SYSTEM DESIGN DOCUMENT

for

AIRPORT MANAGEMENT SYSTEM

Version 1.0

Prepared By: GROUP 20

Aswin S	**B230215CS**	**aswin_b230215cs@nitc.ac.in**
P Gowri Sankar	**B230477CS**	**gowri_b230477@nitc.ac.in**
Jesvin Jaison	**B230359CS**	**jesvin_b230359cs@nitc.ac.in**
Aakansh K Anil	**B230117CS**	**aakansh_b230117cs@nitc.ac.in**

PROFESSOR : Prof. Prabu Mohandas

COURSE : CS2011E DBMS

DATE : 9/3/2024

CONTENTS

1: INTRODUCTION

This Database Design Document for the application Airport Management System establishes a target database management system identified from the analysis of the requirements of the software system maintaining data consistency and integrity. The Entity-Relational model thus created analysing the use case diagram is converted to a relational schema of the target Database Management System (DBMS)

1.1 Document Purpose

The System Design Document has the following objectives :

1. To outline the software design and specification of the Airport Management System database in addition to the system architecture and components that can be accessed by users or system developers via a DBMS.
2. To provide a fundamental approach for implementing the database and related software units, thus aiding in extracting details necessary for the software development of the application.

1.2 Document Scope

The Airport Management System (AMS) is designed as a web-based platform that streamlines various airport operations, including flight tracking, passenger management, baggage handling, employee scheduling, and security enforcement. The document provides in-depth descriptions of the system's architectural components, data flow mechanisms, technology choices, and scalability considerations. By structuring the system design methodically, this document ensures that AMS achieves its goal of automating airport management efficiently and securely.

1.3 Intended Audience

This document is intended for multiple stakeholders, including developers, project managers, testers, and airport authorities. Developers can use it as a guideline for implementing core functionalities, while project managers can reference it to oversee the development progress and maintain alignment with the project goals. Testers will rely on this document to validate system performance and ensure compliance with security requirements. Additionally, airport staff and security personnel will use it to understand the operational structure of the system and its functionalities.

1.4 References and Acknowledgments

The system design follows the IEEE Standard for Software Design Documentation, ensuring best practices in software architecture and system modeling. Official documentation from MySQL, React.js, and Node.js is referenced for database, frontend, and backend development, respectively. Additionally, Vercel documentation is used as a reference for deployment strategies and MySQL documentation was referred for Database

2: SYSTEM OVERVIEW

2.1 High Level Architecture

The Airport Management System is structured as a three-tier web application. The presentation layer is built using Next.js, a React-based framework that offers a dynamic and interactive user experience. The application layer is developed using Node.js with Express.js, which provides a robust server-side environment for handling business logic and API requests. The data layer is managed using MySQL, a relational database that ensures structured storage and efficient retrieval of airport-related data. These layers work together to facilitate seamless communication between users, system components, and external integrations such as airline databases and security systems.

2.2 System Context Design

The AMS operates in an environment that requires integration with several external services to function effectively. It interfaces with airline databases to fetch real-time flight schedules and updates. Security agencies are connected to enforce No-Fly List verification, ensuring that restricted individuals are flagged before they board a flight. Additionally, weather APIs are used to provide real-time weather updates, allowing for better flight scheduling and passenger notifications. Internally, the system integrates with various airport subsystems, such as baggage tracking systems and employee management tools, to ensure smooth operations across different departments.

3: SYSTEM ARCHITECTURE

3.1 Architectural Design

The system follows a simplified architecture to ensure efficient implementation within the project timeline. It is structured as a web-based application with a three-tier architecture consisting of a frontend, backend, and database. The frontend, built with Next.js, handles the user interface and authentication requests. The backend, developed using Node.js and Express.js, manages business logic, authentication, and database interactions. The database is powered by MySQL, which stores and retrieves all relevant airport management data.

To maintain modularity, the system is divided into key components, including user authentication, flight management, passenger check-in, baggage handling, and security verification. The authentication system ensures that only authorized users can access the system, while the flight management module allows administrators to create, update, and track flights. The passenger check-in system facilitates the boarding process, linking passengers to their flights and baggage. The baggage tracking module ensures baggage is associated with the correct passenger, and the security verification module cross-checks passengers with the No-Fly List.

3.2 Subsystems Descriptions

Each subsystem plays an essential role in streamlining airport operations. The authentication subsystem manages user logins and permissions using session-based authentication. The flight management subsystem allows administrators to add, update, and remove flight details. The passenger management subsystem enables check-in, boarding pass generation, and travel record maintenance. The baggage handling subsystem ensures that each passenger’s baggage is linked to their flight, reducing misplacement issues. Lastly, the security verification subsystem cross-checks passenger details against the No-Fly List and alerts authorities when necessary.

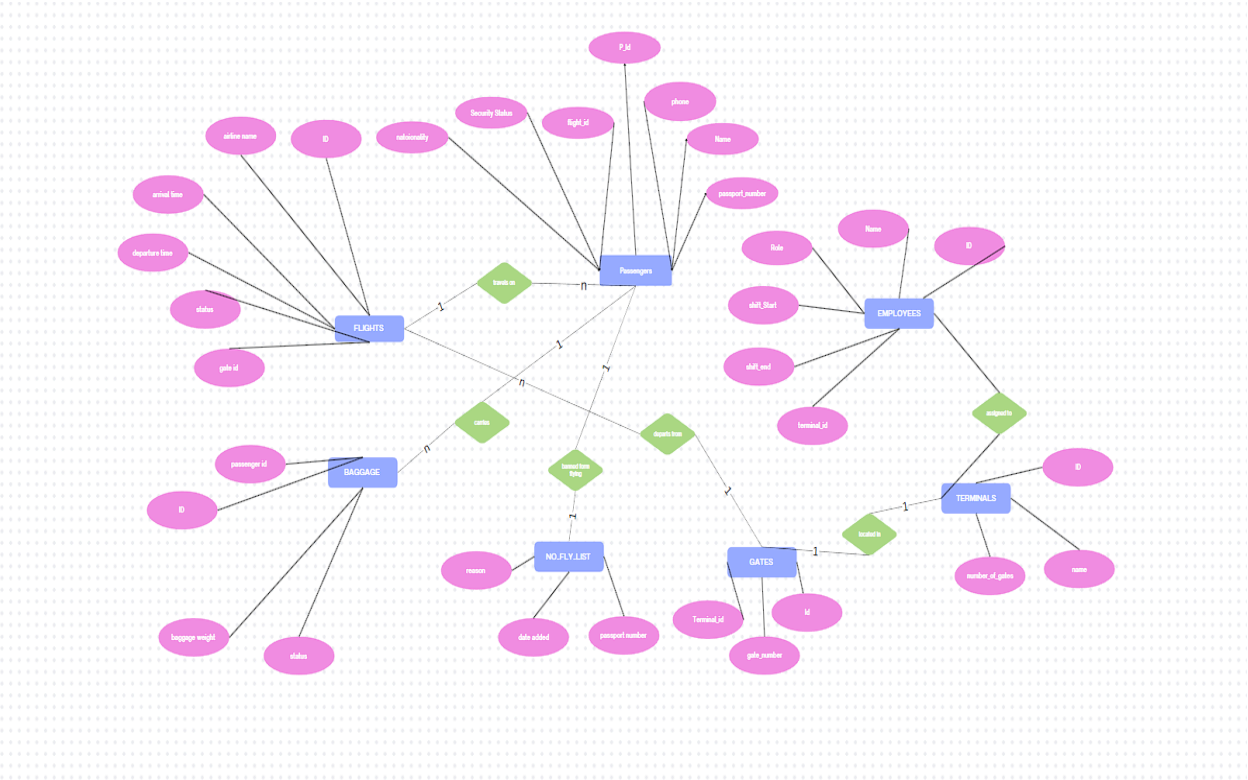
3.3 Technology Stack

The system is developed using a lightweight yet scalable technology stack. The frontend is built with Next.js for easy deployment on Vercel. The backend, using Node.js and Express.js, processes API requests and handles authentication. MySQL serves as the primary database, storing flight schedules, passenger details, baggage information, and security records. The deployment strategy involves hosting the frontend on Vercel, while the backend is deployed on a cloud server or VPS. The database is hosted on a managed cloud platform such as PlanetScale, Supabase, or an Azure MySQL instance.

4: DETAILED DESIGNS

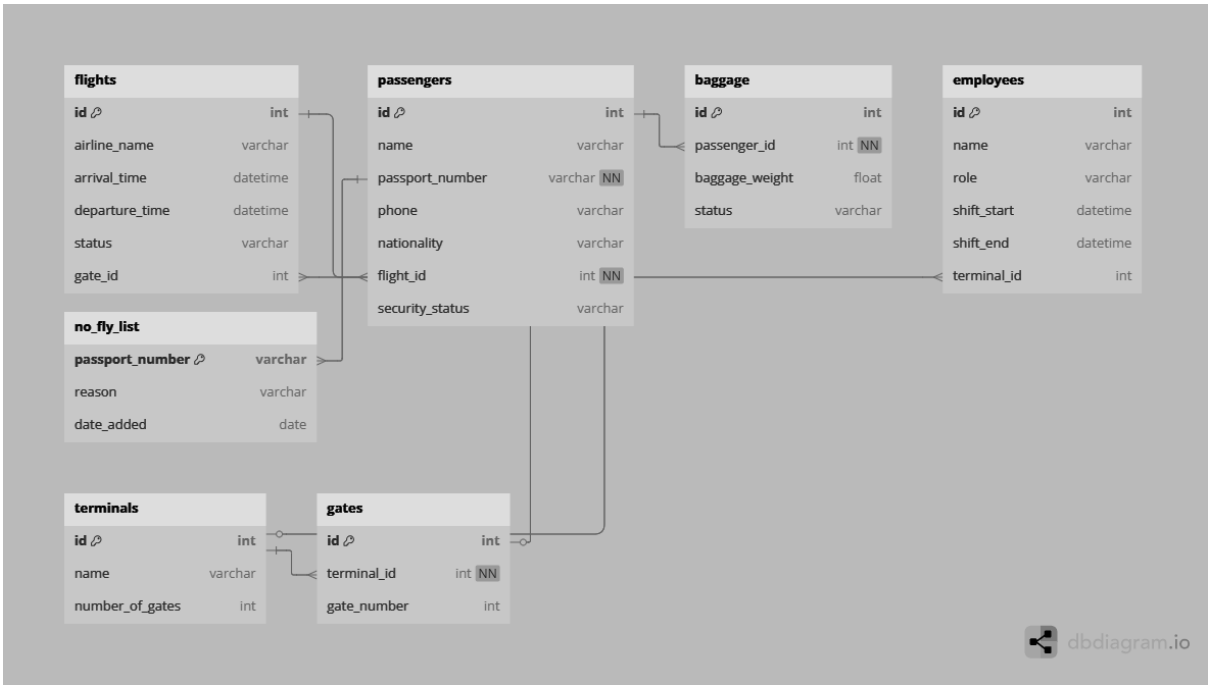
4.1 ER Diagram

The database is designed to efficiently store and manage flight schedules, passenger records, baggage tracking, and security details. The relational schema follows normalization principles to minimize redundancy while ensuring quick data retrieval. The **Entity-Relationship Diagram (ERD)** visually represents relationships between entities such as Flights, Passengers, Employees, Baggage, and Security Checks.



4.2 Relational Schema

The **Relational Schema** defines the structure of database tables, outlining attributes, primary keys, and foreign keys for each table. This schema ensures data integrity and supports optimized queries.



4.3 API Design

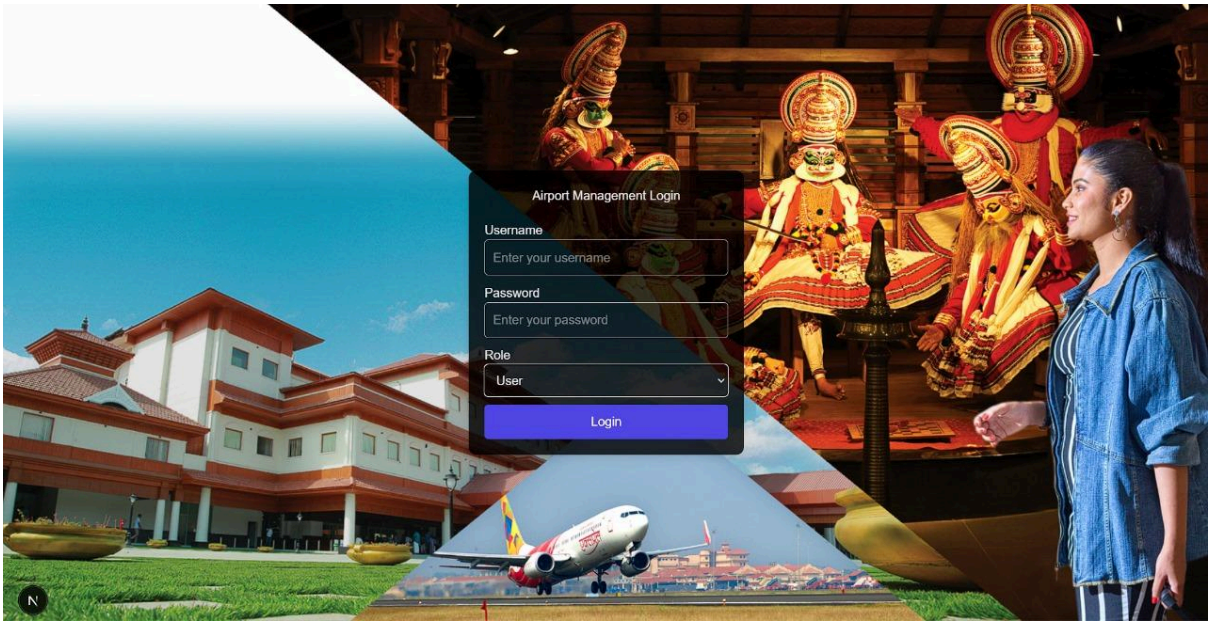
The backend exposes a set of APIs to enable communication between the frontend and database. Each API endpoint follows RESTful principles to ensure modularity and scalability.

- **Authentication API**: Handles user login and session management.
- **Flight Management API**: Allows administrators to create, update, and retrieve flight details.
- **Passenger Check-in API**: Links passengers to flights and generates boarding passes.
- **Baggage Handling API**: Manages baggage tracking and status updates.
- **Security Verification API**: Cross-checks passenger details against the No-Fly List.

Each API ensures secure communication by implementing role-based access control, allowing only authorized users to modify sensitive data.

4.4 User Interface Design

The user interface is designed to be intuitive and responsive, ensuring ease of use for airport staff. The **Admin Dashboard** provides controls for managing flights, employees, and security verification. The **Check-in Portal** allows staff to register passengers and issue boarding passes. The **Security Dashboard** displays No-Fly List alerts and security-related logs. The **Baggage Tracking System** ensures passengers and staff can monitor baggage status in real time.



-Current Opening Web Application Portal

5: SECURITY AND PERFORMANCE

5.1 Authentication

Given the limited team size and the two-week timeframe, the project scope should be scaled down to focus on essential functionality while ensuring feasibility. Instead of implementing a full JWT-based authentication system, a simpler session-based authentication mechanism using Express sessions or NextAuth.js can be used, making integration with Next.js easier. Multi-factor authentication (MFA) can be omitted, as it adds unnecessary complexity for a small-scale project. Role-based access control (RBAC) can also be simplified, limiting user roles to just two: Admin and Staff, rather than implementing multiple role hierarchies.

5.2 Data Protection

For data protection, the focus should primarily be on securely storing passwords using hashing instead of implementing advanced encryption mechanisms like AES-256. While HTTPS is necessary for production, it is not a priority during local development. Security audits and penetration testing, which are typically time-consuming and resource-intensive, can be skipped, as they are not critical for an academic project of this scope.

5.3 Performance Requirements

Performance optimizations should also be streamlined to make the system more manageable. Instead of implementing complex load balancing and database optimizations, the focus should be on ensuring efficient query execution within the database. Rather than implementing real-time updates for flight statuses, a simple polling mechanism or manual refresh can be used to update information at regular intervals. Advanced caching mechanisms like Redis can be avoided unless absolutely necessary.

6: DEPLOYMENT AND SCALABILITY

6.1 Hosting and Cloud Services

In a perfect world, we could have deployed this website on Azure Cloud, due to its high availability, reliability, and scalability. This cloud-based infrastructure supports auto-scaling, enabling the system to adjust resources dynamically based on traffic demands.

However, given the current scope of this project,Vercel provides a more practical hosting solution. While Vercel itself does not offer built-in database support, we can integrate external databases such as PlanetScale, Supabase, Firebase, or MongoDB Atlas to handle data storage efficiently. This approach balances ease of deployment with the flexibility needed to manage dynamic content.

Scalability plays an important role in deployment, making sure that the system can manage growing user traffic and workloads effectively. Ideally, the backend should have some kind of auto-scaling setup so that resources can adjust based on demand, though the exact implementation may depend on the hosting platform. Load balancing can help distribute traffic across multiple backend instances, which should, in theory, prevent the system from becoming overwhelmed. Security is also a major concern, so API endpoints should have authentication measures like JWT and maybe rate-limiting to reduce the chances of misuse or unauthorized access. While these strategies sound good in principle, their effectiveness would depend on how well they are integrated into the system.

By considering these deployment strategies, the Airport Management System can achieve high availability, reliability, and scalability while maintaining security and performance standards.

APPENDIX : GROUP LOG

Date	MEMBERS	TOPICS DISCUSSED	ACTIONS TAKEN
4/03/2025	Jesvin,Aakansh,Aswin, Gowri	Initial planning and the preparation of the SDD document was started.	Defining roles.
5/03/2025	Jesvin,Aakansh,Aswin, Gowri	Creation of ER diagram and the planning of applications and softwares to be used to create the project.	Started working on the ER diagram and schema and compiling the SDD document
9/03/2025	Jesvin,Aakansh,Aswin, Gowri	Changes to the document and real life integration steps.	Completed the SDD document