SOFTWARE ENGINEERING FOR GEOINFORMATICS PROJECT

DESIGN DOCUMENT

Version 1.1

Project name: BIKOD

Prepared by: AbdelGafar, Omar

Ruken Dilara Zaf

Fakherifard Katayoun

Submission Date: 09.06.2019

Table of Contents

Table of Figures

REVISION HISTORY

Name	Date	Reason for Changes	Version

Introduction

The purpose of the Design Document (DD) is to present a technical detailed description of BIKOD website by introducing detailed description of the black box of the web site structure as well as the database of the system. In order to have the full description, the purpose, functions at each program and for the database have been explained in detail.

1. BIKOD Structure

BIKOD web folder is organized as shown in Figure 1.

```
D:.
|   call.py
|   dbConfig.txt
|   forms.py
|   hello2.py
|   output
|
+---static
|       Autolinker.min.js
|       fontawesome-all.min.css
|       L.Control.Locate.min.css
|       L.Control.Locate.min.js
|       labelgun.min.js
|       labels.js
|       leaflet-control-geocoder.Geocoder.css
|       leaflet-control-geocoder.Geocoder.js
|       leaflet-hash.js
|       leaflet-heat.js
|       leaflet-measure.css
|       leaflet-measure.js
|       leaflet-search.css
|       leaflet-search.js
|       leaflet-svg-shape-markers.min.js
|       leaflet-tilelayer-wmts.js
|       leaflet.css
|       leaflet.js
|       leaflet.js.map
|       leaflet.markercluster.js
|       leaflet.pattern.js
|       leaflet.rotatedMarker.js
|       Leaflet.VectorGrid.js
|       leaflet.wms.js
|       main.css
|       MarkerCluster.css
|       MarkerCluster.Default.css
|       multi-style-layer.js
|       OSMBuildings-Leaflet.js
|       qgis2web.css
|       qgis2web_expressions.js
|       rbush.min.js
|       stations_2.js
|       style.css
|
+---templates
|   |   about.html
|   |   home.html
|   |   layout.html
|   |   login.html
|   |   register.html
|   |   welcome.html
|   |
|   \---map
|       +---css
|       |   |   fontawesome-all.min.css
|       |   |   L.Control.Locate.min.css
|       |   |   leaflet-control-geocoder.Geocoder.css
|       |   |   leaflet-measure.css
|       |   |   leaflet-search.css
|       |   |   leaflet.css
|       |   |   MarkerCluster.css
|       |   |   MarkerCluster.Default.css
|       |   |   qgis2web.css
|       |   |
|       |   \---images
|       |           cancel.png
|       |           cancel_@2X.png
|       |           check.png
|       |           check_@2X.png
|       |           focus.png
|       |           focus_@2X.png
|       |           layers-2x.png
|       |           layers.png
|       |           marker-icon-2x.png
|       |           marker-icon.png
|       |           marker-shadow.png
|       |           measure-control.png
|       |           rulers.png
|       |           rulers_@2X.png
|       |           start.png
|       |           start_@2X.png
|       |           throbber.gif
|       |           trash.png
|       |           trash_@2X.png
|       |
|       +---data
|       |       stations_2.js
|       |
|       +---images
|       |       loader.gif
|       |       search-icon.png
|       |
|       +---js
|       |       Autolinker.min.js
|       |       L.Control.Locate.min.js
|       |       labelgun.min.js
|       |       labels.js
|       |       leaflet-control-geocoder.Geocoder.js
|       |       leaflet-hash.js
|       |       leaflet-heat.js
|       |       leaflet-measure.js
|       |       leaflet-search.js
|       |       leaflet-svg-shape-markers.min.js
|       |       leaflet-tilelayer-wmts.js
|       |       leaflet.js
|       |       leaflet.js.map
|       |       leaflet.markercluster.js
|       |       leaflet.pattern.js
|       |       leaflet.rotatedMarker.js
|       |       Leaflet.VectorGrid.js
|       |       leaflet.wms.js
|       |       multi-style-layer.js
|       |       OSMBuildings-Leaflet.js
|       |       qgis2web_expressions.js
|       |       rbush.min.js
|       |
|       +---legend
|       |       stations_2.png
|       |
|       +---markers
|       \---webfonts
|               fa-solid-900.ttf
|               fa-solid-900.woff2
|
\---__pycache__
        form.cpython-35.pyc
        forms.cpython-35.pyc
        forms.cpython-37.pyc
```

Figure 1:BIKOD Folder Tree Diagram

1.1 Independent Programs

```
D:.
|   call.py
|   dbConfig.txt
|   forms.py
|   hello2.py
```

Figure 2: Independent Programs diagram for BIKOD

BIKOD web is run under main three independent programs (hello.py, forms.py and call.py) as shown in Figure 2.

1.1.1 hello2.p

1.1.1.1 Purpose

This program contains the import of libraries ex. (flask, werkzeug. security, werkzeug. exceptions, forms, etc.) as well as flask functions ex. (render_template, request, redirect, flash, etc.). Moreover, it contains the route definition of BIKOD webpages ex. (welcome, home, register and log in).

1.1.1.2 Internal structure

1.1.1.2.1 Functions

- **Flask:** Create an instance of the Flask class for our web app.
- **app.config['SECRET_KEY']:** In order to use session in flask you need to set the secret key in your application settings. secret key is a random key used to encrypt your cookies and save send them to the browser.
- **def get_dbConn():** Function used in order to connect to the Database folder in PostgreSQL.
- **def close_dbConn():** Function used in order to close the connection with the Database folder in PostgreSQL.
- **app.route:** Define the function webpage rout that will be run in.
- **def main():** Definition of the main webpage.
- **def home ():** Definition of the home webpage.
- **def about ():** Definition of the about webpage.
- **def register ():** Definition of the register webpage.
- **RegistrationForm ():** calling the registration form that has been defined in form.py program.
- **request.form:** Accesses the requested object ex. (Username from form).
- **redirect:** Returns a response object and redirects the user to another target location.
- **flash:** Popup a message as one-time alert.

- **Flash(‘f`message{string}’):** Popup a message as one-time alert with a string, as the bootstrap library has been used, success and danger parameters can be applied on the message.
- **render_template:** Reading the html template of the webpage.
- **url_for:** Builds a URL to a specific function. It accepts the name of the function as its first argument and any number of keyword arguments, each corresponding to a variable part of the URL rule.
- **session:** data is stored on server. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in a temporary directory on the server.
- **check_password_hash:** This helper function wraps the eponymous method of Bcrypt
- **generate_password_hash:** Generates a password hash using bcrypt.
- **connect:** Allows database connection.

1.1.2 forms.py

1.1.2.1 Purpose

This program runs the registration as well as the login form using WTFlask. WTFlask not only provide forms but also validation on the form`s fields.

1.1.2.2 Internal structure

1.1.2.2.1 Validator Classes

WTForms package also contains validator class. It is useful in applying validation to form fields.

- **DataRequired:** Checks whether input field is empty
- **Email:** Checks whether text in the field follows email ID conventions
- **EqualTo:** Checks the value is equal to a certain value or
- **Length:** Verifies if length of string in input field is in given range

1.1.2.2.2 Classes

Stores and processes data and generate HTML for a form field. In BIKOD registration and login form have been created ex. (class LoginForm, class RegistrationForm).

- **class LoginForm:** creating a login form API in WTForms.
- **class RegistrationForm:** creating a Registration form API in WTForms.

1.1.2.2.3 Class Field

Fields are used in order to render and convert data in WTForms classes as well as generating HTML for a form field ex. (username, email, password, …etc.). WTforms define carious form field.

- **StringField:** Represents input string element.
- **PasswordField:** Represents <input type = 'password'> HTML form element.

1.1.3 call.py

1.1.3.1 Purpose

This program runs for the connection with database named as “se4g” and the creation of user table which is called as “db_user” in “se4g” database.

1.1.3.2 Internal structure

In this “call.py” script, Psycopg2 adapter is imported and then the parameters of the database such as user, password, host, port and database name are defined.

1.1.3.2.1 Cursor class

cursor = connection.cursor()

Allows Python code to execute PostgreSQL command in a database session. Cursors are created by the connection.cursor() method: they are bound to the connection for the entire lifetime and all the commands are executed in the context of the database session wrapped by the connection.

cursor.execute(create_table_query)

Execute a database operation (in call.py, the operation is creation of a table).

1.1.3.2.2 Create table

This command includes the parameters of the created table. The column parameters specify the names of the columns of the table. The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

1.2 Templates

As shown in BIKOD folder tree shown in Figure 1, the template folder which contains the html files for the webpages shown in Figure 3. The template that flask use is jinja 2. Due to the repetition in the html file templates, creation of a parent and child template ex. layout.html is the parent template for login and register html webpages.

```
+---templates
|   |   about.html
|   |   home.html
|   |   layout.html
|   |   login.html
|   |   register.html
|   |   welcome.html
```

Figure 3: Templates folder for BIKOD

The most common libraries used for styling in the templates are bootstrap and leaflet for the map creation in the home webpage. Some static files have been used and linked in the static folder

1.3 Static

As shown in BIKOD folder tree shown in Figure 1, the static folder which contains java scripts and CSS static files that is used in the templates as shown in Figure 4

```
+---static
|       Autolinker.min.js
|       fontawesome-all.min.css
|       L.Control.Locate.min.css
|       L.Control.Locate.min.js
|       labelgun.min.js
|       labels.js
|       leaflet-control-geocoder.Geocoder.css
|       leaflet-control-geocoder.Geocoder.js
|       leaflet-hash.js
|       leaflet-heat.js
|       leaflet-measure.css
|       leaflet-measure.js
|       leaflet-search.css
|       leaflet-search.js
|       leaflet-svg-shape-markers.min.js
|       leaflet-tilelayer-wmts.js
|       leaflet.css
|       leaflet.js
|       leaflet.js.map
|       leaflet.markercluster.js
|       leaflet.pattern.js
|       leaflet.rotatedMarker.js
|       Leaflet.VectorGrid.js
|       leaflet.wms.js
|       main.css
|       MarkerCluster.css
|       MarkerCluster.Default.css
|       multi-style-layer.js
|       OSMBuildings-Leaflet.js
|       qgis2web.css
|       qgis2web_expressions.js
|       rbush.min.js
|       stations_2.js
|       style.css
|
```

Figure 4: Static folder for BIKOD

2. Database

In the database part of the work, a database is created in **pgAdmin4**, which is a development platform for PostgreSQL. This database is containing two different tables, one is for the user registered and the other is for the number of bikes in each station at a certain time.

The purpose of a database is to store and retrieve information in a way that is accurate and effective. More specifically, in this task, the user database was created in order to keep the information of registered users and use this information for validation during the log in process.

The purpose of creating a database for the bike numbers is to have a dynamic source of information. This database will be used in the statistical part of the work which will supply meaningful information for the users and the admin.

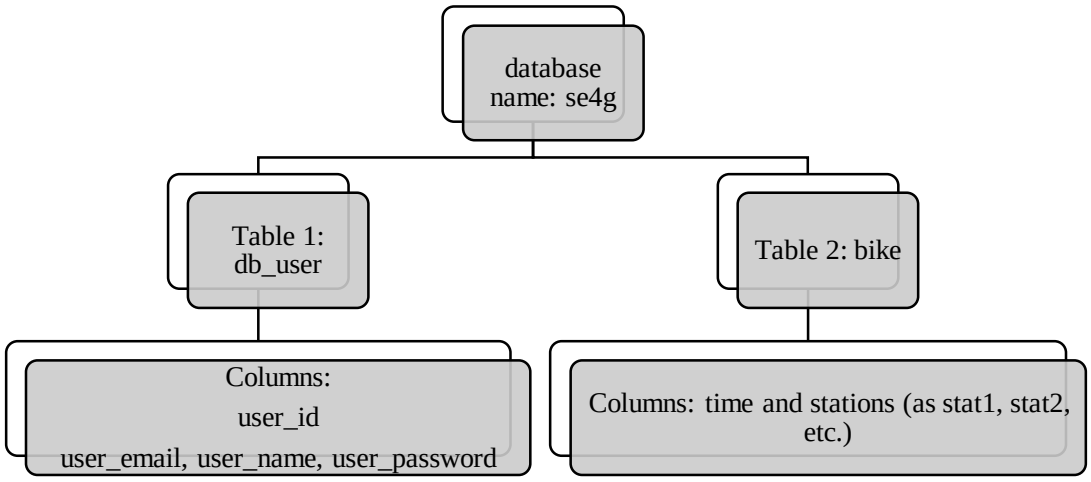

Figure 5: Structure of the BIKOD database

3. Statistical part

This part of the project is still in development phase. However; an introduction will take place here to give an idea about the work planned.

3.1 Bokeh

Several charting libraries are written in python, a language that is more familiar to web developers than most, which makes learning them a less daunting feat. When executed correctly, charting libraries have the power to drive home a powerful message and give us the opportunity to view serious visual candy. However, libraries are never a one-size-fits-all kind of deal. More often than not, our initial assumption that a library is the perfect match will be incorrect. One of those interactive visualization libraries, which targets modern web browsers for presentation is BOKEH.

3.1.1 Purpose

Bokeh is a library primarily intended for creation of visualizations in the browser from large or streaming datasets. These visualizations are created by using Python. Then Bokeh's JavaScript API takes in Python script and renders the plots or charts in addition to handling the UI interactions in the browser, which is one of the purposes in this task to show the stochastics in graph format dynamically. You can also choose to use the Bokeh Server to handle streaming of your data. In the Bokeh 0.12.13 documentation, it states: "This capability to synchronize between python and the browser is the main purpose of the Bokeh Server."

3.1.2 Internal Structure

Bokeh consists of many beneficial modules and libraries that each of them are used for different purposes. Bokeh models are a low level interface that provides high flexibility to application developers.

Some functionalities which will be used

One of them is bokeh.plotting that we will import figure, output_file, and show modules from it to make the desire graph to show or output them in our BIKOD web page (the graph of bike numbers, which varies in each station and time interval that are our variables).
The other modules are ColumnDataSource and Select imported from bokeh.models, for to define the variables (time and station number) of the database to python and also for creating Select Widget. Widgets are interactive controls that can be added to Bokeh applications to provide a front end user interface to a visualization (here we will use this option for selecting a specific station to query the bike number and give value to them in each interaction by updating plots). These prepares a good- functionality for us to query data. Widgets can drive new computations, update plots, and connect to other programmatic functionality.

The next one is importing row from bokeh.layouts, which helps us to create the plot layout with a simple functionality by bar plot and select widget.
The other one is using curdoc imported from bokeh.io, curdoc or current document will eventually hold all the plots, controls and layouts that we created by "curdoc().add_root(layout)" command, which is one of our goals.

Last but not least, for using the functionalities in bokeh we have to import our DataFrame and manipulate it in such a way that would be easy to use, this is under the responsibility of pandas, which is a popular Python package for data science. It offers powerful, expressive and flexible data structures that make data manipulation and analysis easy, among many other things. The DataFrame is one of these structures. By using pandas, we will read our csv data and also we have sliced the data in days and 24 hours each day to prepare them for usage.

Interaction with the main code

By an HTML code for the statistic parts and adding a functionality in the python code we can make a connection between the map visualization page and statistical graphs. On the right side of the web page after logging in or signing in you can see a key named graphs when you click on it, it will connect to the html page related to the bike graphs.

4. Use Cases and Implemented Functionalities

In the RASD file, there are 6 main use cases defined. At this certain point, three of them have been implemented to the project. These use cases are Registration, sign in and Sign out (UC01, UC02, UC03 respectively). The remaining (UC04, UC05, UC06) are in development phase.

UC01: Registration

User can now be registered with their username, email and password. This information can be seen in the database in db_table as updated after every registration.

UC02: Sign in

After the registration, users can use their registered username and password to reach the home page including the map.

UC03: Sign out

Users can return to the welcome page by using this functionality.

5. Organization of team

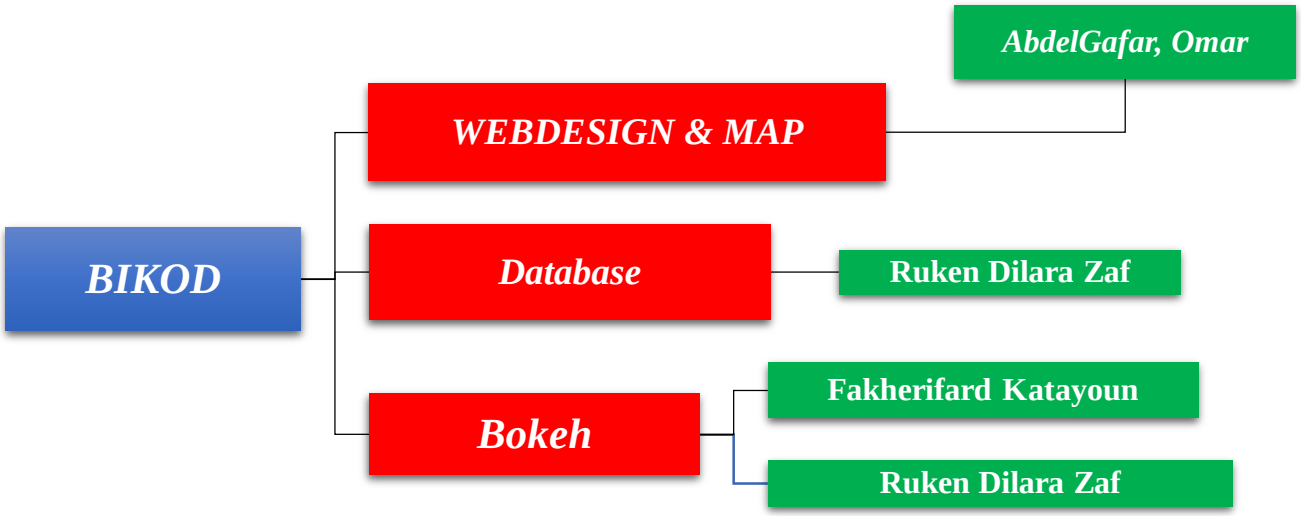

Figure 6 Team Organization