CS673 Software Engineering
Scrumble Bug - Campus Exchange
Software Design Document

Team Member	Role(s)	Signature	Date
Yingtong Zhou	Team leader	*Yingtong Zhou*	Oct 17, 2024
HungHsu(Allen) Chen	Configuration Leader	*HungHsu(Allen) Chen*	10/17/2024
Yuanbin Man	Requirement Leader	*Yuanbin Man*	10/17/2024
Ang Li	Design and Implementation Leader	*Ang LI*	10/17/2024
Yueyihan Qi	Security Leader	*Yueyihan Qi*	10/17/2024
Srujana N	QA Leader	*Srujana N*	10/17/2024

Revision history

Version	**Author**	**Date**	**Change**

Introduction - Yingtong

The secondhand trading app - Campus Exchange - is designed to facilitate easy and secure exchanges of goods between students, faculty, and staff within the campus community. By providing a streamlined platform for listing, browsing, and purchasing second hand items, the app will enhance the experience of campus-based trading, ensuring trust, convenience, and efficiency.

This document outlines the system's overall structure, including its user interface, business logic, database design, and security measures. The design goals of our software system is to ensure that the application is built with scalability, security, and user needs in mind.

Software Architecture - Yuanbin, Allen

Campus Exchange Software Architecture.drawio

CS673 Software Eng Project doc..pdf

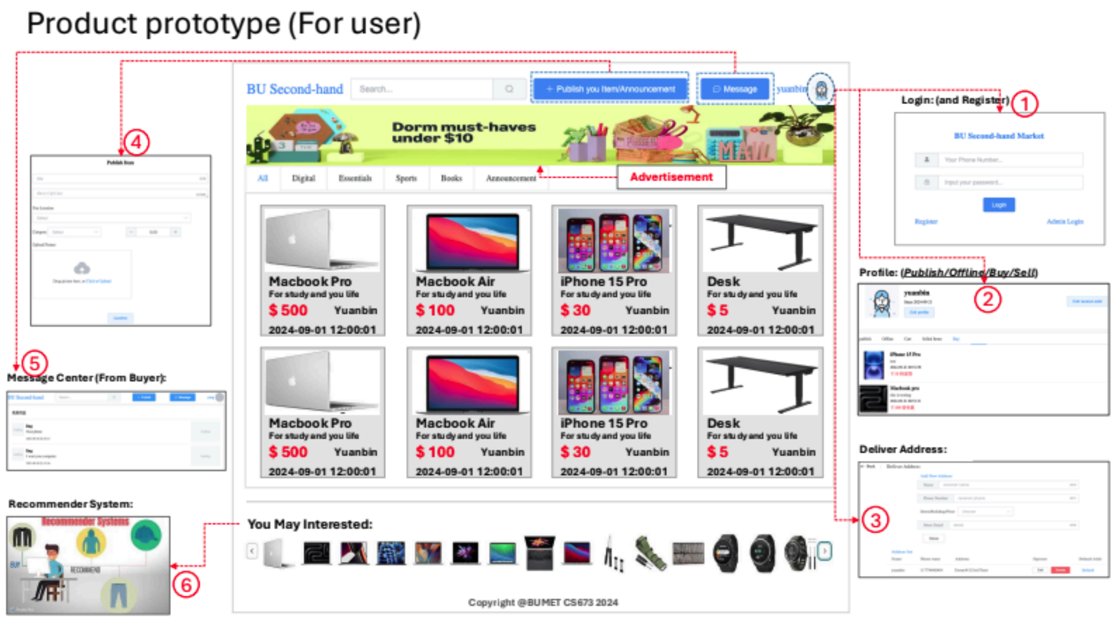
We constructed a RESTful API on our backend and accessed the data in the database through the mapper objects. Objects on the domain are used to represent object entities and the example objects generated by Mybatis represent the query that is used for querying the database. Then a model is based on the objects on the domain for returning data in a preferred format to the front end. Services are paired with interface and implementation. Service provides functionality to manipulate data through domain objects and mappers. Controllers provide the interface for the front end with a RESTful

API architecture. Vo is similar to the model where it contains data in an organized fashion for ease of utilization. Util contains special classes that are utilized for special purposes. Other packages like enums, exceptions, and handlers are used as a standard when generating errors and exceptions.
The application's front end interacts with the data through the API and presents the data to the users.

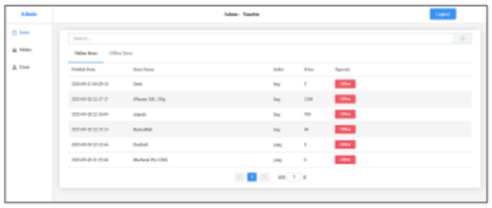
- ## Class Diagram - All

  In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.
  [Campus Exchange UML.drawio]

- ## UI Design (if applicable) - Yingtong

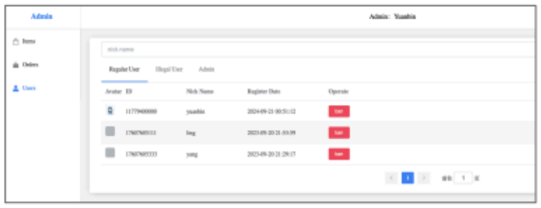


Product prototype (For administrator)



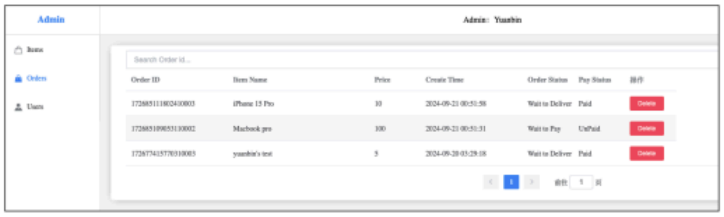
Publish Date	Item Name	Seller	Price	Operate
[illegible]	Desk	[illegible]	1	Offline
[illegible]	iPhone XR, 128g	[illegible]	1200	Offline
[illegible]	airpods	[illegible]	700	Offline
[illegible]	Basketball	[illegible]	80	Offline
[illegible]	Football	yang	5	Offline
[illegible]	Macbook Pro 13RG	yang	6	Offline

(a) Manage Items (Online/Offline)



Avatar	ID	Nick Name	Register Date	Operate
	11779400000	yuanbin	2024-09-21 00:51:12	Ban
	17607605111	ling	2023-09-20 21:03:09	Ban
	17607605333	yang	2023-09-20 21:29:17	Ban

(b) Manage Users (Ban or Grant/Revoke Access)



Order ID	Item Name	Price	Create Time	Order Status	Pay Status	操作
1726851118024100003	iPhone 15 Pro	10	2024-09-21 00:51:58	Wait to Deliver	Paid	Delete
1726851090531100002	Macbook pro	100	2024-09-21 00:51:31	Wait to Pay	UnPaid	Delete
1726774157703100003	yuanbin's test	5	2024-09-20 03:29:18	Wait to Deliver	Paid	Delete

(c) Manage Orders (Delete Some Illegal orders)

The user interface (UI) design for the secondhand trading app, as shown in the provided product prototype, is structured to provide an intuitive and seamless user experience. Here is a breakdown of the key components:

1. Login and Registration Panel
The login and registration section appears at the top right corner of the interface, allowing users to either log in, register a new account, or access the admin login. This feature ensures secure access to the platform, enabling users to view and manage their personal profiles, post listings, and make purchases.

2. User Profile Section
Once logged in, users have access to their profile, where they can manage their activity such as publishing items, reviewing previous orders, and editing profile information. The profile provides an overview of the user's online/offline status, selling history, and buying history.

3. Address
This section allows users to add or manage their preferred meeting locations, which will be used to coordinate in-person exchanges. This ensures convenience during the transaction process, enabling sellers and buyers to easily agree on where to meet offline for the handover of items.

4. Publish Item/Announcement
A central feature of the UI is the "Publish Item/Announcement" button at the top center of the page, allowing users to quickly post new items for sale. This function includes fields for item descriptions, price settings, and categorization. Users can upload images and specify additional details about the product.

5. Message Center (From Buyer)
The message center provides direct communication between buyers and sellers. This section is accessible from the interface, allowing users to inquire about items, negotiate prices, or confirm transaction details. Real-time messaging enhances user engagement and ensures smoother transactions.

6. Recommender System
Located towards the bottom of the screen, the recommender system suggests items that may be of interest to the user based on their browsing or buying history. This personalized recommendation feature enhances user experience by providing relevant suggestions, making it easier for users to discover items they may want to purchase.

7. Main Product Display
The homepage displays a grid of products with a brief description, price, and seller information. The UI organizes items by categories such as Digital, Essentials, Sports, Books, and Announcements, allowing users to filter through various product types easily.

8. Advertisement Banner
At the top of the homepage, there is space allocated for advertisements. This section allows sellers or third-party advertisers to promote products or announcements directly to the app's users.

9. Footer Section
In the footer, users can browse additional recommendations based on popular or recently viewed products. This horizontal scrolling section enhances visibility for more items and helps promote user interaction with the product offerings.

● Database Design (if applicable) - Ang Li

1. From the perspective of architecture
   1.1 We do not recommend developers use a local database for development. Differences in local environments can lead to inconsistent data across databases, and issues with IP addresses and configuration files may arise during the final code merge, causing unexpected problems.

   1.2 We are using Alibaba Cloud servers to set up our own database. The database is MySQL 5.7, utilizing the clustered index InnoDB. It is configured with 4 CPU cores, 4GB of memory, and 40GB SSD cloud storage. Since there is no significant amount of test data at this stage, we have evaluated that this configuration is fully sufficient to support the development needs at the current phase.

1.3 From a security perspective, our database is currently deployed within a CentOS server. The cloud platform can mitigate small-scale DDoS attacks. Both the database and server follow a strong password policy (meeting the three-out-of-four principle). To further enhance security, we plan to restrict access to specific IP addresses in Iteration 2. This functionality will be implemented during the second phase.

2. From the perspective of business

The database is designed using MySQL 5.7, with a focus on efficient data management and scalability. It includes multiple tables to handle different aspects of the platform, such as user management, order processing, and product listings.

1. sh_user (User Table)
The sh_user table stores the information of all registered users on the platform.
Fields:

- id: Primary key (auto-incremented).
- account_number: The user’s account identifier.
- user_password: The encrypted password of the user.
- nickname: The user's display name.
- avatar: Link to the user's profile picture.
- sign_in_time: Last login time.
- user_status: User account status (1 indicates banned).
- email: User's email address.
- is_active: Indicates whether the account is active.
- activation_token: Token used for account activation.
- token_expiry: Expiration time of the activation token.
- created_at: Timestamp when the user was created.
- updated_at: Timestamp for the last update of the user record.

2. sh_order_address (Order Address Table)
This table stores the delivery address details for each order.
Fields:

- id: Primary key (auto-incremented).
- order_id: Foreign key linking to the corresponding order.
- consignee_name: Name of the recipient.
- consignee_phone: Recipient's phone number.
- detail_address: Detailed delivery address.

3. sh_order (Order Table)

The sh_order table contains records of all transactions made on the platform.

Fields:

- id: Primary key (auto-incremented).
- order_number: Unique order identifier.
- user_id: Foreign key referencing the buyer (user).
- idle_id: Foreign key linking to the product being purchased.
- order_price: Total amount for the order.
- payment_status: Status of the payment (e.g., pending, completed).
- payment_way: Method of payment used (e.g., card, PayPal).
- create_time: Time when the order was created.
- payment_time: Time when payment was completed.
- order_status: Current status of the order (e.g., shipped, delivered).
- is_deleted: Soft delete flag for record management.

4. sh_message (Message Table)

The sh_message table stores communication between users regarding a specific product or order.

Fields:

- id: Primary key (auto-incremented).
- user_id: Foreign key referencing the sender of the message.
- idle_id: Foreign key referencing the product discussed.
- content: Message content.
- create_time: Timestamp when the message was sent.
- to_user: Recipient of the message.
- to_message: Reference to a previous message in the thread.

5. sh_idle_item (Product Listing Table)

This table manages the details of the items listed by users for sale on the platform.

Fields:

- id: Primary key (auto-incremented).
- idle_name: Name of the product.
- idle_details: Description of the product.
- picture_list: List of images for the product.
- idle_price: Price of the product.
- idle_place: Location of the product.
- idle_label: Label or category tag for the product.
- release_time: The date and time when the product was listed.
- idle_status: Status of the listing (1 for active, 0 for inactive).
- user_id: Foreign key referencing the seller.

6. sh_favorite (Favorites Table)
The sh_favorite table tracks the items that users have favorited.
Fields:

- id: Primary key (auto-incremented).
- create_time: Timestamp when the item was favorited.
- user_id: Foreign key referencing the user who favorited the item.
- idle_id: Foreign key linking to the favorited product.

7. sh_admin (Administrator Table)
The sh_admin table contains records of platform administrators.
Fields:

- id: Primary key (auto-incremented).
- account_number: Administrator's login identifier.
- admin_password: Encrypted password for the admin account.
- admin_name: Name of the administrator.

8. sh_address (User Address Table)
This table stores the addresses associated with each user for delivery purposes.
Fields:

- id: Primary key (auto-incremented).
- consignee_name: Name of the person receiving the order.
- consignee_phone: Phone number of the consignee.
- province_name: Province of the consignee.
- city_name: City of the consignee.
- region_name: Region or district of the consignee.
- detail_address: Full delivery address.
- default_flag: Indicates whether this address is the default.
- user_id: Foreign key referencing the user.

● Security Design - Yihan

1. Security Objectives

- Protect user data, including passwords and personal information.
- Ensure secure user authentication and authorization.
- Prevent unauthorized access to sensitive resources.
- Maintain data integrity and confidentiality.

2. Authentication

- Email-based Authentication: The system requires users to register and log in using their Boston University email addresses (ending with @bu.edu). This restriction helps ensure that only members of the BU community can access the system.
- Password-based Authentication: Users must provide a password during registration and login, which is used to verify their identity.
- Password Hashing: User passwords are not stored in plain text. Instead, they are hashed before being stored in the database using a password encoder.

3. Authorization

- Session Management: Upon successful login, the system generates a JWT (JSON Web Token), which is used for maintaining user sessions and authorizing subsequent requests.
- Role-Based Access Control (RBAC): The application uses role-based access control to manage user permissions, ensuring that only authorized users can perform certain actions.

4. Data Protection

- Secure Communication: All API communications are conducted over HTTPS to encrypt data in transit.
- Email Verification: The registration process includes an email verification step, where users must confirm their email address before their account is fully activated.
- Password Reset Functionality: A secure password reset process is implemented, allowing users to safely regain access to their accounts if they forget their passwords.

5. Input Validation and Output Encoding

- Input Validation: The controllers use annotations like @NotNull and @NotEmpty to validate user input, ensuring that required fields are provided.
- Output Encoding: Ensure that any output rendered to the user is properly encoded to prevent XSS.

6. Error Handling and Logging

- Error Handling: The controllers handle errors gracefully, returning appropriate error messages without exposing sensitive information.
- Logging: The system implements comprehensive error handling and logging, which helps in detecting and responding to potential security issues.

7. Account Security

- Secure Logout: The system includes a logout endpoint, which, when fully implemented, will invalidate the user's session token, preventing unauthorized access through old tokens.
- Access Control: There are placeholders for authorization checks on changing and removing items, suggesting that only authorized users (likely item owners) will be able to modify or delete their own items.

8. Security Testing

- Testing Strategies: Unit tests are implemented for service classes to ensure that methods behave as expected. Postman is used for API testing to validate the functionality and security of endpoints.
- Tools Used: Unit testing is performed using JUnit.

9. Compliance and Best Practices

- Regulatory Compliance: The project adheres to relevant regulations such as GDPR and CCPA.
- Security Best Practices: Follow best practices such as the least privilege principle and secure coding guidelines.

10. Future Enhancements

- Implement token blacklisting or short-lived tokens to fully secure the logout process.
- Add rate limiting on sensitive operations to prevent brute-force attacks.
- Implement more granular authorization checks for various operations within the system.
- Consider adding multi-factor authentication for enhanced account security.

Business Logic and/or Key Algorithms - Yingtong

CS673_ Second hand Trading App.pdf

As shown in the flowchart above, we break the main business logic into 5 parts:

1. User Management Algorithm:

```
# User Login
1. Start
2. Input: User submits login credentials (email and password)
3. Fetch the corresponding user data from the database using the input email.
4. If the user exists:
   4.1 Validate the email:
     -  If the email is end with “@bu.edu”:
          -  Show “Registration successful” message.
     -  Else, show “Invalid Email” message.
   4.2 Validate the password:
     -  If the password matches the stored hash:
          -  If the user is not banned or deactivated, proceed to the dashboard.
          -  Else, show "Access Denied: User Banned" message.
     -  Else, show "Invalid Credentials" message.
5. Else, show "User Not Found" message.
6. End

# User Sign In
1. Start
2. Input: User enters BU email and password.
3. Check if the email is end with “@bu.edu”:
   - If invalid, show "Invalid Email" message.
   - If valid, proceed to step 4.
4. Fetch user data from the database based on the email.
5. If the user is found:
    5.1 Verify the password by comparing the input password with the stored hashed
password:
        - If the password is correct:
            5.1.1 Check if the user is banned:
                - If banned, show "Access Denied: User Banned" message and log the
attempt.
                - If not banned, proceed to step 6.
            5.1.2 Check if the user has verified their email:
                - If not verified, show "Please verify your email" message.
                - If verified, log the user in and proceed to step 7.
        - If the password is incorrect, show "Incorrect Password" message.
6. If the user is not found in the database, show "User Not Found" message.
7. Set the session token or create a JWT token for the user (for session management).
8. Redirect the user to the dashboard or home page.
9. End
```

2. Item Management Algorithm

```
1. Start

# Item Publishing
2. Function publishItem(ownerID, title, description, location, category, price, picture):
    2.1 Validate if all required fields (title, description, location, price) are filled.
    2.2 Upload the picture and generate a picture URL.
    2.3 Create a new item entry in the database with the ownerID, title, description,
location, category, price, and picture URL.
    2.4 Set item status to "online".
    2.5 If publishing is successful, return "Item published successfully".
    2.6 Else, return "Error publishing item".

# Search Items
3. Function searchItems(keyword, category, location, priceRange):
    3.1 Fetch all items from the database where:
        - title or description contains the keyword
        - AND category matches the provided category
        - AND location matches the provided location
        - AND price is within the provided priceRange
    3.2 If items are found, return the list of matching items.
    3.3 Else, return "No items found".

# View Item Details
4. Function viewItemDetails(itemID):
    4.1 Fetch item details from the database by itemID.
```

```
   4.2 If item exists:
      - Retrieve owner information (seller info).
      - Retrieve item information such as title, description, location, category, price,
picture URL, and item status.
   4.3 If item doesn't exist or is offline, return "Item not available".
   4.4 Else, return item details and owner info.

# Update Item
5. Function updateItem(itemID, ownerID, updatedData):
   5.1 Fetch item by itemID.
   5.2 If item exists and ownerID matches the item owner:
      - Update item details (title, description, location, category, price) based on the
provided updatedData.
      - If a new picture is provided, upload the picture and update the picture URL.
   5.3 If successful, return "Item updated successfully".
   5.4 Else, return "Error updating item" or "Unauthorized action".

# Delete Item
6. Function deleteItem(itemID, ownerID):
   6.1 Fetch item by itemID.
   6.2 If item exists and ownerID matches the item owner:
      - Change item status to "offline".
      - Remove the item from the public listings.
   6.3 If successful, return "Item deleted successfully".
   6.4 Else, return "Error deleting item" or "Unauthorized action".

7. End
```

3. Order Management Algorithm:

```
1. Start
2. Input: User selects an item and clicks "Purchase"
3. Fetch the item details from the database using the item ID.
4. Check if the item is still available:
   - If item is **available** (status = "online"):
      4.1 Change the item status to "On Hold"
      4.2 Create a new order record with the following details:
         - Order ID
         - Buyer ID
         - Seller ID
         - Item ID
         - Status: "On Hold"
         - Transaction address (user input for offline transaction)
```

```
            - Payment method: "Offline" (cash or other offline options)
            - Date and Time of order
        4.3 Notify the seller about the new order.
        4.4 Notify the buyer that the item is now on hold and to proceed with the offline
transaction.
    - Else if item is **unavailable** (status != "online"):
        4.5 Show error message: "Item is no longer available for purchase."
5. End
```

4. Messaging Algorithm

```
1. Start
2. Input: User initiates a message (buyer or seller), selects an item
3. Check if the item exists in the system:
    - If the item exists, proceed to Step 4
    - Else, show error: "Item not found"
4. Input: Message content and time are provided by the user
5. Create a new message record in the database with:
    - Message ID
    - Item info (ID, name)
    - Buyer info (User ID, name) or Seller info (User ID, name)
    - Message content
    - Timestamp (current time)
6. Bind the message to the selected item
7. Notify the recipient (buyer or seller) of the new message
8. Wait for reply:
    8.1 If the recipient replies:
        - Fetch the previous message (if applicable)
        - Append the reply message with the same item info, user info, and timestamp
        - Store the reply in the message history
    8.2 Else, continue without reply
9. Allow users to:
    - List all messages related to a specific item:
        - Fetch all messages with the item ID from the database and display
    - Delete a message:
        - Remove the message from the database
10. End
```

5. Index Page Algorithm: This is the homepage where users are provided with personalized item recommendations and can interact with an AI assistant. The page dynamically fetches user behavior data to suggest items similar to what they have browsed or purchased before. It uses a User-to-Item (U2I) recommendation algorithm to generate

suggestions.

The modules of index are as follows:
1. Load the index page components:
   - Header (navigation, user login/logout)
   - Item search bar (optional for user input)
   - AI assistant widget
   - Recommended items section

2. User behavior-based item recommendations:
   - Fetch the current user's behavior data (e.g., past views, purchases).
   - Input: User interaction data (clicks, views, past purchases)
   - Use the U2I (User-to-Item) model to analyze behavior and recommend similar items.
   - Display recommended items to the user.

3. Handling recommendations:
   - If user clicks "Recommend Items":
     3.1 Collect the current user's item preferences.
     3.2 Run the recommendation algorithm based on:
       - Similar items to previous interactions.
       - User-to-Item (U2I) model trained on behavior data.
     3.3 Display the recommended items on the page.

● Design Patterns - Srujana N

We use basically 3 kinds of Design patterns, which are further categorized into the following:

- **Creational:** Factory Method, Singleton, Prototype
- **Structural:** Facade, Flyweight, Composite
- **Behavioral:** Template Method, Observer, Strategy

MVC (Model View Controller) is often considered a compound design pattern comprising three key patterns: composite, observer, and strategy.

The Spring Web MVC framework implements this architecture, offering components to create flexible, loosely coupled web applications. By using the MVC pattern, the application's different aspects—input logic, business logic, and UI logic—are separated, promoting loose coupling among them.

- **Model:** Encapsulates application data, typically represented by Plain Old Java Objects (POJOs).
- **View:** Renders model data, usually generating HTML output for the client's browser.

- **Controller:** Handles user requests, constructs the appropriate model, and passes it to the view for rendering.

Applicability: The Model-View-Controller-Repository (MVC-R) architecture is well-suited for the SecondHandTrading application due to its need for clear separation of concerns, effective data persistence, and a responsive user interface. This design pattern allows each component (Model, View, Controller, and Repository) to focus on its specific responsibilities:

1. **Separation of Concerns:**
   - MVC-R clearly delineates user interface logic, data handling, and business logic. This separation enables easier maintenance and scalability, as changes to one component (e.g., user interface) do not significantly impact others (e.g., database interactions).
2. **Data Persistence:**
   - The Repository layer abstracts data access, allowing the application to perform CRUD (Create, Read, Update, Delete) operations seamlessly. This is crucial for managing user data effectively, especially in a trading platform where user transactions must be securely stored.
3. **Responsive User Interface:**
   - Using frameworks like Vue.js for the View component ensures a dynamic and responsive user interface that enhances user experience during interactions, such as registering or searching for items.

This structure enhances maintainability and scalability in web application development.

Example Implementation:

When a user requests to retrieve a message in the SecondHandTrading platform, the process involves multiple layers:

1. **Model Layer:** The `Message` model in the SecondHandTrading application encapsulates essential data, including identifiers, content, timestamps, and relationships with users and idle items. By implementing `Serializable`, it facilitates efficient data transfer. The use of private fields with public accessors ensures data integrity and controlled interactions.
2. **View Layer:**
   The Vue.js View sends a request to fetch message details based on the provided message ID.
3. **Controller Layer:**
   The `MessageController` handles the request via the `/info` endpoint. It retrieves the message using the `messageService.getMessage(id)` method, which checks for the existence of the message and handles any errors, responding with appropriate HTTP

status codes.

4. **Repository Layer:**
   The controller calls the `MessageMapper` interface, which abstracts the data access logic. The `selectByPrimaryKey(Long id)` method queries the database to retrieve the message corresponding to the given ID.

• Any Additional Topics you would like to include?

• References

https://google.github.io/styleguide/javaguide.html
https://www.llama.com/docs/get-started/
Shvets, Alexander. Dive Into Design Patterns. 2021.
Spring Framework. "Web MVC." *Spring Framework Documentation*, https://docs.spring.io/spring-framework/reference/web/webmvc.html.
Tutorialspoint. "Spring Web MVC Framework." *Tutorialspoint*, https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm.

• Glossary

1. **CRUD (Create, Read, Update, Delete)**: The four basic operations of persistent storage in software applications.
2. **Serializable**: A Java interface that allows an object to be converted into a byte stream, facilitating data transfer or persistence.