SDEV 2401

Rapid Backend App Development

Authentication Authorization and Users - 1 and 2

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

Agenda

On the right is what we will cover today.

- Why Authentication Matters
- The Authentication Flow (Movie Review Example)
- Custom User Models: Why and How
- Implementing a Custom User Model (Code Example)
- Registration with Custom Fields
- Registration View (Detailed Example)
- Login and Logout: Built-in and Custom Views
- Custom Login View (Code Example)
- Template Integration: Dynamic Navigation
- Restricting Access: Decorators in Action
- Challenge: Apply What You Learned
- Conclusion

Why Authentication Matters

- Authentication is the process of verifying a user’s identity before granting access to resources.
- Example: In a **university portal**, only students can view their grades, while professors can submit them.
- Authorization is different: it determines what an authenticated user can do.
- Frameworks like Django and Flask provide secure, reusable authentication systems.

The Authentication Flow (Movie Review Example)

1. **User visits** a movie review site.
2. If not logged in, they can only browse reviews.
3. If they want to write a review, the site checks if they’re authenticated.
4. **Middleware** intercepts the request:
   - If not authenticated, redirects to login.
   - If authenticated, allows access to the review form.
5. After login, the user is redirected back to the review page.

Diagram will be provided in the example

- Client (browser) → Server (middleware) → View (permission check) → Response

Custom User Models: Why and How

- Built-in user models are limited to basic fields (username, password, email).
- Custom models let you add fields like `membership_level`, `department`, or `date_of_birth`.
- Example: In a **gym app**, add a `membership_level` ("basic", "premium", "trainer").
- Steps:
  1. Create a new app (e.g., `members`).
  2. Inherit from `AbstractUser` in `models.py`.
  3. Add custom fields.
  4. Update `settings.py`: `AUTH_USER_MODEL = 'members.Member'`.
  5. Register the model in the admin for management.

Implementing a Custom User Model (Code Example)

```python
from django.db import models
from django.contrib.auth.models import AbstractUser

class Member(AbstractUser):
    MEMBERSHIP_CHOICES = [
        ("basic", "Basic"),
        ("premium", "Premium"),
        ("trainer", "Trainer"),
    ]
    membership_level = models.CharField(max_length=10, choices=MEMBERSHIP_CHOICES)
    date_of_birth = models.DateField(null=True, blank=True)

    def __str__(self):
        return f"{self.username} ({self.membership_level})"
```

Registration with Custom Fields

- Use Django's `UserCreationForm` as a base for your registration form.
- Add custom fields (e.g., `membership_level`, `favorite_genre`).
- Example: In a **book club app**, users select their favorite genre during registration.
- Always use Django's built-in forms for password handling for security.

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import Member

class MemberRegistrationForm(UserCreationForm):
    class Meta:
        model = Member
        fields = ["username", "email", "membership_level", "password1", "password2"]
```

Registration View (Detailed Example)

Below is an example of a registration view that handles user sign-up with custom fields.

```
from django.shortcuts import render, redirect
from .forms import MemberRegistrationForm
from django.contrib.auth import login

def register(request):
    if request.method == "POST":
        form = MemberRegistrationForm(request.POST)
        if form.is_valid():
            member = form.save()
            login(request, member)  # Log in after registration
            return redirect("dashboard")
    else:
        form = MemberRegistrationForm()
    return render(request, "members/register.html", {"form": form})
```

Login and Logout: Built-in and Custom Views

- **Built-in views:** Quick setup, less code, secure by default.
- **Custom views:** More control (e.g., logging login attempts, custom redirects).
- Example: In a **conference app**, redirect speakers to a dashboard, attendees to the schedule.

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

urlpatterns = [
    path("register/", register, name="register"),
    path("login/", auth_views.LoginView.as_view(template_name="members/login.html"), name="login"),
    path("logout/", auth_views.LogoutView.as_view(), name="logout"),
]
```

Custom Login View (Code Example)

Below is an example of a custom login view that uses Django's `AuthenticationForm` to handle user authentication.

```
from django.contrib.auth import authenticate, login
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm

def custom_login(request):
    if request.method == "POST":
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect("dashboard")
    else:
        form = AuthenticationForm()
    return render(request, "members/login.html", {"form": form})
```

Template Integration: Dynamic Navigation

- Show different navigation links based on authentication status.
- Example: In a **recipe app**:
  - Logged-in users see "Add Recipe" and "Logout".
  - Guests see "Login" and "Register".
- Use Django template tags:

```
{% if user.is_authenticated %}
  Hello, {{ user.username }} ({{ user.membership_level }})
  <form method="post" action="{% url 'logout' %}">{% csrf_token %}<button>Logout</button></form>
{% else %}
  <a href="{% url 'login' %}">Login</a>
  <a href="{% url 'register' %}">Register</a>
{% endif %}
```

Restricting Access: Decorators in Action

- Use `@login_required` to protect views.
- Example: Only logged-in users can submit event feedback.

```python
from django.contrib.auth.decorators import login_required

@login_required
def submit_feedback(request):
    # Only logged-in users can access this view
    ...
```

This is authentication, where we are checking if the user is logged in before allowing access to certain views.

In the next class and example we'll be taking a look at authorization, where we check if a user has permission to do something based on their role or group or custom permission.

Challenge: Apply What You Learned

- Take an app that we've built earlier in the course.
- Add authentication to specific views like creating, updating, or deleting items.
- Add custom fields to the user model relevant to the app's context.
- Test the registration, login, and logout flows.

Conclusion

- Authentication is more than just login/logout.
- Custom user models and role-based access make your app flexible and secure.
- Try adding authentication to your next project with new custom fields and roles!

Example

Let’s go do an example together

Authentication Fundamentals: New Perspectives

A LEADING POLYTECHNIC COMMITTED TO YOUR SUCCESS