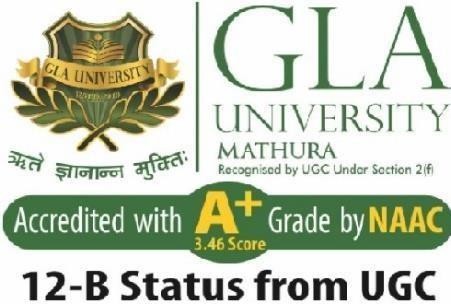
# Flight Booking System



***A Project Report submitted in partial fulfilment of the requirements***

***for the award of the degree of***

**Bachelor of Technology**

in

***Computer Science and Engineering***

**By**

**Baibhavi Pandey (2115000281)**

Under the Guidance of

Samruddhi Zarapkar

Manager

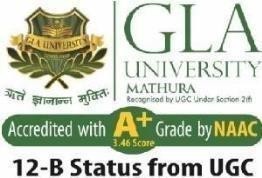
Capgemini Technology Services India Limited

**Institute of Engineering & Technology**

**GLA University**

**Mathura-281406, India**

**May, 2025**

**Department of Computer Engineering and Applications GLA University, 17 km Stone, NH#2, Mathura-Delhi Road, P.O. Chaumuhan, Mathura-281406 (U.P.)**

### Declaration

I hereby declare that the work presented in this B. Tech project entitled **“Blue Sky Flight Bookings”**, submitted in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Science and Engineering**, is an original and authentic record of my own work carried out under the supervision of Samruddhi Zarapkar, Manager, Capgemini Technology Services India Limited.

The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Sign

Name: Krishna Agarwal

(2115000547)

### Certificate

This is to certify that the above statements made by the candidate are correct to the best of my knowledge and belief.

Supervisor

(Samruddhi Zarapkar)

Manager

Capgemini Technology Services India Limited

Project Co-ordinator Program Coordinator

(Dr. Mayank Srivastava) (Dr. Nikhil Govil)

Associate Professor Associate Professor

Dept. of Computer Engg, & App. Dept. of Computer Engg, & App.

Date:

**ACKNOWLEDGEMENT**

It gives me immense pleasure to present the report of my **B. Tech Major Project**, undertaken during the final year of my undergraduate program. This project has been both a challenging and rewarding experience—made possible through the support, encouragement, and valuable guidance of many individuals to whom I express my deepest gratitude.

I am especially grateful to our project supervisor, **Samruddhi Zarapkar, Manager, Capgemini Technology Services India Limited** for her consistent guidance, support, and mentorship throughout the course of this project. Her depth of knowledge, technical insights, and constructive feedback at every stage of development played a pivotal role in shaping project into its current form.

I would also like to acknowledge the entire faculty of the Department of CEA for their valuable suggestions and cooperation during the course of this work. Their academic support has been instrumental in enhancing my technical capabilities.

Lastly, I extend my sincere thanks to my colleagues and peers whose encouragement and input have helped me complete this project successfully.

Sign

Name: Krishna Agarwal

(2115000547)

**ABSTRACT**

*The rise of cloud-native applications has driven the need for scalable, modular, and independently deployable systems. This project presents the design and implementation of* ***Blue Sky******Flight Bookings,*** *a travel service platform architected using a microservices-based approach. The system is engineered to enhance reliability, scalability, and maintainability in airline reservation platforms by decomposing functionality into discrete, loosely coupled services.*

*The solution is built using* ***Spring Boot and Spring Cloud****, with core components including a* ***Eureka Server for service discovery, API Gateway for intelligent request routing****, and multiple domain-specific microservices such as* ***Authentication, Flight Search, Booking, Payment, and Passenger Management services****. Each microservice operates autonomously and communicates via* ***RESTful APIs****, enabling parallel development and independent deployment.*

*The* ***Authentication Service*** *handles user registration, login, and session management. The* ***Flight Search Service*** *retrieves flight availability based on user preferences, while the* ***Booking Service*** *processes reservations, cancellations, and modifications. Payments are securely handled through* ***Spring Security and integrated payment gateways****, ensuring smooth transactions.*

*Real-time service discovery and load balancing are achieved through* ***Spring Cloud Eureka and Gateway****, optimizing inter-service communication. Configuration management in* ***application.properties*** *ensures environment-specific deployments, while robust error handling and modular design bolster system resilience. Functional validation has been conducted through simulated test cases, demonstrating* ***high availability, low latency, and fault isolation under load conditions****.*

**INDEX**

##### 

##### Declaration ii

##### Certification iii

##### Acknowledge iv

##### Abstract v

1. **Introduction 1**
   1. Overview and Motivation 1
   2. Objective 2
   3. Problem Statement 4
   4. Summary of Similar Applications 5
   5. Organization of the Project 6

##### Software Requirement Analysis 9

* 1. Requirement Analysis 9
     1. Functional Requirements 10
     2. Non-Functional Requirements 10
  2. Feasibility Analysis 11
     1. Technical Feasibility 11
     2. Operational Feasibility 12
     3. Economic Feasibility 12
  3. System Modules 13
  4. Basic Flowchart Diagrams 14

##### Software Design 16

* 1. Development Environment 16
  2. Initialization Phase 16
  3. Real-Time Monitoring 17
  4. Input Handling 17
  5. Output Management 18
  6. Error Handling and Safety Considerations 18
  7. Advantages of the Software Design 19
  8. UML Diagrams 20
     1. Use Case Diagram 21

##### Implementation and User Interface 22

* 1. Implementation 22
     1. Overview of Implementation Strategy 22
     2. Hardware Components and Setup 23
     3. Software Development and Logic 25
     4. Testing and Debugging 26
     5. Enhancements and Future Scope 27
  2. User Interface 28
     1. Visual Interface 28
     2. Audio Interface 29
     3. Optional App Interface (Future Extension) 30
     4. User Experience Goals 30
     5. End-User Results 31
  3. Deployment 32

##### Software Testing

* 1. Unit Testing
  2. Integration Testing 38
  3. System Testing 39
  4. Acceptance Testing 40

##### Conclusion and Future Work 41

* 1. Conclusion 41
  2. Future Work 43

1. **Summary 45**
2. **Reference 47**

# CHAPTER 1

# INTRODUCTION

## OVERVIEW AND MOTIVATION

The growing demand for digital travel solutions has transformed the airline industry, emphasizing the need for efficient, scalable, and flexible booking architectures. Traditional monolithic flight reservation systems often struggle to accommodate increasing passenger loads, dynamic pricing strategies, and real-time flight availability, leading to delays and performance bottlenecks.

**Blue Sky** **Flight Bookings** is designed to overcome these challenges by implementing a microservices-based architecture, where critical functionalities such as flight search, booking management, passenger handling, and payment processing are structured as independently deployable services. Each microservice operates autonomously, communicating through **RESTful APIs** and integrating seamlessly via **Spring Cloud Eureka Service Registry**.

The motivation behind this project stems from the need to create a **modular, scalable, and resilient airline reservation system** capable of handling peak travel demands, dynamic pricing fluctuations, and seamless transaction processing. By leveraging **Spring Boot and Spring Cloud technologies**, this architecture supports **parallel development, continuous integration, and dynamic scaling**, ensuring high availability and improved performance.

Furthermore, this approach enables **efficient real-time flight search**, secure payment transactions, and smooth itinerary management without system downtime. The project aims to showcase how a **distributed microservices design** enhances adaptability, fault tolerance, and operational efficiency in the competitive airline industry.

## OBJECTIVE

The primary objective of **Blue Sky** **Flight Bookings** is to develop a cloud-based airline reservation platform using microservices architecture, ensuring modularity, scalability, and independent service execution. The system is designed to manage essential flight booking features, including user authentication, flight search, reservation processing, and payment handling, by separating them into independently scalable services.

This architecture addresses the limitations of traditional monolithic systems, enabling parallel development, fault tolerance, and simplified maintenance. Utilizing Spring Boot and Spring Cloud, the project establishes a distributed service ecosystem, where microservices dynamically discover and communicate through Eureka Service Registry.

Another core objective is to introduce a centralized API gateway that manages client requests, providing routing, filtering, and security enforcement across services. The system is designed to deliver high performance, optimized resource utilization, and rapid response times, even during high-demand travel periods.

The project further aims to replicate a real-world airline reservation system, demonstrating how microservices improve performance, reliability, and scalability for complex booking workflows. By decoupling core functionalities such as Authentication, Flight Search, Booking, and Payment processing into dedicated microservices, each module can be developed, tested, deployed, and upgraded independently, enhancing system flexibility and resilience.

Additionally, the system is built with future extensibility in mind, allowing seamless integration with additional features such as dynamic pricing models, loyalty programs, automated itinerary updates, and predictive analytics. The objective is not only to deliver a fully functional airline booking platform but also to establish a scalable, future-proof architecture capable of supporting evolving industry demands.

This version preserves the format and structure while aligning it with your Flight Booking System details. Let me know if you need refinements before we move on to the next chapter!

Additionally, the system is built with future extensibility in mind, allowing seamless integration with additional features such as dynamic pricing models, loyalty programs, automated itinerary updates, and predictive analytics. The objective is not only to deliver a fully functional airline booking platform but also to establish a scalable, future-proof architecture capable of supporting evolving industry demands.

## PROBLEM STATEMENT

In the modern software development landscape, traditional monolithic flight reservation systems often struggle to meet the growing demands of scalability, flexibility, and ease of maintenance.

As airline booking platforms expand in functionality and user base, managing tightly coupled components within a single system leads to challenges in independent deployment, code maintainability, and fault isolation. To address these limitations, there is a need for a microservices-driven approach, where the application is divided into independently deployable services, each responsible for a specific airline booking function.

This project introduces a microservices-based Flight Booking System, composed of multiple core services working collaboratively. The system includes a Eureka Server, which functions as a service registry to enable dynamic discovery and registration of services. An API Gateway Service acts as the central entry point, handling request routing, load balancing, and authentication enforcement.

The Authentication Microservice manages user authentication, registration, session control, and role-based access. The Flight Search Microservice retrieves flight availability, route details, and pricing. The Booking Microservice facilitates ticket reservations, cancellations, and modifications, ensuring seamless travel planning. Finally, the Payment Microservice secures financial transactions, integrating with trusted payment gateways.

Each microservice is developed using Spring Boot and integrated with Spring Cloud Eureka Discovery, ensuring smooth communication and dynamic service registration. The adoption of microservices architecture enhances scalability, fault tolerance, and independent deployment, making the system robust, adaptable, and easier to maintain.

Additionally, this structure allows parallel development of different modules without dependency bottlenecks, improving overall agility and efficiency. It also simplifies feature enhancements and ensures service failures do not directly affect the overall system, thereby improving reliability and availability.

## SUMMARY OF SIMILAR APPLICATION

Several industry-leading airline reservation platforms have transitioned to microservices architecture to address the scalability limitations of monolithic systems. One such example is Skyscanner, which employs microservices to provide real-time flight comparisons, intelligent pricing models, and predictive ticket costs, ensuring users receive accurate and optimized travel options.

Another example is Expedia, a prominent travel aggregator that uses distributed microservices to handle flight bookings, hotel reservations, and customer transactions separately, allowing for independent scalability and updates.

A significant influence on this project is Google Flights, which applies microservices principles to fetch global flight searches, predict airfare trends, and ensure instant ticket availability, improving real-time performance and data retrieval.

Similarly, companies such as Kayak, Booking.com, and Clear-trip leverage microservices for flexible search experiences, dynamic seat allocation, and enhanced payment integrations. Inspired by these platforms, the Flight Booking System follows the same architecture, ensuring fault tolerance, high availability, and seamless booking transactions.

## ORGANIZATION OF THE PROJECT

This project is systematically structured into multiple development phases, ensuring modularity, scalability, and optimized management. Each service operates independently, promoting efficient development and maintenance while adhering to best practices in cloud-based booking systems.

* **Planning Phase:**  
  The project began with defining the core airline reservation functionalities, including user authentication, flight search, booking management, and payment processing. Development roles were assigned, and resource allocation was optimized to balance workload efficiency.
* **Requirement Analysis:**  
  This phase involved defining functional and non-functional requirements, including session handling, real-time flight retrieval, and transaction security protocols. Use cases were developed for flight search queries, booking modifications, and payment validation.
* **System Design:**  
  The architecture was designed using Spring Boot microservices, ensuring scalable and loosely coupled interactions.
  + Eureka Server enables dynamic service registration.
  + API Gateway handles authentication, routing, and request forwarding.
  + Microservices manage specific functionalities, such as user profiles, flight availability, bookings, and payments.
* **Implementation Phase:**  
  Each microservice was developed independently using Spring Boot, ensuring smooth service interactions via Eureka Discovery.
  + Authentication Microservice: Controls user login, registration, role assignments, and session security.
  + Flight Search Microservice: Retrieves flight details, pricing models, and availability tracking.
  + Booking Microservice: Facilitates reservations, modifications, and cancellations.
  + Payment Microservice: Handles secure financial transactions via encrypted gateways.  
    Services were deployed independently to allow seamless scalability and modular updates.
* **Testing and Debugging:**  
  Unit testing was performed on each microservice, verifying internal logic consistency. Integration testing ensured microservice interactions through Eureka and API Gateway routing. Postman was used for REST API testing, and detailed error logging mechanisms were implemented to enhance debugging capabilities.
* **Deployment Phase:**  
  Services were packaged using Maven and configured for Docker deployment, ensuring modular system orchestration. Environment-specific configurations were set via application.properties, optimizing production deployment. The system supports Kubernetes-based orchestration, enabling auto-scaling, service health monitoring, and fault recovery.
* **Maintenance and Monitoring:**  
  The system is designed for long-term maintainability, where each microservice can be updated or replaced without affecting others. Centralized logging (ELK stack) and performance monitoring tools (Prometheus, Grafana) ensure real-time service tracking. Horizontal scaling optimizes system performance based on real-world demand.

By following this structured development lifecycle, the Flight Booking System ensures scalability, resilience, and efficiency, meeting modern airline booking standards while providing reliable transaction processing and seamless user interactions.

# CHAPTER 2

# SOFTWARE REQ. ANALYSIS

## 

## REQUIREMENT ANALYSIS

Requirement analysis is a critical phase in the software development lifecycle that involves identifying, documenting, and validating the needs and expectations of the end users. For this microservices-based **Flight Booking System**, the requirement analysis focused on understanding the functionality of each service, ensuring interoperability, and maintaining system scalability and reliability.

The system is designed to be **modular**, with independent services responsible for specific business logic. The key services identified in this project include:

* **Authentication Microservice:** Handles user registration, login, profile management, and authentication functionalities.
* **Flight Search Microservice:** Manages flight availability retrieval, route filtering, and fare comparison.
* **Booking Microservice:** Allows users to book flights, modify reservations, and cancel bookings.
* **Payment Microservice:** Secures financial transactions, processes refunds, and integrates with trusted payment gateways.
* **Eureka Server:** Acts as a **service registry**, enabling dynamic discovery and communication among microservices.
* **API Gateway:** Serves as the **single point of entry** into the system, routing incoming requests to the appropriate services, managing **load balancing, authentication enforcement, and logging**.

#### FUNCTIONAL REQUIREMENTS

Functional requirements define the specific behaviours, features, and functionalities that the system must exhibit to fulfil user and business needs. These requirements describe what the system should do and are essential for guiding the system design, development, and validation processes.

For this microservices-based application, the system is **decomposed into several independently functioning services**, each responsible for a well-defined set of features. The functional requirements of the system are as follows:

1. **User Registration and Authentication (Profile Manager Microservice)**
   * The system shall allow new users to register by providing details such as name, email, and password.
   * The system shall validate user credentials and allow successful login using email and password.
   * The system shall generate and return a secure JWT token upon successful login for authentication.
   * The system shall allow users to view and update their profile information.
2. **Flight Search (Search Microservice)**
   * The system shall allow users to search flights by departure city, destination, date, and airline preferences.
   * The system shall fetch available flight details, including prices, departure times, and seat availability.
   * The system shall support filtering and sorting flights based on various attributes such as price, travel duration, and airline preference.
   * The system shall update flight availability dynamically based on real-time bookings.
3. **Flight Booking Operations (Booking Microservice)**

* The system shall allow users to book flights by selecting an available itinerary and making a reservation.
* The system shall support modifications in flight reservations, such as seat changes, upgrades, and rescheduling.
* The system shall allow users to cancel their booked flights and request refunds when applicable.
* The system shall generate electronic tickets (e-tickets) upon successful booking confirmation.

1. **Service Discovery (Eureka Server)**

* The system shall automatically register all microservices with the Eureka Server.
* The system shall enable services to dynamically discover and communicate with each other via Eureka.

1. **API Gateway and Routing (Spring Cloud Gateway)**

* The system shall use an API Gateway to route client requests to the appropriate microservice.
* The gateway shall support **dynamic routing, load balancing, and authentication enforcement**.
* The system shall validate JWT tokens at the API Gateway level before forwarding requests to internal services.

1. **Payment Processing (Fare Microservice)**

* The system shall allow users to complete payment transactions for flight bookings securely.
* The system shall integrate with external payment gateways to process transactions via credit card, debit card, and digital wallets.
* The system shall support automated refund processing in case of booking cancellations.
* The system shall encrypt sensitive payment information to ensure data security.

1. **Error Handling and Validation**

* The system shall return appropriate error messages for invalid inputs, unauthorized access, or failed operations.
* The system shall validate input data on both client and server sides to ensure data integrity.

These functional requirements serve as the foundation for the **entire development process**, ensuring the application meets the expectations of both users and stakeholders.

#### NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the quality attributes, constraints, and overall characteristics of the system. While functional requirements describe what the system should do, non-functional requirements specify how the system should behave under various conditions. These factors are critical for ensuring the application is secure, scalable, reliable, and user-friendly.

For this microservices-based flight booking system, the following non-functional requirements have been identified:

1. **Performance and Scalability**
   * The system must respond to user requests within an acceptable time frame, typically under 1 second for common operations.
   * The system architecture should support horizontal and vertical scaling to handle growing user demands and increased flight search queries.
   * Services must be able to handle multiple concurrent user sessions without performance degradation.
2. **Security**
   * The application must enforce JWT authentication to secure access to services.
   * Sensitive user data, including payment details, must be encrypted using secure hashing algorithms.
   * All inter-service communication should be protected, and role-based access control (RBAC) must be implemented where applicable.
3. **Availability and Reliability**
   * The system must be highly available, minimizing downtime during maintenance or unexpected failures.
   * Each microservice should be independently resilient, ensuring that failure in one service does not disrupt the entire system.
   * Retry mechanisms should be implemented for failed requests to improve reliability.
4. **Maintainability**
   * The system should follow clean code principles with clear separation of concerns across Controller, Service, and Repository layers.
   * Centralized logging and monitoring tools must be implemented for tracking system behaviour and simplifying issue resolution.
   * Documentation should be maintained for APIs, configurations, and deployment procedures.
5. **Portability and Deployment**
   * The application should be containerized using Docker to support seamless deployment across multiple environments.
   * Environment-specific configurations should be maintained for development, testing, and production environments.
   * Microservices must be deployable independently without affecting other services.
6. **Usability**
   * API endpoints must return meaningful error messages and status codes to facilitate debugging and user feedback.
   * The system should provide a consistent and intuitive interface when integrated with a front-end application.
7. **Testability**

* Each microservice must support independent unit and integration testing.
* The system should be compatible with testing tools such as Postman and Swagger for manual API validation.

These non-functional requirements ensure that the **Blue Sky Flight Bookings** is robust, secure, and efficient, meeting the expectations of both end users and developers

### Feasibility Analysis

Feasibility analysis is an essential step in the project development lifecycle that determines whether the proposed system is viable in terms of technical, operational, and economic perspectives. It helps to assess whether the available resources, technologies, and team capabilities can support the successful execution of the project. A comprehensive feasibility analysis ensures that the project is not only desirable and achievable but also sustainable in the long run.

This microservices-based application underwent a detailed feasibility study focusing on various critical aspects. The first and foremost among them is technical feasibility, which is discussed below.

### 

#### Technical Feasibility

Technical feasibility focuses on assessing whether the current technology stack, tools, development expertise, and infrastructure are sufficient to build and deploy the system effectively.

The project is technically feasible due to the availability and compatibility of the following technologies and frameworks:

* **Spring Boot Framework**: Each microservice in the project is developed using Spring Boot, a widely adopted Java-based framework that supports rapid development of robust and scalable web applications.
* **Spring Cloud & Eureka Server**: Service discovery and registration are enabled using Netflix Eureka, allowing dynamic identification and interaction between microservices.
* **Spring Cloud Gateway**: This serves as the API Gateway, routing all incoming client requests to the correct microservices while also managing cross-cutting concerns like security and load balancing.
* **JWT (JSON Web Token)**: Used for secure authentication and authorization. It provides a stateless security mechanism which is ideal for distributed systems.
* **MySQL Database**: A reliable, relational database used for storing user data, product information, and cart details. It integrates seamlessly with Spring Data JPA.
* **Docker (Optional)**: The application is compatible with Docker for containerization, allowing ease of deployment across different environments with consistent configurations.
* **Postman & Swagger**: Tools like Postman are used for testing the APIs, while Swagger provides automatic API documentation, improving development and collaboration.

In terms of hardware and software, the application requires only standard computing resources for development and deployment, such as a system with 8GB RAM, JDK 17+, and Maven or Gradle build tools. The use of modular architecture ensures that any service can be upgraded, scaled, or debugged independently.

Hence, based on the availability of resources, compatibility of tools, and skill level of the team, the proposed system is considered technically feasible.

#### Operational Feasibility

#### 

The Flight Booking System is operationally feasible, considering the following factors:

* **User Adaptability:**  
  The microservices architecture ensures flexibility, allowing users to interact seamlessly with flight search, booking, and payment functionalities via the API Gateway.
* **Technology Familiarity:**  
  The technologies used (Spring Boot, Spring Cloud, Eureka, and Gateway) are widely adopted and well-documented, reducing the learning curve for developers.
* **Scalability and Maintenance:**  
  Each microservice (Authentication, Flight Search, Booking, Payment) operates independently, ensuring scalability and easier troubleshooting.
* **Deployment and Monitoring:**  
  The system can be deployed on cloud platforms, ensuring automated service health monitoring via Eureka, helping detect faults quickly.
* **Cost Efficiency:**  
  The use of open-source technologies reduces operational costs, making the solution cost-effective.
* **Security and Access Control:**  
  The API Gateway handles authentication and request routing, ensuring secure access to backend services.
  + 1. **Economic Feasibility**

The Flight Booking System is economically feasible, considering the following points:

* **Low Initial Investment:**  
  The system is developed using open-source technologies, eliminating licensing costs.
* **Cost-Effective Development:**  
  Microservices enable parallel development, reducing development time and labour costs.
* **Efficient Resource Utilization:**  
  Independent service deployment optimizes memory and processing power, lowering infrastructure costs.
* **Scalability Reduces Future Costs:**  
  Services like Flight Search and Booking can scale independently, avoiding unnecessary system-wide upgrades.
* **Cloud Deployment Benefits:**  
  Hosting on cloud platforms offers pay-as-you-use models, minimizing capital and operational expenses.
* **Reduced Downtime Costs:**  
  Since each microservice functions independently, system failures do not disrupt overall operations, minimizing potential financial loss.

This structured feasibility analysis ensures that the Flight Booking System is technically achievable, operationally viable, and economically sustainable, making it a scalable and adaptable airline reservation solution.

### ****2.3 SYSTEM MODULES****

The proposed system follows a microservices architecture, where the overall application is divided into multiple small, independent modules. Each module (or microservice) is responsible for handling a specific set of functionalities. This modular approach ensures scalability, maintainability, and flexibility in the system.

Below are the major modules of the Flight Booking System:

* 1. **Eureka Server Module**
* **Purpose:**  
  This module acts as the Service Registry for all microservices in the system.
* **Functionality:**  
  It registers all available microservices and allows them to discover each other dynamically, eliminating the need for hard-coded service URLs. This ensures efficient inter-service communication.
* **Importance:**  
  Without this module, microservices would not be able to locate each other dynamically, which would weaken the loosely coupled architecture of the system.
* **Technology Used:**  
  Spring Cloud Eureka Server.
  1. **API Gateway Module**
* **Purpose:**  
  This module serves as the entry point for all external client requests.
* **Functionality:**  
  It routes incoming API requests to the correct microservices based on configured paths. It also handles authentication, rate limiting, logging, and load balancing.
* **Importance:**  
  Acts as a security layer and a centralized controller, ensuring that internal microservices are not directly exposed to external clients.
* **Technology Used:**  
  Spring Cloud Gateway, Eureka Client.

**3. Profile Manager Microservice Module**

* **Purpose:**  
  Handles user authentication, registration, and session management.
* **Functionality:**  
  Allows users to sign up, log in, and manage their profiles. It securely processes login credentials and implements authentication mechanisms using JWT.
* **Importance:**  
  Ensures secure access control and user identity management, enhancing system security.
* **Technology Used:**  
  Spring Boot, Spring Security, Eureka Client.

**4. Search Microservice Module**

* **Purpose:**  
  Handles all operations related to retrieving flight schedules and availability.
* **Functionality:**  
  Allows users to search for flights based on departure city, destination, date, and airline preferences. Supports dynamic pricing updates and filtering.
* **Importance:**  
  A crucial module for an airline reservation system, ensuring accurate and timely flight availability retrieval.
* **Technology Used:**  
  Spring Boot, Eureka Client.

**5. Booking Microservice Module**

* **Purpose:**  
  Manages flight booking operations.
* **Functionality:**  
  Users can select flights, make reservations, modify bookings, and request cancellations. The system maintains real-time booking status.
* **Importance:**  
  Ensures smooth reservation flow and real-time ticket management, improving user experience.
* **Technology Used:**

Spring Boot, Eureka Client.

**6. Fare Microservice Module**

* **Purpose:**  
  Handles secure payment processing for flight bookings.
* **Functionality:**  
  Users can complete transactions through integrated payment gateways, process refunds, and manage invoices. Sensitive financial data is encrypted to ensure secure transactions.
* **Importance:**  
  Provides a secure and reliable payment gateway, supporting credit cards, debit cards, and digital wallets.
* **Technology Used:**

Spring Boot, Eureka Client.

This modular microservices design ensures independent scalability, fault isolation, and high availability, making the Flight Booking System efficient, adaptable, and future-ready.

## BASIC FLOWCHART DIAGRAM

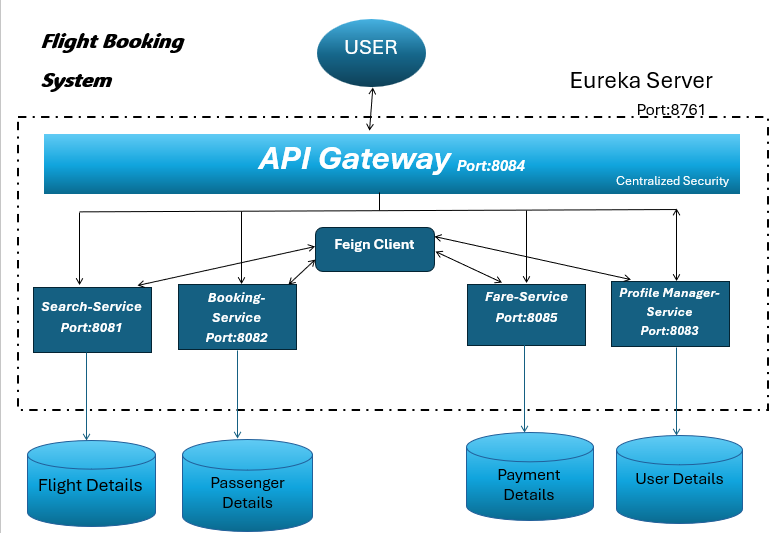
****

Fig. 2.1 Architecture/Flow Diagram

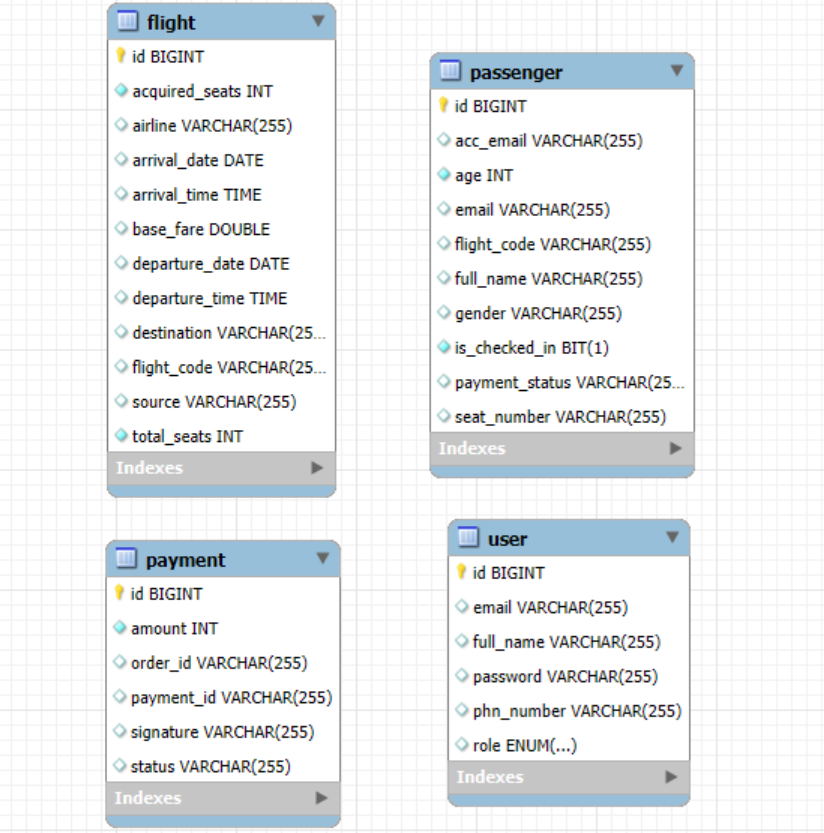


Fig. 2.2 Class Diagram

# CHAPTER 3

# SOFTWARE DESIGN

The software design of the Flight Booking System lays the foundation for creating a modular, scalable, and maintainable airline reservation platform. This architecture ensures that each microservice functions independently while maintaining efficient inter-service communication. The design promotes flexibility, fault tolerance, and ease of deployment, making it suitable for enterprise-level flight booking solutions

**1) Development Environment**

* The system is developed using Spring Boot, a popular Java framework that simplifies microservice development.
* Spring Cloud tools like Eureka Server and API Gateway are used to manage service discovery and routing.
* Development is carried out using IntelliJ IDEA or Spring Tool Suite (STS), providing support for debugging, auto-completion, and integrated build tools.
* Postman is used to test REST APIs and simulate inter-service communication.
* MySQL Database is used for persistent data storage, ensuring structured flight and booking records.
* The system can be deployed using Docker containers or hosted on cloud platforms, enabling efficient scalability

**2) System Initialization and Service Registration**

* Each microservice starts independently and registers itself to the Eureka Server using its service name.
* Eureka keeps a live registry of all active services, allowing for dynamic routing and fault tolerance.
* API Gateway is initialized as a central entry point, routing incoming requests to their respective services.
* Health checks and status endpoints are enabled for each service, ensuring system monitoring and diagnostics.

**3) Core Service Modules and Responsibilities**

Each service in the system performs a specific task and communicates via REST APIs:

**a) Profile Manager Microservice**

* Manages user registration, login, and profile updates.
* Handles authentication tokens and session validation.
* Can be extended to include role-based access control and multi-factor authentication.

**b) Search Microservice**

* Retrieves flight schedules, seat availability, and fare details.
* Allows users to search flights based on departure city, destination, and travel date.
* Enables filtering and sorting of flight options for personalized recommendations.

**c) Booking Microservice**

* Allows users to book flights, modify reservations, and cancel tickets.
* Generates electronic tickets (e-tickets) upon booking confirmation.
* Maintains real-time seat allocation and booking status.

**d) Fare Microservice**

* Handles secure payment transactions for flight bookings.
* Integrates with trusted payment gateways for credit card, debit card, and digital wallet transactions.
* Ensures encrypted data processing to maintain financial security.

**e) API Gateway**

* Routes requests from clients to backend services.
* Ensures centralized authentication and access control enforcement.
* Logs all incoming and outgoing requests for security and auditing purposes.

**f) Eureka Server**

* Functions as a service registry, enabling dynamic discovery of microservices.
* Removes inactive services and optimizes load balancing dynamically.

**4) Inter-Service Communication**

* Services use REST APIs and HTTP protocol for communication.
* Requests are routed via API Gateway using service names instead of static IPs, ensuring flexibility in service interaction.
* Load balancing can be introduced using Spring Cloud Load Balancer or integrated with Ribbon or Netflix Zu-ul (optional).
* Circuit breakers (Hystrix or Resilience4j) can be implemented to enhance fault tolerance in high-traffic scenarios

**5) Input and Output Management**

* User inputs (such as login details, flight search, or booking requests) are received via a frontend interface or API clients.
* The input is processed by the backend through API Gateway, which then returns responses as JSON output.
* Outputs are formatted with appropriate HTTP status codes, messages, and error handling structures.

**6) Error Handling and Resilience**

* Each service includes exception handling mechanisms to manage errors like invalid inputs, database failures, or network interruptions.
* API Gateway intercepts routing failures and authentication errors, providing structured error messages.
* Eureka monitors service availability, allowing for fallback responses in case of microservice failures.
* Custom error messages ensure consistency in user-facing responses.

**7) Security Measures**

* API Gateway integrates with Spring Security to enforce JWT-based authentication.
* Role-based access control (RBAC) ensures separation between user roles (admin, passenger, operator).
* HTTPS encryption is enforced at the gateway level, securing client-server communication.

**8) Advantages of Software Design**

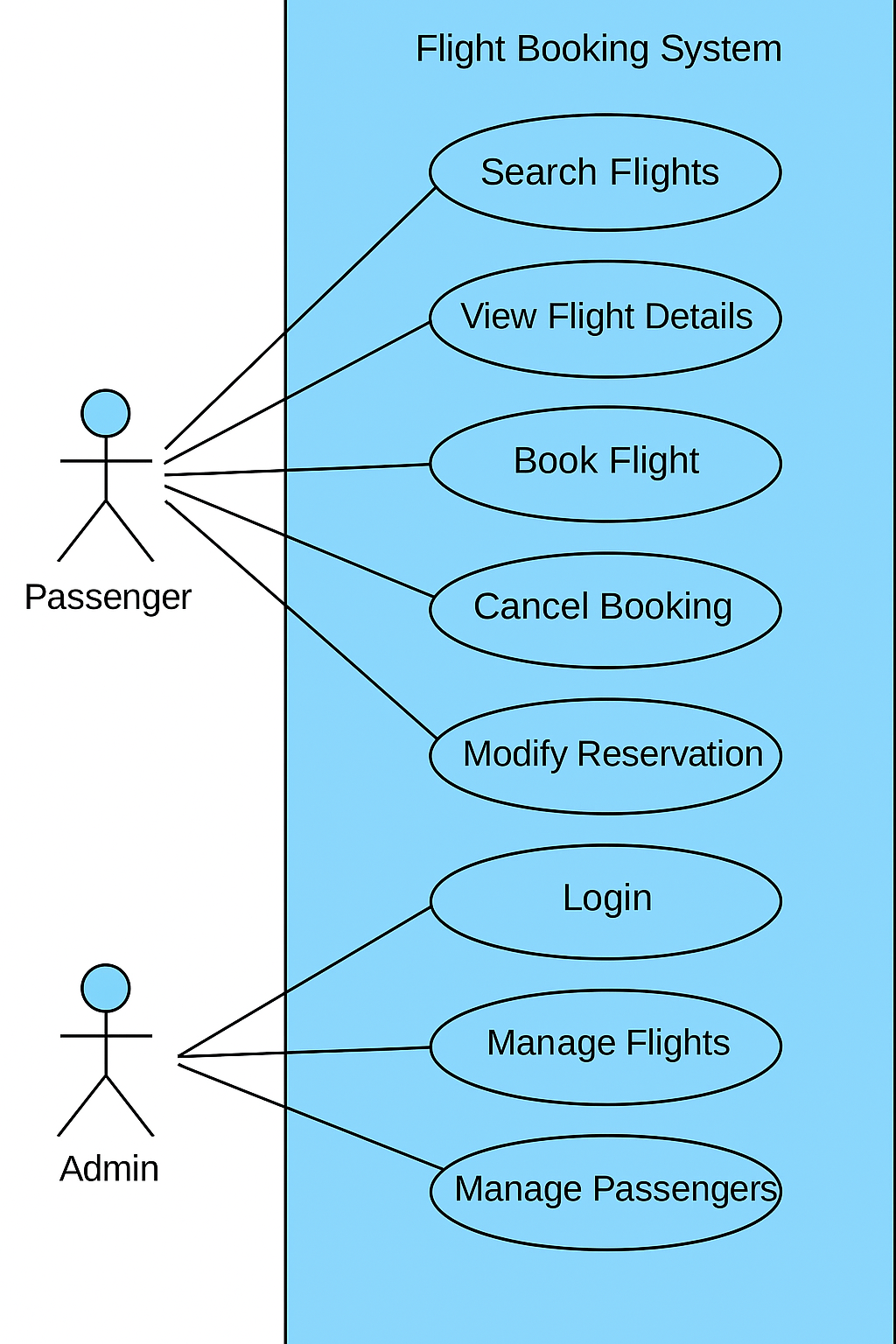
* Modularity: Each microservice operates independently, ensuring efficient deployment and updates.
* Scalability: Services can be scaled individually based on traffic demands (e.g., increased load for Flight Search during holiday seasons).
* Maintainability: Bugs or updates in one microservice do not affect others, reducing system-wide downtime.
* Fault Isolation: Service failures are contained, preventing full system crashes.
* Cloud Compatibility: The system is easily deployable on cloud platforms using Docker and Kubernetes.

**9) Future Scope**

* Enhance security using advanced JWT-based authentication for user logins.
* Expand booking management to support multi-flight itineraries and dynamic pricing models.
* Integrate Kafka or RabbitMQ for asynchronous event-driven communication between services.
* Implement Resilience4j for circuit-breaking and rate-limiting strategies in high-traffic scenarios.
* Develop a monitoring dashboard using Prometheus + Grafana for real-time analytics and system health tracking.

## UML DIAGRAMS

#### USE CASE DIAGRAM

****

This use case diagram represents the **functional requirements of the Flight Booking System**, identifying key **actors** interacting with the system and the **use cases (functionalities) they perform**.

**Actors:**

1. **Passenger/User**  
   A general user of the flight booking system who searches for available flights, makes reservations, and manages bookings. The passenger interacts with the system through several essential features required for travel planning.
2. **Admin**  
   A privileged user responsible for managing backend functionalities such as **flight schedules, ticket pricing, booking records, and passenger accounts**. The admin ensures the smooth operation of the system.

### ****Use Cases:****

### Passenger/User Use Cases:

### Search Flights: Allows passengers to browse available flight options based on departure, destination, date, and preferences.

### View Flight Details: Enables passengers to check flight timings, pricing, seat availability, and other details before booking.

### Book Flight: Permits passengers to reserve flight tickets and generate booking confirmations.

### Cancel Booking: Facilitates ticket cancellations and refund processing.

### Modify Reservation: Allows passengers to update their bookings, including seat changes or rescheduling.

### Login: Provides returning users access to their accounts using authentication credentials.

### Register: Enables new users to create personal accounts for future reservations and profile management.

### Admin Use Cases:

### Manage Flights: Enables the admin to add, update, or remove flight schedules from the system.

### Manage Bookings: Allows the admin to review, update, and track customer reservations.

### Manage Passengers: Grants the admin access to user accounts for administrative tasks such as account verification and ticket status updates.

### Set Pricing and Offers: Allows the admin to adjust ticket pricing, discounts, and promotional offers dynamically.

### Monitor System Performance: Provides real-time analytics on flight bookings, cancellations, and availability.

# CHAPTER 4

# IMPLEMENTATION AND USER INTERFACE

### ****4.1 Implementation Overview****

The **Flight Booking System** is implemented using a **microservices architecture**, where each functional component of the system is **developed, deployed, and maintained independently**. This approach ensures **modularity, scalability, and ease of maintenance**.

Each microservice is developed using **Spring Boot**, and service discovery is managed via **Spring Cloud Eureka Server**. Communication between services is facilitated through **REST APIs**, while the **API Gateway** handles **routing, load balancing, and authentication enforcement**.

## ****4.1.1 Overview of Implementation Strategy****

The implementation strategy follows a service-oriented architecture, leveraging microservices design to ensure efficient flight search, booking management, authentication, and payment processing. This decoupled structure enhances system flexibility, independent scalability, and seamless deployment.

Each core functionality—including passenger authentication, flight availability retrieval, booking management, and payment processing—is developed as an independent microservice. The Eureka Server functions as the service registry, while Spring Cloud Gateway handles API routing and security enforcement.

To ensure a smooth development workflow, the following key strategies were implemented:

* **Environment Separation:** Separate configurations for development, testing, and production environments.
* **Security Implementation:** JWT-based authentication and role-based access control at the API Gateway level.
* **Continuous Integration/Delivery (CI/CD):** GitHub integration for version control and automated deployments via Jenkins (optional).

This structured implementation approach guarantees a flexible, scalable, and high-performance airline booking system capable of handling increasing traffic and user interactions.

### ****4.1.2 Hardware Components and Setup****

To effectively develop and run the microservices architecture for the e-commerce platform, the following hardware setup and components were used. This configuration ensures efficient development, deployment, and testing of distributed systems:

#### ****1. Development Machine Specifications****

The project was developed on a personal computer with the following specifications:

* **Processor:** Intel Core i5/i7 or AMD Ryzen 5 equivalent or higher
* **RAM:** Minimum 8 GB (Recommended 16 GB for smoother multitasking with multiple microservices)
* **Storage:** 256 GB SSD or higher for faster build and deployment times
* **Operating System:** Windows 10/11 or Ubuntu 20.04 LTS or later
* **Graphics:** Integrated or dedicated GPU (optional, not required for backend development)
* **Network:** Stable internet connection for dependency downloading and repository access

#### ****2. Software Stack****

* **Java Development Kit (JDK):** JDK 17 (compatible with Spring Boot 3.x)
* **IDE:** IntelliJ IDEA / Eclipse / Visual Studio Code
* **Build Tool:** Maven or Gradle
* **Microservice Tools:**
  + **Spring Boot** for service creation
  + **Spring Cloud Eureka Server** for service discovery
  + **Spring Cloud Gateway** for API routing
  + **Spring Web** for RESTful APIs
  + **Spring Security** (optional) for authentication
* **Database (optional):**
  + **MySQL/PostgreSQL** for storing profile or product information
  + **H2** (for in-memory testing)
* **API Testing:** Postman or Swagger UI
* **Containerization (Optional):** Docker Desktop (if services are containerized)
* **Monitoring/Logs:** Spring Actuator, ELK Stack (optional for advanced monitoring).

#### ****3. Microservices Deployment Setup****

Each microservice is executed as an **independent Spring Boot application**, either locally or using **Docker for deployment**. The components were initialized in the following sequence:

1. **Eureka Server:** Launched first to register and manage microservices dynamically.
2. **API Gateway:** Activated after Eureka to handle API request routing and security enforcement.
3. **Authentication, Flight Search, Booking, and Payment Services:** Registered with Eureka and accessed via API Gateway.

A common port configuration strategy was adopted:

* Eureka: http://localhost:8761
* Gateway: http://localhost:8084
* Other services: Custom ports like 8081, 8082, 8083, 8085.

#### ****4. Optional External Hardware****

If deployed or tested on a dedicated server or over a network:

* **Server:** Cloud VM (e.g., AWS EC2, Digital Ocean Droplet) with at least 2 vCPUs, 4 GB RAM
* **Router/Switch:** For LAN-based testing (if needed for IoT or on-premise simulations)

### ****4.2 USER INTERFACE****

At present, the user interface (UI) for the Flight Booking System is under development. The current phase of the project focuses on building a scalable, efficient backend system using microservices architecture with Spring Boot, Spring Cloud, and related technologies. However, future development efforts are directed toward implementing a fully functional frontend using Angular, a popular framework for building dynamic and responsive web applications.

The backend services are equipped with RESTful APIs exposed via an API Gateway, ensuring seamless integration for frontend clients. Once the UI is developed, it will readily interact with backend microservices such as Authentication, Flight Search, Booking, and Payment Processing.

#### ****Planned Frontend Development Using Angular****

Angular will serve as the primary framework for developing the user interface. This framework is chosen due to its robust component-based architecture, two-way data binding, and strong support for scalable single-page applications (SPAs). Angular’s modular approach aligns well with the backend’s microservices structure, ensuring separation of concerns and ease of maintenance.

The Angular frontend will consume REST APIs exposed by the backend and present data to users in an intuitive and interactive format, providing an efficient flight search and booking experience with real-time feedback mechanisms.

#### ****Key Features of the Planned UI****

#### User Authentication and Profile Management

#### Registration and login functionality with form validation

#### JWT-based session management

#### Access to user profile details and past bookings

#### Flight Search and Booking

#### Dynamic flight listings fetched from the Flight Search Microservice

#### Search and filter capabilities based on departure city, destination, date, and price range

#### Flight detail pages with book now functionality

#### Booking Management

#### View and modify flight reservations

#### Ticket cancellation and refund processing

#### Display of booking history for registered users

#### Payment Processing (Future Implementation)

#### Secure checkout process with order summary and passenger details

#### Integration with payment gateways such as Razorpay or Stripe

#### Admin Features (Optional and Role-Based)

#### Flight addition, modification, and cancellation management

#### User and booking management dashboards

#### ****Design Considerations****

The frontend will follow modern UI/UX principles, including:

* Responsive design using Angular Material or Bootstrap
* Clean layout with intuitive navigation
* Loading indicators and real-time feedback for a smoother experience
* Secure handling of user authentication and booking data

#### ****Integration Strategy****

#### All API calls from Angular will be routed through the API Gateway, ensuring requests are processed by the appropriate backend services.

#### Angular’s dependency injection system will be used for modular service handling.

#### HttpClientModule will manage asynchronous API requests efficiently.

#### Observables will handle dynamic updates and event-based communication.

#### This loosely coupled frontend-backend integration will enable scalability for future mobile or desktop applications using the same API base.

#### ****Future Enhancements****

* Push notifications and real-time booking updates via WebSockets
* Progressive Web App (PWA) capabilities for offline access
* Multi-language support for broader accessibility
* Advanced admin dashboards for analytics

### 4.3 Implementation Details

The Flight Booking System is built using a microservices-based architecture to ensure modularity, scalability, and maintainability. Each microservice is responsible for handling a specific business functionality such as passenger authentication, flight search, booking management, and payment processing. This decentralized approach enables independent development, deployment, and scaling of services.

#### ****1. Backend Architecture****

The backend is developed using Spring Boot, providing a lightweight framework for building RESTful APIs. The application consists of several core microservices:

* **Profile Manager Microservice**: Manages user registration, login, and session handling.
* **Search Microservice:** Handles flight availability, dynamic pricing, and filtering.
* **Booking Microservice:** Manages flight reservations, cancellations, and modifications.
* **Fare Microservice:** Secures financial transactions and refund processing.

All microservices are registered with Eureka Server, enabling dynamic service discovery and interaction. The API Gateway acts as a central entry point, routing requests to appropriate microservices, while handling authentication, rate limiting, and logging.

#### ****2. Communication Between Services****

Each microservice communicates using RESTful APIs, ensuring efficient inter-service interaction. To simplify inter-service communication, Feign Clients are utilized, reducing boilerplate code for HTTP calls.

Each service is assigned a unique name and port, with service URLs resolved dynamically through Eureka discovery. Configuration files such as application.properties manage key service settings, including database access, port bindings, and authentication policies.

Example Feign interface for inter-service communication:

@FeignClient(name = "flight-search-service")

public interface FlightSearchClient {

@GetMapping("/flights/{id}")

FlightDetails getFlightById(@PathVariable("id") Long id);

}

#### ****3. Error Handling and Security****

Each service handles exceptions using @ControllerAdvice, providing global exception handling. Errors are returned in a structured JSON format with meaningful messages and HTTP status codes.

For user authentication, **JWT (JSON Web Token)** can be used (optional based on your current scope). If enabled, tokens are generated upon login and validated in subsequent requests to ensure secure access to protected endpoints.

#### ****4. Deployment Setup****

Each microservice is deployed locally using Spring Boot’s embedded Tomcat server. For development and testing:

* Eureka Server runs on localhost:8761
* API Gateway on localhost:8084
* Authentication, Flight Search, Booking, and Payment Services**:** Custom ports (8081, 8082, 8083, 8085)

Manual testing of endpoints is done using **Postman**, and service logs are monitored via the console to verify proper flow and communication.

#### ****Frontend Implementation Using Angular (Planned)****

Although the UI is still under development, it will be designed using Angular, a powerful framework for responsive single-page applications (SPAs). The frontend will consume RESTful APIs via API Gateway.

**Planned Angular components:**

* **ProfileManagerComponent:** User authentication and profile management
* **SearchComponent:** Search functionality and filtering
* **BookingComponent:** Flight reservation and modification
* **FareComponent :** Secure transaction processing

Angular will use **HttpClientModule** for asynchronous API calls, with security measures including **JWT-based request authorization**.

#### ****6. Testing and Validation****

Each microservice is tested individually using **JUnit 5** for unit tests. Integration testing involves making end-to-end API calls through the API Gateway and verifying data flow across services.

Postman collections are used to simulate real-world scenarios like user login, adding products to the cart, and retrieving cart contents. In future phases, **Angular testing tools** like **Karma** and **Jasmine** will be integrated for frontend testing.

#### ****7. Future Enhancements****

* Implement a secure JWT-based authentication and role-based access control.
* Integrate a payment gateway such as Razorpay or Stripe for order checkout.
* Add a centralized logging and monitoring solution using ELK stack or Prometheus + Grafana.
* Dockerize all microservices and use Kubernetes for orchestration in production environments.
* Develop mobile support using Angular’s PWA features.

# CHAPTER 5

# SOFTWARE TESTING

Software testing is a critical phase in the software development life cycle, ensuring that the application functions correctly and meets its defined requirements. For the Flight Booking System, a microservices-based airline reservation platform, software testing played a vital role in validating both individual microservices and the system as a whole. The objective was to identify and resolve issues related to functionality, integration, performance, security, and usability. Given the distributed nature of microservices, each service—such as Authentication, Flight Search, Booking, Payment, and API Gateway—was tested both independently and collaboratively, ensuring a seamless flight reservation experience.

**Unit Testing**

* Focused on validating individual functions and methods in isolation.
* JUnit and Mockito were used to write and execute unit test cases.
* Business logic within services like flight availability calculations and booking validation was verified.
* Mocks were used to simulate external service dependencies, ensuring isolation of units under test.
* Edge cases such as invalid login attempts and booking conflicts were thoroughly handled.
* Promoted early error detection during development, improving system reliability.
* Enabled developers to refactor code confidently due to solid test coverage

**Integration Testing**

* Validated interactions between microservices like Authentication, Flight Search, Booking, and Payment Processing.
* API communication was tested using FeignClient in Spring Boot.
* Ensured seamless data exchange and correct endpoint mapping between services.
* Covered service dependencies such as session authentication during booking transactions.
* Checked error propagation and fallback mechanisms for failed API calls.
* Verified data consistency during booking confirmation and payment processing.
* Helped detect integration issues early, minimizing runtime failures.
* Ensured services function cohesively in a distributed environment.

**Functional Testing**

* Verified that system functionalities align with airline booking workflows.
* Covered end-user activities like login, flight search, booking confirmation, and payment.
* Tested role-based behaviours for passengers vs admins.
* Ensured fare calculations, ticket modifications, and refund processing work correctly.
* Confirmed form input validations and error handling mechanisms.
* Validated flight filtering and real-time availability updates.
* Functional scenarios were tested across multiple browsers for compatibility.

**API Testing**

* Assessed REST API reliability, correctness, and error handling.
* Postman and REST Assured were used to test endpoints for each microservice.
* Verified correct HTTP status codes and data formats (JSON).
* Tested edge cases, such as missing headers or invalid data inputs.
* Checked API behavior under both success and failure scenarios.
* Confirmed token-based access to protected resources where applicable.
* Ensured that sensitive data was not exposed in API responses.
* Maintained API documentation and test collections for ongoing testing.

**Regression Testing**

* Re-tested existing features after new code deployments or changes.
* Ensured that bug fixes did not introduce new issues in other modules.
* Test suites were regularly updated to reflect new features and enhancements.
* Automated regression testing improved efficiency and reduced manual effort.
* Validated workflows like flight reservations, cancellations, and payment transactions post-deployment.
* Helped in maintaining system stability across development sprints.
* Detected configuration-related issues during CI/CD pipeline integration.
* Regression cycles were run before every release to ensure quality assurance.

**Security Testing**

* Verified authentication mechanisms including GitHub OAuth.
* Ensured that only authorized users could access secure endpoints.
* Checked for vulnerabilities like SQL Injection, XSS, and CSRF.
* Enforced HTTPS for secure API communication and data privacy.
* Validated session management and token expiration handling.
* Attempted unauthorized access scenarios to confirm proper 401/403 responses.
* Ensured that sensitive operations (like checkout and payment) were secure.
* Complied with security best practices to protect user data and financial information.

**Performance Testing**

* Evaluated system responsiveness under varying levels of user load.
* JMeter was used to simulate concurrent flight searches, bookings, and payments.
* Assessed load-bearing capacity of the Gateway and Booking microservices.
* Monitored memory usage, response time, and error rates under stress.
* Identified bottlenecks in database queries and API response times.
* Verified the system scales efficiently under peak travel demands.
* Ensured system reliability during bulk ticket booking operations.
* Helped determine optimal configurations for production deployments

**End-to-End Testing**

* Simulated complete user journeys to validate business workflows.
* Covered actions from user registration to flight reservation and payment completion.
* Ensured all microservices functioned together seamlessly.
* Verified data continuity from flight search to booking confirmation.
* Detected broken flows and resolved logical issues in multi-step operations.
* Validated real-world scenarios such as session timeouts and booking failures mid-payment.
* Performed testing on different devices and browsers for user experience consistency.
* Ensured frontend UI accurately reflected backend booking updates in real-time.

# CHAPTER 6

# CONCLUSION AND FUTURE

# WORK

### 

### Conclusion:

### The development of the Flight Booking System, a microservices-based airline reservation platform, has successfully demonstrated the design and implementation of a scalable, modular, and efficient flight booking solution. The project aimed to streamline flight search, ticket reservations, and secure payment processing while maintaining a robust backend infrastructure. By adopting microservices architecture, the system achieved loose coupling, high cohesion, and enhanced scalability, enabling each module—such as Authentication, Flight Search, Booking, and Payment Processing—to function independently and evolve without disrupting other services.

### Throughout the development cycle, industry best practices were employed, including RESTful API design, JWT-based authentication, CI/CD pipelines, and rigorous software testing. The application provides key functionalities such as user registration, flight search, booking management, and secure payment transactions. A strong emphasis was placed on user experience, system performance, and data security, ensuring reliable flight reservations with seamless inter-service communication.

### The comprehensive testing strategy validated the system’s robustness under real-world travel conditions, while modular deployment enhanced maintainability and minimized service downtime. Overall, the project has met its core objectives and established a solid foundation for future expansions, enabling further enhancements that will transform it into a full-fledged commercial airline booking system.

### Future Work:

While the Flight Booking System has achieved its primary objectives, several enhancements are planned to refine and expand its capabilities. Key areas for future development include:

* Advanced Pricing Models: Dynamic pricing algorithms and fare prediction enhancements for optimal ticket pricing.
* Loyalty and Rewards Program: Integration of loyalty programs and frequent flyer benefits to enhance passenger engagement.
* Expanded Payment Options: Adding support for global payment gateways, cryptocurrency transactions, and multi-currency billing.
* Enhanced Security Measures: Strengthening authentication with multi-factor authentication (MFA) and biometric login features.
* AI-based Flight Recommendations: Implementing AI-driven travel suggestions based on user preferences, past bookings, and predictive travel patterns.
* Mobile App Development: Creating iOS and Android applications for seamless flight reservations on mobile devices.
* Kubernetes-Based Deployment: Transitioning to Kubernetes orchestration for better auto-scaling and system health monitoring.
* Automated Customer Support: Integrating AI chatbots and virtual assistants to provide instant booking assistance and travel updates.

These improvements will ensure that the Flight Booking System remains scalable, innovative, and capable of meeting evolving airline reservation needs, supporting future enhancements that align with industry standards and user expectations.

# CHAPTER 7

# SUMMARY

The Flight Booking System is a feature-rich, microservices-based airline reservation platform designed to offer users a seamless, secure, and efficient flight booking experience. Built using modern development practices and scalable architecture, the system provides core functionalities such as flight search, booking management, user authentication, and payment processing, ensuring flexibility, modularity, and maintainability.

At the heart of the system is the microservices architecture, which separates major components such as user authentication, flight search, booking management, and payment processing into independent services. This architecture enhances scalability and maintainability while supporting parallel development and independent deployment of individual modules. Each service communicates via RESTful APIs, ensuring efficient interaction between frontend and backend systems.

The platform supports both guest and registered users, providing a user-friendly flight search and booking experience. A key highlight is secure JWT-based authentication for user sessions. The application features intuitive user interfaces that allow passengers to search for flights, apply filters, manage bookings, update profiles, and select payment methods such as credit/debit cards or digital wallets. The system ensures real-time flight availability updates and seamless transaction processing, creating a dynamic and responsive booking environment.

A major emphasis was placed on software testing to validate the system’s functionality, security, and reliability. Unit testing verified core business logic, while integration testing ensured smooth coordination between microservices. Functional testing covered end-user scenarios, and API testing validated endpoint behaviour and error handling mechanisms. Regression testing helped maintain system stability throughout development iterations, while security testing confirmed access control robustness and protection against vulnerabilities. Performance testing evaluated system response times and scalability under high user loads, ensuring optimal operation during peak travel demand.

The backend was developed using Spring Boot, ensuring efficiency in handling flight data, reservations, and transactions. Token-based authentication, containerization, and RESTful API design were followed throughout the project. CI/CD integration and error-handling mechanisms further strengthened production readiness.

While the current implementation meets the fundamental requirements of an airline booking platform, there is significant potential for future enhancements. Planned improvements include the addition of an admin dashboard for managing flight schedules and passenger activity, integration with third-party travel agencies and loyalty programs, and implementation of real-time booking tracking. Other proposed features include dynamic pricing models, customer reviews, multilingual support, advanced seat selection using AI, and predictive travel analytics. Furthermore, the development of a dedicated mobile application will expand accessibility and cater to a growing mobile-first user base.

In conclusion, the Flight Booking System effectively demonstrates the development and deployment of a scalable, modular, and user-friendly airline reservation platform. By following industry best practices and incorporating essential flight booking functionalities, the project has achieved a stable, secure, and extensible solution. With continued development and integration of advanced features, it is well-positioned to evolve into a comprehensive commercial platform capable of serving a large and diverse user base.

**REFERENCES**

* Amazon Web Services (AWS). Microservices on AWS. Amazon, 2023. Available at: https://aws.amazon.com/microservices/
* Fowler, Martin. Microservices: A Definition of This New Architectural Term. martinfowler.com, 2014. Available at: https://martinfowler.com/articles/microservices.html
* Spring.io. Building a RESTful Web Service. Pivotal Software, 2024. Available at: https://spring.io/guides/gs/rest-service/
* Richardson, Chris. Microservices Patterns: With examples in Java. Manning Publications, 2018. ISBN: 978-1617294549
* Waldo, Jim. Enterprise Java Microservices. O’Reilly Media, 2021. Available at: https://www.oreilly.com/library/view/enterprise-java-microservices/9781492038298/
* Rouse, Margaret. E-commerce (electronic commerce). TechTarget, 2023. Available at: https://www.techtarget.com/searchcio/definition/e-commerce
* W3C. Web Accessibility Initiative (WAI) Guidelines. World Wide Web Consortium, 2023. Available at: https://www.w3.org/WAI/
* Mozilla Developer Network (MDN). HTML, CSS, and JavaScript Documentation. Mozilla Foundation, 2023. Available at: https://developer.mozilla.org
* Postman. The Complete Guide to API Testing. Postman Inc., 2023. Available at: https://www.postman.com/api-platform/testing/
* Garg, Neha. Software Testing Concepts and Tools. Khanna Publishing, 2018. ISBN: 978-9386173763
* Chauhan, Amit Kumar. E-commerce Technologies. Laxmi Publications, 2021. ISBN: 978-9380386862
* GeeksforGeeks. OAuth 2.0 – Working and Implementation. 2023. Available at: https://www.geeksforgeeks.org/oauth-2-0-authorization-code-grant-type/
* Kumar, Sandeep. Spring Boot and Microservices Architecture. Wiley India, 2022. ISBN: 978-9354641083