1. Introduction:

1.1 Project Title: Airlines management sytem

1.2 **Team ID :** LTVIP2025TMID31350

Team Leader : Sree Satya Kammula

Team member : Rajulapati Satyasree

Team member : Renuka Maddi

Team member : Polukonda Renuka Devi

2. Project Overview:

The Airlines Management System is a software-based application designed to simplify and automate the key operations of airline companies. It enables efficient management of flight schedules, reservations, passenger records, ticketing, and staff assignments. The system provides a centralized platform where administrators, staff, and customers can interact securely and seamlessly.

The project will be implemented using a user-friendly interface and a database back-end to store and retrieve essential flight and passenger information. It supports real-time booking, cancellation, seat availability checks, and flight status updates.

2.1 Purpose of the Project:

The primary purpose of the Airlines Management System is to:

- ✅ Automate airline operations such as flight scheduling, ticket bookings, and staff management.
- ✅ Improve customer experience by providing a fast, transparent, and reliable booking process.
- ✅ Minimize manual errors and reduce workload by shifting from paper-based systems to digital records.
- ✅ Enable real-time access to critical data such as seat availability, flight status, and passenger records.
- ✅ Secure data handling for both administrative tasks and customer transactions.

2.2 Features:

🛫 1. Flight Management:

- Add, update, or delete flight details.
- Manage flight numbers, departure/arrival times, routes, duration, and aircraft type.
- Track active and cancelled flights.

2. Staff Management:

- Manage pilot and cabin crew assignments.
- Store staff details such as name, role, experience, and shift schedules.

3. User Registration & Authentication:

- Secure login and registration for admins, staff, and passengers.
- Role-based access: Admin, Staff, and Customer dashboards.

4. Ticket Booking System:

- Real-time flight search and seat availability.
- Book tickets with automatic seat assignment.
- Payment simulation (basic, for academic purposes).

5. Ticket Cancellation:

- Cancel booked tickets.
- Update seat availability automatically upon cancellation.
- Refund simulation or tracking.

6. Seat Management:

- Visual representation of seat layout (optional GUI feature).
- Track occupied vs available seats for each flight.

7. Passenger Management:

- View passenger details and booking history.

• Maintain a secure record of customer information.

8. Reports Generation:

• Generate reports on:

o Booked vs available seats.

o Flight occupancy.

o Daily/monthly bookings and revenue (simulation).

9. Search and Filter Options:

• Search flights based on origin, destination, date, and time.

• Filter bookings and records for quick admin access.

10. Optional: Web-Based Interface:

(If you are building it as a web application)

• Responsive user interface for mobile and desktop.

• Online booking and admin portal via browser.

3.Architecture:

3.1 Frontend:

1. Component-Based Structure:

• The UI is split into reusable and self-contained components, each handling a specific part of the application.

• Components include:

o Login, Register, FlightList, BookTicket, CancelTicket

o AdminDashboard, UserDashboard, FlightForm, etc.

• This makes development modular, testable, and easy to maintain.

2. Routing System:

• React Router is used for client-side routing.

• It handles navigation between views like:

o /login, /register, /flights, /book/:id, /dashboard

• Ensures smooth transitions without reloading the page.

jsx

```
<Route path="/login" element={<Login />} />
<Route path="/flights" element={<FlightList />} />
```

3. State Management:

• Local state is managed using React’s useState and useEffect.

• Context API or Redux (optional) can be used for global state:

o User authentication state

o Booked tickets data

o Flight details cache

4. API Integration:

• React frontend interacts with the backend through RESTful APIs.

• Communication is handled using:

o Axios or native fetch()

• API endpoints for:

o Fetching flight data

o Booking/canceling tickets

o User authentication

5. Form Handling & Validation:

• Forms for login, registration, booking use controlled components.

• Libraries like React Hook Form or Formik may be used for:

o Input validation

o Form submission

o Error handling

o CSS Modules, Tailwind CSS, or Bootstrap for responsiveness

o Icons and visual aids using Font Awesome or Material Icons

6. Authentication and Protected Routes:

• JWT (JSON Web Token) is used to store login tokens in localStorage.

• Protected routes ensure only authorized users can access certain pages (e.g., Admin Dashboard).

```
src/
 ├── components/
|    ├── FlightList.jsx
|    ├── BookTicket.jsx
|    ├── Login.jsx
|   └── ...
 ├── pages/
|    ├── AdminDashboard.jsx
|    ├── UserDashboard.jsx
 ├── services/
|    ├── api.js
 ├── App.js
 ├── index.js
```

3.2 Backend:

The backend of the Airlines Management System is developed using Node.js with the Express.js framework. This architecture is designed to efficiently handle RESTful API requests, perform database operations, and manage business logic securely and reliably.

1. Server Framework – Express.js:

• Express.js acts as the HTTP server for handling API requests and routing.

• Provides clean structure for defining endpoints like:

o GET /api/flights – List all flights

o POST /api/bookings – Book a flight

o DELETE /api/bookings/:id – Cancel ticket

o POST /api/login – Authenticate user

2. Routing Layer:

• Routes are modularized by feature:

o authRoutes.js – Login & registration

o flightRoutes.js – Flight CRUD operations

o bookingRoutes.js – Ticket booking and cancellation

o adminRoutes.js – Admin-specific controls

3. Controllers / Business Logic Layer:

• Contains functions to handle actual business logic:

o Validating inputs

o Processing booking logic

o Checking seat availability

o Sending response to frontend

3.3 Database:

The Airlines Management System uses MongoDB as its NoSQL database to store and manage dynamic, scalable data such as users, flights, bookings, and tickets. MongoDB collections are used instead of tables, and documents instead of rows.

4.Setup Instructions:

4.1 Prerequisites: Software Dependencies & Requirements

To develop and run the Airlines Management System the following software and tools are required:

System Requirements:

• Operating System: Windows 10/11, Linux, or macOS

• RAM: Minimum 4 GB (8 GB recommended)

• Node.js Version: v14 or higher

• MongoDB Version: v5.0 or higher (can use MongoDB Atlas or local setup)

• Backend Dependencies (Node.js & Express.js)

Dependency

Node.js

Express.js

Mongoose

Dotenv

Cors

Bcryptjs

Purpose

JavaScript runtime for backend

Web framework for routing and APIs

ODM for MongoDB (schema & DB interaction)

Environment variable management

Enable cross-origin requests

Password hashing

Jsonwebtoken Token-based authentication (JWT)

Nodemon

Auto-restart server on file changes (dev)

4.2 Features:

Step-by-step guide to clone, install dependencies, and set up the environment variables.

✅ Step 1: Prerequisites:

Before starting, make sure you have the following installed:

• Node.js (v14 or higher)

• MongoDB (local or Atlas)

• Git

• VS Code

• Postman (optional for API testing)

✅ Step 2: Clone the Repository:

• git clone https://github.com/your-username/airlines-management-system.git

• cd airlines-management-system

• Replace the GitHub link with your actual repository URL.

•

✅ Step 3: Set Up the Backend:

• Navigate to the backend folder:

• bash

CopyEdit

cd backend

Install backend dependencies:

- bash

CopyEdit

npm install

Create a .env file inside the backend folder:

- bash

CopyEdit

PORT=5000

MONGO_URI=mongodb://localhost:27017/airlinesDB

JWT_SECRET=your_jwt_secret_key

Replace your_jwt_secret_key with a strong secret.

For cloud MongoDB, replace the MONGO_URI with your MongoDB Atlas

connection string.

- Start the backend server:
- CopyEdit

npm run dev

Uses Nodemon to auto-restart the server on changes.

✅ Step 4: Set Up the Frontend:

- Open a new terminal and go to the frontend folder:

bash

CopyEdit

cd ../frontend

Install frontend dependencies:

bash

npm install

(Optional) Create a .env file for API base URL:

- bash
- CopyEdit

REACT_APP_API_URL=http://localhost:5000/api npm start

Runs the app on http://localhost:3000

✅ Step 5: Test the Application:

• Open http://localhost:3000 in your browser.

• Register a new user or login.

• Try booking a flight, viewing dashboards, etc.

• Use Postman to test backend endpoints like GET /api/flights.

5. Folder Structure:

5.1 Client:

✅ Main Folders and Files:

CSS

CopyEdit

```
frontend/
 ├── public/
 ├── src/
→ Static files like index.html
 → Main source code
|   ├── components/    → Reusable UI parts (Navbar, Footer)
|   ├── pages/
→ Main screens (Login, Flights, Book)
|   ├── App.js
|   ├── index.js
|  └── .env
→ Main app layout
 → Starting point of the app
→ API URL setup
```

✅ Important Pages:

• Login Page – For user/admin login

• Register Page – New user registration

• Flight List – Shows all available flights

• Book Ticket – Book a seat on a flight

• User Dashboard – See your bookings

• Admin Dashboard – Add or manage flights

✅ How It Works:

• You open the app in a browser (http://localhost:3000)

• React shows different pages using React Router

• Pages talk to the backend using Axios to:

o Get flights

o Book or cancel tickets

o Login/register users

5.2 Server:

The backend of the Airlines Management System is built using Node.js

with Express.js. It handles things like user login, flight data, bookings, and database

operations with MongoDB.

Folder Structure:

```
backend/
 ├── controllers/
 ├── models/
 ├── routes/
→ Logic for handling requests (e.g., booking, login)
→ MongoDB schema for users, flights, bookings
→ API endpoints (e.g., /login, /flights)
 ├── middleware/      → Functions like auth check (JWT)
 ├── config/
→ Database connection setup
 ├── .env
 ├── server.js
→ Environment variables (DB URL, secret keys)
```

6. Running the Application:

Step 1: Start the MongoDB Server .

Step 2: Start the Backend (Server)

Step 3: Start the Frontend (Client)

Step 4: Access the Application

• Open your browser and go to:

http://localhost:3000

• You can:

o Register a new user

o Log in as user or admin

o View flights

o Book or cancel tickets

o Use Admin Dashboard to add/manage flights

6.1 Frontend:

Steps to Start the Frontend

1. Open a terminal window.

2. Navigate to the frontend or client folder:

3. Start the React app:

4. The app will automatically open in your default browser at:

6.2 Backend:

1. Open a terminal window.

2. Navigate to the backend or server directory:

3. Install backend dependencies (only once):

4. Start the server:

7. API Documentation:

7.1Document endpoints exposed by the backend

1. User (Login/Register):

• POST /auth/register → Register new user

• POST /auth/login → Login and get token

• GET /auth/me → Get current user details

2. Flights:

• GET /flights → Show all flights

• GET /flights/:id → Show one flight

• POST /flights → Add flight (Admin)

• PUT /flights/:id → Update flight (Admin)

• DELETE /flights/:id → Delete flight (Admin)

3. Bookings:

• POST /bookings → Book a ticket

• GET /bookings/user → Show your bookings

• DELETE /bookings/:id → Cancel a booking

7.2 Include request methods, parameters, and example responses.

This section lists all the backend API endpoints of the Airlines Management System, including request methods, required parameters, and sample responses.

8. Authentication:

8.1 Explain how authentication and authorization are handled in the project.

✅ What is Authentication?

Authentication means verifying who the user is (e.g., login with email & password).

✅ What is Authorization?

Authorization means checking what the user is allowed to do (e.g., only admin can add flights).

8.2 Include details about tokens, sessions, or any other methods used.

1. User Registration & Login:

• Users register with:

o name, email, password

• Password is hashed using bcrypt before saving in the database.

• On login:

o User provides email and password.

o If valid, the server generates a JWT token (JSON Web Token).

o Token is sent to the client for future requests.

2. Token-Based Authentication:

• After login, the token is stored in the frontend (e.g., localStorage).

• For secure APIs (like booking or admin actions), this token is sent in the header:

3. Role-Based Authorization:

Special middleware functions are used to protect routes:

• authMiddleware.js – Checks if the token is valid

• adminMiddleware.js – Checks if the logged-in user is an admin

8.2 Include details about tokens, sessions, or any other methods used.

This project uses token-based authentication with JWT (JSON Web Tokens) to securely manage user login and role-based access.

✅ 1. Authentication Flow (Login/Register):

• When a user registers, their password is hashed using bcryptjs and stored in the database.

• When a user logs in, the server:

o Verifies the email and password.

o If correct, it generates a JWT token.

o The token is returned in the response and stored on the frontend (usually in localStorage).

✅ 2. Token (JWT) Details:

• The token contains:

o User ID

o User role (user, admin)

o Expiry time

• Token is signed with a secret key stored in .env:

env

CopyEdit

```
JWT_SECRET=your_secret_key
```

✅ 3. Sending the Token (Frontend → Backend):

• The token is sent in every request that requires authentication using the Authorization header:

makefile

CopyEdit

Authorization: Bearer <token>

• The backend checks the token using middleware before giving access.

4. Middleware (Auth Check)

• Middleware checks the token and extracts user info:

CopyEdit

9. User Interface:

To include screenshots or GIFs of your Airlines Management System UI, follow this simple plan for your documentation or project report.

Since I can't take screenshots of your actual project, here's how you can structure and collect them — and I'll also give sample captions and layout you can copy-paste into your report.

10. Testing:

10.1 Describe the testing strategy and tools used.

Testing Strategy

Testing was done to ensure that the application works correctly, securely, and without errors. The project follows a basic manual testing approach with plans to extend it using automated testing tools in the future.

1. Manual Testing:

Manual testing was performed to check each major feature:

Feature

User Registration

User Login

Flight List View

Flight Booking

Cancel Booking

Admin Flight

Add/Edit

Unauthorized

Access

Test Performed

New user creation with

valid/invalid data

Login with correct/wrong

credentials

Display of all flights

Booking ticket with seat selection

Canceling booked ticket

Admin adding/editing flights

Restricted actions blocked for

non-admin

Result

Passed

Passed

Passed

Passed

Passed

Passed

Passed

2. API Testing with Postman:

All backend API endpoints were tested using Postman to verify:

• Correct request and response formats

• Validations and error handling

• JWT token verification

• Role-based access control

Each endpoint returned the correct status codes and data (200, 401, 403, 404).

3. Frontend Testing (Basic):

The frontend was manually tested in the browser to confirm:

• Navigation works properly (React Router)

• Forms validate input before submitting

• Errors show up for invalid actions

• UI displays correct data from the backend

4. Future Improvements:

• Add unit tests for backend using Jest or Mocha

• Add component tests for React using React Testing Library

• Integrate automated testing in CI/CD pipeline (optional)

11. Screenshots or Demo:

• Provide screenshots or a link to a demo to showcase the application.

This section provides visual proof of the working application and demonstrates its key features. Screenshots are taken from both the user and admin views.

Home / Flight List Page:

Allows users/admins to securely log in.

User Dashboard:

Displays user's bookings and their current status.

Book Ticket Page:

User can book a seat on a selected flight.

Cancel Ticket Page:

If you've uploaded a video demo to YouTube, Loom, or Google Drive, you can include a link like this:

12. Known Issues:

12.1 Document any known bugs or issues that users or developers should be aware of.

1. No Seat Conflict Check:

• Issue: Multiple users can book the same seat at the same time if they click quickly.

• Impact: May cause double bookings for the same seat.

• Planned Fix: Add a backend check to lock or reserve seats during booking.

2. No Email Verification:

• Issue: Users can register with any email without verification.

• Impact: May lead to fake or invalid user accounts.

• Planned Fix: Add email verification using OTP or confirmation links.

3. Basic UI Validation:

• Issue: Some input fields (like seat number or date) are not fully validated on the frontend.

• Impact: Users may submit empty or wrong values.

• Planned Fix: Add proper form validation and alerts using React or libraries like Formik/Yup

.

4. No Pagination for Flights or Bookings:

• Issue: All flights and bookings are shown at once.

• Impact: Performance may reduce if there are many records.

• Planned Fix: Add pagination to flight and booking lists.

5. Admin Role Hardcoded:

• Issue: There’s no way to attach files during booking.

• Impact: Limits real-world usage.

• Planned Fix: Integrate file upload for documents (optional future feature).

13. Future Enhancements:

13.1 Outline potential future features or improvements that
could be made to the project.

Here are some ideas and features that can be added to improve the project in the future:

1. Real-Time Seat Availability:

• Send booking confirmation emails to users.

• Notify users when their flight is delayed or canceled.

2. PDF Ticket Download:

• Allow users to download a printable PDF version of their flight ticket after booking.

3. Online Payment Integration:

• Add options to search flights by date, time, and destination.

• Implement sorting and pagination.

4. User ID Upload:

• Allow users to upload ID proof (passport, Aadhar, etc.) for verification during booking.

5. Admin & Staff Roles:

• Create different dashboards for staff to check boarding status.

• Let admins assign flights to staff members.