# 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 but do so only **after** all team members agree that the requirements for that section and current Sprint have been met. **Do not** delete future Sprint expectations.*

## Team Information

- Team name: Autobots
- Team members
  - Johanna Wichmann
  - Conner Meagher
  - Isaac Ingram
  - Darius Nakis

## Executive Summary

We are making a website with a working frontend and backend for an organization that needs donations and items to help run it.

### Purpose

> ***[Sprint 2 & 4]*** *Provide a very brief statement about the project and the most important user group and user goals.* For sprint 2, we focused on completeing the basket, cupboard, and login system. This gives us the ebasics of the functioning website to be able to differentiate who is loging in to the website and what is being added to and from carts.

### Glossary and Acronyms

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

| Term | Definition |
| --- | --- |
| SPA | Single Page |

## Requirements

Login features, statistic features, basket features, cupboard features, need features

> *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

> ***[Sprint 2 & 4]*** *Provide a simple description of the Minimum Viable Product.* For sprint 2, our MVP includes a working basket that you can add and remove items from, a working login to allow admin

> and helpers to login seperately, and a cupboard that stores needs and allows them to move into and out of cart.

### MVP Features

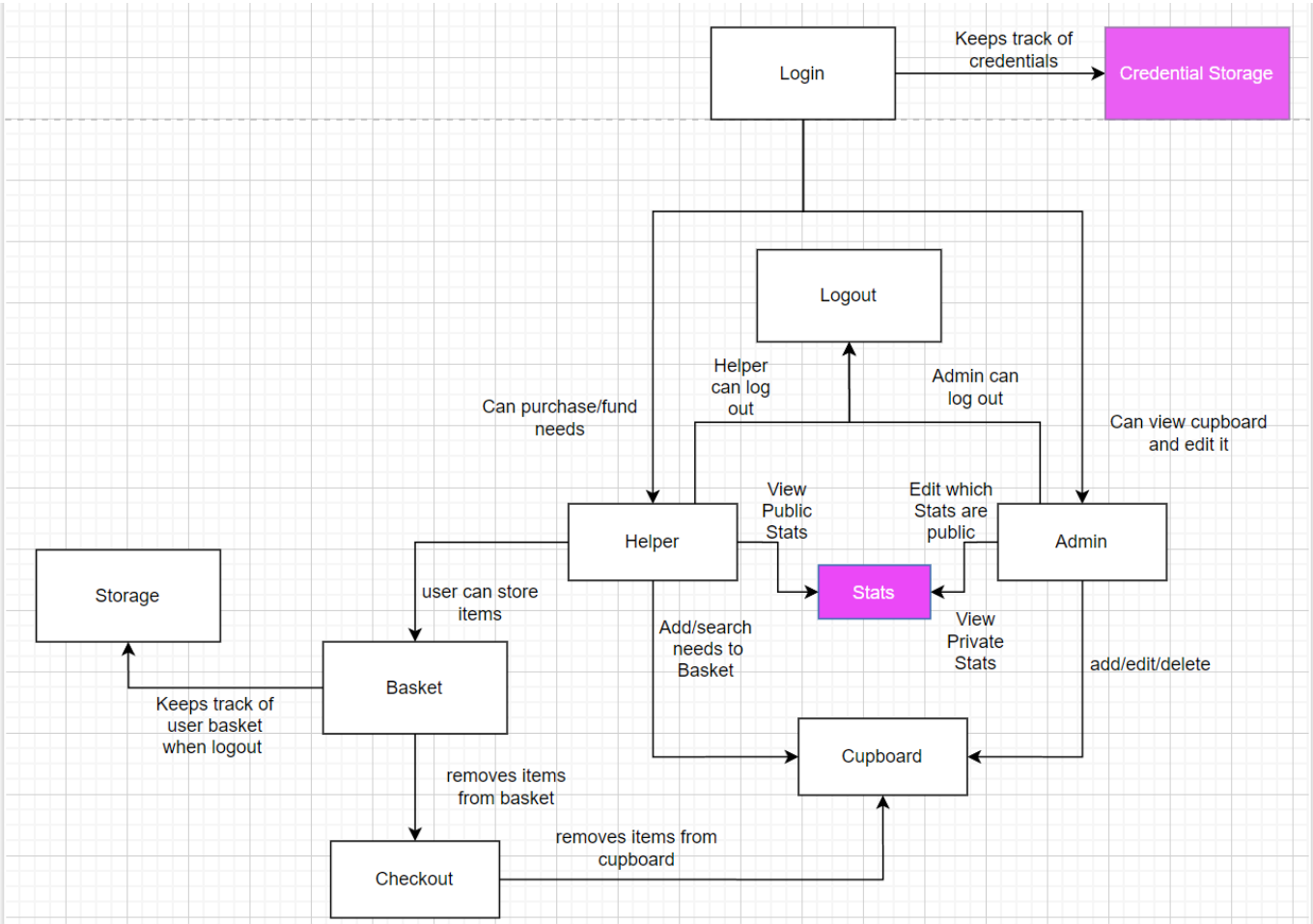
> ***[Sprint 4]*** *Provide a list of top-level Epics and/or Stories of the MVP.*

### 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.* For sprint 2, we worked on the login, basket, and cupboard part of the domain. The login component is where a user is differentiated as either a helper or an admin. When a helper logs in, they can view needs and add them to their cart and remove them from their cart. Helpers are the ones that are purchasing needs to fund the organization. When an admin logs in, they can edit the needs in the cupboard. The basket component is where a helper can see what they have added to their cart and they can edit it there. A helper can remove items from their cart or increase/decrease the quanitity of the item they have chosen. AN admin cannot see what is in helpers carts. From the basket component there is also the checkout button that removes the items that are checked out from the cupboard count. Finally the cupboard which is where all the

> needs are displayed. Helpers can view the cupboard and search for needs and add them to the basket. Admins can edit the cupboard and put in more needs or remove needs that are no longer needed.

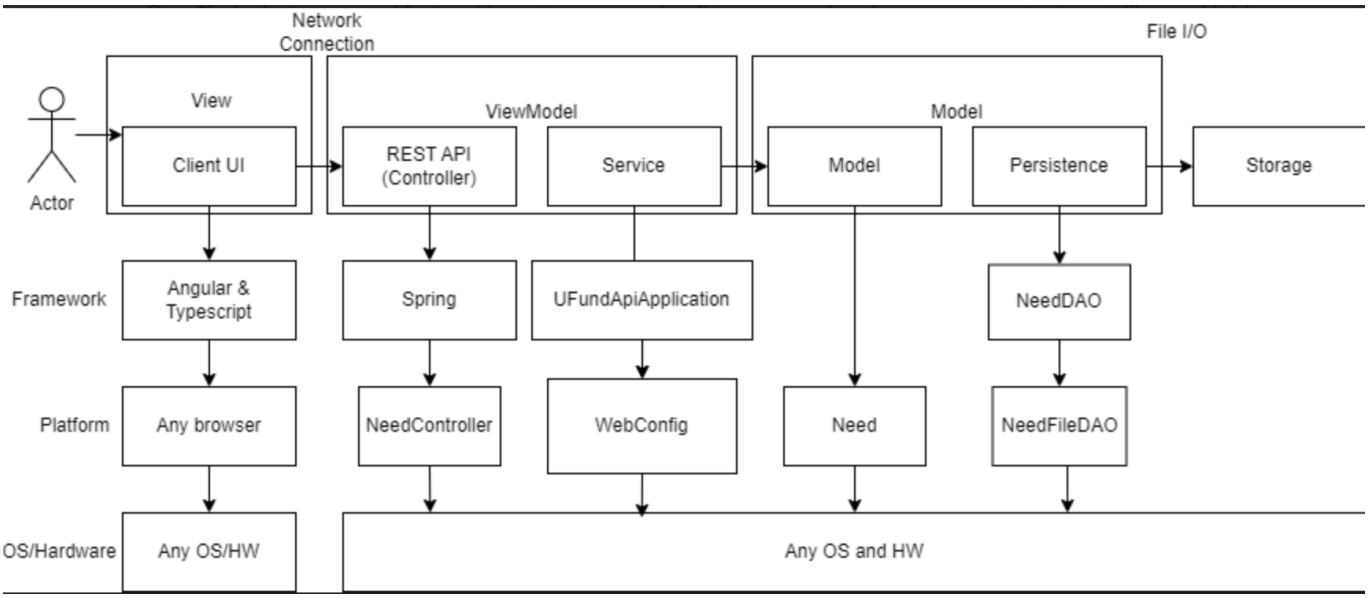
# Architecture and Design

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.

> ***[Sprint 1]*** *(Augment this diagram with your* ***own*** *rendition and representations of sample system classes, placing them into the appropriate M/V/VM (orange rectangle) tier section. Focus on what is currently required to support* ***Sprint 1 - Demo requirements****. Make sure to describe your design choices in the corresponding* ***Tier Section*** *and also in the* ***OO Design Principles*** *section below.)*




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.* A user will enter the login page which will make them login as either a admin or a helper. From their, they can go to the cupboard to add things to their basket or to their basket to edit the things they have selected to put into their basket.

## 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.*

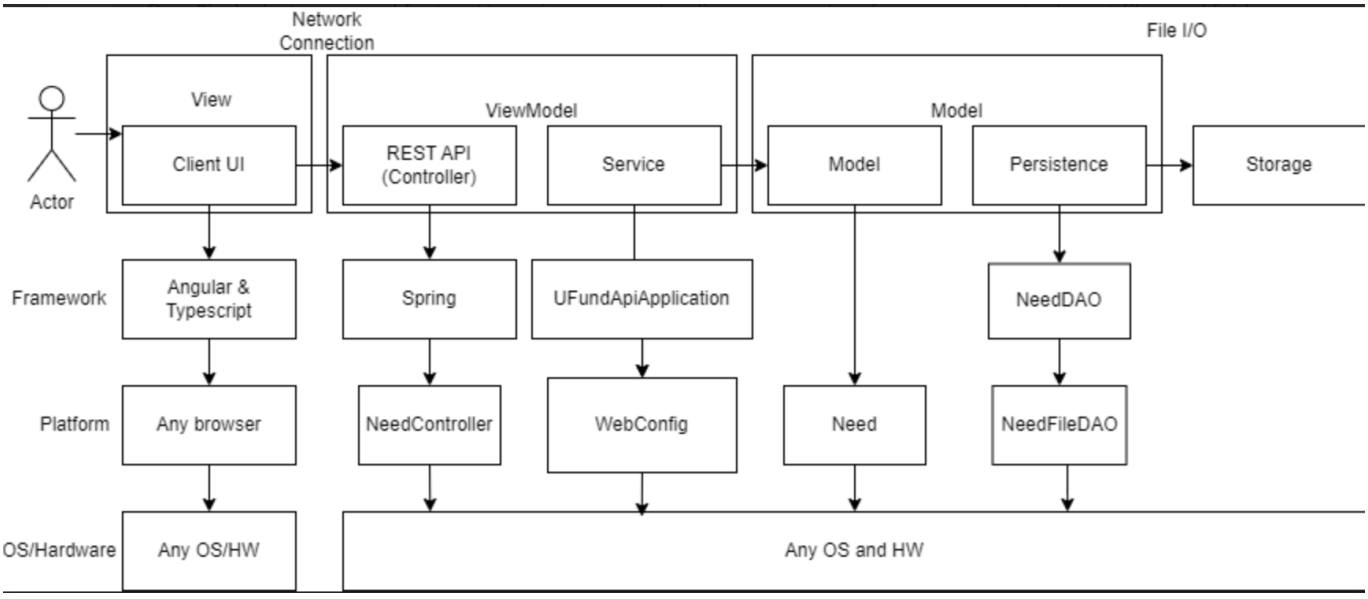
> ***[Sprint 4]*** *To adequately show your system, you will need to present the **class diagrams** where relevant in your design. Some additional tips:*
>
> - *Class diagrams only apply to the **ViewModel** and **Model** Tier*
> - *A single class diagram of the entire system will not be effective. You 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

Within our viewmodel, we have our REST API and the Service. Our REST API uses spring framework and uses the NeedController as its platform. Service uses UFundAPIApplication as its framework and the WebConfig as its platform. Both of these can use any operating system or hardware to run.

> ***[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.*

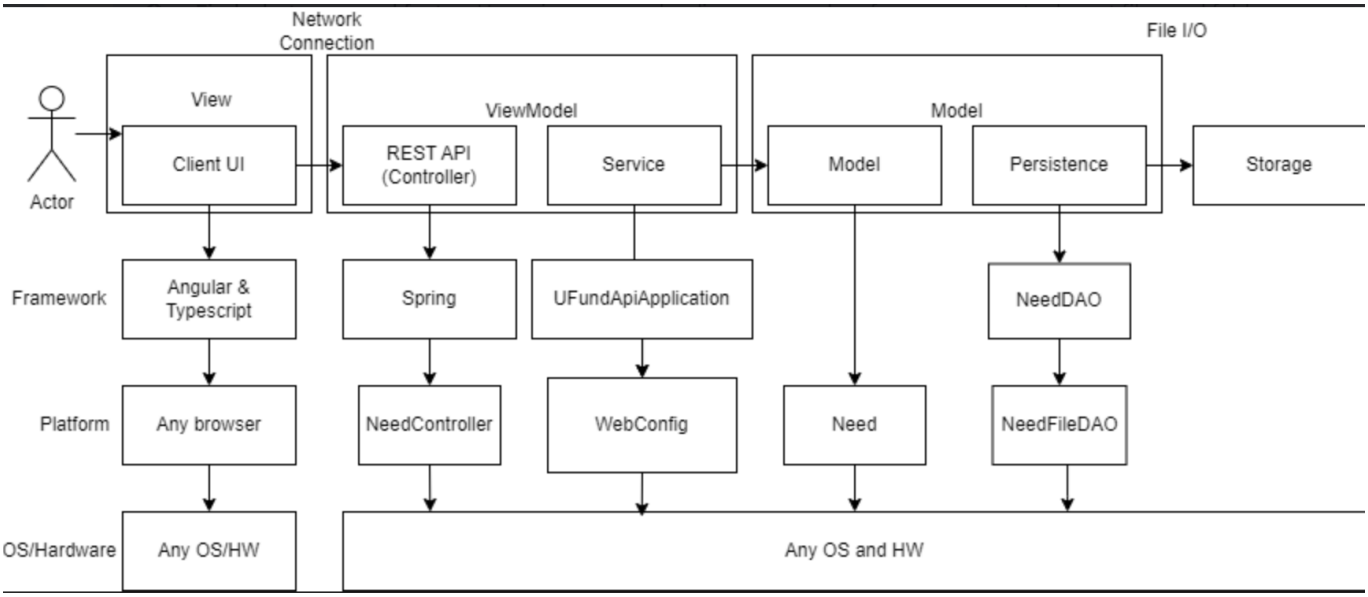


### Model Tier

Within our model, we have the Model and the Persistence. Our model contains the main Needs file that defines a need and all of the functions Needs does. Our Persistence uses the NeedDAO and the NeedFileDAO. The NeedDAO is the interface for the NeedFileDAO to use to help implement all of its functions.

> ***[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.*




## OO Design Principles

We used low coupling as a design principle between the NeedDAO and the NeedFileDAO. This allowed us to use an interface, the NeedDAO, to implement the functions in another file, the NeedFileDAO. It helps us to stay organized and know which file is having problems with the separation of the two. We also used single responsibility for our functions. Each function in the API only computes one function at a time. Our creatNeed only creates a need and nothing else. It allows the API to not have to worry about what's going on around the rest of the program and just what is going on in that single function. We use the information expert like in our AccessControlService is in control of all of the information about the login process and who the user is. We use the Law of Demeter in our project as shown by our use of components and how everything is very seperated from each other. Take for example the LoginComponent. The only thing outside of the LoginCompenent that anything in the LoginComponent talks to is the AccessControlService. Everything else is dealt with only in the LoginComponent

> ***[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.*

> ***[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

Our testing tests all functions within the API. All testing is functional for both the File DAO and the Controller.

## Acceptance Testing

> ***[Sprint 2 & 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.*
> Our NeedsControllers that includes createNeed, updateNeed, getNeed, deleteNeed, searchNeeds, and getNeeds is fully tested and passed all tests.

Our Need class is mostly tested but is just missing an equals testing to test two items in different classes. The getID, getName, getType, getPrice, and getQuantity are all fully tested at 100%.

Our NeedFileDAO is mostly tested except for createNeed which is not tested at all with a 0%.

## 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 & 4] Include images of your code coverage report.*** *If there are any anomalies, discuss those.*
>
> **com.ufund.api.ufundapi**
>
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UfundApiApplication | | 37% | | n/a | 1 | 2 | 2 | 3 | 1 | 2 | 0 | 1 |
| WebConfig | | 100% | | n/a | 0 | 2 | 0 | 4 | 0 | 2 | 0 | 1 |
| Total | 5 of 43 | 88% | 0 of 0 | n/a | 1 | 4 | 2 | 7 | 1 | 4 | 0 | 2 |

> **com.ufund.api.ufundapi.model.persistence**
>
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NeedFileDAO | | 80% | | 85% | 4 | 22 | 12 | 60 | 2 | 12 | 0 | 1 |
| Total | 56 of 287 | 80% | 3 of 20 | 85% | 4 | 22 | 12 | 60 | 2 | 12 | 0 | 1 |

> **com.ufund.api.ufundapi.model**
>
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Need | | 97% | | 50% | 4 | 16 | 1 | 25 | 0 | 12 | 0 | 1 |
| Total | 3 of 106 | 97% | 4 of 8 | 50% | 4 | 16 | 1 | 25 | 0 | 12 | 0 | 1 |

> **com.ufund.api.ufundapi.viewmodel**
>
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NeedController | | 100% | | 100% | 0 | 12 | 0 | 48 | 0 | 8 | 0 | 1 |
| Total | 0 of 194 | 100% | 0 of 8 | 100% | 0 | 12 | 0 | 48 | 0 | 8 | 0 | 1 |

> Our NeedFileDAO is not fully tested but when we add a few more tests it should be at 100% coverage.

# Ongoing Rationale

2/21/24: Sprint 1 completed 3/20/24: Sprint 2 completed