

Software Engineering 4 Geoinformatics Project

Design Document

**Date:** 23-June-2020

**Deliverable:** DD

**Title:** Design Document

**Version:** 2.0

**Authors:** Indusekhar Ravilla, Abdoul Kasim Sogore, Firmino M. Mucucete, © 2020, All rights reserved

Contents

[1**.Introduction** 3](#_Toc44129226)

[**1.1** **Purpose:** 3](#_Toc44129227)

[2. **Structure Design:** 4](#_Toc44129228)

[2.1 **Database Design** 4](#_Toc44129229)

[2.2 **Flask Application:** 6](#_Toc44129230)

[2.2.1 **Template:** 6](#_Toc44129231)

[2.2.2 **Python** 7](#_Toc44129232)

[4.**Mapping of the Requirements Components using use case Flow Diagrams**: 10](#_Toc44129233)

[5.**Effort Spent:** 23](#_Toc44129234)

# **1.Introduction**

The aim of the design document is to guide the development and provide high level information on the structure of the software. This document describes a set of design characteristics required for the implementation of the web application.

In details this document contains a full overview of:

1. Database’s design
2. Software’s structure, such as independent programs, Template and Python script

## **Purpose:**

The Purpose of this project is to develop a web App that shows the data of the trees that has been collected from the Epicollect in an interactive way.

The Web App should be accessible through applied technology (Computer or mobile phone).

The Web App should be available 24h/7 .

# **2. Structure Design:**

The structure of this web-application has been designed on two independent and distinct levels: a script in charge of creating the database and the Flask application.

## **2.1 Database Design**

The design of database largely determines the usability and robustness of the system.

**PostgreSQL** -open source object-relational database-has been chosen for managing, updating, and accessing information.

**Conceptual Model**

User

Login

Register

User

Post Data Comments

Remove comments

Login

View Data

Download Data

Modify Comments

Post Comments

View Comments

**Physical Model**

**User:**

|  |  |
| --- | --- |
| **USER\_ID** | INTEGER PRIMARY KEY |
| **USERNAME** | VARCHAR (20) NOT NULL |
| **EMAIL** | VARCHAR (120) NOT NULL |
| **PASSWORD** | VARCHAR (60) NOT NULL |

**Post:**

|  |  |
| --- | --- |
| **POST\_ID** | SERIAL PRIMARY KEY |
| **AUTHOR\_ID** | INTEGER NOT NULL |
| **CREATED** | TIME STAMP |
| **TITLE** | VARCHAR (350) NOT NULL |
| **BODY** | VARCHAR (350) NOT NULL |
| **FOREIGN KEY** | AUTHOR\_ID |
| **REFERENCES** | USER\_ID |

**Data:**

|  |  |
| --- | --- |
| **DATA\_ID** | SERIAL PRIMARY KEY |
| **AUTHOR\_ID** | INTEGER NOT NULL |
| **CREATED** | TIME STAMP DEFAULT NOW() |
| **NAME** | VARCHAR (350) NOT NULL |
| **HEIGHT** | INTEGER NOT NULL |
| **DIAMETER** | INTEGER NOT NULL |
| **CIRCUMFERENCE** | INTEGER NOT NULL |
| **CONDITION** | VARCHAR (20) NOT NULL |
| **LATITUDE** | VARCHAR (350) NOT NULL |
| **LONGITUDE** | VARCHAR (350) NOT NULL |
| **FOREIGN KEY** | (AUTHOR\_ID) |
| **REFERENCES** | Data (data\_id) |

The data Used here is selected from the Epicollect which provides the users about the information of the trees: it is geographic position, name, condition, diameter, height. However, the user can also contribute in data collection, by inserting the new data into the system that has been surveyed and collected by the user. Those data will have the same attribute of the ones present in Epicollect and will be stored in a table present in the Database

## **2.2 Flask Application:**

The Flask application includes the template and Python part, both have their purpose, internal structure and, most important, they interact with each other.

### **2.2.1 Template:**

The templates control the overall look and layout of a site. They provide the framework that brings together common elements, modules and components as well as providing the cascading style sheet for the site.

Dedicated templates were created depending on their function for the web-application.

|  |
| --- |
| Index.html |
| Login.html |
| Register.html |
| Create.html |
| Update.html |
| Interactive\_map.html |

* **Index.html**: It is the user home page of the web Application where users can see the information about the web app
* **Login.html:** Registered users use this form to log-in with their credentials.
* **Register.html**: Users use this form to register themselves into the Web Application by providing their Data such as Username, Email, and Password.
* **Create.html:** This allows the user to create new Posts.
* **Update**.html: This allows the user to update their posts.
* **Interactive\_map.html:** This form shows the user **the map** with the info about the trees (Position, condition, name etc.), A histogram with the name of the species on X and circumference of the trees on Y.

### **2.2.2 Python**

Inside the Flask web application different modules/application packages have been distinguished, referring to: the users, the posts and the main pages of which the web app is composed of.

The libraries and modules used for the web app development are the following:

**Flask:** Itis a microframework that allows us to create servers in Python. From flask we import:

Flask, render\_template, redirect, request, url\_for, session, g., ...etc.

* **Render\_template:** Render our html pages.
* **Request:** is a python HTTP library. That is useful on many fronts in our work.
* **Session:** Helps us track the session data, comes handy when the member wants to:
  + Logout or Add a comment.
* **Redirect:** Helps us redirect users to other pages (mainly the homepage), usually used with (url\_for).
* **Flash ():** it is used to generate informative messages and keep the user updated of every development, for example error messages, as can be seen in the web application
* g: a global variable that allows us to interact with the html templates.

**Bokeh:** It is an interactive data visualization library to visualize data and create interactive plots and applications running on a web browser.

**Werkzeug.security:** Werkzeug is python library which contains lot of development and debugging tools for implementation of web application gateway interface(WSGI) applications. The good part is you can use this system not only for your web applications but also for standalone python applications like desktop apps, scripts, mobile apps and so on.

There are various security functions available in the werkzeug.security but we are interested in generate\_password\_hash and check\_password\_hash.

1. **generate\_password\_hash:** generate\_password\_hash takes plaintext password, hashing method and salt length as an input to produce hashed password. By default, it produces salt string with length 8.

This allows us to store hashed values of passwords in the database, instead of storing them in clear.

1. **check\_password\_hash:** check\_password\_hash takes two inputs password hash and plaintext password and returns True if hash matches actual hash of plaintext password else returns False.

This allows us to check whether the password provided matched the hashed\_value that is stored in the database.

* When you allow user to signup use generate\_password\_hash and store hashed version of password in database under column **hashed\_pwd**.
* When you authenticate user via sign in use check\_password\_hash by passing already **hashed password** for user from database and **input password** (plaintext) to test veracity of password.

**werkzeug.exceptions:** This module implements several Python exceptions you can raise from within your views to trigger a standard non-200 response. The one of the Exceptions we use is Abort.

**Abort () :** werkzeug.exceptions import abort This function is used to terminate the process if a certain condition is met, it returns the chosen HTTP error.

**Psycopg2:** Psycopg2 is a DB API 2.0 compliant PostgreSQL **driver** that is actively developed. It is designed for multi-threaded applications and manages its own connection pool. Other interesting features of the **adapter** are that if you are using the PostgreSQL array data type, Psycopg will automatically convert a result using that data type to a Python list

**“From psycopg2 import (connect)”**

Simply it is used to connect to the PostgreSQL Database.

**Json:**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight format for storing and transporting data
* JSON is often used when data is sent from a server to a web page
* JSON is "self-describing" and easy to understand

**CSS:**

* **Cascading Style Sheets** (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup Language like HTML.
* CSS is a cornerstone technology of the World Wide Web alongside HTML and Java Script
* Similarly, The CSS file in the Blog folder is the style presentation of our web application

# **3**.**Mapping of the Requirements Components using use case Flow Diagrams**:

* Use case 1**: To able to Register. The flow Chart explains the Registration Process**

A close up of a map

Description automatically generated

@app.route(‘/Register’, methods=(‘GET’,’POST’)).

@app.route(‘/register’, methods=(‘GET’,’POST’)).

\*The function must answer to two HTTP requests:

* 1. **If the request is POST**:
     1. Get the Information sent by the User i.e. Username and Password
     2. Verify that required fields are inserted correctly.
     3. If both are inserted, connect to the database with the function **get\_dbConn (),** give the position on Database to a parameter (curr) and execute the query to search the inserted username.
     4. Check if the username exists in the database using the fetchone() function.
     5. If the Username exists give the message "User is already registered".
     6. If the username not found, then save the data into the database and return a message saying “Registration Complete”
     7. Redirect the user to the login page.
  2. **If the Request is GET**:

1. The system redirects the user into the registration page using the function redirect() and, also use the function render\_template() to load the style of the Registration page.

* **Use Case 2:** To be Able to Login. The Flow chart Explains the Registered User Login Processes

A close up of a map

Description automatically generated

@app.route(‘/Login’, methods=(‘GET’,’POST’))

@app.route(‘/login’, methods=(‘GET’,’POST’))

\*The function must answer to two HTTP requests:

* 1. **If the Request is POST**:

1. Get the information sent by the User i.e username and password.
2. connect to the database with the function **get\_dbConn ()**, and check for the entered username using the function **fetchone()** .
3. If the username does not exist, then send error message to user.
4. If the username exists, then verify the entered password using the function **check\_password\_hash().**
5. If the entered password is does not exit send an error message
6. If both Username and Password are correct then clean the session variable with the function **clean(),** store the user\_id in the session variable and the redirect user on the index page.
   1. **If the Request is GET**:
7. It redirects the user to the Login Page using the function **redirect()** and, also use the function **render\_template()** to load the style of the Login page

* **Use Case 3:** User Logout Process. The Flow chart Explains the User Logout Flow

**A close up of text on a white background

Description automatically generated**

@app.route(‘/Logout’)

@app.route(‘/logout’)

The function must:

1. Clean the session variables.

2. Redirect the user on the application homepage.

* **Use Case 4**: Post Data, The Post Data Process is Explained in the Below Flow chart

**A close up of a piece of paper

Description automatically generated**

The Function **upload()** is used in this process. Also, this function works through the HTTP POST request and the HTTP GET request.

**4.1 If the Request is POST:**

* If the user is logged in, acquire information from the User and save them into variable (comment).
* Check that they are complete (if not send an error message).
* Check they are written in a correct way.
* Connect the database and execute the query to store the features in the database.
* Close the connection and save with the **commit()** function.
* Redirect on the main page using the function **redirect()**

**4.2 If the Request is GET:**

* if user is logged in, render the upload page using the function render\_template()
* If user is not logged, the system sends an error message to the user and redirect him/her to the login page.

**Use Case 5**: Leave and View Comments. The Flow Chart Explains the Process of User viewing the Comments and Posting Comments.

**A close up of a map

Description automatically generated**

This use case is represented in the code by **create()** function. Also, this function works through the HTTP POST request and the HTTP GET request.

**5.1 If the Request is POST:**

* **Check the user is logged in**
* If the user is logged in (its ID is in the session), acquire information from the client saving them into variable (comment).
* Check that they are complete (if not send an error message).
* Connect the database and execute the query to store the comment in the database.
* Close the connection and save with the **commit()** function.
* Redirect on the main page.

**5.2** **If the Request is GET:**

* If the user is Logged in Render the Creation Page
* If the user is not logged, the system sends an error message to the user and redirect him/her to the login page using the function **redirect()**

**If the User Want to Modify Comments:**

**Use case 6**: Modify a comment

This use case is represented in the code by **update()** function. Also, this function works through the HTTP POST request and the HTTP GET request.

-This function receives a parameter: the id of the post to be updated

**6.1 If the Request is POST:**

* Firstly, check if the user is logged in.
* If the user is logged in, check whether the post with the specified id exists.
* if it does not exist, send an error message and redirect the user to the main page.
* If the post exists, connect to the database to search and update the comment.
* Close the database and save the operations with **commit()** function.
* Redirect to the main page.

**6.2 If the Request is GET:**

* if user is logged in, render the update template
* if user is not logged in, send an error message, and redirect him/her to the login page

**A close up of text on a white background

Description automatically generated**

* **Use case 7:** Delete Comment. The following Flow chart explains the Process of Deleting a comment Uploaded by the User.

This use case is represented in the code by **delete()** function. This function works only through the HTTP **POST** request.

- Connect to the database and execute a query to delete a comment.

- Redirect on the main page.

**A close up of a map

Description automatically generated**

* **Use Case 8**: View Data. The View Data Flow Chart explains the Flow of Vie data Process when the user clicks the View Data Button.

@app.route('/ViewMap')

@app.route('/viewmap')

The function must:

* Redirect the user to the imported Open Street Maps with the contribution of CartoDB.
* Render the basemap from Open Street Maps with the contribution of CartoDB.
* Shows the locations of dataset entries.

The process is:

This use case consists of plotting the data (previously saved in a shapefile) and adding a basemap.

**Create a shapefile starting from epicollect5:**

* Send a GET request using the function **get()** from the library “request” and store the raw text of the response in a variable.
* Encode the raw response text into a JSON variable with the function **loads()** from the package “json”.
* Use the function **json\_normalize()** to fit the variable to a Pandas Data frame.
* Create new columns with numeric values, one for each feature (latitude, longitude, diameter, height, Name, address, and condition of the tree).
* Transform the DataFrame into a GeoDataFrame and plot it.

Set the CRS and save all in a shapefile using the function **to\_file().**

Then use to **Bokeh** (It is an interactive data visualization library to visualize data and **create interactive plots and applications running on a web browser**).

* Open the shapefile
* Create a function to extract point coordinates (x,y) from the geodataframe and save them as attributes in a new one dataframe.
* Use the dataframe as Bokeh ColumnDataSource (this function maps names of columns to sequences or arrays).
* Create a map plot in the chosen CRS and add title, labels and glyphs to it.
* Tell Bokeh where to generate output with the function **output\_file()** , then use the function **show()** to save the graph in an HTML file and view it in a browser.

# **5.Effort Spent:**

* Sogore Abdoul Kassim: Web Application Implementation, RASD, DD (Comments on RASD and DD)
* Mucucete Firmino Manuel: Web Application Implementation, RASD, DD (Comments on RASD and DD)
* Indusekhar Ravilla: RASD and DD Implementation, Web Application(Web App Design View comments)