



Universitat de Lleida
Escola Politècnica Superior

WIKIDATA LIQUID GALAXY VISUALIZATION

Guillem Barbosa Costa

Sisco Giné De Solà

Màster Enginyeria Informàtica

September 2017

Acknowledgments

I would like to thank Francesc Giné De Solà, my project tutor, for having patience and dedication to guide me through all of this work. He has given me constant support during these months and he helped me so much with going ahead with the project.

Thanks also to the organization of Lleida Liquid Galaxy to be able to develop the project in the Lab and give me all the possible facilities. Especially, very close to the Andreu Ibañez, I have been very pleased and the treatment has been perfect.

I also thank all my colleagues for teaching me so much about these months, my family to be always present, and friends for their encouragement.

Thanks to all to make it possible

Contents

Acknowledgments	1
1. Introduction	2
1.1. Project motivation	3
1.2. Work objectives	4
2. Task planning and project cost	5
3. Technologies	8
3.1. Liquid Galaxy System	8
3.2. KML Language	11
3.3. Wikidata	13
3.3.1. What is Wikidata?	13
3.3.2. The Wikidata repository	14
3.3.3. Working with Wikidata	17
3.4. SPARQL Language	18
4. Design and software development	21
4.1. Software languages and tools	21
4.1.1. What is Python?	21
4.1.2. Django	23
4.1.3. Material Design	24
4.1.4. Atom	25
4.2. Project idea	27
4.3. Project structure	29
5. Web Application Interface Design	36
5.1. Index page	37

5.2.	Most Populated Cities page	39
5.3.	Premier League Stadiums page	40
5.4.	Longest Rivers page	42
5.5.	Spanish Airports page	43
5.6.	Summer Olympic Games page	44
5.7.	Try the Demo page	46
5.8.	Use case example	48
6.	Conclusions and future work	51
6.1.	Future work	53

Chapter 1

Introduction

During this summer I have been participating in *Google Summer of Code 2017* [1], working in the Lleida *Liquid Galaxy* [2] Lab. A few months before, thinking about what project to do I came up with the option of working in this organization and learning about *Liquid Galaxy* technology. I had some project proposals and finally I decided to work with the *Wikimedia foundation* [3] projects and show the data through the *Liquid Galaxy* that offers a good visualization so any user of the application would appreciate. The project idea is simple, and what is intended is that the application is very usable and that the interaction is as simple as possible.

The *Wikidata Liquid Galaxy Visualization* (WDLGV) project has the purpose of showing some data from the *Wikidata* in the *Liquid Galaxy*. This information is obtained from *Wikidata* [4], who acts as central storage for the structured data in Wikimedia projects. The visualization is represented in *Liquid Galaxy* system, and the information appears in a bubble and different shapes. There is also the possibility to generate a tour that shows the place and starts an orbit around it. The user has at his disposal five options to select and the user experience is funny, interesting and easy.

This report contains a small explanation of the different points that make up the development of this project. First, objectives must be established, which will guide in order not to deviate from what you want to achieve. The usability of the application is a very important feature that should not be overlooked, and an application with good usability has been developed. Before beginning with the design, tasks must be planned and estimating the hours and the budget, and once this has happened, it goes to design and implementation. It explains the design of the application to have an idea (architecture, code, classes, ...), and on the screens of the interface detail the most important aspects. Finally, talk about use cases examples to understand how the

application works. The outlook for the future and the conclusions are the sections that close this document on the WDLGV application.

1.1. Project motivation

When I was offered the option to start a project with **Liquid Galaxy** technology and also have the option of being able to participate in **Google Summer of Code**, I didn't doubt to get me started on this project. It was also interesting to learn new forms of programming, take responsibility and have new knowledge. The main topic of the project was defined with the days, first began with understanding how **Liquid Galaxy** works, and then I thought what I wanted to do related to this technology with an application that would have a good base and different functionalities. The option to work directly with data from Wikipedia, which is worldwide known, I liked it, so it had a wide choice of data and very complete. It was Andreu Ibañez who spoke to me about the option to develop this project, so he had contacts in the Wikipedia foundation and there was the possibility to work with them. After speaking at some meetings, I finally decided to start my project and work for a few months.

The project approach should be clear, because in order to participate in **Google Summer of Code**, the proposal had to be submitted for acceptance. This aspect motivates me to start with the project and define well the objectives and functionalities. Working for **Google** for a few months was a great opportunity to learn and start a new experience very well valued.

The project consists of developing a web server, a web application and it must be able to connect with the **Liquid Galaxy** system and send **KML** [5] files so that they can be showed on the multiple screens that are part of the **Liquid Galaxy**. This integration and relationship between the different parts is the programming body of the project that in the end must be well developed and available for use by any person. The project, once completed will be available in Lleida **Liquid Galaxy** Lab and in all visits, exhibitions, demo days, ... the application will be visible to everyone who wants to interact with it.

1.2. Work objectives

It is important to establish the objectives of the project, these are the basis for making decisions during the development and to avoid any deviation over the main purpose. From the beginning it must be clear what it is that is wanted and how to do it. The exposed objectives help to understand the application and what is needed to achieve everything that was proposed at the beginning of the project.

According to the project motivation, ***the main objective of this project is to develop a framework to show the Wikipedia data in a cluster display wall, such as the Liquid Galaxy is.*** This objective can be divided in the following work objectives:

- Know and understand the different technologies related with the project: ***Liquid Galaxy, Material Design*** [6], ***Django*** [7], ***SPARQL*** [8] language, ...
- Develop a web application that makes queries to the ***Wikidata Query*** database.
- Implement a server that runs the application correctly.
- Define the different use cases that the application will offer.
- Develop internal code that generates a ***KML*** file to be sent to ***Liquid Galaxy*** system.
- Make an SSH connections between the web application and the ***Liquid Galaxy*** for sending files.
- Implement an application easy to use, usable by any user and with a comprehensible language.
- Design a responsive application to be able to adapt to any type of device.

Chapter 2

Task planning and project cost

The objective of this planning is to provide a framework of work for a project of these dimensions that allows reasonable estimates of resources, cost and temporary planning. These estimates are made in time limited to the beginning of the project and are updated as the project advances. In addition, the estimates must define the scenarios of the best case and the worst case so that the results of the project can be limited. The goal of planning is achieved through a process of discovery of information that leads to reasonable estimates.

Planning is an important step in any development, compels to collect and order the tasks necessary to carry out the project and allows a global vision of the whole. It's a software level production planning, so all tasks are for programmers and analysts.

This project has been developed during Summer Google of Code period which has a duration of 3 months, but it must be said that I have been working on this project for a few more weeks. In total, the duration of the project has been about 4 months, working on it 4 hours at the beginning, and 8 during the months of GSoC17.

Before starting with the project, I planned a little the tasks to do to be clear where I wanted to go. It is important to define well the tasks to follow step by step with the planning and to go ahead at a good pace with the development of the project. We must also consider other aspects with the available resources, the quality of these, possible risks, ... We must have everything controlled before starting the project. In this way, we make sure that the project will possibly have a good future and satisfies the goals marked in the beginning.

Below are the planning table with all the tasks that have been defined and that have been fulfilled to advance satisfactorily with the project. The tasks can be divided into 3 phases. The first one to organize and raise the project, the second contains the whole

body of the project, implementation, development and design, and the last one is the review and presentation phase.

PROJECT TASKS	MAY 2017				JUNE 2017				JULY 2017				AUGUST 2017				SEPTEMBER 2017	
	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2
Install the working environment (Liquid Galaxy, libraries, drivers, ...)																		
Research on Liquid Galaxy technology.																		
Research on Wikimedia API and Wikidata Query Service.																		
Plan project tasks.																		
Analyze and decide the use cases.																		
Start with the Django application development.																		
Start with obtaining data (Wikidata SPARQL Queries).																		
Application implementation (classes and functions).																		
Generate KML files for data visualization.																		
Design and development of the interface (Material Design).																		
Apply the SSH connection between the application and the Liquid Galaxy.																		
Check for any errors or malfunctions.																		
New application Demos.																		
Project documentation.																		

Figure 1: Tasks planning

PROJECT TASKS	Hours
Install the working environment (Liquid Galaxy, libraries, drivers, ...)	18
Research on Liquid Galaxy technology.	26
Research on Wikimedia API and Wikidata Query Service.	36
Plan project tasks.	10
Analyze and decide the use cases.	12
Start with the Django application development.	45
Start with obtaining data (Wikidata SPARQL Queries).	54
Application implementation (classes and functions).	180
Generate KML files for data visualization.	104
Design and development of the interface (Material Design).	110
Apply the SSH connection between the application and the Liquid Galaxy.	42
Check for any errors or malfunctions.	18
New application Demos.	28
Project documentation.	50
TOTAL PROJECT HOURS	733

Figure 2: Estimated hours per task

This project has been done with work and effort and the planned tasks have been completed during the planning time. This is why the cost of the project has been planned considering that it has been developed by a single worker, in this case myself as a programmer. During the month of May, which was the month of initiation in the laboratory, I have been working in the mornings about 5 hours a day. Then, at the end of this month, when Google Summer of Code started, I have been working in the morning and in the afternoon, in total about 8 hours each day. In the Figure 2 it has estimated the hours that has been dedicated to each task to be able to calculate the total hours and get the cost of the entire development of the project. The price per hour of a junior programmer with a minimum experience is 12 euros per hour, so the total cost is (the documentation doesn't include it):

$$733 \text{ total hours} - 50 \text{ documentation hours} = 683 \text{ hours} * 40 \text{ €/h} = \mathbf{27320 \text{ euros}}$$

It's important to keep in mind that there are other costs, for example those associated with the Liquid Galaxy system, or the hardware used for the project development. The software used is free and has no cost. At least, a *Liquid Galaxy* system must have 3 basic computers to be able to launch Google Earth. The estimated cost for each computer is 350 euros, in total a minimum Liquid Galaxy system has a cost of $350 * 3 = \mathbf{1050 \text{ euros}}$.

Chapter 3

Technologies

3.1. Liquid Galaxy System

In this project, *Liquid Galaxy* works as a key tool that will be connected to the web application to show all the data with good visualization in the screens.

As an important part of the project, it needs a proper explanation about how it works and how to understand their environment.

Liquid Galaxy is an open source project founded by Google. Created in 2008 by Google employee Jason Holt. It's a general data visualization tool for operations, marketing, and research. *Liquid Galaxy* is a cluster display wall originally built to run *Google Earth* [9] to create an immersive experience for the user. Liquid Galaxy gives the ability to fly around *Google Earth*, view panoramic video and photos, develop interactive tours, and graphically display GIS data in an immersive, panoramic environment with its 6-axis controller, allowing you instantly to zoom in, out, and turn around with completely fluid motion.



Figure 3: Lleida Liquid Galaxy

As you can see in the Figure 3, **Liquid Galaxy** combines high-definition displays, multiple computers and the 3Dconnexion **Space Navigator** [10] control to create an immersive global navigation experience. The view of the user, which spans across all the screens, curves around to fill the user's peripheral vision with the globe, countries, streets—essentially whatever the user is looking at. Each node runs the **Google Earth** application and the user only interacts with the master node, which is in the center. Each movement of the mouse makes the master node launches a synchronization protocols that consists of the following steps:

- The master node captures the coordinates of the position in **Google Earth** indicated by the user. These coordinates are codified in a UDP packet, named ViewSync, which contains the following information from the view in the application: latitude, longitude, altitude, heading, tilt, roll and planet name.
- When the master's view is moved, it sends ViewSync packets to broadcast with the objective of sending it to all the nodes in the cluster network.
- Every slave node has a configuration file, which holds an offset from the original master's view. When the slaves receive a ViewSync packet from the master, they automatically calculate and adjust their relative view by adding the local offset.
- Every node accesses Internet with its own coordinates to download the required data (maps, imagery, 3D layer, ...) independently from the other nodes.

The **Liquid Galaxy** system presented in this section was built specifically to give service to run **Google Earth**. However, the immersive visualization environment that **Liquid Galaxy** provides opens this kind of system for use in a wide range of applications that can benefit from an immersive visualization environment. Some

examples of applications that can be run in this system are WebGL with Aquarium, video streaming, videogames like Quake 3 Arena, videoconferences, ...

Any user who wants to interact with **Liquid Galaxy** just be clear that their interaction will be through the **Space Navigator**, that allows to fly and see whatever they want. The navigation is done from the master node, and this one is who is synchronizing with the rest of nodes to provide a pleasant experience and show a panoramic visualization in all the screens.



Figure 4: Space Navigator

In Figure 4, we can see the **Space Navigator** which is specifically designed to manipulate digital content or camera positions. Simply push, pull, twist or tilt the 3Dconnexion controller cap to intuitively pan, zoom and rotate.

Getting deeper into the **Liquid Galaxy** filesystem, there are two configuration files that are so important to understand how it works. The first one is **kmls.txt** that contains the KML file path to download it and then show all the representation in **Liquid Galaxy**. The other one is **query.txt**, it's saved in a temporal folder, and once used is removed from the system. In this file you specify where you want to go, with the coordinates and the type of trip you want to perform. This information will be so useful when we start to explain the design and development of the project.

3.2. KML Language

Keyhole Markup Language (KML) is an XML-based markup language designed to annotate and overlay visualizations on various two-dimensional, Web-based online maps or three-dimensional Earth browsers (such as **Google Earth**). **KML** was developed for be used with **Google Earth**, which was originally named Keyhole Earth Viewer. **Google Earth** was the first program able to view and graphically edit **KML** files. A **KML** file includes specifications for various features for display within **Google Earth**, Maps and Mobile, and other three-dimensional Earth or geo browser programs.

KML's feature set includes placemarks, 3D models, text descriptions, images, polygons, and so forth. Each location has an associated longitude and latitude and view-specific data such as heading, altitude and tilt may be provided to define a so-called "camera view" for geospatial data. **KML** shares some of its grammar with the geography markup language, or GML, an Open XML markup language defined to express geographical data and features. **KML** documents are often distributed in the form of KMZ files, which are nothing more than a zipped **KML** document inside a file with kmz extension. A KMZ file usually contains a single **KML** document, invariably named "doc.kml" along with images for overlays and icons it may reference internally.

An example **KML** document is:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
  <name>Document.kml</name>
  <open>1</open>
  <Style id="exampleStyleDocument">
    <LabelStyle>
      <color>ff0000cc</color>
    </LabelStyle>
  </Style>
  <Placemark>
    <name>Document Feature 1</name>
    <styleUrl>#exampleStyleDocument</styleUrl>
    <Point>
      <coordinates>-122.371,37.816,0</coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>Document Feature 2</name>
    <styleUrl>#exampleStyleDocument</styleUrl>
    <Point>
      <coordinates>-122.370,37.817,0</coordinates>
    </Point>
  </Placemark>
</Document>
</kml>
```

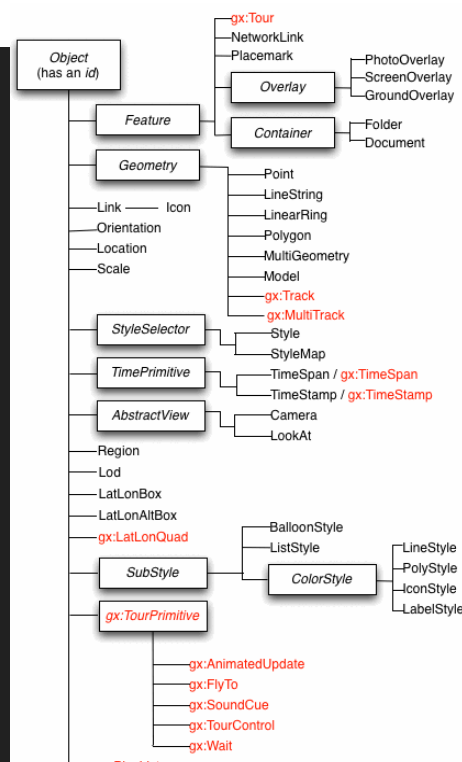


Figure 5: KML file example / KML elements

The Figure 5 example, shows only the Placemark tag with the coordinates point (longitude, latitude, altitude). The Style defines colors, icons, sizes, ... and it's necessary to define with <styleUrl> when the Placemark it is created. There are other features, geometries, abstract views, ... All the **KML** elements allow to create files with a great variety of options that once thrown on **Google Earth** in the **Liquid Galaxy** generate a spectacular visualization and experience to the user.

To generate the **KML** files has been used the **pyKML** [11] library. **pyKML** is a **Python** package for creating, parsing, manipulating, and validating **KML**, a language for encoding and annotating geographic data. **pyKML** adds additional functionality specific to the **KML** language. **pyKML** is open source and packaged releases can be found on the Python Package Index (PyPI).

3.3. Wikidata

3.3.1. What is Wikidata?

Wikidata is a free, collaborative, multilingual, secondary database, collecting structured data to provide support for Wikipedia, Wikimedia Commons, the other wikis of the Wikimedia movement, and to anyone in the world. The content of **Wikidata** is available under a free license, exported using standard formats, and can be interlinked to other open data sets on the linked data web.

Wikidata is a central storage repository that can be accessed by others, operated by the Wikimedia Foundation. Content loaded dynamically from **Wikidata** does not need to be maintained in each individual wiki project, and provides storage for media files and access to those files for all Wikimedia projects, and which are also freely available for reuse.

- **Free.** The data in **Wikidata** is published allowing the reuse of the data in many different scenarios. You can copy, modify, distribute and perform the data, even for commercial purposes, without asking for permission.
- **Collaborative.** Data is entered and maintained by **Wikidata** editors, who decide on the rules of content creation and management. Automated bots also enter data into **Wikidata**.
- **Multilingual.** Editing, consuming, browsing, and reusing the data is fully multilingual. Data entered in any language is immediately available in all other languages. Editing in any language is possible and encouraged.
- **A secondary database.** **Wikidata** records not just statements, but also their sources, and connections to other databases. This reflects the diversity of knowledge available and supports the notion of verifiability.
- **Collecting structured data.** Imposing a high degree of structured organization allows for easy reuse of data by Wikimedia projects and third parties, and enables computers to process and understand it.
- **Support for Wikimedia wikis.** **Wikidata** assists Wikipedia with more easily maintainable information boxes and links to other languages, thus reducing

editing workload while improving quality. Updates in one language are made available to all other languages.

- **Anyone in the world.** Anyone can use *Wikidata* for any number of different ways by using its application programming interface.

3.3.2. The Wikidata repository

The *Wikidata* repository consists mainly of items, each one having a label and a little description. Items are uniquely identified by a Q followed by a number, such as Lleida (Q15090).

Statements describe detailed characteristics of an item and consist of a property and a value. Properties in *Wikidata* have a P followed by a number, such as with instance of (P31) or image (P18).

There are many possibilities of items, and each one has their own properties. Usually the “instance of” is present in all items to indicate a relationship with another similar item. The other properties will depend on the type of item, according to which we refer we have an information or another. For example, for a person item the information will be as place of birth, achievements, image, ... for a city or building, the coordinates (longitude, latitude) will be the most important, and also there are more proprieties as country, population, ...

All this information can be displayed in any language, even if the data originated in a different language. The information shown was always the most recent and therefore any query that is done will return the most current data.

ITEM	PROPERTY	VALUE
Q15090	P17	Q29
Lleida	country	Spain
Q15090	P1376	Q12727
Lleida	capital of	Segrià

Table 1: Wikidata statement group example

The Table 1 shows an example of how the item is related to its property and value. The city of Lleida, for example, has the properties “country” and “capital of” and each of the properties has an associated value. In this case, the value has a code which represents another item, but there is also the possibility that the value is a numerical or simple text that doesn’t have any associated identifier. Property identifier always starts with P, and item or value identifier starts with Q.

Lleida (Q15090) **LABEL** (item identifier)

city in Catalonia, Spain **description**
 Lérida | Lleida, Spain | Lérida, Spain | Lerida

[In more languages](#) **Multilingual**

Statements STATEMENT GROUP

instance of **PROPERTY** **municipality of Spain** **VALUE** [edit](#)
 0 references [+ add reference](#)

image **Lleida - La Seu Vella (des de Cappont).jpg** [edit](#)
 media legend La Seu Vella cathedral (English)
 La Seu Vella katedrális (Hungarian)
 0 references [+ add reference](#)

Figure 6: Wikidata repository. Lleida (Q15090)

In the Figure 6 we can see how the repository is distributed. To the top there are the item, its identifier, the description and the multilingual option to deploy. The statement group contains some information about the item. The first one is present in all because it refers to relation items, for example Lleida instance of a city (Q515), or Leo Messi instance of human (Q5). Then, the other information it depends on the amount of data, and each property will have a statement group. There is the option to be able to edit any data to improve the tool, in the right band of the page there are the “edit” option to do this.

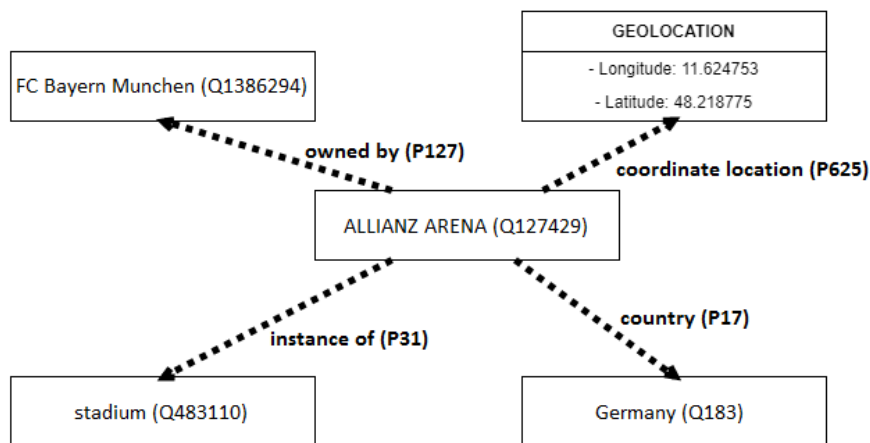


Figure 7: Allianz Arena. Items and properties interconnected.

The Figure 7 shows some properties of Allianz Arena stadium (Q127429). For example, the value of “coordinate location” isn’t an item, and the value is geolocation coordinates that mark the position where the stadium is in the world. On the other hand, the other properties its value is another item with the identifier. The property “country” can have different values, and each of the possible countries has its own identifier that makes it an item. As it has been said before, the main item is always instantiated with “instance of” property.

3.3.3. Working with Wikidata

There are different ways to access **Wikidata** using built-in tools, external tools, or programming interfaces. **Wikidata Query Service** [12] and Reasonator are some of the popular tools to search for and examine **Wikidata** items. The tools page has an extensive list of interesting projects to explore.

Wikidata Query Service works with **SPARQL** language, and it's the tool to make queries toward **Wikidata** and obtain the data. The interface is simple, and it has some facilities that helps the user. Figure 8 shows the **Wikidata Query Service** interface. To the right, the query is written in **SPARQL** language, and there is the possibility to help with query help that allows to filter and show the items and properties that you want. It's so useful because in this way it's easier to know and discover which identifier needs in each case, item and property. Once the query is ready, just press the play button to run the query and then the query result will appear on the bottom. The result data is returned as a double dictionary, first there are the key which identifies the label, and then it's necessary to select the value (result ["population"]["value"]). To handle the result, it is simple and at the same time easy to save because the data are ordered very clearly.

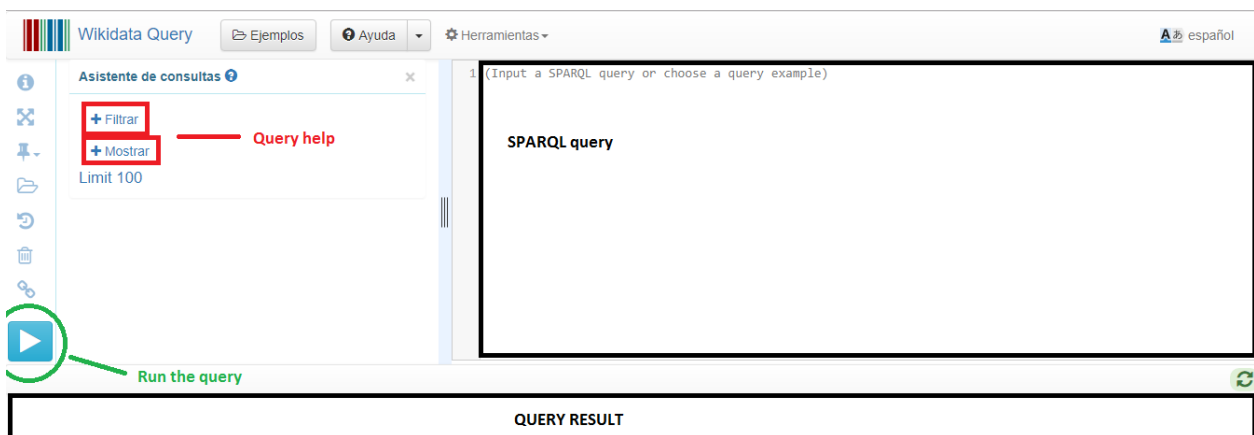


Figure 8: Wikidata Query Service

3.4. SPARQL Language

SPARQL (SPARQL Protocol and RDF Query Language) is an RDF query language, that is a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web. **SPARQL** allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

SPARQL allows users to write queries against what can loosely be called "key-value" data or, more specifically, data that follows the RDF specification of the W3C. The entire database is thus a set of "subject-predicate-object" triples. RDF data can also be considered in SQL relational database terms as a table with three columns – the subject column, the predicate column, and the object column. Unlike relational databases, the object column is heterogeneous: the per-cell data type is usually implied (or specified in the ontology) by the predicate value. **SPARQL** thus provides a full set of analytic query operations such as JOIN, SORT, AGGREGATE for data whose schema is intrinsically part of the data rather than requiring a separate schema definition.

SPARQL has four types of queries. It can be used to:

1. **ASK** whether there is at least one match of the query pattern in the RDF graph data.
2. **SELECT** all or some of those matches in tabular form (including aggregation, sampling and pagination through OFFSET and LIMIT).
3. **CONSTRUCT** an RDF graph by substituting the variables in those matches in a set of triple templates.
4. **DESCRIBE** the matches found by constructing a relevant RDF graph.

```

sparql.setQuery("""SELECT ?city ?cityLabel (SAMPLE(?population) AS ?population) (SAMPLE(?area) AS ?area)
(SAMPLE(?coord) AS ?coord) ?countryLabel (SAMPLE(?image)
AS ?image) (SAMPLE(?elevation) AS ?elevation)
WHERE
{
  ?city wdt:P31/wdt:P279* wd:Q515 .
  ?city wdt:P1082 ?population .
  ?city wdt:P2046 ?area .
  ?city wdt:P625 ?coord .
  ?city wdt:P17 ?country .
  ?city wdt:P18 ?image .
  ?city wdt:P2044 ?elevation .

  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "en" .
  }
}
GROUP BY ?city ?cityLabel ?countryLabel
ORDER BY DESC(?population)
LIMIT """+str(NUM_CITIES))

```

Figure 9: SPARQL Query most populated cities example

Figure 9 shows the **SPARQL** query to obtain the top 10 most populated cities in the world. The SELECT clause contains all the labels that will be in the response, but before it is necessary to get this labels with its property. For example, first of all, city label is an instance of/subclass of city (Q515), and the label city has some properties like population (P1082), area (P2046), coordinates (P625), ... Also, is indicated the language in which we work, English in our case. This query is ordered by population number, and the limit of response is 10 so we want only the top 10 cities.

For each use case available in the application, a new query is made in **Wikidata Query Service**, and for each case, logically, the query change and different properties and items are filtered to get the values out of the result.

In the same code, once the query is defined, it is executed internally and the result obtained is treated also in the same code. This step takes some time, and being a backend process it's positive to inform to the user during the query, so in this way the user knows what is happening.

city	cityLabel	population	area	coord	countryLabel	image	elevation
wd:Q11725	Chongqing	30165500	82403	Point(106.506944444 29.55)	People's Republic of China	commons:Chongqing montage.png	237
wd:Q8660	Karachi	24300000	352700000	Point(67.01 24.86)	Pakistan	commons:Karachi Montage 02.PNG	8
wd:Q8686	Shanghai	24152700	6341	Point(121.466666666 31.166666666)	People's Republic of China	commons:Shanghai montage.png	4
wd:Q956	Beijing	21705000	16410.54	Point(116.391388888 39.905)	People's Republic of China	commons:Beijing montage.png	43.5
wd:Q8673	Lagos	21324000	1171280000	Point(3.4 6.45)	Nigeria	commons:Lagos Island.jpg	34
wd:Q1353	Delhi	16314838	1484	Point(77.216666666 28.666666666)	India	commons:Lotus Temple in New Delhi 03-2016.jpg	200
wd:Q11736	Tianjin	15469500	11920	Point(117.205555555 39.146666666)	People's Republic of China	commons:Tianjin montage.jpg	5
wd:Q406	Istanbul	14657434	5343	Point(28.960277777 41.01)	Turkey	commons:Istanbul collage 5j.jpg	100
wd:Q1156	Mumbai	12442373	603	Point(72.833333333 18.966666666)	India	commons:Mumbai 03-2016 10 skyline of Lotus Colony.jpg	14
wd:Q649	Moscow	12380664	2562	Point(37.617777777 55.755833333)	Russia	commons:MSK Collage 2015.png	156

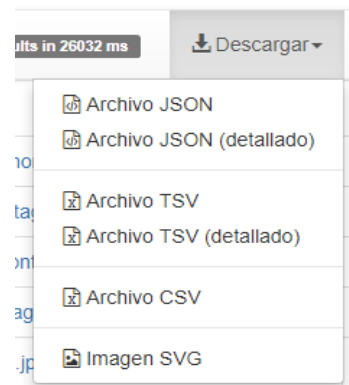
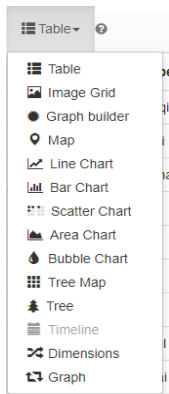


Figure 10: Wikidata Query result

In the Figure 10 we can see how the results are represented by a table, each of the rows are the cities, and each of these has its own values of the items. With this type of representation is seen clearly the results, but there are some other options as image grid, graph, map, tree, ... There is also the option of being able to download the results in different formats like JSON, TSV, CSV, but in my case, I don't download them as the query is done in the same code, and the result can already be analyzed with a simple loop and save each item in a variable.

Chapter 4

Design and software development

The whole project has been developed in python language, all the code is distributed in different classes and functions. The Web Application is in *Django*, and the interface design follows *Material Design*. The user interaction is simple, and has been include the *Google Assistant* [13] interaction (has been configured in the Lleida *Liquid Galaxy* Lab to test with app demos).

4.1. Software languages and tools

4.1.1. What is Python?

Python [14] is a general-purpose programming language created in the late 1980s, and named after Monty Python, that's used by thousands of



people to do a lot of things. Its syntax allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. It's small, very closely resembles the English language, and has hundreds of existing third-party libraries.

There are 3 characteristics of this language that give a very good opinion and make it a very used language within the programming.

- **Readability:** *Python* very closely resembles the English language, using words like ‘not’ and ‘in’ to make it to where you can very often read a program, or script, aloud to someone else and not feel like you’re speaking some arcane language. This is also helped by *Python*’s very strict punctuation rules which means you don’t have curly braces { } all over your code. Also, *Python* has a set of rules, known as PEP 8, that tell every *Python* developer how to format their code. This means you always know where to put new lines and, more importantly, that pretty much every other *Python* script you pick up, whether it was written by a novice or a seasoned professional, will look very similar and be just as easy to read.
- **Libraries:** *Python* has been around for over 20 years, so a lot of code written in *Python* has built up over the decades and, being an open source programming language, a lot of this has been released for others to use. Almost all of it is collected on <https://pypi.python.org>, pronounced “pie-pee-eye” or, more commonly called “the CheeseShop”. You can install this software on your system to be used by your own projects. For example, if you want to use *Python* to build scripts with command line arguments, you’d install the “click” library and then import it into your scripts and use it. There are libraries for pretty much any use case you can come up with, from image manipulation, to scientific calculations, to server automation.
- **Community:** *Python* has user groups everywhere, usually called PUGs, and does major conferences on every continent other than Antarctica. PyCon NA 2013 also started a trend of offering “Young Coder” workshops, where attendees taught *Python* to kids between 9 and 16 years of age for a day, getting them familiar with the language and, ultimately, helping them hack and mod some games on the Raspberry Pis they were given. Being part of a such a positive community does a lot to keep you motivated.

4.1.2. Django

Django is a high-level *Python* Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source. Some *Django* strengths are:



- **Fast:** *Django* was designed to help developers take applications from concept to completion as quickly as possible.
- **Loaded:** *Django* includes dozens of extras you can use to handle common Web development tasks. *Django* takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks.
- **Secure:** *Django* takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.
- **Scalable:** Some of the busiest sites on the planet use *Django's* ability to quickly and flexibly scale to meet the heaviest traffic demands.
- **Versatile:** Companies, organizations and governments have used *Django* to build all sorts of things — from content management systems to social networks to scientific computing platforms.

4.1.3. Material Design

Created and designed by Google, **Material Design** is a design language that combines the classic principles of successful design along with innovation and technology. Google's goal is to develop a system of design that allows for a unified user experience across all their products on any platform.



As of 2015, most of Google's mobile applications for Android had applied the new design language, including Gmail, YouTube, Google Drive, Google Docs, Sheets and Slides, Google Maps, Inbox, Google+, all of the Google Play-branded applications, and to a smaller extent the Chrome browser and Google Keep. The desktop web-interfaces of Google Drive, Docs, Sheets, Slides and Inbox have incorporated it as well. More recently, it has started to appear in Chrome OS, such as in the system settings, file manager, and calculator apps.

The canonical implementation of **Material Design** for web application user interfaces is called Polymer.^[8] It consists of the Polymer library, a shim that provides a Web Components API for browsers that do not implement the standard natively, and an elements catalog, including the "paper elements collection" that features visual elements of the **Material Design**.

The **Material Design** guidelines contain evolving visual, interactive, and motion guidance that can be customized to your app and website.

Material Components provide a reliable development environment for apps and websites across Android, iOS, and the web. Components are updated as the **Material Design** system evolves, ensuring consistent pixel-perfect implementation and adherence to Google's front-end development standards, such as internationalization and accessibility support. Material has certain immutable characteristics and inherent behaviors; solid, occupies unique points in space, impenetrable, mutable shape, changes in size only along its plane, unbendable, can join to other material, can separate, split and heal, can be created or destroyed,

moves along any axis, ... Understanding these qualities of material will help you manipulate material in a way that's consistent with the vision of material design.

In this project **MUI** (Material User Interface) is used for the design of the interface, which is associated with **Material Design**. **MUI** is a lightweight CSS framework that follows Google's **Material Design** guidelines. The **MUI** package includes all the necessary code to use **MUI** components on the web and over email.

4.1.4. Atom

The platform for the project development has been **Atom** [13]. **Atom** is a text editor that's

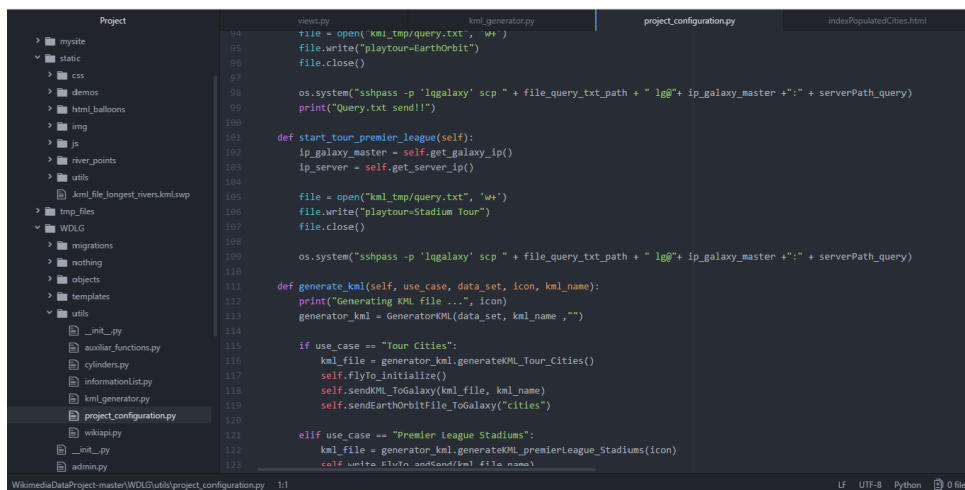


modern, approachable, yet hackable to the core, a tool you can customize to do anything but also use productively without ever touching a config file. Here there are some features of the used tool:

- **Cross-platform editing:** **Atom** works across operating systems. You can use it on OS X, Windows, or Linux.
- **Built-in package manager:** Search for and install new packages or start creating your own, all from within **Atom**.
- **Smart autocompletion:** **Atom** helps you write code faster with a smart, flexible autocomplete.
- **File system browser:** Easily browse and open a single file, a whole project, or multiple projects in one window.
- **Multiple panes:** Split your **Atom** interface into multiple panes to compare and edit code across files.
- **Find and replace:** Find, preview, and replace text as you type in a file or across all your projects.

Other features to highlight are the following:

- **Packages:** There is the possibility to choose from thousands of open source packages that add new features and functionality to **Atom**, or build a package from scratch and publish it for everyone else to use.
- **Themes:** **Atom** comes pre-installed with four UI and eight syntax themes in both dark and light colors. You can also install themes created by the **Atom** community or create your own.
- **Customization:** It's easy to customize and style **Atom**. It's possible to tweak the look and feel of your UI with CSS/Less and add major features with HTML and JavaScript.
- **Under the hood:** **Atom** is a desktop application built with HTML, JavaScript, CSS, and Node.js integration. It runs on Electron, a framework for building cross platform apps using web technologies.
- **Open source:** **Atom** is open source. With a good community it can be improved.



```
Project
├── mysite
├── static
├── css
├── demos
├── html_balloons
├── img
├── js
├── river_points
├── utils
├── kml_file_longest_rivers.kmlswp
├── tmp_files
├── WDLG
├── migrations
├── nothing
├── objects
├── templates
├── utils
├── _init_.py
├── auxiliary_functions.py
├── cylinders.py
├── informationList.py
├── kml_generator.py
├── project_configuration.py
├── wikimap.py
├── _init_.py
└── admin.py

views.py
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123

kml_generator.py
file = open(kml_tmp_query.txt, 'w+')
file.write("playtour-EarthOrbit")
file.close()

os.system("sshpass -p 'lqgalaxy' scp " + file_query_txt_path + " lg@" + ip_galaxy_master + ":" + serverPath_query)
print("Query.txt send!!")

def start_tour_premier_league(self):
    ip_galaxy_master = self.get_galaxy_ip()
    ip_server = self.get_server_ip()

    file = open("kml_tmp/query.txt", 'w+')
    file.write("playtour-Stadium Tour")
    file.close()

    os.system("sshpass -p 'lqgalaxy' scp " + file_query_txt_path + " lg@" + ip_galaxy_master + ":" + serverPath_query)

def generate_kml(self, use_case, data_set, icon, kml_name):
    print("Generating KML file ...", icon)
    generator_kml = GeneratorKML(data_set, kml_name, "")

    if use_case == "Tour Cities":
        kml_file = generator_kml.generateKML_Tour_Cities()
        self.flyto_initialize()
        self.sendKML_ToGalaxy(kml_file, kml_name)
        self.sendEarthOrbitFile_ToGalaxy("cities")

    elif use_case == "Premier League Stadiums":
        kml_file = generator_kml.generateKML_premierLeague_Stadiums(icon)
        self.writeToFileAndSendToGalaxy(kml_file, name)
```

Figure 11: Atom interface

4.2. Project idea

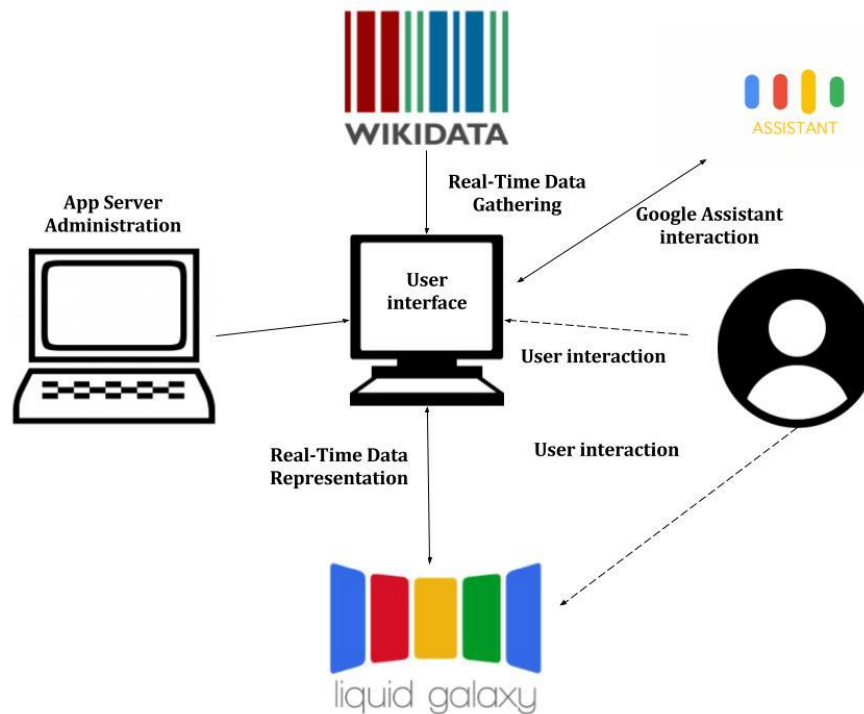


Figure 12: Project idea design

Figure 12 graphically and visually represents the distribution of the project to give us an idea of its structure. The project idea is to develop a web application that collects data from **Wikidata** through real-time queries. It's possible to obtain any kind of information, so **Wikidata** storage a large amount of data that can be consulted at any time. Once this data is well analyzed and treated, the next step is generating a KML file, and then sends this file to the **Liquid Galaxy**. Finally, the KML file is received by the Liquid Galaxy and the user can see all the data in a more visual way through the Liquid Galaxy system screens. The data representation is varied like as trips, tours, flights, placemarks, cylinders, balloons, ...

As it has already been said, the **Liquid Galaxy** will launch the **Google Earth** application and on this will be shown the **KML** file that has been sent. **Google Assistant** interaction is also included in the diagram, but it's only configured in

Lleida **Liquid Galaxy** LAB so there is available a **Google Home** to try the voice recognition and interact with the application.

At all times the server is running to launch the application, and the web application interface is available in any type of format device; tablet, mobile, computer, ... The user has two ways of interacting with the **Liquid Galaxy**. When the application is launched, the most recommended is to use the interface that the application offers, so it has its functionalities and will show the information to **Liquid Galaxy**. The other option is to use the space navigator, in this way the interaction is directly with the **Liquid Galaxy** and the user can travel in any direction, position and altitude.

4.3. Project structure

To know a little more about the application, in this section we will quickly pass through the structure of the project. The Python classes with their code and their functions, the *KML* generated files, the *HTML* templates, ... The whole logic of the application is in the code of each class, and that's why it is also important to know what's behind each screen, so at first glance, a user sees a very simple functionality, but the reality is that there are hours of work, fixing errors, modifying and improving functions, ...

The Figure 13 represents the application design and also specifies in detail that piece of the code that is associated with each part. In this way, we can get an idea of how the project code is structured and what is the function of the classes, functions, styles, files, ... To explain point by point the different files and folders in the project, are organized by groups and each of these is associated with a part of the design.

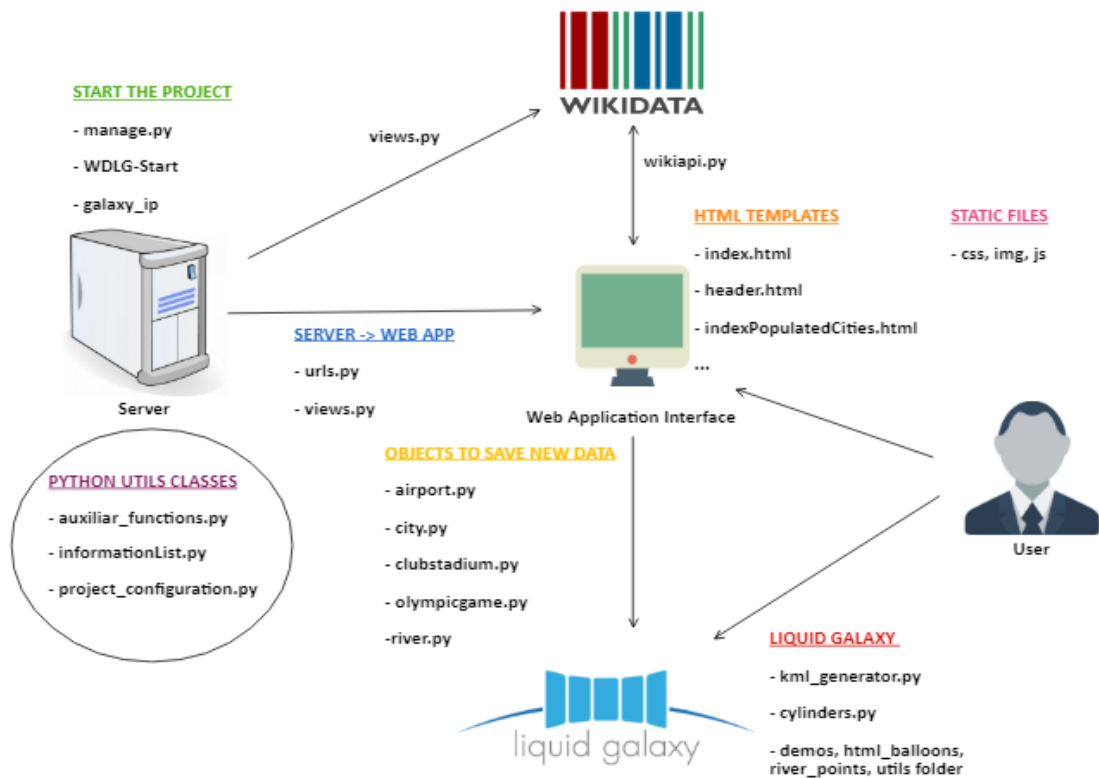


Figure 13: Structure of project files

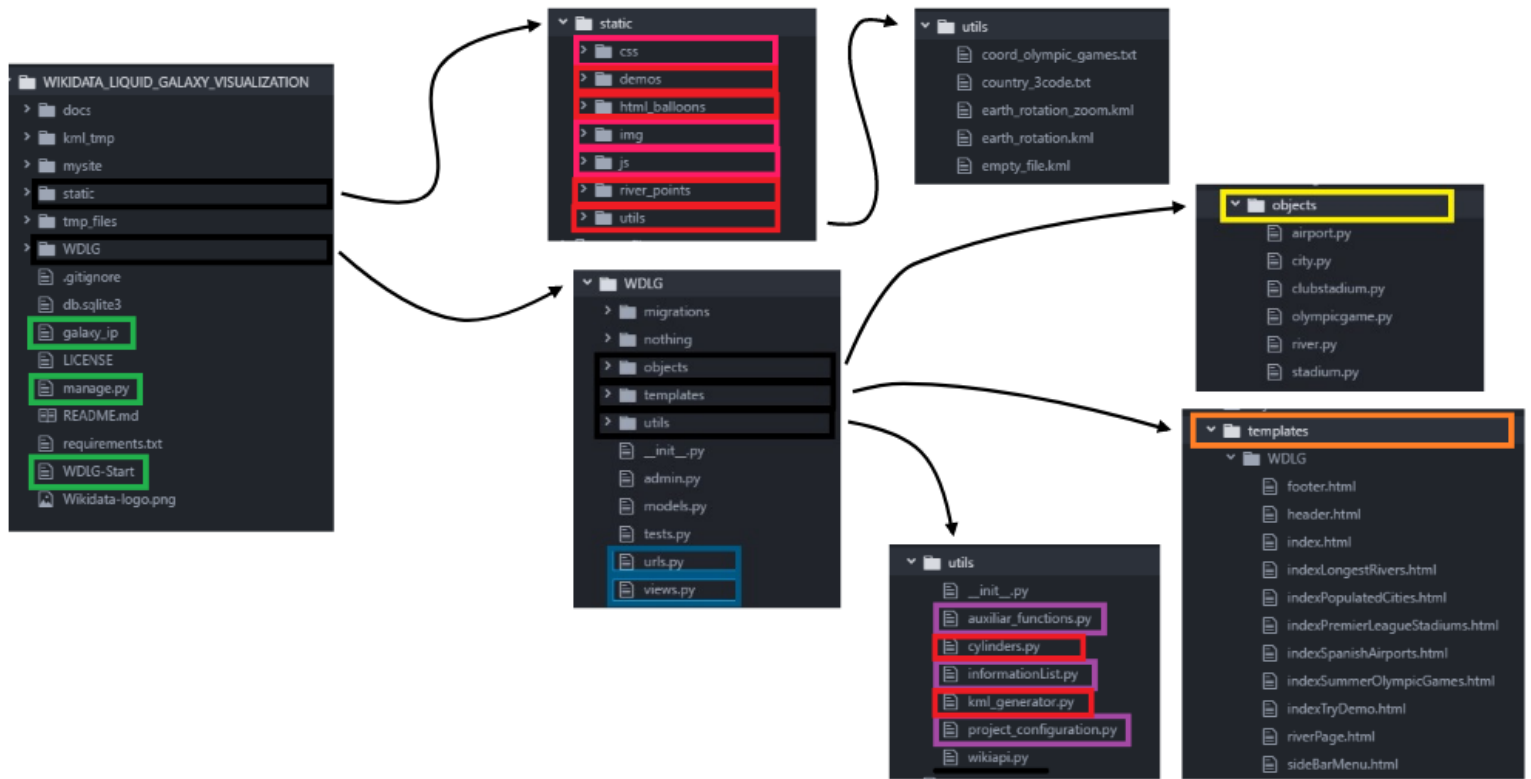


Figure 14: Project files and folders

Figure 14 is related to Figure 13, each file has a color that indicates where it is and can be identified in the project design. Figure 14 represents the entire file system of the project and each important folder is displayed to show the files it contains.

Inside the folder "WDLG" there are the **Python** classes of the project, all the logic is defined here and it's structured so that the functions of each of the classes are related to each other. It's important to maintain a good quality of code, since for future revisions or corrections to be much easier to return to understand the code.

objects folder (Figure 15) contains the classes that represent each of the objects that the use cases implemented. In these classes are saved the data of the real-time queries, and once they are stored in their respective class, then it is possible to do any kind of treatment with them, so it's easier to query on a class and get the data according to the desired variable. For example, generate the **KML** file, all the data is storage in the classes and for each use case the file is generated.

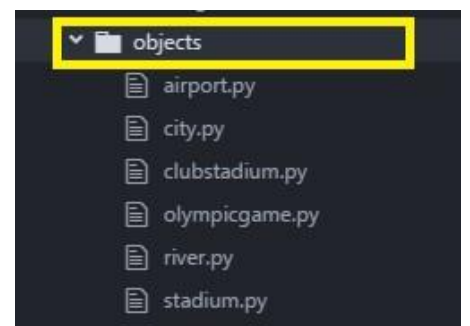


Figure 15: objects folder

All classes have the same structure, a constructor to add the values to each variable, and some extra function like adding the coordinates or an image. For example, **city.py** variables are city name, population, area, image, ... and has a function to add the coordinates “def coordinates (self, longitude, latitude)”.

There are two files which have a great importance for the connection between the server and the web application. **views.py** and **urls.py** are the main part when developing a **Django** project. In the Figure 16, it can be seen the diagram of how it works and how the different components are related between them.

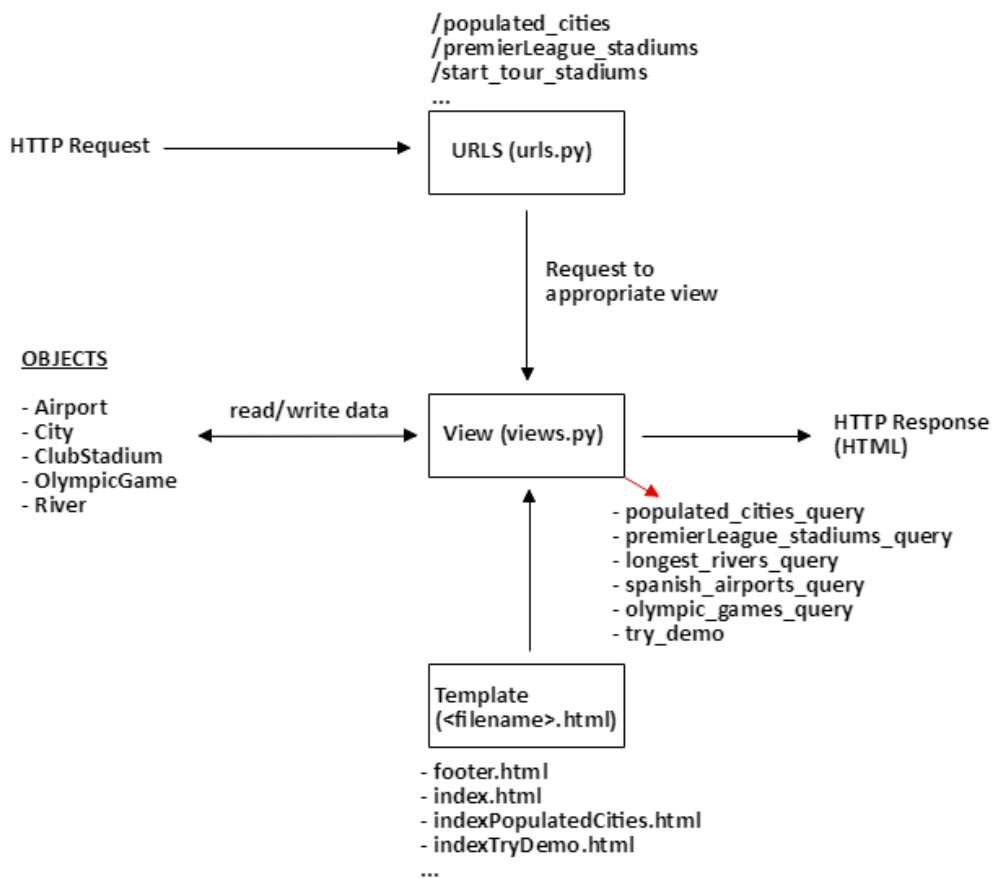


Figure 16: Django project structure

views.py it is one of the most important class and contains an important part of the code with its functions. All functions defined in **views.py** will be the once that will be executed once the user selects an option in the interface. These functions mainly are the ones in charge to make the queries to the **Wikidata** (`populated_cities_query`, `premierLeague_stadiums_query`, `longest_rivers_query`,

...). The data obtained in the generated query is saved in objects classes (airport, city, clubstadium, olympicgame, river). This data can be read at any time, and it's useful for the HTML templates. Also, other functions are defined such as starting a tour, stopping a tour, cleaning the *KML* folder, relaunch de *Liquid Galaxy*, ... All the functions have in common an interesting aspect, once arrived at the end of the function, this must do a return HTML page, each function has its associated template. When the HTML page it's returned to be shown, it's possible to pass parameters as simple values, lists, ... so that in the new loaded page can appear the new information obtained in the function. For example, in our case, after the query, some of the data obtained is displayed in the interface to give an extra information to the user.

The function response is the HTML page that is obtained according to the function and before, it reads the data that are stored in the objects classes for each use case. The request is through the URL that is according to the option that the user chooses. In the *urls.py* file it is defined.

urls.py is responsible for defining the route or URL path for each function implemented in the *views.py*. It's necessary to know which function to launch when the user selects an option (each option has associated a URL).

```
url(r'^$', views.index, name='index'),
url(r'^populated_cities', views.populated_cities_query, name='populated_cities'),
url(r'^start_tour_cities', views.start_tour_cities, name='start_tour_cities'),
url(r'^stop_tour_cities', views.stop_tour_cities, name='stop_tour_cities'),
url(r'^premierLeague_stadiums', views.premierLeague_stadiums_query, name='premierLeague_stadiums'),
url(r'^start_tour_stadiums', views.start_tour_stadiums, name='start_tour_stadiums'),
url(r'^stop_tour_stadiums', views.stop_tour_stadiums, name='stop_tour_stadiums'),
```

Figure 17: urls.py file

For example, as it can be seen in Figure 17, when the user wants to get the most populated cities option, the URL path will be “<server_ip>:port/**populated_cities**” and thanks to *urls.py*, the application knows that in the *views.py* must to execute the function “populated_cities_query”. Now the function will do everything that has been defined. Each URL path has associated one function, and in this way the application knows at all moment which function launches depending on what the user chooses on the web application.

The classes that are in **utils folder** (Figure 18) are the ones that complement the *views.py* functions. To make the code more structured, and easier to read and follow, it has defined other classes that contain different functions that are necessary to complete the queries, analyze the data, ... According to the colors, each class has importance in a part of the code design. As it has already been said, are complementary classes for other project functions. There are 6 files:

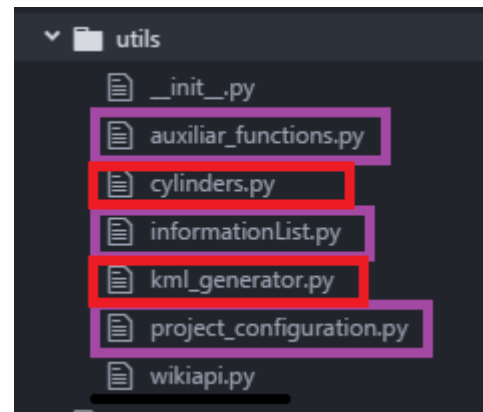


Figure 18: utils folder

- **auxiliar_functions.py:** Auxiliary functions to obtain values through another and functions to complete a dictionary with key and values. It's a useful class for all the project.
- **cylinders.py:** Class that generate the cylinders for the case of us of the Olympic games. When this class is called, 3 large cylinders are generated to simulate the podium, and for each country 3 cylinders more for the different medals types (golden, silver and bronze). Has a great importance in the *Liquid Galaxy* visualization step.
- **informationList.py:** This class works to save the generated objects once the query is called. For example, when the populated cities have already been obtained, a list is generated and stored temporarily in this class. In this way, to get any data we can obtain the list and obtain information of any of the cities.
- **kml_generator.py:** Is responsible for generating the *KML* file for each option. There are many common functions that differ only in the parameters that are passed to it. In general, the structure is very similar in all cases. To generate the *KML* file it has been used *pykml* library, and the file is sent to the *Liquid Galaxy* to be visualized.
- **project_configuration.py:** There are the functions that are responsible for obtaining the server IP, the Liquid Galaxy master IP, set the SSH connection,

... Is also the intermediate step between the class `views.py` and the `kml_generator.py`, here it checks the case being treated and define the **KML** file type to generate.

- **wikiapi.py**: Class for the whole topic of obtaining data through **Wikidata**. In some cases, it is necessary to do a scraping or cut a piece of code to get some special data. Here the connection is established, the query is called and the results are made the necessary operations to obtain the desired information. It's associated in **Wikidata** step, so it's useful to obtain data when the query is running.

`manage.py` is automatically created in each Django project. This file runs the Django application and start the server. Then, the Web Application is ready to be used. In addition, once the application is launched (specifying the **Liquid Galaxy** IP), a `galaxy_ip` text file is created with this IP and a command is launched so that the SSH connection knows the host and it is done correctly.

To launch the application in a single step easier, there is a script `WDL-Start` that automates all the steps to

launch the server and to be able to start using the application. First a folder is created and inside it is created the two necessary files that are sent to **Liquid Galaxy** (kmls.txt, query.txt). Once this it runs `manage.py` on port 8000 and as a parameter you must pass the **Liquid Galaxy** IP on which we want to connect. **`python3 manage.py $1 runserver 0.0.0.0:8000 --noreload`**

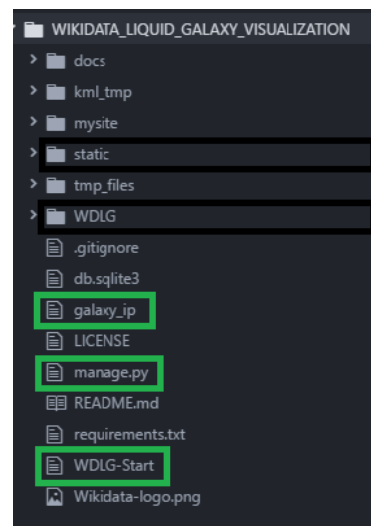


Figure 19: project structure

In the **static folder** (Figure 20) there are some folders like:

- **css**: is the css style, different styles are implemented.
- **demos**: the folder where the generated demos are stored. These are sent to **Liquid Galaxy**.
- **html_balloons**: the html templates of balloons that shows the information in the **Liquid Galaxy**.

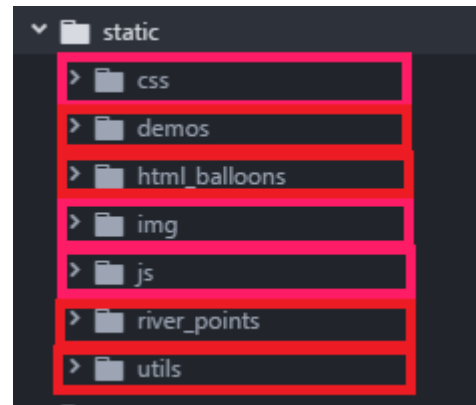


Figure 20: static folder

- **img**: static and dynamic images included in the application.
- **js**: JavaScript functions for the all application.
- **river_points**: the detailed points of the obtained rivers, are necessary to implement the experiences for all rivers and to have a good visualization in the **Liquid Galaxy**.
- **utils**: some files that are useful for application. For example, empty file is used to clean **KML** files in the **Liquid Galaxy**, or earth_rotation file is to generate a rotation while the user doesn't select an option. These files are basically related to the **Liquid Galaxy** for a better project presentation.

templates folder (Figure 21) contains the **HTML** files for the design of the screens of the interface. Each use case has its own screen, in addition to the side menu. Header and footer templates are always important to maintain an identical style across all application screens. Each function in **views.py** has associated a template, and when the function of the selected use case is called, then the HTML page is loaded and is the response that is returned.

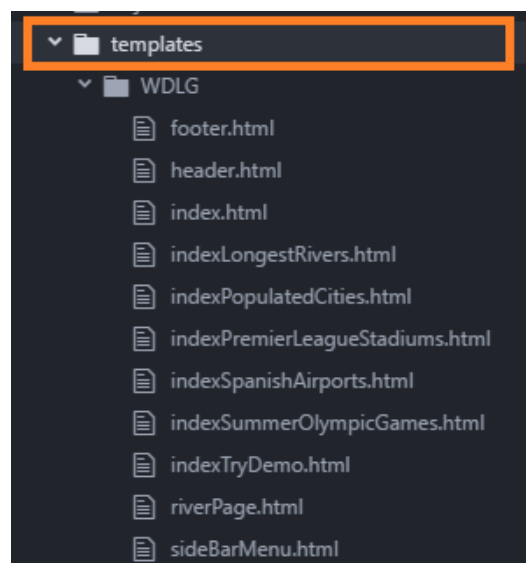


Figure 21: templates folder

Chapter 5

Web Application Interface Design

In this chapter will detail the entire design of the application interface. The different screens developed, the modifications, the improvements, ... The interface has been developed with **Material Design** and it tries to be as simple as possible and with good usability. It's important that a normal user can use and interact with the application without difficulty.

It will be explained how the different screens work, and in this way, will review the functionality of the application and each of the options that the user must interact.

To ensure that the user is always aware of what the application is running, an overlay has been developed. It must consider that every time a **Wikidata** query is made, this takes a few seconds, so during this time it's necessary to inform the user of what is happening. It shows an overlay that informs the user step by step of what is happening. According to the chosen option the text suffers some variation, but always follow the same steps. First, use case **Wikidata** query, then data is analyzed and saved in classes, and finally, **KML** is generated to be send in **Liquid Galaxy**. This information is shown to the user each time the application performs an internal action and have to report what is doing. Figure 22 shows the different overlay steps that inform to the user which internal application process is running.

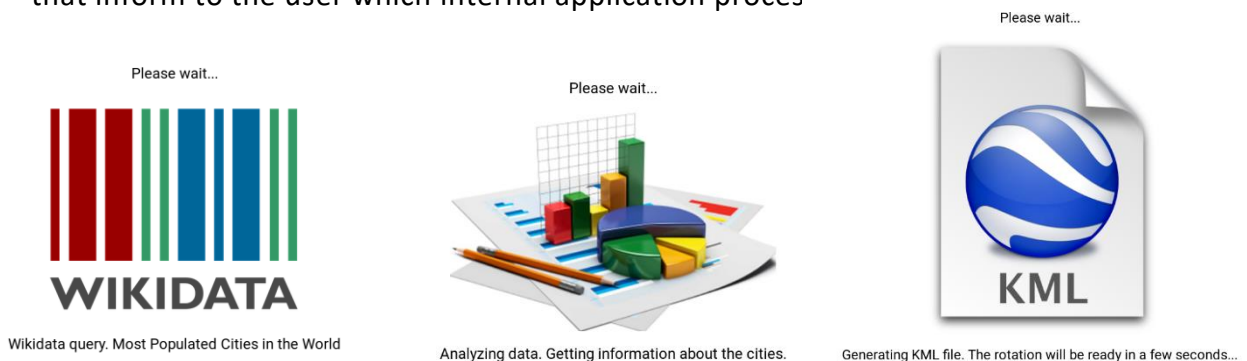


Figure 22: Overlay informative steps

5.1. Index page

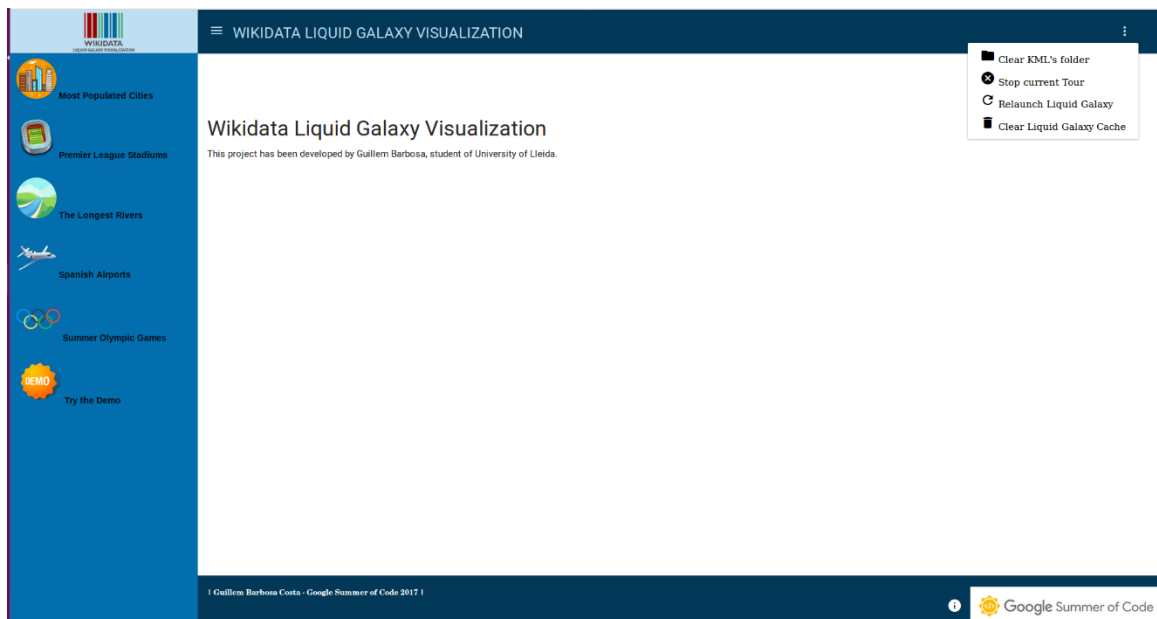


Figure 23: index page

The **index page** (Figure 23) is the page that is showed when the application is started. It's simple, the main thing is the side menu that offers the options to the user to start interacting with the application. There are six possible options to select, and each of them has a different functionality. They are the following:

- Most Populated Cities
- Premier League Stadiums
- The Longest Rivers
- Spanish Airports
- Summer Olympic Games
- Try The Demo

Significant names have been chosen so there is no doubt and it is clearly understood that each of the use cases offered. Whenever you click on the upper left application icon is accessed on this home page, this way it returns to the beginning and from here it starts again.

Another interesting feature, is on the right side, there is a dropdown with 4 options of configuration. These are:

- **Clear KML folder:** Removes all the *KML* files generated by the application. These are storage in *static/kml* folder.
- **Stop current Tour:** Stops any tour that is running. It's recommended use the stop button in each page, but this option it is in case of need.
- **Relaunch Liquid Galaxy:** *Liquid Galaxy* restarts, "*lg-relaunch*" command launches.
- **Clear Liquid Galaxy Cache:** All files are deleted from the cache in the *Liquid Galaxy* master.

5.2. Most Populated Cities page

WIKIDATA LIQUID GALAXY VISUALIZATION

Most Populated Cities In The World

Take a tour of the top 10 most populated cities in the world. Current information of each of them during the trip

1. Click "Start Tour" button.
2. The application will generate a KML file to show in the Liquid Galaxy.
3. Once everything is ready, the application will automatically start the most populated cities tour.
4. If you want to stop the tour, click the "Stop Tour" button.

1. **Chongqing** (People's Republic of China) -> 30,165,500 people.
2. **Karachi** (Pakistan) -> 24,300,000 people.
3. **Shanghai** (People's Republic of China) -> 24,152,700 people.
4. **Beijing** (People's Republic of China) -> 21,705,000 people.
5. **Lagos** (Nigeria) -> 21,324,000 people.
6. **Delhi** (India) -> 16,314,838 people.
7. **Tianjin** (People's Republic of China) -> 15,469,500 people.
8. **Istanbul** (Turkey) -> 14,657,434 people.
9. **Mumbai** (India) -> 12,442,373 people.
10. **Moscow** (Russia) -> 12,380,664 people.

START TOUR STOP TOUR

Guillem Barbons Costa - Google Summer of Code 2017 | Google Summer of Code

Figure 24: most populated cities page

This page, **populated_cities** (Figure 24), offers the possibility to take a tour of the ten most populated cities in the world. It can be clearly see where we are thanks to the title and the icon, and below is a small explanation so that the user has a guide to know how to act and avoid any type of problem. It shows the cities with their country and population. All this information is obtained in real time thanks to the query made in the **Wikidata**. Below, we find two buttons, one to start the tour and the other to stop it. Once the tour has started, then the **Liquid Galaxy** takes the lead. The button "Stop Tour" can be clicked whenever the user wants, and automatically **Liquid Galaxy** receives the command "**exittour=true**".

5.3. Premier League Stadiums page

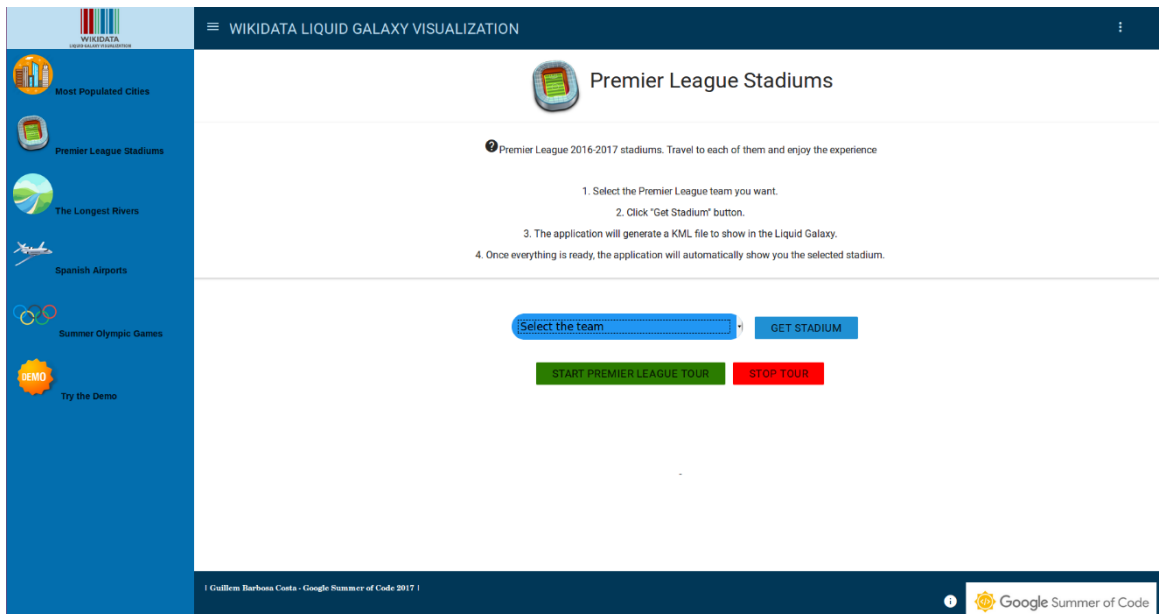


Figure 25: Premier League stadiums page

`premier_league_stadiums` page (Figure 25) gives the option to take a tour of all the Premier League stadiums, or select a specific team and fly to his stadium. There is a combo dropdown which offers all the teams to be selected. Once the team has been selected, then it's necessary to click "Get Stadium" button to start flying to the stadium.

When the *Liquid Galaxy* arrives at the stadium, then the body of the page adds the shield of the selected team, and text with the name of the club and the name of the stadium. In the Figure

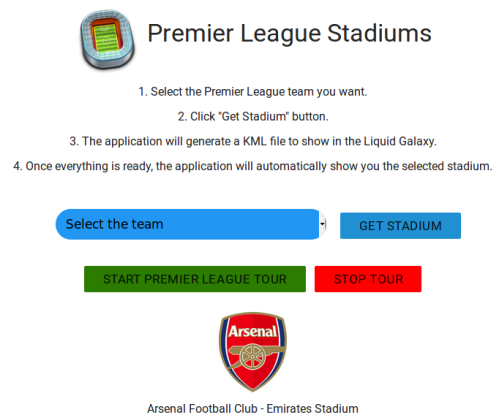


Figure 26: Premier League stadium selected page

26, we have selected Arsenal Football Club, so in the new page appears the Arsenal shield and text with the name of the club and the stadium.

In the case of selection the tour of the 20 stadiums, then the button “Start Premier League Tour” must be selected and automatically will start a tour for all stadiums in order of capacity. To stop the tour just click the “Stop Tour” button.

In all the pages there is the helping part for the user, it is of great utility so that they have a guide of how to go step by step.

In this case, Figure 27 is the overlay it shows the Premier League logo to inform that the information is about football and England teams. When the team is selected in combo, while the application is generating the *KML* file, in the overlay appears the club shield selected and text (“Fly to the stadium... <Club name>”). In the example, the selected club is Arsenal Football Club, and fly to Emirates Stadium.



Figure 27: Special overlay for Premier League stadiums

5.4. Longest Rivers page

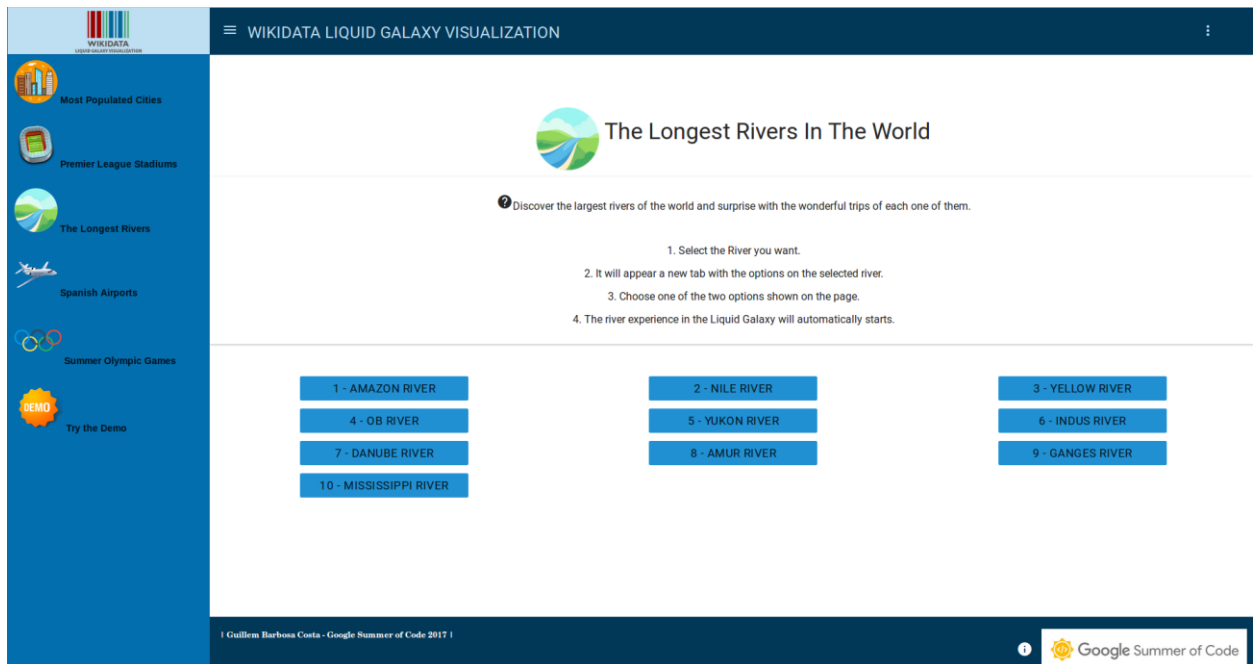


Figure 28: longest rivers page

longest_rivers page (Figure 28) offers the top 10 longest rivers for the user to select the one he wants. When the user clicks on the river button, then opens a new window that offers two possibilities with their explanations. While no river is selected, in the *Liquid Galaxy* appears the rivers marked with a line and an earth rotation starts.

It has been selected Amazon River, and there are two possible experiences to launch. Figure 29 shows the river page that is opened for each river, depending on which river is selected.

Tour experience runs a tour on the river from a height where it's possible to follow all the progress in detail.

Track experience runs a track line that follows all over the river from a height where it's possible to see the entire route. At any time, the user can stop the tour or close the window and click to another river on the rivers main page.

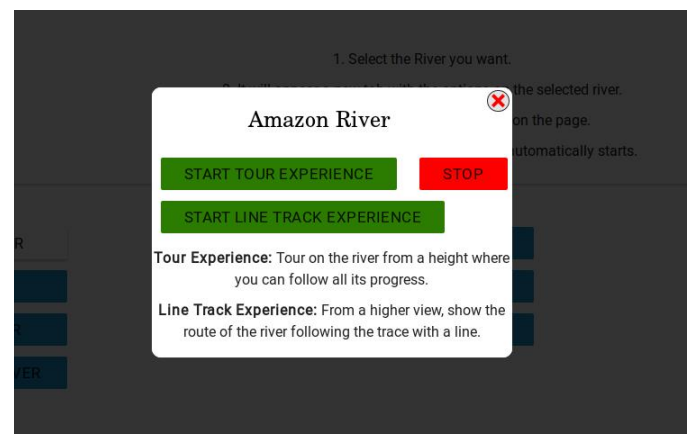


Figure 29: river page

5.5. Spanish Airports page

The screenshot shows the 'Spanish Airports' page in the Wikidata Liquid Galaxy Visualization application. The page features a dark blue header with the Wikidata logo and the text 'WIKIDATA LIQUID GALAXY VISUALIZATION'. A sidebar on the left contains navigation options: 'Most Populated Cities', 'Premier League Stadiums', 'The Longest Rivers', 'Spanish Airports' (highlighted), 'Summer Olympic Games', and 'Try the Demo'. The main content area is titled 'Spanish Airports' and includes an airplane icon. Below the title, there is a help icon and the text 'Here are the Spanish airports, travel on them!'. A numbered list of instructions follows: 1. Click 'Start Tour' button. 2. The application will generate a KML file to show in the Liquid Galaxy. 3. Once everything is ready, the application will automatically start the spanish airports tour. 4. If you want to stop the tour, click the 'Stop Tour' button. Below the instructions is a list of airports with their names and founding years: Barcelona–El Prat Airport – Barcelona – (1916), Almería Airport – Almería – (1968), Palma de Mallorca Airport – Mallorca – (1960), Málaga Airport – Málaga – (1946), Tenerife South Airport – Tenerife – (1978), Alicante Airport – Alicante – (1967), Tenerife North Airport – Tenerife – (1946), Gran Canaria Airport – Gran Canaria – (1930), San Pablo Airport – Sevilla – (1919), and Lanzarote Airport – Lanzarote – (1936). At the bottom of the list are two buttons: 'START TOUR' (green) and 'STOP TOUR' (red). The footer contains the text 'Guillem Barbons Costa - Google Summer of Code 2017' and the Google Summer of Code logo.

Figure 30: Spanish airports page

spanish_airports page (Figure 30) is like the use case of the most populated cities. The page shows a list of the Spanish airports obtained with the query. On the bottom there are the two buttons, one to start the tour and the other to stop it. If the user has already interacted with the most populated cities page, then this use case will be easy so it is very similar.

5.6. Summer Olympic Games page

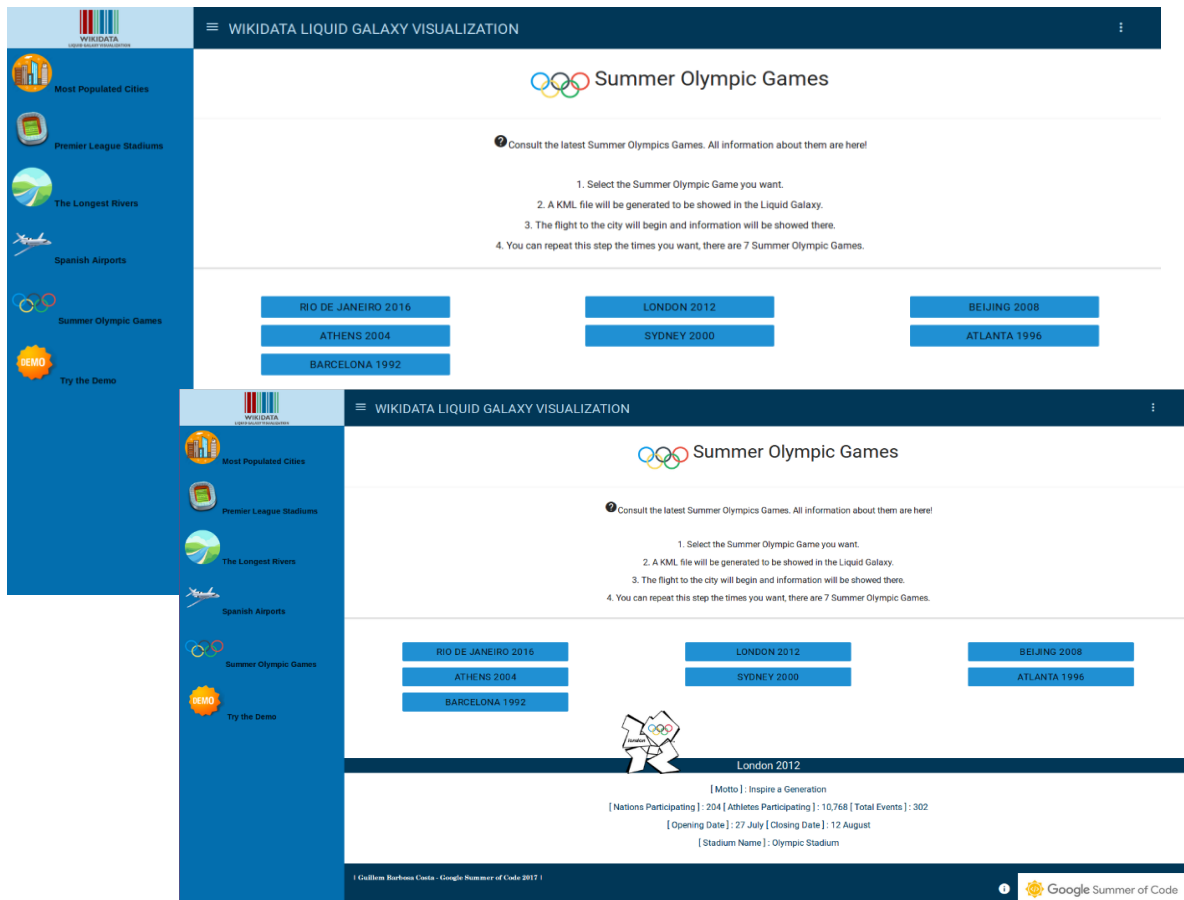


Figure 31: summer olympic games

summer olympic games page (Figure 31) shows the last 7 Summer Olympic games. There is a button for each game, and its necessary click in to get information. In the *Liquid Galaxy* is shown cylinders representation above the host city of the event. The representation is the podium of the countries with more medals, and for each country the type of medals achieved.

The overlay that appears in this use case is the Olympic rings, and a text that contains the Olympic Game selected. Figure 32 is an example by London 2012 selected game.

Please wait...



You have selected -> London 2012
Now it starts the flight to the city.

Figure 32: summer olympic game overlay

Once the **KML** file is generated and the cylinders are loaded in the **Liquid Galaxy**, the same page is reloaded and new data and information are showed. In the image example, London 2012 has been selected, and some extra information has appeared on the page.

5.7. Try the Demo page

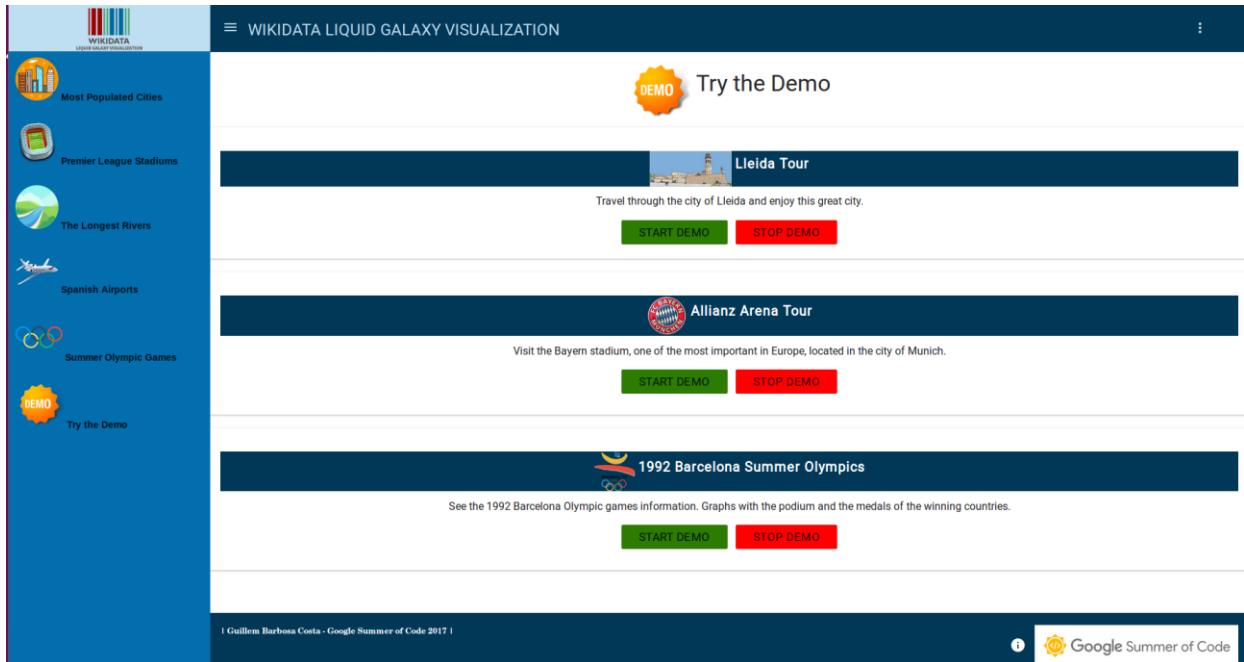


Figure 33: try demo page

demo page (Figure 33) is the section to test the application, where we can find 3 possible demos already generated that allow to test the application and to see how it behaves. **Lleida tour** allows to travel through the city of Lleida and get some extra information in the balloon. **Allianz Arena Tour** offers a view of Bayern Munich stadium. **1992 Barcelona Summer Olympics** flies towards the city of Barcelona and shows the cylinders representation that shows medals data and the countries of the medal table. Each of the demos options have their own buttons to start or stop the tour. It's simple because what is intended is that the user without knowing anything about the application could prove it without any problem and learn easily.

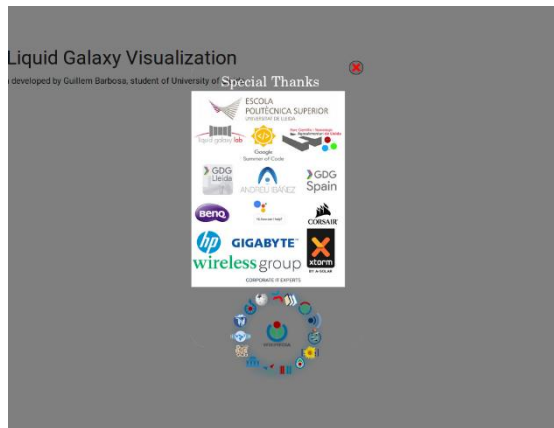


Figure 34: footer.html

As can be seen in the Figure 34, the foot of the page is formed by the name of the author of the project (Guillem Barbosa Costa), and naming **Google Summer of Code** for giving me the opportunity of these summer scholarships. The icon information will open a new window where appears collaborators logos.

5.8. Use case example

Use cases are a technique for capturing potential requirements for a new system or for a software update. Each use case provides one or more scenarios that indicate how the system interacts with the user or another system to achieve a specific objective. Normally, in cases of use, it is avoided to use terms or technical words, since a language that is closer to the end user of the product is preferable. Sometimes, for the development of user cases, along with analysts there are also inexperienced users who help professionals with the analysis

As it has been said, a use case is a sequence of interactions that are developed between a system and its actors as a response to an event that initiates a main actor on the system itself. Usage case diagrams are to specify the communication and behavior of a system through its interaction with users and / or other systems. Or what is the same, a diagram that shows the relationship between the actors and the cases of use in a system. A relationship is a connection between the elements of the model. We find the association, which is the most basic relationship, dependence and generalization. You can also use "include" and "extend".

Here is an example of a well-detailed use case of the application:

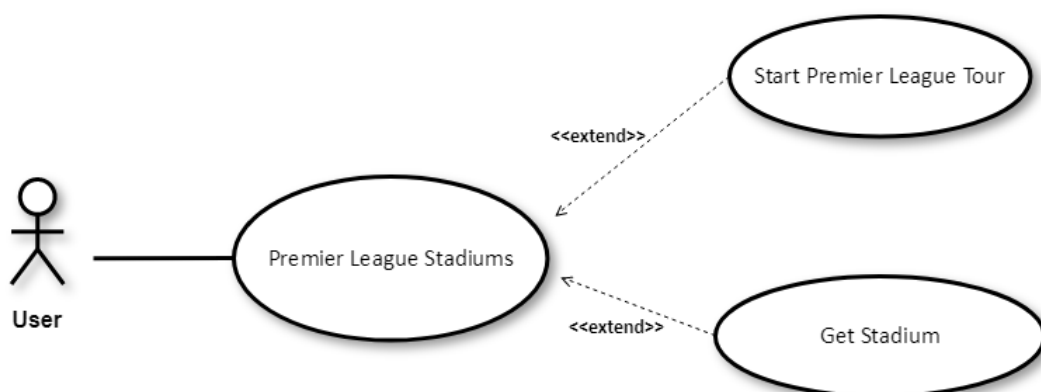


Figure 35: Premier League Stadiums use case

Figure 35 represents the action when the user selects the Premier League Stadiums option in the side menu. Inside this action it extends two new possible actions, Start Premier League Tour, a tour with the 20 Premier League stadiums, or Get Stadium, where previously a team of the Premier League is chosen and then flying to the selected stadium.

Figure 36 details the specification of the use case chosen to develop.

Use case: Premier League Stadiums (select team stadium).	
Actors: Main: User (initiator).	
Purpose: See the Premier League stadiums in the Liquid Galaxy.	
Description: The user has a combo list with all the Premier League teams, choose the team and start to travel to the stadium with “Get Stadium” button click.	
Type: Primary and essential.	
Typical course of events	
User	System
1. The user selects the Premier League Stadiums option.	
	2. The system shows the HTML page (premier_league_stadiums.html).
3. The user open the Premier League teams combo list.	
	4. The system shows the list of teams obtained with the <i>Wikidata</i> query.
5. The user selects one team and click “Get Stadium” button. (extends “Get Stadium”).	
	6. The system generates KML file where there is information about the stadium and the team.

	7. The system shows the overlay during this time. The shield and the name of the stadium are shown in the overlay.
	8. The system returns to show the html page, and this time with the shield, the name of the team and the name of the stadium selected.
9. The user can see the web page and in the Liquid Galaxy the flight to the stadium.	

Alternative courses:

5.b. The user does not select any team from the list.

1. The user clicks the “Get Stadium” button.
2. The system can’t generate the KML file, so no team has been selected.
3. Liquid Galaxy doesn’t receive any file, there is no flight.
4. The user can choose a team and continue with the course of the events.

5.c. The user clicks the “Start Premier League Tour” button.

1. The system generates a KML file tour with the 20 stadiums.
2. The system, sends the KML file to the Liquid Galaxy and starts the tour.
3. The user can see in the Liquid Galaxy the Premier League stadiums tour.

3.b. The system makes the query, but the network fails and no results are obtained.

1. The system shows the HTML page, but the combo teams list is empty.
2. The user can’t select any team.

3.a. The user clicks the “Get Stadium” button.

1. 5b alternative course.

3.b. The user selects another time the Premier League Stadiums option.

1. Start the typical course again (2).

Figure 36: Premier League Stadiums use case specification

Chapter 6

Conclusions and future work

Now, after months of work and effort, with the completed application, conclusions can be drawn regarding everything that surrounds the development of a project of these characteristics. The objectives that were detailed at the beginning have been perfectly achieved, and the application has all the functionalities that had been raised when initiating the project.

The planning of the tasks has been fulfilled and this has allowed to be able to move forward with little complications. The features have been completed and tested to certify how well the application works. In addition, the usability and the ease of use of the application has been achieved. The application interaction will be very easy and with a simple and understandable language.

Some of the tasks, because of there have been some difficulties, have had to be extended, but other tasks have been carried out and finished more ahead of schedule. Some complicated errors or the complexity of some part of the development, have also made some tasks longer, but in general, the planning established at the beginning has been quite fulfilled.

In general, the project has been a success, during the application has been developed, you could see the evolution and test the application so that you could see or discover some error or improvement to do at the same time of development.

This project has been submitted to Google Summer of Code. During this summer it has been developed following the timeline, the evaluations by parts and the help of the mentors in any problem. Figure 37 shows that the project has been certified and completed, so it's a good recognition to the work done and a great experience.



Figure 37: Google Summer of Code certificate

As I said, the experience has been very good, I have learning developing the project, I have discovered new techniques and languages, and my colleagues have made me discover many things that have been useful to me. I have gained in knowledge, but also in responsibility, perseverance and sacrifice. A project of this caliber brings many positive aspects, and the truth is that soon, this time that I have worked with the project will value and serve me to improve.

It is very satisfying and gratifying to see how the application has been growing and improving is a great joy. The application will be available in the Lleida Liquid Galaxy Lab, and can be used by all types of users. This is another part that makes me happy, so the work of this months will continue to be present and active, and I hope to offer a good product and that they enjoy it.

6.1. Future work

To finish and close the project, there is some work that can improve it. Small touch-ups in the design of the interface, to make it even simpler, and add other new features that give a more varied air of options. *Liquid Galaxy* offers many possibilities, in this application we have focused especially on displaying *Wikidata* information from certain places, but we can also add other more complex cases that bring new experiences. Thanks to the large amount of data that *Wikidata* offers, we have good ideas for new implementations, so the work that remains to be done is to think and plan new tasks to improve the functionalities offered by the application. For the moment, we have the operational application, is working well, and any changes that implements will be done with the maximum perfection and always within the possibilities that have. Finally, it could improve and permanently implement the interaction with *Google Assistant*, this would give an innovative and modern touch to the application. It has worked with this new technology, but is not completely finished, so it would be a good way to improve the application.

References

[1] Google Summer of Code.

- <https://summerofcode.withgoogle.com/about/>

[2] Liquid Galaxy.

- <https://liquidgalaxy.endpoint.com/>
- https://en.wikipedia.org/wiki/Liquid_Galaxy

[3] Wikimedia Foundation.

- <https://wikimediafoundation.org/wiki/Home>

[4] Wikidata.

- https://www.wikidata.org/wiki/Wikidata:Main_Page
- <https://en.wikipedia.org/wiki/Wikidata>

[5] Keyhole Markup Language (KML).

- <http://searchmicroservices.techtarget.com/definition/Keyhole-Markup-Language-KML>
- <https://developers.google.com/kml/documentation/kmlreference>

[6] Django.

- <https://www.djangoproject.com/>
- <http://www.pythonforbeginners.com/learn-python/what-is-django/>

[7] Material Design.

- <https://material.io/guidelines/material-design/introduction.html#introduction-goals>
- <https://envato.com/blog/introduction-material-design/>

[8] SPARQL language.

- <https://ontotext.com/knowledgehub/fundamentals/what-is-sparql/>
- <https://www.w3.org/TR/rdf-sparql-query/>

[9] Google Earth.

- <https://www.google.es/intl/es/earth/index.html>

[10] Space Navigator.

- <https://www.3dconnexion.es/index.php?id=26&redirect2=www.3dconnexion.es>

[11] pyKML library.

- <https://pythonhosted.org/pykml/>

[12] Wikidata Query Service.

- https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/A_gentle_introduction_to_the_Wikidata_Query_Service
- <https://query.wikidata.org/>

[13] Google Assistant.

- <https://assistant.google.com/>

[14] Python language.

- [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- <http://whatis.techtarget.com/definition/Python>

[15] Atom.

- <https://atom.io/>