

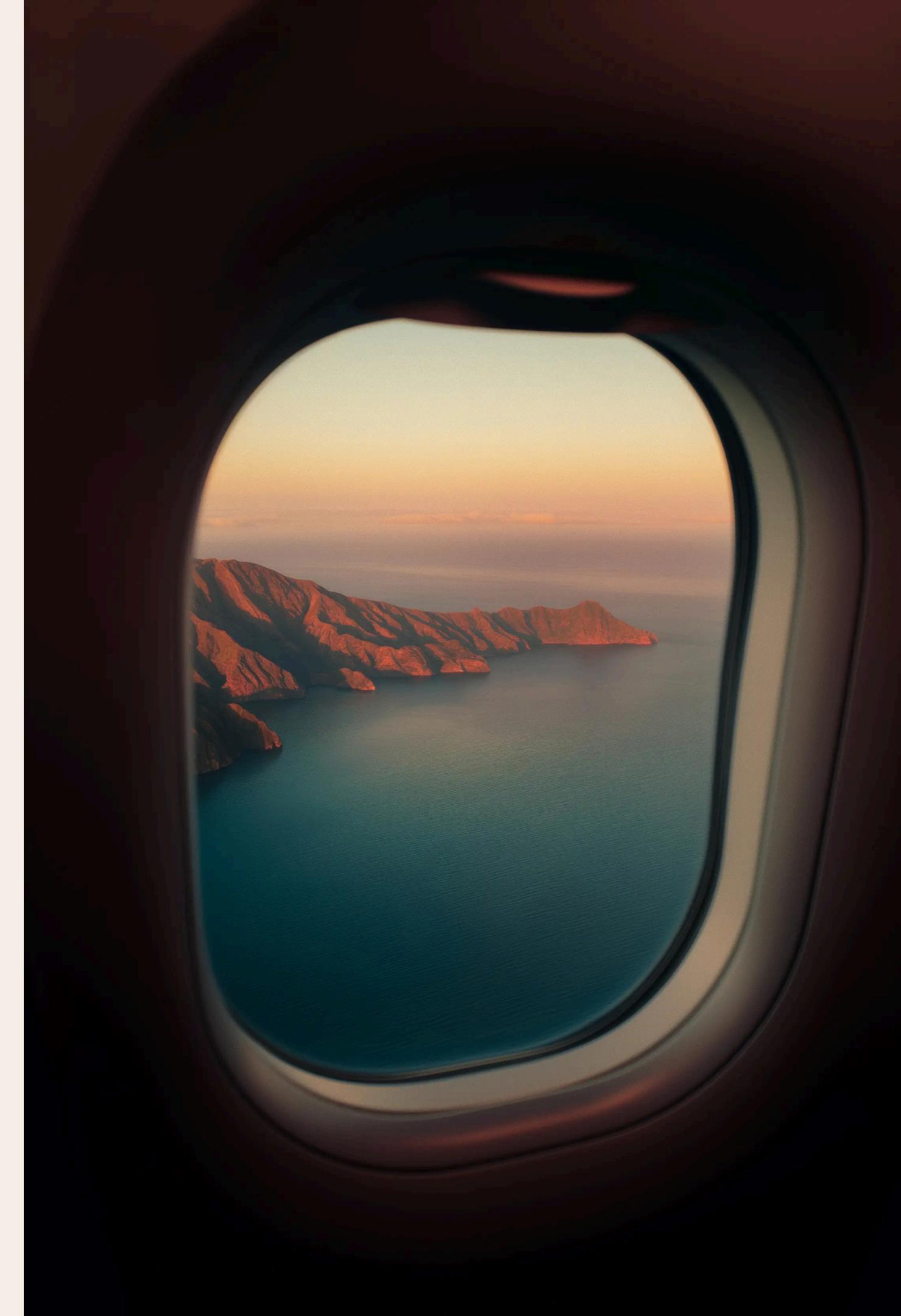
# AIR TRANSPORTATION SYSTEM

PRESENTED BY **TEAM ALPHA**

TEAM LEADER: **ASHWIN NAGAPURE**

TEAM MEMBERS:

- **SARANSH BHOLE**
- **AAYUSH MESHRAM**
- **AMARBIRSINGH RANDHAWA**
- **JANAVI JADHAV**



# INTRODUCTION

## Overview:

- Designed to simplify and enhance the passenger and airline management experience.
- Offers a complete solution to handle flights, bookings, baggage, crew assignments, and passenger interactions, while ensuring system scalability and code quality.

## Technologies Used:

- Backend Framework: **Spring Boot (Java)**
- Frontend Framework: **ReactJS**
- Collaboration Tools: **Swagger, Jira, SonarQube**
- Version Control: **GitHub**
- Database: **MySQL**
- IDE: **IntelliJ IDEA**

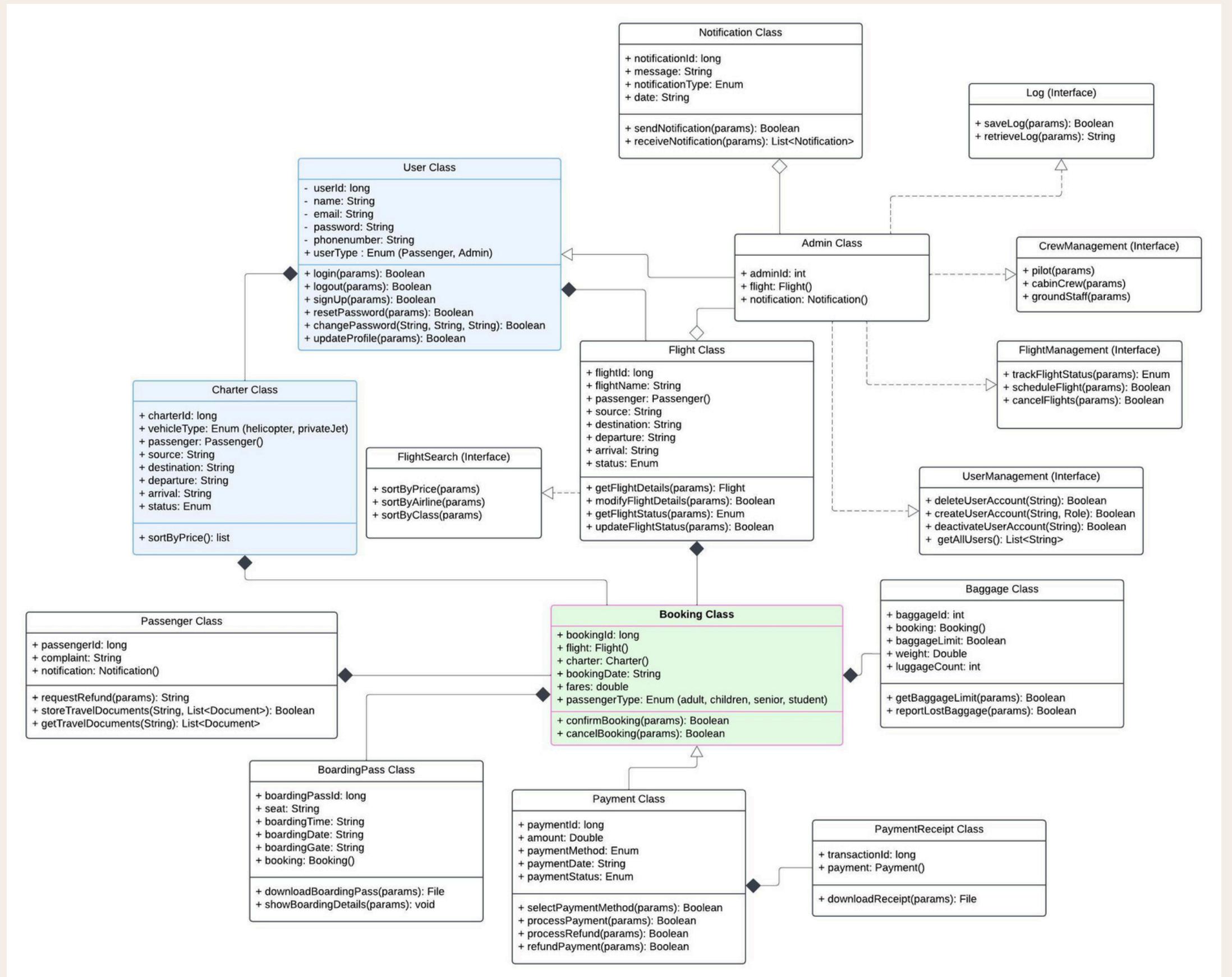


# INTRODUCTION (CONTD.)

## Project Goals:

- Enhance passenger convenience with personalized travel options.
- Optimize airline operations, including crew and flight management.
- Provide a robust and scalable architecture for future growth.

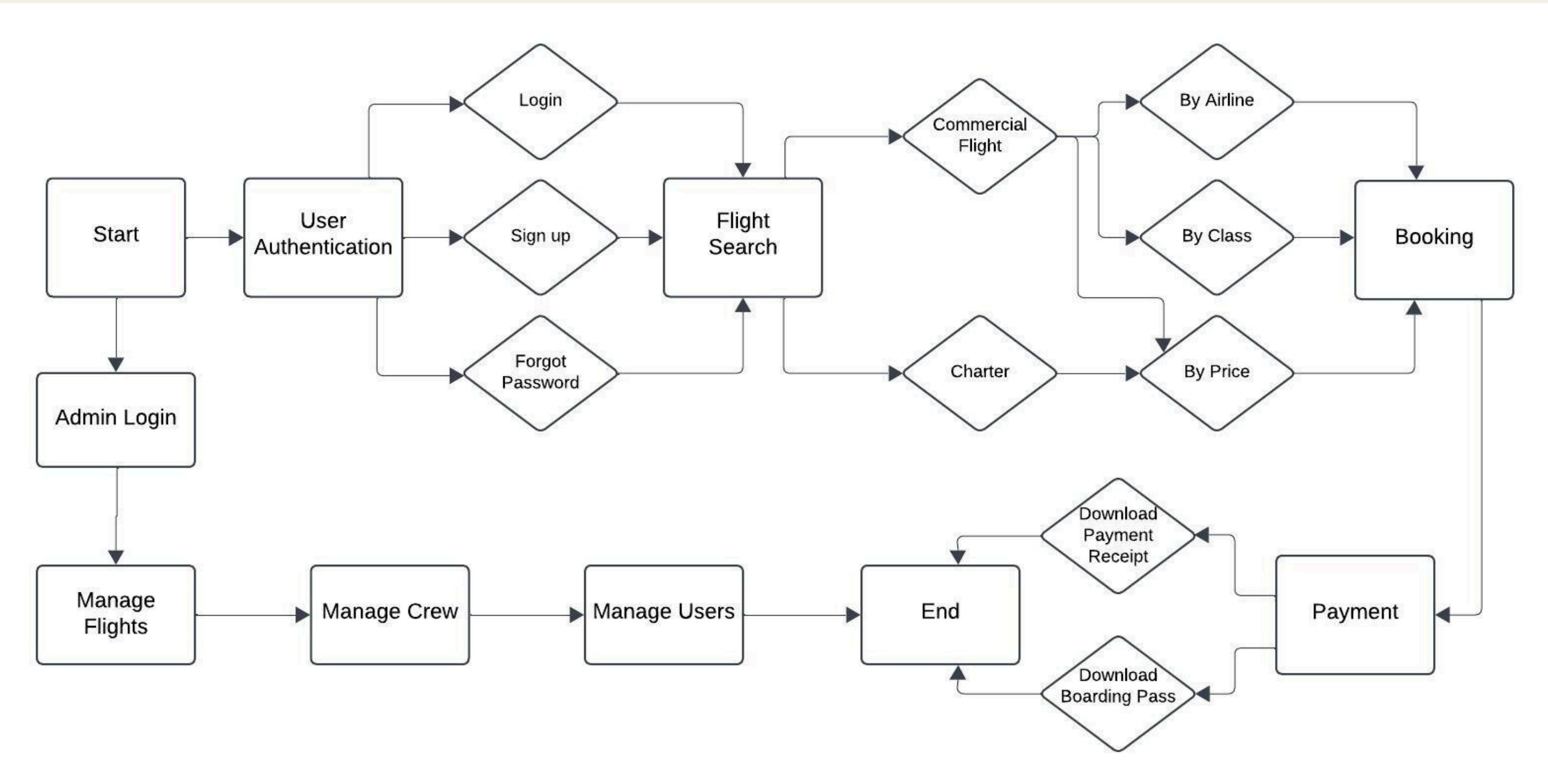
# SYSTEM DESIGN & WORKFLOW



UML Class Diagram



# SYSTEM DESIGN & WORKFLOW (CONTD.)



Workflow  
Representation  
Diagram



# DEVELOPMENT WORKFLOW WITH AGILE

The project was developed following Agile principles to ensure iterative and incremental progress.

## Scrum Framework:

- Sprint duration: 3–4 days per sprint.
- Daily stand-ups to track progress and address issues.

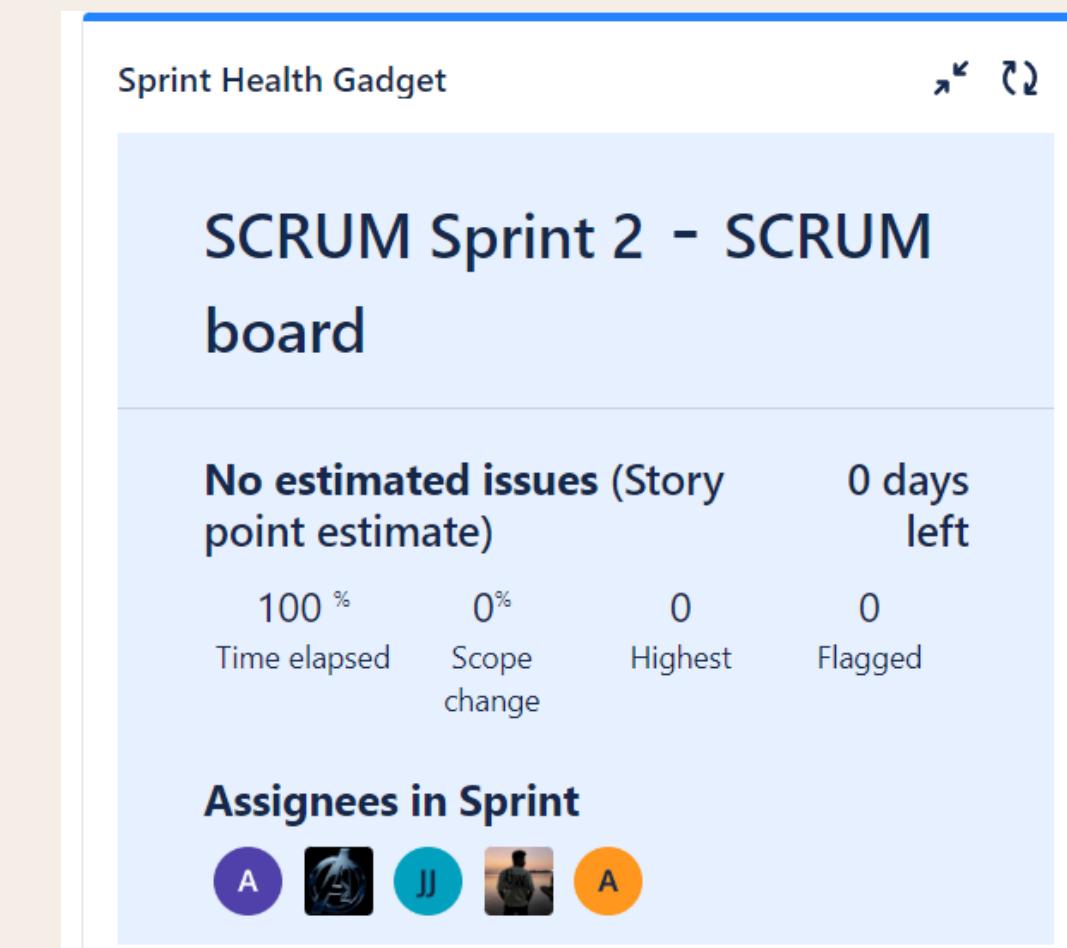
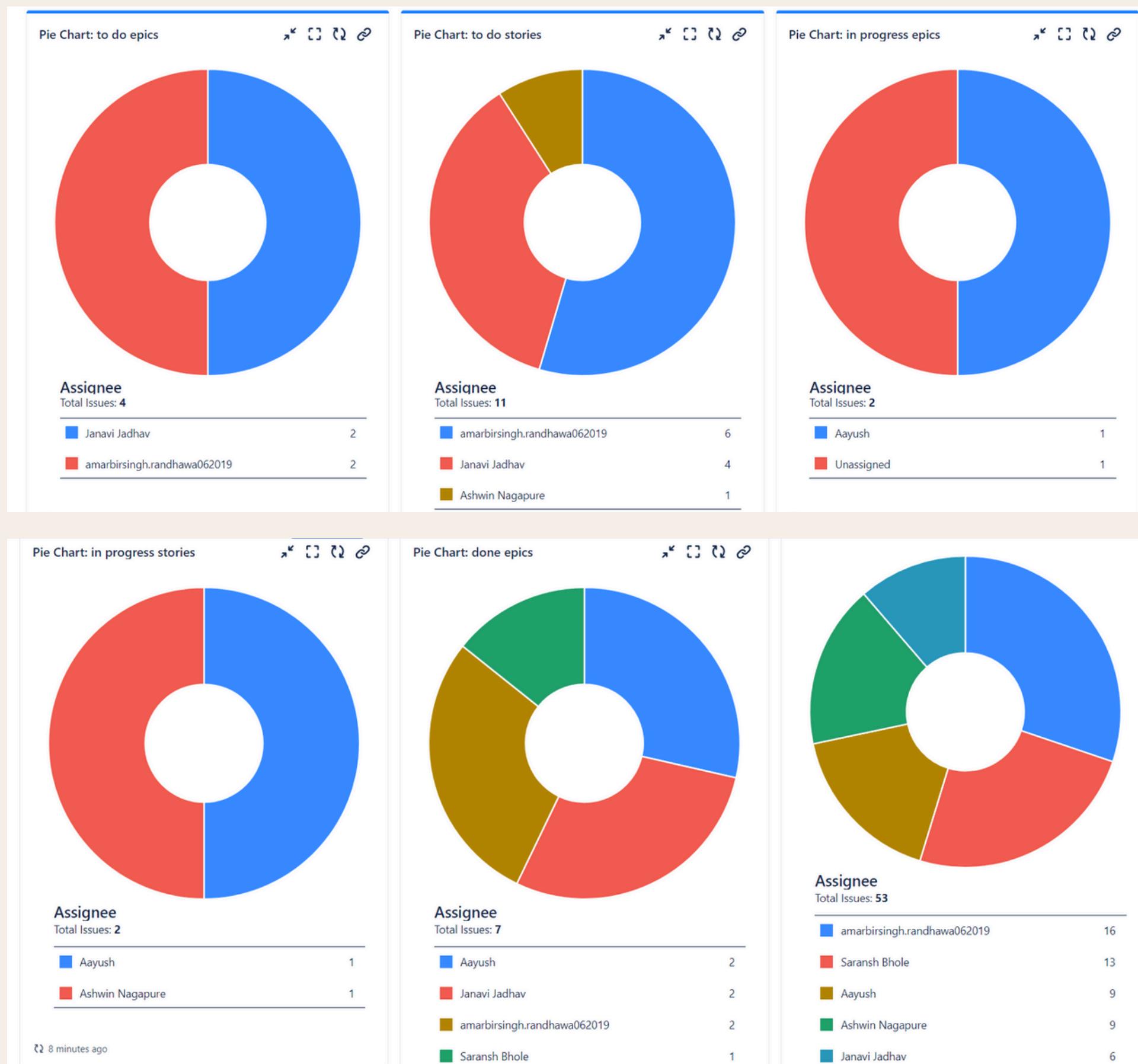
## Jira Software:

- Used to frame epics and user stories.
- Managed sprints, assigned tasks, and tracked progress.
- Ensured transparency and collaboration within the team.

## Distribution of Modules:

- **Ashwin:** User, Passenger, Admin, User Management
- **Saransh:** Booking, Baggage, Boarding
- **Aayush:** Flight, Flight Search, Flight Management, User Authentication
- **Amarbir:** Crew Management, Charter, Travel Preferences
- **Janavi:** Payment, Payment Receipt, Notification

# JIRA DASHBOARD



Two Dimensional Filter Statistics: done ...

Assignee	DONE	T:
Aayush	9	9
amarbirsingh.randhawa062019	16	16
Ashwin Nagpure	9	9
Janavi Jadhav	6	6
Saransh Bhole	13	13
<b>Total Unique Issues:</b>	<b>53</b>	<b>53</b>

Grouped by: Status Showing 5 of 5 statistics.

# IMPLEMENTATION HIGHLIGHTS

- **Swagger UI Integration**

The image shows a desktop screen with two browser windows side-by-side, illustrating the integration of Swagger UI across multiple API controllers.

**Left Browser Window:** Displays the Swagger UI for the **passenger-controller**. It lists the following endpoints:

- GET /passenger** (highlighted in blue)
- PUT /passenger** (highlighted in orange)
- POST /passenger** (highlighted in green)
- GET /passenger/{passengerId}**
- DELETE /passenger/{passengerId}** (highlighted in red)

**Right Browser Window:** Displays the Swagger UI for the **travel-preferences-controller**. It lists the following endpoints:

- GET /travel-preferences/{travelPreferencesId}**
- PUT /travel-preferences/{travelPreferencesId}** (highlighted in orange)
- DELETE /travel-preferences/{travelPreferencesId}** (highlighted in red)
- GET /travel-preferences**
- POST /travel-preferences** (highlighted in green)
- GET /travel-preferences/passenger/{passengerId}**

Both browser windows have a header bar with various icons and a system tray at the bottom showing weather, search, and system status.

# IMPLEMENTATION HIGHLIGHTS (CONTD.)

The image displays two side-by-side browser windows, both showing the Swagger UI interface for a RESTful API. The left window has a dark theme and lists the following controllers:

- charter-controller**
  - GET /charters/{id}
  - PUT /charters/{id}
  - DELETE /charters/{id}
  - GET /charters
  - POST /charters
  - POST /charters/assign-passenger/{charterId}/{passengerId}
  - GET /charters/passenger/{passengerId}
- booking-controller**
  - PUT /bookings/{id}/confirm
  - POST /bookings/{passengerId}/flight/{flightId}
  - POST /bookings/{passengerId}/charter/{charterId}
  - GET /bookings
  - GET /bookings/{id}
  - DELETE /bookings/{id}
- log-controller**
  - GET /logs
  - POST /logs
  - GET /logs/{id}

The right window also has a dark theme and lists the following controllers:

- payment-controller**
  - POST /payments/{bookingId}
  - GET /payments/{id}
- payment-receipt-controller**
  - POST /payment-receipts
  - GET /payment-receipts/{id}
- notification-controller**
  - POST /notification/{adminId}/send/{passengerId}
  - GET /notification
  - GET /notification/{id}
  - DELETE /notification/{id}
- log-controller**
  - GET /logs
  - POST /logs
  - GET /logs/{id}

# IMPLEMENTATION HIGHLIGHTS (CONTD.)

## Purpose:

- Swagger UI provides an interactive API documentation interface, allowing developers and testers to visualize, test, and debug APIs directly from the browser.

## Benefits of Swagger UI:

- Centralized documentation for all APIs.
- Easy to test APIs with built-in tools.
- Improves collaboration between developers and stakeholders.
- Helps identify issues early in development.

## Impact:

- Simplifies collaboration by enabling team members to understand and test APIs without requiring external tools.



# IMPLEMENTATION HIGHLIGHTS (CONTD.)

- SonarQube Integration

The screenshot shows the SonarQube community interface for the project 'ATM / main'. The navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', 'Administration', 'More', and a search icon. The top right corner has a help icon and a user profile. The breadcrumb path is 'ATM / main'. The main menu tabs are 'Overview' (selected), 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity'. On the right, there are 'Project Settings' and 'Project Information' dropdowns. A message at the top right says 'Last analysis 10 minutes ago'. A large green checkmark indicates the 'Passed' status of the quality gate. A yellow warning box states 'The last analysis has warnings. [See details](#)'. Below this, there are two tabs: 'New Code' (selected) and 'Overall Code'. The 'Overall Code' section displays various metrics:

Category	Value	Grade
Security	1 Open issues	E
Reliability	31 Open issues	C
Maintainability	180 Open issues	A
Accepted issues	0	(grey)
Coverage	0.0%	O
Duplications	11.9%	(red)
Security Hotspots	2	E

Details for coverage: 'On 982 lines to cover.' Details for duplications: 'On 4.4k lines.'

# IMPLEMENTATION HIGHLIGHTS (CONTD.)

## Purpose:

- SonarQube ensures high-quality code by spotting bugs, security risks, and areas for improvement.

## Metrics:

- **Code Coverage:** How much of the code is tested.
- **Maintainability:** Measures complexity or technical debt.
- **Reliability:** Highlights potential bugs.
- **Security:** Flags vulnerabilities.

## Impact:

- It helps keep the system strong, secure, and easy to maintain, ensuring long-term success.



# SOLID PRINCIPLES

- **Single Responsibility Principle (SRP):** A class should have one reason to change.
  - DTOs like BaggageDTO, BookingDTO, and CharterDTO focus solely on transferring data.
  - **Benefit:** Improves maintainability and clarity.
- **Open-Closed Principle (OCP):** Open for extension, closed for modification.
  - Interfaces like PaymentService and BookingService allow adding new features (e.g., payment methods) without modifying existing code.
  - **Benefit:** Facilitates easy addition of new functionality.
- **Liskov Substitution Principle (LSP):** Derived classes should substitute base classes without altering program correctness.
  - CharterController can work seamlessly with any implementation of CharterService interface, adhering to LSP.
  - **Benefit:** Ensures consistent behavior across implementations.

# SOLID PRINCIPLES (CONTD.)

- **Interface Segregation Principle (ISP):** Classes should not be forced to implement interfaces they don't use.
  - BookingServiceImpl is only responsible for implementing BookingService methods.
  - FlightServiceImpl and PassengerServiceImpl deal only with their specific interfaces.
  - **Benefit:** Prevents bloated and unused interface methods.
- **Dependency Inversion Principle (DIP):** High-level modules depend on abstractions, not concrete implementations.
  - BaggageController depends on the BaggageService interface, not its BaggageServiceImpl.
  - The BoardingPassServiceImpl uses the BoardingPassRepository abstraction.
  - **Benefit:** Increases flexibility and testability.



# OOPS CONCEPTS

- **Encapsulation:** Grouping data and methods in classes, restricting access.
  - DTOs like BookingDTO encapsulate fields with getters and setters.
  - Services like BookingService encapsulate business logic.
  - **Benefit:** Prevents unintended changes and enforces data integrity.
- **Abstraction:** Hiding implementation details, focusing on essential features.
  - Repositories like BookingRepository abstract database operations.
  - **Benefit:** Increases code modularity and reusability.



# OOPS CONCEPTS (CONTD.)

- **Inheritance:** Reuse code via parent-child relationships.
  - User is extended by Passenger and Admin.
  - **Benefit:** Reduces code duplication and improves maintainability.
- **Polymorphism:** A single interface represents different implementations.
  - At runtime, the interface type CrewManagementService is used to hold an object of the implementing class CrewManagementServiceImpl, enabling polymorphism.
  - **Benefit:** Simplifies switching between implementations.





# CHALLENGES FACED

- Integration of Swagger UI and SonarQube with Spring Boot
- Resolving dependency conflicts and maintaining compatibility across modules
- Managing team collaboration and resolving version control merge conflicts
- Designing scalable system architecture and clear workflows
- Ensuring adherence to Agile methodology and timely sprint completion
- Mapping of all the modules with each other for smooth functioning



# ACHIEVEMENTS

- Created APIs and CRUD operations for all modules in backend
- Performed mapping among all the modules in the backend
- Designed UML class diagram for our system
- Learnt and integrated SonarQube with Spring Boot
- Learnt and integrated Swagger with Spring Boot
- Learnt and used Jira software for managing sprints and framing epics and user stories
- Implemented SOLID principles and OOPs concepts in the project



# CONCLUSION

- Developed a scalable and maintainable system with robust modular architecture
- Followed Agile methodology using Jira for effective planning, tracking, and collaboration
- Ensured code quality and documentation with SonarQube and Swagger UI
- Implemented SOLID principles and OOP concepts for modular, reusable, and testable code



THANK YOU!  
TEAM ALPHA