



# Deusto

**Facultad de Ingeniería**  
Universidad de Deusto

**Ingeniaritza Fakultatea**  
Deustuko Unibertsitatea

## **Ingeniero en Informática** **Informatikako Ingeniaria**

**Proyecto fin de carrera**  
**Karrera amaierako proiektua**



## **Asbtract**

The goal of this project is the combination of semantic, web and geospatial technologies for the development of and on-line GPS community.

A web application will be developed using NodeJS, a server side JavaScript programming platform, and AngularJS a single page application building framework based on the model-view-controller pattern. This system will allow users to share trails and points of interest which will be structured according to a ontology developed for this system and stored on a triplestore.

The semantic nature of the data will allow the users for a more intelligent and precise search, beyond the classic key word search. Besides, with the use of GeoSPARQL, a standard for the representation and querying of geospatial linked data the implicit relations among this data, such as which points of interest surround a trail, can be obtained.

The web application will allow the users to share their trails and enrich their information. Besides, a mobile application will give support to the system allowing the users to follow the trails of others, discovering information near them in real time and allowing them to post location aware notes on their actual location.

## **Descriptores**

Semantic Web, GeoSPARQL, Linked Data, NodeJS, AngularJS.



# Contents

1	Introduction	1
1.1	Motivation . . . . .	1
1.2	State of the art . . . . .	2
1.2.1	The Semantic Web . . . . .	2
1.2.2	Linked Open Data . . . . .	5
1.2.3	Introduction to GIS systems . . . . .	5
2	Project Goals	7
2.1	Overview . . . . .	7
2.2	Project definition . . . . .	7
2.2.1	Goals . . . . .	7
2.2.2	Scope of the project . . . . .	8
2.3	Final product . . . . .	9
2.3.1	Server . . . . .	9
2.3.2	Web application . . . . .	10
2.3.3	Mobile application . . . . .	11
2.4	Project realization . . . . .	11
2.4.1	Realization methodology . . . . .	11
2.4.2	Intermediate products . . . . .	14
2.5	Organization and team . . . . .	16
2.5.1	Organizational structure . . . . .	16
2.5.2	Human resources plan . . . . .	16
2.6	Execution conditions . . . . .	18
2.6.1	Hardware . . . . .	18
2.6.2	Software . . . . .	18
2.6.3	Change control . . . . .	19
2.6.4	Product reception . . . . .	19
2.7	Planning . . . . .	20
2.8	Project budget . . . . .	25

3	Technological Research	27
3.1	Overview	27
3.2	RDF	27
3.3	OWL	32
3.4	SPARQL	35
3.5	GeoSPARQL	37
3.6	NodeJS	38
3.7	Leaflet	43
3.8	Phonegap	45
3.9	GPX	46
3.10	GeoJSON	48
4	Requirements Specification	51
4.1	Overview	51
4.2	Requirements for the ontology	51
4.3	Requirements specification for the server	53
4.4	Requirements of the web application	55
4.5	Requirements of the mobile application	56
4.6	Non-functional requirements	57
5	Design specification	59
5.1	Overview	59
5.2	System architecture	59
5.2.1	Central server	59
5.2.2	Semantic spatial storage system	61
5.2.3	Web client	61
5.2.4	Mobile application	61
5.2.5	Functioning of the system	61
5.3	Design of the ontology	62
5.3.1	Class hierarchy	63
5.3.2	Properties	64
5.3.3	Advantages of ontology based design	68
5.4	Design of the central server	72
5.4.1	Web server	72
5.4.2	REST API	75

5.4.3	Data aggregator . . . . .	76
5.4.4	Socket server . . . . .	76
5.5	Design of the web application . . . . .	77
5.5.1	API connector . . . . .	79
5.5.2	Web . . . . .	80
5.5.3	Map . . . . .	81
5.5.4	Accounts . . . . .	84
5.5.5	Dialogs . . . . .	84
5.5.6	Graphical design . . . . .	85
5.6	Design of the mobile application . . . . .	87
5.6.1	Class design . . . . .	87
5.6.2	Real time communication protocol . . . . .	88
5.6.3	Plugin design . . . . .	89
5.6.4	Graphical design . . . . .	90
5.7	Development environment . . . . .	91
6	Implementation details . . . . .	95
6.1	Coding style . . . . .	95
6.2	Development environment . . . . .	95
6.2.1	Atom text editor . . . . .	95
6.2.2	Git . . . . .	96
6.2.3	Yeoman . . . . .	96
6.2.4	Grunt . . . . .	97
6.2.5	JSLint . . . . .	97
6.2.6	Protégé-OWL . . . . .	97
6.3	Development of the server . . . . .	97
6.3.1	NodeJS application structure . . . . .	98
6.3.2	Semantic database . . . . .	99
6.3.3	Database connector . . . . .	99
6.3.4	Geography Utilities . . . . .	103
6.3.5	Controllers . . . . .	105
6.3.6	Routes . . . . .	106
6.3.7	Recommendations . . . . .	106
6.3.8	Authentication . . . . .	107
6.3.9	Real time communication . . . . .	108

6.4	Development of the web application . . . . .	108
6.4.1	Angular application structure . . . . .	108
6.4.2	Application . . . . .	109
6.4.3	API . . . . .	109
6.4.4	Accounts . . . . .	110
6.4.5	Web . . . . .	111
6.4.6	Map . . . . .	113
6.4.7	Dialogs . . . . .	117
6.5	Development of the Mobile application . . . . .	119
6.5.1	Real time functionality - Server side . . . . .	120
6.5.2	Real time functionality - Client side . . . . .	122
6.6	Application functionality . . . . .	128
7	Testing . . . . .	129
7.1	Overview . . . . .	129
7.2	Test plan . . . . .	129
7.2.1	Unit tests . . . . .	129
7.2.2	Integration tests . . . . .	130
7.2.3	Installation tests . . . . .	130
7.3	Special test cases . . . . .	131
7.3.1	Ontology . . . . .	131
7.3.2	Trail recording and locator service . . . . .	131
7.3.3	User interaction . . . . .	132
8	User guide . . . . .	133
8.1	Overview . . . . .	133
8.2	Web application guide . . . . .	133
8.2.1	Homepage . . . . .	133
9	Conclusions and future lines of work . . . . .	135
	Bibliography . . . . .	137



## List of Figures

### Capítulo 1

1.1 The semantic web architecture . . . . .	3
1.2 The Linked Data cloud . . . . .	5

### Capítulo 2

2.1 Iterative development phases . . . . .	12
2.2 Work breakdown structure . . . . .	15
2.3 Organizational structure . . . . .	16
2.4 Gantt diagram . . . . .	23
2.5 Network diagram . . . . .	24

### Capítulo 3

3.1 An RDF graph describing a person . . . . .	28
3.2 The OWL levels of expressivity . . . . .	32
3.3 View-controller binding through the scope . . . . .	41
3.4 Leaflet with OSM mapnik tile layer . . . . .	43
3.5 Phonegap Build process . . . . .	46

### Capítulo 5

5.1 Architecture design of the system . . . . .	60
5.2 The SORELCOM ontology . . . . .	63
5.3 Inference of the husband relation as inverse of the wife relation . . . . .	70
5.4 Inference of all the classes of a Trail . . . . .	70
5.5 Inference of the husband relation as inverse of the wife relation . . . . .	71
5.6 Class diagram of the web server . . . . .	74
5.7 Resources and operations of the API . . . . .	75
5.8 Socket server class diagram . . . . .	77
5.9 Service processing sequence diagram . . . . .	77

5.10 High-level view of the web application components . . . . .	78
5.11 API connector class diagram . . . . .	80
5.12 Web subsystem class diagram . . . . .	81
5.13 Map subsystem class diagram . . . . .	83
5.14 Accounts component class diagram . . . . .	84
5.15 Dialog component class diagram . . . . .	85
5.16 Responsive web design on SORELCOM . . . . .	86
5.17 Class diagram of the new components of the mobile . . . . .	87
5.18 Nearby checking flow diagram . . . . .	89
5.19 Trail following flow diagram . . . . .	90
5.20 Class diagram of the mobile application native plugin . . . . .	91
5.21 Service subscription sequence diagram . . . . .	91
5.22 SASS to CSS transformation . . . . .	93

## Capítulo 6

6.1 The Atom text editor . . . . .	96
6.2 The Protégé OWL ontology editor . . . . .	98
6.3 Web interface of the parliament triple store . . . . .	100
6.4 Sequence diagram for most controller requests . . . . .	105

## List of Tables

### Capítulo 2

2.1 Intermediate products of the project. . . . .	14
2.2 Workload estimation for the project manager. . . . .	20
2.3 Workload estimation for the system architect. . . . .	20
2.4 Workload estimation for the researcher. . . . .	20
2.5 Workload estimation for the back-end developer. . . . .	21
2.6 Workload estimation for the front-end web developer. . . . .	21
2.7 Workload estimation for the mobile application developer. . . . .	22
2.8 Workload estimation for the graphics designer. . . . .	22
2.9 Real work plan. . . . .	22
2.10 Hardware related budget. . . . .	25
2.11 Human resources budget. . . . .	25
2.12 Summary of budget. . . . .	25

### Capítulo 5

5.1 Points of interest on the SORELCOM data model. . . . .	64
5.2 Trails on the SORELCOM data model. . . . .	65
5.3 Points of interest on the SORELCOM data model. . . . .	66
5.4 Geolocated notes on the SORELCOM data model. . . . .	66
5.5 Persons on the SORELCOM data model. . . . .	67
5.6 Media on the SORELCOM data model. . . . .	67
5.7 Posts on the SORELCOM data model. . . . .	67
5.8 Categories on the SORELCOM data model. . . . .	68
5.9 Inferences used on the SORELCOM ontology classes. . . . .	69
5.10 Inferences used on the SORELCOM ontology classes. . . . .	69
5.11 Real time communication protocol. . . . .	88



## Listings index

### Capítulo 1

1.1 Set of triples in pseudocode representing Bob. . . . .	4
--	---

### Capítulo 3

3.1 Different types of triple statements. . . . .	29
3.2 Triple statements with a blank node. . . . .	29
3.3 RDFS class and property hierarchy. . . . .	31
3.4 OWL property characteristic example. . . . .	33
3.5 OWL property restriction example. . . . .	34
3.6 OWL property restriction example. . . . .	35
3.7 SPARQL query for retrieving all triples on the dataset. . . . .	36
3.8 SPARQL query for retrieving names on the dataset. . . . .	36
3.9 A feature in the GeoSPARQL vocabulary. . . . .	37
3.10 Spatial query in SPARQL. . . . .	38
3.11 NodeJS "hello world" web server. . . . .	39
3.12 Express "hello world" program. . . . .	40
3.13 AngularJS example. . . . .	42
3.14 AngularJS example. . . . .	43
3.15 Leaflet layers. . . . .	44
3.16 Leaflet layers. . . . .	44
3.17 Reading battery status with Phonegap. . . . .	45
3.18 GPX way point representation. . . . .	47
3.19 GPX track representation. . . . .	47
3.20 GPX route representation. . . . .	47
3.21 A GeoJSON <code>GeometryCollection</code> object. . . . .	49
3.22 A GeoJSON <code>Feature</code> object. . . . .	49

### Capítulo 5

5.1 CSS3 media query example. . . . .	86
---------------------------------------	----

### Capítulo 6

6.1 NodeJS module requiring . . . . .	99
6.2 Jetty server configuration . . . . .	99
6.3 SPARQL Select structure . . . . .	100
6.4 SPARQL Modify structure . . . . .	100
6.5 SPARQL Modify structure . . . . .	101
6.6 The definition of the connector objects . . . . .	101
6.7 A query factory method . . . . .	102

6.8	Query object to SPARQL transformation . . . . .	102
6.9	RDF-JSON to GeoJSON transformation . . . . .	103
6.10	JavaScript implementation of the haversine formula . . . . .	103
6.11	Score calculation algorithm . . . . .	104
6.12	Controller method for obtaining the trails of a user . . . . .	105
6.13	API implementation for the trails resource . . . . .	106
6.14	Criteria for route recommendations in SPARQL queries . . . . .	107
6.15	AngularJS application configuration . . . . .	110
6.16	API service class . . . . .	111
6.17	Remote User class . . . . .	111
6.18	The search view document . . . . .	112
6.19	Leaflet usage with angular on feature pages . . . . .	113
6.20	Map service implementation . . . . .	114
6.21	Explorer service implementation . . . . .	115
6.22	Map drawing function . . . . .	115
6.23	Map point of interest marking function . . . . .	116
6.24	Code for adding markers to allow edition of polylines . . . . .	117
6.25	Code for removing markers to avoid performance issues . . . . .	117
6.26	Add coordinates on a editable line . . . . .	117
6.27	Remove coordinates on a editable line . . . . .	118
6.28	Midpoint formula implementation . . . . .	118
6.29	Modal invoking shortcuts . . . . .	118
6.30	Modal closing and communicating functions . . . . .	119
6.31	Modal closing and communicating functions . . . . .	119
6.32	Socket.IO server initialization . . . . .	120
6.33	Socket.IO authorization . . . . .	121
6.34	Subscribing functionality . . . . .	122
6.35	Trail following service implementation . . . . .	123
6.36	Socket server main process . . . . .	124
6.37	Android manifest file . . . . .	124
6.38	Recorder service implementation . . . . .	125
6.39	Recorder service implementation . . . . .	125
6.40	Socket communication client . . . . .	126
6.41	Service subscriptions and server message reading . . . . .	127
6.42	Android service creation from the application . . . . .	127
6.43	Trail recording service starting from the application . . . . .	127
6.44	Trail recording timer start and data reception implementation . . . . .	128
6.45	Location retrieval using Phonegap . . . . .	128

## **1. INTRODUCTION**

---

The goal of this report is to describe the process of design and implementation of the Final Undergraduate Project "SORELCOM: Design and Development of a semantic geospatial web and mobile application".

This project has been carried by Aimar Rodríguez Soto, 4th year student of an undergraduate degree in computer science engineering.

The different chapters that form this document are the following:

1. Introduction: Brief descriptions of the contents of the report, as well as a small introduction to the concepts of Semantic Web and GIS System.
2. Project goals: The goals, the limits, the tasks to perform, working methodologies and the working plan are presented.
3. Technological research: Results of a initial technological research are given through a detailed description of the most relevant of the technologies that affect the project.
4. Requirement Specification: Requirements of the project and the final product are listed and analyzed.
5. Design specification: Design considerations of the project are described, both from the technological and the user experience point of views. The technologies to be used in the implementation are listed.
6. Development: Usage of the development tools, style rules and the process of development of the project is detailed.
7. Testing: Testing plan, tools, methodologies and special test cases are described.
8. User guide: Document explaining how to use the developed tools and platforms at a user level.
9. Conclusions and future lines of work: Analysis of the results of the project and further considerations, such as improvements and future work to be done.

### **1.1. MOTIVATION**

Geospatial data is becoming more and more relevant in our world. In roughly 30 years, we have evolved from a region based spatial information, to human and device centric information. The amount of location-aware devices has multiplied and with this, location-based services are thriving. Users can retrieve directions and information based on their current location, the number of social media users which set their devices to include location information on their phones is growing. There is no doubt that the amount of spatial data is increasing and that it is making an impact in our societies.

## 1. INTRODUCTION

As a consequence, the amount of Linked Open Data with spatial context made available on the web has also increased. The principles of Linked Open Data[7] encourage the usage of RDF and SPARQL to query and model data. This is useful for querying for the explicit relations on the datasets, however, the implicit spatial relationships cannot be easily queried.

To fight this issue, the Open Geospatial Consortium(OGC) has defined a standard, named GeoSPARQL to address the issues of spatial information representation and querying on the semantic web. Thanks to this, it is possible to include a spatial factor in the reasoning done over RDF datasets.

The platform SORELCOM (Social Routes Empowered by Linked Contents and Context Mining) aims to use GeoSPARQL and data from over the web to create a user-centred GPS community which is able to offer location aware service to users and to publish the data generated in a spatially aware linked dataset.

The platform aims to combine semantic and spatial technologies to create a tool to capture routes and enrich their information from sources around the web, providing in addition a powerful location and preference based recommendation service.

## 1.2. STATE OF THE ART

This section introduces the main fields related to the projects and gives a small insight on the current state and work done on this fields.

### 1.2.1 The Semantic Web

The semantic web is a extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [6]

The idea behind the semantic web is to bring meaning and structure to the contents of web pages, creating an environment where software agents can roam the web carrying out sophisticated tasks for end users. Thanks to the semantic web, this agents will be able to understand the relationships among the information on a web page, beyond simply identifying certain keywords.

This machine-readable web of data is based standard languages which allow a uniform representation of the information throughout the different semantic data sources. Through this common infrastructure it is possible to share and process information in a relatively simple way, enabling better solution to the usual problem of searching through the web of information. In short, the Semantic Web attempts to give meaning and structure to the contents on the web, allowing computers to better understand this information to do more useful work.

#### **The fundamentals**

The Semantic Web is based on standard formats used for a universal representation and manipulation of data. This standards are the following:



RDF: The Resource Description Framework (RDF) is a standard language for representing information about resources in the World Wide Web. Resources can be anything, including documents, people, physical object and abstract concepts. [44]

OWL: The Web Ontology Language (OWL) is a semantic markup language for creating and sharing ontologies [4]. It is used to define ontologies when the content needs to be processed by applications or agents, instead of just presented to people [45].

SPARQL: the SPARQL Protocol and RDF Query Language is a language designed to make queries over resources in RDF format [55].

The idea of the semantic web is based on a stack architecture, illustrated on figure 1.1. This stack includes more layers than the mentioned, such as RDF Schema (RDFS) and the eXtensible Markup Language (XML). For further detail into these technologies and their role in the semantic web refer to chapter 3.

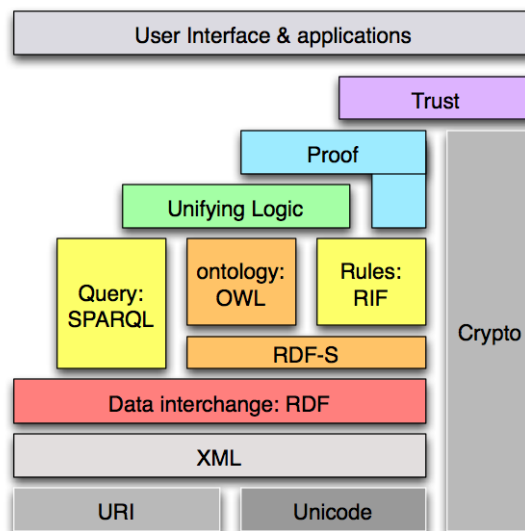


Figure 1.1: The semantic web architecture

source: [http://www.w3.org/2006/Talks/1023-sb-W3TechSemWeb/#\(19\)](http://www.w3.org/2006/Talks/1023-sb-W3TechSemWeb/#(19))

## Introduction to RDF

RDF provides a framework for representing information which can be exchanged between application without loss of meaning and it is intended for situation in which this information needs to be processed by machines instead of displayed to humans, thus making it fitting for the goals of The Semantic Web [69].

This framework has several uses, such as:

- Adding machine readable information to web pages, enabling them to be displayed in enhanced formats or to be processed by other agents.
- Enriching a database by linking it to third party datasets.
- Interlinking API feeds, so that clients can easily discover how to access more information.

## 1. INTRODUCTION

- Using datasets labeled as Linked Data to build aggregations of data around specific topics or concepts.

RDF information is stored in triples which follow a simple subject-predicate-object pattern. One example of RDF triples which represent knowledge about a person named Bob can be found in listing 1.1. Note that in this set of triples, most attributes appear between brackets, these are International Resource Identifiers (IRIs) linking to other resources, while the other attributes are just literal values.

```
1 Bob> <is_named> "Bob"
2 Bob> <is_a> <Person>
3 Bob> <is_a> <Student>
4 Bob> <knows> <Alice>
5 Bob> <is_interested_in> <Computer_Science>
```

Listing 1.1: Set of triples in pseudocode representing Bob.

These IRIs are identifiers for resources on the Semantic Web. Up to now the concepts of URI and IRI have been mixed, to make it clear, an IRI is a generalization of an URI. The form of a IRI is similar to that of the Uniform Resource Locators (URL) used in the World Wide Web, in fact these last are just a form of IRI, thus they look like the following: [http://dbpedia.org/resource/Leonardo\\_da\\_Vinci](http://dbpedia.org/resource/Leonardo_da_Vinci)

### Introduction to Ontologies

In the Semantic Web, vocabularies or ontologies define the concepts and relationships used to describe and represent a certain area of knowledge, they define the terms that can be used on a application, the possible relationships and their constraints. They are used to help data integration when ambiguity may exist between terms in different datasets, or simply to organize knowledge [68].

According to Tom Grubber [35] An ontology is a formal specification of a shared conceptualization. The concept of Ontology comes from the field of philosophy, however they have been adopted into Artificial Intelligence and Knowledge Representation [15]. Since ontologies are used to create conceptual structure on a domain, they can be used by software for problem solving and reasoning among other tasks.

We can find three types of ontologies:

Top-level ontologies : Describe very general notions which are independant of a particular domain and are applicable to different areas, for example; time, space, events, etc.

Domain ontologies : The knowledge represented in these ontologies is particular to a certain field of knowledge, such as computers, persons, forestry, etc. They provide information about concepts in the domain and about the theories governing the domain.

Application ontologies : This is the least general type of ontology. It describes knowledge specific to a certain a application or task, due to this, they are useful for problem solving.

### 1.2.2 Linked Open Data

Linked Data refers to a pattern for interlinking machine-readable data sets to each other, specially via the use of RDF and URIs. In summary, Linked Data is about using the web to create typed links between data from different sources [8].

Linked data relies on technologies that are fundamental to the web, mainly Uniform Resource Identifiers (URIs) and the Hypertext Transfer Protocol (HTTP). URIs are used as a unique identifier across the web to the resources on the Linked Data sets and are looked up using the HTTP protocol.

These technologies are supported by a technology that is crucial for the web of data - RDF. RDF and the ontology definition language OWL can be used to create vocabularies used to describe concepts and entities and how they are related.

This idea's more visible adoption is the Linking Open Data project [63] whose aim is to bootstrap the Web of Data by identifying existing data sets that are available under open licenses, converting these to RDF according to the Linked Data principles, and publishing them on the Web. An cloud representing the already linked data sets can be found in figure 1.2.

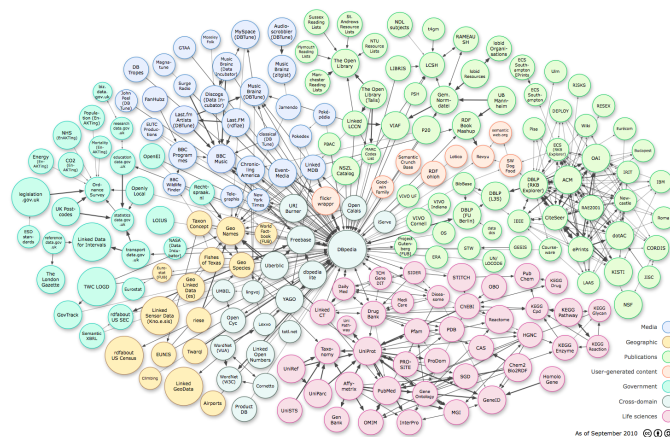


Figure 1.2: The Linked Data cloud

source: <http://linkeddatabook.com/editions/1.0/>

### 1.2.3 Introduction to GIS systems

A Geographical Information System (GIS) integrates hardware, software and data for capturing, managing and displaying all forms of geographically referenced information [25] A GIS provides a framework for gathering and organizing spatial data and related information so it can be displayed and analyzed.

Data on such a system is a digital representation of real world objects. Since this kind of data can be divided into two abstractions, discrete objects (a house, a park) or continuous fields (elevation), two methods are used to store data of this abstractions: Raster and Vector. [32]

## *1. INTRODUCTION*

### **Raster**

Raster data types consist of rows and columns of cells each of them storing a single value, in the say way that a image is composed by pixels. In fact, a Raster data type is, in essence, a type of digital image represented in grids. On a Raster Image each pixels stores a color value which represents certain data, for example, the amount of rain in that region.

This kind of data may be stored in different ways, for example, as a regular .JPEG image or directly in a relational database.

### **Vector**

Vector data is used to express geographical features, which are represented as geometrical shapes of different types.

- Points: They are used to describe features that can be represented with a single point, that is, the simplest of features, for example, the location of a monument, the peak of a mountain, etc.
- Lines or polylines: They are used to represent linear features, such as rivers, roads, trails, etc. It is possible to measure distance on line features.
- Polygons: They are used for the representation of features that cover a particular area of the earth's surface, for example, a country. It is possible to measure area and perimeter of polygons.

## **2. PROJECT GOALS**

---

### **2.1. OVERVIEW**

An overview of the project goal definition document is presented below. The different sections which compose it are the following:

- Project definition: Statement of the goals of the project, specification of its functional aspects and scope.
- Final product: Description of the products that are to be developed on the project.
- Project realization: Definition of the different activities to be carried out in order to fulfill the goals of the project.
- Organization and team: Description of the Work Team that will carry out the project, as well as its organizational structure and human resources plan.
- Execution conditions: Project related work conditions, definition of the criteria over which product reception as well as the Change Control.
- Planning: Establishing of the phases and tasks needed to carry the project, together with orientative dates for their realization.
- Project budget: Determination of the budget of the project and the associate costs.

### **2.2. PROJECT DEFINITION**

#### **2.2.1 Goals**

The main goal of this project is the combination of geospatial and semantic technologies to build a location aware service based on linked data principles.

The main purpose of the service is to provide the users the capacity to share their knowledge in form of routes and points of interest and to enrich the information obtained using data from the web. Then the generated knowledge can be used to recommend the users new routes or points of interests based on their location and preferences.

The knowledge obtained on the system will be published following the Linked Open Data best practices, so that it can be used by other developers or interested groups.

In addition, it will be possible to obtain an objective difficulty rating for the routes or trails that the users share in the system, relieving from the users the problem of rating their own routes and eliminating possible subjective interpretations.

## 2. PROJECT GOALS

Together with this, several other services will be provided, allowing users to follow trails from others, to post notes at certain location and to obtain in any moment information regarding their surroundings; all in order to build a GPS community around this service.

So, in short, the goals of the project can be stated in the following way:

- To build a semantic location-aware service
- To publish the data of the system following the pattern of Linked Open Data
- To create a model for the semantic representation of trails and points of interest
- To build a service that allows the correction and evaluation of a route
- To build a GPS community based on the location-aware service

### 2.2.2 Scope of the project

In order to fulfill the stated goals, a GIS system based on semantic data will be built. The scope of the project is limited to the following:

- Research over the areas, technologies and standards that the project touches.
- Design of a model for the semantic representation of trails and points of interest and a infrastructure that uses and supports this model.
- Development of a service that makes use of the researched technologies and standards and that follows the previously created design.

#### Scope of the research

Research will be carried over several topics, listed below in order of relevance:

- The Semantic Web
- Linked Open Data
- GIS
- GPS devices
- Geographic information standards
- Development tools and platforms
- Similar systems

#### Scope of the development

The design and development will cover the following issues:

- Design and creation of a semantic ontology
- Development of a location aware web service
- Creation of a web application to access the service
- Development of a mobile application to complement the web
- Extraction of data from third party sources on the web

## 2.3. FINAL PRODUCT

The project will result in a GIS system which relies on a semantic back end, that is, it stores the data on RDF format and is able to reason over it. The system will be composed by a server, a web client and a mobile application.

This section describes the characteristics of the products that will compose the system.

### 2.3.1 Server

The server will be formed by two main components, a semantic data store and a web server. The web server will expose the a endpoint that allows direct read-only access to the database, so that other developers are able to query it. In addition, the server will have a NoSQL database used to store sensible information about the users of the system, with the purpose of providing secure authentication.

#### **Semantic data store**

The main function of this component will be the storage of the information existing on the system. Since semantic databases - called triplestore because they store the data using a data structure called triples - use ontology oriented data modeling (see section 3.3), the knowledge kept will not belong solely to domain related data, but will also include a ontology that represents the schema of the database and the relations among the resources.

This data store will be accessed through the HTTP protocol, even locally. In fact, it will be possible to query the data base through a read-only endpoint, accessible using HTTP. This way, any developer will be able to use the data in their application directly, without having to follow any predetermined access functions.

Besides, the component will be complemented by a reasoner. This reasoner will be able to infer new knowledge from the existing one. In this context, this means that it will be able to analyze the statements in the database and using the ontology as a basis it will be able to produce new statements.

In addition to regular reasoning, it will also be possible to reason spatially over the data. This way, it will be possible to use the spatial representations of resources in the system to make spatial queries, based on distance and geometric relations.

#### **Web server**

The web server is the component that will contain most of the functionality of the server. First, it will implement most of the functions encountered in typical web systems, including user authentication, media upload and read, create and modify operations. All these functions will be exposed through a API, following the REST style.

In addition to these basic functionality, the server will also be in charge of receiving the information about trails and points of interest and analyzing it to obtain additional data. This includes applying a mathematical method to calculate a difficulty score for trails.

## 2. PROJECT GOALS

The API will also expose function allowing spatial queries without any need of previous knowledge or expertise on this area. For example, a function will allow asking for all the points of interest at a certain distance from another feature. This queries will be limited to retrieval of routes, points of interest and geolocated notes.

The spatial information that the API provides will be exposed following the GeoJSON format.

The server will communicate with the data store using an HTTP interface and sending SPARQL queries for managing the data. In addition to this, the server will carry the task of analyzing the profile of each user and recommending routes that can interest them.

In addition to the mentioned functionality, the server will also expose a HTML5 web socket based API that the mobile application will use to provide real time services to the users.

### 2.3.2 Web application

The web application will be the main interface through which the users will be able to access the service. The web application will be developed taking into account that it will have to be visualized correctly in all kind of devices and that minimum loading times are desired.

The client will allow users to upload routes and points of interest from GPX files, a standard format for the interchange of GPS information (see section 3.9). In addition, user will also be allowed to draw the routes on a map or mark the points of interest.

It will be possible to search information about the different features and users of the system, taking advantage of the semantic data, which allows to easily build search engines which go beyond the classic keyword search. The application will offer a detailed view of the data and the possibility to comment on the different user generated features. All this information is to be presented in a easy to read format, that is, using interactive maps to show geographical information.

In addition to all of these, the web will provide a interactive map of the world. Moving and zooming this map will show the different routes and points of interest that the current view of the map covers, this way it will be possible to explore the data on the system without the need of a text search.

A exclusive functionality that the client will offer is a interactive editor. This editor will allow drawing trails and points of interest, providing a way to share knowledge without the need of a file recorded from a GPS. In addition, it will be possible to edit existing routes or trails imported from a file, providing the users with a way of correcting possible GPS generated errors.

The editor will allow the following functions:

Draw a route on the map: By repeatedly clicking on the map, specifying the coordinates which form the trail, the users will be able to generate data without a need of GPX files, however, it will be impossible to analyze this trails due to the lack of altitude data.

Mark a point of interest in the map: With a single click on the map, it will be possible to mark a point of interest in the specified point. The user will be prompted with a form, and by filling it, the point of interest will be uploaded to the server.

Edit a existing route: Whether it is downloaded from the platform or imported from a file, it will be possible to edit the coordinates of a route by dragging the points that form it.



Fusing two routes: It will be possible to join the end of a route with the start of another to create a new route.

### **2.3.3 Mobile application**

The mobile application will provide the users with a mean of viewing and browsing though the data on the system, in a similar manner to the web client. However, many of the functions present on the browser base client, for example the editor, will no be present on the mobile client, for the web application will be designed to be usable in mobile browsers.

In addition to the basic functionality, this component will allow the users to record their own trail as they traverse it using the GPS on the mobile device. It will be possible to upload this record to the server or to export it as a GPX or GeoJSON.

The application will also provide a function that allows user to receive real time notifications about the nearby points of interest and geolocated notes. These features will be filtered according to the user preferences, before notifying the user.

It will also be possible to follow the trails that another user has shared, with the mobile application sending a notification to the person using it every time that he/she deviates to much from the established route and indicating how to correct its trajectory.

Finally, the application will allow the creation of geolocated notes. To do this, the user will simply need to write a text, take a photo or record a video and after specifying some privacy settings, the note will be recorded in the system on the current coordinates of the user.

## **2.4. PROJECT REALIZATION**

### **2.4.1 Realization methodology**

The development method to be followed through the project consists on a prototype based iterative development methodology. This procedure allow to develop a project splitting it into different phases and developing a functional part of the project at each phase. Figure 2.1 shows a graphical conceptualization of this development methodology.

Thanks to this working methodology, it should be possible to develop independently every components that forms the system and it will be possible to realize effective unit testing over these components.

The development of the project has been split into six phases, whose tasks are listed below. A Work breakdown structure containing the tasks of the project can be found in figure 2.2.

#### **Project opening**

This phase consists on the start, organization and planning of the project. The tasks on this phase are not directly related to the development of the final product, yet they are necessary for the development of the project. The tasks in this phase are the following:

## 2. PROJECT GOALS

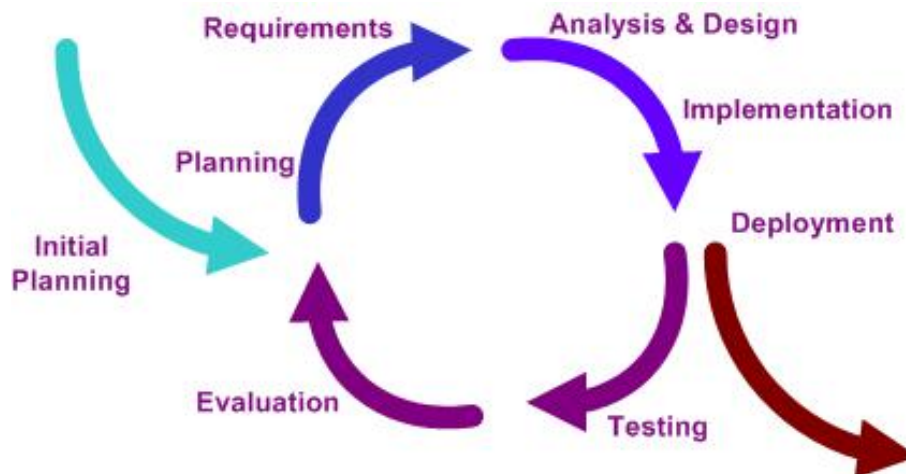


Figure 2.1: Iterative development phases

source: <http://www.wikipedia.org/>

### T1.1 Planning and organization

The resources necessary for the realization of the project will be determined and the tasks needed to carry it out will be organized. The development of the project through time will be established.

### T1.2 Monitoring

This task will be carried through the whole project. The current status of the project will be monitored to determine the possible existing issues and to ensure that the goals are being reached.

## **Initial research**

The research related with the areas that the project touches will be carried out. The goal of the phase is to obtain the sufficient information to develop a efficient and valid system.

### T2.1 Research on the Semantic Web and Linked Open Data

Research will be done on topics and technologies related to the Semantic Web and Linked Open Data. Other topics will also be investigated, for instance, the representation of spatial data using RDF.

### T2.2 Research on technologies

GIS systems, and web and application development tools will be investigated with the goal of establishing an adequate development environment for the project.

### T2.3 Analysis of standards and similar system

Existing standards for trail and point of interest representation will be analyzed, as well as the use of this on other platforms and the functionality that similar services offer.

### **Base system development**

The design of the architecture of the system and the development of the software that will be present on the server will be done in this phase. The tasks consist mostly on design, coding and testing.

#### **T3.1 System design**

The technological architecture of the system and of its components will be designed.

#### **T3.2 Test design**

The tests to be carried on every component will be determined.

#### **T3.3 Database installation and testing**

The semantic database will be installed and tested to guarantee that the desired functionality is granted.

#### **T3.4 Web server implementation and testing**

The basic functionality of the web server and related software will be developed and tested. An example of these additional software pieces is the data store connector.

#### **T3.5 API implementation and testing**

The functions that the API of the web server exposed will be developed and tested.

### **Web application development**

The web application will be developed using the server produced on the previous phase. Tasks involve mainly interface design, web development and testing.

#### **T4.1 Interface design**

The design of the web interface will be done and the needed resources, such as images and fonts, will be obtained.

#### **T4.2 Implementation and testing**

The logic on the web application will be coded and test will be carried out over the developed functions.

#### **T4.3 Editor implementation and testing**

The functionality regarding the map editor will be developed and tested.

### **Mobile application development**

The mobile application will be developed using the API of the produced server . Tasks involve mainly interface design, HTML5 development and testing.

#### **T5.1 Interface design**

The design of the mobile interface will be done and the needed resources, such as images and fonts, will be obtained.

## 2. PROJECT GOALS

### T5.2 Implementation and testing

The logic on the mobile application will be coded and test will be carried out over the developed functions.

### Project finalization

The last phase will take care of the tasks related to project closure and deployment.

#### T6.1 System deployment

The web server will be deployed to a accessible URL and the mobile application will be published on the corresponding markets.

#### T6.2 Project closure

A final check of the completed goals will be done, future lines of work and possible upgrades will be established and the project will be closed.

### 2.4.2 Intermediate products

The project is divided on several phases. An intermediate product is obtained from each of the phases or tasks. Table 2.1 offers a reference of the intermediate products of the project and the tasks or phases they relate to.

Table 2.1: Intermediate products of the project.

Name	phase	task code
Technological research	2. Initial research	T1.1, T1.2
Ontology specification	3. Base system development	T3.1
System architecture	3. Base system development	T3.1
Design of the web server	3. Base system development	T3.1
Design of the web application	3. Base system development	T3.1
Design of the mobile application	3. Base system development	T3.1
Test suit	3. Base system development	T3.2
Design of the web application interface	4. Web application development	T4.1
Design of the mobile application interface	5. Mobile application development	T5.1

These intermediate products are all documents, their contents are listed below:

Technological research: Contains a list and of the main technologies that will be involved in the project and a description for each of them.

Ontology specification: Describes the classes, properties and relations that will appear on the ontology and how they are used to model the data on the system.

System architecture: Defines which are the components that form the system and how they are related among them.

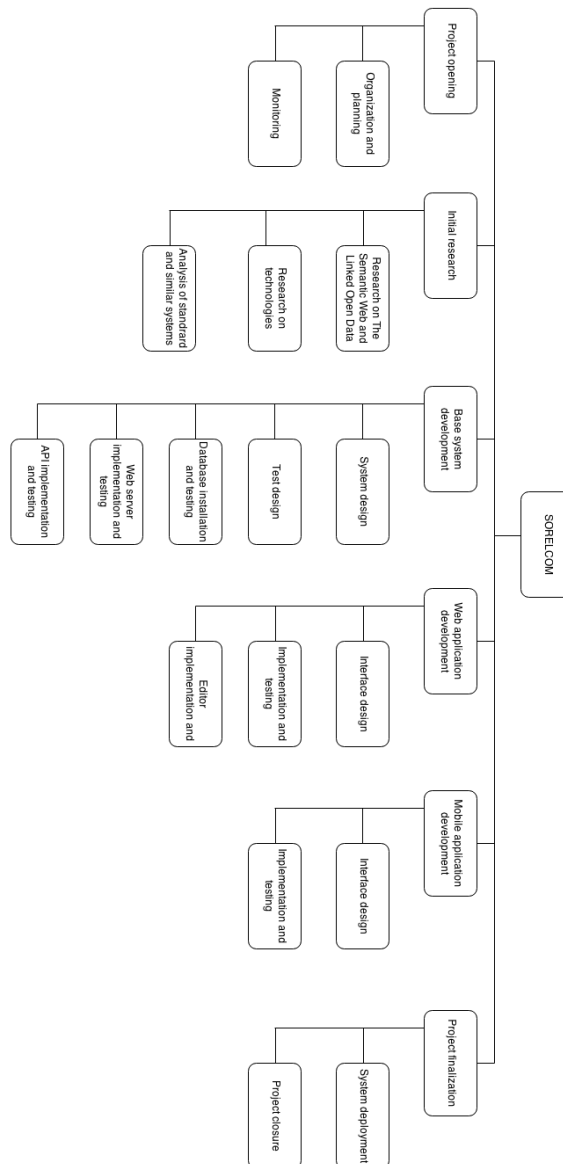


Figure 2.2: Work breakdown structure

Design of the web server: Describes the logical architecture of the web server.

Design of the web application: Describes the logical architecture of the web application.

Design of the mobile application: Describes the logical architecture of the mobile application.

Test suite: Describes the tests to be carried on each of the components of the system.

Design of the web application interface: Describes the appearance of the web interface.

Design of the mobile application interface: Describes the appearance of the mobile interface.

## 2.5. ORGANIZATION AND TEAM

### 2.5.1 Organizational structure

The organizational structure is relatively simple. The organization involves two main groups, the project manager and the work team.

The project manager is the one in charge of the organization of the project and his function is the direction of the work to do. It takes care of the monitoring and planning of the work as well as most of the validations.

The work group carries out the tasks related to the development of the products of the project. This tasks require research, designs, testing and implementation. The work group will be formed by a single student performing all the required roles.

The organizational schema including all the roles on the project is presented in figure 2.3.

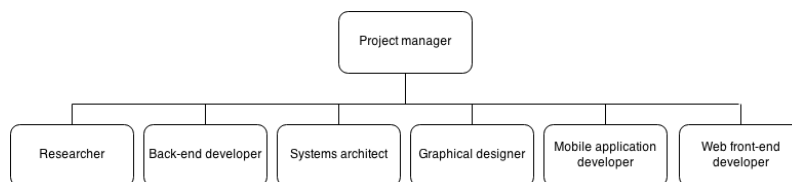


Figure 2.3: Organizational structure

### 2.5.2 Human resources plan

The organization will be formed by the following profiles, each related to the different areas of competence that this project comprises.

#### **Project manager**

The project manager is the person in charge of the opening, organization, monitoring and closure of the project. In addition, it is the person in charge of communicating with the management team.

The skills of the project manager should include at least the following:

- Planning
- Organizing
- Communication

In addition, it is convenient for it to have knowledge on the areas that the project touches.

#### **Researcher**

Due to the innovative nature of the project and the amount of work related to the semantic web, the profile of a researcher is required. The researches takes care of investigating the current state in the areas that the project touches and elaborating state of the art documents needed for the development of the project.

The researcher needs to have knowledge on the areas related to the project, that is, The Semantic Web and Linked Open Data. Learning and writing capabilities are also convenient.

### **Back-end developer**

This profile takes care of developing the back-end of the application, the web server. Due to the nature of its work and the technologies to be used in the project, the following knowledge is required.

- JavaScript programming skills
- Knowledge of the NodeJS programming environment
- Knowledge of the SPARQL query language

### **Front-end web developer**

The web developer takes care of creating the web application. Its work involves development of web based graphical user interfaces, due to this and the technologies chosen for the project, the following skills are required.

- HTML5, CSS3 and JavaScript
- The AngularJS web development framework
- The CSS framework Twitter Bootstrap
- The JavaScript map development library Leaflet

### **Mobile application developer**

This profile takes care of developing the mobile application. Proficiency on mobile application programming is needed, however, it is also necessary to know how to use HTML5 native development tools. The required skills can be summarized in the following:

- HTML5, CSS3 and JavaScript
- The HTML5 web sockets protocol
- The HTML5 native mobile development library Phonegap
- The JavaScript map development library Leaflet

### **Graphical designer**

The graphics designer takes care of designing the interface of the application. In order to present this design it is required that the designer knows how to use some graphic design tool such as GIMP or Photoshop.

### **Systems architect**

The systems architect takes care of designing the system architecture and the logical design of each component. It should have the following skills:

- Software design
- The modeling standard UML

### 2.6. EXECUTION CONDITIONS

The usual workplace will be the DeustoTech office located in the 4th floor of the ESIDE building in the university of Deusto. In this environment ethernet network connection is provided as well as a power source for computers. Still, a personal laptop is to be used for the development instead of office computer. Mobile devices used for testing will also be properties of the student.

The weekly schedule is of 18 hours, divided in three hours per day.

#### 2.6.1 Hardware

Since web and mobile applications are to be developed, there is a need to carry out tests and preview results in various kind of devices, such as mobile phones and tablets. The hardware that is available is the following:

- Laptop computer, Inter i7 processor, 15 inch screen
- Android Smartphone, Nexus 5 model
- iPad tablet

#### 2.6.2 Software

The software tools to be used in the project will be open source in most cases, and just free in the rest. In this section the software that is available is listed:

##### Programming tools

- NodeJS 0.10.25 - JavaScript runtime environment
- NPM 1.3.10 - NodeJS package manager
- Bower 1.3.3 - JavaScript library management tool
- Yeoman 1.1.2 - JavaScript project scaffolding tool
- Express 3.4.3 - NodeJS web development library
- AngularJs 1.2.11 - JavaScript web application development framework
- Leaflet 0.7.3 - JavaScript map developing library
- Twitter Bootstrap 3 - CSS responsive web development framework
- SASS 3.3.8 - CSS development framework
- Phonegap 2.9.1 - Mobile HTML5 programming platform

##### Web browsers

- Google Chrome 27
- Mozilla Firefox 22
- Safari 6
- Opera 12
- Internet Explorer 10



## **Others**

- Ubuntu 14.04 - Open source desktop operating system
- Atom 0.105 - JavaScript based code editor
- Parliament 2.6 - Semantic spatial database, reasoner and endpoint
- GIT 2.0.0 - Version control system
- GIMP 2.8 - Image manipulation program

### **2.6.3 Change control**

The changing requirements or petitions will follow the proceeding described below:

1. The change will be communicated to the work group, in case it comes from outside stakeholders.
2. The work group will study the request and evaluate the impact of the potential change in the project.
3. The work group will produce a report that will be sent to the project manager.
4. The project manager will take the final decision on whether to accept the changes or not.

### **2.6.4 Product reception**

During this project two types of product will be created, documents and software. Both of them will be accepted following different procedures.

The documents will follow a previously determined structure. This structure requires each document to have a index, a figure and table index and a list of references used. There is only one requirement regarding the content, documents must start with a overview of the contents of the paper. The author and title of the document must appear at the beginning.

These documents will be sent to the project manager, who will have to accept it in a period of 5 working days. In case of a lack of response, the work team will assume that the document has been accepted so that the development of the project is not stopped.

The software on the other side will follow more rigorous criteria. On the task SYS2, a document specifying the test suite for each component will be created. Software components will only be accepted when they have passes all the specified tests.

## 2.7. PLANNING

This section presents several aspects related to the project planning. The planning considers a work schedule of 3 hours each day. The project starts Monday 3 of march 2014 and will end the 10th of July of the same year.

### Workload estimation per profile

The estimated work to be done by each profile is shown on this section.

Table 2.2 contains the work to be done by the project manager.

Table 2.2: Workload estimation for the project manager.

Task	hours
T1.1	6
T1.2	16
T6.2	3
Total	25

Table 2.3 contains the work to be done by the system architect.

Table 2.3: Workload estimation for the system architect.

Task	hours
T3.1	45
T3.2	6
Total	51

Table 2.4 contains the work to be done by the researcher.

Table 2.4: Workload estimation for the researcher.

Task	hours
T2.1	15
T2.2	12
T2.3	15
Total	42

Table 2.5 contains the work to be done by the back-end developer.

Table 2.6 contains the work to be done by the front-end web developer.

Table 2.7 contains the work to be done by the mobile application designer.

Table 2.8 contains the work to be done by the graphics designer.

Table 2.5: Workload estimation for the back-end developer.

Task	hours
T3.3	3
T3.4	30
T3.5	24
T6.2	3
Total	60

Table 2.6: Workload estimation for the front-end web developer.

Task	hours
T4.2	27
T4.3	21
Total	48

## Network and Gantt diagram

Figure 2.4 shows the gantt diagram for the project, which establishes the temporal planning. Figure 2.5 shows the network diagram of the tasks on the project, in order to identify the dependencies among them.

## Work plan

The work plan obtained after establishing the schedule and the interdependencies among tasks is presented in table 2.9.

## 2. PROJECT GOALS

Table 2.7: Workload estimation for the mobile application developer.

Task	hours
T5.2	21
Total	21

Table 2.8: Workload estimation for the graphics designer.

Task	hours
T4.1	12
T5.1	9
Total	21

Table 2.9: Real work plan.

Identifier	start	end	days	profile	work (h)
T1 - Project initiation					
T1.1	03/03	05/03	2	Project manager	6
T1.2	10/03	13/06	8	Project manager	16
T2 - Initial research					
T2.1	05/03	12/03	5	Researcher	15
T2.2	19/03	25/03	4	Researcher	12
T2.3	12/03	26/03	5	Researcher	15
T3 - Base system development					
T3.1	26/03	18/04	15	System architect	45
T3.2	18/03	22/04	2	System architect	6
T3.3	22/04	23/04	1	Back-end developer	3
T3.4	23/04	08/05	10	Back-end developer	30
T3.5	08/05	21/05	8	Back-end developer	24
T4 - Web app. development					
T4.1	21/05	28/05	4	Graphics designer	12
T4.2	02/06	16/06	9	Front-end developer	27
T4.3	26/06	08/07	7	Front-end developer	21
T5 - Mobile app. development					
T5.1	28/05	02/06	3	Graphics designer	9
T5.2	16/06	26/06	7	Mobile app developer	21
T6 - Project finalization					
T6.1	08/07	09/07	1	Back-end developer	3
T6.2	09/07	10/07	1	Project manager	3

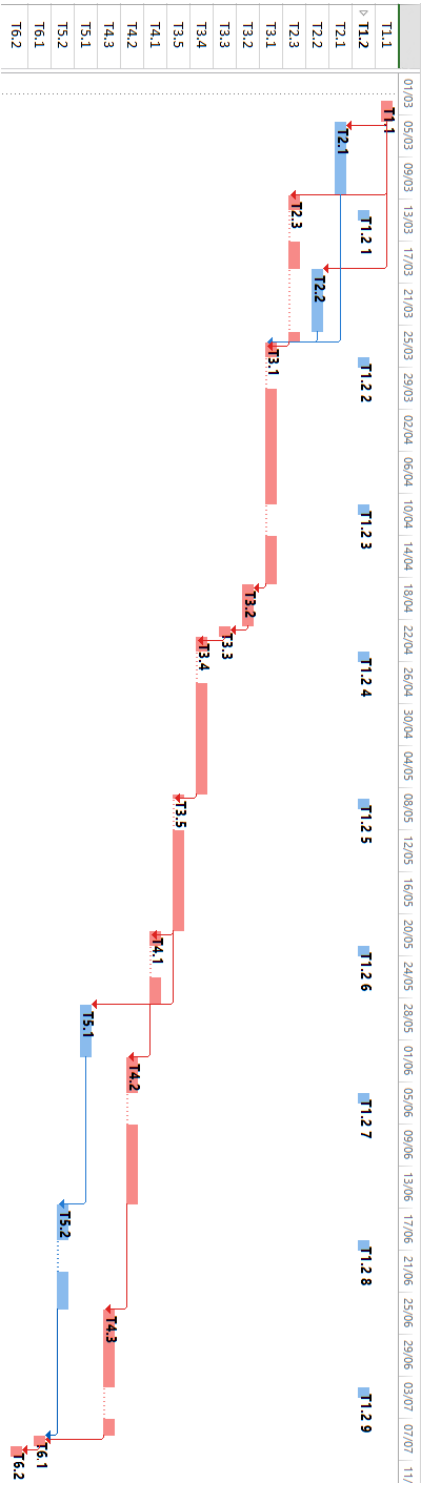


Figure 2.4: Gantt diagram

## 2. PROJECT GOALS

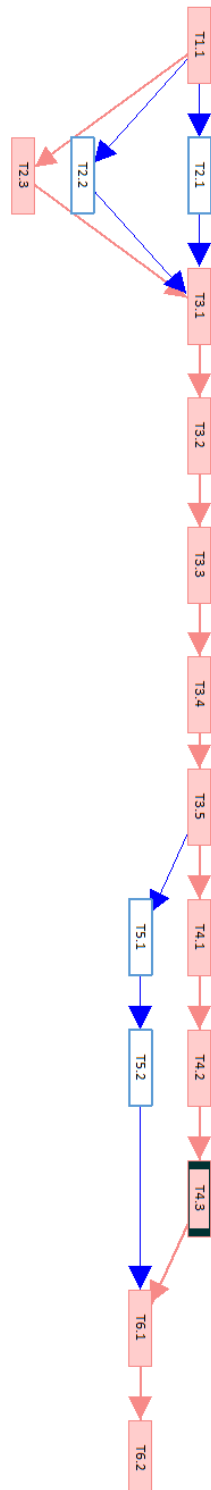


Figure 2.5: Network diagram

## 2.8. PROJECT BUDGET

The costs associated with the project are divided into two main sources. The first group is the hardware related costs, which comprises the budget required for the physical technological resources used in the realization of the project. Table 2.10 reflects these costs.

The second cost is the one destined to the human resources team. An overview of the costs generated by each profile may be found in table 2.11.

No budget will be destined to cover software cost, for the tools used on the project will all be either open source or free-ware. A summary of the project budget can be found in table 2.12.

Table 2.10: Hardware related budget.

Name	units	unit price (€)	cost (€)
Laptop computer	1	1000.00	1000.00
Android smart phone	1	450.00	450.00
iOS tablet	1	500.00	500.00
Total			1950.00

Table 2.11: Human resources budget.

Name	work (h)	cost (€)
Front-end web developer	48	240.00
Back-end developer	60	300.00
Mobile application programmer	21	105.00
Graphics designer	21	105.00
Researcher	42	210.00
System architect	51	255.00
Project manager	25	125.50
Total		1340.00

Table 2.12: Summary of budget.

Name	cost (€)
Hardware	1950.00
Human resources	1340.00
Total	3290.00





## 3. TECHNOLOGICAL RESEARCH

---

### 3.1. OVERVIEW

This chapter presents the results of the initial technological research done to carry out the project. In here, the main technologies used in the project are explained in detail, as well as the benefits of using them and the reasons for doing so.

The most relevant tools and frameworks used on the project are the following:

- RDF
- OWL
- SPARQL
- GeoSPARQL
- NodeJS
- AngularJS
- Leaflet
- Phonegap
- GPX
- GeoJSON

There are many other software tools and libraries are used through the project, however, only the most relevant of them, and the ones on which the initial investigation has been carried are shown in this chapter. The rest of the tools are briefly described in chapters 5, 6 and 7.

### 3.2. RDF

RDF stands for Resource Description Framework, and it is a framework for expressing information about resources. These resources can be anything, documents, people, physical objects, even abstract concepts.

RDF is intended for situation in which data is to be processed by machines and not simply displayed to humans, due to this, it is very useful for the goals of the Semantic Web. It can be used to publish and interlink data on the web. For example, retrieving a resource which represents a person, say Bob, could provide us with the fact that Bob knows another person, say, Alice, who is represented by her URI. Retrieving the resource representing Alice may then yield links to datasets about other persons and so on. A computer can automate this process and follow these links, aggregating data from different resources. This kind of uses are often known as Linked Data [69].

There are many reasons to choose using RDF, the following are some of them:

### 3. TECHNOLOGICAL RESEARCH

- Adding machine readable information to Webpages. This allows them to be displayed in enhanced format or to be processed by roaming agents.
- Enriching datasets by linking them to other sources of data.
- Interlinking APIs so that clients are able to discover new sources of information.
- Using the datasets currently published as Linked Data
- Providing a standards compliant way of exchanging data between applications.
- Interlinking various datasets within an organization and allowing cross-dataset queries by using SPARQL.

## RDF data model

Data in RDF is formed by statements. These statements are called triples and follow a subject-predicate-object structure. A statement on RDF represents a relationship between two resources, the subject and object represent the resources being related while the predicate represents the nature of the relationship. These relationships are phrased in a unidirectional way, from subject to object, and are called properties.

The same subject is often referenced in multiple triples and can of course be the object of other triples. These ability to have the same resource as subject and object in different triples makes it possible to find connections between RDF resources and is one of the most powerful features of the framework. A set of triples can be viewed as a connected graph, in figure 3.1 an example of the visual representation of an RDF graph is given.



Figure 3.1: An RDF graph describing a person

source: <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>

There are three types of RDF data that can occur in triples: IRIs, Literals or Blank nodes [69].

## IRIs

IRI is an abbreviation for International Resource Identifier, which is used to identify a resource. On RDF, IRIs may appear in any component of triples, be it subject, object or predicate. The URLs (Uniform Resource Locators) which identify web pages are a form of IRI, however other forms of IRI

can identify a resource without specifying its location. An IRI is a generalization of URIs allowing the use of non-ASCII characters unlike the latter. Thus, an IRI is defined as a set of characters unreserved characters which can include characters of the UCS [5, 24](Universal Character Set, [ISO10646]).

An example of a IRI from dbpedia can be the following: `http://dbpedia.org/resource/Leonardo_da_Vinci`. Even if RDF does not explicitly specify what a certain IRI represents, they can take meanings from vocabularies or conventions.

In SPARQL (see 3.4) and in some RDF serialization models prefixes can be defined. A dataset, for example `http://example.org`, could be given the name `my`. By giving this name we can reference `http://example.org/Bob`, a resource on the dataset as `my:Bob`. This is useful for it helps to avoid clutter on the query and helps human understanding, thus, from now on, prefix notation will be used to express datasets and vocabularies on the document.

## Literals

Literals are the basic values that are not IRIs. Literals are associated with data types, for example, the Integer and Float data types are equivalent to the traditional types from the programming languages. The string typed literals may optionally have a language associated allowing internationalization. Most of the regularly used data types are defined by XML Schema [49].

One particularity of RDF data types is that they can be defined on ontologies. Due to this, it is possible to find custom types, such as a string representing the serialization in WKT [19] format of a geometry.

An example of different types of triples, including literals can be found in listing 3.1.

```

1  <my:Bob> <foaf:name> "Bob"
2  <!--The object is a literal of type String -->
3  <my:Bob> <rdf:type> <foaf:Person>
4  <my:Bob> <foaf:age> "19"^^xsd:Integer
5  <!-- The literal is of type Integer -->
```

Listing 3.1: Different types of triple statements.

## Blank nodes

Blank nodes are used to refer to resources who don't have or need a global identifier (an IRI). For example, we may want to refer to the right arm of a person, however, we may think that it is too verbose or unnecessary to actually give a identifier to this arm, so we could use a blank node to represent it, as illustrated in listing 3.2. In this example it makes sense to have a blank node, for the right arm is directly dependent on the person, an arm (usually) makes no sense detached from a person.

```

1  <my:Bob> <rdf:type> <foaf:Person>
2  <my:Bob> <my:hasRightArm> :_b1
3  :_b1 <rdf:type> <my:Arm>
```

Listing 3.2: Triple statements with a blank node.

## RDF vocabularies

The data model of RDF does not make any assumptions about the meaning of the IRIs representing the resources. However, in the reality, RDF is usually used in conjunction with a series of vocabularies or other conventions that give semantic information about these resources.

Many vocabularies are currently considered standard due to their wide use among Linked Open Data systems. Some of these ontologies are the following.

Friend of a Friend (FOAF): One of the first vocabularies used worldwide, it is used to represent persons, the properties that identify them and the relations among them [12].

Dublin Core: A metadata element set for describing a wide range of resources. Contains properties such as the creation date of the resource [39].

schema.org: A vocabulary for marking up web pages so that search engines can more easily find them [57].

## RDF schema

RDF Schema (RDFS) is an extension of RDF. It provides a data-modeling vocabulary for RDF data, in this way, RDFS extends RDF by giving externally specified semantics to specific resources, semantics which cannot be captured in plain RDF [13]. RDF Schema provides information about interpretation of the statements in a data model, however, it does not constrain the syntactical appearance of RDF descriptions.

The extension provides mechanisms to describe groups of related resources and the relationships between these resources, as well as classification hierarchies. The class and property system that can be expressed is similar to the type systems of object oriented programming, however, instead of defining classes in terms of the properties their instance have, they are defined in terms of classes of resource to which they apply [11].

RDF schema is written in plain RDF, thus it takes form of a vocabulary. The classes and properties provided are listed below:

### Classes

`rdfs:Resource` All things described by RDF are called resources and are instances of this class. It is the class of everything and all other classes are subclasses of it.

`rdfs:Class` It is the class of all the resources that are classes, thus, `rdfs:Class` is an instance of `rdfs:Class`.

`rdfs:Literal` It is the class of literal values such as strings and integers. It is an instance of `rdfs:Class`.

`rdfs:Datatype` It is the class of datatypes, such as `integer`.

`rdf:langString` It is the class of language tagged string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:HTML` It is the class of HTML string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:HTML` It is the class of HTML string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:XMLString` It is the class of XML literal values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

## Properties

`rdfs:range` It is used to indicate that the values of a property are instances of a certain class.

`rdfs:domain` It is used to indicate that any resource that has the given property is a instance of a certain class.

`rdfs:type` It is used to state that a resource is a instance of a certain class.

`rdfs:subClassOf` It is used to state that all instances of one class are also instances of another.

`rdfs:subPropertyOf` It is used to state that all resources related by a class are also related by another.

`rdfs:label` It is used to provide a human readable version of the name of the resource.

`rdfs:comment` It is used to provide a human readable description of a resource.

## Use of RDFS

RDFS provides with a vocabulary to express relations among RDF resources, which allow for reasoning and inference over datasets. However, this vocabulary is very limited, it does not allow much more than the definition of classification hierarchies. These hierarchies are defined in regular RDF triples, as shown in listing 3.3.

```

1  <foaf:Person> <rdf:type> <rdfs:Class>
2  <my:knows> <rdf:type> <rdf:Property>
3  <my:knows> <rdfs:domain> <foaf:Person>
4  <my:knows> <rdfs:range> <foaf:Person>
5  <my:friendOf> <rdfs:subPropertyOf> <my:knows>

```

Listing 3.3: RDFS class and property hierarchy.

In addition to ability to express relations, RDFS provides additional constructs for the representation of resources. The vocabulary defines containers, such as bags and sequences, in addition to properties to indicate membership to these. Besides collections, the ontology defines a more sophisticated mechanism for data structuring; RDF lists, which are similar to the Linked List data structure. However, the usage collections bring an additional layer of complexity to the dataset, thus they are scarcely used.

## Why RDF?

Using RDF can greatly benefit this project. It can give a representation of the relationships on the system which can be easily analysed by machines. This can help building important functions, such as the recommendation system.

However, the use of RDF on this project is almost mandatory, for one of the goals is achieving data interoperability and publishing the data as Linked Open Data on the web. Anyway, the expressive level provided by RDF schema is not enough for the needs of the project, so another vocabulary must be used.

## 3.3. OWL

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies in the World Wide Web [4]. The language is designed for applications that need to process the content of the information instead of just presenting it to humans. OWL provides greater machine interpretability than RDF or RDF schema by providing additional vocabularies along formal semantics.

OWL is used to explicitly represent the meaning of terms in vocabularies and the relationships among these terms. These representations are called ontologies [45]. The standard is part of the stack of W3C recommendations for the semantic web, together with XML, XML Schema, RDF and RDF Schema.

OWL provides three sub-languages with increasing expressive power. Each subset of OWL includes all the previous ones, in addition to RDFS and RDF, so that there is no loss on expressive power as illustrated in figure 3.2.

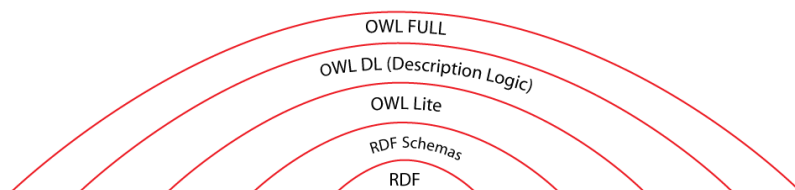


Figure 3.2: The OWL levels of expressivity

source: <http://www.w3.org/>

## OWL Lite

OWL Lite is designed to support those users who need only a classification hierarchy and simple constraints. For example, while supporting cardinality constraints, it only allows cardinality 0 or 1, for the sake of simplicity. Due to this it should be simpler to provide tool support for OWL lite than to its other, more expressive, relatives. It also has a lower level of formal complexity than its more expressive siblings.

This sub-language contains all the features of RDF schema, such as category definitions through classes and hierarchical classification through subclasses, plus additional restrictions and expressions. OWL Lite provides tools for defining equality and inequality relations, property characteristics, property restrictions, restricted cardinalities and class intersections.

### Equality and Inequality

They are used to indicate equality and inequality relations among classes. One example

**equivalentClass** Two classes may be stated as equivalents. This equality is used to create synonyms.

**equivalentProperty** It is used to define synonymous properties, the same way as with classes.

**sameAs** It states that two individuals are the same.

**differentFrom** The opposite to **sameAs**, used to indicate that two individuals are different.

**allDifferent** It can be used to state that a number of individuals are different from each other.

### Property characteristics

Property characteristics are special identifiers in OWL that provide information concerning the properties and their values. These characteristics are placed on properties, with relation to no class, as in listing 3.4.

```

1  <owlx:ObjectProperty owlx:name="adjacentRegion" owl:symmetric="true">
2    <owlx:domain owlx:class="#Region" />
3    <owlx:range owlx:class="#Region" />
4  </owlx:ObjectProperty>

```

Listing 3.4: OWL property characteristic example.

**inverseOf** Describes that a property is inverse to another. If A is related to B by the property P1 and P2 is the inverse of P1, then B will relate to A by property P2.

**TransitiveProperty** States that a property is transitive. For example, if A is related by transitive property P to B and B is related by P to C, then A will also be related by P to C.

**SymmetryProperty** If a property is stated to be symmetric, then if A is related to B by the property, B will also be related to A by the same property.

**FunctionalProperty** If a property is functional, then it can have no more than one value for each individual. This means that there will be no two triples of the same individual containing this property as a predicate.

**InverseFunctionalProperty** It states that the inverse of a property is a functional property.

### 3. TECHNOLOGICAL RESEARCH

```
1 <owlx:Class owlx:name="Wine" owlx:complete="false">
2   <owlx:Class owlx:name="&food;PotableLiquid" />
3   <owlx:ObjectRestriction owlx:property="#hasMaker">
4     <owlx:allValuesFrom owlx:class="#Winery" />
5   </owlx:ObjectRestriction>
6 </owlx:Class>
```

Listing 3.5: OWL property restriction example.

#### Property restrictions

OWL Lite allows to place restrictions regarding on how properties can be used inside of instances of classes. One example can be found in listing 3.5

**allValuesFrom** This restriction is placed on a property in relation to a class. It indicates that a property on a particular class has a local range restriction associated with it. For example, the class *Person* may have a property *hasDaughter*, restricted to have all values from class *Woman*. This means that if an individual of type *person* is related by a relation *hasDaughter* to other individual, we can infer that this individual will be a woman.

**someValuesFrom** This restriction is placed on a property in relation to a class. It indicates that at least one value from the property is of a certain type. For example, the class *SemanticWebPaper* may have a property *hasKeyword* restricted to have some values from *SemanticWebTopic*. This would mean that a semantic web paper can have any number of keywords from any topic, as long as at least one of them belongs to a semantic web topic.

#### Restricted cardinalities

OWL includes a limited form of cardinality restrictions. These are placed on properties with respect to classes, that is, the restrictions constrain the cardinality of the property in instances of a specific class.

**minCardinality** If a minimum cardinality of 1 is stated on a property with respect to a particular class, then any instance of that class will be related to at least one individual by that property. For example, a person must have some sort of identifying card, thus, a minimum cardinality of 1 should be placed on the relation *hasIDCard* with respect to *Person*.

**maxCardinality** If a maximum cardinality of 1 is placed on a property with respect to a particular class, then any instance of that class will be related to at most one individual by that property. For example, a person should have at most one ID card, thus, the relation *hasID* would have a maximum cardinality of 1.

**cardinality** It is used as a convenience to indicate that a property has both a maximum and a minimum cardinality with the same values.

#### Intersection

OWL Lite offers a very limited expressibility for intersections through the property *intersectionOf*. With this expression one could express that a *EmployedPerson* is a intersection of a *Employee*



and a Person. An example for this is shown in listing 3.6

```

1  <owlx:Class owlx:name="WhiteWine" owlx:complete="true">
2    <owlx:IntersectionOf>
3      <owlx:Class owlx:name="#Wine" />
4      <owlx:ObjectRestriction owlx:property="#hasColor">
5        <owlx:hasValue owlx:name="#White" />
6      </owlx:ObjectRestriction>
7    </owlx:IntersectionOf>
8  </owlx:Class>

```

Listing 3.6: OWL property restriction example.

## OWL DL and OWL Full

OWL DL is the second among the OWL sub-languages. It includes all the expressive power of OWL lite and adds on top of it. Similarly OWL Full adds on top of OWL DL. However, OWL Full is usually deemed as undecidable, in fact, there are no reasoners that can guarantee a solution in finite time. Due to this, many reasoners for OWL DL, such as Pellet try to find a way to convert OWL Full ontologies to a more a previous level of expressivity, since most ontologists use the highest expressive power when they only need OWL DL at most [61].

## Why OWL?

RDF schema provides a very limited way of expressing ontologies, it only allows to build classification hierarchies. Due to this, using OWL to describe the ontology would be the most recommendable thing. More specifically, the expression level of OWL DL, which allows to describe disjoint relations between classes (among other relation) plus all of the OWL Lite characteristics listed above, will allow to build a complete and correct ontology.

## 3.4. SPARQL

SPARQL is the recursive acronym for SPARQL Protocol and RDF Query Language. As its name indicates, it is a protocol and query language for RDF data. The idea behind this language is to allow the querying over RDF datasets in a similar language in which triples are expressed [18].

SPARQL allows querying [55] and updating [59] of RDF datasets. Similar to RDF, this language is composed of triples, however among the subjects predicates and objects, variables can be introduced. The logic behind SPARQL is stating a set of triples and introducing a variable somewhere among these triples. By doing so, the query engine will look and retrieve for all the cases which satisfy the set of triples in the query. Nothing impedes to query for a certain triple without specifying any variable, which can be useful when the goal is to know if the triple actually exists, but it is not the usual case.

In SPARQL prefixes can be used to indicate name spaces. By using the keyword prefix, it is possible to specify a URI which will be prepended to all triples using that resource. Variables are

### 3. TECHNOLOGICAL RESEARCH

specified by using an interrogation mark (?) before an arbitrary name. Two examples can be found in listings 3.7 and 3.8.

```
1  SELECT ?s ?p ?o WHERE{
2      ?s ?o ?o .
3  }
```

Listing 3.7: SPARQL query for retrieving all triples on the dataset.

This example is very basic, it does not actually specify any concrete triple it states subject object and predicate as variables. Due to this, the query engine will match every existing statement in the database and will retrieve all triples.

```
1  PREFIX my: <http://example.org>
2  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5  SELECT ?name WHERE{
6      ?person rdf:type foaf:Person .
7      ?person foaf:name ?name .
8  }
```

Listing 3.8: SPARQL query for retrieving names on the dataset.

This case is slightly more elaborated than the previous one. First the keyword PREFIX appears, which specifies namespaces for the query, similar to how XML namespaces work. After the prefix declaration, there is a SELECT keyword. Here the operation to be done is specified, among the options of SELECT for regular queries, ASK for boolean queries, INSERT for data insertion and DELETE for data removal. After the select, the variables to be retrieved, bound in SPARQL terms, are specified. After that, the actual triples of the query are written. In this case a person variable is used to retrieve every resource of type Person in the dataset. Then if that resource contains a name, it is bound to the variable.

It is possible to use more advanced operations in SPARQL. Every triple statement is a implicit intersection, however, it is possible to use the keyword UNION as well. It is also possible to count the amount of occurrences of a variable, to order by variable values and even to filter with regular expressions. Utilities for retrieving only a certain subset of query matching elements are also provided, allowing to limit the result to a certain amount of triples or event to offset the start of the result to a certain triple.

## Why SPARQL?

Since the system will store data in RDF format, the use of SPARQL as a query language becomes obligatory. Still, there are several benefits for the use of SPARQL over a more traditional language such as SQL or over using a ORM. This protocol is designed for advanced queries, thus it allows to find complex relations among data in a relatively easy manner.

Besides, since SPARQL is a standard language, the same queries used for the system could be used to retrieve data from external datasets.

### 3.5. GEOSPARQL

RDF and SPARQL allow for reasoning in a large domain of applications, however, these reasoning is only concerned about relations explicitly represented in the datasets. This does not include the spatial relations we aim to represent in the system, such as nearby relations.

GeoSPARQL is a standard which aims to provide support from representing and querying spatial data over the semantic web. This standard has been presented by the Open Geospatial Consortium (OGC) and is designed modularly [2, 51]. The components that form the specification are listed below [3]:

- A vocabulary to represent spatial features, geometries and their relationships.
- A set of domain-specific spatial functions to use in SPARQL queries.
- A set of query transformation rules

#### Vocabulary

The ontology used to represent spatial data is domain independent in the sense that it is only concerned in representing spatial objects and relations, without specifying the type of resources it represents. Thanks to this it can be used to model any kind of GIS system, whether it represents nature, a country's road infrastructure or cultural points of interest.

The ontology provides two type of top-level classes, Features and Geometries. Features are simply entities in the real world with some spatial location. A feature can have any kind of spatial form that cannot be precisely defined, for example, a lake or a forest. A Geometry on the other hand is any geometric shape, such as a point or a polygon used as a representation of the spatial location of a feature.

Together with these types, a medium for representing the exact geometries of the object is provided. By using the properties `asWKT` and `asGML` it is possible to provide the exact spatial information as a WKT or GML [21] string, as in the listing 3.9.

```

1  <my:Point> <rdf:type> <geo:Feature>
2  <my:Point> <geo:hasGeometry> <my:geom>
3  <my:geom> <rdf:type> <geo:Geometry>
4  <my:geom> <geo:asWKT> "POINT(0, 0)"^^geo:wktLiteral

```

Listing 3.9: A feature in the GeoSPARQL vocabulary.

Besides, there is also a way to provide represent non-exact relation, using qualitative properties, such as the `within` property.

#### SPARQL extension

Exploiting the qualitative relations in the system is simple, a regular query can be used. However, in order to take advantage of the serialization of geometries, GeoSPARQL defines a series of functions that allow the manipulation and querying of spatial data. For example, the function `geof:distance` will return the shortest distance between two geometries.

### 3. TECHNOLOGICAL RESEARCH

This way, an extension for SPARQL is provided, which allows to find the implicit spatial relations in the dataset, as in the query shown in listing 3.10

```
1  SELECT ?p
2  WHERE{
3      ?p a geo:Feature
4      ?p geo:hasGeometry ?pgeo .
5      ?pgeo geo:asWKT ?pwkt .
6      ?w a geo:Feature .
7      ?w a my:Park .
8      ?w geo:hasGeometry ?wgeo .
9      ?wgeo geo:asWKT ?wwkt .
10     FILTER(geof:distance(?pwkt ?wwkt units:m)
11             < 3000)
12 }
```

Listing 3.10: Spatial query in SPARQL.

## Why GeoSPARQL?

While there are some semantic storage systems that implement their own spatial indexes and representation system there has been no standard in the semantic web until the appearing of GeoSPARQL.

However due to the innovative character of this standard, there are few systems which actually implement it. Anyway, the usage of this protocol in the system will benefit the system, for in the future, if the standard is adopted worldwide, the data will be more fitting to the Semantic Web.

### Parliament

Parliament™[60] is a triple-store, SPARQL endpoint and reasoner created by SemWebCentral in 2009. It is one of the few existing semantic databases which support the GeoSPARQL protocol, and it does so without then need of a relational spatial database on the back-end.

Parliament provides a SPARQL endpoint over which selection or update queries can be done. Besides it provides a reasoner which allows to make spatial queries. This database-reasoner-endpoint bundle makes this software very adequate to support the semantic GIS system to be produced. In addition, the reasoner that the bundle provides supports OWL lite, thus it is not necessary to use any additional software to reason over the data.

## 3.6. NODEJS

NodeJs is a framework based on Chrome's runtime engine [67] for developing high-performance, concurrent programs that instead of relying on the mainstream multi-threading approach use asynchronous I/O [65].

The reason behind the development of this framework is mainly that Thread-Based networking is inefficient and difficult to use. Node will be much more memory efficient under high load than systems which allocate threads for each connection.

Node differs from other frameworks such as Django or Rails in that it uses an event driven programming model. In Node, the event model is taken further than in the rest of frameworks, the event loop is presented as a language construct instead of a library. In other languages the event loop is typically started through a blocking call such as `EventMachine.run()` however, in Node, there is no event loop start call, it simply enters it after executing the input script and exits it when there are no more callbacks to execute.

One example of Node's programming model is shown in listing 3.11. In the example, a web server listening on port 1337 is created. To when creating this server a callback is passed; every time a connection is made the web server responds with a "Hello world". The process will tell the operating systems to notify it when a new connection is made, and then go to sleep. When new connections are made the callback will be executed, each connection is just a small memory allocation.

```

1  var http = require('http');
2  http.createServer(function (req, res) {
3    res.writeHead(200, {'Content-Type': 'text/plain'});
4    res.end('Hello World\n');
5  }).listen(1337, "127.0.0.1");
6  console.log('Server running at http://127.0.0.1:1337/');

```

Listing 3.11: NodeJS "hello world" web server.

The framework is powered by a module system. Modules make it possible to include other JavaScript files into an application, using the keyword `require` to load them. This functionality allows easy use of external libraries, in fact, most of the core functionality is written using modules.

NodeJS treats HTTP as a first class protocol, attempting to correct most of the problems that arise in other web frameworks such as HTTP streaming. Thanks to this approach, the platform gives a good foundation for creating web development libraries such as Express.

## NPM

NPM stands for Node Package Manager. It is a tool built to allow for easy installation and control of NodeJS modules. The software offers a simple command line interface to work with. In addition, it offers a way of managing node projects by using JSON configuration files.

By creating a file named `package.json`, it is possible to specify the modules used on the project and their version, plus several other configuration options such as a remote repository.

When using Node to build applications it is almost unavoidable to use NPM to install the needed libraries.

## Express

Express is a minimalist and flexible web application development library built on top of NodeJS. It offers a URL routes, template engine utilities and middle ware among other things.

Express is currently the de facto standard in Node web application development, mainly due to the ease of use it offers. The library offers an `express` object on which callback functions for different HTTP requests can be registered. These callbacks are provided with a request and response

### 3. TECHNOLOGICAL RESEARCH

object which can be used to analyse the request made to the server and to respond respectively. The example on listing 3.12 creates a express server on port 3000, which will respond to queries on the route "/hello".

```
1  var express = require('express');
2  var app = express();
3  app.get('/hello', function(req, res){
4    res.send('Hello World');
5  });
6  var server = app.listen(3000, function() {
7    console.log('Listening on port %d', server.address().port);
8  });
```

Listing 3.12: Express "hello world" program.

Aside from the basic functionality, the library offers a series of shortcuts for rendering parameterized HTML templates, and for building JSON based APIs.

## Why NodeJS

There are several frameworks which could have been used for this project. Jena, a Java framework for Semantic Web application is typically used to manipulate data, and other frameworks such as Django offer very complete geospatial API. Node offers none of these, however, the advantages on performance and scalability it offers out weight other tools. Besides, the student's previous experience with Node has also been taken into account when choosing Node as the working platform.

## 3.6. ANGULARJS

In the last years, there has been a trend to develop web applications instead of developing natively for each operating system. The rise in computational power allows for complex processes to be executed in web browsers and makes the software installation times meaningless, which is causing a shift from native development to web development.

Developing applications for the web has several advantages: there is no need to think in multi-platform development, it is granted; there is no need for installation, etc. Still, there are several issues that arise when developing on the web. The main issue is related to complexity on big JavaScript apps; when projects grow enough it becomes nigh impossible to manage them, due to the lack of structure inherent to most JavaScript projects.

Due to this, tools have been developed to ease web application development. One of the most prominent among these tools is AngularJS, a framework with focus on SPA (Single Page Application) development following the MVC (Model-View-Controller) pattern [23, 34].

Angular has been developed with several goals in mind: First, reducing the amount of DOM manipulation on the code, separating the client side of the application from the server side, giving structure to web applications and improving the capability to test the code.

## Model View Controller

The MVC pattern is implemented in this framework by separating the application into three main components [29].

The view: The HTML templates which can be found in any web page are the view, however, unlike in regular applications, they are extended using directives provided by the framework.

The controller: Angular allows the definition of controllers in a application. They are used to define operation which will manipulate the data and the views. The views and the controllers are related through an object called `$scope` (see figure 3.3).

The model: Any data which belongs to the application domain is considered part of the model. Usually the data is obtained from the server, through REST APIs, since function to easy asynchronous requests are provided.

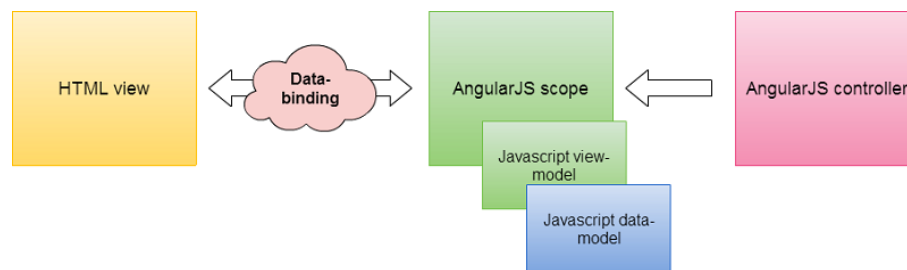


Figure 3.3: View-controller binding through the scope

source: <http://s.codeproject.com/>

In addition to the mentioned, the framework allows the definition of Services and Directives. Services are object that can be injected into a controller and have a wide array of uses, from code reusing to data sharing among controllers. Directives on the other side are utilities to manipulate the views and aren used in the HTML views. One example of such directives is `ng-click` which is used to specify which operation will be executed when a element on the DOM is clicked.

The MVC in Angular works using a two-way binding. Data is bound to the view through the `$scope` object; if a change happens in the view, the data in the models is automatically updated and if a controller modifies the data in the models, the view is automatically updated. An example is shown on listing 3.13. In this example. when the button is clicked, the function `change` will be executed, changing the value on the variable `text` which will automatically update the paragraph on the view.

## Modules

Similar to NodeJS, Angular provides it's own module system. Modules act as libraries, providing the application with services and directives. The framework does not provide with any tool for installation and management of modules, however, external package manages such as bower [10] have been built.

There are other tools that offer support for this framework. Yeoman is a widely used tool for scaffolding web applications. It is used to build skeletons for different types of projects such as Angular projects.

### 3. TECHNOLOGICAL RESEARCH

```
1  <div ng-controller="ExampleCtrl">
2    <p>{{text}}</p>
3    <button ng-click="change()">
4      Change text
5    </button>
6  </div>
7  <script type="text/javascript">
8    var app = angular.module('exampleApp');
9    app.controller('ExampleCtrl', function($scope){
10      $scope.text = 'No text yet';
11      $scope.change = function(){
12        $scope.text = 'Button has been clicked';
13      };
14    });
15  </script>
```

Listing 3.13: AngularJS example.

For the following modules have been considered:

ui-router: Since angular is used for developing single page applications, a way to navigate through the views without reloading the full page is needed. Ui-router provides a more complete than the default way of doing this, offering functionality such as nested view and parametrized views.

ui-bootstrap: Twitter Bootstrap is a CSS and JavaScript framework which aims at cutting the web application development time by reducing the amount of time dedicated to creating stylesheets [43]. This module provides angular directives and services for manipulating the elements provided by the framework.

restangular: Angular offers a way of querying REST API through the resources module, however, it falls short when the functionality offered is more complex than simple CRUD (Create, Read, Update, Delete) operations. Restangular is a library that replaces the default module with a more complete functionality.

flowjs: Uploading images and other kind of files is a harder task in SPAs than in regular web pages. This module helps in the tasks of uploading files with a series of angular directives, although it can be used in other applications.

leaflet-angular: Leaflet, detailed in section 3.7 is a library for creating interactive HTML5 based maps. This module is used to manipulate leaflet maps using a angular directives and services, which is more convenient than the usual way.

### Why AngularJS?

The project involves a good amount of client side JavaScript programming, so it is expected of the application to become quite difficult to manage. Due to this, there is no doubt that some kind of web application development framework is needed.

Some alternatives exist currently, such as Ember or Knockout, however Angular is the most mature of them. The huge community working with this library and the amount of exiting modules are



some of the reasons that decline the balance in favor of angular. Other factors include the focus on developing a Single Page Application, which are more mobile friendly than regular web pages; and the amount of learning tools available for the framework.

### 3.7. LEAFLET

Leaflet is a JavaScript library for building mobile friendly interactive maps. It is designed with simplicity, usability and performance in mind and it works efficiently among all major mobile and desktop browsers, taking advantage of HTML5 and CSS3 when it can [42].

The library has quickly become one of the most if not the most popular mapping library since its release in 2011, due to its small size, ease of use and extendability. The example in listing 3.14 shows the creation of a map.

```
1 var map = L.map('map');
2 var layer = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');
3 layer.addTo(map);
```

Listing 3.14: AngularJS example.

### Layers

Leaflet works with the concept of Layer. A layer is a visual representation of a geographic feature or set of features. There are three types of layers in the framework: tiles, vectors and layer groups.

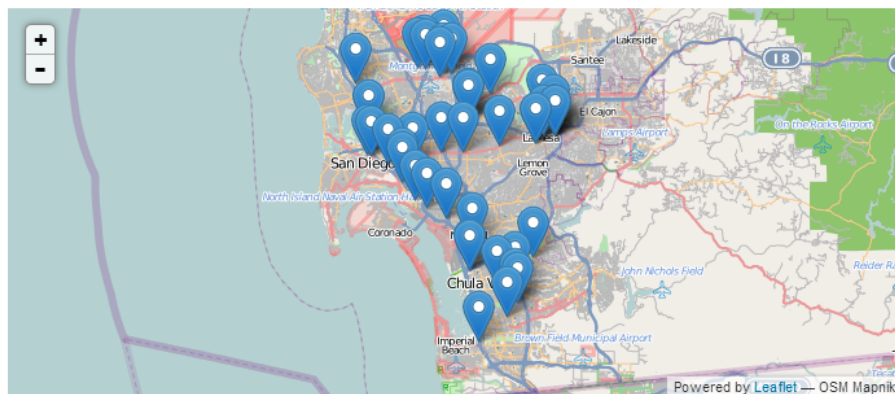


Figure 3.4: Leaflet with OSM mapnik tile layer

source: <http://drupal.org/>

**tiles:** Tiles are square bitmap graphics displayed on a grid fashion in order to show a map. Tile layers are defined by a URL from which the images will be obtained and the library will take care of displaying the appropriate grid depending on the zoom level and the view box. The tiles used change drastically the looks of the map, an example of the OSM mapnik tiles can be found in figure 3.4.

### 3. TECHNOLOGICAL RESEARCH

vectors: These layers are used to display information about features on the map in form of geometries. Vector layers can be classified into five categories, three of them corresponding to the typical vector representation on GIS systems: lines, polygons, points, rectangles and circles. Layers can be added and removed from the map easily as shown in example 3.15

layer groups: The last kind of layers are simply used as a convenience to manipulate more than one feature at the same time. One kind of layer group is the GeoJSON layer which allows the creation of layers from GeoJSON files, detailed in section 3.10.

```
1  var bounds = [[54.559322, -5.767822], [56.1210604, -3.021240]];
2  L.rectangle(bounds, {color: "#ff7800", weight: 1}).addTo(map);
3  var point = L.marker([50.5, 30.5]);
4  //A popup can be bound to be shown on click
5  point.bindPopup("This is a popup message");
```

Listing 3.15: Leaflet layers.

## Interaction

Leaflet is built with the idea of creating interactive maps, thus, it offers methods for manipulating the map and firing and receiving events. It is possible to register listener on map actions, such as when a map is moved or zoomed, or to register listeners on layers, to check when a layer is added, removed, clicked, etc.

This way, it is possible to respond to user actions via callbacks. The framework provides some shortcuts, for example, it is possible to bind the showing of a small popup to the click on a marker without the need to register a callback, as shown in listing 3.15

Another example on the usage of leaflet events can be found in listing 3.16. In the example, a callback is registered for the event of the user moving the map. This is useful for updating the map with information from the layers on the current view.

```
1  // The moveend event is fired every time the
2  // map is moved, dragged or zoomed
3  Map.map.on('moveend', function(){
4      //Obtain the current bounding box of the map
5      var bbox = Map.map.getBounds().toBBoxString();
6      var feature = queryTheServer(bbox);
7      map.addLayer(features);
8  });
```

Listing 3.16: Leaflet layers.

## Why Leaflet?

There are not many alternatives when it comes to create JavaScript based maps. Google maps can be used to embed a map on a web page, however, it does not offer much interaction and can hardly be used for other than displaying static data. OpenLayers is another alternative, however, it is known to be hard to learn and the community using it is smaller than Leaflet's one. The amount

of developers using leaflet provides the framework with a good deal of extensions and its ease of use make it easy to work with even for novice developers.

### 3.8. PHONEGAP

Desktop computers are not the only devices that have seen a rise in computational power in the last years, smartphones and other mobile devices have also improved significantly.

Interaction, screens and even browser support are drastically different in desktop and mobile; and the latter still suffers from a lack of computational power. Due to this web development is not so easily applicable to these devices, building a mobile friendly web application is a difficult task.

However, due to the shift from native to web development happening on desktops, most developers have already learned the tools for developing HTML5 applications. This together with the popularity of mobile apps and difficulty of developing multi-platform applications, has given rise to a number of tools designed to code native mobile applications using HTML5, CSS3 and JavaScript.

The leading among these is Phonegap. Phonegap is a framework that allows to create mobile apps using standardized web APIs for all the major platforms [31].

#### HTML5 development

The development tools used on a Phonegap application are identical to those used on a regular web page. HTML documents are used to structure the views, CSS stylesheets are used to define the appearance of the application and JavaScript is used to program the behavior.

This is possible because what the library does is opening a enhanced web view (a browser) on the mobile device and loading the files into it. Normally, this would mean that much of the sensors of the device, such as the camera and the gyroscope are not usable, however, the browser created by the framework is extended to allow access to all the hardware on the device.

Phonegap is based on Apache Cordova [20] which works with a set of plugins. These plugins provide the web view with the functionality it lacks, and expose it as JavaScript APIs. One example of this is the reading of the battery status, as shown on listing 3.17. Aside from the plugins that come shipped with the core libraries, the community has developed a considerable amount of them to provide functionalities that mobile browsers usually lack, such as HTML5 web socket support.

```

1  window.addEventListener("batterystatus", onBatteryStatus, false);
2
3  function onBatteryStatus(info) {
4      console.log("Level: " + info.level + " isPlugged: " + info.isPlugged);
5  }
```

Listing 3.17: Reading battery status with Phonegap.

#### Phonegap Build

As convenient using HTML5 for developing cross platform applications is, there is still a need to configure the build for each of the platforms and to install the corresponding Software Development

### 3. TECHNOLOGICAL RESEARCH

Kits (SDKs) which can be tedious and time consuming.

In the face of this issue, Phonegap offers a platform called Phonegap Build, a cloud based service which helps agile development and allows to compile applications for six of the seven platforms supported by the framework. The Phonegap team illustrated this process in figure 3.5 on their web page.

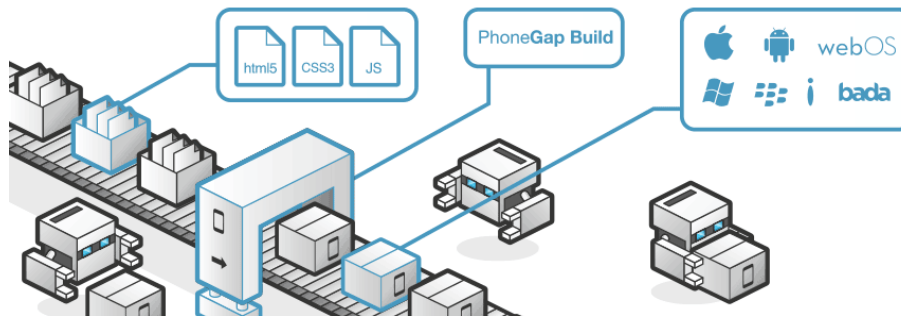


Figure 3.5: Phonegap Build process

source: <http://phonegap.com/about/>

### Why Phonegap?

In the world of mobile HTML5 development there are not many competitors. Phonegap is the leading platform, followed by Appcelerator and Ionic. Still, Phonegap is the most mature of them all and has the biggest community, which results in a higher amount of plugins offering more functionality.

On the other side, native development is out of consideration. One of the requirements of the project is for the mobile application to be multi-platform and while developing it natively may increase performance, the payback in development times would be too much. Besides, since the mobile application interfaces and interaction are heavily map based, the code used on other parts of the project can be reused.

Thus, the decision to use Phonegap is made, based on the cut on development times, the re usability of code and the fact that there is no need to learn other development languages.

### 3.9. GPX

GPX is a lightweight XML data format for the interchange of GPS data (tracks, routes and way points) between applications and web services on the Internet [28]. GPX was released in 2004 and has since then been the de facto standard when it comes to the encoding and interchange of GPS data.

The format represents three types of data, in the following manner:

Way points: A way point is a set of coordinates that identifies a point in space. They are represented using the `<wpt>` tag, as shown in listing 3.18

```

1 <wpt lat="42.438878" lon="-71.119277">
2   <ele>44.586548</ele>
3   <time>2001-11-28T21:05:28Z</time>
4   <name>Example waypoint</name>
5 </wpt>

```

Listing 3.18: GPX way point representation.

Tracks: A track is a segment containing way points, that is, a set of coordinates that describe a path. In GPX tracks are represented by the `trk` element and point inside one by `trkpt` elements (see listing 3.19).

```

1 <trk>
2   <name>Example GPX Document</name>
3   <trkseg>
4     <trkpt lat="47.644548" lon="-122.326897">
5       <ele>4.46</ele>
6       <time>2009-10-17T18:37:26Z</time>
7     </trkpt>
8   </trkseg>
9 </trk>

```

Listing 3.19: GPX track representation.

Routes: A route is identical to a track in its encoding, with the difference that the tags `rte` and `rtept` are used (see listing 3.20). However, there is a difference in the meaning of this concepts; while a track is a record of where a person has been, a route is a suggestion about where someone might go in the future. For this reasons, tracks may have timestamps attached while routes will not.

```

1 <rte>
2   <name> Example route </name>
3   <rtept lat="47.644548" lon="-122.326897">
4     <name>Example route point</name>
5   </rtept>
6 </rte>

```

Listing 3.20: GPX route representation.

## Why GPX?

GPX has been the de facto standard when it comes to representing and interchanging trail and points of interest on the web for ten years. Because of this, it is unreasonable to think in a system which allows interchange of GPS data without using this format. Together with the completeness of the information provided by a GPX file, these are two solid reasons for the usage of GPX on the platform.

## 3.10. GEOJSON

GeoJSON is a format for encoding a variety of geographic data structures [14]. GeoJSON objects may represent a geometry, a feature or a collection of features (see section 3.5, GeoSPARQL). The following geometry types are supported:

- `Point`
- `LineString`
- `Polygon`
- `MultiPoint`
- `MultiLineString`
- `MultiPolygon`
- `GeometryCollection`.

More than a file format in itself, it is a specification of how to encode geographical features in JSON format. A complete GeoJSON data structure is always a JSON object, which consists in a collection of key-value pairs, being the key a string and the values any other kind of data.

The specification defines GeoJSON objects in the following way:

- A GeoJSON object may have any number of members (name/value pairs).
- The GeoJSON object may have a member with the name `type`. Its value will be a string that determines the type of the object.
- The value of the type member must be one of: `Point`, `MultiPoint`, `Polygon`, `MultiPolygon`, `LineString`, `MultiLineString`, `GeometryCollection`, `Feature` Or `FeatureCollection`.
- A GeoJSON object may have an optional member of name `crs` which specifies the coordinate reference system.
- A GeoJSON object may have a `bbox` member, whose value must be a bounding box array.

### Geometries

Geometries are GeoJSON objects whose type is different from `Feature` Or `FeatureCollection`. These elements, unless they are of type `GeometryCollection`, must have a member of name `coordinates` which specifies the coordinates of the object in an array of positions.

Positions are represented by an array of numbers, one for the longitude, one for the latitude and an optional one for the elevation, in that order. The coordinates of a object are formed by a array of positions, except for `Point` element, which have a single position.

A `GeometryCollection` must have a member named `geometries`, an array of GeoJSON geometry objects. An example of this can be found on listing 3.21.

### Features

A `Feature` is a object with some spatial representation. A feature must have two essential members, aside from the type, which are `geometry` and `properties`. The first of these members must contain a GeoJSON geometry object, the spatial representation of the object.

```

1  { "type": "GeometryCollection",
2    "geometries": [
3      { "type": "Point",
4        "coordinates": [100.0, 0.0]
5      },
6      { "type": "LineString",
7        "coordinates": [ [101.0, 0.0], [102.0, 1.0] ]
8      }
9    ]
10  }

```

Listing 3.21: A GeoJSON GeometryCollection object.

The properties member on the other side, is a regular JSON object (a name/value dictionary) and contains all the non-spatial data belonging to the feature. A FeatureCollection does not contain a geometry member, instead it has a features object, an array of feature object in the collection. An example of a feature is shown on listing 3.22

```

1  { "type": "Feature",
2    "bbox": [-180.0, -90.0, 180.0, 90.0],
3    "geometry": {
4      "type": "Polygon",
5      "coordinates": [[
6        [-180.0, 10.0], [20.0, 90.0], [180.0, -5.0], [-30.0, -90.0]
7      ]]
8    },
9    "properties": {
10     "name": "Example feature"
11   }
12 }

```

Listing 3.22: A GeoJSON Feature object.





## 4. REQUIREMENTS SPECIFICATION

---

### 4.1. OVERVIEW

This chapter provides a specifications of the requirements, both functional and non-functional, that the project to be developed must satisfy, which define the overall functioning of the system to be produced. Each requirement will have a code associated for identification purposes.

This requirements specify the minimum possible constraints or characteristics that the system must fulfill. Most technology is kept outside of the requirement specification so that it does not affect the implementation or the design. To achieve a better understanding of these requirements, they are divided into the following sections, each corresponding to a different phase of product of the project.

- Requirements of the ontology: The requirements to be satisfied by the ontology as well as a minimum subset of classes and properties it must have are listed.
- Requirements of the server: The requirements to be satisfied by the server and the system overall are defined in this section.
- Requirements of the web application: The requirements to be satisfied by the web client are defined in this section.
- Requirements of the mobile application: The requirements to be met by the mobile client are defined in this section.
- Non-functional requirements: The requirements which don't specify any particular .

### 4.2. REQUIREMENTS FOR THE ONTOLOGY

The requirement for the ontology are the ones which define the characteristics that the vocabulary to be produced must meet. This requirements are listed below:

RONT1 The ontology must support spatial reasoning.

RONT2 The ontology must support the following inference mechanisms:

- disjoint classes
- subclasses
- subproperties
- inverse properties
- symmetric properties
- functional properties
- maximum cardinalities

#### 4. REQUIREMENTS SPECIFICATION

RONT3 The ontology must be designed following Linked Open Data best practices, so it should reuse existing vocabularies.

RONT4 The ontology must represent the following types of resources:

- Features
  - Trails
  - Points of Interest
  - Geolocated Notes
- Persons
- Multimedia Resources
  - Images
  - Video
- User reviews

RONT5 The resources of type Trail must contain the information below:

- name
- description
- difficulty score
- maximum altitude
- minimum altitude
- ascending slope
- descending slope
- posts
- images
- author
- persons who traversed it
- spatial representation

RONT6 The resources of type Point of Interest must contain the following information:

- name
- description
- altitude
- category
- posts
- images
- author
- spatial representation

RONT7 The resources of type Geolocated Note must contain the following information:

- text
- multimedia
- author
- privacy level
- creation time
- duration
- action radius

- spatial representation

RONT8 Resources of type Person must contain the following information:

- nickname
- email
- first name
- family name
- avatar
- description
- trail buddies
- optionally external homepage
- traversed trails
- added trails
- added points of interest
- added notes
- added multimedia

RONT9 Multimedia resources of type Image and Video must contain the following information.

- author
- addition date
- download link
- feature they belong to

RONT10 Resources of type Review must contain the following information.

- author
- addition date
- textual content
- rating
- related feature

### **4.3. REQUIREMENTS SPECIFICATION FOR THE SERVER**

The requirements for the server specify the functionality that the server must implement as well as the function and resources exposed by the API and overall inter-component communication, for the server is the central piece of the system.

RSV1 The system must be able to operate on Points.

RSV2 The system must be able to operate on LineStrings.

RSV3 The server must be able to obtain the following data from a LineString representing a trail.

- distance
- difficulty
- ascending slope
- descending slope

#### 4. REQUIREMENTS SPECIFICATION

- maximum altitude
- minimum altitude

RSV4 The system must be able to create and send SPARQL queries to a data store.

RSV5 The system must be able to retrieve, transform and aggregate it to the dataset, spatial data from the following sources:

- OpenStreetMap
- Geonames

RSV6 The server must expose a SPARQL read-only endpoint.

RSV7 The server must expose a public API.

RSV8 The public API must follow the REST style.

RSV9 The API must expose the following resources

- Trails
- Points of Interest
- Geolocated Notes
- Users

RSV10 The resources must expose the following operations:

- Read
- Update (Except Geolocated Notes)
- Create

RSV11 The resources must expose information about other related resources, such as posts, images, etc.

RSV12 The API must offer additional operations which expose the following functionality:

- Search
- Features within a area
- Features near each other
- Information about the system
- Information about the API

RSV13 The server must offer secure authentication.

RSV14 The system must be able to provide recommendations based on preferences and location.

RSV15 The system must be able to store user preferences, for recommendation and filtering purposes.

RSV16 Signing in the system will be done via e-mail and password.

## 4.4. REQUIREMENTS OF THE WEB APPLICATION

The requirements for the web application detail the constraints that the web based client must fulfill, as well as the functionality that need to be implemented.

RWEB1 The web application must work on most major browsers. The minimum browsers specified are the following:

- Google Chrome 26
- Mozilla Firefox 22
- Safari 6
- Internet Explorer 10
- Opera 12

RWEB2 The web application must communicate with the server using the public API.

RWEB3 The web application will have a homepage presenting the platform and linking to the rest of the sections.

RWEB4 The web application must have a section allowing the following features:

- Explore data based on its location
- Draw and edit trails and points of interest

RWEB5 The web application must have a section which provides the following functionality.

- Search over the data on the system
- Upload a trail or point of interest from a GPX file
- View detailed information about any data on the platform

RWEB6 The web application must have a section which provides the following features:

- Register a user on the system
- Log in the system
- Modify the profile of a user

RWEB7 All geographic data must be shown on a interactive map.

RWEB8 All operations that require manipulation of spatial data have to be performed without any need of GIS knowledge from the user.

RWEB9 Signing in only has to be needed for write operations.

RWEB10 Registering a user via web platform will require the following information:

- User name
- Email
- Password

RWEB11 Creating a trail via web platform will require the following information:

- Name
- Description
- Geographic representation

#### *4. REQUIREMENTS SPECIFICATION*

RWEB12 Creating a point of interest via web platform will require the following information:

- Name
- Description
- Category
- Geographic representation

### **4.5. REQUIREMENTS OF THE MOBILE APPLICATION**

The requirements to the mobile specify the operations that can be done with the mobile application and the functionality it must implement.

RMOB1 The application must allow the recording of a trail.

RMOB2 The application must allow exporting trails in GPX format or uploading them to the system when a network connection is available.

RMOB3 The application must allow the creation of Geolocated Notes on the current position of the user.

RMOB4 The application must keep track of the current position of the user at all times.

RMOB5 The application must be able to receive real time notifications of nearby features.

RMOB6 The application must be able to notify the user when a new feature is discovered, even if it is not currently active.

RMOB7 The application must allow search and detailed view of information on the system.

RMOB8 The application must provide log in and registration functionalities.

RMOB9 The application can only be used by registered users.

RMOB10 Registering a user via mobile platform will require the following information:

- User name
- Email
- Password

RMOB11 Creating a trail via mobile platform will require the following information:

- Name
- Description
- Recording of the trail

RMOB11 Creating a geolocated note via web mobile platform will require the following information:

- Text, Image or video
- Privacy level
- Range
- Coordinates

RMOB12 Once signed in the mobile application, the system will store the password and username, for convenience purposes.

## 4.6. NON-FUNCTIONAL REQUIREMENTS

The requirements not belonging to a specific category are listed in this sections. This refers to information visibility and privacy, performance and user interaction issues mainly.

RNF1 All the information, except a few sensible data, has to be published according to Linked Open Data best practices.

RNF2 The following data must be kept private:

- User passwords
- The current location of the users

RNF3 The server should be able to handle high loads of connection.

RNF4 The interface on the web application must be responsive, that is, the interface must adapt to mobile and tablet devices.

RNF5 The web application must have a fast loading time and small memory footprint, in order to be adapted to mobile devices.

RNF6 The applications must have intuitive and easy to use and read interfaces.

RNF7 The style of the mobile and web applications must be similar.





## 5. DESIGN SPECIFICATION

---

### 5.1. OVERVIEW

This section describes the design work done through the project as well as the tools used to carry that labor and the results of it. The chapter covers the following topics:

- Architecture of the system: The chosen architecture of the system is described, together with the reasons and advantages of the design.
- Design of the ontology: The final design of the ontology is described, as well as its advantages in front of other data modeling paradigms.
- Design of the central server: The class design of the server is detailed in this section.
- Design of the web application: The class design of the web application is detailed in this section, as well as the interface design.
- Design of the mobile application: The class design of the mobile application is detailed in this section, as well as the interface design.
- Development environment: The list of technologies employed in the project is detailed, briefly explaining the role of each tool.

### 5.2. SYSTEM ARCHITECTURE

Figure 5.1 shows the architecture chosen for the system. This architecture comprises the following elements:

- A central server
  - Web server
  - SPARQL endpoint
  - REST API
  - Data aggregator
  - Database connector
- A semantic spatial storage system which contains the ontology
- A web client used as the main interface of the system
- A mobile application to support the browser based one

#### 5.2.1 Central server

The central server is the core component of the system. It implements most of the functionality on the system and provides data to the client. It also analyzes the trails uploaded to the system

## 5. DESIGN SPECIFICATION

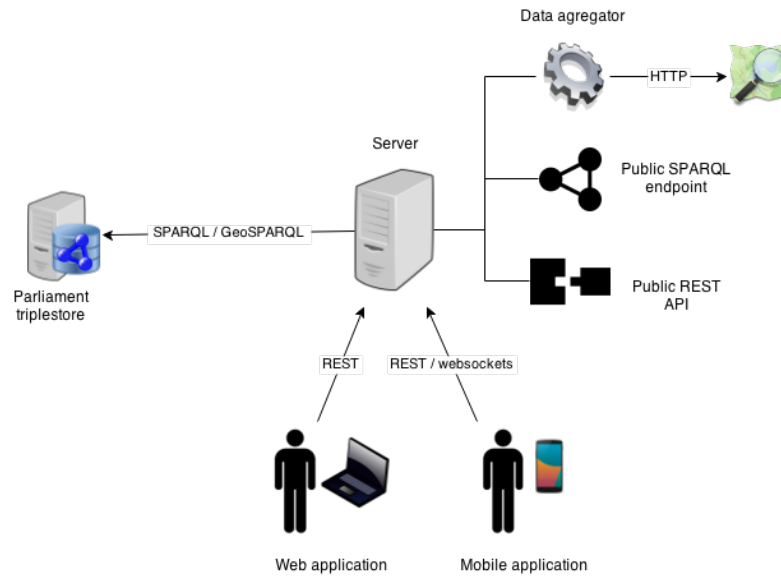


Figure 5.1: Architecture design of the system

and takes care of the communication with the database. The server is divided into several logical software pieces, which are described below.

### Database connector

The database connector implements the functionality needed to connect with the semantic data store. In essence, this means that it takes the requests from the users and transforms them into SPARQL queries. Then it takes the results provided by the database and converts them into a format that the server can process.

### The data agregator

This component has the function of querying external data sources and adapting the results to the data model of the system. It is independent of the rest of components of the server, since instead of being available to receive requests it will just run on demand. The component makes HTTP queries to the different APIs and endpoints on the web, for example OpenStreetMap and Geonames, and aggregates the data to the database.

### REST API

The API provides the basic means of accessing the information on the system. It is used by both clients to communicate with the server. It can be used for read, write and update operations, however, delete operations are not contemplated.

### **The SPARQL endpoint**

This endpoint just routes SPARQL queries done to it to the database. Its role is related to filtering the queries that seek to write and to ensure that no petition is going to break the integrity of the data in the database.

### **5.2.2 Semantic spatial storage system**

This component of the system is mostly third party software. It consists on a triple store, a data storage system that stores RDF data; a reasoner supporting OWL and GeoSPARQL and an HTTP SPARQL endpoint. This component also englobes the ontology designed to define the data model of the system. This ontology is described in detail in section 5.3.

### **5.2.3 Web client**

The web client is the main interface through which the system is accessed. It communicates with the server through a REST API and offers most of the functionality of the system to the users.

The web application provides the usual functionality on web platforms, such as registering, browsing data, uploading data, editing the users profile and modifying data. Most of the operations have to be directly communicated to the server, however, there are certain functions such as the edition of trails that can be done offline and later uploaded to the server.

The client may be used from desktop browsers or mobile browser with no restrictions. More details on the design of the client can be found on section 5.5.

### **5.2.4 Mobile application**

The mobile client provides the functionality that the web application cannot. It offers real time functions by using a websocket API that the server exposes. It provides many of the browsing functionality of the web application on a interface adapted to be as mobile friendly as possible.

In addition, the mobile client exposes new functions typical of similar applications. These functions include the recording of a route and exporting it to a interchangeable format, that is GPX. In addition, the mobile application allows the creation of geolocated notes, which is not possible through the web application.

The design of this component is detailed in section 5.6.

### **5.2.5 Functioning of the system**

This section aims to provide an overview of how the system works and how the different components communicate among them.

The web and the mobile application will access the server through the API it exposes in every moment. The client will restrict the requests that can be sent to the server if the user is not logged

## 5. DESIGN SPECIFICATION

in the system, however, for security reasons, the server will always check if the user is logged in the system if the operation modifies data.

All the requests result in a query to the semantic database. For this, the server creates a specific SPARQL query depending on the request done and sends it to the local URL where the endpoint is listening through an HTTP request. Then, the server parses the response, transforms it to a regular JSON object and replies it to the client.

The normal flow of information on the system is as follows:

1. The client sends a request to the server
2. The server performs some operations and sends a request to the data store
3. The data store answers and the server parses the request into a JSON file
4. The data is returned to the client
5. The client uses the data

However, there is an exception to this flow when exercising real time communication in the mobile application. In this case, there is not always a response to the client. Since this communication is done through a persistent connection, the flow of information is the following:

1. The client establishes a connection
2. The client sends its coordinates to the server
3. The server queries the database for nearby features
4. The server checks if there is any new information compared to the previous requests
5. If there is new information, the server replies only with the new data, if there is not there is not response

In any case, the client will send periodic responses without worrying if the server actually responds or not.

### 5.3. DESIGN OF THE ONTOLOGY

The ontology designed in the project represents specific knowledge related to the domain of Trails, Points of Interest and GPS data in general. It uses external ontologies as a base, in order to ease the sharing of information and the publishing of the dataset at Linked Open Data.

Figure 5.2 provides a graphical representation of classes on the SORELCOM ontology. As shown on the figure, classes from four vocabularies are reused. The FOAF vocabulary is used to obtain a representation of the users of the platform, the GeoSPARQL vocabulary is used as a base for all the resources with a spatial character, the W3C media vocabulary is used to represent the media resources on the platform and the Review vocabulary, representing the reviews and comments done by the users.

The focus of the design is to reuse as many well known vocabularies as possible and defining the needed classes, relations and properties to satisfy the requirements of the ontology 4 to 10, defined in chapter 4.

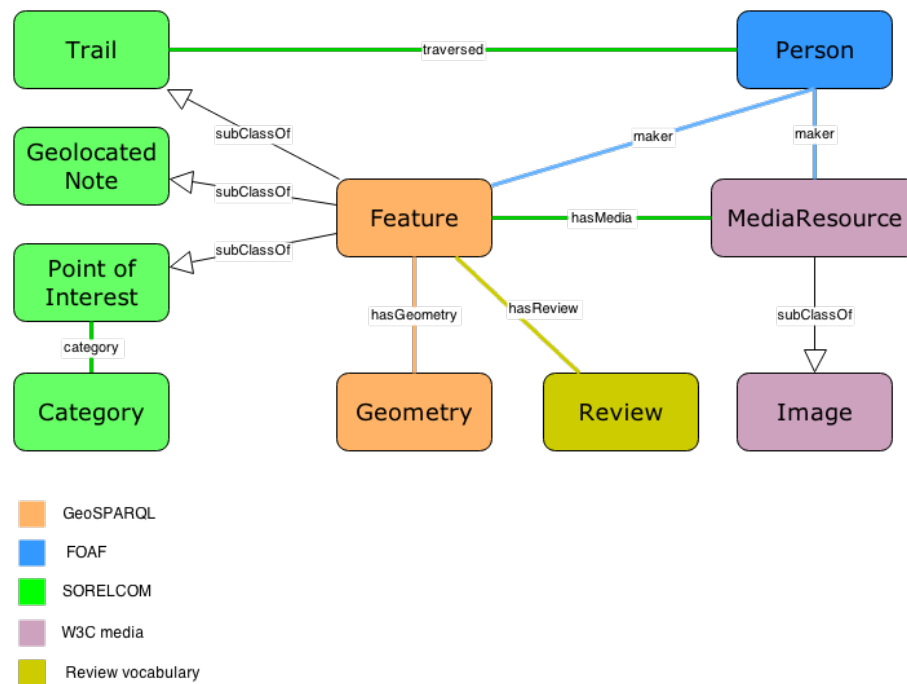


Figure 5.2: The SORELCOM ontology

### 5.3.1 Class hierarchy

The ontology defines a series of classes for the representation of specific spatial features present in the domain of the application. The classes are listed below:

**sorelcom:Trail** A trail represents a path in the world. In the context of the application, a trail can be interpreted as a path that a user has traversed or as a simple route that is indicated by a user, however, the ontology does not specify that a trail should have any of these meanings.

**sorelcom:PointOfInterest** A Point of Interest is a feature in the world that may result of some interest for a person. There is no specification on what can be the interest a person may have on the feature, thus a point of interest may be anything from a monument to a restaurant.

**sorelcom:GeolocatedNote** A Geolocated Note is a message with a spatial context. This message is left by a person at a specific location, and can be viewed by any person or just a specific group depending on its privacy settings. The message on a note may be formed by text, images, video or any combination of them.

**sorelcom:Category** A category used to classify the types of point of interest. Categories contain a human readable name, a depiction and a description.

In addition to the core types on the vocabulary, other classes obtained from external ontologies will be used or extended to represent different types of resources on the ontology. The additional classes used are the following:

**foaf:Person** This class is used to classify resources as persons. Any user on the system will be classified as person. The class is defined in the FOAF vocabulary.

## 5. DESIGN SPECIFICATION

**geo:Feature** A feature is any object in the real world that has a spatial representation. Trails, Points of Interest and Geolocated Notes are all types of Features. This class is defined in the GeoSPARQL vocabulary.

**geo:Geometry** In the GeoSPARQL ontology, a geometry is used to give a spatial representation to a Feature. Without a associated feature, geometries are not resources on the real world.

**media:MediaResource** A media resource is a representation of a media object, such as a image or a video. Media resources refer to the images, posts and videos saved in the system.

**media:Image** A image is a specialized media resource, which refers to a image.

**rev:Review** A review of a resource. Used for the representation of user posts on the features of the system. It may include comments, ratings or both.

### 5.3.2 Properties

Most of the work on the design of the ontology consists on the creation of properties which relate the resources to the information that they must contain. Still, many of the properties used in the data model of the platform are obtained from external vocabularies.

Namespace notation is used to represent external ontologies. The relation of vocabularies, their namespaces and the URIs they represent is shown in table 5.1.

The properties and classes are represented by a IRI. This IRI is just the namespace appended with the class or property name. For example, foaf:name and <http://xmlns.com/foaf/0.1/name> are equivalent.

Table 5.1: Points of interest on the SORELCOM data model.

Ontology	namespace	URI
SORELCOM	sorelcom	<a href="http://morelab.deusto.es/ontologies/sorelcom">http://morelab.deusto.es/ontologies/sorelcom</a>
FOAF	foaf	<a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a>
GeoSPARQL	geo	<a href="http://www.opengis.net/ont/geosparql#">http://www.opengis.net/ont/geosparql#</a>
W3C media	ma	<a href="http://www.w3.org/ns/ma-ont">http://www.w3.org/ns/ma-ont</a>
Review	rev	<a href="http://purl.org/stuff/rev">http://purl.org/stuff/rev</a>
Dublin Core	dcterms	<a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a>
RDF	rdf	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>
RDF schema	rdfs	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>
OWL	owl	<a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a>

The properties defined for the SORELCOM vocabulary are listed below, together with a table representing how the information about those resources is stored in the data model:

#### Feature properties

**sorelcom:name** Property representing the human readable name of a feature.

**sorelcom:description** Property representing a textual description of a feature.

`sorelcom:hasMedia` Property relating a feature to the media associated. It is the inverse of `sorelcom:mediaOf`.

### Trail properties

`sorelcom:difficulty` Property representing the difficulty score of a trail. This score is a integer from 0 to 100, the higher the difficulty the higher the number. A trail may only have one difficulty.

`sorelcom:maximumAltitude` Property representing the altitude of the highest point of a trail.

`sorelcom:minimumAltitude` Property representing the altitude of the lowest point of a trail.

`sorelcom:totalDistance` Property representing the total distance in meters that must be traversed from the starting point of a trail to the end of it.

`sorelcom:ascendingDistance` Property representing the sum of the distance in meters of the segments of the trail which are ascendant .

`sorelcom:descendingDistance` Property representing the sum of the distance in meters of the segments of the trail which are descendant.

`sorelcom:circular` Property representing if the starting point and end point of a trail are the same.

`sorelcom:traversedBy` Property relating a trail to the users who have traversed it.

Information about how trails are represented on the system can be found on table 5.2.

Table 5.2: Trails on the SORELCOM data model.

Property	domain	range	information
<code>sorelcom:name</code>	<code>geo:Feature</code>	string	name
<code>sorelcom:description</code>	<code>geo:Feature</code>	string	description
<code>sorelcom:difficulty</code>	<code>sorelcom:Trail</code>	integer	difficulty
<code>sorelcom:maximumAltitude</code>	<code>sorelcom:Trail</code>	float	maximum altitude
<code>sorelcom:minimumAltitude</code>	<code>sorelcom:Trail</code>	float	minimum altitude
<code>sorelcom:ascendingDistance</code>	<code>sorelcom:Trail</code>	integer	ascending distance
<code>sorelcom:descendingDistance</code>	<code>sorelcom:Trail</code>	integer	descending distance
<code>sorelcom:hasMedia</code>	<code>geo:Feature</code>	<code>ma:Media</code>	images and videos
<code>foaf:maker</code>	Thing	<code>foaf:Person</code>	author
<code>sorelcom:traversedBy</code>	<code>sorelcom:Trail</code>	<code>foaf:Person</code>	persons who traversed it
<code>geo:hasGeometry</code>	<code>geo:Feature</code>	<code>geo:Geometry</code>	spatial representation
<code>rev:hasReview</code>	<code>rdfs:Resource</code>	<code>rev:Review</code>	posts

### Point of interest properties

`sorelcom:altitude` Property representing altitude of the point.

`sorelcom:category` Property representing the category of the point of interest.

## 5. DESIGN SPECIFICATION

Information about how points of interest are represented on the system can be found on table 5.3.

Table 5.3: Points of interest on the SORELCOM data model.

Property	domain	range	information
sorelcom:name	geo:Feature	string	name
sorelcom:description	geo:Feature	string	description
sorelcom:altitude	sorelcom:PointOfInterest	float	altitude
sorelcom:category	sorelcom:PointOfInterest	sorelcom:Category	category
sorelcom:hasMedia	geo:Feature	ma:Media	images
foaf:maker	Thing	foaf:Person	author
geo:hasGeometry	geo:Feature	geo:Geometry	spatial data
rev:hasReview	rdfs:Resource	rev:Review	posts

### Geolocated Note properties

`sorelcom:range` Property representing range of action of a Geolocated Note.

`sorelcom:public` Property representing if the note is public.

`sorelcom:targets` Property relating the note to the persons that should receive it. It exists only when the note is not public.

Information about how geolocated notes are represented on the system can be found on table 5.4.

Table 5.4: Geolocated notes on the SORELCOM data model.

Property	domain	range	information
dcterms:created	Thing	date	creation time
dcterms:valid	Thing	date or integer	duration
sorelcom:description	geo:Feature	string	textual content
sorelcom:public	sorelcom:GeolocatedNote	boolean	privacy level
sorelcom:radius	sorelcom:GeolocatedNote	integer	action radius
sorelcom:hasMedia	geo:Feature	ma:Media	text, multimedia
foaf:maker	Thing	foaf:Person	author
geo:hasGeometry	geo:Feature	geo:Geometry	spatial representation

### Person properties

`sorelcom:hasTraversed` Property relating a Person to the trails it has traversed.

`sorelcom:trailBuddyOf` Property relating a Person to his/her trail buddies.

Information about how persons are represented on the system can be found on table 5.5.



Table 5.5: Persons on the SORELCOM data model.

Property	domain	range	information
foaf:nick	foaf:Person	string	nickname
foaf:mbox	foaf:Person	string	email
foaf:firstName	foaf:Person	string	first name
foaf:familyName	foaf:Person	string	family name
foaf:depiction	foaf:Person	foaf:Image	avatar
sorelcom:trailBuddyOf	foaf:Person	foaf:Person	trail buddies
sorelcom:hasTraversed	foaf:Person	sorelcom:Trail	traversed trails
foaf:weblog	foaf:Person	URI	external homepage
foaf:made	foaf:Person	Thing	features and media

### Media properties

`sorelcom:mediaOf` Property relating a media resource to the feature it belongs to.

Information about how media resources are represented on the system can be found on table 5.6. Media resources are divided into images and video, however, for convenience purposes both of them are presented in a single table.

Table 5.6: Media on the SORELCOM data model.

Property	domain	range	information
foaf:maker	Thing	foaf:Person	author
ma:description	ma:MediaResource	string	content
ma:locator	ma:MediaResource	URI	download link
dcterms:created	Thing	datetime	addition date
sorelcom:mediaOf	ma:MediaResource	geo:Feature	feature

### Review properties

Information about the representation of reviews can be found on table 5.7. Reviews are formed by a rating of the feature and a possible textual comment on it. All properties used for review representation and relation are external.

Table 5.7: Posts on the SORELCOM data model.

Property	domain	range	information
rev:reviewer	rev:Review	foaf:Person	author
rev:text	rev:Review	string	textual content
rev:rating	rev:Review	integer	user evaluation
sorelcom:mediaOf	ma:MediaResource	geo:Feature	feature
dcterms:created	Thing	datetime	creation date

### Category properties

Categories are not specified by any requirement, however a representation for them is created in order to provide information about these categories and what they represent to the user. Information about these resources can be found in table 5.8.

Table 5.8: Categories on the SORELCOM data model.

Property	domain	range	information
rdfs:label	rdfs:Resource	string	name
rdfs:comment	rdfs:Resource	string	description
foaf:depiction	Thing	foaf:Image	icon representation

### Inference mechanism

In order to get the maximum benefit from the semantic dataset the ontology has been defined using OWL. Due to this, several inference mechanism have been available during the design process, however, only a subset of them have been used. The following inference mechanisms have been used:

**Sub classes** All resources which are instanced of a certain class A are also instances of the classes A is subclass of. This property is provided by RDF schema (see section 3.2 for more information). In RDF it is possible to use multiple inheritance, meaning that a class can be subclass of more than one class.

**Disjoint classes** A group of disjoint classes indicate that a resource of one class of the group cannot be of another class on the group

**Sub properties** Subproperty relations are the analogous of subclass relations when referring to resources of type property. It is also provided by RDF schema.

**Function properties** A functional property is a property that can only appear once in the triples of a certain resource. It is provided by OWL lite.

**Inverse properties** When a resource A is related to B by a property P1, then B is related to A by the property P2 inverse of P1. It is provided by OWL lite.

**Symmetric property** When a resource A is related to B by a symmetric property P, then B is related to A also by P.

Table 5.9 shows the inference rules used on the classes of the ontology and 5.10 shows the inference rules used on the properties of the data model.

### 5.3.3 Advantages of ontology based design

The data model on the platform has been designed based on a ontology, mainly to follow the best practices of Linked Open Data. However, there are other alternatives, for instance, using

Table 5.9: Inferences used on the SORELCOM ontology classes.

Class	inference type	Related class
sorelcom:Trail	subclass of	geo:Feature
sorelcom:PointOfInterest	subclass of	geo:Feature
sorelcom:GeolocatedNote	subclass of	geo:Feature
sorelcom:Text	subclass of	ma:MediaResource
sorelcom:Trail	disjoint with	sorelcom:PointOfInterest
sorelcom:Trail	disjoint with	sorelcom:GeolocatedNote
sorelcom:GeolocatedNote	disjoint with	sorelcom:PointOfInterest

Table 5.10: Inferences used on the SORELCOM ontology classes.

Property	inference type	Related property
sorelcom:trailBuddyOf	sub property of	foaf:knows
sorelcom:trailBuddyOf	symmetric property	
sorelcom:hasTraversed	inverse property of	sorelcom:traversedBy
sorelcom:hasMedia	inverse property of	sorelcom:mediaOf
sorelcom:difficulty	functional property	
sorelcom:maximumAltitude	functional property	
sorelcom:minimumAltitude	functional property	
sorelcom:totalDistance	functional property	

a relational database and a RDF mapper. Mappers, for example, D2RQ expose the data on a relational database as a virtual RDF graph, allowing to query this data using SPARQL. This mappers exist to allow Semantic Web application to access the information stored on relational systems [9].

Even with this alternatives, the choice to use a completely semantic system over a relational database was made. This choice came with some disadvantages, the main of them being an increased of the complexity of the system. RDF lacks support on programming platforms compared to relational database systems, thus the amount of coding needed increases. Besides, the current standard for the creation of GIS systems, the PostGIS [52] extension for the PostgreSQL [53] cannot be used. The rest of the spatial databases, including the semantic ones lack support and tools to work with, which has caused an increase in development times.

However using ontologies to define the data model of the platform brings several benefits. The main benefits are detailed in this section.

### **Inference of new knowledge from the existing one**

The main advantage of the ontology based data model design is the possibility of inferring knowledge, that is, creating new knowledge from the existing data. This process usually implies the creation on new triples from the existing triples on the dataset which can be done in two different manners. The first way consists on running the inference process every time a query is made and aggregating the new triples to the results returned. This saves space on the data store but in exchange it increases computation time, for which it is not a very used technique. The more popular approach is running

## 5. DESIGN SPECIFICATION

the inference when data is added to the store and generating new statements which will also be saved. This increases space used and time for processing write operation, however, since read operations are usually way more common than read operations it pays off.

This inference is usually done by specialized reasoning software such as the Pellet OWL 2 reasoner [50]. The data store to be used, Parliament, provides its own reasoner. Parliament's reasoner supports OWL DL inference done when triples are inserted into the data store. The process exploits inference properties like the ones detailed on the previous section to deduce new knowledge, for example, if a property is specified to have a certain domain, then when a triple with that property is inserted into the dataset it is possible to infer that the subject is of the types on the domain of the property. A graphical example of a simple inference can be found in figure 5.3. This example, shows how using inverse relation properties it is possible to obtain inverse properties needing only to specify one end of the relation.

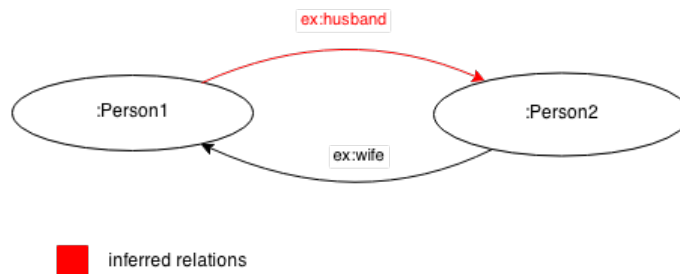


Figure 5.3: Inference of the husband relation as inverse of the wife relation

Many types of inferences are possible. The inference of the classes of a resource depending on a class hierarchy can be done; the example in figure 5.4 shows the usage of the SORELCOM ontology to infer the classes of a resource. Combinations of different inference properties can be used to express complex relations. In figure 5.5 inverse and transitive properties together with ranges and domains, are combined to infer knowledge on a small RDF graph.

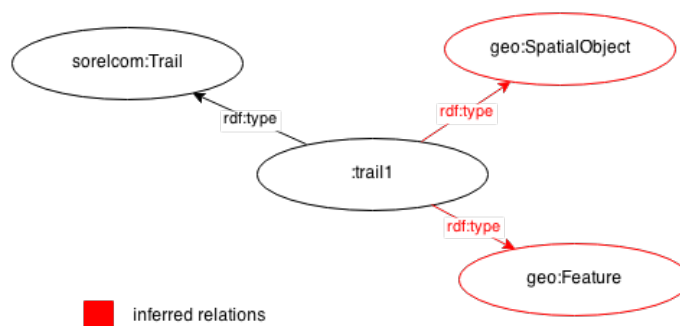


Figure 5.4: Inference of all the classes of a Trail

Depending on the describing language used, it is possible to create very complex rules, and the inference engine will automatically process all of them. This is one of the biggest advantages over relational database systems where a high amount of relations would need to be defined to encode the complex relations that can be represented in a ontology based system and allows to build complex queries.

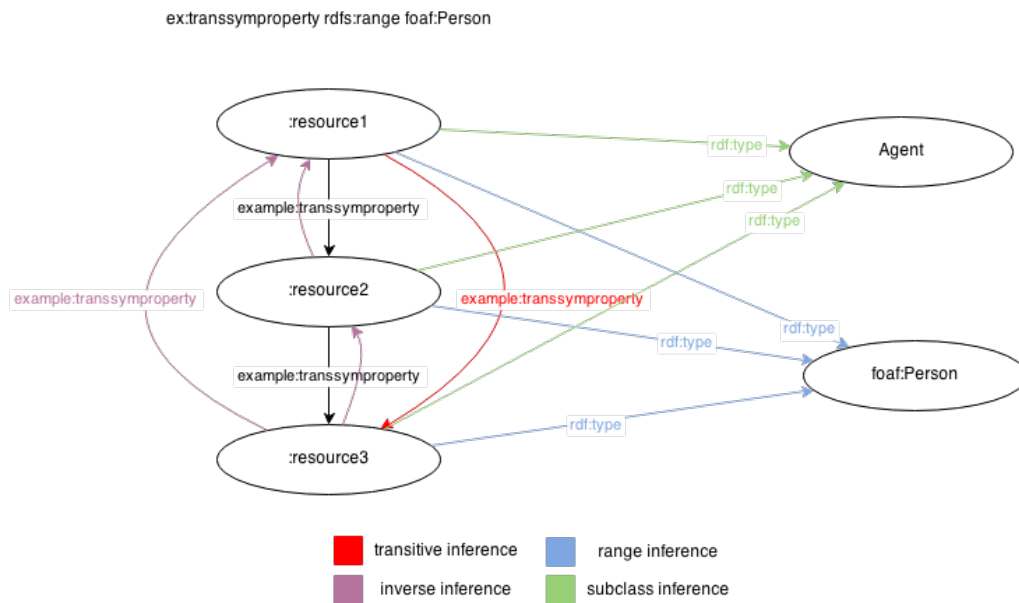


Figure 5.5: Inference of the husband relation as inverse of the wife relation

### Schema and data separation and reusability

Ontologies provides mean for reusing other data models, since it is possible to import ontologies into another one. This way, it is possible to use already defined concepts without the need to develop vocabularies for them. One example of this can be found in the SORELCOM vocabulary, where the FOAF, GeoSPARQL, DC and W3C media ontologies are reused.

Besides, this allows separation between the data model and the actual data. In relational database management systems, the structure of the data is physically created on the system itself (for example, a file for each table) and it is not fully supported to import the model of one database to another. When using ontology based databases, the data model is represented by a ontology which can be imported into any data store, thus it is possible to reuse the schema among several different databases.

### Linked Open data

Ontologies are a tool that facilitate the publishing of Linked Open Data. The inference mechanisms used on these ontologies are not so relevant in this case, however the concepts expressed are crucial. Ontologies allow reusing other vocabularies to express the data model, which is a key point when developing a linked model.

Linked data refers to a set of best practices to publish and interlink structured data for access by both humans and machines via RDF. In order to do this it is necessary to make use of URIs as a real unique identifier, there is no point if a relation that expresses that two persons know each other has a different URI in every dataset. In order to avoid this, well known vocabularies, such as FOAF, and these concepts and relations are uniform among most if not all datasets. This is only possible when using ontology based design, for only ontologies allow this level of reusing.

## 5.4. DESIGN OF THE CENTRAL SERVER

### A note on JavaScript program modeling

All of the software implemented through the project has used JavaScript as the programming language of choice. Due to the single-threaded nature of the JS runtimes, most of the JavaScript programming is done asynchronously.

In addition, the language is not strictly object oriented (although it can be used in that way), however, JS programming paradigms and platforms have tried to bring structure by using modular approaches.

Due to this, there are some considerations to take into account in the class diagrams presented in the following sections.

First, the classes represented are not always classes in the strict sense. Many times these classes are tagged as services, controllers or factories since they follow the pattern of some framework. Even if they are not strictly classes, they behave in very similar ways and they can be represented with the same relations.

Second, in JavaScript most of the data manipulated is an Object, even functions. These objects act as name/value dictionaries. Due to this, it is not necessary to define classes when the object only requires data manipulation and has no operation associated, unlike in object oriented languages such as Java. What would seem as a poor decision choice in other languages (dictionaries instead of classes) is the standard way of working in JavaScript.

Finally, due to the asynchronous nature of the programming environments, the event-oriented programming model is followed. This consist on passing callbacks as parameters to functions, which will be executed when the operations are done (see chapter 6 for more details). Due to this, it is usual to find parameters of type Function in the modeled operations, these arguments are the event callbacks.

### 5.4.1 Web server

The role of the central server is that of implementing most of the functionality of the system and acting as a mediation among the other components. It provides the client application with access to the database; it captures the data sent from the applications, transforms it to RDF and stores it in the database and it is responsible of the analysis of geographical features and of recommendations.

Figure 5.6 shows a diagram of the classes of the server. Most operations are not shown on the diagram because they are simply wrappers around other functions found on related classes. The role and relation of the classes is detailed below:

**Routes:** This class takes care of the routing on the web server. It reads the requests and calls an adequate function depending on the specified route. It implements the REST API, detailed on section 5.4.2.

**Controllers:** There are several controllers present on the server. These modules are the ones implementing the functions that the router calls. Most of these functions perform some analysis

or transformation on the data received, using the functions on the module `GeographyUtils` and then call the `QueryMaker` class in order to retrieve or store data from the database.

Two of the controllers differ a little from the rest. The first one is the `UserController` which makes use of a model in order to store user password hashes on a NoSQL database. It also provides authentication.

The second of these is the `Endpoint` module. This controller exposes a single operation, used to route a SPARQL query to the database. This function also takes care of transforming the response of the database into a format that can be sent to the applications.

**GeographyUtils:** This module provides several utility functions used to analyze and manipulate spatial data. Most of the functions are used to obtain relevant data, for example, the `haversine` operation obtains the distance between two points taking into account the spheric shape of the earth. Other functions transform between the different formats used to represent spatial data in the system, namely `GeoJSON`, `GPX` and `WKT`.

**QueryMaker:** The `QueryMaker` is the most complex class on the server. It takes care of elaborating SPARQL queries for the different functions implemented in the API.

In order to query the database the following chain is followed:

1. A function of the `QueryMaker` is called.
2. The query maker calls a function on the `QueryFactory` or one of its components in order to produce a `Object` representing the query string.
3. This object is sent to the `SparqlClient` which will form a query string and send it to the endpoint exposed by the database in a HTTP request.
4. The response is transformed into a regular JavaScript object or into a `GeoJSON` object.
5. The object is sent to the requester.

**QueryFactory:** This module has the task of producing a object representing a query depending on the request done. It is related by composition to other three objects, which are just specialized factories that translate the read, write and update operations to SPARQL queries.

**SparqlClient:** It is a HTTP client for the semantic datastore. It can be initialized with a set of prefixes used to shorten the queries and with the URL of a endpoint. The task of this object is to transform the `SelectQuery` and `ModifyQuery` items received into SPARQL query strings, sending them to the database and then parsing the response into an adequate format.

## 5. DESIGN SPECIFICATION

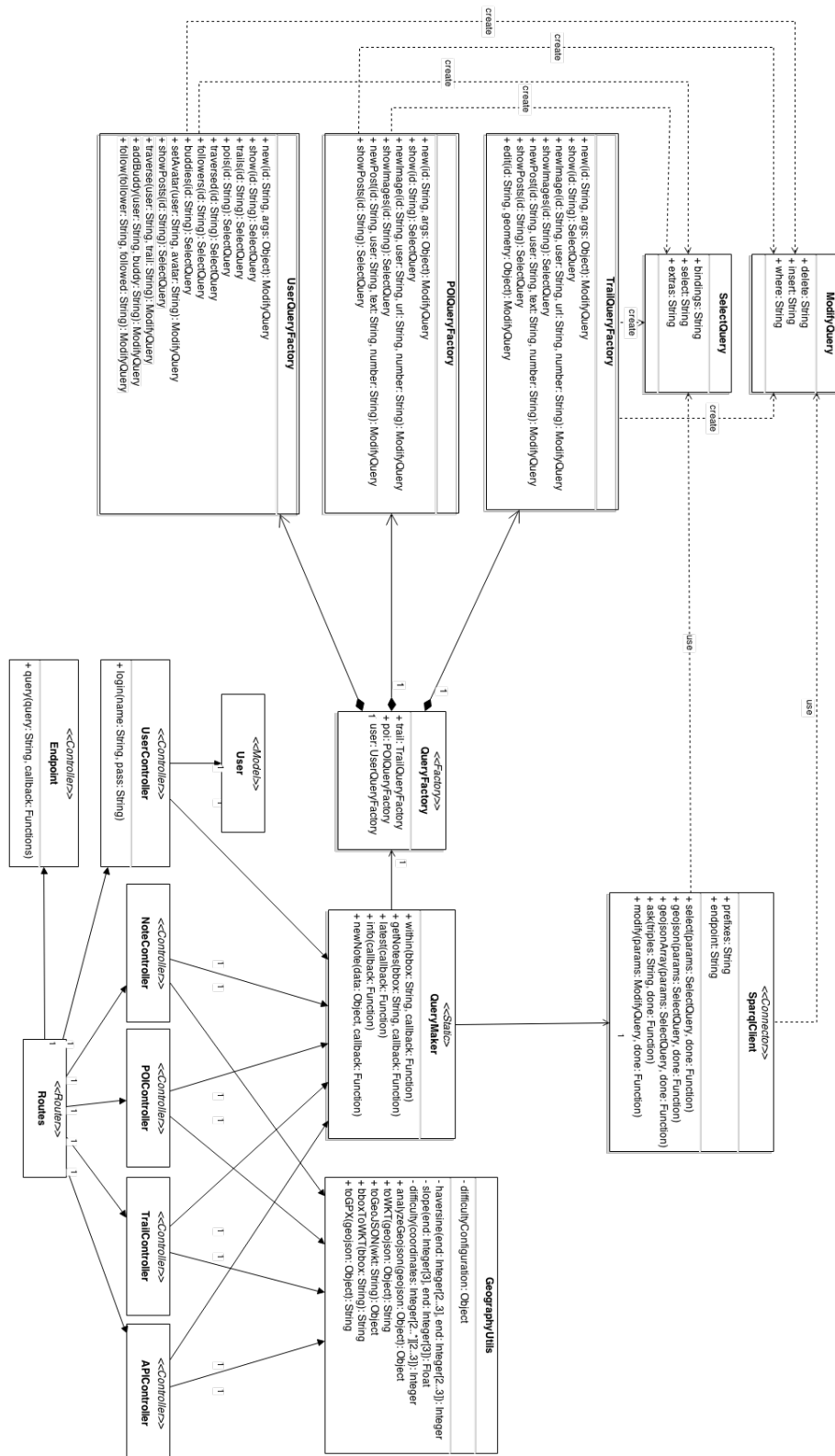


Figure 5.6: Class diagram of the web server



### 5.4.2 REST API

The API on the servers is designed following the REST (Representational State Transfer)[27] style. This style is characterized for the following:

- The server and the client are stateless, all the information needed to give a response is provided in the request. This is not always completely followed, due to the use of cookies. however, some of the practices, for example URL rewriting, are unacceptable in REST.
- Information on the system is exposed as resources which have well defined operations.
- There is a universal syntax for resource identification, each resource is only available through its URI.

The advantages of using REST over other approaches to API styling are various. First, since there is no state to be maintained on client or server, the server scales more easily. In addition, it is possible to provide links to the operations of the object in the API responses, to make the automatic analysis and usage of the API possible.

The graph in figure 5.7 represents the resources and operations exposed by the API.

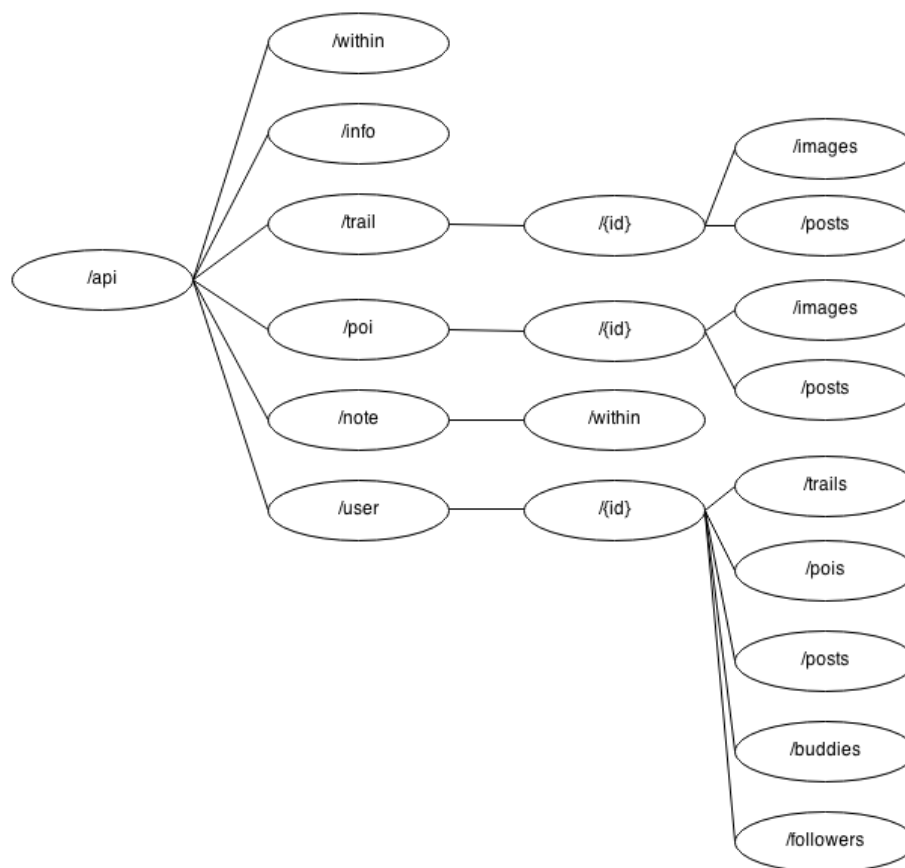


Figure 5.7: Resources and operations of the API

The responses given by the API are JSON objects. This choice is made on the basis that JSON objects are faster to send over a network than other formats such as XML files. In addition, JSON is usually deemed as easier and faster to parse than XML. When sending spatial object, these JSON objects are formatted to follow the GeoJSON standard (see section 3.10).

### 5.4.3 Data aggregator

In order to build a truly linked data application, it is necessary to extract knowledge from the different datasets around the web. Two services have been used to obtain information, OpenStreetMap and Geonames.

OpenStreetMap is a initiative which aims to create and provide free geographic data from around the world. It is a completely collaborative initiative, so all the data is user generated, which may cause inconsistencies in a few occasions.

Geonames is a geographical database which is available for free download under the creative commons license. It consist of over 8 million features from around the world, which provides high performance services. It server more than 30 million web service requests per day.

Both applications provide two ways of accessing their data. The first of them is downloading this data in a file, but they also provide APIs to access different functions.

The platform uses the APIs of these services to access the data. This is done in order to automate the data access and enable the update of the data every period of time without the need for a manual download.

OpenStreetMap provides two different APIs. One of them is optimized for writing operations, while the other ones offers read only functionality. This platform receives data from the Overpass API, which allows for complex queries over large areas.

The Geonames API is read only, however it offers additional functionality such as geocoding and reverse geocoding. It also exposes a set of feature codes which identify the types of features on the database. The API is queried using solely the feature codes for the categories which are relevant to the application.

The updates are run in a script independant to the rest of the server and works following a set of steps:

- For every desired category a request is made to the URL of the API.
- The response is parsed and the features are extracted.
- The features are transformed to the format specified on the ontology. The OpenStreetMaps taxonomy is followed to specify the category.
- The response is checked to see if there are additional pages that the server could not send due to size constraints.
- If there are more pages to query, the process is repeated.

At the current time, the queries are specific to the region of Spain. This is because it would be unreasonable to expect to have a database that can handle features from the whole world in such small development time.

### 5.4.4 Socket server

In order to provide real time communication with the mobile application, the server needs another component. For this, a socket server has been designed, which will establish permanent connection between the server and the client.

The diagram of the socket server is shown in figure 5.8.

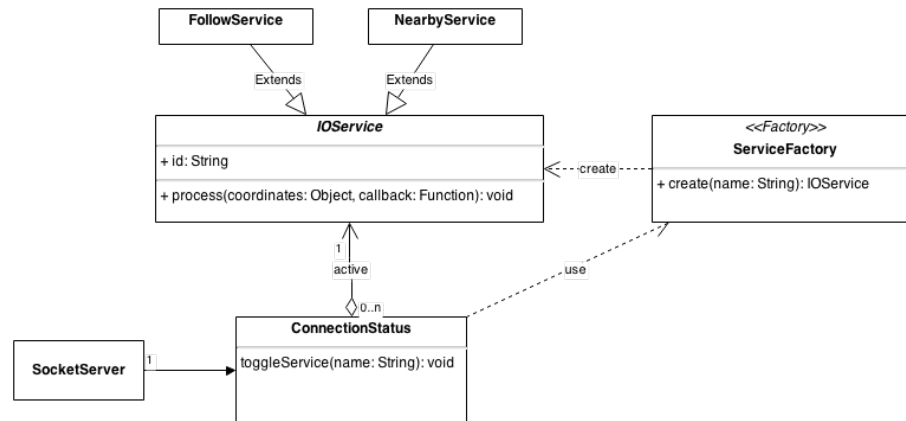


Figure 5.8: Socket server class diagram

**Socket server** The socket server is a process separate from the web server that runs on the same machine. This server takes care of establishing permanent connections with the mobile clients that which to use the real time services.

**Connection status** Each connection needs to keep track of an status, that is, the services it is subscribed to and the option configuration for those services. Through the status, a client can subscribe to different services.

**Services** Services in the socket server are functionalities offered by the platform. One of such services can be dedicated to tracking the features near the user or to follow a certain trail.

The server takes care of receiving messages from the client in the form of coordinates and calling the processing function of every service to which the user is subscribed. This is done following a protocol specified in section 5.6. The sequence diagram for this functionality is shown in figure 5.9.

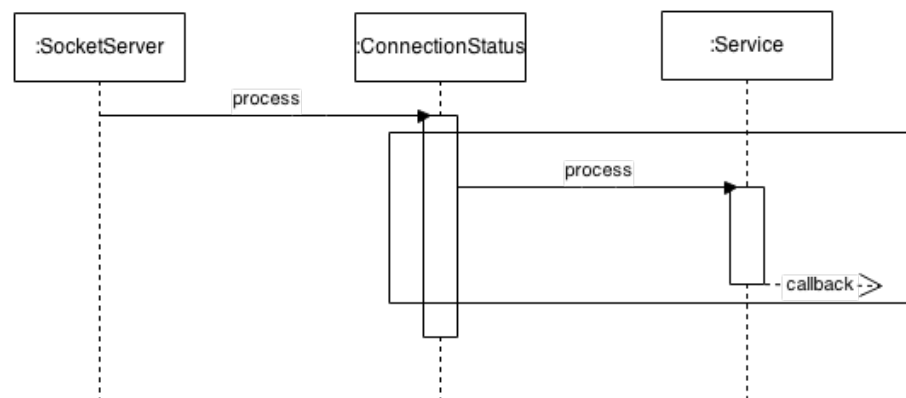


Figure 5.9: Service processing sequence diagram

## 5.5. DESIGN OF THE WEB APPLICATION

The design of the web application has been highly influenced by the framework of choice, AngularJS, (see section 3.6). This framework does not only provide a structure to a project, it imposes it. The

## 5. DESIGN SPECIFICATION

architecture of the application is thus conditioned by this tool, which is why many classes appear as controllers, services or factories; all of them patterns widely used on Angular applications.

The web application gives the user access to the following actions:

1. Register on the platform
2. Log in
3. Browse through the features and users of the application
4. View details about features and users
5. Upload a trail or point of interest
6. Draw a trail on a map
7. Mark a point of interest in a map
8. Edit a trail in a map
9. Explore the world through a map
10. Upload images to a feature
11. Comment on a feature
12. Follow or add a user as a buddy

Figure 5.10 provides a high-level overview of the components that form the web application and the way they depend on each other.

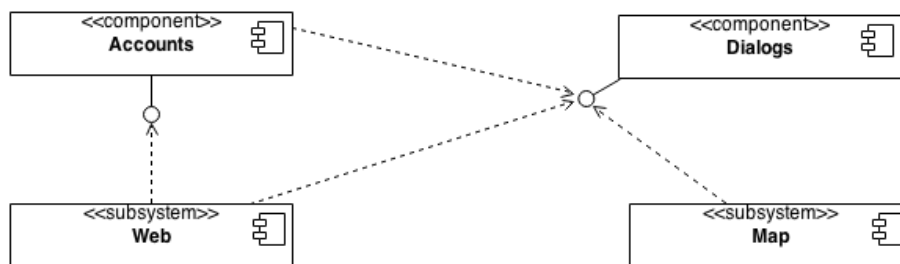


Figure 5.10: High-level view of the web application components

**Dialogs:** Provides the other components with utilities for user interactions. These take form of UI dialogs that allow the user to log in, to load files, etc.

**Accounts:** Implements the functionalities needed to log the user in, to register new users and to store locally the information about the users, to avoid excessive requests to the server.

**Web:** Most of the user interface is implemented in this component. It is a subsystem that contains the views and the functionality needed to browse the information on the server.

**Map:** The map provides the application with a interactive map which allows to explore the trails and points of interest of the world and provides access to the functions of the editor.

In addition to these components, an API service is defined and used through all the components to access the server.

### AngularJS notations

Before describing the internal design of the different components it is necessary to give some notes about how AngularJS structures applications.

**View:** In the context of this framework, a view is an HTML document, enhanced with directives that allow it to access data.

**Controller:** The controllers contain the functionality needed to manipulate the view. The data shown on the HTML documents and the functions used are stored on the controllers.

**Scope:** Named `$scope` in the framework, the scope is a object that binds the view and the controller. The view does not have access to the controller directly, it accesses the scope, thus, when the controllers want to expose information it needs to save it in the scope.

**Model:** A model is any kind of domain related data on the server. It is usually obtained through REST APIs.

**Service:** Services are objects defined for a particular application which can be reused through all the controllers. They are used for data sharing, code reuse, etc. There are several types of services. The most common one used is the plain service pattern, which exposes a singleton object. Since most of the services follow the singleton pattern, it is not explicitly noted on the class diagrams.

**Factories:** Factories are a type of service. The difference with the regular services is that factories just expose a set of functions, they cannot be used to share data. On the other hand they are simpler to implement than regular services.

### 5.5.1 API connector

Figure 5.11 provides a class diagram showing the API connector and the object it creates, which represent the data model on the application.

It is noteworthy to mention how the models in the system work. Since all the data is extracted from the server in a asynchronous way, there is a need to register an event in order to know when the data has arrived from the server. The objects in the model are created as soon as the remote call to the server is done. Then a callback is registered via the `load` method, which specifies how the object will be read and it's data populated.

This way it is possible to bind the data to the views immediately without needing to wait for these object to be populated. Below the role of each class is explained:

**API object** The API class exposes an object that can be used to access the API of the server. This object is obtained from the Restangular module and it provides a way to navigate REST API and query them using the different HTTP methods.

In addition, convenience methods which create and return model objects are provided.

**RemoteObject** It is the superclass of all models obtained from the server. It contains an id, which is a string representing the URI of the resource and a object that allows to query this URI with different HTTP methods.

**User** It represents the users of the application. Users are identified by both their username and their email. The trails and points of interest added, as well as trail buddies and followers.

## 5. DESIGN SPECIFICATION

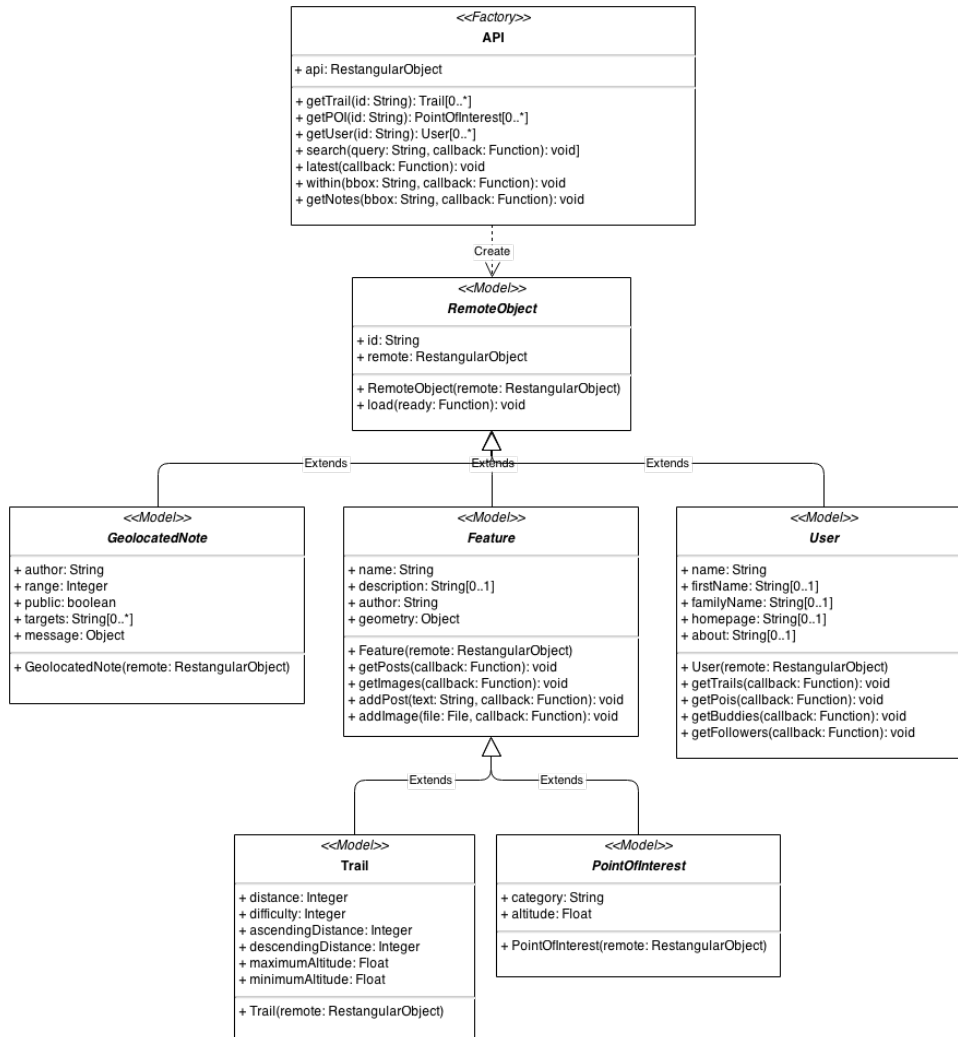


Figure 5.11: API connector class diagram

**Feature** Features are the trails and points of interest on the system. Each of them contains data specific to the type of feature it is, but all allow to access the media and the posts on them.

**GeolocatedNote** These objects represent the geolocated notes of the systems. The difference with features is that they do not contain posts and media is treated differently, for it is the main content.

### 5.5.2 Web

Figure 5.12 shows the class diagram for the web subsystem. This component contains the views and the functionality needed to browse through the information on the platform. The subsystem consists solely on a series of views controllers which make use of the API and dialogs.

**Home page** It is the first page that a new user will find when entering the platform. It shows some guidelines on how to use the application and provides information about the latest changes on the platform.

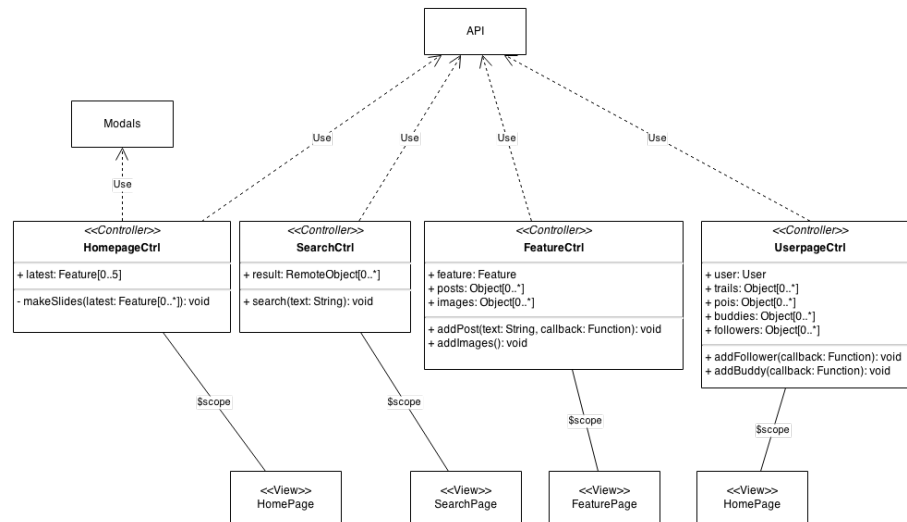


Figure 5.12: Web subsystem class diagram

**Search page** It allows the user to perform text based search on the platform. The retrieved information will adapt its display to the type of resources found. The search takes into account not only the content on the resources but also related resources, however, only trails, points of interest and users are displayed.

**Feature page** It offers a detailed view of the features on the platform and the related information. It provides a map to view the geographic information of the features and allow to create posts.

**User page** It offers an overview of the users on the platform. It allows to see the information that a user has shared on the page and provides a way of following and adding users as buddies.

All the views in the subsystem contain a navigation bar which allows to log in, register and browse in addition to give quick access to the subsections of the page.

### 5.5.3 Map

Figure 5.13 shows the class diagram for the web subsystem. It provides a interactive map through which features on the world can be viewed and trails can be edited.

**Map** The map view provides a map which takes the whole space on the browser. A sidebar menu contains the needed elements to access the functionality of the component. This sidebar provides the editor and the explorer which are build at components of the map view.

**Explorer** The explorer does not provide any direct functionality, however, while this view is active the controller takes care of retrieving information about features that can be found on the current viewport of the map.

**Editor** The editor provides functionality to upload and modify routes and points of interest. All this is done through a point and click interface on the map.

## 5. DESIGN SPECIFICATION

The map is built as a service, in order to be able to share the map element through all the controller that access it. The editor and explorer functionality and data are all contained in their own services which are accessed by the respective controllers.

In addition, a `Tooltip` service is provided which is represented on the map view. It shows information about the current action being taken by the user, such as editing a trail. It acts as a tutorial.

The map works making use of the Leaflet framework. Since the API responds using GeoJSON when querying for spatial data and the library supports direct representation of this format on the map, adding data to it is straightforward. The Leaflet map is contained in the `Map` service and it is accessible by both the editor and the explorer. These two classes however include their own data on the map and must make sure that when the controller is changed the data created by these classes is cleaned from the map. This is done by the controllers themselves, which register and function that destroys the data on the map when the controller is destroyed.

Most of the functionality on the map is done through the registration of events. For the exploring of the world, a request is done to the server every time the user zooms or moves the map. For the drawing of routes and marking of points of interest, an event is registered for clicks on the map. This way is how the interaction with the user is provided in the application, by registering callbacks for the actions it does.

### **Trail edition**

Most of the functionality of the map can be implemented using the events and utilities provided by the leaflet framework. The edition of trails however is a task that has a high complexity related.

In order to provide this functionality, a Leaflet plugin has been developed to create editable trails. Trails are represented as polylines in a map. These polylines are lists of coordinates that are represented in a map by the lines that join them. Editing a trail implies moving these coordinates, which is not supported by default in Leaflet.

There are several plugins already developed which allow this editing, however, they all suffer from the same problem which is performance. Most of this solutions add draggable markers to the coordinates of the line, which results in a lack of performance when there are more than around 200 points. In order to avoid this, the plugin has been developed to show only a limited number of markers, which is necessary considering that trails usually have over 500 points.

Detail about the implementation of this plugin can be found on chapter 6.



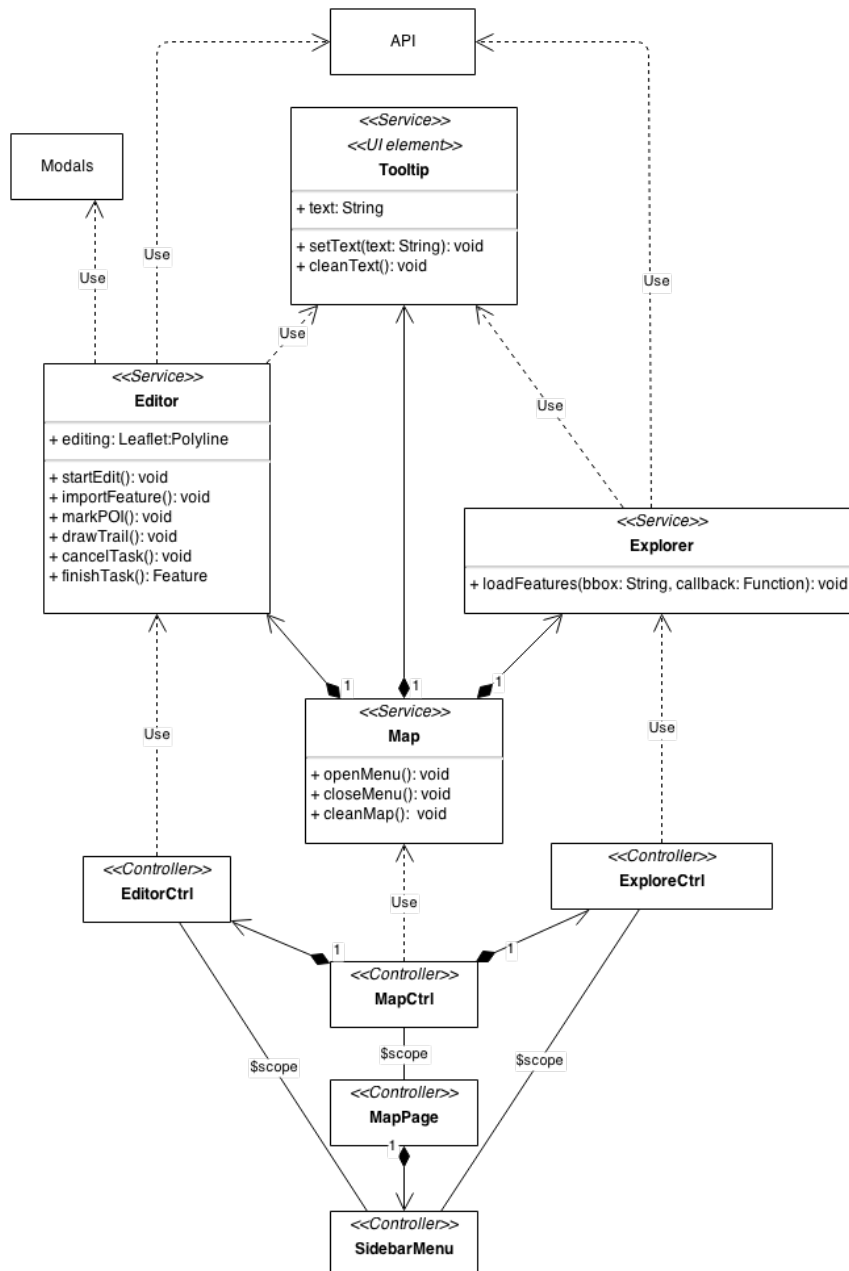


Figure 5.13: Map subsystem class diagram

### 5.5.4 Accounts

Figure 5.14 shows the class diagram for the accounts component. It provides the functionality needed to register and authenticate users as well as storing the logged user locally to avoid excessive queries to the server.

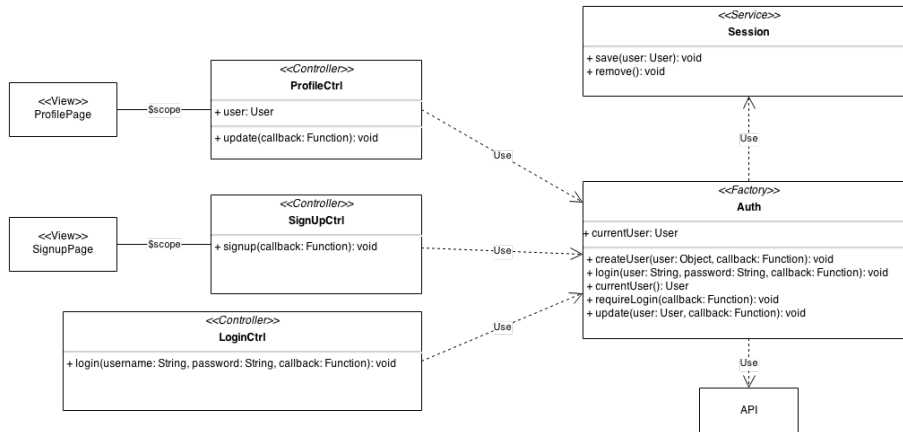


Figure 5.14: Accounts component class diagram

**Profile** The profile allows a user that is already logged in to view his own profile and to change his preferences and personal data.

**Sign up** The sign up page allows a person to register as a user in the platform. The information needed for registering is only a user name, a email and a password. Once registered it is possible to add additional information through the profile page.

**Login** The log in controller is not associated to any view in this diagram for it is used by the dialog system. This is because login may be required in several parts of the application and it is more convenient for the user to have it offered as a modal.

**Auth** It is a service used through all the platform that allows to query the currently logged user, to ask for updates on his information and to log in and register.

**Session** A wrapper over the HTML5 SessionStorage API, which is used to store locally the information of the user once logged in.

### 5.5.5 Dialogs

Figure 5.15 shows the class diagram for the dialogs component. This component provides utilities to facilitate user interaction in the form of dialogs that give access to various functionalities used across the system.

**Log in** The log in dialog provides a very simple interface for the users to log in to the platform. It requires an email and a password for the authentication.

**Feature load** The loading dialog allows downloading trails from the server. It is used on the editor to select a trail to edit.

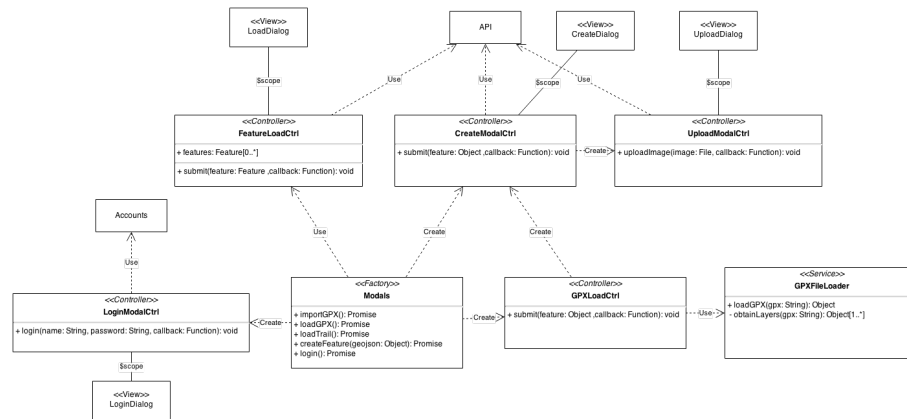


Figure 5.15: Dialog component class diagram

**Feature creation** The feature creation dialog provides a form to create points of interest or trails. It is used after loading a GPX file or marking the feature on the mark. Once the feature is saved, it calls the upload dialog which allows the uploading of images to the platform.

**GPX loading** This dialog provides functionality for loading a GPX file. It analyzes the contents of the file using the GPXLoader service and provides an interface for the user to chose one of the features contained in the file. Once the feature is selected the creation modal will be used in order to obtain the data about this feature.

The `Models` class exposes all the functionality listed, thus, it is the interface through which other components can invoke the dialogs. In essence, this class provides shortcuts for creating dialogs and returns `Promises`, objects that are notified when the modal is closed.

### 5.5.6 Graphical design

The graphical design of a web application is sometimes as important as the logical design and implementation of it. Web pages are usually created to be used by a wide array of users, whether they are tech savvy, they can handle a computer or not. In addition, many different devices have to be able to access the applications, which is also something to take into account when creating the interface.

#### Responsive web design

Responsive web design is an approach to web design which aims to provide optimal viewing experience across a wide range of devices. It is not just about providing a uniform easy to read interface, it has to respond gracefully on every device, optimizing its view for the screen on which it is being displayed [46].

This can be achieved with a series of techniques. Fluid grid layouts are used to reposition elements depending on the size of the screen on which the document is displayed. Responsive graphics are used in order to achieve uniform views through all viewport sizes.

## 5. DESIGN SPECIFICATION

The CSS3 technology supports this design approach with the use of media queries. These constructs allow to specify a output type such as a printer or a screen and a viewport size (width and height). An example of the usage of media queries can be found in listing 5.1.

```
1  @media (min-width: 481px) and (max-width: 768px){
2      /** Styles for the specified viewport */
3  }
4
5  @media screen and (min-width: 780px){
6      . . .
7  }
```

Listing 5.1: CSS3 media query example.

In addition to media queries, the CSS3 framework Twitter Bootstrap [48] has been used to create a responsive application. For this, the framework provides a series of responsive elements, for example, navigation bars which hide their buttons on a submenu when the screen is not big enough. However, the most important element provided by this framework is its grid system, which allows to structure the contents of a web page on different ways depending on the type of screen (desktop, tablet or mobile).

This tool together with custom responsive style sheets has been used to build an application that can be displayed effectively in many devices. An example of this behavior can be found on figure 5.16, which depicts how the map interface adapts to different devices.

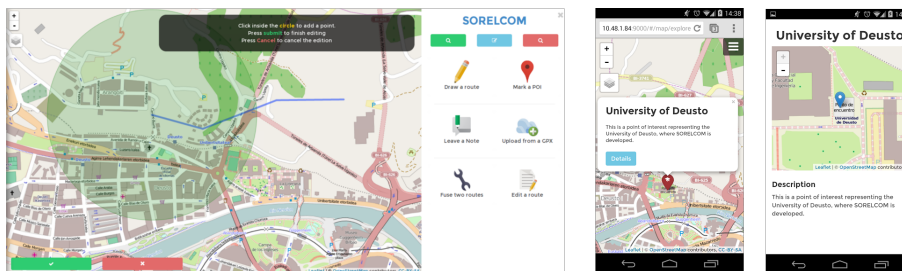


Figure 5.16: Responsive web design on SORELCOM

### Mobile first

When designing a responsive application, the usual train of thought it to think on the structure the interface will follow on desktop devices and then transform it to mobiles and tablets. When following a mobile first approach, contents are designed to be displayed in mobile first and then adapted to tablets and desktop devices.

However, mobile first design implies more that just interfaces. Some platforms offer different URLs for their mobile and desktop versions, however, this causes loss of visibility in search engines, mobile first approaches discourage this practice.

Single pages applications can also be used for mobile first design. The downloading and displaying of a web page takes more time in mobile devices compared to more powerful computers which can be a real issue. SPAs load the document only once and then change the views of the interface using AJAX, resulting in a faster and more fluid execution.

## 5.6. DESIGN OF THE MOBILE APPLICATION

The mobile application has been developed using the HTML5 development framework Phonegap. Due to this, it has been possible to reuse most of the existing functionality from the web application. In fact, the mobile application is structured the same way as the web and uses the same frameworks and libraries.

Most of the components of the web application have been included, with the exception of the map editor. The dialogs of the web application are changed to become regular views instead of pop-up elements.

The mobile application gives access to the following actions:

1. Register on the platform
2. Log in
3. Browse through the features and users of the application
4. View details about features and users
5. Record a trail
6. Export a trail as GPX
7. Upload a trail
8. Receive notification about nearby features and notes
9. Mark a note on the current location
10. Follow a trail

### 5.6.1 Class design

In addition, a new component is added to provide the mobile specific functionality. Figure 5.17 shows the structure of the new component.

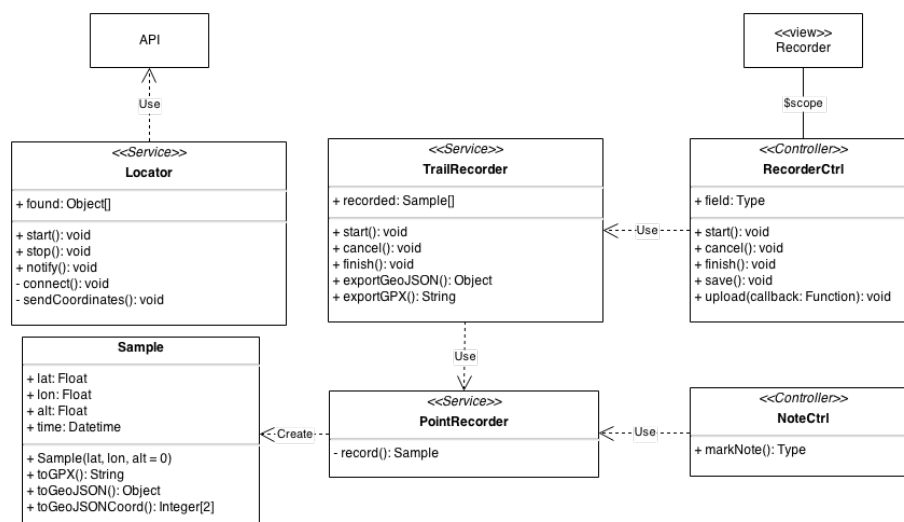


Figure 5.17: Class diagram of the new components of the mobile

**Trail recorder** The trail recorder takes care of periodically saving points by using the **PointRecorder** class. This service is used to record trails that the user is doing. Once a trail is recorded it is

## 5. DESIGN SPECIFICATION

possible to save it as a GPX file or to upload it to the platform.

**Note marker** The function of the note marker is a simple one. It makes use of the same point recorder as the trail recorder in order to mark the current location of the user and saving a geolocated note on the platform.

**Locator service** The locator service is in charge of various tasks. The first of them is always active, unless the user explicitly stops it and takes care of finding nearby points of interest, trails and notes and notifying the user about them. It also checks if the user is inside a route, and in that case it will notify the user when it gets out of the trail.

This component is the one handling real time communication with the server. It does so through the use of WebSockets, however, when the operating system does not provide support for this technology it starts a timer to make periodic calls to the API, providing a pseudo real-time functionality.

### 5.6.2 Real time communication protocol

In order for the mobile client to be able to communicate in real time with the server, a very simple protocol has been designed. When the client wants the server to start tracking its location and sending notifications of nearby features it will send a NEARBY command. When the client wants the server to check if it is following a trail and not deviating it will send a TRAIL command. The server can be in any combination of these two states at any moment.

In order to cancel any of these tracking functions, the command is sent again to the server. The STATE command is used by the client to determine the current state of the server, in order to avoid canceling a function by accident, for example.

The server will not send data to the client by itself, it will wait for it to send its current location with the COORDS order. In addition, the client will be able to configure some parameters such as the distance of on which points of interest are considered nearby with the CONFIG command. A summary of the commands and their parameters is provided in table 5.11.

Table 5.11: Real time communication protocol.

Command	parameters	result
NEARBY	none	toggle tracking nearby features
TRACK	trail id or none	toggle following trail
COORDS	latitude, longitude	response depending on states
CONFIG	key, value	OK or ERROR response
STATE	none	returns the state of the server

The diagrams on figure 5.18 and 5.19 provide a graphical description of the usual procedures on the server. Usually, the client establishes a connection, sends either the NEARBY or TRACK commands, sends the configuration and starts sending coordinates at regular intervals.

These show the expected usual flow on the server, however, they are not the only possibility. It is possible to follow a trail and at the same time receive the information of nearby features. These diagrams however, do distinguish the nature of both commands. While the nearby checking

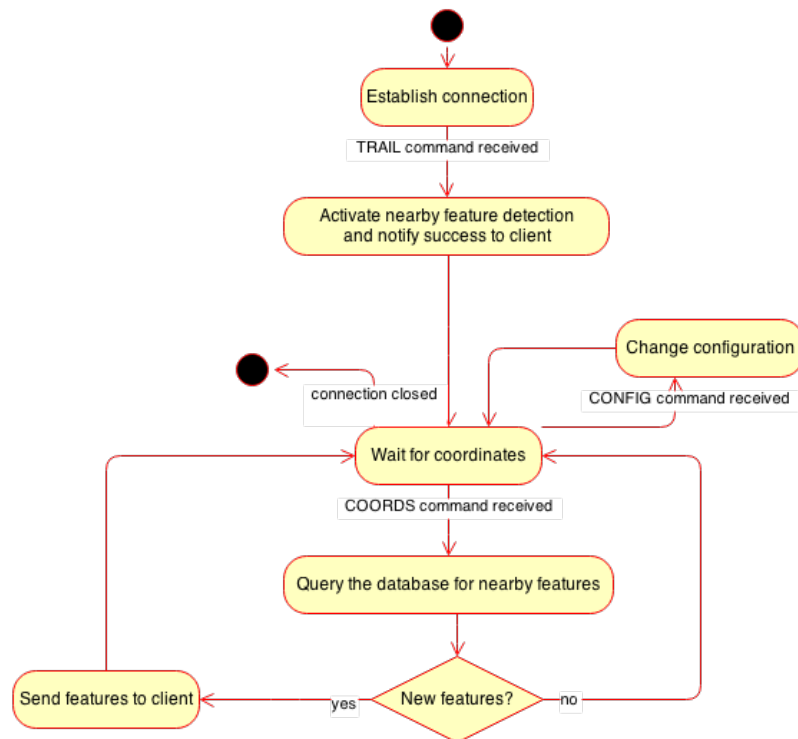


Figure 5.18: Nearby checking flow diagram

functionality has a undetermined duration, it will only stop when told so or when the connection is closed; the trail functionality will stop when the user arrives at the end of the trail.

### 5.6.3 Plugin design

Due to the way the Phonegap library works and the required functionality, it will be necessary to create a series of background services for the application. This services need to be programmed in native language, due to this the application will initially be only developed for Android devices.

Still, a design has been created for the development of these background plugins which can be reused among different applications. This design is shown in figure 5.20. The diagram just shows class relations and very few operations due to the possible implementation differences among different platforms. The architecture for the plugin is identical to that on the socket part of the server, for it implements the protocol to establish real time communication.

**Trail recorder** The trail recorder uses the Point recorder to access the GPS of the mobile device at regular intervals and record a full trail.

**Locator service** The locator is the part of the component that implements the real time communication protocol. It is designed in a identical manner to the socket server, with the difference that includes a client-only service taking care of processing the acknowledgements of the server (sent when adding or removing a service, for example).

The protocol described on the previous section is implemented on this component, as well as in the socket component of the server. This protocol requires to be able to add or remove services to

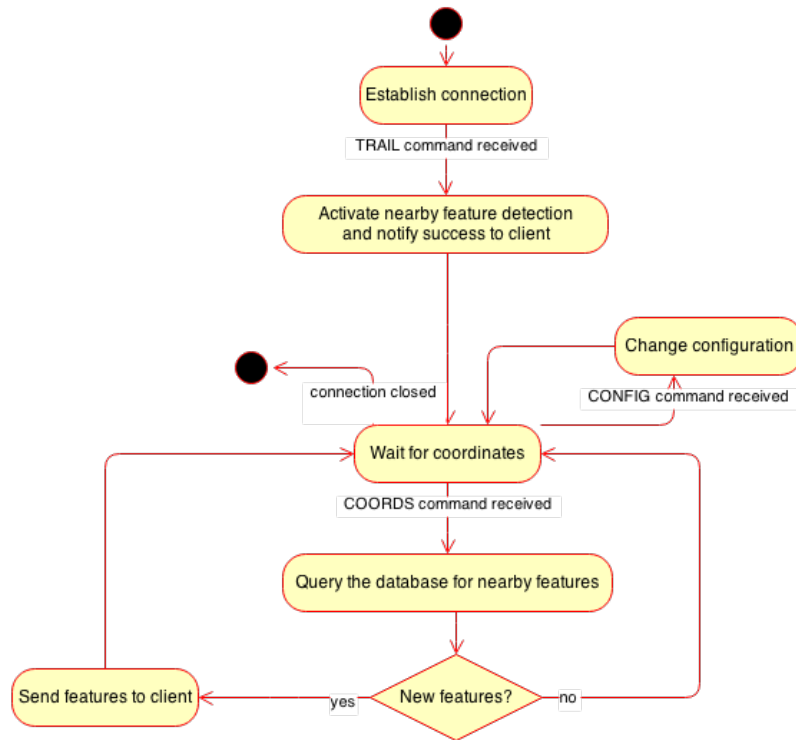


Figure 5.19: Trail following flow diagram

which the user is subscribed. This is done using service identification names. When a status is asked to toggle a service it is checked if the service is already subscribed, if it is then the service is removed, if it is not then it is added.

The sequence diagram that shows this process of subscription can be found in figure 5.21.

### 5.6.4 Graphical design

The mobile application has its appearance defined by CSS style sheets in a similar way to the web application. This, however, does not mean that the appearance of the web can be reused on the application.

Part of the application design is common to both clients, in order to keep a uniform style. Examples of this uniformity are the buttons, fonts, icons and maps used. However, the structure of the application changes radically from web to mobile clients.

Since the mobile application will only be used by smart phones which have a similar aspect ratio and screen size, it is not necessary to use responsive design at the same level. Due to this, the interfaces and the views have been adapted in order to be much more mobile friendly, sacrificing responsiveness in the process.



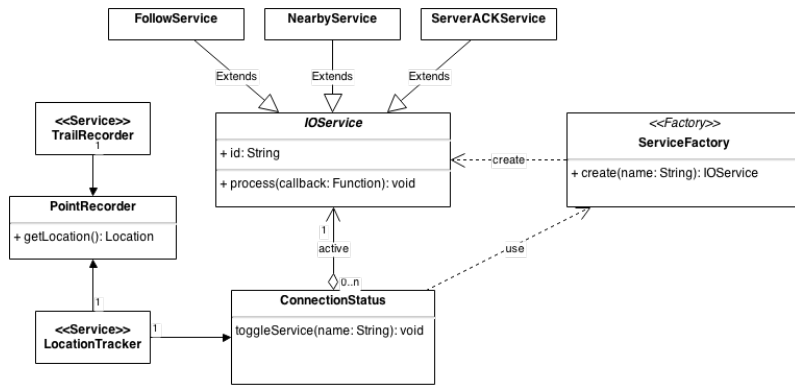


Figure 5.20: Class diagram of the mobile application native plugin

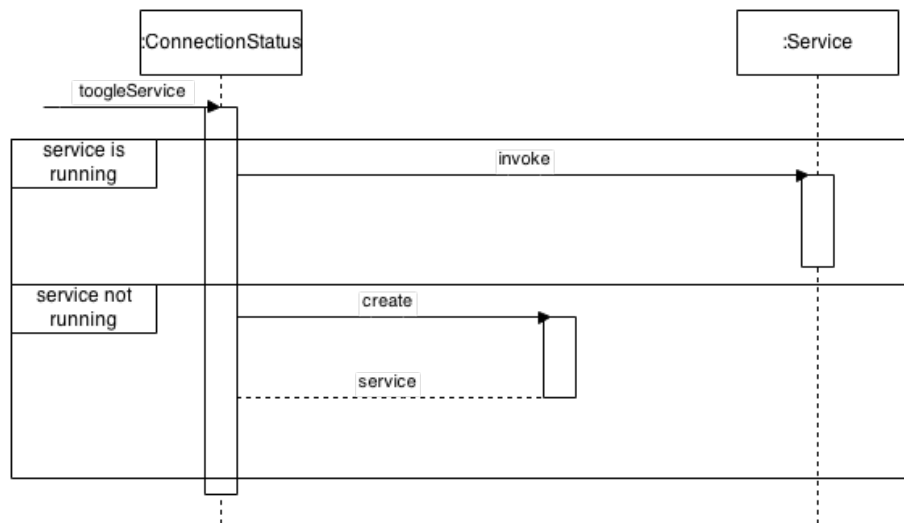


Figure 5.21: Service subscription sequence diagram

## 5.7. DEVELOPMENT ENVIRONMENT

Chapter 3 presents a list of technologies and standards over which a initial research has been carried in order to shape the project. Still, these do not comprise the full list of technologies used on the design and development of the platform.

This section provides a list of the technologies and tools used in the realization of the project, complementing the one on chapter 3 and providing a brief description of each of them and their role on the project.

### HTML5

HTML5 (HyperText Markup Language)[37] is a core technology on the World Wide Web, used to structure and present documents. It is the fifth revision of the HTML standard, it includes semantic tags for the HTML markup language and new APIs.

Its role in this project is to define the interfaces of the clients and its APIs have been used to enhance the functionality.

### **CSS3**

CSS3 (Cascading Style Sheets) is a descriptive language used to specify the look and formatting of documents written in a markup language. It is used together with HTML in order to create visually appealing web pages. CSS3 is the third revision of this languages, which includes new style descriptors, element selectors and functionality.

It has been used in the project to define the appearance of the client applications.

### **JavaScript**

JavaScript, usually abbreviated as JS, is an interpreted programming language, based on the ECMAScript standard. It is defined as an imperative prototype based language, with weak typing. It can be considered object oriented.

It is usually used to provide interaction to web pages, which are just plain documents without JS scripts on them. Due to this, all major browser have a JavaScript runtime implemented. Even though in its origins it was purely a client-side development language, in the last years tools have appeared that allow back-end programming with this language.

It has been used to code all the logic on the developed systems.

### **SASS**

and preprocessor for the CSS language which aims to cut on developing times and improve stylesheet maintainability.

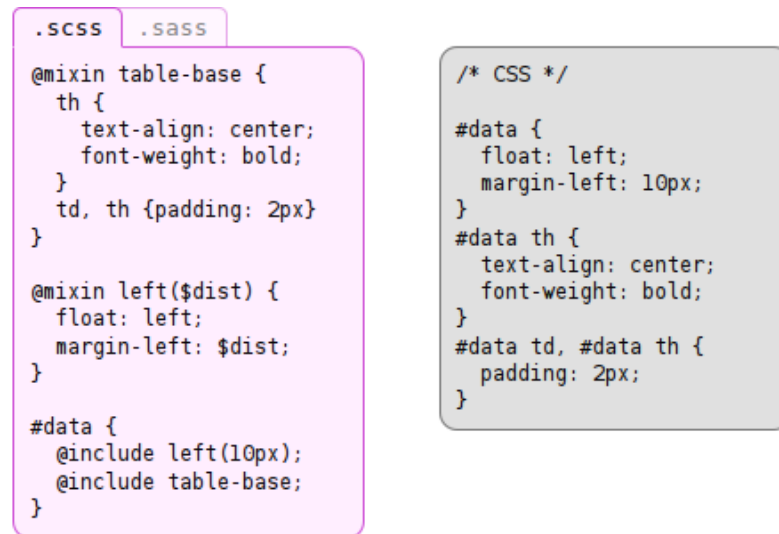
CSS preprocessors have become very popular lately among web developers for various reasons. First, they provide functionality that allows to programatically define styles for a web page instead of just describing, such as variables and functions. Second, these extensions improve code readability and reduce the size of the produced stylesheets. Finally, they follow the principle of DRY (dont repeat yourself), allowing to define the appearance of document without repeating directives. An example of SASS to CSS transformation can be found in figure 5.22.

This framework has been used trough the project to create the style sheets.

### **MongoDB**

MongoDB [16] is a high performance NoSQL document oriented database. It is very used on Node applications due to the similarity of the documents it stores to the object of the JavaScript language. It provides APIs to work with the database using JS directly. In addition, there are node modules such as Mongoose, that provide a ORM using this backend.

MongoDB, together with Mongoose, has been used in the to store in a safe manner the sensible information of the users.



```

.scss      .sass
@mixin table-base {
  th {
    text-align: center;
    font-weight: bold;
  }
  td, th {padding: 2px}
}

@mixin left($dist) {
  float: left;
  margin-left: $dist;
}

#data {
  @include left(10px);
  @include table-base;
}

/* CSS */

#data {
  float: left;
  margin-left: 10px;
}
#data th {
  text-align: center;
  font-weight: bold;
}
#data td, #data th {
  padding: 2px;
}

```

Figure 5.22: SASS to CSS transformation

## Passport

Passport is a authentication framework for NodeJS (see section 3.6). It provides secure authentication using both local account information or several APIs provided by well known services, such as Google, Facebook or Twitter.

It provides the authentication functionality on the server.

## WebSockets

The WebSocket protocol enables a full-duples two way communication method between a client running untrusted code and a server [26]. It is designed to be implemented across web browsers and servers and its API is being normalized by the W3C. It provides real time communication facilities in environments in which it was impossible, that is, web applications.

It has been used to provide real time functionalities in the mobile application when the operating system allows it.

## Socket.IO

Socket.io is a WebSocket API that enables real-time bidirectional event base communication [56]. It consists on two parts, a client library that runs in a browser and a server library for NodeJS, however, both have a nearly identical API.

This library uses primarily WebSockets, however, since this is a relatively new protocol and it is not implemented in every browser and device, it may fall back to other mechanisms such as AJAX polling. In addition to providing a wrapper over WebSockets, it offers several other features such as broadcasting to multiple sockets and storing data associated to each client.

### **AJAX**

AJAX [30] (Asynchronous JavaScript And XML) is a web development technique used to create interactive applications. It allows asynchronous querying to a server, that is, requesting information without the need of updating the whole HTML document.

It has been used through all the project to access the information on the server from the clients. In addition it is used on the mobile application to provide pseudo real-time functions when WebSockets are not supported.

### **UML**

The Unified Modeling Language (UML) is a general-purpose visual modeling language used to specify, visualize, construct and document the artifacts of a software system. It is used to capture decisions and provide understanding of the architecture and class design of the system [58].

UML can be used to capture information about the static design and dynamic behavior of the system and its components through different types of diagram. It is a standard used and accepted worldwide, thanks to which it can be used to expose and interchange information about system designs.

All the component and class diagrams presented in this document have been modeled according to UML standards.

## 6. IMPLEMENTATION DETAILS

---

This specifies the implementation details of the project, that is, the most significant aspects regarding the coding of the system and its functionalities.

The description of this issues aims at easing possible future enhancements of the system and the maintenance of the code. Some specific aspects of the programming of the code are described and some of the choices made during the implementation process are justified.

### 6.1. CODING STYLE

There are several well known coding patterns for JavaScript programs. In this project the Google JavaScript Style Guide [33] has been followed. Below some of the relevant guidelines, and the reasons behind them are listed.

- Always use `var` for variable declarations. This is done to avoid the runtime placing the variable in the global context, which can cause overriding of other variables.
- Variables and functions are named in camelCase, that is, any multi-word name will be written as a single word with the first letter of every additional word in uppercase.
- Always use semicolons. JS runtimes can run code without the semicolons, however, in some cases they may cause subtle hard to debug errors.
- Use `foo.bar = null;` to delete. This is done because the `delete` operator in JS is slower on many runtime environments.
- Use single quotes instead of double quotes. This is a standard across most JS coding styles, for it makes easier to write XML and HTML string, a common task.

This are some of the most common guidelines followed, however, there are more rules and small tricks followed, such as caching the array lengths in loops.

### 6.2. DEVELOPMENT ENVIRONMENT

This section describes the tools used on the development process. This tools are directly related to the development process but not to the final product.

#### 6.2.1 Atom text editor

Atom is a JavaScript coding oriented configurable text editor developed by GitHub [1]. It is available for all major operating systems, however it only has prebuilt packages for OS X. Anyway, it provides a repository for Ubuntu OSs which has been used to install it.

## 6. IMPLEMENTATION DETAILS

The editor is very similar to Sublime Text 2 [62] and while Sublime has more years of development and a bigger community behind, Atom offers native NodeJS and Git integration, which declines the balance in favor of Atom for this project.

It has been used on the project as the sole code editor. An overview of the interface it offers is shown in figure 6.1.

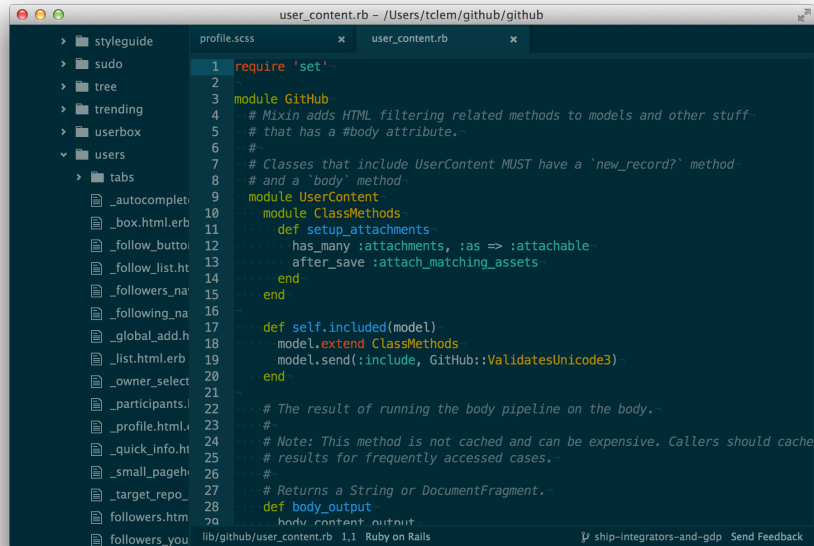


Figure 6.1: The Atom text editor

### 6.2.2 Git

Git [66] is a free and open source distributed version control system with emphasis on speed.

It has been a very popular tool on open source projects ever since it was developed by Linus Torvalds. It supports the classic functionalities found on all version control systems, in addition to several additional functionalities which make it a superior choice.

The main advantage of GIT is being distributed. Thanks to this it is possible to have local repositories in different machines and synchronize all of them on a single repository, having each local repository keep copies of the whole project.

This project is hosted in the GitHub platform, a free open source project hosting site which supports GIT. A student account, providing three private repositories has been provided by the organization itself.

### 6.2.3 Yeoman

Yeoman [64] is a generator ecosystem that helps kickstarting new projects by prescribing best practices and productive tools.

This tool acts as a command line interface that scaffolds projects, that is, it automatically builds project skeletons which incorporate modern tools. It is based on generators, which specify how an project is created and which technologies are to be used in it.

The development team of Yeoman has created a series of official generators, however, there are several community defined generators that can be used to scaffold different types of projects. This project has used the angular-fullstack generator, used to scaffold a express application which uses AngularJS on the client side.

#### **6.2.4 Grunt**

Grunt [36] is a JavaScript automation tool that allow to run tasks such as compilation, testing and minification. It becomes incorporated in the majority of the yeoman generators and it is very useful for the deployment tasks of a project, such as code minification and compilation to JavaScript.

It has been used to run the testing, serving and building processes of the project.

#### **6.2.5 JSLint**

JSLint [22] is a static code analysis tool that promotes and tries to enforce code quality on JavaScript. It works by ensuring that the source code on JavaScript application follows coding rules and best practices.

Grunt can be used to automate the running of JSLint on code changes. In this project, all the source code has been checked using this tool, and no code has been deployed without passing a linter with no errors.

#### **6.2.6 Protégé-OWL**

Protégé is an open-source ontology editor and framework for building intelligent systems[54]. The traditional architecture of the editor is based on frames [47], however, with the standardization of OWL (see section 3.3) support for this language was added to the framework [40].

The current version of Protégé can be used to edit classes and their characteristics, to access different reasoning engines, to edit and execute queries and rules and to visualize relationships between concepts. The tool has been widely adopted by the Semantic Web ontologists, for all their utilities and because it has been adopted among the W3C recommendations [41].

The tool provides a graphical user interface, which can be seen in figure 6.2. Through this editor, other vocabularies can be imported, in addition to create classes and properties for a new ontology and defining restrictions and relations among them. It has been used to create the SORELCOM ontology on the project.

### **6.3. DEVELOPMENT OF THE SERVER**

The first step, after the technology research and the design consideration for the development of the project has been the creation of a project using the Yeoman tool. Even though initially only the

## 6. IMPLEMENTATION DETAILS

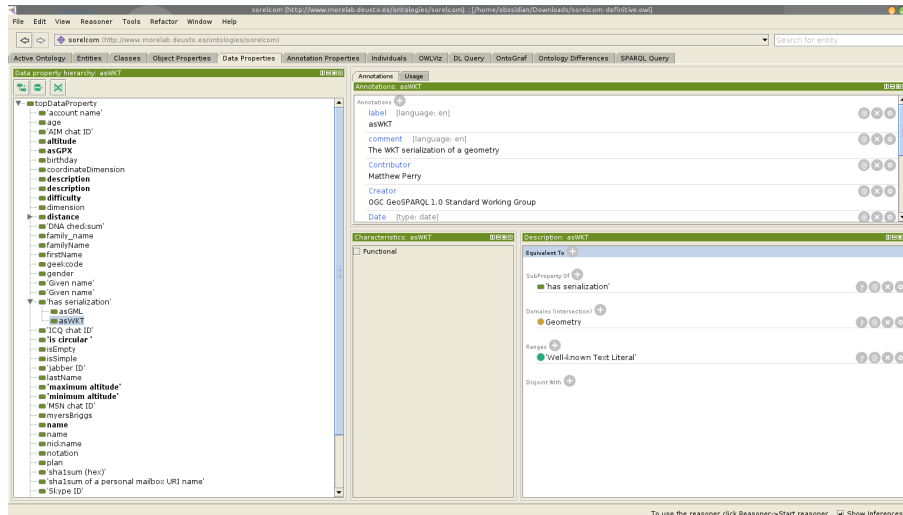


Figure 6.2: The Protégé OWL ontology editor

server is developed, the project skeleton contains a section for the web application, for convenience purposes.

The generator used is the angular-fullstack generator, hosted on and offered through the generator-angular-fullstack npm package.

This command creates the following folder structure.

app Contains the code for the web application. Its structure is detailed in section 6.4.

lib Contains the code for the web server.

test Contains the tests to be run.

node\_modules Contains the libraries used in the server. It is a mandatory folder in any NodeJS application.

dist This folder does not exist initially, however, after running the build task it will be populated with the distributable code.

### 6.3.1 NodeJS application structure

All the code for the server is found inside the lib folder. There is still no consensus by the community on which is the file structure that this kind of servers should follow, thus the following structure has been used.

utils Utility classes, such as the geographic analyser.

connector Classes related to the database connector

controllers Controllers of the server

models Models of the server

routes.js Specification of the URLs that the server will route. It is a file, not a folder.



## Modules

External libraries, modules, are used in the Node projects through the keyword `require`. When a module is downloaded through the NPM package manager it is stored in the `node_modules` folder and can be referenced by a previously defined name. An example of this can be found in listing 6.1

```
1  /** A module called requests is required, an will later be used
2  by referencing the variable where it is stored. */
3  var requests = require('requests');
```

Listing 6.1: NodeJS module requiring

In order to use code defined on files different from the main server, the same mechanism is used. The `require` command obtains an object after running the code on the specified module. The object obtained through the `require` command is called `module.exports`. This object is just a dictionary where the function, objects or classes to be exported are appended.

### 6.3.2 Semantic database

For the server to be able to implement its functionality, it is necessary to first install the database. The storage system to be used is named Parliament, it includes a triple store, a reasoner that supports OWL and GeoSPARQL and a HTTP SPARQL endpoint.

Two types of packages are offered, which can be found in , the QuickStart bundles and the source files. In this project, a QuickStart bundle for a Linux environment has been used. The advantage of this is that there is no need to install any file to the machine, everything is provided and the database can be running by executing the `StartParliament.sh` file provided.

This bundle uses Jetty, a Java based HTTP server, `jetty.xml` file provided. The default configuration is usually enough, however, since large SPARQL queries are to be done, the form submission size has been changed. The element in listing 6.2 has been added in order to allow the upload of large trails.

```
1  <Call class="java.lang.System" name="setProperty">
2    <Arg>org.mortbay.jetty.Request.maxFormContentSize</Arg>
3    <Arg>500000</Arg>
4  </Call>
```

Listing 6.2: Jetty server configuration

Once the server is configured and running, it can be accessed on the port 8080, using the URL `http://localhost:8080/parliament`. Doing so offers a interface for the administration and exploration of the triple store. The final step for the installation of the database is configuring the RDF graph on which the information will be saved to generate spatial indexes, which can be done from this interface, as shown in figure 6.3

### 6.3.3 Database connector

After installing the database, the next step is to develop the database connector. This connector is used to programatically send queries to the data store. Three types of queries will be sent to the

## 6. IMPLEMENTATION DETAILS

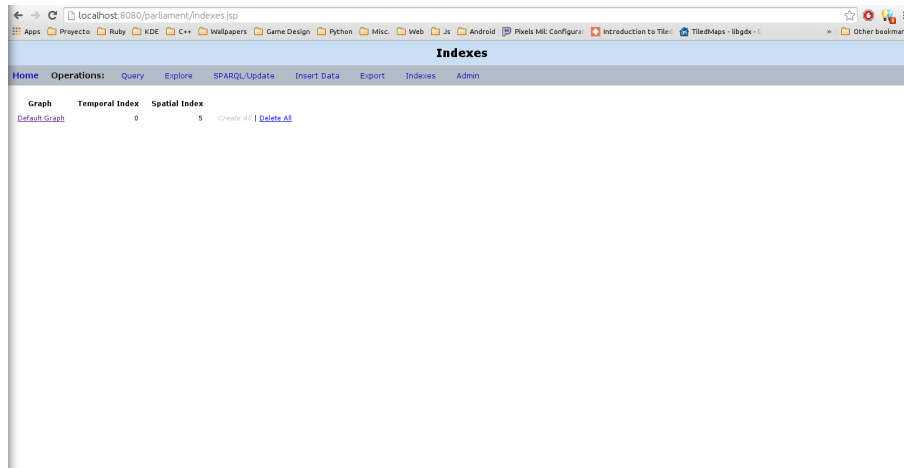


Figure 6.3: Web interface of the parliament triple store

endpoint, accessible through the URL `http://localhost:8080/parliament/sparql:`

**Select** Used to retrieve data from the database. Select queries need to specify a series of bindings, variables that will be retrieved from the database; and a where clause which will contain the information actual query. The structure followed is shown on listing 6.3

```
1 SELECT ?s ?p ?o #variables to be bound separated by a whitespace
2 WHERE {
3     ?s ?p ?o .
4 }
```

Listing 6.3: SPARQL Select structure

**Modify** Used to insert or delete data from the database or both at the same time. SPARQL does not allow to modify existing data, however, it is possible to delete old data and insert the new version in the same query. In order to just insert or delete data, the corresponding part on these queries is omitted. The of the queries is shown in listing 6.4:

```
1 MODIFY
2 DELETE {
3     ?s foaf:name ?name .
4 }
5 INSERT {
6     ?s foaf:name "new name" .
7 }
8 WHERE {
9     ?s foaf:name ?name .
10 }
```

Listing 6.4: SPARQL Modify structure

**ask** Ask queries are used to check if a set of triples exist in the data store or not. They are identical to Select queries, however, they just return a boolean value. In this platform they are used to check the uniqueness of resources. The structure of these queries is shown in listing 6.5:

```
1 ASK { ?s ?p ?o }
```

Listing 6.5: SPARQL Modify structure

## Intermediate Objects

The controller uses a SPARQL client to communicate with the database, however, before the actual query is sent to the database, it has to be built by the application. The query maker that is used by the rest of the application uses a factory to build objects which represent those queries. Then the client will transform these objects into query strings that can be sent to the application.

The reason to use objects instead of directly writing the query is for convenience. It is easier to handle a JavaScript object in the application than a plain string, thus, the server can handle objects until the moment to send the query through HTTP is reached. These objects are defined in the `connector_object.js` file, whose contents are shown in listing 6.6.

```
1 function ModifyQuery(del, insert, where){
2   if(del)
3     this.del = del;
4   if(insert)
5     this.insert = insert;
6   if(where)
7     this.where = where;
8 }
9 module.exports.ModifyQuery = ModifyQuery;
10
11 function ModifyQuery(bindings, where, others){
12   if(bindings)
13     this.bindings = bindings;
14   if(where)
15     this.where = where;
16   if(others)
17     this.others = others;
18 }
19 module.exports.SelectQuery = SelectQuery;
```

Listing 6.6: The definition of the connector objects

## Query factory

The query factory is the class that creates the intermediate objects which will be transformed into the final queries. The implementation of the factory is divided into four classes; the main class containing the methods for creating the general queries and retrieving notes and other three classes containing the methods for creating the query objects that retrieve the defined models from the database.

A good amount of functions has been defined in this cases, one for each possible request and even several for the same request. One example of this methods is shown on listing 6.7, which is used to insert a user in the data store.

## 6. IMPLEMENTATION DETAILS

```
1  this.new = function(args){
2    var userid = calculateURI('user', args.name);
3    var avatar = calculateURI('user', args.name, 'foaf_depiction', 'avatar');
4
5    var where = 'BIND (' + userid + ' AS ?s) . BIND (' + avatar + ' AS ?avatar)';
6    var insert = '?s rdf:label "' + args.name + '" . \
7    ?s rdf:type foaf:Person . \
8    ?s foaf:nick "' + args.name + '" . \
9    ?s foaf:mbox "' + args.email + '" . \
10   ?s foaf:depiction ?avatar . \
11   ?avatar rdf:type sorelcom:Image . \
12   ?avatar sorelcom:storedOn "/images/icon/user.png" . ';
13
14   return new ModifyQuery(null, insert, where);
15 };
```

Listing 6.7: A query factory method

### SPARQL client

The SPARQL client is just a regular HTTP client that uses a series of templates to transform the objects created by the factory into actual SPARQL queries. In order to create a SPARQL client, a endpoint to which queries will be thrown and the prefixes that will be perpended to these queries are specified.

This approach allows to use the client on any endpoint, even if it is not part of the system. An example of the query transformation done by the client can be found in listing 6.8, where a object and the corresponding SPARQL query are shown.

```
1  /** Before the transformation process */
2  var query object = {
3    bindings: '?name ?email'
4    where: '?s a foaf:Person . \
5           ?s foaf:name ?name . \
6           ?s foaf:email ?email .';
7    other: 'LIMIT 5'
8  };
9  /** After the transformation process */
10 var query =
11   'PREFIX foaf: <http://xmlns.com/foaf/spec/> \
12   SELECT ?name ?email \
13   WHERE { \
14     ?s a foaf:Person . \
15     ?s foaf:name ?name . \
16     ?s foaf:email ?email . \
17   } LIMIT 5';
```

Listing 6.8: Query object to SPARQL transformation

In addition to sending requests to the endpoint, the client also has the task of transforming the response of the database into JSON objects to be sent to the requester. The responses of the database is a JSON file following the specification for serializing SPARQL responses as JSON [17], the client just takes that serialization and transforms it into a JSON manageable by the application or into a GeoJSON. An example of this transformation is shown in listing 6.9

```

1  {
2    "head": { "vars": [ "name" , "geometry" ]
3    } ,
4    "results": {
5      "bindings": [
6        {
7          "name": { "type": "literal" , "value": "Example feature" } ,
8          "geometry": {
9            "type": "literal" ,
10           "value": "POINT(0 0)",
11           "datatype": "http://www.opengis.net/ont/geosparql#wktLiteral"
12         }
13       }
14     ]
15   }
16 }
17
18 {
19   "type": "Feature",
20   "geometry": {
21     "type": "Point"
22     "coordinates": [0, 0]
23   }
24   "properties": {
25     "name": "Example feature"
26   }
27 }

```

Listing 6.9: RDF-JSON to GeoJSON transformation

### 6.3.4 Geography Utilities

Located in the util folder, the geography utilities module provides functionality to analyze features in GeoJSON format, as well as transforming them to WKT in order to store them in RDF.

Most of this analysis consists in obtaining from a set of coordinates parameters such as maximum and minimum height, ascending distance, descending distance or total distance. A method to calculate the distance between two coordinates has been programmed. This method uses a mathematical formula called the haversine formula, which can be found on listing 6.10.

```

1  function haversine(pt1, pt2){
2    var lon1 = pt1[0],
3        lat1 = pt1[1],
4        lon2 = pt2[0],
5        lat2 = pt2[1],
6        dLat = numberToRadius(lat2 - lat1),
7        dLon = numberToRadius(lon2 - lon1),
8        a = Math.pow(Math.sin(dLat/2), 2) + Math.cos(numberToRadius(lat1)) *
9        Math.cos(numberToRadius(lat2)) * Math.pow(Math.sin(dLon/2), 2),
10       c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
11   return (6371 * c) * 1000;
12 }

```

Listing 6.10: JavaScript implementation of the haversine formula

## 6. IMPLEMENTATION DETAILS

### Difficulty score

The difficulty score of a trail is the most relevant data that can be obtained. This difficulty score not only gives an estimation of how hard can it be to a user to follow a route, it can also be used as a deciding factor when trail recommendations have to be done.

In the current iteration of the algorithm, the difficulty score is just based on the slopes on the trail. This value, which ranges from 0 to 100, is calculated by evaluating the different slopes of a trail, grouping them by steepness and comparing the length of each steepness level to the total distance of the trail.

The implementation of this method is shown on listing 6.11.

```
1  var score = 0;
2  var meterValue = maxDifficulty/distance;
3  for(var i = 0; i < maxSteep; i++){
4      var steepValue = k / (maxSteep-1);
5      steepValue = ((1 - steepValue) + 1) * steepValue;
6      score += steepValue * meterValue * slopes[i];
7  }
8  return Math.round(score);
```

Listing 6.11: Score calculation algorithm

This implementation will first calculate a value dividing the maximum difficulty, 100, by the total distance of the trail. This can be seen as the score value for each meter of the trail.

After this, it will start iterating every integer from 0 to the maximum slope calculated, usually 20%. A proportion for that steep is calculated, in a way that the highest steep gets a bigger value. This value differs more between steep levels the lower the steepness of the levels, that is, the difference between the 0% slope and the 1% slope values is greater than the difference between the 19% slope and the 20% slope. This allows to differentiate more between trails with low steepness values, which are the most common.

Finally, the distance traversed in that steepness level, multiplied by the proportion of the steepness and the value for each meter, is added to the score. This way, a value that ranges between 0 and a 100 is obtained, being 0 the value of a trail with no slope and 100 the value of a trail with a constant slope of 20% steepness.

The route calculation could take more parameters into account, such as the type of terrain , however that is left for future versions of the system.

Some factors, however, cannot be added so easily. The most notable of them is the distance. Distance is usually considered a major factor when evaluating how difficult a route is, however, in this case it presents several problems.

First, distance can vary a lot, one can find trails that cover solely 500m and others that are as long as 100km. Due to this, it is difficult to limit the range of scores when distance is taken as a parameter. Besides, the actual value of the distance is a very subjective factor. For a person that is traversing the trail by foot it may not count as much as for another person that is running through it. Due to this, it has been decided to leave distance outside of the parameters that affect difficulty.

### 6.3.5 Controllers

The implementation of the controllers is the most trivial task on the development of the server. The job of the controllers is to take the data received on the request, usually a parameter on the URL or an object on the body of the request, extract the data and call a function of the query maker.

Some functions, usually the ones which insert features to the database, need to be analyzed in order to extract the spatial information implicit on their coordinates. In this cases, the controller routes the information to the `GeographyUtils` class, receives the information and sends it to the database. Figure 6.4 shows a usual control flow for a controller.

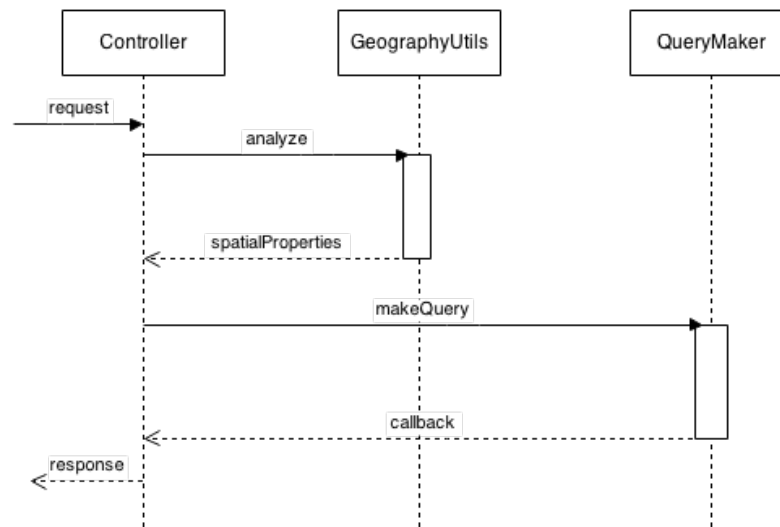


Figure 6.4: Sequence diagram for most controller requests

Controllers also need to take care of analyzing the response. The Express framework middleware parses the HTTP requests into a JSON object and passes it as a parameter to the controller. In addition, a response object is provided to the controller which can be used to send a response. An example is shown in listing 6.12.

```

1      module.exports.addImage = function(req, res, next){
2          if(!req.user)
3              return res.send(401);
4
5          if(req.body.flowChunkNumber !== req.body.flowTotalChunks)
6              return res.send(200);
7
8          short.saveImage(req.files.file, function(err, name){
9              if(err) return res.send(500);
10             q.Poi.addImage(req.params.id, req.user.name, name,
11                 function(err, data){
12                     short.returnOK(err, data, res);
13                 });
14             });
15     };
  
```

Listing 6.12: Controller method for obtaining the trails of a user

## 6. IMPLEMENTATION DETAILS

### File upload

Listing 6.12 is an example of a controller method that can handle file uploads. Files are uploaded using the `flowjs` [38] library, which sends the files on chunks to the server.

It is only possible to read the file once all the chunks have arrived. Due to this, it is necessary for the controllers which upload images or videos to the file to check if the chunk of data is actually the last chunk to be sent. In case it is not, then a response acknowledging the client that the data has been received must be sent.

Once the file is fully uploaded, it can be accessed through the `req.files.file` object. As a sidenote, it is impossible to send more than one file in the same request using this method, however, on the client side it is possible to send multiple files but they will have to be processed one by one on the server.

### 6.3.6 Routes

The routes of the server are crucial to implement the REST API. In order to define the routes that will be parsed by the server, the Express object offered by the framework must be used. This object exposes five methods to define the URLs that will be read: `get`, `post`, `put`, `delete` and `all`; each of them corresponding to a HTTP method, except for `all` which accepts all methods.

These methods receive two parameters, the first of them is a string or a JavaScript regular expression defining the URL that can be requested and the second argument is the function that is called when a request to that URL is sent.

Listing 6.13 defines the URLs from the trail resource of the API. Some of these receive parameters, by specifying a name after a colon; this parameter will be read by the controller on the request object. The same URL can be called from different HTTP operations, however, if no callback is defined for a certain method, the server will send a 404 response.

```
1 app.get('/api/trails', trails.getList);
2 app.post('/api/trails', trails.new);
3 app.get('/api/trails/:id', trails.get);
4 app.post('/api/trails/:id/images', trails.addImage);
5 app.get('/api/trails/:id/images', trails.getImages);
6 app.post('/api/trails/:id/post', trails.addPost);
7 app.get('/api/trails/:id/post', trails.getPosts);
```

Listing 6.13: API implementation for the trails resource

### 6.3.7 Recommendations

Recommendations, which are currently shown on the profile page of a user, are obtained using SPARQL queries. It is possible to do so thanks to the level of complexity that can be embedded into a single query.

In the first version of the recommendations implemented, the following factors are taken into account:

- Score of trails



- Routes traversed by buddies
- Routes traversed by followed persons
- Routes added by buddies
- Routes traversed by followers
- Distance of the route to the usual location of the user

Listing 6.14 shows some criteria used on queries to obtain recommendation candidates. It is noteworthy mentioning that these queries are not done independently, they are merged into a single query; in this document they are shown separately for clarity.

In addition to the filtering shown, the distance of the routes to the usual location of the user is calculated. This is not an actual filtering factor, however, it server to order the candidate routes by their distance to the user. Once this ordering is done, only the few firsts are picked and sent to the user as actual recommendations.

```

1  SELECT DISTINCT ?trail WHERE{
2    ?trail a sorelcom:Trail ; geo:hasGeometry ?tg ;
3    sorelcom:difficulty ?difficulty .
4    FILTER(abs(?difficulty-50) < 25)
5    ?tg geo:asWKT ?twkt .
6  }
7
8  SELECT DISTINCT ?trail WHERE{
9    ?user rdf:label "username"; sorelcom:trailBuddyOf ?buddy .
10   ?buddy sorelcom:traversed ?trail .
11 }
12
13 SELECT DISTINCT ?trail WHERE{
14   ?user rdf:label "username"; sorelcom:follows ?followed .
15   ?followed sorelcom:maker ?trail .
16   ?trail a sorelcom:Trail .
17 }

```

Listing 6.14: Criteria for route recommendations in SPARQL queries

By following this method, the recommendation system can analyse the existing relationships between users. In addition to this, the location of the user is taken into account to a certain point, recommending only nearby routes when there are sufficient candidates. This way, the expressiveness of RDF is combined with spatial queries in order to build a location aware recommendation service.

Usually, recommendation systems use some sort of artificial intelligence, however due to time constraints it has been impossible to implement a more complex recommendation methods. The only possible AI used is the inference engine of the database. Note however, that this is a first version of the final system and there are plans to improve this recommender system with better algorithms and with the addition of more parameters, such as user reviews.

### 6.3.8 Authentication

The generator used provides a already built authentication service. This service uses the passport and mongoose NodeJS modules to provide a way of authenticating and storing users.

## 6. IMPLEMENTATION DETAILS

Mongoose is a object database mapper (ODM) that provides a way of modeling models to be used on mongodb databases. It provides a schema based solution for modeling the application data of NodeJS applications.

Passport is a flexible authentication middleware for NodeJS. It is based on the concept of strategy, which is a way of representing how the users will be authenticated. Different strategies allow using Facebook or Twitter as authentication services, however, in this project only the local strategy based on local database authentication is used.

The generator used provides a User modules (in the models folder) which contains a nick name, a email and a password. In addition, it comes with a pre built controller for managing users and authenticating them.

In order to take advantage of this, the project uses the provided model and controllers and builds methods for saving and retrieving from the semantic storage on top of them. An example of this is the creation of users, which follows the following procedure:

1. Retrieve data from the user and save it to the mongodb database
2. Check that the data has been correctly saved
3. Save the user to the triple store
4. Check that the user has been correctly saved
5. If the user is not correctly saved on the triple store, delete it from the mongodb database
6. Return a response according to the result of the operation

### 6.3.9 Real time communication

The real time communication to be provided has not been implemented in the first phase of the server development. Instead, in order to be able to provide an early prototype of the web application, this functionality has been delayed until it is needed on the development of the mobile application.

## 6.4. DEVELOPMENT OF THE WEB APPLICATION

### 6.4.1 Angular application structure

The app folder of the project contains the code for the web application. It follows a standard Angular folder structure, consisting on the following:

bower\_components Third party libraries installed through bower

styles Style sheets of the application

scripts JavaScript code of the project

    Controllers Controller source code

    Services Service source code

    app.js Application configuration source code

vendor Third party libraries used

views HTML views of the application

### 6.4.2 Application

The application file is the first thing to define in a AngularJS project. It is used to create the application itself, to specify which external modules will be used and which will be the routes and views it accepts.

In order to create single page applications, the framework uses the concept of views. When a Angular application is loaded in a browser, its index.html file is shown, however, this document usually has a view component defined by the `ng-view` angular directive.

The router of the framework uses the current URL of the browser to embed a different HTML document, called partial view, in the place where the view should be. In this project, the module `ui-router`, an extension of the regular routes has been used.

This module offers enhanced routing capabilities, such as defining nested views. In order to configure this component, the concept of state is used. A state of the application has a URL and a partial view associated. In addition, a controller can be defined to manage that partial view. Some states however, are abstract, meaning that they are only intermediate states with no specific controller and are used to route to nested views. Listing 6.15 shows the contents of the `app.js` file of the project. Only two state definitions are shown for clarity purposes, since the real number of states is way bigger.

First, an application is created using the `angular.module` function, which receive a name for the application and an array with the names of the modules to use. All the source files, even the modules, are included as regular JavaScript scripts in the index.html file and the framework will wait for the full loading of the document to execute the code.

Then, a configuration function is used to specify the states that the application will accept. It is also possible to indicate where the browser should be routed when a URL that does not match any state is introduced.

### 6.4.3 API

The API component is used by the rest of the application almost everywhere, thus it is necessary to build it the first. The API acts as a service which comes uses a module called Restangular to query the server.

The service offers a object to the rest of components that can be used to retrieve any resource from the API, in addition to some utility functions. The usage of this object, as well as its definition can be found in listing 6.16

This API uses the concept of remote object to provide a means of receiving data from the server without the need of defining a callback each time the API is called. This objects are just containers for data, initially empty. The objects are created by receiving Restangular object which they call immediately after their creation.

A callback is registered to populate the object when the server responds. This callback will call the `digest` method of the AngularJS root scope, which causes the application to modify the views in order to show the new data.

## 6. IMPLEMENTATION DETAILS

```
1    var app = angular.module('sorelcomApp', [  
2        'ngCookies',  
3        'ngResource',  
4        'ngSanitize',  
5        'ngAnimate',  
6        'ui.bootstrap',  
7        'ui.validate',  
8        'ui.router',  
9        'ui.router.util',  
10       'leaflet-directive',  
11       'flow',  
12       'restangular',  
13   ]);  
14  
15   app.config(function ($stateProvider, $urlRouterProvider) {  
16  
17       $stateProvider.state('web', {  
18           abstract: true,  
19           url: '',  
20           templateUrl: 'partials/layout.html'  
21       });  
22       $stateProvider.state('web.signup', {  
23           url: '/signup',  
24           templateUrl: 'partials/signup.html',  
25           controller: 'SignupCtrl'  
26       });  
27   });
```

Listing 6.15: AngularJS application configuration

In addition, a method is provided so that another callback can be registered for when the data arrives to perform additional operations. If the data has already arrived, the callback will be invoked immediately.

Functions of the data, such as getting points of interest and trails can be executed immediately, for they don't depend on the data to be retrieved from the server, only on the provided id.

Listing 6.17 shows the implementation of one of such objects.

### 6.4.4 Accounts

Most functions of the accounts component are already provided in the application in a similar manner to the server. The `Auth` and `Session` services are already created, however, a few changes have been made to the first of them in order to include the functionality needed to require login on some functions.

In addition, the sign up controller and view are provided, and have been left almost unchanged. This controller allows to register a user using a nick name, a email and a password. The only change made has been introducing a double password check before sending the data to the server.

The profile on the other hand is not created during the scaffolding. The profile controller simply downloads the information of the current logged user, using a convenience method of the user resource on the API. This information is displayed in a simple manner in the view and a form is provided to modify it.

```

1      angular.module('sorelcomApp')
2          .service('API', function Geo(Restangular) {
3              this.api = Restangular.all('api');
4
5              this.getTrail = function(id){
6                  return new Trail(this.api.all('trails').one(id))
7              };
8          });

```

Listing 6.16: API service class

```

1      function User(remote){
2          var that = this;
3          this.prototype = new RemoteObject(remote);
4
5          this.remote.get.then(function success(data){
6              that.name = data.name;
7              that.firstName = data.firstName;
8              that.familyName = data.familyName;
9              that.homepage = data.homepage;
10             that.about = data.about;
11             that.ready = true;
12             $rootScope.$digest();
13         });
14
15         this.getTrails = function(callback){
16             this.remote.all('trails').getList(callback);
17         };
18
19     }

```

Listing 6.17: Remote User class

When a user signs up it will be immediately redirected to the profile page, in order for it to update the data of his account, such as names and profile picture. It is also possible to see all the activity that the user has done on the platform through this page. Of course, the access is restricted in a way that each user may only access his own profile.

The profile is also used to show recommendations for the user, however, it is not the only place where these can be viewed. If a person is logged in and goes to the homepage, it will be able to see the five trails that are considered most fitting.

### 6.4.5 Web

The component that has been called "Web" due to its similarity to that of a regular web page consists on four pieces, the home page, the search page and the feature and user views.

#### Home page

The home page consists on a simple controller that downloads the latest activity on the platform or the user recommendations if the user is signed in. The results are displayed on the page using a peculiar slider.

## 6. IMPLEMENTATION DETAILS

Since the home page is the first view where a user will arrive when entering the platform, it is important that it is visually appealing. Due to this, the information obtained from the server is not displayed as is.

Since this information will always consist on geographical features which have been created or modified recently, the view shows them on a map. A slider is created which will allow the user to see different maps containing the features obtained. The interaction of these maps has been disabled so that there is no interference with the interaction with the slider. In addition, a event checks when the slide has changed in order to recalculate the size of the map, necessary due to the implementation of the Leaflet library.

### Search page

The search page offers a simple interface to make text based search on the server. The controller's only task is to send the query string to the server and to wait for the response to arrive.

Most of the work is done on the view. Since the received data is heterogeneous, users, trails or points of interest may be received; the view must take care of showing the different information of any of the results. This is done using angular directives to hide or show HTML elements. The code for this is shown on listing 6.18.

```
1  <main>
2  <ul>
3    <li ng-repeat="item in searchResults" class="row">
4      <a ui-sref="{makeRef(item)}" class="col-sm-12 col-xs-12">
5        <article>
6          <div class="icon col-md-2 col-sm-2 col-xs-2">
7            
8          </div>
9          <div class="info col-md-9 col-sm-10 col-xs-10">
10           <h5>{{item.name}}</h5>
11           <div class="details">
12             <p ng-if="item.content">{{item.content | limitTo: 40 }}...</p>
13             <p ng-if="item.comments">
14               <span class="fa fa-comment"> {{item.comments || 0}} </span>
15             </p>
16             <p ng-if="item.author">
17               <span class="fa fa-user">&nbsp;</span>
18               {{item.author}}
19             </p>
20           </div>
21         </div>
22       </article>
23     </a>
24   </li>
25 </ul>
26 </main>
```

Listing 6.18: The search view document

Several directives have been used in this (and other) views, including the following:

**ng-repeat** Iterates over a array on the scope and creates a element for each instance.

`ng-if` Creates a HTML element when the condition is fulfilled and destroys it when not.

`ui-sref` Specifies the state to which the link leads. Provided by the ui-router module.

`ng-src` Alternative to the `src` directive which admits brackets (used to indicate angular variables).

`ng-show` and `ng-hide` Hides or shows a html element depending on a condition. Similar to `ng-if` however it does not create or destroy elements. More efficient when there is a high change ratio on the data. Not used on the shown view.

## Feature page

The feature page is just a view showing the details of a certain feature, that is, a point of interest or a trail. The page uses a map to provide a representation of the feature.

To create this map, the Leaflet-angular-directive module is used, which allows to embed map following an Angular programming style. It is possible to create a map without using this module, however, for maps which provide solely viewing functionality is is way more convenient to use the module. The usage of this directive is shown in listing 6.19.

```
1 <leaflet width="100%" height="300px" geojson="geojson" id="viewMap"></leaflet>
```

Listing 6.19: Leaflet usage with angular on feature pages

Images and videos of a feature are shown on this page on a slider. Reviews are also shown on this view, and the uploading of actual posts is implemented through a form which requires a text and a numerical value for the rating.

The information is displayed in a similar manner to the search page, by using angular directives the information that is available for different resources is filtered.

## User page

The user page is almost identical to the feature page. The view on the user page does not show any map, for there is no spatial data. On the other side, a navigation menu is provided to view the trails, points of interest, buddies and followers of a user.

An option is provided to follow and add a user as a buddy when the user being viewed is different from the user logged in. In essence it works in the same way as the feature page, all the data is loaded at the start and if the user makes a write action a query is sent to the server.

### 6.4.6 Map

The Map uses a single general view. This view in addition contains other two nested views, one used for exploring functionalities and the other one offering editing operations.

The general or parent view contains a map which takes the full browser screen. In addition, a sidebar containing the HTML for the nested views. The sidebar can be toggled to be displayed, function that has been implemented completely in CSS.

## 6. IMPLEMENTATION DETAILS

```
1  angular.module('sorelcomApp').service('Map', function Map(API){
2    var that = this;
3    this.initMap = function(id){
4      var baseLayers = {
5        'OpenStreetMap': L.tileLayer(
6          'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
7        )
8      };
9      this.map = L.map(id, {
10        layers: [baseLayers.OpenStreetMap],
11        minZoom: 3,
12        worldCopyJump: true
13      });
14      L.control.layers(baseLayers, null, {position: 'topleft'})
15        .addTo(this.map);
16      this.map.locate({ setView: true, maxZoom: 15 });
17    };
18  });
```

Listing 6.20: Map service implementation

The controller on the parent view implements only the functionality to show and hide the sidebar. All the map creation is done through a service, shown on listing 6.20

This service is used by the rest of the explorer and controller services to access the map object and add or manipulate data on it. The controllers on the map component are just gateways that call the methods defined on the services from the views, the actual computation is done on the services.

### Explorer

The explorer does offers a single function. This function is called every time the map is moved or zoomed and queries the server for the features that can be found on the current viewport of the map, represented by its bounding box.

The explorer also generates HTML elements that will be bound to every feature shown in the map. This is used to show the text data (name, description) of a element and to link to the corresponding feature page.

The basic functions of the explorer are shown in listing 6.21. The `loadGeoJSON` function takes the results and parses them to create the layers and the popups that will be added to the map.

Trails are only shown when the zoom level is enough to distinguish the routes from points. This is determines by the `canViewRoutes` function, which check the zoom level of the map. If it is not enough, then the server is asked to return only points of interest.

The explorer provides no direct interaction with the user, except for the map zooming and moving event. Instead, the space available on the sidebar is used to show links to the detailed views of each feature and to show a legend of the icons used to represent each element on the map.

### Editor

The editor provides several functions related to the manipulation of trails and points of interest.



```

1  angular.module('sorelcomApp')
2    .service('Explorer', function Explorer($rootScope, $q, API, Map, $compile){
3
4      this.init = function(){
5        Map.map.on('moveend', showView);
6      };
7
8      function showView(geojson){
9        if(!that.categories)
10         return;
11        var query = { bbox: Map.map.getBounds().toBBoxString() }
12        if(!canViewRoutes())
13         query.type = 'POI';
14
15        that.canceler = $q.defer();
16
17        API.withHttpConfig({timeout: that.canceler}).get('within', query).then(
18         loadGeoJSON);
19      }
20    });

```

Listing 6.21: Explorer service implementation

**Uploading of a GPX file** The upload of GPX files is independent of the map, however, it is accessible in this component because all functions that upload data are grouped on it.

The upload opens a modal which will load a GPX file specified by the user. This GPX is converted to a GeoJSON object to be manipulated by the application and the different features are extracted. Only one can be picked to be created.

Once a feature is picked, a form is used to obtain the relevant data such as the name of the feature or the category if it is a point of interest. This data is sent through a post request to the server.

**Drawing a trail** A trail is drawn by adding coordinates to a feature. For this, the editor creates a empty polyline and adds it to a map. After that, a callback is bound to the map click event which adds a coordinate to the polyline. This only happens if the distance between the previous coordinate and the new one is less than 500 meters. The method for doing this is shown on listing 6.22

```

1  this.startDraw = function(){
2    Tooltip.setText('Click on the map to start editing');
3    that.editing = L.polyline([]).addTo(Map.map);
4    var latlngs = that.editing.getLatLngs();
5
6    Map.map.on('click', function(e){
7      if(latlngs.length > 0 && e.latlng.distanceTo(latlngs[latlngs.length-1]) > 500)
8       return;
9      that.editing.addLatLng(e.latlng);
10     that.editing.redraw();
11     Tooltip.setText('');
12   });
13 };

```

Listing 6.22: Map drawing function

## 6. IMPLEMENTATION DETAILS

Stopping the editing of the trail requires only to unbind the click event. Once this is done, the editing variable can be transformed into a GeoJSON object and the procedure to follow is the same as when uploading a GPX file.

**Marking a point of interest** The process for marking a point of interest is very similar to that of creating a trail. The difference on implementation is that the click callback must be unbound as soon as the click is done. Besides, once the point is marker, the creation dialog is called instantaneously. The implementation of this functionality is shown in listing 6.23.

```
1  this.markPOI = function(){
2    Tooltip.setText('Click on the map to choose location');
3    Map.map.on('click', function(e){
4      this.stopTask();
5      Modal.create(L.marker(e.latlng).toGeoJSON());
6    });
7  };
```

Listing 6.23: Map point of interest marking function

**Editing a existing trail** In order to edit a trail, a dialog is prompted to ask the user to import from a GPX file or to download it from the platform. If it is imported from a GPX, the functionality mentioned before is used, however, instead of uploading to the platform, it is shown in the map. If it is downloaded from the platform, a list of trails is obtained using the API and the GeoJSON for one of them is obtained when the user selects it.

In this case the JSON file cannot be directly added into the map. It has to be processed first into a editable polyline, which will be the layer added to the map. This editable polyline is developed as a plugin to the Leaflet library.

### **Editable polyline plugin**

In order to edit trails, a means to do so with a point and click interface is provided through the plugin. The goal is to let the users drag the coordinates of a polyline in order to edit them. The library does not provide any event callback that would allow this in any way, the only draggable elements are markers, which represent single points.

A workaround to providing editable lines is creating a polyline and binding each of its coordinates with a marker that can be dragged. When the marker is dragged, then the coordinates of the line can be updated with the new position. Listing 6.24 shows the code for this.

This however does not serve to improve the performance of the editable line. In order to avoid excessive computation time, markers are removed from the map when too many of them are shown at the same time. To do this, a check must be made every time the map is moved or zoomed and if the number of markers is above the maximum, the layer where they are kept is removed. Code for this functionality is shown on listing 6.25.

In addition to coordinate dragging, it is necessary to provide functionality to add and remove coordinates from the map. Adding functionality is provided by setting additional markers in the midpoint between each adjacent coordinate pair. When one of these extra markers is dragged, a new coordinate is added to the polyline.

```

1  this.markerGroup = L.LayerGroup();
2  this.markers = [];
3  this.latlngs = this.line.getLatLngs();
4  for(var i = 0, l = this.latlngs.length; i < l; i++){
5      var marker = L.marker(latlngs[i]);
6      this.markers.push(marker);
7      marker.addTo(this.markerGroup);
8      marker.on('dragend', function(e){
9          latlngs[markers.indexOf(marker)] = e.latlng;
10         line.redraw();
11     });
12 }
13 this.markerGroup.addTo(map);

```

Listing 6.24: Code for adding markers to allow edition of polylines

```

1  var that = this;
2  map.on('moveend', function(){
3      var count = 0;
4      for(var i = 0, l = that.markers.length; i < l; i++){
5          if(map.getBounds().contains(that.markers[i].getLatLng())){
6              count++;
7          }
8      }
9      if(count > maxMarkers && map.hasLayer(that.markerGroup))
10         map.removeLayer(that.markerGroup);
11     else if(count <= maxMarkers && !map.hasLayer(that.markerGroup))
12         map.addLayer(that.markerGroup);
13 });

```

Listing 6.25: Code for removing markers to avoid performance issues

Removing markers on the other hand is simpler. When the user right clicks a marker the corresponding coordinate is removed. This however implies calculating a new midpoint, so that it is still possible to add coordinates in that segment. Pseudo code for these actions are shown on listings 6.26 and 6.27, the actual code is not shown for clarity purposes.

```

1  extraMarker.on('dragend', function(e){
2      var index = this.extraMarkers.indexOf(e.target) + 1;
3      this.latlngs.splice(index, 0, e.latlng);
4      this.markerGroup.splice(index, 0, L.marker(e.latlng));
5      coordinateAdded(index);
6  });

```

Listing 6.26: Add coordinates on a editable line

The calculation of the middle point of two coordinates is done through the midpoint formula, as shown in listing 6.28.

### 6.4.7 Dialogs

The dialogs component offers means of providing interaction with the user through pop-up boxes, also called modals.

## 6. IMPLEMENTATION DETAILS

```
1 marker.on('click', function(e){
2     var index = this.markers.indexOf(e.target);
3     this.latlngs.splice(index, 1);
4     coordinateRemoved(index);
5 });
```

Listing 6.27: Remove coordinates on a editable line

```
1 function midpoint(latlng1, latlng2){
2     return L.latLng((latlng1.lat+latlng2.lat)/2,
3                     (latlng1.lng+latlng2.lng)/2);
4 };
```

Listing 6.28: Midpoint formula implementation

The central class on this component is called the `Modals` service, used to invoke this dialogs. The Twitter Bootstrap framework offers functionality and styles to create dialogs and the module `ui-bootstrap` provides Angular services to use them.

This is done through the `$modal` service, which allows creating a dialog from a defined HTML template and a given controller. The central class of the component uses this service to create different predefined modals. One example of this is shown on listing 6.29

```
1 angular.module('sorelcomApp').service('Modal', function ($modal){
2     this.loadGPX = function(){
3         var modal = $modal.open({
4             templateUrl: 'partials/modals/load.html',
5             controller: 'GPXLoadCtrl',
6         });
7         return modal.result;
8     };
9 });
```

Listing 6.29: Modal invoking shortcuts

Every method that creates a shortcut returns the result of that shortcut. This object is what is denominated a promise. Promises are event listeners that listen to a single success or failure from an asynchronous callback. It is possible to define what the promise will do in case of success and what it will do in case of error, in a way similar to callbacks.

By returning the promise object to the controller summoning the modal, it is possible to let it define how the application will behave once the modal finishes its task. This way it is possible to reuse the same code for different functions. For example, the GPX loader modal will return a GeoJSON as a result, which can be used to create a new modal and upload it or can be directly embedded in a map.

The controllers of the modals will return a result using a service called `$modalInstance` provided by the same Angular module. Using the function `$modalInstance.close()` it is possible to return values to whoever is listening the promise. Example 6.30 shows how this works.

Most of the implementation of the modals consists in simple functions, mostly to get some data from the user, taken from a form or a button. Most of the implementation of the modals consists on this way of communication, to provide ease of use and reusability.

One modal, however, performs a relatively complex function by itself.

```

1  $scope.continue = function(){
2      $modalInstance.dismiss();
3      Modal.create($scope.selected);
4  };
5
6  $scope.import = function(){
7      $modalInstance.close($scope.selected);
8  };

```

Listing 6.30: Modal closing and communicating functions

### GPX loading modal

The GPX loading modal is in charge of parsing GPX files and extracting the features contained within. In order to do this, a small JavaScript library called togeojson is used, which converts from well known geographic encoding formats, such as GPX and KML to a GeoJSON object.

This library will always return a FeatureCollection even when only one feature is found. The controller for this modal delegates on a service called Loader, which uses this library to generate a GeoJSON and then parses the GeoJSON extracting all of its features.

Once all features are extracted, the user is shown a menu with various maps, each containing one of the features. Once one is selected, the rest of the creation procedure is just standard form data extraction and AJAX calls to the server.

The code for the feature extraction is shown on listing 6.31

```

1  angular.module('sorelcomApp').service('Loader', function Loader(){
2
3      this.extractLayers(text, format) {
4          var geojson = toGeoJSON.gpx(jQuery.parseXML(text));
5          if(!geojson) return null;
6          if(geojson.type === "FeatureCollection"){
7              var json = [];
8              for(var i = 0, len = geojson.features.length; i < len; i++){
9                  geojson.features[i].properties.gpx = text;
10                 json.push(geojson.features[i]);
11             }
12             return json;
13         } else {
14             return [geojson];
15         }
16     }
17 });

```

Listing 6.31: Modal closing and communicating functions

## 6.5. DEVELOPMENT OF THE MOBILE APPLICATION

The mobile application reuses many of the classes present on the web application. Thanks to this, the accounts and web components have been reused without any major change. The folder structure of the mobile application is the following:

## 6. IMPLEMENTATION DETAILS

**app** Contains the application, in a similar way to the server. The contents will be compiled into the **www** folder in order to deploy the application.

**www** Contains the scripts, views and styles that the deployed application will use.

**test** Contains the tests

**platform** Contains the code specific to the platform. In the **src** folder inside is where the Java classes of the application can be found.

The changes for the API have been minimal too, instead of defining a relative URL to make requests to, the URL has been changed to the absolute URL of the server where the application is hosted. The change simply means that instead of having **/api** as the base URL of the API service, this URL will be **apps.morelab.deusto.es/sorelcom/api**.

The functionality offered by the modals has also slightly changed. Since the application will only be executed on the mobile, the dialogs are no longer pop up boxes. This implies slight changes to the views and to the controllers in order to remove the communication functions. However, most of the modals, such as the GPX importing have been removed for there is no need of them in the mobile application.

### 6.5.1 Real time functionality - Server side

In order for the mobile component to communicate in real time with the server, it is necessary to implement this capability on the web server. This is done using the **socket.io** library.

It is possible to make **Socket.IO** application and a **Express** application work in the same machine and port, however, for this it is necessary to modify the server initialization code, found on the root of the server, in the file **server.js**.

For this it is only needed to require the **socket.io** module and to tell the library object to start listening on the web server, as seen on listing 6.32.

```
1  var io = require('socket.io');
2  var express = require('express');
3
4  var app = express();
5
6  ...
7  app.listen(config.port, function () {
8    console.log('Express server listening on port %d in %s mode',
9      config.port, app.get('env'));
10 });
11 var sio = io.listen(app);
12 require('./lib/socket/init')(sio);
```

Listing 6.32: Socket.IO server initialization

The **require** keyword is used after the initialization of the socket in order to define the behavior of the **WebSocket** once connections arrive. The file specified in the **require** call will be passes the **Socket.IO** object as a parameter and its code will be executes as in a normal module, with the difference that it will not need to export anything.

## Authentication

Due to the way the real time functionality works, it is necessary to identify the user using the service. Instead of authenticating the user through the socket, the application uses the session established on the web server.

In order to do this, there is no need to modify code on the existing web server. A secret key is defined on the configuration files when specifying the behavior of the Express session manager. This secret key, `sorelcom secret` during the development of the project, has to be used to configure Socket.IO authorization mechanisms. The code on listing 6.33 shows how it has been done.

```

1  sio.configure(function(){
2    sio.set('authorization', passportSocket.authorize({
3      key: 'connect.sid',
4      secret: 'sorelcom secret', //In lib/config/express
5      store: config.sessionStore, //In lib/config/express file
6      fail : function(data, accept) { accept(null, false); },
7      success: function(data, accept) { accept(null, true); }
8    }));
9  });

```

Listing 6.33: Socket.IO authorization

## Communication protocol

The communication protocol defined in section 5.6.2 is implemented mostly on the server. In order to support this protocol, the server uses a subscription mechanism.

With this mechanism, a connection can subscribe or unsubscribe from a service, such as following a determined trail, at will. This enables to have a undetermined amount of services functioning at the same time, for example, a client may be checking for nearby features and at the same time following trail A and B. Besides, it is relatively easy to implement using JavaScript dictionaries.

When a client connects to the server, a State object is created, which will contain the configuration options and the services it is subscribed to. It is noteworthy to mention that the configuration options are global, thus it is not possible to configure trail following distance differently for each trail. The implementation of this class is shown in listing 6.34

Services follow a pattern similar to regular object oriented interfaces. The idea behind this is to use polymorphism to call the services to which the connection is subscribed. However, JavaScript does not provide support for inheritance out of the box. Still, it is possible to implement polymorphism redefining methods of the prototype of the class. The implementation of the service that implements the trail following service is shown on listing 6.35.

Services implement a function called `process`, which processes the operations needed to implement the service. The example in listing 6.35 queries the database to check if the coordinates are inside of the trail being followed. If they are it checks if the trail has finished. The result is sent through a callback to the main process which will send it as a response to the client.

Due to the way JavaScript works, having a base class in this moment makes no difference with just creating the different services without any inheritance. This approach however, provides a higher extensibility if there is a need to add data to the base class in the future.

## 6. IMPLEMENTATION DETAILS

```
1  function State(){
2      this.subscriptions = {}
3      //Load configuration from the user profile
4      this.configs = loadConfiguration();
5      this.subscribe = function(service){
6          if(this.active.service){
7              var id = this.subscriptions.service.id;
8              this.subscriptions.service = null;
9              return id;
10         } else {
11             this.subscriptions.service =
12                 ServiceFactory.createService(service);
13             return this.subscriptions.service.id;
14         }
15     }
16     this.setConfig = function(key, value){
17         if(this.configs.key)
18             this.configs.key = value;
19     };
20 }
```

Listing 6.34: Subscribing functionality

The main loop of the socket server implements the functionality needed to read the data. This function checks the command received to perform the different operations. When a subscription is asked it will call the State object of the client and subscribe a new service. When coordinates are sent to the server it processes all the services it is subscribed to and sends the responses back to the client.

The data received is a JSON object, thus there is no need to parse the query done by the server, it is already presented as a object to the process. A callback is registered for each command sent by the client and the argument members of the data sent are read. The code for the listening process can be found in listing 6.36.

The code subscribes (or unsubscribes) a service on the state of the current connection when the appropriate command is received. When coordinates are received the server loops through all the subscribed services and returns the response when each service is finished processing.

It is important that the server returns the identity of the service every time a new subscription is created or one is deleted. This is used in the client to keep a registry of the services running currently.

There is an additional callback used when the socket is disconnected. It is used to clean all the data accumulated by the user.

### 6.5.2 Real time functionality - Client side

The client side mobile exclusive functionality are directly related to the real time functionality of the system. This includes tracking nearby features, detecting when the user is deviating from a trail or recording a GPX file.

These functionalities are share something in common: they are continuous and executed during large periods of time. Due to this, it is unreasonable to pretend this functionality to work only when



```

1  function FollowService(trail){
2      this.trail = trail;
3      this.prototype = new Service('follow ' + trail);
4
5      this.prototype.process(coordinates, callback){
6          q.isInside(coords, trail, function(err, data){
7              if(err) return callback(err);
8              if(data.boolean === false) return callback(null, 'FALSE');
9              q.isLast(trail, function(err, data){
10                 if(err) return callback(err);
11                 if(data.boolean === false)
12                     return callback(null, 'TRUE');
13                 callback(null, 'FINISH');
14             });
15         });
16     }
17 }

```

Listing 6.35: Trail following service implementation

the application is on screen.

In native mobile applications, this kind of batch functions are implemented using service processes. Phonegap, however, does not allow this. Since the applications created using this library are run on a WebView, similar to a browser, JavaScript code can only be executed while the application is open.

There is a workaround for this issue however. Phonegap implements native functionality with something called plugins. These plugins are pieces of native code that can be called from the application itself. Therefore, it is necessary to implement the real time functions of the mobile application as plugins.

This implies coding in Java, the native language for the Android platform. The GPS of the device is accessed natively in this classes to provide the server with the current coordinates of the device, and a Socket.IO Java library is used to communicate this to the server.

Due to this, the application ends up using HTML5 development for interface manipulation and interaction (as well as for the functions borrowed from the web application) and using native code for the background processes.

### Background service plugin

Two background services need to be implemented in the application. One of them is used for the recording of coordinates of a trail, the other one for the communication with the server.

The first service is the most straightforward one, it just needs to call the GPS on the device every interval of time. To develop the services, a Phonegap plugin called Background Service Core Plugin, found in , has been used. Through the usage of this plugin, it is possible to develop a Android background service using a simple class structure.

In the Java class representing the service, a doWork method has to be written that provides a functionality to be executed at each interval. The interval at which it is executed is specified in the JavaScript code.

## 6. IMPLEMENTATION DETAILS

```
1  sio.on('connection', function(socket){
2      var state = new State();
3
4      socket.on('nearby', function(){
5          var service = state.subscribe('nearby');
6          socket.emit('OK', service.id);
7      });
8      socket.on('follow', function(trail){
9          var service = state.subscribe('follow ' + trail);
10         socket.emit('OK', service.id);
11     });
12     socket.on('config', function(values){
13         for(var key in values){
14             state.setConfig(values[key].name, values[key].value);
15         }
16         socket.emit('OK');
17     })
18     socket.on('coords', function(coords){
19         for(var key in state.subscription){
20             var service = state.subscriptions[key];
21             service.process([coords.lat, coords.lon], function(err, result){
22                 if(err) return socket.emit('error', err);
23                 socket.emit(service.id, result);
24             });
25         }
26     })
27 });
```

Listing 6.36: Socket server main process

In order to use this service with the application, it is necessary to edit the android manifest file. This file presents information about the application to the operating system such as the permissions needed and the services it uses. The lines on listing 6.37 have to be added in order to be able to use the services and the GPS of the device. The services and the permissions are located in different parts of the XML document, however they are put here together for convenience purposes.

```
1  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
2  ...
3  <service android:name="com.sorelcom.services.TrailRecorder">
4      <intent-filter>
5          <action android:name="com.sorelcom.services.TrailRecorder" />
6      </intent-filter>
7  </service>
8  <service android:name="com.sorelcom.services Locator">
9      <intent-filter>
10         <action android:name="com.sorelcom.services Locator" />
11     </intent-filter>
12 </service>
```

Listing 6.37: Android manifest file

The trail recorder service will just pick a point every regular interval of time through the `doWork()` method and will add it to a array, which will be returned to the application as a JSON. This way, once the recording finishes, all the GPX data has been received.

The implementation of this method is shown on listing 6.38.

```

1  @Override
2  protected JSONObject doWork() {
3      Location loc = pointRecorder.getLocation();
4      if(loc == null)
5          return trail;
6      JSONArray coords = new JSONArray();
7      coords.put(loc.getLatitude());
8      coords.put(loc.getLongitude());
9      JSONArray array = trail.getJSONArray( "coordinates");
10     array.put(coords);
11     return trail;
12 }

```

Listing 6.38: Recorder service implementation

The PointRecorder appearing in the recorder service is different from the one on the design specified in section 5.6 on chapter 5. The one used on the service is just a utility class that encapsulates the functionality of the LocationManager class used in android to retrieve the current location of the user.

This point recorder uses the created services as a android context to gain access to the GPS of the device. The code for creating point recorders is shown on listing 6.39. Since the GPS device may be temporarily disabled, the location manager records the last location and sends it to the user as a fall-back if there is no GPS.

```

1  public class PointRecorder{
2      LocationManager manager;
3      Location lastLocation;
4
5      public PointRecorder(Context context){
6          manager = (LocationManager) context
7              .getSystemService(LOCATION_SERVICE);
8      }
9
10     public Location getLocation(){
11         if(locationManager
12             .isProviderEnabled(LocationManager.GPS_PROVIDER)){
13             lastLocation = locationManager
14                 .getLastKnownLocation(LocationManager.GPS_PROVIDER);
15         }
16         return lastLocation;
17     }
18 }

```

Listing 6.39: Recorder service implementation

The last service to implement is the location tracker service. This service has to communicate with the Socket.IO server, for which a client library is needed. A Java implementation of the Socket.IO library has been used for the creation of this functionality. This library can be found in . The service is very similar to the tracker service regarding implementation, however, it uses the doWork() and configuration methods to communicate information to a class implementing the protocol on the client side. The code of this class is shown on listing 6.40.

## 6. IMPLEMENTATION DETAILS

```
1  public class SocketIOClient{
2      SocketIO socket;
3      ConnectionStatus status;
4
5      public SocketIOClient(){
6          socket = new SocketIO("http://apps.morelab.deusto.es/sorelcom");
7          ConnectionStatus status = new ConnectionStatus();
8      }
9      public void start(){
10         socket.connect(status.callback);
11     }
12     public void sendCoordinates(Location loc){
13         JSONObject object = new JSONObject();
14         object.put("lat", loc.getLatitude());
15         object.put("lon", loc.getLongitude());
16         socket.emit("coords", object);
17     }
18     public void addService(String name){
19         status.toggleService(name);
20     }
21 }
```

Listing 6.40: Socket communication client

The client implementation works in an identical way to the server. When the user wants to subscribe to a new service, the socket subscribes adds it to a list of services. The difference is that instead of having a function encoded to a service, the client services will subscribe a callback to the IOCallback object used on the socket. Note that the services mentioned here are different from the android Services.

When a new service is added, a new callback is subscribed to the socket, as shown in listing 6.41.

Most of the code of the document is not shown on this section, however, the design specifies how the control flow works among the different classes on figure 5.21 on page 91.

### Application-to-service communication

The interface needs to provide a way to communicate the user with the services in order to access this functionality. This includes allowing a user to start or stop following a trail, activating or deactivating the location tracking process and starting to record a GPX.

This is all done through the AngularJS services specified for the mobile client (see 5.17 on page 87). The services, programmed in the native language, are accessed on the JavaScript code through the use of the Cordova library. The code for creating a service can be found on 6.42. This code is called when the Angular service handling the Android service is created.

Once the service is started, the application can access a uniform API to set intervals on which the code of the service will be run and to receive results. For this, the application should register for updates on the service. This will cause the application to receive a JSON object, returned by the `doWork()` method on the service, every interval of time specified.

In addition, the service can be stopped by disabling its timer. Once this is done, a JSON object

```

1  Map<String, IOLService> active;
2
3  public ConnectionStatus(){
4      active = new Map<String, IOLService>();
5      final ConnectionStatus st = this;
6      callback = new IOCallback(){
7          @Override
8          public void on(String event, IOAcknowledge ack, Object... args) {
9              if(st.active.contains(event))
10                 active.get(event).process(args);
11          }
12      }
13  }
14  public toggleSubscription(name){
15      if(active.contains(name)){
16          active.get(name).destroy();
17      } else {
18          active.put(name, ServiceFactory.createService(service));
19      }
20  };

```

Listing 6.41: Service subscriptions and server message reading

```

1  var factory =cordova.require(
2      'com.red_folder.phonegap.plugin.backgroundservice.BackgroundService');
3  var trailService = factory.create('com.sorelcom.services.TrailRecorder');

```

Listing 6.42: Android service creation from the application

corresponding to the result of the last interval is returned. This is used on the GPX tracker to obtain the finished route once the user decides to stop recording.

In addition, the service can receive configuration options through another method on the API. This configuration methods are used on the Location tracker to enable and disable different services, that is, configuration is changed when the user activates or deactivates the nearby feature discovery or when the user decided to start or stop tracking a service. Initialization of the trail recording service is shown on listing 6.43.

```

1  trailService.getStatus(function(status){
2      startService(status)
3  });
4
5  function startService(status) {
6      if (status.ServiceRunning) {
7          enableTimer(status);
8      } else {
9          myService.startService(function(data){
10              enableTimer(data)
11          });
12      }
13  }

```

Listing 6.43: Trail recording service starting from the application

Once the service is started, it is possible to enable a timer interval. Functions for starting the trail recorder timer and receiving data from the service are shown on listing 6.44

## 6. IMPLEMENTATION DETAILS

```
1  function enableTimer(status) {
2    if (status.TimerEnabled) {
3      registerForUpdates(status);
4    } else {
5      this.trail = [];
6      myService.enableTimer(recordInterval, function(data){
7        registerForUpdates(data)
8      });
9    }
10 }
11
12 function registerForUpdates(status) {
13   if (!status.RegisteredForUpdates) {
14     myService.registerForUpdates(function(data){
15       if(data.latestResult != null){
16         //Push last coordinates to the trail
17         this.trail.push(data.latestResult);
18       }
19     });
20   }
21 }
```

Listing 6.44: Trail recording timer start and data reception implementation

## 6.6. APPLICATION FUNCTIONALITY

The implementation of the rest of the application is almost identical to that of the web application. The work necessary is just a few modifications on the views, eliminating the dialog communication functionality from the modal controllers and creating the new controllers to communicate with the plugins.

In addition, the functionality to create a geolocated note must be created. This function however can be done using only Phonegap provided functions. The reason for creating background services is just providing the user with a functionality that persists when the phone is blocked. Since posting a note is done in a single point in time, while the application is on screen, there is no need to create a service for this.

Instead, the Phonegap geolocation plugin is used. This plugin accesses the GPS device and provides a single coordinate set representing the current position of the user, perfect for the note posting functionality. The code to obtain this coordinates is shown on listing 6.45.

```
1  navigator.geolocation.getCurrentPosition(function(position){
2    note.addPosition(
3      position.latitude,
4      position.longitude,
5      position.coords.altitude || 0);
6  });
```

Listing 6.45: Location retrieval using Phonegap

Once the location is obtained, the note will just be sent as a GeoJSON to the server through a HTTP POST request to the REST API. This is the only functionality noteworthy on the server (besides the services), the rest is a slight modification of the web application functionality.

## 7. TESTING

---

### 7.1. OVERVIEW

During the project, several tests have been carried out in order to guarantee the quality of the code and minimizing the amount of errors that may appear in the development process. This chapter describes the tests carried out through the project.

The following sections are described in the chapter:

Test plan Different kind of tests and procedures to be carried out on the project.

Special test cases Special cases that require more intensive or just different testing.

### 7.2. TEST PLAN

In order to test the application, different types of tests have been defined. The following test types have been carried out through the project.

Unit tests Units tests are test procedures that verify the functionality at of a specific section of code, such as a class.

Integration tests Tests are carried the correct integration of different kinds of components. These tests have been carried out iteratively, checking the integration of a component every time it is added.

Installation tests Tests to check that the installation of a certain component has been carried out correctly.

Special test cases Test carried out in cases where no other tests can be carried.

#### 7.2.1 Unit tests

The units test carried on the project usually have as a goal the verification of the implemented functionality when confronted with both regular values and extreme values or special cases. The tests are focused on the following aspects:

- Array indexes: Checking if loops and other functions don't try and access values outside of the array bounds. This is important in JavaScript programming because no index out of bounds exception will be thrown, but a undefined value will be returned.

## 7. TESTING

- Undefined and null variables: Specially in condition checking it is important to check how variables behave. This is because in some browser a undefined, a null and a false are completely equivalent while in others no. Besides, some runtimes provide undefined values when others provide null values.
- Type casting: On dynamic typed languages it is important to check cases sensible to the automated type casting done by the runtime. For example, when checking equality using a "==" instead of a "===" operator, a string may be cast to number. These errors are usually hard to find a debug, so it is important to check for them.
- String building: String building is a very common operation in this project, due to the need of creating SPARQL queries. Thus, it is vital to check operations that perform this type of task.
- Iterations, conditions and callbacks: Any project needs to check that the control flow is correct. In event oriented environment which make exhaustive use of callback functions it takes a higher importance due to the rise on the control flow complexity.

The unit test for each component has been carried out as the development of the component was carried out. This is done to detect early errors that may affect other parts of the component.

However, due to the heavy amount of communication carried through the project, the amount of unit test has been relatively small, and integration tests have taken a bigger role.

In order to carry these test units, two tools have been used:

### 7.2.2 Integration tests

Integration tests check that the integration of the different components on the system is correct. These test check that the APIs offered by these components are usable by other parts of the system, but not only this, they also check that the errors on one side do not affect the others.

The integration tests carried can be divided in the following parts:

- Database connector: The database is a central piece of the application, thus it is important to check that the communication of the server with it is correct. These tests include checking that the server can handle database errors, that the data sent to the database is valid and that the responses of the database can be always parsed correctly.
- API: Testing the correctness of the API functions is important since it is used as a central point of communication. As usual this implies checking the validity of data received and the extreme cases. In addition, the connectors on the clients also need testing, in order to validate that they query the correct resources of the database.
- Socket communication: The socket server is prone to error, for there are many operations involved, such as connection establishing and destroying. Authorization, session data storage and communication in itself must be checked, both in the client and in the server.
- 

### 7.2.3 Installation tests

The installation tests aim to verify the installation of one or more components of the project. This means checking the validity of the installation of the database. This is needed because the data storage system includes uncommon functionality, that is, a reasoner, a triple store and a endpoint.



The following tests have been carried out:

- SPARQL endpoint: The endpoint has to accept queries and answer in the desired format in order to pass these tests. In addition, it must accept the different SPARQL protocols to be used on the project.
- Reasoner: The correct functioning of the reasoner, that is, the capability of the system to generate knowledge from the provided triples must be validated.

Some of these tests, such as the SPARQL endpoint can be carried at the start of the project, however, they require hand made testing. This means manually writing SPARQL queries to the database to check it works.

Testing the reasoner, on the other side, cannot be done so early. First it is needed to develop the ontology and create the spatial indexes of the database. Once this is done it is possible to elaborate spatial queries and to insert data to check the functioning of the inference engine.

### **7.3. SPECIAL TEST CASES**

The test plan comprehends different kinds of standard tests that are carried out in any project. Some tests, however, cannot be automated so easily and require manual work to check their validity. The following test have been considered special cases:

#### **7.3.1 Ontology**

When a ontology is not well designed errors may arise during the inference process. A typical example of this is inferring that a resource belongs to two different classes that should be disjoint.

In order to test the validity of the ontology, data has been inserted to the storage system and then the generated triples have been analyzed. Analyzing the triples means just looking at them and determining if the results obtained conform with what was expected on the ontology.

This approach however has a issue. It is very difficulty to check all the cases that may cause problems, thus, common sense and careful analysis are important when carrying out these tests. Still, it can be guaranteed that most of the cases are checked and that all major errors are solved.

#### **7.3.2 Trail recording and locator service**

While it is possible to test the correct functioning of the communication protocol and the validity of server and client side functions, it cannot be certainly said that the services work correctly until they have been tested by a user.

This means that even if the platform can query the server for nearby points of interest and read the response, there is no guarantee that the functionality provided is actually the desired one. In these cases, a certain amount of real data is needed to check that the provided service is the one required.

This is usually done by alpha and beta testing, however, before these phases can begin in the project a small amount of field tests have been carried. This test consist in simply uploading a small amount of test data to the server and using the mobile application on a real environment.

## 7. TESTING

### 7.3.3 User interaction

The interface of, the means of user interaction, the application need some testing too, as well as the logical components. This however does not imply testing for the validity of the interface, but for other features such as ease of use and learning.

In order to do this, external testing and feedback is needed. It is impossible for the interface to be tested by the developers since they are biased, they know exactly what the functionality is, thus they will find it very hard to signal the unintuitive features of the interface.

Due to this, this testing has been delayed to the beta phase of the project.

## **8. USER GUIDE**

---

### **8.1. OVERVIEW**

This chapter shows a step-by-step guide on the usage of the tools developed for the users. Two guides are presented, one corresponding to each final product of the project:

- Web application guide: Guide to the usage of the browser based client.
- Mobile application guide: Guide to the usage of the functions offered exclusively by the mobile application.

### **8.2. WEB APPLICATION GUIDE**

#### **8.2.1 Homepage**



## **9. CONCLUSIONS AND FUTURE LINES OF WORK**

---

### **9.1. OVERVIEW**

This chapter compiles, as a conclusion, the opinions of the work group after the realization of the SORELCOM project, showing their impressions on the work done and taking into account the contributions its development has made both at a personal and professional level.

### **9.2. GOALS ACHIEVED**

After following the development process, it can be stated that all the goals listed in section ?? have been accomplished:

- A location-aware service has been created, which takes advantage of the advantages that the semantic web offers (linked-data, inferences, etc.). This service is offered to the users through a GPS community and a mobile service.
- Several Linked Open Data vocabularies have been used on the definition of the SORELCOM data model, thus allowing the publication of data as Linked Open Data.
- A model for the representation of trails and points of interest has been created. In addition, this model supports point of interest categories, compatible with OpenStreetMaps taxonomies and Geolocated Notes.
- A method for calculating the score of a route has been created, thus giving the possibilities of evaluation. The means of corrections are provided through a trail editor interface.
- The tools for a GPS community have been created. Whether a community is built around them, only time will tell.

### **9.3. CONSIDERATIONS**

### **9.4. FUTURE LINES OF WORK**



## Bibliography

- [1] Atom - A hackable text editor for the 21st century. URL: <https://atom.io/> (visited on 06/21/2014).
- [2] Robert Battle and Dave Kolas. "Enabling the geospatial semantic web with Parliament and GeoSPARQL". In: Semantic Web 3.4 (2012), pages 355–370.
- [3] Robert Battle and Dave Kolas. "Linking geospatial data with GeoSPARQL". In: Semantic Web Journal of Interoperability, Usability, Applicability 24 (2011).
- [4] Sean Bechhofer. "OWL: Web ontology language". In: Encyclopedia of Database Systems. Springer, 2009, pages 2008–2009.
- [5] Tim Berners-Lee, Roy Fielding, and Larry Masinter. "RFC 3986: Uniform resource identifier (uri): Generic syntax". In: The Internet Society (2005).
- [6] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The semantic web". In: Scientific american 284.5 (2001), pages 28–37.
- [7] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data: Principles and state of the art". In: World Wide Web Conference. 2008.
- [8] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data-the story so far". In: International journal on semantic web and information systems 5.3 (2009), pages 1–22.
- [9] Christian Bizer and Andy Seaborne. "D2RQ-treating non-RDF databases as virtual RDF graphs". In: Proceedings of the 3rd international semantic web conference (ISWC2004). Volume 2004. 2004.
- [10] Bower - A package manager for the web. URL: <http://bower.io/> (visited on 06/17/2014).
- [11] Dan Brickley and Ramanathan V Guha. "{RDF vocabulary description language 1.0: RDF schema}". In: (2004).
- [12] Dan Brickley and Libby Miller. "FOAF vocabulary specification 0.98". In: Namespace Document 9 (2012).
- [13] Jeen Broekstra, Michel Klein, Stefan Decker, Dieter Fensel, Frank Van Harmelen, and Ian Horrocks. "Enabling knowledge representation on the web by extending RDF schema". In: Computer networks 39.5 (2002), pages 609–634.
- [14] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. The GeoJSON Format Specification. 2008.
- [15] Balakrishnan Chandrasekaran, John R Josephson, V Richard Benjamins, et al. "What are ontologies, and why do we need them?" In: IEEE Intelligent systems 14.1 (1999), pages 20–26.
- [16] Kristina Chodorow. MongoDB: the definitive guide. " O'Reilly Media, Inc.", 2013.

## 9. BIBLIOGRAPHY

- [17] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. "Serializing sparql query results in json". In: World Wide Web Consortium, Note NOTE-rdf-sparql-json-res-20070618 (2007).
- [18] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. "SPARQL protocol for RDF". In: World Wide Web Consortium (W3C) Recommendation (2008).
- [19] Open Geospatial Consortium et al. "OpenGIS® Implementation Specification for Geographic information-Simple feature access-Part 1: Common architecture". In: OGC document (2011).
- [20] Apache Cordova. About Apache Cordova. 2013.
- [21] Simon Cox, Adrian Cuthbert, Paul Daisey, John Davidson, Sandra Johnson, Edric Keighan, Ron Lake, Marwa Mabrouk, Serge Margoulies, Richard Martell, et al. "OpenGIS® Geography Markup Language (GML) Implementation Specification, version". In: (2002).
- [22] Douglas Crockford. "JSLint: The JavaScript code quality tool". In: URL <http://www.jslint.com> 95 (2010).
- [23] Peter Bacon Darwin and Pawel Kozlowski. AngularJS web application development. Packt Publishing, 2013.
- [24] Martin Dürst and Michel Suignard. Internationalized resource identifiers (IRIs). Technical report. RFC 3987, January, 2005.
- [25] ESRI. What is GIS - Overview. URL: [http://www.esri.com/what-is-gis/overview#overview\\_panel](http://www.esri.com/what-is-gis/overview#overview_panel) (visited on 06/02/2014).
- [26] Ian Fette and Alexey Melnikov. "The WebSocket protocol". In: (2011).
- [27] Roy Fielding. "Representational state transfer". In: Architectural Styles and the Design of Network-based Software Architecture (2000), pages 76–85.
- [28] Dan Foster. "GPX: the GPS exchange format". In: Retrieved on 2 (2007).
- [29] Adam Freeman. "The Anatomy of an AngularJS App". In: Pro AngularJS. Springer, 2014, pages 207–231.
- [30] Jesse James Garrett et al. "Ajax: A new approach to web applications". In: (2005).
- [31] Rohit Ghatol and Yogesh Patel. Beginning PhoneGap: mobile web framework for JavaScript and Html5. Apress, 2012.
- [32] GIS Wiki - GIS. URL: <http://wiki.gis.com/wiki/index.php/GIS> (visited on 06/02/2014).
- [33] Google JavaScript Style Guide. URL: <http://javascript.crockford.com/code.html> (visited on 06/21/2014).
- [34] Brad Green and Shyam Seshadri. AngularJS. O'Reilly Media, Inc., 2013.
- [35] Tom Gruber. What is an Ontology. 1993.
- [36] Grunt: The JavaScript Task Runner. URL: <http://gruntjs.com/> (visited on 06/21/2014).
- [37] Ian Hickson and David Hyatt. "Html5". In: W3C Working Draft WD-html5-20110525, May (2011).
- [38] HTML5 File Upload. URL: <https://github.com/flowjs> (visited on 06/22/2014).
- [39] Dublin Core Metadata Initiative et al. "Dublin core metadata element set, version 1.1". In: (2008).



- [40] Holger Knublauch, Ray W Ferguson, Natalya F Noy, and Mark A Musen. "The Protégé OWL plugin: An open development environment for semantic web applications". In: The Semantic Web--ISWC 2004. Springer, 2004, pages 229–243.
- [41] Holger Knublauch, Matthew Horridge, Mark A Musen, Alan L Rector, Robert Stevens, Nick Drummond, Phillip W Lord, Natalya Fridman Noy, Julian Seidenberg, and Hai Wang. "The Protege OWL Experience." In: OWLED. 2005.
- [42] Leaflet - An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps. URL: <https://leafletjs.com/> (visited on 06/15/2014).
- [43] Reuven M Lerner. "At the forge: Twitter bootstrap". In: Linux Journal 2012.218 (2012), page 6.
- [44] Frank Manola, Eric Miller, Brian McBride, et al. "RDF primer". In: W3C recommendation 10 (2004), pages 1–107.
- [45] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: W3C recommendation 10.2004-03 (2004), page 10.
- [46] Kailashkumar V Natda. "Responsive Web Design". In: Eduvantage 1.1 (2013).
- [47] Natalya Fridman Noy, Ray W Ferguson, and Mark A Musen. "The knowledge model of Protege-2000: Combining interoperability and flexibility". In: Knowledge Engineering and Knowledge Management Methods, Models, and Tools. Springer, 2000, pages 17–32.
- [48] Mark Otto, Jacob Thornton, Chris Rebert, Julian Thilo, et al. "Twitter Bootstrap". In: Dostupné z: <http://twitter.github.io/bootstrap/base-css.html> (2012).
- [49] XML Schema Part. "XML Schema - 2: Datatypes". In: W3C Recommendation 2 (2001).
- [50] Pellet: OWL 2 Reasoner for Java. URL: <http://clarkparsia.com/pellet/> (visited on 06/18/2014).
- [51] Matthew Perry and John Herring. "OGC geosparql, a geographic query language for rdf data". In: OGC Implementation Standard, ref: OGC (2011).
- [52] PostGIS - Spatial and Geographic Object for PostgreSQL. URL: <http://postgis.net/> (visited on 06/18/2014).
- [53] PostgreSQL: The world's most advanced open source database. URL: <http://www.postgresql.org/> (visited on 06/18/2014).
- [54] Protégé. URL: <http://protege.stanford.edu/> (visited on 06/18/2014).
- [55] Eric Prud'Hommeaux, Andy Seaborne, et al. "SPARQL query language for RDF". In: W3C recommendation 15 (2008).
- [56] Guillermo Rauch. Socket. IO-the cross-browser WebSocket for realtime apps. 2013.
- [57] Jason Ronallo. "HTML5 Microdata and Schema. org." In: Code4Lib Journal 16 (2011).
- [58] James Rumbaugh, Ivar Jacobson, and Grady Booch. Unified Modeling Language Reference Manual, The. Pearson Higher Education, 2004.
- [59] Andy Seaborne, Geetha Manjunath, Chris Bizer, John Breslin, Souripriya Das, Ian Davis, Steve Harris, Kingsley Idehen, Olivier Corby, Kjetil Kjernsmo, et al. "SPARQL/Update: A language for updating RDF graphs". In: W3C Member Submission 15 (2008).
- [60] SemWebCentral. Parliament homepage. URL: <http://parliament.semwebcentral.org/> (visited on 06/08/2014).

## 9. BIBLIOGRAPHY

- [61] Evren Sirin and Bijan Parsia. "Pellet: An OWL DL reasoner". In: Proc. of the 2004 Description Logic Workshop (DL 2004). 2004, pages 212–213.
- [62] Sublime Text: The editor you'll fall in love with. URL: <http://www.sublimetext.com/> (visited on 06/21/2014).
- [63] The Linking Open Data project. URL: <http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData> (visited on 06/18/2014).
- [64] The web's scaffolding tool for moders webapps | Yeoman. URL: <http://yeoman.io/> (visited on 06/21/2014).
- [65] Stefan Tilkov and Steve Vinoski. "Node.js: Using JavaScript to Build High-Performance Network Programs." In: IEEE Internet Computing 14.6 (2010).
- [66] Linus Torvalds and Junio Hamano. "Git: Fast version control system". In: URL <http://git-scm.com> (2010).
- [67] v8 Javascript Engine. URL: <https://code.google.com/p/v8/> (visited on 06/14/2014).
- [68] W3C. Ontologies - W3C. URL: <http://www.w3.org/standards/semanticweb/ontology> (visited on 06/01/2014).
- [69] W3C. RDF 1.1 Primer. Feb. 2014. URL: <http://www.w3.org/TR/rdf11-primer/> (visited on 06/01/2014).