Hes·so VALAIS WALLIS
Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

CAMPING PROJECT

Module 646, Option GIS-Python

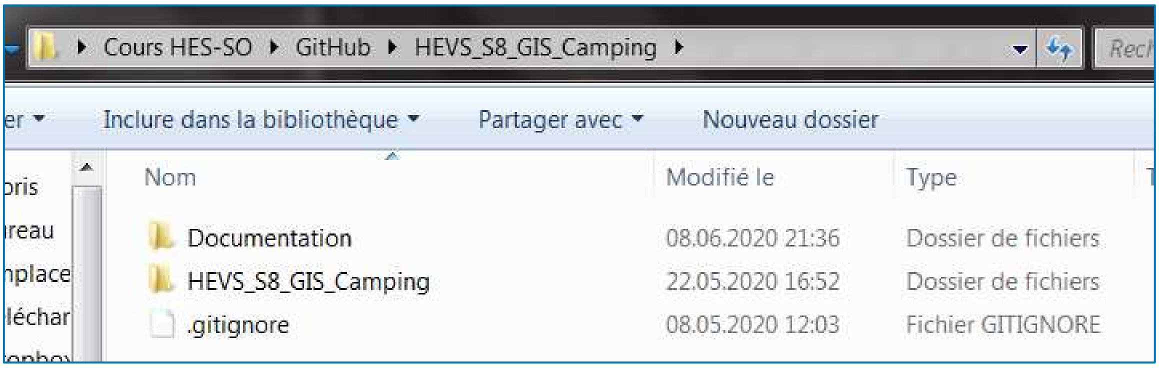
Pierre ANKEN – Montaine BURGER – Nghi TRAN
the 12th of June 2020

TABLE OF CONTENTS

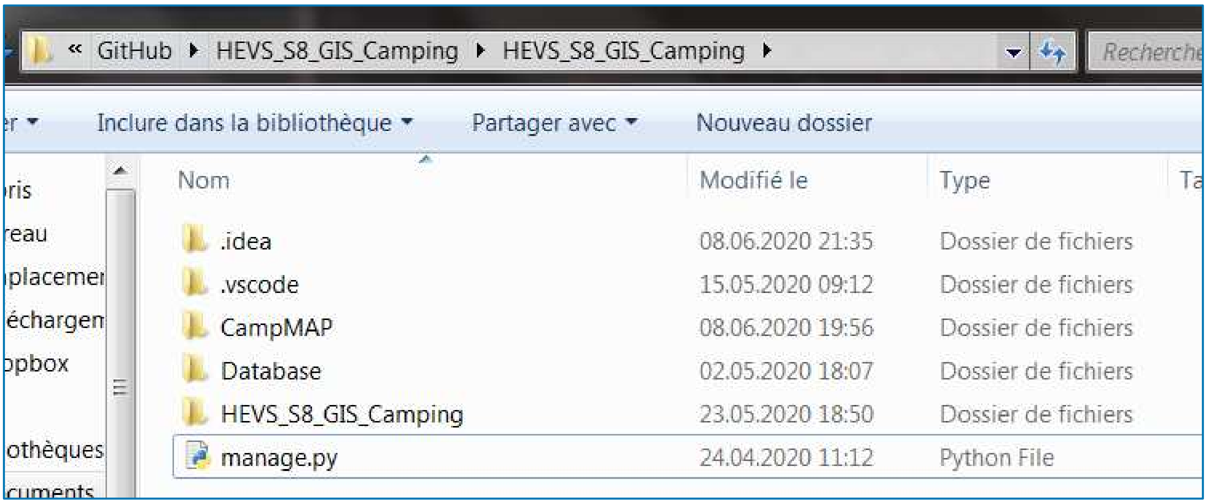
STRUCTURE OF THE PROJECT

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*

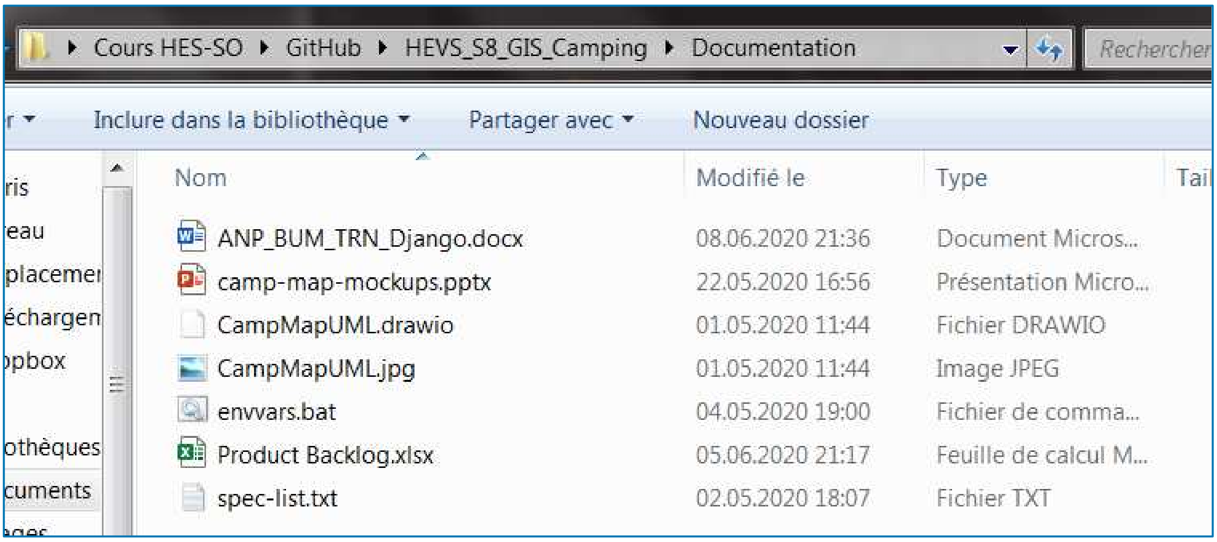
Here is the structure of our project: we have created two separate directories, one for the documentation (with all the files mentioned in this report) and one for the Django application.


Cours HES-SO ▸ GitHub ▸ HEVS_S8_GIS_Camping ▸
Rech
Inclure dans la bibliothèque ▾
Partager avec ▾
Nouveau dossier

Nom	Modifié le	Type
Documentation	08.06.2020 21:36	Dossier de fichiers
HEVS_S8_GIS_Camping	22.05.2020 16:52	Dossier de fichiers
.gitignore	08.05.2020 12:03	Fichier GITIGNORE


« GitHub ▸ HEVS_S8_GIS_Camping ▸ HEVS_S8_GIS_Camping ▸
Recherche
Inclure dans la bibliothèque ▾
Partager avec ▾
Nouveau dossier

Nom	Modifié le	Type
.idea	08.06.2020 21:35	Dossier de fichiers
.vscode	15.05.2020 09:12	Dossier de fichiers
CampMAP	08.06.2020 19:56	Dossier de fichiers
Database	02.05.2020 18:07	Dossier de fichiers
HEVS_S8_GIS_Camping	23.05.2020 18:50	Dossier de fichiers
manage.py	24.04.2020 11:12	Python File


Cours HES-SO ▸ GitHub ▸ HEVS_S8_GIS_Camping ▸ Documentation
Rechercher
Inclure dans la bibliothèque ▾
Partager avec ▾
Nouveau dossier

Nom	Modifié le	Type
ANP_BUM_TRN_Django.docx	08.06.2020 21:36	Document Micros...
camp-map-mockups.pptx	22.05.2020 16:56	Présentation Micro...
CampMapUML.drawio	01.05.2020 11:44	Fichier DRAWIO
CampMapUML.jpg	01.05.2020 11:44	Image JPEG
envvars.bat	04.05.2020 19:00	Fichier de comma...
Product Backlog.xlsx	05.06.2020 21:17	Feuille de calcul M...
spec-list.txt	02.05.2020 18:07	Fichier TXT

HOW TO SETUP YOUR PROJECT LOCALLY

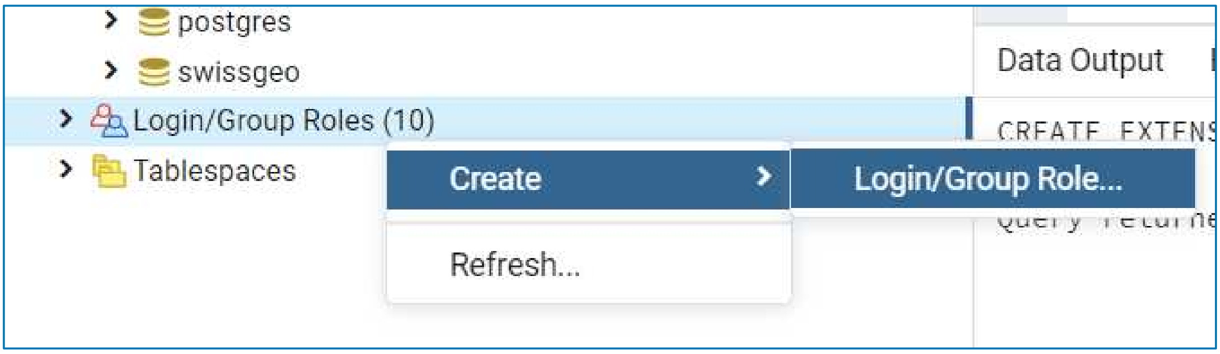
Pre-requirements

- PostgreSQL V12 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

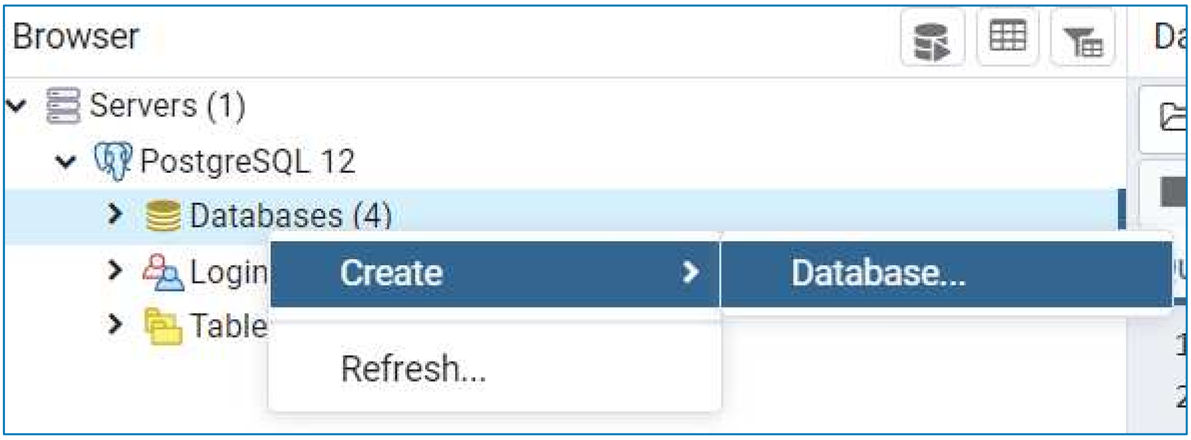
First, create a new admin user:

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

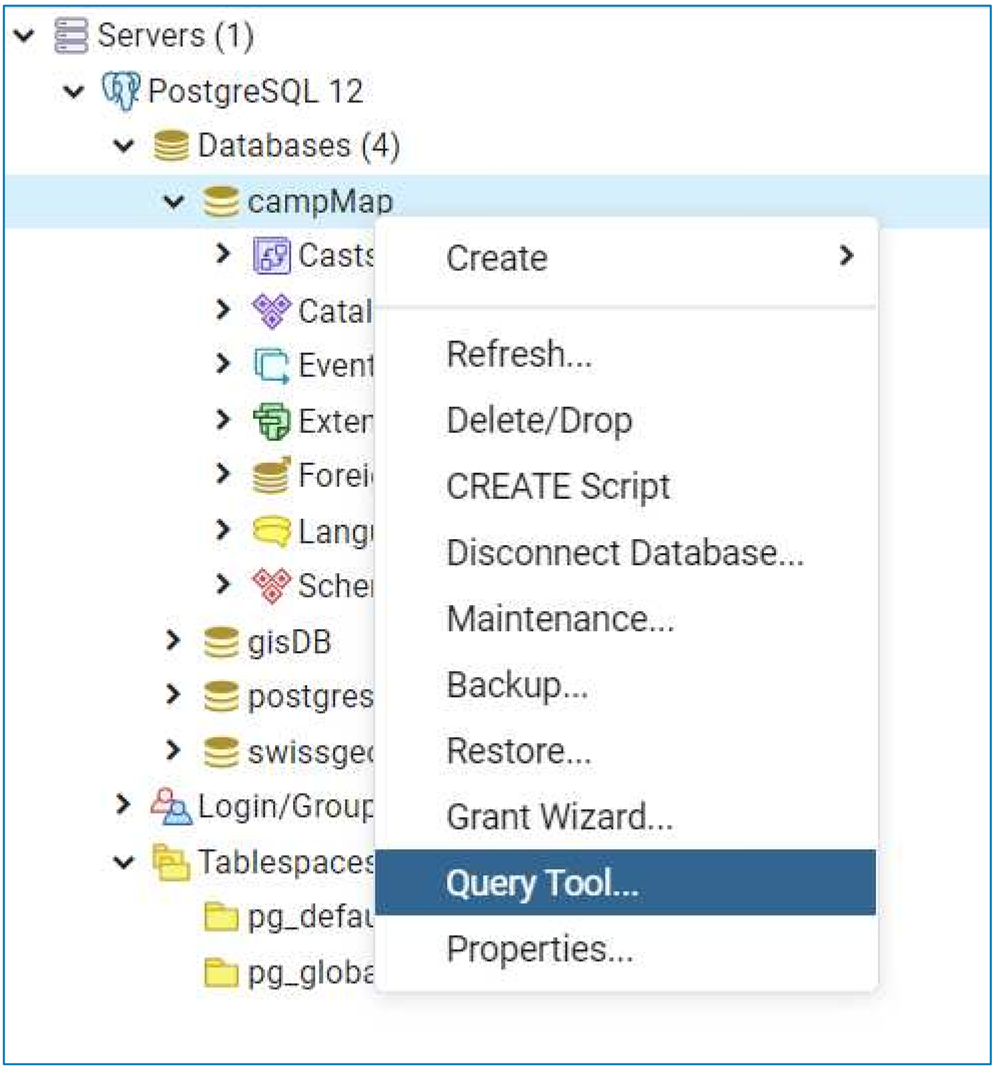


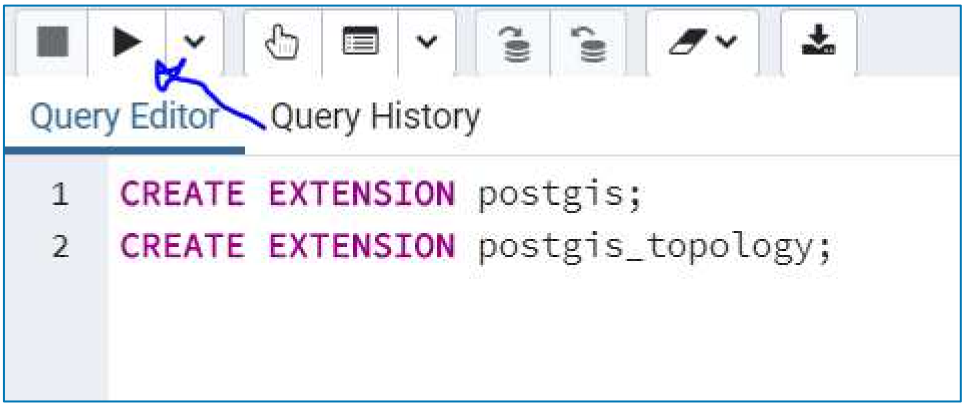
Then, create a new database:

- Database: campMap
- Owner: admin



You have to enable PostGIS in your new database:





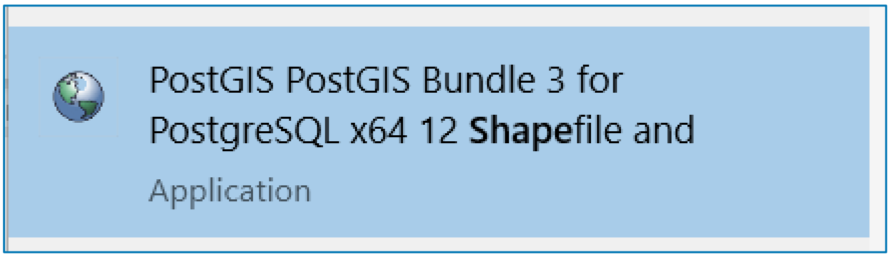
```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
```

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

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
```

Import camping shapes

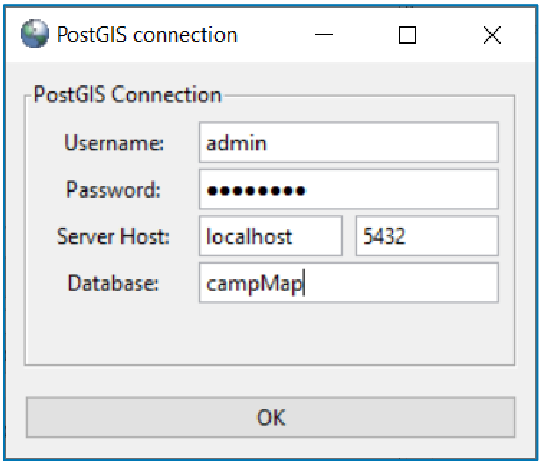
Search for “shapes” in windows search menu:



PostGIS PostGIS Bundle 3 for PostgreSQL x64 12 **Shape**file and

Application

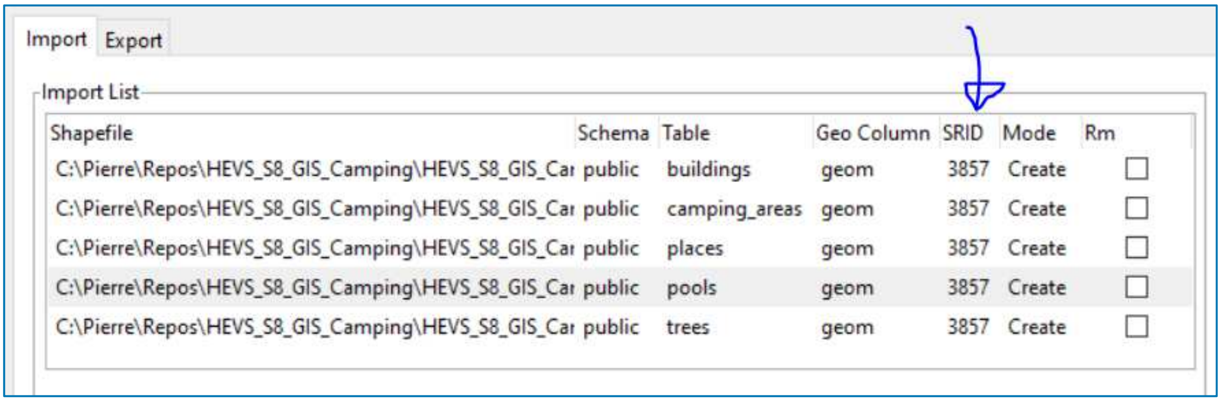
Then, you can edit connection details as created before:



PostGIS connection

PostGIS Connection

Username:	admin	
Password:	●●●●●●●●	
Server Host:	localhost	5432
Database:	campMap	



Import List

Shapefile	Schema	Table	Geo Column	SRID	Mode	Rm
C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Car	public	buildings	geom	3857	Create	□
C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Car	public	camping_areas	geom	3857	Create	□
C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Car	public	places	geom	3857	Create	□
C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Car	public	pools	geom	3857	Create	□
C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Car	public	trees	geom	3857	Create	□

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*

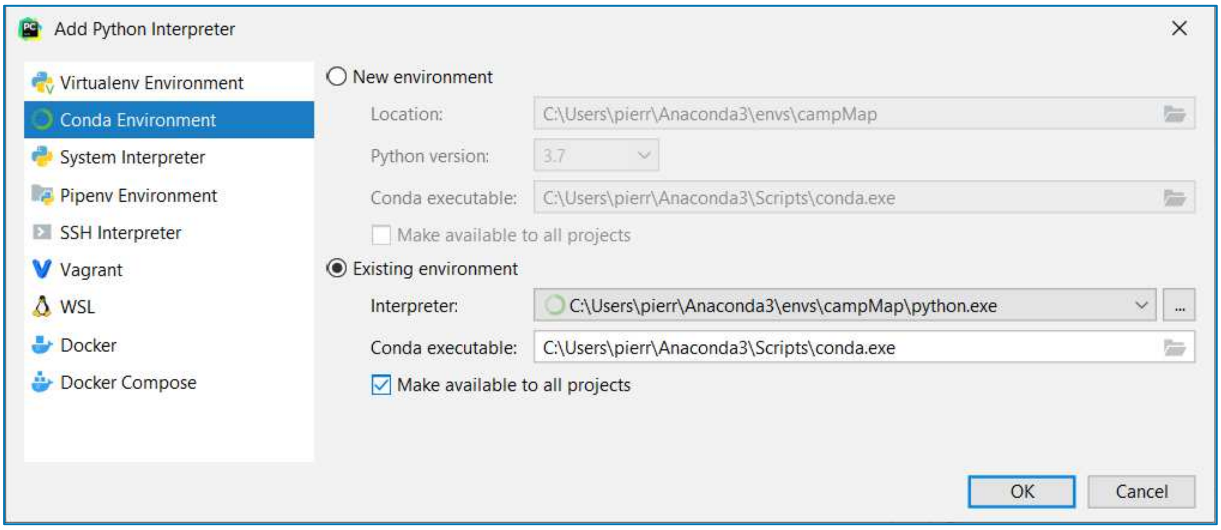
Setup PyCharm

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.



Add Python Interpreter

○ New environment

Location: C:\Users\pierr\Anaconda3\envs\campMap

Python version: 3.7

Conda executable: C:\Users\pierr\Anaconda3\Scripts\conda.exe

□ Make available to all projects

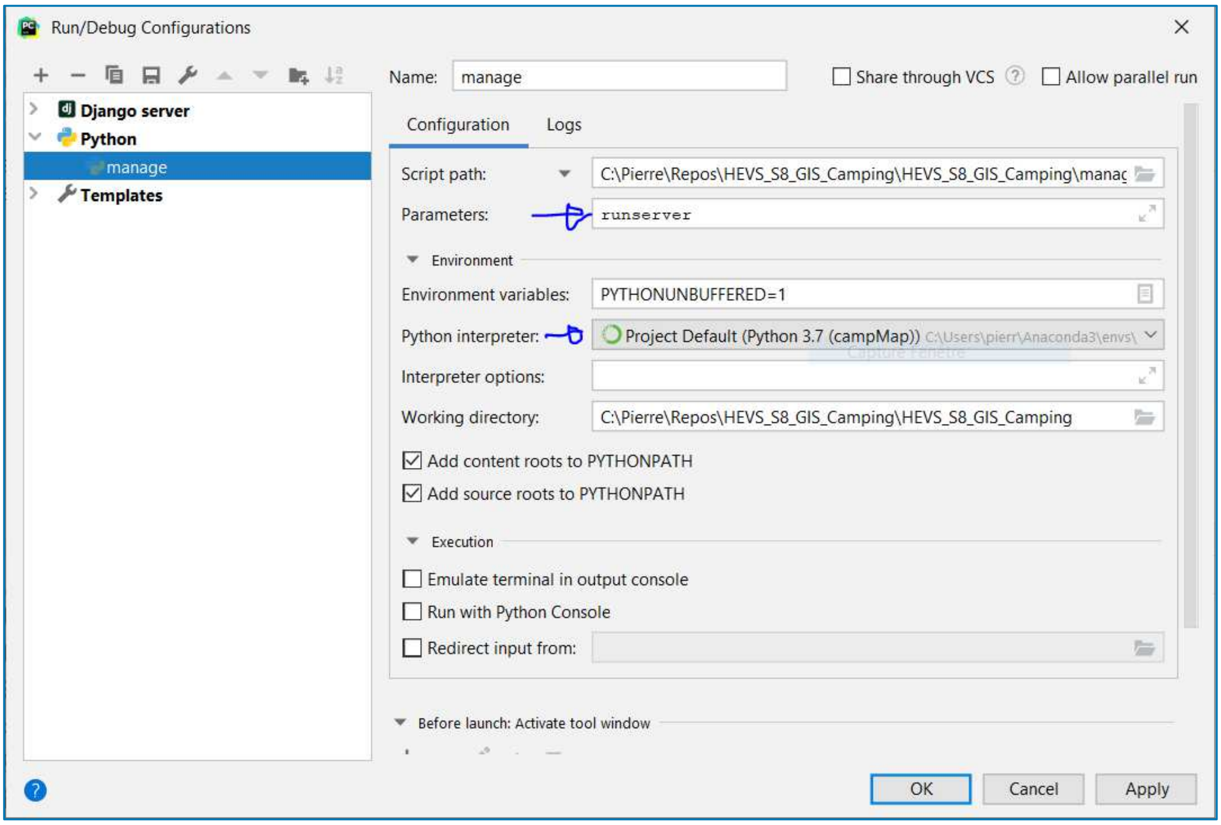
● Existing environment

Interpreter: C:\Users\pierr\Anaconda3\envs\campMap\python.exe

Conda executable: C:\Users\pierr\Anaconda3\Scripts\conda.exe

☑ Make available to all projects

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:



Run/Debug Configurations

Name: manage

□ Share through VCS □ Allow parallel run

Script path: C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Camping\manag

Parameters: runserver

Environment

Environment variables: PYTHONUNBUFFERED=1

Python interpreter: Project Default (Python 3.7 (campMap)) C:\Users\pierr\Anaconda3\envs\

Interpreter options:

Working directory: C:\Pierre\Repos\HEVS_S8_GIS_Camping\HEVS_S8_GIS_Camping

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

Execution

□ Emulate terminal in output console

□ Run with Python Console

□ Redirect input from:

Before launch: Activate tool window

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.

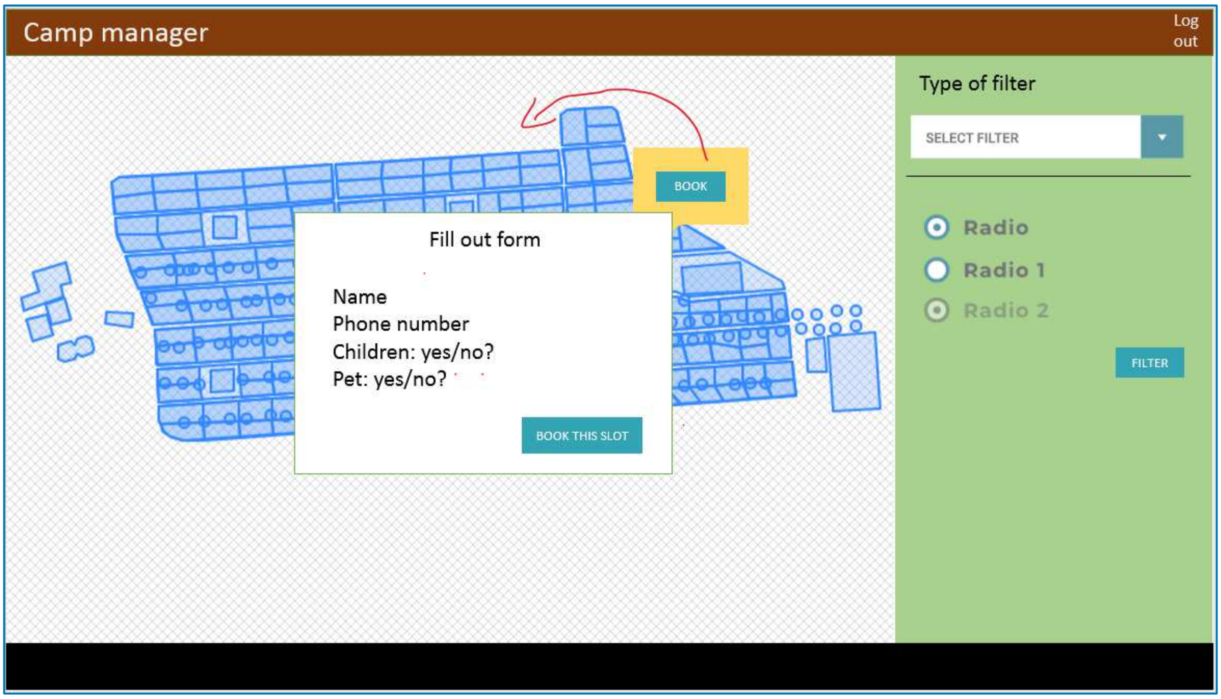
```
set OSGEO4W_ROOT=C:\OSGeo4W64
set PYTHON_ROOT=C:\Users\[USER]\AppData\Local\Programs\Python\Python38-32
set GDAL_DATA=%OSGEO4W_ROOT%\share\gdal
set PROJ_LIB=%OSGEO4W_ROOT%\share\proj
set PATH=%PATH%;%PYTHON_ROOT%;%OSGEO4W_ROOT%\bin
reg ADD "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment"
/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*

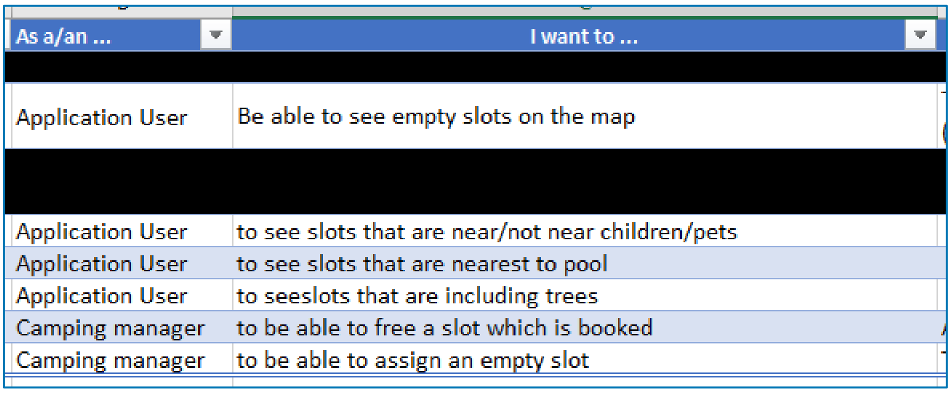


Fill out form

Name
Phone number
Children: yes/no?
Pet: yes/no?

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*



As a/an ...	I want to ...
[redacted]	[redacted]
Application User	Be able to see empty slots on the map
[redacted]	[redacted]
Application User	to see slots that are near/not near children/pets
Application User	to see slots that are nearest to pool
Application User	to seeslots that are including trees
Camping manager	to be able to free a slot which is booked
Camping manager	to be able to assign an empty slot

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.

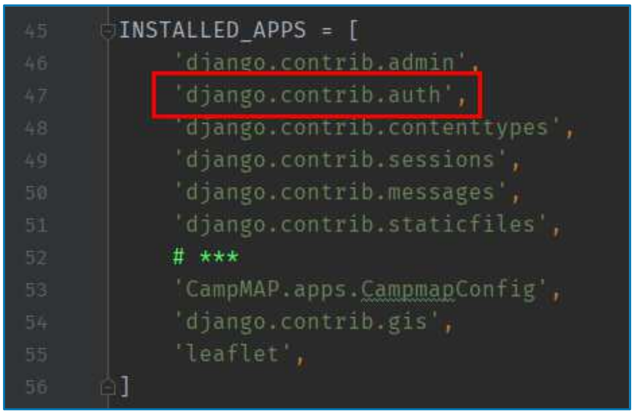
For the camper accounts, we wanted something as in “reality”. That is, users can create their account from the frontend.

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

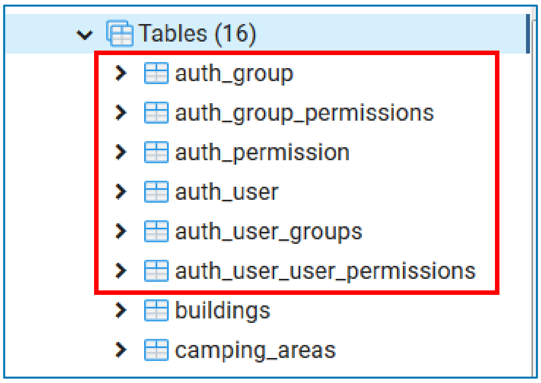
We used the module proposed by Django for the user authentication, which is *django.contrib.auth*.

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



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # ***
    'CampMAP.apps.CampmapConfig',
    'django.contrib.gis',
    'leaflet',
]
```

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.



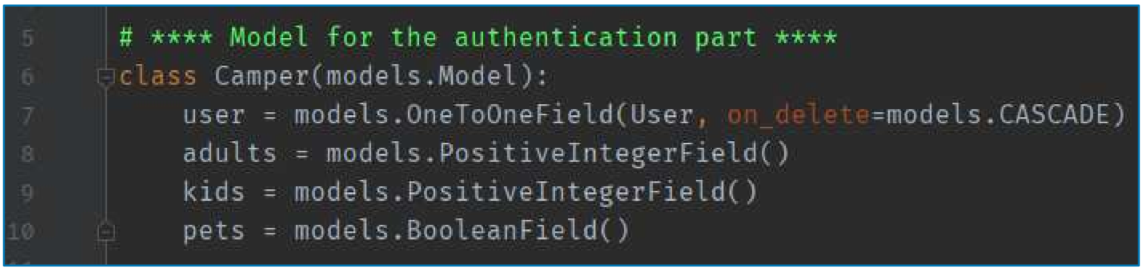
Tables (16)
- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissions
- buildings
- camping_areas

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*.

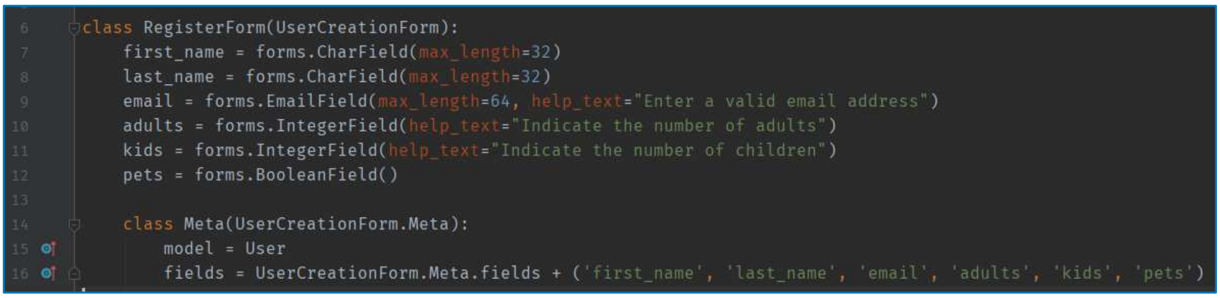
To make this configuration work, we just had to add a one to one relationship:



```python
# **** Model for the authentication part ****
class Camper(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    adults = models.PositiveIntegerField()
    kids = models.PositiveIntegerField()
    pets = models.BooleanField()
```

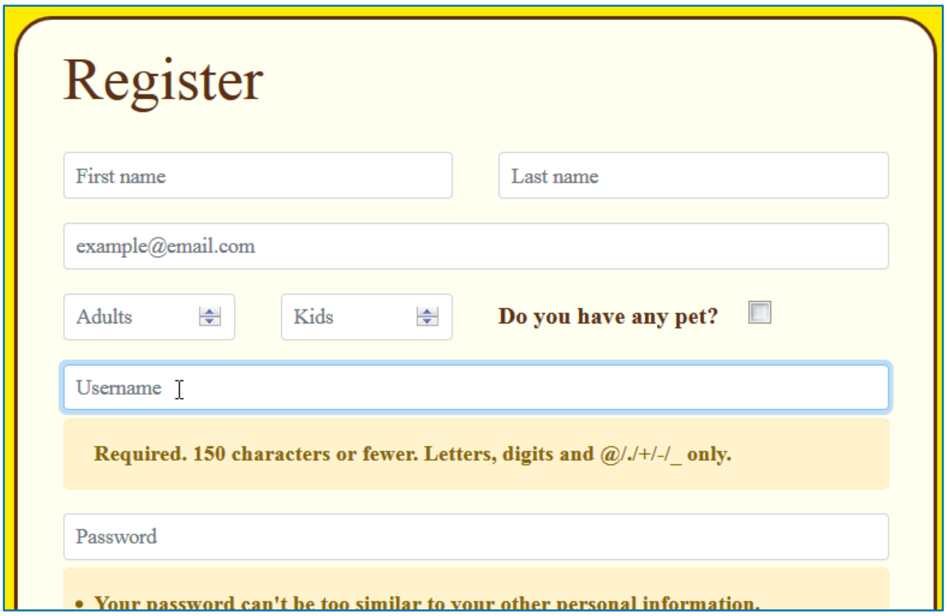
Custom registration form

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.



```python
class RegisterForm(UserCreationForm):
    first_name = forms.CharField(max_length=32)
    last_name = forms.CharField(max_length=32)
    email = forms.EmailField(max_length=64, help_text="Enter a valid email address")
    adults = forms.IntegerField(help_text="Indicate the number of adults")
    kids = forms.IntegerField(help_text="Indicate the number of children")
    pets = forms.BooleanField()

    class Meta(UserCreationForm.Meta):
        model = User
        fields = UserCreationForm.Meta.fields + ('first_name', 'last_name', 'email', 'adults', 'kids', 'pets')
```

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



Register

First name

Last name

example@email.com

Adults

Kids

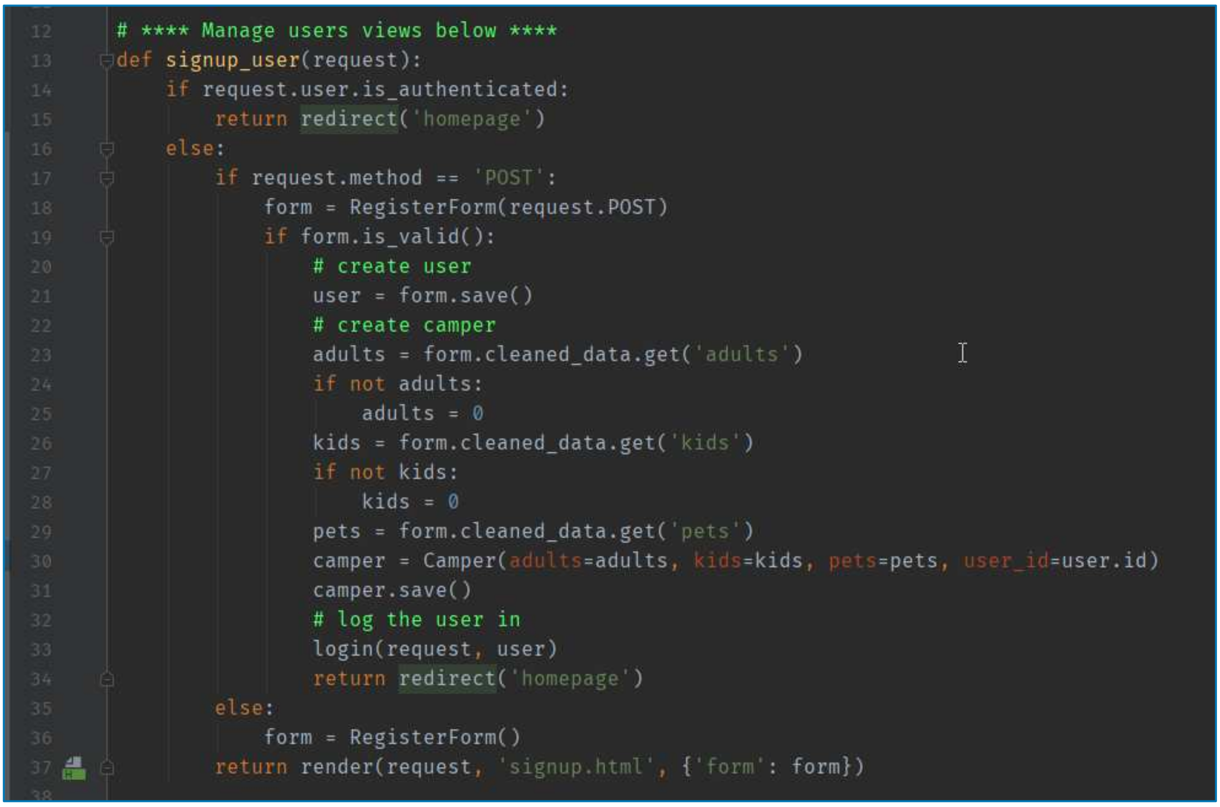
Do you have any pet? □

Username

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

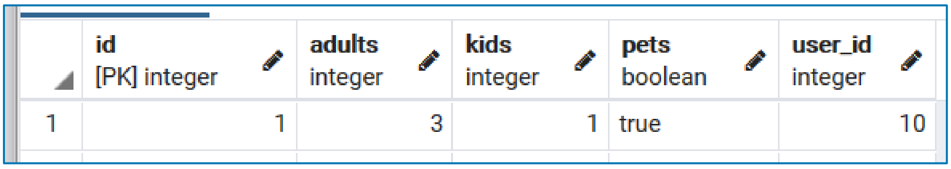
Password

- **Your password can't be too similar to your other personal information.**



```python
# **** Manage users views below ****
def signup_user(request):
    if request.user.is_authenticated:
        return redirect('homepage')
    else:
        if request.method == 'POST':
            form = RegisterForm(request.POST)
            if form.is_valid():
                # create user
                user = form.save()
                # create camper
                adults = form.cleaned_data.get('adults')
                if not adults:
                    adults = 0
                kids = form.cleaned_data.get('kids')
                if not kids:
                    kids = 0
                pets = form.cleaned_data.get('pets')
                camper = Camper(adults=adults, kids=kids, pets=pets, user_id=user.id)
                camper.save()
                # log the user in
                login(request, user)
                return redirect('homepage')
        else:
            form = RegisterForm()
        return render(request, 'signup.html', {'form': form})
```

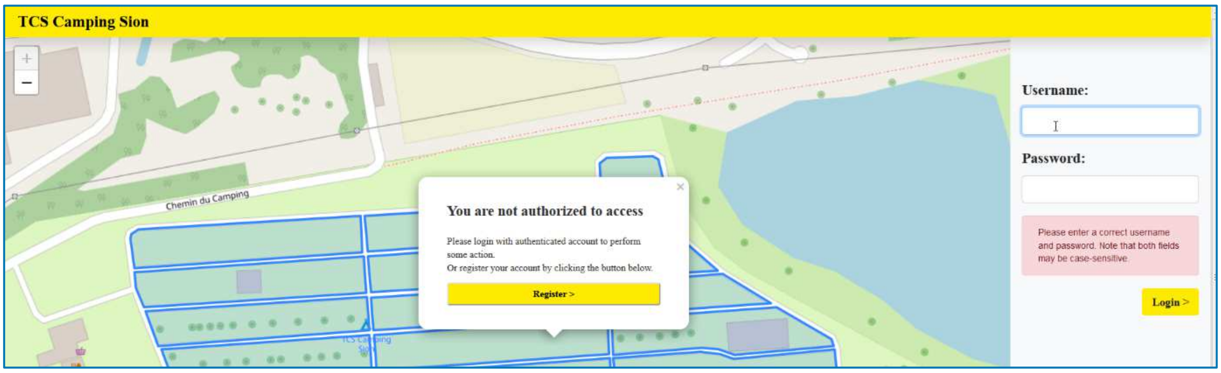
Finally, we can find the campers registered on the application in our database.



	id [PK] integer	adults integer	kids integer	pets boolean	user_id integer
1	1	3	1	true	10

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.



TCS Camping Sion

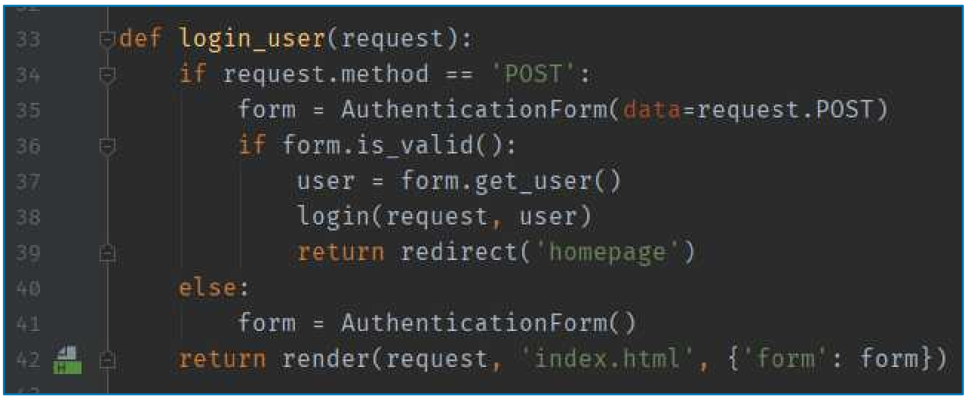
You are not authorized to access

Please login with authenticated account to perform some action.
Or register your account by clicking the button below.

Username:

Password:

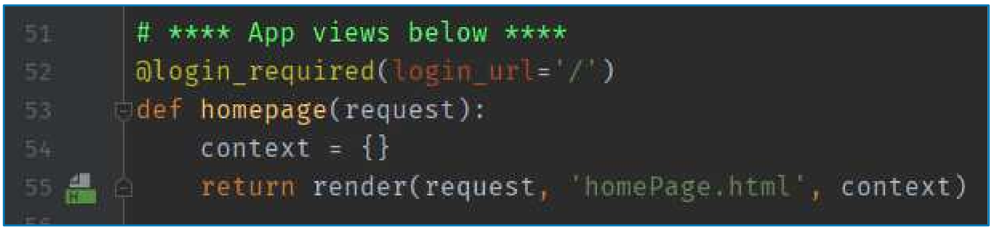
Please enter a correct username and password. Note that both fields may be case-sensitive.



```python
def login_user(request):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('homepage')
    else:
        form = AuthenticationForm()
    return render(request, 'index.html', {'form': form})
```

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.

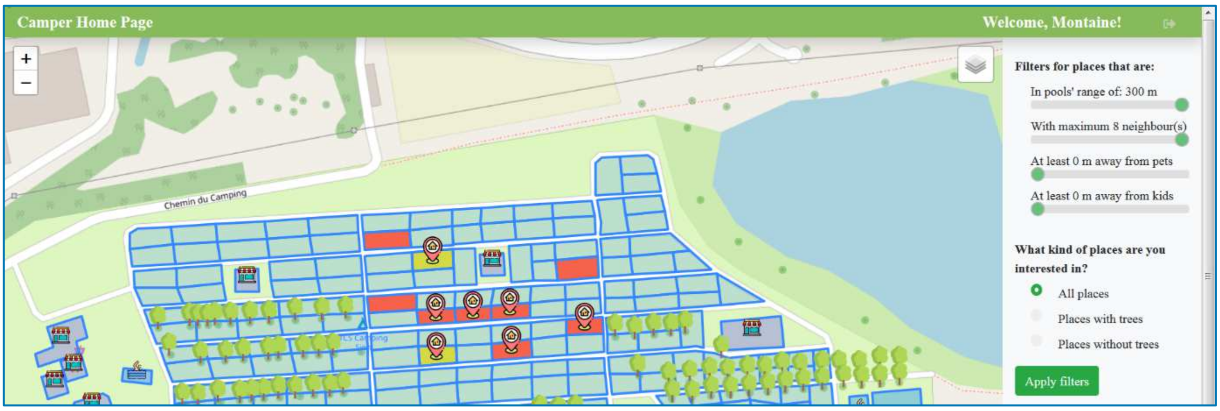


```python
# **** App views below ****
@login_required(login_url='/')
def homepage(request):
    context = {}
    return render(request, 'homePage.html', context)
```

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.



Filters for places that are:

In pools' range of: 300 m

With maximum 8 neighbour(s)

At least 0 m away from pets

At least 0 m away from kids

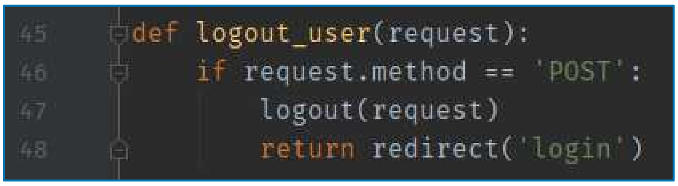
What kind of places are you interested in?

● All places

○ Places with trees

○ Places without trees

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.



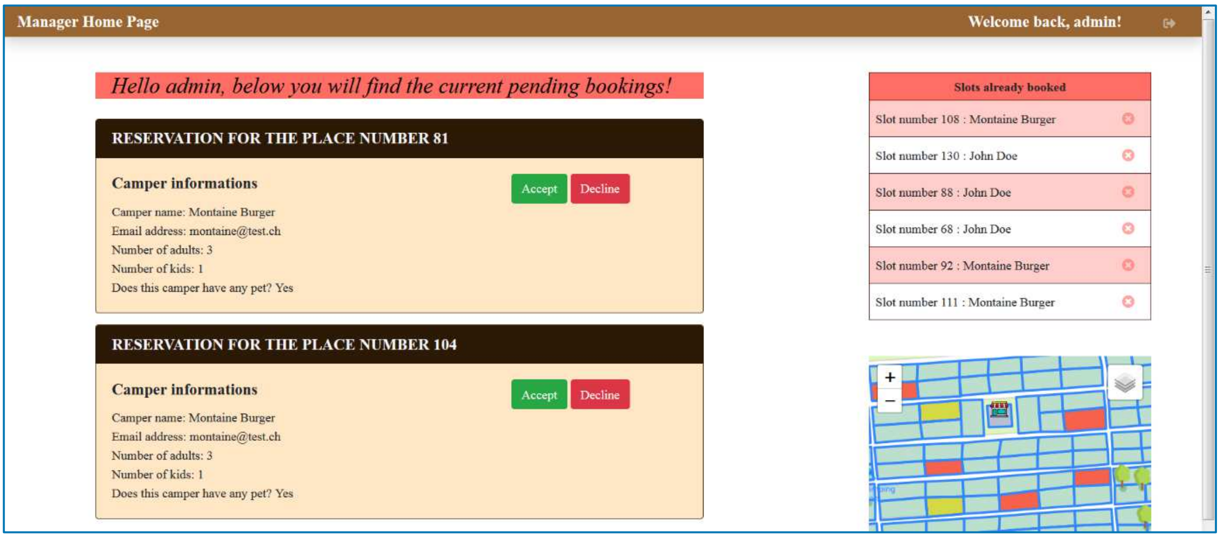
```python
def logout_user(request):
    if request.method == 'POST':
        logout(request)
        return redirect('login')
```

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.



Manager Home Page

Welcome back, admin!

Hello admin, below you will find the current pending bookings!

RESERVATION FOR THE PLACE NUMBER 81

Camper informations

Camper name: Montaine Burger
Email address: montaine@test.ch
Number of adults: 3
Number of kids: 1
Does this camper have any pet? Yes

RESERVATION FOR THE PLACE NUMBER 104

Camper informations

Camper name: Montaine Burger
Email address: montaine@test.ch
Number of adults: 3
Number of kids: 1
Does this camper have any pet? Yes

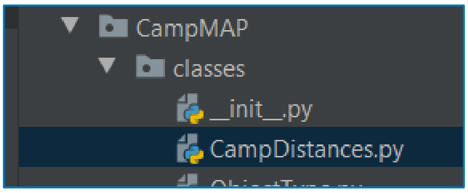
Slots already booked

Slot number 108 : Montaine Burger
Slot number 130 : John Doe
Slot number 88 : John Doe
Slot number 68 : John Doe
Slot number 92 : Montaine Burger
Slot number 111 : Montaine Burger

GEOGRAPHIC INFORMATION SYSTEM PART

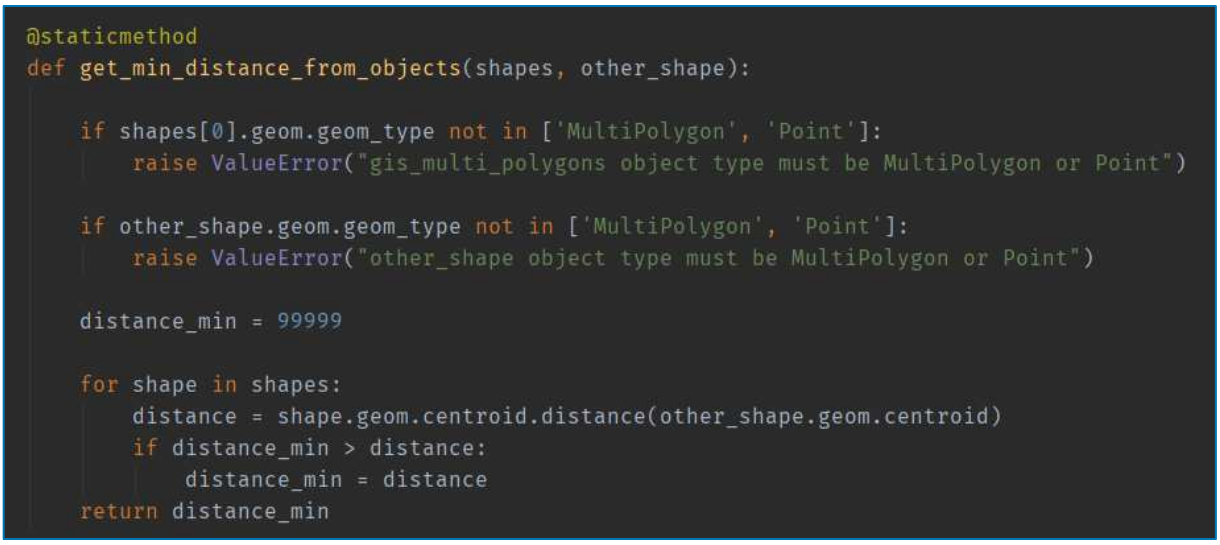
Our toolbox

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.



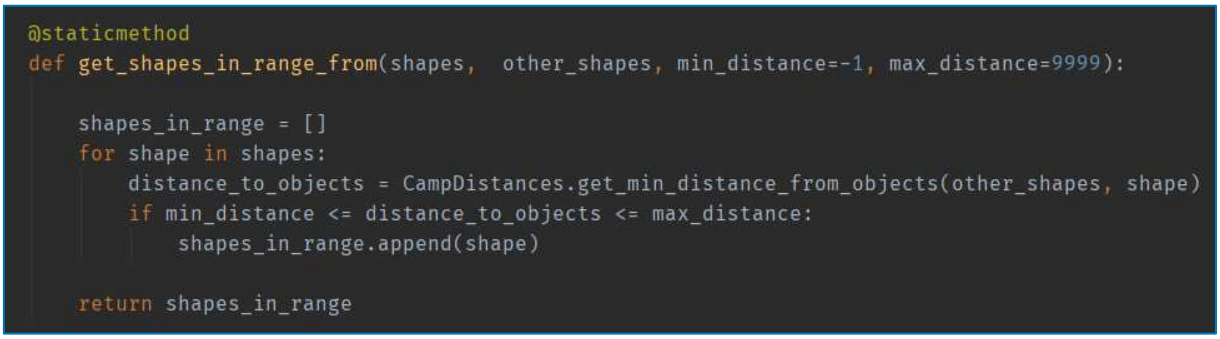
Functions used

- **get_min_distance_from_objects:** this function computes the minimal distance between two collections of shapes.
Geo use: centroid, distance() and geom_type.



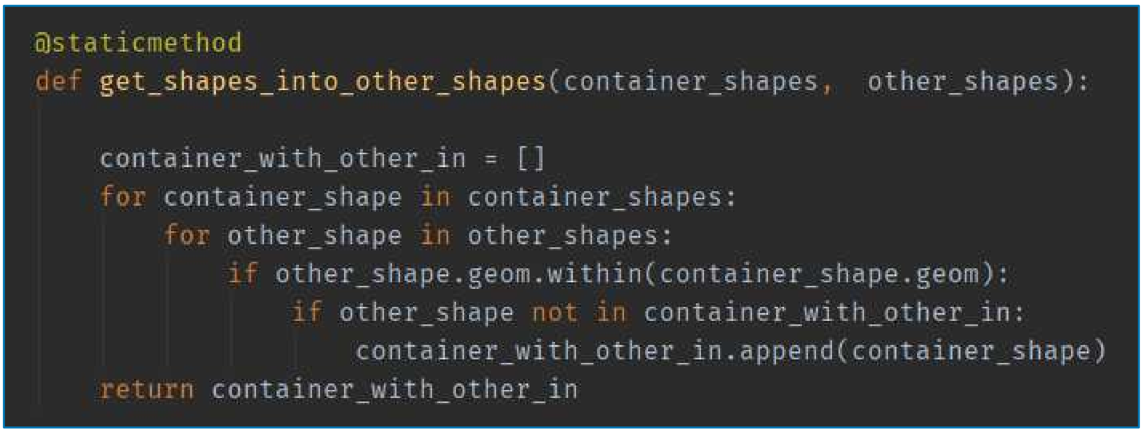
```python
@staticmethod
def get_min_distance_from_objects(shapes, other_shape):

    if shapes[0].geom.geom_type not in ['MultiPolygon', 'Point']:
        raise ValueError("gis_multi_polygons object type must be MultiPolygon or Point")

    if other_shape.geom.geom_type not in ['MultiPolygon', 'Point']:
        raise ValueError("other_shape object type must be MultiPolygon or Point")

    distance_min = 99999

    for shape in shapes:
        distance = shape.geom.centroid.distance(other_shape.geom.centroid)
        if distance_min > distance:
            distance_min = distance
    return distance_min
```

- **get_shapes_in_range_from:** this function retrieves all the shapes in range (min – max) from other shapes collection.



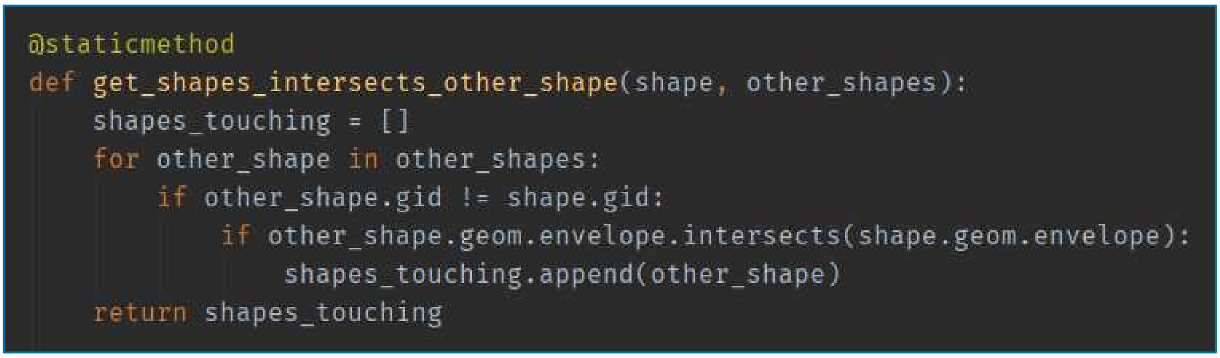
```python
@staticmethod
def get_shapes_in_range_from(shapes,  other_shapes, min_distance=-1, max_distance=9999):

    shapes_in_range = []
    for shape in shapes:
        distance_to_objects = CampDistances.get_min_distance_from_objects(other_shapes, shape)
        if min_distance <= distance_to_objects <= max_distance:
            shapes_in_range.append(shape)

    return shapes_in_range
```

• **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.



```
@staticmethod
def get_shapes_into_other_shapes(container_shapes,  other_shapes):

    container_with_other_in = []
    for container_shape in container_shapes:
        for other_shape in other_shapes:
            if other_shape.geom.within(container_shape.geom):
                if other_shape not in container_with_other_in:
                    container_with_other_in.append(container_shape)
    return container_with_other_in
```

• **get_shapes_intersects_other_shape**: this function checks if a shape intersects with a shape of another collection.
Geo use: envelope, intersects.

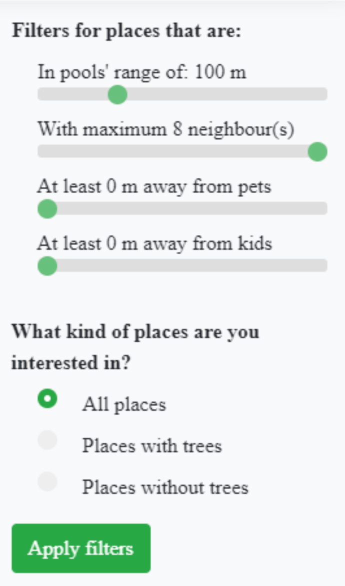


```
@staticmethod
def get_shapes_intersects_other_shape(shape, other_shapes):
    shapes_touching = []
    for other_shape in other_shapes:
        if other_shape.gid != shape.gid:
            if other_shape.geom.envelope.intersects(shape.geom.envelope):
                shapes_touching.append(other_shape)
    return shapes_touching
```

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



Filters for places that are:

In pools' range of: 100 m

With maximum 8 neighbour(s)

At least 0 m away from pets

At least 0 m away from kids

What kind of places are you interested in?

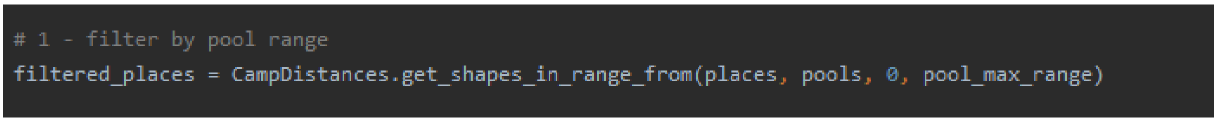
● All places

○ Places with trees

○ Places without trees

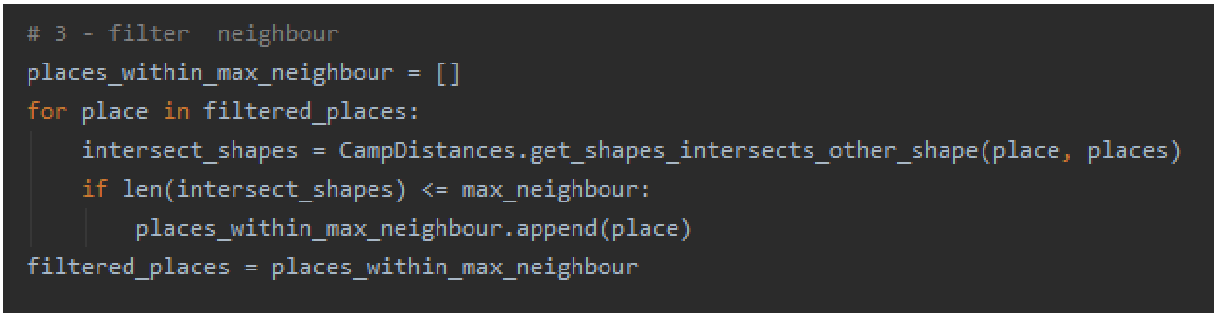
Explanation

- **Pool filter:** we used the ***get_shapes_in_range_from*** explained above.



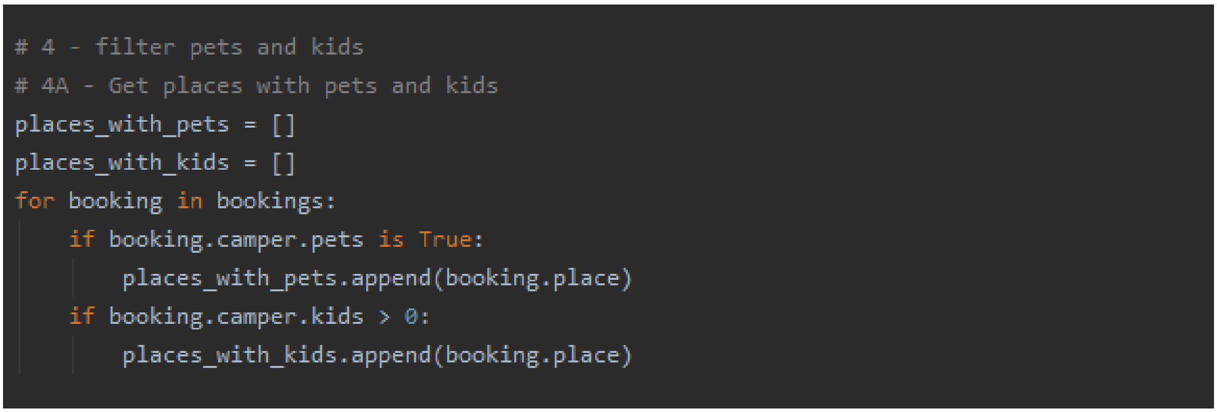
```
# 1 - filter by pool range
filtered_places = CampDistances.get_shapes_in_range_from(places, pools, 0, pool_max_range)
```

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

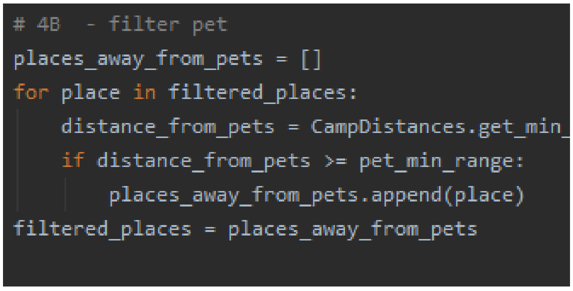


```
# 3 - filter  neighbour
places_within_max_neighbour = []
for place in filtered_places:
    intersect_shapes = CampDistances.get_shapes_intersects_other_shape(place, places)
    if len(intersect_shapes) <= max_neighbour:
        places_within_max_neighbour.append(place)
filtered_places = places_within_max_neighbour
```

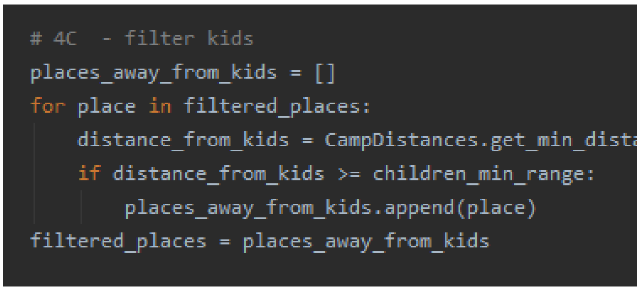
- **Pet/Children filter:** We first get places booked by campers with pets/children, then filter for places that are X meters away from those places.



```
# 4 - filter pets and kids
# 4A - Get places with pets and kids
places_with_pets = []
places_with_kids = []
for booking in bookings:
    if booking.camper.pets is True:
        places_with_pets.append(booking.place)
    if booking.camper.kids > 0:
        places_with_kids.append(booking.place)
```

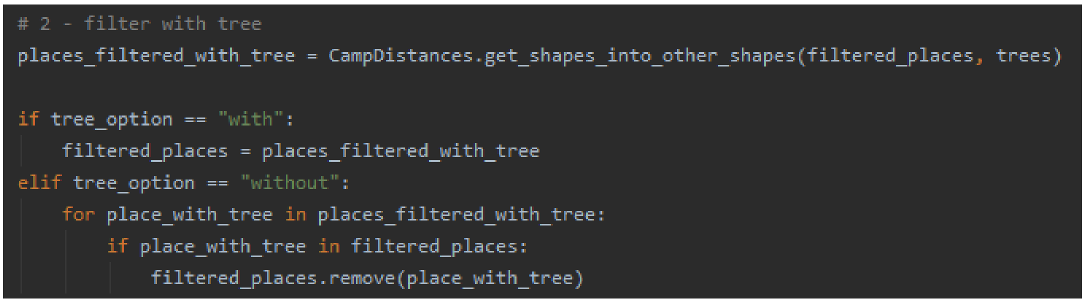


```
# 4B  - filter pet
places_away_from_pets = []
for place in filtered_places:
    distance_from_pets = CampDistances.get_min_
    if distance_from_pets >= pet_min_range:
        places_away_from_pets.append(place)
filtered_places = places_away_from_pets
```



```
# 4C  - filter kids
places_away_from_kids = []
for place in filtered_places:
    distance_from_kids = CampDistances.get_min_dista
    if distance_from_kids >= children_min_range:
        places_away_from_kids.append(place)
filtered_places = places_away_from_kids
```

- **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.



```python
# 2 - filter with tree
places_filtered_with_tree = CampDistances.get_shapes_into_other_shapes(filtered_places, trees)

if tree_option == "with":
    filtered_places = places_filtered_with_tree
elif tree_option == "without":
    for place_with_tree in places_filtered_with_tree:
        if place_with_tree in filtered_places:
            filtered_places.remove(place_with_tree)
```

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