SOFTWARE RELEASE

DOCUMENT

Version 1.0

Politecnico di Milano – Software Engineering for Geoinformatics  
7th June 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

1. **INTRODUCTION**
   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](https://github.com/dragun0/SE4GI_webapp/blob/main/RASD_Group4.pdf) (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. **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. **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 webpages. 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.

1. **DATABASE ARCHITECTURE**
   1. **EPICOLLECT5 DATASET**
2. **SYSTEM ARCHITECTURE**
   1. **DATABASE SERVER, EPICOLLECT5 DATA RETRIEVAL AND  
       PREPROCESSING**
   2. **WSGI-SERVER (APP-SERVER)**
3. **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.

1. **TEAM ORGANIZATION**

**REFERENCES**

Flask.palletsprojects.com. 2021. *Welcome to Flask — Flask Documentation (2.0.x)*. [online] Available at: <https://flask.palletsprojects.com/en/2.0.x/> [Accessed 24 May 2021].

Fullstackpython.com. 2021. *WSGI Servers*. [online] Available at: <https://www.fullstackpython.com/wsgi-servers.html> [Accessed 25 May 2021].