Politecnico di Milano
AA 2020-2021

Geoinformatics Engineering
Software Engineering for Geoinformatics Project

iUrban
Design Document

version 1.0 - 25/5/2021

Authors:

Gao Jiawei

Ding Guangyu

Ernesa Ymeri

Song Xiangyang

Professors:

Di Nitto Elisabetta

Oxoli Daniele

Section 1 - Introduction

1.1 Design Document

As documentation plays an important role in software engineering, design documents would have a variety and a wide array of types of design document as well as explanations on why each of them is important, also defining how type of design document would precisely fit your computer software.

It is important to emphasize in the fact that this document builds upon the Requirements Analysis Specification Document (RASD), written by: Gao Jiawei, Song Xiangyang, Ding Guangyu, Ymeri Ernesa in 2021 which can be found in the following link: SE4G Group1 Github RASD

This document will describe the following characteristics of the project:

1) Project Database: The process database is a permanent repository of the process performance data from projects; it can be used for project planning, estimation, analysis of productivity and quality, and other purposes. For this reason a detailed explanation of the database characteristics will be described.
2) Software Structure: The design of the database is the base of what functions this application can achieve. Depending on Web Server (HTTP Server), Logical Server (Application Server) and Database Server we could access, manage and update the information.
3) Use Cases Application: This part illustrates use cases and requirements map on the components of the software. Further detail in RASD.
4) Organization: The development team is composed of four students. Although each of the members will participate in all the parts of the whole project, it is important to assign roles and activities to each engineer.

1.2 Product Description

The purpose of developing this product is to inform and involve communities potentially who want to contribute to the project through an online desktop application. This web application will allow users to access, analyze and visualize the available iUrban data of a given area through an interactive mapping tool. On the website, the users will be able to find general information about the issue of average noise level, wind direction and instructions on how to contribute to data collection, as this project aims not only to inform people , but also to involve local communities into contributing.

The online application will present different types of information, both static and dynamic. Static type displays the information which remains unchanged during user interaction, stored as HTML code and served when requested by the clients. Dynamic type refers to all the information that the user will be able to interact with and that changes on demand, all of which is achieved by using the Python language.

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

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

2.1 EpiCollect5 Dataset

This description of the **Average noise level** dataset in EpiCollect5 constitutes a snapshot of data as they are available on 10 May 2021, hence the possibility for changes to the dataset size or attributes occurring between this date and the implementation phase, despite no such changes having been observed in the last month.

The dataset consists of 451 measurement points of environmental noise pollution. The data are geolocalized in a certain area in the municipality of **San Isidro, Peru**. Here is an example of the data entry:

Entry: macarena valdivia

Question	Answer
Name	macarena valdivia
Location	-12.102838, -77.039238
Date	23/10/2020
Time	15:35:00
Average noise level	58
Average light intensity	5797
Wind direction	E
Wind speed (Beaufort scale)	1
Cloud cover	8
Cloud type	stratus
Photo of cloud conditions	
Visibility	2
Traffic count	16
Temperature	20
Humidity	77
Make a note of anything that may cause an anomaly	
Air pollution (number of particles) PM only.	1

Fig 1. Example of an Entry Showing the Attributes Table

A data entry contains the following attributes:

1) Name : The name of the surveyor ;
2) Location: Geodetic coordinates (Latitude and Longitude) in WGS84[1] reference system. Coordinates are expressed in degrees, generated automatically by Epicollect5.
3) Date:DD/MM/YYYY format;
4) Time:hh:mm:ss format;
5) Average Noise Level: in dB;
6) Average Light Intensity: is the perceived power per unit solid angle,in Candela (cd);
7) Wind Direction: reported by the direction from which it originates;
8) Wind Speed (Beaufort Scale): is an empirical measure that relates wind speed to observed conditions at sea or on land
9) Cloud Cover: refers to the fraction of the sky obscured by clouds when observed from a particular location
10) Cloud Type: Clouds are given different names based on their shape
11) Visibility:
12) Traffic Count: count of vehicular or pedestrian traffic, which is conducted along a particular road, path, or intersection..
13) Temperature: is a measure of the average kinetic energy of the particles in an object, in Celsius;
14) Humidity: is the amount of water vapor in the air,in fahrenheit;
15) Note: anything that may cause an anomaly;
16) Air Pollution (Number of Particles): PM only.

Additional fields generated automatically by EpiCollect5 when the user uploads a data entry include: EpiCollect5 unique ID, creation date and time, location.

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:

1) Geometry types for Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons and GeometryCollections;
2) Spatial predicates for determining the interactions of geometries;
3) Spatial operators for determining geospatial measurements like area, distance, length and perimeter;
4) 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:

2.2.1 Table TData

1) data_id: Primary key
2) author_id: integer not null
3) created_date: timestamp default not()
4) name: varchar(50) not null
5) date: varchar(50) not null
6) time: varchar(50) not null
7) latitude: float (16) not null
8) longitude: float (16) not null
9) average_noise_level: Integer not null
10) average_light_intensity: Integer not null
11) wind_direction: varchar(16) not null,
12) wind_speed: Integer not null
13) cloud_cover: Integer not null
14) cloud_type: varchar(16) not null
15) cloud_photo_id: varchar (64)
16) visibility: Integer not null,
17) traffic_count Integer not null,
18) temperature: Integer not null
19) humidity Integer not null,
20) note_of_anomaly: varchar(255)
21) air_pollution Integer

2.2.2 Tabel TUser

1) user_id: serial primary key
2) user_name: varchar(20) unique not null;
3) user_password: varchar (255) not null;
4) user_email: varchar(30);

2.2.3 Tabel TComment

1) comment_id: serial primary key
2) author_id: Integer not null;
3) data_id: Integer not null:
4) created_date: Timestamp default now():
5) body: varchar(50) not null;
6) foreign key(data_id) references TData(data_id);

Section 3 - Structure Design

3.1 HTTP Server

HTTP Web Server routines are used to run and configure services of the embedded web server.
A HTTP or web server processes requests via HTTP, a network protocol used to exchange information on the World Wide Web (WWW). The main function of a HTTP server is to store, process and deliver web pages to clients. Pages delivered are usually HTML documents, which may include images, style sheets and scripts in addition to text content. Using HTML, you describe what a page must look like, what types of fonts to use, what color the text should be, where paragraph marks must come, and many more aspects of the document.
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.

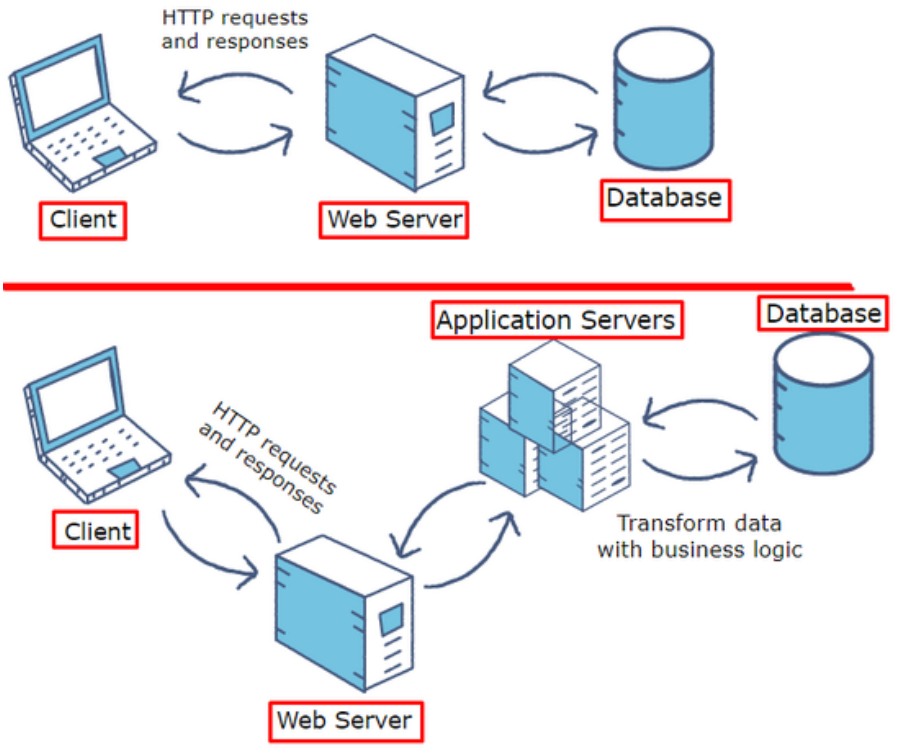

Fig.1 Topological Graph of Servers and Client

3.2 Application Server

An application server is a mixed framework of software that allows both the creation of web applications and a server environment to run them. It can often be a complex stack of different computational elements running specific tasks that need to work as one to power multiple clouds and web-based software and applications.

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 average noise level must be a dB within reasonable values, the wind direction must be a varchar 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 JSON
Inputs	➔ URL of EpiCollect5 REST API ➔ List of attributes of EpiCollect5 dataset that are relevant for this app and to be stored in the DataFrame
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.
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 Psycopg2
Input	➔ 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.

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 …….library which is aimed to develop interactive visualization for web applications**.**

Used Libraries	➔ Matplotlib ➔ GeoPandas ➔ Numpy
Input	➔ Python List of elements to plot: it can contain a BootStrap(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	“BootStrap 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. cloud type, wind direction,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 BootStrap 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:

User Library	➔ Shapely ➔ Pandas ➔ Geopandas ➔ Numpy
Input	
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	➔ MatplotLib ➔ GeoPandas ➔ Numpy
Input	➔ Geodataframe containing the whole available data (queried from database) ➔ 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 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 a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine

WSGI

The Web Server Gateway Interface (Web Server Gateway Interface, WSGI) has been used as a standard for Python web application development. WSGI is the specification of a common interface between web servers and web applications.

Werkzeug

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeg as one of its bases.

jinja2

jinja2 is a popular template engine for Python.A web template system combines a template with a specific data source to render a dynamic web page.
These are important function from flask that enable us to made the 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 Skelton 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.
Generally, there are four main html files for generating the web request and displaying retrieved data from database:
1.Base:
2.Index:

3.Comment: users use this page to leave comments on the recordings
4.Table: users use this page to view the attributes of all samples and do some operations

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
{% **if** users %}	The if statement in Jinja is comparable with the Python if statement
<title>	Displaying the title of a body.
text	Displaying specific text
{% block content %} {% endblock %}	Another block that will be replaced by content depending on the child template (templates that inherit from base.html) that will override it.
{{ ... }}	for Expressions to print to the template output
  	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.
 	This tag defines an unordered list(a bulleted list).
<svg> </svg>	SVG stands for Scalable Vector Graphics
<nav> </nav>	This tag defines a set of navigation links.
 	This tag defines smaller text (like copyright and other side-comments).
<button> </button>	This tag defines a clickable button.
  	This tag is an inline container used to mark up a part of a text, or a part of a document.

3.2.6.2. CSS codes

Like HTML, CSS is not a programming language. It's not a markup language either. CSS is a **style sheet language**. CSS is what you use to selectively style HTML elements.
In the table below mentioned the most important tags and expression for defining templates:

Tags	**Description**
<p class="my-class">	The element(s) on the page with the specified class. Multiple instances of the same class can appear on a page.
➔ Width ➔ Background-color ➔ color ➔ text-shadow ➔ display	➔ width (of an element). ➔ background-color, the color behind an element's content and padding. ➔ color, the color of an element's content (usually text). ➔ text-shadow sets a drop shadow on the text inside an element. ➔ display sets the display mode of an element. (keep reading to learn more)

3.3. Database Server

The WSGI 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.

Section 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 both server-side and client-side on each use case by specifying the different actions or events that take place in these situations.

4.1 Use Case: Discover about Noise Pollution and How to Contribute in Data Collection

Providing users with a platform to learn about the topic of average noise level, average light intensity 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.

4.2 Use Case: 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 BootStrap 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, location, visibility, etc) and the corresponding picture are displayed as a pop-up.

6.1. By using bootstrap5.0 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 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 tooltips. 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.

4.3 Use Case: Map Data with a Custom Visualization

The user will be able to visualize data with different forms of maps and different symbol properties to find more information and get a clearer recognition on the distribution of these sample's attribution spatially.

1. The user can view the data with numerical attributes such as noise pollution, light intensity, humidity by heat map.
2. The user can visualize the points with numerical attributes using different sized symbols to indicate the difference between the properties.

4.4 Use Case: Data Analysis of the Noise Pollution

The user will be able to discover data of iUrban Test also by means of interactive analytic tools (ex. histogram or other charts). The reason for these tools are meant to be interactive in the sense that the user can choose which kind of data he wants to visualize, for example: date of the surveing, humidity, temperature.
This use case describes the analytic tool 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 DBMS,
3. 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 ex: the user can filter the data by humidity, average noise level or by the time of the data.
   In this case, the user clicks in the Filter to display just the subset of needed data;
5. The settings request is taken by Filter Manager, which performs the logical operations and the filtering information is input in the Data Analysis Tools.
6. Then, the user clicks in the Data Analysis service and selects the option to get statistics about user query data.
7. The software calls the statistics function and returns the general statistics of each data subset feature.
8. The statistic data is passed to the Engine and the page is rendered with the requested Statistic display.

4.5 Use Case: Someone added invalid data to the Epicollect5 project

In this one 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, and data must be checked during the preprocessing.

Section 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
	Templates Engine	
	DBMS	
	Filtering Tools	
	Map Visualization	

Section 6 - References

No references yet.