Flight Booking App Documentation

Project Overview

- **Project Name:** Flight-Booking-App
- **Project Type:** Web-based Airline Reservation System
- **Technologies Used:**
    - **Frontend:**
        - HTML5 (Structure)
        - CSS3 (Styling)
        - JavaScript (Client-side Logic)
    - **Backend:**
        - Node.js (Runtime Environment)
        - Express.js (Web Framework)
    - **Database:**
        - MongoDB (NoSQL Database Management System)
- **Project Description:**
    - The Flight-Booking-App is a web-based application designed to facilitate the booking of flights, management of user information, and handling of flight schedules.
    - The application aims to provide a user-friendly interface for customers to search, select, and book flights, while also offering administrative functionalities for managing flights and user accounts.

System Architecture

- **Architecture Pattern:** Microservices-inspired, with a focus on scalability
- **Components:**
  - **Client-Side (Web App):** User Interaction, Booking Management
  - **Server-Side (Node.js/Express):** API Gateway, Business Logic, Database Integration
  - **Database (MongoDB):** Storage for User Data, Flight Schedules, Bookings
- **Data Flow:**
  - **Client** → **Server** (API Requests)
  - **Server** → **Database** (CRUD Operations)
  - **Database** → **Server** (Data Retrieval)
  - **Server** → **Client** (Response with Data)
- **Key Technologies:**
  - **Node.js** for scalable server-side operations
  - **Express.js** for efficient API routing and management
  - **MongoDB** for flexible and scalable data storage
- **Architecture Benefits:**
  - Scalability and Flexibility
  - Efficient Data Management
  - Enhanced User Experience

Key Features

- **User Management:**
    - **Registration:**
        - Secure user registration with email verification
        - Optional: Social media login integration (e.g., Google, Facebook)
    - **Login/Logout:**
        - Session-based login with automatic logout after inactivity
        - Optional: Two-Factor Authentication (2FA) for enhanced security
- **Flight Management:**
    - **View Available Flights:**
        - Filter flights by departure, arrival, date, and time
        - Display flight details, including duration and layovers
    - **Search Flights by:**
        - Destination
        - Date (specific or range)
        - Flight number
- **Booking Management:**
    - **View Booking History:**
        - Users can view their past and upcoming bookings
        - Booking status updates (e.g., confirmed, cancelled)
    - **Cancel Bookings (with constraints):**
        - Time-sensitive cancellation policy (e.g., 24-hour window)
        - Automatic refund processing (if applicable)
- **Admin Dashboard:**
    - **Manage:**
        - User accounts (activate, deactivate, or delete)
        - Flight schedules (update, cancel, or add new)
        - Booking settings (cancellation policies, payment gateways)

Backend Implementation

- Node.js Version: 14.17.0
- Express.js Version: 4.17.1
- MongoDB Version: 3.6.3
- API Endpoints:
  a. /users (CRUD Operations for Users)
  b. /flights (CRUD Operations for Flights, Booking Management)
  c. /bookings (Booking-specific Operations)

User Authentication (Login)

- File: controllers/authController.js
- Endpoint: POST /users/login
- Code:

```
const bcrypt = require('bcrypt');
const User = require('../models/User');

exports.login = async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });

  if (!user ||!(await bcrypt.compare(password, user.password))) {
    return res.status(401).send('Invalid credentials');
  }

  // Generate JSON Web Token (JWT) for authenticated user
  const token = generateToken(user);
  res.json({ token, user: { id: user._id, email: user.email } });
};
```

- This code snippet handles user login by verifying the provided email and password against the stored user data in the MongoDB database.
- It uses the bcrypt library for secure password comparison.
- Upon successful authentication, a JSON Web Token (JWT) is generated for the user.

Frontend Implementation

- Frontend Framework: Vanilla JavaScript with HTML/CSS
- User Interface Components:
    - Navigation Bar
    - Flight Search Bar
    - Flight Listing Component
    - Booking Form
    - User Profile Dashboard

```
<!-- index.html (simplified example) -->
<form id="flight-search-form">
  <input type="text" id="destination" placeholder="Destination">
  <input type="date" id="travelDate">
  <button id="search-flights-btn">Search Flights</button>
</form>
```

HTML form for users to input destination and travel date.

```
// script.js (simplified example)
const searchFlightsBtn = document.getElementById('search-flights-btn');

searchFlightsBtn.addEventListener('click', async (e) => {
  e.preventDefault();
  const destination = document.getElementById('destination').value;
  const travelDate = document.getElementById('travelDate').value;
  try {
    const response = await fetch(`/flights?
destination=${destination}&date=${travelDate}`);
    const flights = await response.json();
    // Display flights
  } catch (err) {
    console.error(err);
  }
});
```

HTML form for users to input destination and travel date.

Database Schema

- Database Management System: MongoDB

Collections:

Users

- _id (ObjectId, primary key)
- username (String, unique)
- email (String, unique)
- password (String, hashed)
- role (String, default: 'user')

Flights

- _id (ObjectId, primary key)
- flightNumber (String, unique)
- departure (String)
- arrival (String)
- departureTime (Date)
- arrivalTime (Date)
- seatsAvailable (Number)

Bookings

- _id (ObjectId, primary key)
- userId (ObjectId, references Users._id)
- flightId (ObjectId, references Flights._id)
- bookingDate (Date)
- status (String, default: 'pending')

```
// models/User.js (simplified example)
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, default: 'user' }
});

module.exports = mongoose.model('User', userSchema);
```

Deployment Steps

- **Prerequisites:**
- Node.js (14.17.0+) installed
- MongoDB (3.6.3+) installed and running
- Git CLI for repository cloning

- **Deployment Steps:**
  - **Clone Repository:**

    - `git clone https://github.com/seshandonan/Flight-Booking-App.git`

  - **Install Dependencies:**

    - `npm install`

  - **Configure Environment Variables:**
    - Create **.env** file with **MONGODB_URI** and **PORT** settings
  - **Start Application:**
    - **node app.js** (development mode)
    - Use PM2 or similar for production environments

- **Deployment Options:**
- **Local Machine** for development and testing
- **Cloud Platforms** (e.g., AWS, Google Cloud, Microsoft Azure)
- **Containerization** (e.g., Docker) for efficient deployment
- **Post-Deployment:**
- Verify application functionality
- Configure logging and monitoring tools (e.g., Prometheus, Grafana)

Additional Resources

- **Node.js Documentation:** https://nodejs.org/en/docs/
- **Express.js Documentation:** https://expressjs.com/en/starter/installing.html
- **MongoDB Documentation:** https://docs.mongodb.com/

Team Members

GODWIN JOSEPH V

GOKULNATH B

GOPI A

HARSHA VARTHAN