



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

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

Table of contents

[How to setup your project locally 3](#_Toc41061708)

[Pre-requirements 3](#_Toc41061709)

[Setup local DB in pgAdmin 3](#_Toc41061710)

[Import camping shapes 5](#_Toc41061711)

[Import environment 5](#_Toc41061712)

[Setup PyCharm 5](#_Toc41061713)

[WARNING: Windows users 7](#_Toc41061714)

[Design of our project 8](#_Toc41061715)

[Mockups 8](#_Toc41061716)

[Product Backlog 8](#_Toc41061717)

[User authentication part 9](#_Toc41061718)

[Integrated module inside Django 9](#_Toc41061719)

[Custom registration form 10](#_Toc41061720)

[Other parts…? 11](#_Toc41061721)

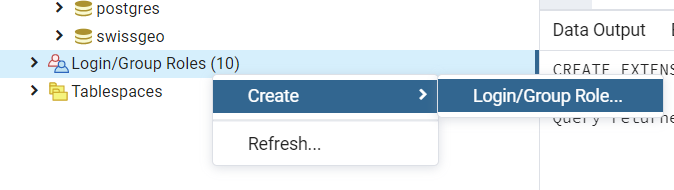
# How to setup your project locally

## Pre-requirements

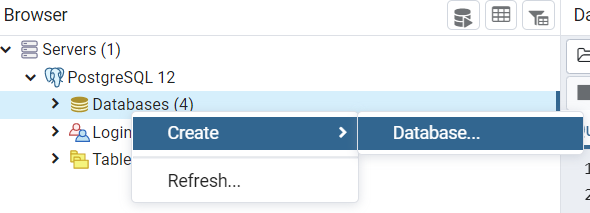
* [PostgreSQL V12](https://www.postgresql.org/download/windows/) installed with PostGIS and pgAdmin modules
* Clone Github Project: <https://github.com/PierreAnken/HEVS_S8_GIS_Camping.git>

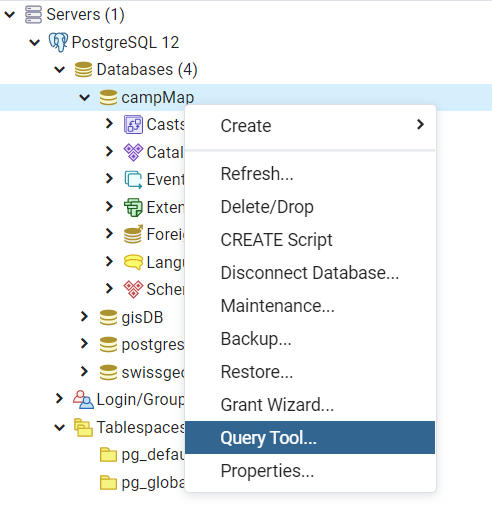
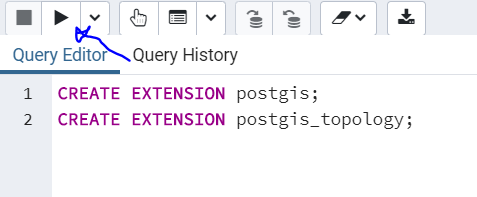
## Setup local DB in pgAdmin

First, create a new admin user:

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

Then, create a new database:

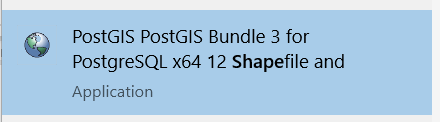
* Database: campMap
* Owner: admin

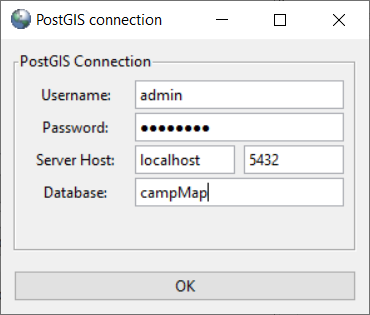
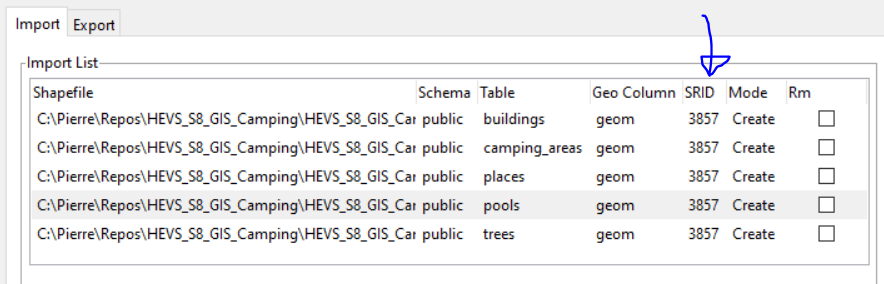
You have to enable PostGIS in your new database:

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

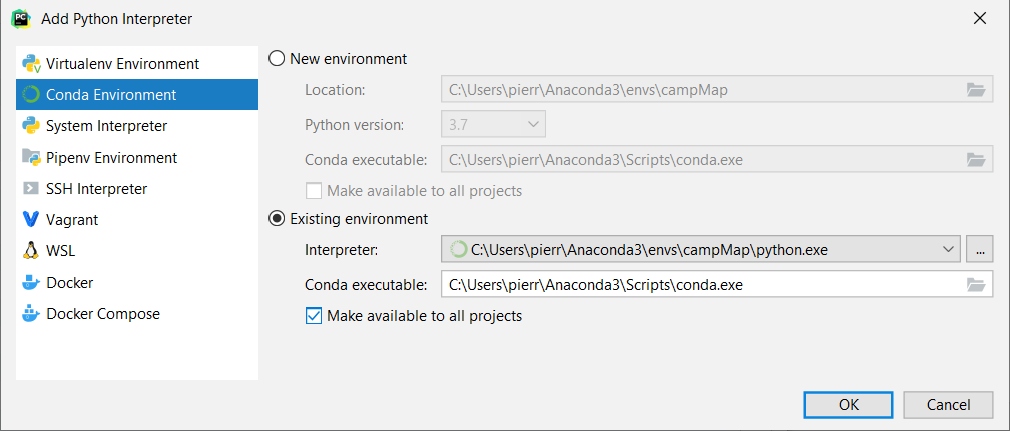
Open Anaconda Navigator interface and import env. from the file in GitHub, under:  
*HEVS\_S8\_GIS\_Camping \Documentation\spec-list.txt*

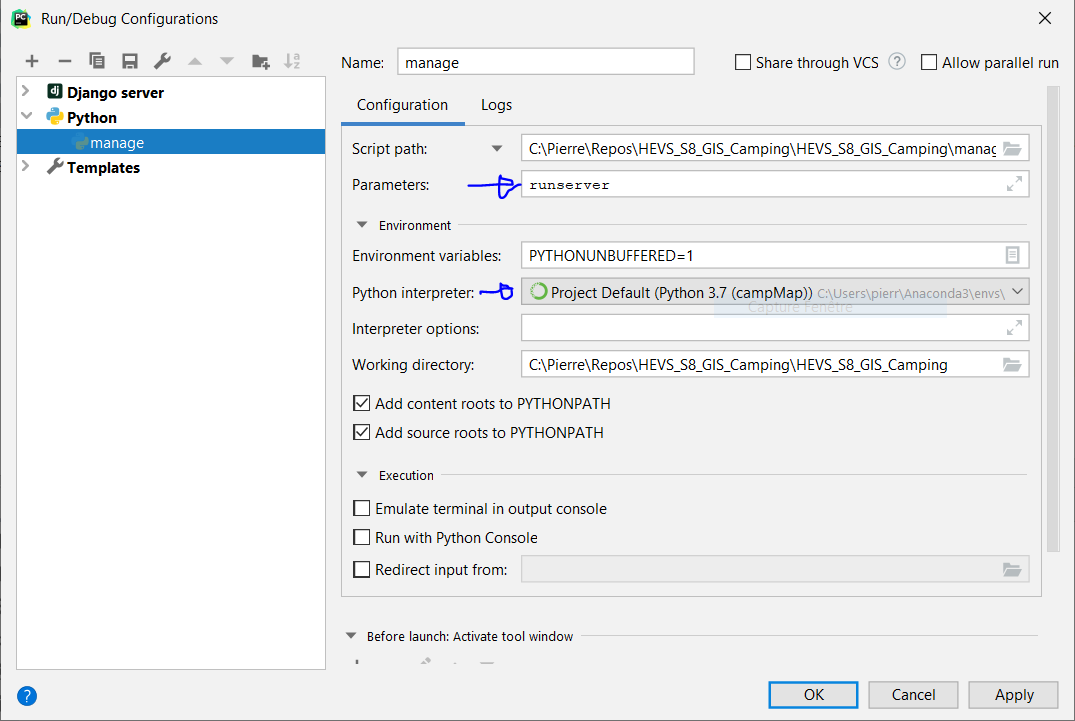
## Setup PyCharm

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%**"

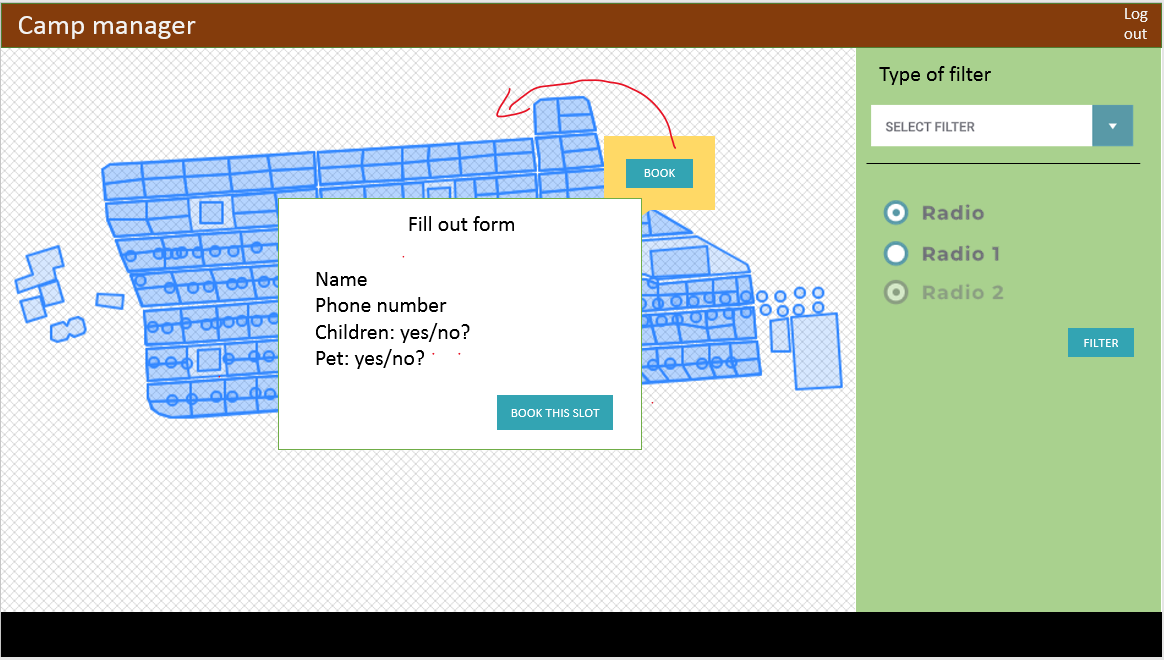
reg ADD "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /v GDAL\_DATA /t REG\_EXPAND\_SZ /f /d "**%GDAL\_DATA%**"

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

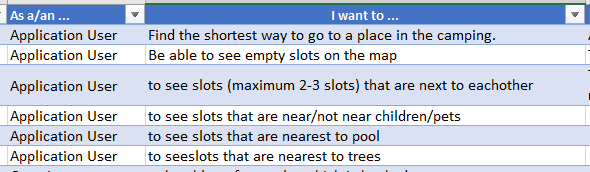
# Design of our project

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*

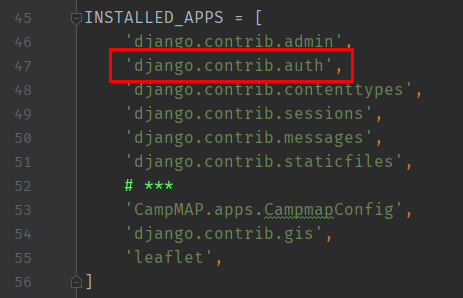
# User authentication part

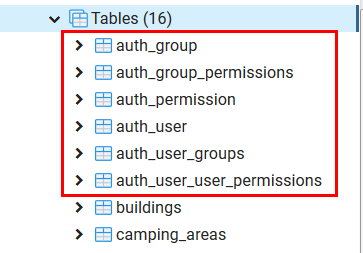
Our wish was to manage users (especially camper and manager roles). The manager can be created from the admin console, since it is the superuser proposed by Django. We have already created the default superuser: **name** **>** admin and **password >** adminPWD.

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

## 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:** when it’s done, it’s important to change one property inside the *auth\_user* table. To perform this, you can right click on the table > Properties. Then, under *Columns* tab, just change *last\_login* field to ***Not NULL = No*** (otherwise it can cause some bugs).

Note:

If manage.py migrate does not add the appropriate models to the database, it’s also possible to manually create the tables using pgAdmin. The scripts for the tables are:

For Camper model:

CREATE TABLE public."CampMAP\_camper"

(

id SERIAL,

adults integer NOT NULL,

kids integer NOT NULL,

pets boolean NOT NULL,

user\_id integer NOT NULL,

CONSTRAINT "CampMAP\_camper\_pkey" PRIMARY KEY (id),

CONSTRAINT "CampMAP\_camper\_user\_id\_key" UNIQUE (user\_id),

CONSTRAINT "CampMAP\_camper\_user\_id\_d3c4413b\_fk\_auth\_user\_id" FOREIGN KEY (user\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "CampMAP\_camper\_adults\_check" CHECK (adults >= 0),

CONSTRAINT "CampMAP\_camper\_kids\_check" CHECK (kids >= 0)

)

TABLESPACE pg\_default;

ALTER TABLE public."CampMAP\_camper"

OWNER to admin;

For Reservation model:

CREATE TABLE public."CampMAP\_reservation"

(

id SERIAL,

status integer NOT NULL,

camper\_id integer NOT NULL,

place\_id integer NOT NULL,

CONSTRAINT "CampMAP\_reservation\_pkey" PRIMARY KEY (id),

CONSTRAINT "CampMAP\_reservation\_camper\_id\_key" UNIQUE (camper\_id),

CONSTRAINT "CampMAP\_reservation\_place\_id\_key" UNIQUE (place\_id),

CONSTRAINT "CampMAP\_reservation\_camper\_id\_88e6ba23\_fk\_auth\_user\_id" FOREIGN KEY (camper\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "CampMAP\_reservation\_place\_id\_02e8f4f1\_fk\_places\_gid" FOREIGN KEY (place\_id)

REFERENCES public.places (gid) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "CampMAP\_reservation\_status\_check" CHECK (status >= 0)

)

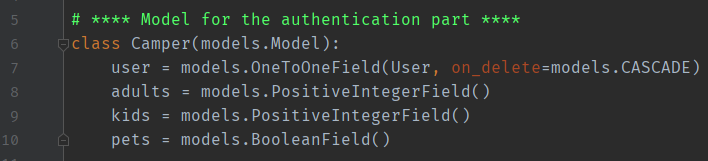
TABLESPACE pg\_default;

ALTER TABLE public."CampMAP\_reservation"

OWNER to admin;

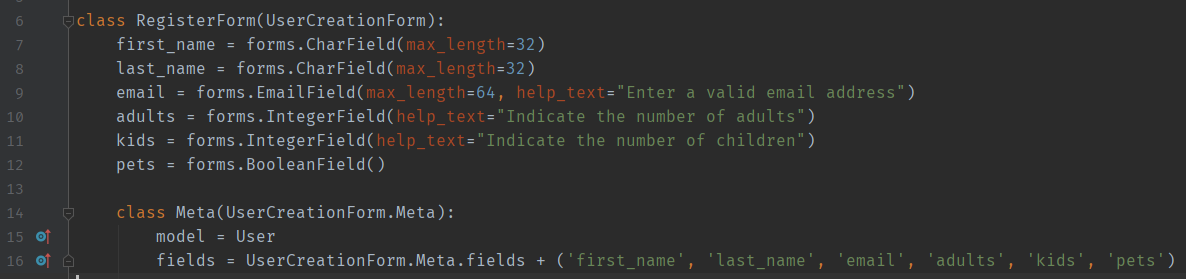
## Custom user model

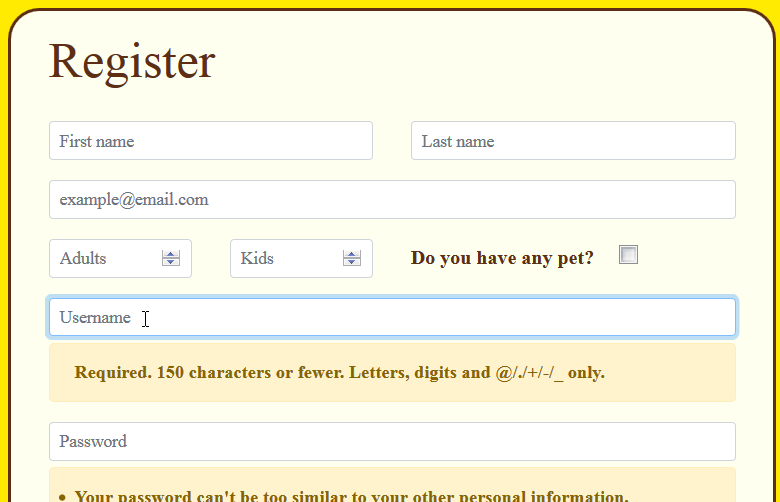
As shown in our database diagram, the *Camper* inherits from the base Django *User*. This allows us to use all the basic authentication features through the *User*, but also to add the fields that we want to save for our *Camper*.

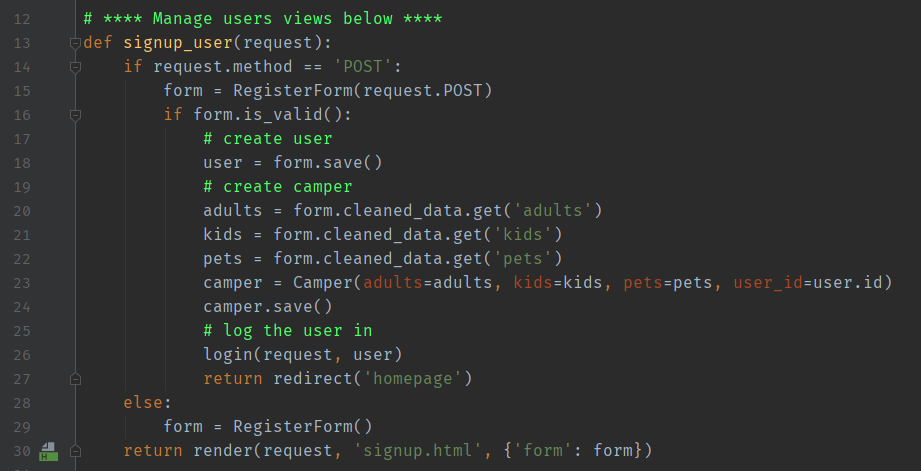
To make this configuration work, we just had to add a one to one relationship:

To retrieve this model in the database, simply migrate it with the usual commands. However, it is important to remove the *NOT NULL* constraint on the *name* column of the *django\_content\_type* table before running them (otherwise it will cause a bug).

## Custom registration form

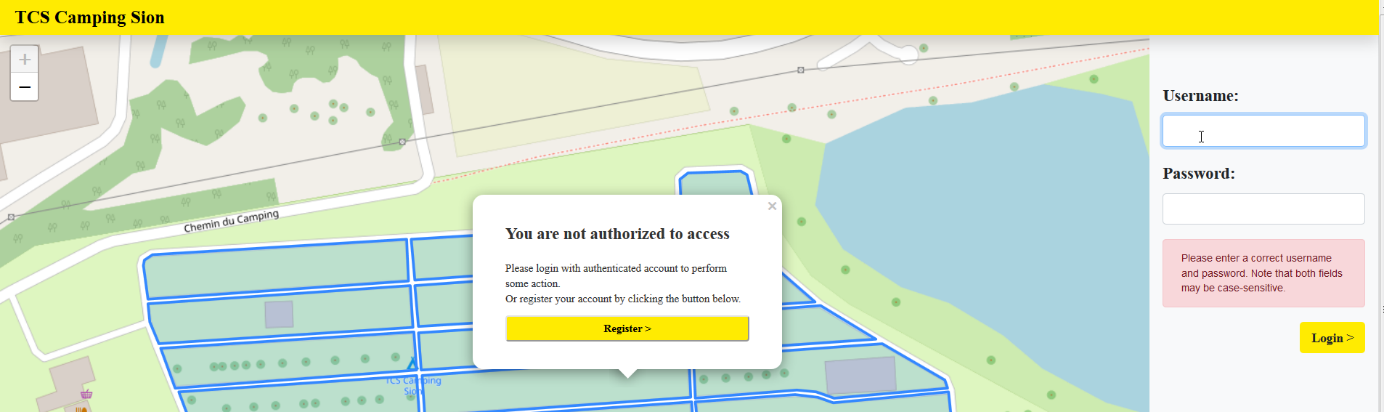
In order to be able to save these new added fields, it was also necessary to rewrite the basic registration form. The goal was to use the basic form, with its functionalities, but with the addition of our custom fields.

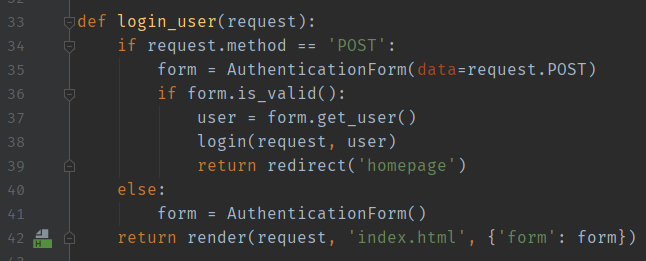
Then we were able to create the registration form on the front end and save the new registrations by creating linked user and camper.



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

## Login on the application

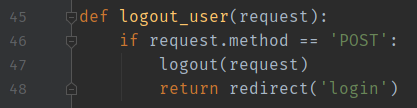
Obviously, our custom users, i.e. the campers, can identify themselves on the application by logging in from the principal page.



## 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.

**TODO: screenshots of the camper/manager pages (when finished)**

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

# Other parts…?