# DESIGN DOCUMENT AND TEST PLAN

Version 1.0

**Politecnico di Milano – Software Engineering for Geoinformatics**
25th May 2021

WEB-BASED APPLICATION FOR THE VISUALISATION AND ANALYSIS OF THE ALPHA CITIZEN SCIENCE STUDY IN LAGOS, NIGERIA

**Authors:**
M. Abd Alslam Mohammed Elkhalifa, M. Abdalla Eldouma Mohamed,
D. Aguirre, L. Dragun

## TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 PROJECT DESIGN

Design clarification and documentation are an indispensable phase in the successful development of a software system. It succeeds the stage of requirement analysis and illustrates how solutions to the formerly identified client's needs shall be implemented. The present document has been composed to elaborate on the software design and test plan of the web-based application dedicated to the communication and visualization of the citizen science data collected in the context of the ALPhA study, conducted by Urban Better | Oni et. al. in Lagos, Nigeria and Yaoundé, Cameroon.

The following pages will assume the reader's familiarity with the ALPhA web-application Requirement Analysis Specification Document (RASD) written by M. Abd Alslam Mohammed Elkhalifa, M. Abdalla Eldouma Mohamed, D. Aguirre and L. Dragun (2021).

The Software Design and Test Plan document primarily aims at providing guidance to the development team by outlining the system's overall architecture, illustrating the workflow of what needs to be built and how, and clarifying the relationship and connection between different software components. Although it is a technical document serving as a blueprint of the software's code, not necessarily aimed at the stakeholders, it can just as well be used by the client party to better understand the underlying technology of the product.

## 1.2 OVERVIEW OF THE DOCUMENT

The document will address the structure and details of the following project areas:

- Project Database:

  An overview of how the Epicollect5 data is being retrieved, edited and synchronized with the web-application's own database and how the latter is structured and administered.

- System architecture:

  The server-side architecture of the web-application is structured in three layers; the database (server) and database management system (DBMS), a WSGI compliant web server and a WSGI application server.

- User cases:

  The user cases previously defined in the RASD are now described in terms of which and how software components are activated/used in the various user case scenarios.

- Team Organization

  Describes the internal organization and work allocation of the development team. Although specific tasks are assigned to each developer, the system needs to be considered as a whole and the general means of interactions between components are to be understood by the whole team.

### 1.3. PRODUCT DESCRIPTION

Based on the previously analysed project requirements, the ALPhA web-application demands implementation as a dynamic website, since the majority of page components require data visualisation/customization that exceeds the abilities of static web-pages. To support the development of such a website, the use of a software framework can be helpful, since it provides libraries for automations such as templating engines or session management, together with predefined classes or functions that can be used to process user input or interact with databases. Furthermore, as specified in the RASD, the application shall be developed in python, therefore demanding a development framework that is compliant with WSGI, which is the specified protocol that describes how a web server communicates with web applications written in python. For these reasons, the Flask framework has been chosen to support the development of the ALPhA web-application. It is one of the most popular WSGI microframeworks used for web-application development with python, since it is simple yet extensible. It depends on the Jinja template engine, which allows the generation of dynamic html pages, the Werkzeug toolkit, needed to write WSGI-compatible applications in python, and it does not prescribe a database backend, therefore preserving the system's flexibility. Essentially, Flask provides all the means necessary to meet the project's requirements.

## 2. DATABASE ARCHITECTURE

### 2.1 EPICOLLECT5 DATASET

The web-application ultimately aims at displaying data collected on three different Epicollect5 datasets. Since they are composed in different languages, which complicates the administration of the database to meet user requests demanding filtered data based on specified attributes, the development team has decided to focus on the dataset of Lagos in the alpha phase of the software. The possibility of adding the French and English Yaoundé dataset to the project's database is left open and shall be addressed after the pilot is developed.

A crucial decision in the system design of the web-application is where, when and how the data from Epicollect5 is retrieved. As mentioned in the development team's RASD, one option would be to retrieve data in real-time from Epicollect5 via its REST API on every HTTP request sent by the client, process the raw data, extract the desired entries and sent the results as a HTML response back to the client. This solution has been rejected due to its inefficiency, meaning increased loading-times for the user and its limiting effects on the application's abilities. The system will therefore comprise its own database, that the web-application will make use of. Not only does this intermediate storage of data decrease waiting-times it moreover increases the system's reliability, since it is then independent from the Epicollect server and can uphold service even when the latter fails. The epicollect raw data retrieval through the Epicollect API, data preprocessing, and storage in the inherent database is carried out by a module separate from the flask web-application, which will run on the database server on deployment. This 'synchronizer' process retrieves the Epicollect5 data, filters them according to the project's interest and stores them in the project's inherent database (a more detailed description can be found in section 3.1).

## 2.1. PROJECT DATABASE

The ‘synchronizer’ process should store the extracted Epicollect5 data on a database server that can interact with the application through a Database management system (DBMS). For the development of the project, PostgreSQL has been chosen as DBMS, since it is a freely available open-source system that offers SQL-compliance and increased flexibility through extensions (add-ons) to enable operations on specific data types. Since the Epicollect5 dataset is georeferenced, the PostGIS extension will be exploited to allow the handling of spatial data. PostgreSQL is both a DBMS and inherently contains a Database server with a local database on which the project’s data will be stored during development.

The ‘synchronizer’ process must be run once before deployment of the web-application and is then executed with a frequency of 24 hours, as the Epicollect dataset is still active and growing. Moreover, the process is a one-way synchronization, therefore only appending new entries to the Postgre database, but never replacing any. This is done because the Epicollect user cannot edit their entry, therefore existing entries can never change and thus needn't be updated in Postgre. Entries in Epicollect5 can only be added or deleted. Designing the synchronization to work in one way only, helps avoid the risk of replacing the Postgre database with empty entries, in case a majority or all entries will ever be deleted from Epicollect. Consequently, the applications database will mirror everything that has ever existed on the Epicollect database.

The project’s database will contain:

- one geodataframe containing the georeferenced Epicollect5 data with 68 attributes.
- one dataframe dedicated to storing user login information.

Possible extension:

- one dataframe saving the users’ comments and ratings left on the ALPhA spaces. Since these comments need to be displayed whenever an ALPhA space has been selected, this dataframe needs to be connected to the geodataframe. This is achieved through identical indexing of data entries.

Since both comment and rating attributes are related to both the users and the geodataframe entries, this extension of the project would require extensive object relation management in the database and is therefore excluded from the pilot version of the application.

A list of the columns contained by the geodataframe stored on the PostgreSQL database can be seen here.

# 3. SYSTEM ARCHITECTURE

The server side of the system's client-server architecture is structured into three interconnected layers, that altogether provide the network that capacitates the final product (web-application):

1. Database server and DBMS (PostgreSQL):

   SQL-server that stores and administers the systems data. The web-application and the 'synchronizer' interact with the DBMS through CRUD operations.

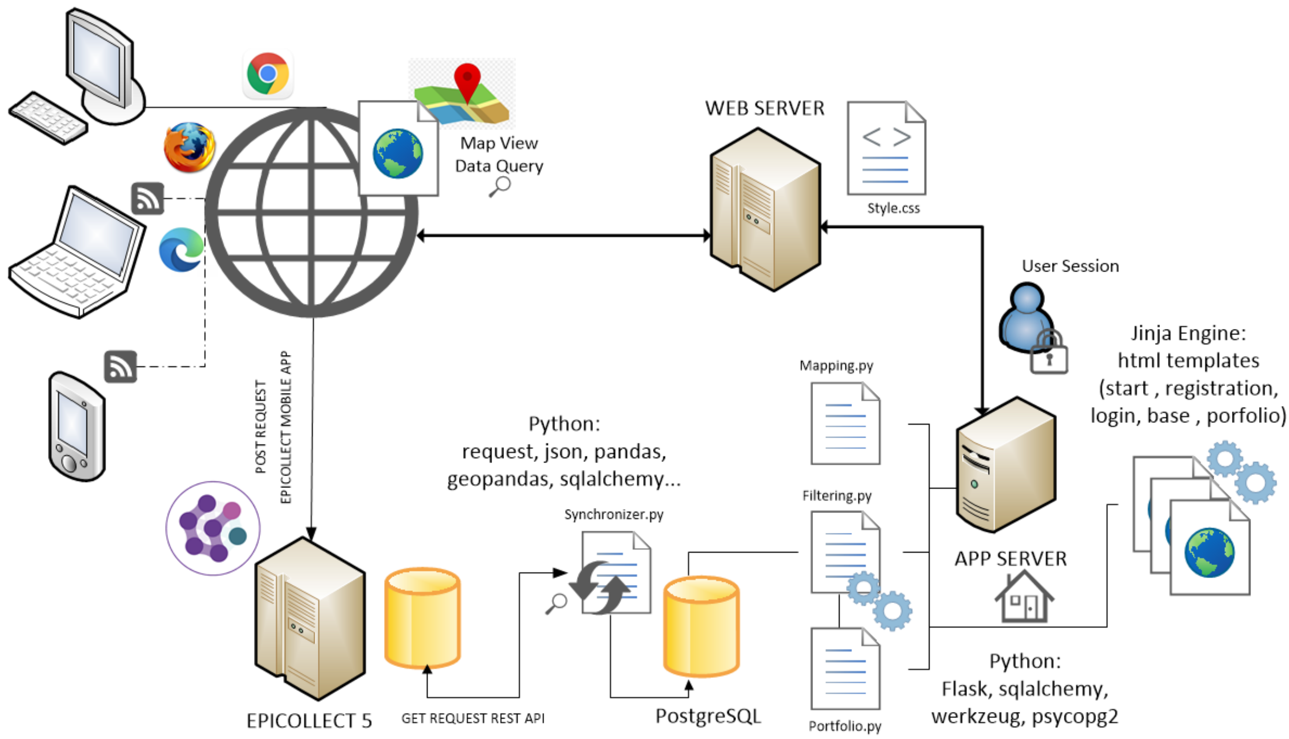
2. WSGI compliant web server:

   Handles all static assets (HTML and CSS) and passes all other client HTTP requests (for the dynamic content) to the WSGI server, which runs the flask web application and passes the response (HTML with CSS reference) back to the web server. The web server then serves the HTML and needed CSS files to the client.

3. WSGI compliant application and WSGI server:

   The WSGI server receives requests from the webserver and runs the WSGI python web-application i.e. the system's business logic (by invoking the callable object 'app'). The application executes functions that alter the values of the web-page's dynamic elements according to the request (by the means of Jinja templates) and sends the response (HTML) back to the web server.

As previously mentioned, the web-application will be developed within the Flask framework. Flask comes with a WSGI server, and the Werkzeug library provides a simple WSGI compliant web server (both for development purposes only) which will be used during the project's development stage for testing.



## 3.1. DATABASE SERVER, EPICOLLECT5 DATA RETRIEVAL AND PREPROCESSING

The system's inherent Postgres database is fed by the 'synchronizer' process, once per day and serves data to the flask application functions through the DBMS, on request. The synchronizer is responsible for the retrieval, filtering, clean-up, and formatting of the raw Epicollect5 data.
The Epicollect5 data is retrieved through the website's REST API (by customising the generic URL 'https://five.epicollect.net/api/export/entries/{project_name}?per_page={number of entries}'), which serves the requested raw data in the form of a json file. The dataset of the ALPhA study conducted in Lagos has been collected through an extensive questionnaire frequently utilizing follow-up questions, which results in data points exhibiting over 150 attributes, far more than what is of interest to the web-application. Therefore, the attributes of interest have been defined in a csv file based on which the 'synchronizer' filters the raw json data of the Epicollect5 API response. Besides unusable columns, entries without associated coordinates are also dropped, since the application only shows data in a map view. The dataset is then formatted into a data frame using the Pandas library. The data pre-processing also requires a basic outlier rejection mechanism, since certain data points are heavily lacking accuracy, placing them far outside the bounds of Lagos and its surroundings. Those outliers are removed using a statistical approach by the means of the SciPy library and the z score function for both latitude and longitude. A geometry column is added to the data frame transforming it into a geodata frame by the means of the Geopandas library. At this stage the resulting geodata frame contains a geometry column that holds shapely geometry objects of type point that display the associated geodetic coordinates (WSG 84) of each entry. For the application to be able to display these on a plane, they must be projected to cartesian coordinates (UTM 32N) The data pre-processing is finalised by indexing the geodata frame, starting from 0 for the oldest entry. This index does not replace the unique Epicollect5 index column (ID_EPC5), since it is needed to allow synchronization with the Epicollect5 dataset. Finally, the Sqlalchemy library is used to create a connection engine with the Postgres database, while the Geoalchemy2 library allows the geospatial data transfer to the database.

The PostgreSQL database is accessed by the web-application by the means of the DBMS through CRUD operations. While the geodata frame containing the Epicollect5 data is static and will mainly receive READ commands, the user information data frame is dynamic and will therefore be read, updated and written into.

## 3.2. WSGI-COMPLIANT WEBSERVER

The web server needs to be WSGI compliant. This means that it has to understand that HTTP client requests can be mapped to both static sources (HTML) as well as to sources that are dynamic i.e. to python code. The HTTP requests concerning static HTML are still held and served by the webserver itself, even when it is WSGI-compliant, but it will forward the HTTP requests for dynamic sources to a WSGI-server, that runs the python application, which will also return an HTML, which can then be served to the client by the webserver just like a static HTML would be. The webserver is also holding and serving the CSS files, referenced by the HTML responses.

### 3.3. WSGI-SERVER (APP-SERVER)

The WSGI server runs the WSGI compliant business logic of the system (i.e. the python code), by invoking a flask object that contains the web-application. The flask application object contains all the data about the app together with defined object functions governing the applications behaviour. These python functions are mapped to HTTP requests. Whenever Flask detects a match between the URL path provided and a defined function, the latter is executed and the results are returned as an HTML to the web browser and displayed in the client browser.
To avoid having to manually define every possible data querying result as an HTML for countless functions (which would be the equivalent to a static HTML), flask provides a template engine. One HTML template is defined for every page of the web-application and is rendered by the Jinja template engine on HTTP request, by passing it the modified variables defined for that template (see section 3.3.7).
The defined functions and their associated URL paths serve different purposes, which are further elaborated on in the following section.

#### 3.3.1. REGISTRATION FUNCTION

The function must answer to two both get and post HTTP requests. The user is prompted for information input (username, password, email) an the registration.html page. The user input is passed to the registration function as arguments, the function checks the user dataframe for already existing data associated with the provided email. If it finds a match, it returns an error message, if it does not it writes the user input to the user dataframe of the database and redirects the user to the login.html page.

#### 3.3.2. LOGIN FUNCTION

The login function operates with the same principle. It must answer to both get and post HTTP requests. The get HTTP request calls the function that returns the login.html page. On this page the user is asked to provide username and password, which are passed to the login function as arguments by a post HTTP request. The function checks if all necessary user input was provided and then reviews the user dataframe in the database. If the function can match the provided user input to an entry of the dataframe, it returns the base.html page (which the user is thus redirected to). If it does not find a match, it returns an error message.

#### 3.3.3. LOGOUT FUNCTION

This function clears the user session variable and redirects them to the application’s start page “start.html” .

### 3.3.4. FILTERING TOOL

The filtering tool reads entries from the database (geodataframe) according to defined filters. The function receives an input of user defined settings (list of strings) and returns a boolean series indicating true or false for every entry of the dataset, depending on whether or not they contain a specific string in a specific column. The filtering options and results are displayed to the user on the base.html (map view). The definition of the filters can only be based on (preferably compulsory) questions of the questionnaire which must be answered through the selection of one multiple-choice option.
Based on this logic, subsets of the ALPhA spaces dataset can be retrieved and displayed by applying one or more of the following filters:

→ Nature of the exercise
- Running or jogging
- Team sports
- Cycling
- Swimming
- Aerobics
- Other activity not listed
- Has the potential to be an ALPhA space

→ Organized activities
- Spaces that host organised group activities
- Spaces used for spontaneous individual activities'

→ Covid-19
- Spaces that are used during Covid-19 lockdowns
- Space that have not been used during Covid-19 lockdowns

→ Health
- Spaces that increase risk of injury or disease
- Spaces that decrease risk of injury or disease

→ Safety
- Spaces showing increased safety factors
- Spaces showing decrease safety factors

→ user drawing for spatial filtering

(This selection of filters is not yet final and may experience slight changes over the course of the following development stages.)

The Pandas boolean series is the basis on which data is subsequently read from the database. After reading the defined subset from the database, the filtering function passes an array of coordinates to the mapping tool, which in turn passes the data as a plot to the template engine, which renders the base.html with the user's data selection. The rendered HTML is sent to the web server, which combines it with the specified CSS file and serves both to the client.

### 3.3.5. MAPPING TOOL

The mapping function is called by the filtering tool and is responsible for providing a visualisation of all or a subset of the data points on a basemap. The tool will mainly make use of the bokeh library which provides basemaps as bokeh_tiles and puts out an interactive Bokeh plotting figure. This bokeh plot of the cartesian coordinates on a basemap is passed as a variable, containing an array, to the template engine, which renders the base.html with the passed variable.

### 3.3.6. ALPHA SPACE INDIVIDUAL PORTFOLIO

The Alpha space's individual portfolio is requested, when the user clicks on a datapoint on the base.html page. The corresponding HTTP request of this user input is mapped to the portfolio function. The function takes the user input, reads predefined attributes of the chosen data entry from the geodataframe and returns a list of strings. The list is passed as a variable to the template engine which renders the portfolio.html, which is then turned over to the web server. Attributes that the portfolio function retrieves include; visual, audio and text descriptions of the ALPhA space and of potential health and safety hazards observed there.

### 3.3.7. JINJA TEMPLATE ENGINE

As previously mentioned, the Jinja template engine, which is part of the flask framework, enables the creation of dynamically changing HTML templates. The template contains variables and expressions, whose values are replaced with the input passed to the engine from the above-described python functions. The Jinja template engine renders the dynamic HTML pages using basic python concepts such as variables, loops and lists resulting in a customized HTML file. The web-application displays content with three static HTML pages, concerning user registration and login (1,2,3) and two dynamic HTML templates (4,5) for the main pages.

1. Start.html:
   Visualises the options to either sign in or sign up to the system, and provides information on the ALPhA-study and Oni et al.
2. Registration.html
   Contains all fields necessary for user registration to the system
3. Login.html:
   Contains all fields necessary for user login
4. Base.html:
   Displays an interactive map, that changes upon HTTP request and a user friendly filter interface
5. Portfolio.html:
   Contains several fields dedicated to display data point attributes such as photo, audio and text descriptions of the ALPhA space

The HTML templates reference specific CSS files that dictate the style and appearance of the web page, including component colours, text alignment and border styles etc. The CSS files are held and served to the client by the web server.

## 4. USE CASES AND IMPLEMENTED REQUIREMENTS

To explain the functionalities of the software, the interactions between the components and possible exceptions, this section provides an explanation of the actions performed by the software and the user in a list of cases that are useful to explain the internal processes of the application.
This section describes what happens on the server and client side when the user cases occur by indicating the different actions that take place in these situations.

UC1: Registration:

- Registration.html page
- The user is prompted to provide personal information such as username, password, and email.
- The registration function checks that the entered username does not match any other username in the user information dataframe on PostgreSQL.
- If it does not, the registration function stores the user input (username and password) in the user dataframe
- The user is redirected to the login.html page.
- If it does the software will provide an error message to the user" wrong username or wrong password".
- The user got the ability to reset the password.

UC2: Login:

- Login.html page
- The user is asked to provide username and password.
- The login function receives this input and compares it to the information stored in the PostgreSQL user information dataframe.
- If the user's login information matches the ones in the database, the function redirects the user to the base.html page.

UC3: Default Map view

- Base.html page
- The filtering tool did not receive input yet, thus the default Panda Boolean series with all true values is used by the filter function to extract data entries from the geodata frame.
- The function retrieves all data entries and passes the list to the mapping function.
- The mapping function creates a bokeh plot of the coordinates on a base map and passes it as a variable to the jinja template engine.
- The base.html template is rendered with all available data points being displayed on the map.

UC4: Data filtering

- base.html
- The user sets one or more filters on the filter interface of the base.html page.
- The filter function receives the user input string and returns a Panda Boolean series indicating true or false for every entry in the geodata frame depending on whether or not they contain a specific string in a specific column (specified in the user input).

- The filter function reads entries from the geodata frame in the database based on the previously generated Boolean series (reads only entries tagged with true)
- And extracts the data entries cartesian coordinates, which are stored in an array as a variable that is passed to the mapping function.
- The mapping function plots the array of coordinates on a base map (bokeh plot) and passes the plot to the jinja template engine, which renders the base.html using the provided variable.

UC5: ALPhA portfolio exploration

- The user is browsing on the base.html page, which was rendered to display a subset of data points that the user selected by applying filters.
- The user then clicks on any one datapoint, which causes a HTTP request to be forwarded to the flask application, which thereupon executes the portfolio function.
- The portfolio function extracts an array of predefined attributes from the selected entry of the geodata frame.
- This list is passed as a variable to the jinja template engine, which renders the portfolio.html
- The selected ALPhA space customised portfolio is sent to the webserver and served to the web-client.

## 5. TEAM ORGANIZATION