***Software Engineering for Geoinformatics, 2020***

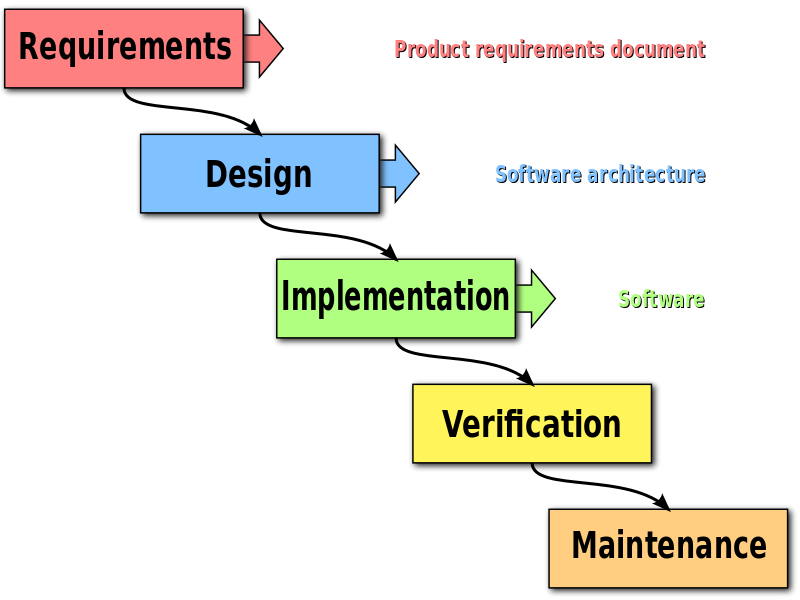
|  |  |  |
| --- | --- | --- |
| **Design and Test Plan Document**  **GROUP MEMBERS** | | |
| **Name** | **Student ID** | **E-mail** |
| Felix Enyimah Toffah | 10647752 | felixenyimah.toffah@mail.polimi.it |
| Sahar Gholami Bourani | 10718755 | sahar.gholami@mail.polimi.it |
| Phesto Osano | 10714566 | phesto.osano@mail.polimi.it |
| Arli Ardiandy Nurseramika | 10693181 | arliardiandy.nurseramika@mail.polimi.it |

**Lecturer**

**Professor Di Nitto Elisabetta**

Delivery date: May 13, 2020

# Architectural Design



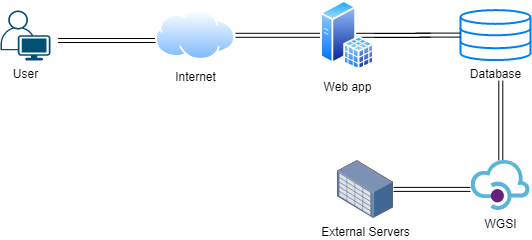
## 1-High level components and their interaction

High-level design (HLD) explains the architecture that would be used for [developing a software product](https://en.wikipedia.org/wiki/Software_development). The architecture diagram provides an overview of an entire system, identifying the main components that would be developed for the product and their interfaces.

The general architecture of the system, in backend and frontend, here includes three main components. user, database, API.

User or client which employs a host like HTML browser to connect to the web app through the internet.

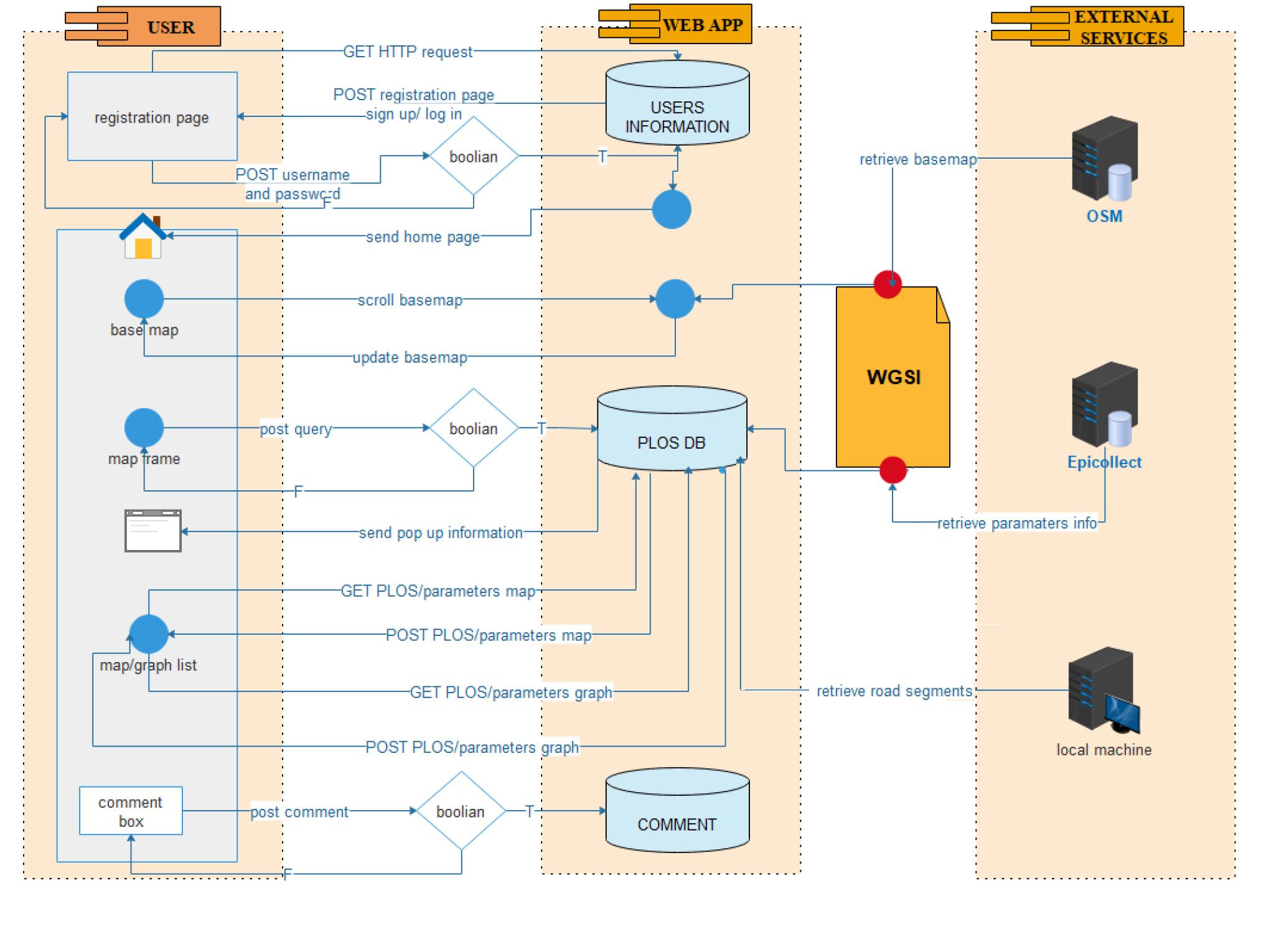
The web app retrieves databases from different external services(which here are Eppicollect and OSM) through API . Here is also called WGSI which is the one with python. The connections between these main components are illustrated below.



### Figure 2.1: High level components of the system and their interactions

## 2-Component view

A general overview of the whole components and connections between them is provided in the figure 2.2. the web app deploys several databases. First is the users’ information in which the information for the registration is stored in such a way that each user has its own unique id . The users’ comments are stored in the comment database, maintaining user id and the time of the post. Third and the main database is PLOS DB. Using this database the web app manages to do its main function which is exploring through maps and graphs using a dataset retrieved dynamically from an external service- which is here Epicollect platform. Let us not forget that the auxiliary platform OSM boosts the visualization process. In order to connect and retrieve data from the external platforms WGSI manages API. in other words, interaction between the DBMS of the webapp and the external services conceived by WGSI. The use of an API for retrieving data is functional to the fact that if new data is collected by the Epicollect, the web app is capable of showing the new data without manual intervention. furthermore an auxiliary static data is needed (which is here the road network of the area), this is included in the web app using PostgreSQL. So using this database in the backend the app can exploit it for auxiliary data.



### Figure 2.2: A general overview of the components

So depending on the request of the user, the app responds by referring to the specific database.

following are the main components and actors of the system:

**User/client**: a user requests through POST and GET action to the webapp. This interaction has been happening in an internet bed.

**Database:** the data bank of the web app which stores all data needed for the app function. It includes three main databases. two of them are created by the users, including “User Information DataBase” and “Comment database”. The other one which stores data retrieved from external services.

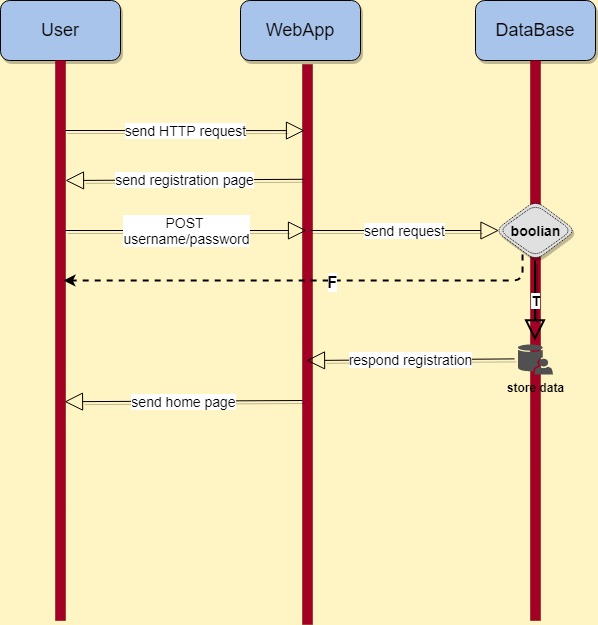
**WGSI:** this interface provides interactions between the web app and external services. It consists of API of different external services including OSM API, Epicollect API and Postgres to interact with the local machine.

**boolean tool:** some requests need to be diagnosed whether following the requirements or not.

**External Services:** these include several platforms which provide the main database of the web application. They are Epicollect which contains the parameters for PLOS calculation, OSM provides the basemap of the map frame. Also the data retrieved from Postgres.

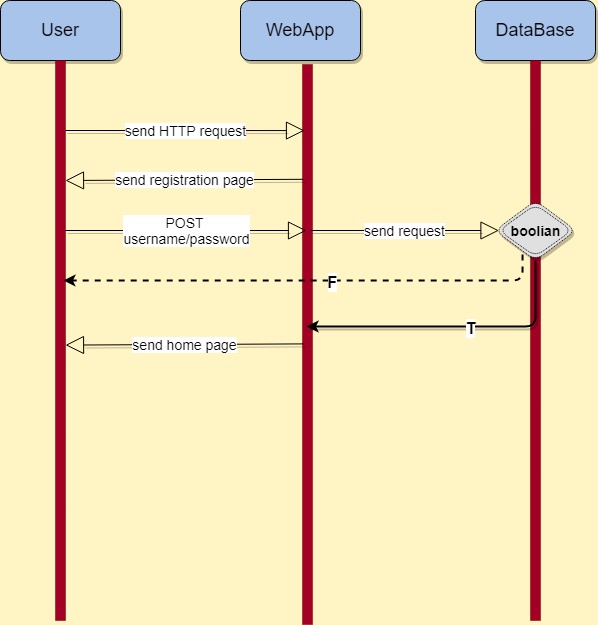
## 4-System Sequence Diagram (SSD)

In this section we follow the behavior of each component of the system for each use case. system sequence diagram (SSD) is a [sequence diagram](https://en.wikipedia.org/wiki/Sequence_diagram) that shows, for a particular [use case](https://en.wikipedia.org/wiki/Use_case), the events that external actors generate, their order, and possible inter-system events.System sequence diagrams are visual summaries of the individual use cases.



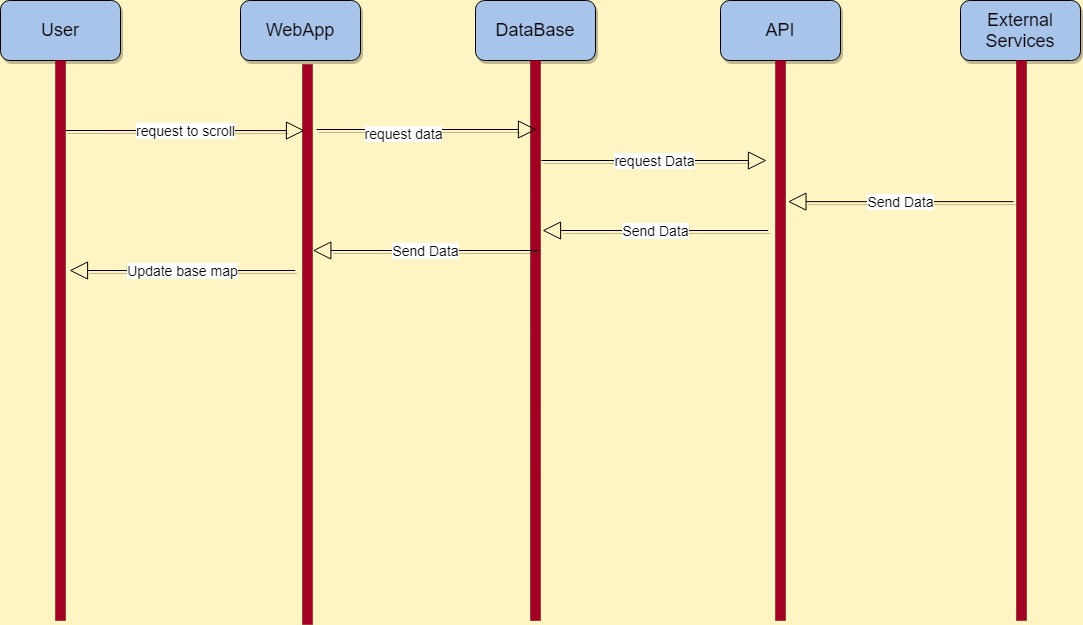
### figure 2.3: sequence diagram of Sign Up

As shown in the figure 2.3 the registration starts with the user sending the HTTP request through an internet browser. then the web app responds by sending the registration page in which the user is able to sign up or log in. In case of latter the sequence illustrates below. the user requests for sign up by posting the user and password or any other information required for registration. so the web app receives the request and searches in the database to check if it is available or if it follows the regularization for user and password. In order to do that a boolean tool analyzes the request. so if it does not follow the regularization codes or the username is not unique the boolean tool responds back invalid username or password. unless the data posted would be stored in the user information database. so the webapp sends the user the homepage.



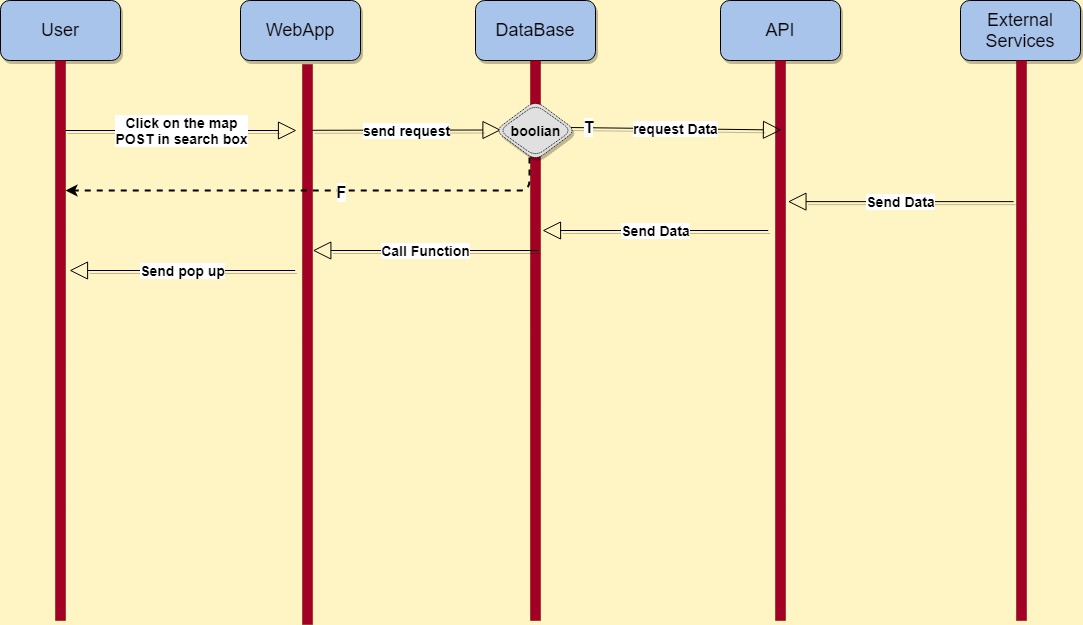
### figure 2.4: sequence diagram of Log in

In case the user has been registered before, he post the username and password in login part and again if it is valid the home page will be sent to the user.



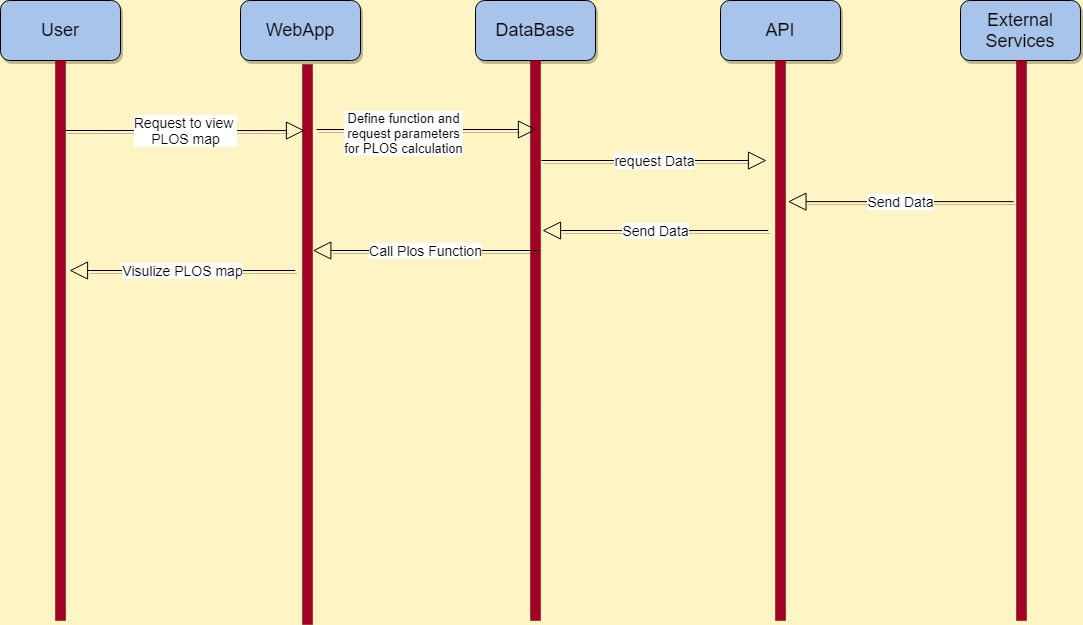
### figure 2.5: sequence diagram of Scrolling

The user scrolls the map to a new position and the app responds by providing the user with an updated map. The interactions between components here are: the request of the user for moving the map is transfers to the WGSI and the OSM API so the new map frame data is sent to the web app and it will be updated in the UI.



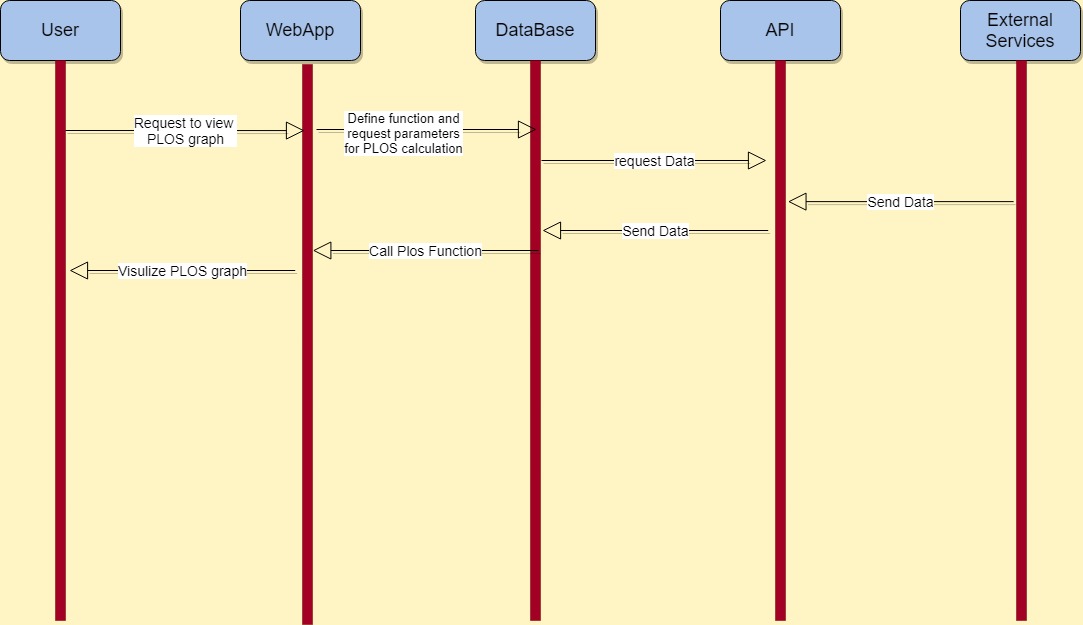
### figure 2.6: sequence diagram of Query

The user is able to query in two ways by location or by attribute. In the first way he clicks on a point or road on the map while in the second way he writes the segment number in the search box. In each case the request for the information of the segment is sent to the webapp. The request is then analysed in a boolean tool to check if it is available in the database. If so the request will be received by the database unless the invalidation message returns to the user. The database interacts with Epicollect API through WGSI. so the data is sent to the webapp internal database which then appears as a pop up for the requested segment.



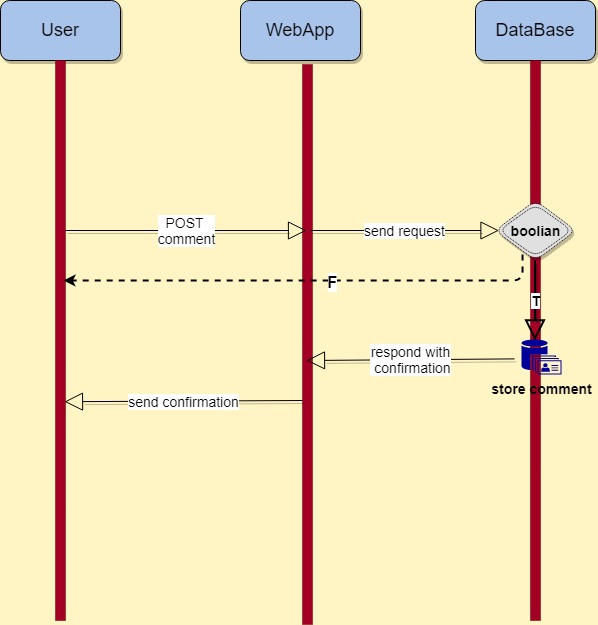
### figure 2.7: sequence diagram of map visualization

The user requests a view of the PLOS map of the area and the app provides a pop up with the PLOS of the desired area as shown below. the interactions between components are illustrated in the figure below. starting from the user sending the request the webapp retrieves the parameters needed for PLOS calculation from API then calculates the PLOS and returns the result as a map to the user.



### figure 2.8: sequence diagram of Graph Visualization

The sequence of graph visualization is the same as map visualization but the difference is the way PLOS is plotted.



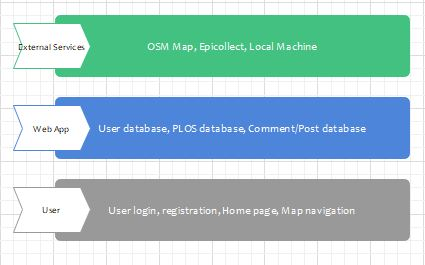
### figure 2.9: sequence diagram of Posting Comment

In order to comment on the app the user first needs to post a comment and send it to the webapp. The characters of the comment are defined beforehand so here they are put in a boolean tool and in case of being valid will be stored in the comment database, along with username and time of commenting.

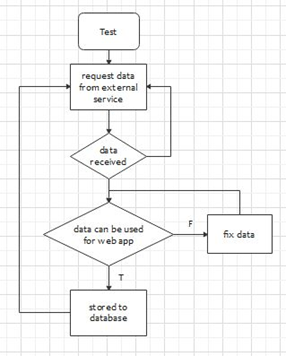
## 5-Component interfaces

TEST PLAN DOCUMENT

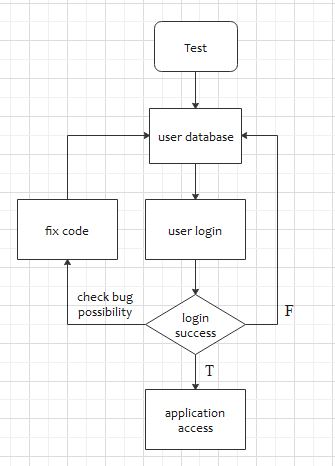
This part of the test plan we consider to divide it into three part, which is the external services, the web app, and the user. In each part we divide it to a smaller tasks so it can be evaluated correctly for the development of the web application. Dividing to a smaller tasks we can know directly if there are any problem with the code or the data so it can be fix immediately. By doing this we can keep track of the web application development.



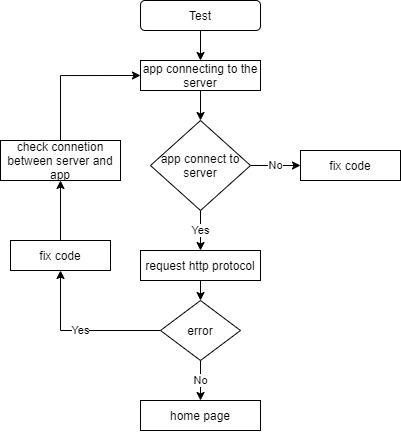
For the external service we need to make sure that the data we will receive is correct and can be used in the web app. There are three source of external services that we use, map from OSM, data collected from epicollect website and calculated data from local machine. For each service we evaluate the data received, if the data could be used for the web app then it will be stored to the database. Map from OSM will be received by requesting the map to the OSM website. In the database we create a separate table for PLOS and its parameters, this data are the one that will be used whenever user querying from the web app.



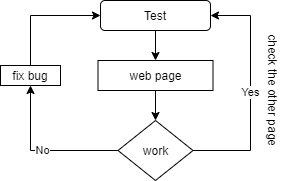
In the web application section we create a different database table for each information we want to use. On the user information database we stored all the username and password of the user so whenever a user try to login to the web application the server will look into the database if the username exists. To check the validation of this we create a fake user database and check the login process. If the login validation is working properly then we can use a real user database and if there are some error in the validation we check the table in the database also check the code if there is a bug. For the PLOS table database to test it we put a fake calculated data from local machine and check the querying process of the user. In this step the expected outcome is to be able to show all the information queried by the user. In the comment/post table database we create an empty table because it is something the user post to the database so when a user put comment on the web application the comment will be automatically stored into the comment/post table database. To test this we try to make a random comment/post from the application and check if the comment is stored correctly inside the database.



On the last part which is the user part, we try to make sure that the interface of the web application can work properly by checking every page that we have to load the page as we wanted. First checking the connection between the web application and web server connection, if there is a connection error we can fix it immediately then can proceed to evaluate each pages on the web application. We separate the page on the web application in to several pages; home page, login page, registration page, and map navigation page.



check connection between web application and server



evaluating each web page