MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

Kodambakkam, Chennai-600024

(An Autonomous Institution)

MERN STACK POWERED BY MONGODB

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

PROJECT TITLE: FLIGHT BOOKING APPLICATION USING MERN

TEAM ID: NM2024TMID16007

FACULTY MENTOR: Mrs. P.Muthulakshmi

Project submitted by,

NAAN MUDHALVAN ID	NAME	REGISTER NUMBER
29A6B070E1C039EB0CE413B330193F2D	MOHAMED ARSHAD AHAMED T	311521243033
33A7CC79A5CDC35952D45A5C3412C8EC	ABISHEK P	311521243003
02D5E3FD395E4A7A44B46C16AD0FDB02	PHEROSE PANDIAN V	311521243037
56311EC7E5020BC9EE846F60136F80D0	KARTHIK G	311521243022

BONAFIDE CERTIFICATE

Certified that this project report "**FLIGHT BOOKING APPLICATION USING MERN** " is the bonafide work of "**MOHAMED ARSHAD AHAMED T (311521243033),ABISHEK P(311521243003), PHEROSE PANDIAN V (311521243037), KARTHIK G (311521243022)**" who carried out the project work under my supervision.

SIGNATURE

Mrs. P.MUTHULAKSHMI M.E.,

ASSISTANT PROFESSOR

Artificial intelligence and data science

Meenakshi Sundararajan Engineering College

No. 363, Arcot Road, Kodambakkam,

Chennai -600024

SIGNATURE

Mrs N.Mathangi,M.E.,M.B.A.,[Ph.D]

HEAD OF THE DEPARTMENT

Artificial intelligence and data science

Meenakshi Sundararajan Engineering College

No. 363, Arcot Road, Kodambakkam,

Chennai – 600024

Submitted for the project viva voce of Bachelor of Engineering in Computer Science and Engineering held on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our beloved Secretary **Mr. N. Sreekanth**, Principal **Dr. S. V. Saravanan** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Mrs.N.Mathangi,** Associate Professor Head of the Department, Department of Artificial intelligence and data science, for her profound inspiration, kind cooperation and guidance.

We're grateful to **Mrs. P. Muthulakshmi** ,Internal Guide, Assistant Professor as our project coordinator for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and Blessings it wouldn't have been possible.

ABSTRACT

The **Flight Booking System** is a modern web application developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), designed to facilitate a user-friendly and efficient flight booking experience. The primary objective of this project is to allow users to browse available flights, book tickets, and select seats with ease. The system provides a streamlined interface for both registration and login, ensuring secure access to the platform. Once logged in, users can search for flights by specifying criteria such as departure and destination cities, travel dates, and the preferred class (economy, business, or first class). The application is designed to handle multiple users and supports flight searches based on real-time data stored in the MongoDB database.

MongoDB serves as the database for storing critical information related to flights, users, and bookings. It efficiently manages and retrieves data, ensuring fast response times for user queries. The system supports complex queries such as filtering flights by price, availability, and time of departure. This allows users to make informed decisions when selecting the best flight options. Additionally, the database is optimized for storing multiple flight schedules, passenger data, and seat availability, ensuring that the system remains scalable as the volume of users and data grows.

On the backend, **Express.js** and **Node.js** work in tandem to handle server-side operations and API requests. The backend manages user authentication, handles requests for flight searches, and processes booking information. It ensures that all user data and booking details are securely stored and processed, providing a safe environment for transactions. The system is built with a focus on modularity, ensuring that new features or changes can be easily integrated into the backend architecture without affecting the performance or integrity of the application.

The frontend, developed using **React.js**, provides a dynamic and responsive user interface. React's component-based architecture ensures that the system remains flexible and easy to maintain. Users can search for flights, view available seats in a visual format, and select their desired seats with ease. The interface also provides users with clear information about flight pricing, seat classes, and availability. Visual feedback during the booking process enhances the overall user experience, ensuring that users can quickly complete their bookings without confusion or delays.

TABLE OF CONTENTS

CHAPTER 1

1. PROJECT OVERVIEW

1.1 PURPOSE

The Flight Booking System is designed to simplify and enhance the process of booking flights online while providing a robust platform for managing flight operations. This project, built using the MERN stack (MongoDB, Express.js, React.js, Node.js), allows users to search for available flights, select seats, and book tickets in a user-friendly, responsive web interface. The system ensures a smooth experience across devices, making it accessible to a broad range of users.

In addition to the user-facing booking functionalities, the project includes a Flight Operator Page, where operators can add new flights, update schedules, and manage seat availability in real-time. This feature streamlines flight management for service providers, giving them control over flight operations through a simple interface.

The project also supports admin functionalities, allowing for comprehensive management of flight data, bookings, and user information. The system leverages MongoDB for efficient data storage and retrieval, ensuring real-time access to flight and booking details.

By integrating these features, the system aims to provide an efficient, scalable solution that caters to both users and flight operators. The flexible architecture allows for easy integration of additional features, such as payment gateways or loyalty programs, making the system adaptable for future growth.

1.2 FEATURES

- **User Registration & Authentication**:

  Users can create accounts and securely log in to manage their bookings and view flight history.

- **Flight Search**:

  Users can search for available flights by entering details like departure and destination locations, travel dates, and the number of passengers (adults and children).

- **Flight Booking**:

  Users can select flights from the search results and proceed to book tickets, viewing flight details such as departure and arrival times, flight duration, and price.

- **Flight Operator Page**:

Flight operators have a dedicated dashboard to add new flights, update schedules, and manage flight availability and details, including departure times, routes, and capacity.

- **Admin Panel**:

Admins can manage the entire system, including viewing all bookings, managing flight information, handling user data, and overseeing flight schedules and operations.

- **Responsive Design**:

The system is designed to be fully responsive, ensuring an optimal user experience across various devices, including desktops, tablets, and mobile phones.

- **Real-Time Flight Data**:

Flight availability, schedules, and details are updated in real-time, ensuring users receive the most up-to-date information when searching for flights.

- **Booking Management**:

Users can view their past bookings, check upcoming flights, and cancel their reservations if necessary, with updated status notifications.

- **Flight Management**:

Operators and admins can create, update, or delete flight entries, providing full control over flight operations.

- **Efficient Database Integration**:

MongoDB is used to store and retrieve user, flight, and booking data efficiently, ensuring fast performance and scalability.

- **Scalability**:

The system, built with the MERN stack, is designed to scale easily as the number of users and flights grows, ensuring smooth operation even as demand increases.

.

CHAPTER 2

2. ARCHITECTURE

The **Flight Booking System** is built on the **MERN stack** (MongoDB, Express.js, React.js, Node.js), providing a full-stack solution that ensures efficient data management, real-time flight information, and seamless user interactions. Each component in the stack contributes to delivering a scalable, secure, and responsive flight booking experience. The architecture separates the frontend and backend, with the frontend offering a dynamic user interface for passengers and flight operators, while the backend handles business logic, data processing, and secure communication. This structure supports real-time flight searches, bookings, and flight management, ensuring smooth performance and flexibility for future growth.

2.1 FRONTEND ARCHITECTURE

- **React**: React is used to build a dynamic, component-based interface for the flight booking system. It enables reusable components for features like flight searching, booking, and user management, offering a seamless experience for customers and operators. React's virtual DOM ensures efficient updates by re-rendering only the components that need to be changed.

- **Bootstrap**: Bootstrap is used to quickly design a responsive and visually appealing UI. Its pre-built components like buttons, forms, navigation bars, and modals streamline the frontend development process, while the grid system ensures the application adapts to different screen sizes, providing a consistent experience across devices.

- **CSS**: Custom CSS is used to style the application, ensuring that the design aligns with the brand's aesthetics. It handles layout, typography, colors, and responsiveness, ensuring the UI is optimized for various devices using media queries and CSS Flexbox/Grid for dynamic layouts.

2.2 BACKEND ARCHITECTURE

- **Node.js**: Node.js serves as the runtime environment for the backend, enabling asynchronous, non-blocking operations. It efficiently handles multiple requests, such as user authentication, flight booking, and data retrieval, ensuring optimal performance even with concurrent operations.

- **Express.js**: Express.js is used as the backend framework to manage server-side logic, routing, and handle API requests. It facilitates the creation of RESTful APIs for interactions between the frontend and the database, managing tasks such as user registration, flight search, and booking, as well as inventory management.

- **JWT Authentication**: **JSON Web Tokens (JWT)** provide a secure method for authenticating users and managing sessions. They validate user identities and roles, ensuring that only authorized users, like customers or admins, can access certain features such as booking flights or managing flight schedules.

2.3 DATABASE ARCHITECTURE

- **MongoDB**: MongoDB is used as the NoSQL database for the flight booking system, providing a flexible and scalable way to store and manage data. It handles various collections such as users, flight details, bookings, and payment records, offering high performance and scalability for growing datasets.
- **Document Structure**: MongoDB uses a document-based schema, which allows for the easy management of collections like user accounts, flight schedules, booking records, and payment transactions. Each document can store structured and unstructured data, making it ideal for the dynamic nature of the flight booking system, where data varies by user and booking.

CHAPTER 3

3. SETUP INSTRUCTIONS

3.1 PREREQUISITES

Before setting up the "Book Store" application, make sure the following prerequisites are met:

1. **Operating System**

   A Windows 8 or higher machine is recommended, though the setup should also work on macOS and Linux systems.

2. **Node.js**

   Download and install Node.js (version 14 or above). Node.js is required to run both the backend server (Node and Express) and the frontend server (React).

3. **MongoDB**

   Local MongoDB Installation: Install MongoDB Community Edition from MongoDB's official site if you prefer to use a local database.

   MongoDB Atlas (Optional): Alternatively, you can set up a free MongoDB Atlas account to use MongoDB in the cloud. MongoDB Atlas provides a connection string that will be needed to configure the backend.

4. **Two Web Browsers**

   The application works best with two web browsers installed for simultaneous testing (e.g., Google Chrome, Mozilla Firefox).

5. **Internet Bandwidth**
   A stable internet connection with a minimum speed of 30 Mbps is recommended,especially if using MongoDB Atlas or deploying to a remote server.

6. **Code Editor**

   Visual Studio Code or any other preferred code editor for easy management of the codebase and environment configuration.

3.2 INSTALLATION

1. **Install Node.js and MongoDB**

   Node.js: Download and install Node.js (Version 14 or above).

   MongoDB: Download and install the MongoDB Community Edition and set up MongoDB on your machine. Alternatively, you can use MongoDB Atlas for cloud hosting.

2. **Clone the Repository**

   Open a terminal and clone the project repository:

   git clone <repository_url> cd

   flight booking system

3. **Install Backend Dependencies**

   Navigate to the backend folder and install the necessary Node.js packages:

   cd backend npm

   install

4. **Install Frontend Dependencies**

   Open a new terminal window or navigate back to the root directory, then go to the frontend folder and install dependencies.

   cd ../frontend npm

   install

5. **Configure Environment Variables**

- In the backend folder, create a .env file.
- Add the following environment variables:

  MONGO_URI=<your_mongodb_connection_string>

  PORT=6001

  JWT_SECRET=<your_jwt_secret>

6. **Run MongoDB**

   If using MongoDB locally, ensure the MongoDB server is running:

   mongodb

7. **Start the Backend Server**

   In the backend folder, start the server with the following command:

   npm start

This will start the backend server on https://localhost : 6001 .

8. **Start the Frontend Server**

   In the front end folder, start the React development server:

   npm start

   This will start the frontend server on https://localhost : 3000 .

9. **Access the Application**

   Open your web browser and navigate to https://localhost : 3000 to view and interact withthe application

CHAPTER 4

4. FOLDER STRUCTURE

The "Flight booking" application follows a well-organized folder structure for both the React frontend and Node.js backend. This structure ensures that components, APIs, and utilities are easy to locate, modify, and scale.

4.1 CLIENT: REACT FRONTEND STRUCTURE

The frontend is structured using the following folders:

```
client/
├── package-lock.json        # Lock file for dependencies
├── package.json             # Frontend dependencies and scripts
├── public/                  # Public assets (e.g., index.html, images)
│
└── src/                     # Source files for the React application
    ├── components/          # Reusable UI components (e.g., buttons, forms)
    ├── pages/               # Page components (e.g., Home, Booking)
    ├── services/            # API call handlers using Axios
    ├── context/             # Global state management using React Context
    ├── assets/              # Static files like images, fonts
    ├── App.js               # Main React app component with routing
    ├── index.js             # Entry point for React app
    └── styles/              # Global CSS files and styling
```

- ✓ **public/index.html**: The root HTML file that React renders to, serving as the entry point for the frontend.
- ✓ **src/components/**: Contains reusable UI components such as buttons, input fields, flight cards, or modals, which are used throughout the application for consistent user interaction.
- ✓ **src/pages/**: Houses main pages like the homepage, booking page, and user profile, structured as components to simplify routing and navigation between different views.
- ✓ **src/services/**: Manages API calls using Axios to interact with the backend. This includes fetching flight details, user authentication, booking operations, and more.
- ✓ **src/context/**: Manages global state for user sessions, authentication, and other shared data across different components using React's Context API.
- ✓ **src/assets/**: Stores static files such as images (flight pictures, logos), fonts, and other assets required by the frontend.
- ✓ **src/styles/**: Holds global styling and theme files for consistent UI appearance.

4.2 SERVER: NODE.JS BACKEND STRUCTURE

The backend server is structured as follows:

```
server/
│
├── index.js              # Entry point for the backend server
├── package-lock.json     # Lock file for backend dependencies
├── package.json          # Backend dependencies and scripts
└── schemas.js            # MongoDB schemas (e.g., for user, flight, booking)
```

- **index.js:** The main entry point for the backend server. This file initializes the server, sets up middleware, establishes database connections, and starts the Express server, making it ready to handle incoming requests.
- **package-lock.json:** This file ensures the consistency of your backend dependencies. It locks the versions of packages installed, ensuring that all environments use the same version of dependencies.
- **package.json:** Contains the list of backend dependencies (e.g., Express, Mongoose, etc.), as well as scripts for running and building the backend server. It also defines metadata and versioning for the project.
- **schemas.js:** Contains MongoDB schemas for defining the structure of your collections (e.g., User, Flight, Booking). This file uses Mongoose to define how the data should be organized and validated in the database.

CHAPTER 5

5. RUNNING THE APPLICATION

To run both the frontend and backend servers locally, follow these steps:

5.1 SET UP THE FRONTEND (SERVER)

1. Open another terminal window and navigate to the client directory:

   cd client

2. Install the frontend dependencies:

   npm install

3. Start the frontend server:

   npm start

The frontend server will launch at https://localhost:3000

5.2 SET UP THE BACKEND (SERVER)

1. Navigate to the server directory:

   cd server

2. Install the necessary backend dependencies:

   npm install

3. Create a .env file in the root of the server directory and add your environment variables(e.g.,MongoDB URI, JWT secret, etc.)

   MONGODB_URI=your_mongo_connection_url

   JWT_SECRET=your_jwt_secret

   PORT=6001

4. Start the backend server:

   node index.js

   The server will run on https://localhost:6001 by default (unless you specify a different portin the .env file).

CHAPTER 6

6. API DOCUMENTATION

The following section documents the backend API endpoints for the "Flight Booking System" project. Each endpoint includes details about the HTTP methods, request parameters, and example responses.

6.1 Customer Endpoints

6.1.1 User Registration

- Endpoint: /api/auth/register
- Method: POST
- Description: Registers a new user.

Request Body:

```
{
 "name": "John Doe",
 "email": "john.doe@example.com",
 "password": "password123",
 "user type":admin,customer,flight operator
}
```

Example Response:

```
{
 "message": "User registered successfully",
 "user": {
  "_id": "617f3e014cde611401efe5ff",
  "name": "John Doe",
  "email": john.doe@example.com
  "user type":admin
 }
}
```

6.1.2 User Login

- Endpoint: /api/auth/login
- Method: POST
- Description: Logs in a user and returns a JWT token.

Request Body:

```
{
 "email": "john.doe@example.com",
 "password": "password123"
}
```

Example Response:

```
{
 "message": "Login successful",
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
 "user": {
  "_id": "617f3e014cde611401efe5ff",
  "name": "John Doe",
  "email": "john.doe@example.com"
 }}
```

6.1.3 Search Flights

- Endpoint: /api/flights/search
- Method: GET
- Description: Searches for available flights based on origin, destination, and departure date.

Query Parameters:

- from: Departure city (e.g., "Chennai")
- to: Destination city (e.g., "Banglore")
- date: Departure date (e.g., "2024-12-01")

Example Request:

GET /api/flights/search?from=Chennai&to=Banglore&date=2024-12-01

Example Response:


```
[
{
   "_id": "673f3e014cde611401efe5ff",
   "flightNumber": "123456",
   "departure": "Chennai",
   "arrival": "Banglore",
   "departureTime": "2024-12-01 08:00:00am",
   "arrivalTime": "2024-12-01 12:00:00pm",
   "price": 1000.00,
   }
]
```


6.1.4 Book a Flight

- Endpoint: /api/bookings
- Method: POST
- Description: Books a flight for the user.

Request Body:


```
{
 "userId": "617f3e014cde611401efe5ff",
 "flightId": "673f3e014cde611401efe5ff",
 "passengers": [
   {
    "name": "John Doe",
    "age": 30,
   },
   {
    "name": "Jane Doe",
    "age": 28,
   }
 ],
 "class": "Economy"
}
```


Example Response:


```
{
 "message": "Flight booked successfully",
 "booking": {
   "_id": "123f3e014cde611401efe6aa",
   "userId": "617f3e014cde611401efe5ff",
```

```
    "flightId": "673f3e014cde611401efe5ff",
    "passengers": [
      {
        "name": "John Doe",
        "age": 30,
      },
      {
        "name": "Jane Doe",
        "age": 28,
      }
    ],
    "class": "Economy",
    "totalPrice": 2000.00,
    "bookingStatus": "Confirmed"
  }
}
```


6.1.5 View User Bookings

- Endpoint: /api/bookings/user/:userId
- Method: GET
- Description: Retrieves all bookings made by a specific user.

Example Request:

GET /api/bookings/user/617f3e014cde611401efe5ff

Example Response:


```
[
  {
    "_id": "123f3e014cde611401efe6aa",
    "flightId": "673f3e014cde611401efe5ff",
    "passengers": [
      {
        "name": "John Doe",
        "age": 30,
      },
      {
        "name": "Jane Doe",
        "age": 28,
      }
    ],
    "bookingStatus": "Confirmed",
    "class": "Economy",
    "totalPrice": 2000.00
  }
]
```


6.1.6 Cancel a Booking

- Endpoint: /api/bookings/:bookingId
- Method: DELETE
- Description: Cancels an existing flight booking.

Example Request:

DELETE /api/bookings/123f3e014cde611401efe6aa

Example Response:


```
{
  "message": "Booking canceled successfully"
}
```


6.2 Admin Endpoints

6.2.1 Admin Login

- **Endpoint:** /api/admin/login
- **Method:** POST
- **Description:** Logs in an admin and returns an authentication token.

Request Body:


```
{
  "email": "admin@example.com",
  "password": "adminpassword"
}
```


Example Response:


```
{
  "message": "Admin logged in successfully",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```


6.2.2 View All Users

- **Endpoint:** /api/admin/users
- **Method:** GET
- **Description:** Fetches all registered users.

Example Response:


```
[
  {
    "_id": "617f3e014cde611401efe5ff",
    "name": "John Doe",
    "email": "john.doe@example.com",
    "role": "user"
  },
  {
    "_id": "617f3e014cde611401efe600",
    "name": "Jane Doe",
    "email": "jane.doe@example.com",
    "role": "user"
  }
]
```


6.2.3 View All Flights

- **Endpoint:** /api/admin/flights
- **Method:** GET
- **Description:** Fetches all available flights in the system.

Example Response:


```
[
  {
    "_id": "673f3e014cde611401efe5ff",
    "flightNumber": "123456",
    "departure": "Chennai",
```

```
    "arrival": "Banglore",
    "departureTime": "2024-12-01 08:00:00am",
    "arrivalTime": "2024-12-01 12:00:00pm",
    "price": 1000.00,
  },
  {
    "_id": "673f3e014cde611401efe6ff",
    "flightNumber": "123457",
    "departure": "Chennai",
    "arrival": "Hyderabad",
    "departureTime": "2024-12-02 10:00:00am",
    "arrivalTime": "2024-12-02 11:30:00am",
    "price": 1500.00,
  }
]
```


6.2.4 View All Bookings

- **Endpoint:** /api/admin/bookings
- **Method:** GET
- **Description:** Fetches all flight bookings.

Example Response:


```
[
  {
    "_id": "bookingId123",
    "userId": "user123",
    "flightId": "flight123",
    "status": "Confirmed",
    "seatsBooked": 2,
    "totalPrice": 2000.00
  },
  {
    "_id": "bookingId124",
    "userId": "user124",
    "flightId": "flight124",
    "status": "Cancelled",
    "seatsBooked": 1,
    "totalPrice": 3000.00
  }
]
```


6.2.5 Approve Flight Operators

- **Endpoint:** /api/admin/operators/:operatorId/approve
- **Method:** PUT
- **Description:** Approves flight operators to allow them to manage flights.

Request Body:


```
{
  "status": "approved"
}
```


Example Response:


```
{
  "message": "Flight operator approved successfully"}
```

6.3 Flight Operator Endpoints

6.3.1 Operator Login

- **Endpoint:** /api/operator/login
- **Method:** POST
- **Description:** Logs in a flight operator and returns a token.

Request Body:

```
{
  "email": "operator@example.com",
  "password": "operatorpassword"
}
```

Example Response:

```
{
  "message": "Operator logged in successfully",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

6.3.2 Create Flight

- **Endpoint:** /api/operator/flights
- **Method:** POST
- **Description:** Allows a flight operator to create a new flight.

Request Body:

```
{
  "flightNumber": "123456",
  "departure": "Chennai",
  "arrival": "Banglore",
  "departureTime": "2024-12-01 08:00:00am",
  "arrivalTime": "2024-12-01 12:00:00pm",
  "price": 1000.00,
  "capacity": 150
}
```

Example Response:

```
{
  "message": "Flight created successfully",
  "flight": {
    "_id": "673f3e014cde611401efe5ff",
    "flightNumber": "123456",
    "departure": "Chennai",
    "arrival": "Banglore",
    "departureTime": "2024-12-01 08:00:00am",
    "arrivalTime": "2024-12-01 12:00:00pm",
    "price": 1000.00,
    "capacity": 150
  }
}
```

6.3.3 Update Flight Details

- **Endpoint:** /api/operator/flights/:flightId
- **Method:** PUT
- **Description:** Allows a flight operator to update flight information.

Request Body:


```
{
  "price": 1500.00,
  "capacity": 200
}
```


Example Response:


```
{
  "message": "Flight updated successfully",
  "flight": {
    "_id": "673f3e014cde611401efe5ff",
    "price": 1500.00,
    "capacity": 200
  }
}
```


6.3.4 Delete Flight

- **Endpoint:** /api/operator/flights/:flightId
- **Method:** DELETE
- **Description:** Allows a flight operator to delete a flight.

Example Request:

DELETE /api/operator/flights/673f3e014cde611401efe5ff

Example Response:


```
{
  "message": "Flight deleted successfully"
}
```

CHAPTER 7

7. AUTHENTICATION

The "Flight Booking Application" application employs a JWT (JSON Web Token)-based authentication and authorization approach, allowing secure user sessions without the need for persistent server-side sessions. Here's an overview of the authentication and authorization flow:

7.1 USER AUTHENTICATION

The Flight Booking System uses JWT (JSON Web Token)-based authentication to securely manage user sessions. This approach allows for secure communication between the client and server while avoiding the need for persistent server-side sessions.

Registration:

- When a new user (either a passenger, flight operator, or admin) registers, their credentials (e.g., name, email, and password) are securely stored in the MongoDB database.
- Passwords are hashed using bcrypt to ensure they are not stored in plaintext.

Login:

- During login, the user provides their email and password.
- If the credentials are valid, the server generates a JWT token that includes the following:
- User ID: Unique identifier for the user.
- Role: Specifies the user's role (passenger, flight operator, or admin).
- Expiration Time: The token expires after a certain period to improve security.

7.2 TOKEN GENERATION AND STRUCTURE

The JWT token is generated on the server using a secret key stored in the backend environment variables. The token structure typically contains the following:

- **User ID:** A unique identifier (UID) assigned to the user upon registration. This is used to identify the user when making requests.
- **Role:** This indicates the role of the user, which determines their access level:
- **Passenger:** Can search for flights, book tickets, view booking history, and manage their account.
- **Flight Operator:** Can add new flights, delete flights, and view flight schedules.
- **Admin:** Can view all users, flights, and bookings, approve flight operators, and manage system-wide settings.
- **Expiration Time:** The token has a set expiration time (e.g., 24 hours), after which it becomes invalid. This enhances security by limiting the potential for unauthorized access with expired tokens.

Example JWT Payload:

```
{
  "userId": "5f3e1b1d4f1c1a2f3a3b8e4a",
  "role": "passenger",
  "exp": 1627558290
}
```

7.3 AUTHORIZATION

Role-Based Access Control (RBAC):

The system uses Role-Based Access Control (RBAC) to protect routes and APIs based on the user's role. Each role has access to specific actions and endpoints:

- **Passenger:**
  - Search for flights
  - View flight details
  - Book a flight
  - View booking history

    - Manage personal account settings
- **Flight Operator:**
    - Add new flights
    - Delete existing flights
    - View and update flight schedules
- **Admin:**
    - View all users, flights, and bookings
    - Approve or reject flight operators
    - Manage flight operators and their permissions
    - Perform system-wide maintenance tasks (e.g., database cleanup, access control)

The JWT token is passed in the HTTP headers of each request (typically using the Authorization header). The server verifies the token and ensures the user has the required role to perform the action.

CHAPTER 8

8.USER INTERFACE

8.1 LOGIN AND REGISTRATION SCREENS

Simplified Login Process: The login screen should be designed with ease of use in mind, providing users with a clean interface that allows quick access to their accounts. Use prominent and easily recognizable input fields for username/email and password, with the option to toggle password visibility.

User Registration: Design the registration screen to capture essential details like name, email, password, and User type while keeping it uncluttered. Provide clear password strength indicators and validation messages to improve user experience.

8.2 HOME PAGE

Top Navigation Bar

- Airways Name/Logo: The top two lines should prominently feature the airline or booking system's logo or name in the center. This serves as both branding and a clickable link back to the homepage.
- Quick Access Buttons: On the right side of the top bar, provide quick access buttons for users to log in and navigate back to the homepage. This could include:
- Login Button: A button that redirects to the login or registration page, with a call to action such as "Sign In" or "Log In."
- Home Button: A button that redirects users back to the homepage if they are on any other page.

Flight Search Functionality

- Flight Search Form: This is the centerpiece of the homepage. Include a user-friendly flight search form prominently on the page, allowing users to search for flights by entering:
- Departure City/Location: Input field with suggestions for cities or airports.
- Destination City/Location: Input field with suggestions for destinations.
- Travel Dates: Calendar input for selecting departure and return dates.
- Search Button: A clear, prominently placed search button ("Search Flights") that initiates the flight search based on the entered details.

About Us Section

- Introduction: Beneath the navigation bar, include a brief and engaging "About Us" section. This section can provide a short overview of the airline or platform, explaining its mission, values, and key features.

8.3 ADMIN PAGE

Dashboard Overview: The admin page should feature a comprehensive dashboard that gives an overview of critical metrics like total bookings, revenue generated, flights available, and user activity. Graphical representations (charts and graphs) should be used to present data clearly.

Flight Management: Admins should have a dedicated section to add, edit, or remove flights from the system. This includes setting flight schedules, updating seat availability, and pricing, ensuring real-time data synchronization.

User Management: Include a section where the admin can manage users (customers and operators), with functionalities like viewing user profiles, resetting passwords, suspending accounts, and handling customer queries or complaints.

8.4 FLIGHT OPERATORS PAGE

Flight Scheduling: The flight operator page should provide tools to manage flight schedules, including setting departure and arrival times, updating available seats, and modifying route details. Operators should have an easy way to make quick changes in case of delays or cancellations.

Real-Time Flight Monitoring: Integrate real-time flight monitoring, allowing operators to track current flight statuses, including delays, early arrivals, and cancellations, to manage operations more effectively.

Passenger List Management: The operator should have access to a real-time passenger list, with the ability to update check-in statuses, manage special requests (such as meals or wheelchair access), and handle seat assignments.

8.5 CUSTOMER PAGE

Flight Search Functionality: Users can search for flights by entering departure and destination locations, travel dates, and number of passengers. After searching, they can choose a suitable flight.

Passenger Details: After selecting a flight, users enter passenger details including name, age, email address, and phone number. This section also allows for any special requests (e.g., meal preferences or wheelchair assistance).

Booking Review: Users can review their booking summary, which includes flight details, passenger information, and pricing breakdown. They can confirm the booking before proceeding.

View Bookings: Customers can view all their current and past bookings, with details like flight information and passenger names. They can also download booking confirmations in PDF format.

Booking Cancellation: Users have the option to cancel their booking by clicking a "Cancel Booking" button. A confirmation message will appear, along with details about any cancellation policies.

CHAPTER 9

9.TESTING

To ensure a robust and reliable "Flight Booking" application, a comprehensive testing strategy has been implemented, covering both frontend and backend functionality. Here is an overview of the testing approach and tools used:

9.1UNIT TESTING:

Description: Unit tests are written to verify individual functions and modules. This helps identify any issues at the component level early in the development process**.**

Tools Used:

Jest for testing JavaScript functions, especially on the backend.

Mocha andChai for testing API endpoints and other backend

logic.

Example Tests: Checking API responses for login, registration, and flight booking.

9.2 INTEGRATION TESTING:

Description: Integration tests are performed to verify that different modules and services work well together. This includes interactions between the client and server, as well as interactions with the database.

Tools Used:

Jest and Enzyme (for React) to test component interactions on the frontend.

Supertest incombination with Mocha for testing API routes and their responses.

Example Tests: Testing the flow from user registration to booking an flight and retrievinguser-specific data from the database.

9.3 END-TO-END (E2E) TESTING:

Description:
End-to-end (E2E) testing is used to simulate real-world user behavior and interactions to ensure the "Flight Booking System" functions as expected from start to finish. This testing involves covering the entire user journey, from logging in, searching flights, booking a flight, and making payments.

9.3.1 Tools Used:

- Cypress is used for automating browser-based tests to simulate user interactions such as searching for flights, selecting seats, and completing a booking.

Example Tests:

1. User Journey: Flight Booking
- Verifying that a user can log in, search for flights, view available flights, and successfully book a flight.
2. Admin Dashboard
- Verifying that an admin can log in, view all users, flights, and bookings, and approve flight operators.
3. Flight Operator Panel
- Ensuring that flight operators can log in, create new flights, and delete existing flights.

9.3.2 Manual Testing:

Description:

Manual testing is conducted to check the usability, accessibility, and responsiveness of the "Flight Booking System" on different devices and browsers. It includes ensuring smooth user experience and functionality of all key features such as booking flights, selecting seats, and handling payment processing.

Scope:

- Verifying layout consistency across different pages (e.g., search page, booking page).
- Ensuring that form inputs work correctly (e.g., login, booking forms).
- Checking for proper error messages when invalid data is entered.
- Testing mobile responsiveness to ensure the application looks good on smartphones and tablets.

9.3.3 Code Coverage:

Description:

Code coverage helps ensure that a significant portion of the codebase is tested by both unit tests and integration tests. It identifies areas of the code that may need further testing, ensuring the application is thoroughly tested before deployment.

Tools Used:

- Istanbul: For measuring code coverage when running unit and integration tests with Jest (for React) and Mocha (for backend).

9.3.4 Continuous Integration (CI):

Description:

A CI pipeline is set up to automatically run tests on every commit or pull request to the repository. This helps prevent new changes from introducing bugs or regressions into the system.

CI Tools:

- GitHub Actions: Automatically runs Cypress tests, Jest unit tests, and Mocha tests for every new code change.
- Code Coverage Integration: The CI pipeline also generates code coverage reports to track test coverage for each commit or pull request.

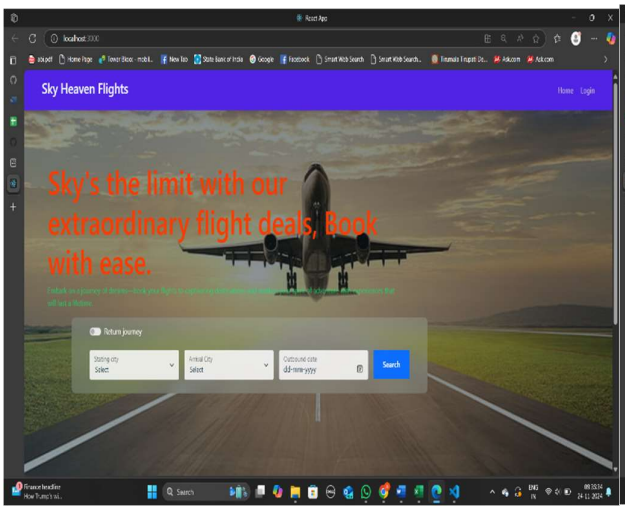
CHAPTER 10

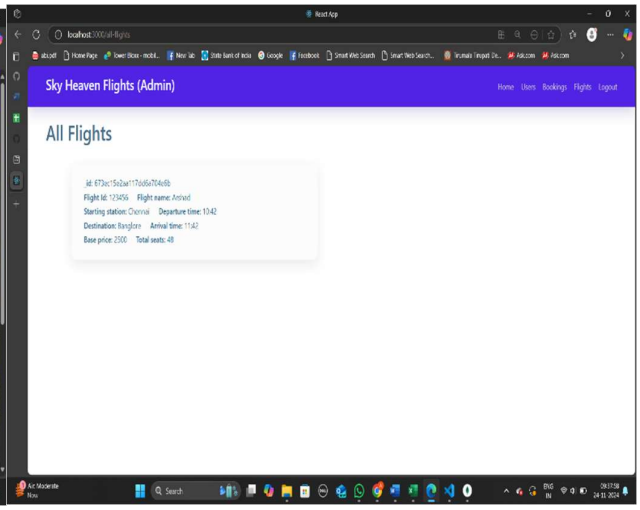
10.SCREENSHOTS AND DEMO:

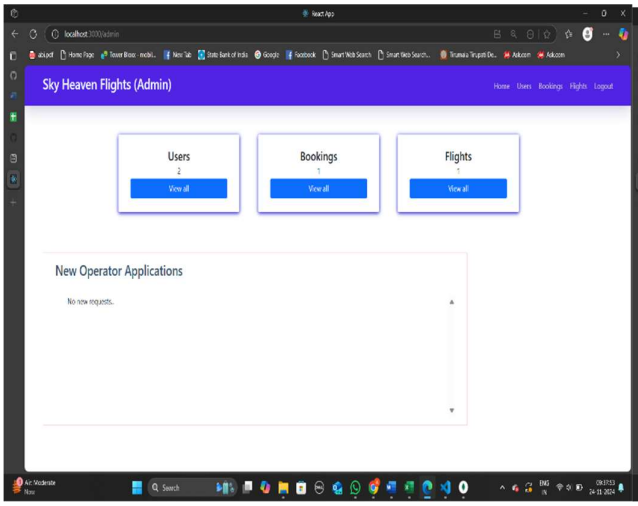
10.1 DEMO VIDEO LINK :

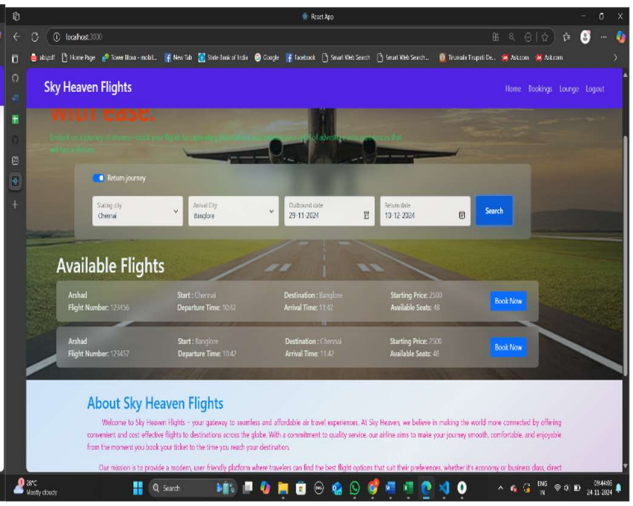
https://drive.google.com/file/d/1wF74aZP2sZDogIck7V0BHkdhRLM3Zvv5/view?usp=sharing

10.2 SCREENSHOTS:









CHAPTER 11

11.ADVANTAGES

A flight booking system developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js) offers several advantages due to the efficiency, scalability, and flexibility of the stack. Here are some key benefits:

- **Full-Stack JavaScript**
  The entire development process uses a single language (JavaScript), simplifying development and enabling seamless integration between frontend and backend.
- **Scalability**
  MongoDB's flexibility and Node.js's ability to handle multiple requests make the system highly scalable, perfect for managing large volumes of flight data and user activity.
- **Real-Time Data Updates**
  React.js allows for dynamic user interfaces that update flight availability, pricing, and seat selection in real-time without page reloads, enhancing user experience.
- **Fast Performance**
  Node.js's non-blocking architecture ensures fast, asynchronous handling of multiple requests, providing a smooth experience even under high traffic.
- **Cost-Effective Development**
  The MERN stack is open-source, lowering development costs, and has a large community, which speeds up development with readily available resources.
- **Cross-Platform Compatibility**
  React enables building responsive web applications that work across desktop and mobile, while React Native can be used for mobile app development if needed.
- **Modular and Maintainable**
  React's component-based architecture allows independent development and updates of key features like flight search, booking, and payments, making the system more maintainable.

.

CHAPTER 12

12.DISADVANTAGES

While the MERN stack (MongoDB, Express.js, React.js, and Node.js) offers many advantages for building dynamic, scalable applications, it comes with certain challenges for a complex system like flight booking. Here are the key disadvantages:

1. **Limited Support for Complex Transactions**
   MongoDB is a NoSQL database, making it highly flexible but less ideal for handling complex, multi-step transactions that require strict data consistency, like those involved in booking a flight. While MongoDB supports ACID transactions, traditional relational databases like PostgreSQL or MySQL are often better suited for such operations due to their more mature transaction handling.
2. **Handling Heavy Server-Side Processing**
   Node.js, being single-threaded, can efficiently handle I/O-heavy tasks but may struggle with CPU-intensive processes. In a flight booking system, features like complex pricing algorithms, real-time availability checks, or route optimization might become bottlenecks, as Node.js isn't well-suited for intensive computations without additional optimization or external services.
3. **Steep Learning Curve for React**
   React.js, while powerful, introduces complexity with its component-based structure, hooks, and state management. For a flight booking system with multiple dynamic components (e.g., search filters, seat selection, and payment processing), managing the state efficiently can be challenging, especially for teams unfamiliar with React's advanced concepts.
4. **Challenges with SEO**
   Single-page applications (SPAs) built with React.js can be less SEO-friendly, which could impact the discoverability of the flight booking system through search engines. Solutions like server-side rendering (SSR) or using frameworks like Next.js can address this issue but add complexity to the development process.
5. **Scalability Issues with MongoDB**
   While MongoDB excels at handling large datasets, querying large amounts of structured data (like flight schedules, passenger data, or payment histories) across multiple collections may lead to performance issues. Traditional SQL databases with better support for complex queries and indexing may be more efficient for large-scale applications.
6. **Complex State Management in React**
   As the flight booking system grows, managing the global state (e.g., flight search results, selected flights, seat availability, and user profiles) can become cumbersome. Libraries like Redux or Context API help manage state, but they introduce complexity, requiring careful planning and implementation.
7. **Security Considerations**
   Express.js is a minimalist framework, meaning many essential security features (like session management, XSS, CSRF protection) need to be manually implemented. This puts additional responsibility on developers to ensure the system is secure, particularly when handling sensitive user data and payment information

CHAPTER 13

13.FUTURE ENHANCEMENTS:

The "Flight booking" application is designed to be scalable and open to future improvements. Here are some potential features and enhancements planned to improve user experience and expand functionality:

1. **Advanced Search and Filtering**
   - Implement personalized flight recommendations using machine learning, which can analyze user preferences, past bookings, and travel history. This will provide users with relevant flight options that align with their preferences.
   - Add advanced filtering options like specific departure times, layover durations, baggage policies, and flight flexibility (refundable vs. non-refundable), allowing users to tailor search results to their needs.
2. **Real-Time Updates and Synchronization**
   - Incorporate real-time seat availability using WebSockets or real-time APIs to ensure users are always viewing the latest seat options and availability. This minimizes the risk of booking conflicts.
   - Include real-time flight tracking, providing users with live updates about delays, gate changes, and cancellations directly within the platform.
3. **Mobile App Development**
   - Extend the platform by developing a mobile app using React Native. The app can offer full functionality such as flight search, booking, seat selection, and payment. This allows users to manage their bookings on-the-go.
   - Integrate push notifications to alert users about important updates such as flight schedule changes, upcoming travel reminders, or promotional deals.
4. **Dynamic Pricing and Prediction**
   - Use machine learning models to predict flight prices based on factors such as seasonality, demand, and market trends. Users can receive insights into the best time to book to get the lowest prices.
   - Implement dynamic pricing algorithms that adjust ticket prices in real-time based on user demand and seat availability, helping airlines optimize revenue.
5. **Enhanced Payment and Security**
   - Support multi-currency payments and integrate popular payment gateways for international users, making the platform accessible to a global audience.
   - Improve security by implementing two-factor authentication (2FA) for login and encrypted payment methods such as Apple Pay and Google Pay to protect user data and ensure secure transactions.
6. **AI-Powered Customer Support**
   - Integrate an AI-powered chatbot capable of handling common customer queries, booking assistance, and even flight cancellations or changes. The chatbot can offer 24/7 support, reducing the need for human customer service.
   - The chatbot could also support multiple languages, providing a more inclusive experience for users from different regions. This can improve overall user engagement and satisfaction.