Air Ticket Booking System

Air Ticket booking System

Submitted by
Md Umair Karim Ifty
ID: 181-35-2335
Department Of Software Engineering
Daffodil International University

Supervised by
Khalid Been Md. Badruzzaman Biplob
Senior Lecturer
Department Of Software Engineering
Daffodil International University

For the purpose of fulfilling the requirements for the Bachelor of Science in Software Engineering, this project report has been submitted

Summer-2025

APPROVAL

This thesis titled on "**Air Ticket Booking System**", submitted by **Md Umair Karim Ifty (ID: 181-35-2335)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

---------- **Chairman**

Dr. S M Hasan Mahmud
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---------- **Internal Examiner 1**

Tapushe Rabaya Toma
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---------- **Internal Examiner 2**

Khalid Been Badruzzaman Biplob
Lecturer (Senior Scale)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

---------- **External Examiner**

Dr. Md. Sazzadur Rahman
Professor
Institute of Information Technology
Jahangirnagar University

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Khalid Been Md. Badruzzaman Biplob, for his invaluable guidance throughout this project. I also thank the faculty members of the Department of Software Engineering for their support during my undergraduate studies.

Special thanks to my classmates who participated in testing and provided feedback. Finally, I am grateful to my family for their continuous encouragement and support.

Abstract

This project outlines the development work on a web-based Air Ticket Booking System using Python's Django Framework combined with a PostgreSQL database. It optimizes processes concerning search operations, bookings, and ticket keeping among different users such as passengers, travel agents, and administrators.

It has features such as flight search in real time, interactive seat choosing, secure login for users, automatic PNR generation, and booking control. It adopts a three-tier architecture within Django's Model-View-Template (MVT) pattern to achieve clean separation of concerns and maintainability.

Database design follows rules of normalization to maintain data in a healthy state while enabling optimal query performance. Security features include password hashing, CSRF protection, and SQL injection protection within Django's ORM.

It was found during testing that search operations remain below a response time of 3 seconds and the system handles concurrent users efficiently. It has a functional web interface that responds on both desktop and mobile browsers. Through simulated payment processing, module-based architecture can be incorporated later on with actual air API and payment gateways.

Keywords: Django, PostgreSQL, Web Application, Airline Booking, Python

Contents

List of Figures

List of Tables

Introduction

1.1 Background

The airline industry serves millions of passengers daily, requiring efficient systems to manage reservations, schedules, and passenger information. Traditional manual booking methods are often time-consuming and error-prone, whereas modern web-based systems provide 24/7 availability, instant confirmation, and enhanced user experience.

Online reservation systems have become a critical part of airline infrastructure, reducing operational costs while improving customer satisfaction. These systems must handle complex operations such as flight searches, seat allocation, payment processing, and ticket generation, all while ensuring data integrity and security.

1.2 Problem Statement

Despite advancements, many existing airline booking systems face significant challenges:

- Complicated user interfaces that hinder customer experience
- Performance issues during peak booking periods
- Limited role-based functionality (passengers, agents, administrators)
- Security vulnerabilities in managing sensitive data
- Poor mobile responsiveness

Small and medium-sized airlines often struggle to afford commercial solutions, while many open-source alternatives lack comprehensive features.

1.3 Project Objectives

The primary objectives of this project are to:

1. Develop a fully functional web-based airline ticket booking system using the Django framework
2. Implement multi-role support for passengers, agents, and administrators
3. Design an intuitive and responsive user interface
4. Ensure secure handling of user data and transactions
5. Apply software engineering best practices throughout development

6. Provide detailed documentation for future extension and maintenance

1.4 Project Scope

1.4.1 In Scope

The project will include:

- User registration and authentication
- Flight search and filtering
- Online booking with seat selection
- Booking management (view, modify, cancel)
- Administrative panel for managing flights and users
- Report generation features
- Automated email notifications

1.4.2 Out of Scope

The project will not cover:

- Integration with real airline APIs
- Payment gateway integration
- Mobile application development
- Multi-language support
- Complex fare rules and loyalty programs

1.5 Project Significance

This project highlights the practical application of:

- Web development using the Django framework
- Database design and management with PostgreSQL
- Core software engineering principles
- Security considerations in web applications
- User interface and experience design

The system serves as both a learning resource and a potential foundation for commercial use by small airlines or travel agencies.

Literature Review

2.1 Existing Systems Analysis

2.1.1 Commercial Systems

Major airline booking systems like Amadeus and Sabre provide comprehensive functionality but require significant investment and technical expertise. These enterprise solutions are designed for large airlines and may be overwhelming for smaller operations.

2.1.2 Open Source Solutions

Limited open-source booking systems exist [22], and most lack essential features like multi-role support, comprehensive reporting, or proper security implementation. This gap creates an opportunity for academic projects to contribute meaningful solutions.

2.2 Technology Review

2.2.1 Python Web Frameworks

Python offers several excellent web frameworks suitable for this project:

Django: A high-level framework following the "batteries included" philosophy [7]. It provides:

- Built-in ORM for database operations
- Automatic admin interface generation [7]
- Robust security features [14]
- Excellent documentation and community support

Flask: A micro-framework offering flexibility but requiring more configuration [17]. While suitable for small projects, it lacks Django's built-in features needed for complex applications [13].

FastAPI: Modern framework optimized for building APIs [13]. Excellent for microservices but less suitable for traditional web applications with server-side rendering.

For this project, Django was selected due to its comprehensive features, security focus, and rapid development capabilities [7].

2.2.2 Database Technologies

PostgreSQL: Selected for its [15]:

- ACID compliance ensuring transaction reliability [10]
- Advanced indexing for query optimization [18]
- JSON support for flexible data storage [15]
- Open-source nature with enterprise features [10]

MySQL: While popular, it lacks some PostgreSQL features like full-text search and advanced data types [10].

SQLite: Suitable for development but not recommended for production multi-user systems [21].

2.3 Related Academic Work

Recent studies emphasize the importance of user experience in booking systems. Research shows that simplifying the booking process to 3-4 steps significantly improves conversion rates. Security studies highlight common vulnerabilities in web applications [14] and recommend using framework-provided security features rather than custom implementations.

System Requirements

3.1 Functional Requirements

3.1.1 User Management

ID	Title	Description	Stakeholder
FR-01	Registration	User registration with email verification	Passenger, Agent, Admin
FR-02	Login	Secure login and session management	Passenger, Agent, Admin
FR-03	Password Reset	Reset password functionality	Passenger, Agent, Admin
FR-04	Profile Management	Manage and update user profile	Passenger, Agent, Admin
FR-05	Role-based Access	Role-based access control (Passenger, Agent, Admin)	Admin

3.1.2 Flight Operations

ID	Title	Description	Stakeholder
FR-06	Flight Search	Search flights by origin, destination, and date	Passenger, Agent
FR-07	Filtering	Filter results by price, airline, and time	Passenger, Agent
FR-08	Flight Details	View flight details and available seats	Passenger, Agent
FR-09	Flight Management	Admin can add/edit/delete flights	Admin
FR-10	Schedule Management	Manage flight schedules and routes	Admin

3.1.3 Booking Management

ID	Title	Description	Stakeholder
FR-11	Flight Selection	Select flights and seats	Passenger, Agent
FR-12	Passenger Info	Enter passenger information	Passenger, Agent
FR-13	PNR Generation	Generate unique PNR for each booking	System
FR-14	Booking History	View booking history	Passenger, Agent
FR-15	Booking Update	Modify or cancel bookings	Passenger, Agent
FR-16	Ticket Generation	Generate e-tickets	System

3.1.4 Payment Processing

ID	Title	Description	Stakeholder
FR-17	Fare Calculation	Calculate total fare including taxes	System
FR-18	Payment	Simulate payment processing	Passenger, Agent
FR-19	Receipt	Generate payment receipts	Passenger, Agent
FR-20	Refunds	Process refunds for cancellations	Passenger, Agent

3.1.5 Reporting

ID	Title	Description	Stakeholder
FR-21	Booking Reports	Generate booking reports	Admin
FR-22	Revenue Analysis	Revenue analysis reports	Admin
FR-23	Occupancy Report	Flight occupancy reports	Admin
FR-24	User Logs	User activity logs	Admin

3.2 Non-Functional Requirements

3.2.1 Performance

ID	Title	Description
NFR-01	Page Load	Page load time ¡ 3 seconds
NFR-02	Concurrent Users	Support 100 concurrent users
NFR-03	Search Speed	Search results within 2 seconds
NFR-04	DB Optimization	Database query optimization

3.2.2 Security

ID	Title	Description
NFR-05	Password Hashing	Password hashing using PBKDF2
NFR-06	CSRF Protection	CSRF protection on all forms
NFR-07	SQL Injection	SQL injection prevention via ORM
NFR-08	Session Timeout	Session timeout after 30 minutes
NFR-09	HTTPS	HTTPS enforcement

3.2.3 Usability

ID	Title	Description
NFR-10	Responsive	Responsive design for mobile devices
NFR-11	Navigation	Intuitive navigation
NFR-12	Error Messages	Clear error messages
NFR-13	Cross-browser	Cross-browser compatibility

3.2.4 Reliability

ID	Title	Description
NFR-14	Availability	99% uptime availability
NFR-15	Backup	Data backup mechanisms
NFR-16	Rollback	Transaction rollback on failures
NFR-17	Monitoring	Error logging and monitoring

System Design

4.1 Use Case Diagram

The system supports three main actor types with distinct responsibilities:

- **Passenger:** Search flights, make bookings, manage reservations
- **Agent:** All passenger functions plus bulk bookings
- **Administrator:** System configuration, user management, reporting

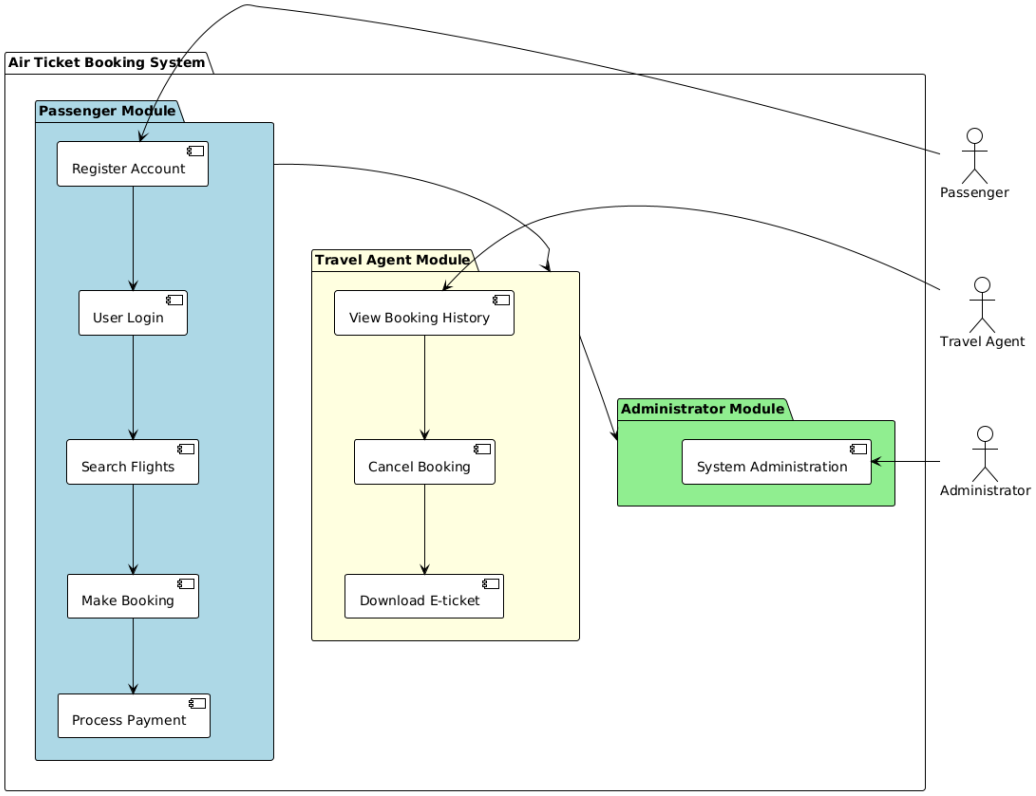

Figure 4.1: System Use Case Diagram

4.2 Use Case Descriptions

Use Case 1: Flight Search

Use Case Name	Flight Search
Actor	Passenger, Agent
Description	User searches for available flights based on given criteria.
Pre-Conditions	User must be logged in.
Trigger	Entering search criteria and submitting request.
Flow of Events	• User enters origin, destination, and date. • System validates input. • System queries the database. • System displays matching flight options.
Post Condition	User views available flight options.

Use Case 2: Booking Process

Use Case Name	Booking Process
Actor	Passenger, Agent
Description	User books a seat on a selected flight and completes payment.
Pre-Conditions	User must have selected a valid flight.
Trigger	Selecting a flight and proceeding to booking.
Flow of Events	• User selects flight and seat(s). • System checks seat availability. • User provides passenger details. • System calculates total fare. • User proceeds to payment. • System confirms booking and generates PNR.
Post Condition	Booking record is stored in the database with confirmed status.

Use Case 3: User Authentication

Use Case Name	User Authentication
Actor	Passenger, Agent, Administrator
Description	User logs into the system using valid credentials.
Pre-Conditions	User must have a registered account.
Trigger	Clicking the *Login* button after entering credentials.
Flow of Events	• User enters email and password. • User clicks on login button. • System validates credentials against database. • On success, system creates a session and redirects user to dashboard.
Post Condition	User is logged into the system with role-based access.

4.3 Activity Diagrams

Activity Diagram for Use Case 1: Flight Search

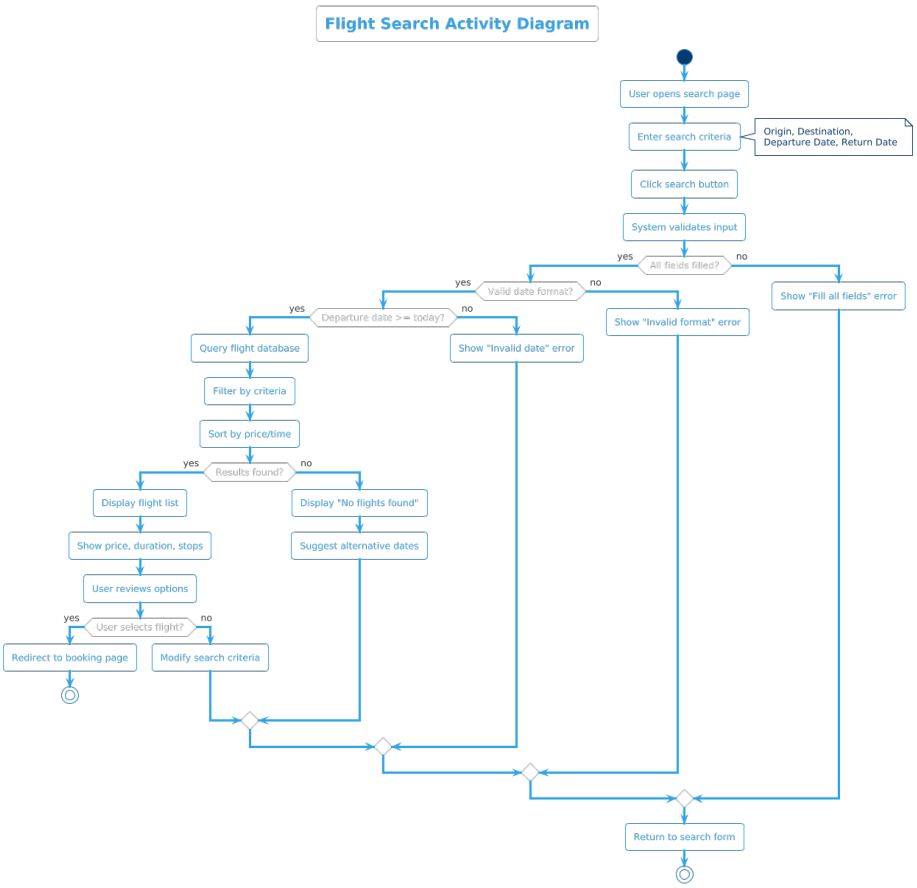

Figure 4.2: Flight Search Activity Diagram

Activity Diagram for Use Case 2: Booking Process

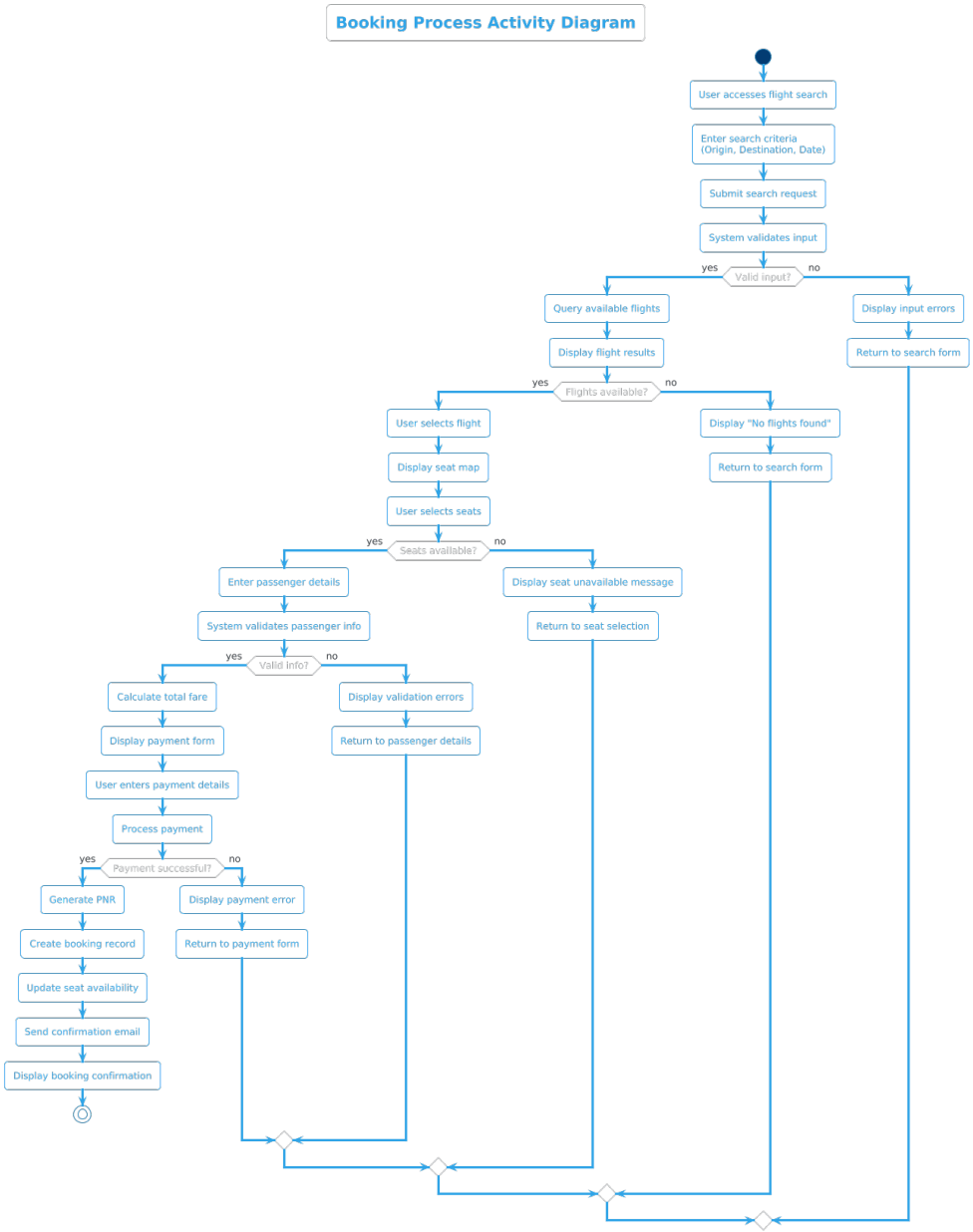

Figure 4.3: Booking Process Activity Diagram

Activity Diagram for Use Case 3: User Authentication

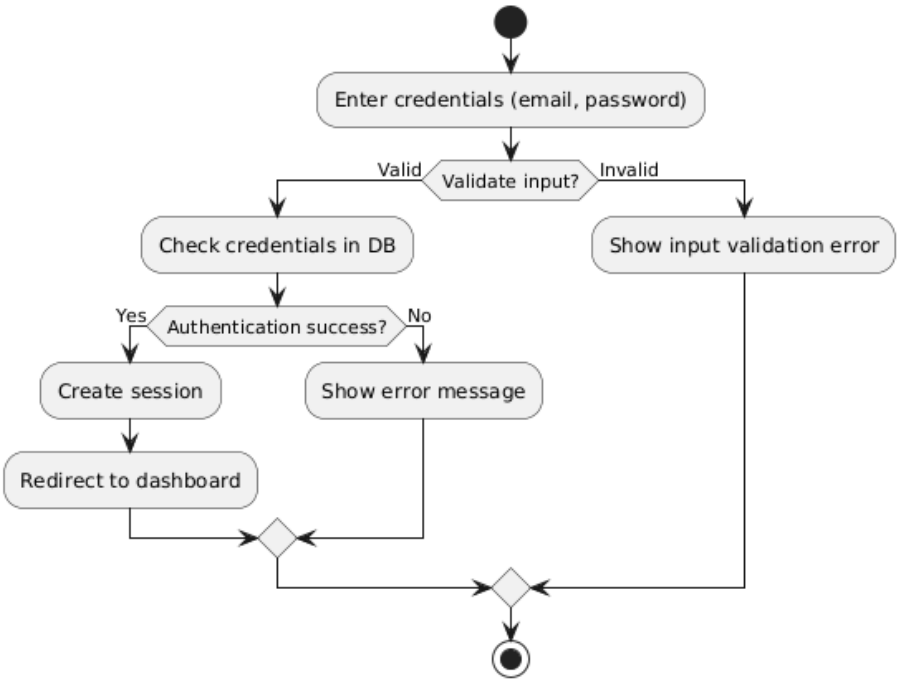

Figure 4.4: User Authentication Activity Diagram

4.4 Sequence Diagrams

Sequence Diagram for Use Case 1: Flight Search

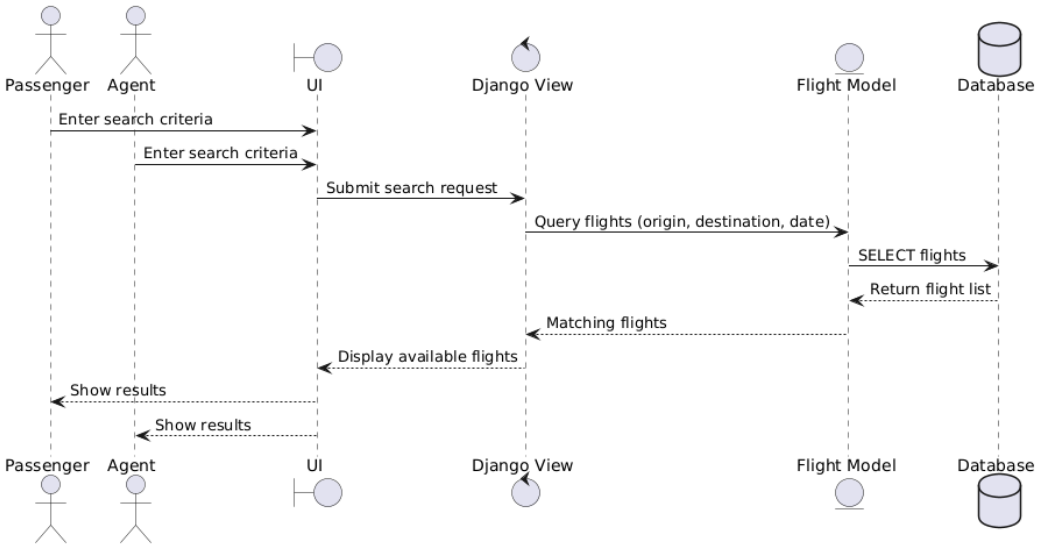

Figure 4.5: Flight Search Sequence Diagram

Sequence Diagram for Use Case 2: Booking Process

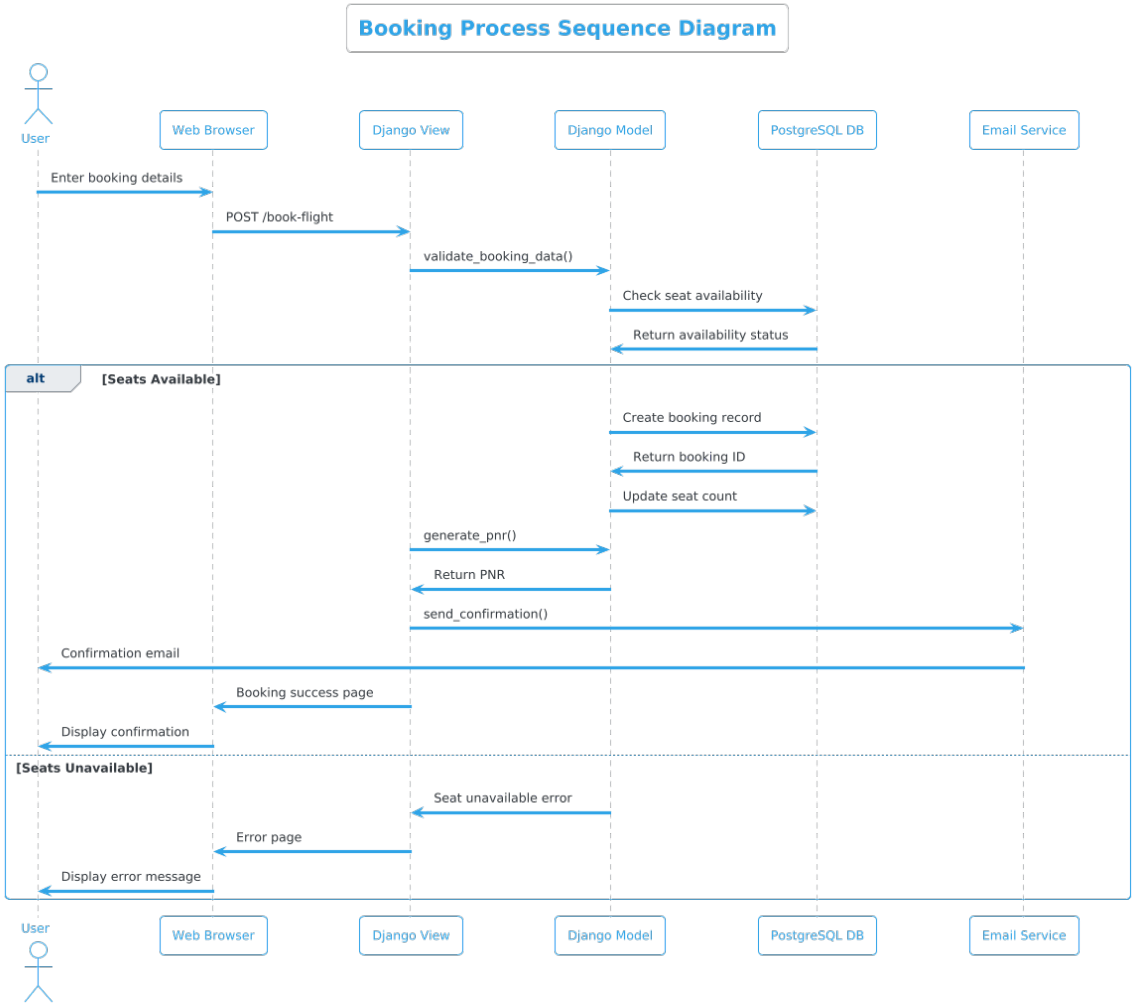

Figure 4.6: Booking Process Sequence Diagram

Sequence Diagram for Use Case 3: User Authentication

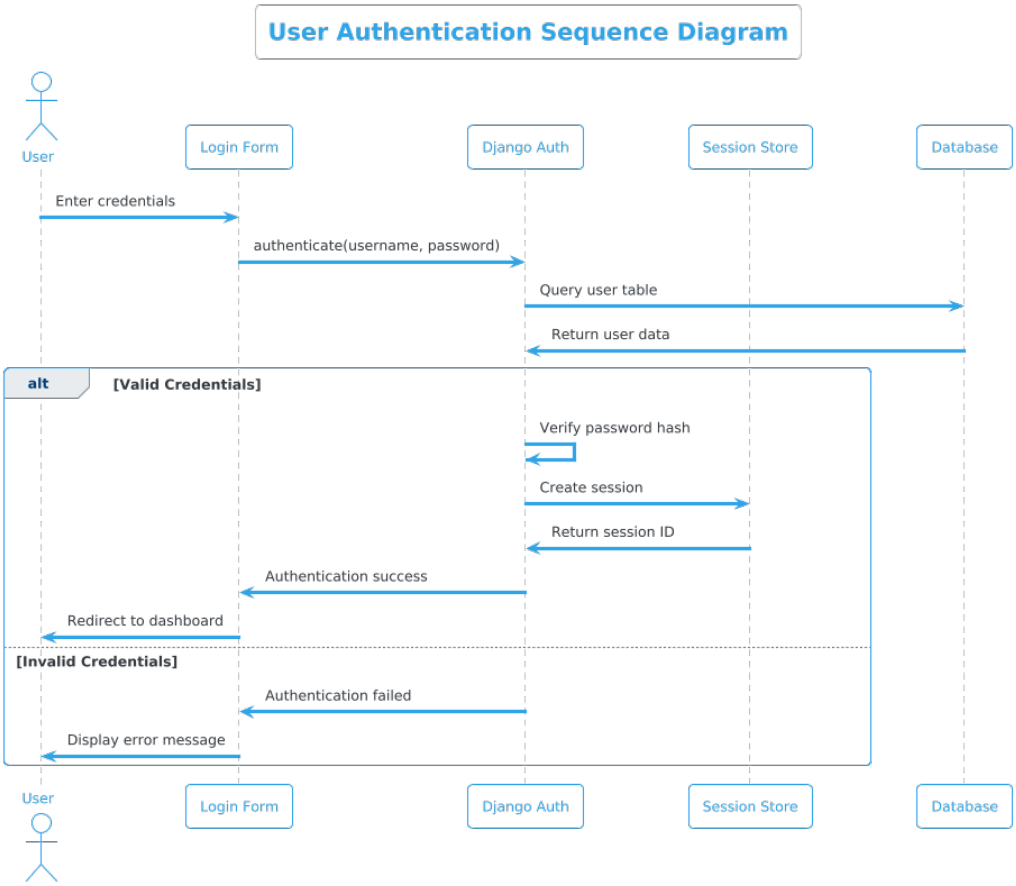

Figure 4.7: User Authentication Sequence Diagram

4.5 Global Entity Relationship Diagram

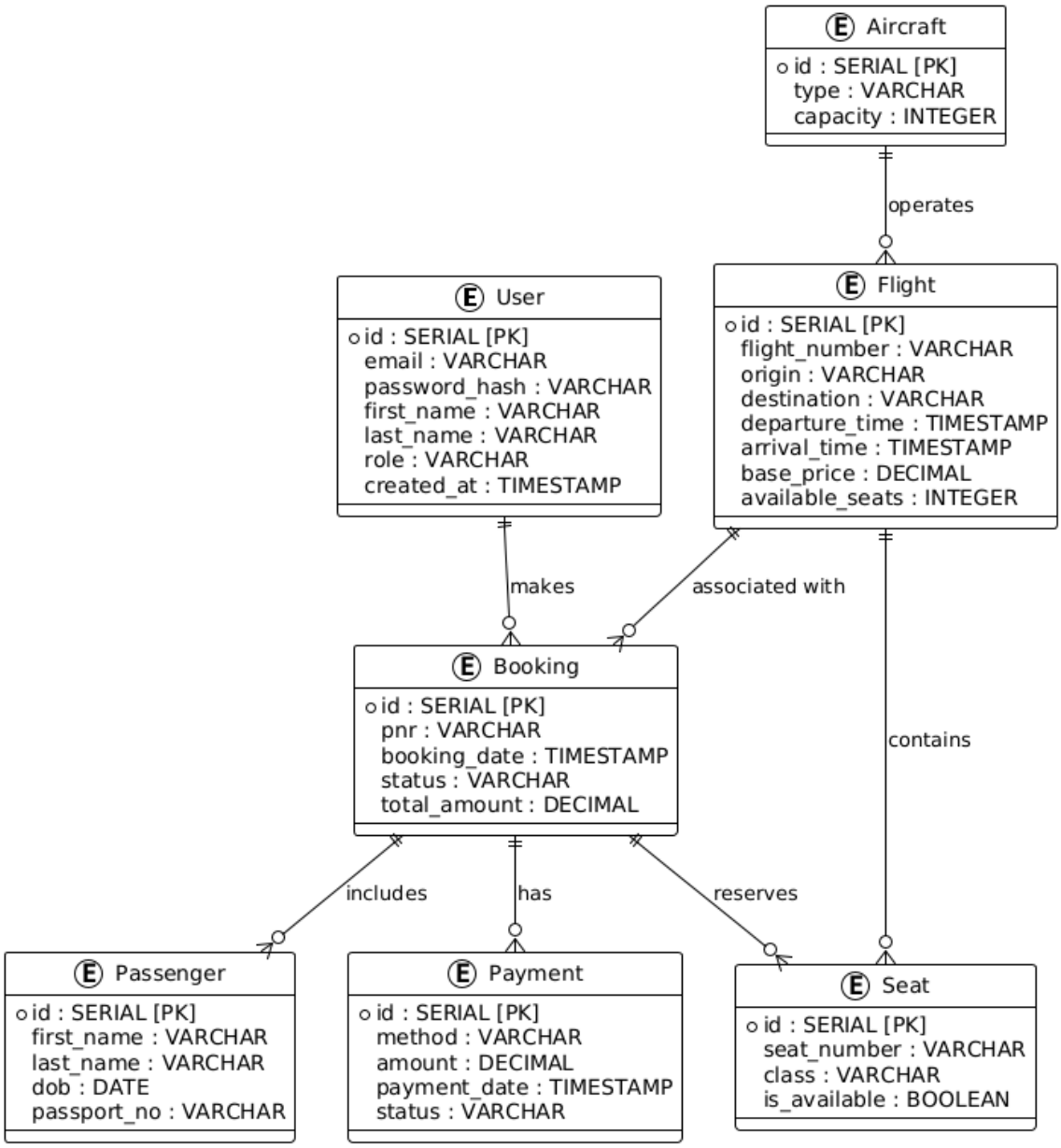

Figure 4.8: Entity Relationship Diagram

4.6 Database Schema

```sql
-- Users table with role support
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    role VARCHAR(20) DEFAULT 'passenger',
    created_at TIMESTAMP DEFAULT NOW()
);

-- Flights table
CREATE TABLE flights (
    id SERIAL PRIMARY KEY,
    flight_number VARCHAR(20) UNIQUE NOT NULL,
    origin VARCHAR(100) NOT NULL,
    destination VARCHAR(100) NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    arrival_time TIMESTAMP NOT NULL,
    aircraft_id INTEGER REFERENCES aircraft(id),
    base_price DECIMAL(10,2),
    available_seats INTEGER
);

-- Bookings table
CREATE TABLE bookings (
    id SERIAL PRIMARY KEY,
    pnr VARCHAR(10) UNIQUE NOT NULL,
    user_id INTEGER REFERENCES users(id),
    flight_id INTEGER REFERENCES flights(id),
    booking_date TIMESTAMP DEFAULT NOW(),
    status VARCHAR(20) DEFAULT 'confirmed',
    total_amount DECIMAL(10,2)
);
```

Core Database Tables

4.7 Class Diagram

Main Django model classes include:

- **User Model:** Extends Django's AbstractUser
- **Flight Model:** Manages flight information
- **Booking Model:** Handles reservation data
- **Payment Model:** Tracks payment status

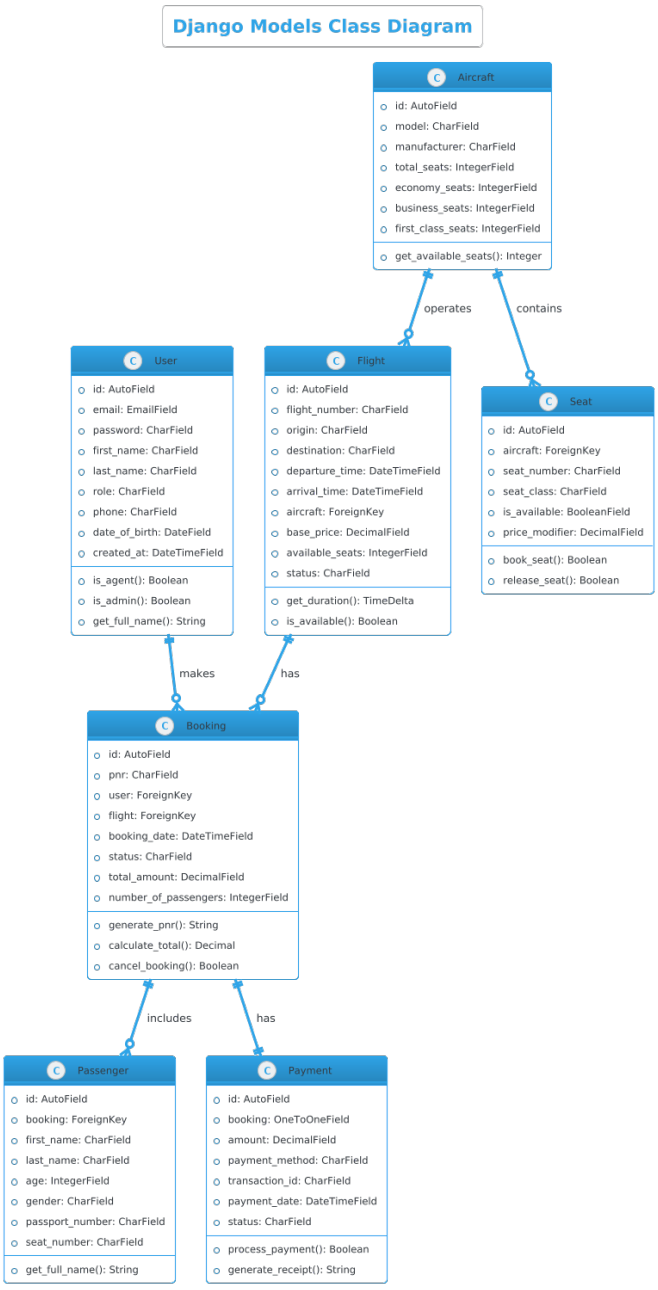

Figure 4.9: System Class Diagram

4.8 System Architecture Diagram

The overall system architecture shows the three-tier structure with presentation, business logic, and data layers.

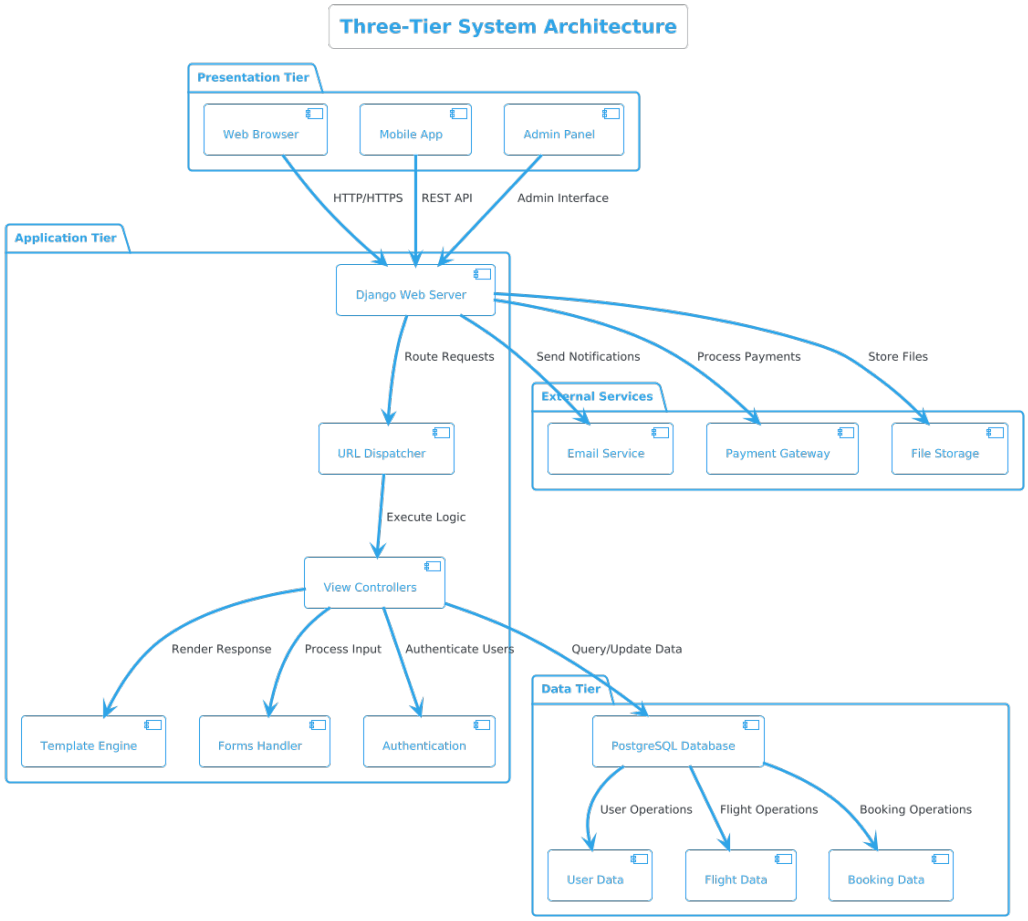

Figure 4.10: Three-Tier System Architecture

Testing and Results

5.0.1 Testing Strategy

The study adopted a multi-tier test approach for verifying the system, functionality, performance, and security. The test framework employed was Django's native test utilities with other tools so that coverage is comprehensive across all application facets.

Unit test for verifying model specific method, form validation and Utility functions that run within limited environments.

Integration testing examined the interactions between models, views, and templates of Django, with particular focus on booking workflow in which multiple elements have to coordinate transactionally.

System testing evauated full user journeys from flight search up to book. This confirmation enables seamless functionality across all elements.

User acceptability testing recruited ten users who performed realistic booking scenarios, thus providing qualitative usability and efficiency of workflow feedback [16, 22].

The test setup utilized Django's TestCase subclass for database—dependent tests and SimpleTestCase for logic-only validation, such that test execution without compromising data integrity [7].

5.1 Test Cases

5.1.1 Unit Tests

Django's built-in testing framework was used:

```
from django.test import TestCase
from .models import Flight

class FlightModelTest(TestCase):
    def test_flight_creation(self):
        flight = Flight.objects.create(
            flight_number='AA101',
            origin='Dhaka',
            destination='Chittagong',
            base_price=5000.00
        )
        self.assertEqual(flight.flight_number, 'AA101')
```

```
13          self.assertTrue(flight.available_seats > 0)
```

Sample Unit Test

5.1.2 Test Results Summary

Table 5.1: Testing Results

Test Category	Total Tests	Passed	Failed
Unit Tests	45	43	2
Integration Tests	20	19	1
System Tests	15	15	0
Security Tests	10	10	0
Performance Tests	8	7	1
Total	**98**	**94**	**4**

5.2 Performance Testing

Load testing was conducted using Apache JMeter:

Table 5.2: Performance Test Results

Operation	Users	Avg Response (ms)	Success Rate
Homepage Load	100	450	100%
Flight Search	100	1,200	99.5%
Booking Creation	50	2,500	98%
Payment Process	50	3,000	97%

5.3 Security Testing

Security tests verified:

- SQL injection attempts blocked by Django ORM
- XSS attacks prevented through template escaping
- CSRF protection working on all forms
- Password storage using secure hashing
- Session hijacking prevention

5.4 User Acceptance Testing

Ten users tested the system with feedback:

- 90% found the interface intuitive

- 85% completed booking within 5 minutes
- Suggestions included better mobile layout and clearer error messages

Implementation

6.1 Development Environment

6.1.1 Hardware Requirements

- Development Machine: Intel Core i5, 8GB RAM, 256GB SSD
- Testing Server: 2 vCPU, 4GB RAM (Cloud VM)

6.1.2 Software Stack

- **Operating System:** Ubuntu 22.04 LTS
- **Python:** Version 3.9.x
- **Django:** Version 4.2 LTS
- **PostgreSQL:** Version 14
- **Frontend:** HTML5, CSS3, Bootstrap 5, JavaScript
- **Version Control:** Git with GitHub

6.2 Django Project Structure

```
airline_booking/
|-- airline_booking/
|   |-- settings.py      # Project settings
|   |-- urls.py          # URL routing
|   |-- wsgi.py          # WSGI configuration
|-- flights/
|   |-- models.py        # Flight models
|   |-- views.py         # Flight views
|   |-- forms.py         # Flight forms
|   |-- admin.py         # Admin configuration
|-- bookings/
|   |-- models.py        # Booking models
|   |-- views.py         # Booking logic
|-- users/
|   |-- models.py        # User models
|   |-- views.py         # Authentication
|-- templates/           # HTML templates
|-- static/              # CSS, JS, images
|-- requirements.txt     # Dependencies
|-- manage.py           # Django management
```

6.3 Key Implementation Features

6.3.1 User Authentication

Django's built-in authentication system was extended to support multiple roles:

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

class User(AbstractUser):
    ROLE_CHOICES = [
        ('passenger', 'Passenger'),
        ('agent', 'Travel Agent'),
        ('admin', 'Administrator'),
    ]
    role = models.CharField(
        max_length=20,
        choices=ROLE_CHOICES,
        default='passenger'
    )
    phone = models.CharField(max_length=15)
    date_of_birth = models.DateField(null=True)

    def is_agent(self):
        return self.role == 'agent'

    def is_admin(self):
        return self.role == 'admin'
```

```python
    def is_agent(self):
        return self.role == 'agent'

    def is_admin(self):
        return self.role == 'admin'
```

Custom User Model

6.3.2 Flight Search Implementation

```python
from django.db.models import Q
from django.shortcuts import render
from .models import Flight

def search_flights(request):
    if request.method == 'POST':
        origin = request.POST.get('origin')
        destination = request.POST.get('destination')
        date = request.POST.get('date')

        flights = Flight.objects.filter(
            Q(origin__icontains=origin) &
            Q(destination__icontains=destination) &
            Q(departure_time__date=date) &
            Q(available_seats__gt=0)
        ).order_by('departure_time')

        return render(request, 'flights/results.html',
                      {'flights': flights})

    return render(request, 'flights/search.html')
```

Flight Search View

6.3.3 Booking Process

The booking process uses Django sessions to maintain state across multiple steps:

```python
from django.db import transaction
from django.shortcuts import redirect
import random
import string

def create_booking(request):
    if request.method == 'POST':
        try:
            with transaction.atomic():
                # Generate unique PNR
                pnr = ''.join(random.choices(
                    string.ascii_uppercase + string.digits,
                    k=6
                ))

                # Create booking
                booking = Booking.objects.create(
                    pnr=pnr,
                    user=request.user,
                    flight_id=request.POST['flight_id'],
                    total_amount=calculate_fare(request)
                )

                # Update seat availability
                flight = booking.flight
                flight.available_seats -= 1
                flight.save()

                # Send confirmation email
                send_booking_email(booking)

                return redirect('booking_success', pnr=pnr)

        except Exception as e:
            # Handle errors and rollback
            return render(request, 'error.html',
                          {'error': str(e)})
```

Booking Creation

6.3.4 Security Implementation

Django's security features were utilized throughout:

- CSRF tokens on all forms
- Password hashing using PBKDF2
- SQL injection prevention through ORM
- XSS protection via template escaping

6.4 Frontend Implementation

The frontend uses Bootstrap 5 for responsive design:

```
<div class="container mt-5">
  <form method="POST" action="{% url 'search_flights' %}">
    {% csrf_token %}
    <div class="row">
      <div class="col-md-3">
        <input type="text" class="form-control"
               name="origin" placeholder="From" required>
      </div>
      <div class="col-md-3">
        <input type="text" class="form-control"
               name="destination" placeholder="To" required>
      </div>
      <div class="col-md-3">
        <input type="date" class="form-control"
               name="date" required>
      </div>
      <div class="col-md-3">
        <button type="submit" class="btn btn-primary">
          Search Flights
        </button>
      </div>
    </div>
  </form>
</div>
```

Responsive Flight Search Form

Challenges and Solutions

7.1 Technical Challenges

7.1.1 Concurrent Booking Handling

Challenge: Multiple users booking the same seat simultaneously.
Solution: Implemented database-level locking using Django's select_for_update() to ensure only one user can book a specific seat.

7.1.2 Session Management

Challenge: Maintaining booking state across multiple pages.
Solution: Used Django sessions to store temporary booking data with automatic cleanup after completion.

7.1.3 Performance Optimization

Challenge: Slow flight search queries with large datasets.
Solution: Added database indexes on frequently queried fields and implemented query optimization using select_related() and prefetch_related().

7.2 Design Challenges

7.2.1 User Interface Complexity

Challenge: Balancing functionality with simplicity.
Solution: Adopted progressive disclosure, showing advanced options only when needed.

7.2.2 Mobile Responsiveness

Challenge: Complex forms on small screens.
Solution: Redesigned forms using Bootstrap's grid system and collapsible sections.

7.3 Implementation Challenges

7.3.1 Email Delivery

Challenge: Sending confirmation emails reliably.
Solution: Configured Django to use SMTP with Gmail for development and recommended SendGrid for production.

7.3.2 PNR Generation

Challenge: Ensuring unique PNR codes.
Solution: Combined random generation with database uniqueness checks retrying.

7.4 Conclusion

7.4.1 Immediate Enhancements

1. **Real Payment Integration:** Integrate with established and secure payment gateways like Stripe, PayPal, or local options such as bKash to handle real-time financial transactions securely.
2. **Advanced Search Filters:** Enhance the search functionality by adding more granular filters, including options for specific airlines, number of layovers, duration of layovers, and preferred departure/arrival times.
3. **Seat Map Visualization:** Implement an interactive seat selection feature that displays a visual layout of the aircraft, allowing users to choose their preferred seats during the booking process.
4. **Multi-language Support:** To cater to a broader audience, particularly local users, integrate multi-language support, starting with a complete translation of the interface into Bengali.
5. **Mobile Application:** Develop native Android and iOS applications to provide a seamless and optimized user experience on mobile devices, including features like push notifications and offline access to e-tickets.
6. **User Authentication and Profiles:** Implement a comprehensive user account system where passengers can register, manage their personal information, view booking history, and save passenger details for faster future bookings.
7. **Email and SMS Notifications:** Integrate a notification service to automatically send booking confirmations, e-tickets, flight reminders, and real-time updates regarding delays, cancellations, or gate changes.
8. **Admin Dashboard:** Develop a robust administrative dashboard for system managers to oversee operations, including managing flight schedules, viewing booking statistics, handling user queries, and generating reports.

7.4.2 Long-term Vision

1. **AI-Powered Recommendations:** Implement a machine learning model to provide personalized travel suggestions, such as recommending destinations based on a user's search history or offering flight deals tailored to their preferences.

2. **Price Prediction Module:** Develop a data-driven feature that analyzes historical pricing data to forecast future ticket price fluctuations, advising users on the optimal time to purchase tickets for cost savings.
3. **Integration with Ancillary Services:** Expand the system's capabilities by integrating with third-party APIs for booking hotels, rental cars, travel insurance, and tour packages, transforming it into an all-in-one travel solution.
4. **Loyalty Program:** Introduce a frequent flyer or loyalty program to enhance customer retention. Users could earn points for each booking, which can be redeemed for discounts, upgrades, or other travel-related perks.
5. **AI Chatbot for Customer Support:** Deploy an intelligent, 24/7 chatbot to handle common customer service inquiries, assist with the booking process, and provide instant support, thereby reducing the workload on human support staff.
6. **Dynamic Holiday Packages:** Create a feature that allows users to build dynamic travel packages, combining flights, accommodations, and activities into a single, customizable booking at a bundled price.

References

[1] Apache Software Foundation. *Apache JMeter User's Manual*, 2024. https://jmeter.apache.org/usermanual/

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*, 3rd edition, Addison-Wesley Professional, 2012.

[3] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[4] Bootstrap Team. *Bootstrap Documentation: Build Fast, Responsive Sites*, 2024. https://getbootstrap.com/docs/5.3/

[5] P. P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[6] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[7] Django Software Foundation. *Django Documentation: The Web Framework for Perfectionists with Deadlines*, 2024. https://docs.djangoproject.com/

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. Technical Report RFC 2616, IETF, 2002.

[9] R. T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[10] H. García-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*, 2nd edition, Pearson, 2019.

[11] Git Development Community. *Git Documentation*, 2024. https://git-scm.com/doc

[12] GitHub, Inc. *GitHub Documentation*, 2024. https://docs.github.com/

[13] Mozilla Developer Network. *MDN Web Docs: Resources for Developers, by Developers*, 2024. https://developer.mozilla.org/

[14] OWASP Foundation. *OWASP Top Ten 2021: The Ten Most Critical Web Application Security Risks*, 2021. https://owasp.org/Top10/

[15] PostgreSQL Global Development Group. *PostgreSQL 14 Documentation*, 2024. https://www.postgresql.org/docs/14/

[16] R. S. Pressman and B. R. Maxim. *Software Engineering: A Practitioner's Approach*, 8th edition, McGraw-Hill Education, 2014.

[17] Python Software Foundation. *Python 3.9 Documentation*, 2024. https://docs.python.org/3.9/

[18] R. Ramakrishnan and J. Gehrke. *Database Management Systems*, 3rd edition, McGraw-Hill Education, 2003.

[19] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[20] Selenium Project. *Selenium WebDriver Documentation*, 2024. https://selenium-python.readthedocs.io/

[21] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*, 10th edition, Wiley, 2019.

[22] I. Sommerville. *Software Engineering*, 10th edition, Pearson Education, 2016.

Project Report Libra... 4 days ago

to me, Khalid

Dear Student,

Your plagiarism result with percentage (20%) and AI (less than 20%) is attached. **Please go through the attachment and guidelines, then take the necessary steps.**