Renewable Energy Development Scenarios

Computer Engineering Project

Alessandro Cogollo - 938296
Politecnico di Milano

June 12 2022

Contents

1 Introduction

This manual aims at describing the software developed for the Computer Engineering project, presenting technologies, tools and architectural choices based on the scenario in which it's located. The resulting software is useful for supporting renewable energy development, providing futures scenarios used to lead changes. The manual represents a brief guide with an explanation on how to install the software.

2 Specifications

2.1 Objectives

The proposed project aims to create an interactive web interface which will be integrated in the RSE S.p.A. dissemination platforms, for supporting future renewable energy development scenarios, which constitutes one of the core activities in RSE. The website will support the computation of the photovoltaic capacity distribution and the expected production in Italy at a province scale for the achievement of the European Green Deal goals. Starting from variable input parameters and spatial indicators, the interface should allow the spatial and graphical representation of the intermediate and final outputs. The project can be found here: `https://github.com/AlessandroCogollo/CompEng-Project2022` and cloned or forked freely. The repository provides an installation guide that can be followed directly from github readme. A database dump can be found as well, to test the application (PostgreSQL server is required).

2.2 Specifications

For this project Python was choosen as the programming language for this project, since it possesses a broad variety of libraries very well suited to develop custom applications in the field of geospatial analysis. More in depth, the libraries and software used were:

- Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the leaflet.js library. Manipulate your data in Python, then visualize it in on a Leaflet map via folium
- Pandas and Psycopg2, are flexible and powerful libraries when it comes to operate with data, which were, in this project in particular, used to interrogate the database.
- Flask, is a web framework built with a small core and easy-to-extend philosophy. It's ideal for a small app like this project.
- Bootstrap, is powerful, extensible, and feature-packed frontend toolkit used do develop a user interface. There is no need to reinvent the wheel each time!

In some pages a cdn hosted version of jQuery has been used, but very lightly, in addiction to vanilla javascript to perform basic checks on inputs.

2.3 Features

As required by the project specifications, the developed web app permits to plot and compare map plots based on parameters inputted by the user; output values can be inspected at a province scale thanks to a tooltip that is triggered and shown on mouse over event.

3 Implementation

3.1 Architecture

The application is meant for web, and is composed by three main modules, allowing any distribution on multiple servers. The first module is the web server, which receives requests from a client and dispatch them to the computation module; it's designed to retrieve parameters inserted by the user, and to display the plotted graphs. Computational module is responsible for querying the database (PostgreSQL, explained later) and obtaining input values to calculate arrays, required to plot graphs and maps. Finally, the computational module passes values to be plotted to the graph module, which displays them, and return to the web server the map as an html file. Pictures explains

The following figure 1 show a basic architecture schema:

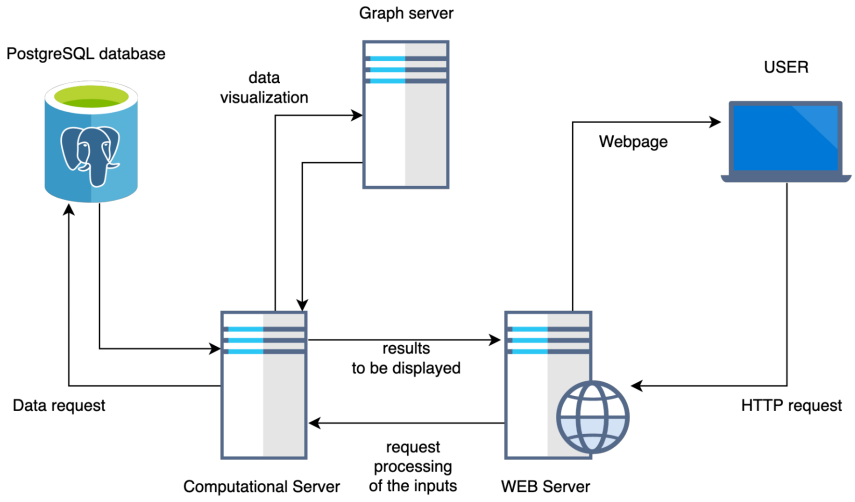

Figure 1: Basic architecture schema

3.2 Values

3.2.1 Datas

Regarding the database, a PostgreSQL database was choosen, since it represents a powerful, open source object-relational database system, and has powerful add-ons such as the popular PostGIS geospatial database extender. In fact, PostgreSQL offers, in addition to the usual MySQL datatypes, the possibility to store geometry types: Point, Line, Circle, Polygon, which has been used to

store datas about provinces' borders. For development purposes, the database has been filled with datas obtained from Istat, containing the administrative borders of italian provinces. They're registered using the WGS84 standard, and can be found on the database dump provided with the project. Below the database schema can be found, reported maintaining the tables' name used in the projects' code, and containing the input used by the computation module.

Table	**Input**
Photovoltaic Installed	Installed Power LAND
Photovoltaic Installed	Installed Power ROOF
Variables	Built surface
Variables	Domestic consumption
Variables	Province population
Variables	Taxable income per capita
Variables	Arable land area
Variables	Agricultural added value
Hourly Producibility	Hourly Producibility

3.2.2 Parameters

Parameters are required by the computation module to perform actions, and have to be greater than zero. Weights are optional and measure unit is none. After the form submission they're stored in parameters object and passed to computation modules.

```
Goals - Scenario PV power [52000 MV]
Goals - Installed Power ROOF [40%]
Coefficients - PV occup. coefficient target ROOF [0,005 km2/MW]
Coefficients - PV occup. coefficient base ROOF [0,008 km2/MW]
Coefficients - PV density target LAND [90,9 MW/km2]
Advance Weight - Weight equivalent hours [no unit]
Advance Weight - Weight domestic consumption per capita [no unit]
Advance Weight - Weight taxable income per capita [no unit]
Advance Weight - Weight Agricultural added value [no unit]
```

Optional inputs not filled by user are automatically set to default values by a javascript function called each time a user submit the form.

3.2.3 Formulas

The formulas used represent the heart of the project; as described, various computational units are used that allow the calculation of the final values of

interest to the user. Below 6 an architecture of the calculation units can be found, followed by formulas used.

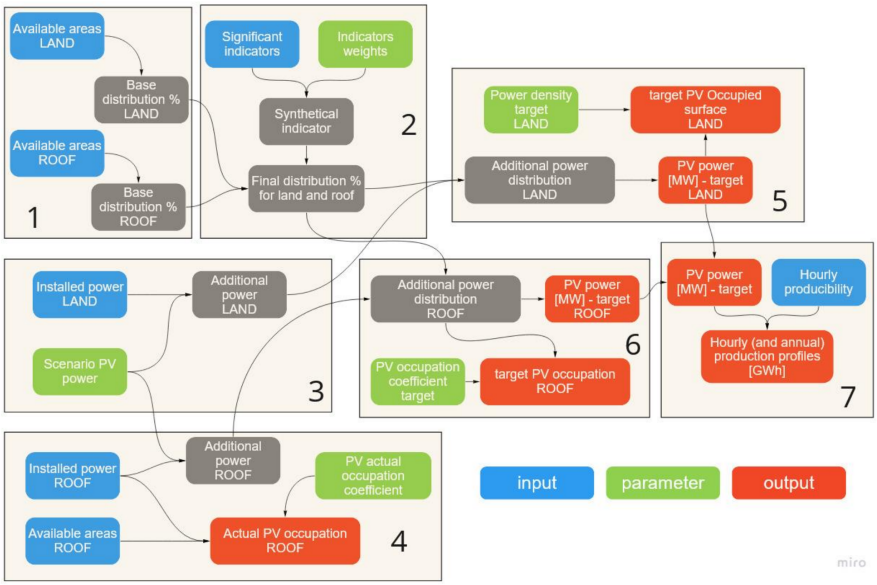

Figure 2: Simple computation module schema

Basic functions used are contained in the `arithmetic module`, and reused multiple times to calculate intermediate values. Such formulas are:

```
def sum_array(to_be_calculated)
def mean(to_be_calculated)
def base_distribution(array_to_process, a)
def array_product(array_first, array_second)
def array_division(array_first, array_second)
```

that are all self-explanatory, except for `base_distribution` which returns an array composed by the quotient of each element of the `array_to_process` and `a`. The `computational module` contains formulas used in the 6 figure above, and are described below. Each of the formulas contained in the aforementioned `computational module` take as arguments three values: `data_object`, `params`, `typology`; passing `data_object params` as arguments simplify computation, and calculating values based on `typology` arg, permits reusability and facilitates code maintenance. The `computational module` provides following formulas (code and related explanation can be found on the GitHub repository).

```
def get_base_distribution(data_obj, params, typology)
def get_synthetical_indicator(data_obj, params, typology)
def get_final_distribution(data_obj, params, typology)
def get_percentage_additional_power(data_obj, params, typology)
def get_additional_power_distribution(data_obj, params, typology)
```

Final values are calculated by the `output values module` object, built with the following methods.

```
def get_actual_pv_occupation_roof(self)
def get_target_pv_occupied_surface_land(self)
def get_pv_power_target_land(self)
def get_pv_power_target_roof(self)
def get_additional_pv_occupation_roof(self)
def get_target_pv_occupation_roof(self)
def get_pv_power_target(self)
def get_production_profiles(self)
```

3.3 Flow

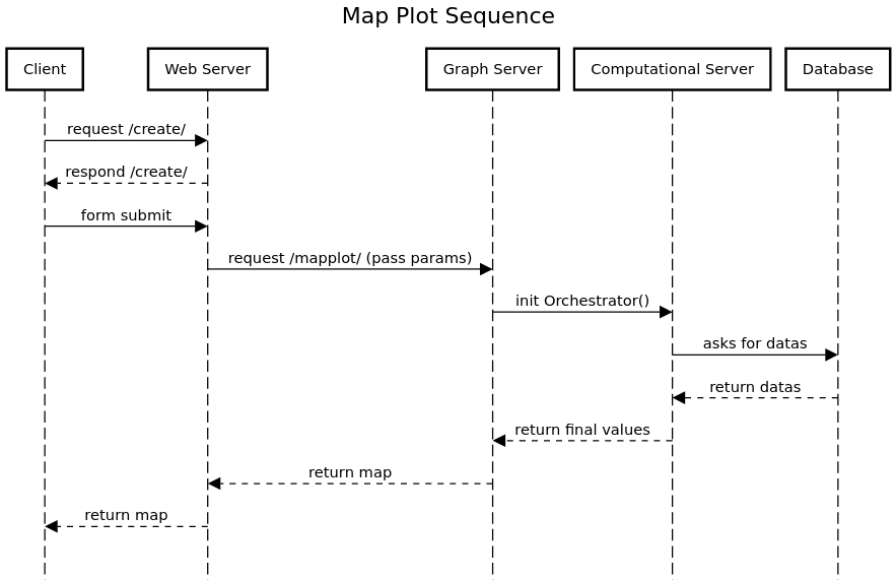

Figure 3: Sequence diagram of map plotting

As described by the sequence diagram above, the process of map plotting is pretty simple, and starts when a client request to the web server (app.py in `flaskr module`) for `/create/` route. Then web server responds with a static page containing two forms which have to be filled in order to permit the plot of maps. As mentioned before, a check on datas is performed client side by javascript, to be sure that every empty value is filled; if not, user is warned if missing input is required, otherwise, missing input is filled with its default value. After form submission, the web server receives datas, and instances a `data_obj` filled with them; two objects are instantiated if a map comparison is required (based on number of forms submitted). After that, the web server instantiate an `orchestrator` object which is deputed to provide final values to the computational server; to do that, the orchestrator has to instantiate a `data_obj` and to fill them with values obtained from the database. Afterwards, the graph server receives values and can plot a single or dual map that is returned to the web server as an html file, and then sent to the client.

3.4 Views

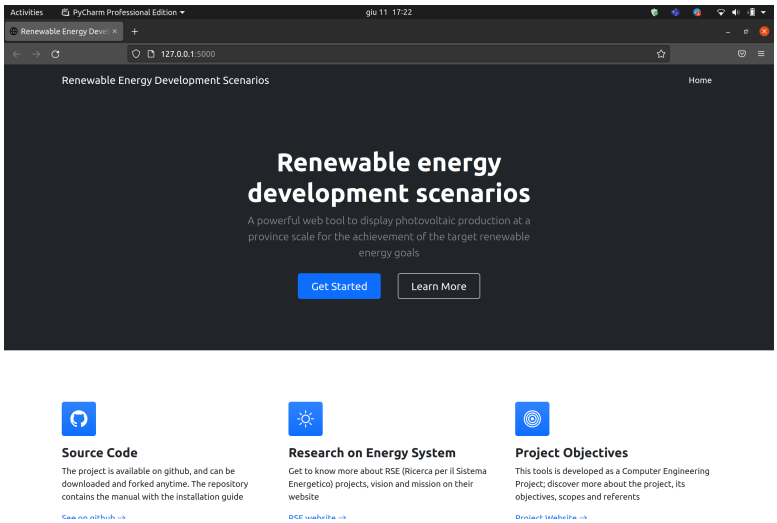

Figure 4: Home page view

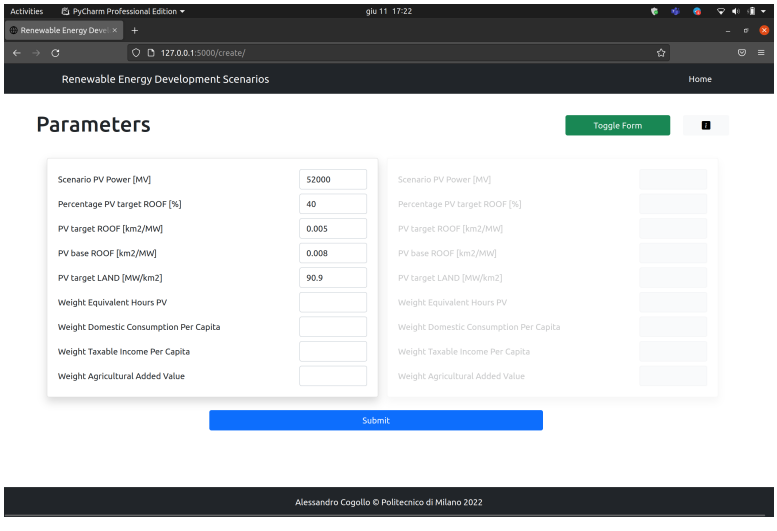

Figure 5: Parameter view

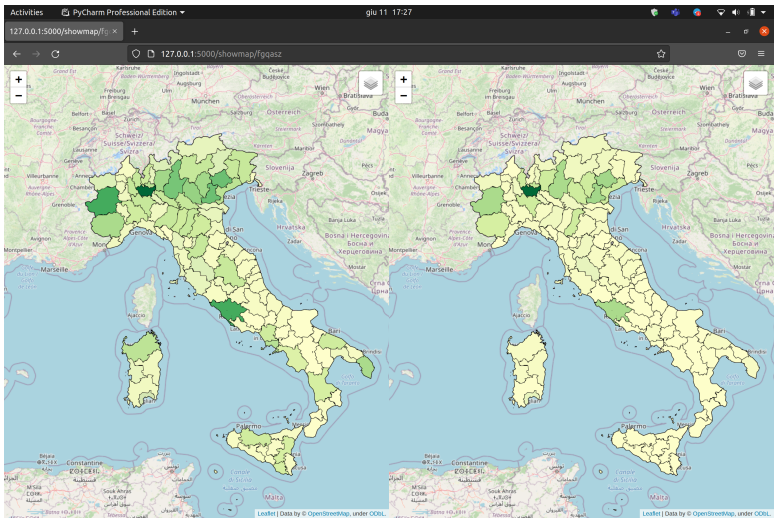

Figure 6: Map comparison view

4 Testing

Tests were carried out during the development phase, exclusively on computational modules.