PROJECT Design Documentation

> *The following template provides the headings for your Design Documentation. As you edit each section make sure you remove these commentary 'blockquotes'; the lines that start with a > character and appear in the generated PDF in italics.*

Team Information

- Team name: Toasters
- Team members
  - Ethan Hartman
  - Albert Abaykhanov
  - Ben Griffin
  - Bevan Neiberg

Executive Summary

A local organization that facilitates the funding of student projects. Volunteers can either provide monetary support or direct goods and services for projects of their choice.

Purpose

Project Statement: A web application which allows users to fund student projects in order to complete the projects. The most important user group is the Student, which adds needs to their project. Their goal is to fulfill their project needs and finish the project.

Glossary and Acronyms

> ***[Sprint 2 & 4]*** *Provide a table of terms and acronyms.*

Term	Definition
SPA	Single Page
MVP	Minimum Viable Product

Requirements

This section describes the features of the application.

> *In this section you do not need to be exhaustive and list every story. Focus on top-level features from the Vision document and maybe Epics and critical Stories.*

Definition of MVP

The MVP allows for users to sign into the application, view student projects, add needs from those projects to their fund basket, and view the fund basket, as well as checking out their basket to fund the student projects. For students, it allows the creation and managing of needs for project funding.

MVP Features

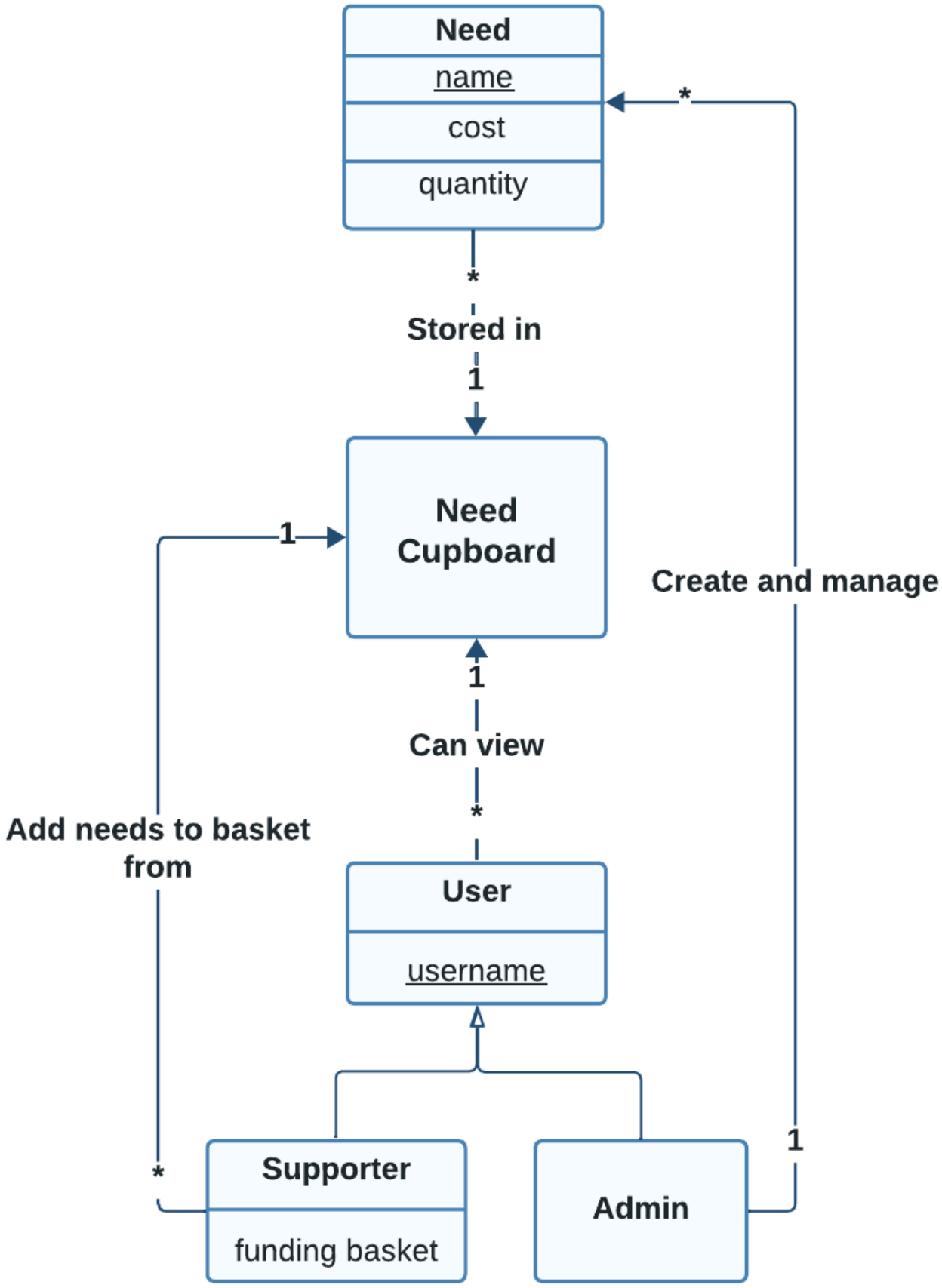
> ***[Sprint 4]*** *Provide a list of top-level Epics and/or Stories of the MVP.*

Enhancements

> ***[Sprint 4]*** *Describe what enhancements you have implemented for the project.*

Application Domain

This section describes the application domain.



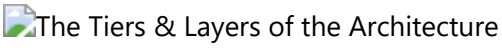
> ***[Sprint 2 & 4]*** *Provide a high-level overview of the domain for this application. You can discuss the more important domain entities and their relationship to each other.* Students can create and manage needs. Students can log in as admins. Supporters can view needs in student projects and they can also add and remove needs from their funding basket. Supporters can check out their fund basket.

Architecture and Design

This section describes the application architecture.

Summary

The following Tiers/Layers model shows a high-level view of the webapp's architecture. **NOTE**: detailed diagrams are required in later sections of this document. (*When requested, replace this diagram with your **own** rendition and representations of sample classes of your system*.)

 The Tiers & Layers of the Architecture

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 persistance.

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.

Overview of User Interface

This section describes the web interface flow; this is how the user views and interacts with the web application.

> *Provide a summary of the application's user interface. Describe, from the user's perspective, the flow of the pages in the web application.*

View Tier

> ***[Sprint 4]** Provide a summary of the View Tier UI of your architecture. Describe the types of components in the tier and describe their responsibilities. This should be a narrative description, i.e. it has a flow or "story line" that the reader can follow.*

> ***[Sprint 4]** You must provide at least **2 sequence diagrams** as is relevant to a particular aspects of the design that you are describing. (**For example**, in a shopping experience application you might create a sequence diagram of a customer searching for an item and adding to their cart.) As these can span multiple tiers, be sure to include an relevant HTTP requests from the client-side to the server-side to help illustrate the end-to-end flow.*

> ***[Sprint 4]** To adequately show your system, you will need to present the **class diagrams** where relevant in your design. Some additional tips:*
>
> - *Class diagrams only apply to the **ViewModel** and **Model** Tier*
> - *A single class diagram of the entire system will not be effective. You may start with one, but will be need to break it down into smaller sections to account for requirements of each of the Tier static models below.*
> - *Correct labeling of relationships with proper notation for the relationship type, multiplicities, and navigation information will be important.*

> - *Include other details such as attributes and method signatures that you think are needed to support the level of detail in your discussion.*

ViewModel Tier

> ***[Sprint 4]*** *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.*

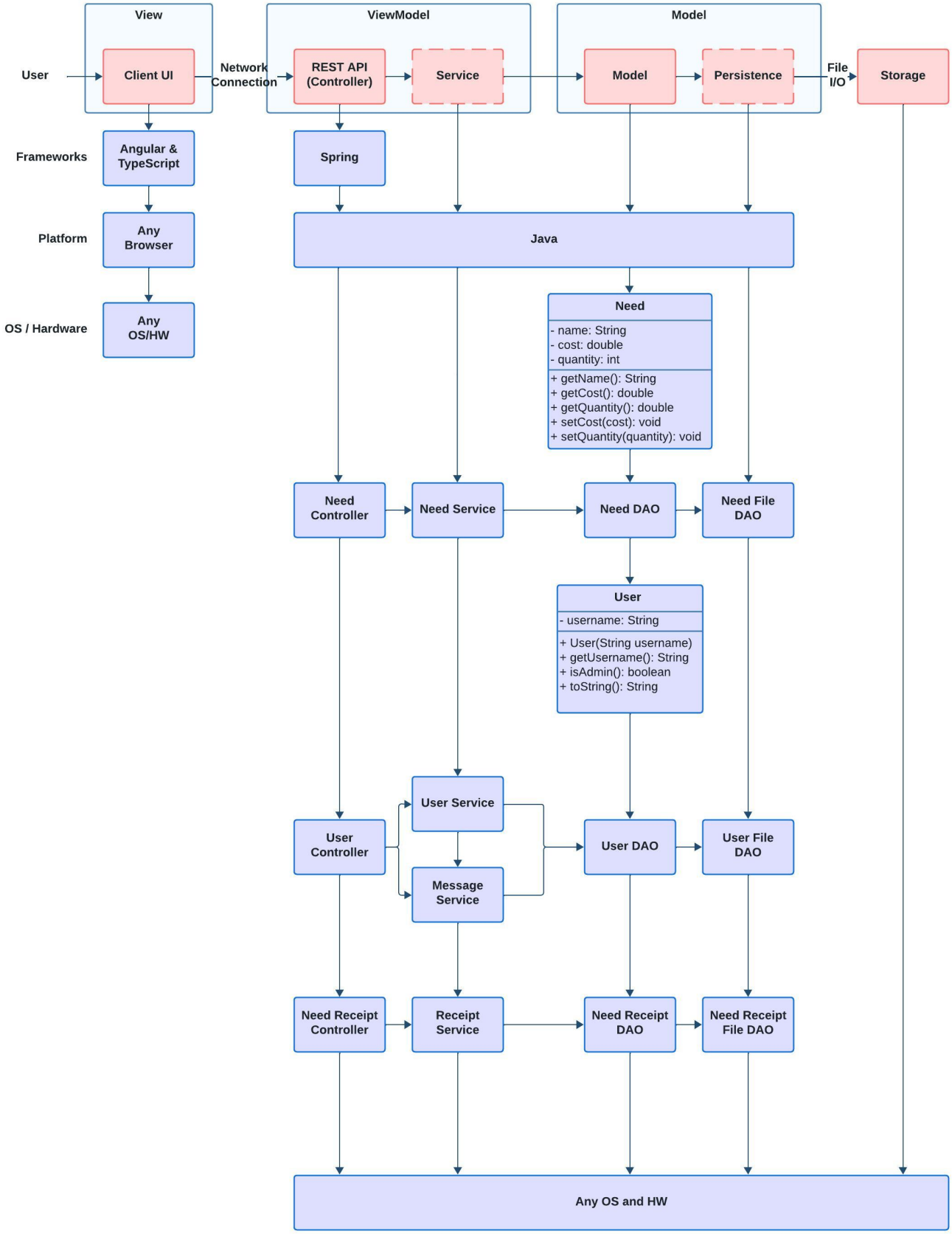
> *At appropriate places as part of this narrative provide* ***one*** *or more updated and* ***properly labeled*** *static models (UML class diagrams) with some details such as critical attributes and methods.*

 Replace with your ViewModel Tier class diagram 1, etc.

Model Tier

> ***[Sprint 3 & 4]*** *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.* For the model portion, we can see we have Need objects, which have the following private properties, name, cost, quantity. They have getters for those 3 properties, and setters for cost and quantity. Needs are used heavily within the Need File DAO, which is an extension of the Need DAO. Users have the private username property, and a getter for getting that username, and a method, isAdmin to check if the user is an admin. Users are used within the User File DAO, which is an extension of the User DAO.

> *At appropriate places as part of this narrative provide* ***one*** *or more updated and* ***properly labeled*** *static models (UML class diagrams) with some details such as critical attributes and methods.*



OO Design Principles

Principle 1: Single Responsibility We can see single responsibility, as we have our controllers, persistence, and model classes all separated. Moving even deeper, we have a User class, which holds usernames for users, and

a Supporter class specifically for supporters, which inherits from the user class and includes a funding basket. We separated the two because an admin is a user but is not a supporter.

Principle 2: Law of Demeter We see the law of Demeter when analyzing the controllers. The two controllers, UserController and NeedController only have access to their specific DAO class, UserDAO or NeedDAO. These controllers are then accessed by their specific Angular service, user service and need service.

Principle 3: High Cohesion All of our classes are named according to their functionality, similarly, they use other classes which have related functionality. For example, if we look at the NeedFileDAO, we can see that this handles all persistence of Need objects

> ***[Sprint 3 & 4]*** *OO Design Principles should span across* ***all tiers.***

Static Code Analysis/Future Design Improvements

> ***[Sprint 4]*** *With the results from the Static Code Analysis exercise,* ***Identify 3-4*** *areas within your code that have been flagged by the Static Code Analysis Tool (SonarQube) and provide your analysis and recommendations.*
> *Include any relevant screenshot(s) with each area.*

> ***[Sprint 4]*** *Discuss* ***future*** *refactoring and other design improvements your team would explore if the team had additional time.*

Testing

> *This section will provide information about the testing performed and the results of the testing.*

Acceptance Testing

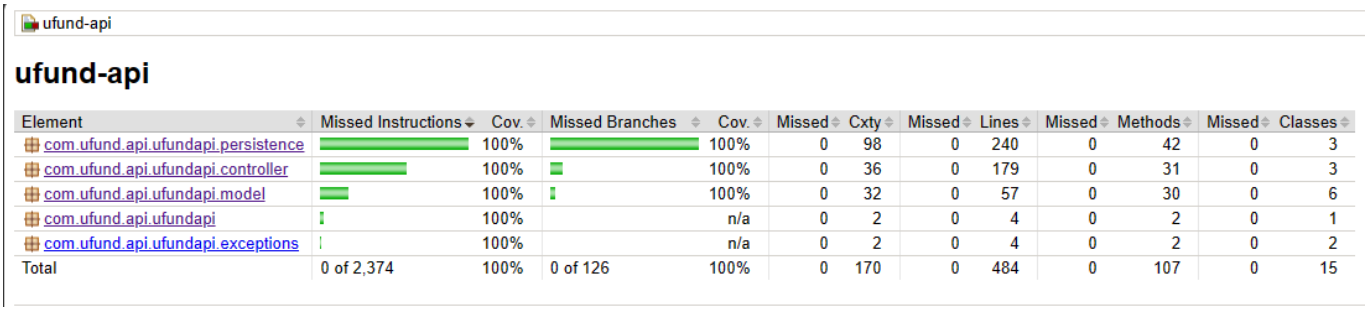
> ***[Sprint 4]*** *Report on the number of user stories that have passed all their acceptance criteria tests, the number that have some acceptance criteria tests failing, and the number of user stories that have not had any testing yet. Highlight the issues found during acceptance testing and if there are any concerns.* Sprint 2: 9/9 user stories passed all acceptance criteria tests. No issues found during acceptance testing.

Sprint 3: 30/34 acceptance criteria passing tests, 2 of which were old acceptance criteria which should have been updated, but they weren't, resulting in failed tests. For the last 2, the testing team couldn't figure out how to use the search bar. However, if used correctly, the acceptance criteria do pass.

Unit Testing and Code Coverage

> ***[Sprint 4]*** *Discuss your unit testing strategy. Report on the code coverage achieved from unit testing of the code base. Discuss the team's coverage targets, why you selected those values, and how well your code coverage met your targets.* Sprint 2: 9/9 user stories passed all acceptance criteria tests. No issues found during acceptance testing.

Sprint 3: 100% code coverage, no missing instructions and no missing branches.



ufund-api

ufund-api

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ufund.api.ufundapi.persistence		100%		100%	0	98	0	240	0	42	0	3
com.ufund.api.ufundapi.controller		100%		100%	0	36	0	179	0	31	0	3
com.ufund.api.ufundapi.model		100%		100%	0	32	0	57	0	30	0	6
com.ufund.api.ufundapi		100%		n/a	0	2	0	4	0	2	0	1
com.ufund.api.ufundapi.exceptions		100%		n/a	0	2	0	4	0	2	0	2
Total	0 of 2,374	100%	0 of 126	100%	0	170	0	484	0	107	0	15

> ***[Sprint 2 & 4] Include images of your code coverage report.*** *If there are any anomalies, discuss those.*