Bus Booking System - Django Project Documentation

Aman Sanjay Poddar

October 24, 2024

Contents

1 Introduction

1.1 Project Motivation

The Bus Booking System is designed to streamline the process of reserving bus seats online. It simplifies the user experience, providing a seamless way for users to browse available buses, check seat availability, and book or cancel reservations. The motivation behind this project is to demonstrate the use of Django's robust features, including authentication, data relationships, and views, in a real-world application.

1.2 Project Overview

This project includes the following features:

- User authentication and authorization system.
- A dynamic admin dashboard to manage buses, routes, and seats.
- Interactive views for browsing buses, booking seats, and managing bookings.
- A bus-like seat layout for better user interaction.

2 Why Django?

Django was chosen for this project due to several key benefits that align well with the requirements of a bus booking system. Below are the main reasons for selecting Django:

2.1 1. Rapid Development

Django's "batteries-included" philosophy provides a wide range of pre-built components like authentication, URL routing, and form handling. This allowed for faster development of the core functionalities, such as user registration, booking flow, and admin dashboard, without having to reinvent common features.

2.2 2. Built-in Authentication System

Django comes with a built-in authentication system, which was essential for this project as secure user logins and registrations were required. It supports password encryption, session management, and various authentication methods (including OAuth, which was used for Google login), ensuring a robust and secure authentication flow.

2.3 3. Model-View-Template (MVT) Architecture

Django follows the MVT architecture, which cleanly separates the business logic from the presentation layer. This ensures maintainability and scalability. Each component of the bus booking system—such as viewing available buses, booking a seat, and canceling a booking—is neatly organized within this structure:

- **Models**: Represent the database schema and relationships (e.g., Bus, Seat, Booking).

- **Views**: Handle the business logic and respond to user interactions.
- **Templates**: Define the user interface and how the data is presented.

2.4 4. Django Admin Dashboard

One of the standout features of Django is its built-in admin interface, which allows admins to manage buses, routes, bookings, and users without writing additional code. This out-of-the-box functionality saved significant development time and provided a clean, extendable way to manage the application's data.

2.5 5. ORM (Object-Relational Mapping)

Django's ORM abstracts away the complexities of SQL queries and database interactions. The ORM made it easier to work with database relationships (such as the relationship between buses and seats) in a more Pythonic way, simplifying code and reducing the chances of SQL injection vulnerabilities.

2.6 6. Security Features

Django comes with several built-in security mechanisms:

- **CSRF Protection**: Cross-Site Request Forgery tokens are automatically handled by Django to secure forms.
- **SQL Injection Protection**: The ORM ensures that all queries are automatically escaped, reducing vulnerabilities.
- **Secure Password Storage**: Passwords are hashed and stored securely, following industry standards.

These features made it easier to implement a secure booking system without worrying about low-level security threats.

2.7 7. Scalability and Flexibility

Django's modularity ensures that the application can scale with ease. New features, such as a payment gateway or real-time notifications, can be added without overhauling the entire system. Additionally, Django supports multiple database backends, making it flexible for future upgrades.

2.8 8. Community and Documentation

Django has a strong, active community and excellent documentation. This was particularly beneficial during development, as most issues encountered had existing solutions or were well-documented in the official Django documentation or community forums.

3 Benefits of this Structure

The bus booking system's structure, which leverages Django's MVT pattern, has several advantages:

- **Maintainability**: The clear separation of concerns makes it easy to update and maintain the system. Changes to one part of the system (e.g., database schema) do not affect other parts (e.g., templates).
- **Reusability**: Components such as templates and forms can be reused across different views, making the development process more efficient.
- **Scalability**: The system is designed to handle increasing user traffic and data without a significant performance drop, thanks to Django's modular and efficient design.
- **Extensibility**: New features (e.g., adding different types of buses, routes, or additional booking options) can be easily added without disrupting the existing codebase.

Overall, Django provided a robust, secure, and scalable framework for developing the bus booking system.

4 User Authentication System

4.1 Login and Registration

The system uses Django's built-in authentication to manage user login and registration. Users can sign up using a username, email, and password. The system also supports social login via Google using Django Allauth.

4.2 Permissions and Access Control

Authenticated users are required to log in to access certain features of the system, such as browsing buses, booking seats, and viewing their dashboard. If an unauthenticated user attempts to access these views, they are redirected to the login page.

Code example for user authentication in views:

```
@login_required(login_url="/accounts/google/login/")
def dashboard(request):
    # User-specific logic here
```

5 Django Admin Dashboard

Django's powerful admin interface is used to manage buses, routes, seats, and bookings. Admins have full control over:

- Adding, updating, and deleting bus details.
- Managing routes and the number of available seats.
- Viewing and managing all bookings made by users.

6 Views

6.1 Dashboard View

The dashboard displays a user's bookings and provides options to cancel existing reservations.

6.2 Browse Buses

Users can search for buses based on source and destination and view available buses for their preferred route.

6.3 Check Seat Availability

For each bus, the system displays a layout of seats with availability status (booked or not). Users can then proceed to book a seat.

6.4 Book Seat

When booking a seat, users select a seat, confirm their details, and complete the booking. If successful, the seat is marked as booked.

6.5 Cancel Booking

Users can cancel a booking, which frees up the seat for other users. The system automatically updates the bus's occupancy.

Example code for booking and canceling:

```
@login_required
def book_seat(request, seat_id):
    seat = get_object_or_404(Seat, id=seat_id, is_booked=False)
    # Booking logic here

@login_required
def cancel_booking(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id, user=request.user)
    # Cancellation logic here
```

7 Models

7.1 Route Model

The `Route` model stores information about the bus route, including the source, destination, distance, and estimated time.

7.2 Bus Model

Each bus is associated with a route and has attributes such as the total number of seats and current occupancy.

7.3 Seat Model

Each seat belongs to a bus and has a seat number and booking status. The `Seat` model is used to track whether a seat is available or booked.

7.4 Booking Model

A booking links a user to a seat, along with the time of booking. Each seat can have only one booking at a time.

8 Relationships Between Models

- A `Route` has many `Buses`.
- A `Bus` has many `Seats`.
- A `Seat` has one `Booking`.
- A `User` can have many `Bookings`.

The relationships are implemented using Django's foreign key and one-to-one fields, which help manage data consistency and relational integrity.

9 Overall Flow of the Application

9.1 User Workflow

1. Users register or log in to the system.
2. Once authenticated, users can search for buses by specifying the source and destination.
3. After selecting a bus, users can view seat availability and proceed to book a seat.
4. Bookings can be canceled if needed, freeing up the seat for others.
5. Users can view all of their bookings in the dashboard.

9.2 Admin Workflow

1. Admins manage the buses, routes, and seats via the Django admin interface.
2. Admins can view all user bookings and manage bus occupancy.

10 How to Run the Project

10.1 Prerequisites

Before running the project, ensure that you have the following installed:

- Python 3.x
- Django 3.x
- PostgreSQL or SQLite (as a database)

10.2 Steps to Run

1. Clone the repository:

   ```
   git clone https://github.com/AmanPoddar04/bus-booking-system
   ```

2. Install the required dependencies:

   ```
   pip install -r requirements.txt
   ```

3. Run migrations to set up the database:

   ```
   python manage.py migrate
   ```

4. Create a superuser for accessing the admin panel:

   ```
   python manage.py createsuperuser
   ```

5. Start the development server:

   ```
   python manage.py runserver
   ```

6. Access the application at `http://localhost:8000`

11 System Architecture

The Bus Booking System follows a client-server architecture, where the client (web browser) interacts with the Django server to request and book bus seats. The system consists of three main components:

- **Client Side**: The user interacts with the system via HTML templates rendered by Django. Bootstrap is used to style the frontend and ensure responsiveness.
- **Server Side**: The Django framework handles all the business logic, including authentication, request handling, and rendering views.
- **Database**: PostgreSQL or SQLite is used to store information related to users, buses, routes, seats, and bookings.

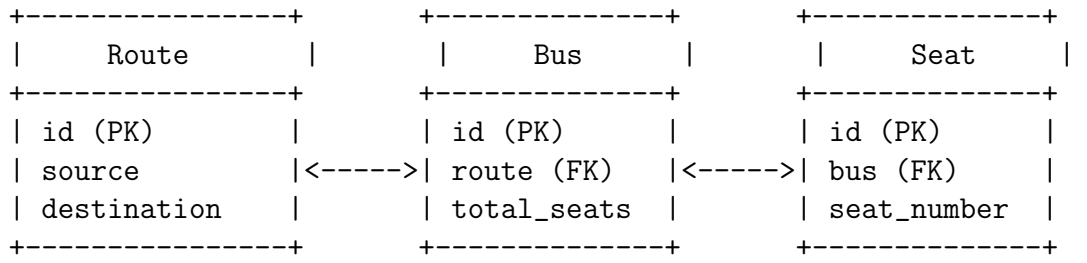
12 Technologies Used

The project uses the following technologies:

- **Django 5.1**: Web framework for handling the backend logic.
- **Bootstrap 5**: For responsive and mobile-first web design.
- **SQLite**: To store persistent data such as buses, seats, and user bookings.
- **Django Allauth**: For user authentication, including OAuth support (e.g., Google login).
- **Django ORM**: For database interaction and schema migrations.

13 Database Schema

The following is the database schema for the bus booking system. The relationships between models are represented below:



14 Error Handling

Error handling is an essential part of this system. It ensures a smooth user experience, even when unexpected situations arise.

- **Booking Conflicts**: If two users attempt to book the same seat simultaneously, the system will display an error message.
- **Form Validation**: The booking form validates user inputs and ensures that only valid seat numbers are selected.
- **Page Not Found**: Custom 404 and 500 error pages are implemented to guide users when they navigate to incorrect URLs.

Here's an example of how booking conflicts are handled:

```
@login_required
def book_seat(request, seat_id):
    seat = get_object_or_404(Seat, id=seat_id, is_booked=False)
    try:
        # Booking logic
    except IntegrityError:
        messages.error(request, "The seat has already been booked.")
        return redirect("browse_buses")
```

15 Security Measures

The following security measures are implemented in the system:

- **Authentication and Authorization**: Only logged-in users can access the booking system. OAuth via Google is integrated for secure authentication.
- **Password Encryption**: Django stores user passwords in an encrypted format using hashing algorithms.
- **Cross-Site Request Forgery (CSRF)**: CSRF tokens are used to protect against cross-site request forgery attacks.
- **Input Validation**: All forms are validated to prevent malicious inputs.

16 Testing

The system has been thoroughly tested using both manual and automated testing methods.

16.1 Unit Tests

Django's built-in testing framework is used to test individual components. Below are a few sample test cases:

- Test if a user can book a seat.
- Test if a user can cancel a booking.
- Test if an unauthenticated user is redirected to the login page.

Example unit test for booking a seat:

```
from django.test import TestCase
from .models import Bus, Seat, Booking
from django.contrib.auth.models import User

class BookingTestCase(TestCase):
    def setUp(self):
        self.user = User.objects.create(username="testuser")
        self.bus = Bus.objects.create(name="Bus A", total_seats=10)
        self.seat = Seat.objects.create(bus=self.bus, seat_number="1")

    def test_seat_booking(self):
        booking = Booking.objects.create(user=self.user, seat=self.seat)
        self.assertEqual(self.seat.is_booked, True)
```

17 Future Scope

Several features can be added to the Bus Booking System in future iterations:

- **Payment Gateway Integration**: Allow users to pay for their bookings via online payment methods.
- **Seat Reservation for Multiple Users**: Implement functionality to book multiple seats in one go.
- **Analytics Dashboard**: Create an admin dashboard to display real-time statistics on bookings, occupancy rates, and more.
- **Email and SMS Notifications**: Send booking confirmation, cancellation, and reminder notifications to users.

18 Conclusion

The Bus Booking System project showcases a real-world implementation of a seat reservation system using Django. It combines essential features such as user authentication, dynamic views, and an admin dashboard to manage bus data efficiently. Future improvements may include the addition of payment integration and more detailed reporting features.