ANIMAL AUXILIARIES 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: 2231-swen-261-05-2b-animal-auxiliaries
- Team members
  - Chase Balmer
  - Sarah Payne
  - Marisa Ortiz
  - Tyler Combs
  - Tszfai Choy

Executive Summary

This project is a U-Fund website that is to be used by a local animal shelter/humane society. This website provides tools for a welcoming and supportive community with the purpose of helping homeless animals. Managers run the needs page, indicating the current needs of the organization. This may include funding equipment purchases, purchasing food, etc. Helpers may view these needs and contribute through funding. Helpers can also view an adoption board (created/updated by a manager) and start the process of adoptiong an available animal. We hope that our product will enable members of our community to help these innocent animals and provide them with the things they need, or even a home. The websites backend is built in Java-Spring, the frontend with Angular.

Purpose

> ***[Sprint 2 & 4]** Provide a very brief statement about the project and the most important user group and user goals.*

The Animal Auxiliaries U-Fund website exists to allow an animal shelter to virtually fundraise for their non-profit by making the process of donating to the animal shelter simple and easy.

Glossary and Acronyms

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

Term	Definition
Need	A single item or action that the non-profit needs to be funded. Needs are **created** by *Managers* and **funded** by *Helpers*.
Cupboard	Virtual storage for the many needs of the non-profit. The cupboard is **edited** by *Managers* and **viewed** by *Helpers*.

Term	Definition
Funding Basket	Virtual shopping basket. Each *Helper* has a personal Funding Basket, they can **add** and **remove** *Needs* from their Funding Basket as well as **Check-out** or buy those *Needs*.
Manager	A website User that represents a person who is responsible for the needs of a non-profit. The Manager has the ability to **edit** the *Cupboard*.
Helper	A website User that represents a person who is donating to the non-profit. The Helper has the ability to **search** the *Cupboard* and **fund** *Needs*.

Requirements

Webpage displaying and enabling all of the following things:

Authentication for Helper/U-fund Manager login & logout will enable necessary privledges for a user. Login page displays community updates posted by the managers.

Helpers have the following abilities: Helper can see list of needs. Helper can search for a need. Helper can add/remove a need to their funding basket. Helper can contribute money to a general surplus. Helper can checkout their funding basket. Helper can view an adoption board. Helper can view whether or not an animal is adoptable. Helper can start the adoption process for an animal.

Managers have the following abilities: Manager(s) can add, remove and edit the data of all their needs stored in their needs cupboard. Manager(s) can add and remove animals to the adoption board. Manager(s) can post and delete updates on the community board. U-fund Manager cannot see contents of funding basket(s).

Data Persistence The system saves all information to files so that changes are reflected for all users.

Your 10% additional feature enchantment(s) Adoption Site Community Board

> *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

- **Minimal Authentication** --> Minimal Authentication allows any user to login to the website without creating an account. Any user attempting to login with the **admin** username will be logged in as a *Manger*. Any user attempting to login with any other username will be logged in as a *Helper*.
- **Helper Functionality** --> Minimal Helper Functionality allows the *Helper* to **search** for a *need*. As well as **Add** and **Remove** *Needs* from their *Funding Basket*.
- **U-fund Manager Functionality** --> Minimal Manager Functionality allows a *Manager* to **add**, **remove**, and **edit** the *needs* in the *cupboard*. And prevent the *Manager* from viewing the *funding baskets*.

MVP Features

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

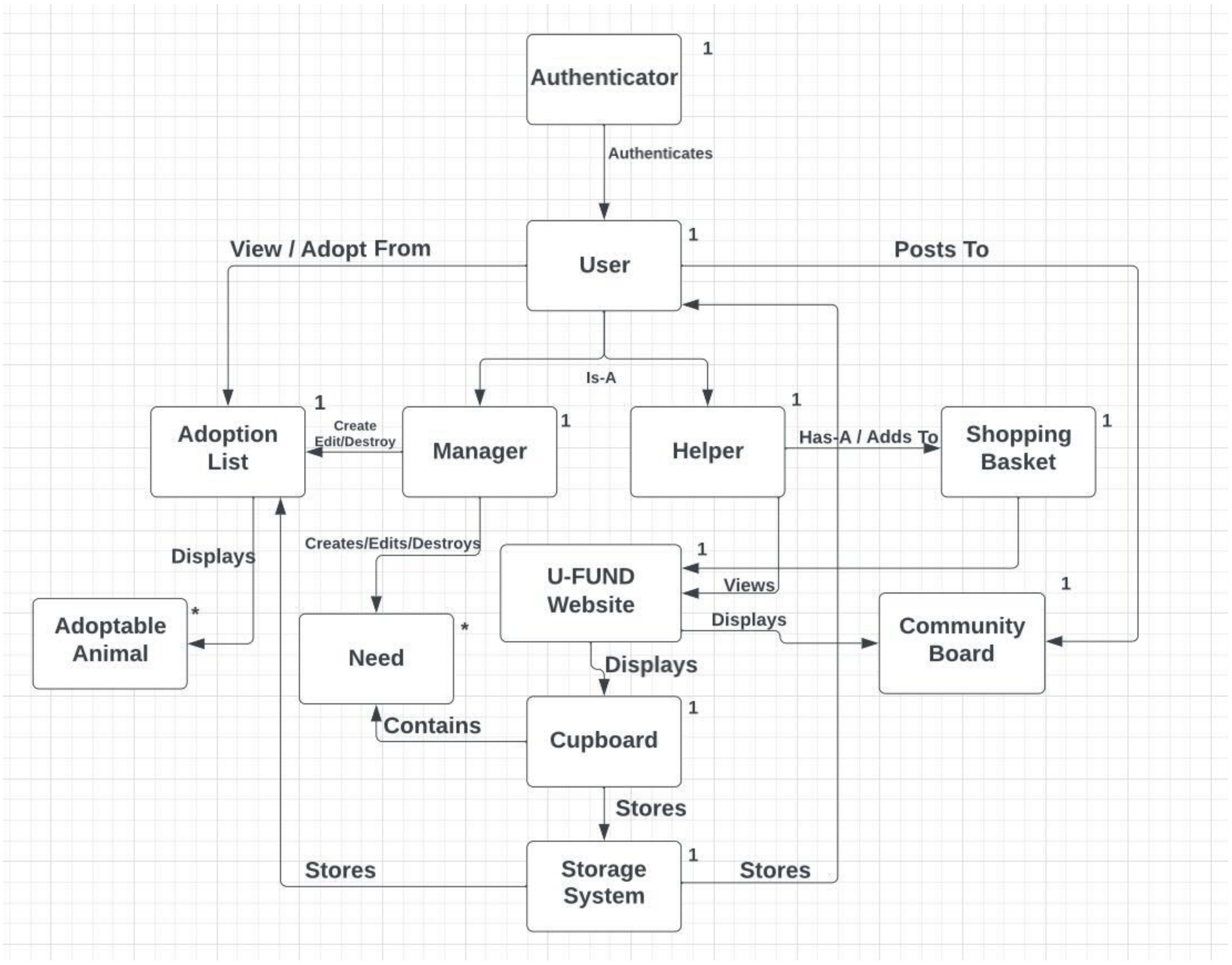
Enhancements

Community Board
The Community Board displays text posts that are posted/deleted by managers. This is viewed on the login page. The hope is for managers to use this to communicate important/timely information and create a tight-knit environment.

Adoption Board The Adoption Board displays a board of adoptable animals that helpers can view. The animals on the board are managed by a manager. Helpers see the board and can elect to start the adoption process on an animal if it is available. If the process is started on an animal, that animal will be labeled as on hold and will be unadoptable by other users. We assume the actual adoption process would be in person/not on the website, so this board is more of a means to research the animal and begin that process of adoption. This enhancement creates a strong identity for our site and provides the ultimate means to help the non-profit.

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 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.*

The application opens on a login page 'http://localhost:4200/login', logging in as an admin will take the user to the manager page 'http://localhost:4200/manager'; logging in as a helper takes you to the helper page 'http://localhost:4200/helper'

> TO DO update this line after bug fixes

Once on the manager page the user sees a 'logout' button at the top of the page was a create need textbox and a list of needs in the cupboard underneath the logout button and a heading 'My Needs'. A manager can create a need by entering a name into the 'Need name: ' labeled text box and clicking the 'Add need' button, this adds the need to the list of all needs underneath. The list of needs displays the need names which link to a 'Need detail' page as well as an 'x' button which will delete the need from the cupboard. The 'Need detail' page displays the name, description, type, price, quantity, and quantity funded properties of the need in editable text boxes. The manager can update the values of the need properties by editing the boxes and clicking the 'save' button, or discard their changes by clicking the 'go back' button.

Once on the helper page the user sees a 'logout' button above a search box with all of the cupboard needs listed below displaying yhe need name, description, price, and quantity propoerties as well as a '+' button that when clicked will add the need to the helper's funding-basket. Below the list of all needs in the cupboard is a list of all needs inside of the helpers funding-basket where the needs are displayed in the same way as they are in the cupboard except with a '-' button which will remove the need from the helper's funding-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

> ***[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.*

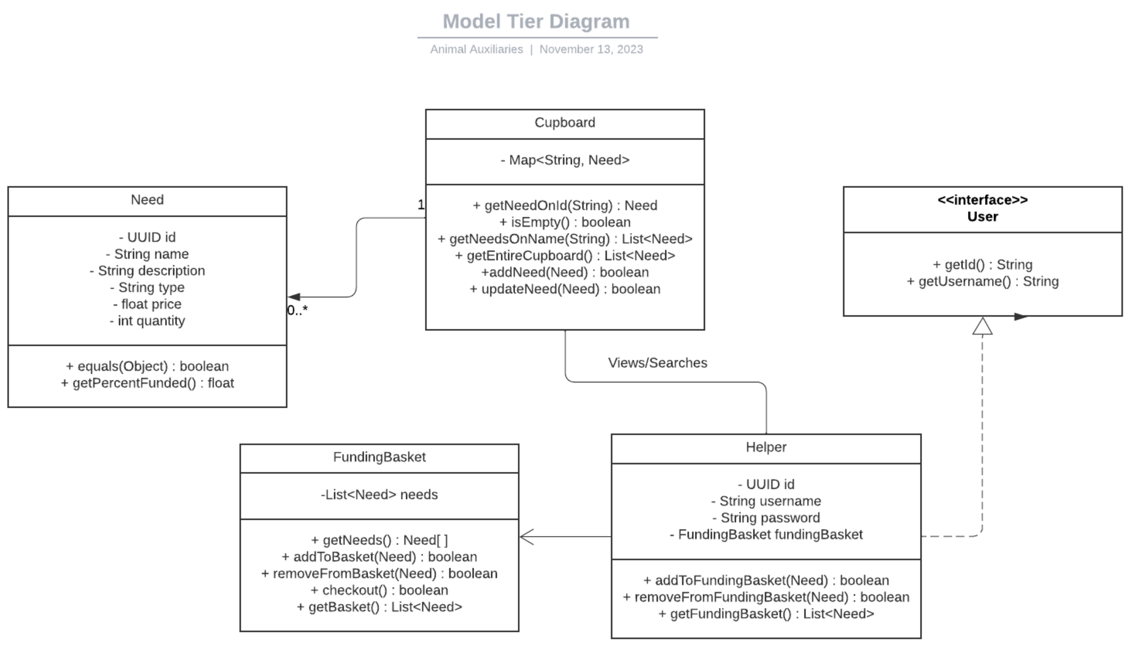
> *At appropriate places as part of this narrative provide* ***one*** *or more updated and* ***properly labeled*** *static models (UML class diagrams) with some details such as critical attributes and methods.*

Replace with your ViewModel Tier class diagram 1, etc.

Model Tier

> ***[Sprint 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.*


Model Tier Diagram
Animal Auxiliaries | November 13, 2023
Cupboard
- Map<String, Need>
+ getNeedOnId(String) : Need
+ isEmpty() : boolean
+ getNeedsOnName(String) : List<Need>
+ getEntireCupboard() : List<Need>
+addNeed(Need) : boolean
+ updateNeed(Need) : boolean
Need
- UUID id
- String name
- String description
- String type
- float price
- int quantity
+ equals(Object) : boolean
+ getPercentFunded() : float
1
0..*
<<interface>>
User
+ getId() : String
+ getUsername() : String
Views/Searches
FundingBasket
-List<Need> needs
+ getNeeds() : Need[ ]
+ addToBasket(Need) : boolean
+ removeFromBasket(Need) : boolean
+ checkout() : boolean
+ getBasket() : List<Need>
Helper
- UUID id
- String username
- String password
- FundingBasket fundingBasket
+ addToFundingBasket(Need) : boolean
+ removeFromFundingBasket(Need) : boolean
+ getFundingBasket() : List<Need>

OO Design Principles

> ***[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 1 Champion Principles

Single Responsibility

- During Sprint 1 our team focused on keeping our classes single responsibility. This means keeping classes as small as possible ad ensuring that every class or object we create should only do one thing.
- Our cupboardController is a good implementation of a single responsibility class; its only job is manipulating the needs in the cupboard. It does not hold the logic for doing these things, its only responsibility is to call the correct methods to make the manipulations happen.

Controllers

- During Sprint 1 our team focused on the controllers design principle. Controllers handle user input/actions and interact with the models and views. They facilitate communication between the user interface and the underlying system components.
- The controller principle is applied in our design by separating the user interface from the business logic of the application.

- Our cupboardController is a good example of a controller class in our code; it handles adding/removing/edting needs in the cupboard

Sprint 2 OO Design Principles

Information Expert

- During Sprint 2, our team focused on making our Need and Cupboard classes 'information experts'. These classes have methods that can be called to find the states of their many properties.
- The Cupboard class has many methods that support searching functionality, so that our cupboardController class can call cupboard methods with search terms and only recieve back needs that fit the search, rather than requesting the full cupboard and having to search through it itself every time.
- The Need class has many methods that describe the standing of the need in terms of quantity available and quantity funded. While they were not used during this sprint as we haven't yet completed our checkout basket functionality, these methods will be useful by alowing the Need object itself to state if it is avaiable to be checked out rather than another class having to deal with the logic of deciding it.

Open/Closed

- During Sprint 2 our team focused on making our Users Open/Closed, meaning that they are 'open for extension, but closed for modification'.
- Our team wrote a User interface to be used when creating types of Users for our website. While at the moment our team has only implemented the User interface in the creation of the Helper class, in futire sprints we see the possibility of including different types of users that will have different abilities and the User interface will be an ideal place to start in creating those new Users.

> ***[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

All aspects of our testing for sprint 3/4 were fantastic. All acceptance tests were passed and both instructions and branches were above 90% coverage. This was a big step up from sprint 2, where our branch percentage was below 90% and not all acceptance tests passed. The team worked very hard to put a stamp of approval and have our tests reflect our confidence.

Acceptance Testing

Sprint 4 Summary Acceptance Testing Summary Every single user story passed all of its acceptance criteria tests(11). Our application works exactly as we expected and planned for. We did not find any bugs in design during our testing.

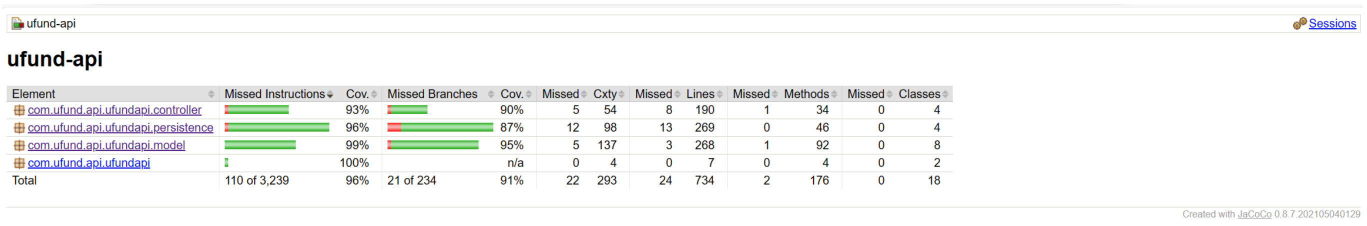
Sprint 2 Summary

User Story	Acceptance Criterion	Sprint 2
As a **helper** I want to see a list of **needs** so that I choose what to contribute to.	Given that I am on the **u-fund** site when I am not on the **Needs** page then I must see a means to navigate to the **Needs** page.	Pass
	Given that I am not on the **Needs** page when I choose the **Needs** page then I am taken to the **Needs** page.	Pass
	Given that I am on the **Needs** page when there are no **needs** in the **cupboard** I see a message indicating that that there are no **needs** available to contribute.	Pass
	Given that I am on the **Needs** page when there are **needs** in the **cupboard** then I see each **need** and short description.	Pass
	Given that I am on the **Needs** page when there are **needs** in the **cupboard** then I see a means to add each **need** to my funding **basket**.	Pass
As a **user** I want to view all of the needs so that I can see what the organization needs.	**Given** I submit a request to the webpage **when** I view an organizations cupboard **then** I view all needs for a given organization	Pass
As a **helper** I want a shopping basket that holds needs so that I can store the things I will fund.	**Given** a user is a helper **when** they view needs **then** I expect them to have a basket to store things.	Pass
	Given that I am a helper on the cupboard page, then I must see a means to naviagte to my basker	Pass
	Given that I am not in my basket, when I choose my basket then I am taken to my basket.	Pass
	Given that I am in my basket I expect to be able to view the needs that I have previously added to the basket	Pass
	Given that I am in my basket I expect to be able to remove needs from my basket	Pass
As a Helper I want to save Needs to my Basket so that I can later fund them	Given that I am a helper on the needs page, when I view a need then I expect to be able to add that need to my basket from the needs page.	Pass
	Given that I have added a need to my basket I expect it to remain in my basket until I remove it	Pass
As a Helper I want to remove a need from my basket so someone else can fund it	Given that I am a helper viewing my basket then I expect to see a means of removing needs from my basket	Pass
	Given that I have requested to remove a need from my basket I expect it to no longer appear listed in my basket	Pass
	Given that I have removed a need from my basket I expect it to remain removed from my basket after I leave and return to the site.	Pass
As a user I want to be able to login to the site so that I can save and view my previous actions	Given a user opens the site, they expect to view a prompt to enter login credentials.	Pass
	Given a user enters Manager login credentials, they expect to be taken to the u-fund manager home page.	Pass
	Given a user enters Helper login credentials, they expect to be taken to the u-fund helper home page.	Pass
	Given a user has been authenticated and taken to their respective homepage, they expect their associated data to be saved from their	Pass

	last interaction with the site.	
As a manager I want to be able to manage the contents of my organizations cupboard.	Given a Manager is viewing their u-fund home screen, they should see a means of navigating to their cupboard.	Pass
	Given a Manager chooses their cupboard, they expect to be taken to their cupboard.	Pass
	Given a Manager is viewing their cupboard, they should see a means of searching through their needs.	Fail
	Given a Manager is viewing their cupboard, they should see a means of selecting a need.	Pass
	Given a Manager has selected a need, they should see a means of editing the selected need.	Pass
	Given a Manager has edited a selected need, they expect the edited version of the need to appear in their cupboard.	Pass
	Given a Manager has edited a need, they expect that need to be removed from any Helper baskets that it was previously in.	Fail
	Given a Manager has selected a need, they should see a means of deleting the selected need.	Pass
	Given a Manager has requested to delete a need, they expect to be prompted with a confirmation that they want to permanently delete the need.	Fail
	Given a Manager confirms they want to delete a need, they expect that need to no longer be shown in their cupboard.	Pass
	Given a Manager confirms they want to delete a need, they expect that need to no longer be shown in any Helper baskets it was previously in.	Pass
	Given a Manager has selected a need, they expect to see a means of retiring the need.	Pass
	Given a Manager has requested to retire a need, they expect to still see the need listed in their cupboard with some indication that it has been retired.	Pass
	Given a Manager has selected a need that is retired, they expect to see a means of unretiring that need.	Pass
	Given a Manager has requested to unretire a need they expect to be able to view it in their cupboard with no indication that it is retired.	Pass
	Given a Manager has requested to unretire a need, they expect it to be visible in their cupboard to Helpers.	Pass
	Given a Manager is viewing their cupboard, they expect to see a means of creating a new need.	Pass
	Given a Manager has requested to create a new need, they expect to view a prompt where they can describe their new need.	Pass
	Given a Manager is viewing the request to create a need they expect to see a means of submitting the new need.	Pass
	Given a Manager submits a complete request to create a new need, they expect the new need to be shown in their cupboard.	Pass
	Given a Manager submits an incomplete request to create a new need, they expect to see a message saying their need could not be created.	Pass
As a Helper I want to fund needs so that I can help that organization	Given a user is logged in as a helper, they should see a means of viewing the cupboard.	Pass
	Given a user is logged in as a helper, they should see a means of viewing the cupboard.	Pass
	Given a helper is attempting to search the cupboard, they should see a means of searching by name, type, or price.	Pass
	Given a helper has submitted a search request, they expect to see a list of Needs from the cupboard that match their search.	Pass

Unit Testing and Code Coverage

[Sprint 3 & 4]



ufund-api Sessions

ufund-api

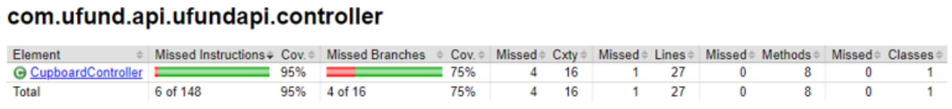
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ufund.api.ufundapi.controller		93%		90%	5	54	8	190	1	34	0	4
com.ufund.api.ufundapi.persistence		96%		87%	12	98	13	269	0	46	0	4
com.ufund.api.ufundapi.model		99%		95%	5	137	3	268	1	92	0	8
com.ufund.api.ufundapi		100%		n/a	0	4	0	7	0	4	0	2
Total	110 of 3,239	96%	21 of 234	91%	22	293	24	734	2	176	0	18

Created with JaCoCo 0.8.7.202105040129

Our coverage for sprint 3 and 4 was levels beyond sprint 2. Instructions was at 96% and branches was at 91%. Our stategy for unit testing was essentially to test everything and anything. Each time a new feature was implemented, we would test the model, followed by persistence, and finish with the controller. This was the best strategy as it mirrored the order in which code was written and also matched the order of dependency. Our model had the highest percentages as it was the easiest to test. We really wanted to hammer this section and make sure the foundation for our code was strong. After that, we moved to testing the persistence tier, which was the most difficult. Many of the persistence methods were full of conditions, which made the branch section hard. The team worked through all of the possible situations that could occur based on input, and used the tests as a way to determine if code could be refactored and made more efficient. We would then finish with the controller tier, which was also very easy after we truly understood mocking. The reason our testing was so effective was because we attacked each component as its own piece, testing each layer sequentially.
We hit above our target percentage of 90%. We did not specifically look to ingore certain peices of code, it really just came down to testing everything to the best of our ability. If it became evident one method was very hard to fully cover, we would acknowledge it and come back to it later if needed. Although we did put a lot of focus on the model, I believe be chose to treat everything equally as we believed that would promote the best coverage in the end, which was reflected.

Sprint 2

Controller Tier



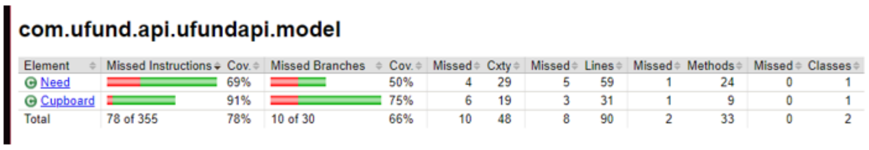
com.ufund.api.ufundapi.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
CupboardController		95%		75%	4	16	1	27	0	8	0	1
Total	6 of 148	95%	4 of 16	75%	4	16	1	27	0	8	0	1

Analysis

Our controller tier is well-tested, though it might potentially benefit from a more thorough set of tests given that our tests overlooked 25% of the branching possibilities.

Model Tier



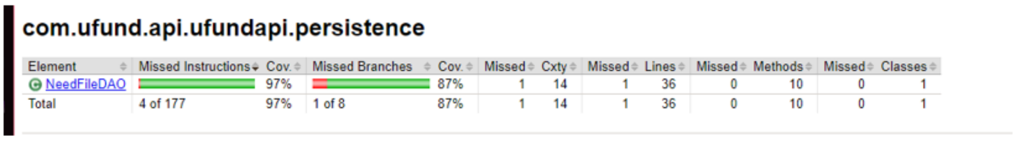
com.ufund.api.ufundapi.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Need		69%		50%	4	29	5	59	1	24	0	1
Cupboard		91%		75%	6	19	3	31	1	9	0	1
Total	78 of 355	78%	10 of 30	66%	10	48	8	90	2	33	0	2

Analysis

Our model tier could benefit from a more through set of tests, our team missed a significant amount of code in the Need class. Although it is important to note that a good number of the missed test cases were simple cases such as missing tests of the toString functions.

Persistence Tier



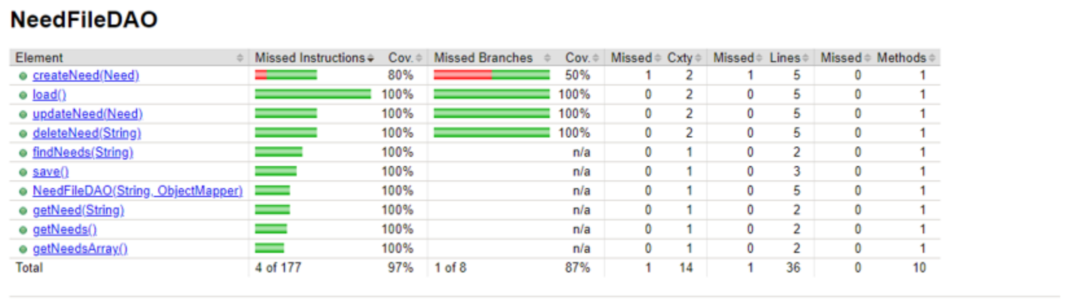
com.ufund.api.ufundapi.persistence

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
NeedFileDAO		97%		87%	1	14	1	36	0	10	0	1
Total	4 of 177	97%	1 of 8	87%	1	14	1	36	0	10	0	1

Analysis

The persistence tier was our most well-tested tier. Our team is happy with our test coverage of the persistence tier.

Well-tested Component



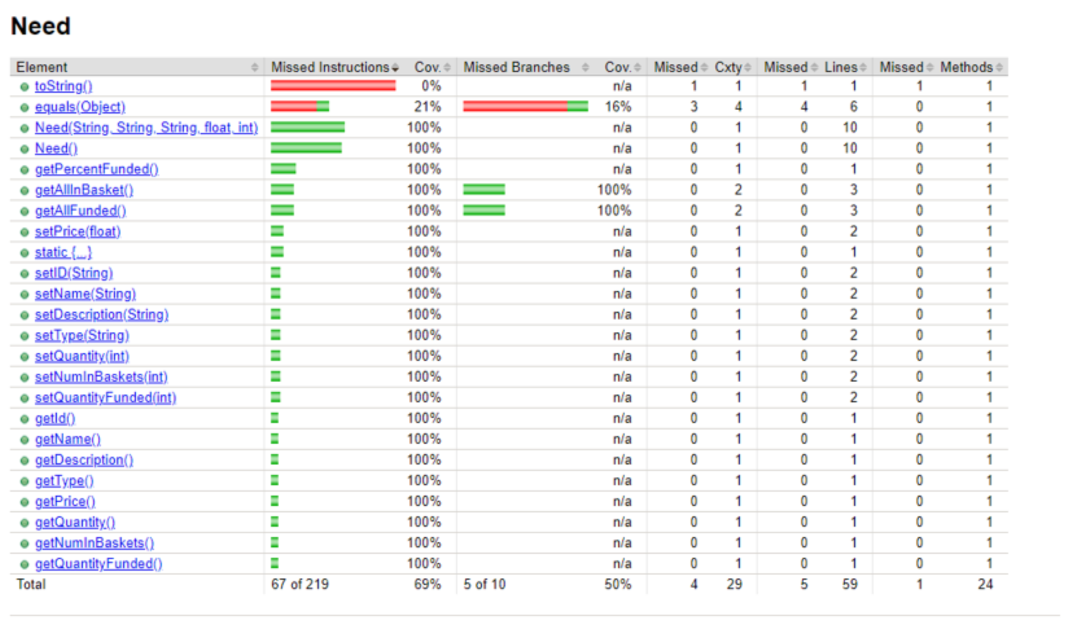
NeedFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
createNeed(Need)		80%		50%	1	2	1	5	0	1
load()		100%		100%	0	2	0	5	0	1
updateNeed(Need)		100%		100%	0	2	0	5	0	1
deleteNeed(String)		100%		100%	0	2	0	5	0	1
findNeeds(String)		100%		n/a	0	1	0	2	0	1
save()		100%		n/a	0	1	0	3	0	1
NeedFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
getNeed(String)		100%		n/a	0	1	0	2	0	1
getNeeds()		100%		n/a	0	1	0	2	0	1
getNeedsArray()		100%		n/a	0	1	0	2	0	1
Total	4 of 177	97%	1 of 8	87%	1	14	1	36	0	10

Analysis

The NeedFileDao was chosen as our example of a well-tested component, we reached a higher percentage of code coverage than we had in any other component. But we should still address the missing branch coverage of the createNeed method.

Poorly tested Component



Need

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		0%		n/a	1	1	1	1	1	1
equals(Object)		21%		16%	3	4	4	6	0	1
Need(String, String, String, float, int)		100%		n/a	0	1	0	10	0	1
Need()		100%		n/a	0	1	0	10	0	1
getPercentFunded()		100%		n/a	0	1	0	1	0	1
getAllInBasket()		100%		100%	0	2	0	3	0	1
getAllFunded()		100%		100%	0	2	0	3	0	1
setPrice(float)		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
setID(String)		100%		n/a	0	1	0	2	0	1
setName(String)		100%		n/a	0	1	0	2	0	1
setDescription(String)		100%		n/a	0	1	0	2	0	1
setType(String)		100%		n/a	0	1	0	2	0	1
setQuantity(int)		100%		n/a	0	1	0	2	0	1
setNumInBaskets(int)		100%		n/a	0	1	0	2	0	1
setQuantityFunded(int)		100%		n/a	0	1	0	2	0	1
getId()		100%		n/a	0	1	0	1	0	1
getName()		100%		n/a	0	1	0	1	0	1
getDescription()		100%		n/a	0	1	0	1	0	1
getType()		100%		n/a	0	1	0	1	0	1
getPrice()		100%		n/a	0	1	0	1	0	1
getQuantity()		100%		n/a	0	1	0	1	0	1
getNumInBaskets()		100%		n/a	0	1	0	1	0	1
getQuantityFunded()		100%		n/a	0	1	0	1	0	1
Total	67 of 219	69%	5 of 10	50%	4	29	5	59	1	24

Analysis

The Need class was chosen as our team's poorly tested class as we almost completely missed covering the equals method. Our team needs to return to these tests and ensure that our equals function is thoroughly tested.