Homework 2: Introduction to Django

1 Objective

You are tasked with building a RESTful Movie Theater Booking Application using Python and Django. The application should allow users to:

- View movie listings via the API
- Book seats via the API
- Check their booking history via the API
- An attractive user interface using Django templates and Bootstrap that displays and manipulates the same data as the API.

REMINDER: Push to your github repo frequently! You should be pushing from your homework folder, created in Homework 1. Double check git status to see that your new work is staged for commit. Write good comments each time you push.

2 Time Requirement

Approximately 4-5 hours.

3 Instructions

The following instructions will help you get started. These are not all inclusive, and you will need to spend some time researching solutions to the requirements.

3.1 Project Setup (30 minutes)

1. Create a new Django project named `movie_theater_booking` inside of the *homework2* directory. Django Getting Started - Docs

2. Set up a virtual environment and install dependencies:

```
    python3 -m venv myenv --system-site-packages
    source myenv/bin/activate
    pip install django djangorestframework
```

3.2 Creating the Booking App (30 minutes)

Within your project folder, create a Django app named `bookings` with the following models:

- **Movie**: title, description, release date, duration.
- **Seat**: seat number, booking status.
- **Booking**: movie, seat, user, booking date.

3.3 Implementing the MVT Architecture (1 hour)

Model:

- Create and run migrations to set up the database tables for your models.

View:

- Develop the following views using Django REST Framework's viewsets:
- MovieViewSet: For CRUD operations on movies.
  - SeatViewSet: For seat availability and booking.
  - BookingViewSet: For users to book seats and view their booking history.

Templates:

- Create HTML templates for user interactions (movie listings, seat booking, and booking history) using Django's templating system.

3.4 Creating an Attractive User Interface (1 hour)

Create a templates directory in your `bookings` app to store HTML files:

```
bookings/
    templates/
        bookings/
            base.html
            movie_list.html
            seat_booking.html
            booking_history.html
```

- Base Template (base.html): Use Bootstrap for responsive design. Include the Bootstrap CSS link in your base template.
- Create Individual Templates: Design the movie listing, seat booking, and booking history pages, ensuring they are visually appealing and user-friendly.

Example of a movie listing template (`movie_list.html`):

```
{% extends 'bookings/base.html' %}
{% block title %}Movie List{% endblock %}
{% block content %}
<h2>Available Movies</h2>
<ul class="list-group">
    {% for movie in movies %}
        <li class="list-group-item">
            <h5>{{ movie.title }}</h5>
            <p>{{ movie.description }}</p>
            <a href="{% url 'book_seat' movie.id %}" class="btn btn-primary">
                Book Now</a>
        </li>
    {% endfor %}
</ul>
{% endblock %}
```

3.5 RESTful API Implementation (1 hour)

You will create an API using serializers and URLs.

- Serializers:
- Create serializers for the Movie, Seat, and Booking models to convert them to JSON format.
- URLs:
- Set up URL routing for the API endpoints in urls.py.
- Endpoints should include:
  - /api/movies/: List all movies and allow CRUD operations.
  - /api/seats/: Check seat availability and book seats.
  - /api/bookings/: View booking history and create new bookings.

3.6 Testing & Running Locally (1 hour)

- You will want to run the application locally inside of the DevEdu environment.
  - Start the server. Then navigate to the "app" button inside of DevEdu dashboard (next to the editor). The link will be similar to *https://app-containerName-SectionID.devedu.io/*

    ```
    python3 manage.py runserver 0.0.0.0:3000
    ```

- Write Unit Tests:
  - Create unit tests for your code in tests.py.
- Integration Testing:
  - Test the API endpoints to ensure they return the correct responses (status codes, data format).
- Behavior-Driven-Design (BDD) Testing:
  - Test the behavior of the application using Behave.

3.7 Deployment (1 Hour)

- Deploy using Render.
  - Render is a unified cloud to build and run all your apps and websites with free TLS certificates, global CDN, private networks and auto deploys from Git.

4 Resources and Examples

Below are some resources that might help with this homework assignment.

- Django Documentation: Django Official Docs
- Django REST Framework Documentation: DRF Docs
- Bootstrap Documentation: Bootstrap
- Testing Documentation: Testing in Django

REMINDER: Any use of AI in your assignments, whether for generating ideas, text, code, or other content, must be explicitly cited in the `README`. This means clearly indicating which AI tools were used, what they were used for, and how the generated content was incorporated into your work.

5 Submission Requirements

1. Submit a zip file using Canvas containing:
   - Project source code with tests.
   - README file with setup instructions.
   - GitHub repository with frequent commits.
2. Push all files to your GitHub repository
3. Have all code in DevEdu for testing and running purposes

6 Assessment Criteria

- **Functionality (25 points)**: Meets all requirements.
- **User Experience (15 points)**: Clean UI with Bootstrap.
- **Code Quality (15 points)**: Follows Django conventions.
- **Testing (20 points)**: Unit and integration tests included.
- **Deployment (20 points)**: Accessible via Render URL.
- **Documentation (5 points)**: Clear README included.

6.1 Rubric (Total: 100 Points)

Criteria	Points	Description
Functionality	35	- Application meets all requirements (view movie listings, book seats, check booking history). - RESTful API is fully functional and responds as expected.
User Experience	15	- User interface is visually appealing and user-friendly. - Effective use of Bootstrap or similar frameworks for design. - Application is easy to navigate.
Code Quality	15	- Code is well-organized and follows Django conventions. - Proper use of comments and documentation. - Efficient use of models, views, and serializers.
Testing	20	- Comprehensive unit and integration tests are written. - Tests cover edge cases and provide adequate coverage for all major functionality (80% Coverage). - Tests are properly organized and easy to understand. - Tests pass.
Deployment	10	- Application is successfully accessible via App in Render.
Documentation	5	- README file is complete, explaining project structure, setup, and how to run the application. - Documentation is clear and helpful for understanding the application.

Table 1: Homework 2 Grading Rubric