ANIMAL AUXILIARIES Design Documentation

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

Our U-Fund website exists to serve small local animal shelters that are in need of a low-cost accessible form of fundraising. Our top goal is to create a website that is accessible and easily navigable to an average community in order to crowd fund for a non-profit through the average person.

Glossary 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*.
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*. They also have the ability to view both community board posts and adoptable animals, and **adopt** animals not on hold

Term	Definition
Adoptable Animal	A single animal that is in the adoption cupboard, who is either adoptable or on hold(not adoptabke)
Adoption Cupboard	Virtual board of adoptable animals
Community Board	Virtual list of text pagraphs **posted** by managers for helpers to view
Post	Individal text paragraph posted in community board by a manager
Adoption Cupboard	Secondary cupboard viewable by Helpers and editable by Managers containing Adoptable Animals

Requirements

Implementation and Adequate Display of the following features:

- Authentication for Helper/U-fund Manager login/logout & grants the correct user abilities based on the status of the logged in user

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 checkout their funding basket.

Managers have the following abilities:

- Manager(s) can add, remove and edit the data of all their needs stored in their needs cupboard.
- 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 Page
- Community Board

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

Epic	User Stories
User Management	User Authentication, Manager Abilities, Helper Abilities
Funding Basket	Add need to basket, Remove need from basket, checkout funding basket
Frontend Angular	Home Page, Manager Page, Helper Page, Basket Page

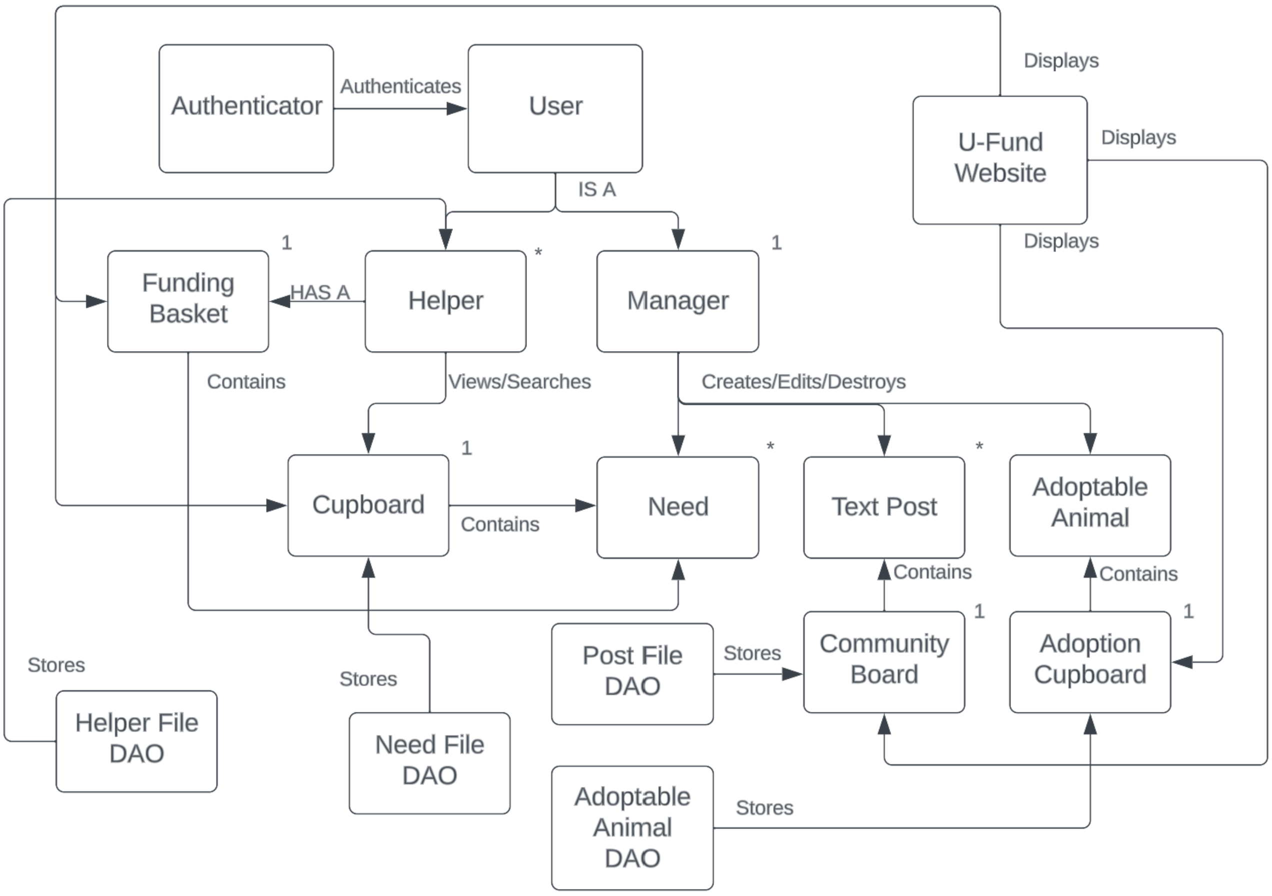
Enhancements

Community Board
The Community Board displays chronological text posts on the home page of the website. Posts can be created and deleted by managers. The purpose of the community board is to act as an engagement tool for Managers to share progress and upcoming events with their helpers as well as potential helpers that have not yet committed to creating a profile on the website. As well as to foster a sense of community among the website users and grant the Helpers insight into the good their donations are doing for the animal shelter.

Adoption Page The Adoption Page stores a collection of adoptable animals. The page can be edited by the Manager and viewed by the Helpers. Using the adoption page, Helpers can view profiles for each of the available adoptable animals as well as begin the adoption process through the U-Fund website. When a Helper begins the adoption process, that animal is placed on a hold for the Helper and no other website user will be able to begin the adoption process on that same animal. The primary purpose of the Adoption Page is not to provide an entirely online adoption process, but to better engage potential adopters and allow them to view the profiles of many avialable adoptable animals at a given time. The formal adoption process will take place outside of the U-Fund website, but the website provides an accessible means of beginning the process.

Application Domain

This section describes the application domain.

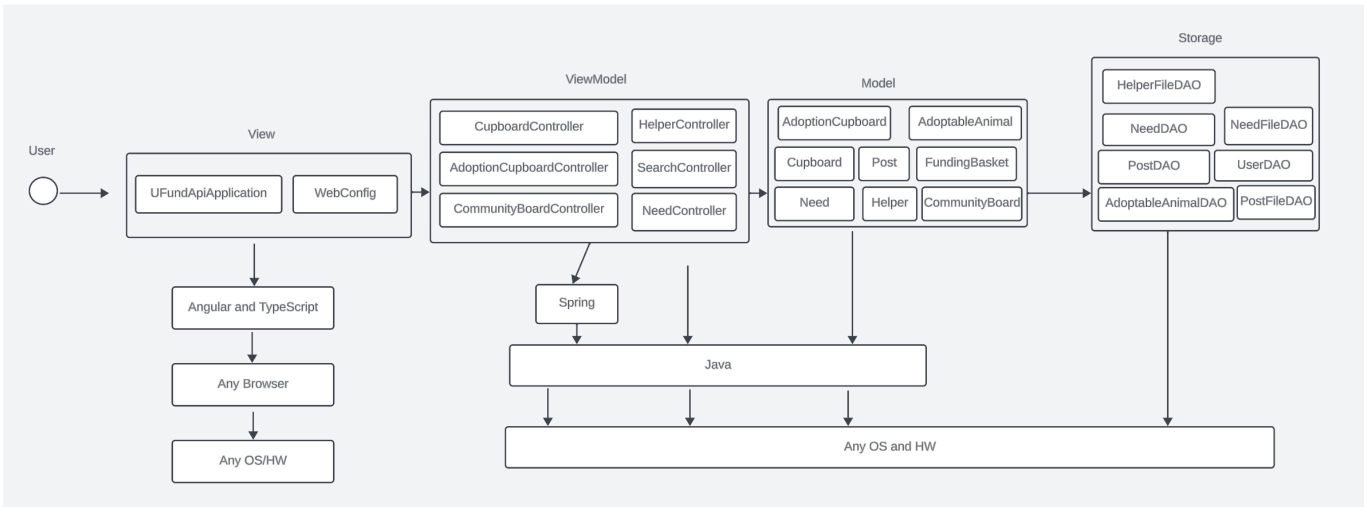


Architecture and Design

This section describes the application architecture.

Summary

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



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.

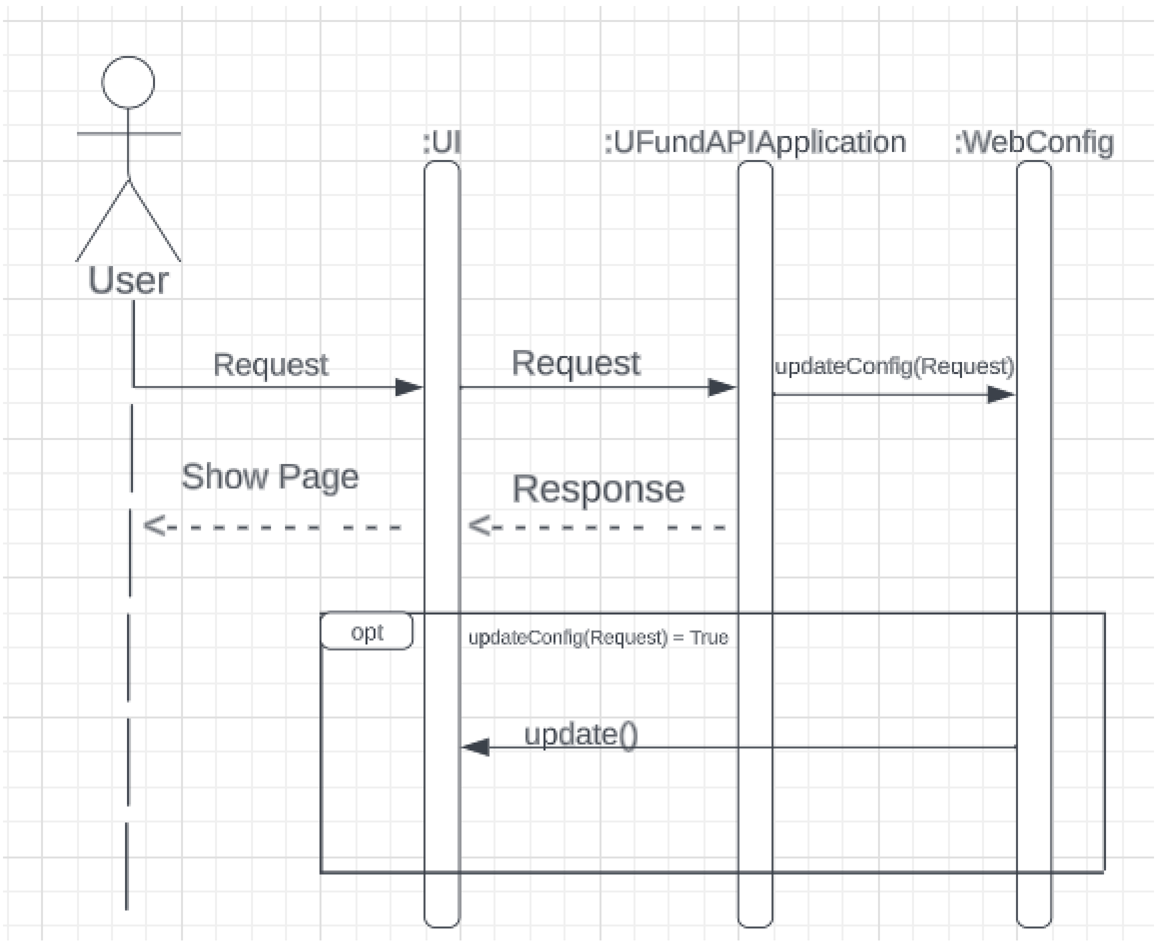
The application opens on a login page 'http://localhost:4200/login', logging in as an admin will take the user to the manager page 'http://localhost:4200/manager'; logging in as a helper takes you to the helper page 'http://localhost:4200/helper']

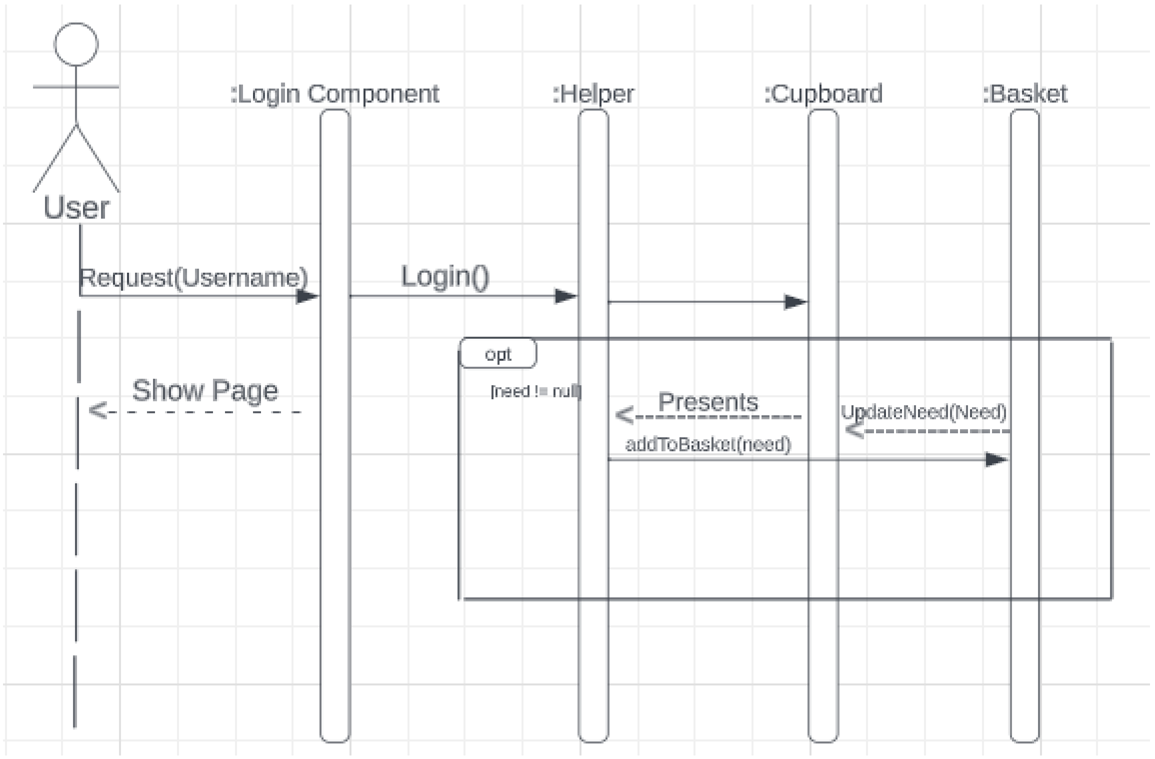
Once on the manager page the user sees a 'logout' button at the top of the page was a create need textbox and a list of needs in the cupboard underneath the logout button and a heading 'My Needs'. A manager can create a need by entering a name into the 'Need name: ' labeled text box and clicking the '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

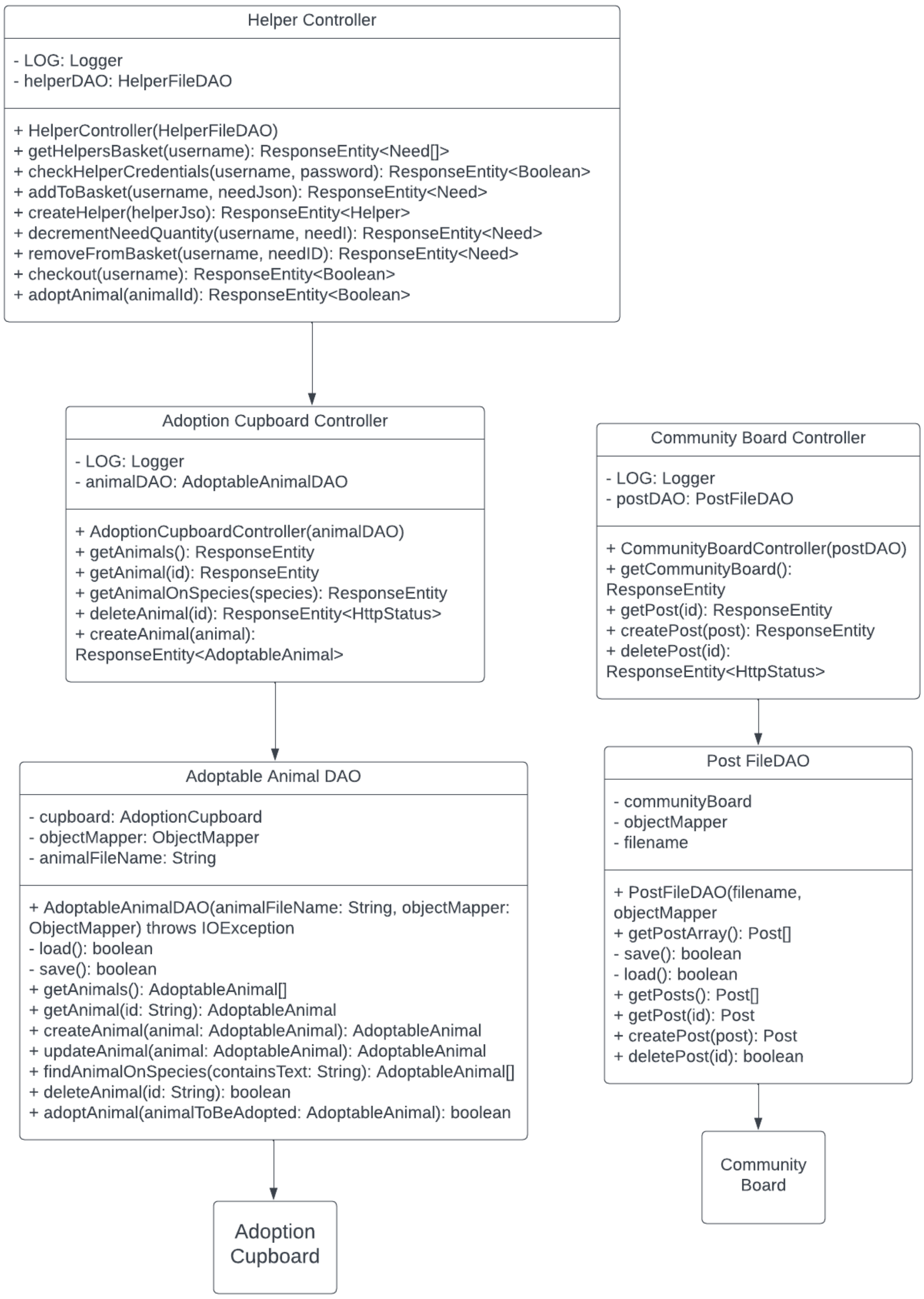
The View Tier UI of our architecture uses two key components. Those being the UFundAPIApplication and our WebConfig. As the user interacts with our UI the API acts as the interpreter for those requests. Then if the API requests any changes be made to the UI it calls upon the WebConfig to make those modifications on the fly allowing flexability and a quick response time.

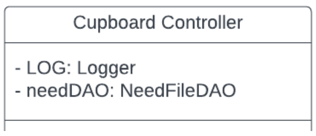


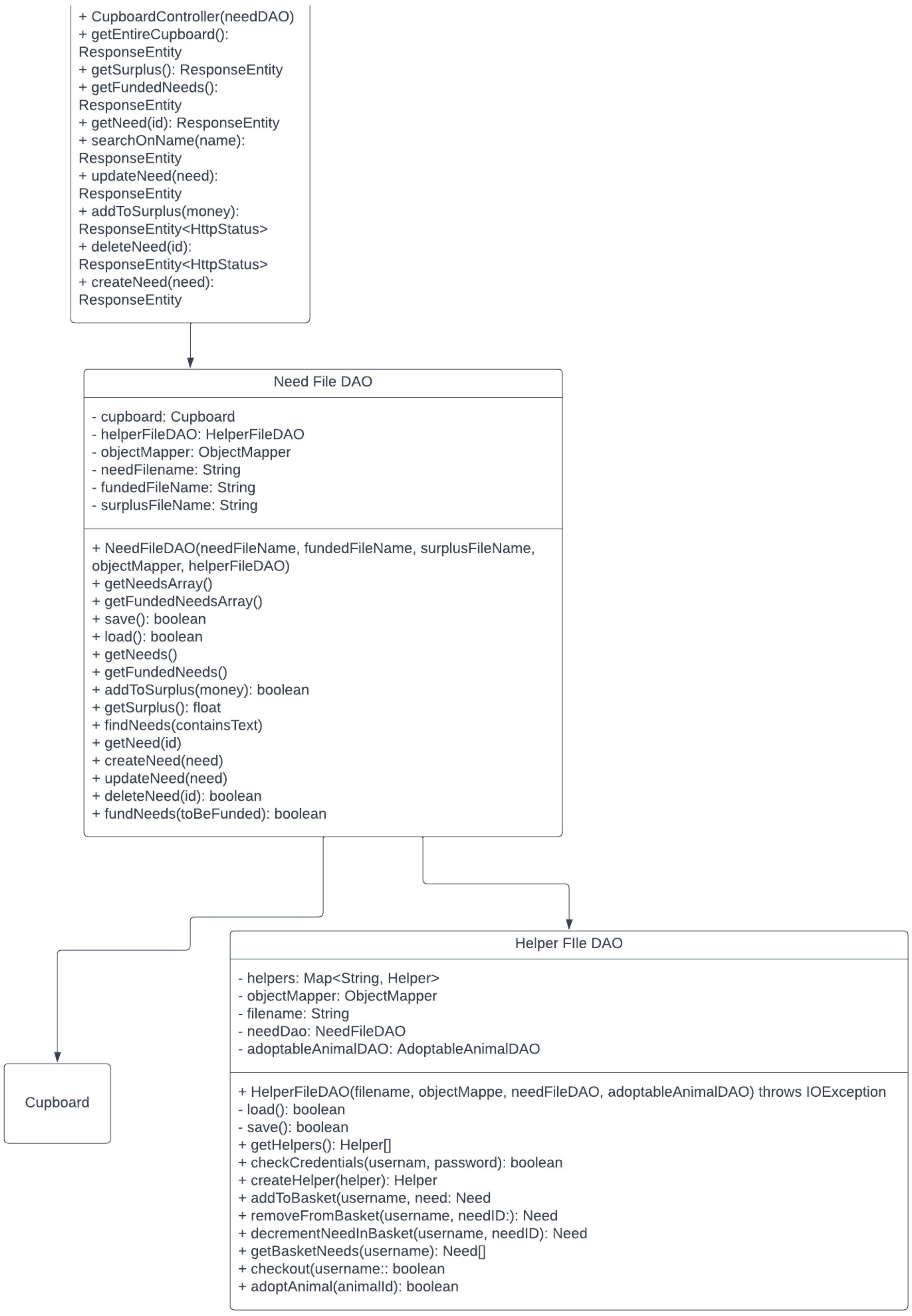


ViewModel Tier

The View Model tier of our architecture consists of 10 components. They are NeedDAO, NeedFileDAO, PostDAO, PostFileDAO, UserFileDAO, AdoptableAnimalDAO, HelperController, CupboardController, CommunityBoardController, and AdoptionCupboardController. This diagram has a repeating sequence among the DAOs, FileDAO, and Controller. The FileDAOs are all based on the architecture of their respective DAOs. Then once created the DAOs parse their respective JSON files into values that can be passed to their respective controllers. Essentially the DAOs act as the bridge between the data stored in the JSONs and the controller which is in charge of changing this data and keeping track of it.

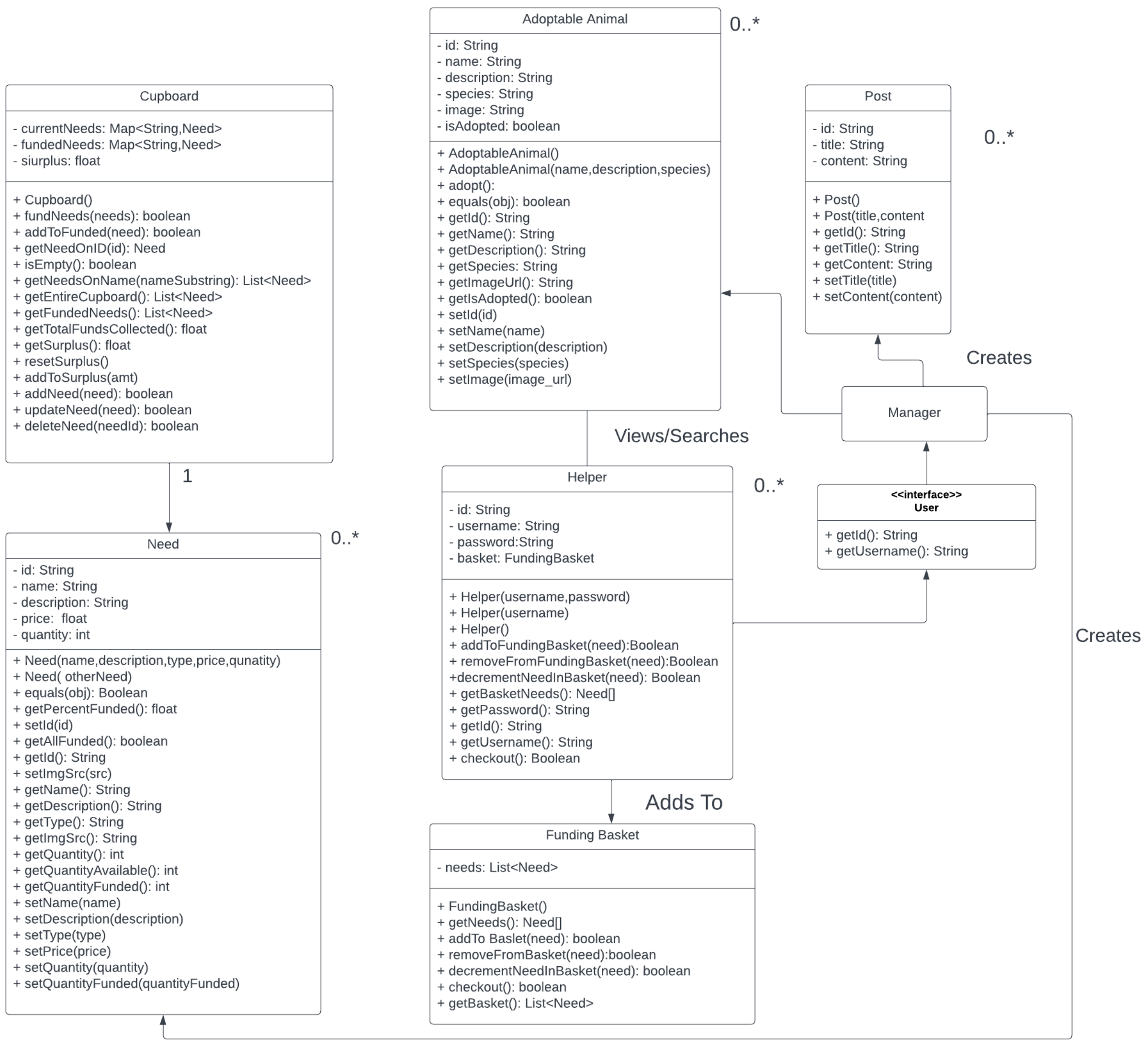

Helper Controller
- LOG: Logger
- helperDAO: HelperFileDAO
+ HelperController(HelperFileDAO)
+ getHelpersBasket(username): ResponseEntity<Need[]>
+ checkHelperCredentials(username, password): ResponseEntity<Boolean>
+ addToBasket(username, needJson): ResponseEntity<Need>
+ createHelper(helperJso): ResponseEntity<Helper>
+ decrementNeedQuantity(username, needI): ResponseEntity<Need>
+ removeFromBasket(username, needID): ResponseEntity<Need>
+ checkout(username): ResponseEntity<Boolean>
+ adoptAnimal(animalId): ResponseEntity<Boolean>
Adoption Cupboard Controller
- LOG: Logger
- animalDAO: AdoptableAnimalDAO
+ AdoptionCupboardController(animalDAO)
+ getAnimals(): ResponseEntity
+ getAnimal(id): ResponseEntity
+ getAnimalOnSpecies(species): ResponseEntity
+ deleteAnimal(id): ResponseEntity<HttpStatus>
+ createAnimal(animal): ResponseEntity<AdoptableAnimal>
Community Board Controller
- LOG: Logger
- postDAO: PostFileDAO
+ CommunityBoardController(postDAO)
+ getCommunityBoard(): ResponseEntity
+ getPost(id): ResponseEntity
+ createPost(post): ResponseEntity
+ deletePost(id): ResponseEntity<HttpStatus>
Adoptable Animal DAO
- cupboard: AdoptionCupboard
- objectMapper: ObjectMapper
- animalFileName: String
+ AdoptableAnimalDAO(animalFileName: String, objectMapper: ObjectMapper) throws IOException
- load(): boolean
- save(): boolean
+ getAnimals(): AdoptableAnimal[]
+ getAnimal(id: String): AdoptableAnimal
+ createAnimal(animal: AdoptableAnimal): AdoptableAnimal
+ updateAnimal(animal: AdoptableAnimal): AdoptableAnimal
+ findAnimalOnSpecies(containsText: String): AdoptableAnimal[]
+ deleteAnimal(id: String): boolean
+ adoptAnimal(animalToBeAdopted: AdoptableAnimal): boolean
Post FileDAO
- communityBoard
- objectMapper
- filename
+ PostFileDAO(filename, objectMapper
+ getPostArray(): Post[]
- save(): boolean
- load(): boolean
+ getPosts(): Post[]
+ getPost(id): Post
+ createPost(post): Post
+ deletePost(id): boolean
Adoption Cupboard
Community Board


Cupboard Controller
- LOG: Logger
- needDAO: NeedFileDAO


+ CupboardController(needDAO)
+ getEntireCupboard(): ResponseEntity
+ getSurplus(): ResponseEntity
+ getFundedNeeds(): ResponseEntity
+ getNeed(id): ResponseEntity
+ searchOnName(name): ResponseEntity
+ updateNeed(need): ResponseEntity
+ addToSurplus(money): ResponseEntity<HttpStatus>
+ deleteNeed(id): ResponseEntity<HttpStatus>
+ createNeed(need): ResponseEntity
Need File DAO
- cupboard: Cupboard
- helperFileDAO: HelperFileDAO
- objectMapper: ObjectMapper
- needFilename: String
- fundedFileName: String
- surplusFileName: String
+ NeedFileDAO(needFileName, fundedFileName, surplusFileName, objectMapper, helperFileDAO)
+ getNeedsArray()
+ getFundedNeedsArray()
+ save(): boolean
+ load(): boolean
+ getNeeds()
+ getFundedNeeds()
+ addToSurplus(money): boolean
+ getSurplus(): float
+ findNeeds(containsText)
+ getNeed(id)
+ createNeed(need)
+ updateNeed(need)
+ deleteNeed(id): boolean
+ fundNeeds(toBeFunded): boolean
Cupboard
Helper FIle DAO
- helpers: Map<String, Helper>
- objectMapper: ObjectMapper
- filename: String
- needDao: NeedFileDAO
- adoptableAnimalDAO: AdoptableAnimalDAO
+ HelperFileDAO(filename, objectMappe, needFileDAO, adoptableAnimalDAO) throws IOException
- load(): boolean
- save(): boolean
+ getHelpers(): Helper[]
+ checkCredentials(usernam, password): boolean
+ createHelper(helper): Helper
+ addToBasket(username, need: Need
+ removeFromBasket(username, needID:): Need
+ decrementNeedInBasket(username, needID): Need
+ getBasketNeeds(username): Need[]
+ checkout(username:: boolean
+ adoptAnimal(animalId): boolean

Model Tier

Our model tier consists of 7 main elements; Cupboard, Need, Helper, Adoptable Animal, Funding Basket, Post, and User. Starting at the beginning if a user is a Helper or manager they have access to a few different things. Helper's may add Needs to their funding basket from the cupboard to be checked out and funded. Helpers may also view and adopt animals and also view posts. If a user is a manager they have access to create these things. Managers may create posts, needs, and adoptable animals.



OO Design Principles

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.
- When adding the enchancements during Sprint 3, we added two more controller classes. The adoptionCupboardController was made with the sole responsibilty of handling the adoptionCupboard and adding/removing adoptableAnimals from the cupboard. The communityBoardController was similar in that it was responsable for handling Posts made and added to the communityBoard.These controllers helped us keep our classes small and efficient

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
- Across Sprint 2 and 3, we continued to use controllers as nessesary to keep the logic of the application far from the user interface. For every major component, a controller was made to help organize and streamline code and logic.

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 future 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.
- We continued using interfaces across the project, especially within the persistance tier. These DAO interfaces creating a jumping off point for creating classes that extended them and also prevented us from adding more than what was nessasary to these classes. It kept our code from getting bloated and difficult to read through.

Sprint 3 OO Design Principles

Low Coupling

- During Sprint 3, our team focused on how classes interacted with eachother and consistently questioned whether one class was holding too much responsability. We focused on low coupling as a way to ease production and maintainability of the code as well as making the code easier to understand.

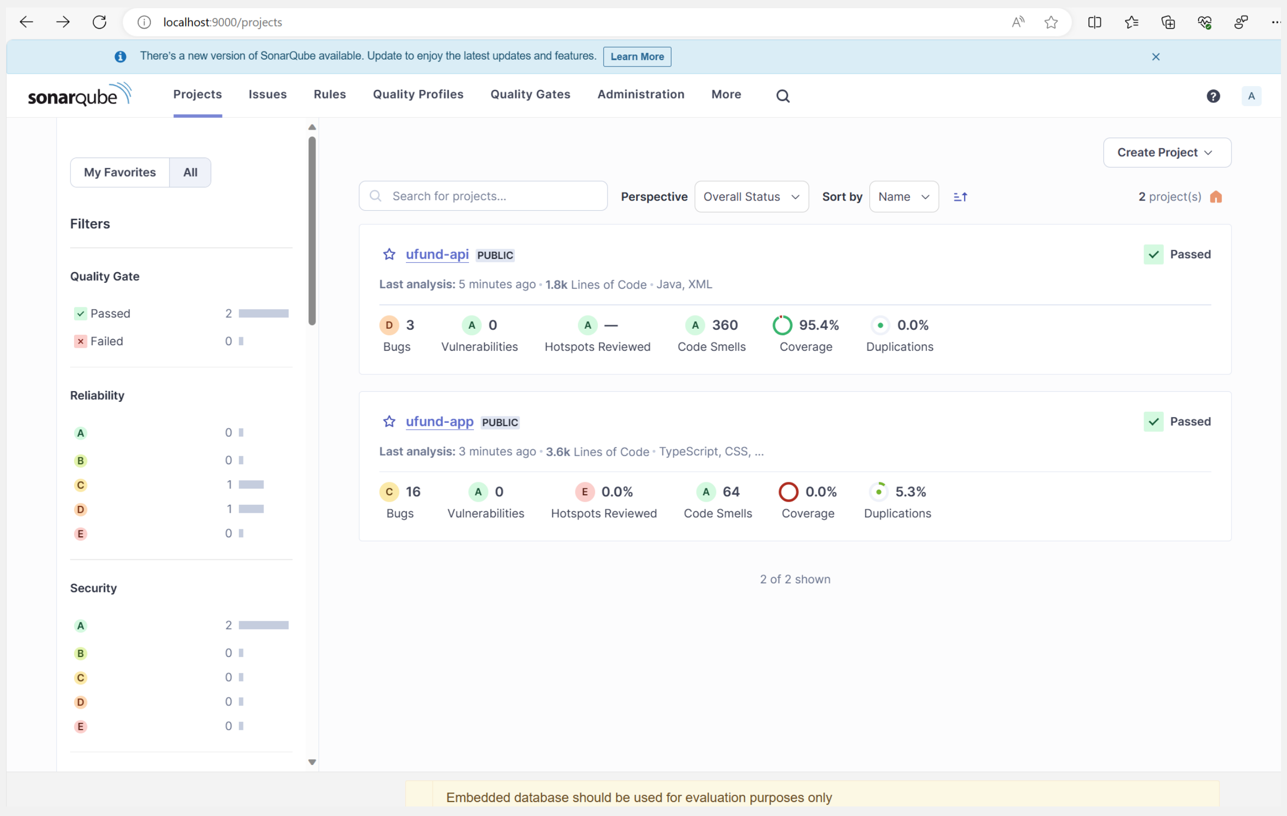
- Because we all worked simultaneously and we all wrote code meant to interact with another person's, low coupling was of greater importance so that we could make the learning process and implementation of different code easier. Low coupling can easily be observed across all tiers, but specifically in the model tier where we seperated the communityBoard class from the Post class. This low coupling allowed for the communityBoard to act as a holder for the posts made by the Post class instead of having it do both which would cause confusion and difficulty when it came to properly implementing the code.

Pure Fabrication

- In our goal to improve our applications design, we chose to focus on Pure Fabrication as another prinicple in Sprint 3. Our team believed it to be important not only as a way to achieve further Low Coupling but as a way to simplify more complex systems in our application.
- In the persistance tier, we made interfaces to be implemented into classes in order to help outline everything the class needed to complete. While these DAO interfaces weren't directly tied to the domain, they served as helpers to help connect classes together

Static Code Analysis/Future Design Improvements

!!!!!!



There's a new version of SonarQube available. Update to enjoy the latest updates and features.

Filters

Quality Gate

✓ Passed	2
✗ Failed	0

Reliability

A	0
B	0
C	1
D	1
E	0

Security

A	2
B	0
C	0
D	0
E	0

Search for projects... Perspective Overall Status Sort by Name 2 project(s)

ufund-api PUBLIC — Passed
Last analysis: 5 minutes ago · 1.8k Lines of Code · Java, XML

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications
D 3	A 0	A —	A 360	95.4%	0.0%

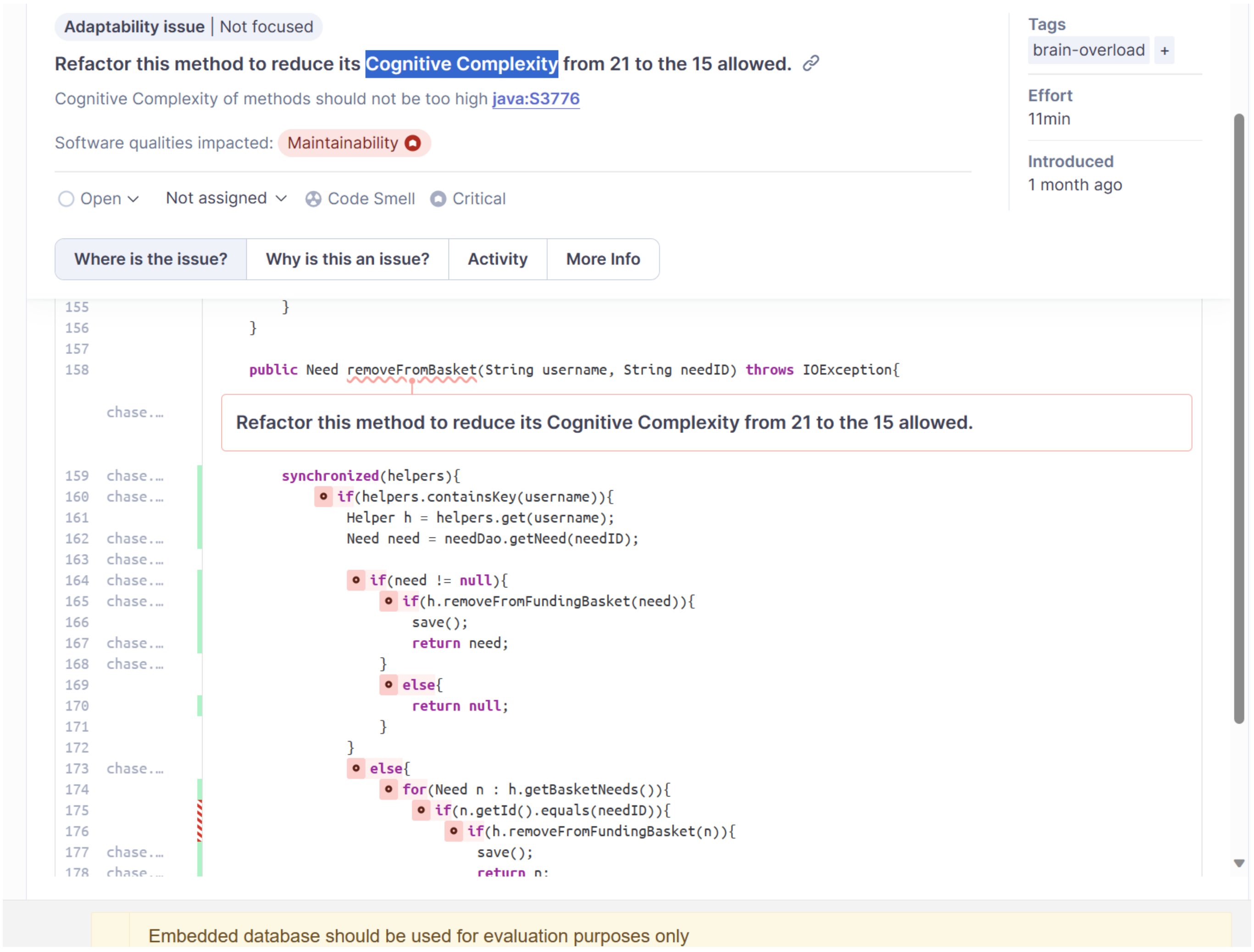
ufund-app PUBLIC — Passed
Last analysis: 3 minutes ago · 3.6k Lines of Code · TypeScript, CSS, ...

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications
C 16	A 0	E 0.0%	A 64	0.0%	5.3%

2 of 2 shown

Embedded database should be used for evaluation purposes only

Issue 1 - Cognitive Complexity


Adaptability issue | Not focused

Refactor this method to reduce its Cognitive Complexity from 21 to the 15 allowed.

Cognitive Complexity of methods should not be too high java:S3776

Software qualities impacted: Maintainability

Open · Not assigned · Code Smell · Critical

Tags: brain-overload · Effort: 11min · Introduced: 1 month ago

```
            }
        }

    public Need removeFromBasket(String username, String needID) throws IOException{
```

Refactor this method to reduce its Cognitive Complexity from 21 to the 15 allowed.

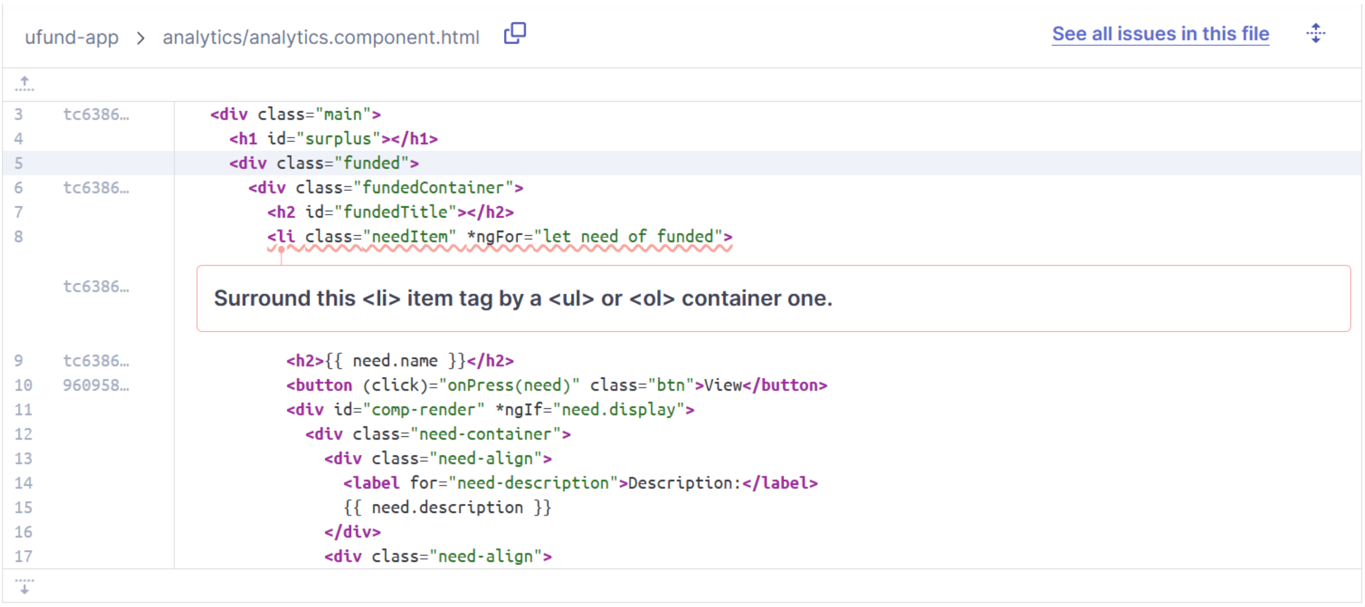
```
        synchronized(helpers){
            if(helpers.containsKey(username)){
                Helper h = helpers.get(username);
                Need need = needDao.getNeed(needID);

                if(need != null){
                    if(h.removeFromFundingBasket(need)){
                        save();
                        return need;
                    }
                    else{
                        return null;
                    }
                }
                else{
                    for(Need n : h.getBasketNeeds()){
                        if(n.getId().equals(needID)){
                            if(h.removeFromFundingBasket(n)){
                                save();
                                return n;
```

Embedded database should be used for evaluation purposes only

This issue surrounds the cognitive complexity of helperFileDAO's removeFromBasket function. The complexity is 21, 6 over the 15 alloted complexity. The problem is that the method is very complex and hard to understand, making maintenance difficult. I believe this issue may appear in some other similar methods from this class. These methods were hard to streamline because we were contstantly updating and changing the logic of how they worked. We would recommend going back into the code and refactoring the approach. Breaking the method into pieces would be very effective here, as removeFromBasket contain multiple steps that could be seperated into different methods. This would signifigantly increase the readability and ability to maintain this code.

Issue 2 - Public Test Methods This issue does not need photo context. We are recieving low risk messages for basically every test we have written instructing us to remove the public modifier from the test methods. The 'why' reads as follows: "JUnit5 is more tolerant regarding the visibilities of Test classes than JUnit4, which required everything to be public. In this context, JUnit5 test classes can have any visibility but private, however, it is recommended to use the default package visibility, which improves readability of code." This appears to be a very small issue and in our opinion is not hurting us in any way. Our code has these tests with the public modifier as our main unit test resource was from the heroes-api project, which has all public test methods. This is likely due to the code being written prior to JUnit 5. With that being said, it would not hurt us to go in and remove all of these public modifiers.

Issue 3 - List Item


ufund-app > analytics/analytics.component.html

See all issues in this file

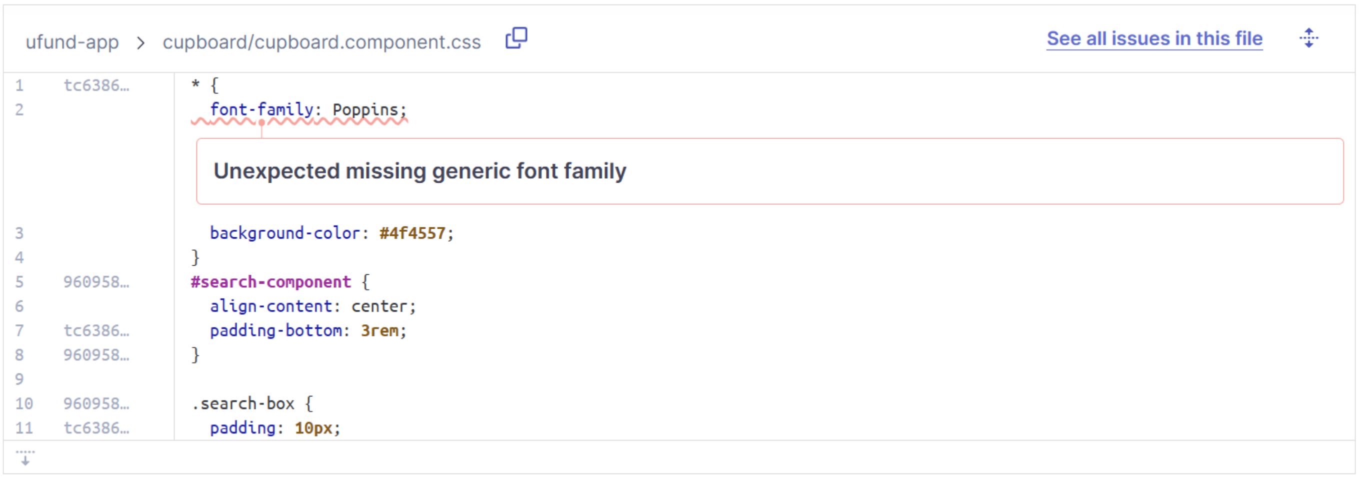
```
tc6386…   <div class="main">
            <h1 id="surplus"></h1>
            <div class="funded">
tc6386…       <div class="fundedContainer">
                <h2 id="fundedTitle"></h2>
                <li class="needItem" *ngFor="let need of funded">
```

tc6386… Surround this <li> item tag by a <ul> or <ol> container one.

```
tc6386…           <h2>{{ need.name }}</h2>
960958…           <button (click)="onPress(need)" class="btn">View</button>
                  <div id="comp-render" *ngIf="need.display">
                    <div class="need-container">
                      <div class="need-align">
                        <label for="need-description">Description:</label>
                        {{ need.description }}
                      </div>
                      <div class="need-align">
```

This issue appears because we have a list item outside of a list container. This is a small but notable issue. It does not make sense for the code to be this way and could confuse a developer. A person looking at this for the first time may thing the code was not finished or had bugs in it. This could be quickly fixed by going in and placing the item inside a unordered list block.

Issue 4 - Missing Generic Font


ufund-app > cupboard/cupboard.component.css

See all issues in this file

```
tc6386…  * {
           font-family: Poppins;
```

Unexpected missing generic font family

```
           background-color: #4f4557;
         }
960958…  #search-component {
           align-content: center;
tc6386…    padding-bottom: 3rem;
960958…  }

960958…  .search-box {
tc6386…    padding: 10px;
```

This issue is regarding a lack of including a generic font. Multiple of our css sections for font styles do not include a generic font to fall back on. This results in a situation where if a browser does not support the fonts listed, the css will fall back on the browsers default font. This could cause a worse outcome if this situation were to occur. It would be suggested to include generic fonts to fall back on where we would still be pleased with the stlye if this situation were to occur.

Future Refactoring

If the team had additional time for improvement, there would definitely be some changes made. When it comes to the logic of our code, we are very pleased and proud of the product we put out. We feel we met the requirements and had logic that functioned at intended. This in saying the changes on the backend we would like to make are not logic focused, rather simplification and efficiency based. There are defintely some methods that could use streamlining or refactoring to make code more readable and efficient(see static code analysis issue 1). We also would like to have been able to scrub through the code and update comments, remove things that no longer are needed or relevant, and polish the classes.

In regards to the front end, we are also happy with the look and feel. Some comments from other teams indicated some minor inconsistencies with margins and white space. It was also suggested to us in the usability review that we update some of the small button animations and polish the look of the manager page and need page. These fixes are not focused on the way it works, rather the spacing and locations on the page. These suggested fixes are pretty small, but they are definitely things we would like to implement.
To wrap up these thoughts, we were very happy with our project, but would like to make finishing touches on some aspects of the site and really add a final polish.

Recommendations for Improvement

Software Architecture

Our cohesion wasn't perfect on the frontend as there were elements we deemed too insignificant for us to create it's own individual componenet class. For example, we decided not to break down various the homepage into too many parts. We decided not to break down the homepages' components such as the community board, the brief description and the login section as weren't going to reuse it in other sections of our website.
When given the chance, we would like to focus on adopting a component-based architecture, breaking down UI elements into reusable components irrespective of immediate use or reusability which will ensure scalability and assist with maintenance.

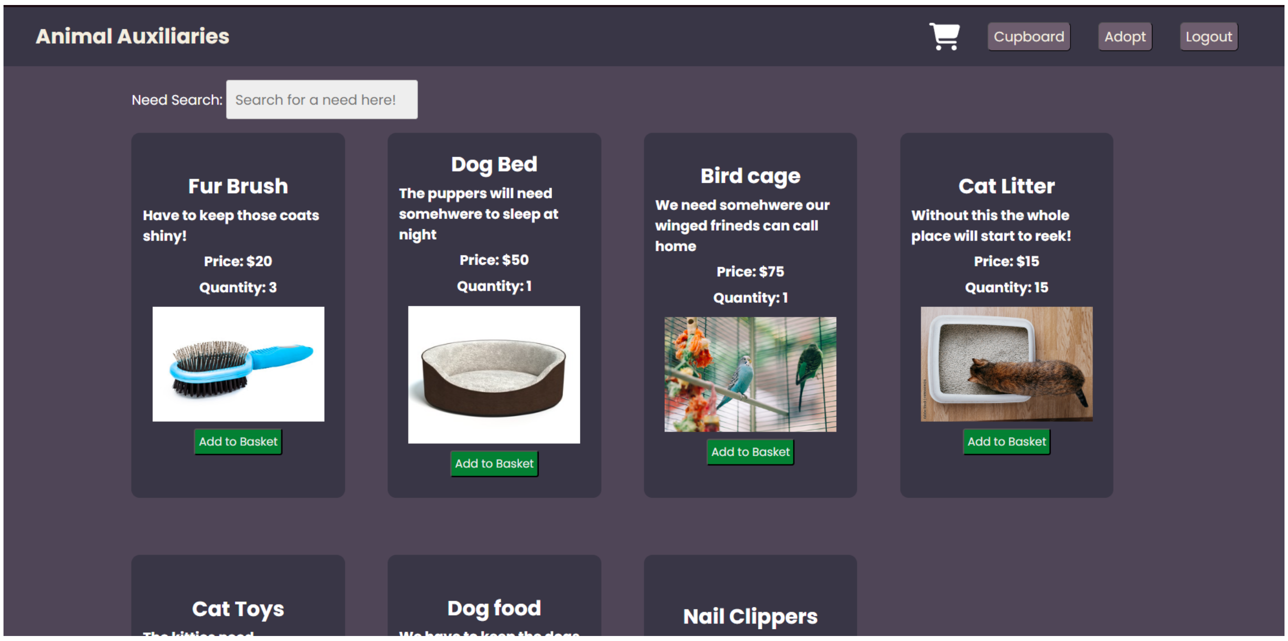
Usability [Front End] Changes

Based on usability feedback, the website needs some fixes on the strategic use of color and texture. Our reviewers suggested changing all text colors to white (currently black) to make the text more readable. This is a good suggestion and would likely increase the readability. Another area of usability we could work on is the purposefulness of page layout. It appears that in some locations on the helper and manager page the margins are not always consistent. We could work on developing the spacial relationships and placing emphasis on the more important areas of each page. Another suggestion from our testers was including a message to indicate if a user is a helper or manager; this would mshow the system communicating what has happened after a login. Below are some example fixes we could make to increase our usability.

Helper Page

- Set the need boxes to uniform sizes

- Set the need text to white



Fur Brush
Have to keep those coats shiny!
Price: $20
Quantity: 3

Dog Bed
The puppers will need somehwere to sleep at night
Price: $50
Quantity: 1

Bird cage
We need somehwere our winged frineds can call home
Price: $75
Quantity: 1

Cat Litter
Without this the whole place will start to reek!
Price: $15
Quantity: 15

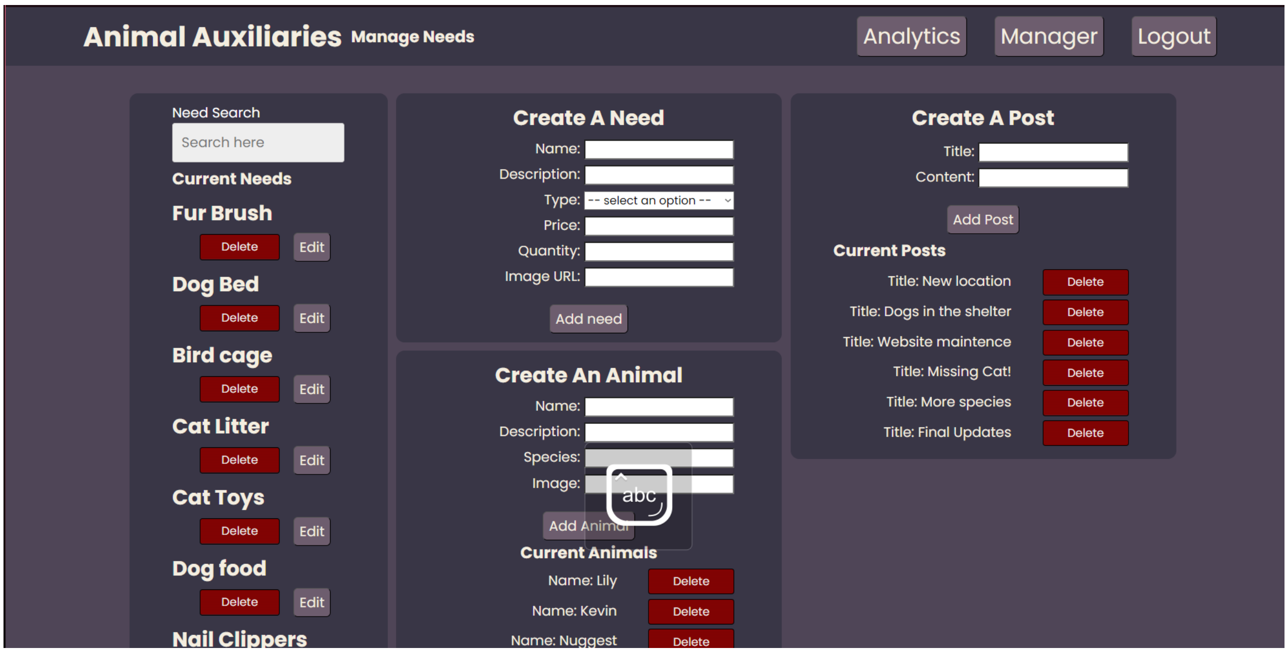
Cat Toys

Dog food

Nail Clippers

Manager Page

- Add text to indicate that you are on the admin page



Fur Brush
Dog Bed
Bird cage
Cat Litter
Cat Toys
Dog food
Nail Clippers

Name: Lily
Name: Kevin
Name: Nuggest

Title: New location
Title: Dogs in the shelter
Title: Website maintence
Title: Missing Cat!
Title: More species
Title: Final Updates

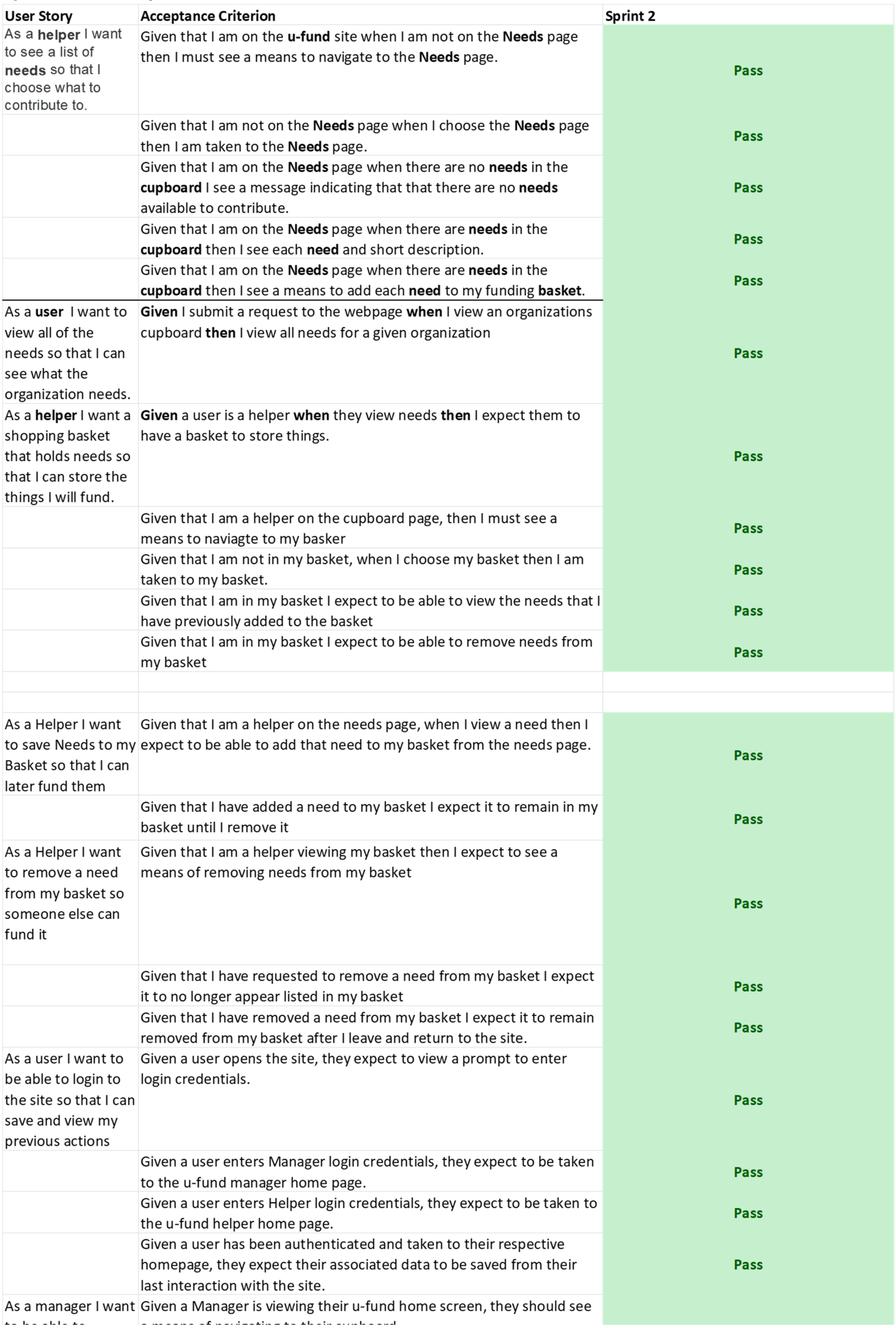
Testing

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

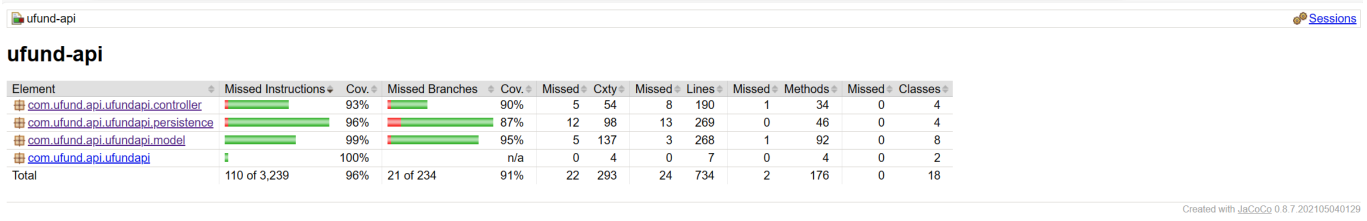


User Story	Acceptance Criterion	Sprint 2
As a **helper** I want to see a list of **needs** so that I choose what to contribute to.	Given that I am on the **u-fund** site when I am not on the **Needs** page then I must see a means to navigate to the **Needs** page.	Pass
	Given that I am not on the **Needs** 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 last interaction with the site.	Pass
As a manager I want	Given a Manager is viewing their u-fund home screen, they should see	

to be able to manage the contents of my organizations cupboard.	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

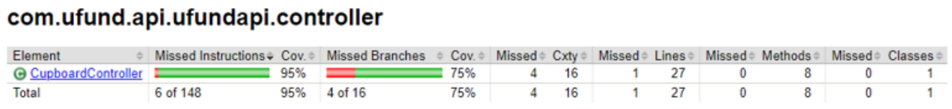
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ufund.api.ufundapi.controller		93%		90%	5	54	8	190	1	34	0	4
com.ufund.api.ufundapi.persistence		96%		87%	12	98	13	269	0	46	0	4
com.ufund.api.ufundapi.model		99%		95%	5	137	3	268	1	92	0	8
com.ufund.api.ufundapi		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 strategy 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



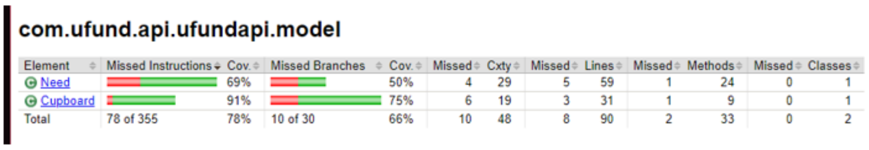
com.ufund.api.ufundapi.controller

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



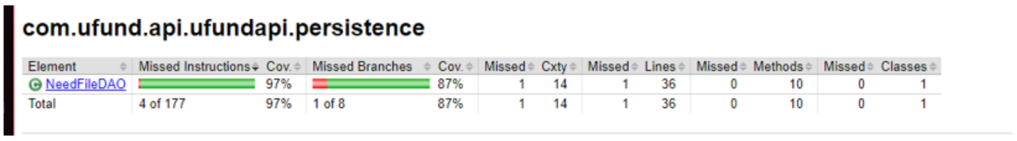
com.ufund.api.ufundapi.model

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



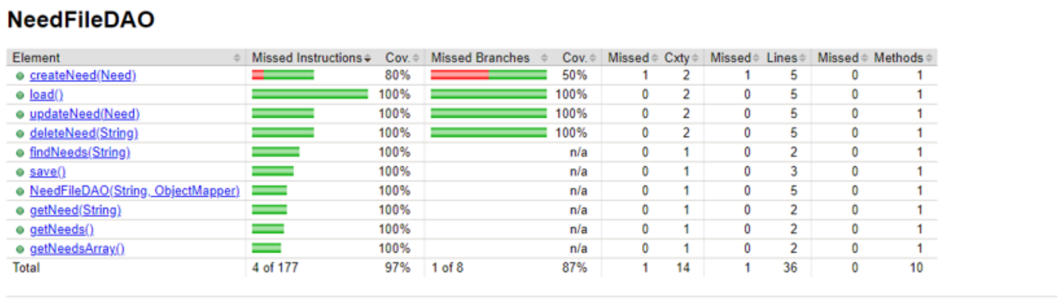
com.ufund.api.ufundapi.persistence

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



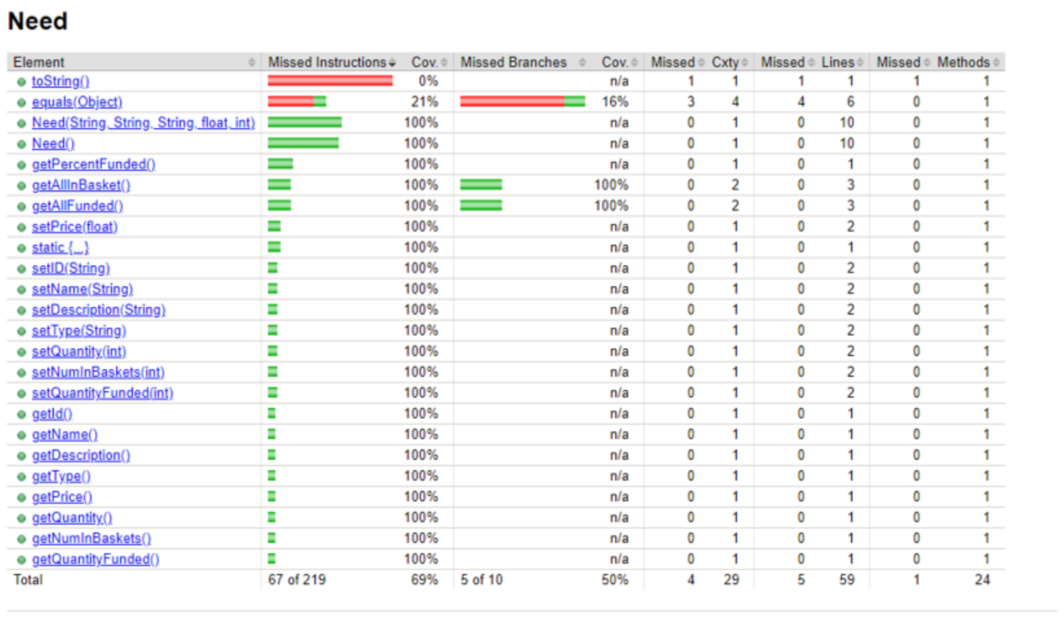
NeedFileDAO

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.