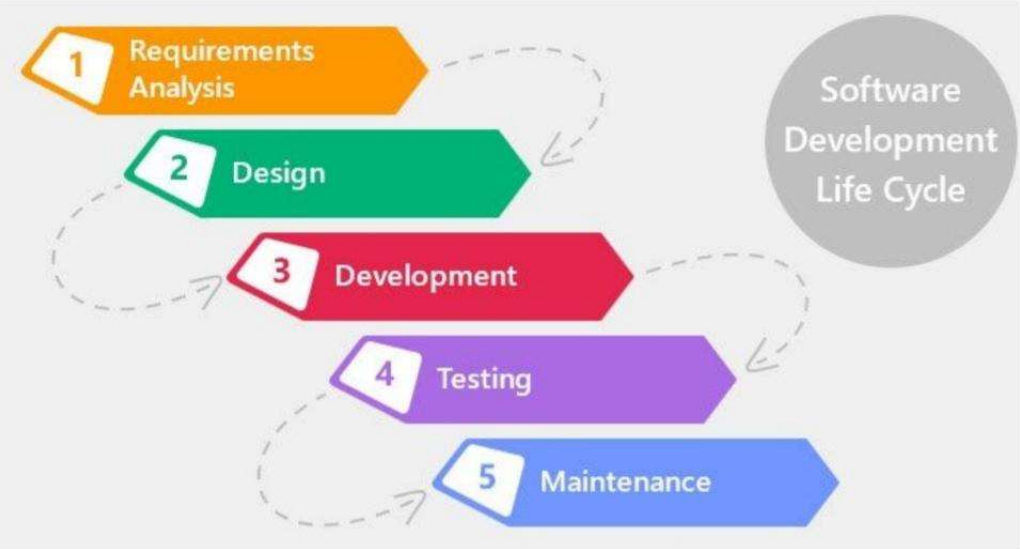
POLITECNICO
MILANO 1863

2020



Design Document

VERSION 1.0

May 12, 2020

Deliverable: DesignDoc

Title: Design Document

Authors:

Wafi Mahdi Eldoud Mahdi

Effort: %25 (Application Functions, HTML Templates, Modules)

MohamedElmustafa Omer

Effort: %25 (Application Functions, HTML Templates, Modules)

Abubakr Albashir

Effort: %25 (Database Design, HTML Templates, Aim, Scope)

Faris Elsmani

Effort: 25% (Database Design, HTML Templates, Aim, Scope)

Date: 12 May 2019

Table of Contents:

1.Introduction:

1.1 Aim:

The design document aims to guide through the various functions carried out in the web application created to access information from the HATUA research.
The document will provide details regarding the web application's modules, templates, functions, and database.

1.2. Scope:

This document is concerned with showcasing technical details regarding the web application. The Application will allow casual users to search for information from a database derived from the HATUA research and gain access to the inquired information through several visualization methods such as pie charts and bar charts. The search function in the application will allow users to draw comparisons between the research subjects' reaction to symptoms with regards to their level of education, tribe, and creed. Moreover, it will also allow users who register as members to interact with each other via a built-in message board.

2. Modules:

2.1 Flask: is a microframework that allows us to create servers in Python. From flask we import: Flask, render_template, redirect, request, url_for, session, g. These libraries/functions help us:

- Render_template: Render our html pages.
- Request: is a python HTTP library. That is useful in many fronts in our work
- Session: Helps us track the session data, comes handy when the member wants to: Logout, Add a comment, Modify a comment or Delete a comment.
- Redirect: Helps us redirect users to other pages (mainly the homepage), usually used with (url_for).

2.2 JSON: is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

From JSON we import:response, raw-data, json.loads and pdf.json_normalize.

2.3 The Dataset:

The dataset UG HATUA PHASE 2 OUTPATIENTS will be retrieved from the internet using JSON, we will use the request library to send requests to the REST API service of the EPICOLLECT platform. The JSON library will be used to encode the raw response text into a JSON variable and then into a Pandas DataFrame.

Finally, the Pandas DataFrame will be converted into a GeoDataframe. Our dataset will be stored as a table in the database (data_table).

This Dataset will be available to all users of the Software.

3. HTML Templates:

3.1 Home.html: This page contains the unified background and design for the web application pages. It also contains the application's Name (should always be visible in the application). With a link to: Register, Login and Search pages. If the user is already logged in it shows a message that says: Logged in as username and provides a link to logout.

3.2 Register.html: This page is accessed from the ('/Register') function. It asks the user to provide personal information such as username, password, email (optional) and age (optional). if the username provided by the user doesn't match any other username in the database then the system allows the registration and returns "Registration complete". The input will be stored in the (sys_table) in the database. Also, it redirects the user to the Login page (Login.html).

3.3 Login.html: This page is accessed from the ('/Login') function. It asks the user to input login information (username and password) and provides submission confirmation. if the user's login information matches the ones in the (sys_table) then the user is allowed to log into the system, it redirects the user to the home page (Home.html) with the Add Comment (AddComment.html) activated.
if the information provided by the user doesn't match the ones in the (sys_table) in the database then it returns an error message that says "Login failed!" and redirects the user to the homepage. (Home.html)

3.4 Logout.html: This page is accessed from the ('/Logout) function. It asks the user to confirm the desire to suspend his/her status as a registered member in the application. If the User chooses to logout it provides confirmation. Then, it redirects the user to the home page (Home.html) with the Add Comment (AddComment.html) deactivated.

3.5 Search.html: This page is accessed from the ('/Search) function. It asks the user to input an inquiry regarding the related attribute field from the attribute data in the dataset. If the inquiry is compatible with the database, it redirects the user/member to the customizeplot.html page in which he/she chooses the prefered method of visualization (line graph, pie chart, bar chart), else it displays a "not found" message.

3.6 ViewMap.html: This page is accessed from the ('/ViewMap) function. It takes the user to an imported map from Google Maps which includes a layer of the database information stored in the (data_table). The user then can access information displayed in the map based on location.

3.7 AddComment.html: This page is accessed only if the member is logged in (userid in session). It is accessed from the ('/Addcomment') function, it asks the user to add a comment if he wishes. The input will be stored in the comment_table in the database. The comment (comment) will be added in the comment_table in the database with the username (userid), the time of the comment (timestamp) and also every comment will be assigned a unique identifier (comment_id).

3.8 EditComment.html: This page is accessed only if the member is logged in (userid in session). It is accessed from the ('/EditComment') function, it allows the user to edit the comment only if the userid of the user is the same as in the comment_table in the database. After the comment is modified it redirects the user to the homepage (home.html).

3.9 DeleteComment.html: This page is accessed only if the member is logged in (userid in session). It is accessed from the ('/DeleteComment') function, it allows the user to delete the comment only if the userid of the user is the same as in the comment_table in the database. After the comment is deleted it is also extracted from the comment_table in the database. The page then redirects the user to the homepage (home.html).

3.10 CustomizePlot.html: This page is accessed only if the member is logged in (userid in session). It is accessed from the ('/CustomizePlot') function, which the user is redirected to from the search.html page. It asks the user to choose the way the user wants to visualize the plot of a certain attribute (field from the dataset). The visualizing ways are requested from a provided list and then the chosen visualizing method will be shown in the page.

4. Application functions:

4.1 User opens the webpage:

@app.route (‘/ ’)

@app.route (‘/Home’)

This function renders the Homepage of the web application (Home.html)

4.2 User/Member searches for data:

@app.route (‘/ ’)

@app.route('/Search', methods=('GET','POST'))

*The function must answer to two HTTP requests:

I) If request is POST:

1. Acquire the search item from the member/user.
2. Cross reference search item with items in database.
3. Retrieve database attributes relevant to a search item.
4. Ask the user to identify a preferred visualization method (line graph, pie chart, bar chart).
5. If a search item is not compatible with the dataset, return message "Not Found".

II) On GET, it redirects the user to the Search page.

4.3 Member registration:

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

*The function must answer to two HTTP requests:

I) If request is POST:

1. Get the information sent by the Member-to-be (username and password)
2. Check if they have been correctly inserted (Username and password filled and follow predefined syntax)
3. Check if the username exists in the database
4. If yes, send an error message to the user. “Username already exists, login if it’s your account”
5. If not, store the information into the database and return a message that says: “Successful registration!”
6. Redirect the user to the login page

II) On GET, it redirects the user on the registration page.

4.4 Member Login:

@app.route('/LoginMember', methods=('GET','POST'))

*The function must answer to two HTTP requests:

I) If request is POST:

1. Get the information sent by the Member (username and password)
2. Check if the username exists in the database.
3. If not, send an error message to the member. "Error! wrong username"
4. If yes, check if the password exists in the database
5. If not, send an error message to the member. "Error! wrong password"
6. If yes, store the Member_id in the session variable.

II) On GET, it redirects the user on the login page.

4.5 Member Adds a Comment:

@app.route('/Addcomment', methods=('GET','POST'))

*The function must answer to two HTTP requests:

I) If request is POST and the member is logged in (userid is in the session):

1. Acquire the comment from the member.
2. Store the comment in the database.
3. Return a message that says "Comment added successfully"
4. Visualize the comment alongside the Member_id
5. Redirect the member to the homepage.

II) On GET, it redirects the user to the add comment page.

4.6 Member modifies his comment:

@app.route('/EditComment', methods=('GET','POST'))

*The function must answer to two HTTP requests:

I) If request is POST and the member is logged in (userid is in the session):

1. Check if the comment was added by the same member (compare userid)
2. If not, return error message "Sorry, can't modify other users' comments"
3. If yes, acquire the comment from the database.
4. Allow the member to edit the comment.
5. Store the edited comment in the database under the same comment_id.
6. Return a message that says "Comment modified successfully!"
7. Redirect the member to the homepage.

II) On GET, it redirects the user to the add comment page.

4.7 Member deletes his comment:

@app.route('/DeleteComment', methods=('GET','POST'))

*The function must answer to two HTTP requests:

I) If request is POST and the member is logged in (userid is in the session):

1. Check if the userid in the session is the same in the comment database.
2. If not, return error message "Sorry, can't delete other users' comments"
3. If yes, acquire the comment from the database.
4. Allow the member to delete the comment.
5. Delete the comment from the database and return a message that says "Comment deleted successfully!"
6. Redirect the member to the homepage.

II) On GET, it must send the user to the delete comment page.

4.8 Member logout:

@app.route('/LogoutMember')

*The function must:

1. Clean the session variables.
2. Redirect the user on the application homepage.

4.9 Member/User conducts map-based search:

@app.route('/ViewMap')

The function must:

- Redirect the user/member to the imported Google Maps map.
- Render the base map from Google maps map.
- Shows the locations of dataset entries.

5.Database Design:

The database management system: The database management system of the web application is PostgreSQL 10 DBMS that works on the local machine.

5.1 Database Schema:

The database schema consists of three tables:

1. The sys_table: the table contains the system users' information needed for registration and login:(user ID, usernames, passwords, and ages).

2. The comment_table: the table contains the user ID, the users' comments and the time of writing or editing the comments.
3. The Data_table: the table contains the dataset of the patients which are the scope of the system including their locations and other attributes related to them.

The connection between the first two tables is done by the foreign key field: userid.

5.2 Tables Design:

1. **The sys_table:**

Field Name	Datatype	Mandatory/ Optional	Note
userid	INTEGER		Serial primary key
username	VARCHAR (255)	Mandatory	NOT NULL, unique
password	VARCHAR (255)	Mandatory	NOT NULL
email	VARCHAR (255)	Optional	
age	INTEGER	Optional	

2. **The comment_table:**

Field Name	Datatype	Mandatory/ Optional	Note
comment_id	INTEGER		Serial primary key
userid	INTEGER	Mandatory	NOT NULL, UNIQUE
timestamp	FLOAT	Mandatory	
comment	VARCHAR (500)	Mandatory	NOT NULL

The userid field is used as a foreign key to connect the first two tables.

3. **The data_table:**

Field Name	Datatype	Optional/Mandatory	Notes
patient_id	INTEGER		NOT NULL, UNIQUE
location	POINT	Mandatory	
intervew_date	DATE	Optional	
gender	VARCHAR (255)	Optional	
martial_status	VARCHAR (255)	Optional	
religion	VARCHAR (255)	Optional	
patient_education_level	VARCHAR (255)	Optional	
ethnic_group	VARCHAR (255)	Optional	
disease_stegma	VARCHAR (255)	Optional	
time_from_first_sym	DATE	Optional	
first_doctor_visit	DATE	Optional	