Django Authentication

Estimated time needed: 30 minutes

In this lab, you will be understanding about django authentication.

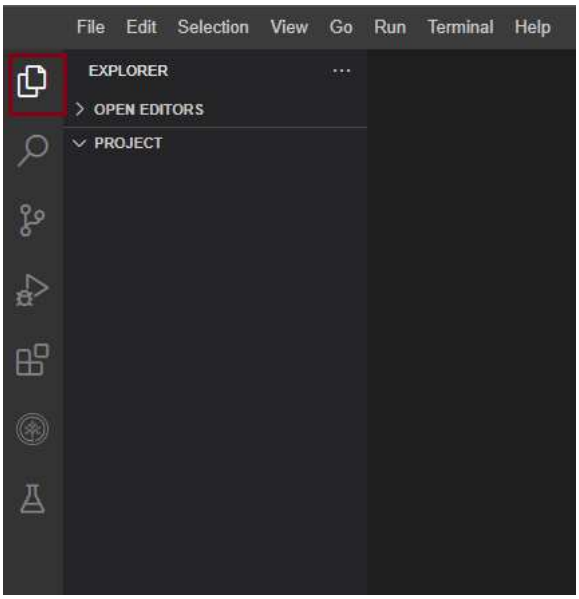
Learning Objectives

- Understand the Django authentication system
- Create views and templates for user log in and log out
- Create views and templates for user registration

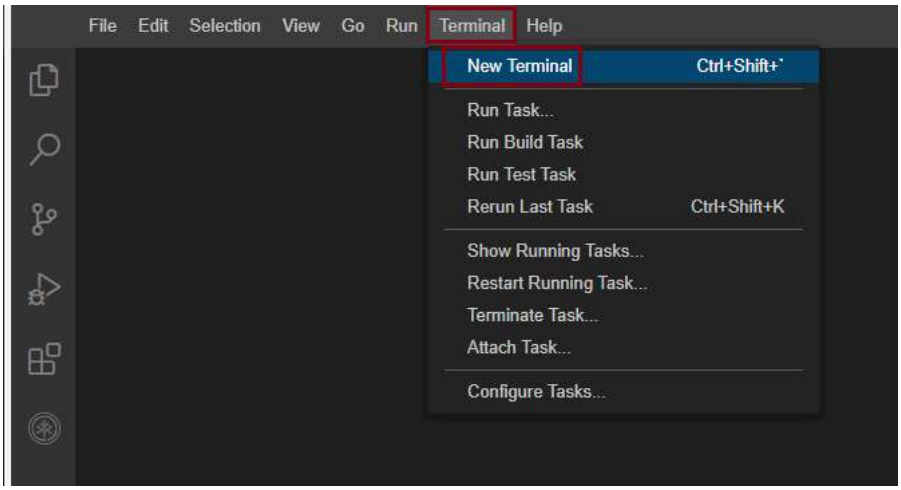
Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your project, in Cloud IDE.

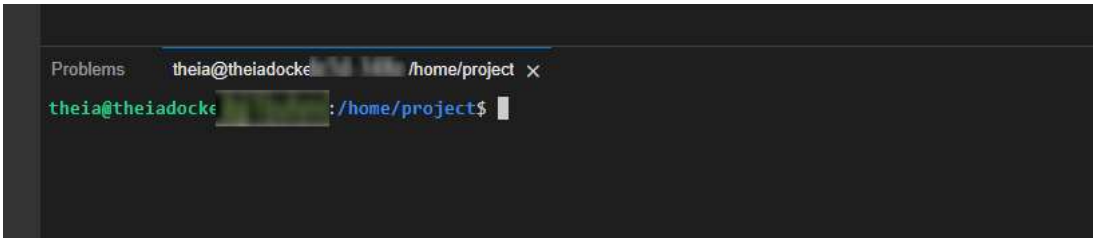
To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



Click on `New`, and then `New Terminal`.



This will open a new terminal where you can run your commands.



Concepts covered in the lab

1. `Authentication`: The process of verifying the identity of a user or system.
2. `User`: Represents an individual who interacts with the Django application. It typically includes information such as username, password, and email.
3. `Registration`: The process of creating a new user account in the application.
4. `Login`: The process by which a user provides credentials (such as username and password) to access a protected area of the application.
5. `Logout`: The process of ending a user's session and removing their authentication status.

Import an onlinecourse App Template and Database

If the terminal was not open, go to `Terminal > New Terminal` and make sure your current Theia directory is `/home/project`.

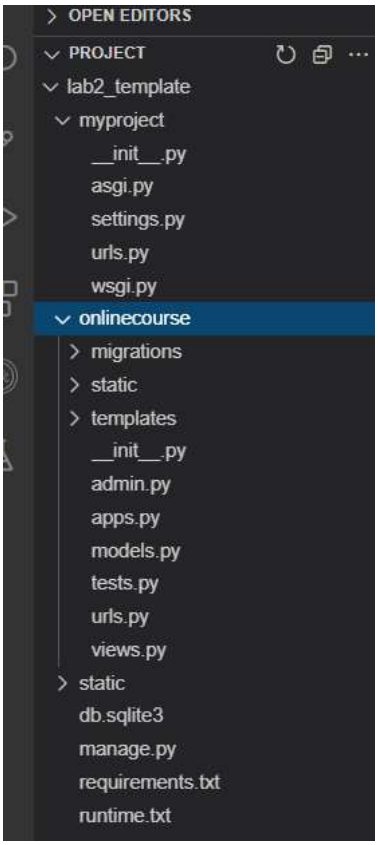
- Run the following command-lines to download a code template for this lab

```
1
2
3
```

```
wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-SkillsNetwork/labs/m5_django_advanced/lab2_template.zip"
unzip lab2_template.zip
rm lab2_template.zip
```

Copied! Executed!

Your Django project should look like the following:



- `cd` to the project folder:

```
1
```

```
cd lab2_template
```

Copied! Executed!

- Next, install required packages for this lab:

```
1
2
3
4
5
6
```

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
pip install -U -r requirements.txt
```

Copied! Executed!

Next activate the models for an `onlinecourse` app.

- Perform migrations to create necessary tables:

```
1
```

```
python3 manage.py makemigrations
```

Copied! Executed!

- and run migration to activate models for `onlinecourse` app.

```
1
```

```
python3 manage.py migrate
```

Copied! Executed!

Now let's test the imported `onlinecourse` app.

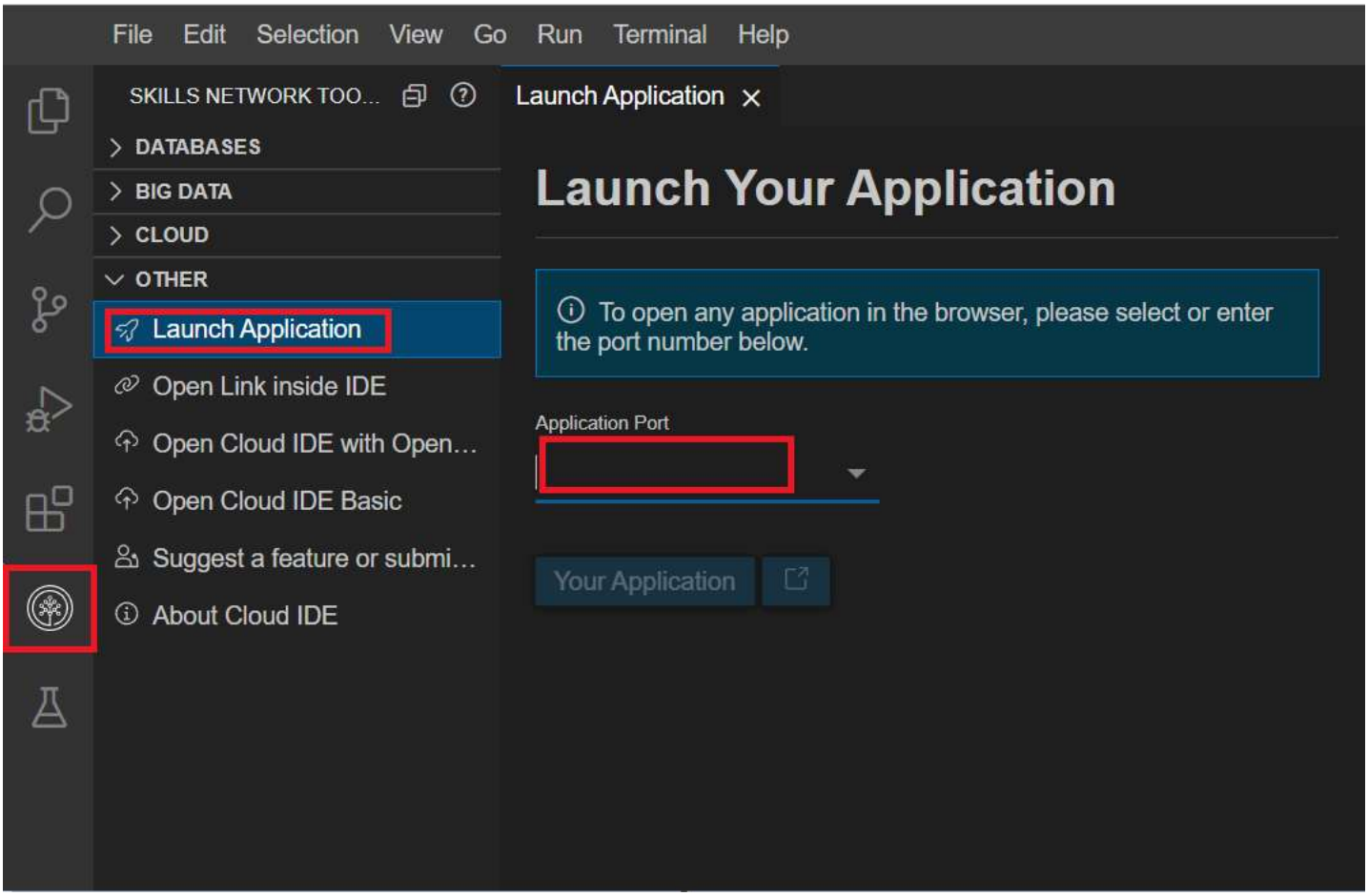
- Start the development server

```
1
```

```
python3 manage.py runserver
```

Copied! Executed!

- Click on the Skills Network button on the left, it will open the **"Skills Network Toolbox"**. Then click the **Other** then **Launch Application**. From there you should be able to enter the port `8000` and launch.



When the browser tab opens, add the `/onlinecourse` path and your full URL should look like the following:

`https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/onlinecourse`

Now you should see the `onlinecourse` app started with a list of courses as the main page.

Display User Information

Django authentication system is a Django built-in app used for managing user authentication and authorization.

Let's start learning Django authentication by quickly creating a superuser and displaying its user information on the main page.

- Stop the server if started and run:

```
1
```

```
python3 manage.py createsuperuser
```

Copied!

With `Username`, `Email`, `Password` entered, you should see a message indicates the super user is created:

```
1
```

```
Superuser created successfully.
```

Copied!

Let's start the server again and login with the super user.

```
1
```

```
python3 manage.py runserver
```

Copied!

- Click `Launch Application` and enter the port for the development server `8000`

After the browser tab opens, add the `/admin` path and your full URL should look like the following

`https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/admin`

- Log in admin site with the credential you just created for the super user.

- Click the `Users` under `AUTHENTICATION AND AUTHORIZATION` section, and find the superuser you just created
- Try to add some profile information such as `First name`, `Last name` and so on which will be shown on the main page.
- Scroll further down and click `Save` to store the profile information.

Next, let's try to retrieve the superuser and show its profile on `onlinecourse/course_list.html` template

- Open `onlinecourse/templates/onlinecourse/course_list.html`, add the following code snippet under comment `<!--Authentication section-->`

```
1
2
3
```

```
{% if user.is_authenticated %}
<p>Username: {{user.username}}, First name: {{user.first_name}}, Last name: {{user.last_name}} </p>
{% endif %}
```

Copied!

In the template, the `user` object will be queried by Django automatically for you based on the `session_id` created after login, and it will be available to both templates and views.

Then, we will check if the user is authenticated or if it is an anonymous user by using a if tag `{% if user.is_authenticated %}` and display the profile such as first name, last name, email, etc if authenticated.

To test, make sure the development server is running and go to:

`https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/onlinecourse`.

You should see the user information on the top.

Login and Logout

Once you log in the superuser, your browser will keep `session_id` in cookie so that the superuser remaining logged in until the session expired.

Next, let's try to logout the superuser manually from main page by adding a logout dropdown button.

- Open `onlinecourse/course_list.html`, update the code between `{% if user.is_authenticated %}` and `{% endif %}` with a dropdown `<div>`:

```
1
2
3
4
5
6
7
8
9
10
```

```
{% if user.is_authenticated %}
<div class="rightalign">
        <div class="dropdown">
            <button class="dropbtn">{{user.first_name}}</button>
            <div class="dropdown-content">
                 <a href="{% url 'onlinecourse:logout' %}">Logout</a>
            </div>
        </div>
</div>
{% endif %}
```

Copied!

The above code block adds a dropdown button to display the user's first name. When you hover the button, it pops up a link referring to a logout view.

Next, let's create the `logout_request` view.

- Open `onlinecourse/views.py`, add a function-based logout view under the comment `# Create authentication related views`:

```
1
2
3
```

```
4. 4
5. 5
6. 6
7. 7
```

```
1. def logout_request(request):
2.     # Get the user object based on session id in request
3.     print("Log out the user `{}`".format(request.user.username))
4.     # Logout user in the request
5.     logout(request)
6.     # Redirect user back to course list view
7.     return redirect('onlinecourse:popular_course_list')
```

Copied!

The above code snippet calls a built-in `logout` method with the `request` as an argument to log out the user (obtained from the request).

- Configure a route for `logout_request` view by adding a path entry in `urlpatterns`:

```
1. 1
```

```
1. path('logout/', views.logout_request, name='logout'),
```

Copied!

You can now test the logout functionality by refreshing the course list page. After you click `logout` button from the drop down, you should see the dropdown button disappeared because the user was not authenticated anymore.

Next, let's try to log in the superuser again.

- Open `templates/onlinecourse/course_list.html`, update the `{% if user.is_authenticated %}` block

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
```

```
1. {% if user.is_authenticated %}
2.     <div class="rightalign">
3.         <div class="dropdown">
4.             <button class="dropbtn">{{user.first_name}}</button>
5.             <div class="dropdown-content">
6.                 <a href="{% url 'onlinecourse:logout' %}">Logout</a>
7.             </div>
8.         </div>
9.     </div>
10.     {% else %}
11.     <div class="rightalign">
12.         <div class="dropdown">
13.             <button class="dropbtn">Visitor</button>
14.             <div class="dropdown-content">
15.                 <a href="{% url 'onlinecourse:login' %}">Login</a>
16.             </div>
17.         </div>
18.     </div>
19. {% endif %}
```

Copied!

Here, we added a `{% else %}` tag to handle the scenario when user is not authenticated and also created a new dropdown button with a link pointing to a `login` view.

The `login` view should return a common login page asking for user credential such as username and password.

Next, let's create a template for such `login` view:

- Open the `templates/onlinecourse/user_login.html`, and add a simple `form` to accept user name and password

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
```

```
13
```

```
<form action="{% url 'onlinecourse:login' %}"  method="post">
  {% csrf_token %}
  <div class="container">
    <h1>Login</h1>
    <label for="username"><b>User Name</b></label>
    <input type="text" placeholder="Enter User Name: " name="username" required>
    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password: " name="psw" required>
    <div>
      <button class="button" type="submit">Login</button>
    </div>
  </div>
</form>
```

Copied!

The key elements of this login form are two input fields for user name and password. After form submission, it sends a `POST` request to a login view.

Next, let's create a login view to handle login request.

- Open `onlinecourse/views.py`, add a `login_request` view:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
def login_request(request):
    context = {}
    # Handles POST request
    if request.method == "POST":
        # Get username and password from request.POST dictionary
        username = request.POST['username']
        password = request.POST['psw']
        # Try to check if provide credential can be authenticated
        user = authenticate(username=username, password=password)
        if user is not None:
            # If user is valid, call login method to login current user
            login(request, user)
            return redirect('onlinecourse:popular_course_list')
        else:
            # If not, return to login page again
            return render(request, 'onlinecourse/user_login.html', context)
    else:
        return render(request, 'onlinecourse/user_login.html', context)
```

Copied!

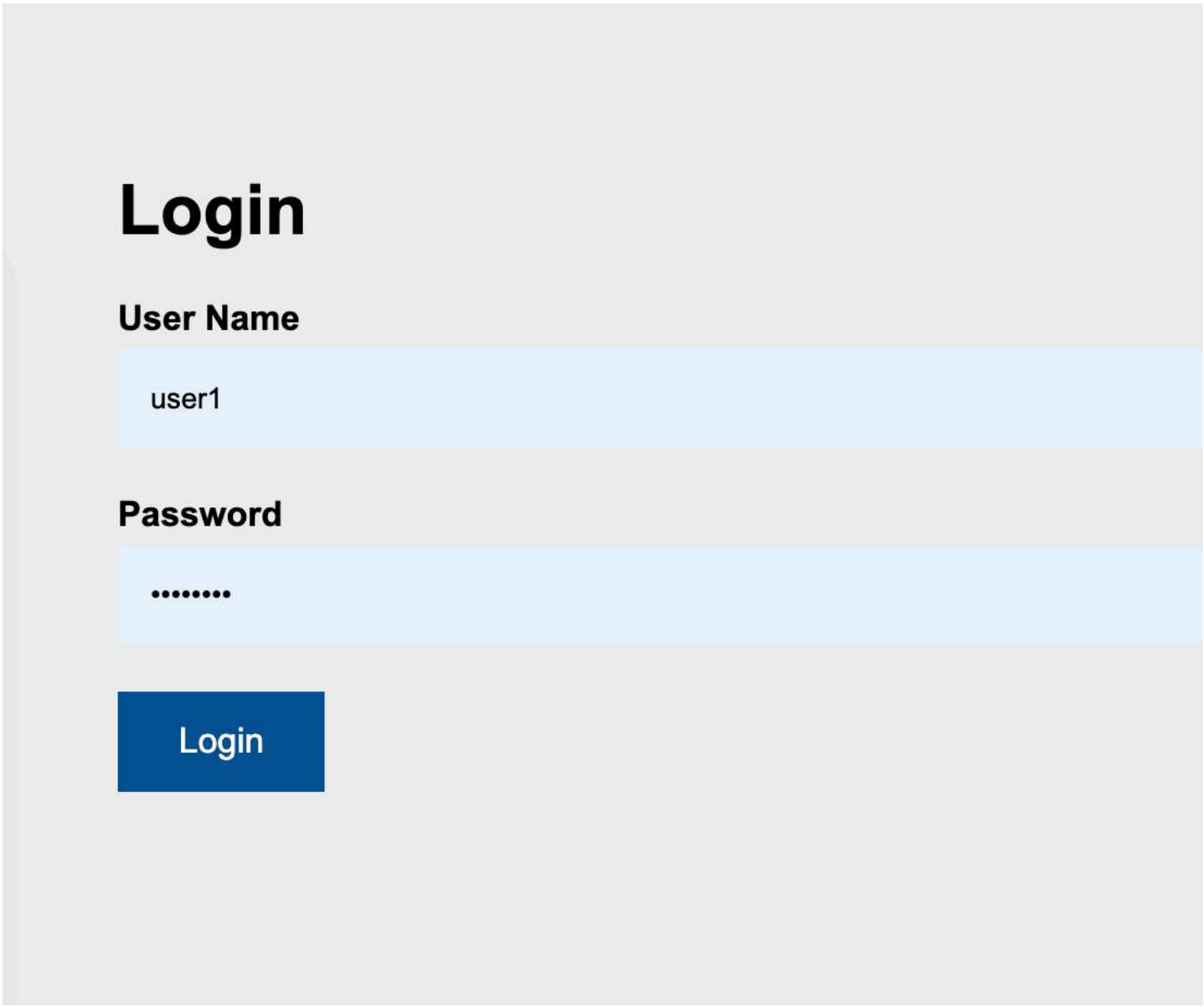
- and configure a route for the `login_request` view by adding a path entry in `urlpatterns` list in `onlinecourse/urls.py`:

```
1
```

```
path('login/', views.login_request, name='login'),
```

Copied!

Now we can test the login function by refreshing the course list page:



You could try both login and logout with the created superuser.

Coding Practice: User Registration

In the previous step, we used the CLI to create a superuser. For regular users,we will need to create a user registration template and view to receive and save user credentials.

At the model level, a user object will be created in the `auth_user` table to complete the user registration. After the user is created, we can log in the user and redirect the user to the course list page.

- Open `onlinecourse/templates/onlinecourse/course_list.html`, update the authentication section by adding a link `<a href="{% url 'onlinecourse:registration' %}">Signup</a>` pointing to a `registration` view to be created.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
```

```
19. 19
20. 20
21. 21
22. 22
```

```
{% if user.is_authenticated %}
    <div class="rightalign">
        <div class="dropdown">
            <button class="dropbtn">{{user.first_name}}</button>
            <div class="dropdown-content">
                <a href="{% url 'onlinecourse:logout' %}">Logout</a>
            </div>
        </div>
    </div>
    {% else %}
    <div class="rightalign">
        <div class="dropdown">
            <form action="{% url 'onlinecourse:registration' %}" method="get">
                <input class="dropbtn"  type="submit" value="Visitor">
                <div class="dropdown-content">
                    <a href="{% url 'onlinecourse:registration' %}">Signup</a>
                    <a href="{% url 'onlinecourse:login' %}">Login</a>
                </div>
            </form>
        </div>
    </div>
{% endif %}
```

Copied!

- Complete the following code snippet to create a `registration_request` view to handle a registration POST request:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
```

```
def registration_request(request):
    context = {}
    # If it is a GET request, just render the registration page
    if request.method == 'GET':
        return render(request, 'onlinecourse/user_registration.html', context)
    # If it is a POST request
    elif request.method == 'POST':
        # <HINT> Get user information from request.POST
        # <HINT> username, first_name, last_name, password
        user_exist = False
        try:
            # Check if user already exists
            User.objects.get(username=username)
            user_exist = True
        except:
            # If not, simply log this is a new user
            logger.debug("{} is new user".format(username))
        # If it is a new user
        if not user_exist:
            # Create user in auth_user table
            user = User.objects.create_user(#<HINT> create the user with above info)
            # <HINT> Login the user and
            # redirect to course list page
            return redirect("onlinecourse:popular_course_list")
        else:
            return render(request, 'onlinecourse/user_registration.html', context)
```

Copied!

▶ Click here to see solution

- Complete the following code snippet to create a `registration` template to accept user information:

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
6
7
8
9
10
11
```

```
<form action="{% url 'onlinecourse:registration' %}" method="post">
  <div class="container">
    <h1>Sign Up</h1>
    <hr>
    <!-- HINT, added inputs for username, firstname, lastname, and password  -->
    <div>
        {% csrf_token %}
        <button class="button" type="submit">Sign Up</button>
    </div>
  </div>
</form>
```

Copied!

▶ Click here to see solution

- Add a route for the `registration_request` view by adding a path entry in `urlpatterns` list in `urls.py`:

```
1
```

```
path('registration/', views.registration_request, name='registration'),
```

Copied!

Now refreshing the main page and try new registration function along with login/logout.

Summary

During this lab, you have gained an understanding of how Django authentication operates and have implemented templates and views for user login, logout, and registration.

Author(s)

Yan Luo