PROJECT

FLIGHT RESERVATION

SYSTEM

Was done by: Alzhan Temirlan,

**laboratory work 1**

**Exercise 01 – SoftWare Requirments specification**

Doc Type: Requirements Description

System: Flight Reservation System

Revision: V.0\_0

Revision Date: << to be filled in by course instructor to make relevant >>

Requested By: Alzhan Temirlan, Tileumbetov Batyrzhan, Kantpay Arsen

The Flight Reservation System will facilitate customers in booking, modifying, and canceling flights. It will interact with a central database containing information about available flights, seats, and customer profiles. The system will also handle payment processing through secure channels.

Customers will be able to search for flights based on various criteria, select seats, and make reservations. The system should provide options for customers to modify their reservations or cancel them, adhering to specified rules and policies.

The Flight Reservation System must integrate with the airline's existing loyalty program, allowing customers to accumulate and redeem points for discounts or upgrades.

The system should generate electronic tickets for customers, including relevant flight details and booking references. It should also send confirmation emails and notifications for reservation changes.

The Flight Reservation System will have an administrative module to manage flight schedules, seat availability, and customer support. Administrators should be able to view and manage reservations, process refunds, and generate reports on system usage.

The security of customer data and transactions is paramount. The system should comply with industry standards for data protection and implement secure authentication mechanisms.

**Functional Requirements:**

Search and Booking:

Customers can search for flights based on criteria such as date, destination, and class.

Customers can select available seats and book flights.

The system must check seat availability and confirm bookings with a unique reference.

Reservation Management:

Customers can modify existing reservations within specified time limits.

Customers can cancel reservations, adhering to the airline's cancellation policies.

Payment Processing:

The system should securely process payments for flight reservations.

Electronic tickets should be generated upon successful payment.

Loyalty Program Integration:

Customers can enroll in the loyalty program.

The system should track and update loyalty points for customers.

Notifications:

Confirmation emails and electronic tickets should be sent to customers upon successful booking.

Customers should receive notifications for any changes or cancellations to their reservations.

**Non-Functional Requirements:**

Security:

The system must comply with industry standards for data protection.

Secure authentication mechanisms should be implemented.

Administrative Module:

Administrators should be able to manage flight schedules and seat availability.

Customer support features should be available in the administrative module.

Reporting:

The system should generate reports on reservation trends, usage statistics, and payment transactions.

This Flight Reservation System should allow customers to easily search, book, modify, and cancel flights. It should integrate with a loyalty program, ensure secure payment processing, and provide robust administrative tools. The security of customer data is a top priority, and the system should comply with industry standards. The specified functional and non-functional requirements form the basis of the software requirements specification.

EXERCISE 02 – Identifying Use Cases

2. Create a list of actors:

Customer: Interacts with the system for flight search, booking, modification, cancellation, and loyalty program.

Administrator: Manages flight schedules, seat availability, reservations, and customer support.

Loyalty Program: Tracks and updates loyalty points for enrolled customers.

Payment Gateway: Processes secure payments for flight reservations.

3. Identify use cases:

Customer:

Search for Flights

Book a Flight

View Booking History

Modify Reservation

Cancel Reservation

View Loyalty Points

Check Flight Status

Redeem Loyalty Points

System Administrator:

Manage User Accounts

Monitor System Performance

Manage Flight Information

Perform Security Audits

Backup and Restore Data

Provide Customer Support

Loyalty Program:

Accrue Loyalty Points

Track Loyalty Points

Update Loyalty Points

Send Promotional Offers

Reservation System:

Process Flight Reservations

Update Seat Availability

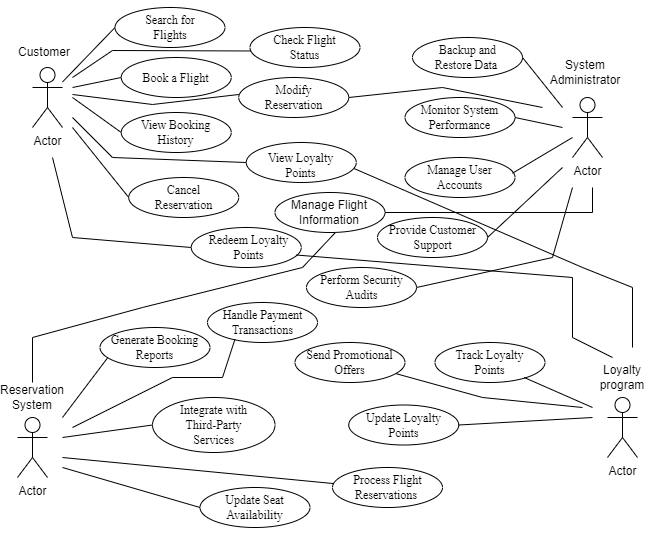
Handle Payment Transactions

Generate Booking Reports

Integrate with Third-Party Services

Summary:

The Use Case diagram for the Flight Reservation System includes actors such as Customers, Administrators, Loyalty Program, and Payment Gateway. Various use cases cover functionalities like searching for flights, booking, reservation management, loyalty program interactions, payment processing, and administrative tasks. The diagram provides a visual representation of the system's interactions with its environment.



**EXERCISE 03 – CREATING USE CASE TEMPLATE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Description | Actors | Assumptions | Steps | Variations | Quality Concerns | Issues |
| 1 | Search for Flights | Customer | Customer has access to the internet | 1. Customer logs in 2. Customer enters search criteria 3. System displays available flights | - No flights found for the entered criteria.  - Customer modifies search criteria. | User Interface: Intuitive search functionality | - Integration with external flight information providers. |
| 2 | Book a Flight | Customer | Customer has selected a flight | 1. Customer selects desired flight 2. Customer enters personal information 3. Customer selects payment method 4. System confirms booking | - Payment failure.  - Customer modifies passenger details. | Security: Payment processing encryption | - Handling of payment gateway downtime. |
| 3 | Modify Reservation | Customer, System Administrator | Customer has a booked reservation | 1. Customer logs in 2. Customer selects reservation 3. Customer selects modification option 4. System processes modification | No reservations found for the customer.  Customer cancels the modification process. | Flexibility: Allow changes to reservations within specified timeframe | - Handling of simultaneous modification requests. |
| 4 | Cancel Reservation | Customer | Customer has a booked reservation | 1. Customer logs in 2. Customer selects reservation 3. Customer selects cancellation option 4. System processes cancellation | - No reservations found for the customer.  - Customer cancels the cancellation process. | Refund Policy: Ensure refunds are processed correctly | - Handling of simultaneous cancellation requests. |
| 5 | Manage Flight Schedules | System Administrator | Administrator has system access | 1. Administrator logs in 2. Administrator accesses flight schedule module 3. Administrator adds, edits, or deletes schedules | - No flight schedules found.  Administrator cancels a flight. | Data Integrity: Ensure accurate schedule updates | - Conflict resolution in simultaneous schedule modifications. |
| 6 | Process Refunds | Reservation System | Administrator has system access | 1. Administrator logs in 2. Administrator accesses refund module 3. Administrator processes refund request | - No refund requests found.  - Refund verification failure. | Accuracy: Ensure correct refund amounts | - Handling of fraudulent refund requests. |
| 7 | Enroll in Loyalty Program | Customer | Customer has a registered account. | 1. Customer navigates to the loyalty program section.  2 System prompts for enrollment details.  3. Customer submits enrollment information. | - Customer declines to enroll.  - Enrollment form submission failure. | - Secure storage of loyalty program enrollment data.  - Promptness in processing enrollment requests. | - Handling of incomplete or incorrect enrollment submissions. |
| 8 | View Loyalty Points | Customer | Assumptions: Customer is enrolled in the loyalty program | 1. Customer checks the loyalty points section.  2. System displays the current points balance. | - Customer has no loyalty points.  - Points retrieval failure. | - Accuracy in displaying the current points balance.  - Responsiveness in updating points information. | - Points synchronization delays. |
| 9 | Redeem Loyalty Points | Customer | Assumptions: Customer has accumulated enough points. | 1. Customer selects items/services for redemption.  2. System deducts points and confirms the redemption. | - Insufficient points for redemption.  - Redemption failure. | - Accuracy in deducting and reflecting redeemed points.  - Confirmation accuracy for the redemption process. | - Inventory update delays for redeemed items. |
| 10 | Manage Reservations | System Administrator | Assumptions: Administrator is authenticated | 1. Administrator searches for reservations.  2. System displays reservation details.  3. Administrator processes reservation changes. | - No reservations found for the specified criteria.  - Reservation changes not allowed for specific flights. | - Data accuracy in displaying reservation details.  - Authorization checks for administrators. | - Handling of simultaneous reservation modification requests. |
| 11 | Manage Seat Availability | System Administrator, Reservation System | Assumptions: Administrator is authenticated. | 1. Administrator selects a flight and date.  2. System displays current seat availability.  3. Administrator adjusts seat availability. | - No flight or date selected.  - Seat availability adjustment failure. | - Data integrity in seat availability updates.  - Authorization checks for administrators. | - Handling of simultaneous seat availability adjustment requests. |
| 12 | Provide Customer Support | System Administrator | Assumptions: Administrator is authenticated. | 1. Administrator receives a customer support request.  2. System displays customer details and request.  3. Administrator provides support and logs the | - No customer support requests found.  - Support interaction logging failure. | - Accuracy in logging customer support interactions.  - Timely resolution of customer support requests. | - Handling of escalated support issues. |
| 13 | Track Loyalty Points | Loyalty Program | Assumptions: Loyalty Program is operational. | 1. Loyalty Program tracks points for eligible customer transactions. | - No eligible transactions for points tracking.  - Points tracking failure. | - Accuracy in tracking and updating loyalty points.  - Timely synchronization of points data. | - Handling of transaction data discrepancies. |
| 14 | Process Payment | Reservation System | Customer has completed booking details. | 1. Customer initiates the payment process.  2. Payment Gateway securely processes the payment. | - Payment failure.  - Customer cancels the payment process. | - Security in processing payment transactions.  - Confirmation accuracy for successful payments. | - Handling of payment gateway unavailability. |
| 15 | Generate Reports | System Administrator | Administrator is authenticated | 1. Administrator selects report parameters.  2. System generates and displays the requested report. | - No report parameters specified.  - Report generation failure. | - Accuracy in report generation.  - Timeliness in providing reports. | - Handling of large data volumes for reports. |
| 16 | Update Loyalty Points | Loyalty Program | Assumptions: Loyalty Program is operational. | 1. Loyalty Program updates customer points based on specified criteria. | - No points update criteria specified.  - Points update failure. | - Accuracy in updating customer points.  - Timely processing of points updates. | - Handling of unexpected points update scenarios. |

**LABORATORY WORK 2**

**EXERCISE 01 – Thinking About Risk**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Risk** | **Importance** | **Difficulty** | **Risklevel** | **Techniques for MS** | **Addressed** | **Questions** | **ViewType** | **Mitigates** | **Risky** |
| System outage due to server failure | 5 | 4 | 4.5 | Implementing a robust backup and recovery system.(Architecture) | Developer | Where are the backup servers located? | Allocation view type | Regularly testing backup and recovery procedures. | - |
| Unauthorized access to customer data | 5 | 4 | 4.5 | Implementing strong encryption and secure authentication mechanisms.(Architecture) | Developer | How often are access controls reviewed and updated? | Module view type | Regularly reviewing and updating access controls. | - |
| Payment processing failure | 4 | 3 | 3.5 | Implementing redundant payment gateways and error handling mechanisms.(Architecture) | Developer | What is the maximum allowable downtime for payment processing? | Runtime view type | Regularly testing payment processing systems. | - |
| Inadequate integration with the loyalty program | 4 | 3 | 3.5 | Conducting thorough testing of loyalty program integration modules.(Management) | USER X | How frequently is the loyalty program data synchronized with the Flight Reservation System? | Allocation view type | Implementing real-time data synchronization. | - |
| Network Connectivity Issues | 4 | 3 | 3.5 | Implementing redundancy in network connections and having failover mechanisms(Architecture) | user X | How often are network connections tested for reliability? | Allocation view type | Regularly testing network redundancy mechanisms. | - |
| Third-Party Integration Problems | 4 | 4 | 4 | Regularly testing and updating third-party APIs, ensuring compatibility.(Architecture) | Electrician | How often are third-party APIs reviewed for updates or changes? | Module view type | Regularly monitoring third-party integrations for issues. | - |
| Flight Schedule Changes | 3 | 3 | 3 | Providing timely notifications to affected customers and allowing flexible rebooking.(Architecture) | Developer | How quickly can the system adapt to unexpected flight schedule changes? | Runtime view type | Implementing real-time updates for flight schedules. | - |
| Inadequate Load Testing | 4 | 4 | 4 | Conducting thorough load testing to ensure the system can handle peak booking periods.(Architecture) | Developer | How often is the system tested under simulated peak loads? | Allocation view type | Regularly updating load testing scenarios. | - |
| Data Inconsistency | 4 | 3 | 3.5 | Implementing data validation checks and regular database audits.(Architecture) | Developer | What is the maximum allowable downtime for payment processing? | Runtime view type | Regularly testing payment processing systems. | - |
| Incomplete Payment Transactions | 3 | 4 | 3.5 | Implementing robust error-handling mechanisms for payment processing.(Architecture) | Developer | How quickly are payment processing errors detected and addressed? | Runtime view type | Regularly reviewing and updating error-handling procedures. | - |
| Inadequate User Authentication | 5 | 3 | 4 | Enforcing strong password policies and implementing multi-factor authentication.(Architecture) | Developer | How often are user authentication mechanisms reviewed for security? | Module view type | Regularly updating user authentication protocols. | - |
| Customer Privacy Violation | 4 | 4 | 4 | Regularly reviewing and updating privacy policies, conducting privacy impact assessments.(Management) | Developer | How is customer data handled in compliance with privacy regulations? | Module view type | Regularly auditing privacy compliance processes. | - |
| Software Compatibility Issues | 3 | 4 | 3.5 | Ensuring the Flight Reservation System is compatible with various browsers and operating systems.(Architecture) | Electrician | ow often are compatibility tests performed with different software configurations? | Allocation view type | Regularly updating compatibility testing procedures. | - |
| Insufficient Backup Testing | 4 | 3 | 3.5 | Regularly testing backup and recovery procedures to ensure data integrity.(Architecture) | Developer | How frequently are backup and recovery tests conducted? | Runtime view type | Implementing automated backup testing processes. | - |

**laboratory work 3**

**EXERCISE 01 – software life cycle modeling**

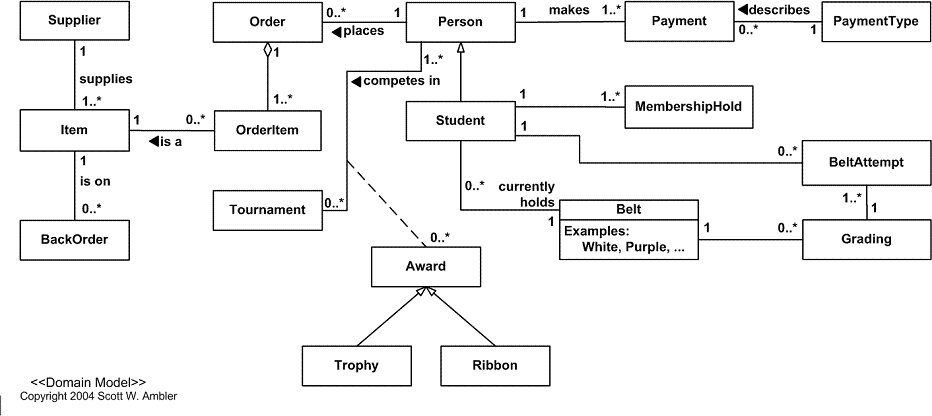
|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Description** | **Advantages** | **Disadvantages** |
| Waterfall | The Waterfall model follows a sequential approach where each phase (requirements, design, implementation, testing, deployment) is completed before moving to the next one. | Simple to use, every phase has a defined result and process review, development stages go one by one, easy to determine the key points in the development cycle, easy to classify and prioritize tasks | The software is ready only after the last stage is over, high risks and uncertainty, not the best choice for complex and object-oriented projects, Inappropriate for the long-term projects Integration Is done at the very end, which does not give the option of identifying the problem in advance |
| Agile | Agile is an iterative software development approach that prioritizes customer collaboration, adaptability, and continuous improvement. It breaks projects into short sprints, with cross-functional teams working on prioritized user stories. | Corrections of functional requirements are implemented into the development process to provide  the competitiveness, Project is divided by short and transparent iterations, Risks are minimized thanks to the flexible change process, Fast release of the first product version | Difficulties with measuring the final cost because of permanent changes, The team should be highly professional and client-oriented, New requirements may conflict with the existing architecture, With all the corrections and changes there is possibility that the project will exceed expected time |

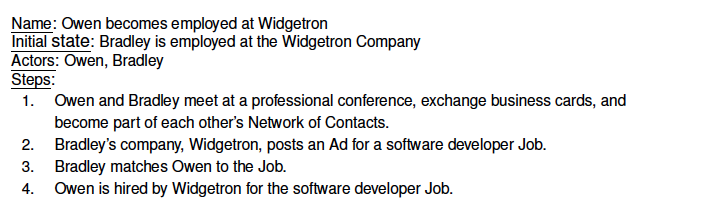
Summary: I have chosen Waterfall, and Agile models because of the diverse needs and constraints involved in developing a flight reservation system. The Waterfall model's structured approach aligns well with projects where requirements are well-defined and changes are minimal. Its sequential nature ensures thorough documentation and systematic progress through analysis, design, implementation, testing, deployment, and maintenance phases. While Agile emphasis on collaboration, adaptability, and frequent customer feedback is particularly beneficial for rapidly changing requirements and complex systems like flight reservation systems, ensuring efficient delivery of high-quality software. Each of these models offers distinct advantages, allowing for tailored approaches to meet the specific needs of the flight reservation system project.

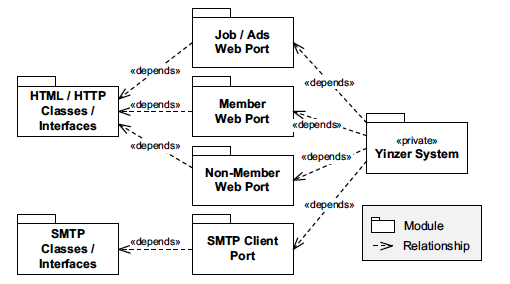
**laboratory work 4**

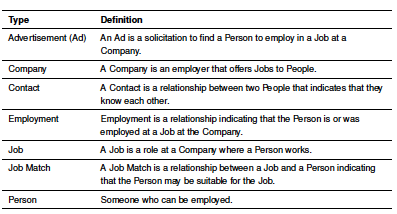
**EXERCISE 1 – Analyze Architecture Diagrams**

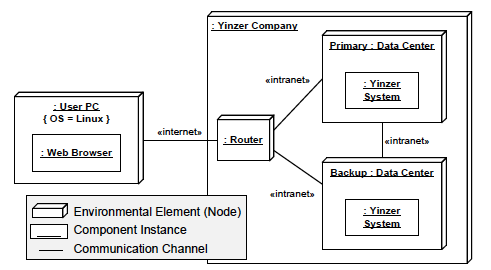
VIEW DIAGRAMS:

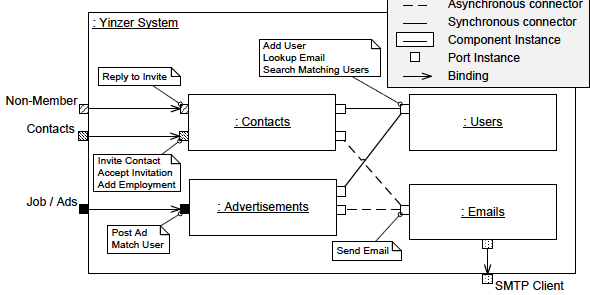
1.  Domain models is a structured visual representation of connected concepts.

1.  Use case diagram description of every step and actors

1. A view of the Yinzer boundary model that shows the externally-visible modules and dependences 

1.  A textual information model for the job ad and business networking domain.

1.  Allocation view type - the view type that contains views of elements related to the deployment of the software onto hardware.
2. A component assembly of Yinzer System from the internals model.



**EXERCISE 2 – Exploring ATM Domain via the Information Model**

|  |  |
| --- | --- |
| Type | Definition |
| Customer | An individual who uses the system to search for and book flights |
| Customer Support | Provides customer support for booking assistance |
| System Administrator | Manages system functionality, oversees transactions, helps customers if needed, and handles system maintenance |
| Loyalty program | A system component that tracks and updates loyalty points for customers. |
| Loyalty points | Customer can exchange loyalty points for discounts and other events |
| Reservation system | The central software that facilitates the interaction between customers, system administrators, and other components of the flight reservation system |
| Booking | A process of reservation of flights |
| Overbooking | The practice of selling more bookings than can be accommodated |
| Payment | A process of making transactions between customer and bank via Reservation System |
| Flight tickets | A document with a price showing the cost of that travel, passenger and journey details. Can be accepted electronic or paper version with passports. |
| Seat availability | Shows the availability of seats on different flights. |

**Relationships Among Roles**

- Customer to Reservation System: Customers interact directly with the Reservation System to manage their travel and loyalty program.

- System Administrator to Reservation System: Administrators use the system to manage and oversee flight-related operations.

- Loyalty Program to Customer: The Loyalty Program interacts with customers through the Reservation System for points tracking and redemption.

**laboratory work 5**

**EXERCISE 1 – Actors, individuals, and the Banking Domain**

1.

a. Customer

b. System Administrator

c. Loyalty Program

d. Reservation System

2.

a. Customer booking flight

b. Customer searches for flight

c. Customer modify reservation

d. Customer modify reservation

f. Customer enroll in loyalty program

e. Customer can view his/her own loyalty points

g. Customer exchange loyalty points

h. System administrator can manage flight schedules

i. System administrator can manage seat availability

j. System administrator can manage reservations

k. System administrator can process refunds

l. System administrator can generate reports

m. System administrator provide customer support

n. Loyalty program track loyalty points

o. Loyalty program update loyalty points

p. Reservation system facilitate search and booking of flights for Customers

q. Reservation system support System Administrators in managing flight schedules, reservations, and customer queries

r. Reservation system Interface with the Loyalty Program to track and redeem points

s. Reservation system Process payments through the Payment Gateway

3.a. If Paul is both a Customer and a System Administrator of the flight reservation system, are there any potential conflicts?

If Paul is both a reservation agent and a customer, conflicts could arise if he gives preferential treatment to his own reservations, manipulates booking availability, or accesses sensitive customer data for personal use.

Software Requirements: Enforce strict role separation, ensuring Paul cannot make reservations for himself while acting as an agent. Access controls, activity logs, and auditing should be in place to monitor and prevent misuse of privileges.

b. If Maria is both a manager of loyalty program and a customer of the flight reservation system, are there any potential conflicts?

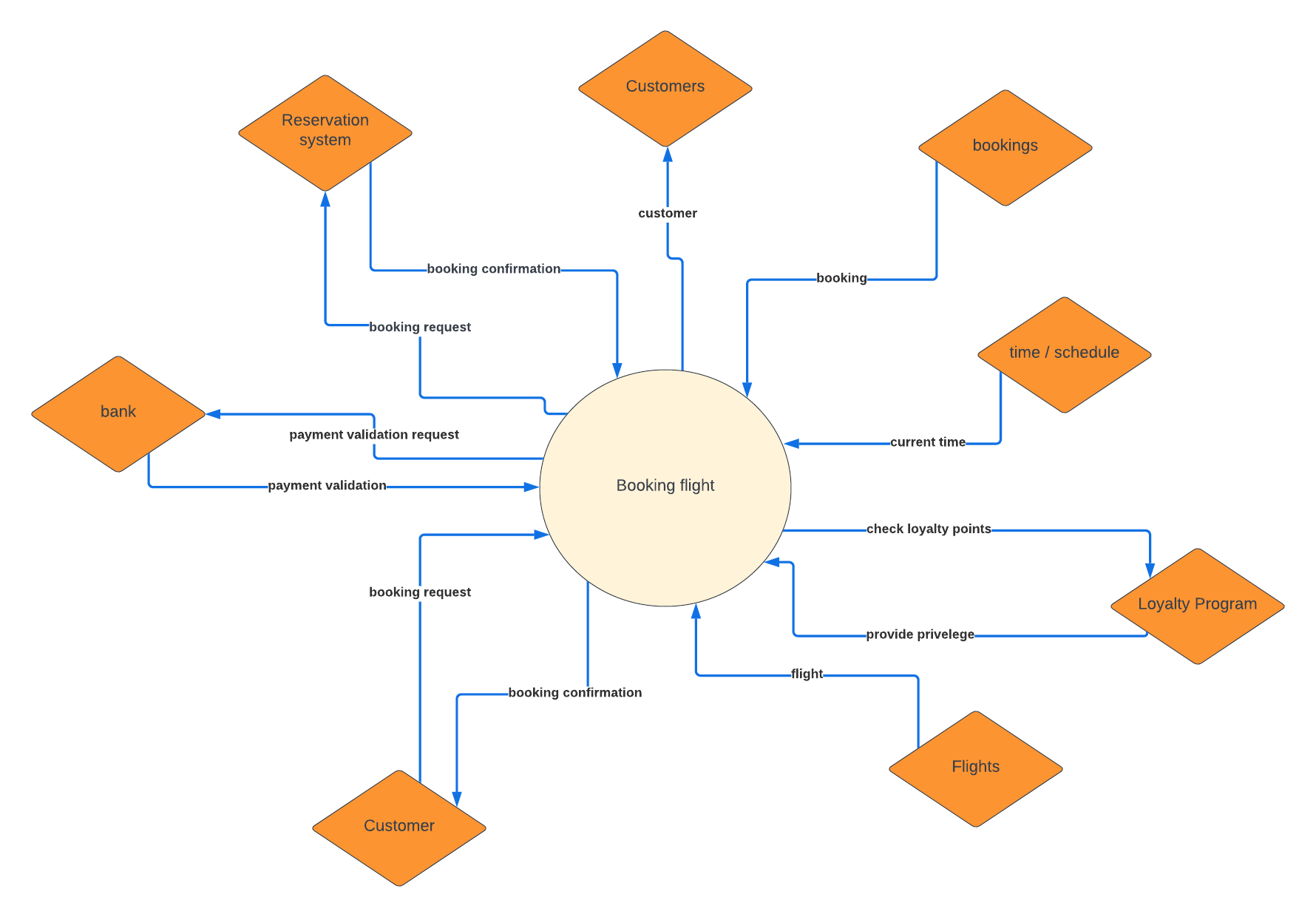
If Maria is both a manager of loyalty program and a customer of the flight reservation system conflict would be in terms of that Marie may change her own loyalty points and will able to have a lot of discounts.

Software requirements: Enforce strict role separation, ensuring marie cannot update loyalty programs for herself while acting as manager of loyalty program. Access controls should be in place to monitor and prevent misuse of privileges.

c. Multiple administrators might inadvertently make conflicting reservations or changes to the same flight. Also, there can be a software overload cause of Increased load on the system when multiple administrators are accessing or modifying flight data simultaneously. Software solution to this is Implement a reservation conflict resolution system that alerts administrators of conflicting actions.  
Ensure the system can handle simultaneous requests by administrators without performance degradation.  
Provide tools for administrators to communicate within the system to avoid conflicts and overlap.

d. The system should support joint bookings and itinerary management for Maria and Paul. Both Maria and Paul should receive notifications for any changes or updates to their shared reservations. To solve this situation we should Implement features for joint reservations where both customers' approvals are required for changes. The second decision is to allow for shared profiles or accounts where Maria and Paul can manage their joint bookings together. Overall, we need to ensure that notifications and updates are sent to both Maria and Paul for transparency and coordination.

**EXERCISE 2 – Creating Context Diagrams**



**EXERCISE 3 – Major FRS Decisions**

Revisiting High-Priority Risks:

1. System outage due to server failure: Implemented redundancy and failover mechanisms.
2. Unauthorized access to customer data: Enhanced security measures, including encryption and access controls.
3. Payment processing failure: Improved transaction processing and backup systems.
4. Inadequate integration with the loyalty program: Ensured API compatibility and robust testing.
5. Network connectivity issues: Implemented network redundancy and monitoring.
6. Third-party integration problems: Standardized API interfaces and thorough testing protocols.
7. Flight schedule changes: Integrated real-time data update mechanisms.
8. Inadequate load testing: Conducted extensive load testing simulations.
9. Data inconsistency: Implemented strong data validation and synchronization protocols.
10. Incomplete payment transactions: Established reliable transaction rollback mechanisms.
11. Inadequate user authentication: Enhanced authentication mechanisms, including multi-factor authentication.
12. Customer privacy violation: Implemented comprehensive privacy policies and data protection measures.
13. Software compatibility issues: Ensured compatibility through extensive testing across platforms.
14. Insufficient backup testing: Conducted regular backup and recovery testing.

Recording Major Architectural Decisions:

For each decision, the following details are recorded:

Decision 1: Implement Redundancy and Failover Mechanisms

* Risk Addressed: System outage due to server failure.
* How Reflected in Architecture: Redundant servers and failover protocols are incorporated.
* Rejected Alternatives: Single server setup.
* Reason for Rejection: Lack of reliability and risk of complete system outage.

Decision 2: Enhance Security Measures

* Risk Addressed: Unauthorized access to customer data.
* How Reflected in Architecture: Use of encryption, secure access controls, and regular security audits.
* Rejected Alternatives: Basic password protection.
* Reason for Rejection: Insufficient protection against sophisticated attacks.

Decision 3: Improve Transaction Processing and Backup Systems

* Risk Addressed: Payment processing failure.
* How Reflected in Architecture: Robust payment gateways with backup systems.
* Rejected Alternatives: Single payment gateway without backups.
* Reason for Rejection: High risk of transaction failures and customer dissatisfaction.

Decision 4: Ensure API Compatibility and Robust Testing

* Risk Addressed: Inadequate integration with the loyalty program.
* How Reflected in Architecture: Comprehensive API documentation and testing protocols.
* Rejected Alternatives: Limited testing and documentation.
* Reason for Rejection: High risk of integration issues.

Decision 5: Implement Network Redundancy and Monitoring

* Risk Addressed: Network connectivity issues.
* How Reflected in Architecture: Multiple network pathways and real-time monitoring.
* Rejected Alternatives: Single network pathway.
* Reason for Rejection: Increased risk of connectivity loss.

Decision 6: Standardize API Interfaces and Testing Protocols

* Risk Addressed: Third-party integration problems.
* How Reflected in Architecture: Standardized API design and extensive testing.
* Rejected Alternatives: Ad hoc API designs.
* Reason for Rejection: Unpredictable integration results and higher maintenance costs.

Decision 7: Integrate Real-Time Data Update Mechanisms

* Risk Addressed: Flight schedule changes.
* How Reflected in Architecture: Real-time data feeds and update protocols.
* Rejected Alternatives: Manual data updates.
* Reason for Rejection: Slow and error-prone process.

Decision 8: Conduct Extensive Load Testing Simulations

* Risk Addressed: Inadequate load testing.
* How Reflected in Architecture: Comprehensive load testing in various scenarios.
* Rejected Alternatives: Limited load testing.
* Reason for Rejection: Inability to ensure system stability under high load.

Decision 9: Implement Strong Data Validation and Synchronization Protocols

* Risk Addressed: Data inconsistency.
* How Reflected in Architecture: Real-time data validation and synchronization mechanisms.
* Rejected Alternatives: Basic validation checks.
* Reason for Rejection: High risk of data corruption and inconsistency.

Decision 10: Establish Reliable Transaction Rollback Mechanisms

* Risk Addressed: Incomplete payment transactions.
* How Reflected in Architecture: Transaction rollback protocols.
* Rejected Alternatives: No rollback mechanisms.
* Reason for Rejection: High risk of incomplete transactions causing financial discrepancies.

Decision 11: Enhance Authentication Mechanisms

* Risk Addressed: Inadequate user authentication.
* How Reflected in Architecture: Multi-factor authentication and secure login processes.
* Rejected Alternatives: Single-factor authentication.
* Reason for Rejection: Insufficient security against unauthorized access.

Decision 12: Implement Comprehensive Privacy Policies and Data Protection Measures

* Risk Addressed: Customer privacy violation.
* How Reflected in Architecture: Data encryption, access controls, and regular audits.
* Rejected Alternatives: Basic data protection.
* Reason for Rejection: Insufficient protection against data breaches.

Decision 13: Ensure Compatibility Through Extensive Testing Across Platforms

* Risk Addressed: Software compatibility issues.
* How Reflected in Architecture: Cross-platform compatibility testing.
* Rejected Alternatives: Limited compatibility testing.
* Reason for Rejection: High risk of software malfunctions on different platforms.

Decision 14: Conduct Regular Backup and Recovery Testing

* Risk Addressed: Insufficient backup testing.
* How Reflected in Architecture: Regularly scheduled backup and recovery drills.
* Rejected Alternatives: Irregular or no testing.
* Reason for Rejection: Unpreparedness for data recovery in case of failure.

Revisiting Project Requirements for Quality Concerns:

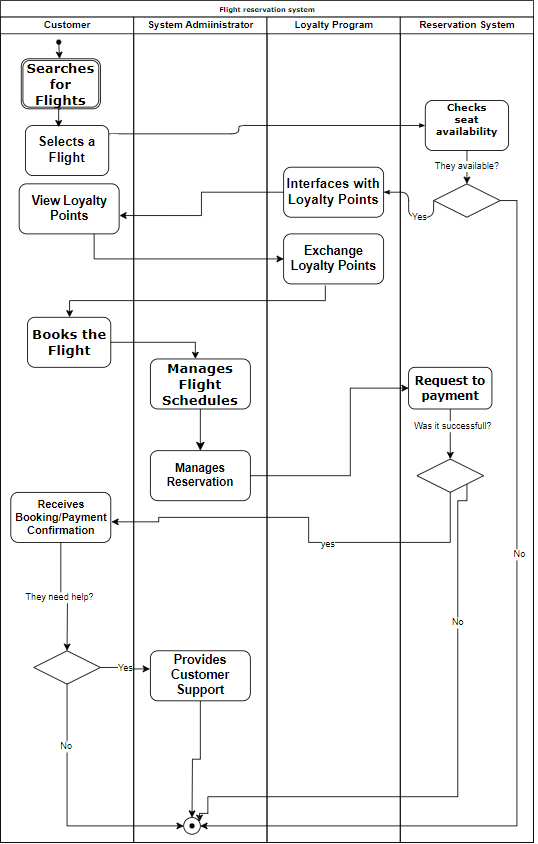
1. Security: Implemented encryption, access controls, and regular security audits.
2. Safety: Ensured robust transaction processing and data integrity.
3. Performance: Conducted extensive load testing and optimized network and server configurations.

Solutions to Quality Concerns:

1. Security: Implemented multi-layered security measures including encryption and multi-factor authentication.
2. Safety: Established reliable backup systems and data recovery protocols.
3. Performance: Conducted extensive performance testing and optimized system architecture for scalability.

**laboratory work 6**

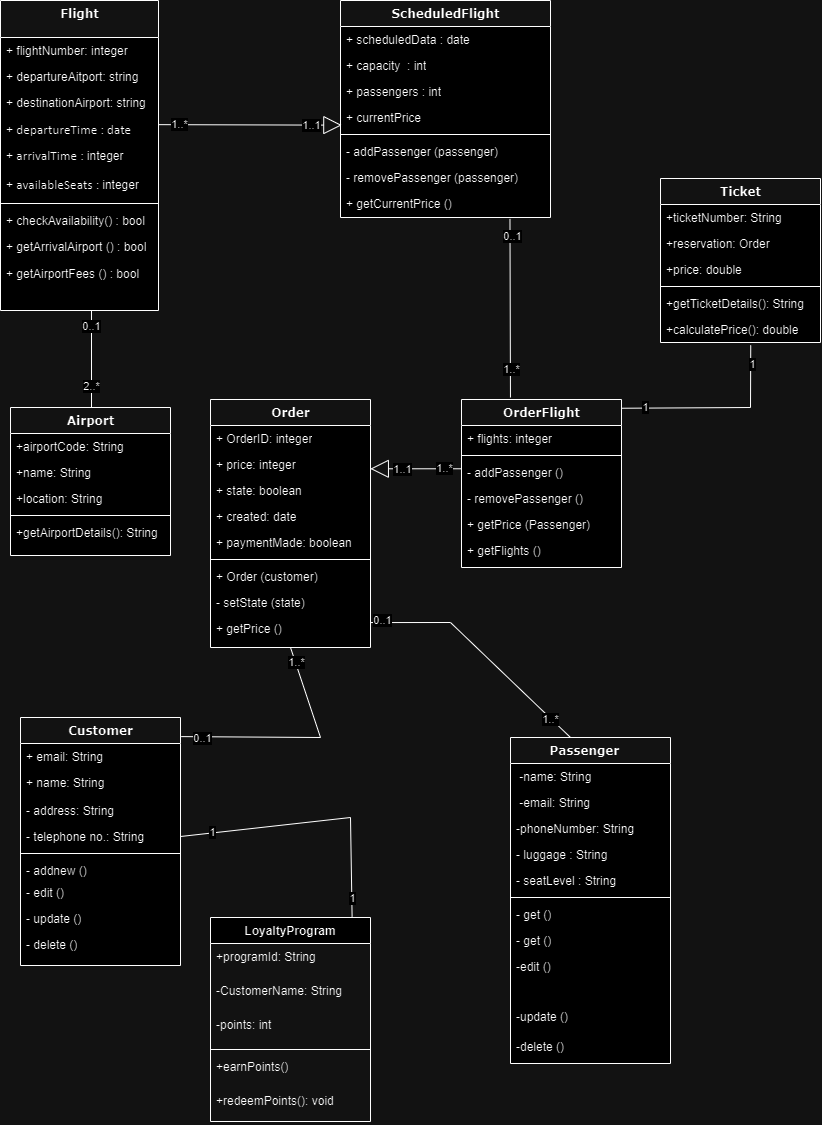
**EXERCISE 01 – Creating dynamic UML diagrams**

****

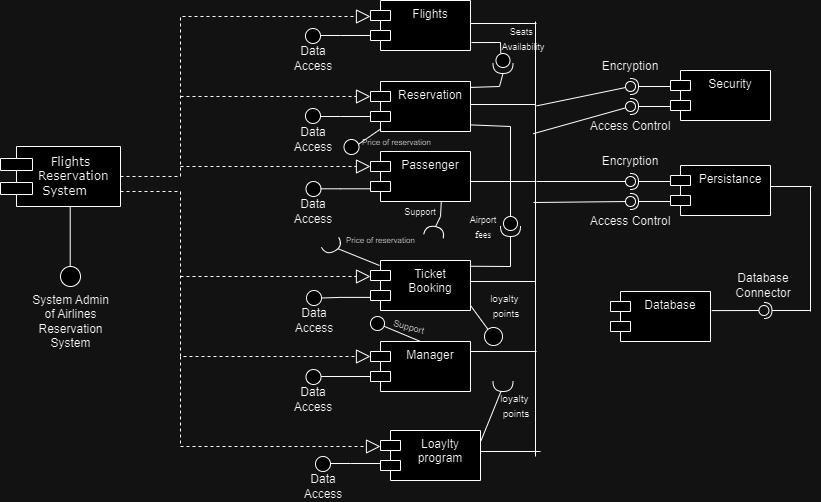
**LABORATORY WORK 7**

**EXERCISE 01 – CREATING UML DIAGRAMS**

Class diagram:

****

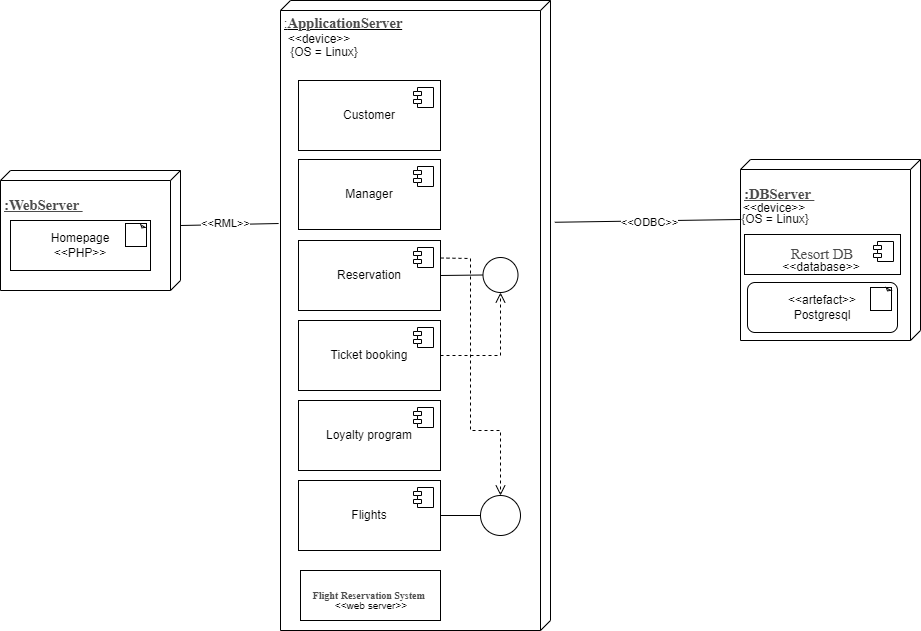
Component diagram:

****

**LABORATORY WORK 8**

**EXERCISE 01 – CREATING UML DIAGRAMS**

Deployment diagram:

****

**LABORATORY WORK 9**

**EXERCISE 01 – CHOOSE THE BEST PATTERN**

1. Creational pattern: Factory

Problem: Need to create a Flight Reservation System (FRS) with booking for domestic and international flights.

Solution: Use Factory Method pattern. Create FlightFactory class with createFlight() method. Implement DomesticFlightFactory and InternationalFlightFactory to create DomesticFlight and InternationalFlight instances.

Advantage:

Flexibility: Easily add new flight types without changing existing code.

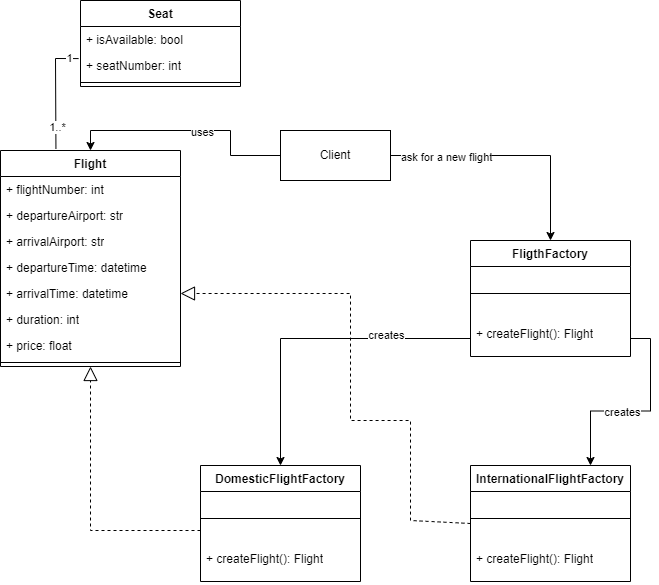
Encapsulation: Creation of flight objects is encapsulated, improving code organization.

Disadvantage:

Complexity: Introducing new types can increase complexity.

Maintenance: Changes to creation process may require modifications to multiple classes.

Graphical representation:



1. Creational patterns: Singleton

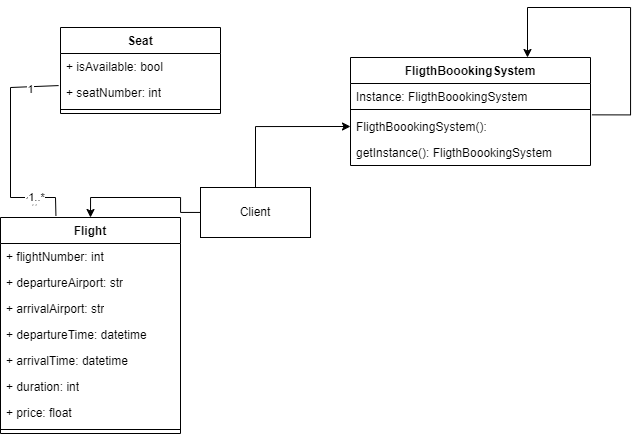
Problem Description: Need to ensure that there is only one instance of a FlightBookingSystem class.

Problem Solution: Implement the FlightBookingSystem class as a singleton.

Advantage: Guarantees that there is only one instance of the class, useful for resources that should be shared.

Disadvantage: Can make testing and debugging more difficult.

Graphical representation:



1. Structural patterns: Composite

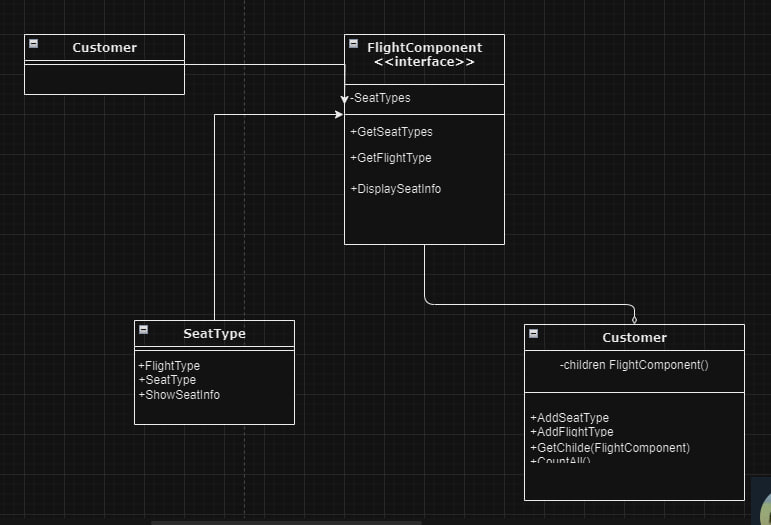
Problem Description: Need to represent flights with different seat types and prices as a single entity.

Problem Solution: Create a FlightComponent interface for flights and seat types, and a FlightComposite class to represent a group of flights or seat types.

Advantage: Allows treating individual flights and groups of flights uniformly.

Disadvantage: Adding new types of components can be complex.

Graphical Representation:



1. Structural patterns: Proxy

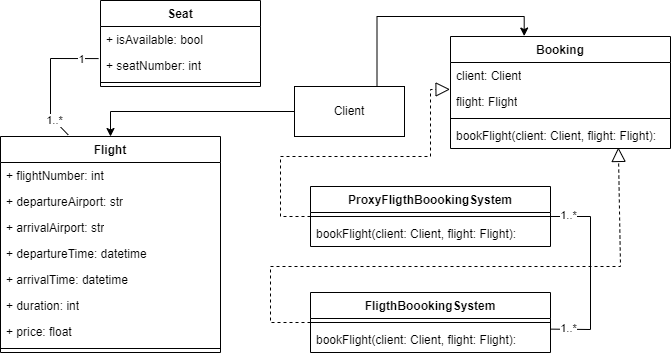
Problem Description: Need to control access to the FlightBookingSystem to manage concurrent bookings.

Problem Solution: Create a ProxyFlightBookingSystem class that delegates booking requests to the real FlightBookingSystem with additional logic for access control.

Advantage: Allows for additional security and access control.

Disadvantage: Can introduce overhead and complexity.

Graphical Representation:



1. Behavioral patterns: Observer

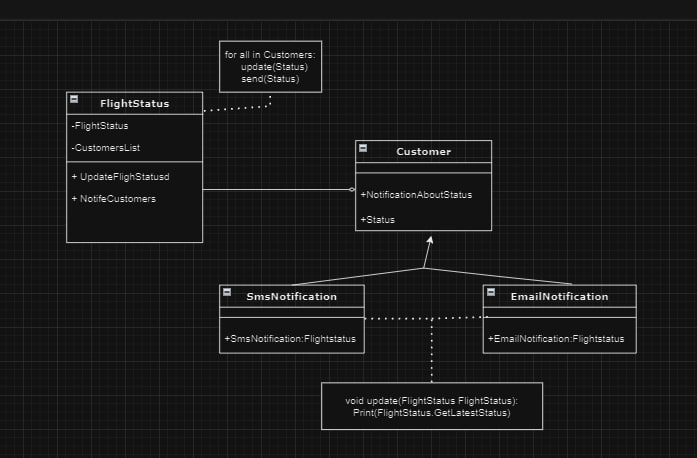
Problem Description: Need to notify passengers when their flight status changes (e.g., delayed, canceled).

Problem Solution: Implement an Observer pattern where passengers register as observers to flight status changes.

Advantage: Allows for a loosely coupled system where flight status changes do not directly affect passengers.

Disadvantage: Requires additional logic to manage observers and notifications.

Graphical Representation:



1. Behavioral patterns: Strategy

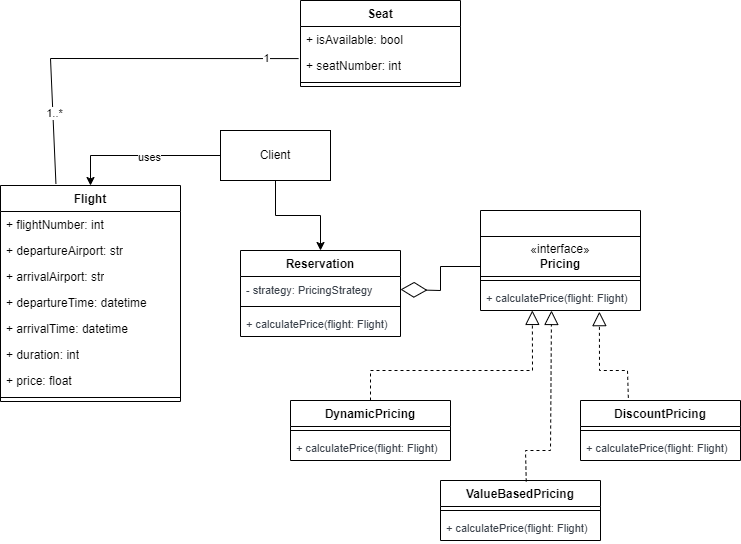
Problem Description: Need to implement different pricing strategies for flights (e.g., dynamic pricing, fixed pricing).

Problem Solution: Define a PricingStrategy interface with a calculatePrice method. Implement specific pricing strategies as concrete classes.

Advantage: Allows for easy swapping of pricing strategies at runtime.

Disadvantage: Can lead to a large number of classes if there are many pricing strategies.

Graphical representation:



**LABORATORY WORK 10**

**EXERCISE 01 – APPLYING ARCHITECTURAL STYLES**

**Layered Style**

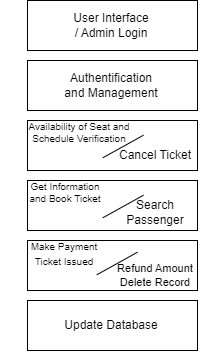
Viewtype : Module

Description: The FRS can be organized into layers such as presentation, business logic, and data access. Each layer represents a distinct set of functionalities and interacts with the layer directly beneath it. For example, the presentation layer handles user interfaces and admin logins, the business logic layer manages flight booking rules like cancel ticket and search passenger and delete record, and the data access layer communicates with the database.

Constraints / Guide Rails: Layers are stacked hierarchically, with higher layers depending only on lower layers. This ensures that changes in one layer do not affect other layers, promoting modifiability and reusability.

Qualities Promoted: Modifiability, portability, and reusability. The layered style allows for easy maintenance and updates to specific layers without affecting the entire system.

Elements&Relationships:



**Client-Server & N-Tier:**

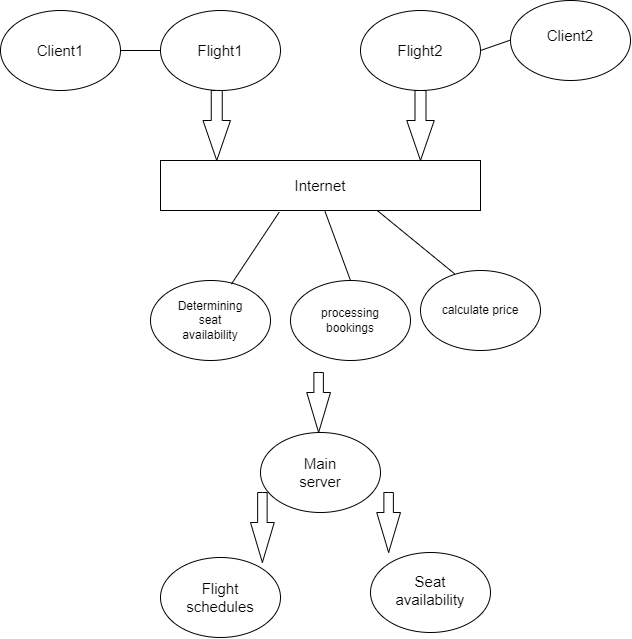
Viewtype: Runtime

Description: The FRS can be structured as a client-server system, where clients interact with servers to request flight information, make reservations, and manage bookings. Additionally, an N-tier architecture can be employed, dividing the system into presentation, application logic, and data management tiers.

Constraints / Guide Rails: Clients can request services from servers, but servers cannot initiate communication with clients. The N-tier architecture ensures that each tier has a specific responsibility, such as user interaction, business logic, and data persistence.

Qualities Promoted: Maintainability, evolvability, and scalability. The client-server & N-tier styles allow for easy management of system components and the ability to scale the system by adding more servers or tiers.

Elements&Relationships:



**Pipe and filter:**

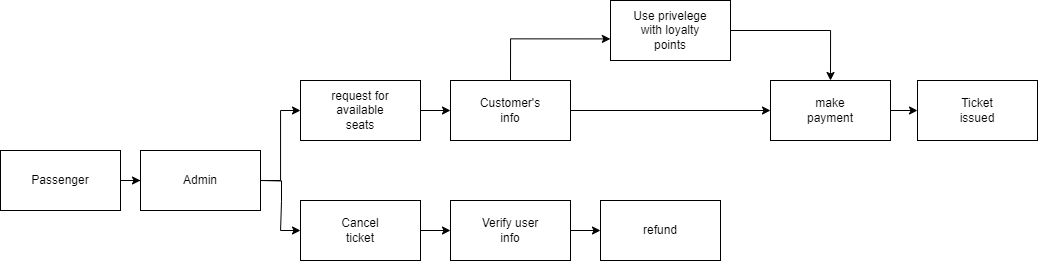
Viewtype: Runtime

Description: The FRS can utilize a pipe-and-filter architecture to process flight search requests and filter results based on various criteria such as price, duration, and airline preferences. Each filter component performs a specific processing task, and pipes connect them to pass data between filters.

Constraints / Guide Rails: Filters are independent and can be added, removed, or modified without affecting other components. The architecture encourages modularity and reusability of filter components.

Qualities Promoted: Modifiability, reusability, and extensibility. The pipe-and-filter style enables the FRS to easily adapt to changing requirements by modifying or adding new filters without impacting the overall system architecture.

Elements&Relationships:

****