**Criterion C: Development**

**Dependencies:**

Frontend:

* Bootstrap – CSS library; used for pre-built elements it provides
* jQuery – JavaScript library; used to manipulate HTML elements

Backend:

* Django – Python-based web framework; used to develop the website
* MySQL – relational database management system; used for storing data
* shortuuid – Python library; used to create ids for posts and users
* Pillow – Python library; used to handle images uploaded by users

Overall structure

Gui. bootstrap

Bootstrap

Mysql database

Cities api

Pillow

Shortuuid

Django framework

Jquery

Generate webpages

**Cities API**

**Perfect crutreion a this sentence**

**Two ways Justification and that’s enough**

**Non nonsense**

**Just write I had to write my own because the other own was not apprioprate**

**Just write sentence describing each ss: ss and sentence desvribing it**

**Do it as captions even**

**Reference the backend**

The client required an autocomplete input on area field of the post as well as on the search-by-area input, meaning that for every input on these fields, a list of cities starting with the input will show up. There were two ways this could have been achieved. Firstly, a third-party API could have been used, such as Google Maps Place Autocomplete API. However, after numerous tests, research and investigation I was unable to use it in plain JavaScript and had to create a ‘bridge’ using Django view between the API and the field. This however was inefficient and allowed others to use the API for which I was paying for. The cost issues were another factor which encouraged me to change my technique.

The second way was to create my own database with a list of cities and an API which would return the list based on an input. This was much easier to make and implement. I found a website with a table with cities as well as their population and voivodeship they are in. With a simple Python program (Figure 13), I was able to download the data and save to a .csv file.

When the data was ready, I was able to import it to a MySQL table based on a model (Figure 14) I created earlier. New view (Figure 15) was created to take input and filter through the objects which start with the given parameter. With a jQuery script (Figure 16) I parse data, get it from the API and manipulate the HTML code.

**Validation**

Each input field has to be validated. E-mail fields need to be check for proper format, character inputs need to be checked for any inappropriate phrases (Figure 17) and proper minimum length, passwords need to fulfil set of rules, such as a minimum length of 8 (Figure 18), not being too common (Figure 21), not being too similar to other fields (Figure 20), and not being completely numerical (Figure 19).

**Authentication**

The default credentials used in Django to login are a username and a password. However, due to the nature of the product and client’s requirements the optimal form of logging-in would be an e-mail and a password. A custom backend had to be written in order to authenticate users based on an e-mail instead of a username.

**Reporting posts**

Due to the lack of a software which checks images uploaded by users for any inappropriate and disturbing scenes, such as nudity, crime, etc. a reporting system had to be developed. An authenticated user can report any post once as shown on Figure 7.