

# Integrated Framework for Dairy Feeding

## **Authors**

Dwarakh Nayam

Kevin Kapros

Mike Silvasy

Oliver Stein

Evan Webb

Ger Vue

Multimedia, Hypertext, and Information Access Report

Instructor: Dr. Edward A. Fox

Clients:

Mark Hanigan, Dairy Science Professor

Wu Feng, CS Professor



May 14, 2021

Blacksburg, Virginia 24061

# Contents

- List of Figures** **vi**
  
- List of Tables** **viii**
  
- Executive Summary** **1**
  
- 1 Project Description** **2**
  
- 2 Requirements** **3**
  - 2.1 Website . . . . . 3
  - 2.2 Data Acquisition . . . . . 4
  - 2.3 Data Reports . . . . . 4
  - 2.4 Anomaly Detection . . . . . 4
  - 2.5 What Tools Did We Use? . . . . . 5
  
- 3 User Manual** **6**
  - 3.1 Overview . . . . . 6
  - 3.2 Purpose . . . . . 6
  - 3.3 Farmers . . . . . 7
    - 3.3.1 User Sign Up Process . . . . . 7

3.3.2	Farms	10
3.3.3	Edit/Update Ingredients	12
3.3.4	Edit/Update Mixes	13
3.3.5	Edit/Update Diets	16
3.3.6	Analysis	17
3.4	Data Management Team	18
3.5	Admin(s)	18
<b>4</b>	<b>Developers Manual</b>	<b>20</b>
4.1	Overview	20
4.2	Getting Started	20
4.3	Front End	21
4.3.1	Design	21
4.3.2	Ingredients	25
4.3.3	Mixes	26
4.3.4	Diets	27
4.3.5	File Uploads	28
4.3.6	R Analysis	28
4.4	Back End	28
4.4.1	Django Framework	28

4.4.2	Data Parsing . . . . .	34
4.4.3	Alerts . . . . .	35
4.5	Docker . . . . .	38
4.5.1	Dockerfile . . . . .	38
4.5.2	Apache Configuration . . . . .	40
<b>5</b>	<b>Methodology</b>	<b>42</b>
5.1	Outline . . . . .	42
5.2	Farmer Goals . . . . .	42
5.2.1	Farm Management Tasks . . . . .	43
5.2.2	Ingredient Retrieval Tasks . . . . .	44
5.2.3	Create Mixtures Tasks . . . . .	45
5.2.4	Create Diets Tasks . . . . .	47
5.2.5	File Upload and Data Injection Tasks . . . . .	47
5.2.6	Retrieve Dairy Production Data . . . . .	48
5.3	Data Management User Goals . . . . .	48
5.3.1	Receive Notifications Upon Dietary Changes Tasks . . . . .	49
5.3.2	Monitor Farmers Production Data . . . . .	50
<b>6</b>	<b>Testing</b>	<b>51</b>
6.1	Farm . . . . .	51

6.2	Ingredients . . . . .	51
6.3	Mixes . . . . .	52
6.4	Diets . . . . .	53
6.5	Storage . . . . .	53
6.6	Data Uploading . . . . .	53
6.7	Staff Notifications and Commit . . . . .	54
6.8	R with Python . . . . .	55
6.9	Troubleshooting . . . . .	55
<b>7</b>	<b>Lessons Learned</b>	<b>57</b>
7.1	Timeline/Schedule . . . . .	57
<b>8</b>	<b>Future Work</b>	<b>60</b>
<b>9</b>	<b>Acknowledgement</b>	<b>62</b>
<b>10</b>	<b>References</b>	<b>63</b>

# List of Figures

3.1	Register for an account . . . . .	7
3.2	Registration example . . . . .	8
3.3	Activation page example . . . . .	9
3.4	Activation email . . . . .	9
3.5	Add farm information . . . . .	11
3.6	Updated farm . . . . .	12
3.7	Ingredient was copied and moved to the local table . . . . .	13
3.8	Adding an ingredient to a mix . . . . .	14
3.9	Create a mix composed of ingredients . . . . .	15
3.10	Created mixes in the mixes page . . . . .	15
3.11	Adding a mixture to a diet . . . . .	16
3.12	Create a diet composed of mixtures . . . . .	17
3.13	An image displaying admin user permissions . . . . .	19
4.1	First design of a farmer's local ingredients . . . . .	22
4.2	First design of the default ingredients that a farmer will choose from . . . . .	23
4.3	Final wire-frame design for viewing user and default ingredients. . . . .	24
4.4	Uploading Files HTML Format. . . . .	28

4.5	An image of the weather model . . . . .	34
4.6	An image of the weather model . . . . .	34
4.7	An image of some of the user profiles . . . . .	35
4.8	An image of an ingredient edit notification email . . . . .	35
4.9	Staffs link to commit user edits . . . . .	36
4.10	Staffs view to see a user edits on an ingredient . . . . .	36
4.11	Staffs view to see possibly affected mixes on an ingredient edit . . . . .	37
4.12	Staffs view to see possibly affected storage ingredients on an ingredient edit . . . . .	37
4.13	Dockerfile for the dairy_feeding_framework Docker image . . . . .	38
4.14	api.conf file used by Docker image to configure Apache Server . . . . .	41
5.1	Farm registration and management work flow . . . . .	44
5.2	Ingredient retrieval task work flow . . . . .	45
5.3	Create mixture task work flow . . . . .	46
5.4	Creating the diets task work flow . . . . .	47
5.5	Retrieve dairy production data work flow . . . . .	48
5.6	Notifications when editing an ingredient work flow . . . . .	49
5.7	Monitoring data in production work flow . . . . .	50
7.1	An image of our timeline . . . . .	57

# List of Tables

4.1	Files related to the front-end . . . . .	25
4.2	Files related to the mixes . . . . .	26
4.3	Files related to the diets . . . . .	27



# Executive Summary

Cattle farmers today have technology such as total mixed rations trackers that gather feeding data with respect to nutrients and total amount of feed consumed by pens of cattle [11]. An online platform that could help farmers manage nutritional diets for an individual cow and/or pens of cattle based on statistical feedback from milk production would serve as a useful complimentary tool to existing total mixed rations tracking technologies. A potential online platform such as the one described could prevent the use of excess nutrients during feeding, mitigate costs, as well as optimize dairy production and herd health. These aforementioned potential benefits are exactly what the Integrated Dairy Feeding Framework aims to provide. The web platform created by our team provides farmers with a simplistic and practical interface that allows them to construct new diets for their cattle based on the statistical feedback of their previous milk composition data and health reports with the intention of generating healthier cattle capable of producing higher quantities of milk as well as a higher quality milk product for the consumer.

# Chapter 1

## Project Description

The goal of this project is to produce a web framework that provides data management and analysis tools for dairy farmers, as well as information and manipulation of data regarding their herds' diets, ingredients, feeding details, and other preliminary and tertiary data aspects such as weather data and particular statistical analyses. The specific target audience/user base is the subset of farmers who are bovine dairy-oriented farmers who currently utilize a sister technology called Select Sires[10] for “developing and delivering innovative genetic solutions”.

Data that is produced from Select Sires by the farmers is used as input to our tool, and is stored in our databases. Additionally, farmers will be able to input data manually regarding their herds' feeding/diet specifics including ingredients, percentages, and mixtures. These are the basic blocks of data that users will interact with and that our website will provide to the interface. Analysis using R will produce statistics designed by dairy scientists working with the development team to provide useful anomaly detection statistics as well as statistical visualizations and analysis of dairy production.

# Chapter 2

## Requirements

The main technological deliverables that our clients required included the following: finishing or redesigning the website, automating data acquisition from multiple sources, creating data-driven reports for the farmer, finishing anomaly detection code, and creating an anomaly event report. Outside of the technical specifications of our clients' requirements, we are required to lead weekly meetings with our clients where we will show our current progress in completing the main deliverables.

### 2.1 Website

Since our clients decided to step away from their initial decision to develop the website with PHP, our team was tasked to develop the website from scratch using Python and the Django Web Framework. The clients required that our website should be implemented with ease of use since farmers are the target users of the website. With this in mind, we were tasked with creating a login and registration system, a page within the website for the farmer to edit or add ingredients from a master ingredients SQL table, a page for farmers to create mixtures from their local ingredients, and a page for farmers to create diets from their mixtures. This will give the farmer the ability to control diet composition and optimization from the website.

## 2.2 Data Acquisition

Our clients originally requested that our web framework server would automatically pull Select Sires[10] data, feeding data, local weather data by farm zip code, TMR Tracker data, and DHIA Milk composition data. However, later our clients decided that the Select Sires data will be collected via template file upload by the user and then stored within the MySQL database. Data collection for weather data is remotely pulled via API calls and should take place every week. Similar to the Select Sires data, the TMR Tracker data will be manually uploaded via template files. Finally, the DHIA Milk composition data will be inserted by the user into the web framework for analysis.

## 2.3 Data Reports

Once our web framework has all of the appropriate data, our clients specified that the framework needs to give the farmer a report that will summarize the milk production by herd, pen, or farmer defined groups of cows. These data reports should be formatted appropriately for readability and should be easily accessed by the farmer on the website.

## 2.4 Anomaly Detection

The web framework is required to support anomaly detection. An anomaly detection report will summarize multiple anomalies that include events like farmer reported health events, environmental stress events, reproductive activity events, nutrition delivery events, and undefined events. We can use the analysis to predict when a dairy animal becomes sick or when a cow is producing less milk. We can then use this data and the history of the dairy

animals to figure out what caused this. We can then suggest a change in certain variables like food and eating habits to help the dairy animal.

## 2.5 What Tools Did We Use?

Following is a list of the tools that our team used to fulfill our clients needs.

- Python (version 3.7.9) and the Django Web Framework (version 3.1.7)
- Bootstrap 4.0 with HTML 5
- MySQL which is supported by rlogin.cs.vt.edu
- rlogin.cs.vt.edu to host the website
- R package (rpy2 3.4.3) so we can use R in Django Framework.

# Chapter 3

## User Manual

### 3.1 Overview

This web app is designed for farmers to be able to upload, track, and make inferences on their data of both their cattle's performance and output, and their diet ingredients and mixtures they give to the cattle. In this user manual you will find how to sign up and start an account with the application, how to utilize the diet mixtures/ingredients, how to upload your own data, how to edit that data, and what kinds of analysis you can find with the web app's visualization measures.

### 3.2 Purpose

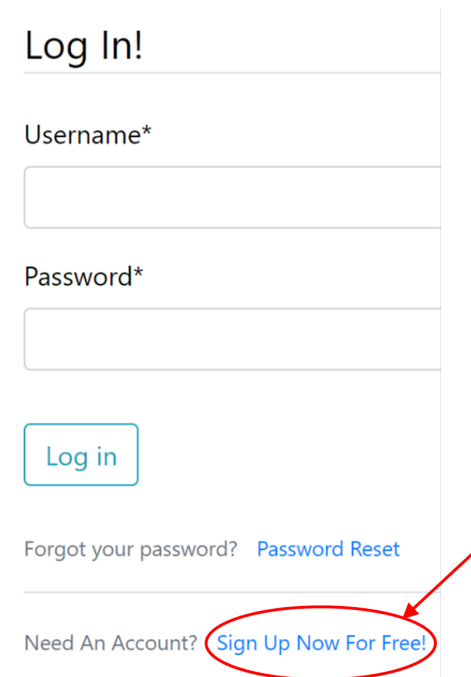
The purpose of this user manual is to inform future users of how to use the Dairy Feeding Framework so that they can employ the appropriate functionalities listed in the Overview (see above).

## 3.3 Farmers

### 3.3.1 User Sign Up Process

#### Registration

If you are new to the website, the first thing you will need to do is create an account. To register, click “Sign Up Now For Free!”



Log In!

Username\*

Password\*

Log in

Forgot your password? [Password Reset](#)

Need An Account? [Sign Up Now For Free!](#)

Figure 3.1: Register for an account

After clicking the “Sign Up Now For Free!” link shown in Figure 3.1, you will be redirected to our registration page shown in Figure 3.2. There you will need to create a unique username, providing an email that does not already have an account and a password. The password cannot be similar to your personal information, must be at least 8 characters long, cannot be a commonly used password (e.g., “Password”) and cannot be completely numeric. You will also have to confirm the password, for verification. If there are any errors, you will receive

an error message.

Join Today! It's completely free!

---

Username\*

Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Email\*

Password\*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation\*

Enter the same password as before, for verification.

[Sign Up](#)

---

Already Have An Account? [Sign In](#)

Figure 3.2: Registration example

After inputting all the required information, click the “Sign Up” button. If you receive any error messages, you will need to rectify the errors. If not, your account has been created and you will now need to verify your email address. Figure 3.3 shows a successful attempt at registering an account.



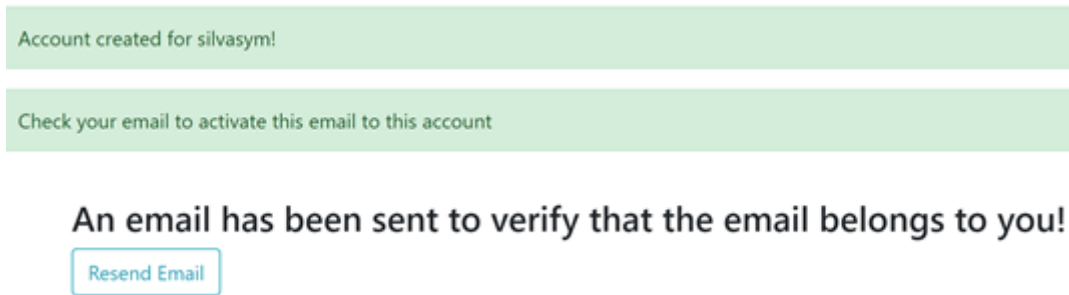


Figure 3.3: Activation page example

## Email Verification

After successfully registering for an account, you will need to activate it before being able to access the website. You should have received an email, but if you would like the email to be resent, you can click the “Resend Email” button. To activate your account, you will need to check your email for the activation link. The email a user can expect to see is shown in Figure 3.4.

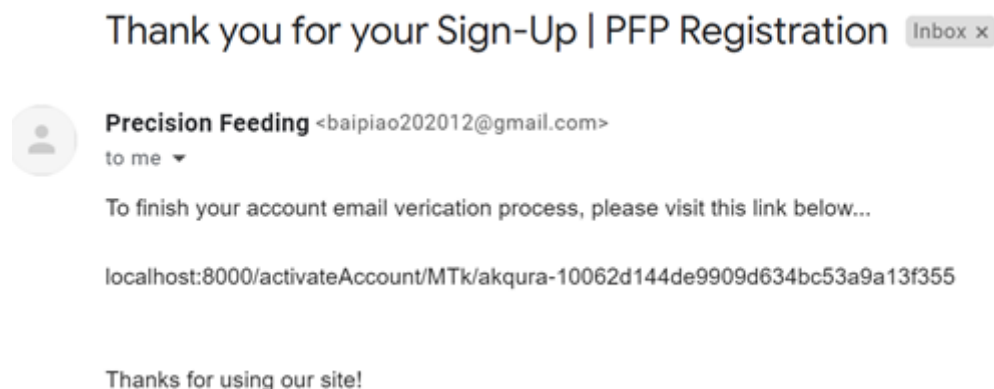


Figure 3.4: Activation email

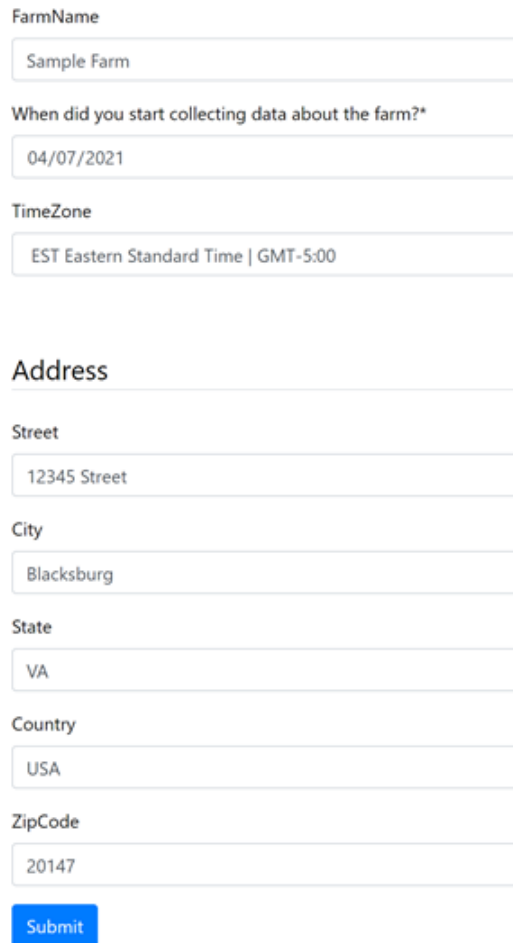
After accessing the link, your account will be activated and you will be redirected to the login page where you can log in and finally access the website.

## Login

Now that you have created and activated your account, you can log in using your username and password. If either of these two are incorrect, error messages will be displayed and you will need to fix the errors before logging in. If the username and password are both correct, log in by pressing the “Log in” button. After logging in, you will be redirected to our home page.

### 3.3.2 Farms

After logging into the website, you will be taken directly to the PFP Dashboard. Here, you will see your farm(s). Each farm has a name, the date when data collection began on the farm, the time zone and address of the farm. To add a farm, simply click the “Add a Farm” button on the dashboard. You will be redirected to our add farm page shown in Figure 3.5, where you must enter the farm name, date when data collection began, time zone and address of the farm. After entering the required information, click the “Submit” button and the farm data will be saved on the back end and displayed on the dashboard.



The form is titled "Add farm information" and contains several input fields. The "FarmName" field contains "Sample Farm". The "When did you start collecting data about the farm?\*" field contains "04/07/2021". The "TimeZone" field contains "EST Eastern Standard Time | GMT-5:00". Below these fields is a section titled "Address" with five input fields: "Street" (12345 Street), "City" (Blacksburg), "State" (VA), "Country" (USA), and "ZipCode" (20147). A blue "Submit" button is located at the bottom of the form.

FarmName

Sample Farm

When did you start collecting data about the farm?\*

04/07/2021

TimeZone

EST Eastern Standard Time | GMT-5:00

Address

Street

12345 Street

City

Blacksburg

State

VA

Country

USA

ZipCode

20147

Submit

Figure 3.5: Add farm information

To update an existing farm, click the “Update Farm” button next to the farm you wish to update. After clicking the button, you will be redirected to our update farm page where you can modify the name of the farm only. After modifying the name, click the “Update” button and the modifications will be saved on the back end and displayed on the dashboard. Figure 3.6 shows the result of an update on “Sample Farm” after its name was changed to “Sample Farm - Update.”

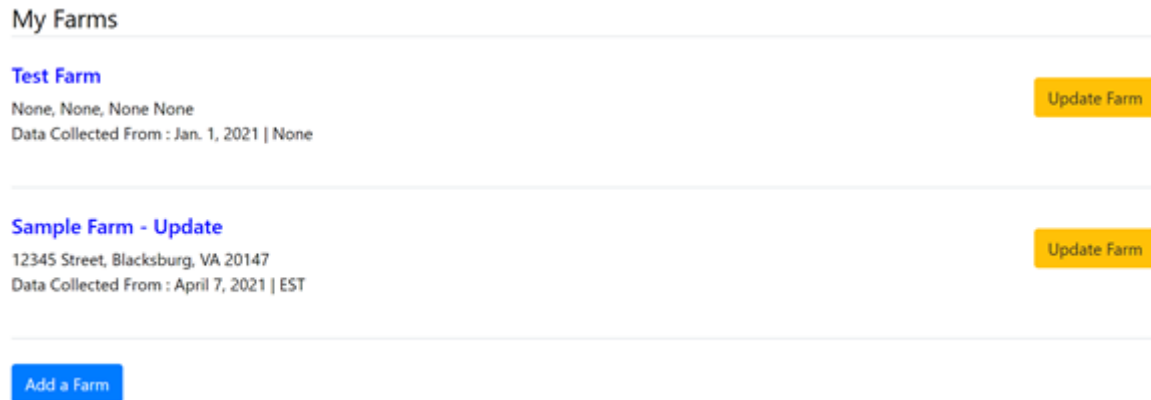


Figure 3.6: Updated farm

### 3.3.3 Edit/Update Ingredients

To edit or update an ingredient, you will need to head over to the ingredients page located under the relevant pages tab. After navigating to this page, you will see two tables, one called “My Ingredients” and one called “Default Ingredients.” The “Default Ingredients” table contains 283 ingredients which you can copy and edit into your “My Ingredients” table to make mixes. To do this, click the “Copy & Edit” button next to the ingredient of your choice, make edits to the ingredient composition (if applicable) and rename the ingredient. You can see in Figure 3.7 that the alfalfa meal ingredient was copied from the “Default Ingredients Table,” renamed to “Alfalfa meal - Edited,” and moved into the “My Ingredients” table.

**My Ingredients**

Click on the Ingredient's name for more ingredient information

Show  entries Search:

Name	Category	Type	DE_base	NDF	ADF	CP	DM Matter	Edit	Make Copy & Update
<a href="#">Alfalfa meal - Edited</a>	Plant Protein	Forage	2.499	42.853	33.882	19.513	90.749	<a href="#">Edit</a>	<a href="#">Make Copy</a>

Showing 1 to 1 of 1 entries Previous **1** Next

**Default Ingredients**

Show  entries Search:

Name	Category	Type	DE_base	NDF	ADF	CP	DM Matter	Copy & Edit
<a href="#">Almond hulls</a>	By-Product/Other	Concentrate	2.434	33.005	27.639	5.25	88.201	<a href="#">Copy &amp; Edit</a>
<a href="#">Ammonium chloride</a>	Vitamin/Mineral	Concentrate	0.0	None	None	163.6	100.0	<a href="#">Copy &amp; Edit</a>
<a href="#">Ammonium phosphate (dibasic)</a>	Vitamin/Mineral	Concentrate	0.0	None	None	115.9	100.0	<a href="#">Copy &amp; Edit</a>
<a href="#">Ammonium phosphate (monobasic)</a>	Vitamin/Mineral	Concentrate	0.0	None	None	70.9	100.0	<a href="#">Copy &amp; Edit</a>
<a href="#">Ammonium sulfate</a>	Vitamin/Mineral	Concentrate	0.0	None	None	134.1	100.0	<a href="#">Copy &amp; Edit</a>

Figure 3.7: Ingredient was copied and moved to the local table

Once you have ingredients in your “My Ingredients” table, you can still edit and make copies of those ingredients. There will be an “Edit” button next to each ingredient so that you can still edit ingredients. You must edit ingredients within 35 days of the last edit/copy and the admin will be notified via email that you have made changes. This is important because any changes to an ingredient will change any mixes and diets that ingredient is in. There is also a “Copy” button next to each ingredient so that you can create a copy of an ingredient.

### 3.3.4 Edit/Update Mixes

To look at your mixes, you will need to head over to the mixes page located under the relevant pages tab. After navigating to this page, you can view mixes you’ve already created

or create a new mix. To create a new mix, click the “Make a New Mix” button on the top right of the page. After clicking this button, you will be redirected to the create a mix page. Here you will see two tables, your “My Ingredients” table from the ingredients tab shown in Figure 3.8, and your “Mixture Ingredients” table shown in Figure 3.9. If you would like to add an ingredient to a mix, click the “+” button next to that ingredient. You can add as many ingredients you would like to a mix.

**My Ingredients**

Show  entries Search:

Name	Location	Category	Type	DE_base	NDF	ADF	CP	DM Matter	Add
<a href="#">Alfalfa meal - Edited</a>	NRCLib	Plant Protein	Forage	2.499	42.853	33.882	19.513	90.749	<a href="#">+</a>
<a href="#">Alfalfa meal - Edited - Copied</a>	NRCLib	Plant Protein	Forage	2.499	42.853	33.882	19.513	90.749	<a href="#">+</a>
<a href="#">Ammonium sulfate - Edited</a>	NRCLib	Vitamin/Mineral	Concentrate	0.0	None	None	134.1	100.0	<a href="#">+</a>

Showing 1 to 3 of 3 entries Previous **1** Next

Figure 3.8: Adding an ingredient to a mix

After adding the ingredient to the mix, you will see it in the “Mixture Ingredients” table as shown in Figure 3.9. Here you will have to input the percentage in dry matter of the ingredient. The total percentage in dry matter of all the ingredients should have a sum of 100%. If the total percentage of dry matter doesn’t sum up to 100%, an error message will pop up and instruct you to fix the percentages so that they sum up to 100% (though there is a slight amount of rounding up allowed if it is not a perfect 100%). If you’d like to remove an ingredient from the mix, click the red “X” button next to that ingredient. After you have your desired mixture, create a name for the mixture and click the “Submit” button to create your mix.

## Mixture Ingredients

Input Mix Name:

Name	Location	Percentage in Dry Matter	Delete
Alfalfa meal - Edited	NRCLib	<input type="text" value="50"/> %	<input type="button" value="X"/>
Ammonium sulfate - Edited	NRCLib	<input type="text" value="50"/> %	<input type="button" value="X"/>

Total = 100

Figure 3.9: Create a mix composed of ingredients

After you create your mix, you should see it in the mixes page. Here, you can edit the mix (changing ingredients or the percentage in dry matter composition), or delete the mix altogether as shown in Figure 3.10.

## My Mixes

My Mix		
Mix ID	Mix Ingr	Ratio DM%
343	<a href="#">Alfalfa meal - Edited</a>	50
345	<a href="#">Ammonium sulfate - Edited</a>	50

Figure 3.10: Created mixes in the mixes page

### 3.3.5 Edit/Update Diets

To look at your diets, you will need to head over to the diets page located under the relevant pages tab. After navigating to this page, you will need to choose the farm where the diet is being used. After choosing the farm, you can view the diets you have already created, or create a new diet. To create a diet, click the “Make a New Diet” button on the top right of the screen. After clicking this button, you should be redirected to our make a diet page. Here you will see two tables, your “My Mixtures” table from the mixes tab shown in Figure 3.11 and your “Diet Mixtures” table shown in Figure 3.12. If you would like to add a mix to a diet, click the “+” button next to that mix. You can add as many mixes you would like to a diet.

**My Mixtures**

Show  entries Search:

Mixture	Location	Add to Diet
Greens	-	<input type="button" value="+"/>
MeatPie	-	<input type="button" value="+"/>
test	-	<input type="button" value="+"/>
test	-	<input type="button" value="+"/>
Test Mix	-	<input type="button" value="+"/>
Test Name 2	-	<input type="button" value="+"/>

Showing 1 to 6 of 6 entries Previous **1** Next

Figure 3.11: Adding a mixture to a diet

After adding the mix to the diet, you will see it in the “Diet Mixtures” table. Here you will



have to input pounds of each mixture in the diet. If you'd like to remove a mix from the diet, click the red "X" button next to that mix. You must also input the diet start date, and animal ID if applicable. After you have your desired diet, create a name for the diet and click the "Done" button to create your diet.

### Diet Mixtures

Input Diet Name:

Mixture	Lbs of Mixture	Delete
Greens	<input type="text" value="90"/> Lbs	<input type="button" value="X"/>
MeatPie	<input type="text" value="70"/> Lbs	<input type="button" value="X"/>

Total = 160

Diet Start Date:

Animal id:

Figure 3.12: Create a diet composed of mixtures

After you create your diet, you should see it in the diets page. Here you can edit the mix (changing mixes or the percentage of mixes) or delete the diet altogether.

### 3.3.6 Analysis

Analysis of the cattle data and the ingredients data will be utilized along with weather data in R. The calculations done on the data are unknown to the development team at the moment, but it can be speculated that the dairy scientists currently formulating the R data

analysis will be targeting statistics such as what diets are correlated with the best dairy output from the given farmer's cattle, what weather makes which herds perform differently, and what ingredients to use to respond to conditions in herd performance or weather changes or possibly other dependencies.

### **3.4 Data Management Team**

The data management team includes the users who handle data flow, data injection, and data modifications. These users have data coming in every day. Not all data needs to be monitored at all times, only at crucial times. This team will have access to run different scripts which will run processes to help with data flow and modifications. This team will also handle any edit commits that farmers may want to make. When farmers need to speak with someone to tackle data errors, this team can be at their service.

### **3.5 Admin(s)**

Admin users are privileged users that have permissions within the Dairy Feeding Framework beyond the permissions shared with regular users. Admin users can: add log entries, view log entries, change log entries, delete log entries, etc. (see Figure 3.13).

**Permissions**

**Active**  
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

---

**Staff status**  
Designates whether the user can log into this admin site.

---

**Superuser status**  
Designates that this user has all permissions without explicitly assigning them.

---

**Groups:**

Available groups ?

Q Filter

Choose all ?

Chosen groups ?

Remove all

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

---

**User permissions:**

Available user permissions ?

Q Filter

admin | log entry | Can add log entry

admin | log entry | Can change log entry

admin | log entry | Can delete log entry

admin | log entry | Can view log entry

auth | group | Can add group

auth | group | Can change group

auth | group | Can delete group

auth | group | Can view group

auth | permission | Can add permission

auth | permission | Can change permission

auth | permission | Can delete permission

auth | permission | Can view permission

auth | user | Can add user

Choose all ?

Chosen user permissions ?

Remove all

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Figure 3.13: An image displaying admin user permissions

Certain admins also have access to view/add/edit/delete data in the tables this website contains, based on what permissions are given to this admin user by the super user.

# Chapter 4

## Developers Manual

### 4.1 Overview

The developers manual will give anyone assigned to start development on the web framework, guidance as to what dependencies are needed to run the web framework locally, front end design choices and implementation, back end design choices and implementation, and methodology for using the application.

### 4.2 Getting Started

To get started on development for this project, the developer will need to make sure that they have the following languages and dependencies downloaded.

- Python version 3.7.9
- GitHub
- R programming language

After installing the programming languages and dependencies above, the new developer must clone the GitHub repository using the following URL.

<https://github.com/dwarakhvt/DairyFeedingFramework/>

The developer must make sure they pull from the master branch. Now the developer should have all files for the web framework. Next, the developer will need to install all of the web framework's dependencies by running the 'setup.sh -all' bash script that will set up the virtual environment and install the Django framework dependencies from the requirements.txt file. Another way to install all the dependencies is to run

```
Python PythonPackageInstaller.py requirements.txt
```

## 4.3 Front End

### 4.3.1 Design

Django has the capability to stitch and merge several HTML documents together and send them to the user as a single HTML file. This allows developers to put sections of the HTML code in separate files. One HTML document can be used for several different page requests. For example, if you wanted your website to look the same with different pages having only a few sections modified, you can have an HTML document act as a base template, where other HTML files will use the base template and fill in the missing sections in the body and/or head. Each section of the code, for a different purpose, is in a separate file. This helps organize the different components that go into creating the final page. Django has template merging capabilities which allow for easy inheritance of the contents of other HTML documents.

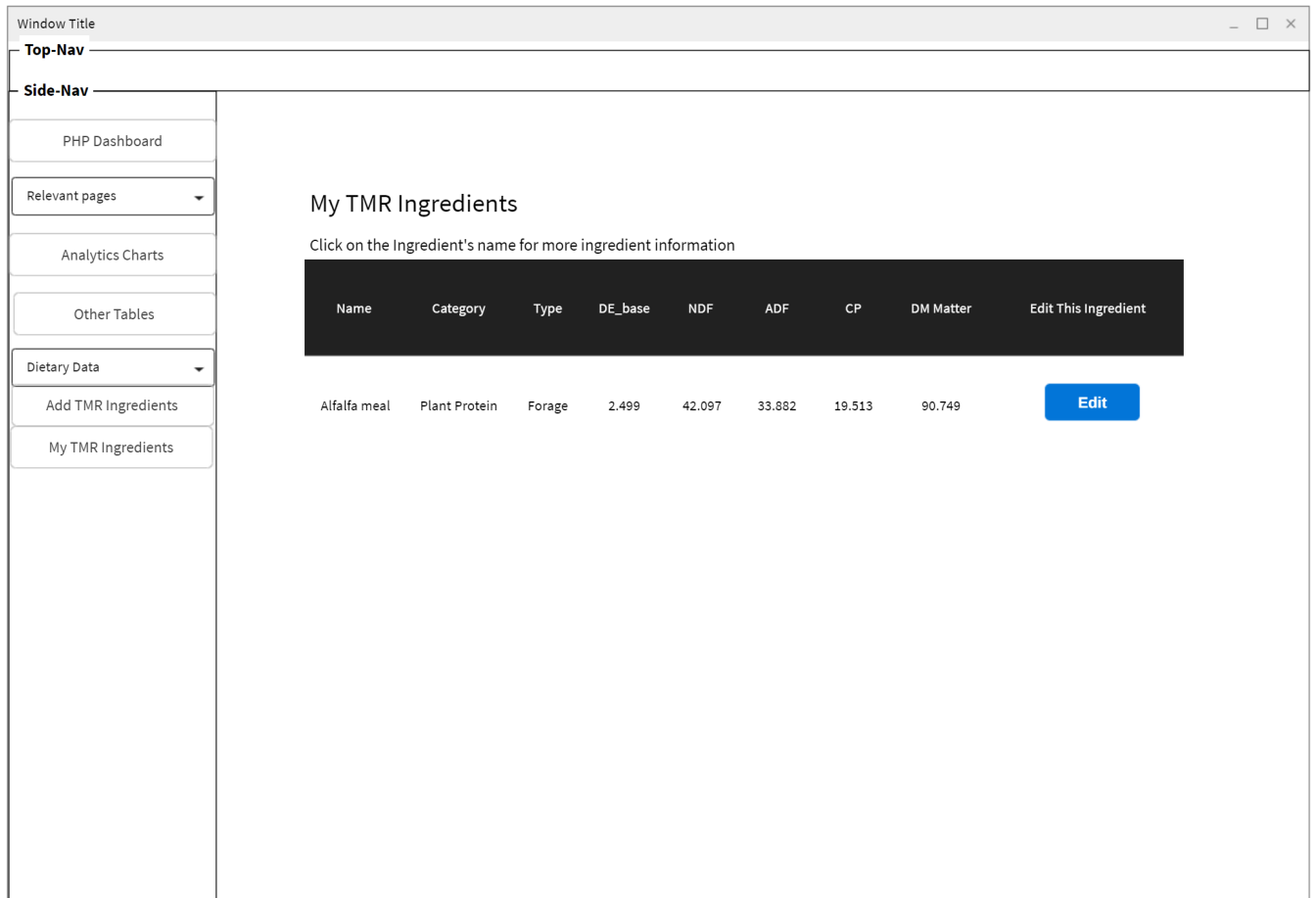


Figure 4.1: First design of a farmer's local ingredients

The team's first design for the farmer to manipulate ingredients was to have two separate pages where the farmer will have one page for adding ingredient, Figure 4.2, and another page for viewing and editing their local ingredients, Figure 4.1.

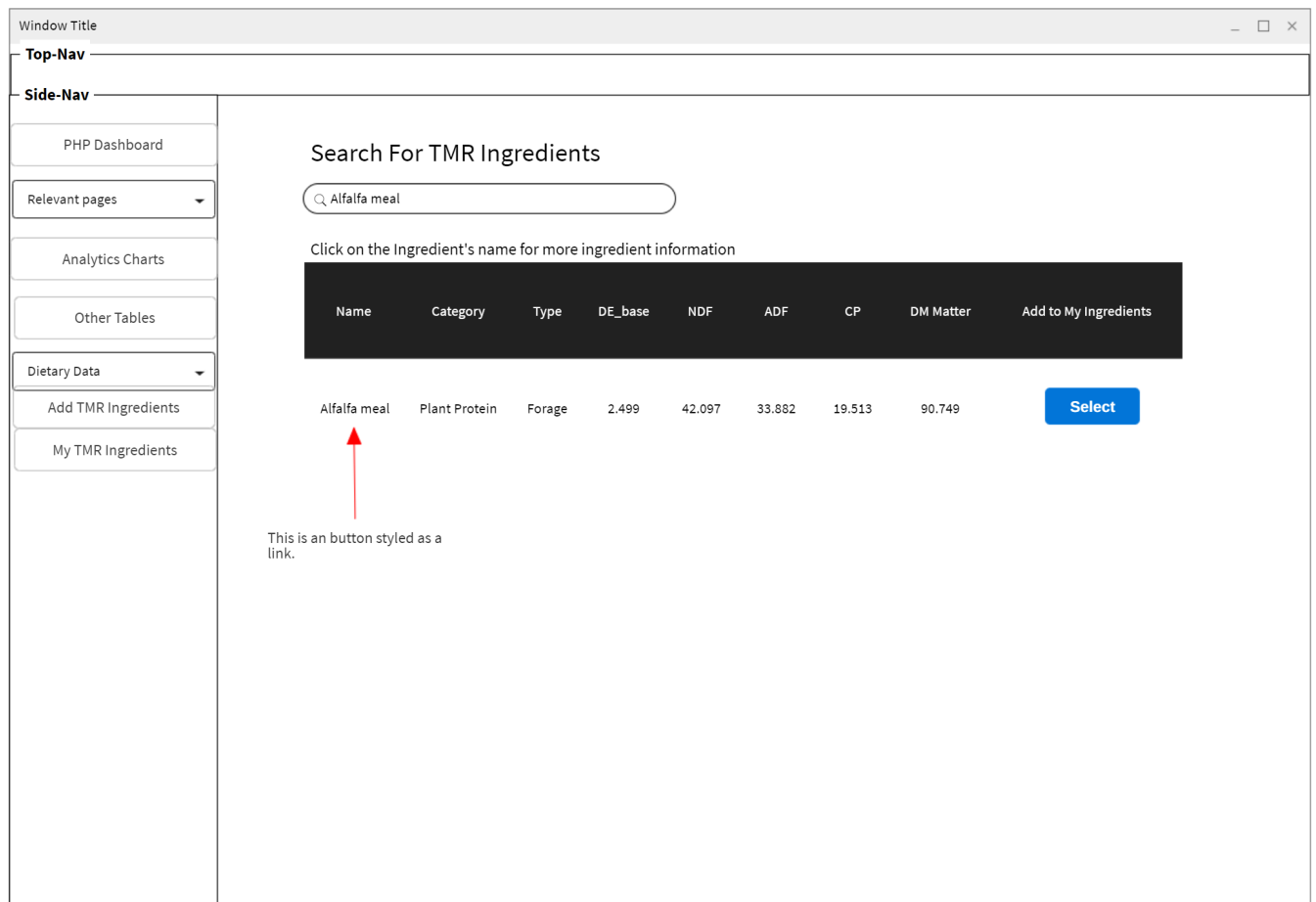


Figure 4.2: First design of the default ingredients that a farmer will choose from

After reviewing the simple solution with our clients, we decided to change the design so that both actions of adding ingredients from the default ingredients table to the farmer's local table and viewing those local ingredients would be in a single page, see Figure 4.3.

Window Title

**Top-Nav**

**Side-Nav**

PHP Dashboard

Relevant pages

Analytics Charts

Other Tables

Dietary Data

### My TMR Ingredients

Click on the Ingredient's name for more ingredient information

Show Ingredients

Name	Category	Type	DE_base	NDF	ADF	CP	DM Matter	Edit This Ingredient	
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>

### Default Ingredients

Click on the Ingredient's name for more ingredient information

Show Ingredients

Name	Category	Type	DE_base	NDF	ADF	CP	DM Matter	Add to My TMR Ingredients
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Add"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Add"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Add"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Add"/>
Alfalfa meal	Plant Protein	Forage	2.499	42.097	33.882	19.513	90.749	<input type="button" value="Add"/>

Figure 4.3: Final wire-frame design for viewing user and default ingredients.



### 4.3.2 Ingredients

Table 4.1 shows the file structure for the ingredients components in the web framework.

<b>Files</b>	<b>Description</b>	<b>File Path</b>
Templates	Contain all of the HTML documents	PFP_WEB/core/templates/core/ingredients
Models	Classes and tables used for ingredients	PFP_WEB/core/models.py
Views	Implements the functionality for ingredients pages	PFP_WEB/core/views.py
URLs	Contains the URL extensions for the ingredients pages	PFP_WEB/core/urls.py
Forms	Allow storing user input into an existing ingredients model	PFP_WEB/core/forms.py

Table 4.1: Files related to the front-end

### 4.3.3 Mixes

Table 4.2 shows the file structure for the mixes components in the web framework.

<b>Files</b>	<b>Description</b>	<b>File Path</b>
Templates	Contain all of the HTML documents	PFP_WEB/core/templates/core/Mixes
Models	Classes and tables used for mixes	PFP_WEB/core/models.py
Views	Implements the functionality for mixes pages	PFP_WEB/core/views.py
URLs	Contains the URL extensions for the mixes pages	PFP_WEB/core/urls.py
Forms	Allow storing user input into an existing mixes model	PFP_WEB/core/forms.py

Table 4.2: Files related to the mixes

### 4.3.4 Diets

Table 4.2 shows the file structure for the diets components in the web framework.

<b>Files</b>	<b>Description</b>	<b>File Path</b>
Templates	Contain all of the HTML documents	PFP_WEB/core/templates/core/Diets
Models	Classes and tables used for diets	PFP_WEB/core/models.py
Views	Implements the functionality for diets pages	PFP_WEB/core/views.py
URLs	Contains the URL extensions for the diets pages	PFP_WEB/core/urls.py
Forms	Allow storing user input into an existing diets model	PFP_WEB/core/forms.py

Table 4.3: Files related to the diets

### 4.3.5 File Uploads

Within the File Upload page, there is a box containing a button that users can click to upload files to the Dairy Feeding Framework. For a more explicit illustration of what this button looks like, please see the Figure 4.4.

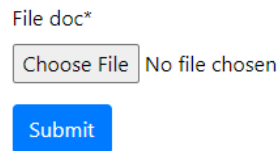


Figure 4.4: Uploading Files HTML Format.

### 4.3.6 R Analysis

We use an artificial R environment that runs with Python [8]. Our website back-end can pass file names and necessary variables to R scripts provided by the dairy scientist admins. This is done to produce statistical analysis output in an HTML format for the dairy science users. Our tool is not only a valuable tool for farmers, but an interactive and useful tool for dairy scientists.

The flow of operations would be that a user navigates to a data analytics page, and a user action/button click takes the user through installing their R script, runs it, and returns the results to their webpage in HTML [9].

## 4.4 Back End

### 4.4.1 Django Framework

#### Overview

This website needs to have high functionality and performance capabilities. Django is an organized Python based web framework platform with many security features and integra-

tions built straight into it. This framework allows us to build apps, templates with special Python/Django capabilities for filling in information, customized URLs, customized forms, and customized functions[1].

## Security

In the main project folder, there is a file called `settings.py`. This file contains all the security and Django application settings. For security, there is a secret security key which is NOT to be revealed to the public. This key is used for any transfers between the user and the server. There are several middleware security packages added by default.

```
django.middleware.security.SecurityMiddleware
django.contrib.sessions.middleware.SessionMiddleware
django.middleware.common.CommonMiddleware
django.middleware.csrf.CsrfViewMiddleware
django.contrib.auth.middleware.AuthenticationMiddleware
django.contrib.messages.middleware.MessageMiddleware
django.middleware.clickjacking.XFrameOptionsMiddleware
```

## Apps

Django can create apps (a collection of methods, templates, URLs, and forms for a particular sub-project of the whole website). All these apps combined make the whole website. Plus it maintains a certain degree of proper organization. Currently, we have two apps (user and core).

1. User App

The user app contains all of the necessary components and information about the user.

This includes the user's profile with active status and "send me alerts". This user's templates directory contains a base template which all app's HTML templates use. There are certain forms in this app for user registration and log-in.

## 2. Core App

The core app contains all of the components for the dairy feeding framework. This includes viewing, making, editing farms, ingredients, mixes, and diets, etc. The templates directory is organized into sub-folders for the different sections such as Diets, Farm, Mixes, etc. Be sure to add new sub-folders for new tasks to maintain organization.

## Models

Models are another term for tables. These models contain the column names (fields) with their corresponding data storage type and restrictions [3]. In `views.py` you can treat these as a query set with class variables (the unique fields), for example:

```
User.objects.first().username
```

This will get the username value for this first record. Changes to the model can be pushed to the database by running two commands.

```
python manage.py makemigrations
```

This will make a staging file with the changes to commit. This also points out any errors that may exist. You must not have any errors to continue.

```
python manage.py migrate
```

This will push the staging changes to the server's database. After this you can see the changes in the admin page.

## Views (Classes and Functions)

The views contains all the classes and functions that would be run for a particular page request [6]. These page request functions must take in at least one parameter (request); additional parameters can be passed in, but it must be configured in the `urls.py` to accept those extra parameters. Along with this, this function should be called from within the `urls.py` file with the URL path and ID set properly. See the URLs section for more information. The page request functions must return either a render of an HTML template or a redirect to another page (URL). Other helper functions can exist, but do not require the request parameter. The request parameter contains all information about the user requesting that page. The render can take in a dictionary of information you want to display on the HTML webpage. This data can be accessed in the HTML using the Python/Django template structures.

## Templates

Apps may or may not have templates. Templates only are required for any page requests that app may have. Templates are stored in the `<APP>/templates/<APP>/` directory. In the templates, you place your HTML code as well as any Python code with special syntax for any areas where you want to pass data to the HTML page. The `views.py` render call can pass in a dictionary which holds data that can be used. Django takes the HTML template document, edits it (to add the dictionary data to the document), then sends the rendered HTML file(s) to the user [4]. You can access the data in the dictionary by using the key in this format `<KEY>`. Below you can find an example of using “if” conditionals and “for” loops.

```
{% for farm in farms %}
    {% if farm.visible == True %}
```

```
        <h2> {{farm.name}} </h2> <br>
    {% else %}
        <h2> Hidden Farm </h2> <br>
    {% endif %}
{% empty %}
    <h2> farms is empty! </h2>
{% endfor %}
```

## URLs

Apps may or may not have a `urls.py` file. The `urls.py` file contains any URLs with matching functions the page is associated with. ‘`urlpatterns`’ holds an array of these paths where the first parameter is for the link, the second parameter is for the method it calls, and the last parameter is for the unique name this link has [5].

```
urlpatterns = [
    path('farms/', views.viewFarms, name="my-farms"),
    ...
]
```

All page requests are directed to these URL paths. Upon a request, the method or class is called, then renders an HTML document for the user and sends it to the user.

## Forms

You may create templates for any forms you may need in the `forms.py` file. Forms classes can use an existing model (table), custom entries, or both together [2]. For the existing model, you can specify which columns of the model you want to display as a form to the



user. It will take care of any input types and restrictions that are present in the models.py for that particular model. Here is an example with Address being a model with columns being Street, City, State, Country, ZipCode, Lat, and Long.

```
class AddFarmAddressForm(forms.ModelForm):  
    class Meta:  
        model = Address  
        fields = ['Street', 'City', 'State', 'Country', 'ZipCode']
```

For custom entries, every entry is assigned a form field from forms.Form.

```
class DocumentForm(forms.Form):  
    docfile = forms.FileField(label='Select a file')
```

In the views.py an instance of this form can be passed into the dictionary for the HTML template to use this. In the document, all you need is the form tag, csrf token, farmForm and your submit button. You can also add crispy (a Django to HTML formatting package) to make use of Bootstrap 4 in the Python form rendering process.

```
{% load crispy_forms_tags %}  
<form method="POST" enctype="multipart/form-data">  
    {% csrf_token %}  
    {{ farmForm|crispy }}  
    <div><button type="submit" class="btn btn-primary">Submit</button></div>  
</form>
```

This will take the fields in the form and make the HTML form for you. This data can then be read in by the request parameter in views functions.

## 4.4.2 Data Parsing

The weather data is pulled from weatherbit.io [12] API. A request to fetch data is done once per week and is stored in the database. The *weather.py* script contains all of the parsing logic. During the data fetch, only information needed is gathered and pushed to the weather info model; see Figure 4.5. If no error occurs then the model gets migrated into the database.

```
class Weather_Info(models.Model): # Need Min and Max temperatures too
    Farm_ID = models.ForeignKey(Farm, on_delete=models.CASCADE, null=True, blank=True)
    Time = models.DateTimeField(default=timezone.now)
    # Time = models.TimeField(auto_now=True, auto_now_add=True)
    ZipCode = models.IntegerField(null=True, blank=True)
    MinTemperature = models.IntegerField(null=True, blank=True)
    MaxTemperature = models.IntegerField(null=True, blank=True)
    Temperature = models.IntegerField(null=True, blank=True)
    Humidity = models.IntegerField(null=True, blank=True)
    DewPoint = models.IntegerField(null=True, blank=True)
    # Timestamp = models.CharField(max_length=50, null=True, blank=True)
```

Figure 4.5: An image of the weather model

For file uploads, a file type restriction is applied. Currently, only file types .csv and .xlsx are accepted. Changes can be made in the future if the restriction needs to expand. The file is parsed by iterating through each row and storing the column's value to its appropriate variable name. Figure 4.6 shows a small code snippet of ingredient data being stored in each iteration. Once the file has been parsed, the data is pushed to the database and updates the library. The systems does not need to differentiate the type of file beings parsed since each file upload is specific to its tab. If a farmer is uploading ingredients, then they can only do it in the ingredients tab.

```
if str(row['Ingr_DM']) != "NaN": abc.Ingr_DM = float(row['Ingr_DM'])
if str(row['Ingr_DE_base']) != "NaN": abc.Ingr_DE_base = float(row['Ingr_DE_base'])
if str(row['Ingr_Conc']) != "NaN": abc.Ingr_Conc = float(row['Ingr_Conc'])
if str(row['Ingr_NDF']) != "NaN": abc.Ingr_NDF = float(row['Ingr_NDF'])
if str(row['Ingr_DNDF48_NDF']) != "NaN": abc.Ingr_DNDF48_NDF = float(row['Ingr_DNDF48_NDF'])
if str(row['Ingr_ADF']) != "NaN": abc.Ingr_ADF = float(row['Ingr_ADF'])
```

Figure 4.6: An image of the weather model

### 4.4.3 Alerts

Users who are given staff status and have the ‘Send Me Alerts’ box checked shown in Figure 4.7 will receive email notifications whenever user(s) make a change to any ingredients after 90 days from when the ingredient was last edited/created.

<input type="checkbox"/> USER	ID	IS ACTIVE	SEND ME ALERTS
<input type="checkbox"/> oli1230	10	✓	✗
<input type="checkbox"/> dvarakh	9	✓	✓
<input type="checkbox"/> silvasym	8	✓	✗
<input type="checkbox"/> admin	3	✓	✗

Figure 4.7: An image of some of the user profiles

The email notifications mentioned when a user modifies an ingredient appear in the format shown in Figure 4.8.

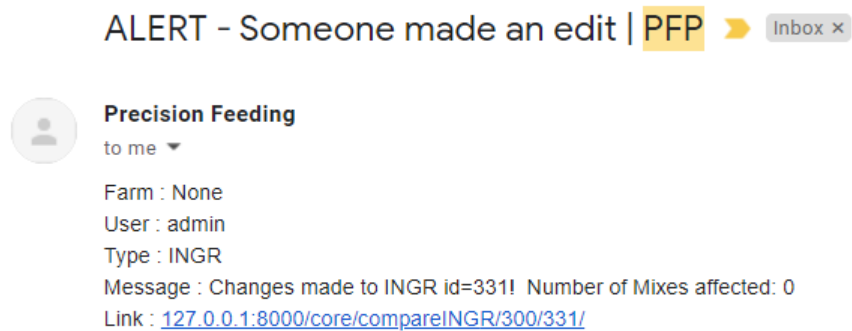


Figure 4.8: An image of an ingredient edit notification email

The staff/admin can click on the link and it will take them to this page as shown in Figure 4.9.



Figure 4.9: Staffs link to commit user edits

The staff/admin can see the edits made in the ingredients. They can view which mixes and storage records need to be edited to use the updated ingredient's data. You can see a sample of the sections in Figures 4.10, 4.11, and 4.12.

The screenshot shows a table titled 'Comparing Ingredients'. The table has three columns: 'Title', 'Old', and 'New'. The rows are color-coded: red for fields that have changed and green for fields that have remained the same. The fields and their values are as follows:

Title	Old	New
id	357	358
user	admin	admin
old_ingr	331	357
timestamp	April 19, 2021, 4:19 p.m.	April 29, 2021, 11:44 a.m.
visible	False	True
Ingr_Loc	NRCLib	NRCLib
Ingr_Code	NRC16F9	NRC16F9
Ingr_Name	Barley hay - Update - Edit 2	Barley hay - Update - Edit 3
Ingr_Category	Grain Crop Forage	Grain Crop Forage
Ingr_Type	Forage	Forage
Ingr_DM	89.401	89.401
Ingr_DE_base	2.618	2.618
Ingr_Conc	0.0	0.0

Figure 4.10: Staffs view to see a user edits on an ingredient

Possibly affected Mixes

**Test Name 2**

Mix ID	Mix Ingr	Ratio DM%
353 <span style="color: green;">●</span>	<a href="#">Sugar - Update 1</a>	49.98
357 <span style="color: red;">●</span>	<a href="#">Barley hay - Update - Edit 2</a>	50

Edit
Delete

**test**

Mix ID	Mix Ingr	Ratio DM%
355 <span style="color: green;">●</span>	<a href="#">Blood meal, high dRUP</a>	30
357 <span style="color: red;">●</span>	<a href="#">Barley hay - Update - Edit 2</a>	70

Edit
Delete

Figure 4.11: Staffs view to see possibly affected mixes on an ingredient edit

Possibly affected Storages

Farm_ID	Ingr_Store_ID	Ingr_Store_Name	Capacity	Unit	Affiliation	Ingr_Type	Ingr_Form	Edit
Sample - admin	<a href="#">Barley hay - Update - Edit 2</a> <span style="color: red;">●</span>	Barley Hay	256	Tons	Mixer	Forage	Liquid	<a href="#">Edit</a>
Test Farm 1 - admin	<a href="#">Barley hay - Update - Edit 2</a> <span style="color: red;">●</span>	Barley Hay	100000	Tons	Mixer	Forage	Solid	<a href="#">Edit</a>
Test Farm 1 - admin	<a href="#">Barley hay - Update - Edit 2</a> <span style="color: red;">●</span>	Barley hay	1	Tons	Mixer	Forage	Liquid	<a href="#">Edit</a>
Test Farm 1 - admin	<a href="#">Barley hay - Update - Edit 2</a> <span style="color: red;">●</span>	Barley Hay 1	4	Tons	Mixer	Forage	Liquid	<a href="#">Edit</a>

Figure 4.12: Staffs view to see possibly affected storage ingredients on an ingredient edit

The staff can click the Commit Changes button which will update all the mixes and storage records shown to use the new/updated ingredient.

## 4.5 Docker

To host the web framework on Virginia Tech’s cloud container cluster, we needed to create a Docker repository to hold a Docker image of the web app. The Docker image is created using the Dockerfile located at ‘DairyFeedingFramework/Dockerfile’ in the project directory. A Docker image declares the state of a Docker container when running the container on a cluster. To get started with Docker, the developer must first install the Docker desktop application from the following link.

<https://docs.docker.com/desktop/>

### 4.5.1 Dockerfile

```
dockerfile > ...
1 FROM centos:8
2
3 RUN yum -y install bash curl git rsync vim httpd epel-release python3 python3-pip python3-ldap python3-mod_wsgi
4
5 COPY api.conf /etc/httpd/conf.d/api.conf
6 COPY start.sh /start.sh
7 COPY ./PFP_WEB /var/www/html
8 COPY requirements.txt /var/www/html
9 COPY PythonPackageInstaller.py /var/www/html
10 WORKDIR /var/www/html
11 RUN python3 PythonPackageInstaller.py requirements.txt
12 ENTRYPOINT ["/start.sh"]
```

Figure 4.13: Dockerfile for the dairy\_feeding\_framework Docker image

The first line of the Dockerfile shown in Figure 4.13 is a ‘FROM’ command that specifies the base Docker image that our Docker image will build from. Next, the ‘RUN’ command will install dependencies in the container once the CentOS8 Docker image is loaded. After the dependencies are installed to our Docker image, the ‘COPY’ command is used to copy files from our local machine to the Docker image so that all of the content needed to run the web framework are present in the Docker image. Once the ‘COPY’ commands are complete, we set the working directory in our Docker image by using the ‘WORKDIR’

command which is set to 'var/www/html'. Next, another 'RUN' command is used to run the PythonPackageInstaller.py with the requirements.txt file that were previously copied from our local machine to the Docker image. This will install all of the web frameworks dependencies so the Web framework can run properly. Finally, the last command in the Dockerfile is the 'ENTRYPOINT' command that will configure the container to run the start.sh file once the container starts [7].

When any changes are made to the Dockerfile, the developer will need to push the changes to the Docker Hub repository that stores the Docker image. The following steps will correctly push the Dockerfile changes to the repository.

On a command line tool, navigate to the project's root directory 'DairyFeedingFramework/' Run the command 'docker build -t ewebb4505/dairy\_feeding\_framework' Run the command 'docker push ewebb4505/dairy\_feeding\_framework'

- on a command line tool, navigate to the project's root directory 'DairyFeedingFramework/'
- Run the command 'docker build -t ewebb4505/dairy\_feeding\_framework .'
- Run the command 'docker push ewebb4505/dairy\_feeding\_framework'

Once the Docker image is successfully pushed to the Docker Hub, the developer will have to redeploy the container on Virginia Tech's cloud container cluster with the following steps.

- Navigate to the following link <https://cloud.cs.vt.edu/p/c-l4mcr:p-z6pfk/workloads>
- Click on the checkbox beside 'pfp\_web' on the namespace table. The namespace will also be labeled 'pfp\_web'
- Next click on the 'Redeploy' button. This will take time to restart the container and pull the image from the Docker Hub repository.

### 4.5.2 Apache Configuration

The Apache Configuration file that is copied from our local working directory to the `etc/httpd/conf.d/` directory in the Docker image will configure the Apache web server that our Docker image runs.

Currently, the `api.conf` file does not display the web framework but the Apache test page is currently displayed at the host URL. Getting the web framework to be served by the Apache Web Server in the Docker container will be a part of the future work for this project since we had to make a switch from our web framework being host by the Virginia Tech rlogin service to the Virginia Tech Cloud Container Cluster. Figure 4.14 shows the current state of the `api.conf` file.



```
WSGIScriptAlias / /PFP_WEB/wsgi.py
WSGI PythonPath /PFP_WEB
# WSGIPythonHome /venv
WSGI PassAuthorization On

Alias /static /PFP_WEB/core/static

ServerName dis.cs.vt.edu

<VirtualHost *:80>

    # ServerName dis.cs.vt.edu
    # ServerAlias www.dis.cs.vt.edu
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    WSGIScriptAlias / /PFP_WEB/wsgi.py
    WSGIDaemonProcess PFP_WEB_app
    # python-path=/PFP_WEB
    # python-home=/PFP_WEB/venv
    WSGIProcessGroup PFP_WEB_app

    # ErrorLog ${APACHE_LOG_DIR}/error.log
    # CustomLog ${APACHE_LOG_DIR}/access.log combined

    Alias /static /core/static
    <Directory /core/static/>
    | Require all granted
    </Directory>

    Alias /media /media
    <Directory /media>
    | Require all granted
    </Directory>

    <Directory /PFP_WEB/>
    | <Files wsgi.py>
    | Require all granted
    | </Files>
    </Directory>

    # RewriteEngine On
    # RewriteCond %{SERVER_PORT} !^443$
    # RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]

</VirtualHost>
```

Figure 4.14: api.conf file used by Docker image to configure Apache Server

# Chapter 5

## Methodology

### 5.1 Outline

For the dairy feeding framework, there will be two types of users: farmers and data management users. Farmers will use the dairy feeding framework to virtually monitor the dairy production at each of their registered farms. Farmers will interact with interfaces that will give them the ability to create diets, download dairy production reports, upload data reports, and view visuals that portray milk production anomalies and other relevant statistics. Data management users will have access to a farmer's dairy production data to make inferences on the quality of the dairy production. Also, data management users will be notified when farmers make changes to a diet.

### 5.2 Farmer Goals

**Farm Management:** The farmer will be able to register multiple farms under their user profile. Each farm will have farm-specific information.

**Ingredient Retrieval:** The farmer will be able to copy ingredients from the default (global) ingredient dataset into their local ingredients table and edit these ingredients if necessary. Farmers will be able to create mixes composed of the ingredients in their local ingredients table.

**Create Mixtures:** The farmer will be able to create ingredient mixtures composed of ingre-

dients from the farmer's local ingredients table. The farmer will determine the amount of each ingredient in the mixture by percentage and will be able to edit and delete mixtures after creation.

Create Diets: The farmer will be able to create diets composed of mixtures from their list of mixtures. Farmers can assign a diet to a particular herd of cows or an individual cow.

File Upload and Data Injection: The farmer will be able to upload DHIA (Dairy Herd Improvement Association) data reports which the web framework will analyze and provide useful information to the farmer.

Retrieve Dairy Production Data: The farmer will be able to view and download a dairy production data report that shows milk production statistics over time including anomalies and visual diagrams of farmer-relevant data.

### 5.2.1 Farm Management Tasks

Refer to Figure 5.1 for the Farm Management Tasks work flow diagram. Following are the task to complete the Farm Management user goal.

- a) Farmer registers a farm via a form.
- b) Update farm information as needed. This task is reliant on task a.

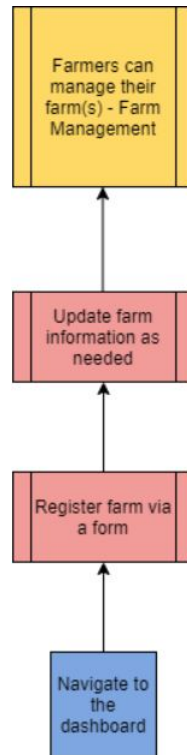


Figure 5.1: Farm registration and management work flow

### 5.2.2 Ingredient Retrieval Tasks

Refer to Figure 5.2 for the Ingredient Retrieval Tasks work flow diagram. Following are the task to complete the Ingredient Retrieval farmer goal.

- a) Display global library of ingredients: retrieve global ingredients data from database.
- b) Display farmer's local ingredients: retrieve the farmer's ingredients stored in the database and present it to the farmer. Reliant on tasks c, d and e.
- c) ingredients from the global library to the local table: the copy functionality is necessary for a farmer to copy an ingredient from the default/global table into their local ingredients table to later create mixes. This task is reliant on task a.
- d) Update/edit ingredient data in the local ingredients table: a farmer will be presented with the tools to edit or update individual ingredient's data from their local ingredients

table. The farmer will be able to edit ingredients data through a form on a separate page when the user indicates that they need to edit an ingredient's data. This task is reliant on task b.

e) Clone local ingredients: a farmer will be presented with the tools to clone ingredients that already exist in their local ingredients table. This will be more efficient than cloning from the default/global table, as the farmer will not have to edit the ingredient all over again. This task is reliant on task b.

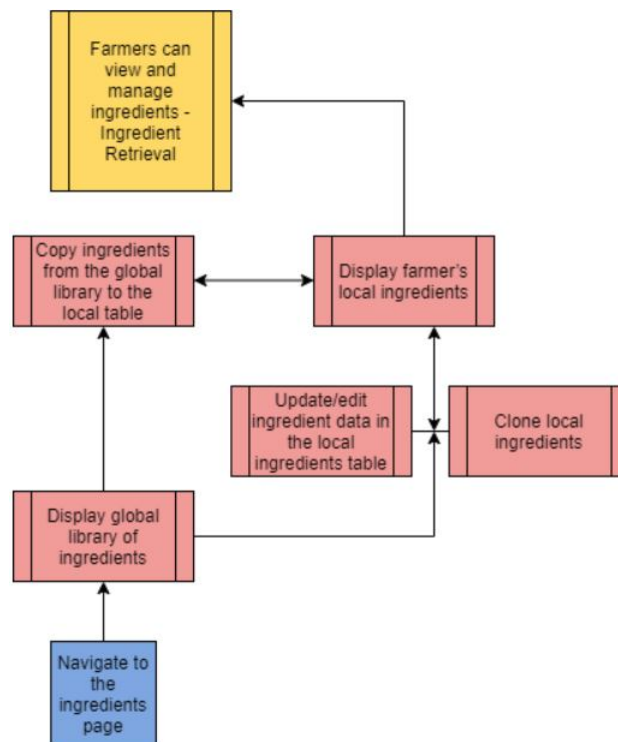


Figure 5.2: Ingredient retrieval task work flow

### 5.2.3 Create Mixtures Tasks

Refer to Figure 5.3 for the Create Mixtures Tasks work flow diagram. Following are the task to complete the Create Mixtures farmer goal.

a) Display farmer's local ingredients: retrieve the farmer's ingredients stored in the database

and present it to the farmer.

b) Add an ingredient to a mixture. This task is reliant on task a.

c) Edit percentage of ingredients in a mixture: the farmer will be given an input field to enter the percentage of an added ingredient in the mixture. This task is reliant on task b.

d) Notify the farmer when mixture composition percentage is not 100%: display a message to the farmer that the sum of ingredient percentages do not equal 100%. This task is reliant on task c.

e) Edit the mixture name: give the farmer the ability to edit the mixture name when creating the mixture. This task is reliant on task c.

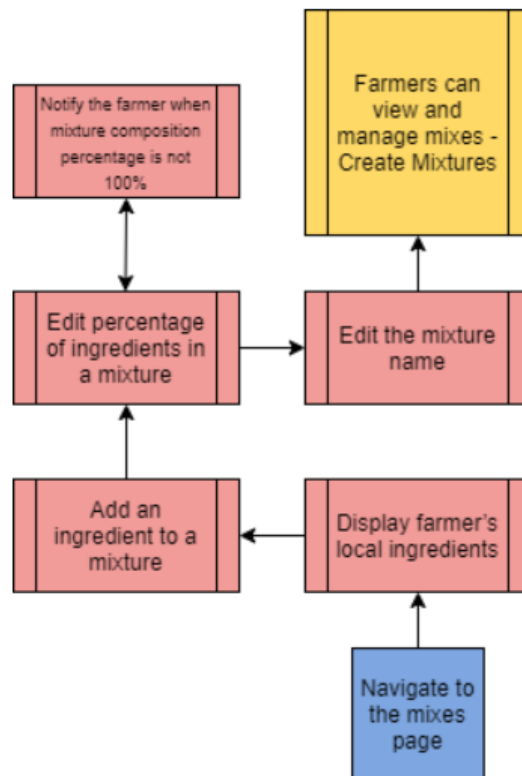


Figure 5.3: Create mixture task work flow

### 5.2.4 Create Diets Tasks

Refer to Figure 5.4 for the Create Diets Tasks work flow diagram. Following are the task to complete the Create Diets farmer goal.

- a) Display the farmer's mixtures: retrieve the farmer's mixtures stored in the database.
- b) Add a mixture to the diet. This task is reliant on task a.
- c) Adjust the appropriate percentages of each mixture for a given diet. This task is reliant on task b.
- d) Assign a start date for the diet. This task is reliant on task c.
- e) Assign diet to a particular pen of cattle or an individual cow. This task is reliant on task d.

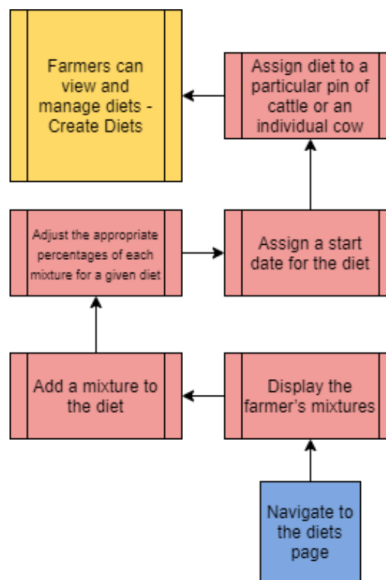


Figure 5.4: Creating the diets task work flow

### 5.2.5 File Upload and Data Injection Tasks

- a) Add a file via file-dialog box.
- b) Receive confirmation that file is stored. This task is reliant on task a.

### 5.2.6 Retrieve Dairy Production Data

Refer to Figure 5.5 for Retrieve Dairy Production Data work flow diagram. Following are the task to complete the Retrieve Dairy Production Data farmer goal.

- a) Pull dairy production data from the database.
- b) Perform data analysis on the dairy production data with R. This task is reliant on task a.
- c) Render the results of data analysis to the web interface. This task is reliant on task b.
- d) A Farmer can download results to their local computer. This task is reliant on task c.

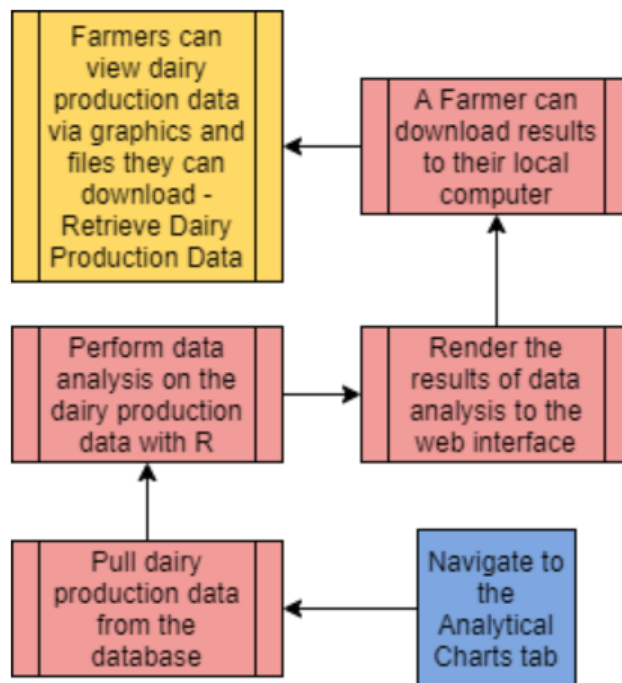


Figure 5.5: Retrieve dairy production data work flow

## 5.3 Data Management User Goals

Data Management User Goals: Receive Notifications Upon Dietary Changes: Data managers will receive notifications when diets have been altered if the change has taken place within



35 days of creation or the last edit. The data managers will then review the change to ensure that it is an acceptable change. Monitor Farmers Production Data: Data managers will be able to monitor milk production data from farms. This data includes the amount of milk yielded, contents of the milk and other additional fields related to the quality of the milk (such as spectral DHIA data).

### 5.3.1 Receive Notifications Upon Dietary Changes Tasks

Refer to Figure 5.6 for Receive Notifications Upon Dietary Changes work flow diagram. Following are the task to complete the Receive Notifications Upon Dietary Changes data management user goal.

- a) Received email that a farmer has changed diet data.
- b) View data changes side by side. This task is reliant on task a.
- c) Confirm that the changes to the diet data are accurate. This task is reliant on task b.

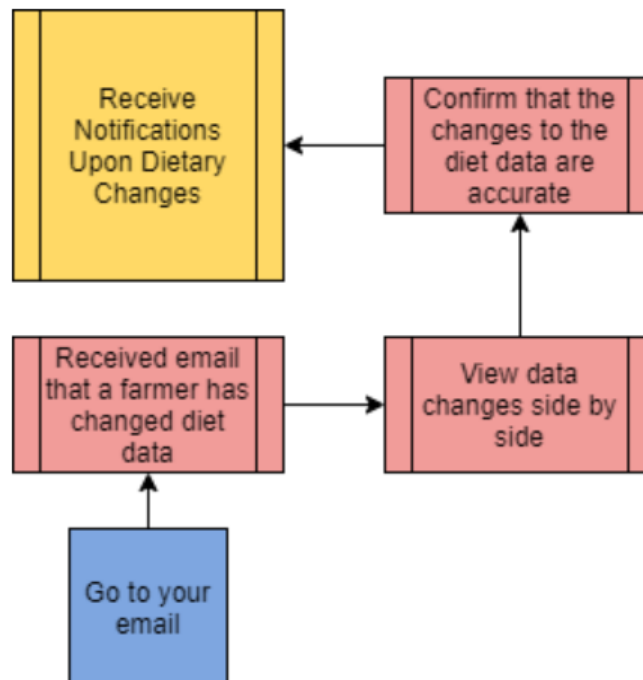


Figure 5.6: Notifications when editing an ingredient work flow

### 5.3.2 Monitor Farmers Production Data

Refer to Figure 5.7 for Monitor Farmers Production Data work flow diagram. Following are the task to complete the Monitor Farmers Production Data data management user goal.

- a) Connect to R with Python to run the anomaly detection tool.
- b) Calculate diet changes and production statistics (run R script). This task is reliant on task a.
- c) Produce visuals based on past/current data statistics, data anomalies, and recommended future changes/actions. This task is reliant on task b.

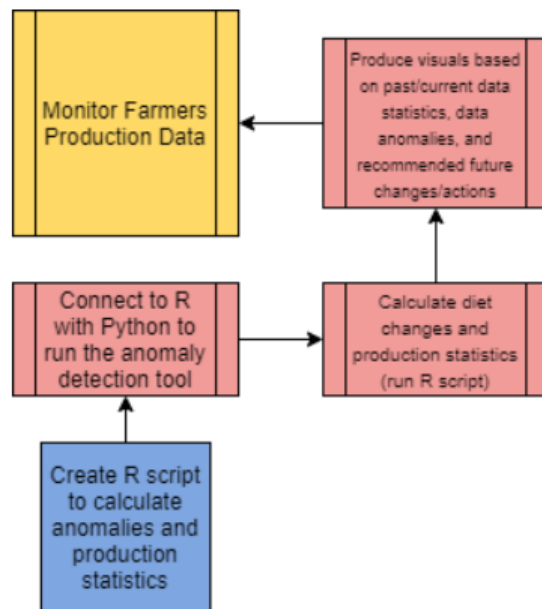


Figure 5.7: Monitoring data in production work flow

# Chapter 6

## Testing

In this chapter, we'll discuss the steps on how to test each function of our website.

### 6.1 Farm

To test the creation of the farm, you will need to do the following:

1. Run server
2. Log in (then the website takes you to the home page)
3. Click the “Add Farm” button
4. Enter your farm information
5. The newly added farm should be displayed once the information is saved.

### 6.2 Ingredients

To test the “Make Copy” of default ingredients, you will need to do the following.

1. Navigate to the Ingredients Tab.
2. Choose “Make Copy” from the defaults ingredients.
3. Make certain changes and save.

4. The edited mix should be in My Ingredients. This mix should have a reference ID to the original default mix (you can check on the admin ingredients tab for “old\_ingr”).

To test the “Edit” of my ingredients, you will need to do the following:

1. Navigate to the Ingredients Tab.
2. Choose “Edit” from my ingredients.
3. Make certain changes and save.
4. Should create a new record for the edited mix.
5. Should set old ingredient’s, (IngrID prior to edit), visible to false.
6. Should see the updated ingredient in my ingredients

To test the “Make Copy” of my ingredients, you will need to do the following:

1. Navigate to the Ingredients Tab.
2. Choose “Make Copy” from my ingredients.
3. Make certain changes and save.
4. Should create a new record for the edited mix.
5. Should keep the old ingredient’s, (Ingr\_ID prior to edit), visible to true.
6. Should see both old and updated ingredient in my ingredients

## 6.3 Mixes

Testing for mixes is similar to testing ingredients. The only difference is that the mixes have a JSON string object to hold the ingredients and its percentages together. They follow the

same structure with the “Edit” button. It creates a new record for every edit and sets the old mix’s visible to false. You can also remove a mix if needed.

## 6.4 Diets

Testing for diets is similar to testing mixes. The only difference is that the mixes have a JSON string object to hold the mixes and its amount together.

## 6.5 Storage

To test the creation of a storage ingredient record, you will need to do the following:

1. Navigate to the Storage Tab.
2. Click on the “Add Storage” button.
3. Fill in the information and save.
4. The newly added storage ingredient should be displayed once the information is saved.

## 6.6 Data Uploading

To test the file uploading, data parsing, and saving, you will need to do the following:

1. Log in as admin/staff.
2. Go to the staff links.
3. If you have Select Sires Animal data, then choose Select Sires, then Upload Animal Data. If you have TMR data, then choose Upload TMR Data.

4. Choose the farm you are uploading this data for. Then upload the file.
5. The process should not take long before it directs you to the files uploaded page.
6. You should see the newly added file at the bottom of the list.
7. If you navigate to the Django Administrator page to the respective table, the data should be uploaded to the farm you chose.

## 6.7 Staff Notifications and Commit

To test the staff notifications and the commit changes option, you will need to do the following:

1. Change `EDIT_NOTIFY_DAYS` in `PFP_WEB/core/views.py` to 0 or a number smaller than the age of the ingredient/mix you wish to edit.
2. Make sure there is at least one staff who has `send_me_alerts` on.
3. Make sure the ingredient's age is larger than `EDIT_NOTIFY_DAYS`.
4. Make sure the ingredient is used in Mixes and Storage tabs. (Not needed)
5. Edit that ingredient.
6. A notification should appear indicating a notification to the staff.
7. Every staff with `send_me_alerts` on, will get an email notification.
8. The email should contain a link only accessible to Staff/Admin.
9. The link will display all the changes made in ingredients. It will also show you where the old ingredient was used in Mixes and Storage tabs if step 4 is complete.

10. There is a commit button at the bottom. Click it.
11. This should commit all the changes. Mixes and Storage records will now use the updated/new ingredient.
12. The link will refresh and should show you no possible changes to Mixes or Storage records.

## 6.8 R with Python

To test the functionality of the R to Python to HTML bridge, you will need to do the following:

1. Login as admin
2. Click the analytics page
3. Select “Add R File”
4. Upload the R file using provided GUI
5. Select “Submit”
6. You should now see a new line of input on the analytics page. Select either “view” or “run” – if you select “view” you must then select “run” and in both situations you will be redirected to the output of the R code that was input in an HTML page format.

## 6.9 Troubleshooting

Django offers its own debugging tool to help developers troubleshoot errors that arise within their websites. When errors occur, the debugger will output useful information so developers/programmers can find where and why these errors occur. The website is currently

running in production, so the debugger has been turned off. However, developers can turn the debugger on by navigating to the `settings.py` folder found at the following path:

```
PFP_WEB/PFP_WEB/settings.py
```

After navigating to the file, programmers will need to change the `DEBUG` and `ALLOWED_HOSTS` settings. `DEBUG` needs to be changed to `true` and the `ALLOWED_HOSTS` setting needs to be changed to an empty list.

```
DEBUG = True
```

```
ALLOWED_HOSTS = []
```

After changing these settings, the debugger will be on and programmers will be able to troubleshoot any errors within the site.

After fixing any errors, it's important for the programmers to change the settings back to what they were before. The `DEBUG` and `ALLOWED_HOSTS` settings should look like this:

```
DEBUG = False
```

```
ALLOWED_HOSTS = ['*']
```



# Chapter 7

## Lessons Learned

### 7.1 Timeline/Schedule

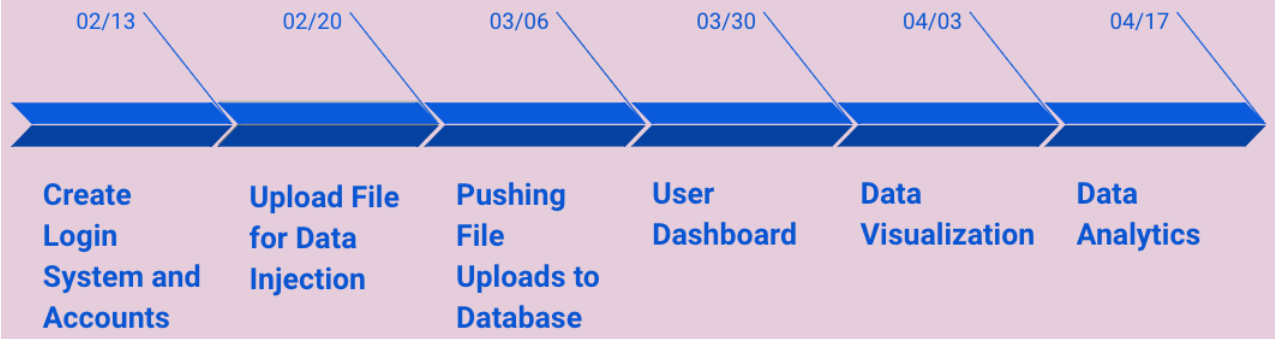


Figure 7.1: An image of our timeline

The timeline shown in Figure 7.1 was the schedule we had set for completing this project. We split up the project into six sub-tasks and organized them based on priority or logical order.

*Create Login System and Accounts:*

We decided to work on the login and accounts functionality first. The reason is so we can build a rough web interface that will act as our foundation. The login system and accounts weren't complicated to implement, but it allowed us to set up some of the basic web functionality into our web page.

*Upload File for Data Injection:*

Next, we supported file upload functionality to our web page. From the front-end perspective, this functionality simply allows users to upload TMRTracker templates and Select Sires files. The file(s) containing the data must be in the range of allowed format. From the back-end perspective, we are making sure that the file is being parsed correctly. This also includes making sure the parsed data are stored in their appropriate table in the database.

#### *Pushing File Uploads to Database:*

Next, after properly parsing data from the uploaded file, we pushed the data to our database. Originally, we are using SQLite as our temporary Database Management System but had moved on to MySQL to support client and server architecture.

#### *User Dashboard:*

The user dashboard covers the majority of the user interface for the website. We made sure that our website is accessible and simple enough for farmers to use. We've organized different pages into tabs on the side menu so farmers can easily access information. We've also made sure to set a common style/format for all the pages. This is to allow design consistency throughout the website and for any new pages that may get added later on.

#### *Data Analytics:*

Admins and staff who need to upload their R scripts can upload their scripts and run said scripts from the website. However, as of current, any data analytics to be gathered from running R scripts would have to be done within the R script itself, as the current functionality of the site does not support otherwise.

#### *Web Hosting:*

Unfortunately, the web framework could not be hosted by the Virginia Tech rlogin cluster since the rlogin cluster's operating system is CentOS7 which is incompatible with the project's version of Django. We were given the task to containerize the project and run the project container on the Virginia Tech cloud container cluster service. Since the cloud container cluster using Kubernetes for container orchestration and workload management, we

started trying to make YAML files that would create the Kubernetes deployment container. Thanks to Chris Arnold in Virginia Tech Tech Staff, we learned that we could manually create the deployment on the cloud container cluster user interface by giving our project Docker image and a couple of other parameters. The deployment container works but the Apache configuration file currently is not set up with our web framework. The lessons learned in this part of the project was getting a high level introduction to containerization with Docker and container orchestration with Kubernetes.

# Chapter 8

## Future Work

As of the point in time of producing this document, the requirements given by the clients have been met in accordance to what has been given. However, while the development team has accomplished meeting expectations for herd diets (ingredients/mixes), front end functionality, farmer sign up and log in, R to Python data analysis connection, and personalized data storage, what is left to be completed are the following items:

1. Interface and data management tool for bovine product progress (this is separate from the completed diet and ingredient data management tool)
2. R anomaly detection
3. Statistical data visualization tool
4. Weather data visualization interface
5. Full data management interface for cumulative data resources (an interface for the farmer to manage **all** of their data)
6. Cumulative interface that links to and incorporates every aspect of the web site (this is a home page that gives a basic interface to each tool the website provides in a convenient manner)
7. Need to add Pen information to diets. Must retrieve Pen information from Select Sires or through other means.

8. Staff notifications for user edits for ingredients and mixes within X number days; you will need to set the value of X (`EDIT_NOTIFY_DAYS`) in `PFP_WEB/core/views.py`.
9. Staff notifications will need to expand to diets.
10. Staff commit user edits may need a backtrack option. Enable the ability to revert committed changes and or see list of all staff commits with detailed information.
11. Edit email settings on `PFP_WEB/PFP_WEB/settings.py` to use an account more professional to the Dairy Feeding Project in production.
12. Create testing scripts to test more efficiently.
13. Correct the Apache Configuration file (`api.conf`) to communicate with Django and display the web framework at the `dis.cs.vt.edu`
14. Add commands to install the R programming language on CentOS8 and set R to the path variable in the Dockerfile.
15. Set up MySQL container with persistent volume on Virginia Tech's cloud container cluster.

# Chapter 9

## Acknowledgement

We would like to acknowledge our clients and members of the Dairy Feeding Framework project.

Mark Hanigan, Dairy Science Professor

Wu Feng, Computer Science Professor

Leticia M. Campos, Dairy Science Ph.D. Student

Sonal Jha, Graduate Research Assistant

Charles Zawacki, Research Assistant

Yanlin Du, Research Assistant

# Chapter 10

## References

- [1] Django Developers, “Documentation Django 3.2,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/>. [Accessed: 04-May-2021].
- [2] Django Developers, “Documentation Django Forms,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/forms/>. [Accessed: 04-May-2021].
- [3] Django Developers, “Documentation Django Models,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/db/models/>. [Accessed: 04-May-2021].
- [4] Django Developers, “Documentation Django Templates,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/templates/>. [Accessed: 04-May-2021].
- [5] Django Developers, “Documentation Django URLs,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/http/urls/>. [Accessed: 04-May-2021].
- [6] Django Developers, “Documentation Django Views,” 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/http/views/>. [Accessed: 04-May-2021].
- [7] Docker Developers, “Orientation and setup,” Docker Documentation, 2021. [Online]. Available: <https://docs.docker.com/get-started/>. [Accessed: 04-May-2021].
- [8] RPY2 Developers “Introduction to rpy2,” rpy2 3.5.0dev documentation, 2017. [Online]. Available: <https://rpy2.github.io/doc/latest/html/introduction.html>. [Accessed: 04-May-2021].
- [9] RStudio Developers, “Introduction R Markdown,” 2020. [Online]. Available: <https://rmarkdown.rstudio.com/lesson-1.html>. [Accessed: 04-May-2021].
- [10] Select Sires Development Team, Select Sires, 2021. [Online]. Available:

<https://www.selectsires.com/> [Accessed: 04-May-2021].

[11] Digi-Star TMR Tracker Solution, Digi-Star, 2021. [Online]. Available: [https://digi-star.com/solutions/2-2/TMR\\_Tracker](https://digi-star.com/solutions/2-2/TMR_Tracker). [Accessed: 13-may-2021]

[12] WeatherBit Developers, Weatherbit, “Weather API for You,” 2021. [Online]. Available: <https://www.weatherbit.io/>. [Accessed: 29-Apr-2021].