Silent Santa

Software Design

Student: Andronescu Raluca

Group : 30431

Technical University of Cluj-Napoca

Contents

Chapter 1

Deliverable 1

1.1 Project Specification

1.1.1 Overview

Silent Santa is a web platform that connects donors with needy children through heartfelt letters posted by school administrators (admins). The platform features two primary user roles:

- **Admin:** Can post christmas letters from needy children, interact with users interested in these letters, review and accept/deny user requests, and perform all actions available to a standard user.
- **User:** Can view all posted letters, add letters to favorites, request to help prepare a gift package for a child (which then appears in a personal "My Letters" to-do list), interact with the admin, and notify when a package is ready to be sent.

1.1.2 Goals

- Facilitate anonymous and meaningful gift exchanges for children in need.
- Provide an intuitive and secure interface for both admins and users.
- Enhance the impact of donations through organized tracking of gift preparation and delivery.

1.2 Functional Requirements

1.2.1 User Authentication and Management

- **Registration and Login:**
  - Secure registration and login for both users and admins.
  - Role-based access control to ensure that admins have additional privileges.
- **Profile Management:**
  - Users can view and edit their profiles.
  - Admin profiles include management options for letters and user requests.

1.2.2 Letter Management

– **Admin Functions:**

- **Post Letters:** Admins can create new letters with details such as the child's needs, wish list items, and any relevant instructions or details.
- **Edit/Delete Letters:** Admins can update or remove letters as necessary.
- **Interaction:** Admins can respond to user inquiries related to a letter.

– **User Functions:**

- **View Letters:** Users can browse all posted letters, each showing a summary of the child's needs.
- **Add to Favorites:** Users can mark letters as favorites for easy access.

1.2.3 Request and Task Management

– **User Requests:**

- Users can request to prepare a gift for a child by selecting a letter.
- Requested letters are added to the user's "My Letters" section, which acts as a to-do list for assembling the gift package.
- Users can mark checklist items as completed as they acquire each item.

– **Admin Request Processing:**

- Admins are notified of new requests for a specific letter.
- Admins can accept or deny a request, and provide feedback if necessary.
- Admins can see user status regarding the package.

– **Status Notifications:**

- Notifications are sent to users when their request status changes (e.g., approved, denied).
- Users can update the status of their gift package (e.g., "in progress," "ready to send"), which triggers additional notifications.

1.2.4 Communication and Interaction

– **Messaging System:**

- Internal messaging allows users and admins to communicate regarding letters or specific requests.

– **Feedback and Comments:**

- Users can comment on letters or ask questions, to which admins can respond directly.

1.2.5 Notification and Reminder System

– **Real-Time Alerts:**

  • Email and in-app notifications inform users of important status changes and deadlines.

– **Automated Reminders:**

  • Reminders prompt users to update their gift package progress and complete tasks in their to-do list.

1.2.6 Admin Dashboard and Analytics

– **Dashboard Overview:**

  • Admins have access to a dashboard showing active letters, pending user requests, and overall progress on gift packages.

1.3 Use Case Model

1.3.1 Use Cases Identification

- **User:**
  - *Login / Signup:* Access the platform via a valid account.
  - *View Letters / Add to Favorites:* Browse children's letters and mark favorites.
  - *Request Letter:* Commit to fulfilling a child's wish.
  - *Message Admin:* Communicate directly with an Admin for clarifications.
  - *View & Edit MyLetters List:* Keep track of letters the user has requested.
  - *View Favorites:* Quickly access previously marked letters.
  - *View Account / Logout:* Manage account details or sign out.
- **Admin:**
  - *Add / Update / Delete Letter:* Maintain the collection of available letters for donors.
  - *Accept / Deny Request:* Approve or reject a user's request to fulfill a letter.
  - *View Status:* Track the progress of gifts and see which items are completed.

1.4 Supplementary Specification

1.4.1 Non-functional Requirements

Non-functional requirements define the system's operational attributes and constraints that must be satisfied, including:

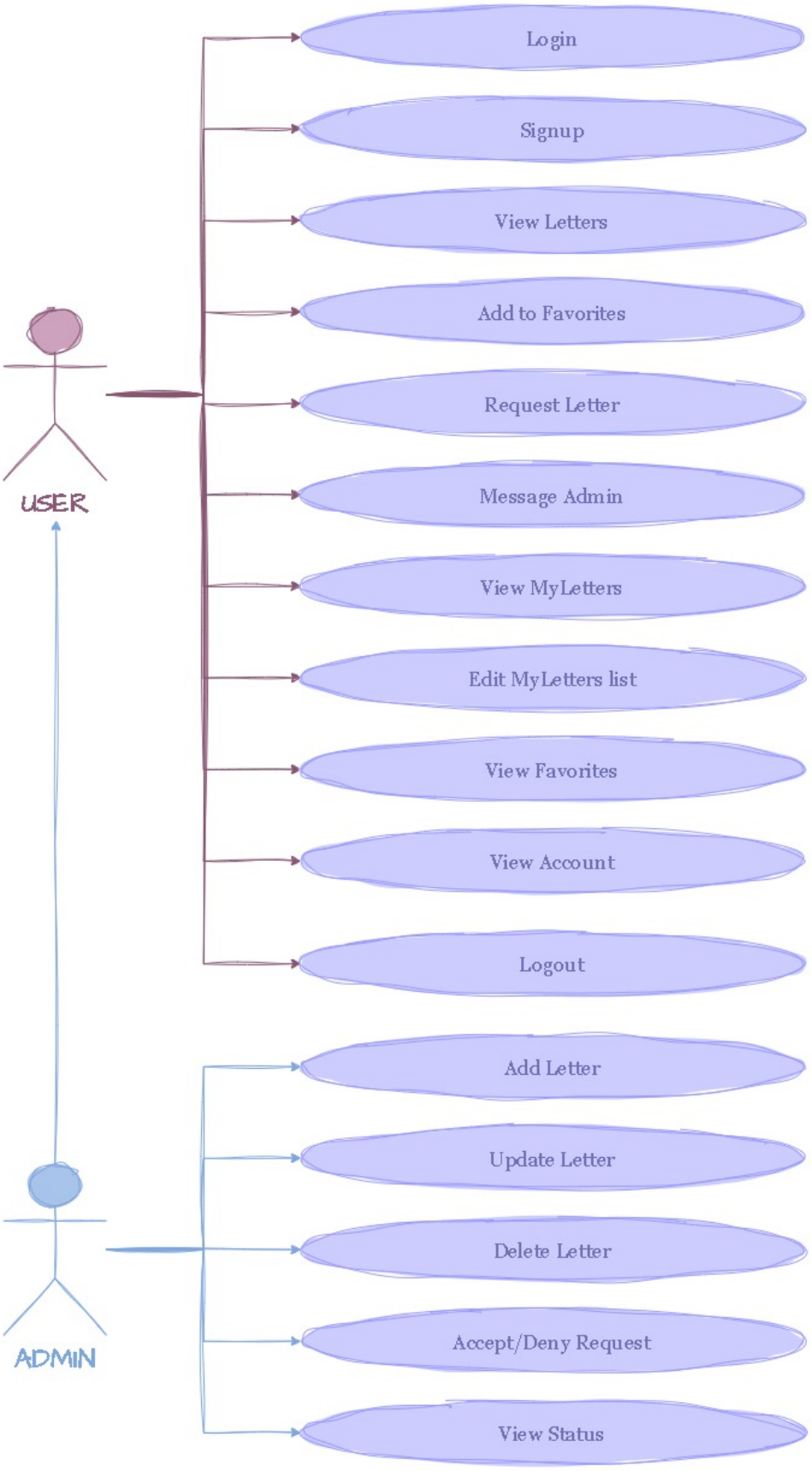

Figure 1.1: Use Case

- **Scalability:** The system must be scalable to accommodate future growth in user base, allowing for additional resources (e.g., servers, databases) to be added as needed.
- **Security:** The system must implement robust security measures, including data encryption, secure authentication, and protection from common vulnerabilities (e.g., SQL injection, cross-site scripting).
- **Usability:** The system must have a user-friendly interface that can be navigated easily by users with minimal technical knowledge.
- **Maintainability:** The system's architecture and codebase should be designed for ease of maintenance, with clear documentation and modular components.

1.4.2 Design Constraints

Design constraints impose restrictions on the system's architecture, functionality, and implementation. These constraints could be:

- **Technological Constraints:** Angular for the frontend, Spring Boot for the backend.
- **Regulatory Compliance:** The system must comply with relevant legal and regulatory standards, such as GDPR for data protection and privacy.
- **Time Constraints:** The system must be completed within a predefined timeframe, such as a set number of months or before a specific deadline.
- **Platform Constraints:** The system must be compatible with the most common operating systems and devices, ensuring cross-platform functionality.
- **Resource Constraints:** The system must be developed with a limited number of developers and computing resources, necessitating efficient coding practices and design choices.

1.5 Glossary

- Admin: A privileged user role with permissions to create, edit, or remove letters and oversee user requests.
- Letter: A post describing a child's needs or wishes. Typically includes personal info or a wishlist.
- User: A donor who can browse letters, request to fulfill a need, and communicate with the admin.
- Request: A user's action to commit to fulfill a specific letter. Moves the letter into that user's "My Letters" section.
- My Letters: A user's personal dashboard showing the letters they have committed to and their progress.

- Notification: An automated alert triggered by status changes, approvals, or other events (via email or in-app).
- Task Checklist: A mini to-do list under each requested letter that helps the user track gift preparation steps.

Chapter 2

Deliverable 2

2.1 Domain Model

The domain model represents the core entities of the system and their relationships. The primary entities in the Silent Santa system include `UsersModel`, `FavoritesModel`, `LettersModel`, `RequestsModel`, and `SubscriberModel`. These entities work together to facilitate letter management, requests from users, and communication between users and admins.

2.1.1 Conceptual Class Diagram

Below is the conceptual class diagram that illustrates the key classes and their relationships in the Silent Santa system.

2.2 Architectural Design

The Silent Santa system follows a layered architecture that organizes the components into different layers for maintainability and scalability.

2.2.1 Architecture Overview

The system consists of the following components:

- **Frontend (Angular):** The user interface is built using Angular. It enables users to interact with the platform, view letters, make requests, and communicate with admins. The frontend communicates with the backend through RESTful API calls.
- **Backend (Spring Boot):** The backend is implemented using Spring Boot. It handles core functionalities like user authentication, letter management, request processing, and email notifications.
- **Database (MySQL):** The database stores user data, letters, requests, and other relevant information. It is optimized for quick retrieval and efficient management of the system's data.
- **Email Service:** The email service handles sending notifications to users about new letters, updates, and other reminders.

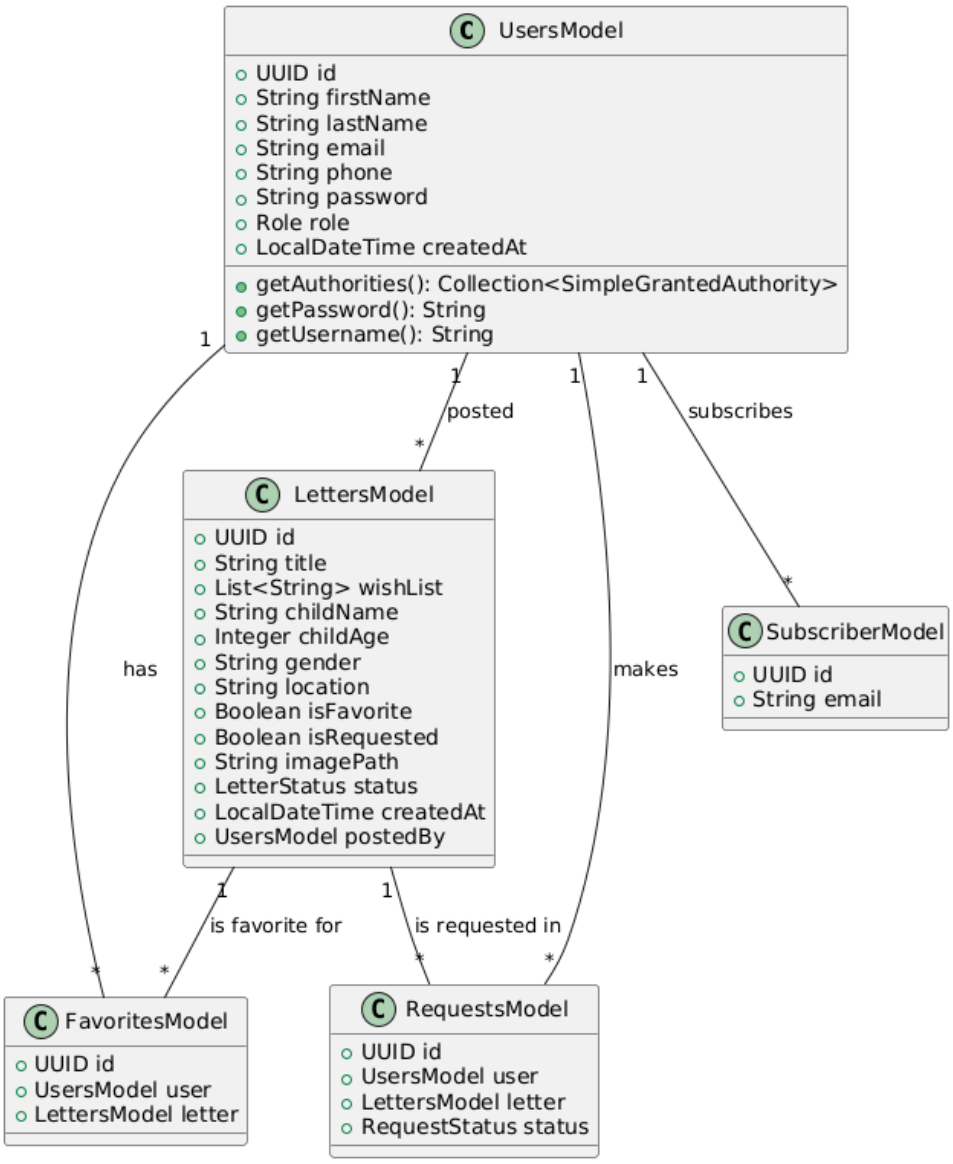

Figure 2.1: Conceptual Class Diagram

2.2.2 Architecture Layers

The system is organized into four main architectural layers:

- **Presentation Layer:** The Angular application serves as the front-end, responsible for handling all user interactions.
- **Business Logic Layer:** The Spring Boot backend implements core business logic such as letter management, user authentication, and request handling.
- **Persistence Layer:** The MySQL database stores all persistent data related to users, letters, requests, and other entities.
- **Communication Layer:** This layer facilitates communication between the front-end, backend, and external services, such as email notifications and API integrations.

2.2.3 Conceptual Architecture

The system's architecture consists of four primary components:

- **Frontend (Angular):** The user interface that allows users to interact with the system, browse letters, make requests, and communicate with admins.
- **Backend (Spring Boot):** The core server-side logic, exposing REST APIs for the frontend to consume and implementing core features like user management, letter creation, and request handling.
- **Database (MySQL):** The database where persistent data is stored, including user profiles, letters, and request statuses.
- **Email Service:** A service that handles sending out notifications to users regarding new letters, status updates, and reminders.

2.2.4 Flow of Data

The flow of data through the system follows these steps:

The user interacts with the frontend (Angular) to browse letters or submit a request.

The frontend sends a request to the backend (Spring Boot) via RESTful APIs.

The backend processes the request, interacts with the database, and sends out necessary email notifications.

The backend sends the response back to the frontend, which displays the updated data to the user.

2.2.5 Package Design

The system is organized into the following packages to maintain modularity:

- **com.group.silent**$_s anta.model : Containstheentityclassesthatdefinethecoredatamodel, includingUse$

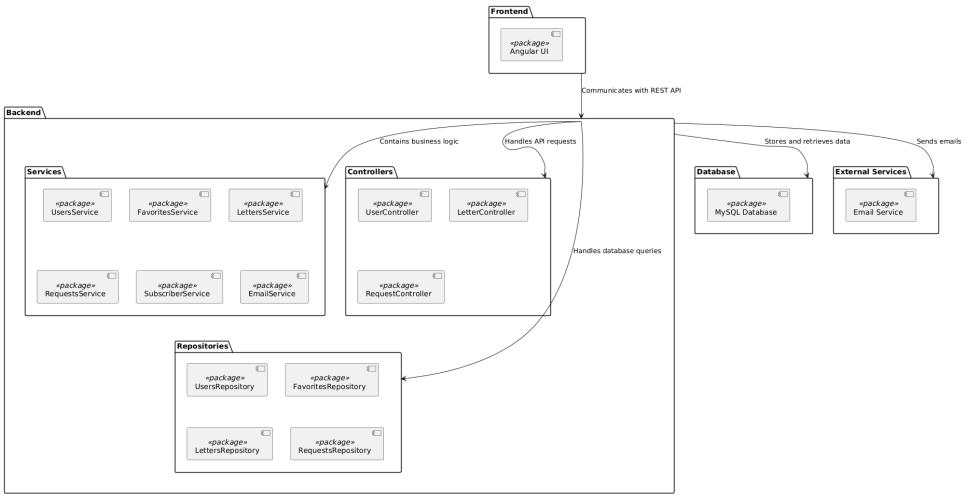

Figure 2.2: Package Diagram

2.2.6 Component and Deployment Diagrams

The system consists of several components, each deployed on different servers, to ensure modularity and separation of concerns:

- **Frontend Component:** This component is deployed on a web server and serves the user interface.
- **Backend Component:** The Spring Boot API, deployed on a web server, handles business logic and API requests.
- **Database Component:** The MySQL database, deployed on a separate server, stores all persistent data.
- **Email Service Component:** The email service, deployed on an email server, sends notifications to users.

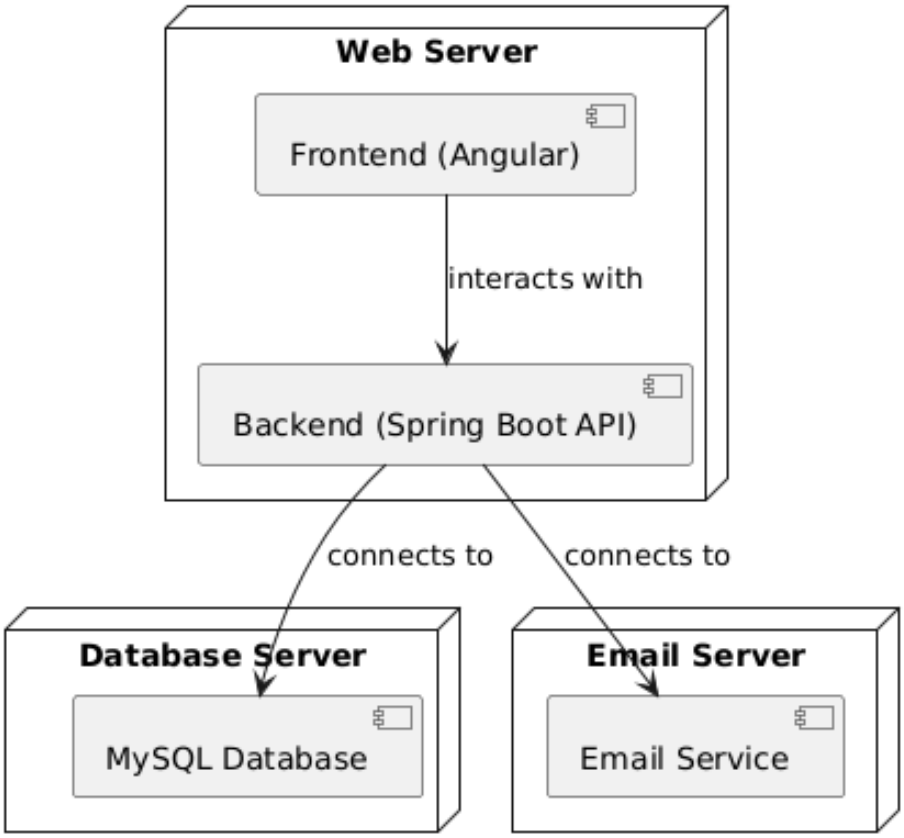

Figure 2.3: Deployment Diagram

Diagram Descriptions: Package Diagram (pk.png): This diagram visualizes how the system is divided into packages. It shows the relationships between different packages such as model, repository, service, controller, and view. Each package contains components that handle specific responsibilities within the system.

Deployment Diagram (dp.png): The deployment diagram shows how the system is deployed across different servers. It visualizes the physical infrastructure, including the

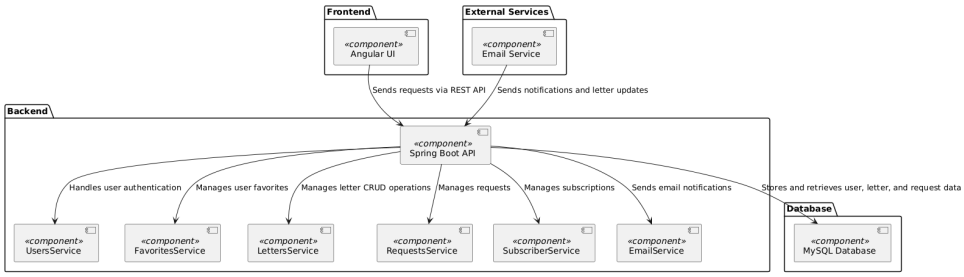

Figure 2.4: Component Diagram

web server, database server, and email service, and how the components interact across these servers.

Component Diagram (cp.png): The component diagram shows the different components within the backend and their relationships. It illustrates how the frontend interacts with the backend through REST APIs, and how the backend communicates with the database and email services.

Chapter 3

Deliverable 3

3.1 Design Model

3.1.1 Dynamic Behavior

For the following two key scenarios, we provide interaction diagrams: one sequence diagram and one communication diagram.

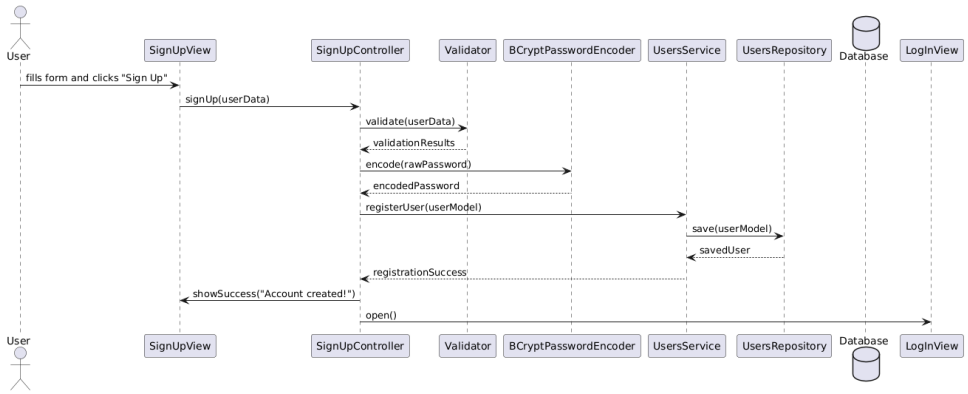

Figure 3.1: Sequence Diagram for the "User Request Letter" Scenario

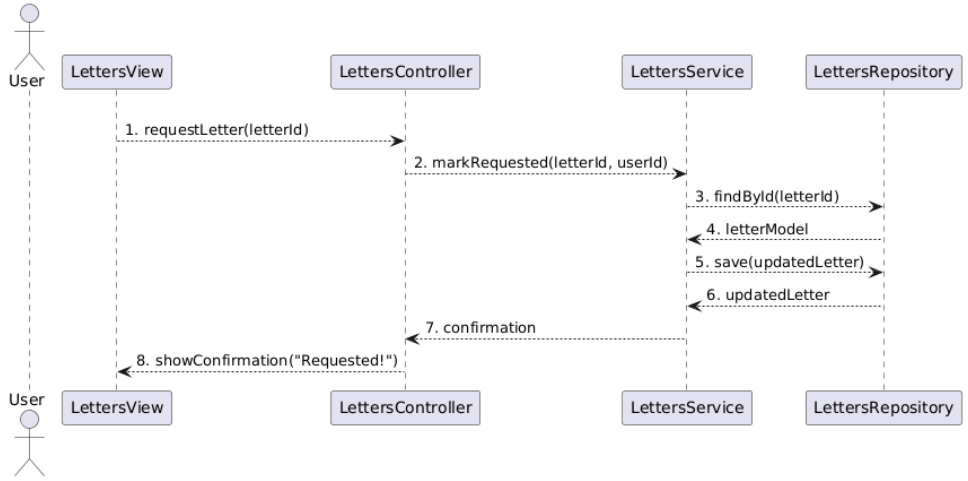

Figure 3.2: Communication Diagram for the "User Sign-Up" Scenario

3.1.2 Applied GoF Patterns

- Observer We decouple request processing from notifications by having `RequestModel` act as subject and `NotificationListener` implementations (e.g. in-app, email) register to receive updates. Adding new channels requires no changes to the core model.
- Factory Method `LettersService` relies on a `LetterFactory` interface to create `Letter` subclasses (e.g. `WishlistLetter`, `NeedsLetter`). Controllers depend only on the factory, so new letter types can be added without modifying existing code.
- Strategy Password encoding and validation rules are injected via `PasswordEncoder` and `ValidationStrategy` interfaces. Swapping implementations (e.g. BCrypt vs SCrypt, admin vs user rules) is done through Spring configuration, not code changes.

3.1.3 Class Diagram

The following UML class diagram applies relevant GoF design patterns (e.g., `Observer` for notifications, `Factory` for creating letter and user objects). Pattern choices are motivated in the accompanying text.

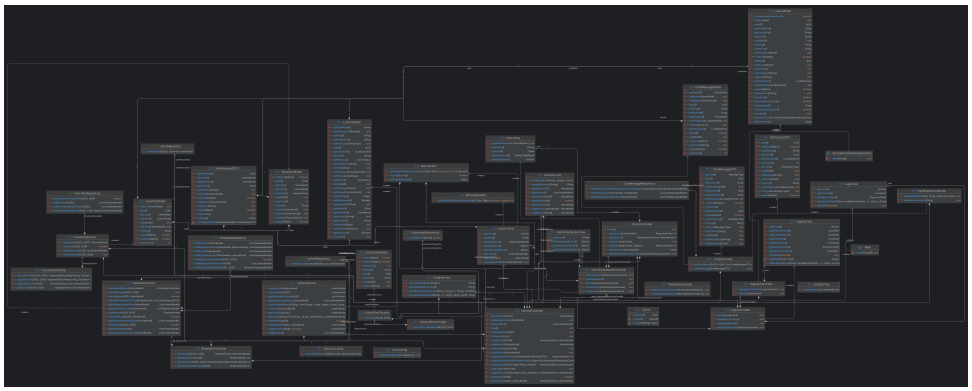

Figure 3.3: UML Class Diagram with GoF Patterns

3.2 Data Model

The system's logical data model defines the core entities and how they relate—Users, Letters, Requests, Favorites, and Notifications—and ensures data integrity and clear navigation between them. See Figure 3.4 for the full model.

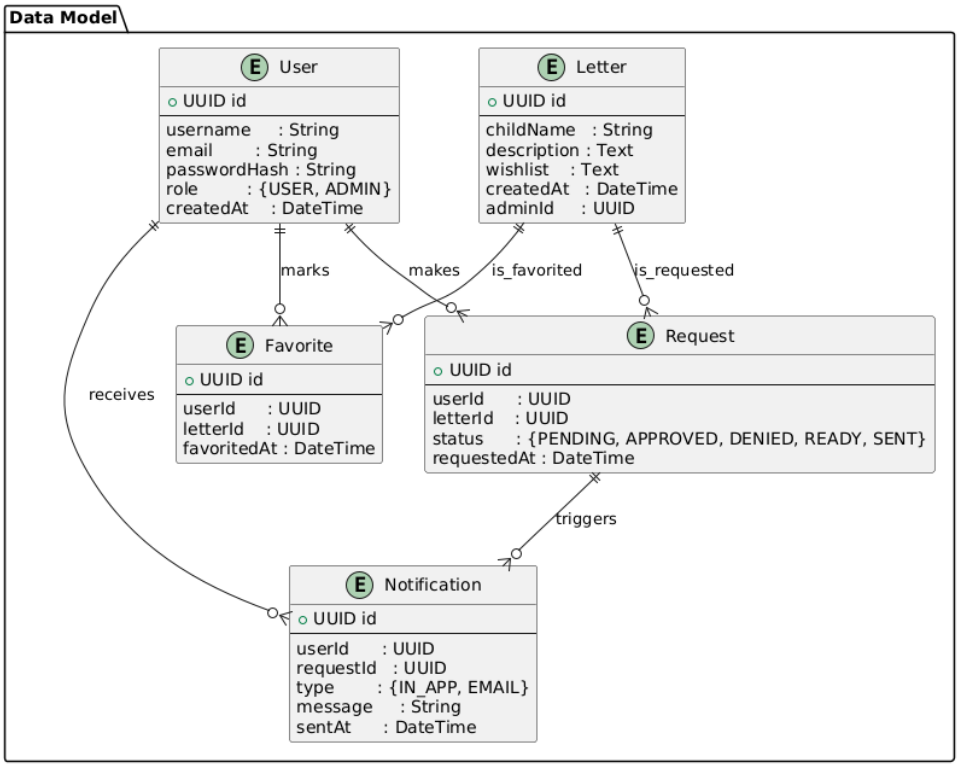

Figure 3.4: Logical Data Model

3.3 System Testing

We employ a mix of unit, API, integration, and acceptance testing to ensure system reliability:

– **API Tests:** Test all REST endpoints with Postman collections to validate request/response workflows.

– **Integration Tests:** Exercise end-to-end flows (e.g. sign-up → request letter → notification) with Spring Boot Test and an in-memory H2 database.

subsection*Sample Test Cases

1. **Valid Sign-Up:**
   – *Input:* new user data with valid email and password.
   – *Expected:* HTTP 201, user persisted, welcome email sent.
2. **Duplicate Email Sign-Up:**
   – *Input:* registration with an existing email.
   – *Expected:* HTTP 400 with “Email already in use” error.
3. **Request Approval Notification:**
   – *Setup:* pending request in database.
   – *Action:* admin approves request.
   – *Expected:* request status = `APPROVED`, in-app and email notifications dispatched.

3.4 Future Improvements

Potential enhancements to extend the platform's impact:

– **Donation Tracking:** Add payment gateway and order tracking for gift purchases.

– **Recommendation Engine:** Suggest letters to users based on past activity and preferences.

– **Mobile App:** Develop native iOS/Android clients for on-the-go participation.

3.5 Conclusion

Silent Santa delivers a secure, scalable solution for anonymous gift exchanges. Our design balances clear separation of concerns with extensibility via GoF patterns. Rigorous testing ensures reliability, and planned features like real-time chat and donation tracking will further enhance user engagement and system value.