Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Grau en Enginyeria Informàtica (GEI)
Software Engineering

Study and development of a device tracking software

Author: Albert Díaz Benitez

Director: Jaume Figueras i Jové
Ponent: Dr. Antoni Guasch i Petit

Thursday 18th January, 2018

Abstract

A device tracking software is a software installed in an electronic device that is capable of reporting the device's location remotely. The average individual is likely to utilize multiple mobile technology applications daily. The applications have gained the ability to process more and more information to further enhance their features. The geolocation is one of the most important feature of these mobile applications.

There are a lot of geolocation software but most of them aren't open source and they need to be bought to be used. Open source software applications are those in which the source code or the base code is available for modification or enhancement by anyone.

InLab FIB has developed the TooPath environment for tracking mobile devices with a GPS. TooPath is the first free tracking system made of 100% free and open source data and technologies. However, the current version of TooPath is no longer maintained and non operative.

This project consists on developing a new version of the TooPath geolocation API using current technologies. The new version will offer a geotracking system for any kind of electronic device and a web to consume and demonstrate the API functionalities. A study and analysis of the open source licenses is done on the project as the API is offered as open source software.

Resum

Un software de seguiment de dispositius és un software instal lat en un dispositiu electrònic capaç de transmetre la ubicació del dispositiu de manera remota. És probable que l'usuari mitjà utilitzi diverses aplicacions mòbils cada dia. Les aplicacions han guanyat la capacitat de processar més i més informació per millorar encara més les seves característiques. La geolocalització és una de les característiques més importants d'aquestes aplicacions mòbils.

Hi ha molts softwares de geolocalització, però la majoria d'ells no són de codi obert o open-source i cal pagar per utilitzar-los. Les aplicacions de software de codi obert són aquelles en què el codi font o el codi base estan disponibles per a la seva modificació o millora per part de qualsevol persona.

InLab FIB ha desenvolupat l'entorn de TooPath per al seguiment de dispositius mòbils amb un GPS. TooPath és el primer sistema de seguiment gratuït fet al 100% de dades i tecnologies de codi obert. No obstant això, la versió actual de TooPath ja no es manté i no està operativa.

Aquest projecte consisteix a desenvolupar una nova versió de l'API de geolocalització de TooPath utilitzant les tecnologies actuals. La nova versió oferirà un sistema de seguiment per a qualsevol tipus de dispositiu electrònic i una web per consumir i demostrar les funcionalitats de l'API. També s'ha fet un estudi i anàlisi de les llicències de codi obert dins del projecte, ja que l'API s'ofereix com a programari de codi obert.

Resumen

Un software de seguimiento de dispositivos es un software instalado en un dispositivo electrónico que es capaz de informar de la ubicación del este de forma remota. Es probable que el individuo promedio use múltiples aplicaciones de tecnología móvil diariamente. Dichas aplicaciones han adquirido la capacidad de procesar más y más información para mejorar aún más sus características. La geolocalización es una de las características más importantes de estas aplicaciones móviles.

Hay un gran cantidad de software de geolocalización, pero la mayoría de estos no son de código abierto y deben comprarse para ser utilizados. Las aplicaciones de software de código abierto son aquellas en las que el código fuente o el código base están disponibles para modificación o mejora por parte de cualquier persona.

InLab FIB ha desarrollado el entorno de TooPath para rastrear dispositivos móviles con un GPS. TooPath es el primer sistema de seguimiento gratuito hecho de tecnologías 100% de código abierto. Sin embargo, la versión actual de TooPath no se mantiene y no es operativa.

Este proyecto consiste en desarrollar una nueva versión de la API de geolocalización TooPath utilizando las tecnologías actuales. La nueva versión ofrecerá un sistema de seguimiento para cualquier tipo de dispositivo electrónico y una web para consumir y demostrar las funcionalidades de la API. También se realiza un estudio y análisis de las licencias de código abierto en el proyecto, ya que la API se ofrece como software de código abierto.

Acknowledgements

I would like to thank inLab FIB for giving me the opportunity to do this project as my Final Degree Project. Specially, to my project manager Jose Francisco who has helped me, taught me a lot of development patterns and given me some tips. Also, I would like to thank my colleague, Sergio Paredes, for helping me improve and giving me his opinion as developer when I had doubts.

I would like to thank Antoni Guasch and Jaume Figueras for supervising the project and giving me some useful advice. Also, to Jaume for the job done as product owner.

I would like to thank my family for support me all these years during the bachelor degree. I would to thank all the classmates and specifically, Gerard Pradas and Pau Vilanova.

Lastly, I would like to thank my girlfriend, Gabriela López, for giving me strength to finish this project and support me on these hard months of work. Without her I couldn't have finished this degree.

Index

List of Figures

List of Tables

List of Listings

Glossary

200 OK Standard response for successful HTTP requests. 33–44, 46

201 Created The request has been fulfilled, resulting in the creation of a new resource. 33, 38, 41, 45, 60

204 No Content The server successfully processed the request and is not returning any content. 34, 40, 43, 46

400 Bad Request The server cannot or will not process the request due to an apparent client error. 33–46, 60

401 Unauthorized Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.. 33–35, 37–46

403 Forbidden The request was valid, but the server is refusing action. The user might not have the necessary permissions for a resource, or may need an account of some sort. 33–35, 37–46, 71

404 Not Found The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible. 33–36, 38–46

409 Conflict Indicates that the request could not be processed because of conflict in the request, such as an edit conflict between multiple simultaneous updates. 33, 38, 41, 45

API Stands for "Application Programming Interface." An API is a set of commands, functions, protocols, and objects that programmers can use to create software or interact with an external system. 4–6, 12, 13, 16,

PATCH The HTTP PATCH method is used to update a resource fields or attributes. In the successful case, PATCH returns the resource representation in JSON and an HTTP status code of 200. In an error case, it most often returns a 404 or 400 status code. 31, 32, 34, 39, 42, 59, 73

POST The HTTP POST method is used to create new resources. In the successful case, POST returns the resource created representation in JSON and an HTTP status code of 201. In an error case, it most often returns a 409 or 400 status code. 31, 32, 35, 36, 38, 41, 45, 59, 60, 70–72

PUT The HTTP PUT method is used to change the representation of a resource. In the successful case, PUT returns the new representation in JSON and an HTTP status code of 200. In an error case, it most often returns a 404 or 400 status code. 31–33, 39, 42, 44, 45, 59, 70–72

Python Interpreted high-level programming language for general-purpose programming. 10, 29, 53, 56, 69, 87

SQL Stands for "Structured Query Language". SQL is a standard language for storing, manipulating and retrieving data in databases. 4, 54, 61

TypeScript Free and open-source programming language developed and maintained by Microsoft. 62–64

Virtual Machine Emulation of a computer system. 51

Acronyms

GDAL Geospatial Data Abstraction Library. 87

GPS Global Positioning System. 1, 2

IDE Integrated Development Environment. 10, 12, 69, 80

JSON JavaScript Object Notation. 29, 30, 33–46, 53, 59, 73, 76

JWT JSON Web Tokens. 31, 53, 72, 88

PSF Python Software Foundation. 80

TDD Test-Driven Development. 9, 13, 14, 19, 29, 70, 76, 86

UML Unified Model Language. 46

URL Uniform Resource Locator. 33–35, 38, 39, 41–46, 54, 56, 58

YAGNI You Aren't Gonna Need It. 9

Chapter 1

Introduction

This project is a Degree Final Project at FIB-UPC, directed by Jaume Figueras i Jové and supervised by Antoni Guasch i Petit. It has been developed at inLab FIB[16] as part of CarGuard project[7].

The main goal of this Degree Final Project is to develop a device tracking software with its software architecture design. The technologies selected to develop this kind of software are also studied.

1.1 Context

Device tracking software is a software installed in an electronic device that is capable of reporting the device's location remotely. Depending on the software and the device in which it is installed, the software may obtain the location of the device by means of GPS, WiFi-location, IP address, or accelerometer readings, and it may report the position by means of e-mail, SMS, or other means of communications.

The average individual is likely to utilize multiple mobile technology applications daily. Some of those applications may be tools to help the user in several different ways (e.g., communication, information gathering, directional navigation, etc.). As applications have become more complicated and advanced with regard to their features, they have gained the ability to process more and more information to further enhance their features.

One source of information required for many applications is device location, and by extension the user's location. This allows certain applications to function more efficiently and improve the user experience (e.g., if a user is searching for a restaurant, the device application can narrow the recommendations based on the user's location). There are a variety of ways to retrieve a user's location information, (e.g., GPS, Galileo[1], Wi-Fi connections, etc.). This project aims to cover the need to have geolocation and other related services for any application.

Apart from the need for applications to use geolocation there is also an economic limitation. Most of the existing software requires a certain amount of money to be able to use it and, besides, they are restrictive and so are the modifications or updates. For that reason, carrying out the project following the guidelines of the open-source[21] software allows to not restrict the use of the software and possible personal modifications.

There is clearly a need to use geolocation software in most applications, computer systems and electronics. This need emerged specifically during the planning of a CarGuard project of the SEAT UPC Chair[29] in collaboration with the inLab FIB.

1.2 Stakeholders

The Stakeholders or actors involved are those people or entities interested in the development of the project. In this case, they can be classified in various groups according to their interests.

Developer

The developer is the person in charge of doing the research and implementation and also, of accomplishing the deadlines. In addition, the developer is responsible for the writing and delivering of the necessary documentation. This actor works as agreed with the director and project manager following the concrete requirements of the client. In particular, Albert Díaz Benitez, student of the UPC and report author.

[1]Galileo is the global navigation satellite system

Director

The director is the person responsible for guiding, giving advice and helping the developer in the project terms and objectives. The director's primary responsibility is to explain and specify the main functionalities that the software must have. This actor works as agreed with the project manager, and for this project the director also works as product owner. In particular, Jaume Figueras i Jové, PDI[2] from the UPC campus of Terrasa and project collaborator for inLab FIB.

Project Manager

The project manager is the person responsible for the management and guidelines of the whole project. This actor works as agreed with the director. The project manager also helps the developer in a more technical way in order to achieve a more correct development while taking into account the methodology chosen. In particular, José Francisco Crespo, PAS[3] from the FIB and project manager of inLab FIB.

Beneficiaries

This actor stands out as its main interest lies on the proper functioning of the project. In this case, being the software open-source everyone has access to it, this adds even more interest and benefits. In addition to the use of the software, the beneficiary actor can also be someone who wants the software to be used and known in order to gain fame and recognition. In this particular case, the development and production of such a software would fit the inLab FIB interests perfectly. In particular, SEAT[28] for the CarGuard project.

1.3 Problem formulation

During the analysis and design of CarGuard some functionalities generated the necessity to develop a new software. These functionalities are: the car's location when it's parked, the car track's history and the management

[2] "Personal Docent i Investigador"
[3] "Personal d'Administració i Serveis"

of the car's location. Also, there is an interest from the inLab FIB to update an old and non functional software that can offer the functionalities mentioned.

1.4 State of the art

As mentioned on the context section, geolocation is highly demanded and used by most applications, therefore, there will be many software that offer a similar solution. The goal of this section is to give information about the state of the art of the technologies that can help to achieve the goals of the project, and also, the existing software that have an approach to the project solution.

One obvious approach is the previous version of TooPath[34]. It was developed on 2010 and released on 2013 by inLab FIB. The technologies used were Apache, Joomla and mySQL. This version had two principal functionalities that the new version will also have: real-time display of the position of the device on the OpenStreetMap[9] and a track representation on the map. However, this software is no longer maintained and non functional in the present time.

When talking about geolocation software, developers always face problems to store raw geographical data on databases. However, there exist solutions and software implemented to solve these problems, one of these is PostGIS[27]. PostGIS is a spatial database extender for PostgreSQL[26] object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.

One existent framework that can be use to develop the software is the Django REST[8], with geographic add-ons. This framework supports full integration with PostGIS and Django REST Framework that allows to develop a API with full integration of geolocation raw data.

Another existent technology that can be useful is OpenStreetMap. It is a collaborative project to create a free editable map of the world, so, it is an open-source map. It can be useful to express the solution.

1.5 Scope

The principal scope of this project is focused on the development and design of a functional new version of TooPath. This new version includes the full API development and testing, the full web development and all the architectural design of the software.

This project will be developed in a team of three people: a project manager, a product owner and an intern as software architect and developer. It will be managed using the Agile[1] methodology, Sprints of 3 or 4 weeks (for specifically reasons), weekly meetings with all the team and daily meetings with the project manager and developer.

The first task will be setting up the environment, installing all the necessary requirements to have the Django REST Framework ready and functional in advance to the development of the software.

After the environment preparation it is important to test if the technologies suggested by the the project director (Django REST Framework and PostgreSQL with PostGIS) work correctly and achieve the expected results. In other words, it is primordial to test if the raw geographical data can be stored and expressed. If not, it can be assumed that the technology selected has been erroneous and therefore the research must continue.

If everything works correctly the full developing of the software integrating all the functionalities can be pursued. These functionalities are: users and external authentication, historic of tracks and geotracking position. It is mandatory to test all these functionalities to ensure their correct functioning.

A web will be developed to demonstrate the API functionalities on the project presentation and offer a user level solution to use the API. This web will be developed with full OpenStreetMap integration and it will use all the functionalities of the API. The Angular framework[2] and the Bootstrap 4[5] library are the chosen technologies for the web development.

Also, because the software developed will be open-source, the whole API will be on a public Git repository (with the correspondent README[4] to achieve its functionality). In consequence, the people interested on the soft-

[4]Repository file that gives information about the project installation, uses, license, etc

ware can pull the project and make all the updates/changes that they want or simply use the functionalities already implemented.

1.5.1 Project goals

The main goal of the project is to develop a new version of the TooPath geolocation API using current technologies. The new version will offer a geotracking system for any kind of electronic device and a web to consume and demonstrate the API functionalities. This main goal can be decomposed on these concrete goals:

Update TooPath with the current geolocation technologies.

Make a website for the direct use of the API functionalities.

Offer this API as open-source software. In consequence, all the technologies used must be open-source or free.

Regarding the functionalities that must be implemented in this project, the technical objectives are:

Design a geolocation API model to store and process.

Develop an authentication and permissions system.

Develop the functionalities for the tracking system: add, update and delete locations and tracks.

Make the software protected and secured.

Develop a web that will consume the API.

1.6 Methodology

This section explains the chosen methodology and all consortia and guidelines used for the development of the project. The Agile methodology has been chosen since it allows total flexibility with respect to future changes or differences of implementation, it implies a total integration of the equipment and therefore, each participating has a global vision of the project for its complete understanding. The recommendations of the project manager have also been taken into account along with the philosophy of work of the company when choosing the methodology and guidelines of the project.

The guidelines and characteristics chosen from the Agile methodology are described below.

Short development cycle

By using an iterative approach with short cycles it is much easier to keep the project on schedule and be conscious about the current state of the project. These short cycles are called Sprints on Agile methodology.

A Sprint is a set period of time during which specific work has to be completed and be ready for review. Each Sprint begins with a planning meeting. During the meeting, the product owner (the person requesting the work) and the development team agree exactly upon what work will be accomplished during the Sprint. The development team has the final say when it comes to determining how much work can realistically be accomplished during the Sprint, and the product owner has the final say on what criteria need to be met for the work to be approved and accepted. At the end of every Sprint, the features and tasks developed are shown on a demonstration with the client (product owner for this project).

Retrospectives

At the end of every Sprint a meeting called retrospective takes place. During the retrospective, the team discusses what went well in the Sprint, what could be improved, and what they commit to improve in the next Sprint.

By the end of the retrospective, the development team should have identified improvements that will be implemented in the next Sprint. Although improvements may be implemented at any time, the Retrospective provides a formal opportunity to focus on inspection and adaptation.

Code Review

Code review is a systematic examination of computer source code. It is intended to find mistakes overlooked in software development, improving the overall quality of software. In this particular project the Code reviews are done via regular meetings with the Development Team. It allows to improve the quality of the software code, share knowledge between the team members, locate possible errors or misuse of the technologies, etc.

Frequent product owner feedback

In this particular project, the product owner acts as a client as it is the actor who knows what must be done and has full interest in the completion of the project. By letting the product owner check the project state as frequently as possible, chances for misunderstandings are greatly reduced and, if they occur, are corrected much faster.

Pair Programming

Pair Programming[25] is an Extreme Programming[11] practice in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently.

In this project, the Pair Programming is used on casual situations like fixing errors and implementations that help to achieve knowledge at least in one of the participants. This type of programming also works as code reviewing and allows to reduce the number of possible errors during the development phase.

TDD

TDD [17] stands for Test-Driven Development, it refers to a style of programming in which three activities are tightly interwoven: coding, testing (through writing unit tests) and design (in the form of refactoring). It can be succinctly described by the following set of rules:

1. Write a unit test describing an aspect of the program
2. Run the test, which should fail because the program lacks that feature
3. Write just enough code, the simplest possible, to make the test pass
4. Refactor the code until it conforms to the simplicity criteria, repeat accumulating unit tests over time

This style of programming allows to accomplish the YAGNI[12] develop philosophy that states a programmer should not add a functionality until deemed necessary. It also allows a better quality code design and an easier error location.

Git Flow

If this work flow is implemented, whenever something is done in the code, a corresponding branch will have to be created, the code worked on and incorporated to the main branch and the created branch closed. Throughout the working day the git, merge, push and pull commands will need to be executed several times a day, as well as checkouts of different branches, delete them, etc. Git-flow[10] is created to solve this lack of time problem, it is a set of extensions that save enough work to execute all these commands, simplifying the management of the branches of the repository.

For this project, using Git-flow model for branching allow to organize the git branches and make the work flow more traceable.

1.7 Monitoring and development tools

Git and GitLab are going to be the tools used to monitor the evolution of the project. Git enforces a short cycle approach (by means of commits), forces the developer to document all changes and helps a lot with the proper use of an Agile methodology. GitLab provides a remote Git repository and an integrated issue tracking system, perfect for setting milestones and tracking the project's progress. In addition Slack[31] tool will be used for the team communication.

PyCharm is going to be the IDE selected for this project. PyCharm is a IDE developed for JetBrains[32] specialized on the use of Python. This IDE allows to integrate all the monitoring tools for this project, and also, the use of Git Flow on the same IDE. Also, WebStorm is the IDE selected for the web development (developed also for JetBrains). This IDE is the most recommended and popular for web development.

Chapter 2

Project management

2.1 Schedule

Estimated project duration

The estimated project duration is approximately of 6 months and a half. The project starts on April 18th, 2017 and the deadline is on December 22th, 2017.

Considerations

Due to the use of Agile methodologies, new requirements for the project may appear. Therefore, the planning and scheduling may change and new tasks and functionalities may appear.

Furthermore, during August 2017 the project was stopped due to summer holidays.

2.2 General planning

Resources

For the project development two types of resources are needed: material and human.

Human resources:

For the development of this project a team of three people was arranged:

Project Manager: Person in charge of the management and leading of the project with a weekly approximated dedication of 10 hours.

Product Owner: Person that has a vision of the final software and functionalities. The product owner has a weekly approximated dedication of 5 hours.

Software Developer: Person in charge of developing the software, and also for this project, designing the software as a software architect. The developer has a weekly dedication of 20 hours.

Material resources:

Resource	Type	Functionality
HP Compaq 8100 Elite SFF	Development Tool	For developing the project and write the documentation
Local server	Development Tool	For the local use and testing of the software
Production server	Development Tool	For the external use of the software
PyCharm	Development Tool	IDE for developing the API
WebStorm	Development Tool	IDE for developing the web
Git, GitLab	Management Tool	Version control of the code
Slack	Communication Tool	Intern and private chat for team communication
ShareLatex	Development Tool	Online LaTeXeditor for the documentation
Workplace	Place Tool	Place to work with good conditions
Django REST	Development Tool	Framework for the API development
django-rest-gis	Development Tool	Geo library for the API development
PostgreSQL with PostGIS	Development Tool	Add on for the Database
OpenStreetMap	Development Tool	Map used for the geolocation visualization on the web
Vagrant	Development Tool	Virtual Machine generator for local tests
Angular	Development Tool	Framework for the web development
Bootstrap	Development Tool	Style library for the web development
TypeScript	Development Tool	Programming language for the web development
Python	Development Tool	Programming language for the API development

Table 2.1: Material resources

Alternatives and action plan

In this section, the execution of the plan that was created will be described.

As it was commented on the previous chapter, the Agile methodology allows to be flexible with the possible changes of requirements and functionalities arranged in every Sprint. So, the Sprint and functionalities aren't fixed. If a Sprint doesn't end well or completes all the functionalities, the planning can be reorganized for a successful project finalization. Also, if extra time was needed for the API proper functionality an extra Sprint should be used, or less hours should be dedicated on the website development.

Since weekly meetings have been arranged with the director/product owner and also, daily meetings with the project manager, there will be enough time to detect possible deviations from the original planning and correct them.

Next, some examples of potential sources of delays are mentioned.

Time:

As mentioned before, in case of lack of time, a simpler web could be made. It is also possible to use an extra Sprint to compensate this lack of time, this extra Sprint must be well analyzed and estimated to finish the software.

Learning:

Due to the inexperience on geolocation software and the programming language used, the learning curve could be considerable. To avoid this important learning curve it is considered to apply Pair Programming or ask for help to experienced people.

Bugs:

Considering that a geolocation framework is a relatively complex software and also, that there is poor documentation, it is easy to introduce bugs while implementing. This may also cause delays if bugs are not located and resolved fast enough. TDD methodology should be able to avoid the appearance of most of the bugs during the development phase.

CarGuard:

During Sprint 2 and 3 the functionalities developed depend on the development of the CarGuard project. Again, these Sprints are dependent on the

CarGuard necessities and development. Also, the CarGuard product owner is from an external company, so meetings aren't that well programmed or frequent. This fact can become a loss of time.

In case this happened an extraordinary meeting would happen to reorganize the Sprint and try to develop and go forward for this project.

2.3 Project planning

The different phases of the project and how every Sprint is divided and managed is described in this section. The phases of project planning and feasibility and project analysis and design start on April 18th and end on May 12th.

2.3.1 Sprints description

As seen previously this project follows the Agile methodologies and also, uses TDD and retrospectives. So, every Sprint has a duration of three weeks (there are exceptions for Sprint 2 and 3) and it is divided in these phases:

Analysis: This initial phase's purpose is to specify the functionalities and use cases. Also, the possible requirements or functionality variations may be specified.

Test Writing: Because of the TDD, it is mandatory to write the tests of the Sprint functionalities before the implementation.

Implementation: This implementation phase must implement everything necessary for successful testing.

Test Validation: After implementation it is very important to review that all the tests were passed and in consequence, the implementation phase successful. If not, it will be necessary to review the implementation and make adjustments.

Integration: These phase's objective is to integrate the functionalities implemented on the public server for the people use.

Retrospective: At the end of every Sprint a Retrospective meeting is done. These will help to perform a better next Sprint.

Next the main objectives of every Sprint are described.

Project planning and feasibility:

This is the first phase of the project. It consists in agreeing on the scope and budget of the project, and also, making the planning. All this is done in one or various meetings with the project manager and the product owner.

Taking advantage of the meetings held with the project manager and product owner the study of the state of art will start.

Sprint 0 - Project analysis and design:

The main objective of this phase is to make an accurate analysis of the project and develop the consequent design.

First of all, in the project analysis it will be necessary to define and set the objectives, requirements, features and the use cases of the software. Furthermore, the state of art is expanded and an analysis and evaluation of the different technologies used for the development is provided.

On the other hand, project design consists on creating the architecture of the software: sequence diagrams, model diagrams (UML), database design and server architecture design. This design will help to perform a high quality software and prevent a waste of time in case this was to be done during the implementation.

Sprint 1 - Project kickoff and environment setup:

This Sprint starts on May 15th and ends on June 2th. The main objective of this Sprint is to have the environment ready to develop. This environment setup consists on the IDE's installation, the frameworks and the respective plugins, add-ons and dependencies. Also, the virtual machine will be created and its functionality checked.

Sprint 2 - CarGuard functionalities:

This Sprint starts on June 5th and ends on June 30th. The objectives of this Sprint and the third are to implement and integrate all the functionalities

that CarGuard project needs. Both Sprints have one extra week of duration due to the complexity of the integration for the CarGuard use.

Sprint 3 - Performance and CarGuard functionalities:

This Sprint starts on July 3rd and ends on July 28th. This Sprint has a more technical specification because it is focused on a better performance of the implementation in the second Sprint and also in following the API patterns for the models implemented.

Sprint 4 - Users and permissions:

This Sprint starts on September 12th and ends on September 29th. This Sprint is focused on following the API patterns of the permissions and users.

Sprint 5 - Google log in and first Tracks model:

This Sprint starts on October 2nd and ends on October 20th. After the implementation of the users and token authentication on the previous Sprint, Google log in functionality is added.

Sprint 6 - Tracks implementation:

This Sprint starts on October 23th and ends on November 10th. This Sprint is focused on the most important functionality of the project, the tracks.

Sprint 7 - API finalization and Web kickoff:

This Sprint starts on November 13th and ends on December 1st. On this Sprint it is mandatory to make all the adjustments, if necessary, to finalize the API implementation. Also, the development of the website used to validate and demonstrate the API functionalities will start. This website has a map, a user/device management and tracks historical.

Sprint 8 - Web implementation:

This Sprint starts on December 4th and ends on December 22th. This Sprint is reserved to implement and finalize the website.

Sprint 9 - Extra sprint for possible bugs or improvements:

This Sprint starts, if necessary, on January 8th and will end on the final deadline. This Sprint is optional as it's only necessary if there are bugs or/and mandatory improvements. Otherwise this Sprint won't happen.

Closing Phase:

The main objective of this phase is to review and close the project. Also, the report and all the necessary documentation will be written during this phase.

2.3.2 Estimated time

Sprint	Time spent (hours)
Project planning and feasibility	30
Project analysis and design	65
Project kickoff and environment setup	45
CarGuard functionalities	70
Performance and CarGuard functionalities	66
Users and permissions	56
Google log in and first Tracks model	50
Tracks implementation	70
API finalization and Web kickoff	58
Web implementation	80
Closing phase	20
Extra sprint for possible bugs or improvements	40
Total	650

Table 2.2: Estimated time of the Sprints

2.3.3 Gantt chart

The Gantt Chart can't be showed in this section partially due to the big size of the chart. See the following link to see it in full.

2.4 Project budget

In order to carry out this project, the resources mentioned previously will be needed. In this section, an estimation of the cost of the project is presented, taking into account the mentioned hardware and software resources, and the corresponding amortizations. In addition, the indirect costs and unexpected costs of the project are taken into consideration.

2.4.1 Human resources budget

This project is going to be developed by three people:

José Francisco Crespo as project manager in inLab FIB.

Jaume Figueras as product owner in inLab FIB.

Albert Díaz as software developer and architect in inLab FIB.

To know the hour/salary of every role the website Page Personnel[24] is used. It is also important to consider that the price of software architect and developer is extracted from the actual salary of an intern in inLab FIB.

The table 2.3 shows the costs of the human resources needed in the development of the project.

Role	**Price per hour**	**Hours**	**Cost**
Project Manager	20 €	305	6100 €
Product Owner	20 €	152	3040 €
Software Architect	8 €	95	760 €
Software Developer	8 €	515	4120 €
Total			14020 €

Table 2.3: Human resources budget

The project manager role is working on the project planning, and also, helping the software developer if it is needed.

The software architect role is working on the software architecture design (model diagrams and database design) during the project planning and feasibility phase and also on the project analysis and design phase.

The software developer role is working on the software development and also, testing it because of the TDD use.

The product owner role is deciding and making clear the specification and functionalities desired for the final software.

Sprint and Tasks	**Hours estimated**	**Roles**	**Cost**
Project planning and feasibility	15	Project Manager	300 €
	7	Product Owner	150 €
	30	Software Architect	240 €
	52		690 €
Project analysis and design	32	Project Manager	640 €
	16	Product Owner	320 €
	65	Software Architect	520 €
	113		1480 €
Sprint 1	23	Project Manager	460 €
	11	Product Owner	220 €
	45	Software Developer	360 €
	79		1040 €
Sprint 2	35	Project Manager	700 €
	17	Product Owner	340 €
	70	Software Developer	560 €
	87		1600 €
Sprint 3	33	Project Manager	660 €
	16	Product Owner	320 €
	66	Software Developer	528 €
	115		1508 €
Sprint 4	28	Project Manager	560 €
	14	Product Owner	280 €
	56	Software Developer	448 €
	98		1288 €

Continued on next page

Table 2.4 – *Continued from previous page*

Sprint and Tasks	Hours estimated	Roles	Cost
Sprint 5	25	Project Manager	500 e
	12	Product Owner	240 e
	50	Software Developer	400 e
	87		1140 e
Sprint 6	35	Project Manager	700 e
	17	Product Owner	340 e
	70	Software Developer	560 e
	122		1600 e
Sprint 7	29	Project Manager	580 e
	15	Product Owner	300 e
	58	Software Developer	464 e
	283		1344 e
Sprint 8	40	Project Manager	800 e
	20	Product Owner	400 e
	80	Software Developer	640 e
	140		1840 e
Sprint 9	20	Project Manager	400 e
	10	Product Owner	200 e
	40	Software Developer	800 e
	70		1400 e
Closing Phase	10	Project Manager	200 e
	5	Product Owner	100 e
	20	Software Developer	400 e
	35		700 e

Table 2.4: Sprints and tasks budget

2.4.2 Hardware budget

In order to implement TooPath a set of hardware will be needed for various purposes. It is important to consider that the hardware used are from inLab FIB, so it has been used by previous workers. This fact is considered to calculate the useful life and amortization. The table 2.5 shows the hardware amortization and cost.

Product	Price	Time	Percentage	Useful life	Amortization
HP Compaq 8100 Elite SFF	660 €	6 month and a half	-	5 years	71.50 €
SkyLab Server	1300 €	6 month and a half	-	5 years	200 €
Total	1960 €				271.50 €

Table 2.5: Hardware resources budget

2.4.3 Software budget

The table 2.6 summarizes the costs and amortization of the software that are going to be needed to develop the project.

Product	Price	Time	Useful life	Amortization
PyCharm License	199.00 €	6 month and a half	1 year	107.79 €
WebStorm License	129.00 €	6 month and a half	1 year	69.87 €
Git	0 €	-	-	0 €
GatLab	0 €	-	-	0 €
Django REST	0 €	-	-	0 €
Django Rest-gis	0 €	-	-	0 €
Postgres with PostGIS	0 €	-	-	0 €
Vagrant	0 €	-	-	0 €
OpenStreetMap	0 €	-	-	0 €
ShareLatex	0 €	-	-	0 €
Total	328.00 €			177.66 €

Table 2.6: Software resources budget

2.4.4 Unexpected costs

Table 2.7 shows the unexpected costs that may appear. This allows to have a certain budget margin that will help against unexpected events that may occur during the project development.

Event	Cost	Probability	Event Cost
Extra Sprint 9	1400 €	10%	140€
PC failure	660 €	5%	33 €
Server failure	1300 €	0.036%	46.80 €
Total	3360 €	6.54%	219.80 €

Table 2.7: Unexpected costs of the project

There might be be two types of unexpected events:

Extra Sprint 9: It is possible to have an extra sprint due to the possible bugs or improvements. This extra Sprint increments the total hours of the project in 40 hours.

Hardware failure: In case of a problem with the hardware, it must be repaired or replaced, increasing the cost of the project.

2.4.5 Indirect costs

Next, table 2.8 provides an estimation of the indirect costs that are not comprised in any of the previous categories.

Product	Price	Percentage	Cost
Workplace	3000 €	1.43%	42.90 €
Internet connection	40.00/month	1.43%	0.57 €
Transport	0.00 €	-	0.00 €
Total			43.47 €

Table 2.8: Indirect costs of the project

These indirect cost are described here:

Internet connection: Each member must have an internet connection to be able to work on the project. Given that the company already has an internet connection, only the bandwidth percentage used is calculated. The project involves 3 members and the company is made up of 70 people, so it will be 4.28% of bandwidth contracted.

Workplace: For the four members of the team there is space available to work in an office with all the necessary facilities. The cost includes: office rent, maintenance (cleaning and repairs) and basic services (electricity, air conditioning, heating, etc). In this section the costs of the office will be taken into account under the same conditions that the internet connection, with 4.28 of the total of the invoices.

Transport: The project will be carried out in the same company, therefore, since aren't trips for the meetings because the product owner will be in the same company, this cost is not taken on account.

2.4.6 Budget monitoring

Constant cost control will be done in order to see if there are deviations in the project. The costs derived from human resources will be controlled for each iteration of the project and it will be observed if there are deviations. These deviations will be calculated with the hours each team member has assigned with the ones performed during each of the iterations.

At the end of the project the estimated initial cost will be compared with the final cost. If the final cost is greater than the estimated initial cost, it will be studied if the difference has been caused by the known unforeseen events and if it can be covered with the corresponding budget. If not, the contingency budget will be used to cover these deviations.

2.4.7 Total budget

Using data shown in the previous tables the following table can be used to describe the total cost of the project. Also, being a long-term project, six months and a half, it was decided to save a part of the budget for the contingency plan, more specifically 15% of the direct and indirect costs obtained previously.

Concept	Cost
Human resources	14020 €
Hardware resources	271.50 €
Software resources	177.66 €
Unexpected costs	219.80 €
Indirect costs	43.47 €
Subtotal	14696.68 €
Contingency	2204.50 €
Total	16936.93 €

Table 2.9: Total costs of the project

2.5 Sustainability

In this section the sustainability of the project will be evaluated in three different areas: economical area, social area and environmental area. This analysis will be based on the application of the sustainability matrix to the project, shown in table 2.10.

	Project Development	**Exploitation**	**Risks**
Environmental	Design consumption	Ecological Footprint	Environmental risks
	9/10	17/20	-20/-1
Economical	Bill	Viability Plan	Economical Risks
	9/10	18/20	-20/-1
Social	Personal Impact	Social Impact	Social Risks
	8/10	15/20	-20/0
Sustainability range	26/30	50/60	-60/-2
	74/90		

Table 2.10: Sustainability of the project

2.5.1 Economical

A detailed quantification of all the costs involved in the project has been done, both of material and human resources, as shown in previous sections of

this document. It's relevant to notice the use of mostly open-source software. So, it is feasible and competitive in economic terms.

The duration of each activity is estimated and fitted in relation of its importance. As stated in the context of the project there is not actual similar functional solution, so no existing technology can be reused.

The proposed solution will be less expensive than current solutions from an economical point of view because the use of all the free software possible and because of open-source, so the updates and improvements will be cheaper. Hence, it is a totally free software, so the end-user won't need to pay to use it. Also, this project may have impact in the economy of organizations: they may have minimum costs in the use of this software.

It is also important to highlight that the development team is formed by three people. This is the minimum possible for a software development team. This fact allows to minimize the human costs as much as possible, and in consequence, the possible hardware and software resources.

2.5.2 Social

This project is done to update an old software that was used to track and manage vehicles fleets and manage their locations. This update is needed because some companies, like Aigues de Mataró, are interested on using it. Also, as it is previously mentioned this new version is needed for inLab and the CarGuard project.

Personally, this software won't have an impact because of the use but it will have an impact because of the development. This impact will be caused by the new technologies that will be used and that will improve the working experience of a software developer.

On the other hand, this new version is gonna be open-source, which means it's going to be a free software. This fact allows free software use for any kind of user, and also, possible improvements or new versions that can be useful for anyone who wants to use it and work on this software. Ultimately, this software will have a good impact and zero social risks.

2.5.3 Environmental

This project will be done using the minimum resources possible to minimize the design consumption. Due to this, the ecological footprint is minimum for the resources of a software development project. This project is developed on a company with existing resources, so there isn't a necessity to buy any new hardware (because of this, the ecological footprint is minimum).

Although, considering that in a software developing project there will be emissions of CO2 , the main objective is to reduce them to the minimum. So, assuming that a desktop computer consumes 250 Watts, the desktop will be used during all the development time. Taking these values into account, the energy consumed will be 123 KW, so the ecological footprint will be of 47.355 kg of CO2. Also, all the documentation that has to be read, will not be printed, instead, it will be read from the computer screen.

On the other hand, this project will have low environmental risks. The only environmental risk is the possibility that the hardware fails or breaks, this will cause a necessity to buy or repair this hardware causing an increment of the ecological footprint.

Chapter 3

Requirements analysis

On this chapter the analysis of requirements is presented to clearly define and concrete the functionalities and restrictions that must be taken into account to start implementing the API and the web.

This requirements analysis was performed on the Sprint 0, project analysis and design. The whole analysis was done by the developer following the patterns learned on the degree.

The following scheme will be followed to define the functional and non-functional requirements:

[Requirement title]

[non] functional requirement #number

Description

Justification

Acceptance criteria

3.1 Non functional requirements

The non-functional requirements represent general characteristics and define the restrictions of the application to implement.

Open Source

Non functional requirement #**1**

Description: The API developed for this project must be open-source. Therefore, the API source code, documentation must be available for free to the public. The technologies used to develop this API must be free or open-source to evade possible restrictions for future users.

Justification: The API is a new version of an old software also open-source, so, to maintain the philosophy it must be open-source too. Also, this API will be public so it has to be open-source to ensure the complete usability and possible modifications and updates from other collaborators and users.

Acceptance criteria: The API source code will have a valid open-source license[22].

Documentation

Non functional requirement #**2**

Description: The API must have documentation, including examples of how developers can use each function, and the constraints that the API has.

Justification: This documentation is needed to provide and improve the performance and use of the API. Also, this documentation is essential for the good use and understanding of the API for the developers.

Acceptance criteria: The API documentation will be created using the Swagger editor[33].

Testability

Non functional requirement #**3**

Description: Specifically, the source code of the API must have unit tests for every view and endpoint. These tests must check the status codes and JSON response, and database instances consistency.

Justification: These tests allow the developers to ensure the correct functionality of every view and endpoint on every moment. These tests will help the correct modifications and upgrades of the API.

Acceptance criteria: The TDD methodology of inLab will be followed.

Readability

Non functional requirement #**4**

Description: The source code of the API will be public and must be readable and understandable for any developer or users with Python and Django notions.

Justification: A readable code will allow a better use for the developers to implement new functionalities and understand the current ones.

Acceptance criteria: Python and TDD convention will be followed during the API development.

3.2 Functional requirements

The functional requirements define the functionalities or services that it must have the software.

User management

Functional requirement #**5**

Description: The system offers a basic user management. All the resources created have an ownership corresponding to the system user that has created the resource.

Justification: With a basic user management and resources ownership the private information will be secured and not visible for other system users.

Token authentication

Functional requirement #**6**

Description: The API must offer authentication via token. This token must be sent on every request and it identifies the user that has sent the request. On the web case, it also must identify the user logged.

Justification: This token authentication will make all the API requests secure and improve the log in system for the API and web communication.

GeoJSON standards

Functional requirement #**7**

Description: The JSON response of the API must follow the GeoJSON[13] standards.

Justification: Following the GeoJSON standards will help the integration of the API to others systems.

OpenStreetMap integration

Functional requirement #**8**

Description: The web must have integration with the OpenStreetMap technology for the visualization of the tracks and devices on the map.

Justification: The OpenStreetMap technology allows the use of an open-source map. This will help to accomplish the non-functional requirement #1.

Chapter 4

Specification

This chapter presents the complete description of the behavior of the implemented system. The functionalities of the Web and API, and the conceptual schemes are also specified and described on this chapter.

4.1 API specification

On this section the different functionalities that the API requires are defined. This functionalities are grouped according to their tasks on the API. Also, every group has a brief explanation and its functionalities are described according the API resources and HTTP methods.

The following scheme will be followed to describe every functionality of every group:

[**Functionality title**]

HTTP Method: [GET, POST, PUT, PATCH, DELETE]

Endpoint

Authentication[1]: [YES, NO]

Description

Required fields

[1]using JWT as authentication

Data returned

Errors handled

The guidelines found on the Web API Design[19] is used to specify the PATCH and PUT methods. In summary, the PUT method is used to change the representation of a resource and the PATCH method is used to update the content fields of a resource that won't affect this representation. For example, if the email of a user needs to be changed (which is unique), a PUT method must be used. If this email isn't part of a resource representation a PATCH method can be used. Also following this guidelines, it has been decided that the identifier of all the resources will be the ID.

Also, the API has been specified following the OpenAPI Specification[20] (formerly known as the Swagger Specification). This specification type is easy-to-learn, language agnostic, and both human and machine readable. This specification can be read on the appendix B.

4.1.1 Functionalities

The different groups are: users management, devices management, tracks management, locations management and authentication.

Users management

This group is formed for the petitions that create, edit, delete and retrieve information about the users resource. On the API the user is identified by the id and also, by its username or email which are unique fields. Every user has an owner permission for the resources created and also, it allows to protect the reading and editing of these resources.

User creation:

HTTP Method: POST

Endpoint: /users/

Authentication: No

Description: This method allows the creation of a user. This method doesn't require authentication. For security reasons the password provided is saved codified and it isn't presented on any response of the API. Also, the username and email are unique fields and the format of this last one is verified.

Required fields: username, email and password in the request body.

Data returned: 201 Created status code and a JSON object with the user fields instance created excluding the password field due to security reasons.

Errors handled: 400 Bad Request and 409 Conflict.

User information retrieving:

HTTP Method: GET

Endpoint: /users/:user_id/

Authentication: Yes

Description: This method allows to retrieve the information of a user identified with the ID passed by parameter. This information can only be accessed if the id of the user authenticated corresponds with the ID passed by parameter.

Required fields: user_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the user fields except the password due to security reasons.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Change user representation:

HTTP Method: PUT

Endpoint: /users/:user_id/

Authentication: Yes

Description: This method allows to change the representation of a user identified with the ID passed by parameter. This change can only be done if the id of the user authenticated corresponds with the id passed by parameter. Also, the request body must contain all the fields that represent the user resource.

Required fields: user_id as URL parameter and username, email and password in the request body.

Data returned: 200 OK status code and a JSON object with the new user representation and all the other fields except the password due to security reasons.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Delete user:

HTTP Method: DELETE

Endpoint: /users/:user_id/

Authentication: Yes

Description: This method allows to delete the user resource identified with the ID passed by parameter. This delete can only be done if the id of the user authenticated corresponds with the id passed by parameter.

Required fields: user_id as URL parameter.

Data returned: 204 No Content status code.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Update user fields:

HTTP Method: PATCH

Endpoint: /users/:user_id/

Authentication: Yes

Description: This method allows to update the fields of a user identified with the ID passed by parameter. This update can only be done if the id of the user authenticated corresponds with the id passed by parameter.

Required fields: user_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the user fields and its new values, except the password due to security reasons.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Authentication

This group is formed by the petitions that perform the user authentication and the token refresh and verification. When a user performs a log in a token is provided, this token identifies and authenticates the user. Also, for security reasons this token will expire on a short period of time and must be refreshed during this period of time. This token will be on every request that needs authentication to identify the user and apply the owner permissions.

User log in:

HTTP Method: POST

Endpoint: /login/

Authentication: No

Description: This method allows the user to authenticate and retrieve the token.

Required fields: email and password in the request body.

Data returned: 200 OK status code and a JSON object with a valid token.

Errors handled: 400 Bad Request and 404 Not Found.

User Google log in:

HTTP Method: POST

Endpoint: /login-google/

Authentication: No

Description: This method allows the user to authenticate and retrieve the token using the Google account.

Required fields: email and Google token in the request body.

Data returned: 200 OK status code and a JSON object with a valid token.

Errors handled: 400 Bad Request and 404 Not Found.

Token verification:

HTTP Method: POST

Endpoint: /api-token-verify/

Authentication: No

Description: This method allows the user to check the token validation. The token that must be checked is passed in the request body.

Required fields: token in the request body.

Data returned: 200 OK status code that confirms the token validation.

Errors handled: 400 Bad Request.

Refresh a token:

HTTP Method: POST

Endpoint: /api-token-refresh/

Authentication: No

Description: This method allows the user to refresh a token. The token that must be refreshed is passed in the request body. If the token is already expired, the user must log in again.

Required fields: token in the request body.

Data returned: 200 OK status code and a JSON object with the new token refreshed.

Errors handled: 400 Bad Request.

Devices management

This group is formed by the petitions related with the devices resource: create, edit, delete and retrieve information of a device. The device resource is the main resource of the API. This resource is associated with the user creator as owner, the tracks resources and its actual location.

Get all devices information:

HTTP Method: GET

Endpoint: /devices/

Authentication: Yes

Description: This method allows to retrieve all the devices information. It only returns the devices owned by the request user. All the devices returned have all the information of the tracks related and its locations.

Required fields: None.

Data returned: 200 OK status code and a JSON array with every device and all the information and fields.

Errors handled: 401 Unauthorized and 403 Forbidden.

Create a device:

HTTP Method: POST

Endpoint: /devices/

Authentication: Yes

Description: This method allows the creation of a device. When a device is created automatically is created and associated an actual location instance (wich is empty until is updated).

Required fields: name, trash, device_type and device_privcay.

Data returned: 201 Created status code and a JSON object with the fields of the device created.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 409 Conflict.

Device information retrieving:

HTTP Method: GET

Endpoint: /devices/:device_id/

Authentication: Yes

Description: This method allows to retrieve information of the device identified with the ID passed by parameter. This information can only be accessed if the id of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the device fields (including tracks and locations).

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Change device representation:

HTTP Method: PUT

Endpoint: /devices/:device_id/

Authentication: Yes

Description: This method allows to change the representation of the device identified with the ID passed by parameter. This change can only be done if the id of the user authenticated corresponds with the ID of the device owner. Also, the request body must contain all the fields that represent the device resource. With this method instances related to this device like the tracks or their locations and the actual location can't be changed.

Required fields: device_id as URL parameter and the name in the request body.

Data returned: 200 OK status code and a JSON object with the device fields (including tracks and locations).

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Update device fields:

HTTP Method: PATCH

Endpoint: /devices/:device_id/

Authentication: Yes

Description: This method allows to update the fields of the device identified with the ID passed by parameter. This change can only be done if the id of the user authenticated corresponds with the ID of the device owner. With this method instances related to this device like the tracks or their locations and the actual location can't be updated.

Required fields: device_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the device fields updated (including tracks and locations).

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Delete device:

HTTP Method: DELETE

Endpoint: /devices/:device_id/

Authentication: Yes

Description: This method allows to delete the device identified with the ID passed by parameter. This delete can only be done if the id of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id as URL parameter.

Data returned: 204 No Content status code.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Tracks management

This group is formed by the petitions related with the tracks resource: create, edit, delete and retrieve information of a track. A track is associated with a device and use its owner permission. Also, the track is associated with various locations.

Get all tracks information:

HTTP Method: GET

Endpoint: /devices/:device_id/tracks/

Authentication: Yes

Description: This method allows to retrieve all the tracks information associated with the device identified by the ID parameter. This information can only be accessed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id as URL parameter.

Data returned: 200 OK status code and a JSON array with every track and all the information and fields.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Create a track:

HTTP Method: POST

Endpoint: /devices/:device_id/tracks/

Authentication: Yes

Description: This method allows the creation of a tracks. This creation can only be performed if the ID of the user authenticated corresponds with the ID of the device owner. Also, when the track is created will be associated to the device identified with the parameter ID.

Required fields: device_id as URL parameter and the name in the request body.

Data returned: 201 Created status code and a JSON object with the track instance created and its fields.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found and 409 Conflict.

Retrieve track information:

HTTP Method: GET

Endpoint: /devices/:device_id/tracks/:tracks_id/

Authentication: Yes

Description: This method allows to retrieve the information of the track identified by the track ID and associated with the device identified by the device ID parameter. This information can only be accessed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id and track_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the track and all the information and fields.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Change track representation:

HTTP Method: PUT

Endpoint: /devices/:device_id/tracks/:tracks_id/

Authentication: Yes

Description: This method allows to change the representation of the track identified by the track ID and associated with the device identified by the device ID parameter. This change can only be done if the ID of the user authenticated corresponds with the ID of the device owner. With this method users can't change instances related to this track like the locations.

Required fields: device_id and track_id as URL parameter and name in the body request.

Data returned: 200 OK status code and a JSON object with the new representation of the track and all the information and fields.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Update track fields:

HTTP Method: PATCH

Endpoint: /devices/:device_id/tracks/:tracks_id/

Authentication: Yes

Description: This method allows to update the fields information of the track identified by the track ID and associated with the device identified by the device ID parameter. This method can only be done if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id and track_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the track and all the updated information and fields.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Delete track:

HTTP Method: DELETE

Endpoint: /devices/:device_id/tracks/:tracks_id/

Authentication: Yes

Description: This method allows to delete the track identified by the track ID and associated with the device identified by the device ID parameter. This delete can only be done if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id and track_id as URL parameter.

Data returned: 204 No Content status code.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Locations management

This group is formed by the petitions related with the actual location of a device (edit and retrieve the actual location) and the track locations resource (delete and retrieve information of a location). This two types of resources (actual location and track location) are associated with a device and use its owner permission.

Retrieve device actual location:

HTTP Method: GET

Endpoint: /devices/:device_id/actualLocation/

Authentication: Yes

Description: This method allows to retrieve the information of the actual location of the device identified by the ID parameter. This information can only be accessed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id as URL parameter.

Data returned: 200 OK status code and a JSON object with the actual location and all the information and fields.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

Change actual location representation:

HTTP Method: PUT

Endpoint: /devices/:device_id/actualLocation/

Authentication: Yes

Description: This method allows to change the representation of the actual location of the device identified by the ID parameter. This change can only be done if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id as URL parameter and the new location in GeoJSON[13] format.

Data returned: 200 OK status code and a JSON object with the new representation of the actual location and all the information and fields.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Create track location:

HTTP Method: POST

Endpoint: /devices/:device_id/tracks/:tracks_id/trackLocations/

Authentication: Yes

Description: This method allows the creation of a track location associated with the track identified by the track ID and associated with the device identified by the device ID parameter. This creation can only be performed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id and track_id as URL parameter and the location in GeoJSON format.

Data returned: 201 Created status code and a JSON object with the track location created.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found and 409 Conflict.

Change track location representation:

HTTP Method: PUT

Endpoint: /devices/:device_id/tracks/:tracks_id/trackLocations/:track_location_id/

Authentication: Yes

Description: This method allows the change the representation of a track location identified by the track location ID passed by parameter and associated with the track identified by the track ID and associated with the device identified by the device ID parameter. This change can only be performed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id, track_id and track_location_id as URL parameter and the location in GeoJSON[13] format.

Data returned: 200 OK status code and a JSON object with the new representation of the track and all the information and fields.

Errors handled: 400 Bad Request, 401 Unauthorized, 403 Forbidden and 404 Not Found.

Delete track location:

HTTP Method: DELETE

Endpoint: /devices/:device_id/tracks/:tracks_id/locations/:track_location_id/

Authentication: Yes

Description: This method allows to delete the track location identified by the location ID passed by parameter and associated with the track identified by the track ID and associated with the device identified by the device ID parameter. This delete can only be performed if the ID of the user authenticated corresponds with the ID of the device owner.

Required fields: device_id, track_id and track_location_id/ as URL parameter.

Data returned: 204 No Content status code.

Errors handled: 401 Unauthorized, 403 Forbidden and 404 Not Found.

4.1.2 Conceptual schema

The API conceptual model is represented by the following UML diagram:

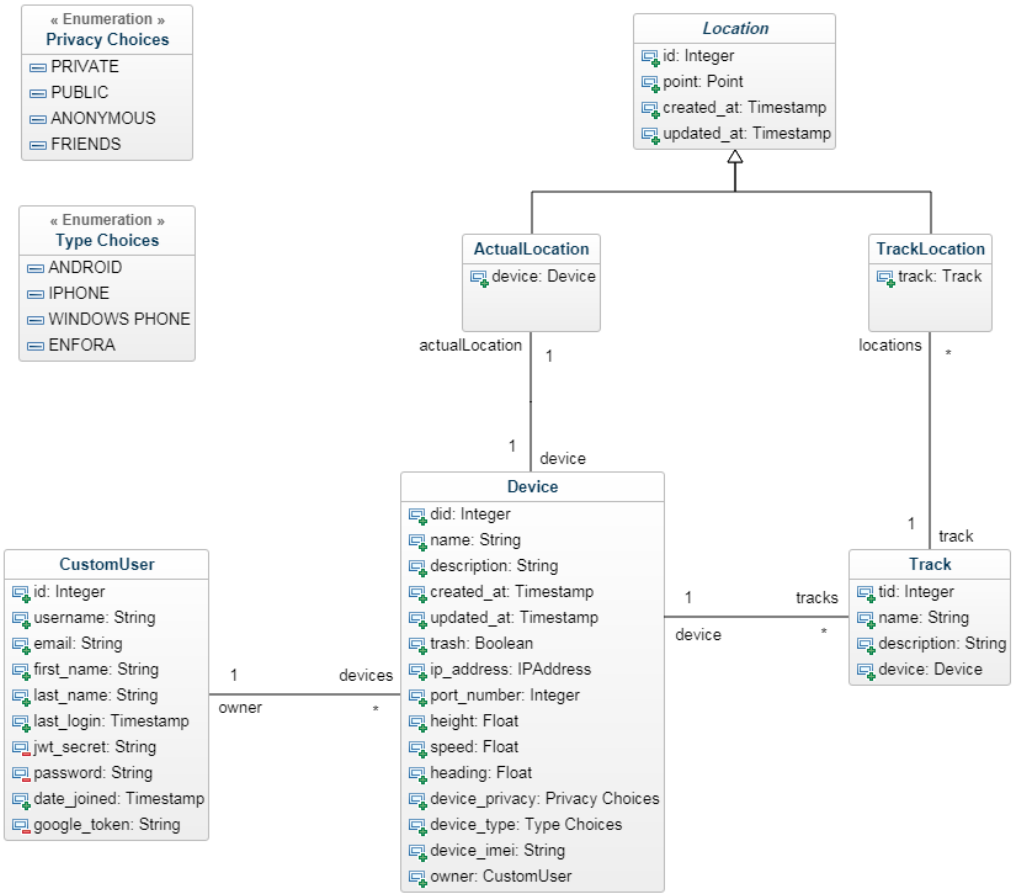

Figure 4.1: API conceptual model

4.2 Web specification

On this section the different functionalities that the web requires are described and defined. This functionalities are grouped according to their tasks. Also, every group has a brief explanation.

4.2.1 Functionalities

The different functionalities groups are: user management, tracks visualization and resources management. These functionalities are related to the API, as the Web will consume and use the API functionalities.

User management

This group is formed by all the functionalities related to the users. As the web isn't a social network it isn't needed to have a user interaction. In particular, the functionalities related to this group are:

Sign Up: The web offers a registration form with the fields that the user API resource accepts. Users can also sign up using the Google credentials.

Log In: Once registered users can log in to access to all the devices and tracks functionalities. If users have registered using Google they can log in with this credentials.

Edit profile: A user can change the email, first and last name.

Resources management

This group is formed by all the functionalities related with the device, tracks and locations resource management. This management references the functionalities of create, edit this resources. Also, the resources raw info can be seen on different views. So, this functionalities will be divided in devices, tracks and locations.

The device resource can be created with a form with the fields that the device API resource accepts. Besides this functionality, the user will be able to see all his devices with all the tracks associated and locations. Also, the device field, device type will be seen as a icon (Android, Apple, Windows or Fedora logo). This resource also allows the possibility to edit all the information (except the id) via a form.

The track resource can be created with a form with the fields that the track API resource accepts. When the track is created it will be associated directly to the correspondent device.

The track location resource can be created with a form with the fields that the track location API resource accepts. Every location created will be associated to the correspondent track.

Tracks visualization

This group is formed by the two most visible functionalities: track view on demand in a map and a map view with the tracks selected of the user logged devices. The map selected for the website is the OpenStreetMap and its technologies.

To view a track the user can navigate on the devices information and click on a track see it on the map. Once the user clicks on a track, it will be seen on the map joining all the points and forming a line on the map. In this way, this functionality can show the track locations in a more visual way.

Besides this first way to visualize a track there will be a second one. there will be a map view with all the devices and their tracks as filters. This map will be empty of tracks as a first state, and when a track is selected it will be painted on the map following the same pattern described bellow. As different tracks on the map can be seen, the color to paint each track will vary depending on their device.

4.2.2 Conceptual schema

The web conceptual model is represented by the following navigability diagram:

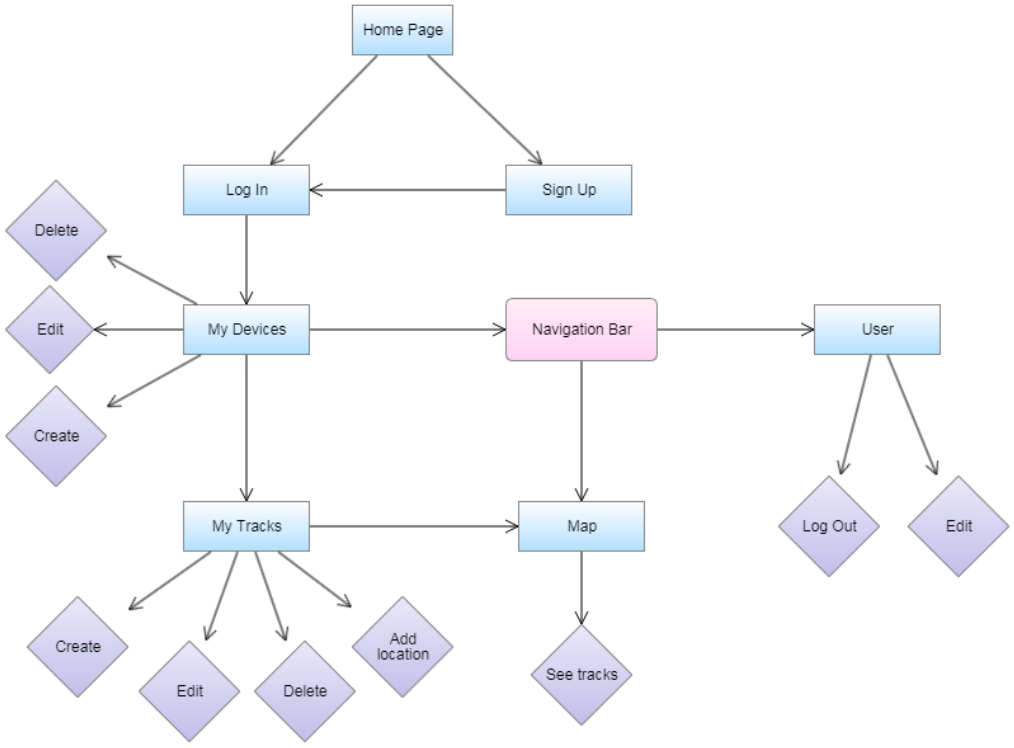

Figure 4.2: Web navigability diagram

Chapter 5

Design

This chapter presents and describes the system design on a physical and logic level and also, the technologies used.

5.1 System architecture

This section is focused on the system's architecture that the software needs to work. The system consists on a server with a database, the API and the Web application.

5.1.1 Server

The system has a unique server. This server is owned by inLab and offered as a service for its collaborators. InLab offers this service using the OpenNebula[23] technology. Concretely, inLab offers a Linux Virtual Machine to work as server. For this system the Virtual Machine selected is an Ubuntu 16.

The database is on the same server so a different connection to retrieve and manage the data isn't necessary.

5.1.2 API

The API is located on the server's side and it works as a service to interact with the server and its database. The communication of this API is via internet and the HTTP protocol.

5.1.3 Web application

This web application will interact with the server using the API. Also, a good server connection will be necessary to ensure the correct use.

5.1.4 Components communication

The users can access the system using the web application from their laptops and computers. This diagram shows the communication of all the components:

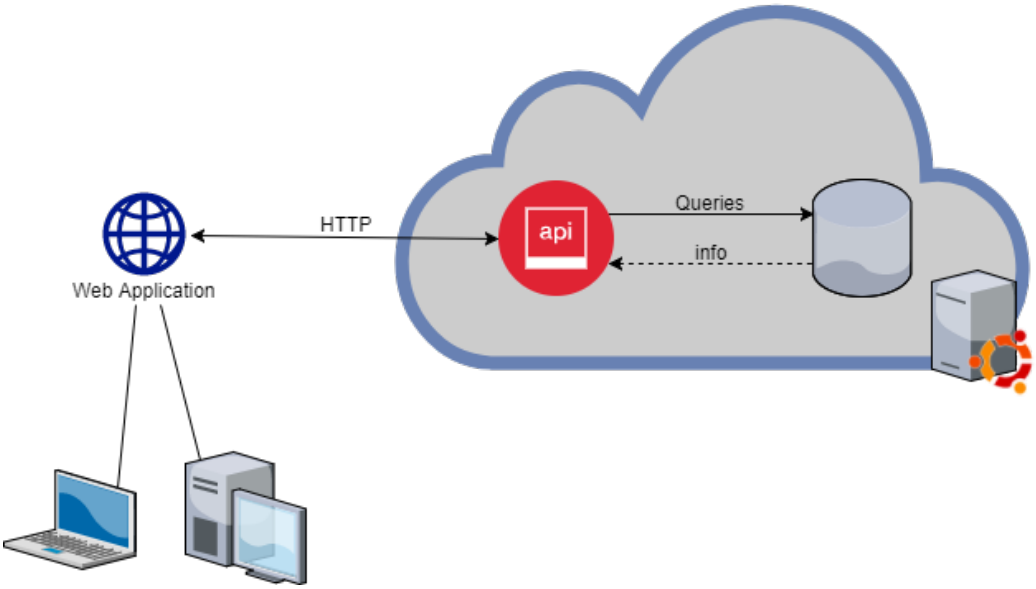

Figure 5.1: System components communication

5.2 API design

This section shows the technologies used on the API and the patterns applied. Also, the database technology is described on this section.

5.2.1 Technologies

The technologies selected for the API and the database were the Django REST Framework using the *django-rest-gis*[6] add-on and the PostgreSQL

with PostGIS. All these technologies allow to solve the geographical data problem.

Django REST

This framework is a powerful and flexible toolkit for building Web APIs. This framework extends from the Django Framework which is a Python ORM[1] Framework.

Django REST has some functionalities that are essential for our software:

Serialization: allow complex data such as querysets and model instances to be converted to native Python datatypes that can be easily rendered into JSON, XML or other content types. Serializers also provide deserialization allowing parsed data to be converted back into complex types, after first validating the incoming data.

Authentication polices: REST framework provides a number of authentication schemes out of the box, and also allows for the implementation of custom schemes. One of the authentication policies, and the selected one is the use of JWT.

Customizable: This framework is entirely customizable. It doesn't have restrictions on the use of all the features.

Client testing: REST framework includes a few helper classes that extend Django's existing test framework, and improve the support for making API requests. For example, any type of client requests and all their work flow can be easily tested.

Also, this framework has a lot of different add-ons and plugins to access to new features and functionalities. For example, the geographical add-on selected for this API (Django REST-gis).

Django-rest-gis add-on

Django-rest-gis is a geographical add-on for Django REST Framework. This add-on adds geographical fields, serializers, pagination and filters to the

[1]Object Relational Model

REST Framework. The most significant features for the software are the geographical fields that allow to manage and store all the geographical data and the geographical serializers that ensure the GeoJSON compatibility.

PostgreSQL with PostGIS

PostgreSQL is an object-relational database management system that uses the SQL language.

PostGIS is a spatial database extender for PostgreSQL. It adds support for geographic objects allowing location queries to be run in SQL.

5.2.2 Pattern applied

The patterns of Web API design book[19] have been followed for the API resource and URL design. Some of the guidelines followed are:

Nouns are good; Verbs are bad: To keep the URL simple and understandable the use of nouns instead of verbs is recommended.

Plural nouns and concrete names: The use of plural nouns makes the API more intuitive for the developers. Also, the use of concrete names to avoid resource abstraction.

Simply associations: It's common to have relations between resources. These relations are represented allowing a better system understanding.

Handling errors: It is very important to handle the errors that might happen on the API. It is recommended to follow and use the HTTP status codes as they are the standards for the developers.

For example, to access on a concrete track resource the correct URL will be */devices/:device_id/tracks/:track_id/*. First of all, the access to the device must be associated with the track (following simply associations guidelines seen above). Also plural nouns must be used for all the resources and concrete names to avoid misunderstandings.

A custom Model-View-Controller pattern has been followed for the API development.

This pattern consists on separating the data and business logic from the user interface and the module in charge of managing the actions and communications. This software architecture pattern is based on the ideas of reusing code and the separation of concepts to facilitate development and the subsequent maintenance. For these reasons there are three different components:

Model: Defines the data models with whom the application will work and it is the only class that has communication with the database. It is in charge of search, retrieve and store the data.

View: Shows the information on a format for a proper user interaction.

Controller: In charge of requesting the necessary data to the model, processing and sending it to the view. It is the key element for the use of this pattern, since it is the component that communicates the other two.

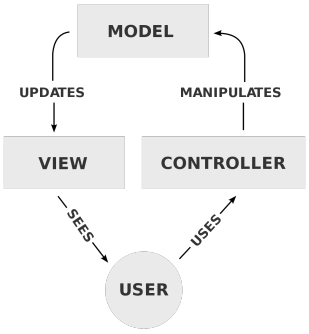

Figure 5.2: Model View Controller structure

5.2.3 Application to software

As it is mentioned before, a custom model view controller is used for the API development. This section describes the differences and adaptations that make it custom.

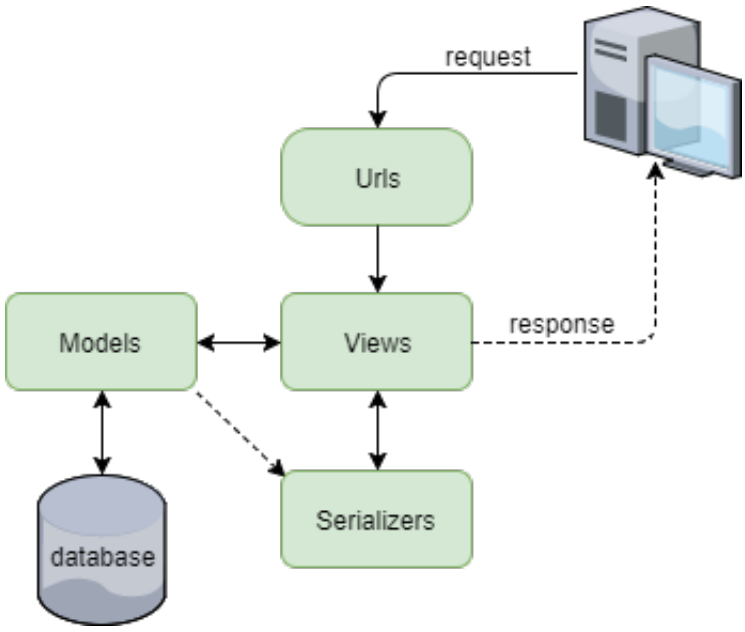

Figure 5.3: custom API Model View Controller structure

On the figure shown above the absence of controllers and new components called URLs and Serializers can be seen. As Django REST provides direct communication using models the existence of a controller component isn't required. These two new components are from Django REST, the URLs defines all the URL accepted and their views and the Serializers components are used to serialize and deserialize data.

An example for every component implemented will be shown for a better understanding of this custom pattern. To avoid complexity a simple request as the creation of a user is used as an example.

Before starting with the example, the next figure shows the structure followed to order the different packages. All the components, except URLs and models, are grouped on Python packages corresponding with a model name. For example, the devices package will have its views, serializers and another components as tests, signals, etc.

```
TooPath3/
|-- apps.py
|-- constants.py
|-- models.py
|-- urls.py
|-- utils.py
|-- wsgi.py
|-- __init__.py
|-- devices/
|   |-- permissions.py
|   |-- serializers.py
|   |-- signals.py
|   |-- tests.py
|   |-- views.py
|   |-- __init__.py
|--locations/
|
|-- migrations/
|   |-- 0001_initial.py
|   |-- 0002_auto_20171107_1657.py
|   |-- 0003_tracks.py
|   |-- 0004_auto_20171107_1921.py
|   |-- 0005_auto_20171108_1252.py
|   |-- 0006_auto_20171108_1628.py
|   |-- __init__.py
|-- settings/
|   |-- base.py
|   |-- local.py
|   |-- production.py
|   |-- __init__.py
|-- tracks/
|
|-- users/
```

Listing 1: API project structure

Also, there are two more packages, settings and migrations. In the set-

tings package there are the configuration files to use the software on local and deploy it to production. In the migrations package there are all the files to migrate the models into the database.

Models

There is a model class for every database table. Every class has predefined methods to create, edit, delete, get and filter the resource (these methods can be overwritten if needed). The existence of these methods make the use of a controller unnecessary for this framework.

The next code listing shows the user model class located on the *models.py* file. As it can be seen, this class extends from an abstract class called AbstractUser. The use of this abstract class allows to get an existent user model class and add the extra fields desired.

```
1        class CustomUser(AbstractUser):
2            jwt_secret = models.UUIDField(
3                 Token secret ,
4                help_text= Changing this will log out user everywhere ,
5                default=uuid.uuid4)
6            email = models.EmailField(null=False, unique=True)
```

Listing 2: CustomUser model

Urls

The Urls component is located on a unique file called *urls.py* (shown on the next code listing). This file has the function of declare all the accepted endpoints (seen on the chapter 4) of the API and its correspondent views. In the user creation example, a request on the URL */users/* will be made and *urls.py* will make a redirect to UserList view.

```
1         urlpatterns = [
2             url(r ^devices/(?P<d_pk>[0-9]+)/tracks/(?P<t_pk>[0-9]+)
                  /locations/(?P<l_pk>[0-9]+)/$ ,
                  locations_views.TrackLocationDetail.as_view()),
3             url(r ^devices/(?P<d_pk>[0-9]+)/tracks/(?P<t_pk>[0-9]+)
                  /locations/$ ,
                  locations_views.TrackLocationList.as_view()),
4             url(r ^devices/(?P<d_pk>[0-9]+)/tracks/(?P<t_pk>[0-9]+)/$ ,
                  tracks_views.TrackDetail.as_view()),
5             url(r ^devices/(?P<d_pk>[0-9]+)/tracks/$ ,
                  tracks_views.TrackList.as_view()),
6             url(r ^devices/(?P<d_pk>[0-9]+)/actualLocation/$ ,
                  locations_views.DeviceActualLocation.as_view()),
7             url(r ^devices/(?P<d_pk>[0-9]+)/$ ,
                  devices_views.DeviceDetail.as_view()),
8             url(r ^devices/$ , devices_views.DeviceList.as_view()),
9             url(r ^users/(?P<u_pk>[0-9]+)/$ ,
                  users_views.UserDetail.as_view()),
10            url(r ^users/$ , users_views.UserList.as_view()),
11            url(r ^login/$ , users_views.UserLogin.as_view()),
12            url(r ^api-token-refresh/$ , refresh_jwt_token),
13            url(r ^api-token-verify/$ , verify_jwt_token),
14        ]
```

Listing 3: API urls

Views

This component manages the requests sent through *urls.py* and returns a response in JSON format to the requests author. On every view the different request methods accepted are defined (GET, POST, PUT, PATCH and DELETE) as functions. Each one of these functions has a common fact, the use of serializers to validate and serialize/deserialize the request data or a model instance.

The next code listing shows the behaviour of the UserList view on a POST request.

```
1         class UserList(APIView):
2             def post(self, request):
3                 serializer = CustomUserSerializer(data=request.data)
4                 if serializer.is_valid():
5                     user_created = serializer.save()
6                     return Response(data=PublicCustomUserSerializer(
                          ↪  instance=user_created).data,
                          ↪  status=HTTP_201_CREATED)
7                 return Response(data=serializer.errors,
                   ↪  status=HTTP_400_BAD_REQUEST)
```

Listing 4: Create user view

On this POST method of UserList view the request data is serialized and after the correct validation (if it's not a 400 Bad Request status is returned with its errors) the user resource is created and returned with a 201 Created status.

Serializers

This is the last component of this custom pattern. This component is used to serialize and deserialize data from a request or form a model instance. Also, a concrete model can be related to a serializer and its update and create functions can be used to update and create an instance of the model related.

On the previous view, the serializer CustomUserSerializer is used to serialize the request data and after save it and create a new user instance with this data. The next code listing shows the implementation of this serializer:

```
class CustomUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomUser
        fields =  __all__

    def validate(self, data):
        if self.partial is True:
            if  pk  in data or  id  in data or  email  in data or
             ↪   username  in data:
                raise serializers.ValidationError(
                 ↪  DEFAULT_ERROR_MESSAGES[ invalid_patch ])
            if bool(data) is False or len(self.initial_data) !=
             ↪  len(data):
                raise serializers.ValidationError(
                 ↪  DEFAULT_ERROR_MESSAGES[
                 ↪   patch_device_fields_required ])
        if  password  in data:
            data[ password ] = make_password(data[ password ])
        return data
```

Listing 5: CustomUser serializer

The previous code of the serializer has the model related on the Meta and the fields that want to be serialized and deserialized. Also, the validate function is the only function of the serializer because the create and update are already implemented from the ModelSerializer class.

5.3 Database design

The database design for the server is described in this section.

As mentioned before, PostgreSQL is the technology selected for the database. On a common database design the tables, restrictions and relations must be previously designed and then applied using the SQL language. But this common design isn't necessary because the Django REST Framework has a migrations system.

This migration system allows to pass the model class declared to the database. For example, in our system there exist the models of User, Device, TrackLocation, ActualLocation and Track. These models are converted to database tables with their restrictions and relations when the migration system are applied.

Due to this system it isn't needed to design database table diagram. This diagram can also be generated automatically using the framework, the result can be seen on the Appendix A.

5.4 Web design

This section shows the technologies used on the Web and the patterns applied. Also, the mockups designed for the web are shown on this section.

5.4.1 Technologies

The technology selected for the web development is the version 5 of Angular[2] Framework which is the last stable version. This framework works with TypeScript language which is also briefly described above. Also, the Bootstrap[5] framework is used to make the web front-end.

Angular

Angular is a framework and platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.

Angular is the framework selected for the web development because of the following reasons:

Leaflet plugin: The existence of a library called *ngx-leaflet*[3] allows to easily integrate the OpenStreetMap technology on the web. This library is installed as module and adds all the directives and components to make use of the maps of OpenStreetMap.

Easy API integration: The framework makes it easy to integrate the API as a service to use on the web.

Good documentation: There is a lot of official documentation about the framework and all its uses. This fact allows to quickly solve possible doubts and bugs when developing the web.

Experience and knowledge: The experience and knowledge acquired from previous projects and university projects make the web development phase easier and faster.

Leading technology: Angular is a very popular framework, with a newly stable version and constant updates, which makes it a leading technology.

TypeScript

TypeScript is a free and open-source typed superset of JavaScript that compiles to plain JavaScript. TypeScript is pure object oriented with classes, interfaces and statically typed like Java.

Bootstrap

Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components. Unlike many web frameworks, it concerns itself with front-end development only.

5.4.2 Mockups

It was a short phase of mockups design before starting the web development. These mockups are simply images we create that show what the website will look like without needing to dive straight into coding. These mockups are just flat images that cannot be interacted with, but look like a screenshot of a website page.

The mockups are used because the process of laying out a mockup is much faster than coding a web. Creating this image mockup allows to try out different design elements, layouts, colors, and fonts quickly to get a better idea of what design is preferred before devoting to code.

The mockups of the project web can be seen on the Appendix C.

5.4.3 Pattern applied

Angular tries to stay pattern-agnostic and can be used with conventional MV* patterns, it was designed with reactive programming in mind and it really shines when used with reactive data flow patterns.

As the framework is pattern-agnostic the web was developed following the official documentation and using these framework object types:

Component: Component decorator allows to mark a class as an Angular component and provide additional metadata that determines how the component should be processed, instantiated and used at run time.

Injectable: It is a type of object applied on the Dependency Injection (DI). DI is a way to create objects that depend upon other objects.

App module: This is the main piece of the framework. All the modules imported and declared, web routes, providers, etc are declared on this object.

TypeScript class: There are object classes based on the TypeScript language.

5.4.4 Application to software

The different component types described previously are shown next using examples of the web. These examples corresponds with the web log in functionality. First of all, a common TypeScript object class is showed:

```
1        export class User {
2          first_name: string;
3          last_name: string;
4          username: string;
5          email: string;
6          password: string;
7        }
```

Listing 6: User TypeScript object class

Next, the App Module file is shown:

```
import ...
const appRoutes: Routes = [
  {path:  signUp , component: AppSignUpComponent},
  {path:  logIn , component: AppLogInComponent},
  ...
];
@NgModule({
  declarations: [
    AppComponent,
    AppLogInComponent,
    ...
  ],
  imports: [
    NgbModule.forRoot(),
    RouterModule.forRoot(appRoutes),
    ...
  ],
  providers: [
    HttpService,
    LogInService,
    ...
    ],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Listing 7: App Module file

The App Module file has three important parts. First, the constant called *appRoutes* is in charge of connecting every valid path with its Component. Second, the declarations, imports and providers arrays are in charge of declare the custom Components implemented for the web, the Components imported and the injectable Providers respectively. Third, *@NgModule* wrap all these arrays as a global module to be accessible for all the files.

Next, the log in Component is showed:

```
1   import ...
2
3   @Component({
4     selector:  app-app-log-in ,
5     templateUrl:  ./log-in.component.html ,
6     styleUrls: [ ./log-in.component.css ]
7   })
8   export class AppLogInComponent implements OnInit {
9
10    user: User;
11
12    constructor(private _logInService: LogInService, private
        _router: Router) {
13    }
14
15    ngOnInit() {
16      this.user = new User();
17    }
18
19    async sendLogIn() {
20      try {
21        await this._logInService.postLogin(this.user);
22        this._router.navigate([  ]);
23      } catch (error) {
24        console.log(error);
25      }
26    }
27  }
```

Listing 8: Log in Component

The *@Component* variable defines the template and the styles files for this Component. The Injectable are passed as parameters on the constructor function. The constructor of the example has two Injectable objects, the *Router* which is explained before and the *LogInService* which is an object

that acts as an intern service and is shown on the next listing. The *ngOnInit* function is called on the Component initialization and allows to initialize and create objects needed for the correct Component behaviour. Last, the asynchronous function, *sendLogIn* is called when a user click on the log in button on the web: This function calls the *LogInService* and perform the log in on the API.

Next, the log in service Injectable is showed:

```
import ...

@Injectable()
export class LogInService {

  constructor(private _logInApiService: LogInApiService) {
  }

  async postLogin(user: User) {
    let response = await this._logInApiService.postLogIn(user);
    if (response.status === 200) {
      localStorage.setItem( currentUser , JSON.stringify({
        id: response.json().id,
        email: response.json().email,
        first_name: response.json().first_name,
        last_name: response.json().last_name,
        username: response.json().username,
        token: response.json()
      }));
    }
  }
}
```

Listing 9: Log in service Injectable

The *@Injectable* variable makes this object Injectable. It has a constructor function with another object Injectable on the parameters. The asynchronous function *postLogin* call the API using the *logInApiService*.

Chapter 6

Development

This chapter describes how the process and the implementation have been in the system development.

The description is divided for every Sprint mentioned on the project planning section. Every task and functionalities mentioned in the project planning section and specification chapter are implemented on these Sprints. Some Sprints had difficulties that are explained next.

6.1 Sprint 0 - Project analysis and design

Duration: 4 weeks (18/04/2017 - 12/05/2017)

The methodologies study and requirement analysis were done during this Sprint. All the study and analysis was performed by the product owner, project manager and developer on several meetings. Also, the software architecture was designed.

6.2 Sprint 1 - Project kickoff and environment setup

Duration: 3 weeks (15/05/2017 - 02/06/2017)

All the environment was prepared and configured on this Sprint. All the software needed and their dependencies were installed: Python, Django, Django REST, django-rest-gis, PostgreSQL, PostGIS, PyCharm IDE, Vagrant, etc. There was a dependency problem with the Django REST-gis installation for windows which ended up in a small waste of time. This problem is described on the section 10.2.

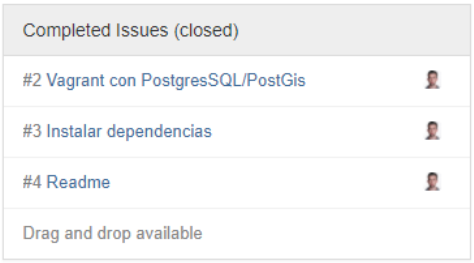

Figure 6.1: Sprint 1 completed tasks

Also, a concept test was done to ensure the correct functionality of the framework, database and server. This concept test was very important because future configuration and dependencies might end up in an important waste of time. Therefore, the critical functionalities of the technologies were focused on this concept test.

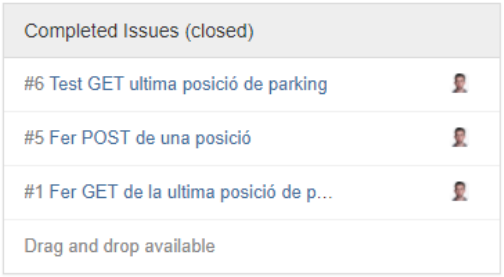

Figure 6.2: Sprint 2 completed tasks

6.3 Sprint 2 - CarGuard functionalities

Duration: 4 weeks (05/06/2017 - 30/06/2017)

The main functionality for CarGuard project was implemented in this Sprint. This functionality is the actual location of a car. This functionality requires the Device and Location models which were implemented and migrated to the database. Location model wasn't abstract on this point because

the functionality didn't require it.

A succesful demo and meeting took place at the end of this Sprint with the CarGuard client, to review and ensure the achievement of the functionalities.

The TDD methodology was applied since this Sprint. The two previous Sprints weren't following this methodology because there wasn't any code implementation.

6.4 Sprint 3 - Performance and CarGuard functionalities

Duration: 4 weeks (03/07/2017 - 27/07/2017)

This Sprint main focus was finishing the functionalities needed for the CarGuard project. CarGuard needed to retrieve and store the car information, so the GET, PUT, POST methods must be implemented. These functionalities didn't require any new model implementation, just a new view.

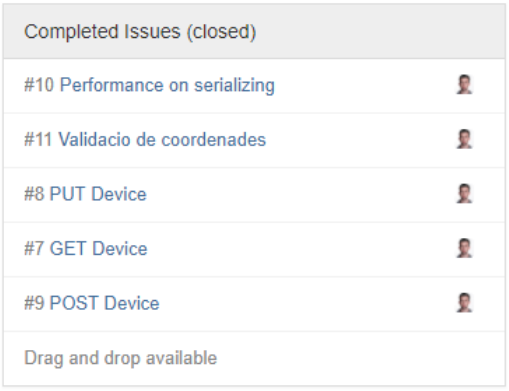

Figure 6.3: Sprint 3 completed tasks

The location storage was implemented on the previous Sprint, but this storage didn't validate the coordinates. This validation was implemented using the Serializer component of Django REST.

There was another succesful demo with the CarGuard client (SEAT) on the end of this Sprint. The development for the CarGuard functionalities finalized in this Sprint, and after, the functionalities were specified by the Product Owner.

6.5 Sprint 4 - Users and permissions

Duration: 3 weeks (12/09/2017 - 29/09/2017)

The main objective of this Sprint was to implement the ownership permissions for the device resource. This type of permission requires of the existence of a user resource and the log in feature. So, the user model class was implemented and migrated before start with the permissions implementation. The device model class was also migrated again to add the owner attribute.

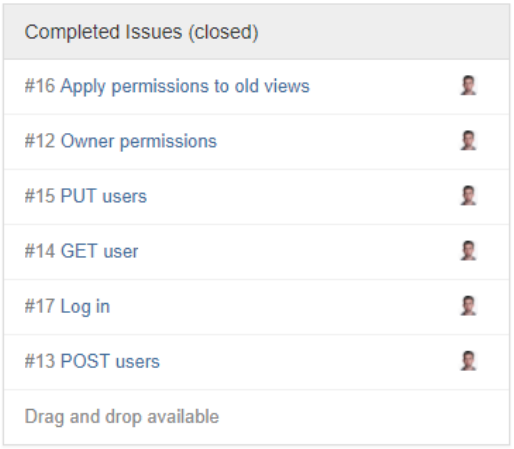

Figure 6.4: Sprint 4 completed tasks

The user model of the API is called CustomUser and as it is mentioned in the previous section 5.2.3 it extends the AbstractUser model class from Django. It was decided to extend this model because the framework has a functional user resource model and it would be a waste of time and effort to implement what has already been implemented. The GET, POST and PUT methods for the user resource view were also implemented.

The log in feature was implemented after the user model and view implementation. This feature allows to identify the user on every request done with a simple token generated from the framework.

The Django REST documentation was followed to implement the permissions of device ownership. These permissions allow to protect the access and manage of the device resource. For example, if a user makes a PUT request for a device that doesn't own the request can't be performed and it returns the 403 Forbidden status code. Also, the permissions were added to the views implemented on the previous Sprints.

6.6 Sprint 5 - Social log in and first Track model

Duration: 3 weeks (02/10/2017 - 20/10/2017)

This Sprint was focused on implementing the social log in feature via Google and the addition of JWT for the user request authentication. There was some difficulties with the social log in feature which delayed its implementation. These difficulties are described on the section 10.2.

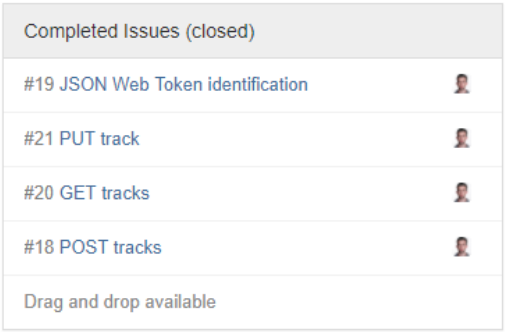

Figure 6.5: Sprint 5 completed tasks

The JWT authentication was implemented using the *django-rest-framework-jwt*[4] library for Django REST framework. These tokens allow to authenticate the user that has made the request. Also, every user created has a secret unique token which identifies the user and generates the tokens sent with the request. The tokens sent with the request have a determined time validation for security reasons. Also, if the tokens are stolen changing the secret token will automatically leave all the previous tokens generated invalid. So, the JWT give a good security level for the API.

Also, the track model class was implemented and migrated on this Sprint. The GET, POST and PUT methods for the tracks resource were also implemented.

6.7 Sprint 6 - Tracks implementation

Duration: 3 weeks (23/10/2017 - 10/11/2017)

The Location model implemented on the Sprint 2 was made abstract to apply the extends model feature. The ActualLocation and TrackLocation model were implemented and migrated to the database. These models extend from the Location model. The POST and PUT methods for the track

location resource were also implemented.

Since all the models were implemented their relations were also integrated on the view responses. For example, the response for the GET method on the devices view returns all the devices with its tracks and tracks locations on the same JSON. Having relations on the responses allows to maintain the models relations on it.

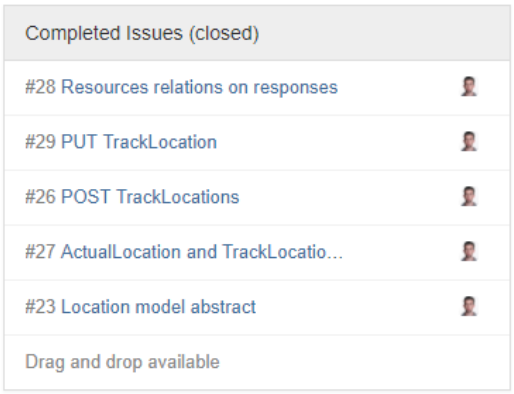

Figure 6.6: Sprint 6 completed tasks

6.8 Sprint 7 - API finalization and Web kickoff

Duration: 3 weeks (13/11/2017 - 01/12/2017)

The API implementation was finished on this Sprint. The PATCH and DELETE methods for all the resources views were implemented on this Sprint. With these methods, every resource can be retrieved, updated, created and deleted.

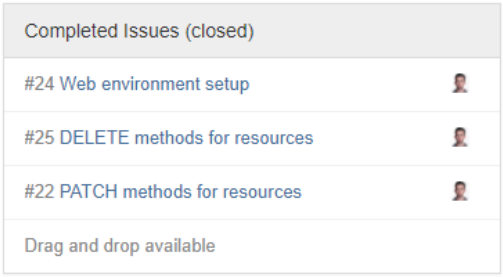

Figure 6.7: Sprint 7 completed tasks

As the API was finished, the tools and the framework with its dependencies needed for the web development were installed. The mockups and the web conceptual schema were designed on this Sprint.

Also, the license study and analysis were performed on this Sprint. This study can be seen on the section 8.2.

6.9 Sprint 8 - Web implementation

Duration: 3 weeks (04/11/2017 - 22/12/2017)

The major part of the web implementation was done on this Sprint. The milestone desired was to finish the web development on this Sprint but the map view couldn't be implemented.

The web views implemented on this Sprint were: Create, edit and delete view for the devices, tracks and users resources, the add track location view, my devices view and the log in and sign up views.

Also, the social log in feature with Google was implemented at the same time as the log in and sign up web view.

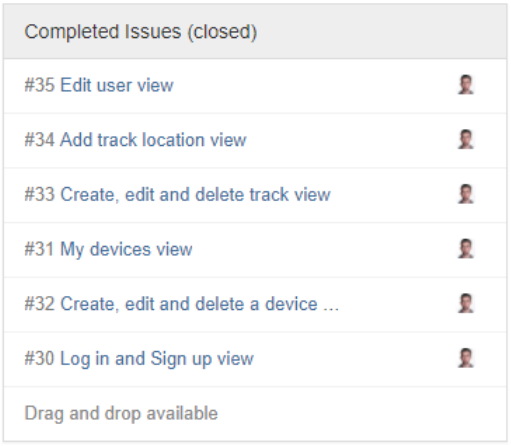

Figure 6.8: Sprint 8 completed tasks

6.10 Sprint 9 - Extra Sprint

Duration: 2 weeks (8/01/2018 - 22/01/2018)

The main focus of this extra Sprint was the map view implementation for the web. This feature wasn't started on the previous Sprint but it is necessary for demonstrating that the routes are stored correctly on the database and that they can be retrieved and used on a map.

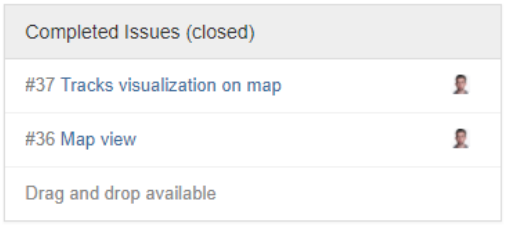

Figure 6.9: Sprint 9 completed tasks

6.11 Closing phase

Duration: 2 weeks (27/10/2017 - 05/01/2018)

The API specification was written on this phase. The project report was written on this phase analyzing the whole project to make some conclusions. Also, the project slides for the presentation were prepared and done during this phase.

Chapter 7

Validation

Different ways of validation are described on this chapter. There are three sections; tests, continuous integration and product owner. The tests section describes more about TDD and how is it used for the system validation. The continuous integration section describes how the correct functionality of the software is tested and validated when it is deployed or pushed to the repository. And last, the validation of the product owner is described on its homonym section.

7.1 Tests

The TDD methodology is used for the tests as it is mentioned on the section 1.6. These tests are used to prove the correct functionality and behaviour of the API views. It is possible to test if the JSON data is returned, if the HTTP status code is returned and corresponds with the expected code, if the database instances are created and updated and other functionalities.

A test writing convention for the test function names has been followed according to team agreement. This allows to give information on the test function name to see what is testing on the function, and helps the users interested to understand the code and its behaviour. The test function names follow the next convention:

```
test_return_[expected-return]_when_[view-or-method-performed]
```

Next, a test class from the API is showed as an example:

```
1        class GetUserCase(APITestCase):
2            def setUp(self):
3                self.client = APIClient()
4                self.user =
                  ↪  create_user_with_email(email= user@gmail.com )
5                self.token = generate_token_for_user(user=self.user)
6                self.client.credentials(HTTP_AUTHORIZATION= JWT   +
                  ↪  self.token)
7
8            def test_return_200_status_when_get_user_is_done(self):
9                response = self.client.get(path= /users/  +
                  ↪  str(self.user.pk) +  / )
10               self.assertEqual(HTTP_200_OK, response.status_code)
```

Listing 10: Fragment of GetUserCase test class

This example shows a *setUp* method and the test method. The main objective of the *setUp* method is to prepare the context and state of the database and python. The *setUp* method of the example creates a user and its token (added to the header credentials). The test method on the example checks if the HTTP status code returned is 200. This validation is done using the *APIClient* testing class from the framework. This class allows to send requests to the API with its data, path, etc.

7.2 Continuous integration

The open-source integration software called Jenkins[18] is used to apply continuous integration[1] on the deploys. This software allows to run all the tests and others programmed tasks automatically when new code is pushed to the repository. The Jenkins for the project follows this behaviour for every new push:

[1]development practice that requires developers to integrate code into a shared repository several times a day.

1. Run all the tests.
   (a) If any test fails, stop the work flow.
2. Deploy the new code to the virtual machine.
3. Communicate on the Slack channel the result of the deploy and tests (success or fail).

Next, a pipeline example of the Jenkins tool is shown:

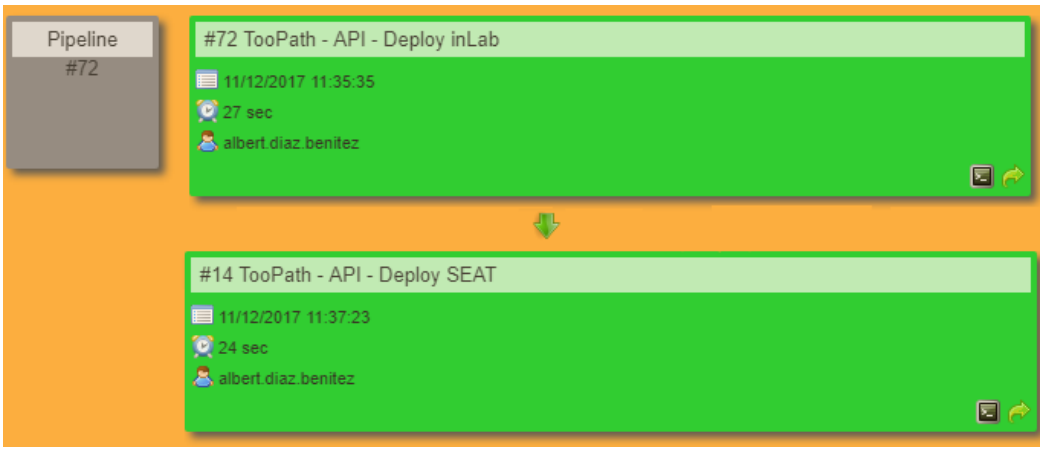

Figure 7.1: Jenkins pipeline tool

7.3 Product owner

Another validation route for this project were the weekly meetings with the product owner, checking on his own if the agreed tasks and features for that Sprint were completed. If that was not the case, these features and tasks were kept up for the next Sprint or relocated on another future Sprint if there were some concrete reasons.

Chapter 8

Software publishing

The guidelines and methods to publish the API are described on this chapter. Also, because this is an open-source software, the study and analysis of the open source license is described here.

8.1 API publication

The API is published on the GitHub repository. It is possible to use, download and view the API from this repository. Also, the open source license is visible and readable from this repository.

SemVer[30] is used for the version number of the releases. The version number follows the pattern *MAJOR.MINOR.PATCH*:

MAJOR: version when you make incompatible API changes.

MINOR: version when a functionality is added in a backwards-compatible manner.

PATCH: version when you make backwards-compatible bug fixes.

8.2 Open-source license study and analysis

Open-source licenses are licenses that comply with the Open Source Definition — in brief, they allow software to be freely used, modified, and shared.

To be approved by the Open Source Initiative (OSI), a license must go through the Open Source Initiative's license review process.

First of all, all the technologies used and its dependencies are open source or free software. This is very important because an open source software can't be developed with private software. There are exceptions like the IDEs used or the operative system of the PC. All these exceptions were previously installed and the project could be implemented with a GNU/Linux based system and the free versions of the IDEs.

A dependencies study was performed before selecting the license for the software. The next table shows all the development dependencies and the software used for the TooPath API and the web with their licenses:

Dependency	**License**
Django	Own (Free)
Django REST Framework	BSD 2-clause "Simplified"
Gunicorn	MIT
PyJWT	MIT
appdirs	MIT
django-cors-headers	MIT
django-rest-framework-gis	MIT
djangorestframework-jwt	MIT
packaging	Apache 2.0 or BSD 2-clause "Simplified"
psycopg2	LGPL
pyparsing	MIT
pytz	MIT
six	MIT
Angular	MIT
Bootstrap 4	MIT
PostreSQL	PostgreSQL license
PostGIS	GPL-2.0
Python	PSF
Typescript	Apache 2.0
GDAL	MIT
Git	GNU-2.0

Table 8.1: Development software and dependencies licenses

All the software and its dependencies have open source licenses which don't cause any problem. Different possibilities for the open source license of the API are studied.

It is important to know the interests of the author company or entity and why the software should be open source in order to know which license choose. For example, the company can be concerned about patents which make an Apache License 2.0 the most appropriate. Instead, if the company is concerned about sharing improvements the GNU GPL 3.0 is the most appropriate. Also it is possible to just want the software to be used as the users wanted as long as the attribution is provided. On the last case the MIT license is the most appropriate.

The main reason to make the software open source for the inLab company is to give free access to the software but always provide the author attribution. Also, inLab isn't concerned about software patents or coerce the distribution of improvements. In consequence, the MIT license was selected. This license has the following contents:

```
Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

Listing 11: MIT license template

Chapter 9

Integration of knowledge

The knowledge obtained from the bachelor degree is described on this chapter. Also, the technical competences are described to justify their grade of achievement.

9.1 Integration of knowledge from the bachelor degree

The subjects of the bachelor degree mentioned bellow are extracted from the university website[15].

Web Applications and Services (ASW): Developing of an API and a web that will consume it. Design of the architecture of a web service.

Software Architecture (AS): Programming patterns that can be applied like: Singleton and Transactional Pattern among others. Also, object oriented patterns and architecture design.

Database Design (DBD): Design of relational databases, schema and tables.

Requirements Engineering (ER): Evaluation of budget and context of software project.

Software Project Management (GPS): Concepts and experience of Agile methodology and a brief introduction of TDD.

Software Engineering Project (PES): Software development experience from a real project.

Knowledge Engineering and Distributed Intelligent Systems (ECSDI): Brief introduction to Python language and first project developed with this language.

Operating Systems Administration (ASO): Linux management.

9.2 Valoration of technical competences of the project

Finally, the technical competences of the specialty will be justified. Next, the table shows the grade of achievement.

	CES1.1	CES1.2	CES1.3	CES1.4	CES1.5	CES1.6	CES1.7	CES1.8
In profundity	X	X			X		X	X
Enough			X					
A little				X		X		

CES1.1 - To develop, maintain and evaluate complex and/or critical software systems and services

The project consists on developing a geolocation API and a web. Also, it will include authentication and secure permissions for user management and a complex model and database integration. This competence will achieve the correspondent grade with the use of good developing patterns and a qualitative management from the development team of the project to ensure the completion of the API and web.

CES1.2 - To solve integration problems in function of the strategies, standards and available technologies

The main integration problem of this project is the use and management of geolocation raw data. This will be solved with the use of the Django REST Framework and the PostGIS add on. There are other integration

problems like the API integration for the web and the server integration with the releases of the software. These problems will be solved following good patterns of web design and implementation and the use of tools like Jenkins to control the integration on the server. Ultimately, this competence will achieve the correspondent grade.

CES1.3 - To identify, evaluate and manage potential risks related to software building which could arise.

A complex software like this project can have potential risks like integration problems with the actual technologies, failures on the specification and proper functionalities. It is also important to consider that this software will be open-source, this fact is another possible risk as we need to use free software and tools to develop this project. To avoid these risks we will have constant meetings and monitoring of the software state to prevent and solve these risks. Also, all the technologies of the project analysis will be specified to ensure that the open-source milestone is possible.

CES1.4 - To develop, maintain and evaluate distributed services and applications with network support

This competence is achieved with the minimum grade due to the minimum use of network support. All the network environment like the server will be developed with the Project Manager help and supervision to ensure the good setup. In fact, the developer will only need few knowledge of the server functioning.

CES1.5 - To specify, design, implement and evaluate databases

This project needs a specific database with a design that can accomplish the API restrictions and store/manage geolocation data. This competence will achieve the correspondent grade because of the use of database schema, it will use add ons (like PostGIS) that will assure the store/manage of geolocation data. The database will be also designed following the knowledge acquired in the degree and following the API models restrictions.

CES1.6 - To administrate databases

The database is already designed and functional and will need minimum administration. Due to this, the selected grade is "a little" and it will be accomplished with a good use of the migration system from Django and possible backups in case of failure.

CES 1.7 -To control the quality and design tests in the software production

This project will be a full quality software with a full final version release. It will also have integration and unit testing of the software. This competence is achieved with the maximum grade because the TDD methodology will be followed for the project development, and this will ensure a quality software with minimum possible bugs.

CES2.1 - To define and manage the requirements of a software system.

The project will be defined on the first weeks through team meetings and specifying the full functionalities and scope. The possible requirements will be studied and managed with the Project Manager to ensure the good development of the project and that the analysis is well done.

Chapter 10

Project obstacles

The problems and obstacles of this project are described on this chapter. Every obstacle has its section. All this obstacles are briefly mentioned on previous. Every obstacle has a homonym section and an explanation with the cause and the solution of the obstacle or problem.

10.1 GDAL installation

This obstacle appeared the Sprint 1. There was an obstacle with the installation of GDAL which is a dependency for the *django-rest-framework-gis* and PostGIS use. The geospatial features can't be used without GDAL so it was an obstacle that needed to be solved with maximum priority.

This obstacle was solved on the same Sprint but spending more time with the environment setup which wasn't planned. The problem was caused by a bit version incompatibility between Python and OSGeo4W[1].

10.2 Google log in feature

This obstacle appeared on the Sprint 5. This feature was planned to be implemented on the Sprint 5 but it wasn't as it is mentioned on the section 6.6. This feature was estimated as a simple feature to implement but it wasn't.

[1]binary distribution for Windows to install GDAL

Initially, some libraries provided by the framework were studied to implement this feature. This libraries were focused on all the social OAuth[2] possibilities as Facebook, Twitter, Google and others. These libraries were rejected for two reasons:

OAuth was unnecessary: The API uses JWT, the use of OAuth system was unnecessary.

Unnecessary features: These libraries have a lot of features and social platform integration which aren't needed to implement a Google log in.

The Google Identity Platform[14] was studied and selected to implement this feature. This secure authentication system allows to implement this feature without dependencies and also to implement it easily on the web. Finally, the Google log in feature was implemented during the Sprint 8 as it is mentioned on the section 6.9.

[2]open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access but without giving them the passwords

Chapter 11

Future work

The possible features and functionalities that can be implemented for the API are described on this chapter. Also, possible improvements and new features of the web are mentioned.

11.1 Tracks algorithm

The tracks algorithm implementation is the main API feature to be implemented in the future. The algorithm will allow the API to know when a track is finished and also, if a new track needs to be created. The main idea that the algorithm helps to check if the distance between the last position sent and the last position saved is possible within the time difference between these positions.

For the algorithm implementation the fact that just one track can be on open state will be considered as a restriction. These will be the simple rules for the algorithm:

Track opened:

- New track location sent and correct distance-time relation: track location added to track.
- New track location sent and incorrect distance-time relation: track closed and new track created with the track location sent as first location.

Track closed:

- New track location sent: track created and track location added to track.

This is just a basic idea of the objectives and behaviour of the algorithm. If this algorithm is implemented a correct specifications of the rules and cases and minimum two more Sprints will be needed.

11.2 Browsable API

A browsable API is an API that allows developers to figure out how to use it without significant documentation. Django REST Framework supports generating human-friendly output for each resource when the format is requested. A possible API improvement is to make it browsable using the tools disposed by the current framework.

11.3 Web improvements

Since the web has been developed as a demo to show the API functionalities, it can be improved. The following list shows some possible improvements for the web:

Implement the password forgotten functionality.

Improve the visualization of resources.

Improve the design.

Add more user fields like the profile image.

Improve the form validation and errors.

Implement the CSV download functionality with the tracks and devices info.

Chapter 12

Conclusions

The conclusions of the project follow the goals mentioned on the section 1.5.1. These goals are analyzed next to see if the project has finished correctly and if all of them have been completed.

The main goal of the project is fully accomplished. TooPath is operative and accessible from the GitHub repository and offered as open-source software under the MIT license. All the requirements analyzed on the chapter 3 have been considered and completed. The web developed for the project presentation is operative and completed. The main goal was decomposed on three different goals which were:

Update TooPath with the current geolocation technologies. **Achieved**.

Make a website for the direct use of the API functionalities. **Achieved**.

Offer this API as open-source software. **Achieved**.

For the technical project goals:

Design a geolocation API model to store and process. **Achieved**, this API model has been designed and specified as it is shown on the chapters 4 and 5.

Develop a user and permissions system. **Achieved**, the user and permissions policy has been implemented on the Sprint 3.

Develop the functionalities for the tracking system: add, update and delete location and tracks. **Achieved**, the functionalities for the tracks have been implemented as it can be seen on the chapter 6.

Make the software protected and secured. **Achieved**, the software uses JWT which make it secure.

Develop a web that will consume the API. **Achieved**, the web has been implemented and will be shown on the project presentation.

TooPath contributes to society as it is offered as open-source software. TooPath offers a geolocation tracking system fully operative and the freedom to modify it as the user chooses to. Secondly, the web developed on this project can be improved and offered at user level to use TooPath.

To close this chapter I have decided to include my final thoughts on the project. First, this project has allowed me to improve my skills as a software developer. The technologies used for this project were new for me and it has been an interesting opportunity to use them. Second, this project is focused on a social technological theme as the open source software is, which is of great interest for me. Lastly, this project has given me the opportunity to see the requirements and the magnitude of a software project.

References

[1] *Agile*. URL: http://agilemethodology.org/.

[2] *Angular Framework*. URL: https://angular.io/.

[3] Asymmetrik. *Leaflet library*. URL: https://github.com/Asymmetrik/ngx-leaflet.

[4] Blimp. *django-rest-framework-jwt library*. URL: https://github.com/GetBlimp/django-rest-framework-jwt.

[5] *Bootsrap library*. URL: https://getbootstrap.com/.

[6] Federico Capoano. *django-rest-framework-gis library*. URL: https://github.com/djangonauts/django-rest-framework-gis.

[7] *CarGuard project*. URL: http://inlab.fib.upc.edu/ca/carguard-comunicant-mobils-amb-automobils.

[8] Tom Christie. *Django REST Framework*. URL: http://www.django-rest-framework.org/.

[9] Steve Coast. *Open Street Map*. URL: https://www.openstreetmap.org/#map=6/40.007/-2.488.

[10] Vincent Driessen. *A successful Git branching model*. URL: http://nvie.com/posts/a-successful-git-branching-model/.

[11] *Extreme Programming*. URL: http://www.extremeprogramming.org/.

[12] Martin Fowler. *YAGNI Philosophy*. URL: https://martinfowler.com/bliki/Yagni.html.

[13] *GeoJSON*. URL: http://geojson.org/.

[14] *Google Identity Platform*. URL: https://developers.google.com/identity/.

[15] *Informatics Engineering subjects*. URL: https://www.fib.upc.edu/en/studies/bachelors-degrees/bachelor-degree-informatics-engineering/curriculum/syllabus.

[16] *inLab FIB*. URL: http://inlab.fib.upc.edu/en.

[17] *Introduction to Test Driven Development (TDD)*. URL: http://agiledata.org/essays/tdd.html.

[18] *Jenkins*. URL: https://jenkins-ci.org/.

[19] Brian Mulloy. *Web API Design. Crafting Interfaces that Developers Love*. apigee, 2012.

[20] *Open API Specification*. URL: https://swagger.io/specification/.

[21] *Open Source*. URL: https://opensource.com/resources/what-open-source.

[22] *Open Source license*. URL: https://opensource.org/licenses/alphabetical.

[23] *OpenNebula*. URL: https://opennebula.org/.

[24] *Page Personnel*. URL: https://www.pagepersonnel.es/.

[25] *Pair Programming*. URL: http://www.extremeprogramming.org/rules/pair.html.

[26] *PostgreSQL*. URL: https://www.postgresql.org/.

[27] Paul Ramsey. *PostGIS*. URL: http://postgis.net/.

[28] *SEAT*. URL: http://www.seat.es/.

[29] *SEAT UPC Chair*. URL: http://catedraseat.upc.edu/?r=site/index&language=en.

[30] *Semantic Versioning 2.0.0*. URL: https://semver.org/.

[31] *Slack tool*. URL: https://slack.com/intl/es-es.

[32] JetBrains s.r.o. *JetBrains*. URL: https://www.jetbrains.com/.

[33] *Swagger Editor*. URL: https://swagger.io/swagger-editor/.

[34] *Too Path*. URL: http://inlab.fib.upc.edu/es/toopath-sistema-de-seguimiento-gratuito-de-dispositivos-moviles.

Appendix A

Database conceptual schema

The database conceptual schema is shown on the next page due to its big size.

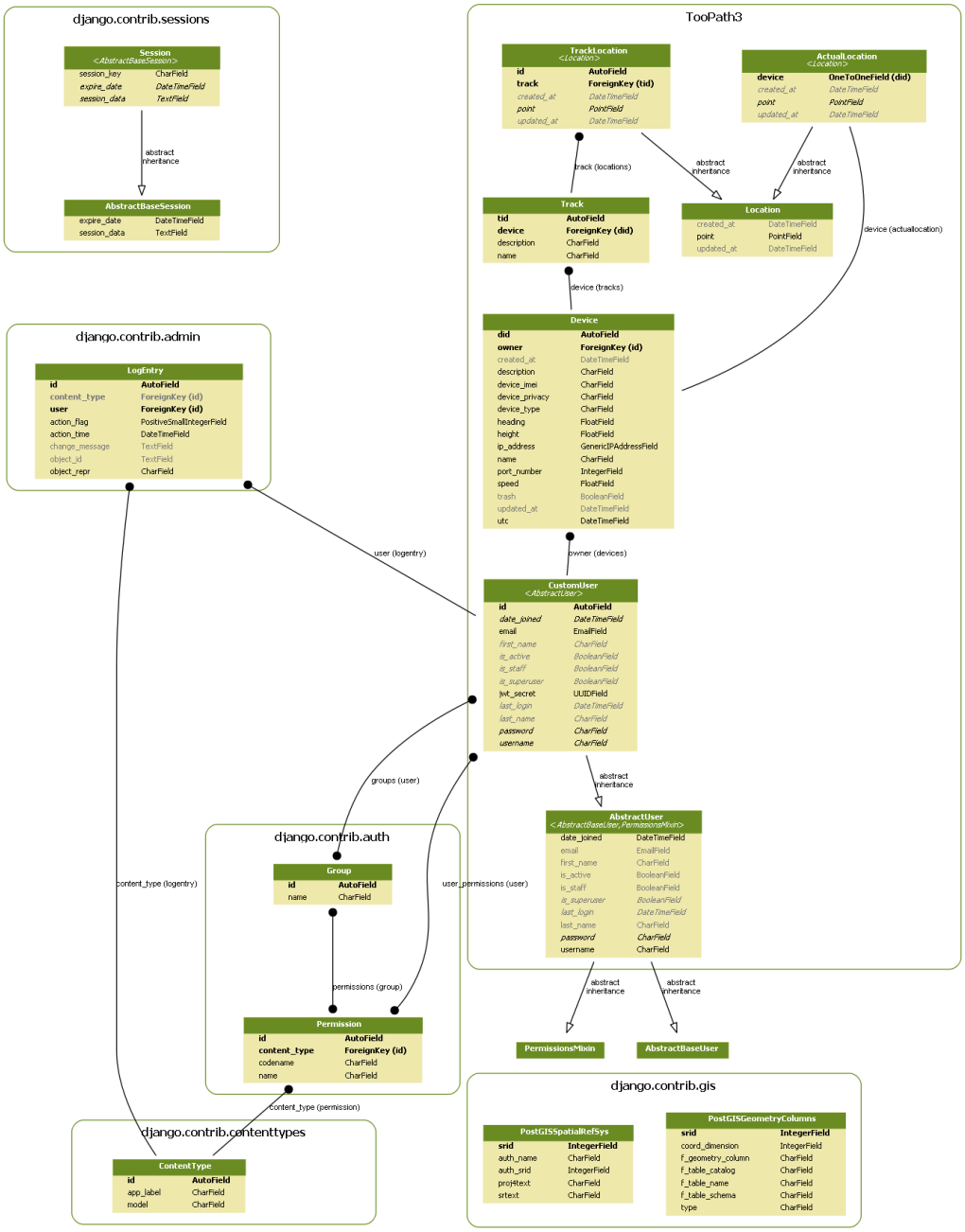

Figure A.1: Database models conceptual schema

Appendix B

OpenAPI Specification

```yaml
swagger: 2.0
info:
  version: 1.0.0
  title: TooPath
  description: TooPath is a device tracking software offered as a
    service.
  contact:
    email: albert.diaz.benitez@fib.upc.edu
  license:
    name: MIT License
    url: https://opensource.org/licenses/MIT
host: 127.0.0.1:8000
schemes:
  - http
tags:
  - name: users
    description: petitions for the user management
  - name: login
    description: petitions for login
  - name: token
    description: petitions to manage tokens
  - name: devices
    description: petitions to manage devices
  - name: track
```

```
24      description: petitions to manage tracks
25    - name: track location
26      description: petitions to manage the tracks locations
27    - name: actual location
28      description: petitions to manage the actual location of a device
29  paths:
30    /devices/:
31      get:
32        tags:
33          - devices
34        summary: Get all the user devices
35        description: >-
36          Petition to retrieve all the devices instances associated with
             the user
37          identified by the JWT
38        produces:
39          - application/json
40        consumes:
41          - application/json
42        security:
43          - JWT: []
44        responses:
45           200 :
46            description: devices information retrieved
47            schema:
48              type: array
49              items:
50                $ref:  #/definitions/DeviceResponse
51           401 :
52            description: Unauthorized
53            schema:
54              $ref:  #/definitions/Error401
55           403 :
56            description: Forbidden resource
57            schema:
58              $ref:  #/definitions/Error403
59      post:
```

```
60      tags:
61        - devices
62      summary: Create a new device
63      description: >-
64        Petition to create a device instance with all the fields passed
           on the
65        body and associated with the user identified by the JWT
66      produces:
67        - application/json
68      consumes:
69        - application/json
70      security:
71        - JWT: []
72      parameters:
73        - in: body
74          name: body
75          description: User info
76          required: true
77          schema:
78            $ref:  #/definitions/Device
79      responses:
80         200 :
81          description: device created succesfully
82          schema:
83            $ref:  #/definitions/DeviceResponse
84         400 :
85          description: Bad Request due to invalid body or fields format
86          schema:
87            $ref:  #/definitions/Error400
88         401 :
89          description: Unauthorized
90          schema:
91            $ref:  #/definitions/Error401
92         403 :
93          description: Forbidden resource
94          schema:
95            $ref:  #/definitions/Error403
```

```
96          409 :
97           description: Resource conflict
98           schema:
99             $ref:  #/definitions/Error409
100   /devices/{deviceId}/ :
101    get:
102      tags:
103        - devices
104      summary: Get device info
105      description: >-
106        Petition to retrieve the device instance associated with the user
107        identified by the JWT and identified by the ID on the URL
108      produces:
109        - application/json
110      consumes:
111        - application/json
112      security:
113        - JWT: []
114      parameters:
115        - name: deviceId
116          in: path
117          description: ID of device to retrieve
118          required: true
119          type: integer
120          format: int64
121      responses:
122         200 :
123          description: Device information retrieved
124          schema:
125            $ref:  #/definitions/DeviceResponse
126         401 :
127          description: Unauthorized
128          schema:
129            $ref:  #/definitions/Error401
130         403 :
131          description: Forbidden resource
132          schema:
```

```
133             $ref:  #/definitions/Error403
134           404 :
135             description: Device not found
136             schema:
137               $ref:  #/definitions/Error404
138       put:
139         tags:
140           - devices
141         summary: Change representation of a device
142         description: >-
143           Petition to change the representation of the device associated
              ↪  with the
144           user identified by the JWT and identified by the ID on the URL
145         produces:
146           - application/json
147         consumes:
148           - application/json
149         security:
150           - JWT: []
151         parameters:
152           - name: deviceId
153             in: path
154             description: ID of device to change the representation
155             required: true
156             type: integer
157             format: int64
158           - in: body
159             name: body
160             description: Device representation to change
161             required: true
162             schema:
163               $ref:  #/definitions/Device
164         responses:
165           201 :
166             description: Device representation changed succesfully
167             schema:
168               $ref:  #/definitions/DeviceResponse
```

```
169         400 :
170          description: Bad Request due to invalid body or fields format
171          schema:
172            $ref:  #/definitions/Error400
173         401 :
174          description: Unauthorized
175          schema:
176            $ref:  #/definitions/Error401
177         403 :
178          description: Forbidden resource
179          schema:
180            $ref:  #/definitions/Error403
181         404 :
182          description: Device not found
183          schema:
184            $ref:  #/definitions/Error404
185     patch:
186       tags:
187         - devices
188       summary: Update fields of a device
189       description: >-
190         Petition to update or change fields of the device instance
             ‹!  associated
191         with the user identified by the JWT and identified by the ID on
             ‹!  the URL
192       produces:
193         - application/json
194       consumes:
195         - application/json
196       security:
197         - JWT: []
198       parameters:
199         - name: deviceId
200           in: path
201           description: ID of device to update fields
202           required: true
203           type: integer
```

```
204           format: int64
205         - in: body
206           name: body
207           description: Device fields to update
208           required: true
209           schema:
210             $ref:  #/definitions/Device
211       responses:
212         201 :
213           description: Device updated succesfully
214           schema:
215             $ref:  #/definitions/DeviceResponse
216         400 :
217           description: Bad Request due to invalid body or fields format
218           schema:
219             $ref:  #/definitions/Error400
220         401 :
221           description: Unauthorized
222           schema:
223             $ref:  #/definitions/Error401
224         403 :
225           description: Forbidden resource
226           schema:
227             $ref:  #/definitions/Error403
228         404 :
229           description: Device not found
230           schema:
231             $ref:  #/definitions/Error404
232     delete:
233       tags:
234         - devices
235       summary: Delete a device instance
236       description: >-
237         This petition allow to delete the device instance identified by
            the ID
238         located on the URL
239       produces:
```

```
240         - application/json
241       consumes:
242         - application/json
243       security:
244         - JWT: []
245       parameters:
246         - name: deviceId
247           in: path
248           description: ID of device to be deleted
249           required: true
250           type: integer
251           format: int64
252       responses:
253          204 :
254           description: Device deleted succesfully
255          401 :
256           description: Unauthorized
257           schema:
258             $ref:  #/definitions/Error401
259          403 :
260           description: Forbidden resource
261           schema:
262             $ref:  #/definitions/Error403
263          404 :
264           description: Device not found
265           schema:
266             $ref:  #/definitions/Error404
267   /devices/{deviceId}/actualLocation/ :
268    get:
269       tags:
270         - actual location
271       summary: Retrieve the Actual Location of a device
272       description: >-
273         Petition to retrieve the actual location instance associated with
            the
274         device identified by the ID on the URL
275       produces:
```

```
276         - application/json
277       consumes:
278         - application/json
279       security:
280         - JWT: []
281       parameters:
282         - name: deviceId
283           in: path
284           description: device ID of the actual location
285           required: true
286           type: integer
287           format: int64
288       responses:
289          200 :
290           description: Actual Location retrieved succesfully
291           schema:
292             $ref:  #/definitions/ActualLocationResponse
293          401 :
294           description: Unauthorized
295           schema:
296             $ref:  #/definitions/Error401
297          403 :
298           description: Forbidden resource
299           schema:
300             $ref:  #/definitions/Error403
301          404 :
302           description: Device not found
303           schema:
304             $ref:  #/definitions/Error404
305     put:
306       tags:
307         - actual location
308       summary: Change representation of the actual location of a device
309       description: >-
310         Petition to change representation of the actual location
            ‹!  associated with
311         the device identified by the ID on the URL
```

```
312       produces:
313         - application/json
314       consumes:
315         - application/json
316       security:
317         - JWT: []
318       parameters:
319         - name: deviceId
320           in: path
321           description: device ID of the actual location
322           required: true
323           type: integer
324           format: int64
325         - in: body
326           name: body
327           description: New Actual Location representation info
328           required: true
329           schema:
330             $ref:  #/definitions/ActualLocation
331       responses:
332          200 :
333           description: Actual Location representation changed succesfully
334           schema:
335             $ref:  #/definitions/ActualLocationResponse
336          400 :
337           description: Bad Request due to invalid body or fields format
338           schema:
339             $ref:  #/definitions/Error400
340          401 :
341           description: Unauthorized
342           schema:
343             $ref:  #/definitions/Error401
344          403 :
345           description: Forbidden resource
346           schema:
347             $ref:  #/definitions/Error403
348          404 :
```

```
349           description: Device not found
350           schema:
351             $ref:  #/definitions/Error404
352     /devices/{deviceId}/tracks/ :
353      get:
354        tags:
355          - track
356        summary: Get all the tracks of a device
357        description: >-
358          Petition to retrieve all the tracks instance associated with the
              device
359          identified by the ID on the URL
360        produces:
361          - application/json
362        consumes:
363          - application/json
364        security:
365          - JWT: []
366        parameters:
367          - name: deviceId
368            in: path
369            description: ID of device to return tracks
370            required: true
371            type: integer
372            format: int64
373        responses:
374           200 :
375            description: Tracks information retrieved
376            schema:
377              type: array
378              items:
379                $ref:  #/definitions/TrackResponse
380           401 :
381            description: Unauthorized
382            schema:
383              $ref:  #/definitions/Error401
384           403 :
```

```
385           description: Forbidden resource
386           schema:
387             $ref:  #/definitions/Error403
388         404 :
389           description: Device not found
390           schema:
391             $ref:  #/definitions/Error404
392     post:
393       tags:
394         - track
395       summary: Create a new track
396       description: >-
397         Petition to retrieve all the tracks instance associated with the
          ‹!  device
398         identified by the ID on the URL
399       produces:
400         - application/json
401       consumes:
402         - application/json
403       security:
404         - JWT: []
405       parameters:
406         - name: deviceId
407           in: path
408           description: ID of device to return tracks
409           required: true
410           type: integer
411           format: int64
412         - in: body
413           name: body
414           description: User info
415           required: true
416           schema:
417             $ref:  #/definitions/Track
418       responses:
419         201 :
420           description: Track created succesfully
```

```
421           schema:
422             $ref:  #/definitions/TrackResponse
423         400 :
424           description: Bad Request due to invalid body or fields format
425           schema:
426             $ref:  #/definitions/Error400
427         401 :
428           description: Unauthorized
429           schema:
430             $ref:  #/definitions/Error401
431         403 :
432           description: Forbidden resource
433           schema:
434             $ref:  #/definitions/Error403
435         404 :
436           description: Device not found
437           schema:
438             $ref:  #/definitions/Error404
439         409 :
440           description: Resource conflict
441           schema:
442             $ref:  #/definitions/Error409
443   /devices/{deviceId}/tracks/{trackId}/ :
444    get:
445      tags:
446        - track
447      summary: Get track information
448      description: >-
449        Petition to retrieve the track instance identified by the ID on
           ‹!  the URL
450        associated with the device identified by the ID on the URL
451      produces:
452        - application/json
453      consumes:
454        - application/json
455      security:
456        - JWT: []
```

```
457       parameters:
458         - name: deviceId
459           in: path
460           description: device ID of the track
461           required: true
462           type: integer
463           format: int64
464         - name: trackId
465           in: path
466           description: ID of the track to retrieve
467           required: true
468           type: integer
469           format: int64
470       responses:
471          201 :
472           description: Track info retrieved succesfully
473           schema:
474             $ref:  #/definitions/TrackResponse
475          401 :
476           description: Unauthorized
477           schema:
478             $ref:  #/definitions/Error401
479          403 :
480           description: Forbidden resource
481           schema:
482             $ref:  #/definitions/Error403
483          404 :
484           description: Device or Track not found
485           schema:
486             $ref:  #/definitions/Error404
487     put:
488       tags:
489         - track
490       summary: Change representation of a track
491       description: >-
492         Petition to cahnge the representation of the track instance
            identified
```

```
493         by the ID on the URL associated with the device identified by the
            <!  ID on
494         the URL
495       produces:
496         - application/json
497       consumes:
498         - application/json
499       security:
500         - JWT: []
501       parameters:
502         - name: deviceId
503           in: path
504           description: device ID of the track
505           required: true
506           type: integer
507           format: int64
508         - name: trackId
509           in: path
510           description: ID of the track to change the representation
511           required: true
512           type: integer
513           format: int64
514         - in: body
515           name: body
516           description: New track representation info
517           required: true
518           schema:
519             $ref:  #/definitions/Track
520       responses:
521          201 :
522           description: Track representation changed succesfully
523           schema:
524             $ref:  #/definitions/TrackResponse
525          400 :
526           description: Bad Request due to invalid body or fields format
527           schema:
528             $ref:  #/definitions/Error400
```

```
529         401 :
530          description: Unauthorized
531          schema:
532            $ref:  #/definitions/Error401
533         403 :
534          description: Forbidden resource
535          schema:
536            $ref:  #/definitions/Error403
537         404 :
538          description: Device or Track not found
539          schema:
540            $ref:  #/definitions/Error404
541     patch:
542       tags:
543         - track
544       summary: Update fields of a track
545       description: >-
546         Petition to update the fields of the track instance identified by
            ‹!  the ID
547         on the URL associated with the device identified by the ID on the
            ‹!  URL
548       produces:
549         - application/json
550       consumes:
551         - application/json
552       security:
553         - JWT: []
554       parameters:
555         - name: deviceId
556           in: path
557           description: device ID of the track
558           required: true
559           type: integer
560           format: int64
561         - name: trackId
562           in: path
563           description: ID of the track to be updated
```

```
564           required: true
565           type: integer
566           format: int64
567         - in: body
568           name: body
569           description: New track fields values info
570           required: true
571           schema:
572             $ref:  #/definitions/Track
573       responses:
574         201 :
575           description: Track fields updated succesfully
576           schema:
577             $ref:  #/definitions/TrackResponse
578         400 :
579           description: Bad Request due to invalid body or fields format
580           schema:
581             $ref:  #/definitions/Error400
582         401 :
583           description: Unauthorized
584           schema:
585             $ref:  #/definitions/Error401
586         403 :
587           description: Forbidden resource
588           schema:
589             $ref:  #/definitions/Error403
590         404 :
591           description: Device or Track not found
592           schema:
593             $ref:  #/definitions/Error404
594     delete:
595       tags:
596         - track
597       summary: Delete a track
598       description: >-
599         Petition to delete the track instance identified by the ID on the
          ‹!  URL
```

```
600         associated with the device identified by the ID on the URL
601       produces:
602         - application/json
603       consumes:
604         - application/json
605       security:
606         - JWT: []
607       parameters:
608         - name: deviceId
609           in: path
610           description: device ID of the track
611           required: true
612           type: integer
613           format: int64
614         - name: trackId
615           in: path
616           description: ID of the track to be deleted
617           required: true
618           type: integer
619           format: int64
620       responses:
621          204 :
622           description: Track deleted succesfully
623          401 :
624           description: Unauthorized
625           schema:
626             $ref:  #/definitions/Error401
627          403 :
628           description: Forbidden resource
629           schema:
630             $ref:  #/definitions/Error403
631          404 :
632           description: Device or Track not found
633           schema:
634             $ref:  #/definitions/Error404
635   /devices/{deviceId}/tracks/{trackId}/trackLocations/ :
636    post:
```

```
637       tags:
638         - track location
639       summary: Create a new track location
640       description: >-
641         Petition to create the track location instance asociated on the
              track
642         identified by the ID on the URL associated with the device
              identified by
643         the ID on the URL
644       produces:
645         - application/json
646       consumes:
647         - application/json
648       security:
649         - JWT: []
650       parameters:
651         - name: deviceId
652           in: path
653           description: ID of device of the track
654           required: true
655           type: integer
656           format: int64
657         - name: trackId
658           in: path
659           description: ID of the track to create the new track location
660           required: true
661           type: integer
662           format: int64
663         - in: body
664           name: body
665           description: Track location info
666           required: true
667           schema:
668             $ref:  #/definitions/TrackLocation
669       responses:
670          201 :
671           description: Track Location created succesfully
```

```
672           schema:
673             $ref:  #/definitions/TrackLocationResponse
674         400 :
675           description: Bad Request due to invalid body or fields format
676           schema:
677             $ref:  #/definitions/Error400
678         401 :
679           description: Unauthorized
680           schema:
681             $ref:  #/definitions/Error401
682         403 :
683           description: Forbidden resource
684           schema:
685             $ref:  #/definitions/Error403
686         404 :
687           description: Device or Track not found
688           schema:
689             $ref:  #/definitions/Error404
690
    /devices/{deviceId}/tracks/{trackId}/trackLocations/{trackLocationId} :
691   put:
692     tags:
693       - track location
694     summary: Change the representation of the track location
695     description: >-
696       Petition to delete the track location instance asociated on the
          track
697       identified by the ID on the URL associated with the device
          identified by
698       the ID on the URL
699     produces:
700       - application/json
701     consumes:
702       - application/json
703     security:
704       - JWT: []
705     parameters:
```

```
706         - name: deviceId
707           in: path
708           description: ID of device of the track
709           required: true
710           type: integer
711           format: int64
712         - name: trackId
713           in: path
714           description: ID of the track to update the new track location
715           required: true
716           type: integer
717           format: int64
718         - name: trackLocationId
719           in: path
720           description: ID of the track location to be updated
721           required: true
722           type: integer
723           format: int64
724       responses:
725         204 :
726           description: Track Location representation changed succesfully
727           schema:
728             $ref:  #/definitions/TrackLocationResponse
729         401 :
730           description: Unauthorized
731           schema:
732             $ref:  #/definitions/Error401
733         403 :
734           description: Forbidden resource
735           schema:
736             $ref:  #/definitions/Error403
737         404 :
738           description:  Device, Track or Track Location not found
739           schema:
740             $ref:  #/definitions/Error404
741     delete:
742       tags:
```

```
743         - track location
744       summary: Delete a track location instance
745       description: >-
746         Petition to delete the track location instance asociated on the
              track
747         identified by the ID on the URL associated with the device
              identified by
748         the ID on the URL
749       produces:
750         - application/json
751       consumes:
752         - application/json
753       security:
754         - JWT: []
755       parameters:
756         - name: deviceId
757           in: path
758           description: ID of device of the track
759           required: true
760           type: integer
761           format: int64
762         - name: trackId
763           in: path
764           description: ID of the track to delete the new track location
765           required: true
766           type: integer
767           format: int64
768         - name: trackLocationId
769           in: path
770           description: ID of the track location to be deleted
771           required: true
772           type: integer
773           format: int64
774       responses:
775          204 :
776           description: Track Location deleted succesfully
777          401 :
```

```
778           description: Unauthorized
779           schema:
780             $ref:  #/definitions/Error401
781         403 :
782           description: Forbidden resource
783           schema:
784             $ref:  #/definitions/Error403
785         404 :
786           description:  Device, Track or Track Location not found
787           schema:
788             $ref:  #/definitions/Error404
789   /users/:
790     post:
791       tags:
792         - users
793       summary: Create a new user
794       description: Petition to create the user instance with the info
          passed on the body
795       produces:
796         - application/json
797       consumes:
798         - application/json
799       parameters:
800         - in: body
801           name: body
802           description: User info
803           required: true
804           schema:
805             $ref:  #/definitions/User
806       responses:
807         201 :
808           description: User created
809           schema:
810             $ref:  #/definitions/UserResponse
811         400 :
812           description: This field is required
813           schema:
```

```
814             $ref:  #/definitions/Error400
815           409 :
816             description: Resource conflict
817             schema:
818               $ref:  #/definitions/Error409
819   /users/{userId}/ :
820    get:
821      tags:
822        - users
823      summary: Get information of the user
824      description: Petition to retrieve the user instance identified by
          ↪  the ID on the URL
825      produces:
826        - application/json
827      consumes:
828        - application/json
829      security:
830        - JWT: []
831      parameters:
832        - name: userId
833          in: path
834          description: ID of user to return
835          required: true
836          type: integer
837          format: int64
838      responses:
839         200 :
840          description: OK
841          schema:
842            $ref:  #/definitions/UserResponse
843         401 :
844          description: Unauthorized
845          schema:
846            $ref:  #/definitions/Error401
847         403 :
848          description: Forbidden resource
849          schema:
```

```
850             $ref:  #/definitions/Error403
851           404 :
852             description: User not found
853             schema:
854               $ref:  #/definitions/Error404
855       put:
856         tags:
857           - users
858         summary: Change representation of the user
859         description: >-
860           Petition to change the representation of the user instance
               identified by
861           the ID on the URL
862         produces:
863           - application/json
864         consumes:
865           - application/json
866         security:
867           - JWT: []
868         parameters:
869           - name: userId
870             in: path
871             description: ID of user to return
872             required: true
873             type: integer
874             format: int64
875           - in: body
876             name: body
877             description: User representation to be changed
878             required: true
879             schema:
880               $ref:  #/definitions/User
881         responses:
882           200 :
883             description: Representation changed succcesfully
884             schema:
885               $ref:  #/definitions/UserResponse
```

```
886         400 :
887          description: Bad Request due to invalid body or fields format
888          schema:
889            $ref:  #/definitions/Error400
890         401 :
891          description: Unauthorized
892          schema:
893            $ref:  #/definitions/Error401
894         403 :
895          description: Forbidden resource
896          schema:
897            $ref:  #/definitions/Error403
898         404 :
899          description: User not found
900          schema:
901            $ref:  #/definitions/Error404
902     patch:
903       tags:
904         - users
905       summary: Update user fields
906       description: >-
907         Petition to update the fields of the user instance identified by
             the ID
908         on the URL
909       produces:
910         - application/json
911       consumes:
912         - application/json
913       security:
914         - JWT: []
915       parameters:
916         - name: userId
917           in: path
918           description: ID of user to return
919           required: true
920           type: integer
921           format: int64
```

```
922         - in: body
923           name: body
924           description: User fields to be updated
925           required: true
926           schema:
927             $ref:  #/definitions/User
928       responses:
929         200 :
930           description: Fields updated succesfully
931           schema:
932             $ref:  #/definitions/UserResponse
933         400 :
934           description: Bad Request due to invalid body or fields format
935           schema:
936             $ref:  #/definitions/Error400
937         401 :
938           description: Unauthorized
939           schema:
940             $ref:  #/definitions/Error401
941         403 :
942           description: Forbidden resource
943           schema:
944             $ref:  #/definitions/Error403
945         404 :
946           description: User not found
947           schema:
948             $ref:  #/definitions/Error404
949     delete:
950       tags:
951         - users
952       summary: Delete user instance
953       description: Petition to delete the user instance identified by the
          ↪  ID on the URL
954       produces:
955         - application/json
956       consumes:
957         - application/json
```

```
958       security:
959         - JWT: []
960       parameters:
961         - name: userId
962           in: path
963           description: ID of user to return
964           required: true
965           type: integer
966           format: int64
967       responses:
968          200 :
969           description: User deleted succesfully
970          401 :
971           description: Unauthorized
972           schema:
973             $ref:  #/definitions/Error401
974          403 :
975           description: Forbidden resource
976           schema:
977             $ref:  #/definitions/Error403
978          404 :
979           description: User not found
980           schema:
981             $ref:  #/definitions/Error404
982   /login/:
983     post:
984       tags:
985         - login
986       summary: Log In into the API
987       description: Petition to log in to the API and retrieve the JWT
988       produces:
989         - application/json
990       consumes:
991         - application/json
992       parameters:
993         - in: body
994           name: body
```

```
995           description: Login credentials
996           required: true
997           schema:
998             $ref:  #/definitions/Login
999       responses:
1000          200 :
1001          description: Log in succesful
1002          schema:
1003            $ref:  #/definitions/TokenResponse
1004          400 :
1005          description: Unable to log in with provided credentials.
1006  /login-google/:
1007    post:
1008      tags:
1009        - login
1010      summary: Log In into the API using a Google account
1011      description: >-
1012        Petition to log in using the Google account to the API and
           ‹!  retrieve the
1013        JWT
1014      produces:
1015        - application/json
1016      consumes:
1017        - application/json
1018      parameters:
1019        - in: body
1020          name: body
1021          description: Google credentials
1022          required: true
1023          schema:
1024            $ref:  #/definitions/GoogleLogin
1025      responses:
1026         200 :
1027          description: Log in succesful
1028          schema:
1029            $ref:  #/definitions/TokenResponse
1030          400 :
```

```
        description: Unable to log in with provided credentials.
  /api-token-refresh/:
    post:
      tags:
        - token
      summary: Refresh the JWT
      description: Petition to refresh a JWT passed on the body
      produces:
        - application/json
      consumes:
        - application/json
      parameters:
        - in: body
          name: body
          description: Login credentials
          required: true
          schema:
            type: object
            properties:
              token:
                type: string
      responses:
         200 :
          description: Token refreshed
          schema:
            $ref:  #/definitions/TokenResponse
  /api-token-verify/:
    post:
      tags:
        - token
      summary: Check if the JWT token is valid
      description: Petition to check the validity of the JWT passed on
       ‹!  the body
      produces:
        - application/json
      consumes:
        - application/json
```

```
      parameters:
        - in: body
          name: body
          description: Login credentials
          required: true
          schema:
            type: object
            properties:
              token:
                type: string
      responses:
         200 :
          description: Token valid
         400 :
          description: Token not valid
securityDefinitions:
  JWT:
    type: apiKey
    name: Authorization
    in: header
definitions:
  Device:
    type: object
    properties:
      name:
        type: string
      description:
        type: string
      trash:
        type: boolean
      ip_address:
        type: string
      port_number:
        type: integer
      height:
        type: number
      speed:
```

```
      type: number
    heading:
      type: number
    utc:
      type: string
    device_privacy:
      type: string
    device_type:
      type: string
    device_imei:
      type: string
Track:
  type: object
  properties:
    name:
      type: string
    description:
      type: string
TrackLocation:
  type: object
  properties:
    point:
      type: object
      properties:
        type:
          type: string
        coordinates:
          type: array
          items:
            type: number
            format: float
            maxItems: 2
            minItems: 2
ActualLocation:
  type: object
  properties:
    point:
```

```
      type: object
      properties:
        type:
          type: string
        coordinates:
          type: array
          items:
            type: number
            format: float
            maxItems: 2
            minItems: 2
Login:
  type: object
  properties:
    username:
      type: string
    password:
      type: string
GoogleLogin:
  type: object
  properties:
    email:
      type: string
    google-token:
      type: string
User:
  type: object
  properties:
    username:
      type: string
    email:
      type: string
    first_name:
      type: string
    last_name:
      type: string
    password:
```

```
        type: string
  DeviceResponse:
    type: object
    properties:
      did:
        type: integer
        format: int64
      owner:
        type: string
      tracks:
        type: array
        items:
          $ref:  #/definitions/TrackResponse
      name:
        type: string
      description:
        type: string
      created_at:
        type: string
      updated_at:
        type: string
      trash:
        type: boolean
      ip_address:
        type: string
      port_number:
        type: integer
      height:
        type: number
      speed:
        type: number
      heading:
        type: number
      utc:
        type: string
      device_privacy:
        type: string
```

```
      device_type:
        type: string
      device_imei:
        type: string
  TrackResponse:
    type: object
    properties:
      tid:
        type: integer
        format: int64
      name:
        type: string
      description:
        type: string
      device:
        type: integer
        format: int64
      locations:
        type: object
        properties:
          type:
            type: string
          features:
            type: array
            items:
              $ref:  #/definitions/TrackLocationResponse
  TrackLocationResponse:
    type: object
    properties:
      id:
        type: integer
        format: int64
      type:
        type: string
      geometry:
        type: object
        properties:
```

```
        type:
          type: string
        coordinates:
          type: array
          items:
            type: number
            format: float
            maxItems: 2
            minItems: 2
    properties:
      type: object
      properties:
        created_at:
          type: string
        updated_at:
          type: string
        track:
          type: integer
          format: int64
ActualLocationResponse:
  type: object
  properties:
    id:
      type: integer
      format: int64
    type:
      type: string
    geometry:
      type: object
      properties:
        type:
          type: string
        coordinates:
          type: array
          items:
            type: number
            format: float
```

```
            maxItems: 2
            minItems: 2
    properties:
      type: object
      properties:
        created_at:
          type: string
        updated_at:
          type: string
        device:
          type: integer
          format: int64
UserResponse:
  type: object
  properties:
    id:
      type: integer
      format: int64
    username:
      type: string
    email:
      type: string
    first_name:
      type: string
    last_name:
      type: string
    date_joined:
      type: string
    last_login:
      type: string
TokenResponse:
  type: object
  properties:
    token:
      type: string
Error403:
  type: object
```

```
    properties:
      detail:
        type: string
  Error401:
    type: object
    properties:
      detail:
        type: string
  Error400:
    type: object
    properties:
      field_error:
        type: string
  Error404:
    type: object
    properties:
      detail:
        type: string
  Error409:
    type: object
    properties:
      detail:
        type: string
```

Appendix C

Web mockups

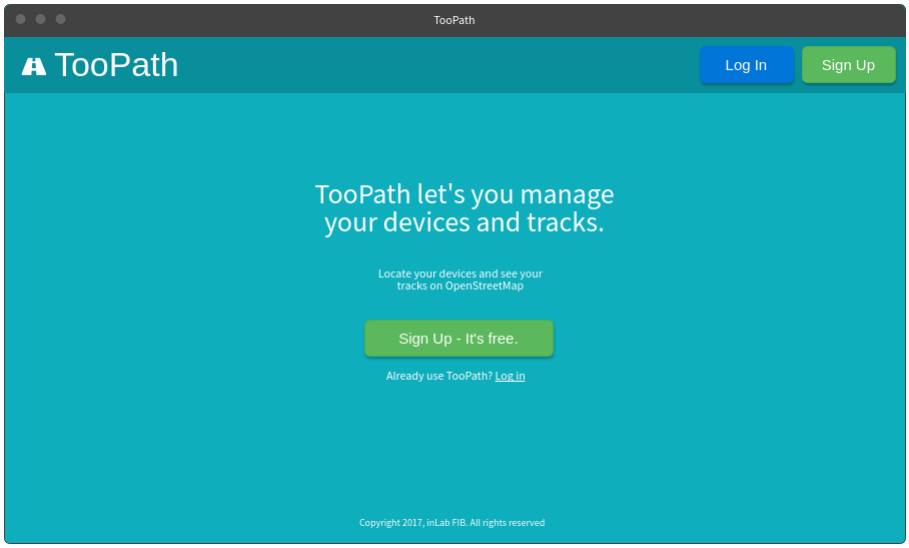

Figure C.1: Home page (not logged)

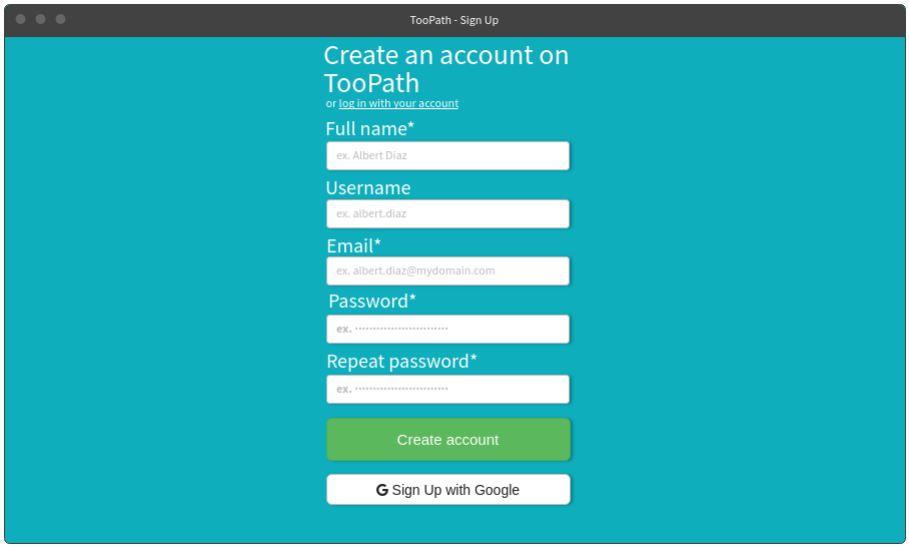

Figure C.2: Sign Up page

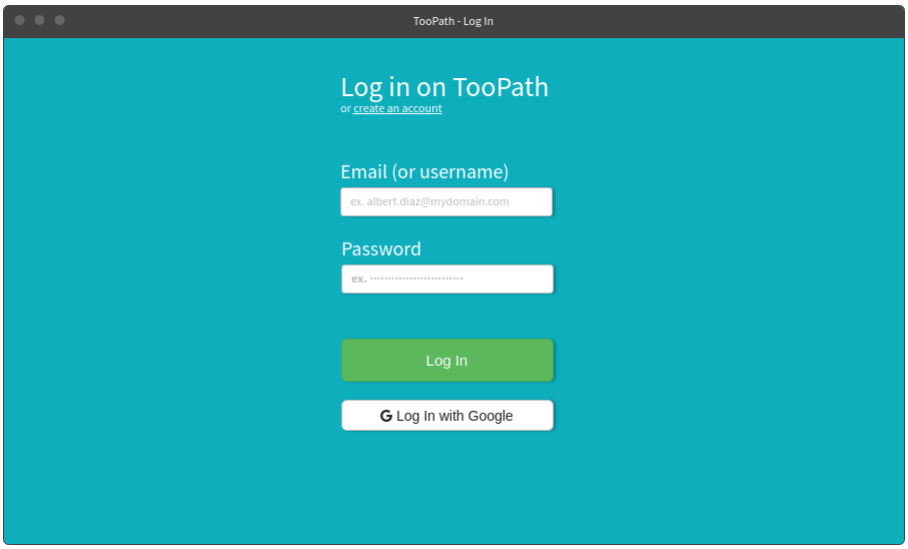

Figure C.3: Log In page

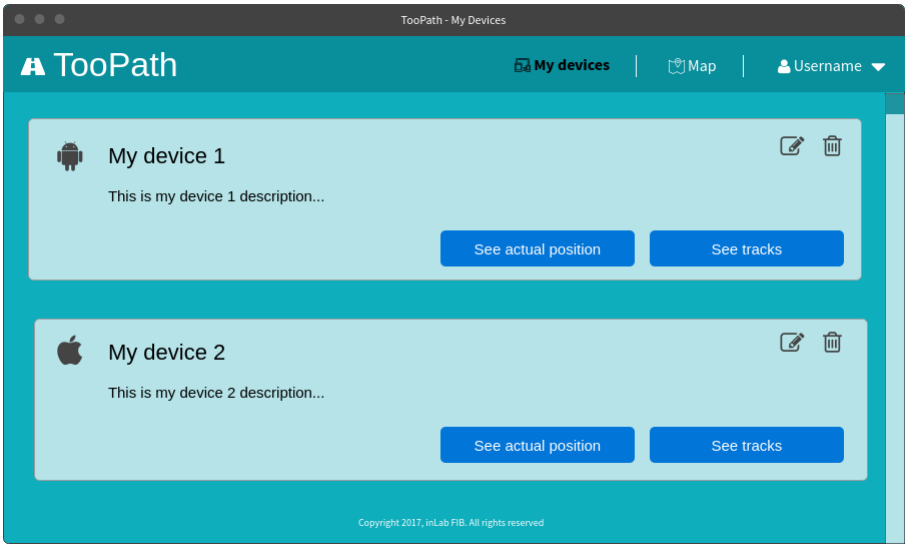

Figure C.4: Home page (logged)

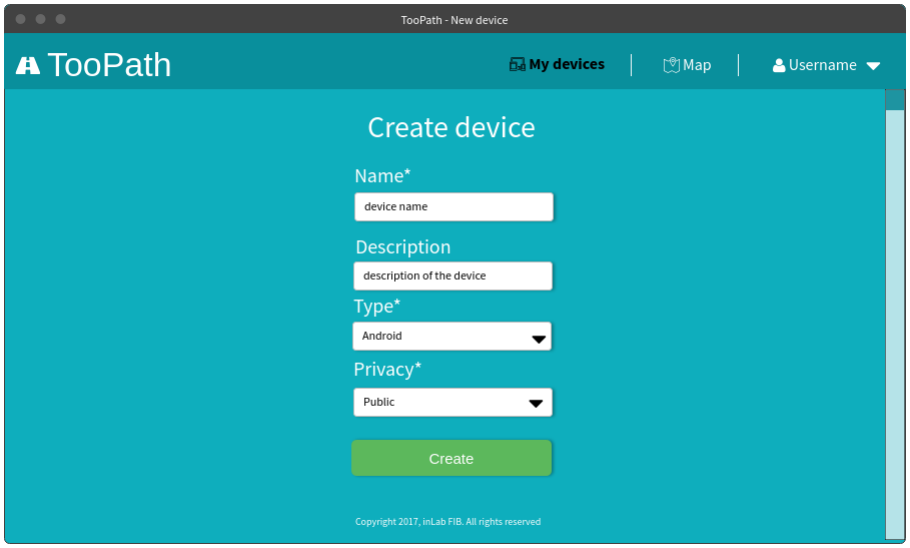

Figure C.5: Create a device page

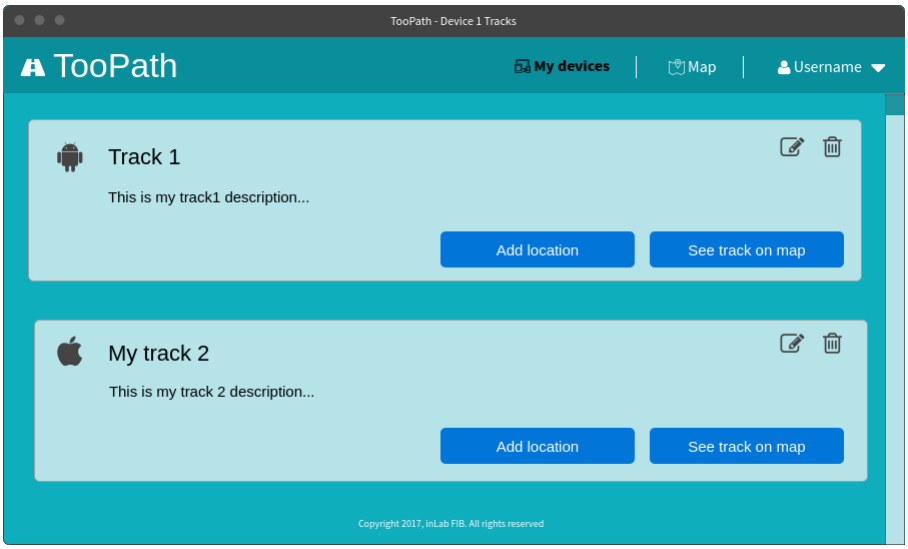

Figure C.6: Tracks of a device page

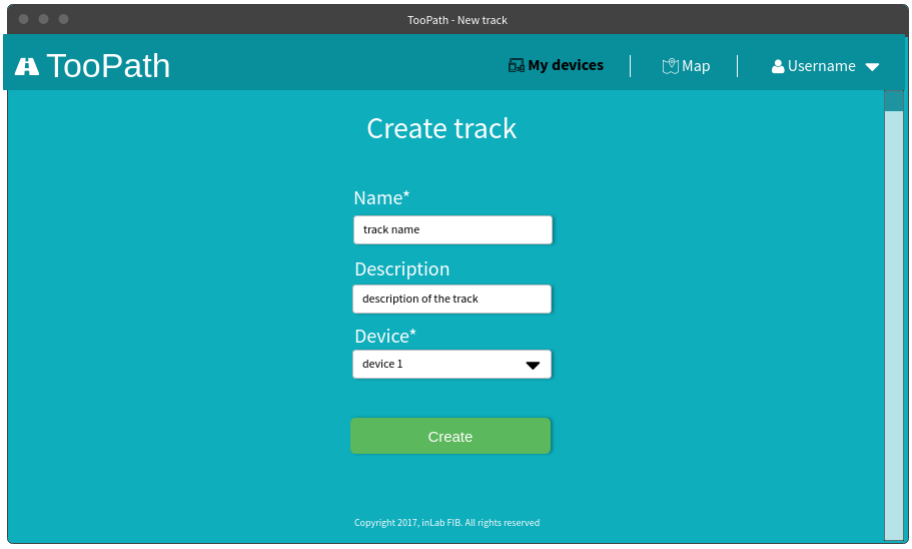

Figure C.7: Create a track page

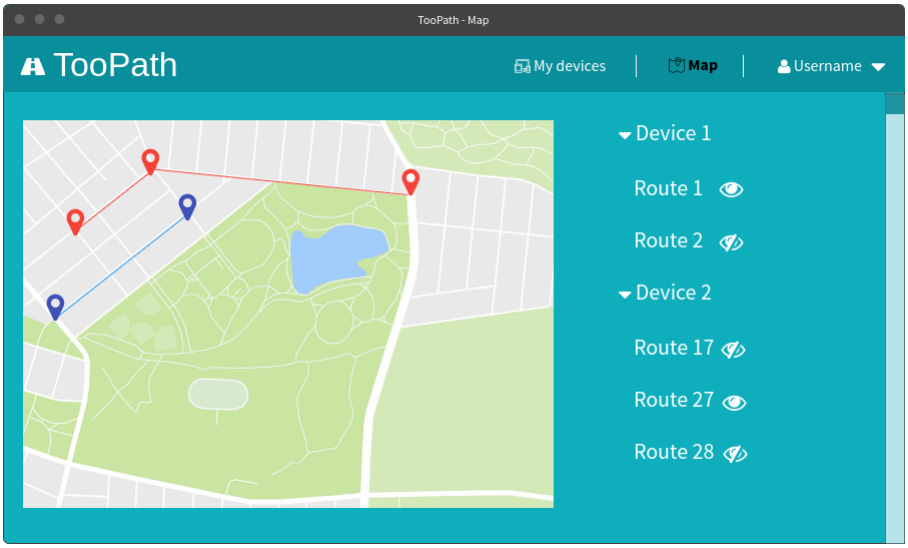

Figure C.8: Tracks visualization on map