PROJECT Design Documentation

Team Information

- Team name: BCNS
- Team members
  - Nam Huynh
  - Saavan Tandon
  - Borneil Gope
  - Cameron Marsh

Executive Summary

This is a project which creates a website for a food charity. In this website, the user is able to login as a user or an admin using a username and password. The admin is able to view the needs cupboard and add or delete needs from it. The Helper is able to view the needs in the needs cupboard that the admin has added. The Helper can also search for needs using a search bar and have access to view and modify the funding basket. Items in the funding basket persist even after the Helper logs out and logs back in. Users can also send feedback to the admin, which the admin can see after they log in. The users can "check out" needs when they have decided to contribute to them, and they can also see a history of previous needs they have contributed to that persists through log in and log outs.

Purpose

The U-Fund Chairty webpage will allow Helpers to donate money to causes and items in order to assist the food bank. This will allow people with money to spare to assist those who are in need.

Glossary and Acronyms

Term	Definition
SPA	Single Page
DAO	Data Access Object
CSS	Cascading Style Sheets
TS	Type Script
Angular	Framework for web application
HTML	Hypertext Markup Language
Admin	Can edit products and orders
Admin	User that can edit the inventory
Helper	User that can edit their cart
Need	A purchasable item in the cupboard

Term	Definition
Cupboard	Access to the needs

Requirements

This section describes the features of the application.

- Admin can add, remove, and modify needs in the needs cupboard
- Helper can browse needs in the needs cupboard and search using keywords
- Helper can add/delete products from their cart
- Helper can log in to an existing account with username and password, or register should the account not exist
- Helper can checkout their needs from the basket

Definition of MVP

- Minimal Authentication for Helper/U-fund Manager login & logout

  - A Helper can create a new account with a new username and password
  - A Helper can log into an existing account
  - The Admin already has an account in the system
  - The Admin and Helper can log out of their account
  - Data Persistence on the Helper Basket
  - Username of user is unique

- Helper functionality

  - Helper can see list of needs
  - Helper can search for a need
  - Helper can add/remove an need to their funding basket
  - Helper can proceed to check-out and fund all needs they are supporting

- Needs Management

  - U-fund Manager(s) can add, remove and edit the data of all their needs stored in their needs cupboard
  - A U-fund Manager cannot see contents of funding basket(s) of Helpers
  - A U-fund Manager does not have a basket

- Data Persistence

  - The contents of the Helper basket in the basket will remain the same when the Helper log out and log back in.
  - When a Helper purchase all the need from the Cupboard, another Helper will not be able to see the needs from the Cupboard anymore until the Admin restock

MVP Features

Create New Need: AS a Developer I WANT to submit a request to create a new need (name [unique], cost, quantity, type) SO THAT it is added to the cupboard.

Get a Single Need: AS a Developer I WANT to submit a request to get a single need SO THAT I can access the cost, quantity and type.

Update a Need: AS a Developer I WANT to submit a request to update a need SO THAT I can change important information.

Delete a Single Need: As a Developer I want to submit a request to delete a single need SO THAT it is no longer in the cupboard.

Get Entire Cupboard: AS A Developer I WANT to submit a request to get the cupboard SO THAT I can update needs in the cupboard.

Search for Needs: AS a Developer I WANT to submit a request to get the the needs in the cupboard whose name contains the given text, SO THAT I have access to only that subset of needs.

Browse Needs: AS a Helper I WANT to see a list of needs SO THAT I choose which ones to contribute to.

Helper Authentication: AS a Helper I WANT to login securely with my credentials SO THAT I can purchase needs from the cupboard.

Manager Authentication: As a Manager I WANT to login securely with my credentials SO THAT I can manage the cupboards.

Saved Funding Basket: AS a Helper I WANT TO be able to logout with the contents of my funding basket saved SO THAT I can manage my support efficiently without the fear of losing progress.

Modify Cupboards (Epic): AS a Manager I WANT to be able to modify and populate the cupboard SO THAT I can present the best needs for the helper to fund.

Add Need (Manager): AS a Manager I WANT to add a need to the Needs Cupboard SO THAT it will be available for Helpers to select.

Delete Need (Manager): AS a Manager I WANT to remove a need from the Needs Cupboard SO THAT it will no longer be available for Helpers to select.

Manage Funding Basket (Epic): AS a Helper I WANT to be able to modify my funding basket by adding or deleting needs from it SO THAT I can effectively contribute to the organization.

Add Needs (Helper): AS a Helper I WANT to add needs to my funding basket SO THAT I can buy multiple needs at once.

Delete Needs (Helper): AS a Helper I WANT to delete needs from my funding basket SO THAT I can easily manage and refine my selection, ensuring my contributions align with my current preferences and priorities.

View Funding Basket: AS A Helper I WANT TO view the items in my Funding Basket SO THAT I can decide what items to fund.

Check Out Needs: AS a Helper I WANT to buy the needs SO THAT I can efficiently assist and address the unique challenges faced by the non-profit organization.

Enhancements

View Purchases: AS a Helper I WANT to view a history of my past purchases so that I can track what needs I have contributed to and make informed decisions.

- With this feature, a Helper is able to see a list of the items they have previously purchased by viewing a new section of the Funding Basket page.

Send Feedback (Helper): AS a Helper I WANT to send a feedback message to the Manager SO THAT I can make suggestions or notify the Manager of problems.

View Feedback (Manager): AS a Manager I WANT to view a list of received feedback messages SO THAT I can keep informed of the Helpers' concerns and suggestions.

- For Helpers, an ability to send a feedback message is added. And the U-Fund Manager has access to a new page in which all feedback messages sent from the Helpers can be viewed.

Application Domain

This section describes the application domain.
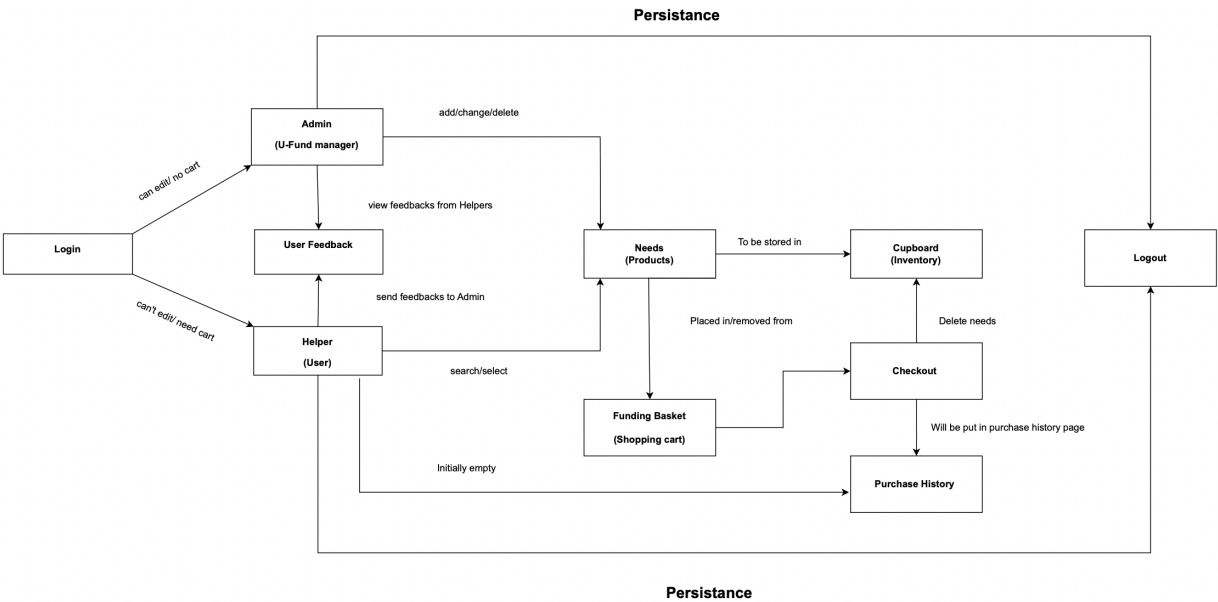
> **[Sprint 2 & 4]** *Provide a high-level overview of the domain for this application. You can discuss the more important domain entities and their relationship to each other.*

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.
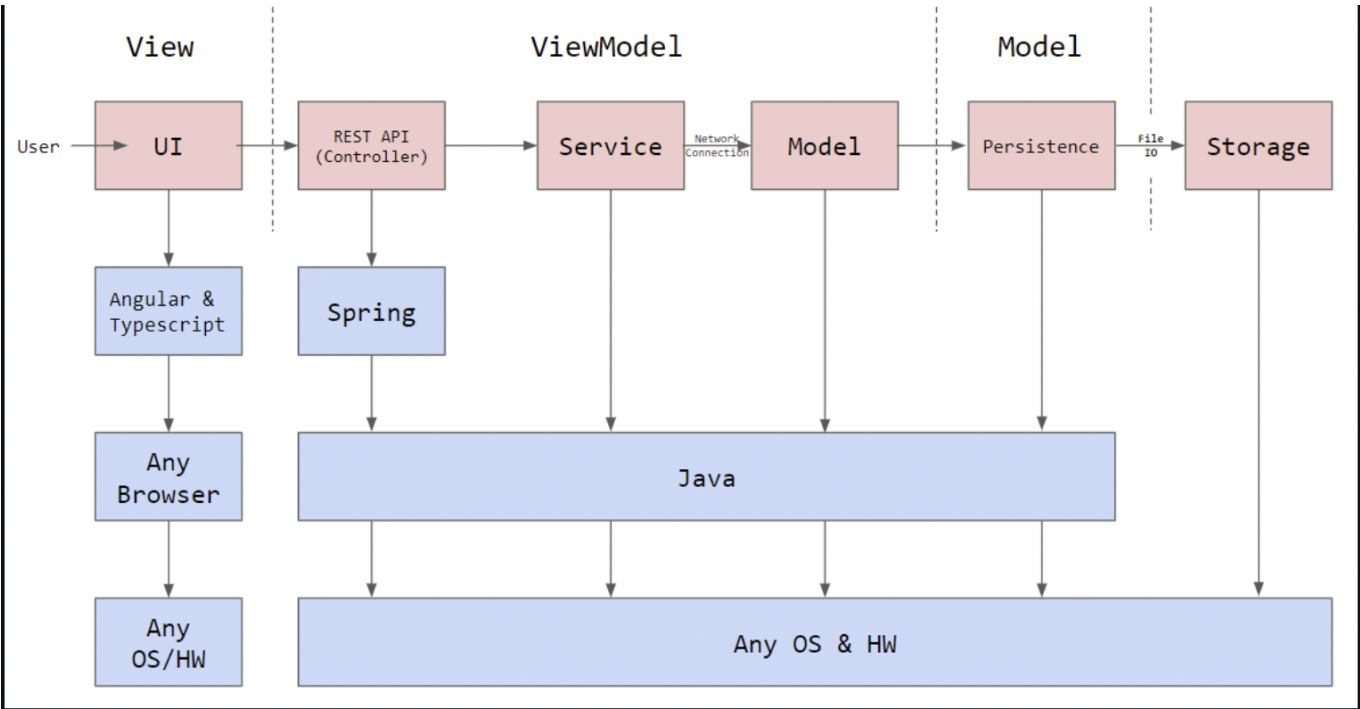
The web application, is built using the Model–View–ViewModel (MVVM) architecture pattern.

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

- **NeedController:** Responds to HTML requests for the Needs resource. It connects the Need UI to the Need model in the backend and will create, update, and get needs from the needs cupboard as necessary, all by accessing methods in NeedDAO objects.

- **HelperController:** Responds to HTML requests for the Helper resource. It connects the Funding Basket and Helper related UI to the backend, with functionality to create, get and authenticate helpers as well as manipulate a Helper's Funding Basket, all by accessing methods in HelperDAO objects.

- **NeedDAO:** An interface for the Data Access Object used to access and modify the underlying storage for the Needs Cupboard.

- **NeedFileDAO:** The specific implementation of NeedDAO. Contains functionality to create and get a need from the underlying storage, as well as search, update, and delete.

- **HelperDAO:** An interface for the Data Access Object used to access and modify the underlying storage for Helpers and their funding baskets.

- **HelperFileDAO:** the specific implementation of HelperDAO. Access the underlying storage to create, get, and update Helpers as well as their Funding Baskets. Also authenticates their login credentials.

[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.*
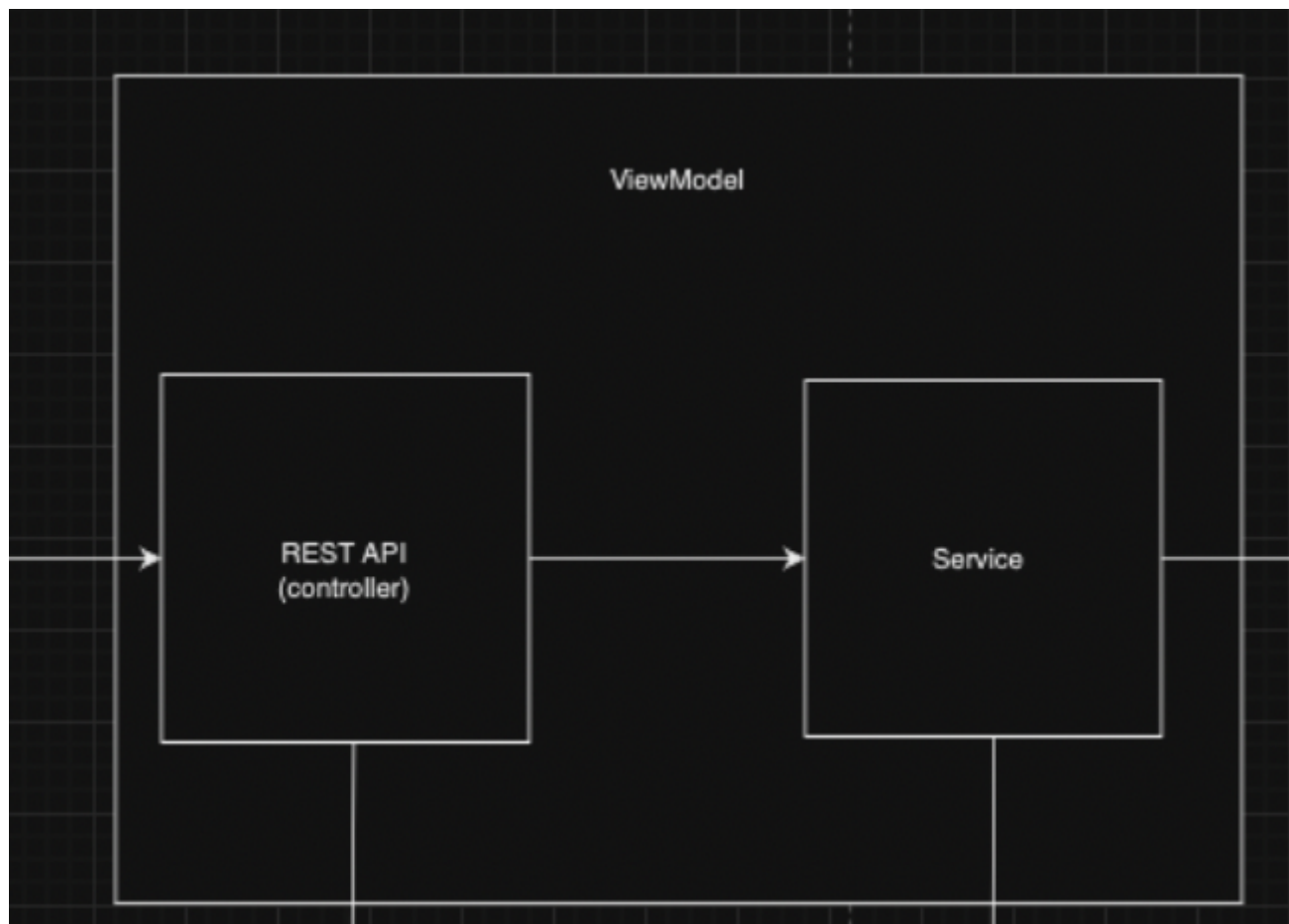
Model Tier

Need: Acts as a Java representation of a single need and its attributes. Works in tandem with NeedFileDAO and NeedController such that needs are loaded from the underlying storage into Need instances.

- **Helper:** Acts as a Java representation for a single Helper, its data, and its Funding Basket. Works in tandem with HelperController and HelperFileDAO to load Helpers and their funding baskets from the underlying storage into Helper instances.

> *[Sprint 2, 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.*

> *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.*
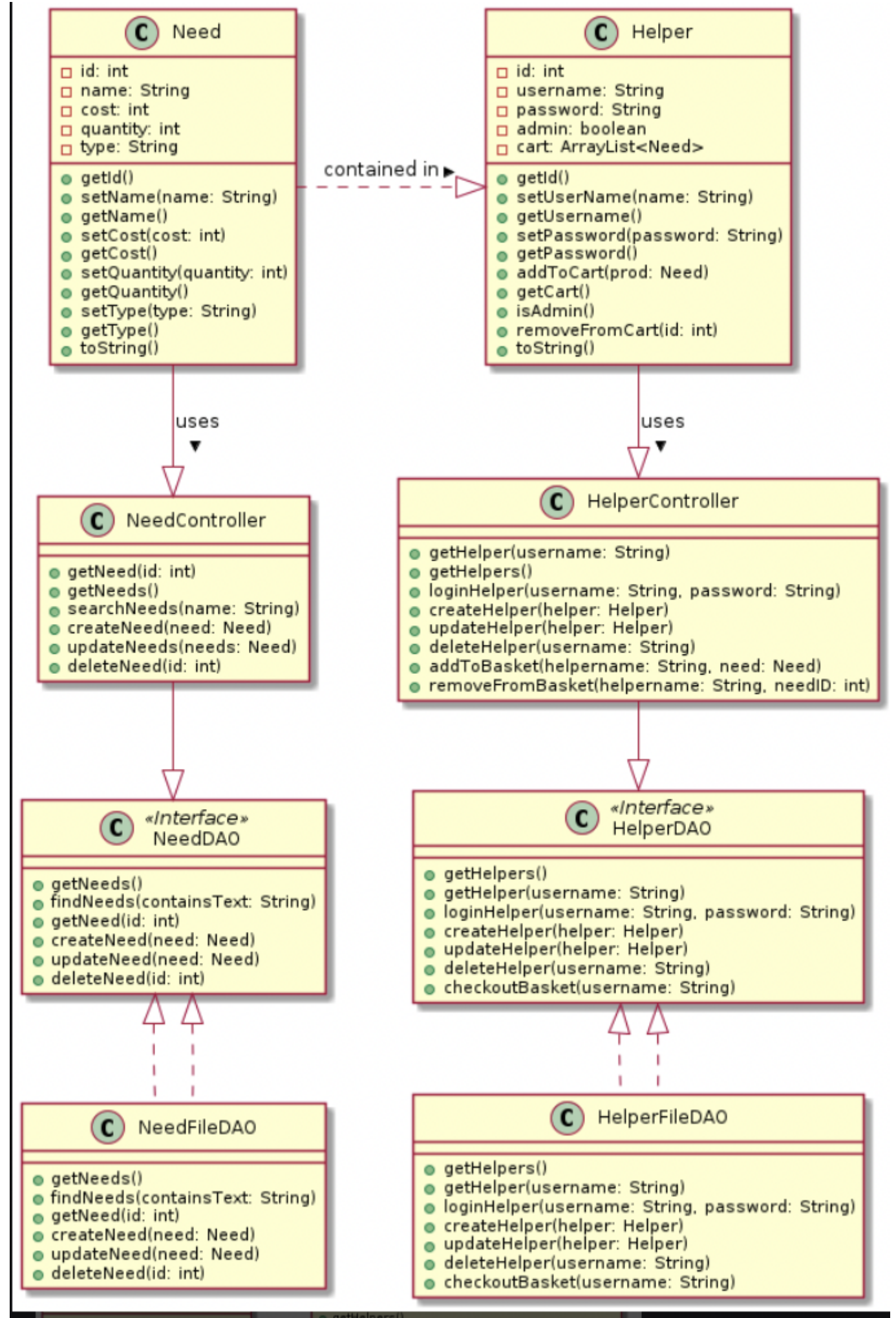
Need

- □ id: int
- □ name: String
- □ cost: int
- □ quantity: int
- □ type: String

- ● getId()
- ● setName(name: String)
- ● getName()
- ● setCost(cost: int)
- ● getCost()
- ● setQuantity(quantity: int)
- ● getQuantity()
- ● setType(type: String)
- ● getType()
- ● toString()

contained in ▶

Helper

- □ id: int
- □ username: String
- □ password: String
- □ admin: boolean
- □ cart: ArrayList<Need>

- ● getId()
- ● setUserName(name: String)
- ● getUsername()
- ● setPassword(password: String)
- ● getPassword()
- ● addToCart(prod: Need)
- ● getCart()
- ● isAdmin()
- ● removeFromCart(id: int)
- ● toString()

uses ▼

uses ▼

NeedController

- ● getNeed(id: int)
- ● getNeeds()
- ● searchNeeds(name: String)
- ● createNeed(need: Need)
- ● updateNeeds(needs: Need)
- ● deleteNeed(id: int)

HelperController

- ● getHelper(username: String)
- ● getHelpers()
- ● loginHelper(username: String, password: String)
- ● createHelper(helper: Helper)
- ● updateHelper(helper: Helper)
- ● deleteHelper(username: String)
- ● addToBasket(helpername: String, need: Need)
- ● removeFromBasket(helpername: String, needID: int)

«Interface»
NeedDAO

- ● getNeeds()
- ● findNeeds(containsText: String)
- ● getNeed(id: int)
- ● createNeed(need: Need)
- ● updateNeed(need: Need)
- ● deleteNeed(id: int)

«Interface»
HelperDAO

- ● getHelpers()
- ● getHelper(username: String)
- ● loginHelper(username: String, password: String)
- ● createHelper(helper: Helper)
- ● updateHelper(helper: Helper)
- ● deleteHelper(username: String)
- ● checkoutBasket(username: String)

NeedFileDAO

- ● getNeeds()
- ● findNeeds(containsText: String)
- ● getNeed(id: int)
- ● createNeed(need: Need)
- ● updateNeed(need: Need)
- ● deleteNeed(id: int)

HelperFileDAO

- ● getHelpers()
- ● getHelper(username: String)
- ● loginHelper(username: String, password: String)
- ● createHelper(helper: Helper)
- ● updateHelper(helper: Helper)
- ● deleteHelper(username: String)
- ● checkoutBasket(username: String)

OO Design Principles

> **[Sprint 1]** *Name and describe the initial OO Principles that your team has considered in support of your design (and implementation) for this first Sprint.*

Single Responsibility: (created a domain model, separating classes based on their individual purpose, which follows Single Responsibilty as each class serves one unique purpose)

Open/Closed: (Incorporated restapi classes into projects which used encapsulation to keep some modules public and others private, which protects data and enhances security)

> **[Sprint 2, 3 & 4]** *Will eventually address upto **4 key OO Principles** in your final design. Follow guidance in augmenting those completed in previous Sprints as indicated to you by instructor. Be sure to include any diagrams (or clearly refer to ones elsewhere in your Tier sections above) to support your claims.*

Low Coupling: (describe how DAO interfaces reduce coupling between Controllers and DAOs)

Dependency Inversion/Injection: (describe how DAO interfaces are injected into Controller classes' contructors rather than being initialized in the controller)

Pure Fabrication: (describe how HelperFileDAO/NeedFileDAO doesn't represent a real entity but is created for the purpose of separating the methods that access Helper and Need storage)

> **[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

- **29** user stories in total, covering:
  - Admins being able to edit needs from the cupboard
  - Helpers being able to login/creating their account
  - Helpers being able to edit needs from their basket
  - Helpers being able to checkout their basket
  - Helpers being able to search for needs
  - Persistence of the shopping carts
  - 10% Features: creating/viewing feedbacks and display purchase history

All user stories have passed their acceptance criteria tests.

Unit Testing and Code Coverage

Strategy We targeted 90% code coverage across the project with our unit tests because this would ensureminimal gaps in the coverage while still being achievable.

U-fund API Class Level Code Coverage:

ufund-api												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ufund.api.ufundapi.controller		76%		71%	9	41	39	160	0	25	0	3
com.ufund.api.ufundapi.persistence		98%		87%	6	61	2	164	0	37	0	3
com.ufund.api.ufundapi		88%		n/a	1	4	2	7	1	4	0	2
com.ufund.api.ufundapi.model		98%		75%	3	35	1	53	1	31	0	3
Total	170 of 1,742	90%	17 of 88	80%	19	141	44	384	2	97	0	11

U-fund API Controller Code Coverage:

com.ufund.api.ufundapi.controller												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
HelperController		66%		61%	7	20	28	78	0	11	0	1
FeedbackController		72%		83%	1	9	10	34	0	6	0	1
NeedController		97%		87%	1	12	1	48	0	8	0	1
Total	151 of 654	76%	9 of 32	71%	9	41	39	160	0	25	0	3

- **Controller**: The controller tier has 76% code coverage overall, primarily due to the large size of HelperController making it difficult to test.

U-fund API Model Code Coverage:

com.ufund.api.ufundapi.model												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Feedback		90%		n/a	1	6	1	9	1	6	0	1
Helper		100%		75%	2	17	0	26	0	13	0	1
Need		100%		n/a	0	12	0	18	0	12	0	1
Total	4 of 246	98%	2 of 8	75%	3	35	1	53	1	31	0	3

- **Model**: The model tier has 100% code coverage overall, meaning it is very well tested. All tests created for this tier passed.

U-fund API Persistence Code Coverage:

com.ufund.api.ufundapi.persistence												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
HelperFileDAO		96%		81%	4	24	2	63	0	13	0	1
NeedFileDAO		100%		93%	1	21	0	54	0	13	0	1
FeedbackFileDAO		100%		90%	1	16	0	47	0	11	0	1
Total	10 of 799	98%	6 of 48	87%	6	61	2	164	0	37	0	3

- **Persistence**: The persistence unit test is well done.

Ongoing Rationale

- (2024/2/10): Sprint 1
  - The 10% feature will be Helper Feedback and Purchase History Page
- (2024/3/19): Sprint 2

- - The team will change the architecture of the project significantly by removing the Basket logic in the API entirely and will be represented as an array of Needs in the Helper
  - Rationale:
    - Reduce the number of unnecessary unit testing
    - Making the Basket as an attribute of Helper would make a significantly improvement on the design and better adherence to GRASP Principles by reducing Coupling Issue
- (2024/4/3): Sprint 3
  - The team will change the architecture of the project by adding the a list of purchase history to the Helper
  - Rationale:
    - Reduce the number of unnecessary unit testing
    - Making it easier to implement and keep track of the purchase history