CSE 3105 / CSE 3137

OBJECT-ORIENTED ANALYSIS AND DESIGN

FALL 2024

COURSE PROJECT: *LIBRARY MANAGEMENT SYSTEM*

System Design Document

Group 5

Berke Alpaslan – 210316016

Mert Doğan Aygün – 210316076

Gül Yeşilgil – 220316055

Birhat Taş – 190315010

Tural Mammadov – 190316001

Aziz Berk Yıldırım - 210316055

2 January 2025

Table of Contents

Table of Figures

1 Introduction

The purpose of this document is to present the system design for the *Library Management System*, which builds upon the requirements outlined in the Requirements Analysis Document (RAD). This document provides a high-level overview of the software architecture, outlining the design goals, system constraints, and overall structure of the solution.

The *Library Management System* is designed to facilitate critical library operations such as book borrowing and returning, book reservation, and inventory management. The architecture ensures scalability, usability, and maintainability while addressing functional and non-functional requirements derived from the RAD.

- **References**

1. Requirements Analysis Document (RAD) – Group5_RAD.pdf
2. Existing Library Management Practices
3. UML Diagrams (Use Case, Class, and Sequence Diagrams)

- **Design Goals**
- Provide an efficient and user-friendly system for library members and staff.
- Ensure data consistency, integrity, and security.
- Support scalability to accommodate library growth and increasing user interactions.
- Enable seamless integration with future modules or enhancements.
- **Constraints**
- The system must comply with the functional and non-functional requirements specified in the RAD.
- Existing library workflows and operational standards must be maintained.
- Design must adhere to best practices for modularity and reusability.

1.1 Purpose of the System

1.2 Design goals

I. **Scalability (Priority: High)**
The system must handle an increasing number of library members, books, and concurrent user operations without a significant decrease in performance. This ensures long-term viability and adaptability to future needs.

II. **Usability (Priority: High)**
A user-friendly interface is critical to ensure library members and staff can efficiently interact with the system. The interface must be intuitive and reduce the need for extensive training.

III. **Reliability (Priority: High)**
The system must provide accurate and consistent results for borrowing, returning, and reserving books. Data integrity and proper error handling mechanisms are essential.

IV. **Maintainability (Priority: Medium)**
The architecture should be modular and well-documented to simplify future updates, debugging, or integration with new features.

V. **Performance (Priority: Medium)**
The system should respond quickly to user actions, such as searches and inventory updates. Although not as critical as reliability, performance should not hinder user experience.

VI. **Security (Priority: High)**
Protecting sensitive information, such as user credentials and borrowing history, is essential. The system must include robust authentication, authorization, and encryption mechanisms.

VII. **Extensibility (Priority: Medium)**
The design must allow for the addition of new features or integration with other systems without major architectural changes.

VIII. **Cost-Efficiency (Priority: Low)**
While cost considerations are important, they should not compromise key system functionalities or quality.

Trade-Offs Discussion:
Achieving high reliability and security may slightly impact performance, especially during peak usage. Similarly, focusing on scalability and maintainability could increase initial development time and costs. However, prioritizing these design goals aligns with the system's long-term requirements and ensures overall robustness.

2 Current Software Architecture

As there is no previous system in place for library management, this section provides an overview of existing architectures for similar systems to guide the design of the new system.

I. **Centralized Database Architecture**
Most library management systems rely on a centralized database to store information about books, users, transactions, and inventory. This architecture ensures consistency and simplifies data management but can lead to bottlenecks under heavy load.

II. **Monolithic Architecture**
Traditional systems often use a monolithic architecture where all components are

tightly integrated. While simpler to implement, such systems are less scalable and harder to maintain as they grow in complexity.

III. **Service-Oriented Architecture (SOA)**
Some modern systems adopt an SOA approach, dividing functionalities into independent services (e.g., user management, book inventory, reservation system). This architecture improves modularity and maintainability but increases development effort and requires careful handling of inter-service communication.

IV. **Web-Based Interfaces**
Current library systems often feature web-based user interfaces for both librarians and members. These interfaces provide flexibility and accessibility but require additional measures for security and performance optimization.

Assumptions and Common Issues Addressed by the New System:

- **Assumption:** The library will require a scalable and secure system to manage growing user and inventory demands.
- **Issues Addressed:**
  - Limited scalability of traditional monolithic systems.
  - Difficulty in maintaining and updating tightly coupled components.
  - Lack of a user-friendly interface for efficient operations.
  - Insufficient security measures in older systems.

The new system will leverage a modular architecture based on the Onion model, ensuring scalability, maintainability, and security. By incorporating modern practices such as web-based access and API-driven communication, the system will address the limitations of traditional architectures while aligning with current technological standards.

3 Proposed Software Architecture

3.1 Subsystem decomposition

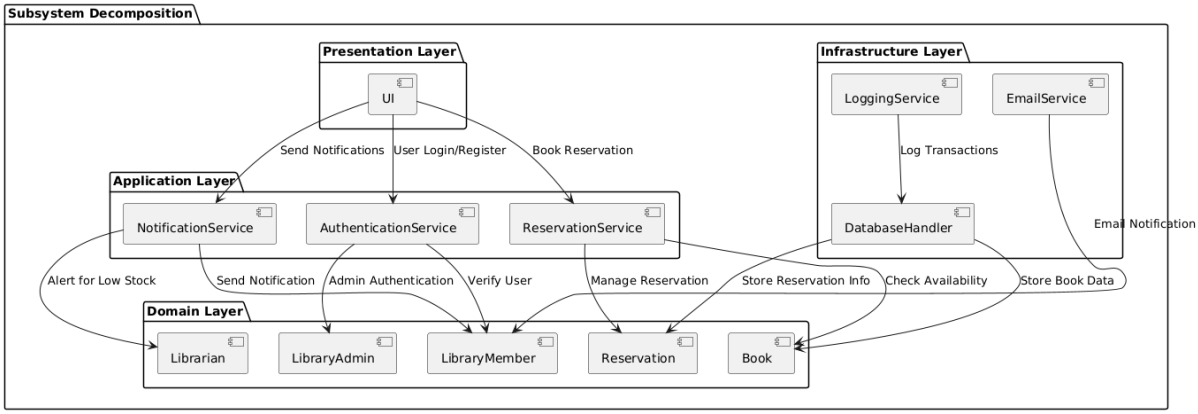

Figure 1 Onion Architecture System Design

Description and Responsibilities

The system is designed using the **Onion Architecture**, a modular and layered architecture that ensures separation of concerns, testability, and maintainability. Below are the identified subsystems and their responsibilities:

1. **Presentation Layer (UI)**
    - Responsible for interacting with the user (librarians and members).
    - Provides forms for search, reservation, and inventory management.
    - Relays user inputs to the application core.
2. **Application Layer (Service Layer)**
    - Coordinates between the UI and domain layers.
    - Manages application-specific logic, such as user authorization and system notifications.
3. **Domain Layer**
    - Represents the core business logic and entities, such as Book, Reservation, and LibraryMember.
    - Enforces business rules and constraints.
    -

4. **Infrastructure Layer**
    - Handles data persistence and external systems integration.
    - Contains repositories for database operations and services for external APIs.

Architectural Pattern Rationale

The Onion Architecture was chosen because:

- It enforces a **clear separation of concerns** with its layered approach.
- It prioritizes the **domain layer**, ensuring the business logic remains independent of external dependencies.
- It supports **testability** by isolating business logic and allowing external systems (e.g., databases) to be mocked during tests.
- It aligns with the **design goals** of modularity, scalability, and maintainability.

3.2 Hardware/software mapping

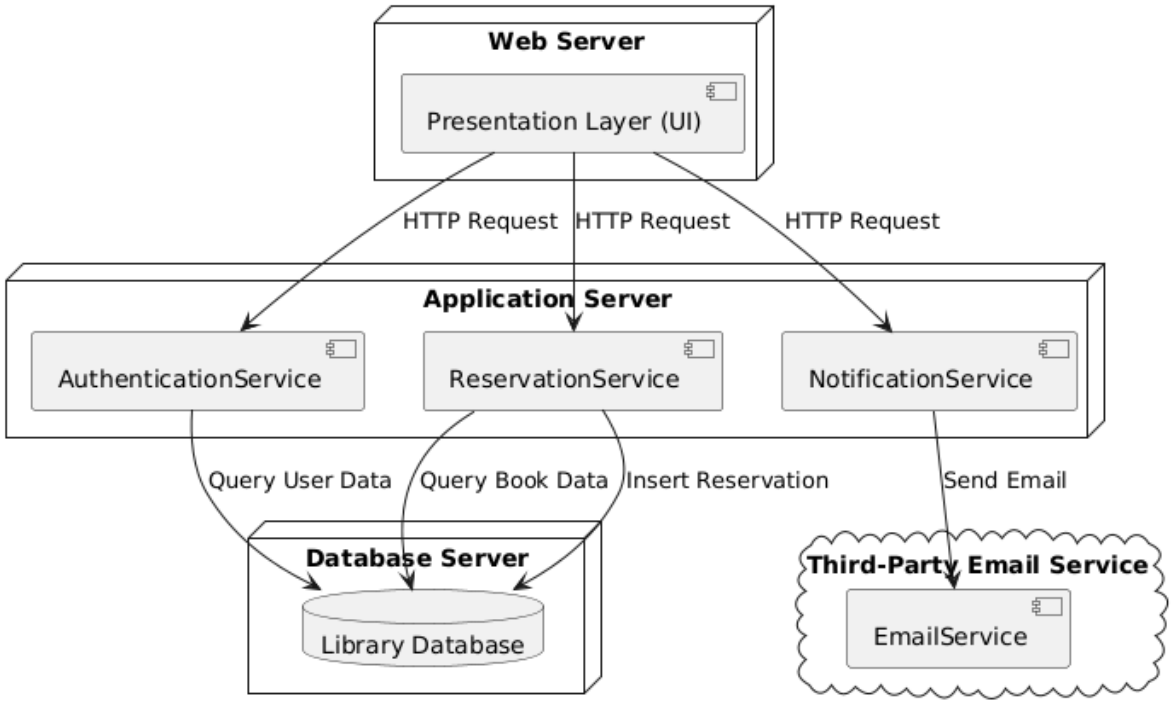

Figure 2 Hardware/Software Mapping

The **Hardware/Software Mapping** section outlines the distribution of software components to hardware infrastructure. This mapping ensures that the system's components are properly

allocated to achieve efficiency, scalability, and reliability. Below is the mapping of our system:

4 Hardware Components

1. **Web Server**: Hosts the system's web application and handles user interactions.
2. **Application Server**: Executes the core business logic and processes user requests.
3. **Database Server**: Manages the persistence of system data, including user records, books, and reservations.

5 Software Components

1. **Presentation Layer (UI)**: Deployed on the **Web Server**, allowing users to access the system via a browser.
2. **Application Services**: Deployed on the **Application Server**, handling authentication, reservations, and notifications.
3. **Database**: Deployed on the **Database Server**, storing data related to users, books, and reservations.
4. **Email Service**: Utilized as a third-party service for sending notifications.

5.1 Persistent data management

The **Persistent Data Management** section outlines how the system handles and stores persistent data, the database infrastructure chosen for the system, and how the database is encapsulated to ensure modularity and maintainability.

- **Database Description**

Our system uses a **PostgreSQL** database to store and manage persistent data. This database was selected due to its scalability, support for complex queries, and strong data integrity features. The database ensures that all critical information, such as user details, book inventories, and reservation records, is securely stored and easily accessible.

- **Data Schemes**

The following are the key entities stored in the database, along with their respective attributes:

1. **LibraryMember**
    - MemberID (Primary Key)

- Name
- Email
- PhoneNumber
- Address

2. **Librarian**
    - LibrarianID (Primary Key)
    - Name
    - Email
    - Role
3. **LibraryAdmin**
    - AdminID (Primary Key)
    - Name
    - Email
    - Role
4. **Book**
    - BookID (Primary Key)
    - Title
    - Author
    - Genre
    - AvailabilityStatus
5. **Reservation**
    - ReservationID (Primary Key)
    - MemberID (Foreign Key)
    - BookID (Foreign Key)
    - ReservationDate
    - ExpiryDate

- **Database Encapsulation**

The database is encapsulated within the **Data Access Layer (DAL)** of the application. This layer abstracts direct database interactions and ensures that:

1. SQL queries are executed efficiently.
2. Any changes to the database schema require minimal changes to the application logic.
3. Only authorized services can access the database, ensuring security and preventing unauthorized modifications.

The **Data Access Layer** employs an Object-Relational Mapping (ORM) tool, such as **Entity Framework**, to facilitate interactions between the application and the database. This approach ensures:

- Improved maintainability and scalability.
- Reduction in boilerplate code for database queries.
- Strict type checking, reducing runtime errors.

- **Summary**

The **PostgreSQL** database, combined with the encapsulation provided by the **Data Access Layer**, ensures efficient, secure, and scalable management of persistent data. This infrastructure allows the system to handle current data requirements while being flexible enough to accommodate future needs.

5.2 Access control and security

The access control and security model for our system ensures that the right users have access to the correct resources and actions, while maintaining the confidentiality, integrity, and availability of the system. Below are the key elements of the model:

- **Access Matrix**

The following access matrix defines the permissions for different user roles within the system:

Resource	Library Member	Librarian	Library Admin
View Books	Read	Read	Read
Reserve Books	Create, Update	None	None
Manage Reservations	None	Read, Update	Read, Update
Manage Inventory	None	Read, Update	Create, Read, Update, Delete
Manage Users	None	None	Create, Read, Update, Delete

- **Authentication Mechanism**
- The system uses **JWT (JSON Web Tokens)** for user authentication.
    - When a user logs in, the system generates a secure token, which must be included in all subsequent API requests to validate the user.
    - The tokens are signed with a strong private key to ensure they cannot be tampered with.
- **Encryption**
- **Data in Transit:** All communication between clients and the server is secured using **HTTPS (TLS encryption)**.
- **Data at Rest:** Sensitive information, such as user passwords, is hashed using a secure algorithm like **bcrypt**. Other critical data is stored in the database with **AES-256 encryption**.
- **Key Management**
- The system securely manages encryption keys using a **Key Management Service (KMS)**. Keys are rotated periodically to enhance security.
- Access to keys is restricted to authorized services and personnel only.
- **Security Considerations**

1. **Role-based Access Control (RBAC):** The system implements RBAC to ensure each user can only perform actions aligned with their role.
2. **Audit Logs:** All security-sensitive operations, such as login attempts and modifications to sensitive data, are logged for auditing purposes.

3. **Session Expiration:** JWT tokens have a fixed expiration time to reduce the risk of token misuse.
4. **Protection Against Common Attacks:**
    - SQL Injection: All database queries use parameterized statements.
    - Cross-Site Scripting (XSS): User inputs are sanitized before rendering.
    - Cross-Site Request Forgery (CSRF): CSRF protection mechanisms are in place for sensitive operations.

This framework ensures that the system is robust and meets modern security standards.

5.3 Boundary conditions

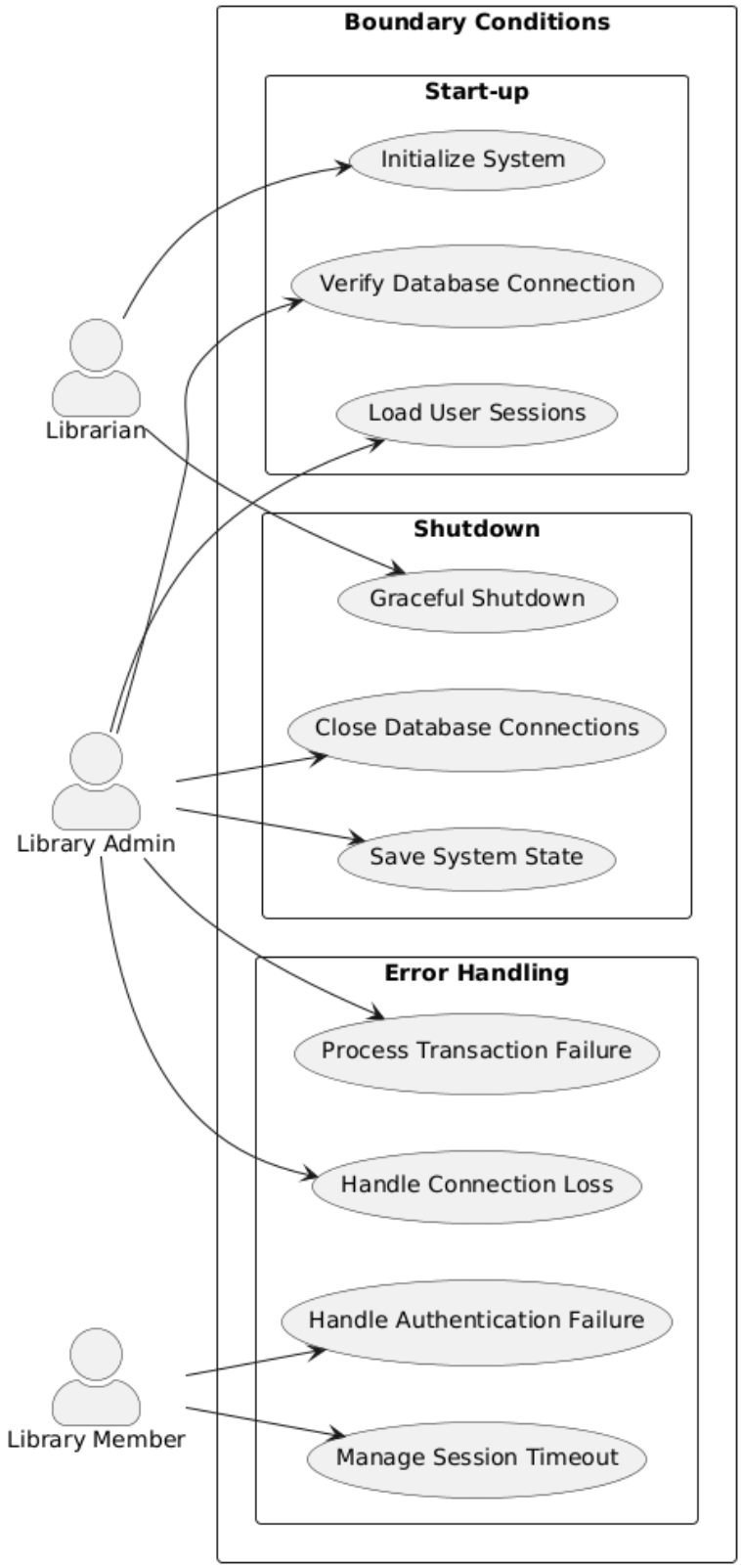

Figure 3 Boundary Conditions

This section describes the system's behavior during start-up, shutdown, and error conditions. The following use cases detail these boundary scenarios.

Start-up Conditions

- **Initialize System**: System performs database connection checks, loads configuration settings, and initializes all necessary services.
- **Load User Sessions**: Restores any interrupted user sessions and cleans up expired sessions.
- **Verify Database Connection**: Ensures stable connection to the database before allowing any operations.

Shutdown Conditions

- **Graceful Shutdown**: System saves all current states, completes pending transactions, and notifies active users.
- **Save System State**: Preserves current system state for recovery during next startup.
- **Close Database Connections**: Properly closes all database connections to prevent data corruption.

Error Conditions

- **Handle Connection Loss**: Manages database or network connection failures with proper error recovery.
- **Manage Session Timeout**: Handles expired user sessions with appropriate cleanup and user notification.
- **Process Transaction Failure**: Manages failed transactions with rollback mechanisms and error logging.
- **Handle Authentication Failure**: Manages failed login attempts and implements security measures.

6 Subsystem Services

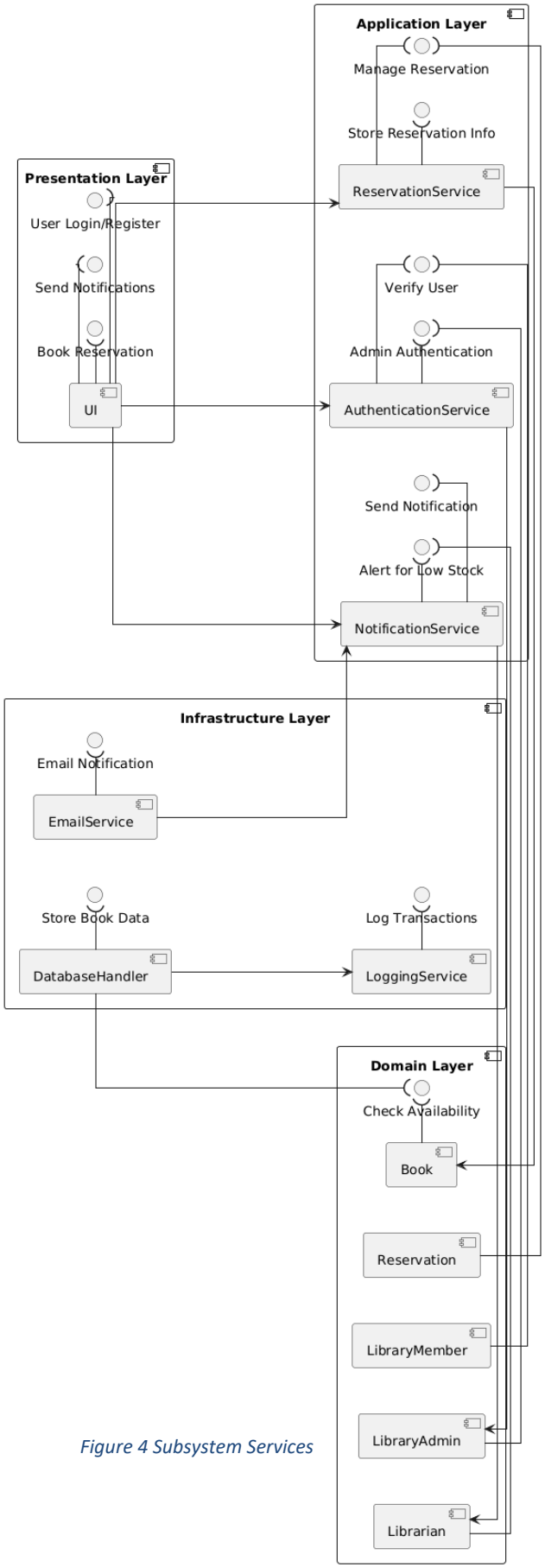

Figure 4 Subsystem Services

The UML component diagram illustrates the services provided by each subsystem using lollipop notation. Each layer and its services are detailed below:

Presentation Layer

The UI component provides the following services:

- **User Login/Register**: Handles user authentication and new user registration processes
- **Send Notifications**: Manages the user interface for sending notifications to users
- **Book Reservation**: Provides the interface for users to make book reservations

Application Layer

This layer contains three main service components:

- **NotificationService**
- **Send Notification**: Manages the delivery of notifications to users
- **Alert for Low Stock**: Handles alerts when book quantities are running low
- **AuthenticationService**
- **Verify User**: Validates user credentials and maintains session management
- **Admin Authentication**: Handles specialized authentication for library administrators
- **ReservationService**
- **Manage Reservation**: Coordinates the book reservation process
- **Store Reservation Info**: Handles the persistence of reservation data

Domain Layer

Contains the core business entities and their services:

- **Book**: Provides **Check Availability** service for verifying book status
- **Reservation**: Implements reservation business rules
- **LibraryMember**: Manages member-specific operations
- **LibraryAdmin**: Handles administrative functions
- **Librarian**: Manages librarian-specific operations including stock management

Infrastructure Layer

Provides fundamental technical services:

- **DatabaseHandler**
- **Store Book Data**: Manages persistent storage of book information
- **Check Availability**: Provides database-level availability checks
- **EmailService**
- **Email Notification**: Handles email delivery functionality
- **LoggingService**
- **Log Transactions**: Manages system-wide transaction logging
- **Service Interactions**

The diagram shows the dependencies between services through connecting lines. Key interactions include:

- UI components depend on Application Layer services for processing user requests
- Application Layer services utilize Domain Layer components for business logic
- Infrastructure Layer provides supporting services to all upper layers
- Email notifications flow through multiple layers to reach users

This service architecture ensures clear boundaries between subsystems while maintaining efficient communication paths between components. Each service has a well-defined responsibility, promoting maintainability and scalability of the system.

7 Glossary

1. **Actor**: An entity that interacts with the system, either as a user or another system, to achieve a specific goal.
2. **API (Application Programming Interface)**: A set of rules and protocols for building and interacting with software applications, enabling communication between different systems or components.
3. **Boundary Condition**: Specific scenarios or edge cases in a system that define its limits or constraints, often tested to ensure robustness.
4. **Class Diagram**: A type of UML diagram that represents the static structure of a system, including its classes, attributes, methods, and relationships.
5. **Database Management System (DBMS)**: Software used to define, create, maintain, and control access to a database.
6. **Entity**: A distinct object or concept in a database that can be uniquely identified and has attributes associated with it.
7. **Framework**: A pre-built collection of code that provides a foundation for developing software applications, often including libraries, tools, and conventions.
8. **Object-Oriented Analysis and Design (OOAD)**: A methodology for analyzing and designing a system using object-oriented concepts like classes and objects.
9. **PlantUML**: A tool for creating UML diagrams from simple textual descriptions.
10. **Primary Key**: A unique identifier for a record in a database table.
11. **Sequence Diagram**: A UML diagram that shows the interaction between system components over time, often used to represent the flow of messages.
12. **Stakeholder**: Any individual or organization that has an interest in or is affected by the system being developed.
13. **System Boundary**: The line that separates the system from its external environment, defining the scope of the system.
14. **Use Case Diagram**: A UML diagram that shows the interactions between actors and the system to achieve specific goals.
15. **User Interface (UI)**: The part of the system through which users interact with the software, including screens, buttons, and input fields.
16. **UML (Unified Modeling Language)**: A standardized modeling language used to visualize, specify, construct, and document the artifacts of a system.

8 References

https://www.gencayyildiz.com/blog/nedir-bu-onion-architecture-tam-teferruatli-inceleyelim/

9 Appendix

- Annex – I: Distribution of Work
- Annex – II: Meeting Minutes

Distribution of Work

Berke Alpaslan (Team Leader)

- Responsible for project coordination and overseeing the overall organization of the document. Worked on the Glossary section and ensured proper formatting of the report.

Gül Yeşilgil (Subsystem Designer)

- Designed the subsystem decomposition and created UML diagrams, including those with lollipop notation. Wrote and refined the PlantUML code.

Mert Doğan Aygün (Boundary Specialist)

- Defined the boundary conditions, including initialization, closure, and error scenarios. Wrote use-case scenarios and created the boundary diagrams using UML.

Birhat Taş (Document Editor)

- Reviewed the written sections of the report, corrected English grammar and syntax errors, and finalized the document layout.

Tural Mammadov (Service Analyst)

- Prepared the explanations for the Subsystem Services section and identified dependencies between the subsystems.

Aziz Berk Yıldırım (Error Handling Expert)

- Analyzed system errors and proposed solutions. Detailed error-handling processes in the boundary conditions section and reviewed the UML diagrams for accuracy.

Annex – II

Meeting Minutes

Date:	28/12/2024
Location:	Microsoft Teams
Duration:	180 min.
Participants:	Berke Alpaslan, Gül Yeşilgil, Mert Doğan Aygün, Birhat Taş, Tural Mammadov, Aziz Berk Yıldırım
Content of the meeting (briefly explain the agenda, decisions, work distributions, etc.)	
At the beginning, we planned the project and, during the subsequent stages of the meeting, we prepared our report in alignment with this plan.	