



|  |
| --- |
| **Camping Project** |
| Module 646, Option GIS-Python |

|  |
| --- |
| Pierre ANKEN – Montaine BURGER – Nghi TRAN  the 12th of June 2020 |

Table of contents

[Structure of the project 3](#_Toc42776998)

[How to setup your project locally 4](#_Toc42776999)

[Pre-requirements 4](#_Toc42777000)

[Setup local DB in pgAdmin 4](#_Toc42777001)

[Import camping shapes 6](#_Toc42777002)

[Import environment 6](#_Toc42777003)

[Setup PyCharm 6](#_Toc42777004)

[WARNING: Windows users 8](#_Toc42777005)

[Design of our project 9](#_Toc42777006)

[Mockups 9](#_Toc42777007)

[Product Backlog 9](#_Toc42777008)

[User authentication part 10](#_Toc42777009)

[Integrated module inside Django 10](#_Toc42777010)

[Custom user model 11](#_Toc42777011)

[Custom registration form 11](#_Toc42777012)

[Login on the application 12](#_Toc42777013)

[Custom homepage and logout from the application 13](#_Toc42777014)

[Geographic Information System part 15](#_Toc42777015)

[Our toolbox 15](#_Toc42777016)

[Functions used 15](#_Toc42777017)

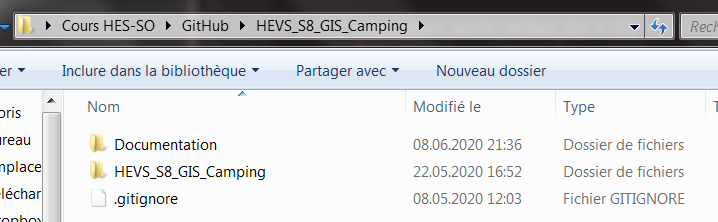
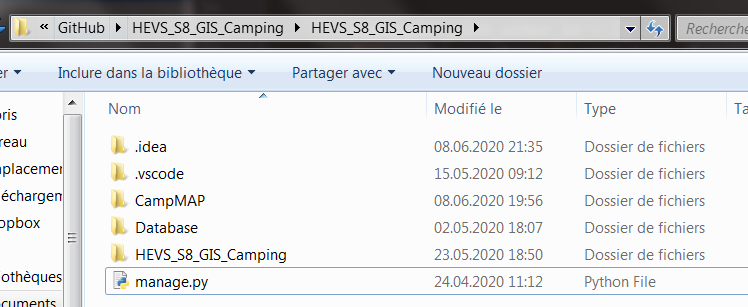
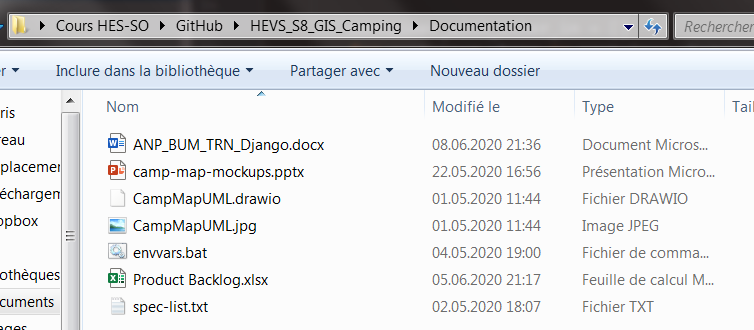
[Use of the different filters 16](#_Toc42777018)

[Explanation 17](#_Toc42777019)

# Structure of the project

We collaborated on this project through a Git repository. You can find it at the following address:

[*https://github.com/PierreAnken/HEVS\_S8\_GIS\_Camping*](https://github.com/PierreAnken/HEVS_S8_GIS_Camping)

Here is the structure of our project: we have created two separate directories, one for the documentation (with all the files mentioned in this report) and one for the Django application.

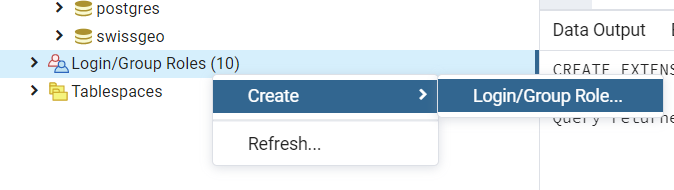
# How to setup your project locally

## Pre-requirements

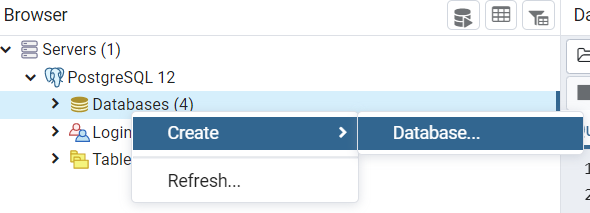
* [PostgreSQL V12](https://www.postgresql.org/download/windows/) installed with PostGIS and pgAdmin modules
* Clone Github Project: <https://github.com/PierreAnken/HEVS_S8_GIS_Camping.git>
* Anaconda 3.7 available at <https://www.anaconda.com/products/individual>

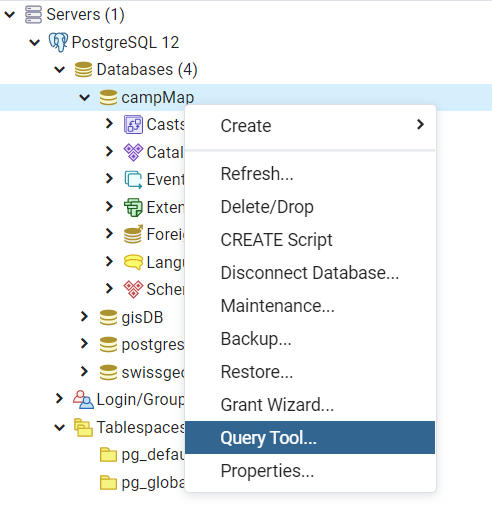
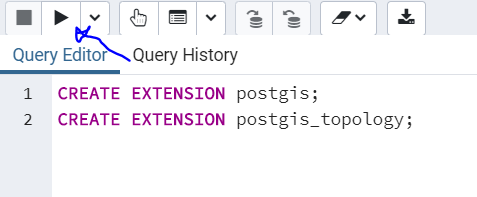
## Setup local DB in pgAdmin

First, create a new admin user:

* Name: admin
* Password: adminPWD
* Privileges: can login + superuser

Then, create a new database:

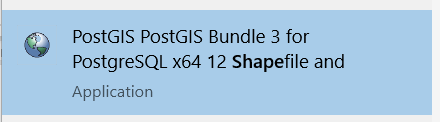
* Database: campMap
* Owner: admin

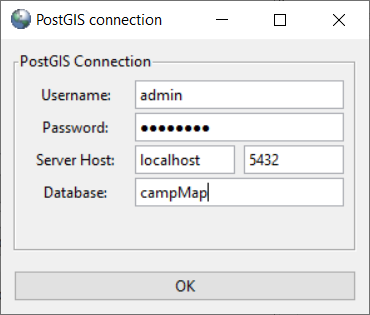
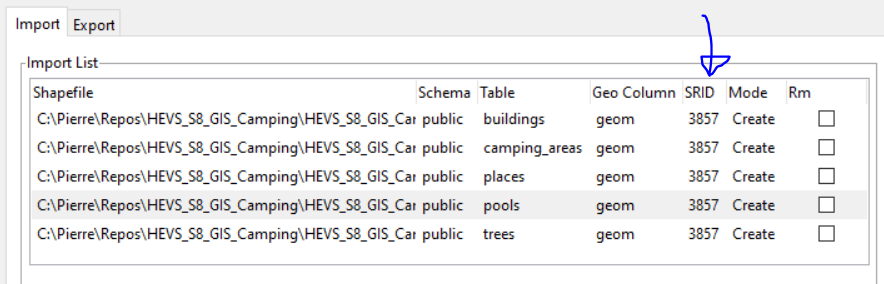
You have to enable PostGIS in your new database:

In text below, so you can copy and paste it:

**CREATE EXTENSION** postgis;  
**CREATE EXTENSION** postgis\_topology;

## Import camping shapes

Search for “shapes” in windows search menu:

Then, you can edit connection details as created before:

**Note:** import files from GitHub to have correct table names!

## Import environment

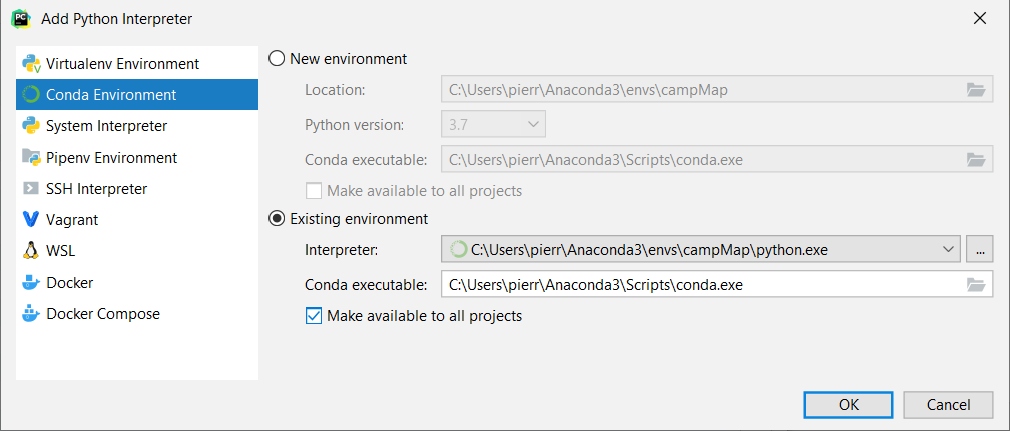
Open Anaconda Navigator interface and import env. from the file in GitHub, under:  
*HEVS\_S8\_GIS\_Camping \Documentation\spec-list.txt*

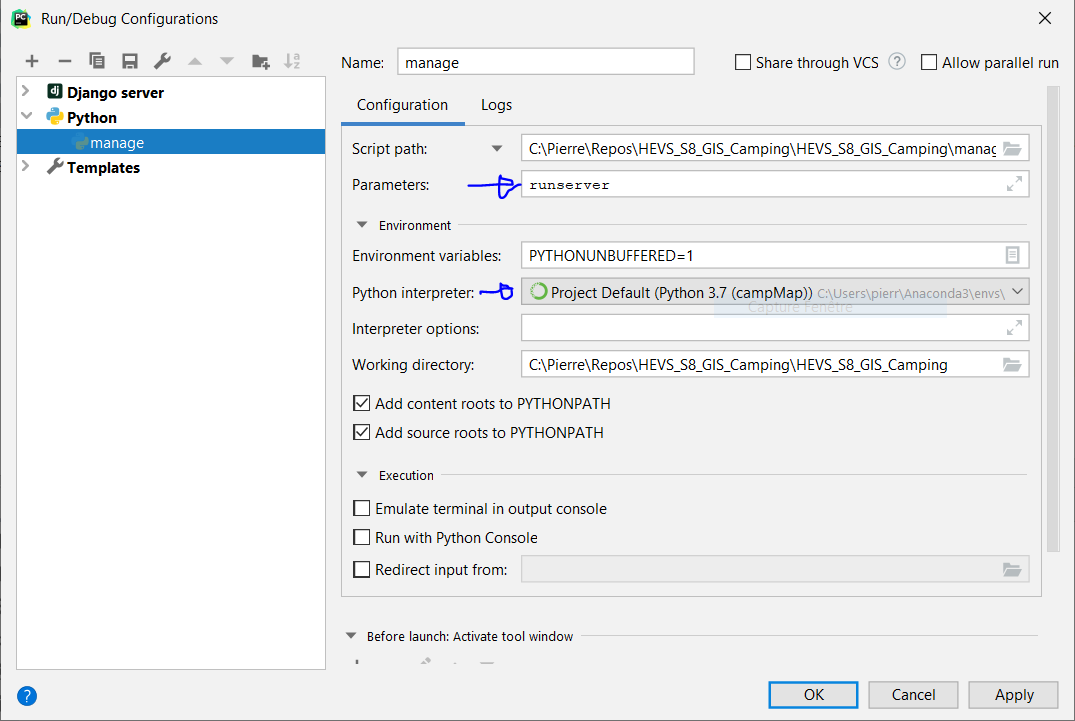
## Setup PyCharm

If you want to work with PyCharm, here are some advices to make it easier:

First, open the folder *HEVS\_S8\_GIS\_Camping\HEVS\_S8\_GIS\_Camping* in PyCharm as a project.

Then, under Settings > Project > Project Interpreters > add a new interpreter (cogs icon top right).

Select the python.exe inside the env. you imported previously.

After that, in PyCharm, right click on *manage.py* and run. It won’t work, but it will create you a run configuration (drop menu top right). You can edit the configuration as below:

Let PyCharm update everything with the new configuration and run your project.  
**Note:** install from PyCharm terminal any missing module with *pip install [package name]*.

## WARNING: Windows users

It is important to follow the procedure provided by Django: <https://docs.djangoproject.com/en/3.0/ref/contrib/gis/install/#windows>

And don’t forget to install *OSGeo4W*!

Then, verify to have the repositories mentioned at the top of *settings.py* installed on your computer (and at the same locations!). If it still doesn't work, as last resort, you can create a batch to change your environment variables.

**set** OSGEO4W\_ROOT**=**C:\OSGeo4W64

**set** PYTHON\_ROOT**=**C:\Users\[USER]\AppData\Local\Programs\Python\Python38-32

**set** GDAL\_DATA**=%OSGEO4W\_ROOT%**\share\gdal

**set** PROJ\_LIB**=%OSGEO4W\_ROOT%**\share\proj

**set** PATH**=%PATH%**;**%PYTHON\_ROOT%**;**%OSGEO4W\_ROOT%**\bin

reg ADD "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /v **Path** /t REG\_EXPAND\_SZ /f /d "**%PATH%**"

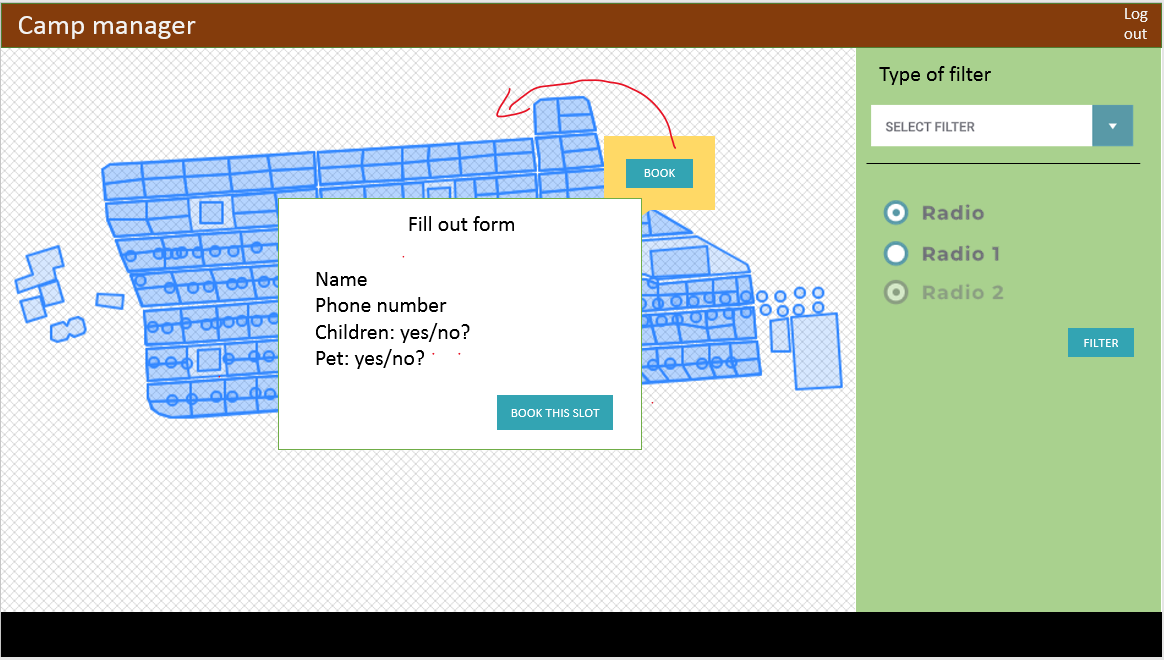
reg ADD "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /v GDAL\_DATA /t REG\_EXPAND\_SZ /f /d "**%GDAL\_DATA%**"

reg ADD "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /v PROJ\_LIB /t REG\_EXPAND\_SZ /f /d "**%PROJ\_LIB%**"

# Design of our project

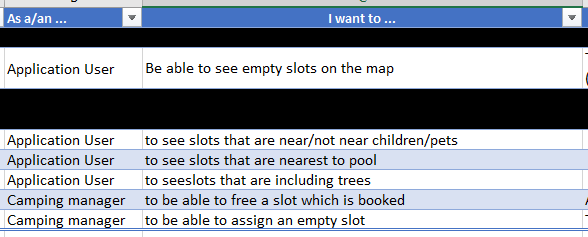
We dedicated our first Teams meetings at designing our project, by creating some mockups and defining a product backlog (user stories).

## Mockups

Here is a sample of the mockups, but you can find the entire file under *HEVS\_S8\_GIS\_Camping\Documentation\camp-map-mockups.pptx*

## Product Backlog

Here is an overview of the product backlog we defined, but again, you can find the entire file under *HEVS\_S8\_GIS\_Camping\Documentation\Product Backlog.xlsx*



# User authentication part

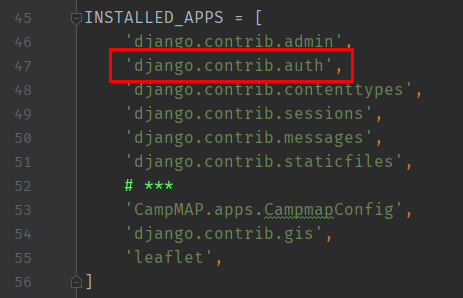
Our wish was to manage users (especially camper and manager roles). The manager can be created from the admin console, since it is the superuser proposed by Django. We have already created the default superuser: **name** **>** admin and **password >** adminPWD.

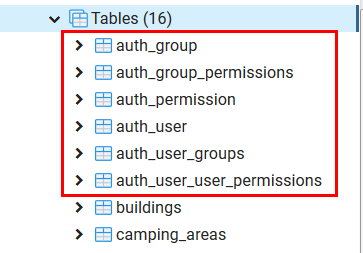
For the camper accounts, we wanted something as in “reality”. That is, users can create their account from the frontend.

**Note:** the command to create a superuser is ***python manage.py createsuperuser*** and then you just have to complete the asked information (user, email and password).

## Integrated module inside Django

We used the module proposed by Django for the user authentication, which is *django.contrib.auth*.

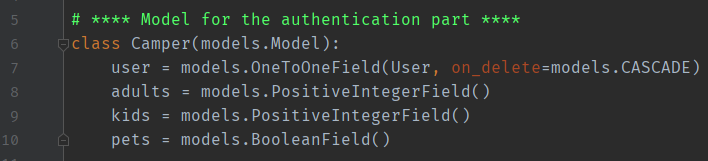
To do this, simply check that the module is in the *INSTALLED\_APPS* of *settings.py* (normally it is there by default).

First, to make this part work in synchronization with the database, you have to make migrations for the application: ***python manage.py makemigrations [app name]*** and then, migrate the project: ***python manage.py migrate***. This action will add all required tables inside the database.

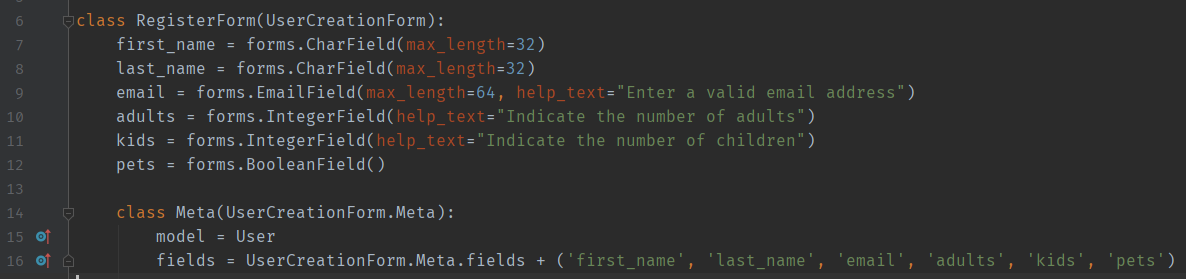
**Note:** you can drop every table which is not a “shape table” in your database and just run the following command to synchronize the database. ***python manage.py migrate --run-syncdb***

## Custom user model

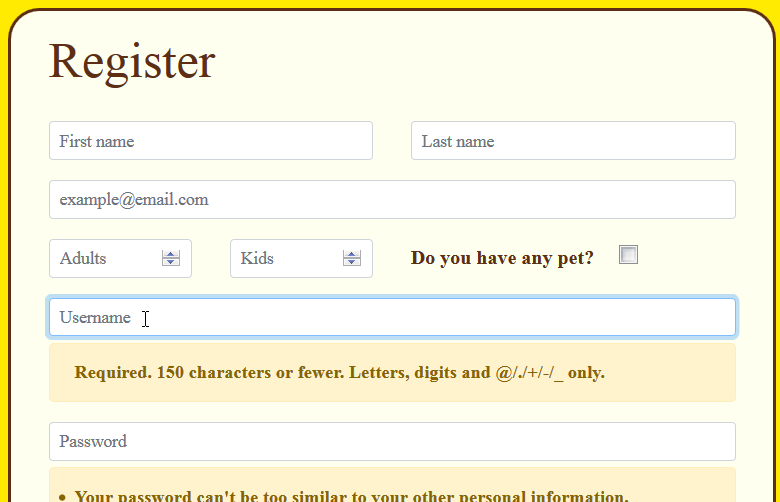
As shown in our database diagram, the *Camper* inherits from the base Django *User*. This allows us to use all the basic authentication features through the *User*, but also to add the fields that we want to save for our *Camper*.

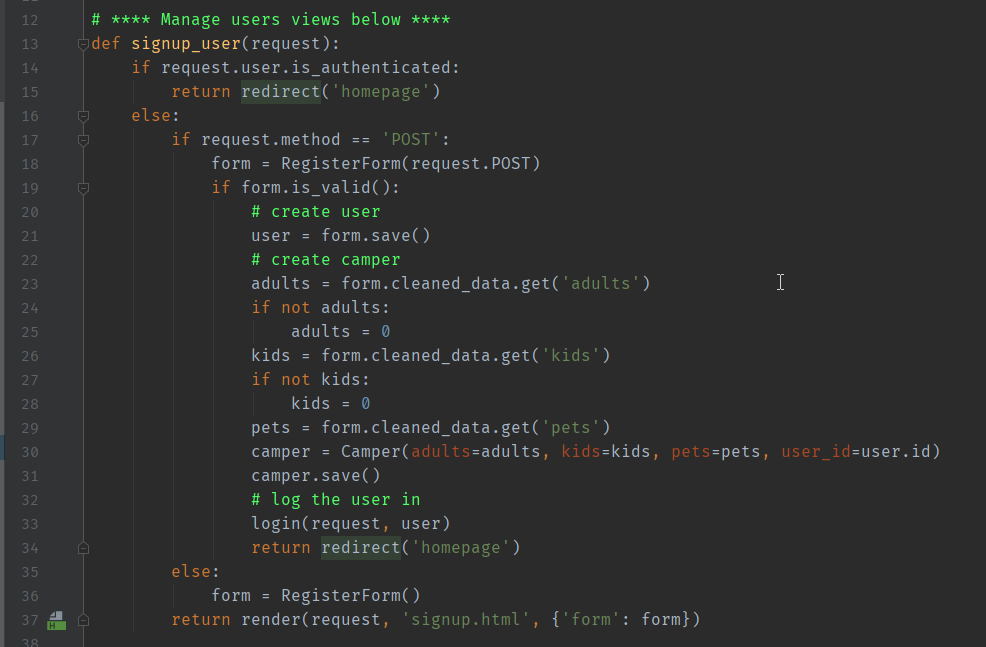
To make this configuration work, we just had to add a one to one relationship:

## Custom registration form

In order to be able to save these new added fields, it was also necessary to rewrite the basic registration form. The goal was to use the basic form, with its functionalities, but with the addition of our custom fields.

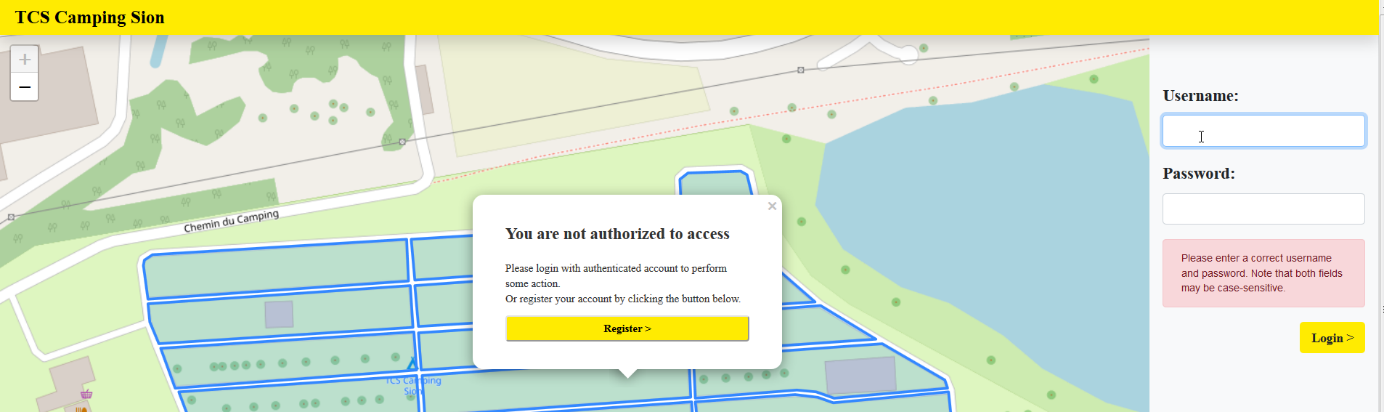
Then we were able to create the registration form on the front end and save the new registrations by creating linked user and camper.

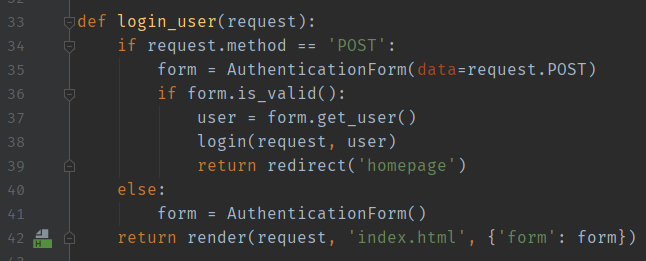




Finally, we can find the campers registered on the application in our database.

## Login on the application

Obviously, our custom users, i.e. the campers, can identify themselves on the application by logging in from the principal page.

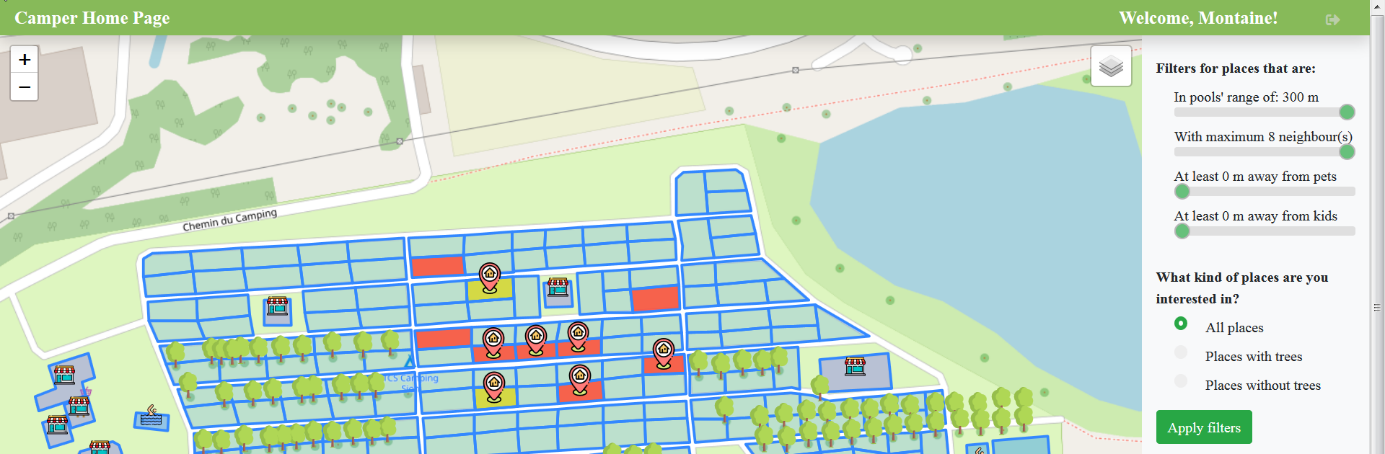


## Custom homepage and logout from the application

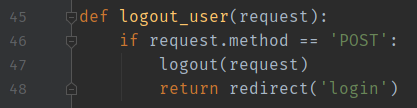
They then land on the camping homepage. This page is modified according to the rights assigned to the user (*manager* or *camper*). From these pages, they will be able to perform several actions described below.

The camper, in his side, gets an overview of the campsite: with the free, reserved and occupied places.

He can also filter the display of the places according to different criteria: distance from the swimming pool, from children or from animals + maximum number of neighboring places+ places with or without trees.

Finally, he’s able to book one or more slots and see his own bookings marked by an icon.

As can be seen in this screenshot, the logged-in user can also log out from the application and be redirected to the main login page.



The manager (who corresponds to the superuser created before) has an overview of all the bookings.

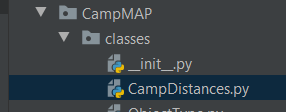
On one side of the screen, he can find the pending reservations of the campers, which he can accept or decline.

On the other side of the screen, he has a list of the booked places and a quick view of the availability of the campsite with the help of a small map displayed as a summary.

He can also delete some reservations and free the places related directly from the list.

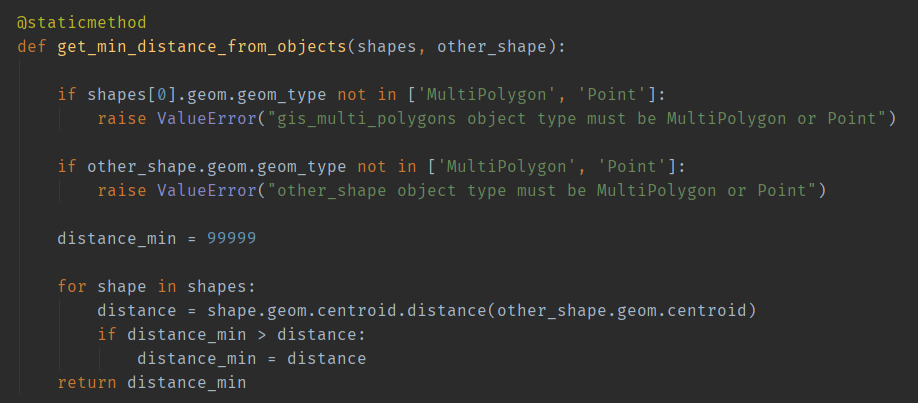
# Geographic Information System part

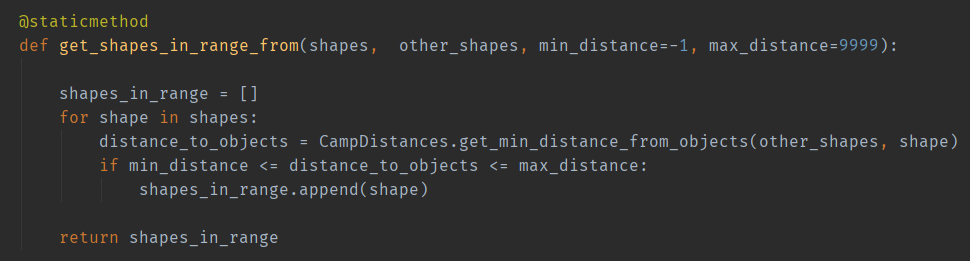
## Our toolbox

We have created a .py file that serves as a toolbox for this project. It is within this file that we have created all the classes that we use to manipulate and perform all the geographic functions used in the project and the display of the shapes.

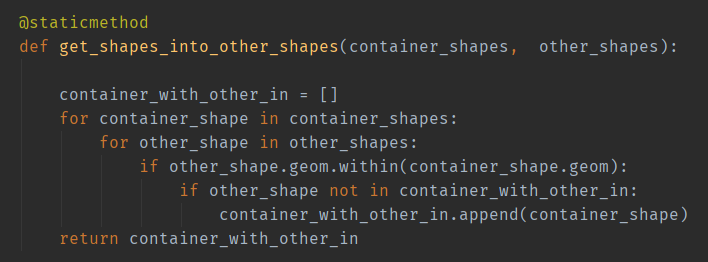
## Functions used

• **get\_min\_distance\_from\_objects:** this function computes the minimal distance between two collections of shapes.

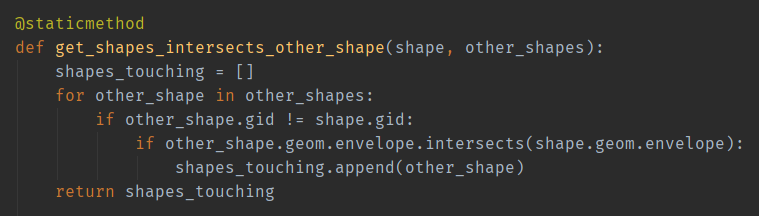
*Geo use: centroid, distance() and geom\_type.*

• **get\_shapes\_in\_range\_from:** this function retrieves all the shapes in range (min – max) from other shapes collection.

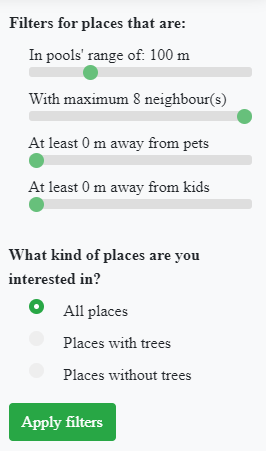
• **get\_shapes\_into\_other\_shapes**: this function checks which shapes of a collection are inside the boundaries of a shape from another collection.

*Geo use: within.*

• **get\_shapes\_intersects\_other\_shape**: this function checks if a shape intersects with a shape of another collection.

*Geo use: envelope, intersects.*

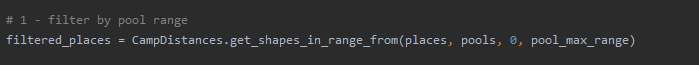
## Use of the different filters

We implemented 4 filters:

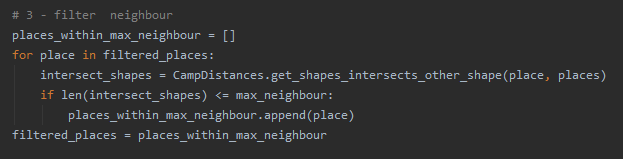
* **Pool filter:** filter for places in user selected range of the pools
* **Neighbor filter:** filter for places that have a certain maximum amount of places right next to it
* **Pet/Children filter:** filter for places that are at least X meters away from booked campers with pets/children
* **Tree filter:** filter for places that have trees inside their slot

## Explanation

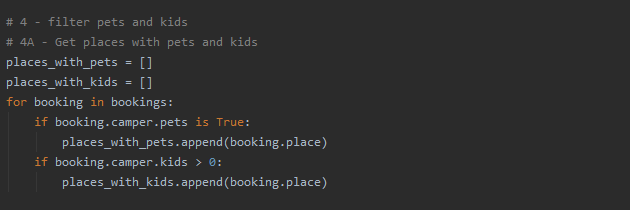
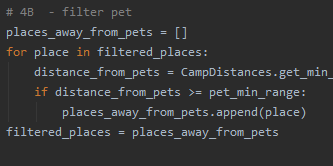
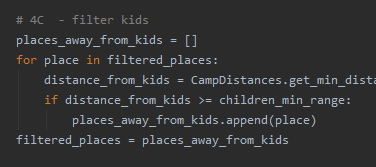
* **Pool filter:** we used the ***get\_shapes\_in\_range\_from*** explained above.



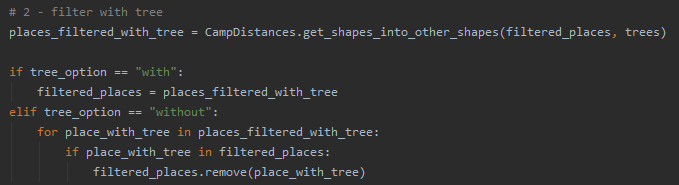
* **Neighbor filter:** we fetch intersecting places for each place and check for the number of intersecting (neighboring) places.



* **Pet/Children filter:** We first get places booked by campers with pets/children, then filter for places that are X meters away from those places.

* **Tree filter:** we get places that contains trees within them. Depending on whether users want places with/without trees, places with tree will be included/excluded to the filtered results.



**How filters are combined:** all filters are applied at once, when users press  button.