**Project Structure**

**bash**

**CopyEdit**

**server/**

**│**

**├── index.js # Entry point**

**├── schemas.js # Mongoose schemas**

**├── routes/**

**│ ├── auth.js # User auth routes**

**│ ├── flight.js # Flight-related APIs**

**│ └── booking.js # Booking APIs**

**└── middleware/**

**└── authMiddleware.js # JWT verification**

**📦 Install Backend Dependencies**

**bash**

**CopyEdit**

**npm install express mongoose body-parser cors bcrypt jsonwebtoken**

**1. index.js (Main Entry Point)**

**js**

**CopyEdit**

**import express from 'express';**

**import mongoose from 'mongoose';**

**import bodyParser from 'body-parser';**

**import cors from 'cors';**

**import authRoutes from './routes/auth.js';**

**import flightRoutes from './routes/flight.js';**

**import bookingRoutes from './routes/booking.js';**

**const app = express();**

**const PORT = 5000;**

**app.use(cors());**

**app.use(bodyParser.json());**

**mongoose.connect("mongodb://localhost:27017/flightbooking")**

**.then(() => console.log("MongoDB connected"))**

**.catch(err => console.error(err));**

**app.use("/api/auth", authRoutes);**

**app.use("/api/flights", flightRoutes);**

**app.use("/api/bookings", bookingRoutes);**

**app.listen(PORT, () => console.log(`Server running on port ${PORT}`));**

**2. schemas.js (Mongoose Schemas)**

**js**

**CopyEdit**

**import mongoose from 'mongoose';**

**export const userSchema = new mongoose.Schema({**

**username: String,**

**email: String,**

**password: String,**

**userType: String,**

**approval: Boolean**

**});**

**export const flightSchema = new mongoose.Schema({**

**flightName: String,**

**origin: String,**

**destination: String,**

**departureTime: String,**

**arrivalTime: String,**

**basePrice: Number,**

**totalSeats: Number**

**});**

**export const bookingSchema = new mongoose.Schema({**

**userId: mongoose.Schema.Types.ObjectId,**

**flightId: mongoose.Schema.Types.ObjectId,**

**seats: Number,**

**status: String,**

**journeyDate: String,**

**seatClass: String,**

**bookingDate: String,**

**price: Number**

**});**

**export const User = mongoose.model("User", userSchema);**

**export const Flight = mongoose.model("Flight", flightSchema);**

**export const Booking = mongoose.model("Booking", bookingSchema);**

**3. routes/auth.js (User Authentication)**

**js**

**CopyEdit**

**import express from 'express';**

**import bcrypt from 'bcrypt';**

**import jwt from 'jsonwebtoken';**

**import { User } from '../schemas.js';**

**const router = express.Router();**

**const SECRET = "FlightAppSecret";**

**// Register**

**router.post('/register', async (req, res) => {**

**const { username, email, password, userType } = req.body;**

**const hashedPassword = await bcrypt.hash(password, 10);**

**const newUser = new User({ username, email, password: hashedPassword, userType, approval: true });**

**await newUser.save();**

**res.status(201).json("User Registered");**

**});**

**// Login**

**router.post('/login', async (req, res) => {**

**const { email, password } = req.body;**

**const user = await User.findOne({ email });**

**if (user && await bcrypt.compare(password, user.password)) {**

**const token = jwt.sign({ id: user.\_id, userType: user.userType }, SECRET);**

**res.json({ token });**

**} else {**

**res.status(401).json("Invalid Credentials");**

**}**

**});**

**export default router;**

**4. routes/flight.js (Flight Management)**

**js**

**CopyEdit**

**import express from 'express';**

**import { Flight } from '../schemas.js';**

**import verifyToken from '../middleware/authMiddleware.js';**

**const router = express.Router();**

**// Add flight (Admin)**

**router.post('/add', verifyToken, async (req, res) => {**

**const newFlight = new Flight(req.body);**

**await newFlight.save();**

**res.status(201).json("Flight Added");**

**});**

**// Get all flights**

**router.get('/', async (req, res) => {**

**const flights = await Flight.find();**

**res.json(flights);**

**});**

**export default router;**

**5. routes/booking.js (Booking)**

**js**

**CopyEdit**

**import express from 'express';**

**import { Booking } from '../schemas.js';**

**import verifyToken from '../middleware/authMiddleware.js';**

**const router = express.Router();**

**// Book a flight**

**router.post('/book', verifyToken, async (req, res) => {**

**const booking = new Booking({ ...req.body, userId: req.user.id });**

**await booking.save();**

**res.status(201).json("Booking Confirmed");**

**});**

**// Get user bookings**

**router.get('/mybookings', verifyToken, async (req, res) => {**

**const bookings = await Booking.find({ userId: req.user.id });**

**res.json(bookings);**

**});**

**export default router;**

**6. middleware/authMiddleware.js (JWT Middleware)**

**js**

**CopyEdit**

**import jwt from 'jsonwebtoken';**

**const SECRET = "FlightAppSecret";**

**export default function verifyToken(req, res, next) {**

**const token = req.headers.authorization?.split(" ")[1];**

**if (!token) return res.status(403).json("Access Denied");**

**try {**

**const verified = jwt.verify(token, SECRET);**

**req.user = verified;**

**next();**

**} catch (err) {**

**res.status(401).json("Invalid Token");**

**}**

**}**

**Backend Development**

**9.1 Database Configuration**

* **Set up a MongoDB database either locally using MongoDB Compass or via a cloud-based service like MongoDB Atlas.**
* **Create a new database and define collections for:**
  + **Users**
  + **Flights**
  + **Bookings**
  + **Any other relevant entities**

**9.2 Create Express.js Server**

* **Initialize an Express.js server to manage HTTP requests and API endpoints.**
* **Install and configure necessary middleware:**
  + **body-parser for parsing request bodies**
  + **cors for managing Cross-Origin Resource Sharing**

**9.3 Define API Routes**

* **Create modular route files for different features:**
  + **flights.js**
  + **users.js**
  + **bookings.js**
  + **auth.js**
* **Define endpoints for:**
  + **Flight listing**
  + **User registration/login**
  + **Managing bookings**
* **Implement route handlers to process requests and communicate with the database.**

**9.4 Implement Data Models**

* **Define Mongoose Schemas for:**
  + **Flights**
  + **Users**
  + **Bookings**
* **Create Mongoose models and define:**
  + **CRUD operations: Create, Read, Update, Delete**
  + **Relationships (e.g., user-booking, flight-booking)**

**9.5 User Authentication**

* **Set up routes for:**
  + **Register**
  + **Login**
  + **Logout**
* **Use middleware to:**
  + **Protect private routes**
  + **Verify authentication tokens (JWT or session-based)**

**9.6 Handle New Flights and Bookings**

* **Create endpoints for:**
  + **Fetching and listing available flights**
  + **Booking seats with validation and database updates**
* **Implement backend controllers to manage this logic efficiently.**

**9.7 Admin Functionality**

* **Admin-specific operations include:**
  + **Adding new flights**
  + **Managing all user bookings**
* **Add role-based access control to restrict admin features.**

**9.8 Error Handling**

* **Use Express error handling middleware to catch exceptions.**
* **Send back proper HTTP status codes and messages (e.g., 400, 401, 404, 500)**

**Backend Development**

1. **Database Configuration:**

* Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
* Create a database and define the necessary collections for flights, users, bookings, and other relevant data.

1. **Create Express.js Server:**

* Set up an Express.js server to handle HTTP requests and serve API endpoints.
* Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

1. **Define API Routes:**

* Create separate route files for different API functionalities such as flights, users, bookings, and authentication.
* Define the necessary routes for listing flights, handling user registration and login managing bookings, etc.
* Implement route handlers using Express.js to handle requests and interact with the database.

1. **Implement Data Models:**

* Define Mongoose schemas for the different data entities like flights, users, and bookings.
* Create corresponding Mongoose models to interact with the MongoDB database. Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

1. **User Authentication:**

* Create routes and middleware for user registration, login, and logout.
* Set up authentication middleware to protect routes that require user authentication.

1. **Handle new Flights and Bookings:**

* Create routes and controllers to handle new flight listings, including fetching flight data from the database and sending it as a response.
* Implement booking functionality by creating routes and controllers to handle booking requests, including validation and database updates.

1. **Admin Functionality:**

* Implement routes and controllers specific to admin functionalities such as adding flights, managing user bookings, etc.
* Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

1. **Error Handling:**

* Implement error handling middleware to catch and handle any errors that occur during the API requests.
* Return appropriate error responses with relevant error messages and HTTP status codes.

**Reference video for backend code:**

<https://drive.google.com/file/d/11iNDCz0lJIv9zAS8nYbjrc1JXTPGtwsi/view?usp=drive_link>