Project Design Documentation:

Team Information:

Team Name:

- EpicMaven - 5E

Team Members:

- Alex Hamadeh - afh5776
- Yousaf Iqbal - yyi5708
- Red Ziogas - ajz1369
- Kevin Huang - kxh8353
- Daniel Panettieri - dmp1452

Executive Summary:

The project focuses on developing a software solution simulating a u-fund structure, catering to admins and helpers. Our team, EpicMaven, is committed to delivering a robust application with features like authentication, user-specific functionalities, and data security. The Minimum Viable Product (MVP) includes login/authentication, helper features for managing needs and funding, and managerial capabilities for data management. We follow the MVVM architecture, adhere to Object-Oriented Design Principles, and conduct rigorous testing for functionality and code coverage. Our ongoing rationale involves regular assessments for progress, addressing challenges, and planning for future enhancements, aiming for a user-focused, reliable, and secure system.

Purpose:

The project aims to develop a software solution that emulates a u-fund structure, catering primarily to two user groups: managers and helpers. The most important user group is the helpers, whose goal is to assist users by managing their needs and funding baskets efficiently. Helpers need functionalities such as searching for needs, adding/removing needs from funding baskets, and accessing detailed need information to fulfill their responsibilities effectively within the system.

Requirements:

Authentication and Roles:

- Implement a secure login/authentication system for users, helpers, and managers. Define distinct roles and permissions for each user type to access relevant functionalities.

Helper Functionalities:

- Provide helpers with the ability to manage a list of needs.
- Enable helpers to search for specific needs, add/remove needs from their funding basket, and view detailed need information.

Manager Functionalities:

- Empower managers to add, remove, and edit data related to all needs stored in the system.
- Restrict manager access to the funding basket, focusing on data management and oversight capabilities.

Definition of MVP:

- User login / logout
- User authentication
- Ufund persistence: user data is saved when the user logs out, closes, or refreshes the tab
- client should see a list of needs in their basket
- client should be able to add and remove needs from their basket
- client should be able to checkout needs from their basket
- admin should be able to add or remove a need from the cupboard
- admin should be able to edit information about a need in the cupboard

MVP Features:

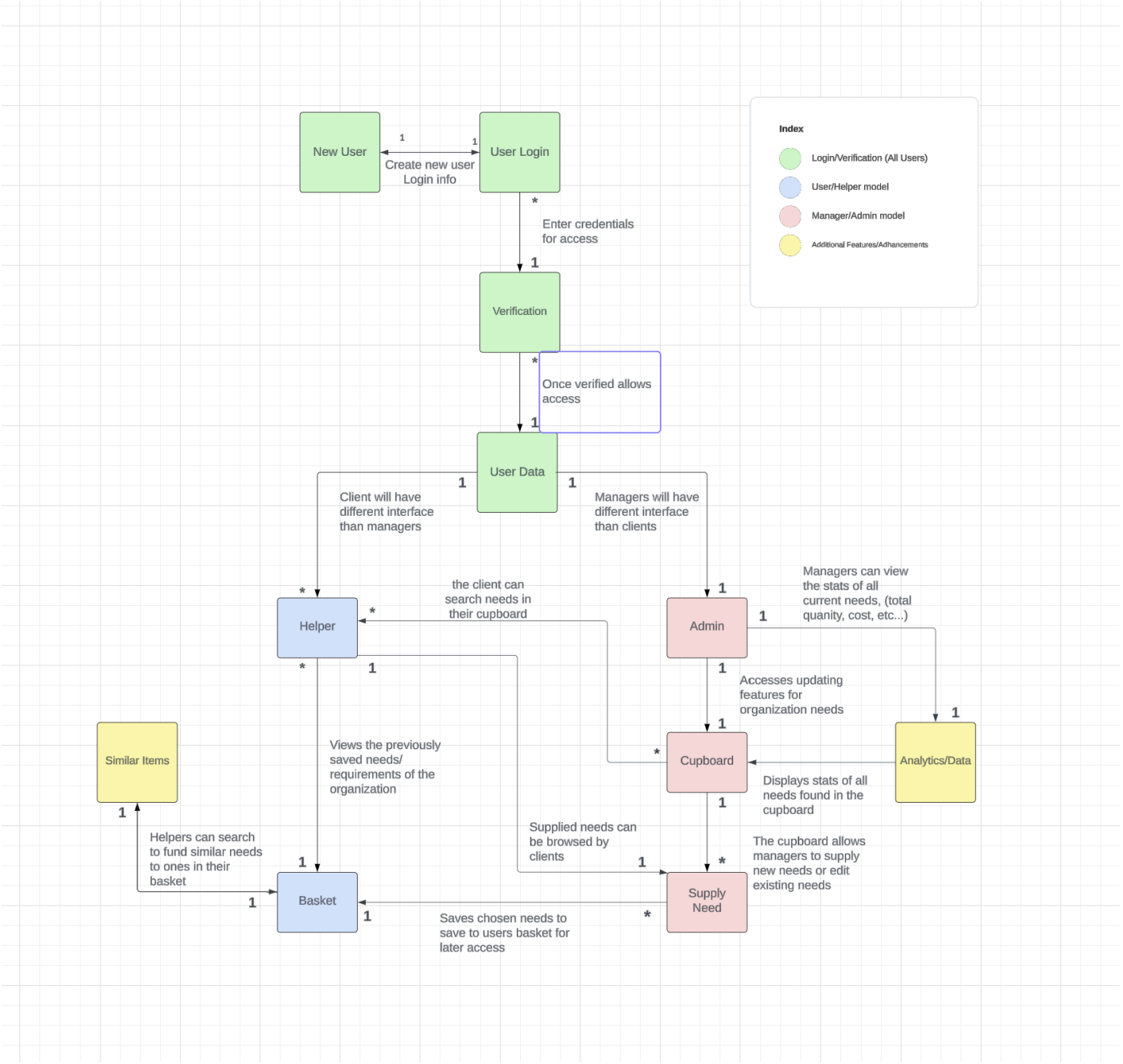
- As a client, I want to view the list of needs so I can choose where to donate my money
- As an admin, I want to add new needs so that users can see what is available
- As an admin, I want to modify the details of a need so that users can be presented with the most up-to-date information

Enhancements:

- Analytics component for the admin where they can see statistics and recent activity of their Ufund store
- Grouping of needs: when a client adds an item into their basket, they have an option to view and add similar items to their basket under the same category.

Application Domain:

Domain Model:

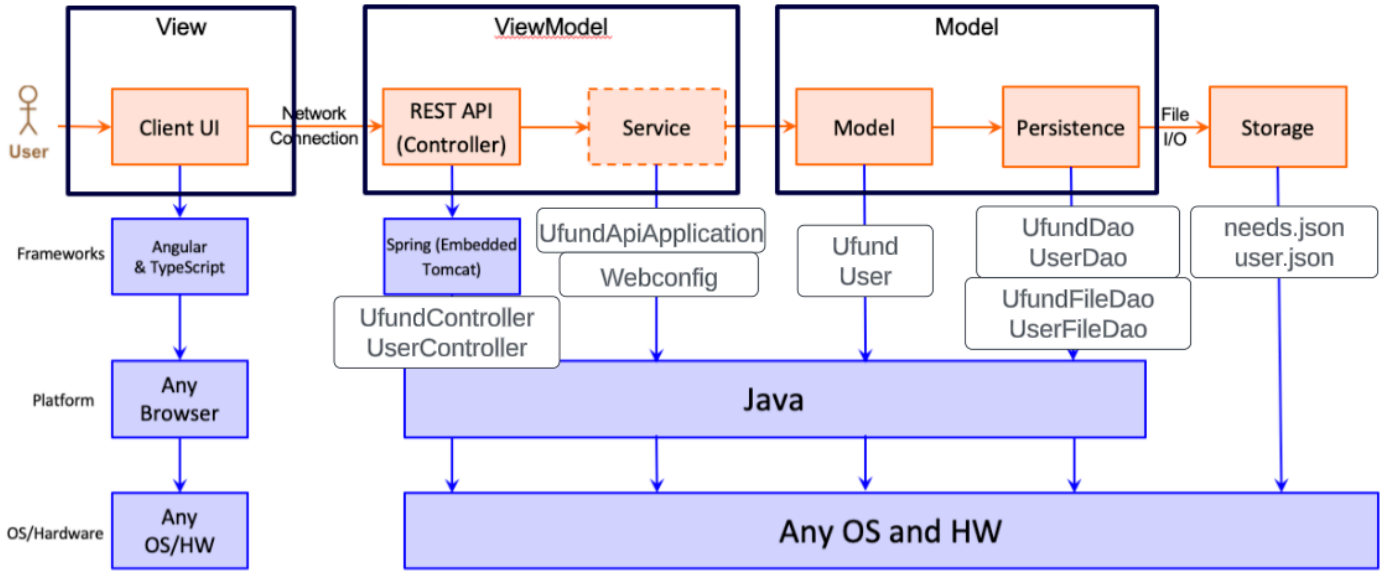


Architecture and Design:

The web application, is built using the Model–View–ViewModel (MVVM) architecture pattern. The Model stores the application data objects including any functionality to provide persistence. The View is the client-side SPA built with Angular utilizing HTML, CSS and TypeScript. The ViewModel provides RESTful APIs to the client (View) as well as any logic required to manipulate the data objects from the Model. Both the ViewModel and Model are built using Java and Spring Framework. Details of the components within these tiers are supplied below.

Summary:

The following Tiers/Layers model shows a high-level view of the webapp's architecture:



Overview of User Interface:

The user interface of the web application is designed to provide a seamless and intuitive experience for users, helpers, and managers, allowing them to interact efficiently with the system and perform their respective tasks. Here is a summary of the application's user interface flow from the user's perspective:

- Login Page (Admin/Helper):

> Users enter their credentials (username/email and password) for authentication. Upon successful login, users are directed to their personalized dashboards based on their roles.

- Dashboard (Helper)

> Clients have dashboards displaying a list of bars below the homepage where the client can have access to different options, advantages, and luxury services. In the UI main page, there is a clock displaying the date and time as well as a calendar followed by a calculator where the user can perform calculations such as adding up prices, calculating percentages, and maximizing profit.

- Basket (Helper)

> A client has access to a search bar where they can search a need by name, category, or description. As the client searches for the need, hints and recommendations pop up. The client can also press the show all needs button

> where the client will be able to see all the needs in the cupboard. After the client is done adding items into their basket, the client can press the checkout button which navigates into a checkout page where the client can enter payment information, or if the client changes their mind, they can press the exit button anytime to take them back to the main page.

- Account Bar (Helper)

> A client has the advantage to delete their account for personal or security reasons. Once the client clicks the tab, the client is directed into a delete account page where the client is provided with enter email and password bars, followed by a delete account button where the client clicks to permanently remove their account. After that button is pressed, the no-longer member is taken back into the login page.

- About Us Page (Helper)

> When the client is curious about the contributors to this software, the client can click the "about" tab and the client can see the names and pictures of the 5 contributors of this software as well as a description of everyone's role in this project.

- Dashboard (Admin)

> the admin has a dashboard displaying a list of bars below the homepage where the admin can have access to the cupboard, analytics, and other customer service features. In the UI main page, there is a clock displaying the date and time as well as a calendar followed by a calculator where the admin can perform calculations such as adding up prices, calculating percentages, and maximizing profit.

- Cupboard (Admin)

> The admin can create a new need for clients where the admin can assign a name, cost, quantity, and the type of need. The admin will see add button where the need will be placed in the cupboard and show up in the list of needs in the top of the page. The admin can press the exit button which takes the admin back to the main page.

- Analytics (Admin)

> Once the tab is clicked, the admin can see statistics of Ufund, including the total number of needs, cumulative cost and quantity, and total types of needs. An exit button will take the admin back into the main page.

- About us Page (Admin)

> When the admin is curious about the contributors to this software, the admin can click the "about" tab and the admin can see the names and pictures of the 5 contributors of this software as well as a description of everyone's role in this project.

- Exit (Admin/Helper)

> while on the main page, if the client or admin finished their business and have no more tasks on Ufund, they can click the exit button which will land them back onto the login page.

- Responsive Design:

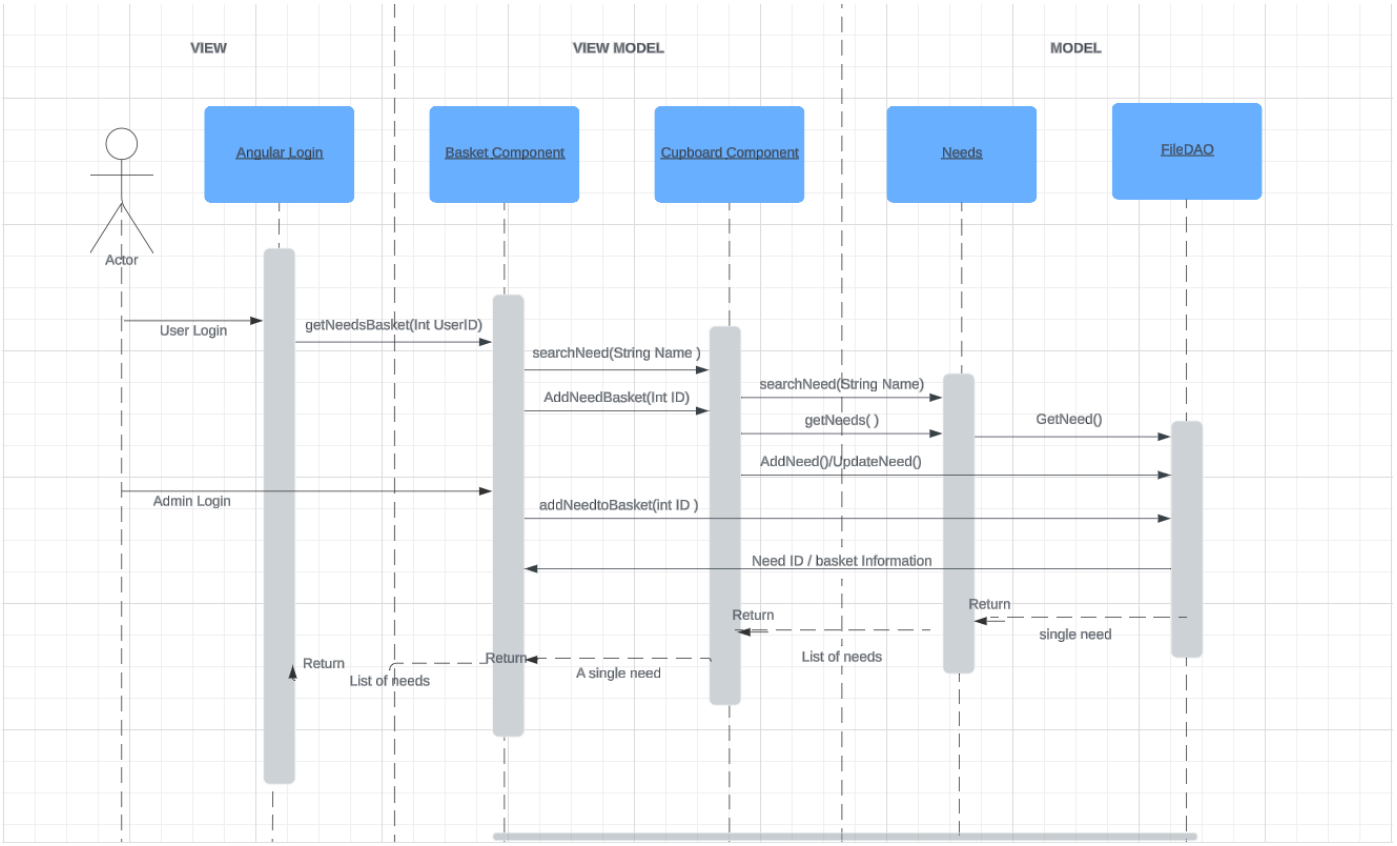
> The user interface is responsive, ensuring compatibility and optimal display across various devices and screen sizes. UI elements like forms, buttons, and navigation components are user-friendly and visually appealing.

View Tier:

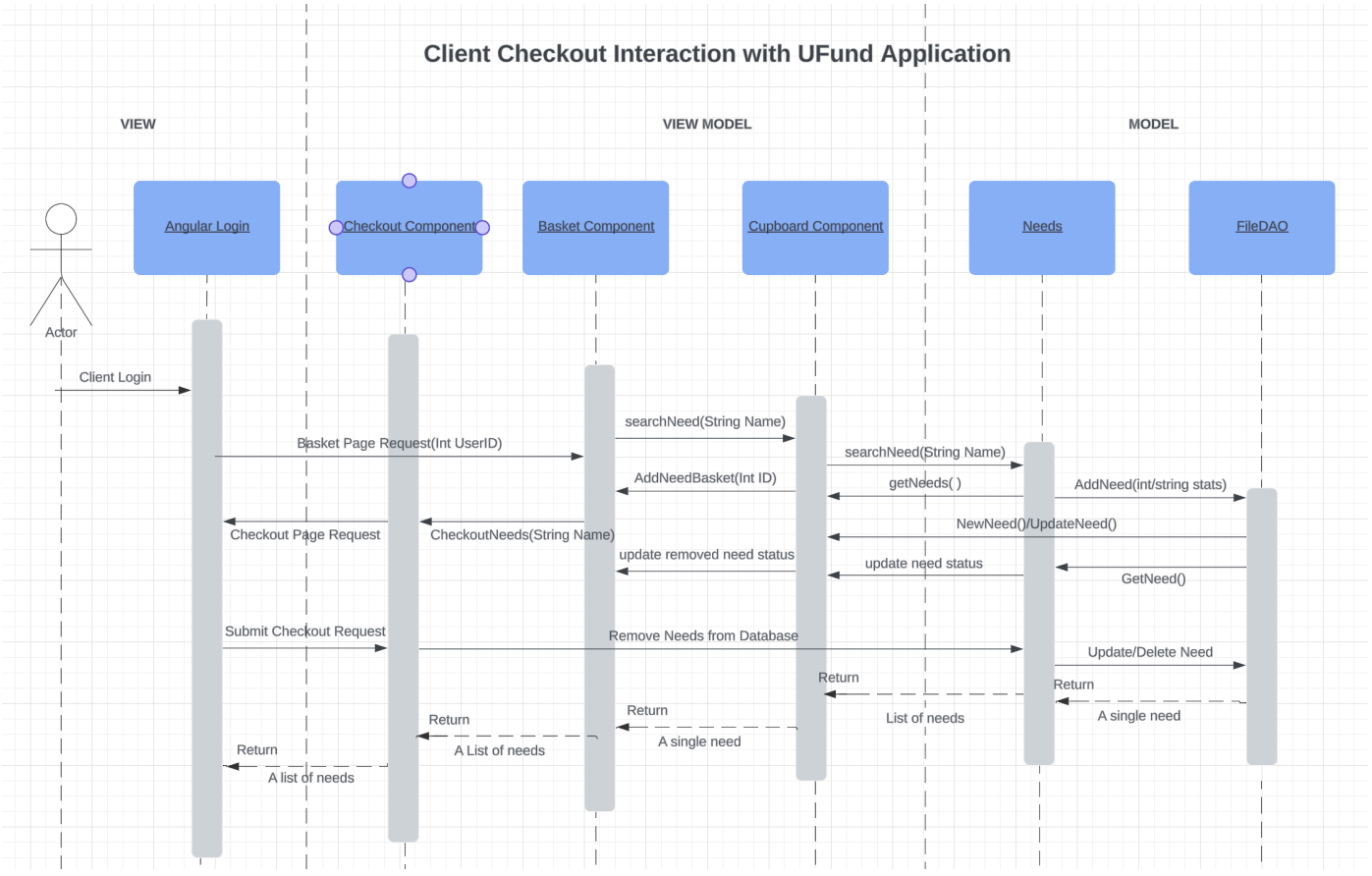
Front End:

- TS files: enable front end communication with back end components.
- HTML files: calls .ts components to translate functionality into interface.
- CSS files: independent from ts and HTML; sets dimensions and color for UI.

Sequence Diagram #1:



Sequence Diagram #2:



ViewModel Tier:

- UfundController.java: handle api requests and provide responses.

- UserController.java: handle api requests and provide responses.
- UfundAPIApplication.java: provides framework for ufund api.
- UserAPIApplication.java: provides framework for user api.
- WebConfig.java: used for mapping.

Model Tier:

- Ufund.java: The purpose of ufund.java is to create the ufund object, which serves as the need.
- User.java: The purpose of user.java is to create the user object, which serves as the login.
- UfundDAO.java: An interface that declares the methods for changing the list of needs.
- UserDAO.java: An interface that declares the methods for changing the list of logins.
- UfundDAOFile.java: Implements UfundDAO.java and implements the methods necessary for changing the needs.
- UserDAOFile.java: Implements UserDAO.java and implements the methods necessary for changing the logins.

OO Design Principles:

The initial OO Principle that we as a team have considered in support of our design and implementation for this first sprint is Single Responsibility:

- Such that, the ufund controller is an example of it and its purpose is to handle api requests and provide responses using HTPP Protocols.
- It is not concerned with the management of the needs data or even the underlying storage mechanism, which the responsibility is delegated to the ufund dao class.
- The same explanation applies to the user controller, etc.

In addition to the Single Responsibility Principle, we also aim to incorporate the following OO Design Principles into our system:

- Open/Closed Principle (OCP): Ensuring that our software entities are open for extension but closed for modification. This allows for future enhancements without altering existing code.

- Low Coupling: We strive to minimize dependencies between modules or classes to improve maintainability and flexibility in our system.
- Information Expert: Assigning responsibilities to classes that have the most information required to fulfill those responsibilities, promoting better encapsulation and cohesion.
- Dependency Inversion Principle (DIP): We aim to depend upon abstractions rather than concrete implementations to reduce coupling and promote flexibility in our system's architecture.

These principles collectively guide our design decisions to create a well-structured, maintainable, and scalable software system.

Our design adheres well to fundamental object-oriented design principles, ensuring maintainability and flexibility. To enhance it further, we'll focus on improving interface cohesion, reducing dependencies, and introducing more abstractions where needed. Regular reviews will help ensure continued alignment with best practices.

Static Code Analysis/Future Design Improvements:

1. Null Pointer Dereference:
    - Analysis: This occurs when the code attempts to access an object property or method through a null reference, leading to a runtime exception.
    - Recommendation: Use defensive programming techniques such as null checks or the null object pattern to handle null references safely.
2. Code Duplication:
    - Analysis: Duplication of code segments increases maintenance overhead and introduces the risk of inconsistencies.
    - Recommendation: Refactor duplicated code into reusable functions or classes to promote code reusability and maintainability.
3. Resource Leak:
    - Analysis: The code fails to release acquired resources properly, such as file handles or database connections, leading to potential resource leaks and system instability.

    - Recommendation: Ensure that acquired resources are released using appropriate mechanisms, such as `try-with-resources` blocks or explicit resource cleanup routines, to prevent resource leaks and improve application reliability.

Discuss future refactoring and other design improvements your team would explore if the team had additional time.

1. Modularization:
    - Break down monolithic components into smaller, more manageable modules to improve code organization and scalability.
2. Dependency Injection:
    - Introduce dependency injection to decouple components and improve testability, maintainability, and flexibility.
3. Performance Optimization:
    - Identify and optimize performance bottlenecks, such as inefficient algorithms or database queries, to enhance system responsiveness.
4. Error Handling and Logging:
    - Enhance error handling mechanisms to gracefully handle exceptions and provide meaningful error messages to users. Implement robust logging to facilitate debugging and troubleshooting.
5. Code Documentation:
    - Improve code documentation to make the codebase more understandable and maintainable, including inline comments and API documentation.

Testing:

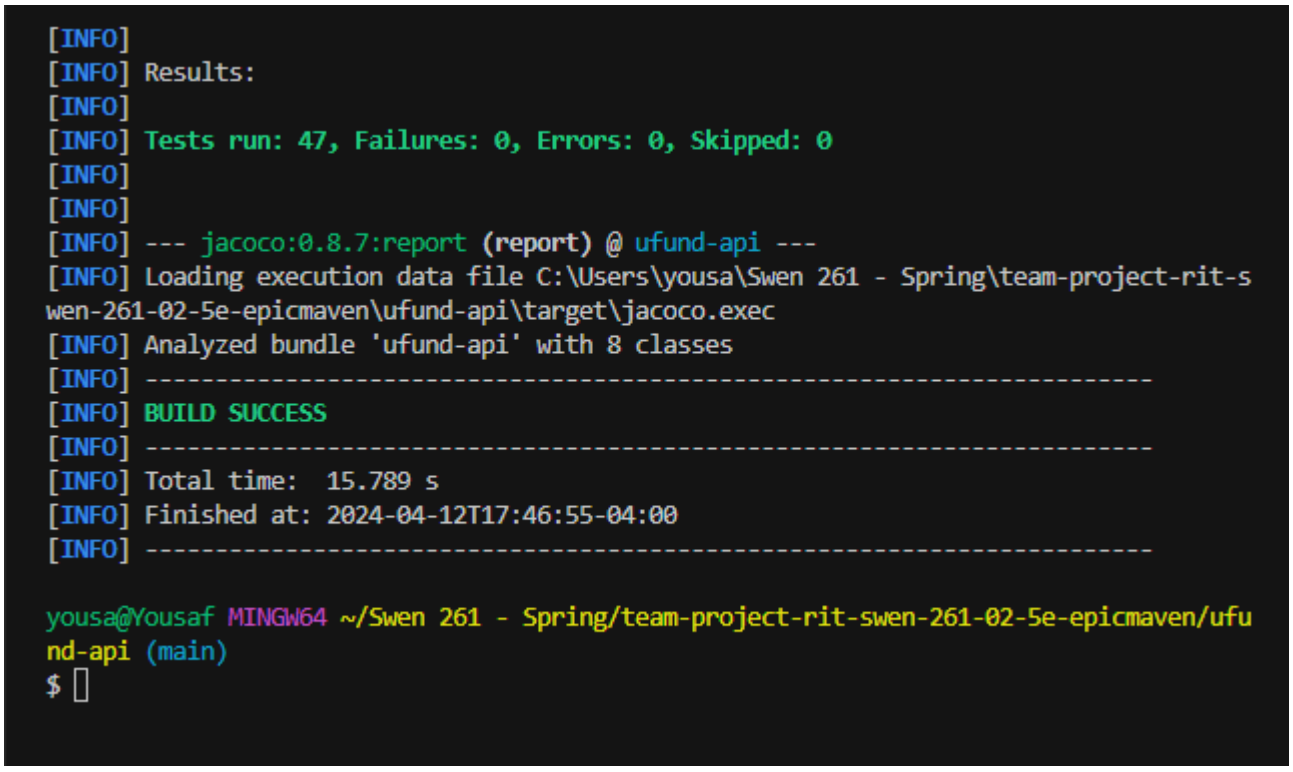
Acceptance Testing:

Report on the number of user stories that have passed all their acceptance criteria tests, the number that have some acceptance criteria tests failing, and the number of user stories that have not had any testing yet. Highlight the issues found during acceptance testing and if there are any concerns.

- Number of tests run: 47
- Number of tests passed: 47

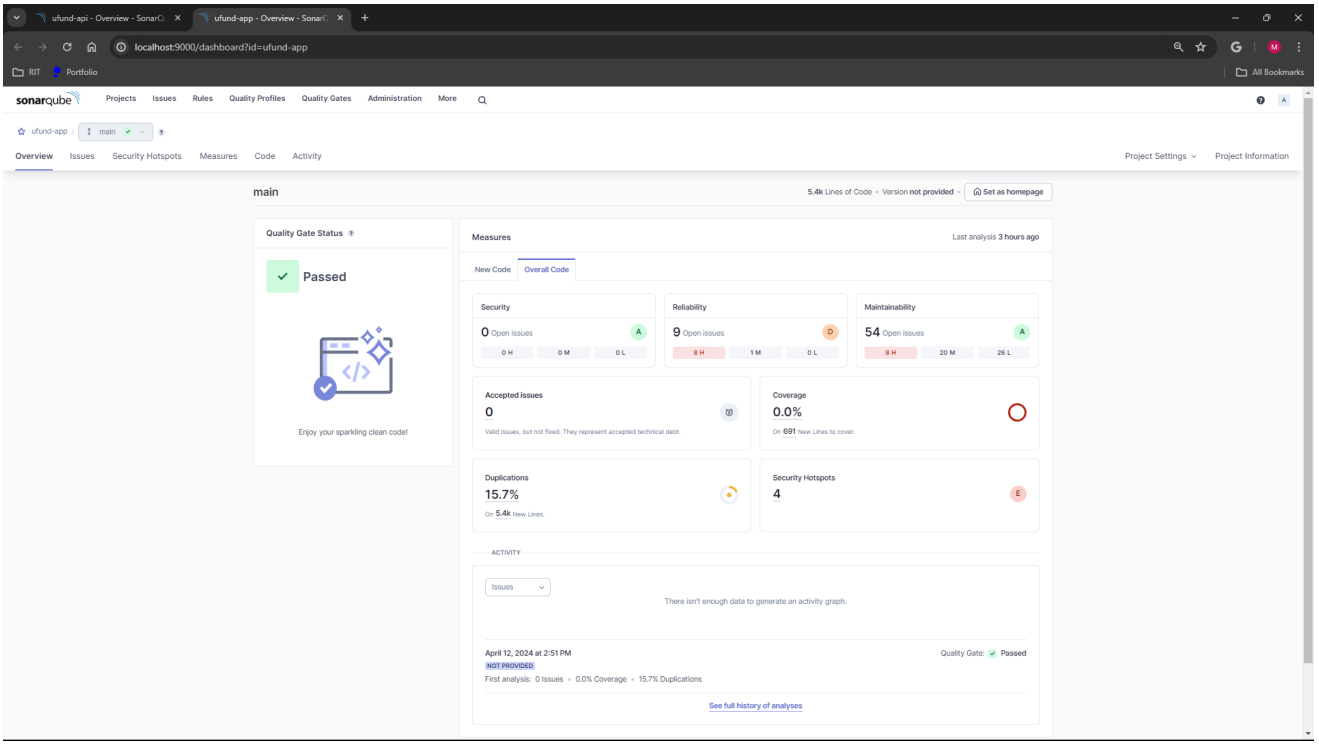
Tests that yet to pass:

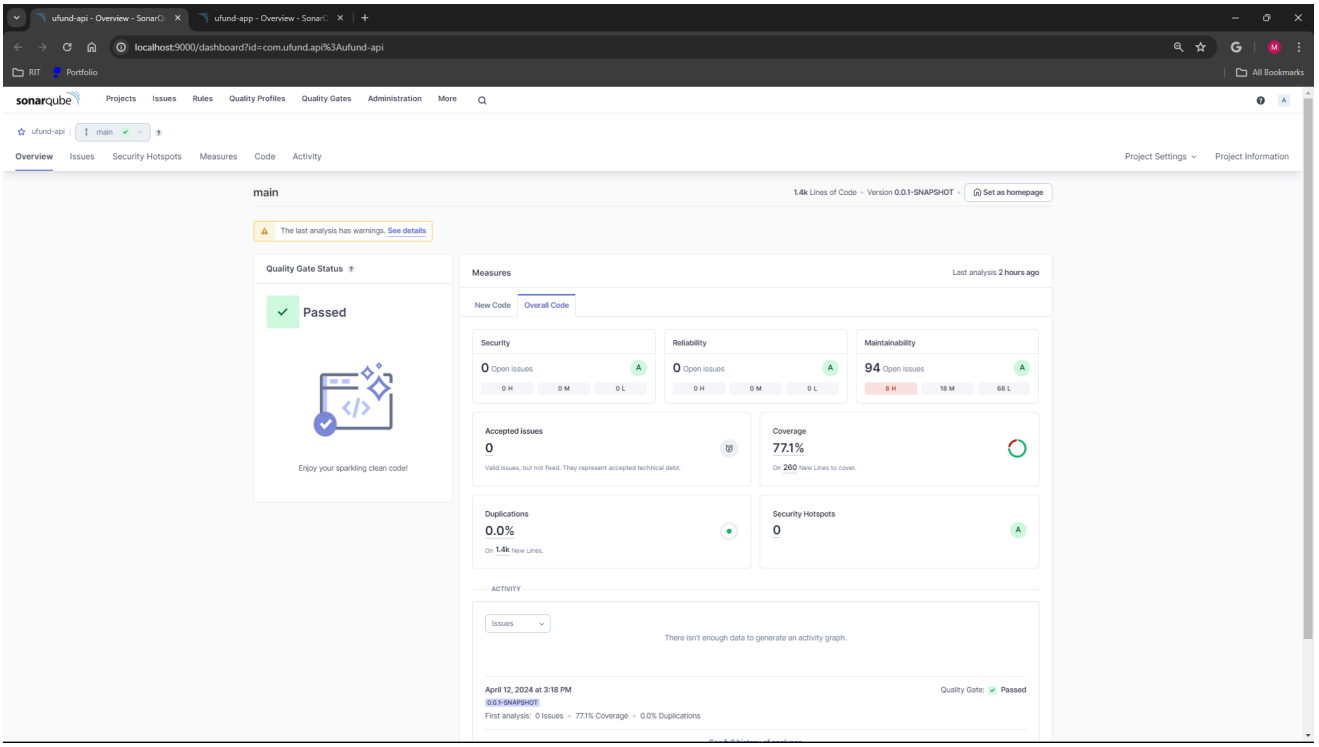
- testFindUser: UserFileDAOTest; problem: difference in ID.
- testGetUser: UserFileDAOTest; problem: difference in ID.
- testCreateUser: UserFileDAOTest; problem: null.
- testFindUsers: UserFileDAOTest; problem: difference in ID.
- testCreateNeed (worked at one point)): UfundFileDAOTest; problem: Null pointer exception.
- testSaceException (worked at one point): UfundFileDAOTest; problem: null pointer exception.

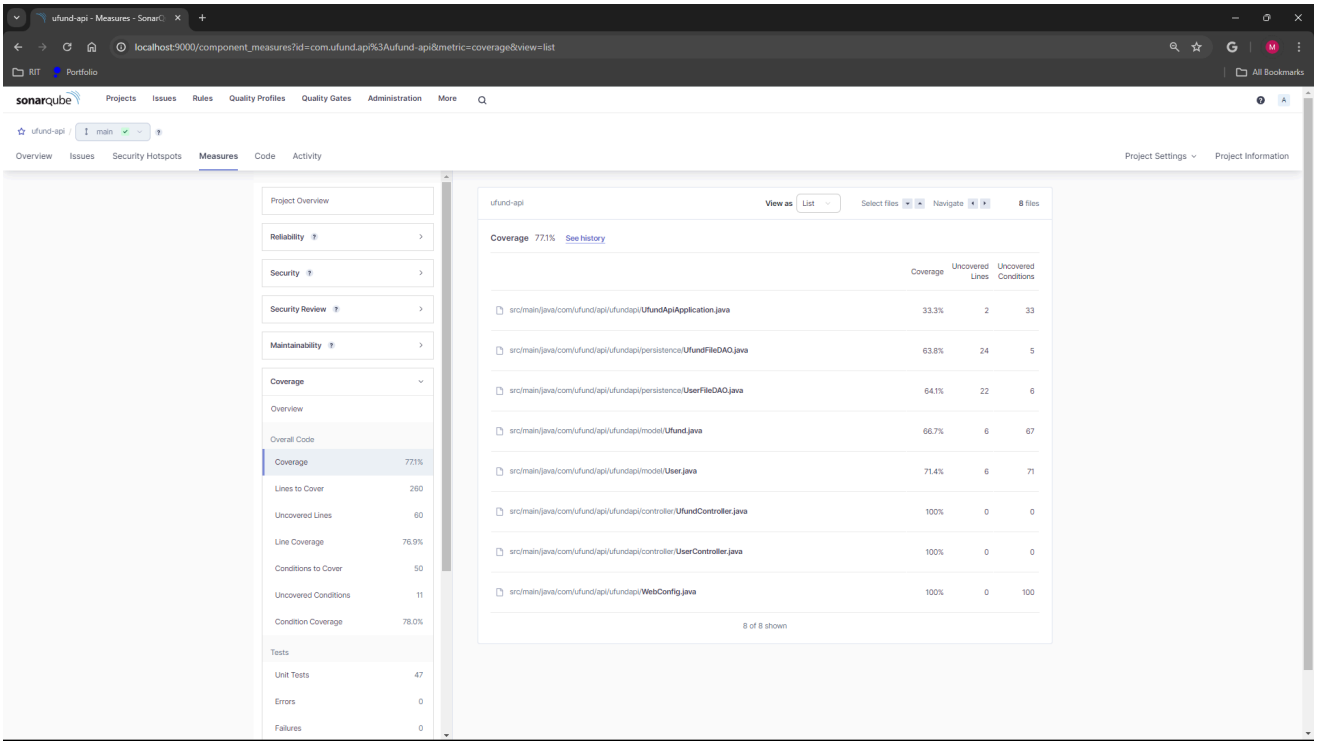


Unit Testing and Code Coverage:

- Coverage Targets: Ufund Controller, User Controller, Ufund model, User model, UfundFileDAO, UserFileDAO.


ufund-api - Overview - SonarQ
ufund-app - Overview - SonarQ
localhost:9000/dashboard?id=ufund-app
RIT
Portfolio
All Bookmarks
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
ufund-app
main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
main
5.4k Lines of Code · Version not provided ·
Set as homepage
Quality Gate Status
Passed
Enjoy your sparkling clean code!
Measures
Last analysis 3 hours ago
New Code
Overall Code
Security
0 Open issues
A
0 H
0 M
0 L
Reliability
9 Open issues
D
8 H
1 M
0 L
Maintainability
54 Open issues
A
8 H
20 M
26 L
Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.
Coverage
0.0%
On 691 New Lines to cover.
Duplications
15.7%
On 5.4k New Lines.
Security Hotspots
4
E
ACTIVITY
Issues
There isn't enough data to generate an activity graph.
April 12, 2024 at 2:51 PM
NOT PROVIDED
Quality Gate: Passed
First analysis: 0 Issues · 0.0% Coverage · 15.7% Duplications
See full history of analyses


ufund-api - Overview - SonarQ
ufund-app - Overview - SonarQ
localhost:9000/dashboard?id=com.ufund.api%3Aufund-api
RIT
Portfolio
All Bookmarks
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
ufund-api
main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
main
1.4k Lines of Code · Version 0.0.1-SNAPSHOT ·
Set as homepage
The last analysis has warnings. See details
Quality Gate Status
Passed
Enjoy your sparkling clean code!
Measures
Last analysis 2 hours ago
New Code
Overall Code
Security
0 Open issues
A
0 H
0 M
0 L
Reliability
0 Open issues
A
0 H
0 M
0 L
Maintainability
94 Open issues
A
8 H
18 M
68 L
Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.
Coverage
77.1%
On 260 New Lines to cover.
Duplications
0.0%
On 1.4k New Lines.
Security Hotspots
0
A
ACTIVITY
Issues
There isn't enough data to generate an activity graph.
April 12, 2024 at 3:18 PM
0.0.1-SNAPSHOT
Quality Gate: Passed
First analysis: 0 Issues · 77.1% Coverage · 0.0% Duplications


ufund-api - Measures - SonarQ
localhost:9000/component_measures?id=com.ufund.api%3Aufund-api&metric=coverage&view=list
RIT
Portfolio
All Bookmarks
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
ufund-api
main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Project Overview
Reliability
Security
Security Review
Maintainability
Coverage
Overview
Overall Code
Coverage 77.1%
Lines to Cover 260
Uncovered Lines 60
Line Coverage 76.9%
Conditions to Cover 50
Uncovered Conditions 11
Condition Coverage 78.0%
Tests
Unit Tests 47
Errors 0
Failures 0
ufund-api
View as List
Select files
Navigate
8 files
Coverage 77.1% See history
Coverage
Uncovered Lines
Uncovered Conditions
src/main/java/com/ufund/api/ufundapi/UfundApiApplication.java 33.3% 2 33
src/main/java/com/ufund/api/ufundapi/persistence/UfundFileDAO.java 63.8% 24 5
src/main/java/com/ufund/api/ufundapi/persistence/UserFileDAO.java 64.1% 22 6
src/main/java/com/ufund/api/ufundapi/model/Ufund.java 66.7% 6 67
src/main/java/com/ufund/api/ufundapi/model/User.java 71.4% 6 71
src/main/java/com/ufund/api/ufundapi/controller/UfundController.java 100% 0 0
src/main/java/com/ufund/api/ufundapi/controller/UserController.java 100% 0 0
src/main/java/com/ufund/api/ufundapi/WebConfig.java 100% 0 100
8 of 8 shown