

# **AN INTERNSHIP REPORT ON FlightFinder Navigating Your Air Travel Options.**

**By**

**Team ID : LTVIP2025TMID43395**

**Team Size : 4**

**Team Leader : Bharathi Madiki**

**Team member : Pabolu Sudheer**

**Team member : Paina Durga Syamalarao**

**Team member : Pangam Satwika**

**College name:**

**Aditya college of engineering & technology, surampalem**

# ABSTRACT

This document presents *FlightFinder*, a digital solution tailored to streamline and modernize the flight booking experience. The platform is built to cater to users who demand convenience, speed, and personalized travel options in an increasingly mobile-first world. Designed with both user experience and system efficiency in mind, FlightFinder bridges the gap between travelers and airline services through an intuitive and responsive web interface.

The technical architecture behind the application consists of a robust frontend interface developed using modern web technologies, a secure and efficient backend powered by RESTful APIs, and a scalable, cloud-ready database that stores critical information such as user profiles, flight schedules, and booking records. The system architecture also includes user authentication modules, role-based access for administrators, and dynamic data handling to ensure a responsive and reliable user experience.

A real-world user scenario featuring John, a frequent business traveler, demonstrates the platform's key strengths—seamless flight discovery based on personal preferences, intuitive seat selection with live seat maps, and secure payment processing via integrated gateways. The application also supports loyalty program integration, real-time flight updates, and multi-device access, allowing travelers to manage their bookings on the go.

On the administrative side, the backend dashboard allows system admins to manage flights, monitor bookings, and maintain user data with ease. The application is designed to scale with increased user demand and supports modular expansion for future features such as hotel booking, travel insurance, and package deals. By combining user-centric design with strong technical foundations, *FlightFinder* delivers a comprehensive, secure, and efficient solution for modern air travel needs.

# FlightFinder: Navigating Your Air Travel Options

## INTRODUCTION:

FlightFinder is an advanced web-based flight booking application designed to enhance and simplify the air travel experience. Whether you're a frequent flyer or an occasional traveler, FlightFinder helps you find the most suitable flights tailored to your preferences. By leveraging a clean user interface, real-time flight data, secure transactions, and customizable options, the app ensures that booking a flight is not just a task — it's a seamless experience.

With smart features like interactive seat selection, airline filters, loyalty program integration, and secure payment gateways, FlightFinder transforms the way users plan and manage their journeys. It enables users to compare multiple flight options from various airlines, apply filters based on convenience and comfort, and make informed travel decisions. The intuitive design supports quick navigation through each stage of the booking process—from search to confirmation—while providing flexibility in modifying or canceling bookings when needed.

In addition, FlightFinder offers a robust back-end system to support administrative tasks such as managing flights, monitoring user activity, and handling booking data, making it a powerful tool for both travelers and system administrators. With scalability in mind, the platform is designed to handle high volumes of users and flight data efficiently. Overall, FlightFinder is not just a flight booking tool; it is a comprehensive travel companion tailored for the modern digital traveler.

## KEY FEATURES:

### 1. User-Friendly Interface

- Clean and intuitive UI for effortless flight booking.
- Responsive design optimized for smartphones and desktops.

### 2. Flight Search & Filters

- Powerful search engine for flights based on destination, date, and class.
- Filters for airline preference, direct flights, departure time, and more.

### 3. Personalized Experience

- Save traveler preferences and frequent flyer information.
- Integration with loyalty programs and preferred airlines.

### 4. Interactive Seat Selection

- Real-time seat availability visualization.
- Choose seats with extra legroom or window preferences.

### 5. Secure Payment Gateway

- Integrated and encrypted payment system.
- Supports multiple payment options for user convenience.
- 6. Booking Confirmation & Itinerary**
  - Instant e-ticket generation and itinerary sharing.
  - In-app access to booking history and travel documents.
- 7. Admin Dashboard & Management**
  - Admin login and authentication.
  - Tools to manage users, flights, and bookings efficiently.
- 8. Scalable Backend Architecture**
  - RESTful APIs for user, flight, and booking management.
  - Modular backend structure for performance and reliability.
- 9. Real-Time Assistance and Updates**
  - Notification system for booking confirmation, delays, or cancellations.
  - Help center integration for customer support.
- 10. Data Security and Privacy**
  - Secure authentication for both users and administrators.
  - GDPR-compliant data storage and handling policies.

## **DESCRIPTION :**

FlightFinder is designed as a modern travel companion for individuals and business professionals alike. At its core, the app simplifies the travel planning process with a powerful search engine, smart filtering options, and a seamless booking system. It's built to reduce the hassle commonly associated with traditional flight booking platforms.

For example, in a practical scenario, a business traveler named John needs to book a round-trip flight from New York City to Paris. With FlightFinder, he can input his details, apply preferred filters like business class and direct flights, and view all suitable flight options in seconds. After selecting his preferred flight, he uses the interactive seat map to book a window seat with extra legroom, proceeds to payment via a secure gateway, and instantly receives his e-ticket and travel itinerary.

From a technical standpoint, FlightFinder is structured into three main layers:

- **Frontend:** Delivers an interactive interface for searching and booking, including user login and profile management.

- **Backend:** Handles all core logic and operations through API endpoints for users, bookings, and flight data.
- **Database:** Manages persistent storage of users, available flights, booking details, and admin data.

This layered architecture ensures scalability, reliability, and modularity, making the system adaptable for future enhancements like hotel bookings, travel insurance, or multi-city travel planning.

## USER SCENARIO: BOOKING A BUSINESS TRIP WITH FLIGHTFINDER

**User:** John – Frequent traveler and business professional

**Purpose:** Booking a round-trip flight from New York City to Paris for a business conference

### Step-by-Step Journey with the App

#### 1. Launching the App

John opens the *FlightFinder* app on his smartphone to begin booking his upcoming business trip.

#### 2. Entering Travel Details

He inputs the following information:

- **Departure City:** New York City
- **Destination:** Paris
- **Departure Date:** April 10th
- **Return Date:** April 15th
- **Travel Class:** Business Class
- **Number of Passengers:** 1

#### 3. Flight Search Results

The app quickly retrieves a list of available flight options based on John's preferences.

Results include various airlines, pricing, total flight duration, direct and layover options, and departure times.

#### 4. Applying Filters

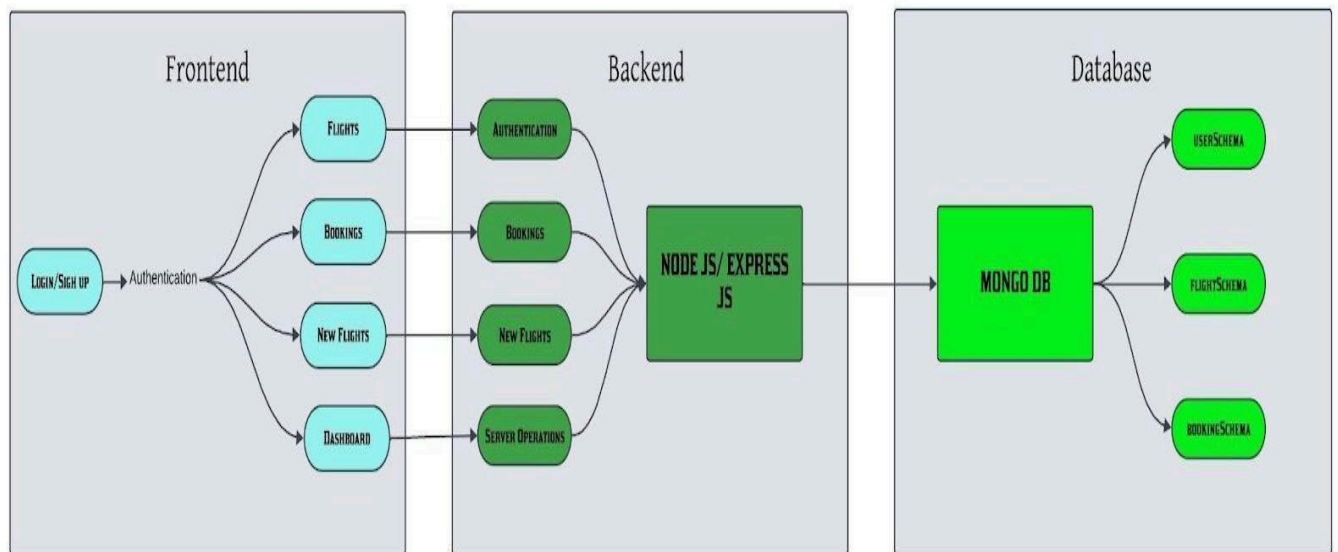
To refine his search, John uses built-in filters to:

- View **only direct flights**

- Select **convenient departure times**
  - Choose a **preferred airline** based on past experiences and loyalty programs
5. **Seat Selection**
- After selecting his flight, John proceeds to pick a seat.
- The app displays an **interactive seat map** of the business class cabin, showing all available seats.
- John chooses a **window seat with extra legroom** for added comfort.
6. **Secure Payment**
- John securely enters his payment details through the app's **integrated and encrypted payment gateway**.
- The app processes the transaction and immediately generates:
- A **booking confirmation**
  - An **e-ticket**
  - A detailed **itinerary**
7. **Conclusion**
- This scenario illustrates how *FlightFinder* streamlines the entire booking process—from flight search and seat selection to secure payment—providing users like John with a fast, reliable, and personalized travel booking experience.

## TECHNICAL ARCHITECTURE

The technical architecture of *FlightFinder* is designed to ensure scalability, performance, and security across all components. It is divided into three primary layers: **Frontend**, **Backend**, and **Database**.



## 1. Frontend

The frontend represents the **user-facing interface** of the application. It includes all the interactive components that allow users to search, book, and manage their flights.

### Key Components:

- **User Authentication:** Enables user login, registration, and session management.
- **Flight Search:** Allows users to search flights based on destination, dates, class, and number of passengers.
- **Booking Interface:** Facilitates flight selection, seat selection, and confirmation of bookings.

## 2. Backend

The backend manages the **core business logic** and handles communication between the frontend and database through RESTful API endpoints.

### Key Components:

- **User API:** Manages user data, authentication, and profile updates.
- **Flight API:** Handles flight listings, search queries, and flight details.
- **Booking API:** Manages booking creation, seat selection, and payment integration.
- **Admin API & Authentication:** Provides secure access for admin users to manage content and operations.
- **Admin Dashboard:** A web interface for administrators to monitor users, flights, and bookings.

## 3. Database

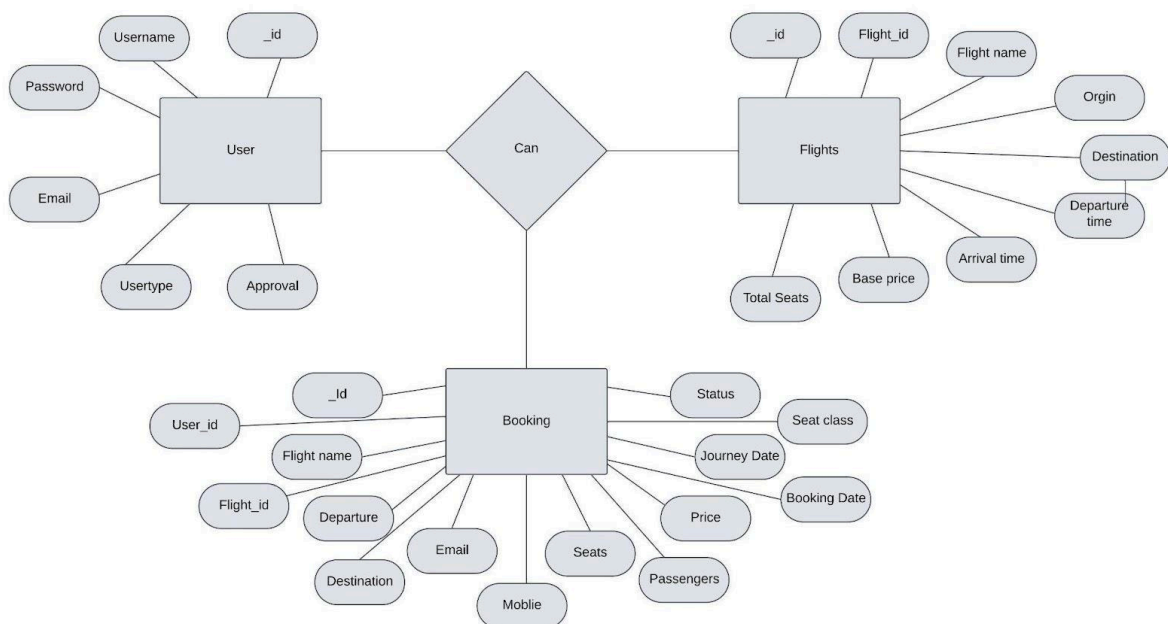
The database is responsible for the **persistent storage** of all application data.

### Key Collections/Tables:

- **Users:** Stores user credentials, profiles, and booking history.
- **Flights:** Contains flight schedules, airline details, pricing, and availability.
- **Flight Bookings:** Records booking transactions, seat selections, and payment details.

This architecture ensures that *FlightFinder* is modular, secure, and scalable—capable of supporting both end-users and administrators with real-time access and control over flight booking operations.

## ER DIAGRAM



The flight booking ER-diagram represents the entities and relationships involved in a flight booking system. It illustrates how users, bookings, flights, passengers, and payments are interconnected. Here is a breakdown of the entities and their relationships:

**USER:** Represents the individuals or entities who book flights. A customer can place multiple bookings and make multiple payments.

**BOOKING:** Represents a specific flight booking made by a customer. A booking includes a particular flight details and passenger information. A customer can have multiple bookings.



**FLIGHT:** Represents a flight that is available for booking. Here, the details of flight will be provided and the users can book them as much as the available seats.

**ADMIN:** Admin is responsible for all the backend activities. Admin manages all the bookings, adds new flights, etc.

## PRE-REQUISITES

To develop a full-stack flight booking app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

### Node.js and npm:

Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- **Download:** [Node.js Download Page](#)
- **Installation Guide:** [Package Manager Installation](#)

### MongoDB:

Set up a MongoDB database to store hotel and booking information. Install MongoDB locally using a cloud-based MongoDB service.

- **Download:** [MongoDB Community Edition](#)
- **Installation Guide:** [MongoDB Manual Installation](#)

### Express.js:

Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- **Installation command:** npm install express

### React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications

- **Setup Guide:** [Create a New React App](#)

### HTML, CSS, and JavaScript:

Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

- **Requirement:** Basic knowledge is required to customize and extend the UI.

## Database Connectivity:

Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

## Front-end Framework:

Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

## Version Control:

Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- **Download Git:** [Git Download Page](#)

## Development Environment:

Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

## SETTING UP THE FLIGHT BOOKING APP LOCALLY

To run the Flight Booking App on your local machine, begin by accessing the project repository hosted on GitHub. Use a terminal or command prompt to download a copy of the project files to your computer. After successfully downloading, open the project folder to continue with the setup.

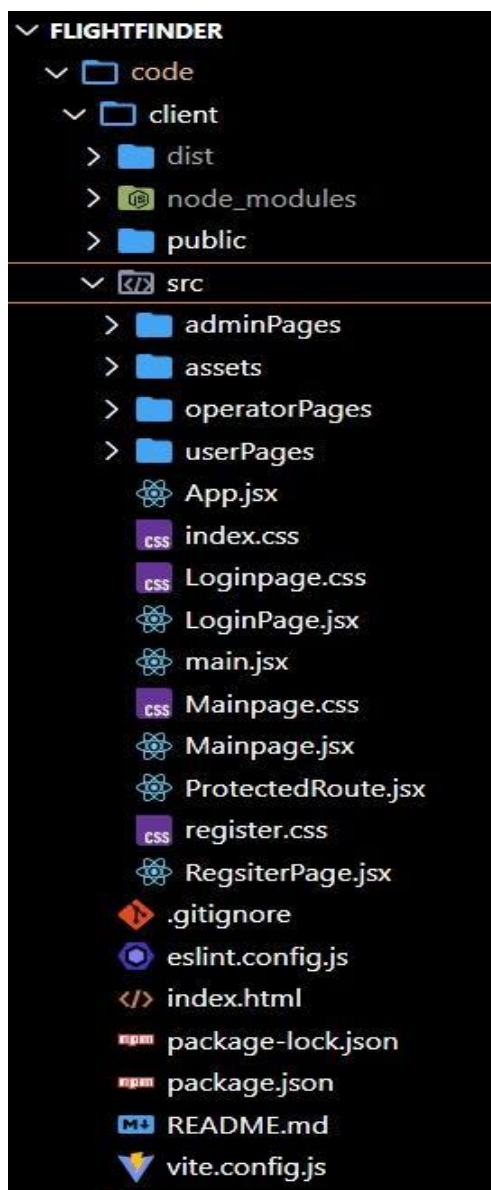
Inside the folder, install the necessary software packages required for the app to function. These packages are managed automatically and include all essential components for both the frontend and backend.

Once all dependencies are installed, launch the development server. The app is configured to run on your local system and will typically open in a browser window at the default address: <http://localhost:3000>. If needed, this port number can be changed by editing the environment configuration file.

After the server starts, open a web browser and visit the specified address. If everything has been set up correctly, you will see the homepage of the Flight Booking App, indicating that the application is running successfully on your local environment.

## PROJECT STRUCTURE

The **Flight Booking App** is divided into two main parts: the **Client** directory and the **Server** directory. Each section has its own responsibilities and technologies to handle the frontend and backend of the application.

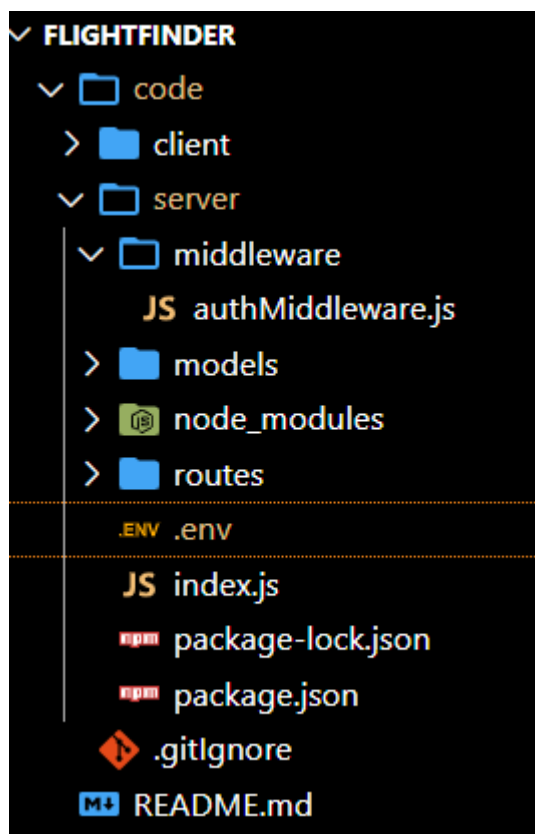


### 1. Client Directory (Frontend)

The **Client** folder contains all the files and subdirectories related to the **frontend** of the application. This part of the app is built using **React.js** and is responsible for rendering the user interface, handling user interactions, and communicating with backend APIs.

Key elements inside the Client directory include:

- Components for user authentication, flight search, booking interface, and seat selection
- Routing configuration for navigation between pages
- API utility files for making HTTP requests to the backend
- Styling files (CSS/SCSS) to design the user interface
- Assets such as images, icons, and logos used throughout the UI



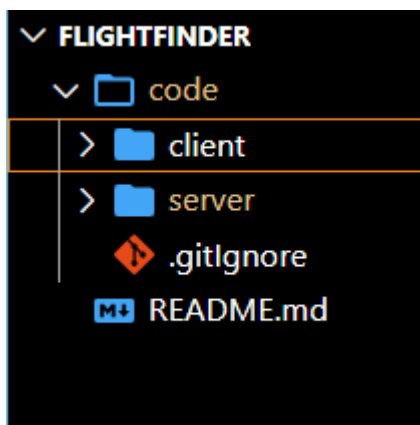
## 2. Server Directory (Backend)

The **Server** folder contains the **backend** logic of the application. It is built using **Node.js**, **Express.js**, and **MongoDB** to handle API requests, database operations, authentication, and business logic.

Key elements inside the Server directory include:

- Route handlers for users, flights, bookings, and admin operations
- Controllers that define the core logic for each API endpoint

- Models for defining MongoDB schemas (e.g., User, Flight, Booking)
- Middleware for authentication, error handling, and request validation
- Configuration files for connecting to the database and managing environment variables



## APPLICATION FLOW

- The **Flight Booking App** follows a straightforward and user-friendly flow for both users and administrators.
- **Users** begin by creating an account on the platform. Once registered, they can search for destinations and browse available flights based on their preferred dates and time of travel. After selecting a suitable flight, users can choose a specific seat, proceed with secure payment, and complete their booking. The system also allows users to view or cancel their bookings if needed.
- **Administrators**, on the other hand, have access to backend functionalities where they can manage all user bookings, add new flights and services, and monitor overall user activity on the platform to ensure smooth operation and service delivery.

## PROJECT SETUP AND CONFIGURATION

The Flight Booking App is structured into two main directories: one for the **frontend** and one for the **backend**. The setup process begins by organizing these directories and installing all necessary dependencies for both parts of the application.

### Folder Structure

To begin development from scratch, create two main folders at the root level of your project:

- **client/** – for frontend (React.js)

- **server/** – for backend (Node.js and Express)

This separation ensures a clean architecture and easy maintainability of the codebase.

```
code > client > package.json > ...
1  {
2    "name": "client",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "build": "vite build",
9      "lint": "eslint .",
10     "preview": "vite preview"
11   },
12   "dependencies": {
13     "axios": "^1.10.0",
14     "react": "^19.1.0",
15     "react-dom": "^19.1.0",
16     "react-router-dom": "^7.6.2",
17     "react-toastify": "^11.0.5"
18   },
19   "devDependencies": {
20     "@eslint/js": "^9.25.0",
21     "@types/react": "^19.1.2",
22     "@types/react-dom": "^19.1.2",
23     "@vitejs/plugin-react": "^4.4.1",
24     "eslint": "^9.25.0",
25     "eslint-plugin-react-hooks": "^5.2.0",
26     "eslint-plugin-react-refresh": "^0.4.19",
27     "globals": "^16.0.0",
28     "vite": "^6.3.5"
29   }
30 }
```

## Frontend Setup (client Folder)

Navigate to the client folder and install all required libraries to build the frontend interface of the application.

The frontend is built using:

- **React.js** – JavaScript library for building user interfaces
- **Bootstrap** – CSS framework for responsive design
- **Axios** – Promise-based HTTP client for API communication

Once these libraries are installed, a package.json file is generated, which manages the project's dependencies, scripts, and metadata.

## Backend Setup (server Folder)

```
code > server > npm package.json > ...
1  {
2    "name": "server",
3    "version": "1.0.0",
4    "main": "index.js",
5    "scripts": {
6      "test": "echo \\\"Error: no test specified\\\" && exit 1",
7      "start": "node index.js",
8      "dev": "nodemon index.js"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "type": "commonjs",
14   "description": "",
15   "dependencies": {
16     "bcrypt": "^6.0.0",
17     "bcryptjs": "^3.0.2",
18     "cors": "^2.8.5",
19     "dotenv": "^16.5.0",
20     "express": "^5.1.0",
21     "jsonwebtoken": "^9.0.2",
22     "mongoose": "^8.15.2",
23     "nodemon": "^3.1.10"
24   }
25 }
26
```

Now, move to the server folder and install the backend libraries required to build the server-side logic of the application.

The backend stack includes:

- **bcrypt** – For password hashing and user authentication
- **body-parser** – To parse incoming request bodies
- **cors** – To enable Cross-Origin Resource Sharing
- **express** – Minimal and flexible Node.js web application framework
- **mongoose** – ODM for MongoDB to define schemas and interact with the database

## FRONTEND DEVELOPMENT

The frontend of the Flight Booking App is built using **React.js** and is responsible for providing a smooth and interactive experience to users, administrators, and flight operators. The key

functionalities implemented in the frontend include user authentication, flight booking, viewing bookings, and admin operations.

The image displays two screenshots of a web application interface for 'SB Flights'. The top screenshot shows the 'Register' form at the URL 'localhost:5173/registration'. The form includes input fields for 'Name', 'Email', and 'Password', a dropdown menu for 'User', a blue 'Sign up' button, and a link 'Already registered? Login'. The bottom screenshot shows the 'Login' form at the URL 'localhost:5173/login'. This form includes input fields for 'Email' and 'Password', a dropdown menu for 'User', a blue 'Login' button, and a link 'Don't have an account Register'. Both forms are centered on a light gray background with a blue header bar containing the 'SB Flights' logo and navigation links.

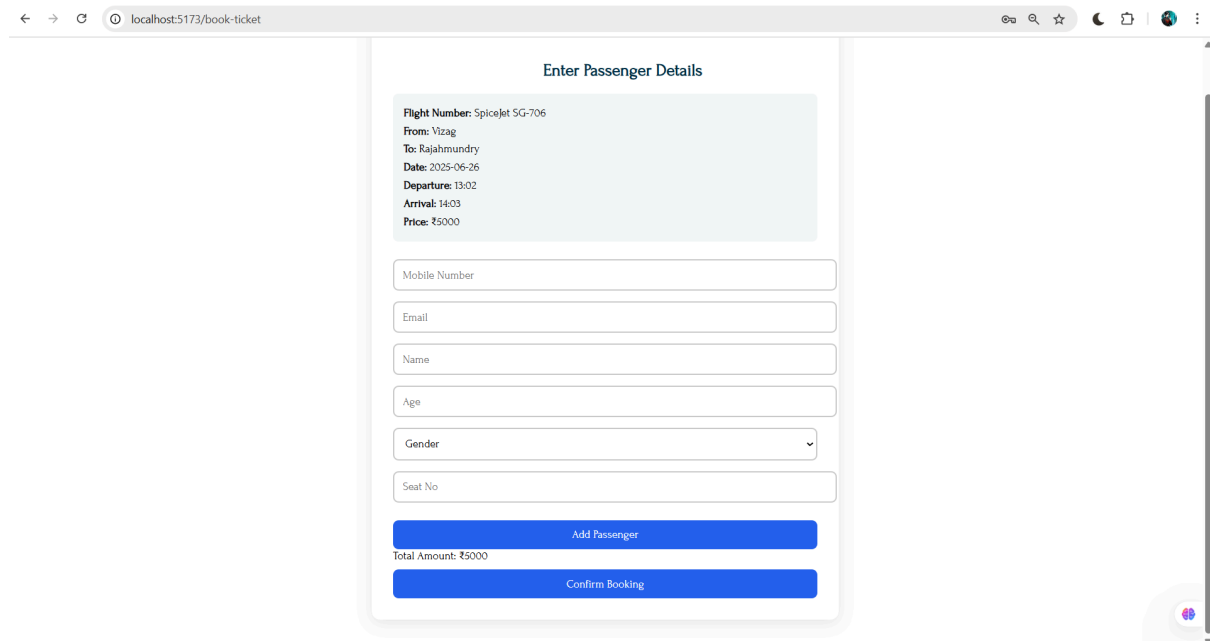
## Login / Register

A dedicated component is created to handle user authentication. This includes a form where users input their **username** and **password**. Based on the provided credentials:

- If the login matches a **regular user**, they are redirected to the user homepage.
- If the credentials belong to an **admin** or **flight operator**, they are navigated to their respective dashboards.



The login functionality is secured and integrated with the backend to validate user roles.



The screenshot shows a web browser window with the address bar displaying 'localhost:5173/book-ticket'. The main content area features a form titled 'Enter Passenger Details'. Inside the form, there is a light blue box containing flight information: 'Flight Number: SpiceJet 5G-706', 'From: Vizag', 'To: Rajahmundry', 'Date: 2025-06-26', 'Departure: 13:02', 'Arrival: 14:03', and 'Price: ₹5000'. Below this box are several input fields: 'Mobile Number', 'Email', 'Name', 'Age', 'Gender' (a dropdown menu), and 'Seat No'. At the bottom of the form, there is a blue button labeled 'Add Passenger', a text label 'Total Amount: ₹5000', and another blue button labeled 'Confirm Booking'.

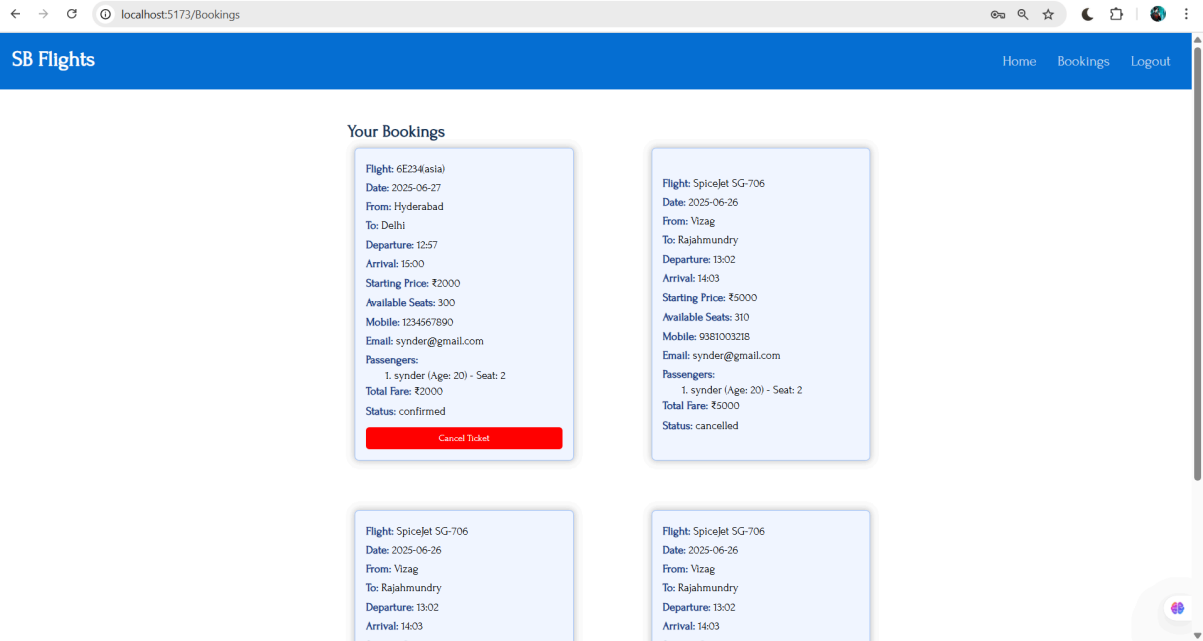
## Flight Booking (User)

Flight booking is managed using a modal component. The process begins with a **flight search** feature, where users can enter details such as:

- Departure city
- Destination
- Travel date
- Preferred class, etc.

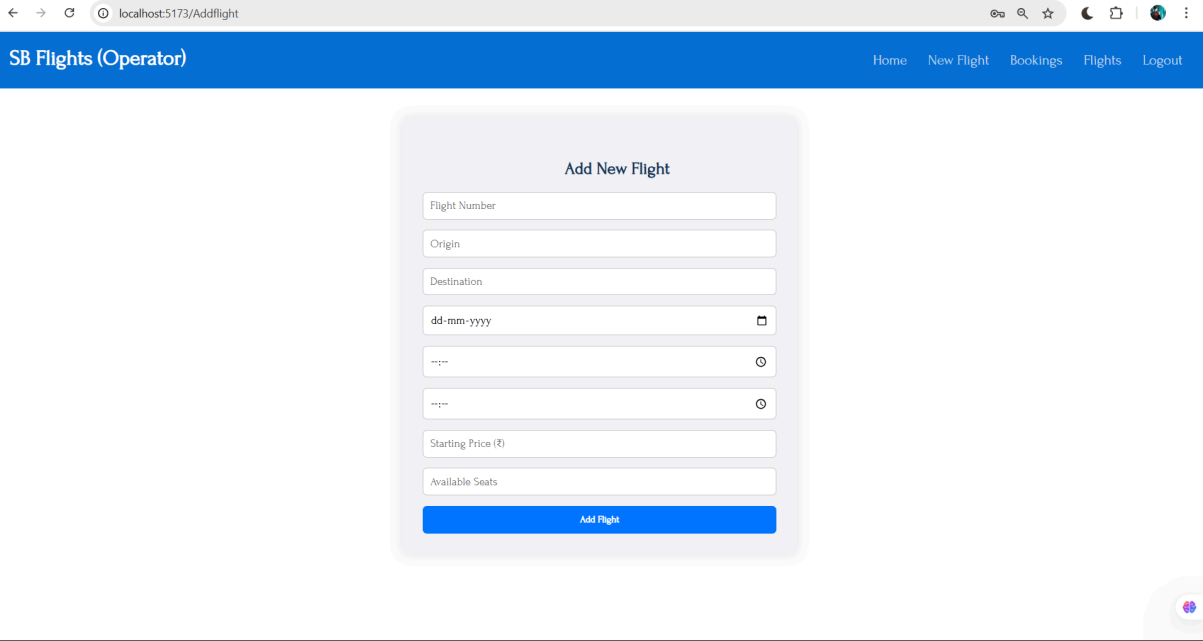
Once the inputs are provided, the app fetches a list of **available flights** from the backend. Each listed flight includes a **“Book”** button that redirects the user to the flight booking modal, where they can complete their reservation.

# Fetching User Bookings



On the **bookings page**, the application displays all the **past and current bookings** made by the user. Alongside each booking, there is an option to **cancel** the reservation if needed. This improves user control and flexibility over their travel plans.

# Add New Flight (Operator)

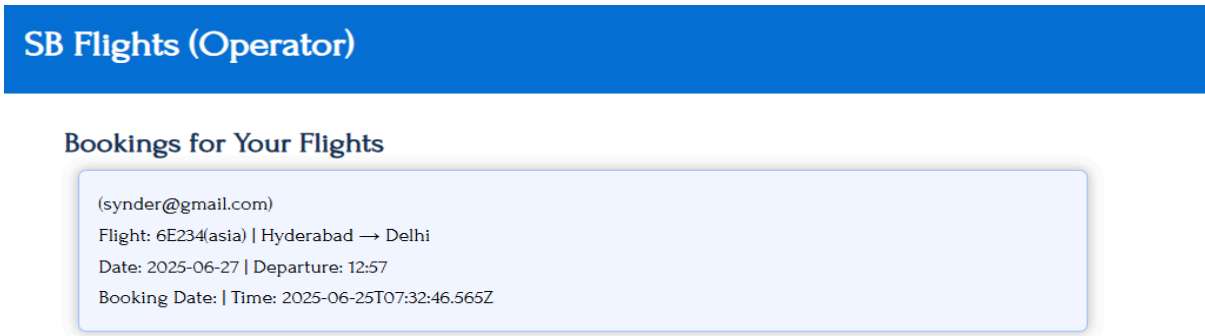
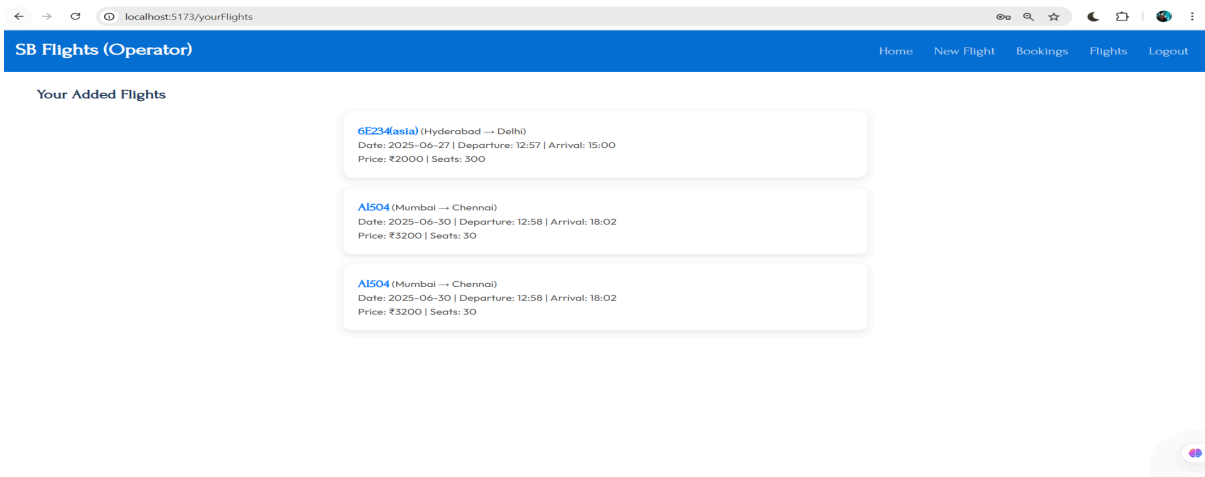


In the **Operator dashboard**, functionality is provided to **add new flights** to the system. This is achieved through a simple HTML form that collects flight information such as:

- Flight number
- Departure and arrival cities
- Date and time
- Seat availability
- Pricing

Once the form is submitted, an **HTTP request** is sent to the backend to save the new flight data into the database.

## Update Flight Details (Operator)

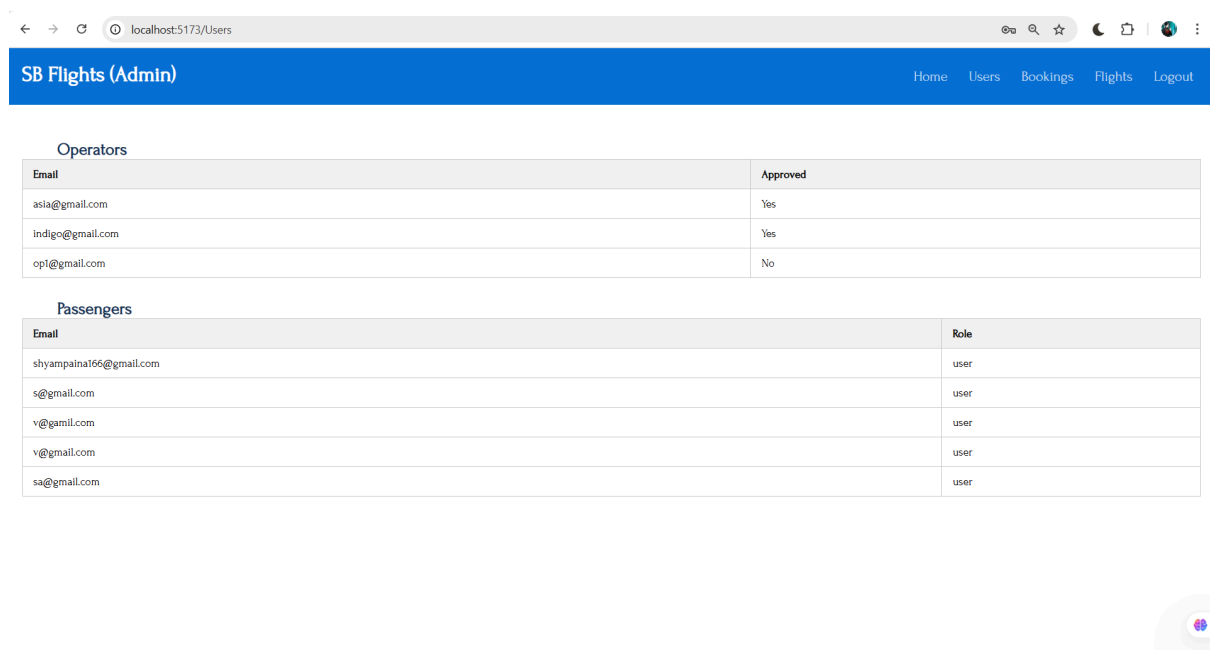


Admins can also **update existing flight details** from the dashboard. This feature allows modifications to be made to any flight information, such as timing, pricing, or availability.

Additionally, the admin dashboard provides options to:

- **View all flights**
- **View all bookings**
- **View all registered users**

These features help administrators manage the system efficiently and maintain full control over the operations.



The screenshot displays the 'SB Flights (Admin)' dashboard. At the top, there is a blue header bar with the title 'SB Flights (Admin)' on the left and navigation links 'Home', 'Users', 'Bookings', 'Flights', and 'Logout' on the right. Below the header, the 'Operators' section contains a table with two columns: 'Email' and 'Approved'. The table lists three operators: 'asia@gmail.com' (Approved: Yes), 'indigo@gmail.com' (Approved: Yes), and 'opl@gmail.com' (Approved: No). Below the Operators table, the 'Passengers' section contains a table with two columns: 'Email' and 'Role'. The table lists five passengers, all with the role 'user': 'shyampaina166@gmail.com', 's@gmail.com', 'v@gmail.com', 'v@gmail.com', and 'sa@gmail.com'. The browser's address bar at the top shows 'localhost:5173/Users'.

Operators	
Email	Approved
asia@gmail.com	Yes
indigo@gmail.com	Yes
opl@gmail.com	No

Passengers	
Email	Role
shyampaina166@gmail.com	user
s@gmail.com	user
v@gmail.com	user
v@gmail.com	user
sa@gmail.com	user

## ◆ User & Operator Management (Admin)

- View list of all registered users and operators.
- Approve or reject operator registration requests.
- Remove or modify user/operator accounts as needed.

← → ↻ localhost:5173/AllBookings

SB Flights (Admin) Home Users Bookings Flights Logout

### All Bookings

Flight: 6E234(asia)  
Date: 2025-06-27  
From: Hyderabad  
To: Delhi  
Departure: 12:57  
Arrival: 15:00  
Starting Price: ₹2000  
Available Seats: 300  
Mobile: 1234567890  
Email: synder@gmail.com  
Passengers:  
1. synder (Age: 20) - Seat: 2  
Total Fare: ₹2000  
Status: confirmed

Flight: SpiceJet SG-706  
Date: 2025-06-26  
From: Vizag  
To: Rajahmundry  
Departure: 13:02  
Arrival: 14:03  
Starting Price: ₹5000  
Available Seats: 310  
Mobile: 9881003218  
Email: synder@gmail.com  
Passengers:  
1. synder (Age: 20) - Seat: 2  
Total Fare: ₹5000  
Status: cancelled

Flight: SpiceJet SG-706  
Date: 2025-06-26  
From: Vizag  
To: Rajahmundry  
Departure: 13:02  
Arrival: 14:03  
Starting Price: ₹5000  
Available Seats: 310

Flight: SpiceJet SG-706  
Date: 2025-06-26  
From: Vizag  
To: Rajahmundry  
Departure: 13:02  
Arrival: 14:03  
Starting Price: ₹5000  
Available Seats: 310

## Booking Oversight (Admin)

- View all bookings made by users.
- Filter bookings by date, user, flight, etc.
- Cancel or review suspicious bookings.

← → ↻ localhost:5173/Admin-dashboard

SB Flights (Admin) Home Users Bookings Flights Logout

Users  
8 users  
[View All](#)

Bookings  
4 bookings  
[View All](#)

Flights  
5 flights  
[View All](#)

### New Operator Applications

Name	Email	Actions
op1	op1@gmail.com	<a href="#">Confirm</a> <a href="#">Reject</a>

## ◆ Admin Dashboard & Settings

- View total number of users, operators, flights, and bookings.
- Manage platform-level settings (e.g., seat limits, fare rules).
- Monitor logs or system activity.

## BACKEND DEVELOPMENT

The backend of the Flight Booking App is developed using **Node.js**, **Express.js**, and **MongoDB**. It is responsible for handling business logic, API routing, user authentication, and data management.

### Database Configuration

The application uses **MongoDB** for storing and managing data. You can either:

- Set up a **local MongoDB instance** using **MongoDB Compass**,  
or
- Use a **cloud-based service like MongoDB Atlas**.

A dedicated database is created with collections for:

- **Users**
- **Flights**
- **Bookings**
- Any other relevant entities

### Express.js Server Setup

An **Express.js server** is configured to handle all incoming HTTP requests and serve API endpoints to the frontend.

Essential middleware is included in the setup:

- **body-parser**: Parses incoming request bodies in JSON format
- **cors**: Enables secure Cross-Origin Resource Sharing between the frontend and backend

## API Route Definition

Separate route files are created to organize and manage API functionality, such as:

- **Flight Routes:** List, add, update, or delete flights
- **User Routes:** Register, login, fetch user profile
- **Booking Routes:** Create, view, or cancel bookings
- **Auth Routes:** Handle secure authentication operations

Each route file is connected to appropriate **route handlers**, built using Express.js to process incoming requests and interact with the database accordingly.

## Data Models and Schemas

Using **Mongoose**, schemas are defined for each data entity:

- **User Schema**
- **Flight Schema**
- **Booking Schema**

Each schema includes all required fields and constraints. Mongoose models are used to perform **CRUD operations** (Create, Read, Update, Delete) on the MongoDB collections, ensuring structured and efficient data handling.

## User Authentication

Authentication routes and middleware are implemented to manage:

- **User Registration**
- **Login**
- **Logout**

Protected routes are secured using **JWT tokens** or session-based authentication to ensure only authorized users can access sensitive data.

## Flight and Booking Management

Controllers are implemented to handle:

- **Flight Listings:** Fetch and display available flights from the database
- **Bookings:** Validate input data, process bookings, and update the database

Both user and admin can interact with these features, depending on their permissions.

## Admin Functionality

Admin-specific APIs are implemented to provide:

- **Add New Flights**
- **Update or Delete Existing Flights**
- **Manage Bookings and Users**

Authorization checks are added to ensure that **only authenticated admins** can access these features. This prevents unauthorized access and maintains system integrity.

## Error Handling

Custom middleware is implemented to handle runtime errors and API exceptions. The system returns:

- Meaningful **error messages**
- Appropriate **HTTP status codes**
- Structured error responses for better debugging and user feedback

## DATABASE DEVELOPMENT

The database layer of the Flight Booking App is built using **MongoDB**, with **Mongoose** as the ODM (Object Data Modeling) library. This section outlines how to structure the data using schemas and connect the database to the backend application.

### Schema Configuration

To ensure consistent data storage and retrieval, MongoDB schemas must be defined clearly using Mongoose. These schemas are based on the **ER (Entity Relationship) diagram** designed for the application and represent real-world entities such as:

- **User Schema** – Stores user details like name, email, password, and role (user/admin)
- **Flight Schema** – Stores flight information such as flight number, origin, destination, date, time, price, and available seats
- **Booking Schema** – Stores user bookings, including user reference, flight reference, seat number, and booking status

Each schema is defined using Mongoose with appropriate data types, validations, and references to ensure relational integrity and smooth CRUD operations.

### Connecting the Database to the Backend

Before interacting with the database through API routes, it's essential to establish a **secure and reliable connection** between the backend server and MongoDB.

The connection setup typically includes:

- Importing Mongoose into the backend server file (e.g., server.js or index.js)



- Connecting to MongoDB using a connection string (either local URI or a cloud-based URI from MongoDB Atlas)
- Handling success and error callbacks to confirm the connection status



## Conclusion

The **FlightFinder and Booking Web Application** offers a reliable, user-friendly solution for searching, managing, and booking flights. With distinct dashboards and role-based access for **Admin**, **Operators**, and **Users**, the system ensures clear functionality and secure operations across all levels.

The integration of features like:

- Real-time flight addition by operators,
- Seamless booking with passenger detail handling,
- Admin oversight for users, operators, and flights,

...makes this platform scalable and suitable for both educational and real-world deployment.

By using modern technologies like **React**, **Node.js**, **Express**, and **MongoDB**, the system achieves fast performance, modularity, and easy maintainability.

The project not only showcases core **MERN stack skills** but also demonstrates the understanding of authentication, authorization, REST API handling, and full-stack architecture — making it a valuable real-world portfolio project.