Flight Management App

Project Overview

Tech Stack: React (Frontend) with Vite – Express (Backend) – MongoDB (Database) – Node.js – TailwindCSS (Styling)

Description

The Flight Tracking App is designed to manage a list of flights across India. This app centralizes flight data and allows users to easily perform CRUD (Create, Read, Update, Delete) operations, ensuring that the flight tracking data in the database is current and accurately reflects the travel ID , travel destination, travel duration and cost of the journey.

Table of Contents

---

1. Project Structure

Final_sk_dev/

── backend/

    ── controllers/

        ── flightController.js

    ── models/

        ── flight.js

    ── routes/

        ── flightRoute.js

    ── index.js

    ──.env

── frontend/

    ── public/

    ── src/

        ── components/

        ── App.jsx

    ── index.html

    ── package.json

- **backend/**: Contains the Node.js and Express.js API server
    - **controllers/**: Handles business logic for CRUD operations
    - **models/**: Mongoose models for MongoDB schema
    - **routes/**: API routes for trainer CRUD operations
- **frontend/**: Contains the React application for UI
    - **components/**: Reusable UI components
    - **src/**: Full-page components (e.g., Add Flight, Edit Flight)

---

2. Database Design

Collections: flightmodels

Field	Type	Description
_id	ObjectId	Unique identifier for each flight
fight_id	String	ID of the flight of the user
airline	String	Airline name of the user
destination	String	Destination of the flight
fare	Number	Cost of the flight
duration	Number	Time of the flight

---

3. API Endpoints

Base URL: http://localhost:3000

Method	Endpoint	Description	Request Body	Response
GET	/api/flightmodels	Fetch all flights	-	Array of flights

GET	/api/flightmodels/:id	Fetch specific flight	-	Flight details
POST	/api/flightmodels	Add new flight	{ flight_id, airline, destination, fare, duration}	Creates new flight
PUT	/api/ flightmodels /:id	Update trainer by ID	{ flight_id, airline, destination, fare, duration }	Updates flight details in the list
DELETE	/api/ flightmodels /:id	Delete trainer by ID	-	Flight detail will be removed from list

---

4. Frontend Components

Pages

1. **HomePage** (/)
    - Displays a list of all flights with add, edit and delete buttons.
2. **Add Flight Page** (/AddFlight)
    - Contains a form for adding a new trainer with fields: flight_id, airline, destination, fare, duration.
3. **Edit Flight Page** (/EditFlight/:id)
    - Form pre-filled with existing flight data for updating.

Components

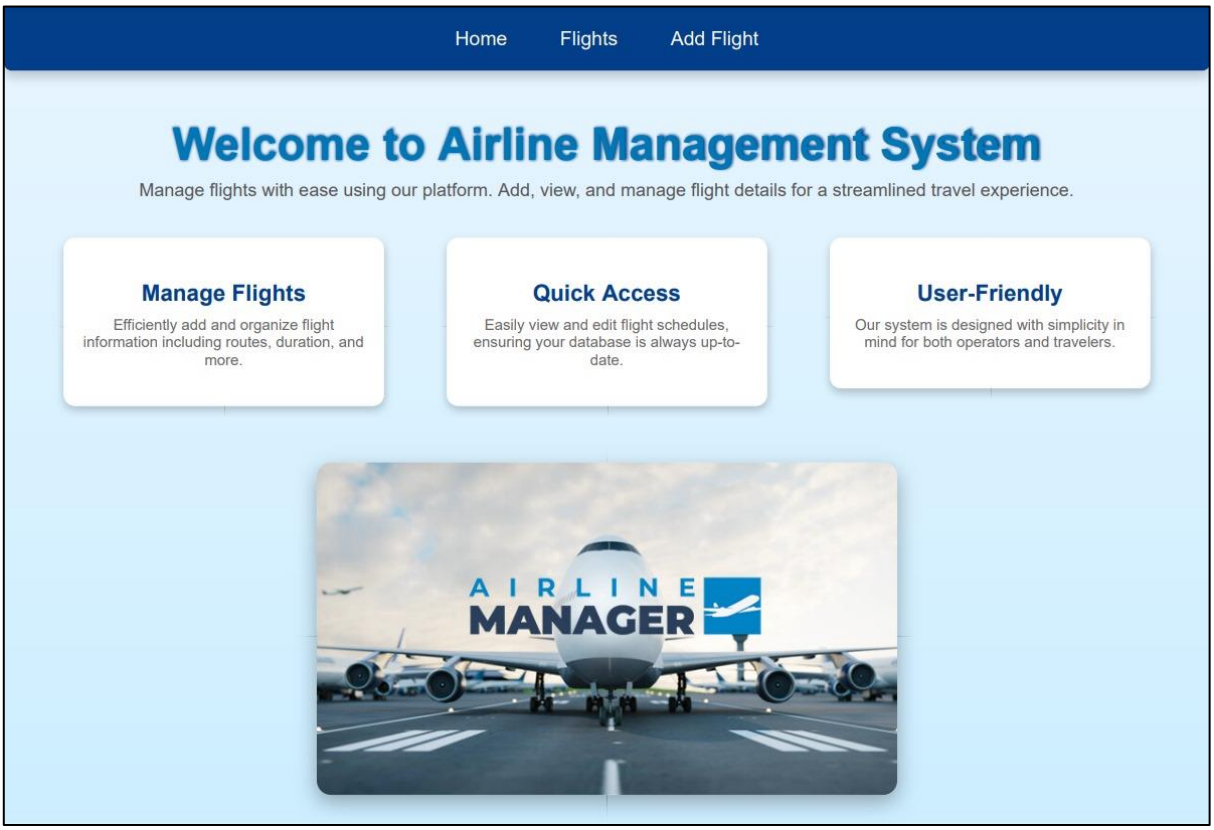
- **AddFlight** : Component to add flight to list of flights.
- **EditFlight** : Reusable form for editing the flight details.
- **GetFlight** : Functionality for deleting a flight from the database
- **HomePage** : Main routing file with routes for performing the CRUD operations and displaying all the flight details
- **Header** : Header display in all the pages
- **Github** : Link to github profile
- **Footer** : Footer display in all the pages
- **Contact** : Contact details to reach out to admins
- **About** : About description of the website

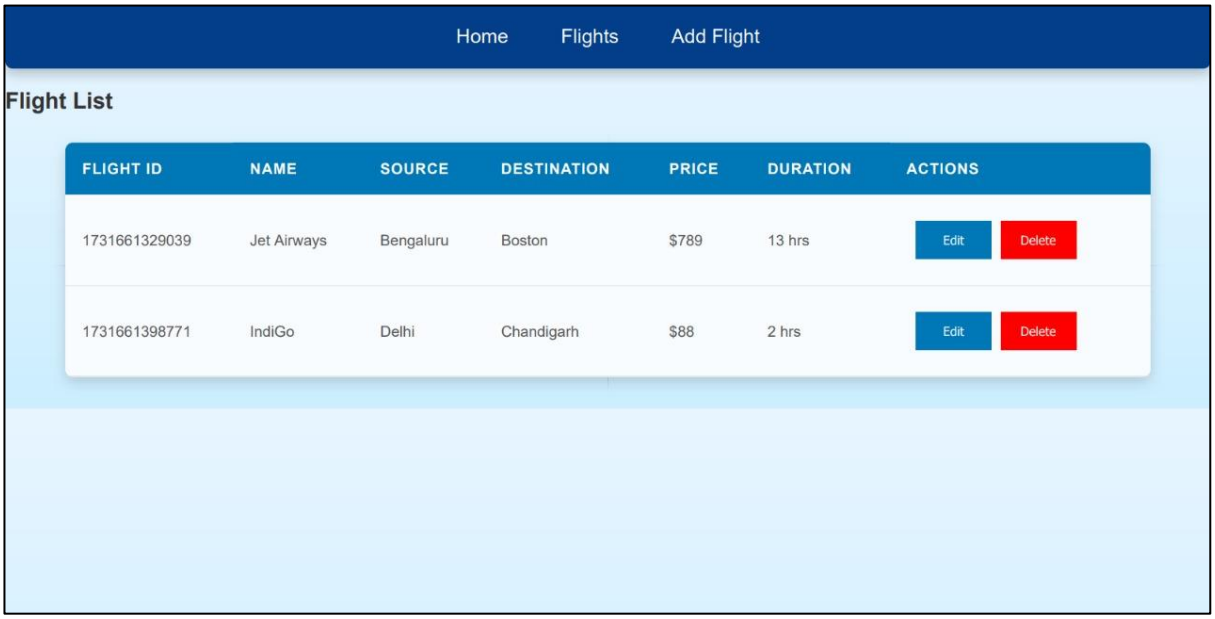
Routes Configuration (App.jsx)

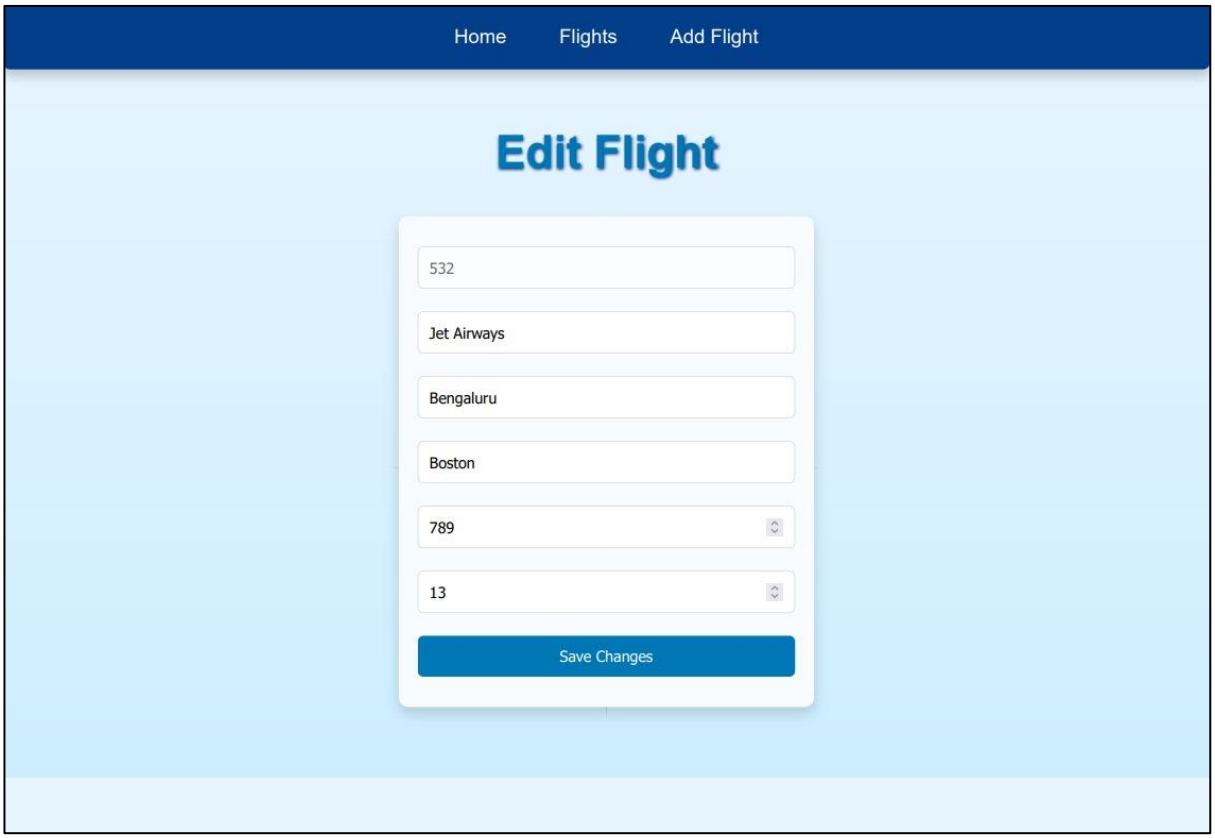
```
import './App.css'
import React from 'react'
import Header from './components/Header/Header'
import Footer from './components/Footer/Footer'
function App() {
  return (
    <>
    <Header/>
    <Footer/>
    </>
  )
}
export default App
```

---

5. Frontend Images




Home
Flights
Add Flight
Flight List
FLIGHT ID
NAME
SOURCE
DESTINATION
PRICE
DURATION
ACTIONS
1731661329039
Jet Airways
Bengaluru
Boston
$789
13 hrs
Edit
Delete
1731661398771
IndiGo
Delhi
Chandigarh
$88
2 hrs
Edit
Delete


Home
Flights
Add Flight
Edit Flight
532
Jet Airways
Bengaluru
Boston
789
13
Save Changes

6. Backend Architecture

1. **Controllers**: Functions that handle CRUD operations for trainers, such as getAllItems, getItemById, createItem, updateItem, and deleteItem.
2. **Routes**: Define routes and attach respective controller functions.
3. **Server Setup (app.js)**:
    - Initializes Express server, connects to MongoDB using Mongoose, cors and sets up API routes.

Example Controller (flight.js)

```
const Item = require("../models/flight");

exports.getAllItems = async (req, res) => {
  try {
    const items = await Item.find();
    res.status(200).json(items);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};


exports.createItem = async (req, res) => {
  try {
    const item = new Item(req.body);
    const savedItem = await item.save();
    res.status(201).json(savedItem);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

```
exports.updateItem = async (req, res) => {
  try {
    const updatedItem = await Item.findByIdAndUpdate(req.params.id, req.body, { new: true });
    if (!updatedItem) return res.status(404).json({ message: "Item not found" });
    res.status(200).json(updatedItem);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

exports.deleteItem = async (req, res) => {
  try {
    const deletedItem = await Item.findByIdAndDelete(req.params.id);
    if (!deletedItem) return res.status(404).json({ message: "Item not found" });
    res.status(200).json({ message: "Item deleted successfully" });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

---

7. Additional Functionalities

- **Validation**: Use validation on both frontend (React) and backend (Express/Mongoose).
- **Styling**: Add TailwindCSS for a user-friendly interface.

---

8. Setup and Installation

Prerequisites

- Node.js and npm
- MongodbCompass and Mongodb Community Server and mongosh installed and running

Steps

1. **Backend Setup**:
   - cd backend
   - npm install mongodb mongoose express dotenv
2. **Environment Variables**:
   - Create a .env file in the backend/ folder with:

     PORT=3000

     MONGO_URI=mongodb://localhost:27017/flightmodels
3. **Frontend Setup**:

   cd frontend

   npm create vite@latest “frontend”

   cd frontend

   npm install
4. **Running the Application**:
   - Start backend server:

     nodemon server.js
   - Start frontend server:

     npm run dev

     The app should be running locally at http://localhost:5173