# A Web-based Editing Interface for Georeferenced Data

**Sunday 6th January, 2019 - 15:32**

Iris Kremer
*University of Luxembourg*
*Email: iris.kremer.001@student.uni.lu*

Jean Botev
*University of Luxembourg*
*Email: jean.botev@uni.lu*

*Abstract*—**This paper presents the bachelor semester project made by Iris Kremer under the direction of her project academic tutor Jean Botev. This project seeks to create a web-based editing interface for georeferenced data, which involves the domains of Geographic Information Systems (GIS) and web-based UI design. The scientific aspect of this project concerns research on applications using georeferenced data, while the technical dimension is focussing on APIs, in particular how they work, emerged and how to implement them in the code. The editing interface developed during this project enables users to handle georeferenced data. The precise goals fixed for this project have been set in the section objectives and detailed in the section requirements.**

## 1. Introduction

It can be obvious to everyone that georeferenced data has become essential in our everyday life, but its importance and the role it plays is still underestimated. Sure, we could not live without interactive maps and navigation systems (e.g. GPS) any more, since it has improved considerably services like route seeking and finding locations, but there are many other domains using georeferenced data, whether in the entertainment field, or in scientific research, for instance statistic data visualization.

The term 'georeferencing' means relating some data, e.g. an image, an object, some information etc., to geographic information, which are coordinates corresponding to a place. Since the georeferenced data is bound to these coordinates, it is assigned to the corresponding place and can be referenced on a map. A special case of georeferenced data is the so-called spatial data or geospacial data, which specifically represents the shape, size and location of an object, like topographic elements or buildings. [2]

As explained in the first paragraph of the section, georeferenced data is used in many domains. To give an example, on Google Maps and some other web-cartography platforms, users can switch between map mode and satellite view. This is only possible thanks to georeferenced data. On the map, some reference points have been set. The corresponding points are then georeferenced on the images taken from the satellite and from the planes, and the border of adjacent images are corrected such that they fit. This is how the satellite view arises and also how streets and places can be displayed on it as an overlay that fits the image. It goes even further. The map itself is also georeferenced with respect to a given system of coordinates, which is in our case the latitude and longitude. Virtual maps in video games are developed using the same process, but maybe in another coordinate system than the one used for our earth.

Another application of georeferenced data is in statistics. Researchers and scientists can create whole databases of georeferenced data and display them on a map for a clear overview. Also, these data can be analysed by data scientists and converted into tables, sorting them for instance by regions using the coordinates assigned to each data. This helps the analysis of the data - and mention that the results of these studies may also be presented using georeferenced data displayed in a map.

## 2. Project description

### 2.1. Domain

The aim of this project is to explore how georeferenced data can be created and managed using web-technologies. Therefore, the project concerns mainly two domains: Geographic Information Systems and Web-based user interface (UI) design.

**Geographic Information Systems.**

GIS or Geographic Information System describes a system handling geographic data. It means, all tools designed to search, manipulate, analyse, structure and display geographic data, for example OpenStreetMap, Google Earth and Google Maps, belong to GIS. Such systems can be developed to enhance our everyday life with GPS and other services, but also to improve analysis of research results in any domain implying georeferenced data.

Through Application Programming Interfaces (APIs) from already mentioned and other GIS, web-interfaces and features to create and manage georeferenced data can easily be developed, which leads us to the second domain of the project.

**Web-based UI design.**

There are hundreds of possible applications of GIS, but this project will focus on the management of georeferenced data from the ordinary user's point of view. It will seek to create a web-based interface handling a basic type of georeferenced data: information bound to points on a map, which could be useful for anyone who wants an overview of personal points of interest. The design of the web page will have to be carefully thought out to keep it clear and avoid overloading the page. Therefore, the second domain concerns the web development of a user-friendly, and at the same time minimal interface, offering many features all accessible on the same page.

## 2.2. Objectives

There are two main objectives to fulfil. First, the concept of APIs should be explored by discovering some APIs for web-cartography. Second, the project should result in a direct application of the newly acquired knowledge, choosing one of these APIs to create a prototype of an interactive, web-based interface for georeferenced data.

**Exploring APIs for web-cartography.**

APIs are widely used in web applications, providing many types of services, but since the main topic of this project is georeferenced data, the APIs for web-cartography are the ones of interest. Discovering how they work is the first step to understand how to complete the project.

To compare the features of each API, a suitable procedure is to build a taxonomy of the available platforms and their APIs. This first deliverable gives a clear overview of the existing features. Therefore, it is necessary to conduct research on other projects using these APIs or comparing them. Also, some APIs will be tested to choose the most suitable one for achieving the second objective of this project.

**Development of an interface for georeferenced data.**

The second part of the project aims to create a prototypical web-based interface for georeferenced data, which will be the second deliverable. Using an API for web-cartography to display a map that opens on the user's location, an interactive, user friendly interface will be created. This interface should be kept minimal, i.e. absolutely all the features must be available on the page with the map, without overloading the interface.

The purpose of this interface is to enable users to pick any point on the map, enter some information about the place in a form, save it and display it. The information can include some text and an image. It will be saved persistently in a database, in relation with the point picked on the map. Once saved, the place will be displayed in form of a pin on the map and related information will be accessible either in an information window bound to the pin, or in a list of all saved places.

To manage these points of interest, other useful features could also be implemented, like an option to delete the place (i.e. delete its pin and remove it from the database as well as from the list) or some ways to visually group the saved places into distinct categories.

## 2.3. Constraints

Theoretically, the only constraints of this semester project are those of the APIs used, i.e. which features they provide. These are not serious limitations, since the APIs used in this project have a large variety of features.

However, to reach the objectives fixed for this project, a basic level of HTML and a solid level of CSS and JavaScript are also required. Knowledge about these three mark-up and programming languages had to be acquired during the production of the deliverables. Most of this part of the work can't be presented as a deliverable, but still had to be done in order to achieve the goals set.

## 3. Background

The scientific dimension of this project deals with research on existing applications using georeferenced data, while the technical aspect approaches the operation modus and implementation ways of Application Programming Interfaces (APIs), with a little focus on JavaScript, since this project uses JavaScript APIs.

## 3.1. Scientific

Applications using georeferenced data are much more widespread than it seems; indeed, there exist numerous of them for many different usages.

The first applications using georeferenced data we think of are technologies using web-cartography, among them Google Maps, Google Earth and OpenStreetMap which we already saw previously, but also Bing Maps, Apple Maps etc., as well as their equivalent mobile applications. These are all GIS, used mainly for global orientation such as finding routes or addresses.

That being said, simple earth maps are often not enough for research purposes. Science often needs special maps to study for example the topography of regions, the earth shape (so-called Geodesy) or the climate. Other GIS are focussing on building these kinds of maps, for example ArcGIS. This application will use specific georeferenced data to build these maps. For example, georeferenced climate data enables to create maps displaying the climate in different regions, eventual climate changes etc.

Another application in science seen previously in the introduction is to display georeferenced statistics information on a map (statistic data visualization).

There are not only scientific applications which use georeferenced data. For instance Snapchat, a social network app, uses an earth map which shows the positions respectively of the user and its friends using the same app. Pokémon Go, a famous game on mobile devices, also uses a map of the real world to display the position of the player, as well as so-called pokémons near him. These only exist in the augmented reality world, but still, the virtual pokémons are georeferenced on the map as well, which shows by the way that the georeferenced data doesn't need to be real.

Coming to the field of games and virtual objects, as already explained in the introduction, virtual maps and even virtual environments are created the same way as the earth map and the 3D model of the earth, and objects on these maps are georeferenced as well.

Fortnite is an example for this. The environment has been created by georeferencing the objects (topography, trees, houses, places where weapons appear etc.) in a coordinate system. The character played is moving in this virtual world. A corresponding flat map has been created to display the users position and orientation.

Another example is the game GTA 5, which uses a virtual environment and a corresponding map, with parts of the American city Los Angeles, that have been recreated for this game.

As we can see, the use made of georeferenced data exceeds the routing searches, but goes over to complex research using specific maps which require certain types of georeferenced data. Also, georeferenced data is used in various apps like Snapchat and plays also an important role in the gaming field.

## 3.2. Technical

APIs are libraries of features, allowing users to access the services of an application through either a programming language, or another application. It is a way of sharing data and/or functionality without giving up the copyrights and at the same time without requiring the users to understand the exact functioning of the application. It is similar to a library in the real world, where customers can borrow books to access information without stealing the rights, or to a car, which everyone can learn to drive and use without having to know exactly how the car works.

The first APIs emerged in the 60s. At that time, APIs were used in operating systems only, since there existed neither internet, nor personal computers. Since they are so useful, they evolved over the years and their importance in computer science increased. Web APIs exist since around the year 2000 and progressed rapidly, also thanks to companies like Amazon and eBay, as well as social media like Facebook and Twitter, which developed APIs for their own applications, such that nowadays, web APIs are of paramount importance in the domain of web-development. [3]

APIs have many advantages. As already mentioned, one of them is that the owner of the application can promote his application and make it available for various uses. Eventually, the owner can ask users to pay to use his API, but there exists also many free APIs.

APIs also advantage the programmers using them, because they neither need to know how the application works, nor have to program it themselves to use it. For instance, this project is using an API to display a map and implement its features. Without this API, this project would have needed to build an entire world map by ourselves, as well as programming each function call, which would have been a huge workload and required very advanced knowledge of programming, as well as access to a large amount of data.

There are two different ways to get access to an API. Sometimes, only one possibility is available, while for other APIs, the programmer can choose the way of implementing they want.

The first possibility is to download one or some file(s) containing all features from the API and link the file(s) in the head of the HTML code of the page using this API.

The second way is to directly access the file on an online server using an URL as source for the file in the head of the HTML document.

In this project, all APIs implemented use JavaScript to access the features. JavaScript is a programming language created to enhance interaction between web pages and users. Programmers use JavaScript to build interactive websites with functions bound to HTML objects. This is possible because JavaScript is able to create, grab, modify and remove HTML elements dynamically.

On load of a web page, all included files in the head of the HTML document are loaded and executed by default before loading the body. This means that the browser is loading each file, then executes it directly (i.e. create global variables, execute functions called on load etc.) before charging the body of the document.

When binding a JavaScript (JS) file in the HTML document, it is important to take care that the JS file is loaded after the body, because some of the JS functions, that are bound directly to HTML elements or modifying them, are executed on load of the file. This means the elements have to be already present on the page when the JS file is executed. Therefore, the attribute 'defer' has to be added to each JS file included in the HTML file.

Once included correctly in the HTML file, the API features can be accessed using the API-specific function calls in a personal JS file. These function calls are not standard JS features. They have been created by the developers of the API and made available inside the API file(s), so each API-specific function call is accessing some feature of the file, which is the purpose of the API.

## 4. A Web-based Editing Interface for Georeferenced Data

### 4.1. Requirements

In the objectives, we already saw that the following deliverables will be produced during this semester project:

- A **taxonomy of APIs** presenting briefly available technologies in the domain of web-cartography
- A prototypical **web-based editing interface for georeferenced data** with features to handle georeferenced data

Each of these deliverables has to fulfil its requirements set in this section.

**Taxonomy of APIs.**
There exist many different APIs for web-cartography. Although the Google Maps JavaScript API remains the most popular one, many performant and competitive alternatives have been developed over the years since the emergence of map interfaces. Yandex Maps has become popular in Russia, while Bing Maps seems to stay in the shadow of Google Maps for now. Apple has developed its own Apple Maps. The free and open-source APIs Leaflet and OpenLayers for OpenStreetMap (OSM) also grew thanks to many contributors and became very popular. Each of these APIs has advantages and drawbacks which have to be explored.

It has to be decided among all these APIs which one will be used for the second deliverable of the project. The end result of the project may depend on this choice, so it has to be smart and founded. The purpose of the taxonomy is to create a clear overview to help choosing the most appropriated API. This is the scientific deliverable of this project.

The requirements for this taxonomy are the following:

1) For lack of time, a first, founded selection has to be made before building this taxonomy to decide which APIs will be compared.
2) The taxonomy shows each chosen platform with all its different APIs.
3) Each API will be presented briefly with a summary of its main functionality and purpose, such that it gives a small insight of the features available for this API.
4) It will be said which ones of the APIs would be suitable for this semester project.

**Interface.**

Creating a web-based editing interface for georeferenced data involves two different aspects. The first one is the design of the web page and the second one is the functionality.

This interface is web-based and designed for common people to georeference points of interest (PoI) on a map. Therefore, the layout is one of the main topics to work on. It has to be simultaneously user friendly and minimal.

A user friendly design means that the look of the page is engaging, such that the user wants to use it when he/she arrives on the page. Therefore, it should be kept simple and clear. The most important thing is to avoid discouraging the user with an unorganised and overloaded interface.

In this project, the minimal design is bound with the user friendly design, because it is required to avoid page overload. The goal is to have all implemented features available on the same page, but in a compact way that doesn't disturb the accessibility of other features and clearness of the layout.

Therefore, requirements concerning the design of the page are the following:

1) Design a clear and aired interface.
2) Keep this interface minimal to avoid overloading the page.
3) Design an attractive layout that invites and encourages users to use the interface.
4) All features have to be accessible on the same page, namely the page with the map.

The functionality of this interface is to enable users to handle simple georeferenced data, like saving points of interest with some information. Therefore, many features have to be implemented. Some of them are features directly related to the map, while other ones are storage features. There are also some which combine and bring together both aspects - the map and the storage.

Some features are required for the web page to fulfil its target functionality - these are the functional requirements. Other features are optional and not truly required for the interface, but still useful to have - the non-functional requirements.

Functional requirements:

1) On click on the map, a marker is set on mouse position.
2) Information about the selected PoI can be entered in a form. This information is composed of at least a name and eventually a comment and/or an image.
3) On click on a 'Save' button, the entered information is saved, in relation with the place selected on the map, persistently in a database.
4) This PoI is displayed in form of a pin on the map. Clicking on the pin displays the associated data in an information window.
5) There is a list of all saved places with all the information saved.
6) On click on a 'Delete' button, the selected place will be deleted and removed from the database.

Non-functional requirements:

1) On click on a 'Reset' button and after having saved a place, the form is reset to its default state.
2) The user can choose a pin type and colour for each place. This way, the places can be visually grouped. Also, the pins can be displayed or hidden according to their type and colour, such that the user can choose to show only one type or one colour of pins.
3) The feature 'Delete all places' can remove all saved places from the map, the list and the database.
4) The feature 'Pan to all places' enables to pan the map such that all saved places are visible at once.

Keep in mind and take in account that this interface remains a prototype, which is not fully developed and secured.

## 4.2. Design

It was important to decide about the design of each deliverable before starting to produce them.

**Taxonomy Design.**

The purpose of the taxonomy of APIs is to compare the different APIs for web-cartography in order to discover the available features handling georeferenced data and to help choosing the most suitable API for this project. Because it is impossible in terms of time to make research about each of the platforms mentioned in the requirements section, it had to be decided which ones would be compared in the taxonomy.

The platform Google Maps is the most well-known and popular one, which implies it logically to be considered in the taxonomy. OpenStreetMap is the greatest open-source API for web-cartography, so it also had to be considered. Lastly, in order to have a comparison with another, not open-source platform, Bing Maps has also been chosen for the comparison, which leaves Yandex Maps and Apple Maps out. These two platforms were less adequate to be compared, because Yandex Maps is less known in western Europe, and since the JavaScript Apple Maps API (MapKit JS) is still in beta version, it was inappropriate to keep it as a possible choice. [4]

To conclude on this part, Google Maps, Bing Maps and OpenStreetMap are the platforms which will be compared in the taxonomy. Research in the documentations and other studies is performed on these three platforms to collect information about their APIs and present them in a table.

Google Maps has ten different APIs, which can be divided into three subgroups according to each of their functionality. OpenStreetMap (which will sometimes be abbreviated as OSM) has one API which is also for developing OSM and its basic maps themselves, but some other APIs have been developed to access the map of OSM and add features without having to deal with the developer tools. These APIs are presented as APIs for OSM as well. Bing Maps has five different APIs, deferring in their supported operating systems and features.

Information about all these APIs main functionality and features will be found on their respective documentation pages. Also, reports of studies comparing these APIs are useful to learn more about them and consider other opinions.

**Interface Design.**

For this project, a small website has been created, using the same font family everywhere - Calibri - and HSL colours of the same hue, to keep a consistent layout. On each page, there is a navigation bar available for easy navigation to the other pages.

The website is composed of three pages: The welcome page, the map page and the saved places page. But to be more precise, the welcome page is only a dummy page - a placeholder for a login page, which could have been added if the project had been carried out further and improved with user accounts. The saved places page for its part is a draft page, which first has been used to generate the list of saved places in a normal sized space. Still, this page could become personal space page to handle the user account in the hypothesis that user accounts would be implemented.

The web-based interface that has to fulfil the requirements given in the section before is the map page, so this page is the actual editing interface for georeferenced data. Therefore, the following part of the section will only focus on the design of this page. Since desired features are already detailed in the requirements section, we will mainly describe the layout of the interface.

The global layout of the interface is inspired from the commonly used mapping interfaces Google Maps, Open-

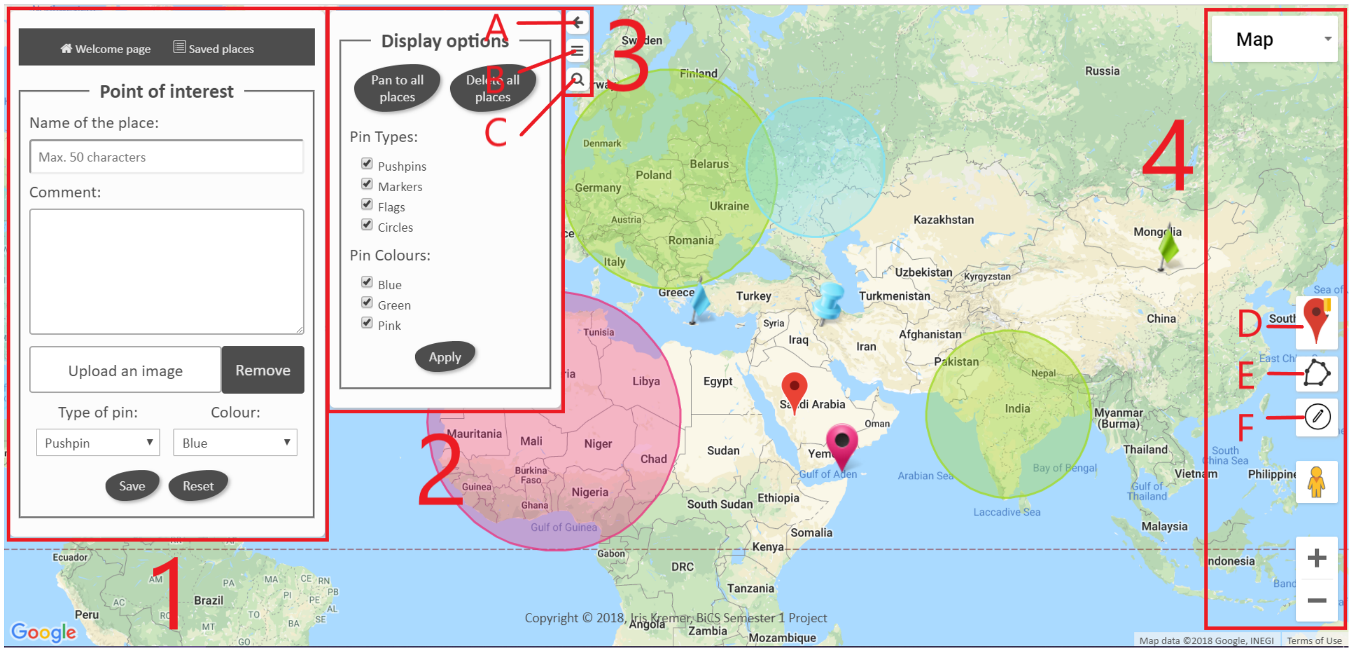


Figure 1. Screenshot of the interface with red labels.

StreetMap etc. First of all, the whole window is filled with the map and the page itself is not scrollable.

Map controls available by default were set to the right side of the window, such that the left side is free for a sidebar. This will contain all features that are not directly available on the map or through map controls.

There is actually not only one, but three independent sidebars - the form, the list of saved places and the display options -, as well as three small buttons next to them to control their display.

As illustrated in Figure 1, the sidebars and map controls are arranged as follows:

Zone 1 is filled either by the form, where users enter data to save a place, or by the list of saved places, with all saved information. On click on the small button B in zone 3, the form switches to the list of saved places, and vice versa on a second click, which saves space in the interface to see the map.

Zone 2 hosts the display options, a thinner sidebar with options to display all or only some pins according to their type and colour.

Zone 3 contains three small buttons to control the display of the sidebars. On click on button A, the first zone is hidden and with a second click, it is displayed again. Button B enables switching between form and list, as already been explained. The last button controls the display of zone 2 and shows or hides it on click, independently from zone 1.

Zone 4 houses all map controls. The default control enabling full-screen view has been disabled and removed, because in this mode, the map overlay was hiding the sidebars. On the other hand, three map controls - D, E and F - have been customized and added manually. D enables to move the pin that marks a saved place, E is for drawing polygonal shapes and F for drawing circles to be saved. Button E is disabled, because this function has not been implemented during the semester project, as it was not part of the requirements. Despite the feature of drawing and saving circles was not part of the requirements either, it still has been implemented in addition to the required features, so the button F is functional. Please refer to the assessment section to learn more about this topic.

Even though the sidebars are narrow, it does not hinder to arrange elements inside of them attractively. To have an airy design, the elements have some padding and margin inside and around them. Colours used match together and make the content clearly readable.

The form has a typical form look. A top input for the name, then a comment section for some longer text and finally a file input for images. Under the file input, some extra options are available to choose the pin type and the colour.

The layout of the list of saved places was a challenge, because it required to provide all informations about each pin, but still fit in the narrow sidebar. To remedy the lack of space, when the list is displayed, each place is represented as a box labelled with the name in the middle and the pin icon on the left, to recognize which type and colour has been chosen for the pin of the place. When there are too many saved places to fit in the space, the sidebar becomes scrollable instead of overflowing out of the page. To display more details of one pin, the user just has to click on the box of the place and it expands to show the comment and image as well. A second click on the label of the box shrinks it again to the initial box with only the name and pin icon. The Figure 2 is showing the list sidebar with one element expanded, displaying the saved information, and the scrollbar which has appeared to avoid overflow.

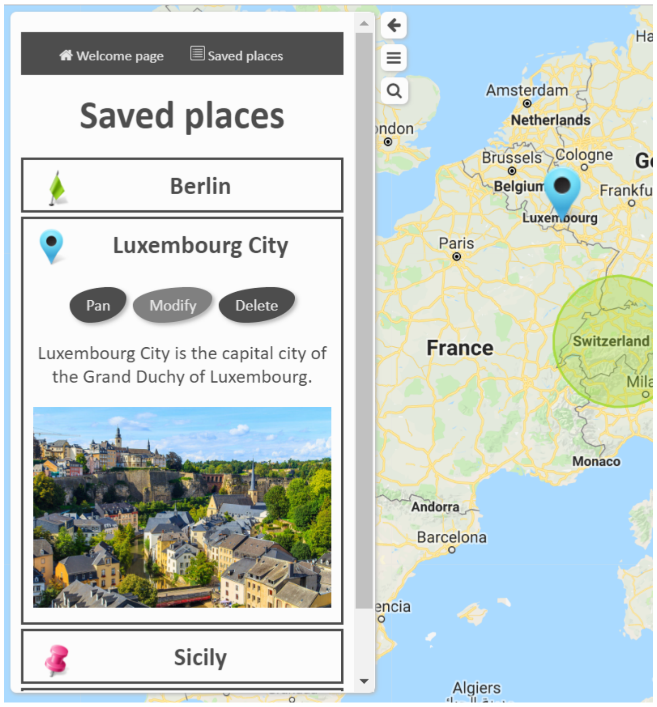


Figure 2. Screen of the list sidebar.

Besides of the look of the form and list, another aspect is the responsive character. On hover on buttons and links, the cursor and colours change. Most of the buttons even change shape. This contributes to make the page more attractive as well.

## 4.3. Production

During this project, two deliverables have been produced: The taxonomy of APIs and the web-based editing interface for georeferenced data. First, we will present the taxonomy of APIs, which is the result of research according to the requirements, with respect to the design described in the previous section. Second, we will describe the production of the interface.

### Taxonomy of APIs.

The taxonomy built during this project is shown in Figure 3. Notice that Google Maps and Bing Maps are more similar since these are big enterprises, while OSM is an open-source software which is entirely build by volunteers. This explains especially the big difference between the types of maps available for OSM (only the basic, customizable map) and the two other ones (basic map, satellite view, street view etc).

When looking at this table, there are some things to analyse about each platform.

#### (1) Google Maps

Google Maps has built many APIs targeting different usages. The first three APIs, i.e. Maps Static, Maps JavaScript and Street View, are general APIs to display maps and access basic features. The next three APIs, Directions, Distance Matrix and Roads, are centring on routes and distance calculations. The last ones - Places Library, Places API, Geocoding and Geolocation - are focussing on finding data about places, geocoding and reverse geocoding.

For this project, the API providing the adequate features is the Maps JavaScript API.

#### (2) OpenStreetMap

The OpenStreetMap API is not made to display a map and implement features, but to modify and develop directly the map and features provided by OpenStreetMap, which is totally different from all other APIs presented in this table.

Leaflet and OpenLayers are independent, also open-source APIs for OSM, which allow to access the map of OSM and to implement features, similarly to the APIs of Google Maps and Bing Maps. The main difference between Leaflet and OpenLayers is the complexity.

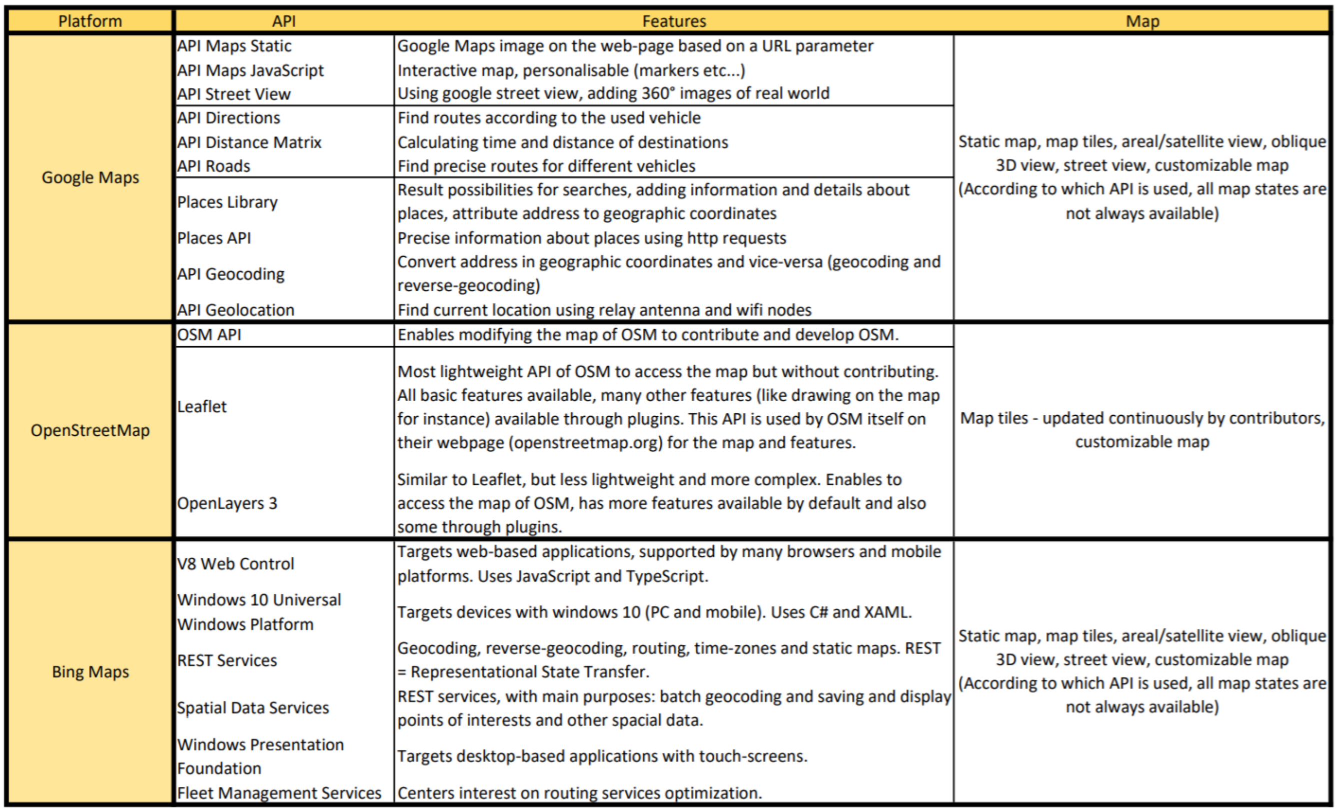
Leaflet is very light-weight because it provides few features compared to all the other APIs. But Leaflet is easily extendable using plugins. The 'lack' of default features is therefore an advantage, because it enables the programmer to use only the plugins he needs and leave out anything unnecessarily complex.

OpenLayers provides more features by default, which makes it less light-weight and more similar to APIs of Google Maps and Bing Maps. It has still plugins available, but many features are already implemented in the default API.

The OSM API is absolutely not appropriate for this project, because the aim is to create an interface to handle georeferenced data, not to build the map itself. Leaflet and OpenLayers would both suit for this project, but since Leaflet is simpler to understand, it is also the most appropriate one.

#### (3) Bing Maps

Unlike Google Maps, Bing Maps has developed its different APIs in the first instance according to the target platform (website or mobile devices), as well as the operating system (OS) used (Windows, since Bing Maps



| Platform | API | Features | Map |
|---|---|---|---|
| Google Maps | API Maps Static | Google Maps image on the web-page based on a URL parameter | Static map, map tiles, areal/satellite view, oblique 3D view, street view, customizable map (According to which API is used, all map states are not always available) |
| | API Maps JavaScript | Interactive map, personalisable (markers etc...) | |
| | API Street View | Using google street view, adding 360° images of real world | |
| | API Directions | Find routes according to the used vehicle | |
| | API Distance Matrix | Calculating time and distance of destinations | |
| | API Roads | Find precise routes for different vehicles | |
| | Places Library | Result possibilities for searches, adding information and details about places, attribute address to geographic coordinates | |
| | Places API | Precise information about places using http requests | |
| | API Geocoding | Convert address in geographic coordinates and vice-versa (geocoding and reverse-geocoding) | |
| | API Geolocation | Find current location using relay antenna and wifi nodes | |
| OpenStreetMap | OSM API | Enables modifying the map of OSM to contribute and develop OSM. | Map tiles - updated continuously by contributors, customizable map |
| | Leaflet | Most lightweight API of OSM to access the map but without contributing. All basic features available, many other features (like drawing on the map for instance) available through plugins. This API is used by OSM itself on their webpage (openstreetmap.org) for the map and features. | |
| | OpenLayers 3 | Similar to Leaflet, but less lightweight and more complex. Enables to access the map of OSM, has more features available by default and also some through plugins. | |
| Bing Maps | V8 Web Control | Targets web-based applications, supported by many browsers and mobile platforms. Uses JavaScript and TypeScript. | Static map, map tiles, areal/satellite view, oblique 3D view, street view, customizable map (According to which API is used, all map states are not always available) |
| | Windows 10 Universal Windows Platform | Targets devices with windows 10 (PC and mobile). Uses C# and XAML. | |
| | REST Services | Geocoding, reverse-geocoding, routing, time-zones and static maps. REST = Representational State Transfer. | |
| | Spatial Data Services | REST services, with main purposes: batch geocoding and saving and display points of interests and other spacial data. | |
| | Windows Presentation Foundation | Targets desktop-based applications with touch-screens. | |
| | Fleet Management Services | Centers interest on routing services optimization. | |

Figure 3. Taxonomy of APIs for web-cartography. Created based on information from: [5], [6], [7], [8], [9], [10], [11], [12].

belongs to Microsoft, or all OS). Note also that Bing Maps has some APIs which use at least partly, sometimes completely other languages than JavaScript, for example C#.

Bing Maps has no API which is really easier to use than the other ones and even its documentation is quite complex, so it seems clearly to target enterprise application development. Therefore, none of the APIs is very appropriate for this project.

**Interface.**

The very first step of the production of the interface was to choose the development environment and tools used. For web-development, the only tools needed were a programming interface and a browser to load the page.

Visual Studio has been used as programming interface in this project, because it has a nice and relatively simple interface.

Chrome has currently the reputation of being the most adequate browser for web-development. Therefore, this browser and its developer tool have been used to work on this project.

The developed interface has been tested on Safari and Firefox. To use the interface on Safari, the user needs to 'Disable local file restrictions' (in the 'Develop' menu). Then, all implemented features seem to work flawless. On Firefox, all features seem to work correctly as well, although one issue has been identified: IndexedDB uses different stores depending on the initial page loaded (index.html or map.html). Therefore, further testing and additional development is required before using this interface on Firefox.

Since the interface has been optimized for Google Chrome, it is still recommended to use the interface on this browser.

The production of the web-based editing interface is involving three parts: the choice of an API for web-cartography to be used, the choice of a storage method and the implementation of the features described in the requirements section. (Remember that the layout of the interface has been designed as detailed in the design section.)

**(1) Choice of API**

To ensure an operative interface, the choice of the API for web-cartography used is crucial.

The first part of the work to choose this API is made by building the taxonomy of APIs, giving a clear overview of provided features of each specific API. Please refer to the taxonomy to get more details about it.

One conclusion made looking at the taxonomy was that Bing Maps has not separated its features the same way on its different APIs than Google Maps, so there was no API which is adapted for this semester project. Also, it is oriented for enterprise use and has a complex documentation, so this platform was less appropriate. Therefore, it left Google Maps and OpenStreetMap to focus on, especially the Google Maps JavaScript API and Leaflet, which are the APIs providing adequate features for this semester project.

To decide between these two technologies, the best option was to try both of them out and choose the most comfortable one. The criteria for the choice were the complexity and the available documentation, because simplicity is required to be able to use the API without any prior knowledge.

The first tests were performed using Leaflet. This API is accessible by either using an URL, or by downloading the files for the API.

Even though Leaflet is lightweight and absolutely not complex, it was more difficult to understand for a beginner than expected. There is not much documentation available: aside from the list of all functions, there is only a starter guide to help users implementing the API and some spare documentation pages helping on advanced features. The rest had to be figured out only using the list of functions or by searching on the web, which may be manageable with some experience, but difficult for an inexperienced programmer. It took time to implement the most basic features like moving a marker on click on the map and open a popup on it.

Secondly, the Google Maps JavaScript API has been tested. Having already gained a bit of experience with Leaflet, reproducing features that had been implemented using Leaflet with the Google Maps API was much more simple. But since this was related to the experience with Leaflet, some more testing had to be performed before deciding which API would be used.

The Google Maps JS API is globally more complex than Leaflet and has some more features, like a satellite view and street view for example. Also, this API can only be accessed directly on their server using an URL source, and has to be loaded at the very end, after all other CSS and JS files, because the URL contains a callback function that initializes the map.

The big positive point of Google Maps is their rich documentation. It does not only have a list of functions and a starter guide, but also details many features and provides a large number of examples and sample codes.

Leaflet may be less complex, but thanks to the detailed documentation provided for the Google Maps API, it is easier to use and understand for beginners in programming. Since the most primordial criteria was to understand how to use the API, Google Maps has been chosen to be used for this project. Additionally, since Google Maps provides more features which are easy to understand with the explanations, this choice may have opened to more possibilities for implementing features.

**(2) Choosing storage method**

Without any knowledge of servers, it would have been one more difficulty to achieve the objectives of the project by trying to save information using servers. Therefore, storage on the clients side has been privileged.

There are many possibilities to store data locally, but most of them like JavaScript variables, cookies and HTML5 web storage are not adequate for this project, either because they are not persistent (JS variables), or because these methods do not provide enough space to save data. There are only two options which are worth consideration for this semester project: Web SQL Database and Indexed Database API.

Web SQL Database is a structured database using SQL to save and recall data. IndexedDB is an API for a local database using its own language.

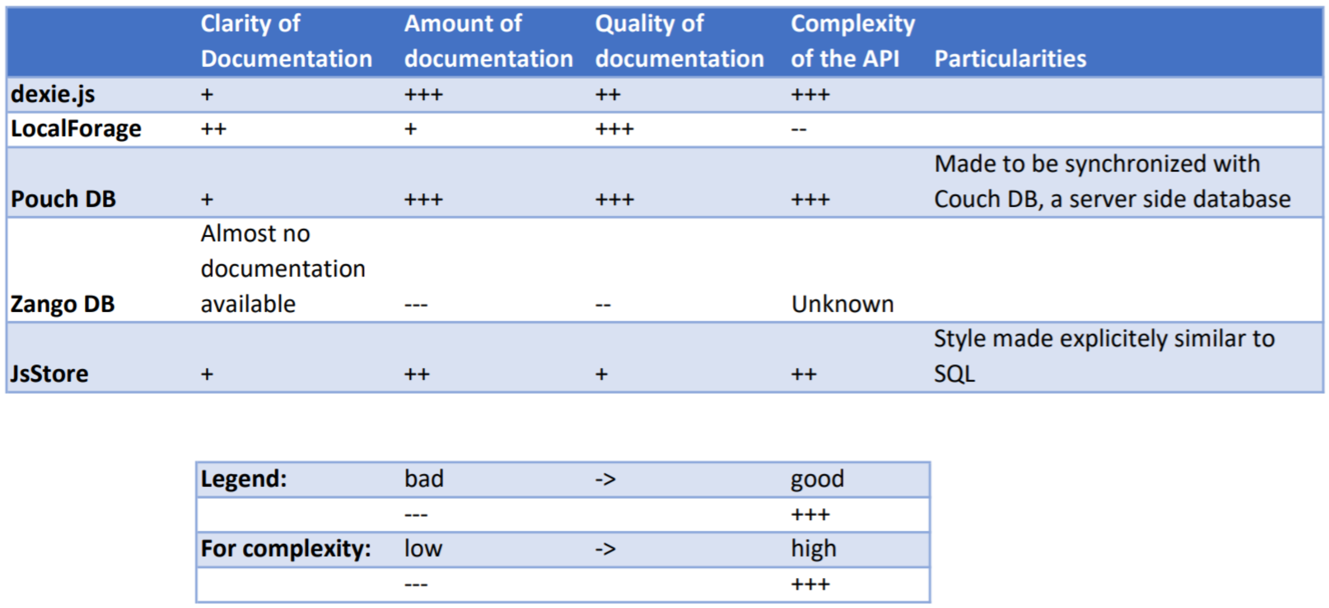
The main difference between Web SQL Database and IndexedDB is the way of storing data. To use Web SQL Database, a specific structure to store the data has to be defined in advance (each table containing objects of the same nature with predefined attributes for the columns). IndexedDB has a simpler structure that assigns a unique key to each object to recall the data and no further constraints is set for the object and its attributes.

In the case of this project, none of the methods is especially more appropriate than the other one, because all objects stored in this prototype are of the same nature. But actually, Web SQL Database is obsolete since 2010. It is not appropriate to use a deprecated tool, so it has been decided to use IndexedDB, which is more recent, currently well maintained and supported by all browsers. [13]

Even though IndexedDB is a very modern technology, it has one big disadvantage, which is that its programming language is low-level. Learning this language is therefore time-consuming and difficult. To avoid having to learn it, there exists JavaScript APIs for IndexedDB. This option is very convenient, since JavaScript is already required to use the API of Google Maps.

On developer.mozilla.org [14], they propose five different JS APIs for IndexedDB. A small comparison has been made of each of them using their documentation pages to get information. The result has been summarized in the table of Figure 4 in order to find out which one is the most suitable API to be used.

As pictured in the table, PouchDB is an API using IndexedDB for local storage and enables the synchronization of data with CouchDB on servers. For instance, it allows



| | Clarity of Documentation | Amount of documentation | Quality of documentation | Complexity of the API | Particularities |
|---|---|---|---|---|---|
| **dexie.js** | + | +++ | ++ | +++ | |
| **LocalForage** | ++ | + | +++ | -- | |
| **Pouch DB** | + | +++ | +++ | +++ | Made to be synchronized with Couch DB, a server side database |
| **Zango DB** | Almost no documentation available | --- | -- | Unknown | |
| **JsStore** | + | ++ | + | ++ | Style made explicitely similar to SQL |

| | | | |
|---|---|---|---|
| **Legend:** | bad | -> | good |
| | --- | | +++ |
| **For complexity:** | low | -> | high |
| | --- | | +++ |

Figure 4. Taxonomy and evaluation of IndexedDB JavaScript APIs. Created based on information from: [15], [16], [17], [18], [19]

programs storing data on a server database to work for some time in disconnected mode as well. This can be very useful for advanced programs, but not in this case. JsStore has been build explicitly to have a syntax which is similar to SQL, for programmers used to SQL. In this case, this is also not very appropriate. Dexie.js is a very exhaustive API, with a big amount of features, and therefore too complex for a beginning programmer. ZangoBD has not enough documentation to even know if it is complex or not. Since documentation is essential to understand how the API works, this choice was also eliminated.

It left only LocalForage, which seemed at first sight not too complex. It has not much documentation, but still enough to understand well the basis of this API. It was the most appropriate API for this semester project, so this one has been used.

**(3) Implementing features**

Different types of features have been implemented in the interface to fulfil the requirements:

1) Show/hide sidebars features - using original JavaScript language only
2) Map features - using Google Maps API function calls
3) Storage and display features - LocalForage and Google Maps API function calls

Show/hide sidebar features:

As explained in the design section, the interface contains three little buttons next to the sidebars with the functionality to control the display of the sidebars, to avoid overloading the page.

The feature bound to button A is checking whether either the form, or the list is displayed. If none of them is displayed, it sets the display style of the form to 'block' to show the form. Else, it sets both the display style of the form and the list to 'none'. Similarly, the feature of button C is checking whether the options are displayed or not and changes the display style from 'block' to 'none' or from 'none' to 'block' according to the original state.

The feature of button B is looking at which one of the list and the form is displayed. If none of them is displayed, no further action is executed. If one of them is displayed, the display style of this element is set to 'none' and the other one to 'block', to switch from the first to the second one.

Map features:

The map is loaded after having charged the HTML page and all other scripts, using the callback of the Google Maps API URL which calls in our case the function 'initMap()'. This function had to be defined in the main script. It creates in the first instance the map tiles and sets the center, zoom and default controls of the map. If the user allows the browser to know its location, the map will center to the current location and sets the red marker on it. Otherwise, it centers and sets the marker by default on Luxembourg City. Also, 'initMap()' calls two other functions which recreate pins for places saved previously and the corresponding list. These features will be discussed later in this section.

One of the foundation features of the interface is the ”moving marker” feature. This feature moves the red marker, which is set by default on load of the map, on click on the map to the clicked position. It is bound with the marker control and activated by default on load of the map.

This feature also sends each time the new coordinates of the marker to a hidden input in the form, such that the coordinates are accessible when the user saves a place.

Some display options available in the option sidebar are also bound to map features:

The button ’Pan to all places’ is bound with a feature that takes all coordinates of the pins and sets zoom and center of the map such that all of them are visible.

On click on the button ’apply’, the checklist information is parsed and pins are displayed or hidden according to the choices checked by the user. This feature enables to visually completely separate the pins by type and/or colour.

Storage and display features:

The most important feature of the interface remains the saving feature. On click on ’Save’, the function ’savePlace()’ is called. This function takes the coordinates, name, comment, image, pin type and colour entered by the user and bounds them together as one place object. This object is then saved using LocalForage with the name of the place serving as key. The function will first check whether the name is empty or already used for another place, and returns an error message for the user, if it is the case.

After having successfully saved a PoI, a pin with the chosen type and colour will directly be set on the exact place on the map. Bound to this pin, an ’infowindow’ (term used by Google Maps for ’information window’) is created, which opens on click on the pin, and contains all informations saved. Also, the PoI is added in the list by calling the function ’createDiv()’. Lastly, the form is reset to its initial state.

The function ’createDiv()’ creates a div element with all saved information about a place, formatted well with CSS, and binds JS features making the div extend and shrink on click on the name. This div is then appended to the list of saved places. This is a good example of HTML elements created dynamically using JavaScript. This function is called after each place is saved, as well as on load of the page, where it is called once for each key through another function: ’createList()’.

Another feature which requires to recall the objects from the database is the ’displaySavedPins()’ function, which is called on load of the page to display all previously saved pins. This function takes the array of all keys, parses this array and recalls the object for each key. Then, it takes the attributes of each object to create a pin of the chosen type and colour, which shows an infowindow on click with the saved information.

Finally, two features to delete the places have been implemented. The first one, ’deletePlace()’, is called using a button ’delete’ in the infowindow of each pin and at the top of each element in the list. On call, this function takes the PoI selected (i.e. the place to which is bound the infowindow or the list element containing the button pressed) and removes the corresponding object from the database, the element from the list and the pin from the map, such that the place is completely deleted.

The second feature is ’clearAllPins()’, bound to the button ’delete all places’. When pressed, this function will remove each pin of the map, clear the list and clear the database.

## 4.4. Assessment

This sections answers the question whether the requirements set for this project have been reached or not.

**Taxonomy of APIs.**

The requirements for the taxonomy were first to make a founded choice among the available APIs for web-cartography and decide which ones would be presented. This choice has been made in the design part of the project and selected three platforms to present.

Then, the taxonomy has to present all different APIs from each platform. For each API, the main functionality is explained briefly, such that its purpose is clearly stated. Finally, it has been explained which of the APIs could be a suitable choice to be used in this semester project and why.

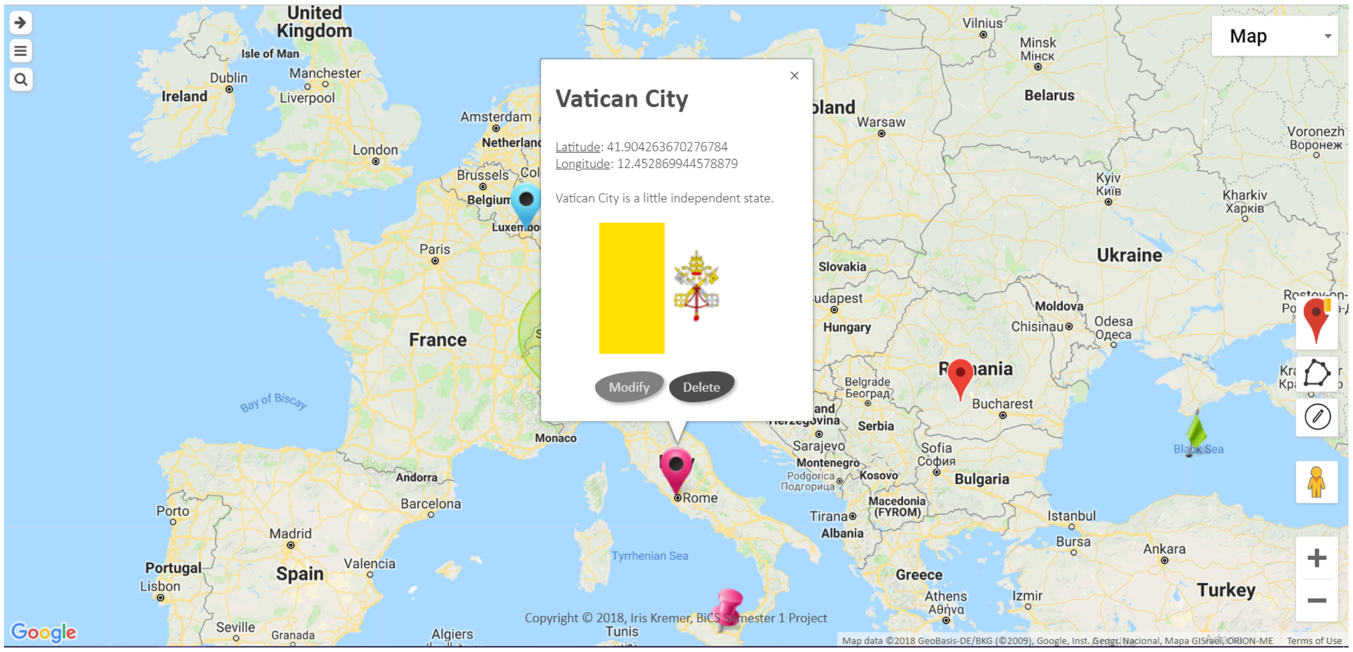
Therefore, the delivered taxonomy and the presentation made of it in this paper fulfil the requirements set for this taxonomy.

**Interface.**

The requirements for the design of the editing interface targeted a user-friendly, attractive and minimal interface. The design has been described precisely in the design section.

The interactive and aired layout built for the web page is attractive and invites users to use the interface. Through the separation of the features in three sidebars - each of them displayed or hidden independently - and the map controls, the page is structured and not overloaded. Figure 5 is showing a picture of the interface when the two sidebars are hidden. We can see that the interface is very clear and the user can focus on the management of his PoI directly on the map.

Also, the separation of the features in the different sidebars has been chosen wisely, such that they are easier to find and accessible. Finally, this structure makes it possible to have all features available on the same page.


Vatican City

Latitude: 41.904263670276784
Longitude: 12.452869944578879

Vatican City is a little independent state.

Modify
Delete
Map
Copyright © 2018, Iris Kremer, BiCS Semester 1 Project
Map data ©2018 GeoBasis-DE/BKG (©2009), Google, Inst. Geogr. Nacional, Mapa GISrael, ORION-ME
Terms of Use
Google

Figure 5. Screenshot of the interface with all sidebars hidden.

Therefore, the chosen layout fulfils all requirements for the design of the page.

The editing interface for georeferenced data also had many functional and non-functional requirements concerning available features.

As described in detail in the production section, all features required for the interface to work properly have been implemented. Also, features which were not necessary for the interface, but still enhanced its functionality to handle georeferenced data, were implemented as well.

A very strong and convincing argument to assess the successful implementation of these features is that, after having finished implementing the required features, other, non-required and slightly more advanced functions have been implemented as well in the code to continue improving and developing the interface.
Concretely, the possibility to save areas of interest (AoI) has been added. These AoI are an additional category of places, namely circles, for which the user can choose the radius and center manually. A new 'saveCircle()' function has been implemented and some features have been recreated for circles, when it was more convenient than adapting the equivalent function for pins. But several functions already existing for PoI have been adapted, such that they work both for pins and circles. This shows that the quality of the code was high enough to facilitate the implementation of new, different features.

**Lacks.**

Remember that the editing interface for georeferenced data remains a prototype. The code certainly could be further optimized in many places.
Also, the interface is not secured. For example, a user who looked at the code knows that writing HTML code into the name and comment fields could partly destroy the layout of the infowindows and list elements. However, since these saved informations are only saved locally and available for the user himself, breaking the website like this would only affect himself and is therefore pointless.

## Acknowledgment

I, Iris Kremer, would like to thank my project academic tutor (PAT) Jean Botev for instantiating this interesting project, as well as for his time spent for me and help during the whole project. I learned a lot about the topic of APIs and the importance of georeferenced data in real life, as well as a rigorous work method.

I would also like to thank the BiCS management and education team for establishing the concept of semester projects in this bachelor, which is a great and rare opportunity for us students to discover simultaneously how to organise ourselves for project work and fields of Computer Science that we may not touch on in regular courses.

Finally, I owe thanks to my family, i.e. my parents, my grand-parents and my sister, for their support and love.

## 5. Conclusion

In this paper we have presented a bachelor semester project, which had as objective to explore how georeferenced data can be created and managed by developing a web-based editing interface for georeferenced data. This task has been fulfilled with success and was a great introduction to GIS and web-based applications.

The created interface enables to save points of interest (PoI) and circular areas of interest (AoI) in relation with information entered manually, and to manage these places and areas - including visual categorization using different pins and colours, viewing the information in an information window (infowindow) or in a list and deleting saved places.

In the future, the editing interface could be expanded with the possibility to save polygonal AoI. The structure of the code and the saving procedure are already adapted for this possible further development (map control for polygons, saving PoI and AoI using indexes - which suggests that all saved objects are on a same level -, 'type' attribute of saved objects, etc).

Furthermore, a feature could be added to modify information or even the location of a saved place. In the prototype, buttons 'Modify' are already foreseen in the infowindows and elements of the list, such that the feature can be bound to these buttons.

Finally, the interface could be hosted on a server, instead of locally. Also, the PoI and AoI could possibly be saved on this server, such that they are accessible from anywhere. Adding this functionality would imply to implement user accounts as well. A logical step would be to allow the users choose if (some of) their saved places should be visible only by him-/herself or by all users.

## References

[1] BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global. University of Luxembourg, BiCS - Academic Bachelor in Computer Science (2017).

[2] Georeferencing? (Small article on spacial data and georeferencing) December 19, 2013. Available online: http://www.gisresources.com/georeferencing-2/

[3] By Rolin Zumeran. The History of APIs and How They Impact Your Future. June 07, 2017. Available online: https://www.openlegacy.com/blog/the-history-of-apis-and-how-they-impact-your-future

[4] Apple Maps API documentation. https://developer.apple.com/maps/

[5] OpenStreetMap API documentation. https://wiki.openstreetmap.org

[6] Google Maps API documentation. https://developers.google.com/maps/documentation/

[7] Leaflet documentation. https://leafletjs.com

[8] OpenLayers documentation. https://openlayers.org

[9] Bing Maps APIs documentation. https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api

[10] Mary Branscombe. How Microsoft Bing Maps API Features compare to Google Maps. August 16, 2018. Available online: https://codematters.online/how-bing-maps-api-features-compare-to-google-maps/

[11] Farkas Gábor. Applicability of open-source web mapping libraries for building massive Web GIS clients. July 01, 2017. Accessible online: https://link-springer-com.proxy.bnl.lu/article/10.1007/s10109-017-0248-z?pds=24122018155326103920103985619875030

[12] Khitrin M.O. Comparison of JavaScript Libraries for Web-Cartography. 2017. Available online: https://vestnik.susu.ru/ctcr/article/viewFile/6619/5590

[13] Craig Buckler. HTML5 Browser Storage: the Past, Present and Future. Oktober 09, 2013. Available online: https://www.sitepoint.com/html5-browser-storage-past-present-future/

[14] IndexedDB documentation by Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[15] LocalForage documentation. https://localforage.github.io/localForage

[16] Dexie.js documentation. http://dexie.org

[17] ZangoDB documentation. https://github.com/erikolson186/zangodb

[18] PouchDB documentation. https://pouchdb.com

[19] JsStore documentation. http://jsstore.net

## 6. Appendix

Sources of the images used in the developed interface:

1) Blue, green and pink pushpins, markers and flags: http://www.icons-land.com/vista-map-markers-icons.php

2) Set marker control icon: https://www.kisspng.com/png-google-map-maker-google-maps-google-logo-map-marke-1103825/

3) Draw polygon control icon: https://m.veryicon.com/icons/system/android-1/maps-polygon.html

4) Draw circle control icon: https://www.shareicon.net/edit-draw-pen-circle-pencil-write-stroke-108587

5) List and map navbar icons: https://www.onlinewebfonts.com/