Bartagamen

Team 2 Bartagamenschen

Janik

Bryce

Race

**Software Design Specification**

**Document**

**Version: (A)** **Date: (05/17/2022)**

Table of Contents

[1 Introduction](#_heading=h.gjdgxs) **5**

[1.1 Goals and objectives](#_heading=h.30j0zll) 5

[1.2 Statement of system scope](#_heading=h.1fob9te) 5

[1.3 Definitions and Acronyms](#_heading=h.2et92p0) 5

[1.3.1 Bart](#_heading=h.6ep30wx3mpmr) 5

[1.3.2 UI](#_heading=h.t9sni3nihznx) 5

[2 Architectural design](#_heading=h.tyjcwt) **6**

[2.1 System Architecture](#_heading=h.3dy6vkm) 6

[2.2 Design Rational](#_heading=h.1t3h5sf) 7

[3 Key Functionality design](#_heading=h.4d34og8) **7**

[3.1 Add Pet Feature](#_heading=h.2s8eyo1) 7

[3.1.1 Add Pet Use Cases](#_heading=h.17dp8vu) 7

[3.1.2 Processing Sequence for Add Pet](#_heading=h.5nmfqwozog3p) 9

[3.1.3 Structural Design for Add Pet](#_heading=h.26in1rg) 9

[3.1.4 Key Activities](#_heading=h.lnxbz9) 10

[3.1.5 Software Interface to other components](#_heading=h.35nkun2) 10

[3.2 Remove Pet Feature](#_heading=h.z5fppedy15wl) 10

[3.2.1 Remove Pet Use Cases](#_heading=h.44sinio) 10

[3.2.2 Processing sequence for Remove Pet](#_heading=h.g46qq1ruj2yz) 12

[3.2.3 Structural Design for Remove Pet](#_heading=h.z337ya) 13

[3.2.4 Key Activities](#_heading=h.3j2qqm3) 13

[3.2.5 Software Interface to other components](#_heading=h.1y810tw) 13

[3.3 Edit Pet Feature](#_heading=h.4i7ojhp) 14

[3.3.1 Edit Pet Use Cases](#_heading=h.2xcytpi) 14

[3.3.2 Processing sequence for Edit Pet](#_heading=h.1ci93xb) 15

[3.3.3 Structural Design for Edit Pet](#_heading=h.3whwml4) 16

[3.3.4 Key Activities](#_heading=h.2bn6wsx) 17

[3.3.5 Software Interface to other components](#_heading=h.qsh70q) 17

[3.4 Add Food Feature](#_heading=h.19f78ahj7mq6) 17

[3.4.1 Add Food Use Cases](#_heading=h.3dgmg87x72gx) 17

[3.4.2 Processing sequence for Add Food](#_heading=h.ypa7ire8ab1k) 19

[3.4.3 Structural Design for Add Food](#_heading=h.z1p18i2fif91) 20

[3.4.4 Key Activities](#_heading=h.iypb2vt2hvbn) 20

[3.4.5 Software Interface to other components](#_heading=h.88rhyrnjd53) 21

[4 User interface design](#_heading=h.3as4poj) **21**

[4.1 Interface design rules](#_heading=h.1pxezwc) 21

[4.1.1 Bart Colors](#_heading=h.mlacmlhhosay) 21

[4.1.2 Color Scheme Rules](#_heading=h.5v15ccsicxhd) 21

[4.2 Description of the user interface](#_heading=h.49x2ik5) 21

[4.2.1 Home Screen](#_heading=h.2p2csry) 21

[4.2.1.1 Screen Images](#_heading=h.147n2zr) 22

[4.2.1.2 Objects and Actions](#_heading=h.ihv636) 22

[4.2.2 Food Bank Screen](#_heading=h.32hioqz) 22

[4.2.2.1 Screen Images](#_heading=h.vrhq8r75fjql) 23

[4.2.2.2 Objects and Actions](#_heading=h.e70fe69udtcm) 23

[4.2.3 Individual Pet Screen](#_heading=h.2grqrue) 23

[4.2.3.1 Screen Images](#_heading=h.umpacjjqyjf5) 24

[4.2.3.2 Objects and Actions](#_heading=h.emn3acaj42g9) 24

[4.2.4 Calendar View Screen](#_heading=h.vx1227) 24

[4.2.4.1 Screen Images](#_heading=h.jzvcvhv2c6gx) 25

[4.2.4.2 Objects and Actions](#_heading=h.32h7gmv3hlnw) 25

[4.2.5 Add Pet Screen](#_heading=h.3fwokq0) 25

[4.2.5.1 Screen Images](#_heading=h.uazwj71qaay8) 25

[4.2.5.2 Objects and Actions](#_heading=h.j5eccjkjj9w4) 25

[4.2.6 Edit Pet Screen](#_heading=h.1v1yuxt) 25

[4.2.6.1 Screen Images](#_heading=h.kofu8k5461nn) 26

[4.2.6.2 Objects and Actions](#_heading=h.ijihkbwevx5a) 26

[5 Restrictions, limitations, and constraints](#_heading=h.4f1mdlm) **26**

[6 Testing Issues](#_heading=h.2u6wntf) **26**

[6.1 Types of tests](#_heading=h.19c6y18) 26

[6.2 List of Test Cases](#_heading=h.28h4qwu) 26

[7 Appendices](#_heading=h.nmf14n) **27**

[7.1 Packaging and installation issues](#_heading=h.37m2jsg) 27

[7.2 User Manual](#_heading=h.46r0co2) 28

[7.3 Open Issues](#_heading=h.2lwamvv) 28

[7.4 Lessons Learned](#_heading=h.111kx3o) 28

[7.4.1 Design Patterns](#_heading=h.3l18frh) 28

[7.4.3 Team Communications](#_heading=h.206ipza) 28

[7.4.4 Task Allocations](#_heading=h.4k668n3) 28

[7.4.5 Desirable Changes](#_heading=h.2zbgiuw) 28

[7.4.6 Challenges Faced](#_heading=h.1egqt2p) 28

# 1 Introduction

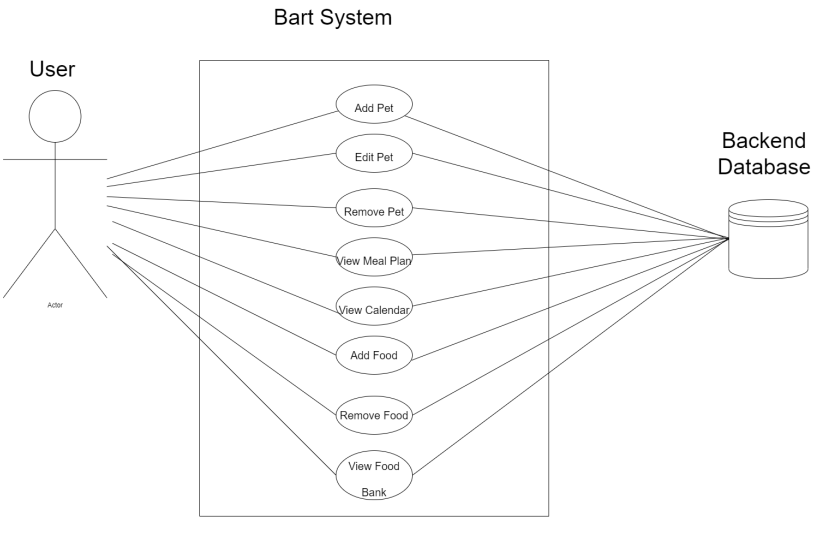
The Bartagamen Mobile Application features a diet plan algorithm which allows users to create a diverse meal plan for their bearded dragons. This document describes all data, architectural, interface and component-level design for the software.

## 1.1 Goals and objectives

The goal of the Bartagamen Mobile Application is to create a centralized place where the user can keep track of the complex meal plans of one or more bearded dragon lizards.

## 1.2 Statement of system scope

The Bartagamen Mobile Application is intended to be used by people who currently own a pet bearded dragon and shall be available for android systems. The system shall generate a diverse meal plan which can be viewed across multiple pets including a complete view for all pets in one place.

**

## 1.3 Definitions and Acronyms

### 1.3.1 Bart

Abbreviation for the Bartagamen Mobile Application

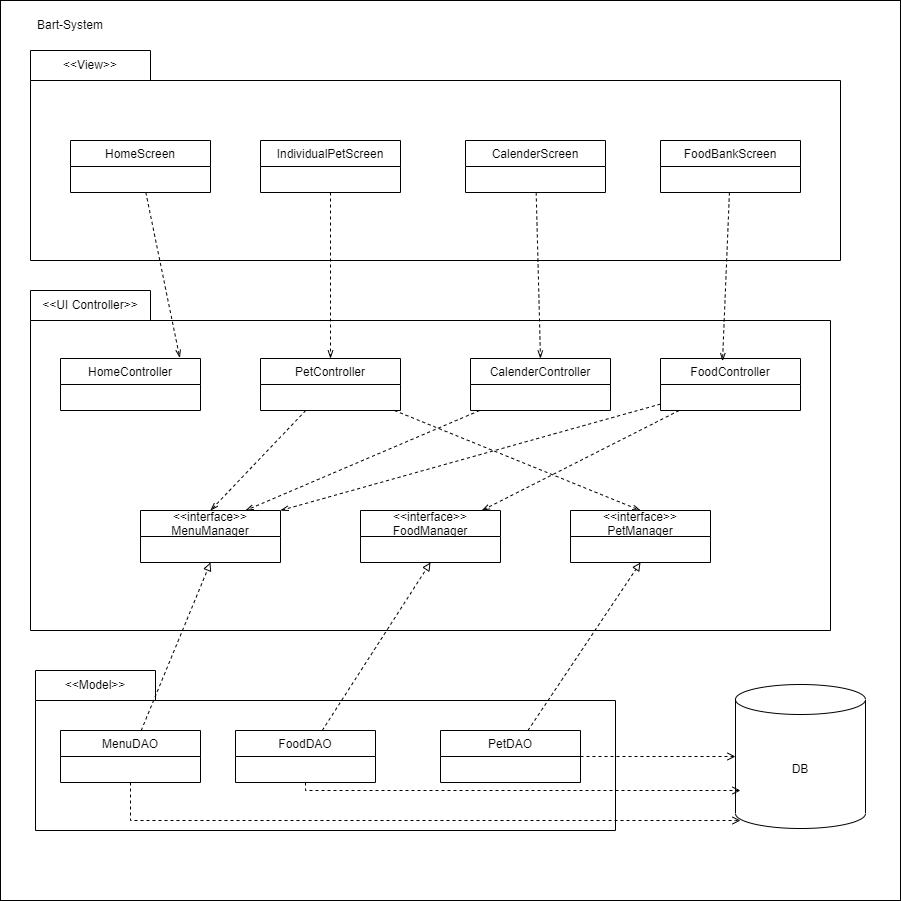
### 1.3.2 UI

User Interface

# 2 Architectural design

## 2.1 System Architecture

The Bart System is made up of the View subsystem, the UI Controller subsystem, the Model subsystem, and the database. The View subsystem displays information about the application to the user, and sends user requests to the UI Controller subsystem. The UI Controller subsystem processes the requests from the View subsystem and manages them when it can. If the UI Controller subsystem doesn’t contain the information it needs to handle a request locally, it sends the request to the Model subsystem. The Model subsystem accepts requests from the UI Controller subsystem, and builds SQL Queries to the Database for the required information to process the request. The Model subsystem then responds to the UI Controller level requests with the appropriate information.



## 2.2 Design Rational

A Model View Controller Layered architecture was the ideal choice for our software application. Layers provide services to the layers above them, and the View subsystem sits at the top of all the layers in the system. It provides no services, and only makes requests to the other subsystems. The Database subsystem sits at the bottom-most level, because it makes no requests to other subsystems, and only provides services to the Model subsystem.

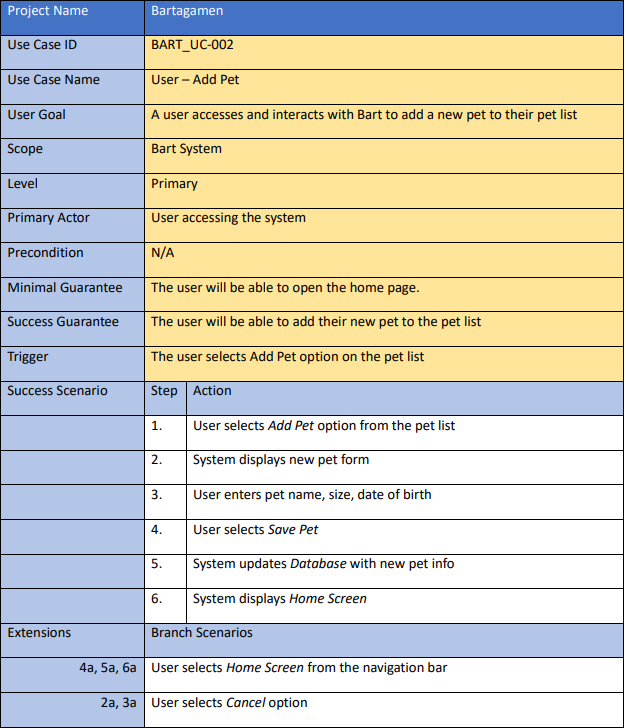
A client-server architecture was not chosen because the information supplied to the user does not need to be centrally located on the internet or a remote database, users all have uniform permissions, and an internet connection is not required. A repository architecture is also suboptimal because multiple modules need to communicate with each other, and forcing that communication through a central repository would impact performance, which is a priority on a mobile application.

# 3 Key Functionality design

## 3.1 Add Pet Feature

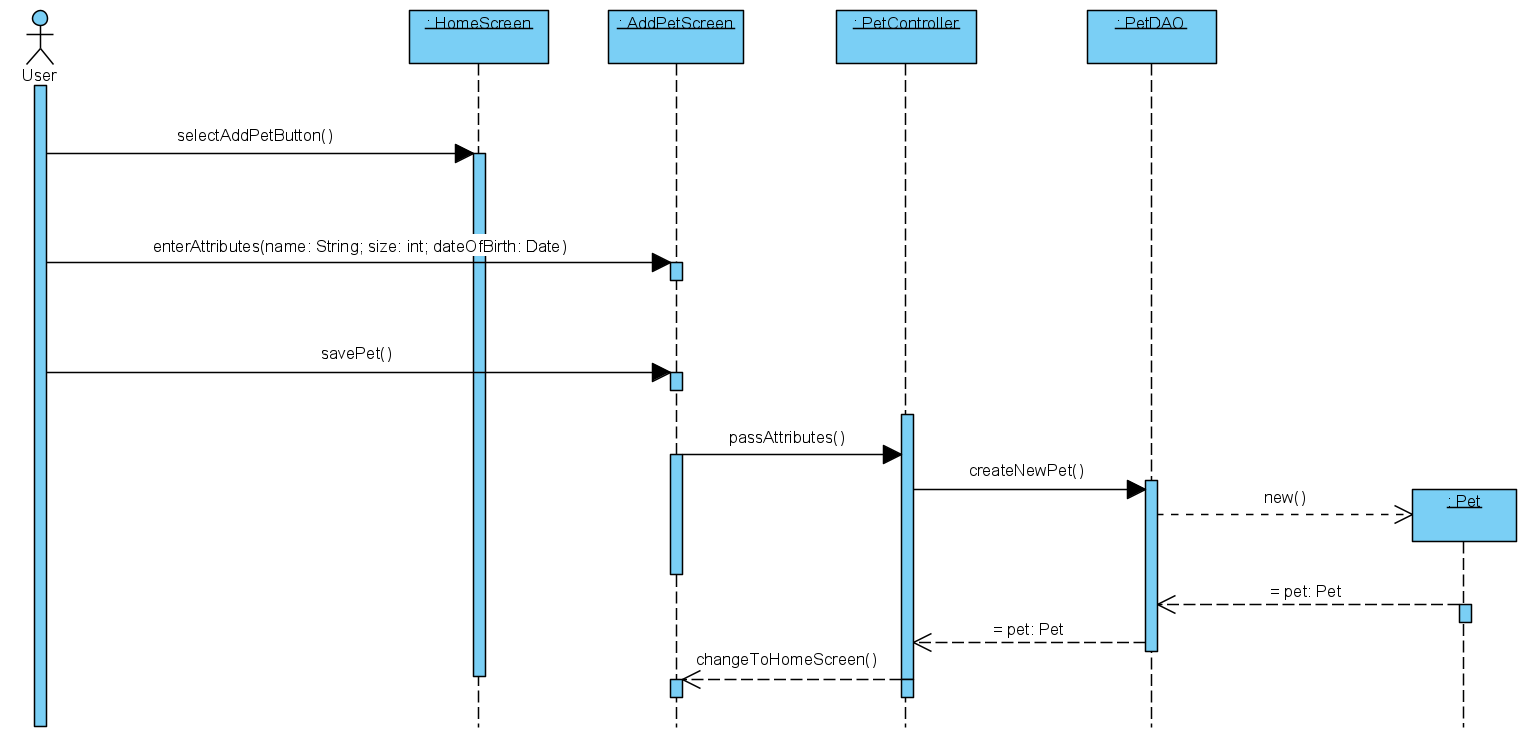
### 3.1.1 Add Pet Use Cases

From the home screen, the user selects the ***Pet List Dropdown Menu*** and from there selects the ***Add Pet*** option. The system then displays a blank pet form and the user enters relevant information based on their specific pet. The user then selects the ***Save*** option and the information is saved to the ***Database***. The user may also hit the ***Cancel*** button which will not save a new pet to the ***Database***. In either case, the system displays the home screen.



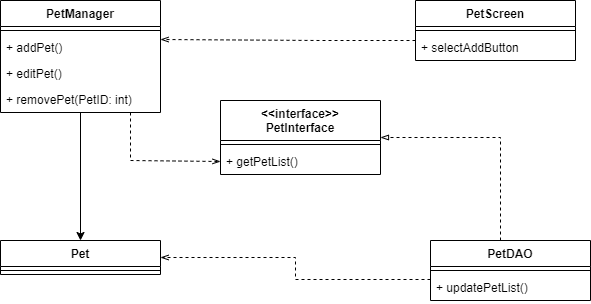
### 

### 3.1.2 Processing Sequence for Add Pet



***Add Pet*** will display a page with a form allowing the user to enter information about the new pet they are adding. They must enter the pet *Name*, *Size*, and *Date of Birth*. The pet object will be sent from the ***AddPetScreen***, through the ***PetController***, through the ***PetDAO***, to the ***Pets*** table in the ***Database***. The user is then able to select the newly added pet from the ***Pet List Dropdown Menu***.

### 3.1.3 Structural Design for Add Pet



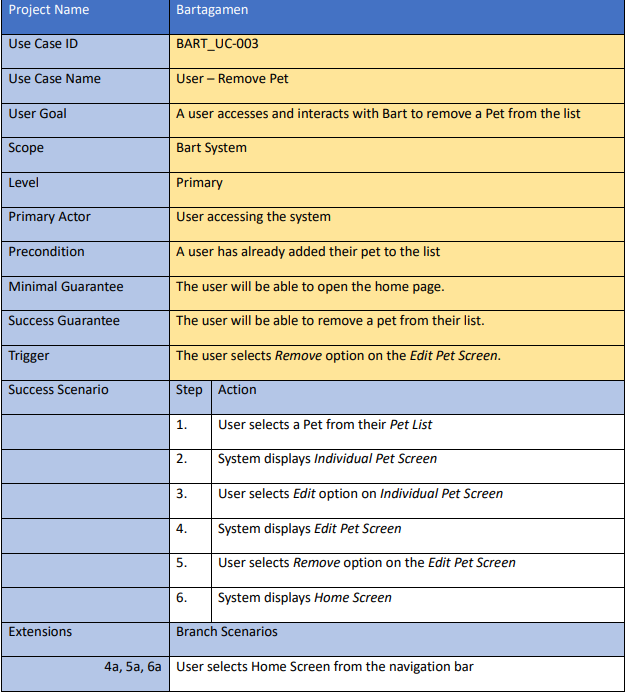
### 3.1.4 Key Activities

### 3.1.5 Software Interface to other components

## 3.2 Remove Pet Feature

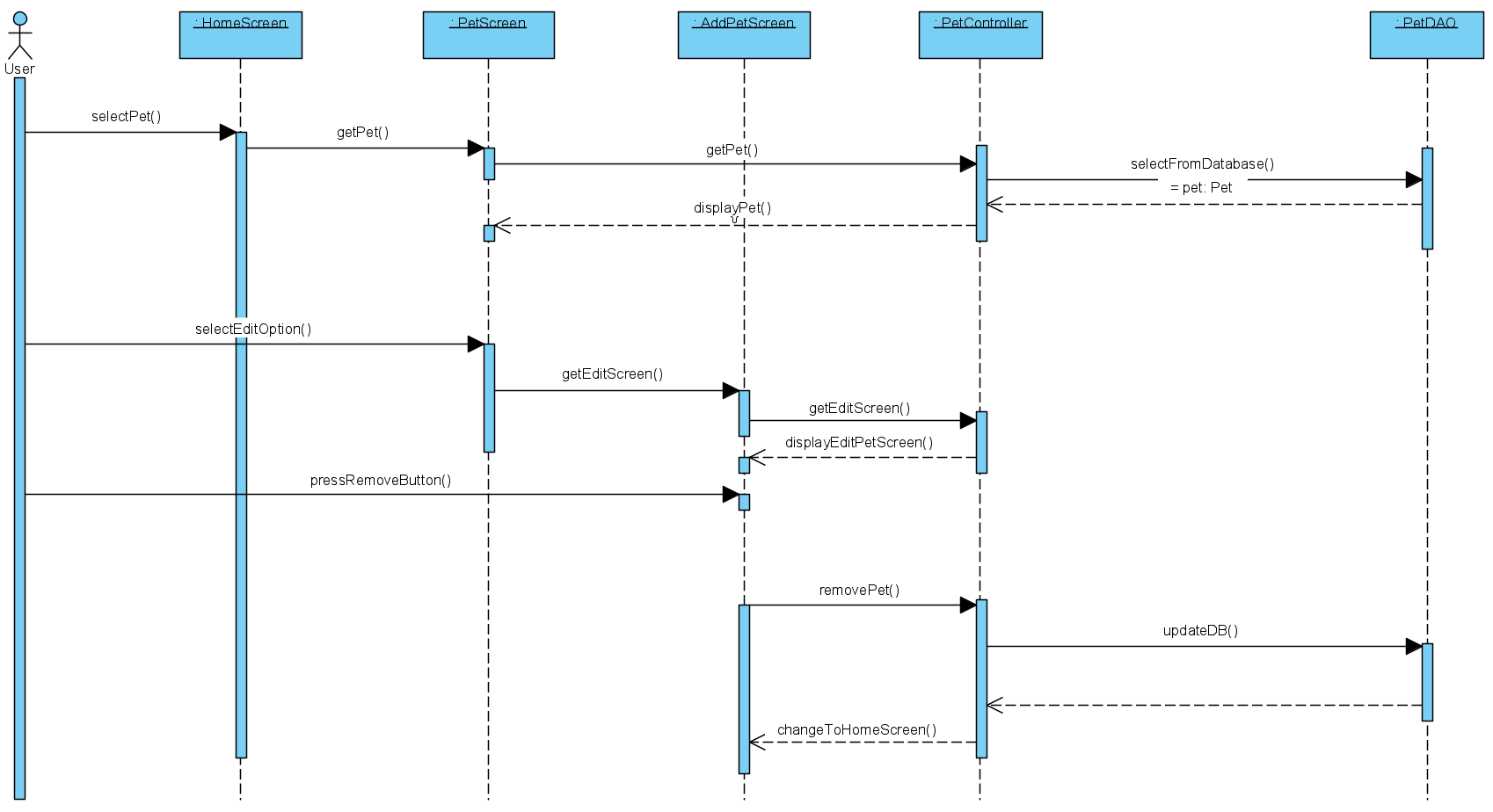
### 3.2.1 Remove Pet Use Cases

From the ***Individual Pet Screen***, the user selects the ***Edit*** option. The system displays the ***Edit Pet Form*** for the selected pet in the ***Individual Pet Screen***. The user selects the ***Remove*** option in the form. The system removes all that pet’s information from the ***Database***. The system displays the ***Home Screen***.



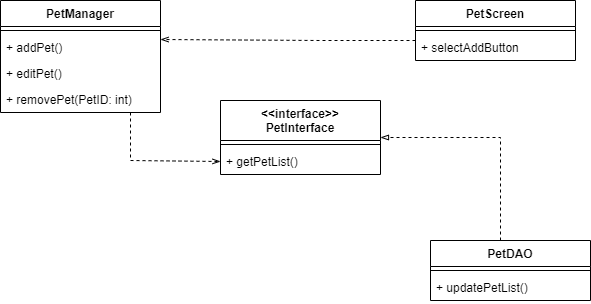
### 

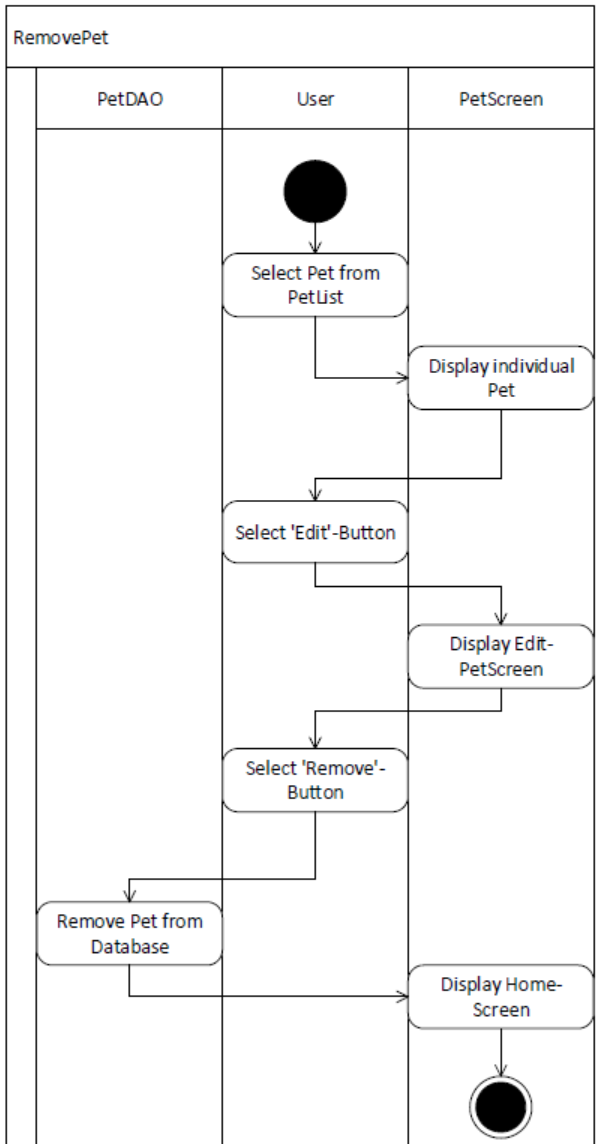
### 3.2.2 Processing sequence for Remove Pet



The ***User*** can Select a Pet through the ***HomeScreen***, which will be pulled by the ***PetController*** and the ***PetDAO*** from the ***Database***. Then it will be shown via the ***PetScreen***. Now the ***User*** can hit the edit-option button. A Remove-button appears on the AddPetScreen and the user hits that button. Afterwards the ***DB*** will be updated through the ***PetController*** and the ***PetDAO***. In the end the Home-Screen-Display will be shown.

### 3.2.3 Structural Design for Remove Pet





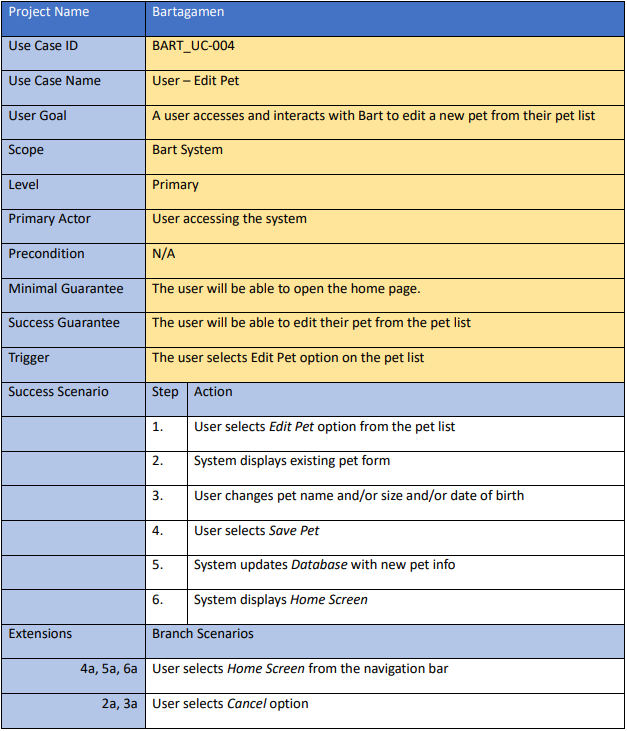
### 3.2.4 Key Activities

### 3.2.5 Software Interface to other components

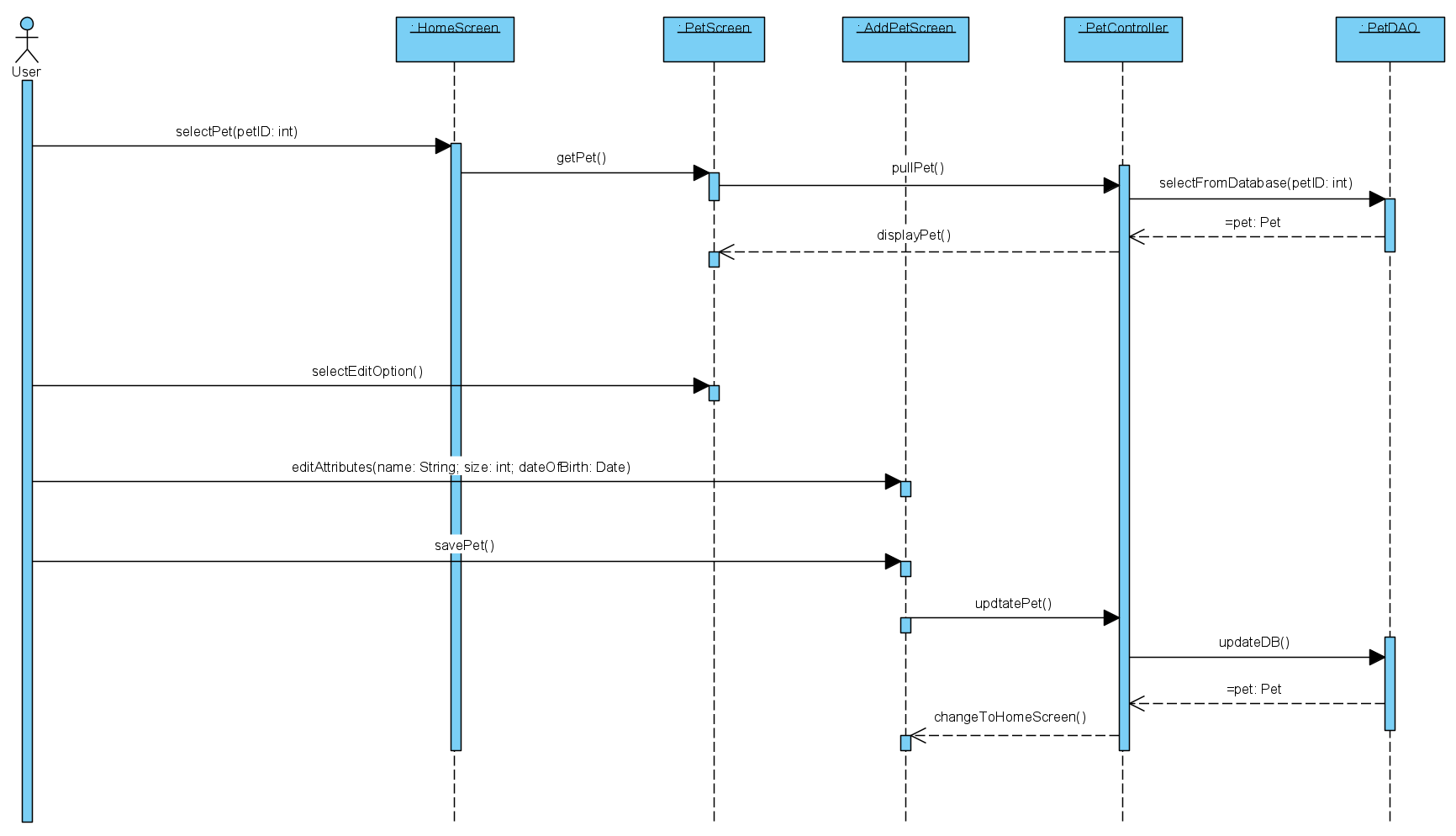
## 3.3 Edit Pet Feature

### 3.3.1 Edit Pet Use Cases

From the ***Individual Pet Screen***, the user selects the ***Edit*** button, and an ***Edit Pet Form*** appears for the selected pet. The user is able to edit features about the pet such as name, size, or date of birth. The user selects ***Save*** and the information for that pet is saved to the ***Database***. If the user selects ***Cancel*** instead of ***Save***, the information changed will not be saved to the database. In either case, the system will display the ***Home Screen***.

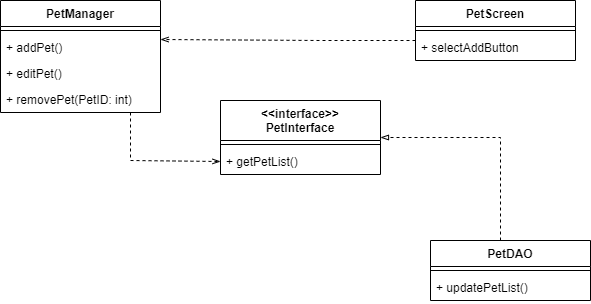


### 3.3.2 Processing sequence for Edit Pet



The ***User*** chooses a screen from the ***HomeScreen***. After that the System gets the pet through the ***PetController*** and the ***PetDAO***. Now the PetForm will be displayed on the ***PetScreen***. Next, the User selects the Edit-Option and enters the information about the Pet in the ***AddPetScreen***. After he/she does that, the User selects the save Button and through the ***PetController*** and the ***PetDAO*** the DB will be updated and the System shows the HomeScreen.

### 3.3.3 Structural Design for Edit Pet



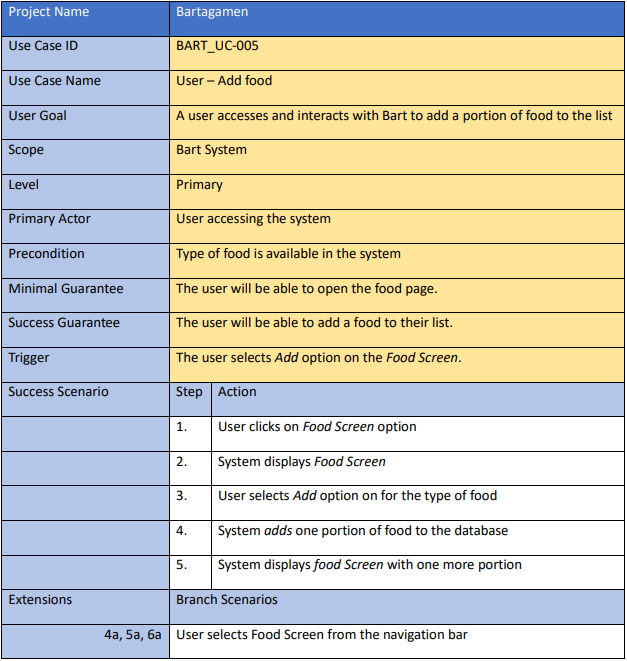
### 3.3.4 Key Activities

### 3.3.5 Software Interface to other components

## 3.4 Add Food Feature

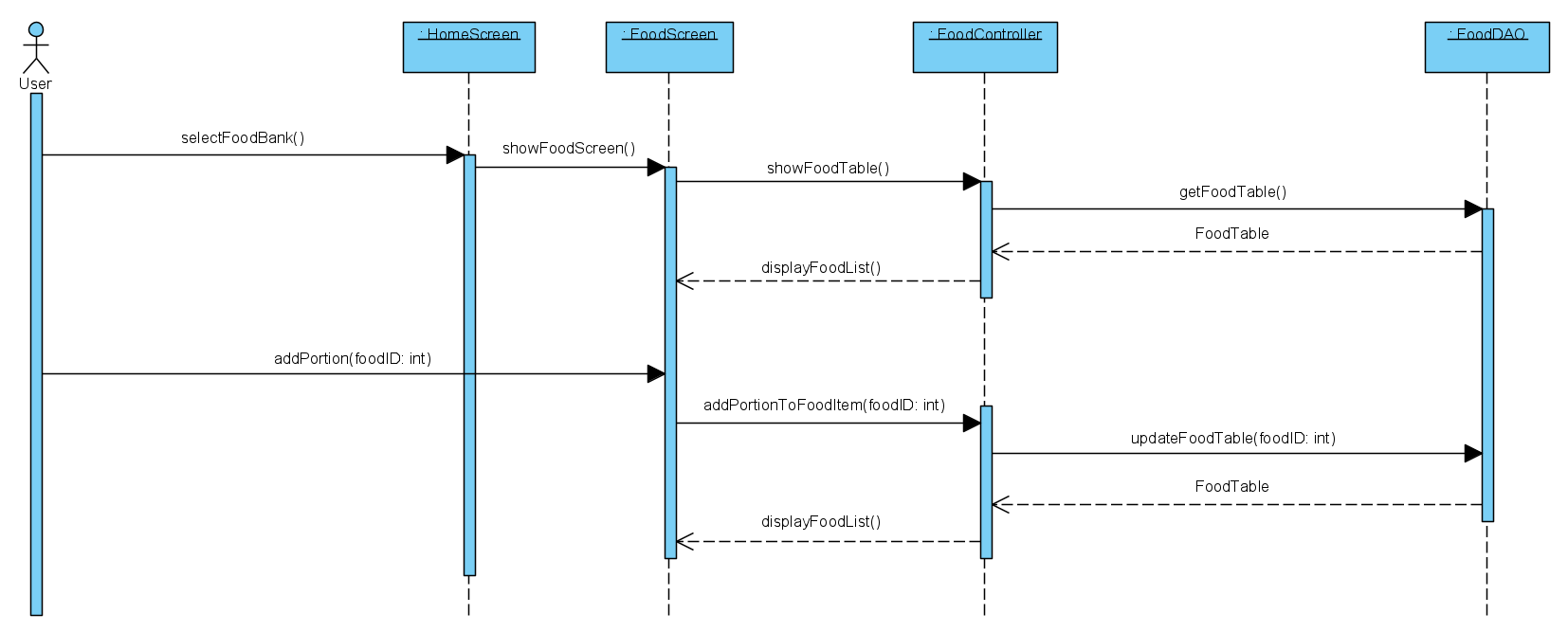
### 3.4.1 Add Food Use Cases

From the ***Home Screen***, the user selects the ***Food Bank*** button. The system displays the ***Food Bank Screen***; a list of food options, and radio buttons to signal that they are either in stock or out of stock. The user searches through the list of foods, and ***Selects*** a food they currently have in supply. The user then exits the ***Food Bank Screen.*** The system updates the ***Database*** with the new food added by the user in the ***Food Bank Screen***.



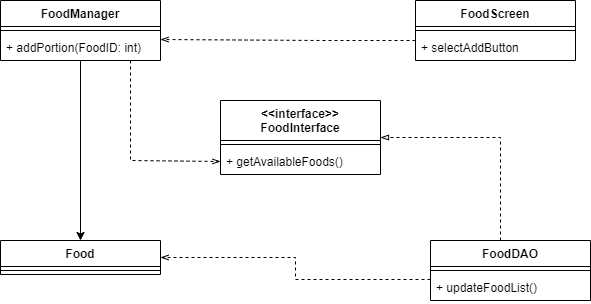
### 

### 3.4.2 Processing sequence for Add Food



***Add Food*** will display the FoodList through the ***FoodController*** and the ***FoodDAO*** to the User by pulling the available Food from the ***Database***. The user can now add a Portion to a chosen FoodItem via the ***FoodController***. Then the FoodTable will be updated through the ***FoodDAO***. In the end the FoodList will be shown via the ***FoodScreen*** to the ***User***.

### 3.4.3 Structural Design for Add Food



### 

### 3.4.4 Key Activities

### 3.4.5 Software Interface to other components

# 4 User interface design

## 4.1 Interface design rules

### 4.1.1 Bart Colors

“Background Blue” - #F1F4FD

“Button Inactive Blue” - #D5DEFF

“Button Active Blue” - #2E5BFF

“Text Background Grey” - #E4E8F1

“Black” - #000000

“White” - #FFFFFF

### 4.1.2 Color Scheme Rules

Overall application background shall be Background Blue.

Actionable Buttons shall be colored Button Inactive Blue, to signify they may be interacted with, but are not currently selected. Text Color of Actionable Buttons shall be Button Active Blue, to contrast with Button color.

Actioned Buttons shall be colored Button Active Blue, to signify they are currently selected. Text Color of Actioned Buttons shall be White, to contrast with Button color.

Text Display Boxes shall be colored Text Background Grey, to separate them both from the Application Background and other Buttons. Grey signifies Text Display Boxes are separated, but not actionable like Buttons. Text Color of Text Boxes shall be Black, to contrast with the Text Display Box color.

## 4.2 Description of the user interface

The user interface (UI) is a mobile application and provides a visual front-end to our client's databases and queries.

### 4.2.1 Home Screen

The home screen allows the user to access several important screens such as the add pet screen and the food bank screen while also allowing the user to select which pet the individual pet screen will display and displaying daily food requirements.

#### 4.2.1.1 Screen Images

#### 

**Figure 1 - Home Screen**

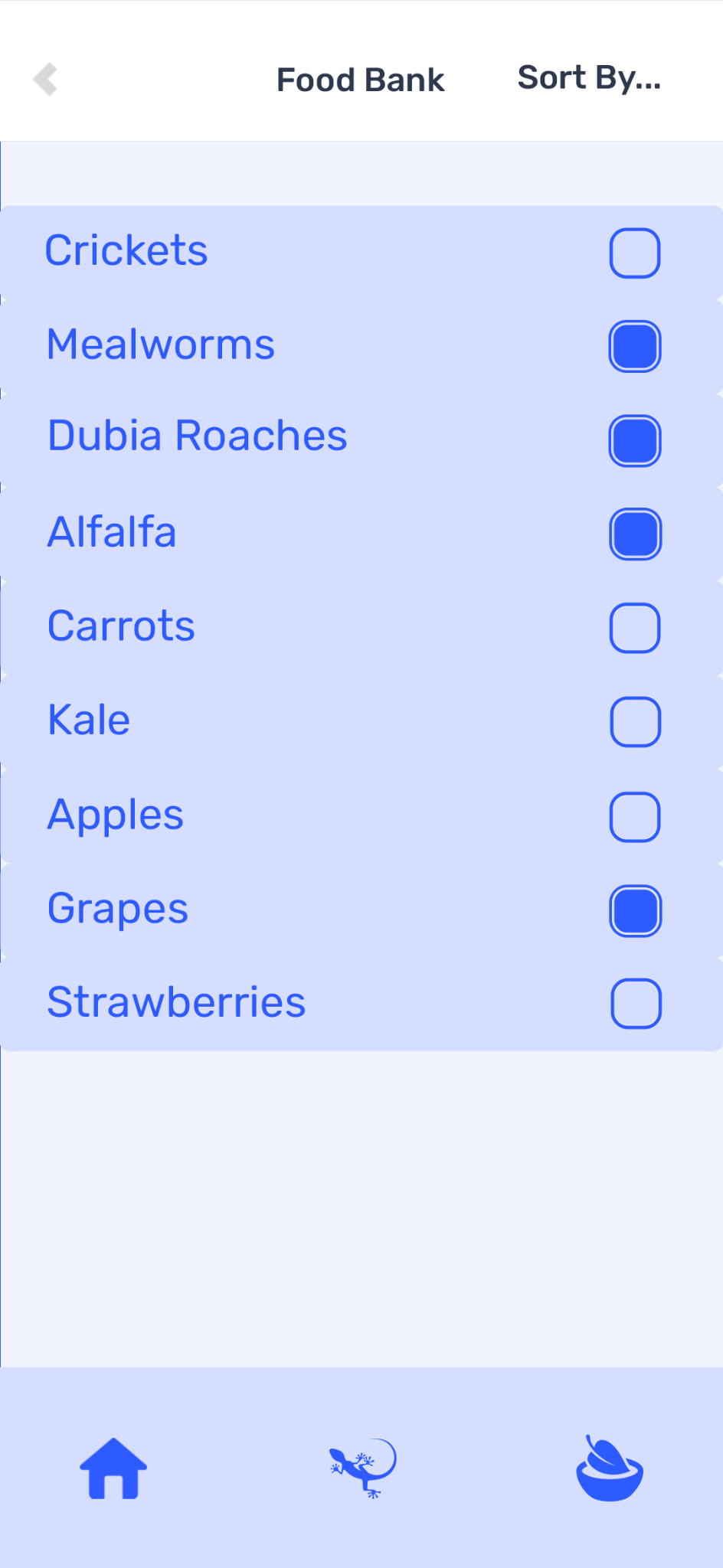
#### 4.2.1.2 Objects and Actions

The goal of the home screen is to provide a centralized point where the user will be able to access several important screens and options, as well as acting as a point where the user will be directed to after some actions on other screens.

### 4.2.2 Food Bank Screen

The food bank screen displays a list of food options that the user can select based on their current food supply.

#### 4.2.2.1 Screen Images

****

**Figure 2 - Food Bank Screen**

#### 4.2.2.2 Objects and Actions

The goal of the food screen is to provide a place where the user can visually see all the options available for the diet of a bearded dragon as well as what they currently have available in their own stock.

### 4.2.3 Individual Pet Screen

The individual pet screen will display a weekly meal plan to the user. The user may select each day in the week to see that respective plan as well as navigate to the monthly view from this screen.

#### 4.2.3.1 Screen Images

#### 

**Figure 3 - Individual Pet Screen**

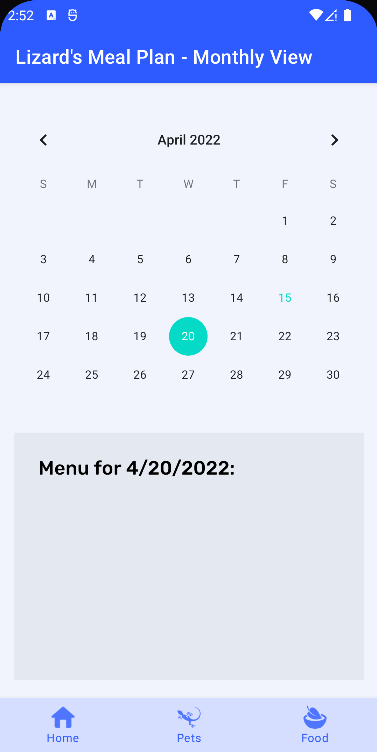
#### 4.2.3.2 Objects and Actions

The main goal of the individual pet screen is to provide an easy way for the user to look at an individual meal plan for one pet as well as provide a gateway to view the monthly view.

### 4.2.4 Calendar View Screen

The calendar view screen will provide an expanded view of a pets meal plan thirty days in the future.

#### 4.2.4.1 Screen Images



**Figure 4 - Calendar View Screen**

#### 4.2.4.2 Objects and Actions

The goal of the calendar view screen is to provide a long term view of a lizard's meal plan so the user can plan for or change their existing food options.

### 4.2.5 Add Pet Screen

The add pet screen allows the user to add a new pet into the system with all the relevant information.

#### 4.2.5.1 Screen Images

#### 4.2.5.2 Objects and Actions

The goal of the add pet screen is to provide a simple way for the user to add a new pet to the system with all the required information for a bearded dragon lizard.

### 4.2.6 Edit Pet Screen

The edit pet screen allows the user to change the information that was added to that pet in the add pet screen as well as allow the user to delete an existing pet.

#### 4.2.6.1 Screen Images

#### 4.2.6.2 Objects and Actions

The goal of the edit pet screen is to provide an easy way for the user to edit information about a specific pet as well as having the ability to delete that pet if necessary.

# 5 Restrictions, limitations, and constraints

* Tool must operate in an Android Mobile environment.
* Minimum Android API Level 26 (Android 8.0 / 89.7% of Android Users as of Apr 2022)
* The application will store information in an SQL Database.

# 6 Testing Issues

Test strategy and preliminary test case specification are presented in this section. Additional test cases are located in the Appendix section in spreadsheet format.

## 6.1 Types of tests

You may consider the following types of tests:

1. **Performance Test** – for example, to ensure that the response time for information retrieval is within an acceptable range. You typically should provide a specific performance bounds. For example, the search process should not take longer than 30 seconds.
2. **Accuracy Test** – for example, to determine if queries return the expected results.
3. **User Interface Test** – for example, to make sure the user interface is clear and easy to use with all types of users. Unfamiliar user can use the interface with minimal instruction and achieve the desired results.
4. **Security test** – for example, to ensure that users can only perform the tasks specified for their user group
5. **Repeatability Test** – for example, the software returns the same result for repeated queries.

## 6.2 List of Test Cases

You should document each test case in the following format:

| Test Type | **Accuracy Test** |
| --- | --- |
| Testing range | User login feature |
| Testing Input | User ID and Wrong password |
| Testing procedure | Enter user ID and wrong password  Click Login |
| Expected Test Result | Prompt “User is not valid” |
| Testers | Tom Bridger |
| Test result | Passed |

Note: you may have more than one test cases for each type of tests. If the tester and test results information is not available, you may add it in the final submission.

# 7 Appendices

## 7.1 Packaging and installation issues

How to install and prepare the system to run

For example, many teams use a database for your system. Some teams use FireBase, some use Google services, others use customized servers (located in dorm). Whatever techniques you are using, you need to describe how you set up the DB and how to create DB connections (showing some code snippet would help).

This part will be used by the instructor to evaluate your self-learning ability. When you describe a technique (even it is a small tool like Postman), provide sufficient information such that it can be used as a tutorial for a novice to quick get it up and running.

## 7.2 User Manual

Give step-by-step description of using the key features of the system.

## 7.3 Open Issues

Features considered but not finished

## 7.4 Lessons Learned

* Difficult if one person makes progress and the other members need to catch up
* Leaving comments helps your team members catch up to what you’re doing
* Leveraging member experience to lean on their strengths
* Project management with tools like Monday.com to distribute tasks
* Communicating progress with team members on individual tasks
* Sprints are helpful to see what we’re currently working on, what still needs to be done, organized by what issues need to be addressed first
* Class diagramming and modeling was helpful for making sure everybody is on the same page about what a module or class does, and any one of us can implement it

### 7.4.1 Design Patterns

We used MVC-Design Pattern, because it makes sense to split our classes in different layers. You can see the class diagram in section 2.1. Furthermore we used a Singleton class for our Data Access Object, to only have one instance of the Database. The Database is a local Database. Another singleton class we used for our navigation bar, which is visible on every Screen of our app.

### 7.4.3 Team Communications

For very superficial communication we used a Smartphone Messenger. There we scheduled meetings for assignments or the app itself. When it came to meetings we used either discord or zoom, where we were able to share our screen to the other team members.

### 7.4.4 Task Allocations

We used the programm ‘Mondays.com’, which is a tool for working with an agile project management approach. First we filled the Product Backlog with all the work packages we figured out and then we divided our timeline in 6 Sprints. When we met and distributed the tasks, we assigned the work packages to the team members within those sprints.

### 7.4.5 Desirable Changes

*Assume that you have another month to work on the project, what aspects of the system would you like to improve? What are the additional features you want to add to the system? [Each student should use a separate paragraph to respond to the questions]*

**Janik**: In my opinion the main functionality is very easy and convenient to use. So if I would like to change something, because we have more time, I would like to work on the FoodListTable, because to see the Proteins (bugs) the user has to scroll down to see them. So there is probably another opportunity to show the Items, by first choosing a radioButton and then only showing the items of the Food-Category. But it's kind of hard to make it suitable for the database. That was the reason we did it how it is, because of lack of time.

**Race**: Given more time to work on the project, I’d like to make it available for Apple smartphones on iOS. The application has been written for Android, and I believe a lot of our existing work could be carried over directly to the iOS platform with minimal work. This would give us the largest increase in the amount of users of our application for the amount of work put in. I would also like to spend more time unit testing our classes, but due to the amount of inheritance and utility of other classes, they proved pretty complicated to test without major reworking.

**Bryce**: If we had more time to work on the project, I think I would want to work on some of the features that we had planned to implement but had to remove due to the time constraints. One of these features was the ability for the user to add a custom image of their pet to go with its profile which would have been nice. A minor thing that I would like to change would be how the drop down menu on the home page doesn’t keep the name of the selected pet shown after you select that pet, a minor detail that would communicate with the user which pet they have selected already from that screen.

### 7.4.6 Challenges Faced

*Among requirements specification, system design, and system implementation, which one you think is the hardest task? Why? [Each student should use a separate paragraph to respond to the questions]*

**Janik**: In my opinion, the hardest task is the system design and implementation, because it has to be consistent and we changed it a few times. But in the end everything is consistent. So if I have to choose one out of them, I would say the implementation is harder, because I still learn a lot of things I hadn't known before. So some tasks are a bit tricky and take a long time.

**Race**: I have prior experience with requirement specification and system design, so I would say the hardest part is implementation. It was the most time consuming part of the project, our first commit to the actual code of the application was February 23, and we all worked on the project up until the submission deadline of May 18. It was entertaining to work together as a team to delegate tasks and create sprint backlogs together.

**Bryce**: The hardest task for me would have to be implementation because going into this project I had no experience with java and android studio, the two things we used to create this project. I had to learn a lot and it took a while for some of the more complex tasks. Even if I came into this with a little more knowledge, I think the implementation would still be the hardest part.