SubClear

(Team 8)

(Minh Bui, Bryan Lam,Braylon Spann and Kevin David)

Software Design Specification

&

Project Delivery Report

Version: (1) **Date: (12/01/2025)**

Table of Contents

1 Introduction....5
1.1 Goals and objectives....5
1.2 Statement of system scope....5
1.3 Reference Material....5
1.4 Definitions and Acronyms....5
2 Architectural design....5
2.1 System Architecture....5
2.2 Design Rational....6
3 Key Functionality design....6
3.1 [Function 1: XXX]....6
3.1.1 [XXX] Use Cases....6
3.1.2 Processing sequence for [XXX]....6
3.1.3 Structural Design for [XXX]....6
3.1.4 Key Activities....6
3.1.5 Software Interface to other components....6
4 User interface design....6
4.1 Interface design rules....6
4.2 Description of the user interface....7
4.2.1 [XXX]Page....7
5 Restrictions, limitations, and constraints....8
6 Testing Issues (SLO #2.v)....8
6.1 Types of tests....8
6.2 List of Test Cases....8
6.3 Test Coverage....9
7 Appendices....9
7.1 Packaging and installation issues....9
7.2 User Manual....10
7.3 Open Issues....10
7.4 Lessons Learned....10
7.4.1 Project Management & Task Allocations (SLO #2.i)....10
7.4.2 Implementation (SLO #2.iv)....10
7.4.3 Design Patterns....10

7.4.4 Team Communications.....10
7.4.5 Software Security Practice (SLO #4.iii)..... 11
7.4.6 Technologies Practiced (SLO #7)..... 11
7.4.7 Desirable Changes.....11
7.4.8 Challenges Faced.....11

1 Introduction

SubClear is implemented as a web plugin application, with the user interface built using HTML/CSS and the backend logic developed in Java. Many people sign up for multiple services (streaming, cloud storage, gyms, apps, etc.) and forget what they are paying for every month. SubClear aims to give users a clear overview of their active subscriptions, how much they are spending, and when upcoming charges are due, so they can avoid unnecessary payments and make better budgeting decisions.

1.1 Goals and objectives

The main goals of SubClear are to:

- Centralize subscription information for each user
- Show monthly and yearly spending summaries
- Send reminders before renewals or billing dates
- Make it easy to spot unwanted or unused subscriptions

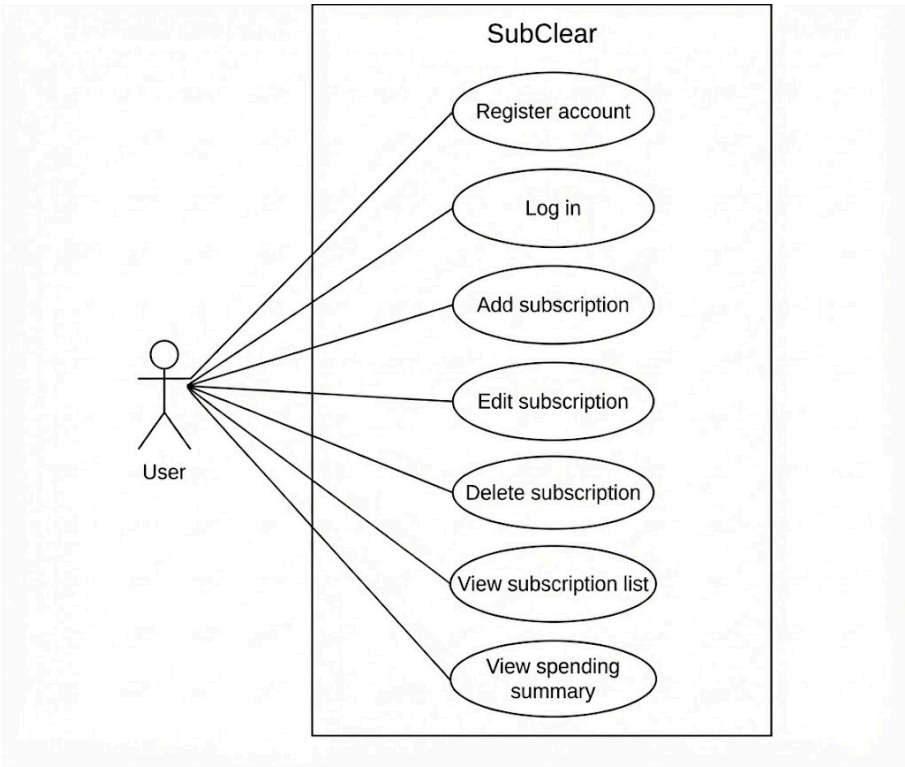
1.2 Statement of system scope

SubClear is a web plugin system that helps users keep track of all their recurring subscriptions in one place. The system focuses on making it easy for users to see what they are paying for, how much they spend each month, and when upcoming charges are due.

The main features and functionalities of SubClear include:

- Allowing users to create an account and log in.
- Adding new subscriptions with details such as name, cost, billing cycle, and next payment date.
- Editing and deleting existing subscriptions.
- Viewing a dashboard that lists all active subscriptions.
- Showing summaries of total monthly and yearly spending on subscriptions.

The system does not handle real payment processing or connect directly to bank accounts. Its purpose is only to help users organize and monitor their subscriptions.



1.3 Reference Material

This section is optional.

1.4 Definitions and Acronyms

This section is optional.

2 Architectural design

2.1 System Architecture

SubClear is organized into three main layers plus a data store. The goal of this architecture is to separate the user interface, the application logic, and the data storage so the system is easier to understand and maintain.

- **UI / Presentation subsystem**
  This layer contains the HTML pages and forms that the user interacts with. It is responsible for displaying the login page, registration page, dashboard, and add/edit subscription forms, and sending user input to the backend.

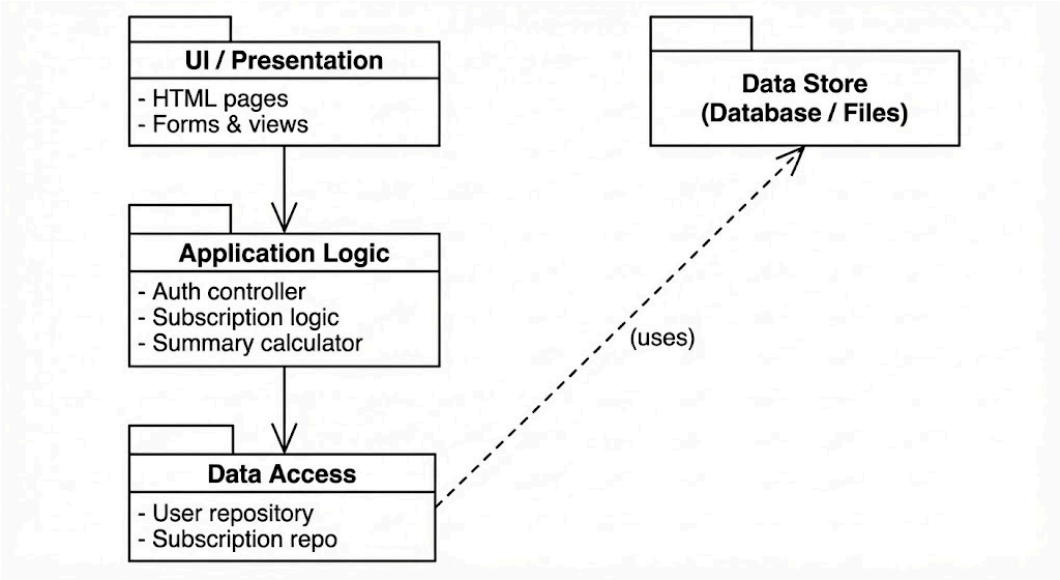
- **Application Logic subsystem**
  This layer contains the main Java classes that implement the behavior of the system. It handles user authentication (register and log in), subscription management (add, edit, delete), and the calculation of monthly and yearly spending summaries. It receives requests from the UI and decides what operations to perform.

- **Data Access subsystem**
  This layer provides methods for saving and loading data. It includes code that reads and writes user and subscription information, and hides the details of how the data is stored.

- **Data Store**
  The data store keeps persistent records for users and their subscriptions (username, password, subscription name, price, billing cycle, next payment date).

These subsystems collaborate as follows: the **UI / Presentation** layer sends user requests to the **Application Logic** layer. The **Application Logic** layer applies the business rules and, when it needs to access stored data, it calls the **Data Access** layer. The **Data Access** layer then communicates with the **Data Store** to read or update records and returns the results back up to the **Application Logic**, which then sends the appropriate response back to the UI to display to the user.

The package diagram shows the major subsystems and the dependencies between them.



2.2 Design Rational

We chose a simple three layer architecture (UI / Presentation, Application Logic, Data Access + Data Store) because it clearly separates the responsibilities of the system. The UI layer only deals with HTML pages and user interaction, the Application Logic layer contains the Java code that implements the main behaviors, and the Data Access layer is responsible for talking to the data store. This separation makes the system easier to understand, debug, and change later, because each layer has a clear role.

Another reason for this architecture is **maintainability and flexibility**. If we need to change how subscriptions are stored (for example, switch from files to a database), most of the changes will be inside the Data Access and Data Store layers, while the UI and Application Logic can stay almost the same. Similarly, if we want to redesign the UI pages, we can do that without touching the business rules in the Application Logic layer.

We also considered a simpler architecture where the UI pages would talk directly to the data store and perform logic in the same module. That approach might be faster to code at the beginning, but it would mix HTML, validation, and data storage code together. This “all in one” style becomes hard to test and extend, especially when more features are added (for example, more complex summaries or extra subscription fields). For that reason, we decided not to use a single layer or tightly coupled design.

More advanced architectures like microservices or a rich single page front end were also not selected. For a small class project like SubClear, splitting the system into many independent services would add a lot of complexity (deployment, communication, etc.) without much benefit. Our three layer architecture is simple enough for our team to implement with HTML/CSS and Java, but still follows good design principles such as separation of concerns and modularity.

3 Key Functionality design

3.1 [Function 1: Subscription Management]

This function lets the user view all of their subscriptions and add, edit, or delete individual subscriptions. It is one of the core features of SubClear because it controls the actual data the system stores.

3.1.1 [Subscription Management] Use Cases

Use Case UC1 – View Subscription List

- **Primary actor:** User
- **Precondition:** User is logged in.
- **Main flow:** The user navigates to the dashboard. The system loads all subscriptions for that user from the data store and displays them in a list with key details (name, cost, billing cycle, next payment date).
- **Postcondition:** The user sees an up-to-date list of their subscriptions.

Use Case UC2 – Add Subscription

- **Primary actor:** User
- **Precondition:** User is logged in and on the dashboard.
- **Main flow:** The user clicks “Add Subscription”. The system shows a form. The user fills in the subscription name, price, billing cycle, and next payment date and submits the form. The system validates the input and, if it is valid, saves the new subscription and refreshes the list.
- **Postcondition:** A new subscription is stored and visible in the list.

Use Case UC3 – Edit Subscription

- **Primary actor:** User
- **Precondition:** User is logged in and has at least one subscription.
- **Main flow:** The user selects a subscription from the list and clicks “Edit”. The system shows a form pre-filled with the existing data. The user updates one or more fields and submits the form. The system validates the changes, saves them, and refreshes the list.
- **Postcondition:** The selected subscription is updated in the data store and on the dashboard.

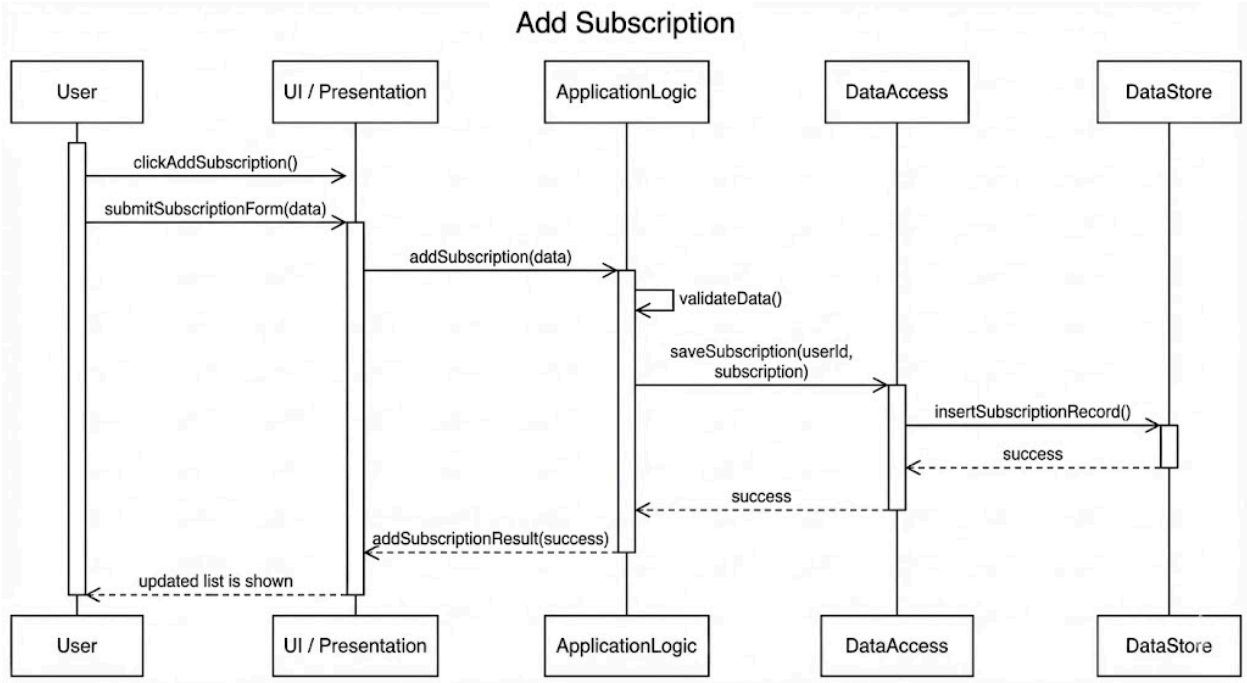
Use Case UC4 – Delete Subscription

- **Primary actor:** User

- **Precondition:** User is logged in and has at least one subscription.
- **Main flow:** The user selects a subscription and chooses “Delete”. The system asks for confirmation. If the user confirms, the system removes the subscription from the data store and reloads the list without that item.
- **Postcondition:** The subscription is removed and no longer appears in the list.

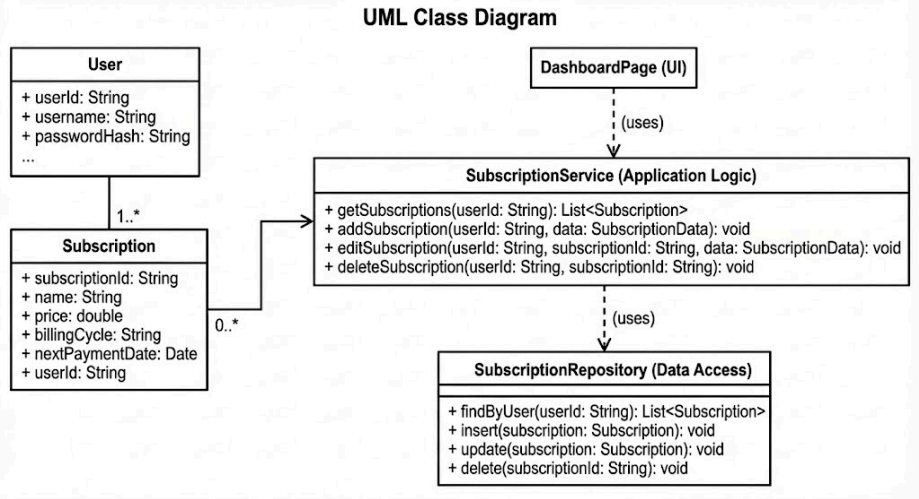
The use case descriptions above define how a logged-in user can view and manage their subscriptions. They show the basic steps, preconditions, and outcomes for viewing the list, adding a new subscription, editing an existing one, and deleting a subscription. These use cases form the basis for the sequence, class, and activity diagrams in the following sections.

3.1.2 Processing sequence for [Subscription Management]



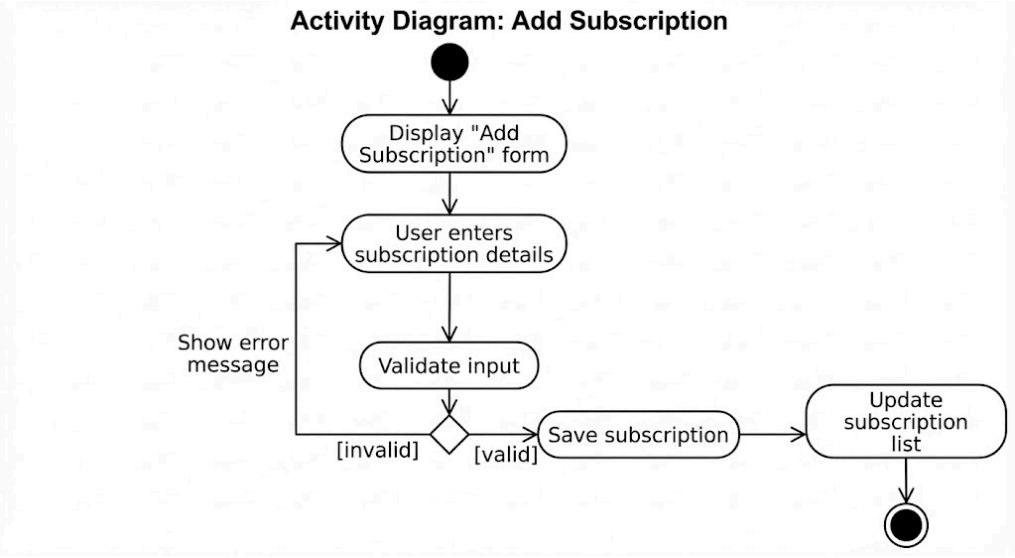
The sequence diagram for “Add Subscription” shows how the different parts of SubClear collaborate to complete this function. The user interacts with the UI to open and submit the add subscription form. The UI forwards the request to the Application Logic layer, which validates the input and calls the Data Access layer to save the subscription. The Data Access layer writes the new record to the data store and returns a success result back through the Application Logic to the UI. Finally, the UI refreshes the page so the user can see the updated subscription list.

3.1.3 Structural Design for [Subscription Management]



The class diagram above shows the main classes involved in subscription management. A User is associated with multiple Subscription objects, representing the subscriptions owned by that user. The DashboardPage in the UI layer depends on the SubscriptionService class in the application logic layer to retrieve and modify subscriptions. SubscriptionService provides operations to get, add, edit, and delete subscriptions, and internally uses the SubscriptionRepository class in the data access layer to perform database or file operations. This structure separates the user interface, business logic, and data access responsibilities while clearly modeling the relationship between users and their subscriptions.

3.1.4 Key Activities



The activity diagram for “Add Subscription” shows the main steps involved in this function from the user’s point of view. The system first displays the “Add Subscription” form and the user enters the subscription details. The input is then validated. If the data is invalid, an error message is shown and the user is sent back to correct the form. If the data is valid, the system saves the new subscription and updates the subscription list on the dashboard, after which the activity ends.

3.1.5 Software Interface to other components

The subscription management function mainly interfaces with the authentication and data access components. It requires a valid logged-in User object from the authentication part of the system before any subscription operations are allowed. All persistent changes are done through the SubscriptionRepository in the Data Access layer, which hides the details of the data store. The UI pages (dashboard and add/edit forms) call the SubscriptionService methods, which form the main programmatic interface for this functionality.

4 User interface design

4.1 Interface design rules

For SubClear, we use a clean, consistent interface across all pages. Every screen uses the same purple gradient background, with a white card in the center that contains the main content. Cards have rounded corners and a light drop shadow so they stand out against the background.

We use a simple, readable font with dark text on a light card for good contrast. Primary actions such as Login, Create Account, and Add Subscription use a purple button style, while destructive actions such as Logout and Delete are shown in red to warn the user. Secondary actions like Cancel use a light gray button.

All forms follow the same layout: labels are placed above the input fields, required fields are marked with an asterisk, and related inputs are grouped together. On the dashboard, information is organized into summary cards at the top and a subscription list below, so users can quickly see their totals and individual subscriptions.

4.2 Description of the user interface

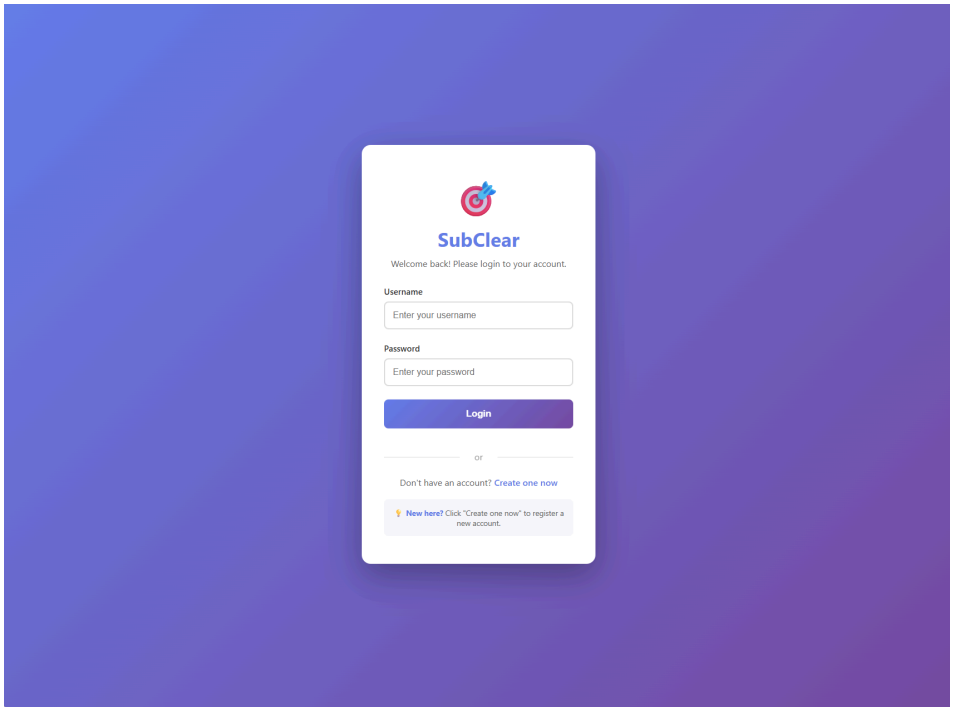
Login page

Register page

Dashboard (empty + with data)

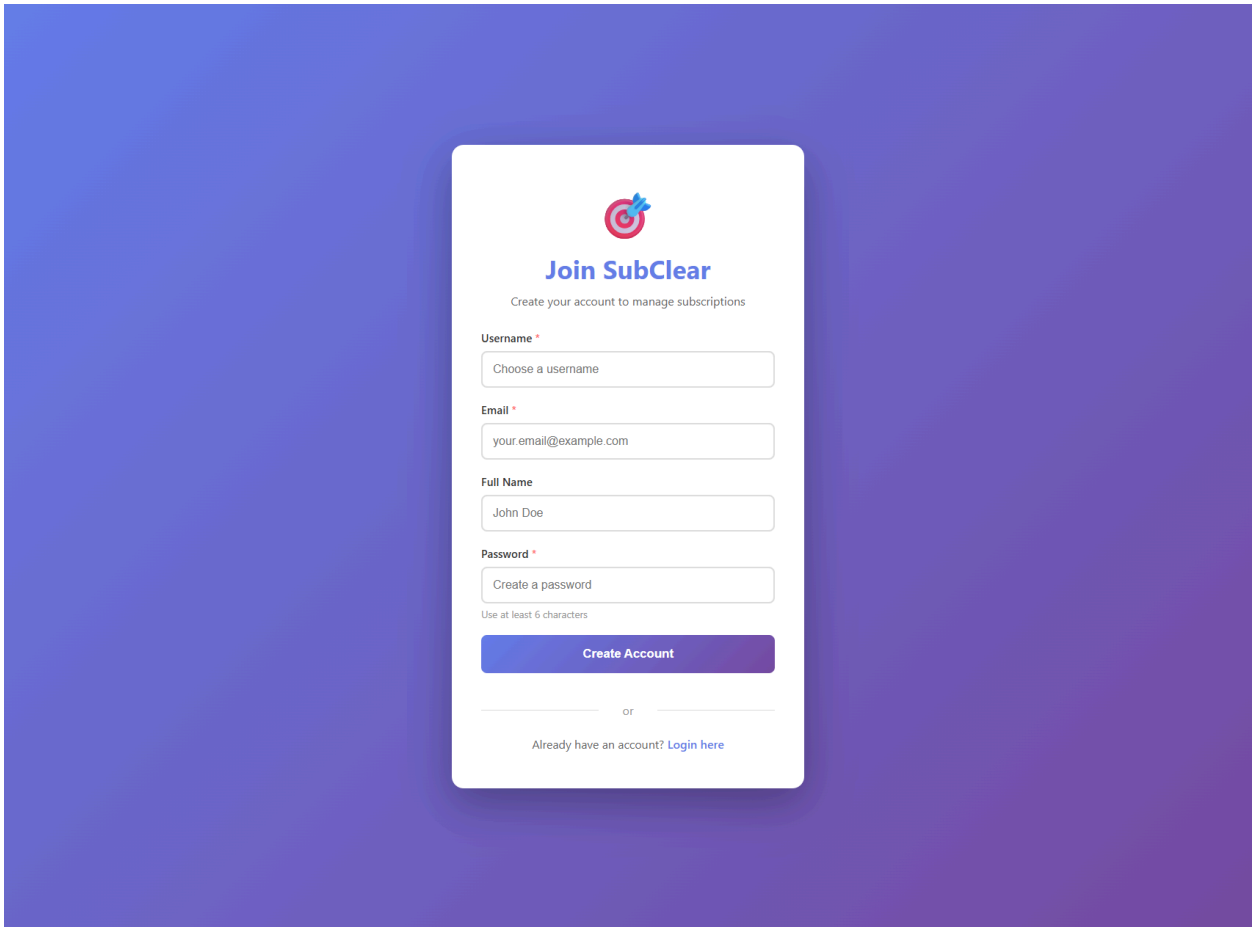
Add Subscription page

4.2.1 [Log In]Page



The login page allows an existing user to sign in to SubClear. The main objects on this page are the centered login card, the username and password input fields, and the purple Login button. At the top of the card there is the SubClear logo and title, with a short welcome message. Under the form there is text that says “Don’t have an account? Create one now” with a clickable link that takes new users to the registration page. There is also a note at the bottom that reminds new users to click the link to create an account. The typical action on this page is for the user to enter their username and password and press Login to continue to the dashboard.

4.2.2 Register Page



4.2.2 Objects and Actions

The register page lets a new user create a SubClear account. The main card at the center of the screen is titled “Join SubClear” and contains a vertical form. The form includes fields for Username, Email, Full Name, and Password, with required fields marked by an asterisk. At the bottom of the form there is a purple Create Account button that submits the information. Below the button there is a line of text with a link: “Already have an account? Login here,” which sends the user back to the login page. On this page the user’s main action is to fill in their details and click Create Account to register.

4.2.3 Dashboard Page

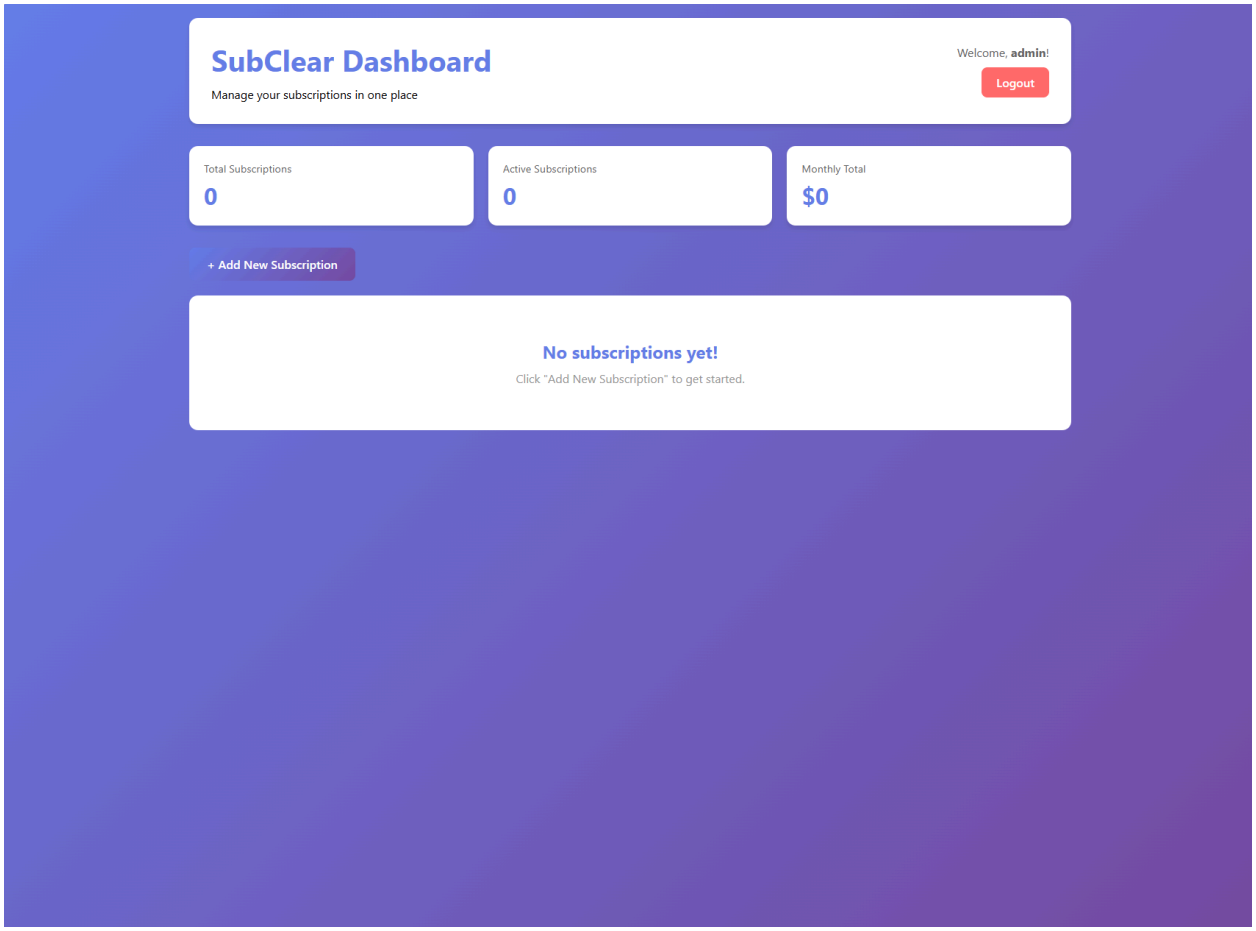
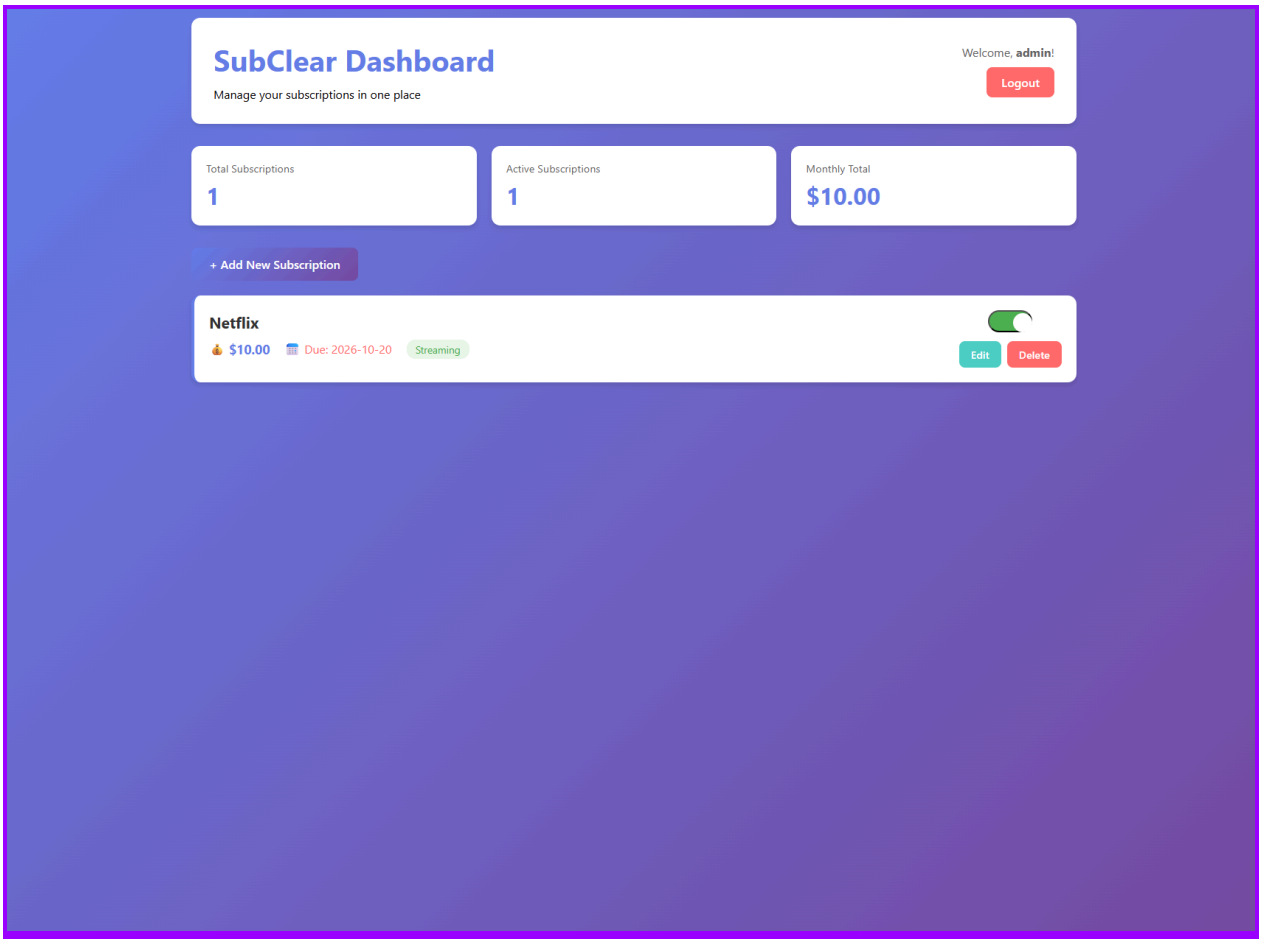


Figure X: Dashboard with no subscriptions



- Figure Y: Dashboard with subscriptions

4.2.3.2 Objects and Actions

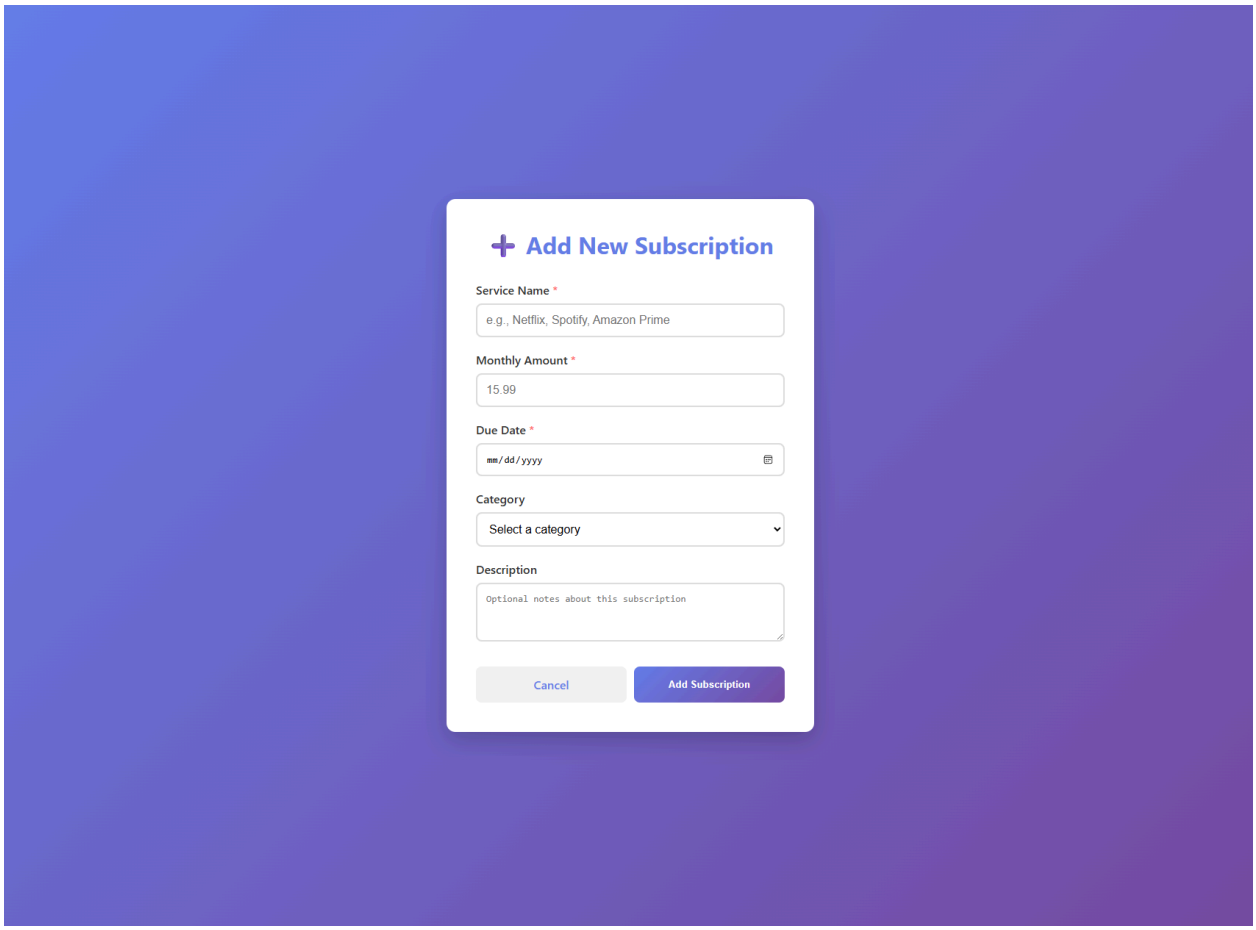
The dashboard page is the main page users see after logging in. At the top of the card, the title “SubClear Dashboard” is displayed, along with a short description “Manage your subscriptions in one place.” On the right side of the header the interface shows a welcome message (for example, “Welcome, admin!”) and a red Logout button that signs the user out.

Below the header there are three summary cards showing Total Subscriptions, Active Subscriptions, and Monthly Total so the user can quickly see their overall spending. Under these cards there is a purple “+ Add New Subscription” button that opens the add-subscription form.

In the empty state (Figure X), a large message area tells the user “No subscriptions yet!” with a hint to click “Add New Subscription” to get started. When the user has one or more subscriptions (Figure Y), this area shows a list of subscription cards.

Each subscription card displays the service name (e.g., Netflix), the monthly amount, the due date, and a small category label such as “Streaming.” Each card also includes a toggle to mark the subscription as active/inactive and two buttons: Edit to modify the subscription and Delete to remove it. The main actions on this page are adding new subscriptions, reviewing the summary totals, editing existing subscriptions, and deleting subscriptions the user no longer wants.

4.2.4 Add Subscription Page



4.2.4.2 Objects and Actions

The add subscription page allows the user to create a new subscription entry. The centered card is titled “Add New Subscription” and contains a form with several fields: Service Name (for example, Netflix or Spotify), Monthly Amount, Due Date (with a calendar picker), Category (a drop-down list where the user can choose a type such as Streaming or Utilities), and an optional Description text area for notes. At the bottom of the form there are two buttons: a gray Cancel button that returns the user to the dashboard without saving, and a purple Add Subscription button

that submits the form. When the user fills in the required fields and clicks Add Subscription, the system saves the new subscription and then shows it on the dashboard list.

5 Restrictions, limitations, and constraints

For this class project, the SubClear system has the following technical restrictions and limitations:

- **Java backend only.**
  The server side logic is implemented in Java and requires a compatible JDK to run. Other backend languages are not supported.

- **Web application running in a browser.**
  SubClear is designed as a browser based tool, not as a native mobile app or desktop application.

- **Modern desktop browsers only.**
  The user interface has been designed and tested for modern browsers such as Google Chrome and Microsoft Edge on a desktop or laptop. Behavior on older browsers or mobile devices is not guaranteed.

- **Single user web session, no external authentication.**
  Authentication is handled by a simple username/password system within SubClear. Third party login methods (Google, Facebook, etc.) are not supported.

- **Local data store only.**
  All user and subscription data is stored in a single data store (e.g., a local SQL database or file storage) configured on the server. The current design does not support distributed databases, replication, or automatic backups.

- **Limited scope of financial features.**
  SubClear only tracks subscription information (name, amount, due date, category) and calculates basic monthly totals. It does not connect to real bank accounts, credit cards, or payment gateways, and it does not perform any real billing.

- **Non-scalable architecture.**
  The system is intended for demonstration and small numbers of users. Performance and scalability for a large production environment were not primary design goals.

These constraints were acceptable for our project because the main objective is to demonstrate the core functionality of subscription tracking and to practice good design and architecture, rather than to deliver a fully production-ready system.

6 Testing Issues (SLO #2.v)

Test strategy and preliminary test case specification are presented in this section. Additional test cases are located in the Appendix section in spreadsheet format.

6.1 Types of tests

For SubClear, we plan to use the following types of tests:

1. **Performance Tests** – to check that common actions such as loading the dashboard and saving a new subscription respond within an acceptable time (for example, within 2–3 seconds in a normal local environment).

2. **Accuracy Tests** – to verify that the system returns correct results, such as successful login with valid credentials, correct error messages for invalid input, and correct monthly totals based on the stored subscriptions.

3. **User Interface Tests** – to make sure the pages are easy to use and that buttons, links, and form fields behave as expected. For example, new users should be able to register and add a subscription without extra explanation.

4. **Security Tests** – to confirm that only authenticated users can access the dashboard and subscription data, and that invalid login attempts are handled safely.

5. **Repeatability Tests** – to ensure that the system gives the same result when the same action is repeated, such as saving the same type of subscription multiple times or refreshing the dashboard.

6.2 List of Test Cases

Test Case 1 – Valid login (Accuracy Test)

- **Test Type:** Accuracy Test
- **Testing range:** User login feature

- **Testing input:** Existing username and correct password
- **Testing procedure:**
    1. Open the Login page.
    2. Enter a valid username and matching password.
    3. Click the **Login** button.
- **Expected test result:** User is redirected to the dashboard page showing the summary cards and subscription list.
- **Testers:** Bryan Lam
- **Test result:** Passed

Test Case 2 – Invalid login (Security / Accuracy)

- **Test Type:** Security Test
- **Testing range:** Login with wrong password
- **Testing input:** Existing username and incorrect password
- **Testing procedure:**
    1. Open the Login page.
    2. Enter a valid username and a wrong password.
    3. Click the **Login** button.
- **Expected test result:** System rejects the login and displays an error message (e.g., “Invalid username or password”) while staying on the login page. Dashboard is **not** accessible.
- **Testers:** Minh Bui
- **Test result:** Passed

Test Case 3 – Add new subscription (Accuracy / Repeatability)

- **Test Type:** Accuracy Test
- **Testing range:** Add Subscription feature
- **Testing input:**
    - Service Name: “Netflix”
    - Monthly Amount: 10.00
    - Due Date: a valid future date
    - Category: “Streaming”
- **Testing procedure:**
    - Log in as a valid user.
    - On the dashboard, click **Add New Subscription**.
    - Fill in the form with the input values above and click **Add Subscription**.
    - Repeat steps 2–3 with a different service name (e.g., “Spotify”).
- **Expected test result:**
    - Both subscriptions appear on the dashboard list.
    - The **Monthly Total** card shows the correct sum of all monthly amounts.
- **Testers:** Bryan Lam , Minh Bui
- **Test result:** Passed

Test Case 4 – Delete subscription (User Interface / Accuracy)

- **Test Type:** User Interface Test
- **Testing range:** Delete Subscription feature
- **Testing input:** Existing subscription (e.g., Netflix)
- **Testing procedure:**

    - Log in as a user that already has at least one subscription.
    - On the dashboard, locate the subscription in the list.
    - Click the **Delete** button on that subscription card and confirm if a confirmation step exists.

- **Expected test result:**
    - The selected subscription disappears from the list.
    - The **Total Subscriptions** and **Monthly Total** values update accordingly.
- **Testers:** Bryan Lam
- **Test result:** Passed

Test Case 5 – Performance of dashboard load (Performance Test)

- **Test Type:** Performance Test
- **Testing range:** Dashboard page loading time
- **Testing input:** User account with several subscriptions (e.g., 10–20 records)
- **Testing procedure:**
    1. Log in as a user with multiple subscriptions.
    2. Measure the time from clicking **Login** until the dashboard is fully displayed.
    3. Repeat the test a few times and record the times.
- **Expected test result:** Dashboard loads and displays all subscriptions within about 2–3 seconds in the test environment.
- **Testers:** Bryan Lam
- **Test result:** Passed

6.3 Test Coverage

Functional requirements

- **FR1: User can register an account.**
  Status: *Partially tested.* The registration form has been manually tested with valid and invalid inputs to confirm that accounts can be created. Additional boundary tests (e.g., very long usernames) may still be required.

- **FR2: User can log in and log out.**
  Status: *Tested.* Test cases 1 and 2 verify successful login with valid credentials and rejection of invalid credentials. Logging out has been manually tested by clicking the Logout button and confirming that the user is returned to the login page.

- **FR3: User can view their subscription dashboard.**
  Status: *Tested.* After login, the dashboard is displayed and the summary cards show the number of subscriptions and monthly total. This is covered in Test Case 1 and 3.

- **FR4: User can add subscriptions.**
  Status: *Tested.* Test Case 3 verifies that new subscriptions can be added and appear on the dashboard with the correct monthly total.

- **FR5: User can delete subscriptions.**
  Status: *Tested.* Test Case 4 checks that a subscription can be deleted from the list and that the totals update.

- **FR6: User can edit subscriptions.**
  Status: *Partially tested.* Basic manual testing has been done to confirm that editing a subscription updates the values on the card and in the monthly total. More detailed test cases can be added later.

Non-functional requirements

- **NFR1: Usability /pages should be easy to understand for new users.**
  Status: *Informally tested.* The interface uses consistent layout, labels, and button styles. Informal user interface tests show that new users can register, log in, and add a

subscription without extra help.

- **NFR2: Performance /basic operations should respond within a few seconds.**
  Status: *Partially tested.* Test Case 5 measures dashboard loading time and shows acceptable performance on the development machine. More extensive performance testing with larger data sets could be added.

- **NFR3: Security /only logged-in users can access subscription data.**
  Status: *Tested at a basic level.* Attempting to access the dashboard without logging in, or with invalid credentials, is blocked as covered by Test Case 2 and manual checks. Advanced security (encryption, password policies) is outside the scope of this project.

7 Appendices

7.1 Packaging and installation issues

How to install and prepare the system to run

1. **Prerequisites**
    - Java JDK installed (version 17 or later).
    - Maven installed.
    - A modern web browser (Chrome, Edge, etc.).
2. **Download the source code**
    - Clone the GitHub repository using git clone or download it as a ZIP and extract it to a local folder.
3. **Configure the application (if needed)**
    - In the current prototype, subscriptions are stored using a simple in memory data store, so there is no separate database setup required.

Build and run

- Open a terminal in the project root (where pom.xml is located).
- Run

These steps are enough for a new developer or tester to get SubClear running on their machine and access the main pages (Login, Register, Dashboard, Add Subscription).

7.2 User Manual

1. Register a new account

1. Open the Login page.
2. Click the **“Create one now”** link at the bottom of the card.
3. On the **Join SubClear** page, fill in the **Username**, **Email**, **Full Name**, and **Password** fields.
4. Click **Create Account**.
5. If everything is valid, the account is created and the user can proceed to log in.

2. Log in

1. On the Login page, enter your username and password.
2. Click the **Login** button.
3. If the credentials are correct, you are redirected to the **SubClear Dashboard**.

3. View the dashboard

- On the dashboard, the top summary cards show:
    - **Total Subscriptions**
    - **Active Subscriptions**
    - **Monthly Total**
- Below the cards, the subscription list shows each subscription’s name, amount, due date, and category.

4. Add a new subscription

1. From the dashboard, click the **+ Add New Subscription** button.
2. On the **Add New Subscription** page, enter:

- Service Name
- Monthly Amount
- Due Date
- Category
- (Optional) Description

3. Click **Add Subscription** to save the subscription, or **Cancel** to go back without saving.
4. After saving, you are returned to the dashboard and the new subscription appears in the list. The monthly total is updated.

5. Edit an existing subscription

1. On the dashboard, locate the subscription you want to change.
2. Click the **Edit** button on that subscription card.
3. Adjust any fields (amount, due date, category, etc.).
4. Save the changes. The dashboard updates with the new values.

6. Delete a subscription

1. On the dashboard, locate the subscription to remove.
2. Click the **Delete** button on that subscription card.
3. Confirm the deletion if prompted.
4. The subscription disappears from the list and the totals are recalculated.

7. Logout

- Click the red **Logout** button in the top right corner of the dashboard header to end the session and return to the Login page.

7.3 Open Issues

The following features and improvements were considered but not fully implemented in the current version of SubClear:

- **Email or notification reminders** before a subscription's due date.
- **More advanced filtering and sorting** on the dashboard (for example, filter by category or show only upcoming renewals).
- **Richer analytics**, such as charts showing spending trends over time.
- **Stronger security features**, such as password reset, password strength checks, and full encryption of sensitive data.
- **Persistent database storage** with backup and restore instead of the current simple data store.

These items are left as future work if the project is extended beyond the course.

7.4 Lessons Learned

7.4.1 Project Management & Task Allocations (SLO #2.i)

Our team divided the work mainly by skill and interest. One member focused on the HTML/CSS front end pages, another worked on the Java back end logic, and another helped with testing and documentation. We used GitHub to share code and track changes, and we communicated through group chat to coordinate tasks.

At the beginning, our planning was a bit loose and we underestimated how long integration between front end and back end would take. Later in the project we improved by breaking down the work into smaller tasks (for example, "implement add subscription API", "wire dashboard totals", "finish login page") and checking in more often. In the future we could improve by setting clearer milestones earlier, assigning owners for each requirement, and doing shorter, more frequent status updates so issues are caught sooner.

7.4.2 Implementation (SLO #2.iv)

During implementation we tried to keep the code aligned with the layered design from the earlier sections: UI pages call a service layer, and the service layer uses a repository layer. When we noticed duplicated logic or overly long methods, we refactored them into smaller helper methods.

We also used discord for basic code review. Before merging, another team member looked over the changes to check for obvious bugs and style issues. Overall, the final implementation is mostly consistent with the system design: we ended up with a clear separation between the **UI / Presentation**, **Application Logic**, and **Data Access** components, as shown in our architecture diagrams. Some shortcuts were taken for the data store and security to save time, but the main structure matches the design.

7.4.3 Design Patterns

The main design pattern we followed was a simple **Model View Controller (MVC) style** separation:

- The HTML pages act as the **View**.
- The Java service classes act like the **Controller/business logic**, handling requests and applying rules.
- The repository classes and data classes act as the **Model**, storing and retrieving subscription and user data.

We also used a **Repository pattern** for data access, so the rest of the code only depends on methods like findByUser or insert, without needing to know how the data is stored. This makes it easier to replace the simple data store with a real database later.

7.4.4 Team Communications

Our team mainly communicated using discord and short in person meetings during lab time. We shared screenshots and code snippets when someone got stuck. This worked reasonably well,

but sometimes people were quiet and tasks got delayed because others assumed someone else was handling them.

In the future, we could improve by scheduling regular 10–15 minute check ins, writing down who owns each task, and being more proactive about asking for help when blocked.

7.4.5 Software Security Practice (SLO #4.iii)

We incorporated security mainly by restricting access to the dashboard so that only logged in users can see subscription data. The login screen is the entry point, and all subscription operations are done under a user account. We also validate input on the server side to avoid saving obviously invalid values.

Because this is a course project, we did not implement more advanced security mechanisms such as full password encryption, HTTPS, or detailed access control. If the system were to be deployed in a real environment, the next steps would be to enforce stronger password handling, store credentials using a secure hash function, and always serve the application over HTTPS.

7.4.6 Technologies Practiced (SLO #7)

This project gave us a chance to practice several technologies and skills we might not have used otherwise:

- Building a multi page HTML/CSS interface with a consistent visual design.
- Structuring a Java application using separate layers (service, repository).
- Using Maven to build and run the project.
- Working with Git and GitHub for version control, branching, and merging.
- Creating UML diagrams (use case, sequence, class, activity) to describe our design.

7.4.7 Desirable Changes

If we had another month to work on SubClear, we would focus on adding more “real world” features. We would like to connect the system to a proper SQL database so data is persistent across restarts and can handle more users. We also want to add email or in app reminders before due dates, and better analytics such as charts showing how subscription spending changes over time. Finally, we would refactor parts of the codebase to reduce duplication and improve error handling and validation.

7.4.8 Challenges Faced

For us, the hardest part of the project was the system implementation phase. Writing the requirements and drawing the design diagrams was straightforward once we agreed on the idea, but actually wiring the front end HTML pages to the Java back-end and making everything work together took more time than expected. Small bugs in routing or data handling sometimes made the whole feature fail, and debugging those issues was challenging. This experience showed how important it is to have a clear design but also to leave enough time for implementation and testing.