FlightFinder – Navigating Your Air Travel Options

1.Introduction

- Project Title: FlightFinder – Navigating Your Air Travel Options
- Team Members:
  1. Koppisett V V Sai Satya Sriram (Role : Frontend Development)
  2. Syed Rasha (Role : Backend Development)
  3. Konduri Sadvi (Role : Backend Developmet)
  4. Manda Varshini (Role : Database Development)

2.Project Overview

- Purpose :

  FlightFinder is a full-stack flight booking web application designed to simplify and enhance the air travel booking experience. The platform allows users to search, explore, and reserve flights efficiently based on their preferences. The goal is to provide a seamless, secure, and user-friendly flight reservation system.
- Features:
  - ➢ Advanced Flight Search (source, destination, date)
  - ➢ User Registration & Login (Authentication)
  - ➢ Flight Booking System
  - ➢ Booking History Management
  - ➢ Admin Dashboard for Managing Flights
  - ➢ Secure Authentication & Authorization
  - ➢ Responsive User Interface

3.Architecture:

- Frontend:

  The frontend of FlightFinder is developed using **React.js** as a Single Page Application (SPA). It follows a component-based architecture where UI elements such as search forms, flight listings, login, and booking pages are built as reusable components. Navigation is handled using React Router, and Axios is used to communicate with the backend APIs. Bootstrap is integrated for responsive design. The frontend manages user interactions and dynamically updates the UI based on API responses.

- Backend:

  The backend is built using **Node.js** and **Express.js**, following a RESTful API architecture. Express handles routing, middleware, and request processing, while controllers manage the business logic for users, flights, and bookings. Security is implemented using bcrypt for password hashing and environment variables for sensitive configurations. The backend acts as an intermediary between the frontend and the database.

- Database :

  The application uses MongoDB as a NoSQL database to store user, flight, and booking information. Mongoose is used as an Object Data Modeling (ODM) library to define schemas and perform CRUD operations. The database layer ensures structured data storage, validation, and efficient querying to support the application's functionality.

4.Setup Instructions

- Prerequisites:
  - ➢ Node.js & npm
  - ➢ MongoDB (Local or Atlas)
  - ➢ Git
  - ➢ VS Code
- Installation:
  1. Clone Repository:
     https://github.com/yourusername/FlightFinder.git
     cd FlightFinder
  2. Install Backend Dependencies
  3. Install Frontend Dependencies
  4. Setup Environment Variables (.env in server folder)

5. Folder Structure

- Client :
  - public/ – Static public files
  - src/ – Main source code folder
  - assets/ – Images and static resources
  - components/ – Reusable UI components
  - context/ – React Context for global state management
  - pages/ – Page-level components (Login, Register, Home, etc.)
  - RouteProtectors/ – Protected route components for authentication
  - styles/ – CSS and styling files
  - App.js – Main application component
  - App.css – App-level styles
  - index.js – Entry point of React app
  - index.css – Global styles
  - logo.svg – Logo file
  - reportWebVitals.js – Performance measurement
  - setupTests.js – Testing setup configuration

    - package.json – Frontend dependencies and scripts
    - package-lock.json – Dependency lock file
- Server:
    - index.js – Main server entry point
    - schemas.js – Mongoose schemas/models
    - package.json – Backend dependencies and scripts
    - package-lock.json – Dependency lock file

6. Running the Application

- Frontend: npm start in the client directory.
- Backend: npm start in the server directory.

7. API Documentation

i. Register User

Endpoint: POST /register

Request Body:

```
                {
"username": "John",
 "email": "john@gmail.com",
"usertype": "user",
 "password": "123456"
}
```

Response:

```
{
  "_id": "userId",
  "username": "John",
  "email": "john@gmail.com",
  "usertype": "user",
  "approval": "approved"
}
```

ii. Login User

Endpoint: POST /login

Request Body:

```
        {
  "email": "john@gmail.com",
  "password": "123456"
}
```

Response:

```
{
  "_id": "userId",
  "username": "John",
  "email": "john@gmail.com",
  "usertype": "user",
  "approval": "approved"
}
```

iii. Add Flight

Endpoint: POST /add-flight

Request body :

```
    {
 "flightName": "Air India",
 "flightId": "AI101",
 "origin": "Chennai",
 "destination": "Delhi",
 "departureTime": "10:00 AM",
 "arrivalTime": "12:30 PM",
  "basePrice": 5000,
  "totalSeats": 120
}
```

iv. Update Flight

Endpoint: PUT /update-flight

Response :

```
{
  "message": "flight updated"
}
```

v. Fetch All Flights

Endpoint: GET /fetch-flights

vi. Book Ticket

Endpoint**:** POST /book-ticket

Request body:

```
{
  "user": "userId",
  "flight": "flightId",
  "flightName": "Air India",
  "flightId": "AI101",
  "departure": "Chennai",
  "destination": "Delhi",
  "email": "john@gmail.com",
  "mobile": "9876543210",
  "passengers": ["John Doe", "Jane Doe"],
  "totalPrice": 10000,
  "journeyDate": "2026-02-20",
  "journeyTime": "10:00 AM",
  "seatClass": "economy"
}
```

Response:

```
{
  "message": "Booking successful!!"
}
```

vii. Fetch All Bookings

Endpoint**:** GET /fetch-bookings

viii. Cancel Ticket

Endpoint: PUT /cancel-ticket/:id

Response :

```
    {
"message": "booking cancelled" }
```

8. Authentication

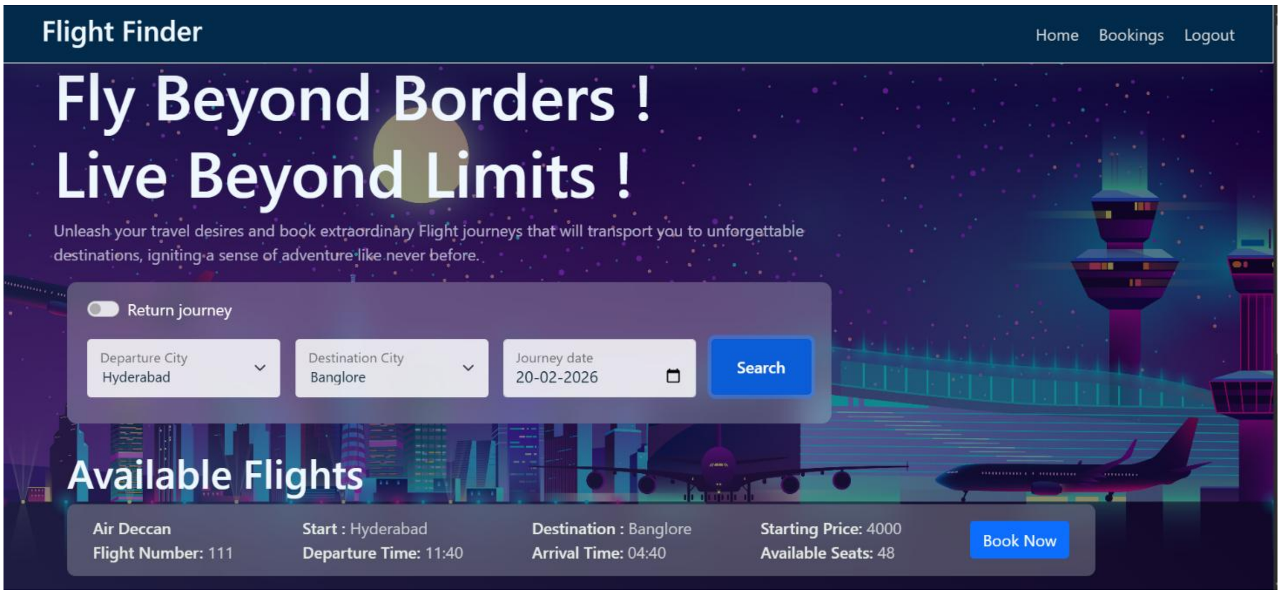
The FlightFinder application implements authentication using email and password verification with secure password hashing. During registration (POST /register), the user's password is encrypted using bcrypt before being stored in the MongoDB database. This ensures that plain-text passwords are never saved, enhancing security.

During login (POST /login), the system verifies the user by checking whether the email exists in the database. If the user is found, bcrypt is used to compare the entered password with the hashed password stored in the database. If the credentials match, the user details are returned as a response.

The system also supports role-based access control using the usertype field (e.g., user or flight-operator). Flight operators require admin approval before gaining full access. Approval status is managed using the approval field (approved, not-approved, or rejected), and administrators can approve or reject operators through dedicated endpoints.

Currently, authentication is session-based through request validation (no JWT tokens implemented). Password encryption, user role validation, and operator approval together ensure basic application-level security.

9. User Interface


Flight Finder

Home Bookings Logout

Fly Beyond Borders !
Live Beyond Limits !

Unleash your travel desires and book extraordinary Flight journeys that will transport you to unforgettable destinations, igniting a sense of adventure like never before.

Return journey

Departure City: Hyderabad

Destination City: Banglore

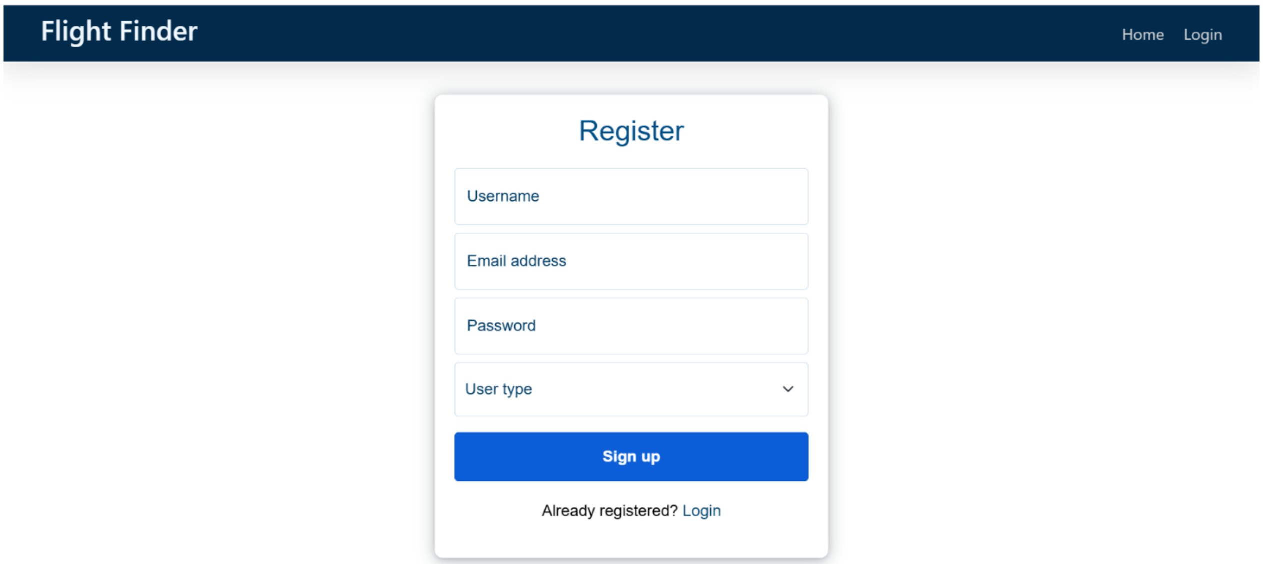
Journey date: 20-02-2026

Search

Available Flights

Air Deccan
Flight Number: 111
Start : Hyderabad
Departure Time: 11:40
Destination : Banglore
Arrival Time: 04:40
Starting Price: 4000
Available Seats: 48

Book Now


Flight Finder

Home Login

Register

Username

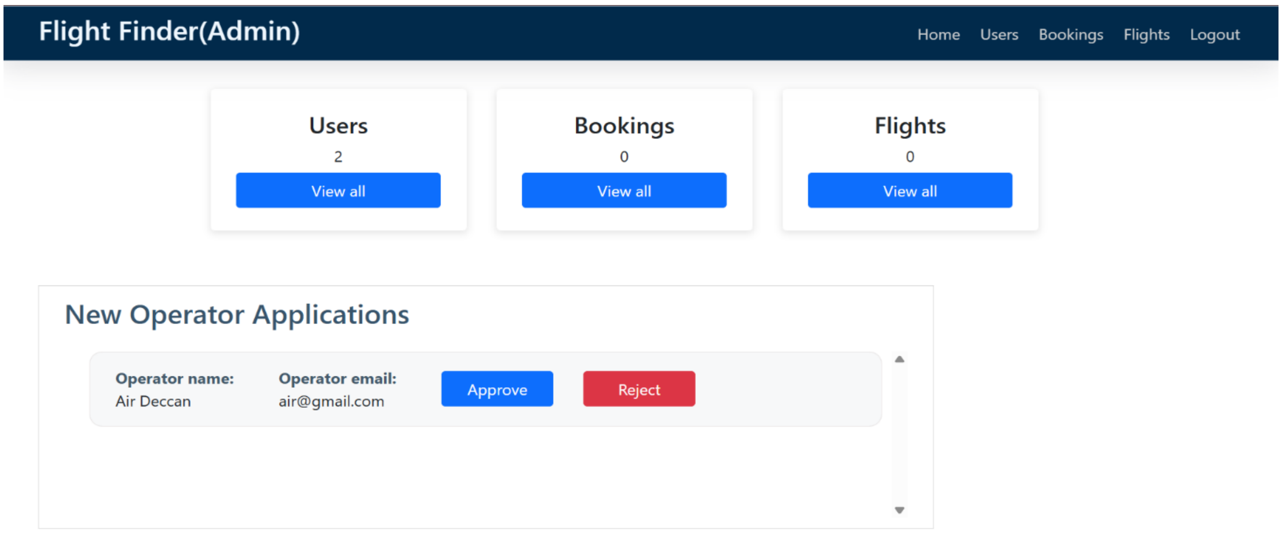
Email address

Password

User type

Sign up

Already registered? Login


Flight Finder(Admin)

Home Users Bookings Flights Logout

Users
2
View all

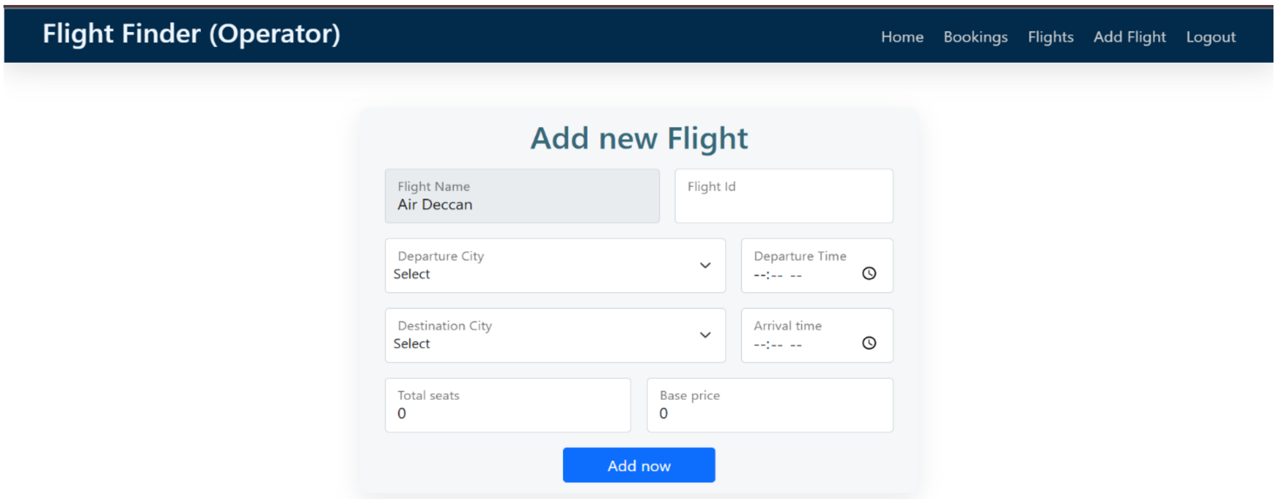
Bookings
0
View all

Flights
0
View all

New Operator Applications

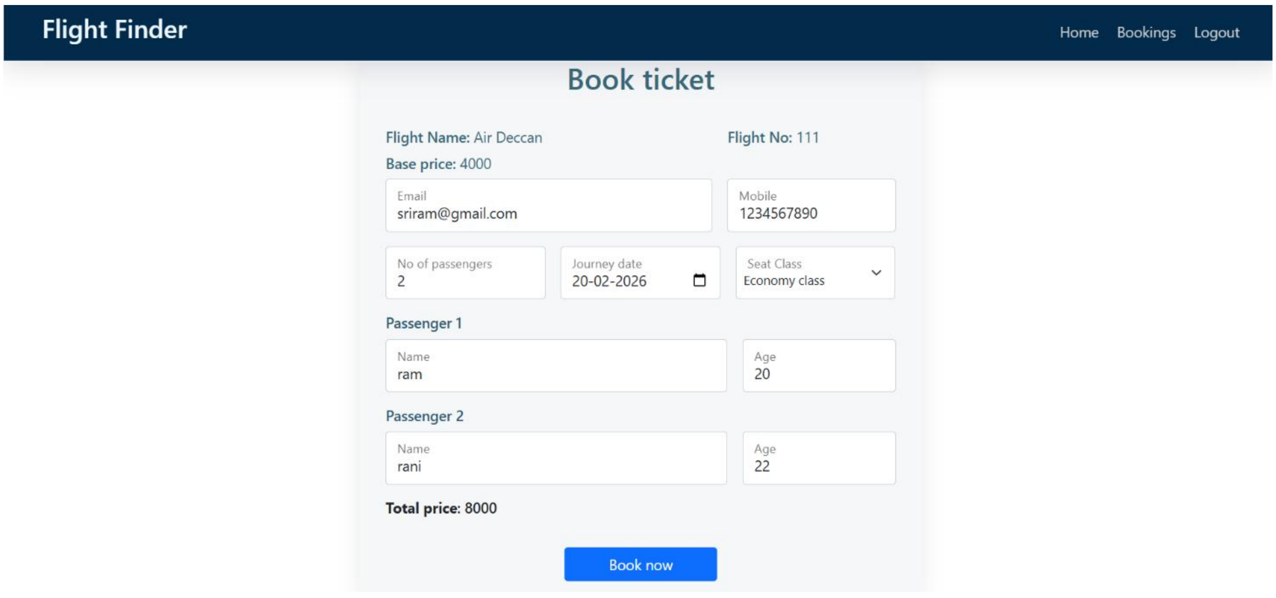
Operator name: Air Deccan
Operator email: air@gmail.com

Approve Reject



Add new Flight

Flight Name: Air Deccan
Flight Id
Departure City: Select
Departure Time: --:-- --
Destination City: Select
Arrival time: --:-- --
Total seats: 0
Base price: 0



Book ticket

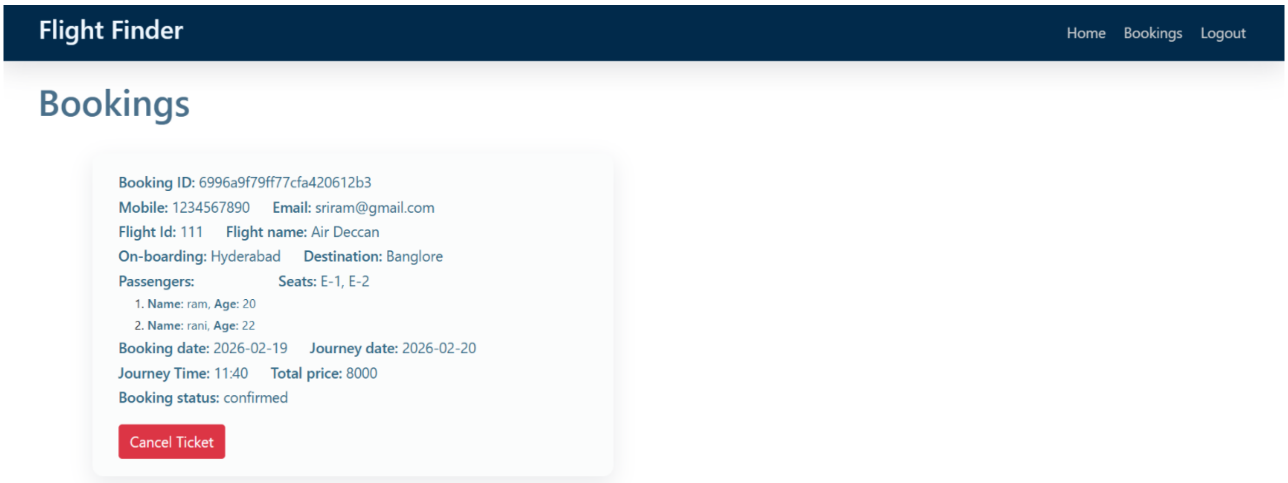
Flight Name: Air Deccan **Flight No:** 111
Base price: 4000

Email: sriram@gmail.com
Mobile: 1234567890
No of passengers: 2
Journey date: 20-02-2026
Seat Class: Economy class

Passenger 1
Name: ram
Age: 20

Passenger 2
Name: rani
Age: 22

Total price: 8000



Bookings

Booking ID: 6996a9f79ff77cfa420612b3
Mobile: 1234567890 **Email:** sriram@gmail.com
Flight Id: 111 **Flight name:** Air Deccan
On-boarding: Hyderabad **Destination:** Banglore
Passengers: **Seats:** E-1, E-2

1. **Name:** ram, **Age:** 20
2. **Name:** rani, **Age:** 22

Booking date: 2026-02-19 **Journey date:** 2026-02-20
Journey Time: 11:40 **Total price:** 8000
Booking status: confirmed

10. Testing

- Manual testing for UI flows
- API testing using Postman
- Error handling and validation checks
- Tested authentication and protected routes

11. Screenshots or Demo

Demonstration video link:

https://drive.google.com/file/d/1Nhksb7cDOC9UIViIJqojxTGZTPeEMzoY/view?usp=sharing

12. Known Issues

- No Payment Gateway Integration (Demo Booking Only):

  The system currently supports demo bookings only and does not process real payments or handle transactions and refunds.

- Limited Filtering Options

  Flight search includes only basic filters (origin and destination) without advanced options like price range, airline, sorting, or time-based filtering.

- Basic Admin Analytics

  The admin panel provides limited insights and lacks advanced reports, revenue tracking, and graphical data visualization

13. Future Enhancements

- Payment Gateway Integration (Razorpay/Stripe)
- Advanced Analytics Dashboard
- Real-time Flight API integration
- Mobile App Version
- Reviews & Ratings System