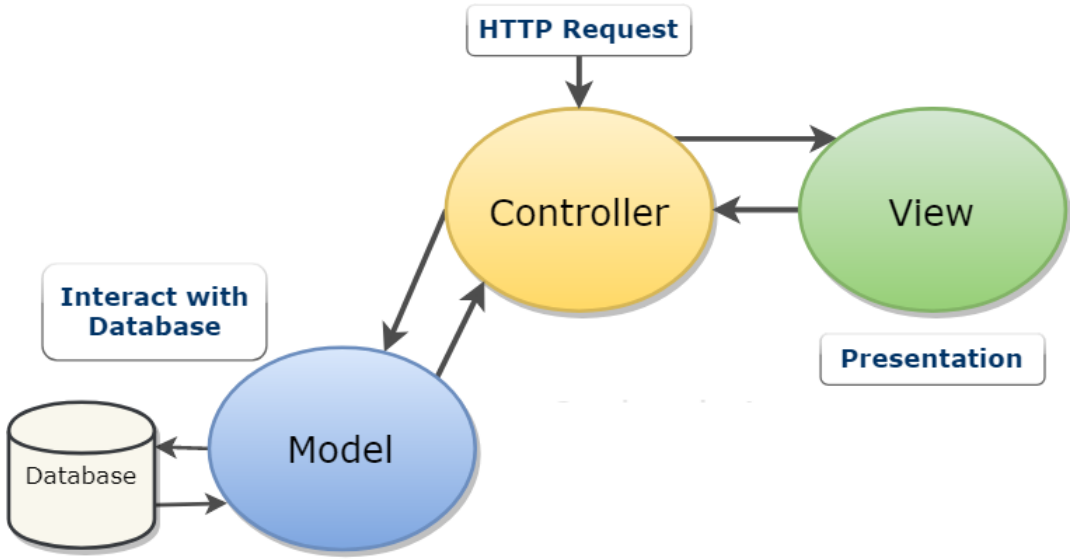


Figure 4.1: Model-View-Controller design pattern.

In just under a year after its development, React has been adopted by major Internet companies in their core products as Facebook, Instagram, Netflix, and AirBnB [22]. Reacts needs two fundamentals packages for routing and private routes: React Router and Redux.

4.2 React Redux

In a multi-component application, the state information should be available for all the components in the application tree. Various state management libraries synchronize state and UI components like Redux. It is a predictable state container that uses a central data store to manage the state of the application. The store act as a source of truth on which components can rely on state management [24].

Three basic concepts are Redux's cornerstone: store, action, and reducers. The whole app state is stored in an object tree inside a single store. The store makes it easier to debug and inspect the application. The only way to change the state tree is to emit an action, an object describing what happened. All changes are centralized and follow a strict order avoiding race conditions. Reducers are functions that specify how the actions transform the state tree. It takes the previous state and generates a new one without modifying the previous one [25].

This patter can be challenging for small applications, but it scales to large and complex apps very well. The interaction between the three components is shown in the following picture.

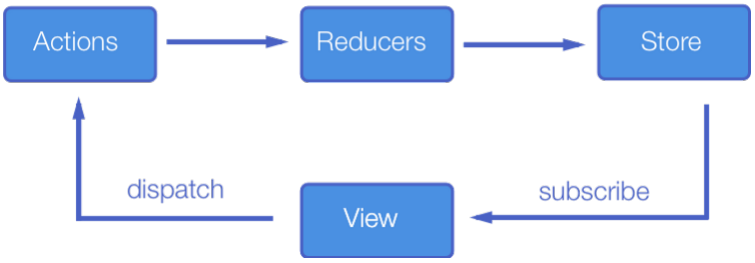


Figure 4.2: Interaction between store, action, and reducer in React Redux.

4.3 React Router

React Router is an API for React application that allows the creation of a single-page web application with dynamic navigation. The difference between static and dynamic routing stands where the routes are declared. Static routing is declared when the app is initialized outside of a running application. Dynamic routing instead, takes place as the app is rendering. This library is built with React and embraces its philosophy of component-based architecture.

There are three main components: router, route matching, and navigation. The router component is the core of the application; it must be the outer component given to the DOM in the render method. It creates a history object and enables these components to interact with it to manage the routing. The router component is in its turn divided into two parts: Route and Switch. The Route is mandatory, and it contains the path and a component to render when a location matches the Route's path. The Switch can be omitted, but it is useful to group the different Routes and iterate among them to do the matching. Navigation links are Link components to create links with styling attributes [23].

React Router also enables the user to utilize the browser button, maintaining the correct application view.

4.4 Node.js

Node.js is a cross-platform and open-source Javascript runtime environment built on Chrome's V8 engine. It allows the usage of javascript both on the client and server-side letting to implement the Javascript everywhere paradigm, unifying the Web application development under a single programming language. This choice makes development faster and bug-fixing more effective.

A Node.js application runs on a single process without creating a new thread for each request going against the model of the classical web server. It accesses the operative system resources through the event-driven model. This model is based on a simple concept: every time there is a change an event is fired. Node.js uses a non-blocking paradigm, making the blocking behavior the exception; in this way, it avoids the waste of memory and CPU

cycles, in favor of increased performance. This characteristic makes it perfect for the creation of a data-driven application where the I/O is put to the test.

Node.js is very flexible in terms of development architecture because it has few dependencies and loosely guidelines. Node.js came along with a standard package management useful for download and installed several plug-in and dependencies. Even if npm is the largest javascript repository, its quality is still under verification. The lack of a control mechanism on these modules leads to a careful choice of them from the developer [27] [28]. Node.js is a low-level platform, and there are thousands of libraries build on it to make the development easier as Express.

4.5 Express

Express is the de facto standard server framework for Node.js. Express is an unopinionated framework; it has few restrictions on how to achieve goals and which component to use. It provides common tasks not directly provided by Node as routing and middleware integration.

Routing determines how the server responds to a request to a particular URI and HTTP request method. Route path can be string, string pattern, or regular expressions. The server listens for requests on a specific route and method when it detects a match execute the callback function associated with it. Route handlers behave like a middleware with the only exception of the next() function, that can be used to skip the remaining route callbacks. The callback function has to send a response back to the client to avoid letting the request hanging. The express.Router class creates modular route handlers that is a complete middleware [26].

Middleware functions can access the request-response object and the next middleware. They can execute code, change the request-response object, end the function exiting the request-response cycle and call the next middleware function in the stack. Express can use different types of middleware:

- Application-level is bind to the app object through the app.use() function. It takes three arguments request, response, and next.
- Router-level is bound to the router object and works in the same way as the previous one.
- Error-handling is bind to the app object but takes four arguments: error and the usual ones.
- Built-in includes the JSON parser for incoming requests.
- Third-party is installed using npm and is applied using the app.use() function.

In the specific case of the thesis, express interact with three leading middleware: cors, passport, and sequelize. Cors implement the cross-origin resource sharing a security policy that allows the server to specify who can access it and which HTTP request use. Passport is an authentication middleware designed to manage both traditional and through an

OAuth provider as Facebook, Twitter, and Google. Sequellize is an Object-Relational-Mapper for Node.js. Its basic idea is to abstract the complexity of interfacing with the database writing SQL queries using the object-oriented paradigm instead of pure SQL [29].

4.6 PostgrSQL

PostgreSQL is an open-source object-relational database management system. It uses a client/server model where the two processes cooperate and communicate through a TCP/IP network connection. The server process, called Postgres, manages the database and performs actions on it on behalf of the client. The client application can have different natures: web server, text-oriented application, or database management tool. PostgreSQL supports standard SQL and adds more complex features that simplify management and prevent data loss and corruption [29]. It introduces [31] [32]:

- Views give a name to a query to access it like an ordinary table.
- Foreign keys maintain the data's referential integrity and improve database quality.
- Transactions ensure the ACID properties: Atomicity, Consistency, Isolation, and Durability. Atomicity guarantees that a transaction is considered as a single unit. It fails or succeeds completely. Consistency prevents database corruption bringing the database from a correct state to another, but doesn't guarantee the transaction correction. Isolation ensures that the concurrent execution of transactions produces the same state as their sequential execution. Durability ensures that a transaction remains committed even in case of a system failure.
- The multi-version concurrency control manage the concurrency. Each transaction makes changes on a snapshot of the database to avoid read lock and ensure the ACID properties.
- PostgreSQL provides by default several data types, but also it offers the possibility to create new types.

Chapter 5

Requirement

The following chapter contains the requirements, both functional and non-functional, that ensure the system quality. In the development, they were the milestones to monitor the progress during the months. Since there wasn't a definite client, the requirements are the result of market research and the opinion of the project advisor.

Before any further specification, it is mandatory to explain the difference between standard and authorized users. A standard user can access the website, navigate through pages, insert categories and places, and generate a tour based on its preference. The authorized user has specific priviledge given by the system administrator. It can review the data inserted by the standard user and validate them.

5.1 Functional Requirement

The functional requirement specifies the detailed requirements which the system shall meet. The different scenarios are presented using the Use Case diagram. Moreover, each use case is described specifing:

- Main actor
- Precondition
- Input
- Output

5.1.1 Use Case UC01: Register

RF01	User registration
Main Actor	User, both standard and authorized
Precondition	The user didn't have an open session, and he is not registered in the system.

Continued from previous page

Input	The user wants to access the system
Output	1. The user opens the application. 2. The system shows the main page. 3. The user selects *"Sign Up"* button. 4. The system shows the sign-up form. 5. The user inserts username, email and the password twice. 6. The user clicks on the *"Sign Up"* button 7. The system validates all the fields. 7.1 If the username or the email already exists, the system shows an alert. The user has to change them to sign-up. 7.2 If the password, the email, or the username doesn't respect the criteria, the system shows an alert. 7.3 If the two passwords inserted, don't match the system shows an alert. 8. The system redirects the user to his main page.

Continues from previous page

Table 5.1: Describes the main actor, the precondition, and the intentions of the Registration process. In the output section are described the steps to achieve the registration.

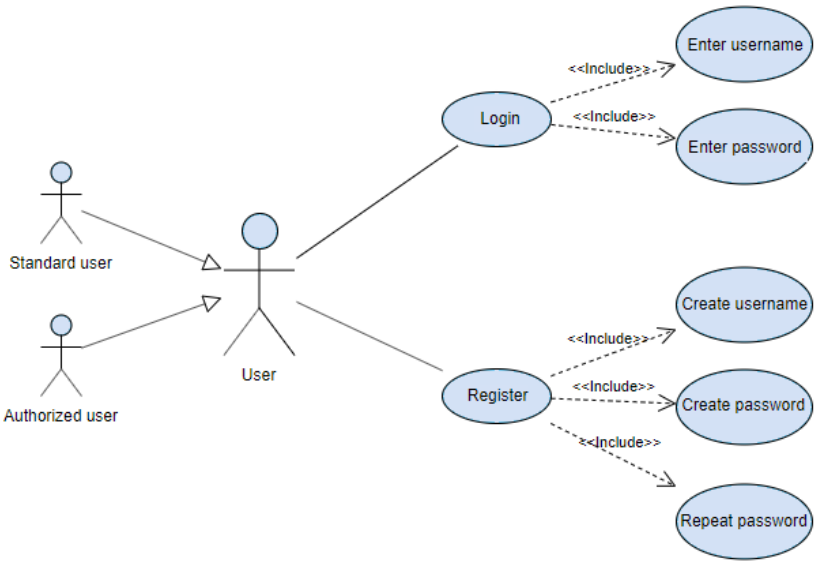


Figure 5.1: The use case describes the access process to the system, both for authorized and standard users. The detailed description is reported in the table 7.3 and 5.2

5.1.2 Use Case UC02: Login

RF02	User Login
Main Actor	User, both standard and authorized
Precondition	The user didn't have an open session, and he is already registered in the system.
Input	The user wants to access the system
Output	1. The user opens the application. 2. The system shows the main page. 3. The user selects *"sign-in"* button. 4. The system shows the sign-in form. 5. The user inserts username and password. 6. The user clicks on the emph"Sign In" button. 7. The system checks if the user exists and validates the two fields. 7.1 If the username or the password is wrong, the system shows an alert. The user has to change them to sign-in. 7.2 If the user with that username doesn't exists, the system shows an alert. 8. The system redirects the user to his main page.

Table 5.2: This table describes the login process in all of its parts. First, define the main actor and the preconditions. Then it explains what the user wants to do. In the last section are reported all the subsequent actions performed by the user and the system with the possible outcome.

5.1.3 Use Case UC03: Add new category

RF03	User add new category
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new category.
Output	1. The user selects, on the navigation bar on the left, the *"Add new Categories"* section. 2. The system shows the related page. 3. The user selects the form and inserts the category name. 4. The user selects the *"Category"* radio button. 5. The user clicks on the *"Insert"* button. 6. The system checks that this new category isn't already in the database. 6.1 If the category already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 6.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 5.3: Describes the main actor, the precondition, and the intentions when the user wants to add a new category. In the output section are described the steps to achieve the action and the possible outcome.

5.1.4 Use Case UC04: Add new subcategory

RF04	User add new subcategory
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new subcategory.
Output	1. The user selects, on the navigation bar on the left, the *"Add new Categories"* section. 2. The system shows the related page. 3. The user selects the *"Subcategory"* radio button. 4. The system shows the validate category already inserted in the database. 5. The user selects the form and inserts the subcategory name. 6. The user selects one or more parent categories between the ones rendered on the page. 7. The user clicks on the *"Insert"* button. 8. The system checks that this new subcategory isn't already in the database. 8.1 If the subcategory already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 8.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 5.4: Describes the main actor, the precondition, and the intentions when the user wants to add a new subcategory. In the output section are described the steps to achieve the action and the possible outcome.

5.1.5 Use Case UC05: Add new point of interest

RF05	User add new point of interest
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to insert a new point of interest.
Output	1. The user selects, on the navigation bar on the left, the *"Add new points"* section. 2. The system shows the related page. 3. The user selects one point on the map. 4. The system renders a marker on the selected point, it blocks the map and it shows a form. 5. The user inserts the place name and its description. It selects one or more categories and subcategories. 6. The user clicks on the *"Insert"* button. 7. The system checks that this new point of interest isn't already in the database. 7.1 If the place name already exists, the system shows an alert. The user can insert a new one or continue to navigate in the system. 7.2 If the insertion succeeds, the system shows an alert to notify the user. The user can repeat this action as many times as he desires.

Table 5.5: Describes the main actor, the precondition, and the intentions when the user wants to add a new point of interest. In the output section are described the steps to achieve the insertion and the possible outcome.

5.1.6 Use Case UC06: Generate a touristic tour

RF06	User generate a tour
Main Actor	Standard User
Precondition	The user has an open session, and he is a standard user.
Input	The user wants to generate a tour.
Output	1. The user selects, on the navigation bar on the left, the *"Create points of interest"* section. 2. The system shows the related page. 3. The user selects one or more categories. 4. The system renders the subcategories related to the category selected in the previous step. 5. The user optionally selects one or more subcategories. 6. The user clicks on the *"Generate Points"* button. 7. The system checks that there is in the database at least one point linked to those categories and subcategories. 8. In case of success: 8.1 The system renders a map with the points found. 8.2 It also shows on the map a path between the points with public transportation and a textual description of the journey. 9. In case of failure: 9.1 The system renders a map with no points. The user can repeat this action as many times as he desires.

Table 5.6: In the tour generation process are described the main actor, the precondition, and the intentions of the user. In the output section are described the steps to achieve the action and the possible outcome.

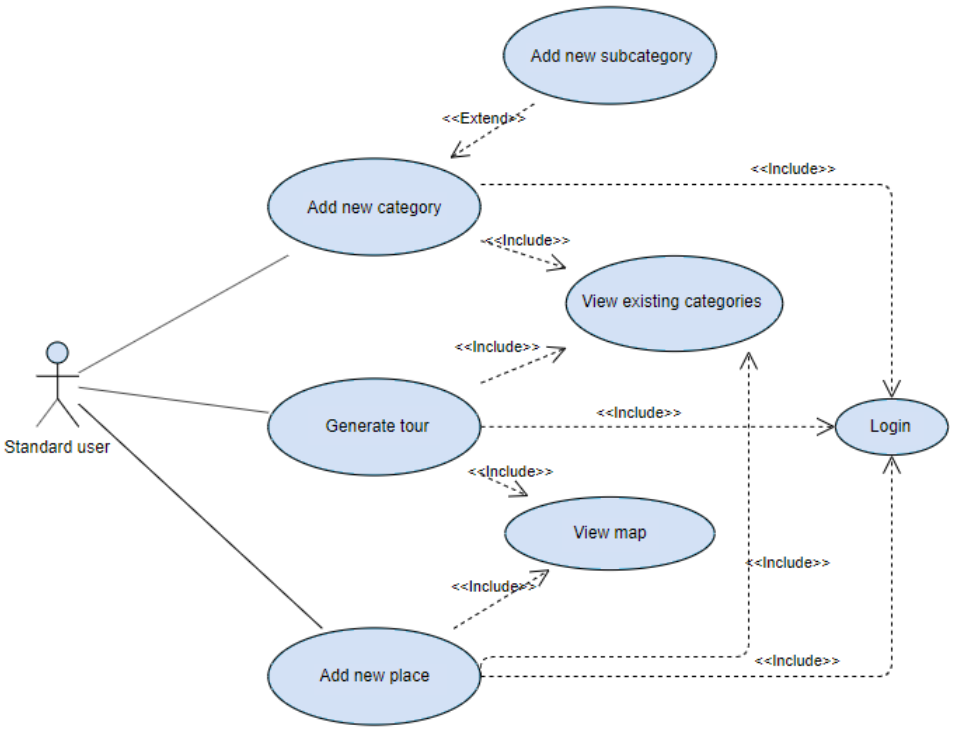


Figure 5.2: The use case describes all the actions that a standard user can execute in the system. All the actions require that the user is logged in and is of type standard. The detailed description is reported in the table 5.3, 5.4, 5.5 and 5.6

5.1.7 Use Case UC07: Review and validation of categories and subcategories

RF07	User reviews and validates categories
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user. The categories and subcategories are still not reviewed by no one.
Input	The user wants to validate a category inserted by standard users.
Output	1. The user selects, on the navigation bar on the left, the *"Review category"* section. 2. The system shows the related page. 3. The user sees two different columns, one for the categories and one for the subcategories. Each row has two buttons: *"Validate"* and *"Invalidate"*. The subcategory column also shows the reference to its parent category. 4. The user can choose to validate or invalidate each category and subcategory. 5. The user clicks on the *"Validate"* button associated to the category that he wants to validate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are categories to review.

Table 5.7: Describes the main actor, the precondition, and the intentions when the user wants to review a category. In this scenario it can validate or invalidate a category, as well as do nothing. In the output section are described the steps to achieve the validation of a category. The process to invalidate a category is exactly the same, the only difference is the button clicked in the fifth step. The same process can be executed as well for the subcategories.

5.1.8 Use Case UC08: Review valid categories and subcategories

RF08	User reviews valid categories and subcategories
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user. The categories and subcategories have already been invalidated.
Input	The user wants to invalidate a validate category.
Output	1. The user selects, on the navigation bar on the left, the *"Validated categories"* section. 2. The system shows the related page. 3. The user sees two different columns, one for the categories and one for the subcategories already validated. Each row has the *"Invalidate"* button. The subcategory column also shows the reference to its parent category. 4. The user can choose to invalidate or not each category and subcategory. 5. The user clicks on the *"Invalidate"* button associated to the category that he wants to invalidate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are validated categories.

Table 5.8: Describes the main actor, the precondition, and the intentions when the user wants to review an already validated category. In this scenario it can invalidate a category, as well as do nothing. In the output section are described the steps to achieve the invalidation. The same process can be executed as well for the subcategories.

5.1.9 Use Case UC09: Review invalid categories and subcategories

RF09	User reviews invalid categories and subcategories
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user.The categories and subcategories have already been validated.
Input	The user wants to validate an invalid category.
Output	1. The user selects, on the navigation bar on the left, the *"Invalidated categories"* section. 2. The system shows the related page. 3. The user sees two different columns, one for the categories and one for the subcategories already invalidated. Each row has the *"Validate"* button. The subcategory column also shows the reference to its parent category. 4. The user can choose to validate or not each category and subcategory. 5. The user clicks on the *"Validate"* button associated to the category that he wants to validate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are invalidated categories.

Table 5.9: Describes the main actor, the precondition, and the intentions when the user wants to review an already invalidated category. In this scenario it can validate a category, as well as do nothing. In the output section are described the steps to achieve the validation. The same process can be executed as well for the subcategories.

5.1.10 Use Case UC10: Review and validate points of interest

RF10	User reviews and validates points of interest
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user.
Input	The user wants to validate a point of interest inserted by standard users.The points of interest are still not reviewed by no one.
Output	1. The user selects, on the navigation bar on the left, the *"Review points"* section. 2. The system shows the related page. 3. The user sees a table with six columns.The columns contain respectively: name, latitude, longitude, categories, subcategories, and description of the point of interest. Each row has two buttons: *"Validate"* and *"Invalidate"*. 4. The user can choose to validate or invalidate each point of interest. 5. The user clicks on the *"Validate"* button associated to the point of interest that he wants to validate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are points of interest to review.

Table 5.10: Describes the main actor, the precondition, and the intentions when the user wants to review a point of interest. In this scenario it can validate or invalidate the point, as well as do nothing. In the output section are described the steps to achieve the validation of a point of interest. The process of invalidation is exactly the same, the only difference is the button clicked in the fifth step.

5.1.11 Use Case UC11: Review valid points of interest

RF11	User reviews valid points of interest
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user.The points of interest have already been validated.
Input	The user wants to invalidate a valid point of interest.
Output	1. The user selects, on the navigation bar on the left, the *"Validated points"* section. 2. The system shows the related page. 3. The user sees a table with six columns.The columns contain respectively: name, latitude, longitude, categories, subcategories, and description of the points of interest already validated. Each row has the *"Invalidate"* button. 4. The user can choose to invalidate or not each point of interest. 5. The user clicks on the *"Invalidate"* button associated to the point of interest that he wants to invalidate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are validated points of interest.

Table 5.11: Describes the main actor, the precondition, and the intentions when the user wants to review an already validated point of interest. In this scenario it can invalidate it, as well as do nothing. In the output section are described the steps to achieve the invalidation.

5.1.12 Use Case UC12: Review invalid points of interest

RF12	User reviews invalid points of interest
Main Actor	Authorized User
Precondition	The user has an open session, and he is a authorized user. The points of interest have already been invalidated.
Input	The user wants to validate an invalid point of interest.
Output	1. The user selects, on the navigation bar on the left, the *"Invalidated points"* section. 2. The system shows the related page. 3. The user sees a table with six columns.The columns contain respectively: name, latitude, longitude, categories, subcategories, and description of the points of interest already invalidated. Each row has the *"Validate"* button. 4. The user can choose to invalidate or not each point of interest. 5. The user clicks on the *"Validate"* button associated to the point of interest that he wants to validate. 6. The system stores the information on the database. The user can repeat this action as many times as he desires until there are invalidated points of interest.

Table 5.12: Describes the main actor, the precondition, and the intentions when the user wants to review an already invalidated point of interest. In this scenario it can validate it, as well as do nothing. In the output section are described the steps to achieve the validation.

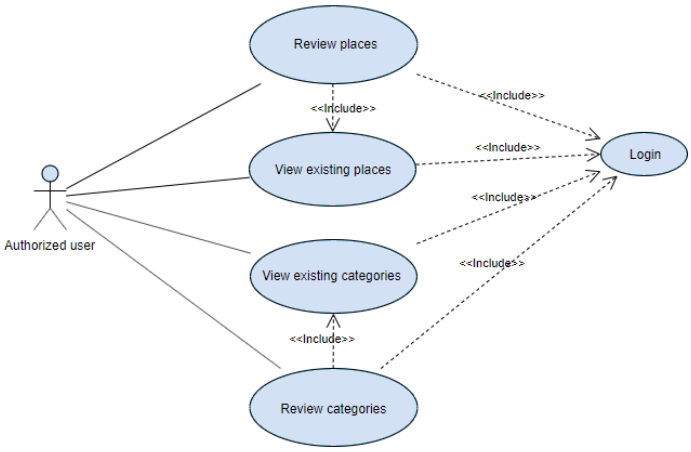


Figure 5.3: The use case describes all the actions that an authorized user can execute in the system. All the actions require that the user is logged in and is of type authorized.The detailed description is reported in the table 5.7, 5.8, 5.9, 5.10, 5.11 and , 5.12

5.2 Non Functional Requirement

Non-functional requirements state how the functional requirements need to be achieved. In the following section, they are presented using tables that specify the identification number, the type, and the description.

5.2.1 Usability

RNF01	Usability
Description	The UI design is attractive for the user. The combination of shapes and colors made the interaction more natural and intuitive. Moreover, UI follows the logic of similar applications making the interaction more comfortable and error-free. The users don't need the training to use the application.

Table 5.13: Describes the usability requirement of the application.

5.2.2 Maintainability

RNF02	Maintainability
Description	The developer has to easy correct defects or their causes, prevent unexpected working conditions, and repair or replace components without having to rewrite the whole code. Besides, the useful system life must be maximized along with its efficiency, reliability, and safety.

Table 5.14: Describes the maintainability requirement of the application.

5.2.3 Performance

RNF03	Performace
Description	The system must have a short response time both in the UI rendering and in the interaction with the backend. It should be acting in the same way with different users' load. The database size and the length of the path for the tour to calculate don't have to affect significantly the response time that should be under 5 seconds.

Table 5.15: Describes the performance requirement of the application.

5.2.4 Platform compatibility

RNF04	Platform compatibility
Description	The software execution isn't affected by the platform where it runs. The system must work well on every web browser. This non-functional requirement also includes the portability one, in terms of abstraction between the application logic and the system interface to reduce the development cost.

Table 5.16: Describes the platform compatibility requirement that also satisfies the portability one.

5.2.5 Reliability

RNF05	Reliability
Description	The system can work under a defined condition for a specified period. It reflects design perfection and resistance to failure of a component or the whole application in terms of probability of success. It is strictly related to availability and the users' behaviors. The possible failure of the system can be related to human interaction, maintenance-induced failure, and software failure.

Table 5.17: Describes the reliability requirement of the application.

5.2.6 Robustness

RNF06	Robustness
Description	The system has to deal with errors during the execution and erroneous input. It has to notify the user and continue without generate misleading behaviors.

Table 5.18: Describes the robustness requirement of the application.

5.2.7 Operability

RNF07	Operability
Description	All the system parts, frameworks, databases, and UI, have to work together to accomplish the common task. The system works without needing application restart or any other non-automated interventions. It is closely related to reliability and maintainability.

Table 5.19: Describes the operability requirement of the application.

5.2.8 Security

RNF08	Security
Description	The data integrity is mandatory for the security of all the information stored in the database. The system has to implement access restrictions and separation of jurisdiction between the users. Further, it has to implement a mechanism to prevent informatic attacks.

Table 5.20: Describes the security requirement of the application.

Chapter 6

Architecture and Design

This chapter explains in detail the system architecture and the design with a focus on the frameworks used. Section 6.1 describes first the general architecture of the project. Then it reports the detailed architecture used for the back-end and the front-end development. Section 6.2 includes the database schema and the data representation used in the project.

6.1 System architecture

The system architecture is composed of sub-systems that work together to implement the overall system. This project is based on a client-server architecture, shown in the figure 6.1, where clients can use any desktop computer with a modern browser. The server can be unique or replicated, and it communicates with a PostgreSQL database and the OTP service. The client-server architecture has several advantages that fit the project scope.

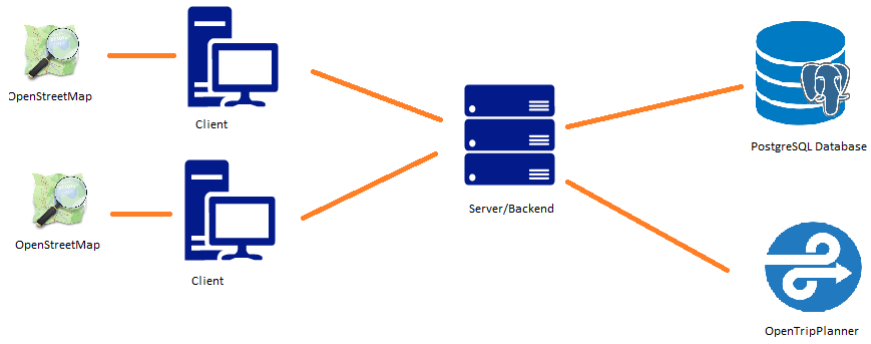


Figure 6.1: Client-server architecture of the project. It shows the connection between the components and also includes the server connection with the PostgreSQL database and the OpenTripPlanner service. Each client has its instance of the OpenStreetMap.

First of all, the system ensures the separation between the presentation and business logic. Business logic includes all the processes invisible to the user that are the core of the application; the presentation logic consists of the UI representation instead. This separation ensures uncomplicated maintainability because any changes in the back-end

don't affect the presentation layer, and the changes are centralized. Moreover, the client can access the system without any specific configuration. The front-end development can be uncoupled because the communication between client and server should agree only on the data representation and the communication protocol. It is a cheap architecture in term of human cost, the security and the maintainability are centralized, so fewer support staff is required [15].

6.1.1 Back-end architecture

The REST architectural style is the foundation for the back-end development. The representation state transfer defines a set of constraints for web services creation that provides interoperability between systems over the internet [16] [17]. A RESTful API is based on the following guideline:

Separation of concern between client and server enables the components to grow autonomously. It improves client portability across different platforms, and it increases the server scalability by streamlining its components.

Responses cacheability can be enabled by clients or servers to improve performance or disabled to avoid to receive stale data.

Stateless server protocol binds the session state to the client. In each request sent, the client must include all the information required for the communication.

Layered system used to improve scalability and security. This feature is liked to the separation of concern explained before. The client is not aware of who he is communicating with because additional layers can be added to the server to improve load balance, security policies, or caching mechanism. Furthermore, the server can call other web services to introduce additional functionality.

Uniform interfaces to simplify and decouple the architecture. Unique URI identifies individual resources that are conceptually separated by the data representation returned. The message returned includes enough metadata information to be manipulated or deleted. A standard interface like HTTP is used for communication.

RESTful systems aim for fast performance, reliability, and scalability, as explained before. Following this architectural style, components can be managed and updated without affecting the system even while it is running.

6.1.2 Front-end architecture

The implicit connection between React and the Model-View-Controller design pattern described in chapter 4 leads to think of embracing the MVC design style. The problem with MVC is the bidirectional communication that made the code less maintainable and challenging to debug. Facebook, which developed React, presented a new architecture called Flux for building React web application with unidirectional data flow [10]. Four elements are the basis of Flux: Action is an object with property and data, Store contains

the application state and logic, View listen to store changes and re-render accordingly, and dispatcher is a process implementing actions and callback functions [?]. Based on this architecture, the thesis is developed using Redux, an architecture based on Flux that differs from it for the absence of the dispatcher and the concept of data immutability.

In the project, React is the View layer, and Redux is the Store.The overall architecture with components and relationships is represented in figure 6.2.

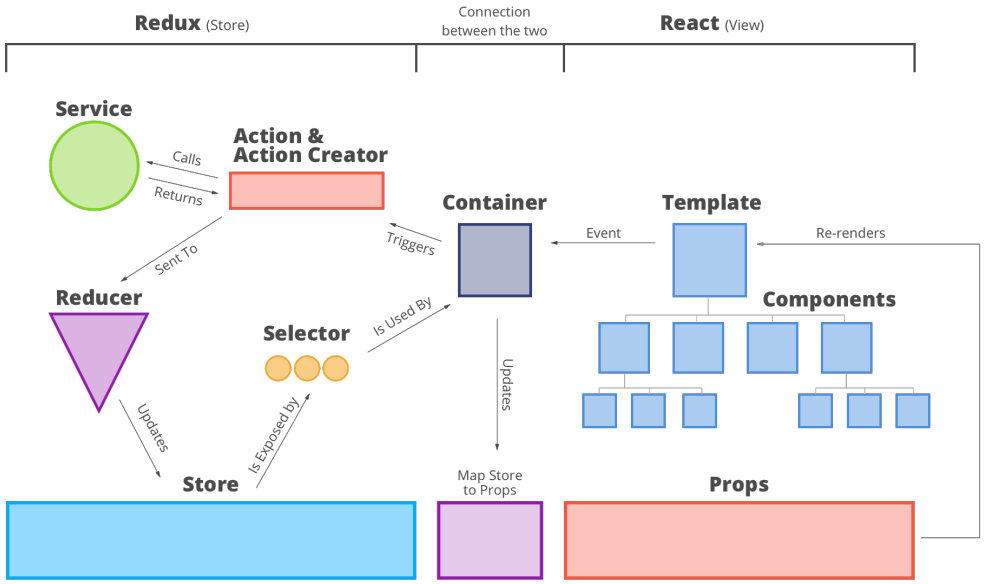


Figure 6.2: React+Redux architecture details with highlights on the components and the relations between them.

The container is an abstraction layer that allows Redux and React to be decoupled, therefore, to change and grow autonomously. It translates the changing in the Redux state to React props, and when in React, an event happens, it triggers the corresponding action in Redux. A Component is a piece of code that renders and uses a set of props passed by its parent that can be a component or a template. A template is merely an additional abstraction level. The Action is an object containing the type of action, and the state changed because of the action. It communicates with the server to execute the action required by the React event, and it sent the result to the Reducer. The Reducer is the only component allowed to change the state creating a new one. The selector is an abstraction level between the Store and the Container [19].

6.2 Data representation

The data used by the application is stored in a relational database. An ORM technique can be used to simplify the development and increase maintainability. The Object Relational Mapping technique allows the developer to manipulate and query the data in the database using an object-oriented approach. It implements a data layer that creates a "virtual object database" and acts as a translator between the object-oriented language and the

database, reducing the need for SQL language. The ORM translates the logical object representation in a suitable form that can be stored in the database preserving the object properties [14].

This project uses Sequelize, an ORM promise-based for Postgres, MySQL, and other relational databases [13]. The first step is to create a Sequelize instance to connect to the database. Then the developer has to define for each database table a model in Sequelize. By default, the *createdAt* and *updateAt* field are inserted in the model to track the modification done on the table. The application will interact with the database directly with this model. After the model creation, it must be deployed in the database using the Sequelize migration.

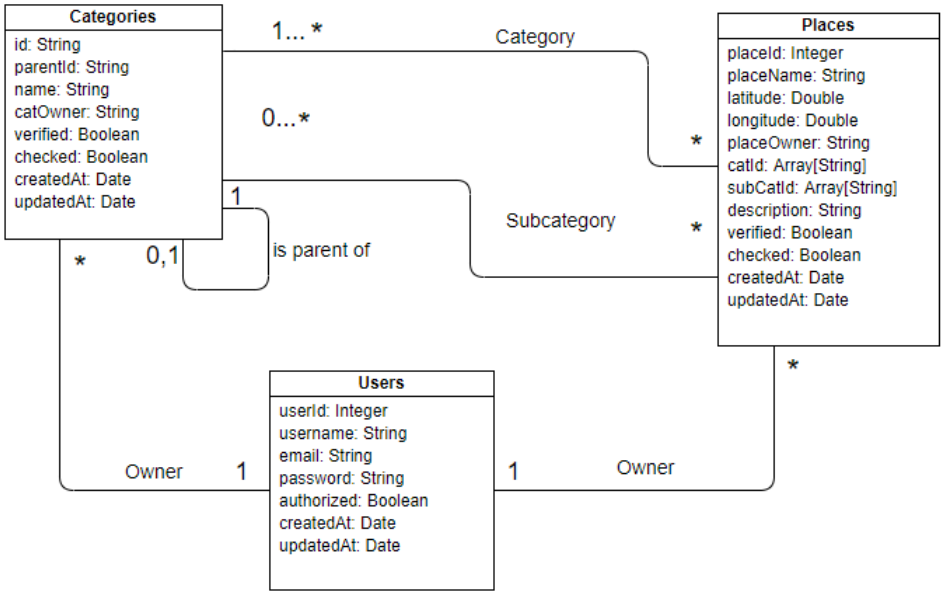


Figure 6.3: Schema of the project database.

Figure 6.3 shows the database model created using the migration. The Categories and Places tables are linked with the Users one through a foreign key on the user *email*. The Places table also has two more foreign keys referring to the categories and subcategories stored in the Categories table. The Categories table includes both categories and subcategories that are defined using a foreign key applied between the *id* and *parentId* attributes.

Chapter 7

Development

This chapter details the implementation of the project, explaining the functionality and the most significant element employed. It includes UML diagrams and snapshots of the application both of code and UI. Before entering the implementation details, let's introduce the development environment.

The development environment is a set of processes and tools used to develop source code and debug it. WebStorm is the JavaScript IDE created by JetBrains that covers all the modern JavaScript languages as React and Node.js. It provides to the developer intelligent code completion and on-the-fly error detection. For these features and the built-in integration with Git and GitHub, it was chosen for the development of the entire project. Git is a revision control system and a source code management that was used in the project to keep track of its history. It was used together with a GitHub account where the project information was stored. GitHub is a web-based hosting service that uses Git.

7.1 Homepage

The first view presented to the user is a map showing the point of interest, inserted and already validated, as markers on the map. This is a public page where both registered and non-registered users can access it. In the top-right corner, there are the Sign-in and Sign-up button to allow access to the restricted part of the system. In the development, Leaflet was used to show the OSM map in the application. Leaflet is an open-source JavaScript library designed to ensure simplicity, performance, and usability. It is highly customizable and supports several plug-ins to extends its features. When a user navigates to the homepage, the browser asks him to allow the localization. If it is enabled, the map will zoom in on his position, and a localization marker appears. Otherwise, nothing happens, but the system could not work correctly in the next phase of tour creation. The user can navigate on the map, and when he goes on a marker, the system shows the place name as shown in figure 7.1.

The system to render the point of interest on the map, interact with the server. The only API call that can be executed by an unregistered user is the *getAll* one showed in figure 7.2. The system asks for the points of interest already validated by the authorized user. Since this page is public, the server returns only the name and the geographic

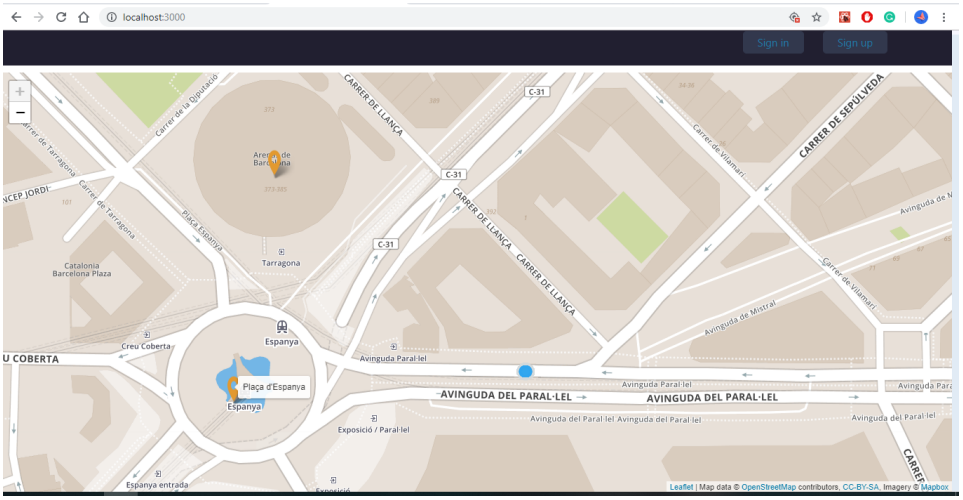


Figure 7.1: App registration page. The user has pointed his mouse on the Plaça d'España marker and the label is shown

coordinates of the point without any information about who inserted it.

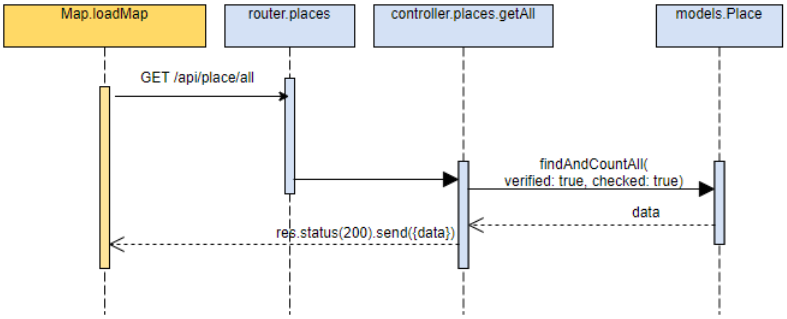


Figure 7.2: Sequence diagram for points of interest retrieval.

7.2 User login

As said before, the user to access all the application features has to log in and start a new session. To handle the authentication, a specific service was developed using JSON Web Tokens. JWT is a compact and self-contained solution for transmitting information between parties that can be verified and trusted because it is digitally signed. JWT is composed of three different parts: header, payload, and signature. The header contains the specification for the type of token and the algorithm used. User-related information and token metadata are stored in the payload. The signature is used to verify the token, and it is created trough the encoded header, the encoded payload, and the secret known by the token issuer. Any changes in the token modifies its signature and invalidate it. The JWT is used as an access token for the protected features of the system.

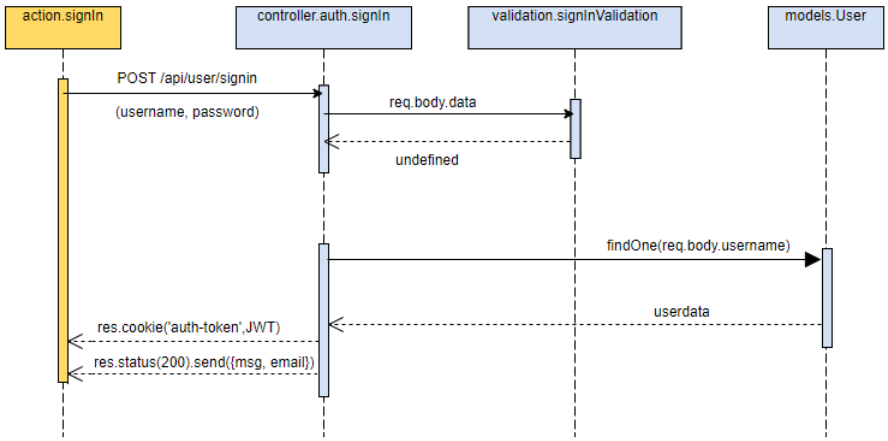


Figure 7.3: Sequence diagram of the login process. The first lifeline on the left it the one related to the client. The others are all calls within the server.

Every time the user enters the application, he receives an access token. For design choices, the client stores the access token and sent it to the server through cookies. Figure 7.3 shows all the calls executed by the system in the login process. This scenario represents the login of an already registered user that inserts all the correct parameters, explained afterward. The *signInValidation* function validates the data inserted by the user and returns the error if it founds some illegal data. It the case represented, the user has inserted the correct ones, so the validation returns an undefined error that means success. The JTW token and the user email are stored by Redux in the client-side.

Figure 7.4: App login page

The user interface showed in figure 7.4 follows the style of the most popular software to make the user comfortable with a familiar scenario. The user as to insert the username

and the password, then he selects the *SignIn* button. In case he miswrote any of the two fields, leaves one of them empty, or inserts illegal code in them, he is notified with an alert. In case the login succeeds, the main page connected with his state of normal or authorized user is automatically rendered.

7.3 User Registration

During the registration process, the user must follow some specifications for username and password creation. The username must be at least 6 characters long and contain only alphanumeric characters and underscores. The password requires at least 8 characters, including an uppercase letter, a digit, and a special symbol. It has to be repeated another time for safety. The form used for the registration process also uses a structure similar to the other popular website. In case any field showed in figure 7.5 is empty or doesn't respect the constraint explained before, the user is notified.

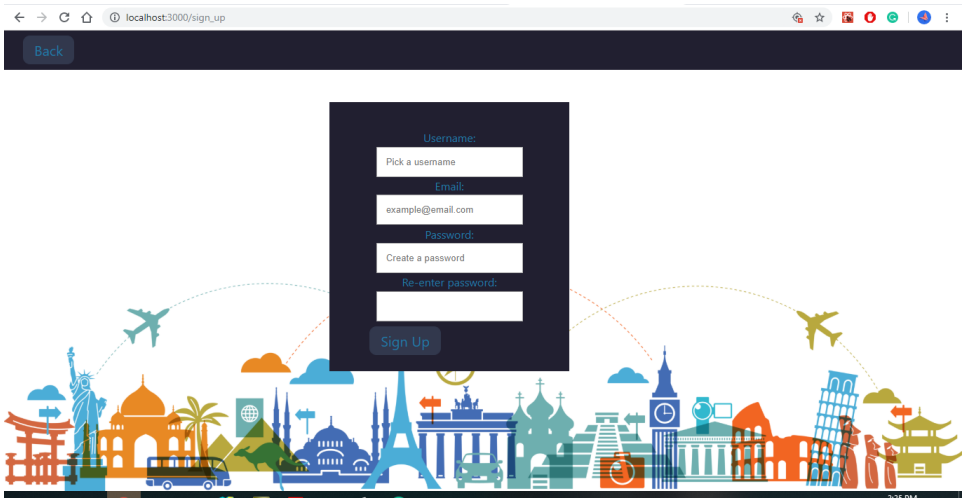


Figure 7.5: App registration page

When the user hits the *SignUp* button, the client sends a POST request to the server. The input validation process is called and permit to proceed with the new user insertion. This process showed in figure 7.6 is practically the same described in figure 7.3. The only differences are the URI of the request, and the parameter send with it. If the user inserted a username already taken or the email already exists, the user is notified to allow him to change them and start the process again. When the process is completed successfully, the user is directly redirected to his homepage and receives with the response the JWT and his email.

7.4 Token Validation

The system has to validate the token every time the user requests a private route. The system specifies three functions to verify the different access privileges. The *verify.normal*

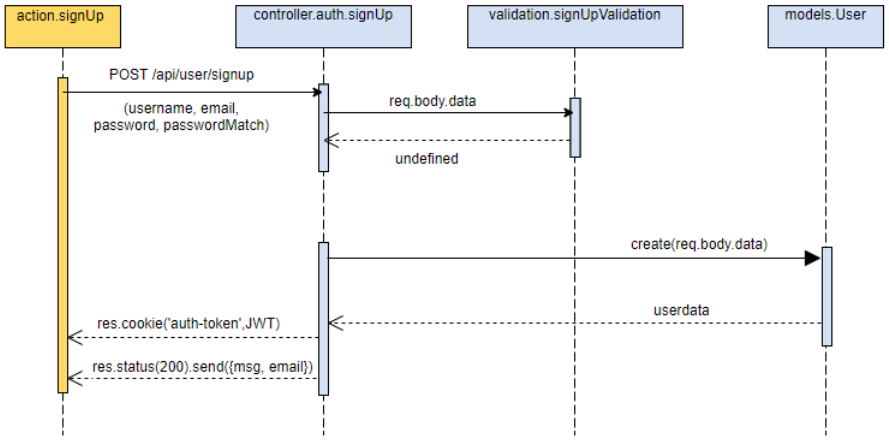


Figure 7.6: Sequence diagram of the registration process.

is used to check if the user that wants to access the resource is registered and is a standard user. The *verify.authorized* one examine if the user has the authorized privilege. There is also a general function called *verify.general*, used to check only if a user is registered in the system because both types of users can access some resources in different contexts.

Figure 7.7 reports the sequence diagram for the token validation. First, the server has to retrieve the JWT from the request cookies. In case the function found it, the token is validated with a proprietary function of JWT that decrypts the payload and takes the *userId* previously stored there. The server then queries the database to find if the user corresponding with that id is registered and notifies the user accordingly. If the user tries to access unauthorized resources, the system automatically redirect him to the login page.

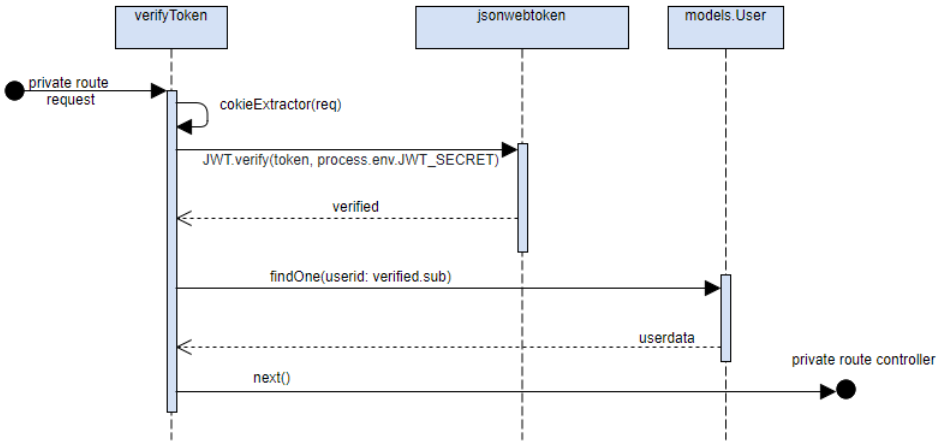


Figure 7.7: Sequence diagram of the authentication process.

7.5 Standard user features

The standard user is the one that can add valuable information on the application and the one with more sophisticated features. On his homepage can choose three different activities explained in the following subsections, or he can logout using the *SignOut* button in the header showed in figure 7.8, 7.11, 7.14 and 7.16 using a straightforward process. The *SignOut* button triggers an action on the Redux reducer and sets the global state to the initial state before the login.

7.5.1 Insertion of a new Category

The first feature of the software is the categorization of point of interest. By default, the system already has six categories: Building, Museum, Business, People, Outdoor, and Artistic movement. Each of them has several subcategories that can be liked to one or more categories. Even if the categories cover many touristic aspects, the possibility for the user to add a new one was inserted.

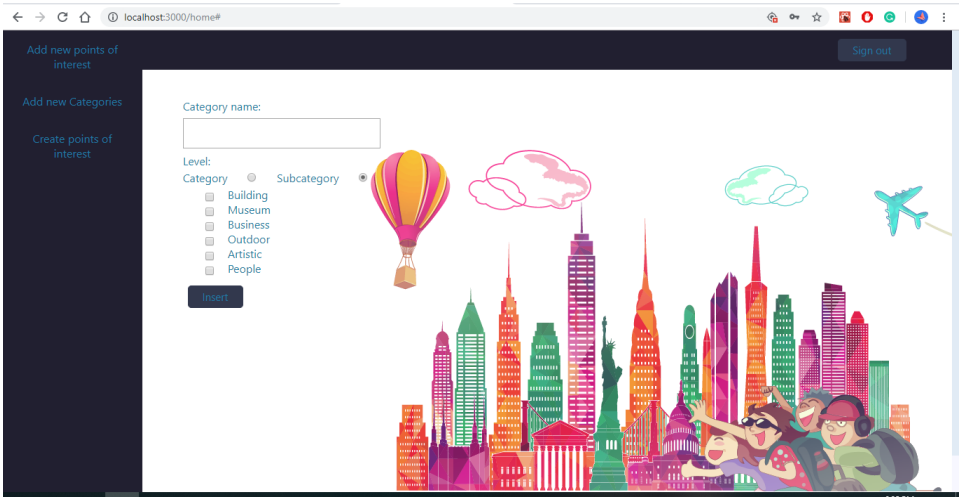


Figure 7.8: Add new Category tab of the application. The user has already selected the subcategory field, and the system has rendered the possible parent categories.

The user to insert the category selects the *Add new category* tab in the left navigation bar showed in figure 7.8. He can choose between the insertion of a category or a subcategory. In the category case, he inserts the name, selects the Category button, and clicks on *Insert*. The sequence diagram, figure 7.9, exposes the API calls executed. First, the system checks if the user is a standard one and is registered, and it validates the input. Then it queries the database three times to ensure that there isn't any category with that name, to count the existing categories to give a sequential id to the categories, and if any error occurs, it inserts the new category. In case of success or error, the user is notified.

For the subcategory insertion, the user has to select the subcategory button; the system requests the main categories through a GET request following the steps reported in figure 7.10. The user then selects one or more categories and continues as explained for the

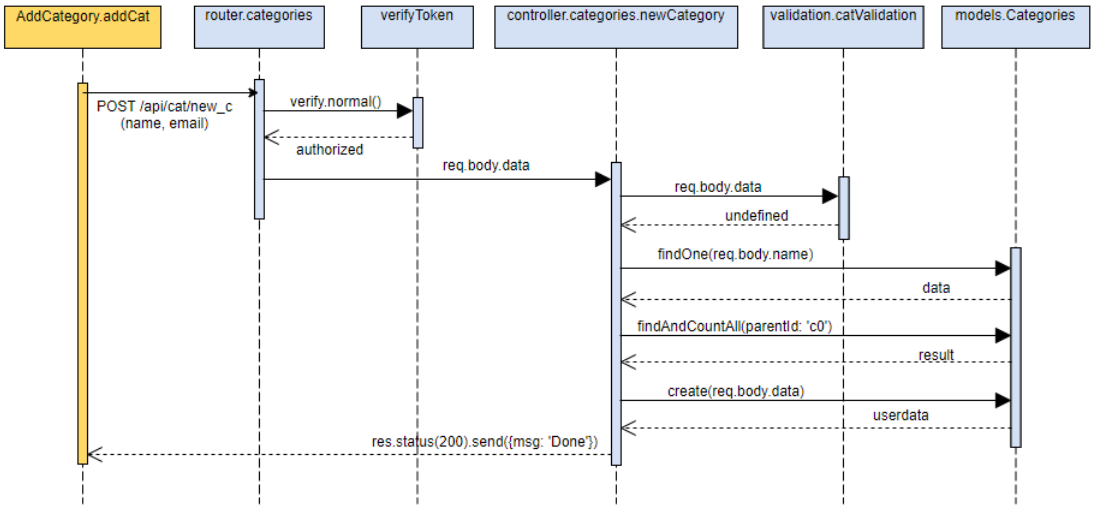


Figure 7.9: Sequence diagram of the category insertion process.

category insertion. To get the categories, the system checks first if the user is registered in the system, and afterward, it retrieves from the database the category already validated. The process to insert a subcategory is the same; the only difference is the request URI, and the parameters passed with it.

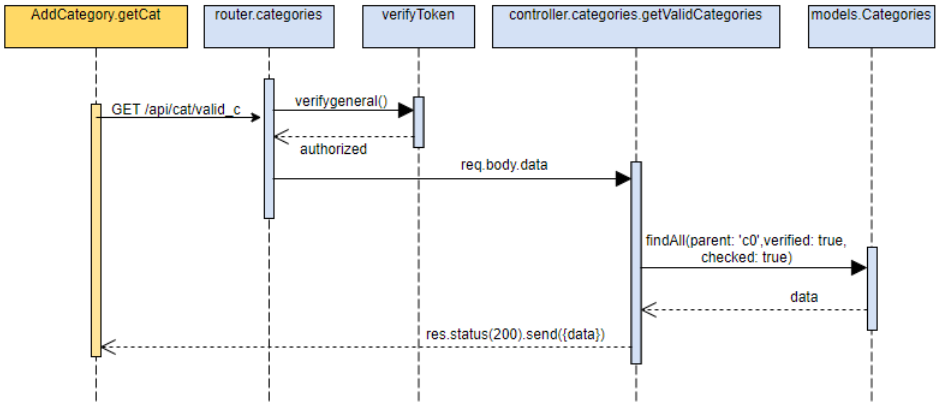


Figure 7.10: Sequence diagram of the process for get the categories already validated by the authorized user.

7.5.2 Insertion of a new point of interest

One of the central purposes of the application is giving value to user knowledge and ensure the most trustworthy and free journey. The idea is to insert points of interest that cover not only the major sightseeing but also hidden or less know places. The user can

do it within the website, selecting the tab Add new point of interest. If he has enabled the localization, it sees a map zoomed on his position; otherwise, he sees the general map, as explained in section 7.1. In this case, the points of interest already validated are represented with orange markers; the new point selected by the user is a blue marker.

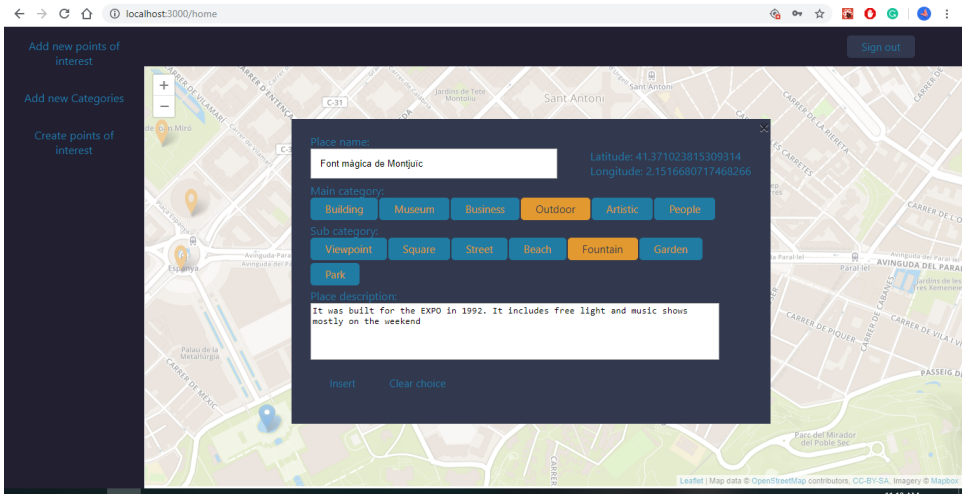


Figure 7.11: Add new point of interest tab of the application. The user has already selected the point on the map and the *Outdoor* category. The subcategory rendered by the system are the one related to the *Outdoor* category already selected. The user has also inserted the name and the description and selected the *Fountain* subcategory.

When the user selects a point on the map, the system blocks the map and renders a popup form, as in figure 7.11, where the user can insert name, categories, subcategories, and a description. Between them, the name and at least one category must be inserted. The system shows in the form latitude and longitude where the user clicked and store them with the user email and all the other information in the database, as reported in figure 7.12.

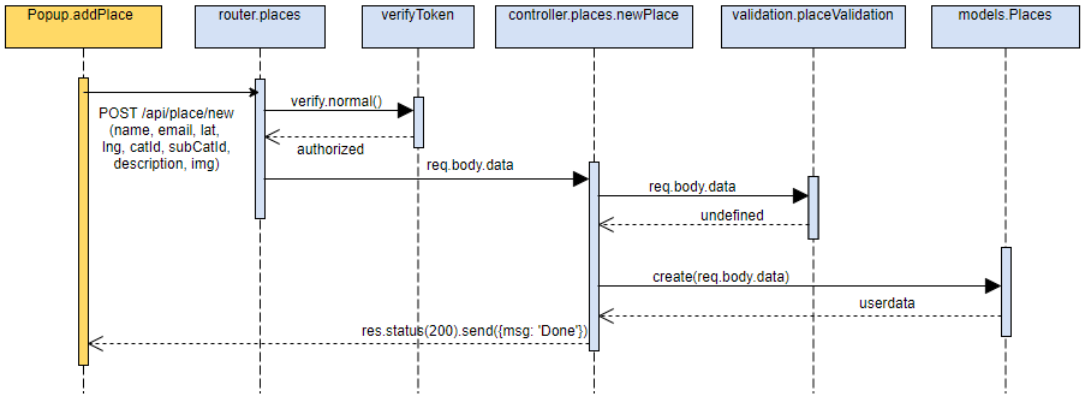


Figure 7.12: Sequence diagram of the point of interest insertion process.

A POST request is sent to the server that validates token and input and creates an instance on the database. The flow of the process is similar to others already explained in the previous sections. But this one requires two other API calls. The first one is included in the map creation, as explained in the 7.1 section, for the retrieval of point of interest already validated. The second API call, described in figure 7.13, can be repeated every time the user selects a category button to retrieve the subcategories linked to the selected category. There is a POST request to the server that verifies the token and asks the database for the subcategories validated that have as parent the one stored in the request body.

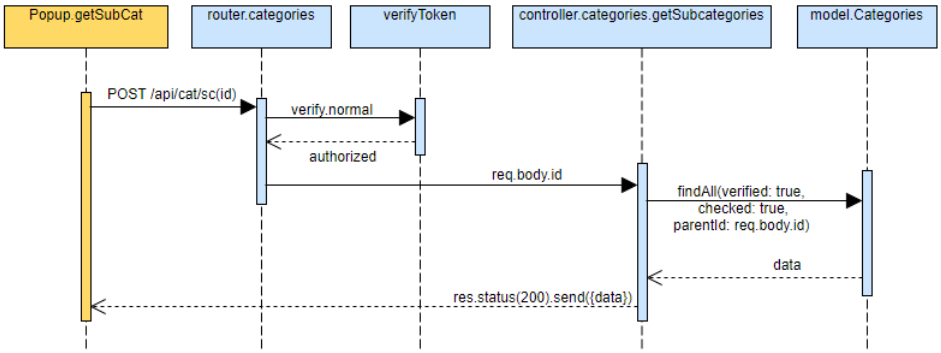


Figure 7.13: Sequence diagram of subcategory request process.

7.5.3 Tour Creation

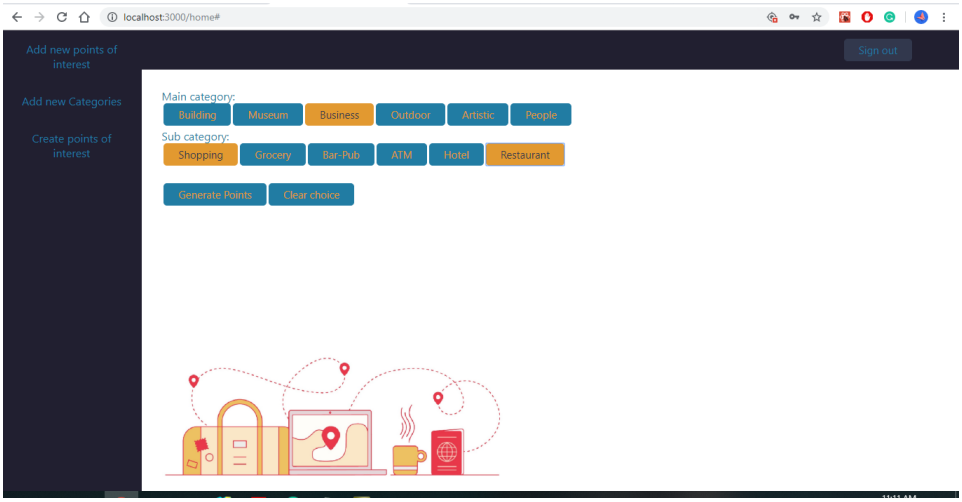


Figure 7.14: Creates point of interest tab of the application. The user has already selected the *Business* category and the system renders the subcategory related to it. The user selects *Shopping* and *Restaurant* subcategories

The last, but not least feature available for a non-privileged user is the customized tour creation. The user navigates to the *Create points of interest* tab on his homepage. The system renders the first screen, reported in figure1, where the user sees and chooses the validated categories. To do so, it executes the GET call explained in section 7.5.1 figure 7.10. For each category selected by the user, the system executes the same call to the API explained in section 7.5.2 figure 7.13. The user can choose many subcategories as well. To help the user in the visualization of the categories chosen, the system renders them with a different color. The user can select and deselect each category manually or undo all by selecting the *Clear choice* button.

When the user is satisfied with the filter selection, it clicks on the *Generate points* button, and the system executes the API call reported in figure 7.15. As usual, the server checks if the user has the right privileges to execute that call. It queries the database to found the places that were categorized with the tag selected by the user. The result of the database aren't returned directly to the user because it is a waste of bandwidth to execute another call to the server for the tour generation. The server automatically calls the *sortPoint* function passing the data retrieved from the database and the current position of the user. In this scenario, the user must allow the localization to use the service.

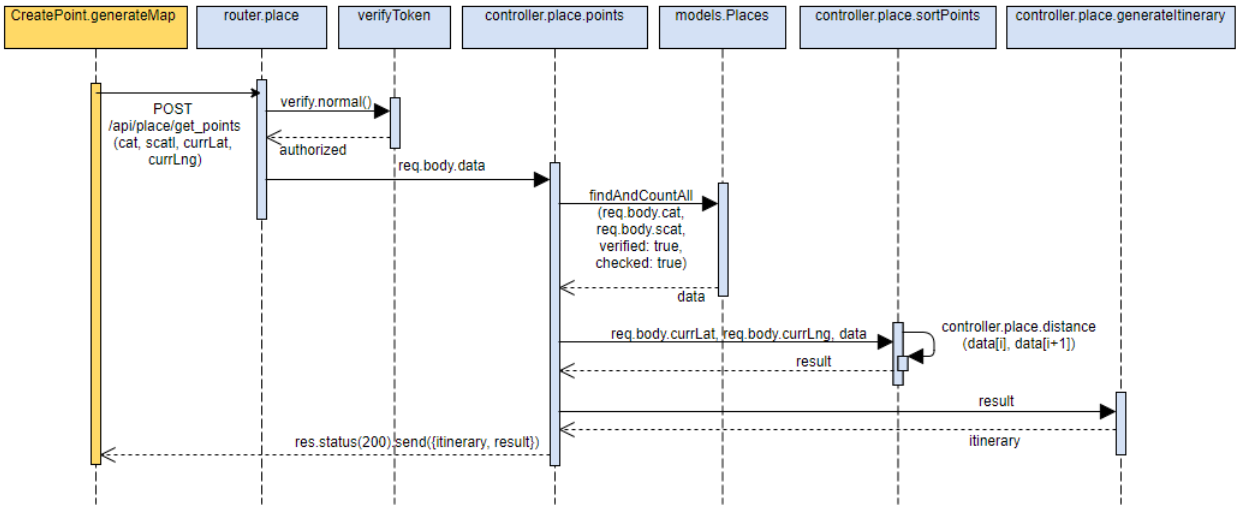


Figure 7.15: Sequence diagram of the tour creation process.

The sort function uses a brute force approach because during the design was assumed that point categorization was focused and careful to give the best performance possible. This function computes in the first iteration, the distance between the current user position and all the points returned by the database. The nearest point goes in the array as the second element, and the process iterate again. This time the system computes the distance between the second element of the array and the remaining point, putting the new solution in the third position. The function ends when there aren't any more points on which iterate.

The OTP service can create the itinerary only between pairs, so it can't receive the array as a whole. Therefore, the system passes the sorted array to the *generateItinerary* function that iterates on the array, takes a point and its following one, and sends them to the OTP. To work the OTP also needs the time and date of departure, which are set in our system by default with the current date and time, the maximum walking distance between the different bus transit, set to 500 meters, and the transport modes to consider, in this case, walk and public transportation. The function creates a new array with the itinerary linked together and returns it finally to the user.

If the research of points didn't produce any result, the user is notified. Otherwise, the system renders a new screen with a map and a description box, as shown in figure 7.16. In the map are present only the point of interest received back from the server and not all the points as in sections 7.1 and 7.5.2. The user can navigate the map and see the point name as in section 7.1. The system renders with two different colors, the bus path and the path where he has to walk, and when he passed over a path with the mouse, the line name is shown. To make the user comfortable was inserted a description box where the journey is described in a discursive way. If the user is not satisfied with the tour, he can select the *Create points of interest* tab on his homepage again.

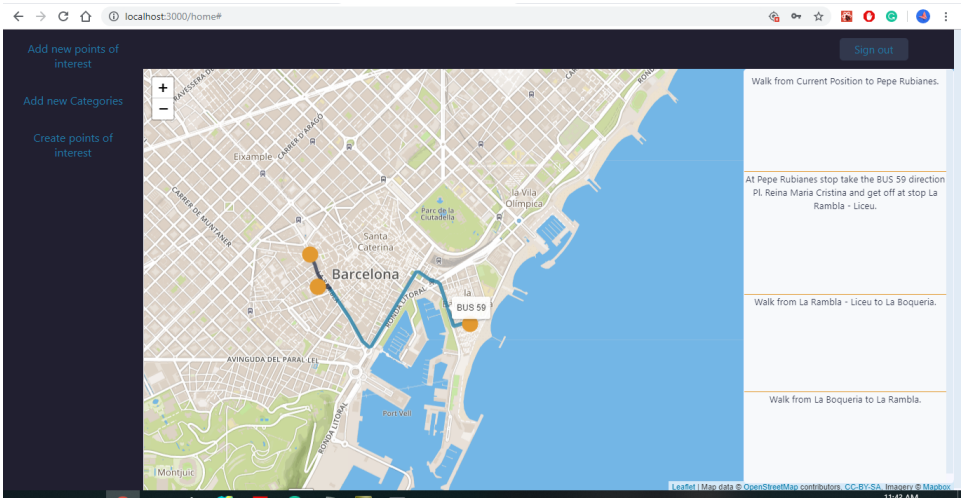


Figure 7.16: Creates point of interest tab of the application after the tour creation. The user navigates on the map and points on the bus line *59*.

7.6 Authorized user features

The administrator of the system gives privileges to the authorized user. It has to ensure the data quality in the application employing the feature described in the following subsections. He can logout as well as the standard user using the *SignOut* button displayed in figures 7.17 and 7.20 that follow the process described in section 7.5.

7.6.1 Validation of a category

The authorized user has different privileges compared to the standard one. It ensures the quality of the system data; for this reason, his job needs a different environment and a different homepage in the application. The system administration gives his privileges at the registration time. The first task and the one rendered by default on the homepage is the category review showed in figure 7.17.

Figure 7.17: Application tab for the category reviews.

The main page is divided in two columns, the left one for the categories validation where the user can see the category name and the two buttons for validate or invalidate the corresponding category. The right column contains the subcategories following the same schema, but we must also have the parent category for each of them in order to complete the validation rightly.The user to validate a category clicks on its *Validate* button and fires the process described in figure 7.18.

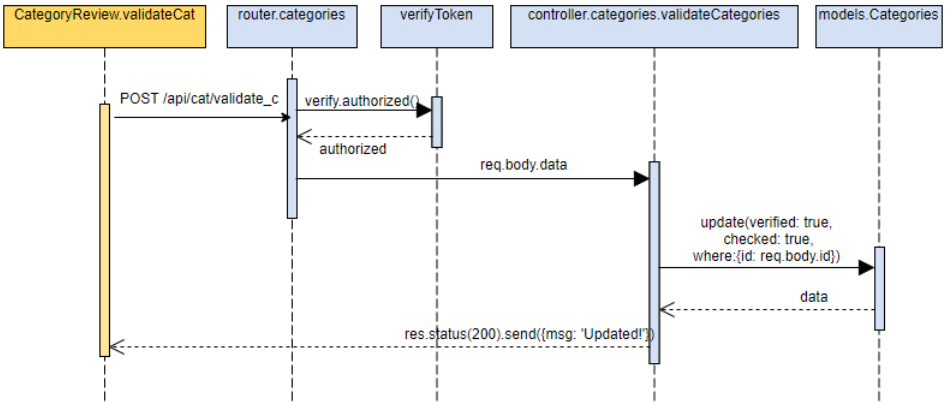


Figure 7.18: Sequence diagram of the validation process of a category.

There is a POST request that sends the category id to the server that checks the user privilege, and in case of success, it updates the corresponding entry in the database. When the success message is sent back to the client, the page re-renders automatically and doesn't show the validated category anymore. The same process is executed to invalidate the category or to validate and invalidate subcategories with the only difference of the request URI. Below the *Review categories* tab, there are two more tabs related to categories: *Validated categories* where the user can invalidate an already validated category and *Invalidated category* where the user can execute the opposite task on an invalid category. This process can be executed for subcategories as well and uses the schema presented in figure 7.18.

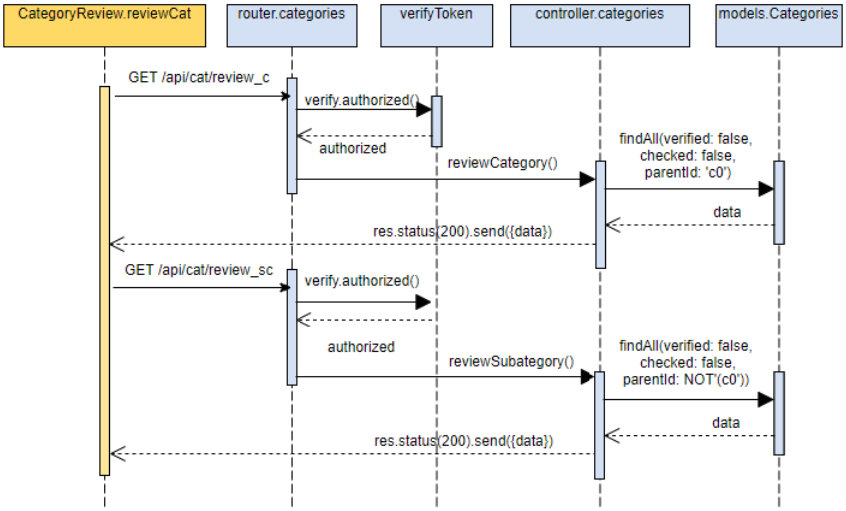


Figure 7.19: Sequence diagram of the category review process.

The system to show the categories and subcategories to be validated has to execute before the page rendering an API call represented in figure 7.19. The client asks first for the categories inserted by the user but not yet reviewed. The server validates the user token to check his privileges, executes the query, and returns the data to the client. The latter sends another request for the subcategories that follows the same pattern. This process is executed every time the user hits one of the buttons between *Review categories*, *Validated categories*, and *Invalidated categories*.

7.6.2 Validation of a point of interest

The second task of the authorized user is the validation of points of interest and can be accessed through the navigation bar on the left on the screen, selecting the Review points tab. the user has to check several parameters, so the page was designed using a table, as reported in figure 7.20.

The authorized user has to check first of all if the name and coordinates are the right one, then if the point was correctly categorized. Each row represents an entry in the database that is still not reviewed and has associated the Validate and Invalidate buttons.

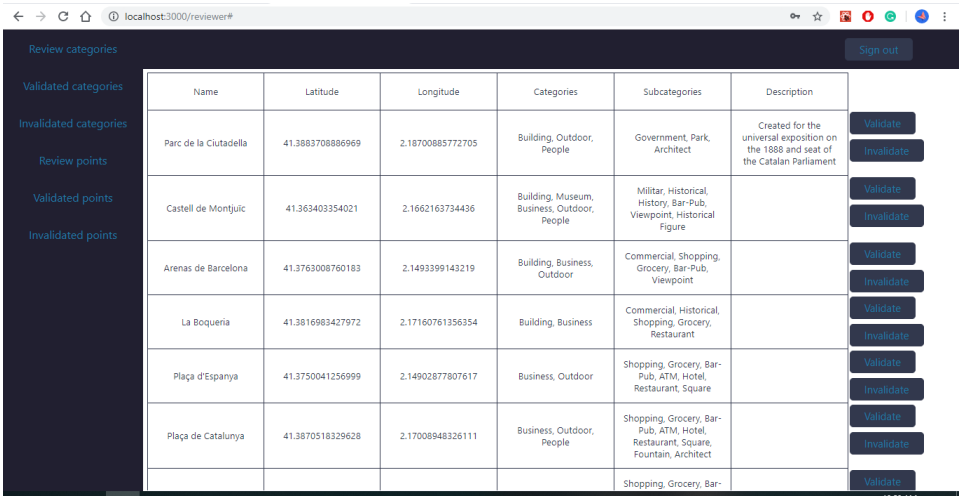


Figure 7.20: Application tab for the points of interest reviews.

The columns are the corresponding attribute stored in the database model. To validate a point of interest, the user has to select the *Validate* button correspondent. This action executes the API call to update the corresponding entry in the database. The process is similar to the one explained in section 7.6.1, and the sequence diagram for the validation process is reported in figure 7.22.The same schema of the UI is used for the *Validated points* and *Invalidated points* tabs. The process is the same as explained in figure 7.22 for both the actions with different URI.

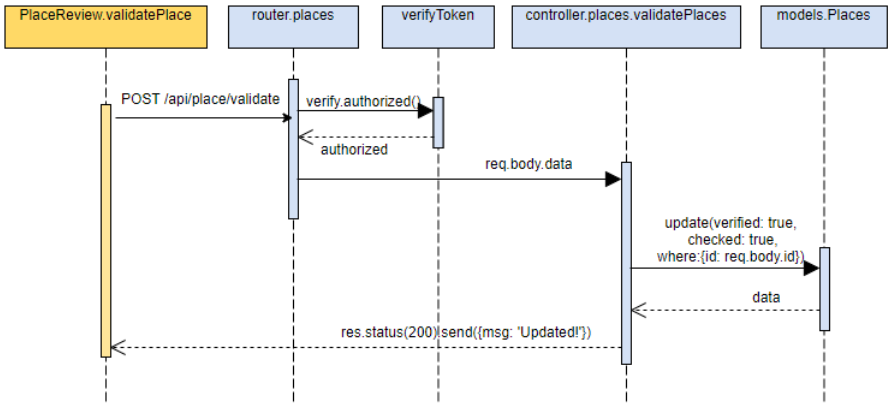


Figure 7.21: Sequence diagram of the point validation process.

To fill the review table, the system executes a GET request. The server checks the user token and then executes the query on the database, as reported in figure 7.22. This time the query includes two joins on the Categories table to have the categories and subcategories name instead of the id. As showed in figure 7.22, this time was impossible to use the Sequelize methods due to the length and complexity of the query, so a pure

SQL query was executed. This decision made the UI more readable and user-friendly and was applied to all the points of interest tab.

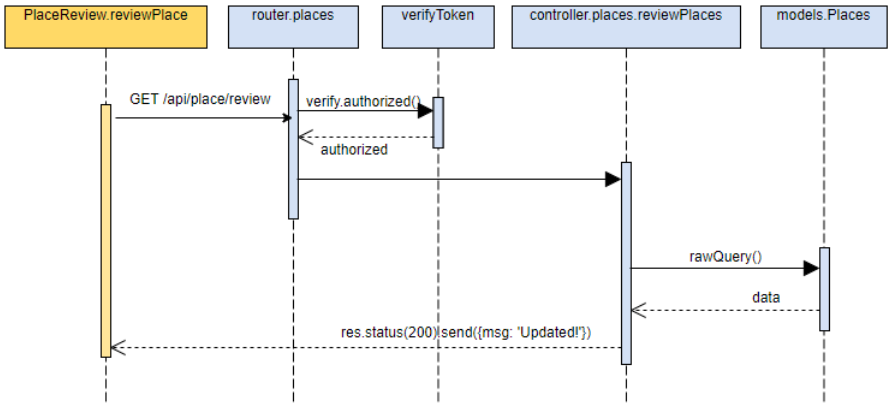


Figure 7.22: Sequence diagram of the points of interest review process.

Chapter 8

Conclusion and future works

The project idea was, since the beginning, appealing for the challenges presented and the possibility as a future user. First of all, it is a project that fits in a trending market because traveling is a pleasant step in everyone's life, and several applications are trying to jump out as market leaders, as explained in chapter 2. Another critical point in the project was the cutting-edge technologies proposed that were challenging but incredibly interesting. As explained in chapter 4, the technologies used are new and widely adopted by the most important world software companies.

During the design and implementation, the project evolved with the emerging of new ideas. At its creation, the project included the iBeacon technologies to create a proximity advertising of the application, but the idea was then shelved for privacy and spam creation problems. Some ideas were valuable and other too difficult for the limited amount of time and resources available, so they are presented as future work.

At the end of the project, fulfilled all the requirements stated in chapter 5 and reached all the goals established even if it is still a demo. The project retrieves, categorizes, and validates the point of interest inserted by the user employing categories focused on the touristic market. Moreover, it creates thematic tours putting together the fetch of points of interest and the journey creation in an individual solution complying with the expected non-functional requirements.

Further upgrades can be done to improve the application quality or automatize it:

Modification user profile to insert additional information and manage the profile. Some functionality as password change or retrieve the one forgot are not inserted in the application because the project is in its embryonic stage, and it needs additional improvements.

Improvement of the authorized user interface mostly for the points of interest review. The system could show the point to review on a map to facilitate the localization and the validation of its coordinates and add the possibility to change the data inserted by the user to correct spelling error or categorization without invalidating the categories.

Translation in different languages to reach more users and leave them the possibility

to choose which language to use. Even though English is the undisputed global language, there are users that prefer their native one.

Creation of a mobile application to use it on the go, reach more users, and expand the market.

Change People category to match the name of the public figure instead of the profession. This kind of categorization is more accurate than the one chosen and creates a more specialized tour.

Usage of data crawling techniques to execute the data validation instead of the manual validation executed by the authorized user.This functionality of the app is a bottleneck, if the insertion number is low, the reviewer can manage the validation, but with the increase of the application's usage, the reviewer can be very slow compared to the input data flow. A data crawling technique can automate and speed up the validation process, but it needs more research and trials of how much it was possible to execute in the project timeline.

Bibliography

[1] https://www.businessmodelsinc.com/data-is-the-new-currency/

[2] https://www.wired.com/story/wired-guide-personal-data-collection/

[3] https://www.predictiveanalyticstoday.com/google-public-data-explorer/

[4] https://datos.gob.es/en/catalogo/

[5] https://data.europa.eu/euodp/en/data/

[6] https://en.wikipedia.org/wiki/OpenStreetMap

[7] N. Borolea, D. Routa, N. Goela, Dr. P. Vedagirib, Dr. Tom V. Mathewb, "Multimodal Public Transit Trip Planner with Real-Time Transit Data ", *Procedia - Social and Behavioral Sciences*, *No. 104* , 2013, pp. 775 – 784

[8] https://beyondtransparency.org/chapters/part-2/pioneering-open-data-standards-the-gtfs-story/

[9] https://github.com/CanalTP/navitia/wiki/OpenTripPlanner-and-Navitia-comparison

[10] https://facebook.github.io/flux/

[11] https://www.techeconomy.it/2016/03/25/openstreetmap-e-google-maps/

[12] https://developers.google.com/maps/documentation/javascript/

[13] https://sequelize.org/master/

[14] https://en.wikipedia.org/wiki/Object-relational_mapping

[15] https://en.wikipedia.org/wiki/Client%E2%80%93server_model

[16] https://searchapparchitecture.techtarget.com/definition/RESTful-API

[17] https://en.wikipedia.org/wiki/Representational_state_transfer

[18] https://medium.com/of-all-things-tech-progress/understanding-mvc-architecture-with-react-6cd38e91fefd

[19] https://medium.com/mofed/react-redux-architecture-overview-7b3e52004b6e

[20] https://reactjs.org/

[21] `https://en.wikipedia.org/wiki/React_(web_framework)`

[22] https://www.html.it/guide/react-la-guida/

[23] https://reacttraining.com/react-router/web/guides/quick-start

[24] https://redux.js.org/api/api-reference

[25] https://medium.com/the-web-tub/managing-your-react-state-with-redux-affab72de4b1

[26] http://expressjs.com/en/api

[27] https://nodejs.org/en/docs/

[28] https://nodejs.dev

[29] https://www.postgresql.org/docs/

[30] https://medium.com/@purposenigeria/using-postgresql-and-sequelize-to-persist-our-data-c86854a3c6ac

[31] https://en.wikipedia.org/wiki/PostgreSQL

[32] http://www.postgresqltutorial.com/what-is-postgresql/

[33] https://agilemanifesto.org/

[34] https://agilemanifesto.org/principles.html