Finance Tracker - Design Document

Table of Contents

Overview

The finance tracker is a web app meant for users to track their income and expenses in various ways. Users will be able to log in, and have a host of options to choose from when it comes to selecting how to manage their income, expenses, and budgets. The main features include investment tracking, automation of income and expenses, planning of budgets, and manual adding of income and expenses. The technologies that will be used are Spring Boot, Spring Security, Thymeleaf, Lombok, JPA, and MySQL.

Architecture

- Backend: Spring Boot
- Frontend: Thymeleaf, Bootstrap
- Database: MySQL, JPA
- Authentication: Spring Security
- Code Efficiency: Lombok

These technologies will be used for the creation of the web application, as they work well together, and are often used together and still supported.

Application Layers

- Controller Layer: Handles HTTP requests and serves information from the database to pages using Thymeleaf
- Service Layer: Handles business logic and things that should not be directly handled by the controller layer.
- Repository Layer: Used for data access
- Database: Localhost with MySQL

These are the four main application layers for this project. Other things that should be taken note of that aren't necessarily application layers are the models, which are templates for how the Users, Transactions, etc. will be stored in the database. There is also a security configuration, which is self-explanatory. Finally, there are a few enumerations for lists of selections.

Transaction

Transactions are comprised of an ID (Long), type (TransactionType enum), category (Category enum), tag (String), amount (Double), date (LocalDate), and user (User). Transactions are joined to the user by user ID and is a many-to-one relationship since transactions are tied to a single user, but a user can have many different transactions.

AutomatedTransaction

AutomatedTransactions are comprised of an ID (Long), type (TransactionType enum), category (Category enum), amount (Double), startDate (LocalDate), frequency (RecurrenceFrequency enum), and user (User). AutomatedTransactions are joined to the user by user ID and is a many-to-one relationship since AutomatedTransactions are tied to a single user, but a user can have many different AutomatedTransactions.

Budget

Budgets are comprised of an ID (Long), category (Category enum), amount (Double), amountSpent (Double), and user (User). Budgets are joined to the user by user ID and is a many-to-one relationship since Budgets are tied to a single user, but a user can have many different Budgets.

Investment

Investments are comprised of an ID (Long), name (String), category (InvestmentCategory enum), originalAmount (Double), currentValue (Double), and user (User). Investments are joined to the user by user ID and is a many-to-one relationship since Investments are tied to a single user, but a user can have many different Investments.

Transaction

Transactions are comprised of an ID (Long), type (TransactionType enum), category (Category enum), tag (String), amount (Double), date (LocalDate), and user (User). Transactions are joined to the user by user ID and is a many-to-one relationship since Transactions are tied to a single user, but a user can have many different Transactions.

User

Users are comprised of an ID (Long), username (String), password (String), transactions (List<Transaction>), investments (List<Investment>), automatedTransactions (List<AutomatedTransactions>), and budgets (List<Budget>). A user is connected to these for things, so that they're specific to a user, and they use the one-to-many relationship since a user can have multiple of these, but not the other way around.

Potential Things To Add In Future

- Email reminders for investments, automatic transactions, and getting close to a budget limit.
- Export functionality for getting investments as a portfolio.
- Merge Transaction and AutomaticTransaction into one and extend.
- Charts for other aspects other than just Income Vs. Expenses over time.