User Authentication in Django

Chapter 7

1

User authentication

- So far we've built a working application that uses forms, but we're missing a major piece of most web applications: user authentication.
- Implementing proper user authentication is famously hard; there are many security gotchas along the way so you really don't want to implement this yourself.
- Fortunately Django comes with a powerful, built-in user authentication system that we can use.

2

Introduction

- Managing users-lost usernames, forgotten passwords, and keeping information up to date-can be a real pain.
- As a programmer, writing an authentication system can be even worse.
- Lucky for us, Django provides a default implementation for managing user accounts, groups, permissions, and cookie-based user sessions out of the box.
- The Django authentication system handles both authentication and authorization.
- Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do.

3

Overview

The authentication system consists of Users with

- Permissions: Binary (yes/no) flags designating whether a user may perform a certain task
- Groups: A generic way of applying labels and permissions to more than one user
- A configurable password hashing system:
  Passwords are stored by default in Django using the PBKDF2 algorithm, providing a good level of security for your user's data. You can read more about this as part of the official Django documentation on how django stores passwords. The documentation also provides an explanation of how to use different password hashers if you require a greater level of security.
- Forms for managing user authentication.
- View tools for logging in users, or restricting content

4

Setting up Authentication

- Before you can use Django's authentication offering, you'll need to make sure that the relevant settings are present in your project's settings.py file.
- Within the settings.py file find the INSTALLED_APPS tuple and check that django.contrib.auth and django.contrib.contenttypes are listed.
- Whenever you create a new project, by default Django installs the auth app, which provides us with a User object containing: *username, password, email, first_name, last_name*.
- While django.contrib.auth provides Django with access to the authentication system, django.contrib.contenttypes is used by the authentication application to track models installed in your database.

5

The User Model

- User objects are the core of the authentication system. They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators and so on.
- Only one class of user exists in Django's authentication framework, that is, superusers or admin staff users are just user objects with special attributes set, not different classes of user objects.
- The primary attributes of the default user are: username, password, email, first_name, last_name.
- The model also comes with other attributes such as is_active.

6

Login Functionality

Django provides a default view for a login page via LoginView. All we need to do is add a project-level urlpattern for the auth system, a login template, and a small update to our settings.py file.

1. Add a project-level urlpattern for the auth system;
2. Create a login template to display the login form;
3. Edit settings.py to set where to redirect the user upon a successful login with the **LOGIN_REDIRECT_URL** setting;
4. Provide a link to login.

7

Login – (1) Add a project-level urlpattern for the auth system

Let's place our login and logout pages at the accounts/ URL. Hence, **edit** the project-level URLConfs to add the following:

```
path('accounts/', include('django.contrib.auth.urls')),
```

8

Login – (2) Creating the login template

- By default Django will look within a templates folder called registration for a file called login.html for a login form. So we need to create a new directory called registration and the required file in it.
- **Create** the template file templates/registration/login.html as follows:

```
<!-- templates/registration/login.html -->
{% extends 'base.html' %}
{% block content %}
<h1>Login</h1>
<form method="post"> {% csrf_token %}
  {{ form.as_p}}
<button type="submit">Login</button>
</form>
{% endblock content %}
```

9

Login – (3) LOGIN_REDIRECT_URL setting

- **Edit** settings.py to set where to redirect the user upon a successful login with the **LOGIN_REDIRECT_URL** setting. At the bottom of the settings.py file add the following:

```
LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'home'
```

- Now after successful login, the user will be redirected to the 'home' template which is our homepage.
- We can take this chance to add the **LOGOUT_REDIRECT_URL** setting to redirect the user upon logout as well.

10

Login – (4) Provide a link to login

- **Edit** base.html file such that if the user is logged in, we say hello to them by name with a link to logout; if not we provide a link to our newly created login page.
- Note the use of the is_authenticated attribute.
- Recall that we did not write the views and URL pattern for 'logout' and 'login'. These are provided by Django by default.
- We have completed the steps needed for Login. Check that it works at the URL P18XXXXX.pythonanywhere.com/accounts/login/
- Next, let's move to user registration (Signup).

```
<!-- templates/base.html -->

{% if user.is_authenticated %}
    <p>Hi {{ user.username }}!</p>
    <p><a href="{% url 'logout' %}">logout</a></p>
{% else %}
  <p>You are not logged in.</p>
  <a href="{% url 'login' %}">login</a>
{% endif %}
{% block content %}
{% endblock content%}
</div>
</body>
</html>
```

11

User registration / SignUp

To set the user registration functionality, we will go through the following steps:

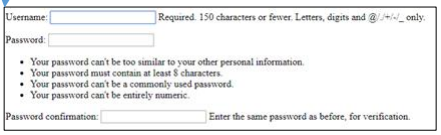
1. **Create** an application "accounts" in your project folder of your virtual environment and register it to settings.py;
2. Add a project-level urlpattern to point to this new app;
   path('accounts/', include('accounts.urls')),
3. **Create** the file accounts/urls.py (application-level URLConfs) with the url mapping to handle signup.
   path('signup/', views.SignUpView.as_view(), name='signup'),
4. Write the logic for the view SignUpView as stated in step 3 above, in views.py.
5. **Create** the template file (templates/signup.html) for the presentation of the Signup page.
6. Provide a link to sign up.
   Sign Up

12

SignUp – (4) Write the logic for the view SignUpView

- **Edit** the file accounts/views.py by adding the code.
- Django provides us with a form class, UserCreationForm, to build the signup page to register new users easily.
- By default it comes with three fields: **username, password1, and password2**.
- The view uses the built-in UserCreationForm and generic CreateView.
- And we use reverse_lazy to redirect the user to the login page upon successful registration.
- We use reverse_lazy instead of reverse because for all generic class-based views, the urls are not loaded when the file is imported, so we have to use the lazy form of reverse to load them later when they're available.
- Next, let's add signup.html to our project-level templates folder.

```
# accounts /views.py
from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy
from django.views import generic
class SignUpView (generic CreateView):
    form_class = UserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'signup.html'
```



13

SignUp – (5) Creating the template file for the presentation of the Signup page

- **Create** the template file templates/signup.html as follows:

```
<!-- templates/signup.html -->
{% extends 'base.html' %}
{% block content %}
<h1> Sign up </h1>
<form method="post">{% csrf_token %}
  {{ form.as_p}}
<button type="submit">Sign up</button>
</form>
{% endblock content %}
```

- We have completed the steps needed for Sign Up. Check that it works at the URL P18XXXXX.pythonanywhere.com/accounts/signup/

14

Django's built-in support for User authentication

- With minimal amount of code, the Django framework has allowed us to create a login, logout, and signup user authentication flow.
- Django provides everything we need for login and logout, but we need to create our own form to sign up new users.
- Under-the-hood it has taken care of the many security gotchas that can crop up if you try to create your own user authentication flow from scratch.

15

Summary

- User Accounts
- Built-in user authentication system

16