# Carnegie Mellon University in Qatar
# 15415 - Spring 2020
# Recitation 8 - Django Part II

## Prerequisites:

In order to complete this lab, you should have the following installed:

1. PostgreSQL 10
2. Python 3.x; with psycopg2 package installed. This is PostgreSQL package for Python.
3. PyCharm Professional (not Community/Edu)

## Project Setup

### Create a new project

We will create a new project where we will create our apps (Recall the difference between a project and an app from the last Recitation)

a. Create a new project…
b. Choose Django and keep the defaults
c. Set the project name (will be the folder name)

### Setup the dependencies

1. Open the console/terminal from PyCharm and run:

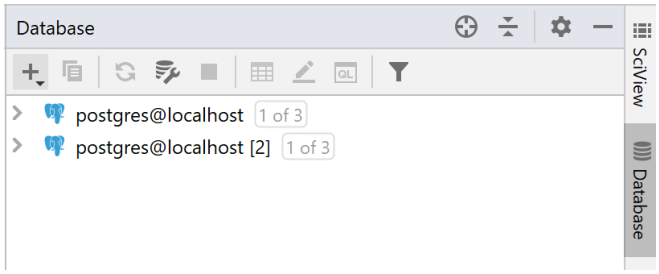
```
$pip freeze > requirements.txt
```

A side note: when using pip freeze, pip reads the versions of all installed packages and then produces a text file (in this case requirements.txt) with the package version for each python package specified. This makes your work environment portable in the future!

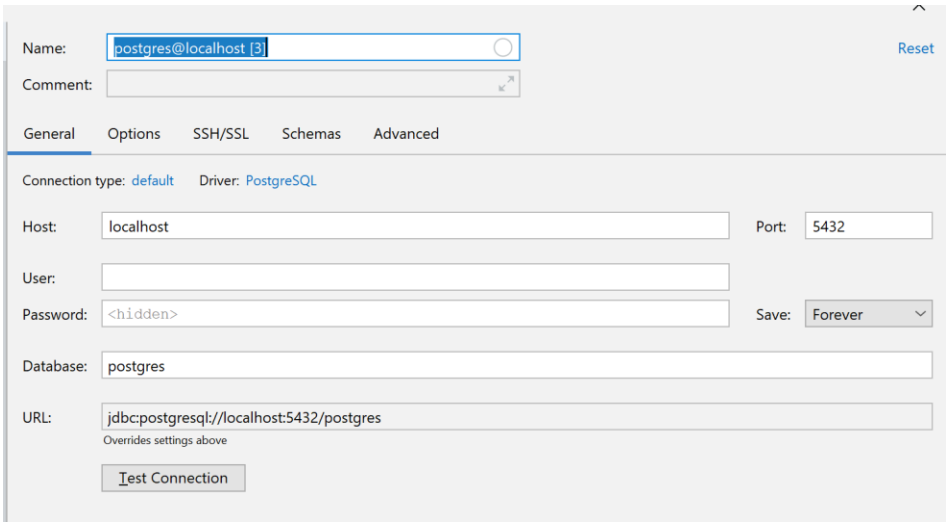
2. Append psycopg2 to "requirements.txt" and then PyCharm shows a command to install the newly added packages.

### Connecting the Database

1. In the PyCharm Database panel on the right, click on the "+" and choose "Data Source" then choose "PostgreSQL":



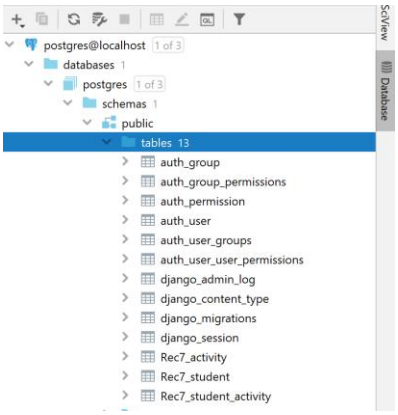

2. Next, we create and test the connection to our database, as in the below diagram:




3. Name your database (i.e., mydb) , add the username and password and test the connection. If everything is correct, the connection test should pass!
4. In **settings.py** change the databases structure as:

```
DATABASES = {
  'default': {
     'ENGINE': 'django.db.backends.
              postgresql_psycopg2',
     'NAME': 'mydb', <your db name>
     'USER': 'postgres',
     'PASSWORD': '***',
     'HOST': 'localhost',
     'PORT': ''
  }
}
```

5. Now you need to run the migrations (propagating the changes to the database), open the manage.py shell from Tools > Run manage.py task) and run “migrate”.
6. Now check in PyCharm DB panel if the “mydb” tables have been created. You can find this in schema **public** (you might have different tables):




## Setting up the admin console

1. Go over urls.py
2. PyCharm has a built-in web server, you can run your server using (Shift + F10) or from the top right panel:

3. Go to http://localhost:8000/. Does it work?
4. Check the admin console: http://localhost:8000/admin/
5. Now, we will create a superuser with password. Open the manage.py console and run the command “createsuperuser” and you will be asked to provide a password:

```
manage.py > createsuperuser
```

6. If the user is created, you can check in the PyCharm DB panel in the **auth_user** table. You can navigate through this table and see the created table from PyCharm UI.
7. Now you can login to the admin console using this username and password.

## Creating a new app

1. Create a new Django app from the manage.py console and name it myapp:

```
manage.py > startapp myapp
```

2. In **settings.py**, append **myapp** to **INSTALLED_APPS:**

```
'myapp.apps.MyappConfig'
```

## Creating the profile page

Now we will create three functionalities of the profile page: Signup, Login and Logout.

### The profile page

1. First, we want our users to land to the profile page as soon as they login to our website. We need to specify the path in urls.py and link it with the profile view, which we will define next in views.py:
   a) In urls.py, import myapp views:

```
from myapp import views as myapp_views
```

   b) Append the profile path to **urlpatterns**:

```
path('', views.profile, name="profile")
```

2. In views.py, write the function view profile. This function profile, takes a request and renders a response with the "profile.html" file that we will define next.

```
def profile(request):
    return render(request, 'profile.html')
```

3. In the templates' directory (it is empty now because we haven't created any templates yet), create a new HTML file and add the following in the body of the HTML:

```
<h2> Profile </h2>
<p> Welcome, {{ user.username }}! </p>
<p> Your email address: {{ user.email }} </p>
```

   **Note:** these {{}} are called handlebars. They are variables that are loaded from the context of the program (we will learn more about them later).

4. Now if you go to http://localhost:8000/, what do you see?
5. Logout from the user admin (from the admin console) and go to http://localhost:8000/ again. What do you see?
6. We need to only show the profile page, if we are logged in. We can achieve this by annotating the profile view with @login_required. You might need to quick import the missing imports using (Alt + Enter)

```
@login_required
def profile(request):
    ...
```

7. Go to http://localhost:8000/ again. What do you get?

## Login Page

1. In urls.py import auth_views:

```
from django.contrib.auth import views as auth_views
```

2. In urls.py, append to urlpatterns:

```
path('login/',
    auth_views.LoginView.as_view(),
    {'template_name': 'login.html'},
    name='login')
```

3. In the directory "templates" create a new directory "registration" and create an HTML file "login.html", and add the following in the body:

```
<h2> Login </h2>
<form method="POST">
  {{ form }}
  <button type="submit"> Login </button>
</form>
```

4. Test the login page: http://localhost:8000/login/
5. Attempt to login with user **admin**
6. Add **{% csrf_token %}** to login form
   Note: CSRF stands for cross-site request forgery.
7. Attempt to login with user **admin**
8. Append to **settings.py**:

```
# User login and authentication
LOGIN_REDIRECT_URL = 'profile'
LOGIN_URL = 'login'
LOGOUT_URL = 'logout'
```

## Let's beautify our pages a bit!

1. Change {{form}} to {{form.as_p}} and rerun the page.
2. Now if we need to create a template for all pages that we will create, let's define a base.html template. Create base.html in the templates, and change its title and body, as:

```
<title>
  {% block title %} Project 2 {% endblock %}
</title>
{% block body %}
{% endblock %}
```

3. In every template we have created (profile.html and login.html), let's simplify them by adding:

```
{% extends 'base.html' %}
```

For example, in profile.html:

```
{% extends 'base.html' %}
 {% block title %} Profile {% endblock %}
 {% block body %}
   <h2> Profile </h2>
   <p> Welcome, {{ user.username }}! </p>
   <p> Your email: {{ user.email }} </p>
 {% endblock %}
```

## Logout

1. In **urls.py**, append to **urlpatterns**:

```
path('logout/', auth_views.LogoutView.as_view(),name='logout')
```

2. In templates > registration directory, create an HTML file "logged_out.html" and add this:

```
{% extends 'base.html' %}
{% block title %} Bye {% endblock %}
{% block body %}
<h2> Logged out </h2>
<p> Thank you for visiting us </p>
<a  href="{% url 'login' %}">Log in again</a>
{% endblock %}
```

3. Attempt to logout user **admin**: http://localhost:8000/logout/
4. Add to **base.html**:

```
<p> <a href="{% url 'logout' %}"> Logout </a> </p>
```

## Signup

1. In **urls.py**, append to **urlpatterns**:

```
path('signup/', myapp_views.signup,
     name='signup')
```

2. In views.py, write function view **signup**:

```
if request.method == 'POST':
  form = UserCreationForm(request.POST)
else:
  form = UserCreationForm()
context = {'form': form}
return render(request, 'signup.html', context)
```

*When will it be a post method? And when will it be the else case?*

3. Define the signup.html template (you can use the extends method or the normal method)

```
<h2> Sign Up </h2>
 <form method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit"> Sign Up </button>
 </form>
```

4. Append to **login.html**:

```
<p> <a href="{% url 'signup' %}">Sign Up </a> </p>
```

5. Attempt to sign up a new user: http://localhost:8000/.
6. Check auth_user table for a new record for the user you have just created. Can you find the user?

## Validating Forms before Committing to the DB

At this point, we have the sign-up form, but we can sign up a user with invalid information. We will need to implement form validation to avoid this. In the sign up view, validate the sign up form, save the validated user and redirect to login view:

```
if form.is_valid():
  form.save()
  print('User created')
  return redirect('login')
else:
  print('Validation failed')
```