

Software Engineering for Geoinformatics 2020

6th of June, 2020

Software Design & Test Plan Document

Web-Based Geospatial Data Analysis
Application on Environmental Noise Pollution
Survey in Pune, India

Version: 2.0

Authors:

R. Cedeno, D. Cumming, H. Mahmoudi, A. Zacchera

Table of Contents

Table of Contents	1
1. Introduction	3
1.1 Design Document	3
1.2 Product Description	4
2. Project Database	5
2.1. EpiCollect5 dataset	5
2.2. PostgreSQL database	6
3. Software Structure	8
3.1. HTTP Server	8
3.2. Application Server	8
3.2.1. Data retrieval from EpiCollect5 and preprocessing	9
3.2.2. Data import into PostgreSQL	10
3.2.3. Mapping Tool	11
3.2.4. Filtering Management	12
3.2.5. Geospatial charts & Statistics	13
3.2.6. Template Engine	14
3.2.6.1. Flask application	14
3.2.6.2. JINJA Template engine	15
3.2.6.2. CSS codes	17
3.2.7. Interaction between application server components	17
3.3. Database Server	19
4. User Cases Application	20
UC1: Discover about noise pollution and how to contribute in data collection	20
UC2: Data points visualization	20
UC3: Map data with a custom visualization	21
UC4: Data analysis of the noise pollution	22
UC5: Repeatedly visualization-filtering requesting	24
UC6: Someone added invalid data to the Epicollect5 project	25

5. Organization	26
6. References	26

1. Introduction

1.1 Design Document

Throughout the design phase of Software Engineering and during the development of any significant project, organization is a key ingredient that makes a difference between a successful project and one that fails. Both structure and organization will help the development team to clarify each of the steps and design decisions made during the whole life of the project. Having said this, one of the most important steps is to develop a Design Document.

It is important to emphasize in the fact that this document builds upon the Requirements Analysis Specification Document (RASD), written by Rodrigo Cedeño, Diego Cumming, Homeyra Mahmoudi and Alessandro Zacchera, written in 2020 which can be found in the following link: https://github.com/azacchera/SE4G_Group_01/blob/master/Documentation/RASD_Group1_20200420_v1.0.pdf)

In essence, the design document is a fundamental piece that will guide the project developers and team members along the subsequent development path of a piece of software, this will include the implementation, integration and testing of system components. Every team and person involved in the project must have knowledge of its contents. According to the IEEE having design documentation is important for the following reason:

"Is an integral part of the software development process, and a vital source of information. Technical documentation is a means to make knowledge about a software system explicit that would otherwise only remain implicit knowledge in the heads of the development team that easily gets lost over time Further, documentation presents information at a higher level of abstraction than the system implementation, which makes it an important means for communication among stakeholders."

In other words, it will provide the design blueprints as a structure in which the project will be carried out by describing the solution of the current problem and requirements that it has. This important part will help to establish and define characteristics that would not be discussed or described in another way and will serve as a guide for any future steps. It is also important to mention that 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.

It is important to mention that the Design Document actually is a technical description, which is not focused directly to be read and interpreted by the client or stakeholders, but if these are expert enough or have sufficient technical knowledge, they can take advantage to understand the system and have more control in the maintenance and implementation phases. This will help the development flow to be more efficient, due to the fact that the client would be able to request changes in earlier phases.

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 it 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.
- 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.

1.2 Product Description

The purpose of developing this product is to inform and to involve communities potentially affected by environmental noise pollution (ENP) by means of a web application for desktop. This web-application will allow users to access, retrieve, analyze and visualize, through an interactive mapping tool, the available ENP data of a certain area. On the website, the user will also be able to find general information regarding the issue of noise pollution, data description and instructions on how to contribute to the data collection since this project aims not only to inform people but also to involve local communities.

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.

This document builds upon the Requirements Analysis Specification Document.

2. Project Database

The data for this project, including information about point positioning, noise level etc., will be retrieved from EpiCollect5, pre-processed and copied to a PostgreSQL database with a given frequency. 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: i) to enable verification, pre-processing and storing of consistent data; ii) to ensure availability of data, decoupling our web application from EpiCollect5; iii) to reduce the risk of data loss; iv) to improve performance; v) 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 Noise Pollution Survey table, and of that in PostgreSQL. For a discussion about the interface between the application and databases and about the data fetching, pre-processing, queries, etc. refer to Section 3.

2.1. EpiCollect5 dataset

This description of the Noise Pollution Survey dataset in EpiCollect5 constitutes a snapshot of data as they are available on 5 May 2020; changes to the dataset size or attributes might occur between this date and the implementation phase, although no such change has been observed in the last month, a fact that has lead us to conclude that the surveying of noise pollution in Pune was halted.

The dataset consists of 971 measurement points of environmental noise pollution. The data are geolocalized in a certain area in the municipality of Pune, India. Here is an example of data entry:

Entry: 111614056 27

Question	Answer
Your MIS No.	111614056
Your sub-sector. (1 to 30)	27
Date of entry (DD/MM/YYYY)	28/02/2019
Time of Entry	11:02:44
Major Land-use around entry point	Mixed (Resi+Comm)
Location (Lat-Long)	18.51934, 73.852987
Noise Pollution Level (in Decibels)	77.1
Take a pic.	

A data entry contains the following attributes:

1. Your MIS No.: ID number of the surveyor;
2. Your sub-sector: an integer ranging from 1 to 30 that indicates the sub-sector where the measurement was taken;
3. Date of entry in DD/MM/YYYY format;
4. Time of entry in hh:mm:ss format;
5. Major Land-use: a categorical variable which indicates the major land use around the measurement point (Commercial, Residential, Recreational, etc.);
6. Location: geodetic coordinates, Latitude and Longitude, in WGS84 reference system. Coordinates are expressed in degrees. EpiCollect5 also automatically computes local projected coordinates, East and North, in meters. The local projected coordinate system used in this case is UTM zone 43N.
7. Position accuracy: measurement error in GPS point positioning in meters. In this dataset point positioning accuracy ranges between 1 m and 65 m, a level of accuracy that is quite typical for GPS positioning based on single-epoch code observation by GPS receivers in most smartphones.
8. Noise Pollution Level in dB;
9. Picture: a photograph of the surroundings at the moment of the measurement.

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.

2.2. PostgreSQL database

The web application running on the WSGI server will interact with a Database Management System for data storing and management. In particular, we decided to use PostgreSQL because it is a free and open-source relational database management system 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. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC). Key features of PostGIS include, among others:

- Geometry types for Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons and GeometryCollections;
- Spatial predicates for determining the interactions of geometries;
- Spatial operators for determining geospatial measurements like area, distance, length and perimeter;

- Spatial operators for determining geospatial set operations, like union, difference, symmetric difference and buffer.

The PostgreSQL-PostGIS database will be composed by one Table with the following attributes, i.e. columns:

1. entry_ID: integer value uniquely identifying the entry. This will be also the table's Key Value;
2. surveyor_ID: integer value indicating the surveyor who has contributed the data measurement;
3. subsector: integer value from 1 to 30 indicating the sub sector where the measurement was taken;
4. date_entry: class datetime.date variable representing the date of the entry;
5. time_entry: class datetime.time variable representing the time of the entry;
6. landuse: a string value (categorical variable) indicating the major land use around the measurement point (e.g. Commercial, Residential, Recreational, etc.);
7. lat: float value representing the geodetic Latitude (in degrees) of surveyed point;
8. lon: float value representing the geodetic Longitude (in degrees) of surveyed point;
9. accuracy: float value representing the measurement accuracy in GPS point positioning (in meters).
10. UTM_easting: float value representing the UTM projected East coordinate (in meter);
11. UTM_northing: float value representing the UTM projected North coordinate (in meter);
12. UTM_zone: string value representing the UTM zone of the projected coordinates reference system;
13. noise_dB: float value indicating the level of noise pollution [in dB];
14. picture_URL: string value containing the URL to the picture of the surroundings at the moment of the measurement.
15. geometry: a PostGIS geometrical attribute, of type POINT, containing the geodetic coordinates (Lon., Lat.) of the surveyed point.

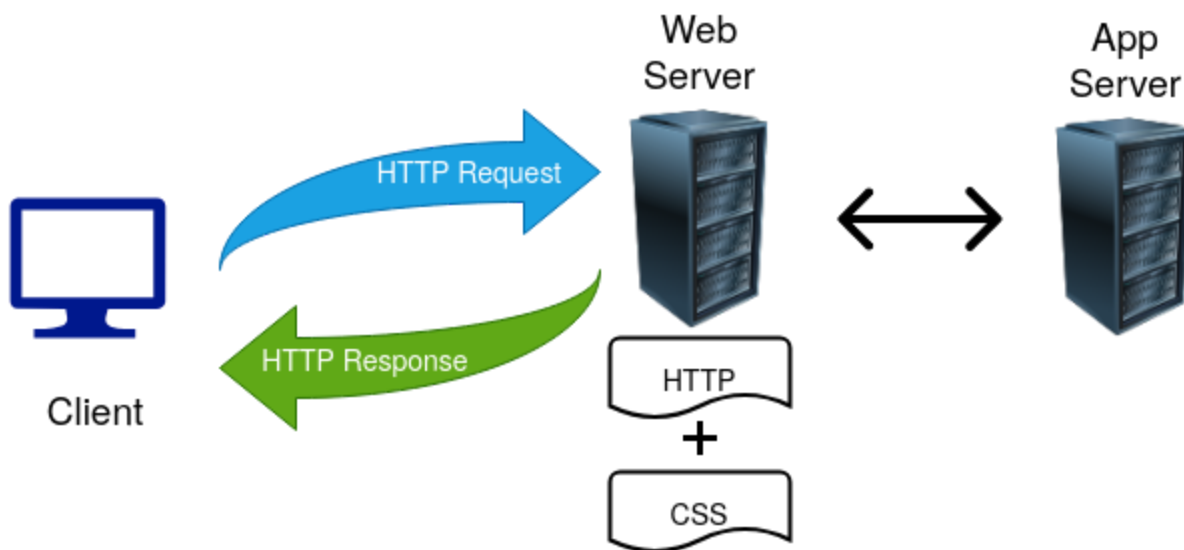
3. Software Structure

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

3.1. HTTP Server

The client is able to communicate with HTTP Server through client requests (URL, clicks, formularies, etc). This server will provide a response to the client server (user browser). The response is composed by HTTP and CSS codes that are run by the client server and the completion status to confirm whether the operation is successful or not.

The HTTP and CSS codes obtained as output from the Web Server are generated by means of requesting to the Application Server all the needed information for producing the required web page. Thus, the Application Server provides the logical outcome as a response to the HTTP Server's request.



3.2. Application Server

This server will provide all the logical operations that will be needed in order to fulfil the software requirements. Given that Python coding will be used for this project, this is a WSGI Server, thus Python applications can run and serve the Web Server in a consistent way¹.

¹ <https://www.fullstackpython.com/wsgi-servers.html>

In this section, the principal logical functions' internal structure is described. Then, there is an explanation about how the principal interactions between the components are carried out.

3.2.1. Data retrieval from EpiCollect5 and preprocessing

First of all the application is required to connect to EpiCollect 5 and to get the dataset of interest using the REST API provided by such a web service. This step will return a string with raw data.

The next step is to parse the raw text response and check whether each data value is properly formatted. For example, the dates and times must be in correct DD/MM/YYYY and hh:mm:ss formats respectively, the land use must be a valid categorical value, the noise pollution must be a float number within reasonable values, etc. This step allows us to correct data entries that might be incomplete, formatted in a not consistent way, present errors or outliers. If a certain data point contains an invalid value, then it is discarded, which means that it is not considered for any purpose (visualization, data analysis, filtering, etc.). The Data is then validated, cleaned and ready to be transformed from JSON format to pandas DataFrame.

Finally, a new GeoDataFrame is created by adding a new geometry attribute of Point-type to the DataFrame.

The main characteristics of Data retrieval from EpiCollect5 and preprocessing are summarized in the table below:

Used Libraries	Pandas GeoPandas Requests JSON re
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
	Data validation and cleaning criteria
Output	GeoPandas GeoDataFrame that contains the validated and cleaned data previously retrieved from EpiCollect5.

3.2.2. Data import into PostgreSQL

Once the data from EpiCollect5 has been retrieved and pre-processed the next task is to import it to a PostgreSQL database table.

The first time the app is run a new table in PostgreSQL database will be created and the entire data contained in the GeoDataFrame copied in this new table. This process will be performed using Python modules SQLAlchemy and GeoAlchemy2, which in return will set up an “engine” for managing the connection and interaction with the database management system.

After the PostgreSQL database table has been created and initialized with the data from EpiCollect5 the app will look for any new entry in the EpiCollect5 dataset. If any new data is found then it will be appended to the PostgreSQL database table. In this way new entries in EpiCollect5 will be reflected into the database while avoiding the need to copy the entire dataset every time the app is run, thus improving performance, and also eliminating the risk of overwriting or deleting data that are already stored in the database.

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

Used Libraries	Pandas GeoPandas SQLAlchemy GeoAlchemy2 Psycopg2
Inputs	GeoPandas GeoDataFrame containing the data retrieved from EpiCollect5 and already preprocessed.
	Parameters (database name, user, password, etc.) to connect to the DBMS
Output	PostgreSQL-PostGIS database table with varchar, numerical and geometry attributes.

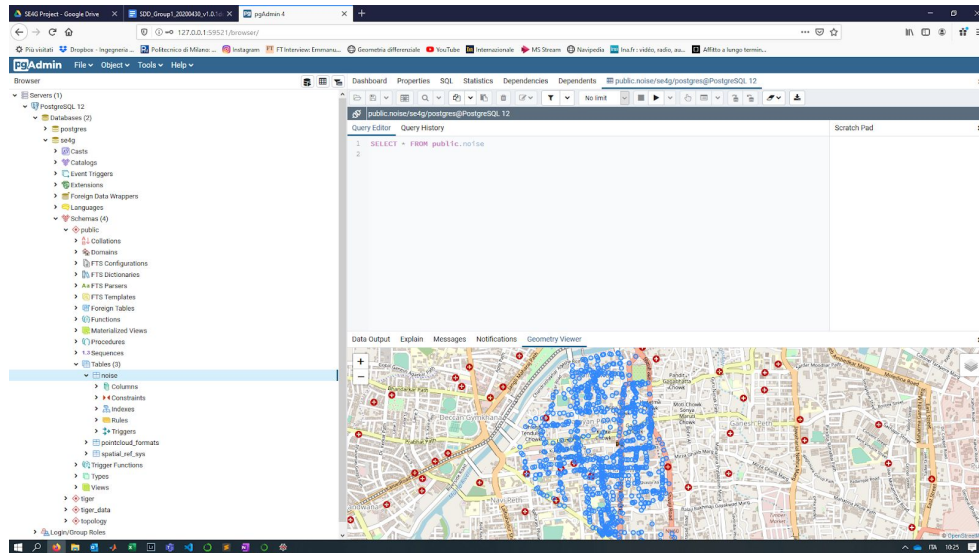


Figure 1: Example of PostgreSQL-PostGIS database

3.2.3. Mapping Tool

The WSGI Server will provide a user-interactive map for showing measurement points, basemaps and geospatial data visualization. The Mapping function will receive a list of elements to be plotted as input as well as specifications of how each element has to be plotted. This function will mainly use the Bokeh library which is aimed to develop interactive visualization for web applications.

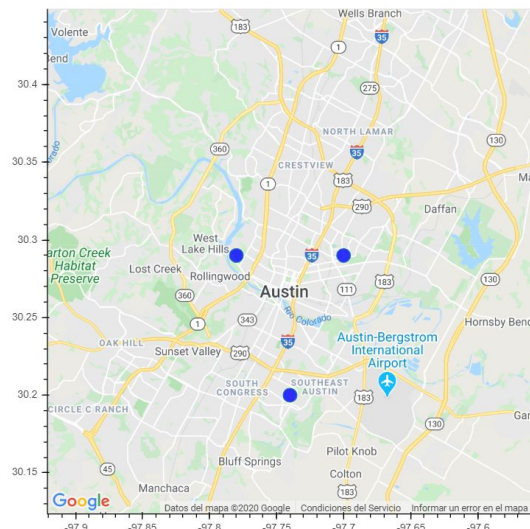


Figure 2: Bokeh Web Map Example

The main characteristics of the mapping function are described in the following table:

Used Libraries	Bokeh Matplotlib Geopandas Numpy
Inputs	Python List of elements to plot: it can contain a Bokeh Tile (for basemaps), Geodataframe or numpy array (for data plotting)
	Python List of labels (as strings) for each element to be used in the map legend.
	Python List of how-to-plot specifications: String format which specify whether the element has to be displayed as a tile (basemaps), points (data measurement points) or as a geospatial chart (heat map, hotspot map, etc)
Output	"Bokeh Plotting Figure" class containing the interactive map ready to be displayed in the web application

3.2.4. Filtering Management

As the software must have a user-controlled data filtering manager, a filtering function must be deployed. The filtering function receives as input the user-controlled filtering settings (e.g. time window, drawing for spatial filtering, noise pollution cutoff, etc). In order to identify the data points that fulfil the filtering, the function returns a Pandas Boolean Series in which there are True values where the measurement point is considered and False values where the data point is not needed.

For spatial filtering, the Bokeh figure will have a user-drawing tool in order to select the desired location for filtering purposes. Once the user has done the drawing, the selecting tool function will retrieve and return the points within the desired area.

The described output allows the other components to easily operate on the filtered data. The following table summarizes the described function:

Used Libraries	Bokeh Shapely Pandas Geopandas Numpy
Inputs	Timestamp tuple to define window time
	Spatial area user-drawing done in the Bokeh Figure with the Hover tool.
	Noise Pollution tuple to define the range of values for filtering
	List of strings which specifies the subset of land use classes that will be used
Output	The function returns a Pandas Boolean Series in which there are True values where the measurement point is considered and False values where the data point is not needed.

3.2.5. Geospatial charts & Statistics

As part of the Data Analysis Tools, the application will make the user able to visualize the already filtered (or the whole by default) geospatial data with different styles (heat map, hotspot map, contour map, etc) and to obtain statistics (mean, standard deviation, quantiles, min/max, histogram, etc) on its features. Therefore, the software will have a set of components to obtain these functionalities which are summarized in the following table:

Used Libraries	Bokeh Matplotlib Geopandas Numpy
Inputs	Geodataframe containing the whole available data (queried from database)
	Pandas Boolean Series which specifies which are the considered data (output of filtering)

	function)
	In the case of Geospatial charts, the style and the layer name must be specified (string arguments)
Output	Geospatial charts: it returns a matplotlib figure which is ready to be added to the bokeh figure (map)
	Statistics: it returns a dictionary with all the available statistics as well as a matplotlib histogram of the relevant data features.

3.2.6. Template Engine

3.2.6.1. Flask application

Flask is one of the python's libraries which provide useful tools and features to create the web application. Flask applications enable us to execute the interaction of the web server and database through WSGI server. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file. Flask application provides a connection within the WSGI server in aim of implementing and managing the request and interaction from client user (which is connected to the web server) to the database. The responses from the database after generating in the flask application will come back to the user client.

Flask will create a directory for establishing the connection between the database and the web server. The directory will be divided into two main parts. The first part of this directory will contain the python codes for generating the functions and architecture of the connection:

- The first part is python code that contains the functions for interacting with the database. This approach allows the user to send the specific request. These requests are the main key for establishing the connection between the user and the database to retrieve information from the database and present them on the client browser in the html format.
- The second part of the python code is for defining the architecture of the database connection and user client. In this structure we will explain the format of the connection and also present values for variables in purpose of better interpreting.

These are more important function from flask that enable us to made the connection:

Flask function	Description
Render_template()	Helper function that allows use of the JINJA template engine.
Request.form()	Assign to the data that requested from specific field or variable
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().
Connect "From psycopg2 library"	Create a new database connection.

3.2.6.2. JINJA Template engine

While flask application retrieves information from the database, it should be displayed on the web application. The web application mainly uses HTML to display information for the clients, so for this purpose we have to integrate codes in HTML files. For managing the appearance of the web application, ease the navigation and browsing for end users, flask will provide us a template engine called JINJA. Templates are an empty *Skeleton* which receive values from Python codes. Templates contain variables and expressions that are replaced with the values. JINJA template engine which dynamically builds HTML pages using familiar Python concepts such as variables, loops, lists, and etc. based on logic and context. Unlike static HTML, templating systems like JINJA empowers us to do things like share snippets between pages, or render pages conditionally based on context. Templates won't execute without an app (Flask application) to serve them. The second part of the directory that mentioned on the flask application section allocated to Templates. Generally, there are two main html files for generating the web request and displaying retrieved data from database:

1. base.html: The base.html 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.

2. index.html: there are other components in the homepage of the web application and we have to define also for them html codes for rendering them. These components are like children and their parent is the main base.html. There are free blocks defined in the parent html file, It's the job of "child" templates to fill the empty blocks with content. This index.html is just like baee.html but without some details that belong to the main homepage (like footer).

These are basic codes for generating the web application through client users(browser). Based on the user cases and the functional requirement on the RASD document we have different components, partials and sub pages. We will define templates(child files) based on the different functions and user cases. In the table below mentioned the most important tags and expression for defining templates:

Tags	Description
{% block %}	tags define blocks that child templates can fill in.
<title>	Displaying the title of a body.
<h1> text </h1>	Displaying specific text.
{% block title %} {% endblock %}	A block that serves as a placeholder for a title, you'll later use it in other templates to give a custom title for each page in your application without rewriting the entire <head> section each time.
{{ url_for('index')}}}	A function call that will return the URL for the index() view function. This is different from the past url_for() call you used to link a static CSS file, because it only takes one argument, which is the view function's name, and links to the route associated with the function instead of a static file.
{% block content %} {% endblock %}	Another block that will be replaced by content depending on the <i>child template</i> (templates that inherit from base.html) that will override it.
<div> </div>	This tag is a block level tag which the content inside the tag is displayed in a new line. With this tag, you can put together a large group of HTML tags and style them collectively or in groups using CSS codes.

<pre> </pre>	This tag defines an unordered list(a bulleted list).
-----------------------------------	---

3.2.6.2. CSS codes

On the time that we will prepare the HTML files for establishing and managing the connection and interaction, the CSS code which derived from the same directory that templates there, will manage the style and the appearance of the template web pages like border style, color of the components, text align and etc. There are toolkits for better interpreting web styling codes such as Bootstrap toolkit, which provides easy-to-use components for styling application.

3.2.7. Interaction between application server components

The different components within the application server are designed in such a way that the described data filtering, mapping and analysis are easily controlled by the user through a rendered dynamic web page. In this section the interaction between the different components within the application server is explained.

In the first instance, the whole Epicollect5 project's available data is prepared doing the following:

1. retrieving, parsing, converting and pre-processing as explained in 3.2.1.
2. importing and initializing in the software DBMS

The Epicollect5 storage is often checked over the lifetime of the software in order to append new data if any. Once the DBMS contains all the available information properly, the data is placed into a Geopandas Dataframe which is distributed in the Data Analysis Tools, Filtering Manager and Mapping Tool as an input.

The Filtering Manager uses the user-setting to identify which data point is going to be considered in the Data Analysis and Mapping Tools by inputting them a Boolean Pandas Series as explained in 3.2.4. The Data Analysis Tools will obtain user requests regarding data visualization and statistics in order to do the following:

1. When a Statistic request is submitted, this set of functions will do the computation and input it to the Template Engine. The Template Engine will generate the display of the statistics to the user in a clear way.
2. When a Geospatial Chart is requested. this set of functions will compute the needed data to obtain the specific visualization (e.g. heat map, hot spot map, contour map) and it will be input to the Mapping Tool which will execute plotting operations (using the filtering information) in

order to add the requested visualization to the figure, which will be finally rendered by the Template Engine.

The Template Engine renders the final client-side web page. Each time this function does it, the engine will have as input some built-in base templates (in form of HTML codes) designed to provide a user-friendly experience in the web application.

In Figure 3, all the principal interactions between the application server components are graphically summarized. Application functions are represented in orange, DBMS in blue, Python Geopandas dataframe in green and Jinja templates set in yellow. The arrows represent the data flow between the different software components.

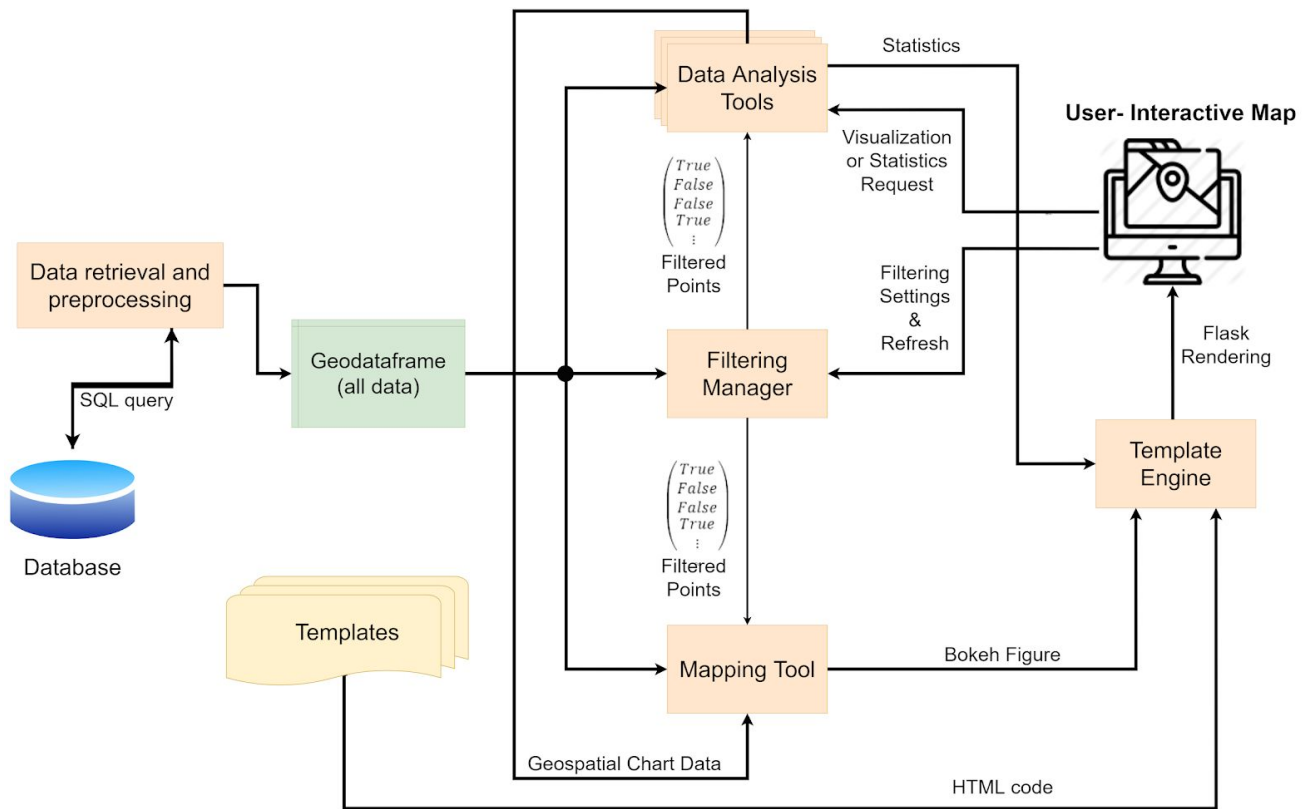


Figure 3: Summary of interaction between application server functionalities.

3.3. Database Server

The WGSJ server will connect to a database server where data is stored. More precisely the web application will interact with PostgreSQL, a free and open-source relational database management system. This interaction will exploit a database adapter for the Python programming language known as psycopg2, which will enable the web app to perform CRUD operations on the database.

Moreover, given the geospatial nature of our data, we decided to exploit PostGIS, an open source software program that adds support for geographic objects to the PostgreSQL object-relational database. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC). See Section 2 for a description of the key features offered by PostGIS.

The following table contains a list of the main functions related to the interaction between the web application and PostgreSQL. Some of these functions are provided out-of-the-box by Python's module psycopg2.

Function name	Description
Establish connection	To establish a valid connection between the web app and a PostgreSQL database. Connection shall enforce ACID rules.
Create table	To create a table with a given set of attributes, including geometry, to store data values.
Copy data	To copy data from a GeoPandas DF to a table in PostgreSQL / PostGIS.
Insert data	To insert new data entries into the database.
Query data	To query/filter data using SQL compliant instructions.
Fetch data	To fetch data from the database to the Python main programme.
Commit changes	To commit changes or queries to the database.
Close connection	To close the connection with the database after all operations have been correctly committed.

4. User Cases Application

In order to explain the software functionalities, interactions between the components and possible exceptions, this section is going to address an explanation about the actions taken by the software and the user in a list of cases that are useful to explain the internal processes of the application.

In this section we describe what is going on from server-side and client-side on when the user cases happen by specifying the different actions that take place in these situations.

UC1: Discover about noise pollution and how to contribute in data collection

Providing users with a platform to learn about the topic of noise pollution and how to contribute to the survey is one of the purposes of our project. However, since it does not involve any dynamic feature, but on the contrary it is rendered as part of the static HTML part of the website, we have chosen not to address it in this document. For more information about this use case refer to the RASD document.

UC2: Data points visualization

Through the interactive map, users can obtain the details about each measurement points, as in this user case is described:

1. The user enters the user interface for interactive mapping.
2. The whole available raw data is retrieved from the Epicollect5 host server, cleaned up (remove strings from numerical values and setting as missing the invalid ones) and placed in the DBMS (section 3.2.1).
3. From the DBMS, a Geodataframe with the whole available data (checked and cleaned up) is generated by the Application Server.
4. The Geodataframe is input in the Mapping Tool, which adds it to the Interactive Map Figure (created with Bokeh Library, as explained in section 3.2.3).
5. The user places the cursor on a certain measurement point (its geographical position in the map).

6. The pop-up function (tabular dataframe) is activated. Therefore, once the cursor is placed in the point, the specific details about that point (measurement date & time, land use classification, ENP value, etc) and the corresponding picture are displayed as a pop-up.

6.1. By using hovertool from Bokeh we are able to generate tabular tooltips for data points.

6.2. This function enables us to retrieve specific data from column data sources like land use, measurement date and time, images and etc for every point that is shown in the map.

6.3. It is also possible to custom HTML templates for tooltip. These HTML templates customize the tabular appearance for better displaying information and images.

7. The user is able to do the same with each available measurement point.

Exceptions:

1. There is a possibility that two different users measure one specific point and enter data in the Epicollect5, hence we have two measurements from one specific point.

UC3: Map data with a custom visualization

The Interactive map is one of the capabilities which is provided on the web-application in the aim of visualizing spatial data. This user case describes how a user can obtain a customized visualization of spatial data:

1. The user enters the user interface for interactive mapping.
2. The whole available raw data which has already been retrieved from the Epicollect5 host server, cleaned up and placed in the DBMS) is obtained by the Data retrieval and pre-processing function (Section 3.2.1.).
3. As explained in UC2, the whole available data points from the DBMS are visible in the interactive map by default (with a certain default basemap).
4. The user wants to consider the data just within a specific area, in a certain time window and with a specific land use classification, so he/she does the corresponding settings in the friendly user-interface for filtering to keep just the needed data subset.

5. The settings request is taken by the Filter Manager, which performs the logical operations (explained in section 3.2.4) in order to obtain the indexes of the needed data subset.
6. The indexes of the filtered data points are delivered to the Mapping Tool, which use these to obtain only the needed data subset in order to add it to the Map Figure (the details of this process are explained in section 3.2.3). The page is rendered again with the updated Map (with the filtered data points).
7. The user clicks the Visualization Options Menu and selects a certain style to plot the spatial data (heat maps, contour maps, dot maps, hot spot maps, etc). The user request is sent to the Data Analysis Tools.
8. The indexes of the filtered data points are also delivered to the Data Analysis Tools, which perform the logical operations (explained in section 3.2.5) to generate the required data to obtain the visualization with the data filtering.
9. The visualization data is input in the Mapping Tool, which computes and adds the custom visualization to the Figure. As before, the page is rendered again with the updated Map.

Exceptions:

1. The user is looking for a certain data point that he has added to EpiCollect5 database but still it doesn't appear in our web application; e.i. the DBMS data has not been updated already.
2. The basemap (loaded from a server such as Open Street Map) is currently not available, thus the web application only loads data points without a basemap.
3. Newly data that has been uploaded to the EpiCollect5 database and as a consequence has been updated to the DBMS has an incorrect format and cannot be displayed on the map.

UC4: Data analysis of the noise pollution

The user will be able to discover information about Environmental Noise Pollution also by means of interactive analytical tools (e.g. histograms and other charts). These tools are meant to be interactive in the sense that the user can choose which data to visualize, for example filtering data by specific land use or time of the day of the surveying. This use case describes the main analytical tools which will be provided by the web application:

1. The user enters the user interface for interactive analytical tools;

2. The whole available raw data which has already been retrieved from the Epicollect5 host server, cleaned up and placed in the DBMS, is obtained by the Data retrieval and pre-processing function (Section 3.2.1.).
3. As explained in UC2, the whole available data points from the DBMS are visible in the interactive map by default (with a certain default basemap).
4. Different operations and filters are available for personalizing these charts, for example the user can filter the data by land use or by the time of day when the noise pollution measure was taken. In this case the user clicks in the Filter Manager to display just the subset of needed data;
5. The settings request is taken by the Filter Manager, which performs the logical operations (explained in section 3.2.4) in order to obtain the indexes of the needed data subset;
6. The filtering information is input in the Data Analysis Tools.
7. Then, the user clicks in the Data Analysis service and selects the option to get Descriptive statistics.
8. The software calls the statistics function which computes and returns the general statistics of each data subset feature.
9. The statistics data is passed to the Template Engine. The page is rendered with the requested Statistic display.

Exceptions:

1. The user applies a filter (e.g. land use = residential) for which there is no data to be displayed in the plot; in this case the plot will result empty.

UC5: Repeatedly visualization-filtering requesting

Now we consider a case similar to UC4, but supposing that the user requests a sequence of data analysis requests (geospatial chart visualization or statistics over filtered data) and filtering updates. If the user does many statistics-filtering requests it is not a problem, since the statistics display will be simply updated and overwritten.

Given that each geospatial chart visualization requests are going to be added to the client-side map, this is an interesting user case. If we consider a scenario in which for each filtering request, each map layer is updated with the new filtering, then it would be very computationally expensive, especially when the number of layers is high. For handling this problem, the previously generated Geospatial Visualizations will not be updated with each filtering request.

The actions carried out by the user and the software in this case are outlined:

1. When the Mapping Tool is initialized, the only default layer is the measurement data points (the basemap is also visible but not as a user-controlled layer).
2. The user can do a filtering request by setting up the filter manager controls. The Filtering Function will obtain the required data point indexes (As Boolean Series) and send these to the Data Analysis and Mapping Tools.
3. The Mapping Tool will update the Bokeh Figure by modifying the Data Points layer according to the new filtering. The page is rendered again with the updated figure by the Template Engine.
4. The user can see the updated Data Points layer. Now he/she does a visualization request by specifying which type and submitting the layer name.
5. The Data Analysis Tools will receive the visualization request and the layer name. It will generate the figure data (taking into account the new filtering!) and send it to the Mapping Tool.
6. The Mapping Tool will update the Bokeh Figure by adding the new visualization layer according to the new filtering. The page is rendered again with the updated figure by the Template Engine.
7. Steps 2-6 can be repeated many times as illustrated in Figure 4.

Visualization - Filtering Sequence

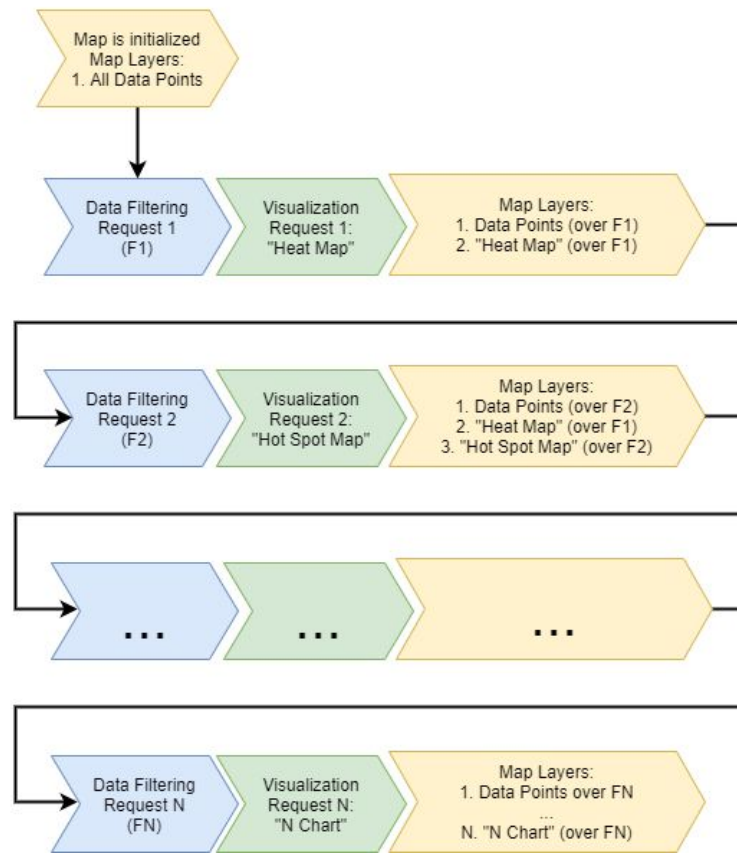


Figure 4: Flowchart which summarizes how the map is updated during a filtering-visualization request sequence.

UC6: Someone added invalid data to the Epicollect5 project

Now we consider a scenario in which invalid data is uploaded in the Epicollect5 platform. If this type of exception would not be taken into account, it could lead to failures in the software because each component expects an specific format for input data. In order to avoid failures, the data must be checked during the preprocessing and invalid data points are discarded as explained in section 3.2.1. When a certain data point is discarded means that it is not considered for any purpose (visualization, statistics, filtering, etc.)

5. Organization

As mentioned in the Introduction, 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. So each team member will act as the primary owner of a specific part, as presented in the following table:

Name	Task/Activity	Description
Homeyra M.	Templates Engine	The Templates Engine will be developed in HTML and CSS and will be the structural backbone of the project. All of the system components will be located on top of this space and will be displaying information to the user at all times.
Alessandro Z.	DBMS	The Database Management System will be designed to work along with EpiCollect5. It will extract all the information from the API tool, pre-process it and store it in the most efficient manner. This database management system will interact with the different interfaces with a defined temporality.
Diego C.	Filtering Tools	The filtering tools will interact with the user in order to receive requests. This will work as an input-output friendly interface that will interpret the user needs and output the correct requests to be visualized with the interactive part of the software.
Rodrigo C.	Map Visualization	This task involves the implementation and development of the map visualization, which is the most important interaction tool that the web application will have. It will display all the information upon user request additional to the default visualization. It will be presented in an efficient, organized and attractive way.

6. References

G. Buchgeher, C. Klammer, B. Dorninger and A. Kern, "Providing Technical Software Documentation as a Service - An Industrial Experience Report," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 2018, pp. 581-590, doi: 10.1109/APSEC.2018.00073.