**CS673 Software Engineering** 

**Team 2 - SoloSavings**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
|  |  |  |  |
| Maryam Hussein | Design and Implementation leader | *Maryam hussein* | 09/19/2023 |
|  |  |  |  |
| Andrew Ouellette | Configuration Leader | *Andrew Ouellette* | 9/25/2023 |
| Christan Pratt | Requirements Leader | *Christan Pratt* | 09/24/2023 |
| Hanwen Zhang | Security Leader | *Hanwen Zhang* | 9/17/2023 |
| XinZhao | Flex Leader | *ZhaoXin* | 9/24/2023 |
| Maryam Hussein | Design and Implementation leader |  | 10/09/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Christan Pratt  Requirements Leader | 10/9/2023 | Add JWT Architecture diagram |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.4d34og8)

[References](#_heading=h.2s8eyo1)

[Glossary](#_heading=h.17dp8vu)

# Introduction

this document, our guide as we set out on the exciting journey of creating the SoloSavings web application. SoloSavings is a budget management web app that will help us handle income, expenses, and track budgets. We will build a maven project using Java Spring Boot. In these pages, we'll explore the details of how this software will work and how we'll design it.

The main aim of this document is to outline our design plan for making SoloSavings a reality. It's not filled, it's a practical guide that lays out the design and Architecture steps we need to take and the insights we need to gather. We'll transform our SoloSavings idea into a fully functional, user-friendly, and secure web app.

# Software Architecture

The SoloSavings web application is designed with a **layered architecture** that separates concerns and promotes modularity and maintainability. It consists of the following key components:

1.Frontend (UI):

The frontend is responsible for the user interface and user experience.

Technologies:Jsp, JavaScript, HTML, CSS.

The frontend communicates with the backend through RESTful APIs.

2.Backend (Java Spring Boot):

The backend handles business logic, data processing, and interacts with the database.

Technologies: Java Spring Boot, Spring Security for authentication and authorization.

It exposes RESTful APIs to the frontend for data retrieval and manipulation.

3.Database (MySQL):

MySQL is used as the database management system to store user data, financial records, and other application data.

The backend interacts with the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.

4.Frontend-Backend Communication:

The frontend communicates with the backend through RESTful APIs.

REST endpoints are defined in the backend to handle HTTP requests from the frontend.

5.Dependency and Interaction:

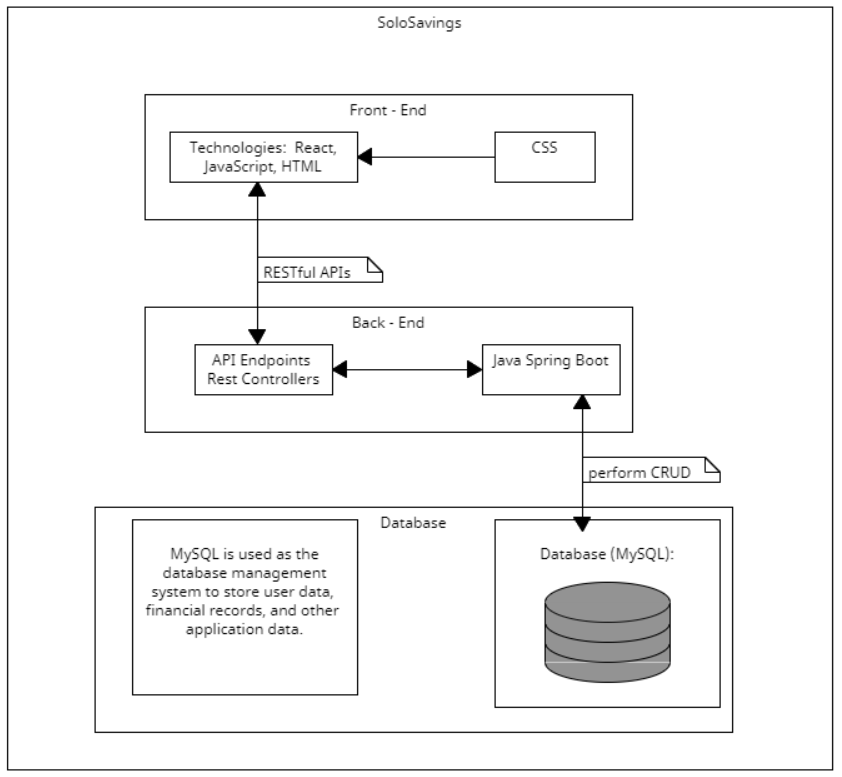
The frontend interacts with the backend through API calls, making HTTP requests to fetch and update data.

Spring Boot acts as the middleware between the frontend and the database, processing requests and managing data retrieval and storage.

MySQL stores user data, financial records, and application data.

6.Framework:

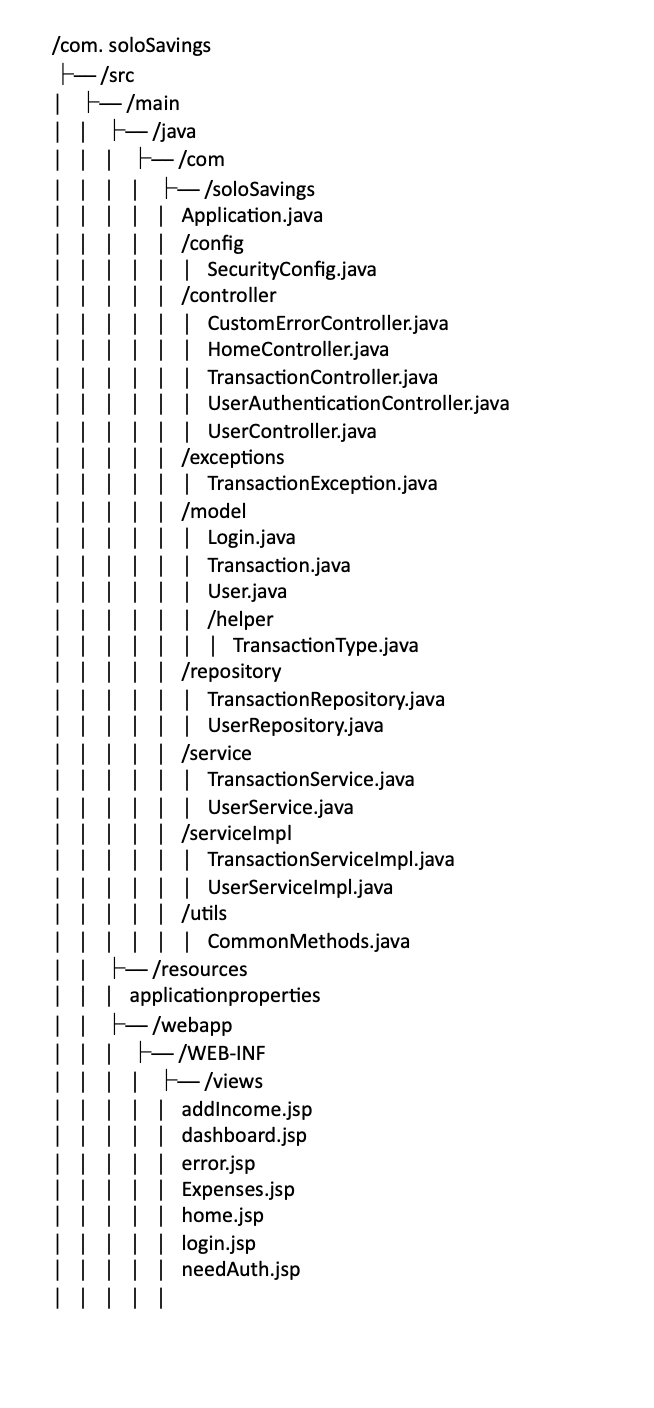
Java Spring Boot is used as the primary backend framework, providing features for handling RESTful APIs, security, and database interactions.



# Class Diagram

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.

# 



**com.solosavings.controller Package:**

contains classes responsible for handling incoming HTTP requests from clients, interacting with the services layer, and returning responses. Controllers act as intermediaries between the user interface and the business logic.

**com.solosavings.service Package:**

houses interfaces that define contracts for various business logic components in your application. These interfaces specify the operations and methods that must be implemented by the service layer. Service classes will implement these interfaces to provide the actual functionality.

**com.solosavings.serviceImpl Package:**

package contains the implementation classes for the service interfaces defined in the com.solosavings.service package. These classes contain the actual business logic and perform tasks such as processing user data, performing calculations, and interacting with the repository layer.

**com.solosavings.model Package:**

holds classes that represent the data structures and entities used in your application. These classes define the structure of your data and may include attributes, relationships, and methods for data manipulation.

**com.solosavings.repository Package:**

contains interfaces that define data access operations for database interactions. These interfaces are used with Spring Data JPA to simplify database operations such as querying, inserting, updating, and deleting records.

**com.solosavings.utils Package:**

contains utility classes that provide common functionality or helper methods used throughout the application.

**com.solosavings.exceptions Package:**

houses custom exception classes tailored to handle specific error scenarios within your application. These exceptions help you manage and respond to exceptional conditions gracefully.

**com.solosavings Package:**

is the root package for your SoloSavings application and contains the Application class, which serves as the entry point of your Spring Boot application.

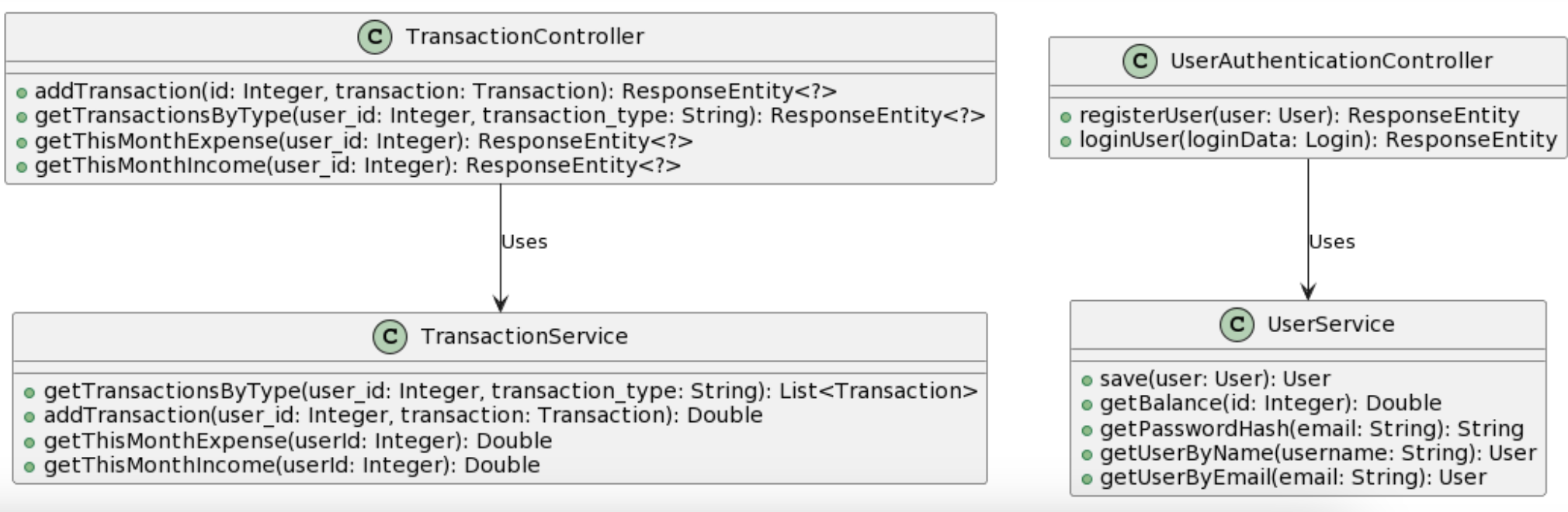
resources/templates Directory:

stores template files used for rendering views in your application. These templates are typically HTML files or other view templates used to generate the user interface.

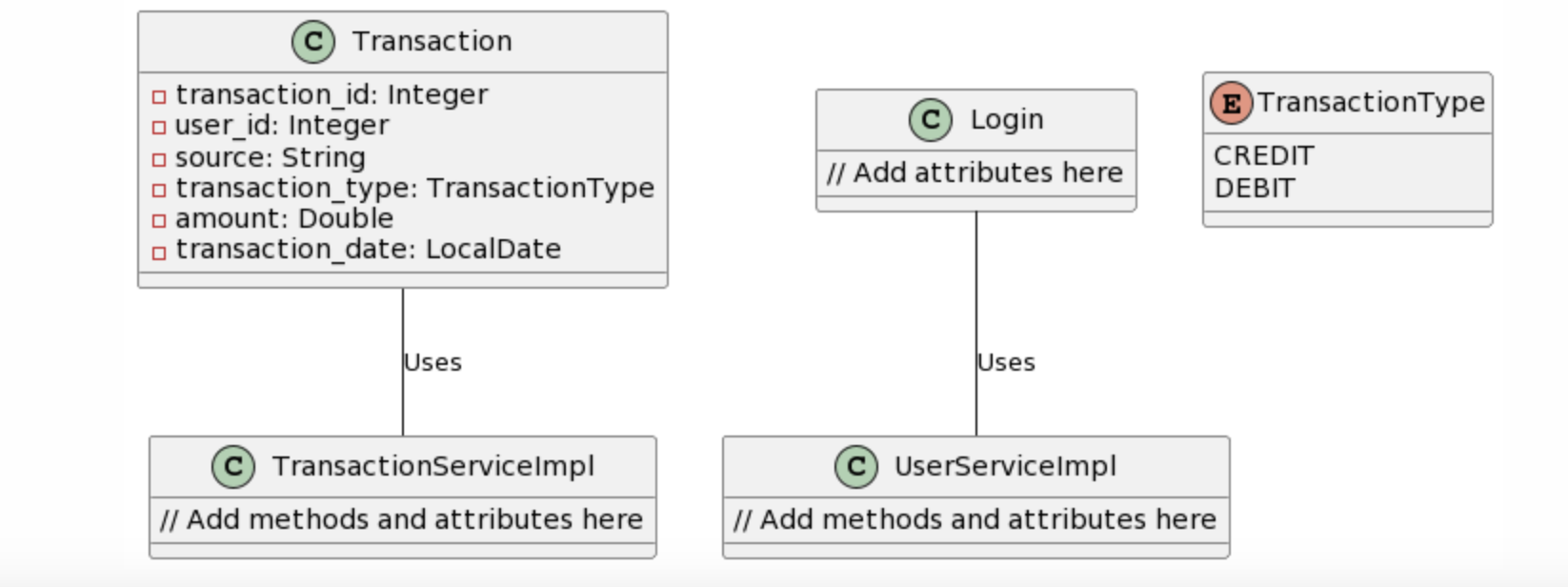
**com.solosavings.test Package:**

is used for writing test classes to perform unit testing of your application components, including controllers and services. It helps ensure the correctness and reliability of your code.

* **Class Diagram for the controller and service Packages**



* **Class Diagram for the serviceImpl and model Packages:**

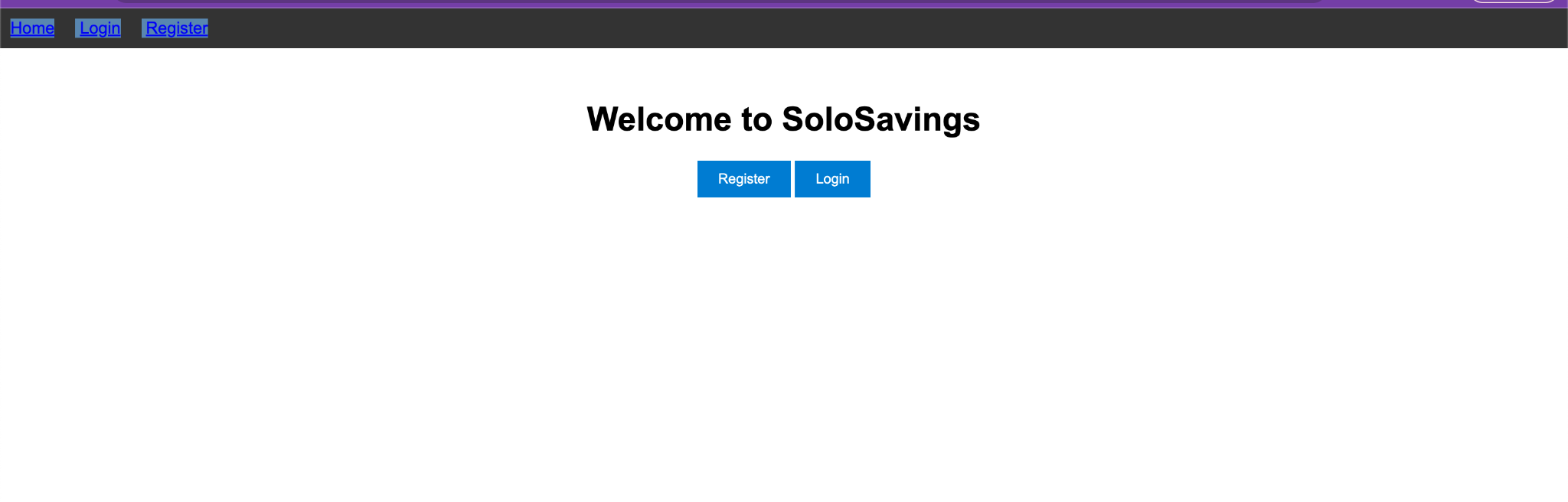


# 

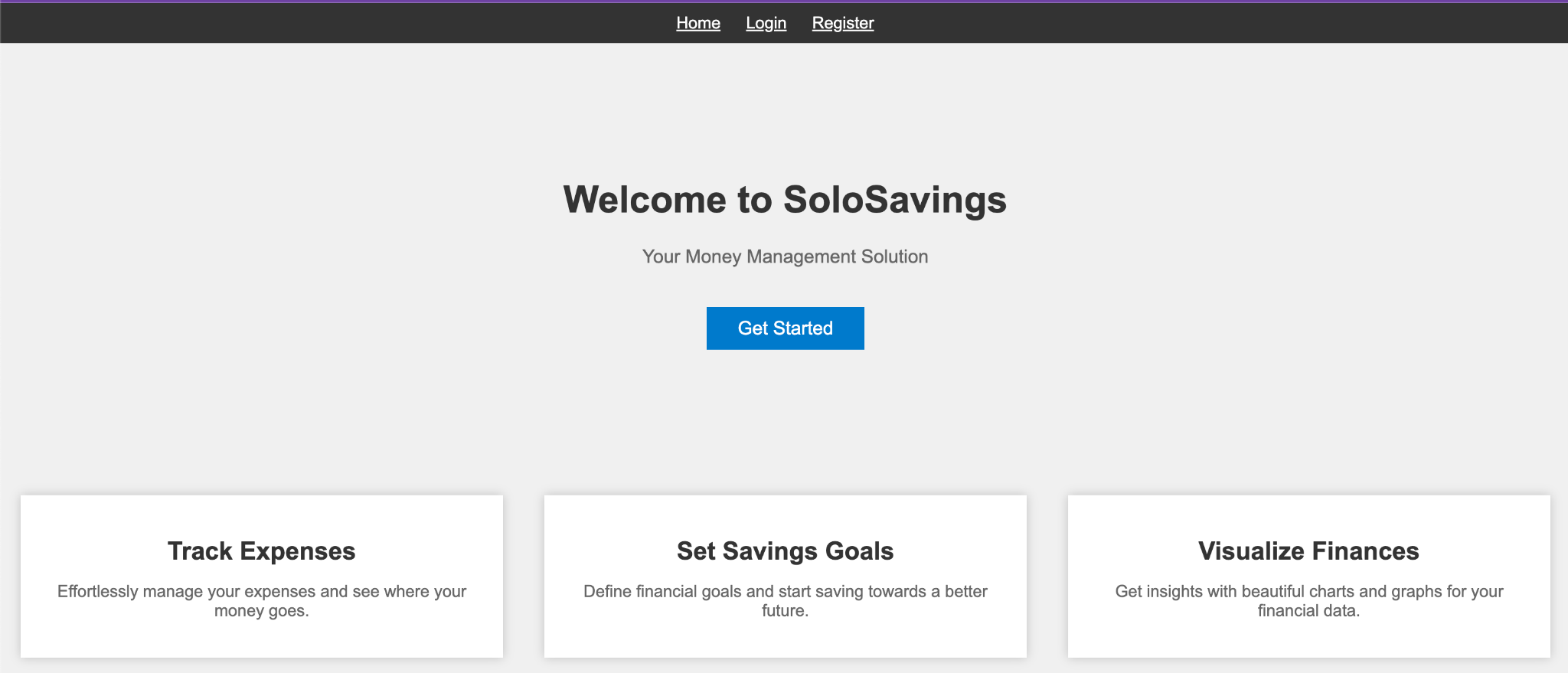
# UI Design (if applicable)

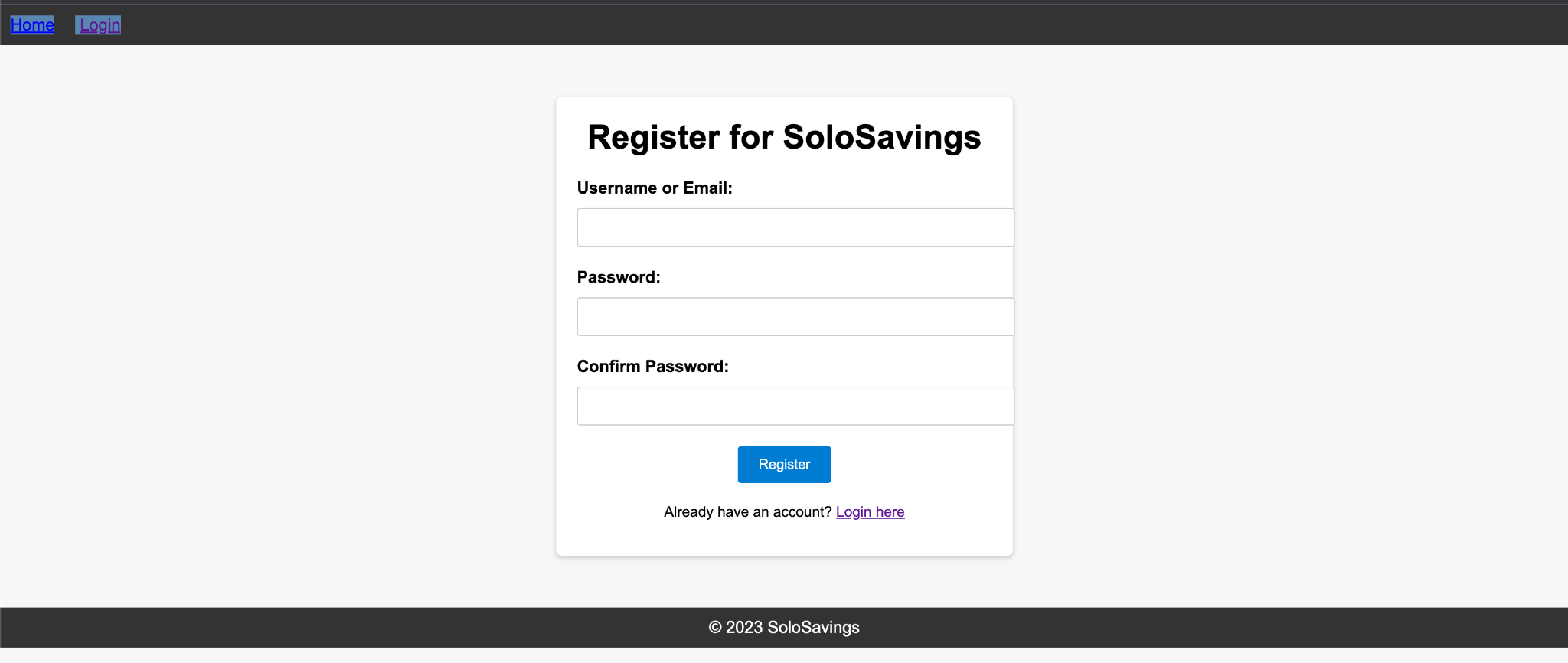
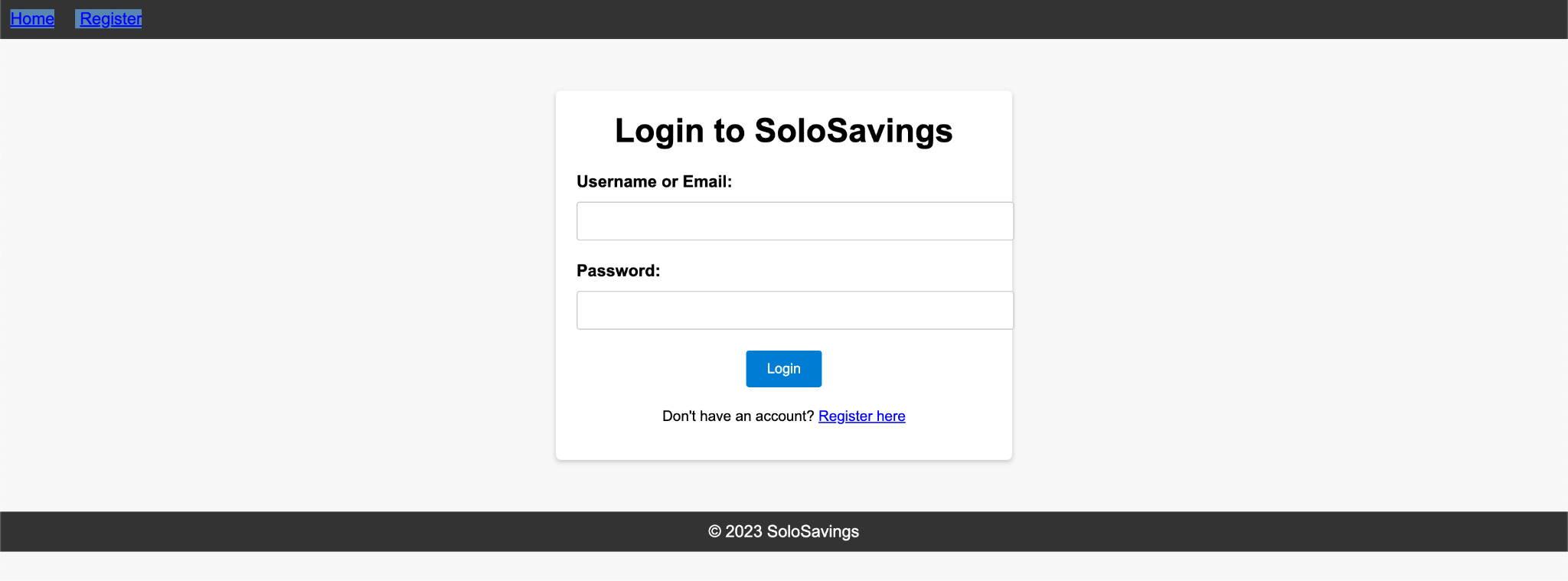
In this section, you can describe your UI design. You can include both your initial design before the implementation and the screenshots of your UI after the implementation.

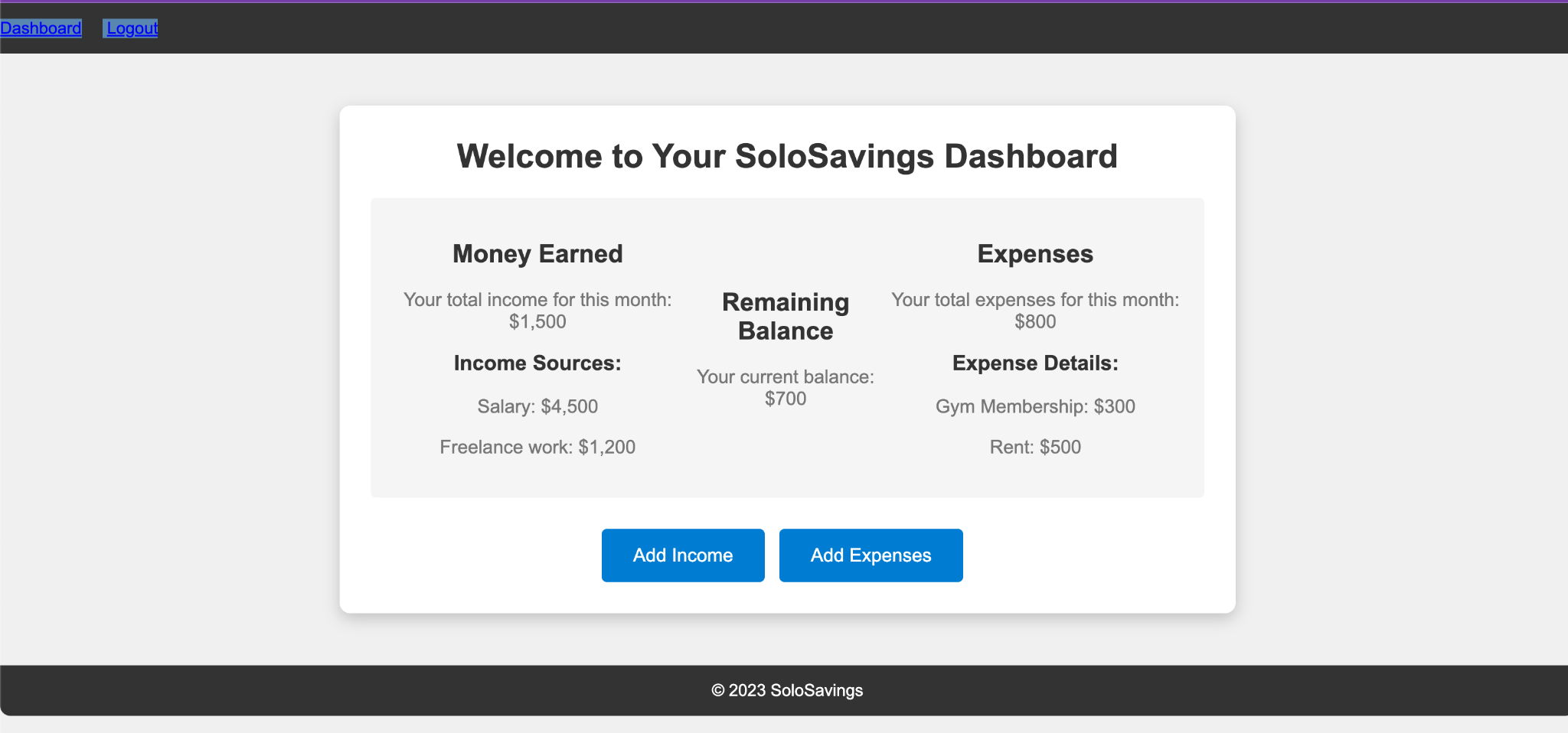
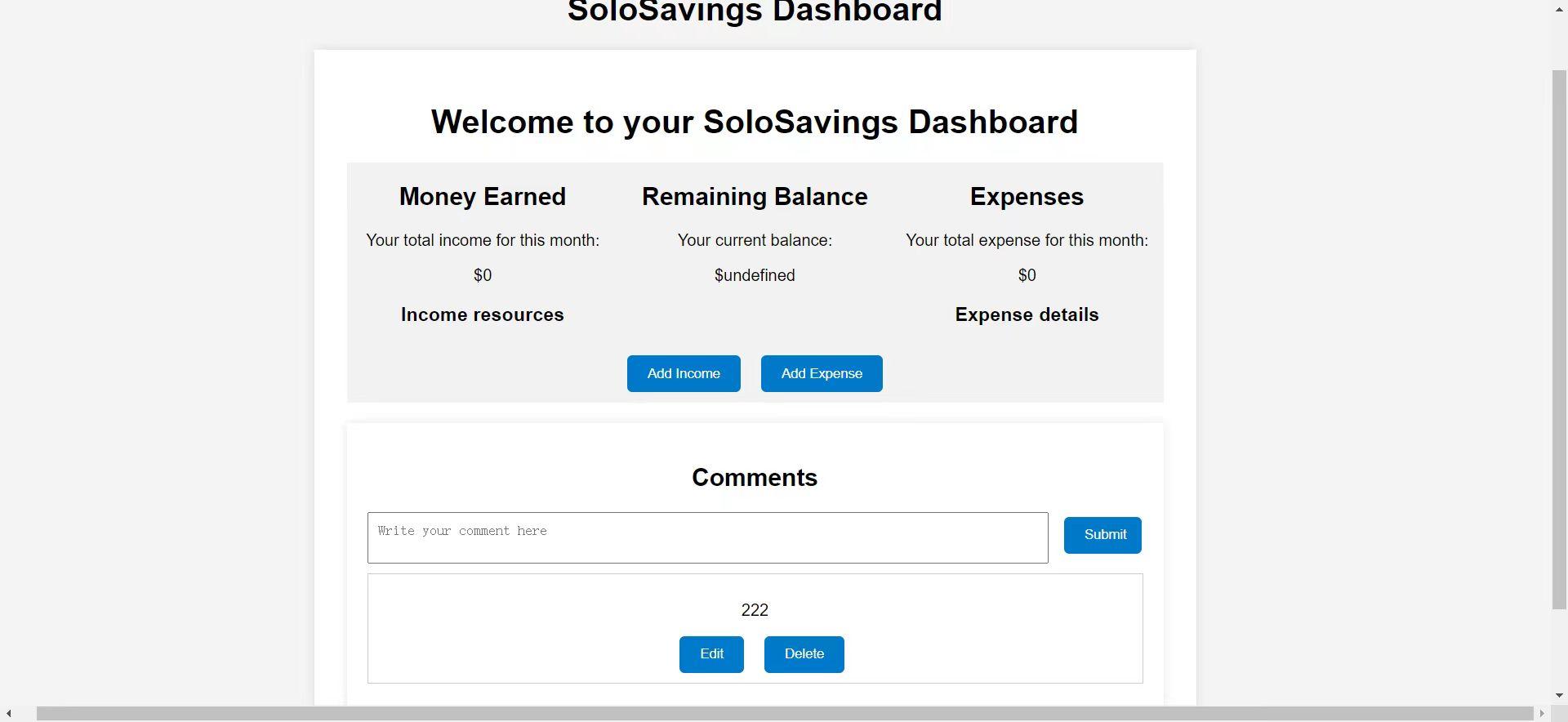
**Header:** Contains navigation links for "Home," "Login," and "Register." Users can click on these links to access different sections of the application.

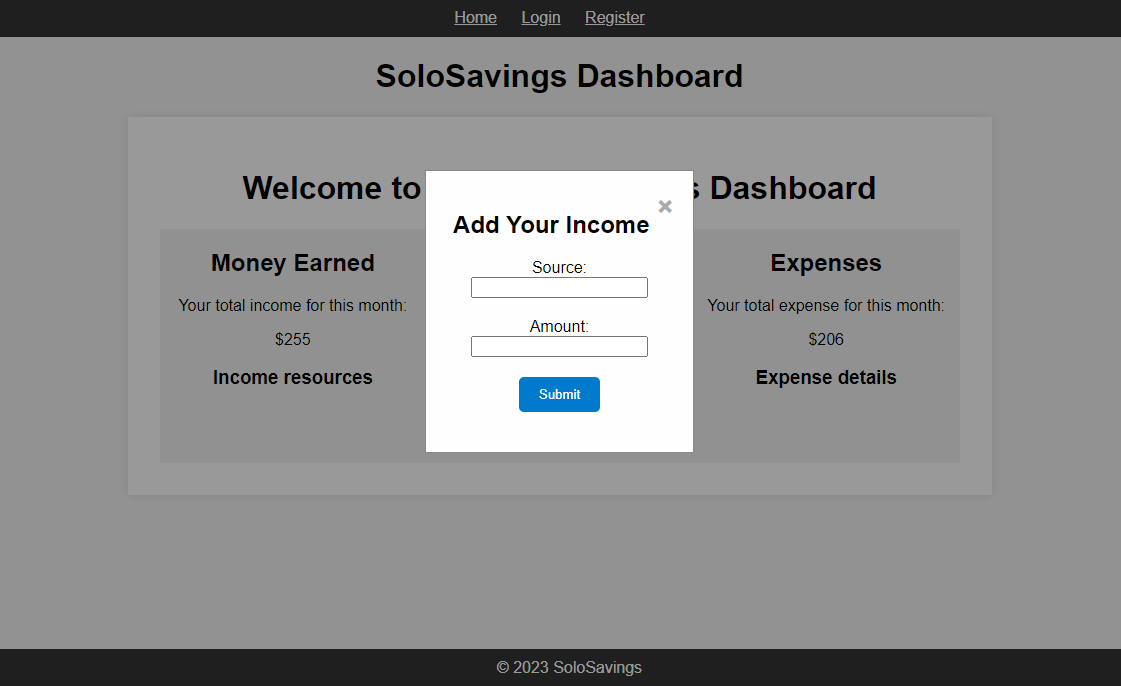
**Home Page:** The landing page with an introduction to SoloSavings, providing an overview of the application. Users can navigate to the "Login" and "Register" pages from here.  
  
  


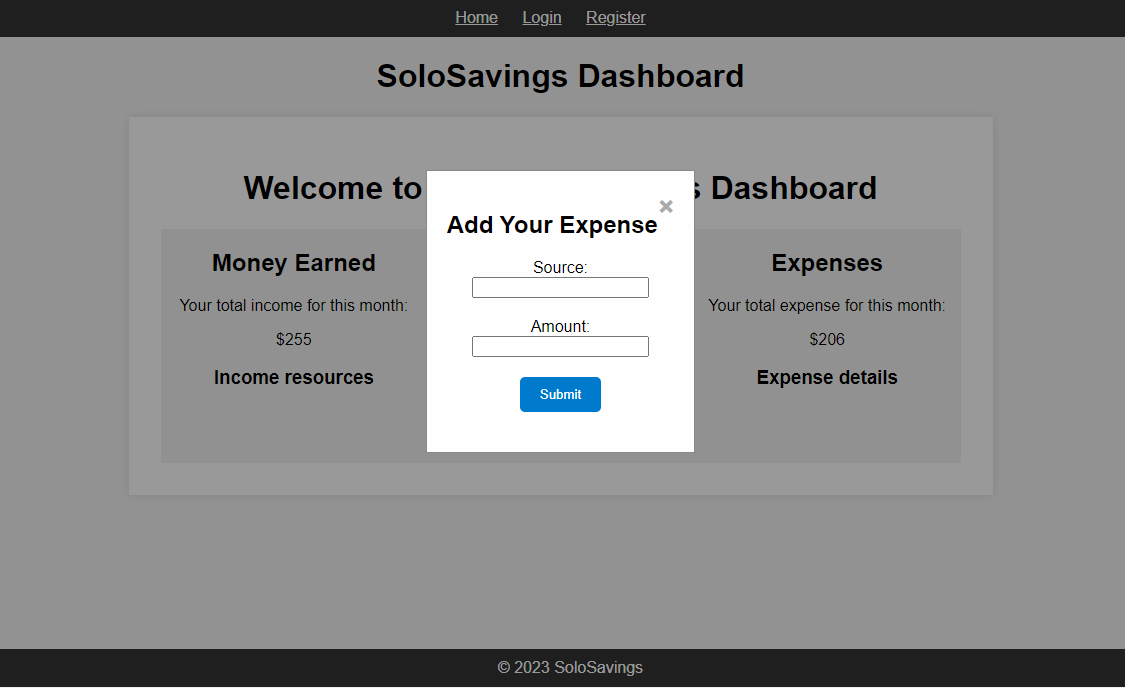
**This is the updated home page design**

****

**Register Page:** New users can create an account with SoloSavings by filling out a registration form. The form includes fields for personal information and account details. Users can also access  
  
  
**Login Page:** Allows registered users to log in by providing their username/email and password. Users can also access the "Register" page from here.  


**users are to be redirected to their respective dashboards after successfully logging in, with dashboard data fetched from a database.  
  
  
  
Updated ,add more feature and UI enhancement   
**

where users can **add income,** save it to the database, update the income amount in the database, and redirect to their respective dashboards

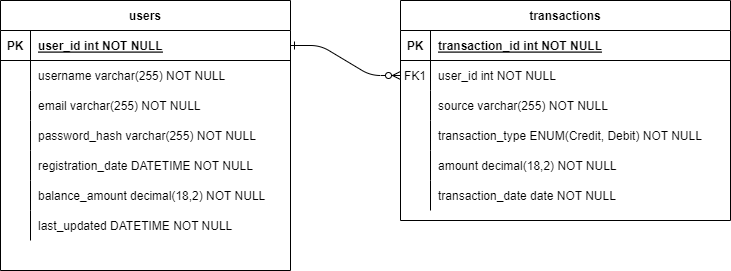
the functionality where users can **add expenses** and have it decrease their remaining balance  
and redirected to their respective dashboards

budget app to keep track of money earned and spent. We can add items from the list and the budget will always be updated.

**Log out** redirected to the Home Page  
 \* these are initial concepts that have not been implemented yet.

# Database Design (if applicable)

In this section, you shall describe any database schema if used in your software system.  
  
Below, we provide an overview of the database schema, including tables, relationships, and key attributes:  
  
Database Design for SoloSavings



**Entities:**

**1. Users table:**

Attributes:

user\_id (Primary Key): Unique identifier for each user.

username: User's chosen username.

password\_hash (or encrypted password): Securely stored password hash.

email: User's email address.

registration\_date: Date and time of user registration.

Balance\_amount: user’s ongoing balance based on transactions

Last\_updated: date and time user record updated

**2. Transaction table:**

Attributes:

transaction\_id (Primary Key): Unique identifier for each transaction record.

user\_id (Foreign Key referencing Users): Identifies the user associated with the income.

source: Source of transaction (e.g., salary, freelance work, electricity bill, cellphone bill).

Transaction\_type: CREDIT for Income/DEBIT for expenses

amount: Amount of transaction.

transaction\_date: Date of the transaction.

**Relationships*:***

Users can have multiple transaction records, and each transaction record belongs to a user. This is a one-to-many relationship.

**Explanation:**

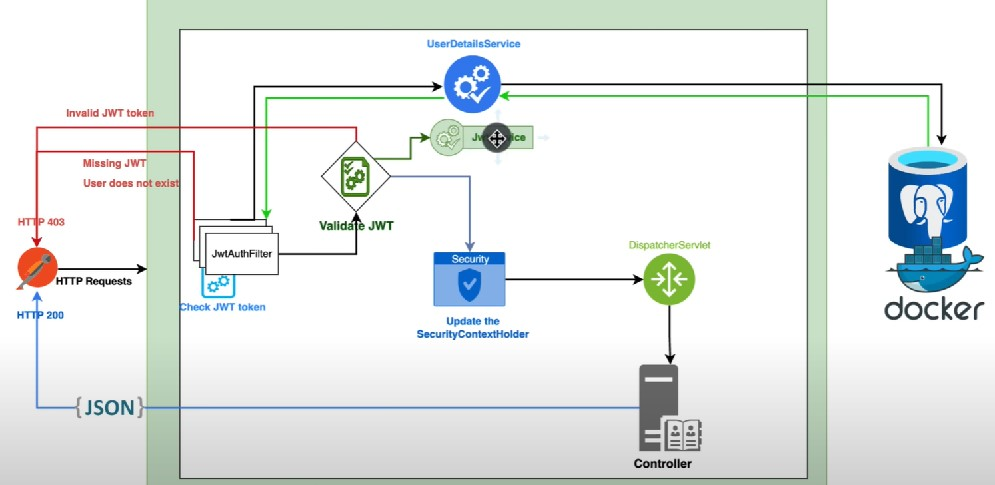
The "Users" table stores user account information, including login credentials and current balance.

The "transaction" table stores income-related transactions and expense-related transactions, including the source of transaction, amount, and date. It is related to the "Users" table through the user\_id foreign key.

This database design serves as a foundation for the "SoloSavings" application, facilitating the management of user accounts and transactions(income, expenses) to manage the user balance.

# Security Design

Designing a robust security system for a personal savings and budgeting app is critical to protect sensitive financial and personal data. It's important to prioritize the most critical and foundational security features.



The Security Architecture will be built upon the Spring Boot security framework leveraging JWT Tokens for registering and authentication.

First, Storing passwords safely in the database is crucial to prevent attackers from accessing them. SoloSavings will implement secure user authentication with strong password requirements by requiring a minimum of eight characters, at least one uppercase letter, one lowercase letter, one number and one special character. The application should check the password using a regular expression, "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$", when the potential user set up the account password with SoloSavings application. When storing the password, SoloSavings will not store passwords in plain text. SoloSavings will use a modern one way hashing algorithm to securely store passwords as hashing is a one-way function that cannot be decrypted. On top of hashing, SoloSavings will also implement the salting to the passwords in order to add an extra layer of security. A salt is a randomly generated string that is added to each password during the hashing process, which ensures that the hash is unique to each password.

When storing passwords, the salt is combined with the password and hashed, and both the hash and salt are stored in the database. When validating a password, the salt for the user is fetched from the database, and it is appended to the password provided by the user and hashed. The password structure would be like hashed(password + salt).

As a full stack application, the SoloSavings would implement API security by rendering information through HTTPS encrypted secure data transmission for all CRUD operations, implement cross-origin resource sharing an rate limiting and throttling to protect against abuse, and implement input validation to prevent common security vulnerabilities like SQL injection.

Moreover, we need to ensure code quality, and Github Security offers code scanning tools which we should take advantage of. Before implementing any third-party library, we should do in depth research to make sure the dependencies are reliable and well maintained.

Finally, error handling logging will be implemented in SoloSavings to log all errors into the log database, warn the user when an error is encountered, and monitor any suspicious activities in the application.

While cognizing that not all security features may be feasible in the given timeframe, here are some features that are nice to have but not as the priority to implement:

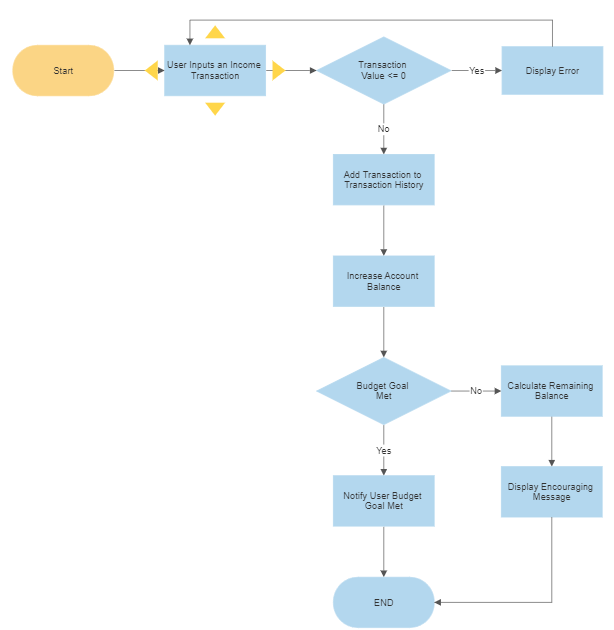
* Session ID generation and timeout management
* Two-factor authentication SSP / OAuth for added security
* Authorization access control - no special user permission or admin needed
* Soft deletion - the application currently does not offer data backup function
* Change the password and forget the password functionalities

# Business Logic and/or Key Algorithms

“Business logic refers to the logic and algorithms serving as the foundation of code in business software.” Put another way, business logic is real-world business rules put into computer code. Some of the initial business logic defined by the SoloSavings key stakeholders are illustrated below for transaction (income/expense) entries:

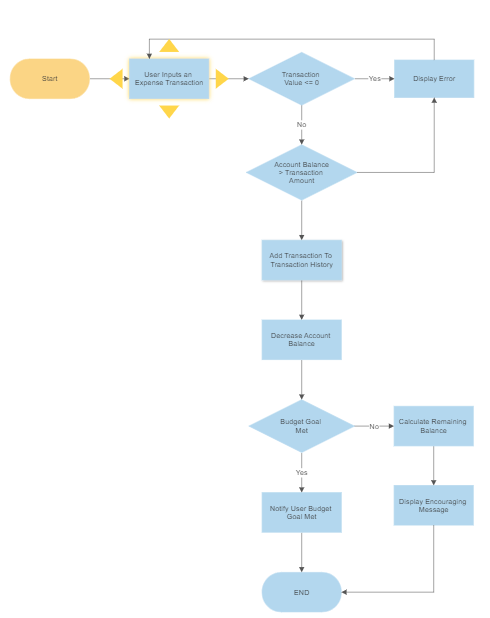
**Income**

* 1. If a transaction value is zero or negative, the transaction should be flagged for the user to review the transaction before saving.
  2. If an income is entered, then the transaction should be stored and the account balance should be increased.
  3. When a transaction is entered, the account balance is checked to see if the budget goal is met,
  4. If the account balance does not hits the budget goal, the user is notified, else the remaining balance is calculated and an encouraging message is displayed



**Expense**

* 1. If a transaction value is zero or negative, the transaction should be flagged for the user to review the transaction before saving.
  2. If an expense is entered, then the account balance should be checked to ensure that the expense can be covered, transaction should be stored and the account balance should be decreased.
  3. When a transaction is entered, the account balance is checked to see if the budget goal is met,
  4. If the account balance does not hits the budget goal, the user is notified, else the remaining balance is calculated and an encouraging message is displayed



# Design Patterns

In this section, you shall describe any design patterns used in your software system.  
  
**Design Pattern: MVC-R (Model-View-Controller with Repository)**

**Intent:**

The MVC-R design pattern, an extension of the classic Model-View-Controller (MVC) pattern, introduces a Repository layer to decouple data access from user interface and business logic in the SoloSavings application. The primary aim is to establish a well-structured, maintainable, and scalable architectural foundation.

**Motivation:**

The SoloSavings application requires an architectural pattern that distinguishes data access concerns from user interfaces and business logic. The MVC-R pattern addresses this by incorporating a Repository layer.

**Components:**

**Model (M):**

* Represents the application's data and business rules.
* Comprises classes like User, Income, Expense, and Balance to model various aspects of financial data.
* Enforces data integrity and encapsulates the application's state.

**View (V):**

* Manages presentation and user interface aspects.
* Leverages React for frontend development, ensuring dynamic and responsive user interfaces.
* Responsible for rendering data from the Model and capturing user input.

**Controller (C):**

* Serves as a mediator between the Model and the View.
* Contains classes like UserAuthenticationController and IncomeAndExpenseController to handle user interactions.
* Processes user input, updates the Model, and controls application flow.

**Repository (R):**

* Focuses on data access and storage.
* Comprises UserRepository, IncomeRepository, ExpenseRepository, and BalanceRepository to interact with corresponding data entities.
* Segregates database interactions from the Controller and Model, fostering a cleaner architecture.

**Collaboration:**

* The View interacts with the Controller to capture user input and present data.
* The Controller communicates with the Model for data retrieval and updates.
* The Controller engages with the Repository for data persistence.
* The Repository manages data access, including database queries and updates.

**Benefits**:

* **Decoupled Concerns:** MVC-R decouples data access concerns into the Repository layer, enhancing modularity and maintainability.
* **Scalability**: By separating components clearly, it facilitates scalability for adding features or modifying existing ones.
* **Testability**: Independent testing of each component simplifies unit testing and debugging.
* **Code Reusability**: Well-encapsulated components can be reused within the application or future projects.

**Considerations**:

Define component responsibilities meticulously to prevent violating the Single Responsibility Principle.

Ensure robust error handling and validation at each layer to uphold data integrity.

Employ version control and continuous integration tools to support collaborative development.

**Applicability**:

MVC-R suits SoloSavings, which demands distinct separation of concerns, data persistence, and a responsive user interface.

**Example:**

When a user registers in SoloSavings, the React View captures registration details. The UserAuthenticationController processes the request, validates input, and coordinates with the UserRepository to store new user data in the database.

# 

# References Baeldung. "Java Password Hashing." [Link](https://www.baeldung.com/java-password-hashing)

# Java Interview Point. "Java Salted Password Hashing." [Link](https://www.javainterviewpoint.com/java-salted-password-hashing/)

# OpenAI. "ChatGPT."

# Head First Design Patterns (Book).

# Investopedia. "Business Logic." [Link](https://www.investopedia.com/terms/b/businesslogic.asp#:~:text=Business%20logic%20is%20the%20custom,constrains%20how%20a%20business%20operates.)

# Integrify. "What Is Business Logic?" [Link](https://www.integrify.com/what-is-business-logic/#:~:text=Many%20financial%20organizations%20rely%20on,you%20with%20a%20final%20total.)

# "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

# "Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects" by Douglas C. Schmidt and Michael Stal.

# Glossary

**SoloSavings**: The name of the budget web application we will develop.

* **Maven Project:** A software project management and comprehension tool used for building and managing Java-based projects.
* J**ava Spring Boot:** A framework for building Java-based web applications, providing features like RESTful APIs and security.
* **User Model:** A component of the software that manages user account information and authentication.
* **Financial Model:** A component responsible for handling financial logic, income, expenses, and balance calculations.
* **MySQL:** A relational database management system used to store user data, financial records, and other application data.
* **Hashing:** A one-way cryptographic process used to securely store passwords.
* **Salt:** A randomly generated string used in password hashing to add an extra layer of security.
* **HTTPS:** Hypertext Transfer Protocol Secure, a secure version of HTTP used for encrypted data transmission.
* **Cross-Origin Resource Sharing (CORS):** A security feature implemented to control which web domains can access resources on a web page.
* **Rate Limiting and Throttling:** Techniques used to limit the number of requests a user or system can make to an API within a specified time frame.
* **Session Management:** The management of user sessions, including session ID generation and session timeout settings.
* **Two-Factor Authentication (2FA)**: An additional layer of security that requires users to provide two different authentication factors before gaining access.