# Flight Booking System Assignment

## Assignment Overview

Design a Flight Booking System that allows users to search and book flights, while implementing secure authentication and email verification. Your team is required to implement everything listed and discussed in the requirements below.

## Technical Requirements

### Backend (Node.js + Express + MongoDB)

- RESTful API design.
- JWT-based authentication.
- Email verification with 6-digit code stored in MongoDB (expires in 10 minutes).
- Secure password hashing using bcrypt.
- Flight search endpoint using MongoDB filtering (no caching).
- Protected routes for bookings.

### Frontend (React)

- Registration page.
- Login page.
- Email verification page.
- Flight search page (Home Page)
- Booking history page.
- Proper error handling and loading states.

### Database Design

- Suggested Collections/Tables:
- Users
- Flights
- Bookings

### Users Collection

- Name
- Email (unique)
- Password (hashed)
- isVerified (boolean)
- verificationCode (String)
- verificationCodeExpires (Date)

Required CRUD Operations: Create, Read, Update (for user info and verification)

## Flights Collection

- Flight Number (unique identifier)
- From (departure city/airport)
- To (arrival city/airport)
- Date (flight date/time)
- Total Seats
- Available Seats
- Price (ticket price)

Required CRUD Operations: Create, Read, Update, Delete

## Bookings Collection (MongoDb)

- User ID (reference to Users collection)
- Flight ID (reference to Flights collection)
- Booking Date
- Number of Seats (optional)
- Total Price (optional)
- Status (optional: confirmed/canceled)

Required CRUD Operations: Create, Read, Update (status)

## Tips for Students

Build the project incrementally. Do not start with everything at once. Follow this logical order:

### 1. Authentication + JWT

Start by implementing user registration and login.

- Hash passwords using bcrypt.
- Generate a JWT token after successful login.
- Protect routes that require authentication (e.g., booking routes).
- Make sure the login works correctly before moving to the next step.

---

### 2. Email Verification Code (stored in MongoDB)

After registration:

- Generate a 6-digit verification code.
- Store it in the user document inside MongoDB:
    - verificationCode
    - verificationCodeExpires (10 minutes from creation time).
- Send the code to the user's email.

When the user enters the code:

- Check if the code matches.
- Check that it has not expired.
- Update isVerified = true.
- Clear the verification code fields after successful verification.

---

## 3. Flights CRUD in Database

Create full CRUD operations for flights:

- Add new flights.
- View all flights.
- Update flight details.
- Delete flights.

Test all endpoints using Postman before connecting to the frontend.

---

## 4. Flight Search using MongoDB Filtering

Implement a search endpoint that filters flights by:

- From
- To
- Date

Use MongoDB queries (e.g., find() with filters).
No caching is required.

---

## 5. Booking Logic with availableSeats Update

When a user books a flight:

- Check if enough seats are available.
- Reduce availableSeats accordingly.
- Create a booking record linked to the user and the flight.

Ensure that users can only see their own bookings.

## 6. Frontend Implementation

After completing and testing all backend APIs:

- Build the React pages.
- Connect frontend to backend APIs.
- Handle loading states and error messages properly.

## Email Testing

Use **Mailtrap** for email testing purposes.
These services allow you to simulate sending emails without using a real email provider.You will be able to view the verification email inside their dashboard.

## Grading Criteria (Total: 100%)

| Criteria | Marks |
|---|---|
| **Backend Implementation** (RESTful APIs, Authentication, Email Verification, Flights CRUD, Booking APIs) | 40 |
| **Business Logic & Data Integrity** (Flight filtering, booking logic, availableSeats update, validation, protected routes) | 15 |
| **Frontend Implementation (React)** (Pages, routing, API integration, proper state management, loading and error handling) | 15 |
| **API Testing (Postman Collection)** (Well-organized collection, meaningful test cases, environment variables, validation of system behavior) | 5 |
| **GitHub & Version Control** (Consistent commits, clear commit messages, proper repository structure and README) | 5 |
| Individual Understanding | 20 |

## Group Rules:

- Assignment must be completed in groups of 2 students.
- Both students must fully understand the entire project (Backend + Frontend).
- During evaluation, any student may be asked to explain any part of the code.

## Deliverable Discussion/Response Policy:

- If a student is asked a question during evaluation and does not answer, **5 points will be deducted**.
- If the student fails to answer a second question, **an additional 5 points will be deducted**.
- If the student fails to answer a third question, **the student will receive 0 for the Individual Understanding mark**.
- If the student fails to answer a fourth question, the student will receive 0 for the deliverable itself, regardless of the team's mark

## Important Rules:

1. No late submission allowed.
2. The deliverable version uploaded to the **Google Classroom** assignment is the one that will be evaluated. If it's a team project, the team representative can just upload the deliverable for other members. The GitHub version will serve only as a reference to track progress.
3. **Version Control:** All projects must use GitHub to track progress. Commits should reflect contributions from all members. GitHub is mandatory in the evaluation process
4. **Academic Integrity Reminder**: Directly copying or pasting content from AI tools (such as ChatGPT or others) into your course deliverables is considered academic misconduct (Up to cheating). Your submitted work should reflect your own effort, writing style, creativity, and uniqueness. You may use AI tools to support your learning or improve your understanding, but the final content must be genuinely your own. All submissions/deliverables are subject to plagiarism and AI-detection checks. If there is reasonable suspicion of AI-generated or copied content, the lecturer/TA reserves the right to:
5. Understanding deliverables and their details will be evaluated individually for each team member.

## GitHub Repository & Version Control Requirement

Each team must create and maintain a dedicated GitHub repository for this project. All development work must be committed regularly throughout the project period. The repository will be used to track progress, evaluate development consistency, and verify individual contributions.

Teams are expected to make meaningful, incremental commits that reflect continuous development rather than uploading the entire project at once. Commit messages should be clear and descriptive. The repository must include a well-organized folder structure, source code, assets (where applicable), and a README file explaining how to run the project.

Failure to demonstrate a consistent development history through GitHub commit activity may impact the project evaluation.