The server part

The server was developped in Python 3. It used the web micro framework Flask to handle the HTTP communication, and the ORM provided by SQLAlchemy, using SQLite, to manage the database.

The server is mainly devided in two parts: the setup server, and the localization server. Some part, like the models, and utility functions are shared by both parts. Each server provides a REST API to communicate with using HTTP.

The shared part

The models

Using SQLAlchemy, it is easy to build an ORM mapping some Python classes to tables in a database.

The following models were done:

- AP: storing the access points, with there IP and a name
- Location: storing x and y, plus a one-to-one link to the associated fingerprint
- Fingerprint: storing the link to its location, and a link to its RSSI sample
- RSSISample: storing its RSSI elements
- RSSIElement: storing the access point IP where it came from, and the associated RSSI value

The utility functions

Again, there was two parts: the functions handling the computing of the location, and the ones taking care of the communication with the access points.

Computing locations

`rssi_distance` is the most important. It is the one that compute the distance between two RSSI sample, given a threshold.
With some pythonic magic, it could handle RSSI either as dictionnaries, or as stored RSSISample classes. It returns the distance as a float.

This value is given to the `compute_location` function, that just iterate though the stored fingerprints, and find the one with the closest distance. Then it returns its location.

Communicating with the access points

The communication was made using TCP sockets, so it went easily with the `socket` package provided by Python.

The first function is the `get_rssi_element`, that takes the access point IP, the device MAC, and an element dictionnary to be filled. It opens the connection, sends the MAC, and wait for a 4 bytes value containing the RSSI value. Then it puts it into the dictionnary at the key “IP”.

The second function, `get_rssi_sample`, takes only the device mac, and iterates though the access points to create a thread for each one, and pass it the right values.

Then, all the threads are launched almost at the same time, to ensure the device to be at the same location for each value received.
The element dictionnary filled is now returned, and it is our RSSI sample.

The setup server

The setup server is made to fill the database with every data needed to make the geopositioning to work. It allows to initialize the database, add access points, and finally add localization, computing the corresponding fingerprint using the utility functions. The device's MAC address is stored in a websession, allowing to have many devices setting up the database at the same time.

The localization server

It is almost the same as the setup server, but provide less function: it never writes the database, but uses it only in read mode.
The only difference is that it performs the inversed operation concerning the localization: it gets the RSSI, then find a location given the fingerprint, rather than store a fingerprint corresponding to the given location.