**Test Cases for Flight Booking Django Application**

**Test Case 1: User Registration**

**Description:** Verify that users can register successfully.

* **Input:** Provide valid user details (username, email, password).
* **Expected Result:** User account is created and a confirmation message is displayed.

**Description:** Verify validation errors during registration.

* **Input:** Provide invalid details (e.g., missing required fields, weak password).
* **Expected Result:** Appropriate error messages are displayed.

**Test Case 2: User Login**

**Description:** Verify successful login with valid credentials.

* **Input:** Valid username and password.
* **Expected Result:** User is redirected to the homepage.

**Description:** Verify error on login with invalid credentials.

* **Input:** Invalid username or password.
* **Expected Result:** Login fails and error message is shown.

**Test Case 3: Flight Search**

**Description:** Verify that users can search for flights.

* **Input:** Provide valid search criteria (e.g., source, destination, date).
* **Expected Result:** Matching flights are displayed.

**Description:** Verify no results when no flights match criteria.

* **Input:** Invalid or unavailable source/destination.
* **Expected Result:** A message indicating no results is displayed.

**Test Case 4: Booking Tickets**

**Description:** Verify successful booking of a one-way ticket.

* **Input:** Select flight, fill in passenger details, proceed with payment.
* **Expected Result:** Booking confirmation is displayed with ticket details.

**Description:** Verify successful booking of a round-trip ticket.

* **Input:** Select outbound and return flights, fill in passenger details, proceed with payment.
* **Expected Result:** Booking confirmation is displayed with ticket details.

**Description:** Verify validation errors during booking.

* **Input:** Leave required fields blank or provide invalid details.
* **Expected Result:** Error messages are displayed.

**Test Case 5: Viewing Bookings**

**Description:** Verify that users can view their bookings.

* **Input:** Log in and navigate to bookings page.
* **Expected Result:** List of booked tickets (confirmed and canceled) is displayed.

**Test Case 6: Canceling Tickets**

**Description:** Verify that users can cancel their tickets.

* **Input:** Select a booked ticket and choose the cancel option.
* **Expected Result:** Ticket status is updated to "canceled" and a confirmation message is displayed.

**Test Case 7: Generating PDF Tickets**

**Description:** Verify that users can download their tickets as PDF.

* **Input:** Select a booked ticket and choose the download option.
* **Expected Result:** A PDF file with ticket details is downloaded.

**Test Case 8: Responsive Design**

**Description:** Verify that the web application is mobile responsive.

* **Input:** Access the application on various devices (desktop, tablet, mobile).
* **Expected Result:** Pages adjust and display correctly across devices.

**Test Case 9: AJAX Functionality**

**Description:** Verify that the "as-you-type" search works.

* **Input:** Type keywords into the search field.
* **Expected Result:** Suggestions appear dynamically without refreshing the page.

**Test Case 10: Payment Processing**

**Description:** Verify successful completion of payment.

* **Input:** Provide valid payment details.
* **Expected Result:** Payment is processed, and user is redirected to confirmation page.

**Description:** Verify error during payment.

* **Input:** Provide invalid payment details.
* **Expected Result:** Payment fails, and an error message is displayed.

**Project Specifications and Requirements**

**1. Project Overview**

The Flight Booking Django application allows users to search, book, cancel, and manage flight tickets. It provides a responsive interface, dynamic search functionality, and the ability to generate PDF tickets. This project is the capstone for CS50's Web Programming with Python and JavaScript.

**2. Functional Requirements**

**User Management**

* Users can create an account.
* Users can log in and log out.

**Flight Search**

* Users can search for flights using source, destination, and date.
* Results should be filtered and displayed dynamically.

**Ticket Booking**

* Users can book one-way or round-trip tickets.
* Users can specify passenger details during booking.
* Payment must be completed to confirm a booking.

**Ticket Management**

* Users can view, cancel, and download their booked tickets.
* Canceling a ticket updates its status.
* Tickets should include flight details, passenger details, and a unique reference number.

**Payment**

* Payment processing should be secure and reliable.
* Users should receive confirmation after successful payment.

**Admin Panel**

* Admins can manage flights, users, and bookings through the Django admin panel.

**3. Non-Functional Requirements**

* **Performance:** Pages should load quickly, with dynamic content updating via AJAX.
* **Scalability:** The system should handle multiple users and bookings simultaneously.
* **Responsiveness:** The application must work seamlessly on desktops, tablets, and mobile devices.
* **Security:** User data and payments must be securely handled.

**4. Technical Specifications**

* **Backend:** Python, Django.
* **Frontend:** HTML, CSS, JavaScript (with AJAX).
* **Database:** SQLite (default Django database).
* **PDF Generation:** xhtml2pdf library.
* **Deployment:** Compatible with Heroku.

**5. Installation Instructions**

1. Install Python 3.9.
2. Clone the repository and navigate to the project directory.
3. Install dependencies: py -m pip install -r requirements.txt.
4. Apply migrations: py manage.py makemigrations and py manage.py migrate.
5. (Optional) Create a superuser: py manage.py createsuperuser.
6. Start the development server: py manage.py runserver.
7. Open the application in a web browser at 127.0.0.1:8000.

**6. Constraints**

* The application currently uses SQLite, which may not be ideal for high-scale production.
* Payments are simulated and should be integrated with a live payment gateway for production use.

**7. Assumptions**

* Users have access to a modern web browser.
* Users provide valid email addresses and payment information.

**8. Future Enhancements**

* Integration with real-world APIs for live flight data.
* Support for multiple payment gateways.
* Enhanced reporting and analytics for admins.
* Push notifications for ticket updates and reminders.