POLITECNICO MILANO 1863

Geoinformatics Engineering

Software Engineering for Geoinformatics

Academic year 2020-2021

Software Design and Test Plan Document

BinEco

Authors:

Elisa Servidio	849417
Dorotea Rigamonti	969365
Mohammad Mahdi Safari	970076
Surendhiran Sundaraj	904909

May 25, 2021

Version 1.0

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

Our project aims to allow PA to create a modern and efficient information system by the help of the citizens, in order to enter, access, retrieve, analyze, visualize and modify, by means of an interactive mapping tool, the available ERP data of a certain local area.

Given data about litter for a certain Public Administration, the main goal is to improve the management of garbage collection in the municipality, for example by putting a bin or an infographic in a critical area (critical areas are found by statistical analysis implemented in the application). A low level goal is to improve the behaviour of people according to the environment and rubbish pollution.

The Web App should be accessible through applied technology (Computer or mobile phone).
The Web App should be available 24h/7.

More detailed information about the project can be found in the RASD document.

The web application will present different kinds of information, both static and dynamic. By static the purpose is to display all of the information that will not change with the user interaction, it will be stored as HTML code and will be served when requested by the client. Dynamic refers to all of the information that the user will be able to interact with and which changes upon request, all of this is developed using Python language.

1.2 Software Design Document

The Design Document provides a specification on the architecture of BinEco's system. It is complementary to the Requirements Analysis and Specification Document already presented, and it provides further description on its components, their interactions and the implementation, integration and testing plan.

This document will describe the following characteristics of the project:

- *Project Database*: Since one of the most important parts of this project is the data management and structure, it is also one of the most complex and composed of several elements. For this reason a detailed explanation of the Database characteristics will be described.
- *Software Structure*: as it will be specified in this section, the software is structured in a 3-layer architecture which will make possible the interaction between the client and the server taking into consideration the static and dynamic units.
- *User cases application*: As it can be seen in further detail in the RASD, it can be seen how use cases or requirements map on the components of the software.
- *Team Organization*: the development team is composed of 4 people, and although each one of the members will participate in building all of the project, it is important to assign roles and activities to each engineer. This will be discussed in this section.
- *Test Plan*: detailed document that describes the test strategy, objectives and schedule required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test.

The Design Document is connected to the Requirements Analysis Specification Document (RASD) by complying with all the requirements established and in order to avoid missing any important functionality.

1.3 Reference Documents

- IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-intensive Systems
- IEEE 1016-2009, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

Chapter 2

BinEco Database

The web app retrieves databases from different external services (which here are Epicollect5, PostgreSQL and OSM) through REST API.

Data from Epicollect5 are pre-processed and copied to a PostgreSQL database with a given frequency.
Data about bins' distribution in a certain locality are retrieved from OSM and copied to a PostgreSQL database referring to a certain municipality.

The web app will then interact with DBMS and perform CRUD operations on the PostgreSQL database.

The advantages for storing the data in a PostgreSQL database as opposed to fetch them directly from Epicollect5 include:

- to enable verification, pre-processing and storing of consistent data
- to ensure availability of data, decoupling our web application from EpiCollect5
- to reduce the risk of data loss
- to improve performance
- to leverage DBMS capabilities and, in particular, the interface between Python and PostgreSQL

In this section we describe both the structures of the table fetched from Epicollect5, also referred to as the Litter table, and of those in PostgreSQL.

2.1 Epicollect5 Dataset

The dataset of environmental rubbish pollution measurements that will be used in this project comes from Epicollect5 and contains data collected in

Australia.
The dataset consists of 58 georeferenced measurement points of ERP geolocalized in the municipalities of Cairns and Townsville.

Dataset: `https://five.epicollect.net/project/bineco-web-application/data`

We retrieved data from an already existing project whose main features are shown in *Table 2.1* and we modified the attributes of the dataset according to the main purpose of our application.
Eventually, we created a new project called *Bineco Web Application* whose main features are summarized in *Table 2.2.*

Table 2.1: Original Epicollect5 project's table

Attribute	**Description**
Point position	expressed in WGS84
Date of creation	expressed in m/dd/yy h:min
Litter type	general information about material of garbage like metal, plastic, etc.
Count	number of piece of garbage
Description (e.g. string)	indication of the type of object
Brand (e.g. MCDonalds)	information about the brand of the litter
Litter preservation	general information about the status of preservation of the litter (degraded, slightly worn, pristine)
Size	three categories based on length of garbage (small, medium, large)
Type Of Infrastructure	the type of place where it was found (e.g. picnic table). It helps to reduce the buffer that using GPS implies (GPS provides geographical coordinates that are not necessarily so precise))
Comment	section not mandatory to be filled in
Photo	a photo of the piece of garbage

Table 2.2: Bineco Web Application project's table

Attribute	**Description**
Point position	expressed in WGS84
Date of creation	expressed in m/dd/yyyy
Time of creation	expressed in h:min:sec
Litter type	general information about material of garbage like metal, plastic, etc.
Quantity	general quantity of litter (low/medium/high)
Type Of Infrastructure	the type of place where it was found (e.g. picnic table). It helps to reduce the buffer that using GPS implies (GPS provides geographical coordinates that are not necessarily so precise))
Comment	section not mandatory to be filled in
Photo	a photo of the piece of garbage

Instead of counting litter Citizens (who are the users in charge to collect data) will have to enter a general quantity. This will make them much more willing to collect data due to the fact that it will take less time to do it.

Description, brand, litter preservation and size attributes are no longer useful for our web application given that they don't provide any valuable piece of information to the PA according to the purpose of our project.

Additional fields generated automatically by EpiCollect5 when the user uploads a data entry include:

- EpiCollect5 unique ID
- creation date and time
- upload date and time

Even if the creation date and time are automatically generated by Epicollect5, we had to add the attributes

- Date of creation
- Time of creation

to simulate Citizens collecting data over an entire month in the past. This mechanism allows us to make statistical analysis which exploit data collected in a certain amount of time.

2.2 PostgreSQL - PostGIS Database

The web application running on the WSGI server will interact with a DBMS for data storing and management. The DBMS of the web application is PostgreSQL 13 DBMS that works on the local machine.

PostgreSQL is a free and open-source DBMS emphasizing extensibility and SQL compliance, and to leverage database adapters for Python programming language, for example psycopg2.

Moreover, PostgreSQL provides useful extensions depending on the specific nature of data. Given that data entries are georeferenced we have decided to exploit PostgreSQL's extension PostGIS, an open source software program that adds support for geographic objects to the PostgreSQL object-relational database.

The PostgreSQL - PostGIS database will be composed by data organized in tables with primary keys which are unique identifiers, those elements that allow us to identify a specific piece of data without any ambiguity. Data are logically related even from a practical point of view by usage of specific identifiers that occur in more than one table as follow:

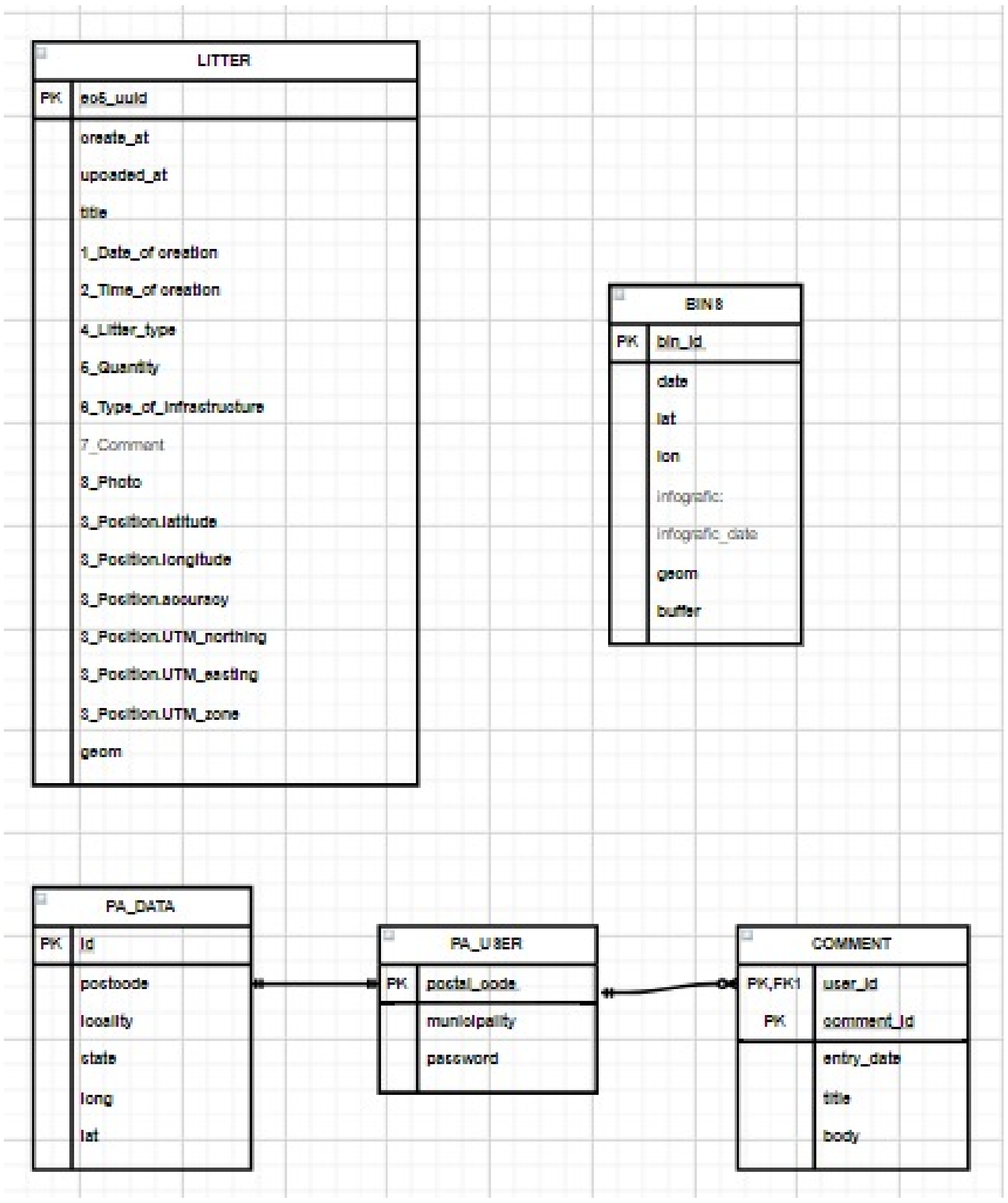

Figure 2.1: Entity Relationship Diagram

BinEco data tables specifications are reported in the next pages.

Table 2.3: Litter table

Attribute	**Description**
ec5_uuid	integer value uniquely identifying the entry. This will be also the table key value
1_Date_of creation	class datetime.date variable representing the date of the entry
2_Time_of creation	class datetime.time variable representing the time of the entry
3_Position.latitude	float value representing the geodetic Latitude (in degrees) of surveyed point
3_Position.longitude	float value representing the geodetic Longitude (in degrees) of surveyed point
3_Position.accuracy	float value representing the UTM projected East coordinate (in meter)
3_Position.UTM_easting	float value representing the UTM projected East coordinate (in meter)
3_Position.UTM_northing	float value representing the UTM projected North coordinate (in meter)
3_Position.UTM_zone	string value representing the UTM zone of the projected coordinates reference system
4_Litter_type	enumeration (plastic, glass,...)
5_Quantity	enumeration (low, medium, high)
6_Type_of_infrastructure	enumeration type (picnic table, bin..)
7_Comment	string value
8_Photo	
geom	a PostGIS geometrical attribute, of type POINT, containing the geodetic coordinates (Lon., Lat.) of the surveyed point

Bins' distribution table stores information about the bin creation, its position and the infographic updates. It refers to a certain municipality for which we retrieve data from OSM through API to make our application help PA in the decision making process.

Table 2.4: Bins' distribution table

Attribute	**Description**
bin_ID	integer value uniquely identifying the bin. This will be also the table key value
bin_date	class datetime.date variable representing the date of the bin installation
lat	float value representing the geodetic Latitude (in degrees) of surveyed point
lon	float value representing the geodetic Longitude (in degrees) of surveyed point
infographic	boolean variable representing the presence or not of an infographic
infographic_date	class datetime.date variable representing the date of the infographic installation
geom	a PostGIS geometrical attribute, of type POINT, containing the geodetic coordinates (Lon., Lat.) of the surveyed point
buffer	a PostGIS geometrical attribute of type POLYGON

PA users table contains the system users' information needed for registration and login.

Table 2.5: PA users table

Attribute	**Description**
postcode	string value uniquely identifying the PA. This will be also the table key value
municipality	string value representing the name of the municipality of PA
password	string value representing the password of the PA user account

PA postcodes table allows BinEco system to check if registration or login is made correctly by PA.

Table 2.6: PA postcodes table

Attribute	**Description**
id	integer value uniquely identifying the municipality. This will be also the table key value
postcode	string value uniquely identifying a municipality
locality	string value representing the name of the municipality
state	string value representing the abbreviation of states
lat	float value representing the geodetic Latitude (in degrees) of surveyed point
long	float value representing the geodetic Longitude (in degrees) of surveyed point

Table 2.7: Comment's table

Attribute	**Description**
user_id	string value uniquely identifying the PA. This will be also the table key value
comment_id	string value uniquely identifying the comment
entry_date	class datetime.date variable representing the date of the creation of the comment
title	string value representing the title of the comment
body	string value representing the body of the comment

Chapter 3

Software Structure

The software's architecture defines its organization in terms of components and connectors.

BinEco software's architecture will consist of a 3-layer system composed of a Web Server (HTTP Server), Logical Server (Application Server) and Database Server.

The software is divided into two independent programs:

- database configuration script in charge of creating the structure of the database
- the flask application script in charge of managing:
  - the template engine
  - the python code which provide all the logical operations that will be needed in order to fulfil the software requirements

3.1 Database configuration

Business logic relies on a database server where data are stored persistently and manipulates them, making them available for the client. The WGSI server connects to this database.

BinEco application will interact with the DBMS of PostgreSQL by using the libraries:

Table 3.1: Libraries and Functions imported for Database Configuration

Library	**Function**	**Description**
psycopg2	connect	creates a new database session and returns a new instance of the connection class
sqlalchemy	create_engine	produces an Engine object based on a URL

The main operations for interacting with a DBMS are:

- Opening a connection with the server
- Sending the request to execute a SQL command
- Sending the request to commit any change
- Closing the connection

3.1.1 Data Preprocessing

BinEco script for the database configuration is in charge of retrieving and preprocessing the initial data which are going to be stored in the database.

The main libraries to preprocess data are:

Table 3.2: Libraries imported for Data Preprocessing

Library	**Description**
requests	allows you to send HTTP requests using Python
json	used to encode the raw response text into a JSON variable
pandas	it offers data structures and operations for manipulating numerical tables and time series
geopandas	used to create geodataframe of the data retrieved
shapely.wkb	used to convert into WKB format the geometry column

3.1.1.1 Epicollect5 litter dataset

1. Data are retrieved from Epicollect5 through REST API provided by the web service, sending it a request. To do this, Requests library is needed.

2. raw data are converted into a json file

3. Data are transformed into a Pandas DataFrame.

4. A new GeoDataFrame is created by adding a new geometry attribute of Point-type to the DataFrame. A GeoDataFrame object is a DataFrame that has a column with geometry, then the GeoData are saved in the variable.

Table 3.3: Epicollect5 litter dataset preprocessing specification

Used libraries	Pandas GeoPandas Requests JSON
Inputs	URL of EpiCollect5 REST API List of attributes of EpiCollect5 dataset that are relevant for this app and will be stored in the DataFrame
Output	GeoPandas GeoDataFrame that contains data previously retrieved from EpiCollect5

3.1.1.2 Municipalities' postcodes dataset

1. Data are retrieved from
   `https://www.matthewproctor.com/australian_postcodes`
   in .cvs format and saved in the folder 'data' attached to the software.
2. Data are directly saved into a pandas dataframe.

Table 3.4: Municipalities' postcodes dataset preprocessing specification

Used libraries	Pandas
Inputs	df_australia_postcode.csv List of attributes of the dataset that are relevant for this app and will be stored in the DataFrame
Output	Pandas DataFrame that contains preprocessed municipalities postcodes data

3.1.1.3 Bins' distribution dataset

1. Bins's distribution data are retrieved in geojson format from OSM by using QuickOSM in Qgis/using the osm api Overpass and we store it in the 'data' folder.

2. The geojson file is directly assigned to a geopandas dataframe from which we extract and create all the attributes needed to fulfill the table of the bin.

3. The geometry column is converted into WKB format because is faster to be stored.

Table 3.5: Bins' distribution dataset preprocessing specification

Used libraries	geopandas shapely.wkb
Inputs	Waste_basket_Cairns.geojson Waste_basket_Townsville.geojson List of attributes of the dataset that are relevant for this app and will be stored in the GeoDataFrame
Output	GeoPandas DataFrame that contains preprocessed bins distribution data

3.1.2 Data import into PostgreSQL

Once data are preprocessed, they are imported to PostgreSQL database tables.

An engine or a connection is set up for managing interaction with the DBMS. Once the application runs for the first time new table are created in PostrgreSQL DB and data are copied into the corresponding tables.

Once data previously retrieved from Epicollect5 are copied and stored in the corresponding PostgreSQL DB table, if any new entry is found in the dataset from Epicollect, it will be automatically appended to the PostgreSQL DB table.

The main characteristics of Data import into PostgreSQL are described in the table below:

Table 3.6: Data import into PostgreSQL specification

Used libraries	sqlalchemy Psycopg2 geopandas
Inputs	GeoPandas GeoDataFrame containing the preprocessed data Parameters (database name, user, password, etc.) to connect to the DBMS
Output	PostgreSQL-PostGIS database table with varchar, numerical, boolean and geometry attributes

3.2 Flask application

FLASK is a set of Python's libraries that allows us to build a web application that will be running in the context of a WSGI server.
It is classified as a microframework which means that, compared to other approaches, is simpler because it contains only the necessary tools that are strictly needed to support us in the interaction with WSGI server.

The Flask application includes the template part and the Python part. The application server will provide all the logical operations that will be needed in order to fulfil the software requirements. We can write business logic of a web server using Python, relying on the standard WSGI which our web server must know in order to be able to execute Python code.

The Web browser interacts with the Web server. Python web application will be running in the context of WSGI. The interaction between WSGI server and Web server is defined according to WSGI specification.

In the next sections we will describe all the principal logical functions and the interactions between the various components.

3.2.1 Python Code

Purpose of the Python code is to answer all the requests coming from the user through the web server, with whom it interfaces through the WSGI, to collect inside all the available functions and recover all the data needed by the database server.

Table 3.7: Libraries and Functions for implementation of Python Code

Library	**Function**	**Description**
flask	Flask	creates an application instance
	Render_template()	Helper function that allows use of the JINJA template engine
	Request	
	Redirect	Returns a response object (a WSGI application) that, if called, redirects the client to the target location
	url_for()	Generates a URL to the given endpoint with the method provided
	flash()	Flashes a message to the next request. In order to remove the flashed message from the session and to display it to the user, the template has to call get_flashed_messages()
	session	
	g	It is a global namespace for holding any data during a single app context
	Abort	to abort a request early with an error code
psycopg2	connect	
werkzeug.security	check_password_hash	This allows us to check whether the password provided matched the hashed_value that is stored in the DB

	generate_password_hash	This allows us to store hashed values of passwords in the database, instead of storing them in clear
werkzeug.exceptions	abort	This function is used to terminate the process if a certain condition is met, it returns the chosen HTTP error
shapely.geometry	point, polygon	create geometric objects
geopandas	used to create geo-dataframe of the data retrieved	
pandas	it offers data structures and operations for manipulating numerical tables and time series	
seaborn	statistical data visualization library	
bokeh	server_document	functions for embedding Bokeh standalone and server content in web pages
	subprocess	to allows the bokeh app running on port 5006 to be accessed by Flask at port 5000

Once all the packages and module are imported, we create a global array which contains three different constant values corresponding to three different thresholds. In our app logic are present all the functions defined for each use case and need that the user can request in his session. First there is the creation of the application instance through the Flask function, to which we pass as argument the name and the template folder that we have implemented for our app; so we set a secret key to some random bytes.

Then we define all the functions necessary for the implementation of our application.

3.2.1.1 Functions

Database connection functions

- **get_dbConn**: if we are not connected yet, we pass to the function the .txt file of our DB configuration containing dbname, username and password, the function reads (*readline*) the text, saves the string and connects with our db
- **close_dbConn**: if connected, the function disconnects from our db, closing the connection and removing the attribute (*pop*)

For each of the functions present in the code which fall in cases in which the is the user that can do a request to the server, we preliminary declare the *@app.route* to map the specific URL with the associated function that is intended to perform some task; with *@app.route('/')* we create a URL route in our application and if we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen; *route()* decorator tell Flask what URL should trigger our function.

By default a route only answers to GET requests, so in order to the function we are implementing and the case we are in we use the methods argument of the route decorator to handle different HTTP methods (GET, POST, if present in the template).

Registration function: *register()*

Table 3.8: *register()* function

Used libraries	**flask**: *redirect()*, *flash(error)*, *url_for()*, *render.template()*, *request*
	werkzeug.security: *generate_password_hash()*
	psycopg2: *fetchone()*, *close()*, *cursor()*, *esecute()*, *commit()*
Used functions	*get_dbConn()*
What it does	it is in charge of connecting to the page of the registration, taking the entries of a new user and in case of correctness of the inputs, connects to the database and creates a new entry. Then redirects to the login.html page. Otherwise it shows an error message and redirects to the register.html page. if the user is already present in the database, it redirects him/her to the login.html page

Login function: *login()*

Table 3.9: *login()* function

Used libraries	**flask**: *redirect()*, *flash(error)*, *url_for()*, *render.template()*, *request*, *session*
	werkzeug.security: *check_ password_ hash()*
	psycopg2: *fetchone()*, *close()*, *cursor()*, *esecute()*, *commit()*
Used functions	*get_ dbConn()*
What it does	It is in charge of connecting to the *login.html* page, taking the entries of a user and in case of correctness of the inputs create a new session and connect to the *index.html* page. Otherwise it shows an error message and redirects to the *index.html* page

Logout function: *logout()*

Table 3.10: *logout()* function

Used libraries	**flask**: *session.clear, g, redirect(), url_for()*
What it does	close the session and redirect to the *about.html* page

Cookies function: *load_ logged_ in_ user()*

Table 3.11: *load_ logged_ in_ user()* function

Used libraries	**flask**: *session.clear()*, *g*, *redirect()*, *url_for()*
	psycopg2: *fetchone()*, *close()*, *cursor()*, *esecute()*
Used functions	*get_ dbConn()*
What it does	It is in charge of verifying and memorizing the presence of a logged user in the website. It get the *use.id* of the session if it is *none* set the global variable *g.user* to *nan* and return *false*; if it is not *nan*, it set the global variable *g.user* to the value in the database and return *true*

Create the customized home page function: *index()*

Table 3.12: *index()* function

Used libraries	**flask**: *render.template()*
Used functions	*load_ logged_ in_ user()*
What it does	It is in charge of connecting to the *index.html* page if the user is logged, otherwise it connects to the *about.html* page

Select temporal window: *query_ temp()*

Table 3.13: *query_ temp()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*
	geopandas
Used functions	*get_ dbConn()*, *now()*
Output	new geodataframe
What it does	it is in charge of creating a new geodataframe considering a certain time lapse. The temporal window is defined starting from the day when calling the function considering the previous 30 days.

Query data by area: *queryByArea()*

Table 3.14: *queryByArea()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*
	geopandas
Used functions	*get_ dbConn()*
Input	a specific area polygon type
Output	new geodataframe
What it does	create a new dataframe by selecting all the raw where the geometry type point is contained in the input area.

Data statistical analysis function: *analysis()*

Table 3.15: *analysis()* function

Used libraries	**NumPy**
	geopandas
Used functions	*queryByArea()*
Input	a specific area polygon type
	id_bin
	global array with constant variables corresponding to thresholds
Output	boolean value
	array with absolute frequency of the different quantity values (low, medium, high)
What it does	starting from the points selected by the *query_temp()* function, it computes the absolute frequency on a specific set of points of litter extracted by using a specific area. If the id of the bin belongs to area (if *id_ bin* is not *none*) • if the computed values overcome the correspondent threshold the boolean variable is set to true • otherwise returns the array of statistics

Update a bin with infographic function: *update_ bin()*

Table 3.16: *update_ bin()* function

Used libraries	**psycopg2**: *close()*, *fetchone()*, *cursor()*, *execute()*, *commit()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*
Input	ID of the bin (on the click of the map)
Output	message that the operation is done
What it does	it is in charge of connecting to the *updateBin.html* page. If the user is logged, the function takes the new information about the bin infographic. In case of correctness of the inputs, it connects to the dataset and updates the raw. Then redirect to the *index.html* page. if the user is not logged in it shows an error message and redirects to the *login.html* page.

Visualize statystical analysis result function: *visualize_ result()*

Table 3.17: *visualize_ result()* function

Used libraries	**matplotlib**
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*analysis()*
Input	area(selected by the user on the interactive map), thresholds
Output	histogram plot
What it does	given a selected area and the threshold, allows the user to visualize an histogram that helps the user in the decision making process of putting a new bin

Data interactive visualization function: *interactive_ map()*

Table 3.18: *interactive_ map()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
	bokeh
Used functions	*query_ temp()*, *analysis()*, *get_ dbConn()*, *update_ bin()*, *visualize_ result()*
Output	interactive map
What it does	it allows the user to access the visualization processed data customized with some statistical analysis and interact with the map and data on it.

Insert a new bin function: *new_ bin()*

Table 3.19: *new_ bin()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*, *load_ logged_ in_ user()*
What it does	it is in charge of connecting to the *newBin.html* page. Check if the user is logged in. If yes, it takes the entries of a new bin and in case of correctness of the inputs connect to the dataset and create a new entry. Then redirect to the *index.html* page. Otherwise it shows an error message and redirects to the login.html page

Create the ‘homepage’ function for the comment section: *help_ us()*

Table 3.20: *help_ us()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*
What it does	it is in charge of connecting to the *help_ us/index.html* page showing all the comments of the users which it retrieves by connecting to the database.

Leave a comment function: *add_ comment()*

Table 3.21: *add_ comment()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*, *load_ logged_ in_ user()*
What it does	it is in charge of connecting to the *help_ us/add_ comm.html* page. Check if the user is logged in. If yes, it takes the entries of a new post and in case of correctness of the inputs connect to the dataset and create a new entry. Then redirect to the *help_ us.html* page. Otherwise shows an error message and redirect to the page *help_ us/add_ comm.html* if no, it shows an error message and redirects to the *login.html* page

Request information about a comment: *get_ comment()*

Table 3.22: *get_ comment()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*
Input	the id of a comment
What it does	given the id, if a post associated exist and the user(saved in the global variable) is the author, then it get all the information about the comment and return the comment otherwise shows an error message

Delete comment function: *delete_ comment()*

Table 3.23: *delete_ comment()* function

Used libraries	**flask**: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*
Input	Id of a comment
What it does	given the id of a comment, connect to the database and delete the correspondent raw

Update comment function: *update_ comment()*

Table 3.24: *update_ comment()* function

Used libraries	**psycopg2**: *close()*, *cursor()*, *execute()*, *commit()*, *fetchone()*, *abort()*
	flask: *redirect()*, *flash()*, *url_for()*, *render.template()*, *request*
Used functions	*get_ dbConn()*, *load_ logged_ in_ user()*, *get_ comment()*
Input	the id of a comment
What it does	given the id of a comment, checks if the user is logged in if yes, it retrieves from the database all the info about a comment with *get_ comment()* in the get method it connects to the *help_ us/update_ comm.html* page with the past info about the post in the post method it asks the user to insert new data and if they are all correct connect to the database and update it then redirect to *help_ us.html* page otherwise shows an error message and redirect to the page *help_ us/update_ comm.html* if no, it shows an error message and redirects to the *login.html* page

3.2.2 Template Engine

Purpose of the template engine is to generate desired content types, such as HTML, while using some of the data and programming constructs such as conditionals and for loops to manipulate the output. Template files that are created by developers and then processed by the template engine consist of prewritten markup and template tag blocks where data is inserted.

3.2.2.1 Jinja template engine

The web application mainly uses HTML to display information for the clients, so for this purpose we have to integrate codes in HTML files. JINJA Engine is a component that can be integrated within the context of Flask application, that receives as input a template that defines the structure of a Web Page and some specific values that need to be incorporated in that page and provides as an output the html document completed with the values that are given as input.

A template defines the structure of our html Page and is a file containing variables and expressions that are replaced with values when the template is rendered. At the time of the creation, we don't know exactly what the values will be. At run time JINJA engine is called by our Flask application, the template and the values are passed to JINJA Engine and then the html page is generated. Templates won't execute without an app (Flask application) to serve them. The templates control the overall look and layout of a site.

- **base.html**: it is considered as a main structure that points to all others. In the language of JINJA engine the **base.html** is referred to as a parent. it will display the main layout of the web application and all the components
- **about.html**: the starting page for the users which are not logged in. it shows information about the web app, what it does and to join the project.
- **index.html**: It is the user home page of the web Application where users (registered and logged in) can see the information about the web app
- **register.html**: Users use this form to register themselves into the Web Application by providing their Data such as Username, Municipality, and Password.
- **login.html**: Registered users use this form to log-in with their credentials.
- **newBin.html**: allows the user to insert a new entry in the bin table
- **updateBin.html**: allows the user to update the infographic attribute of the bin table
- **interactive_map.html**: allows the user to visualize data on an interactive map and interact with the objects present in it
- **visualize_results.html**: allows the user to visualize an histogram with the result of a statistical analysis
- **help_us/index_comm.html**: allows to visualize the page with all the comment and leave new one
- **help_us/add_comm.html**: Users use this form to create a new comment
- **help_us/update_comm.html**: Users use this form to update a specific comment which means edit it or delete it

3.2.2.2 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 code manages the style and the appearance of the template web pages.

Chapter 4

Use Cases Application

4.1 UC1 *PA registration*

- **@app.route('/register', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to
    - Get the information sent by the user client (name of the municipality, username and password)
    - Check if the username exists in the municipalities' postcode database
    - If not, check if the username and the municipality correspond
      - if yes accept the new registration and direct the user on the login page
      - if not, send an error message to the user
    - If yes, send an error message to the user
  - On GET, it has to send the user on the registration page

4.2 UC2 *PA logs in*

- **@app.route('/login', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to
    - Get the information sent by the user client (username and password)
    - Check if the username exists in the database
    - If not, send an error message to the user
    - If yes, store the user_id in the session variable
  - On GET, it has to send the user on the login page

4.3 UC3 *PA enters new bin data*

- **@app.route('/newBin', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to
    - Get the information sent by the user client(ID of the bin, Lat, Lon)
    - Check if it already exists in the database
    - If not, call the function to update the database
    - If yes,send an error message to the user
  - On GET, it has to send the user on the newBin page

4.4 UC4 *PA visualize data on interactive map*

- **@app.route('/Map', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to
    -
  - On GET, it has to send the user on the popup "page" of the selected point

4.5 UC5 *PA requests information about data*

- **@app.route('/Map/popup', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to
    - Get the information sent by the user client (click on one point)
  - On GET, it has to send the user on the popup "page" of the selected point

4.6 UC6 *PA updates infographic*

- **@app.route('/updateBin', methods=('GET', 'POST'))**
- The function must answer to two HTTP requests
  - On POST, it has to

    * Get the information sent by the user client(1 in the infographic attributes)
    * Check if it has the same value in the database
    * If not, call the function to update the database and send a message when done
    * If yes,send an error message to the user
  - On GET, it has to send the user on the updateBin page

4.7 UC7 *PA visualizes statistical analysis results*

4.8 UC8 *PA adds a comment*

4.9 UC9 *PA deletes a comment*

4.10 UC10 *PA updates a comment*

4.11 UC11 *PA logs out*

Chapter 5

Team Organization

Chapter 6

Test Plan

6.1 Scope

This test is only a functional testing, more precisely a API test with which we are going to test all the functions present in the application. Out of scope items are Database testing, hardware and any other external interface.

6.2 Test Cases

6.2.1 UC1 *PA registration*

6.2.1.1 TC1 *Correspondent locality and postcode*

Table 6.1: **UC1** - *TC1*: Correspondent locality and postcode

Inputs	user accesses to the registration page and inserts: postcode: 4870 municipality: Cairns password: yy3322yy
Hypothesis	4870 postcode exists in the system Cairns is the correspondent city
Expected results	the system brings the user in the login page

6.2.1.2 TC2 *Postcode doesn't exist*

Table 6.2: **UC1** - *TC2*: Postcode doesn't exist

Inputs	user accesses to the registration page and inserts: postcode: 12345667 municipality: Sydney password: yy3322yy
Hypothesis	12345667 postcode does not exist in the system
Expected results	the system informs the user that the postal code or municipality is incorrect

6.2.1.3 TC3 *Wrong post_ code or password*

Table 6.3: **UC1** - *TC3*: Wrong post_code or password

Inputs	user accesses to the registration page and inserts: postcode: 4870 municipality: Sydney password: yy3322yy
Hypothesis	4870 postcode exists in the system Cairns is the correspondent city
Expected results	the system informs the user that the postal code or municipality is incorrect

6.2.2 UC2 *PA logs in*

6.2.2.1 TC1 *Correct username and password*

Table 6.4: **UC2** - *TC1*: Correct username and password

Inputs	user accesses to the login page and inserts: username: 4870 password: yy3322yy
Hypothesis	4870 postcode exist in the system yy3322yy is the password
Expected results	the system brings the user in the index page and welcome him/her

6.2.2.2 TC2 *Wrong username or password*

Table 6.5: **UC2** - *TC2*: Wrong username or password

Inputs	user accesses to the login page and inserts: username: 4870 password: xx4455xx
Hypothesis	4870 postcode exist in the system yy3322yy is the password
Expected results	the system informs the user that the username or password is incorrect

6.2.2.3 TC3 *Username doesn't exist*

Table 6.6: **UC2** - *TC3*: Username doesn't exist

Inputs	user accesses to the login page and inserts: username: 12345667 password: yy3322yy
Hypothesis	12345667 username does not exist in the system
Expected results	the system informs the user that the username or password is incorrect

6.2.3 UC3 *PA enters new bin data*

6.2.3.1 TC1 *correct format*

Table 6.7: **UC3** - *TC1*: Correct format

Inputs	user accesses to the newBin page and inserts: lon: 145.742303 lat: -16.923761 ticks 'no'
Hypothesis	lon belongs to [0, 360) lat belongs to [-90, 90) a infographic choice is selected
Expected results	the system informs the user that the operation is done

6.2.3.2 TC2 *Wrong lon/lat*

Table 6.8: **UC3** - *TC2*: Wrong lon/lat

Inputs	user accesses to the newBin page and inserts: lon: -45.742303 lat: -16.923761 ticks 'no'
Hypothesis	lon belongs to [0, 360) lat belongs to [-90, 90) a infographic choice is selected
Expected results	the system informs the user that the lat/lon is incorrect

6.2.3.3 TC3 *Lon/lat/infographic is nan*

Table 6.9: **UC3** - *TC3*: Lon/lat/infographic is nan

Inputs	user accesses to the newBin page and inserts: lon: -45.742303 lat: -16.923761 no tick
Hypothesis	lon belongs to [0, 360) lat belongs to [-90, 90) a infographic choice is selected
Expected results	the system informs the user that the attribute is mandatory

6.2.4 UC4 *PA visualize data on interactive map*

6.2.4.1

6.2.5 UC5 *PA requests information about data*

6.2.5.1

6.2.6 UC6 *PA updates infographic*

6.2.6.1

6.2.7 UC7 *PA visualizes statistical analysis results*

6.2.7.1

6.2.8 UC7 *PA adds a comment*

6.2.8.1

6.2.9 UC8 *PA deletes a comment*

6.2.9.1

6.2.10 UC9 *PA updates a comment*

6.2.10.1

6.2.11 UC9 *PA logs out*

6.2.11.1

Bibliography

[1] Elisabetta Di Nitto (2021), *Software Engineering for Geoinformatics - Lectures.*