***Software Engineering for Geoinformatics***

|  |  |  |
| --- | --- | --- |
| ***Spatial Visualization Analysis (SpaVia)***  ***Web Application***  **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**

**Summer 2020**

# Chapter2: Design Document

The developing web app is retrieving data from two platforms. Which are now called pre-existing references. First is data retrieved from Epicollect5 platform. It is a geo-localized mobile data-gathering platform

Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system. In the model of the software development process, architectural design is the first stage in the software design process. It is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them. The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components. As shown below in software engineering after Requirement Analysis (RASD document in the first chapter) the next step is to the software architecture.

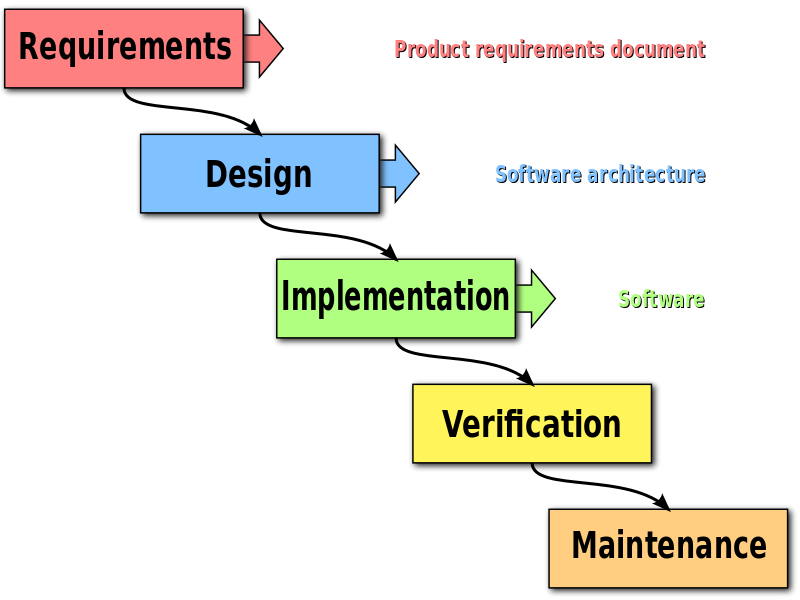


Figure 2.1: Software Engineering Waterfall Process

2.1-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, the web app (including python script and data base), 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 Epicollect5[[1]](#footnote-1), PostgreSQL[[2]](#footnote-2) and OSM[[3]](#footnote-3)) through API. The connections between these main components are illustrated following.



Figure 2.2: High level components of the system and their interactions

2.2-Component view

A general overview of the whole components and connections between them is provided in the figure 2.2. Looking at DBMS, The web app deploys several databases. First is the local database which stores users’ data (in which the information for the registration is stored in such a way that each user has its own unique id) and users’ comments (maintaining user id and the time of the post). These data are stored in SQLlite locally using SQLalchemy module, so it can be retrieved later.

Regarding the goal of the web app, which is visualizing static and dynamic analysis, second and third database are defined. Here it is needed to be explained that there are two kinds of database retrieved from local or external database. One is collected data available in Eppicollect5 and it contains some variables of the roads features. It consists 1917 Points of eight area in Milano City. The other is locally stored PLOS database which contains not only the variables of the roads features but also the PLOS value calculate using variables. The latter includes 20 road segments data.

Using the collected database the web app enables the user to explore through the map and visualize dynamic graphs of variables. Let us not forget that the auxiliary platform OSM boosts the visualization process. Interaction between the DBMS of the web app and the external services conceived by REST API. In order to do dynamic visual analysis, the collected data retrieved from Epicollect5 platform, is stored in local database. Then deploying flask-sqlalchemy module in python script the data is retrieved in python script, we are able to dynamically plot it deploying bokeh module.

Second database is PostgreSQL database. The PLOS data available in local machine, is then stored in PostgreSQL using sqlachemy module. Then the stored data is retrieved to the web app deploying Psycopg2 module in python script. The static analysis is then done using seaborn module in Python.

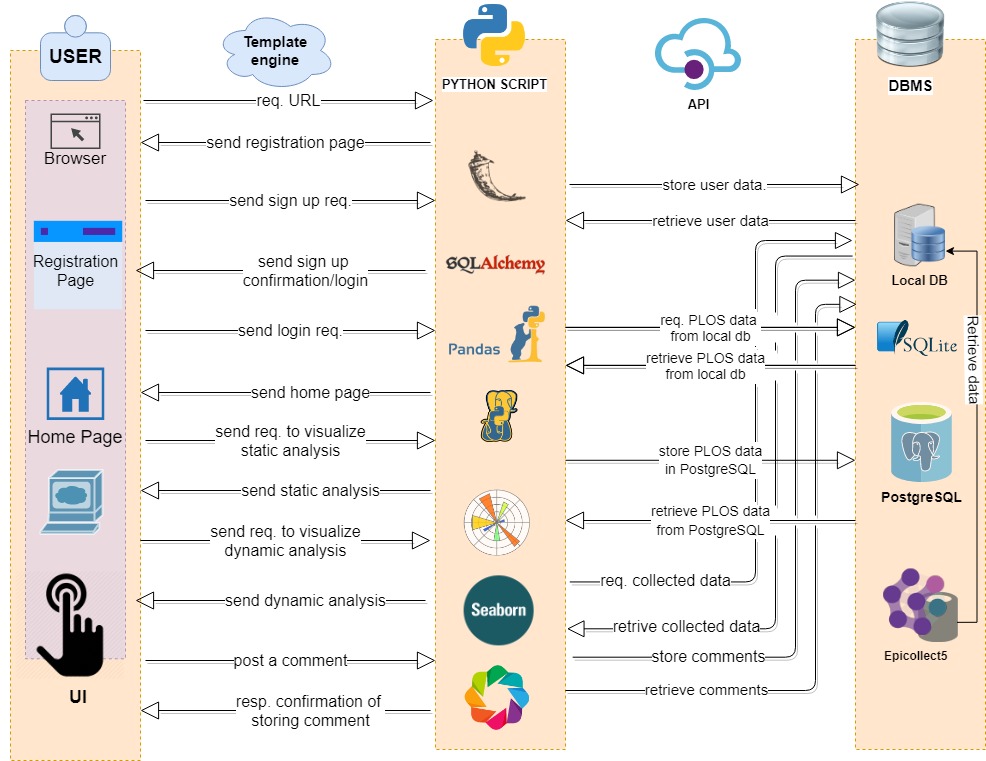


Figure 2.3: A general overview of the components

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. The user interact with web app through browser UI and the web app interact via template engines with the user.

**DBMS:** Data Base Management System of the web app is the data bank which stores all data needed for the app functionality. It includes three main databases. Collected data from Epicollect5 stores locally, so local database includes user data, comment data, and collected data retrieved from Epicollect5. Then is PostgreSQL which stores PLOS data.

**Python Script**: is the main script including functions, libraries, modules, engines, connections to DMBS.

**External Servers:** these include servers which provide the main database of the web application. In this case they are Epicollect5, which contains the variable of the road features, PostgreSQL, which stores the PLOS data, and OSM provides the basemap of the map frame for Bokeh dynamic analysis.

2.3-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.

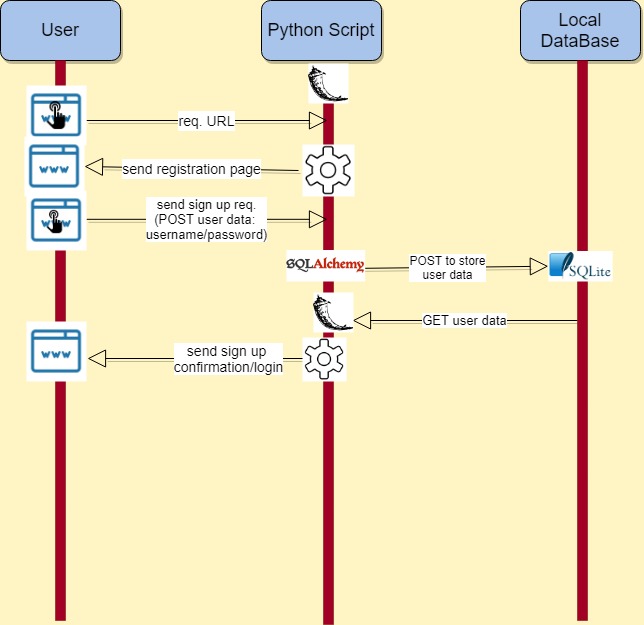
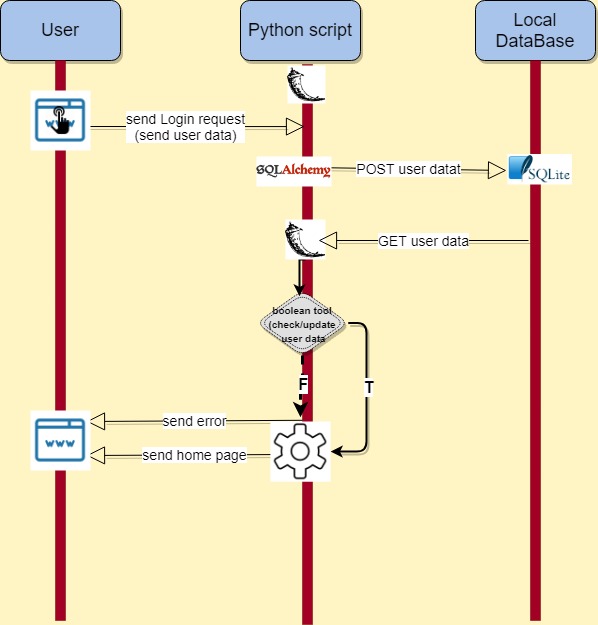


Figure 2.4: sequence diagram of Sign Up

As shown in the figure 2.4 the registration starts with the user sending the HTTP URL request through an internet browser. In the backend, when python script receives the request it connects to template engine and responds by sending registration page to the user through browser in which the user is able to sign up or log in. The user requests for sign up by posting the user and password or any other information required for registration and sends it through browser to the web app. In this stage using sqlalchemy module, the web app request to POST the user data to be stored in SQLite locally so then GET it back in flask python connects to template engine and send the sign up registration/login page.



### Figure 2.5: sequence diagram of Log in

In case the user has been registered before, he/she posts the username and password in login page and sends the login request. In python calling sqlachemy it POST the user data to the sqlite and GET it back. In flask, using this Boolean function if the data was available connecting to template engine, the web app send the home page, if not it sends error in a same way.

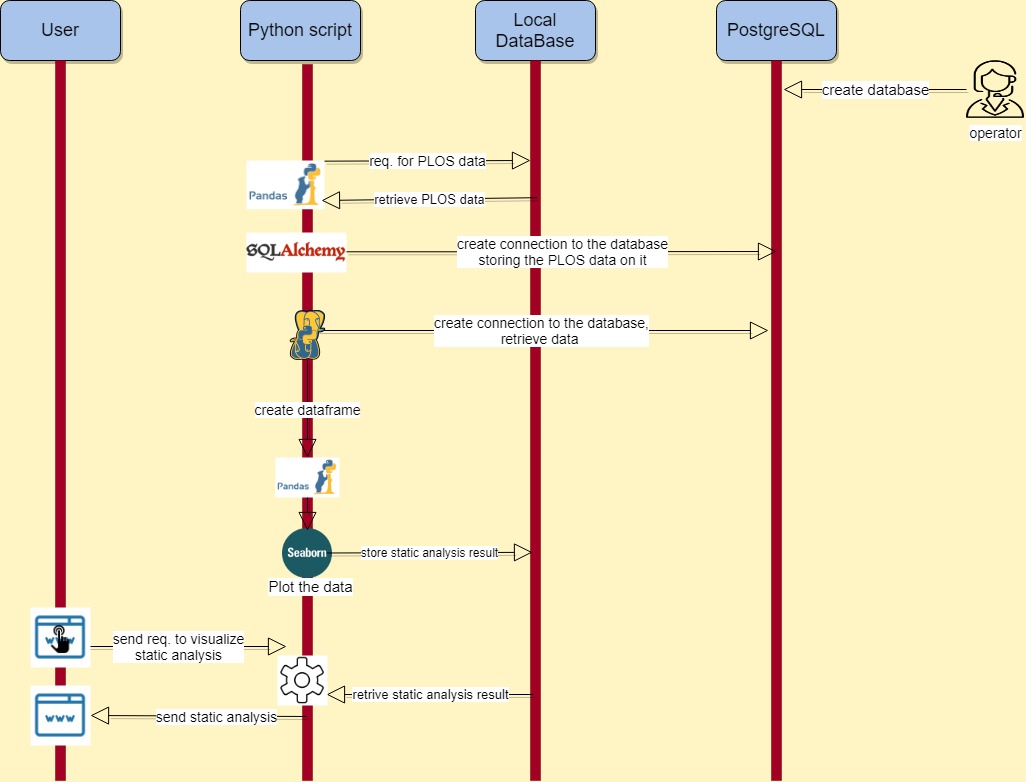


Figure 2.6: sequence diagram of static analysis visualization

In order to static analysis visualization, some interactions need to be done in the back end. The data is PLOS data which is available locally. To have the data on the fly it is stored in PostgreSQL database. So first a new database is created in PostgreSQL. Then using Pandas Python retrieves the PLOS data locally, afterwards it deploys SQLalchemy to connect to the database in PostgreSQL, creates table and stores the PLOS data on this table. Back in Python, pandas create data frame out of the table, and Seaborn plot static graphs and stores them in local database. When the user requests the static analysis visualization, template engine respond the garphs.

### 

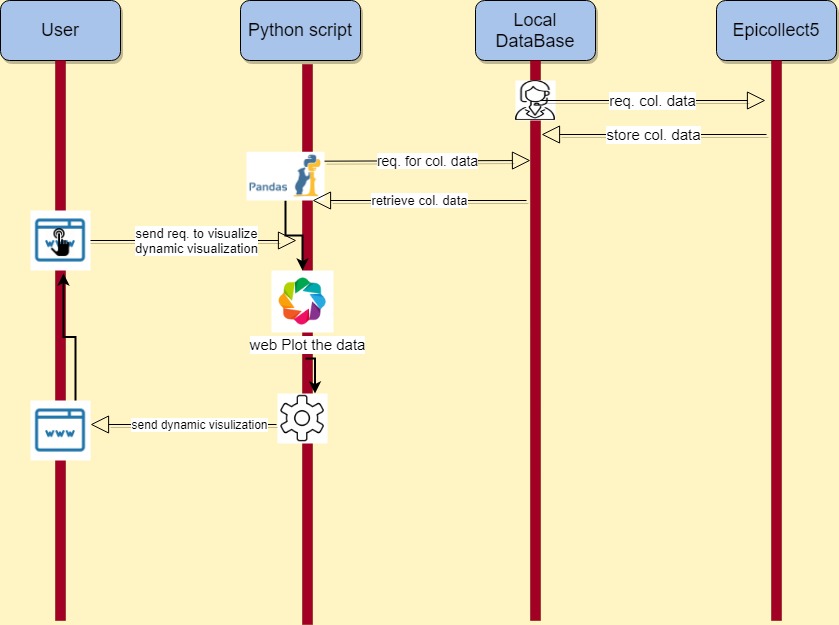


Figure 2.7: sequence diagram of dynamic analysis visualization

As mentioned before the data for dynamic analysis is collected data retrieved from Epicollect5. So it starts with retrieving data and storing it locally from Epicollect5 platform. Then in python pandas calls the data from local database and when the user request to visualize the graphs of collected data, Bokeh plots. Using the template engine the web app sends the graph to the user. At the same time she/he can change the variable of the plot and Bokeh and template engine update the requested data dynamically.

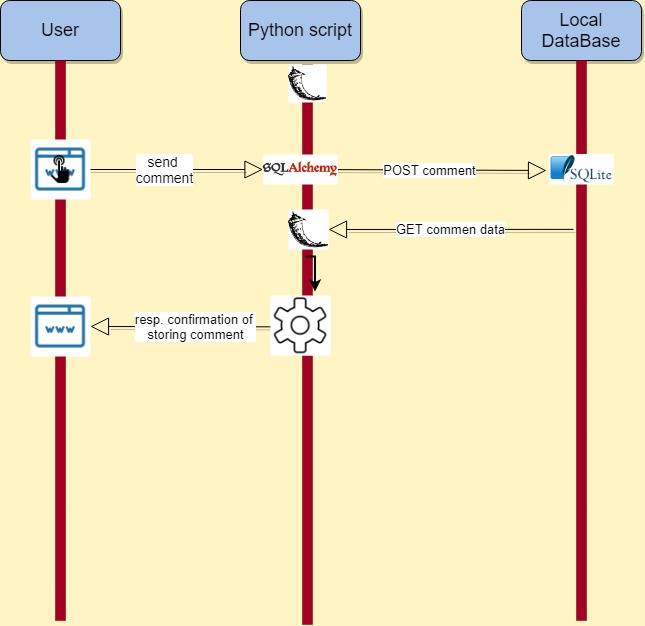


Figure 2.8: sequence diagram of posting comment

In order to comment on the app the user first needs to post a comment. Python then uses SQLalchemy to store it in SQLite locally. Then flask can GET the comment. The confirmation of the comment is sent through the template engine to the user.

1. - <https://five.epicollect.net/> [↑](#footnote-ref-1)
2. <https://www.postgresql.org/> [↑](#footnote-ref-2)
3. <https://www.openstreetmap.org/> [↑](#footnote-ref-3)