Spring 2023

Fridge Tracker and Recipe Provider : FridgeChamp

Matt Dudek
mcd81@uakron.edu

FridgeChamp: Track your fridge's content, recipes, and meal plan

Matthew Dudek

Department of Computer Science

Honors Project

Submitted to

The Williams Honors College

The University of Akron

Table of Contents

Table of Figures

Brief Overview

FridgeChamp is a website geared towards home chefs who want to efficiently manage: Their fridge/pantry contents; All of their recipes and others; And be able to plan meals throughout the week. Online recipes in general contain an egregious amount of “bloat”, see any recipe linked from Pinterest, where the actual recipe is almost buried under stories, ads, and other annoying things. In addition, the fridge tracking component is rare to find as most are geared towards restaurants and nursing homes. The meal planner aspect is fairly easy to find, most do it on a whiteboard on their fridge, but digital is easier and allows an easy way to plan out a week and see all of the recipes available.

Issues Addressed

Many home cooks have to keep track of an exhaustive list of recipe links and try not to lose any or forget about them, or put them into a book which becomes quite large and frustrating to look up a specific recipe. Also many ingredients for specific recipes are not entirely used so they expire and are trashed before they can be consumed, costing money and generating excess waste. Additionally many people will loosely plan their week and either end up with a lot of leftover ingredients or won’t have enough, both bad outcomes.

Solution

FridgeChamp is a web application that home cooks will be able to use to track their food stock. FridgeChamp will also be able to track and sort all the recipes added and provide an option for a user to keep the recipe private or allow it to be public. The food tracked will optionally have an expiration date that will alert the user if an item is going to expire soon. Also food may have an “alert” level to let the user know if the total stock goes under a certain threshold. Currently each user will have their own account and fridge. Also the user will have a planner for meals that can optionally go up to a year in advance and is extremely useful for grocery lists and managing home cooking. FridgeChamp also has a recipe database so that the user can store all of their various recipes in one place without any bloat.

High Level Functions

- Users can login or signup and register for an account
    - Valid email is required
- Overview is a placeholder and currently greets the user
- Fridge
    - Items can be sorted by category
    - Items, Instances, and Alerts can be created
    - Instances can be edited
    - Links to an Expired overview and Alert overview which contain information about Expired/Soon to be expired and instances under their alert value respectively.
- Recipes
    - Users will be able to view, add, edit, remove, and make their recipes public.
        - If the website every becomes more popular a review system will be needed in order to approve recipes before they become accessible by everyone
    - There is a recipe builder for users to create recipes.
    - Every recipe will have a “I made this” button to subtract the used ingredients from their fridge, currently fridge’s with insufficient quantities set subtracted ingredients to 0.
- Meal Planner

- Users can make a list of days with breakfast, lunch, and dinner and can set a recipe for each meal. Users can schedule up to a year in advance and old days are cleared out every time the planner is viewed.

Design

I am focused on the key elements and functionality before proceeding with the frontend development, besides some UI testing with basic CSS. For the backend of the website Django will be used. Django was picked because it is open source, heavily documented, and becoming fairly used in industry, Django is also fast to pick up and based on Python which is easy to develop. The back-end database is PostgreSQL which is also open source and easy to use/integrate with Django. In order to give the user a better experience and make it easier to edit things I plan to implement a frontend coded in React after I graduate. The reason I wasn't able to incorporate the frontend before graduation is due to the fact that I worked on this solo and had to essentially learn Django from the ground up, which is no small task. I have built a server with enough capabilities to become a web server and that is where the website/database will be deployed. A domain of fridgechamp.com has also been acquired and configured.

In Depth Breakdowns

Physical Server

The computer hosting the web server and database is running an AMD ryzen 5 5600G processor with 32GB of DDR4 3200 MHz RAM, on a Gigabyte B450M DS3H Motherboard in a Thermaltake Versa H17 Micro ATX Tower Case. The operating system for the computer is Ubuntu server 20.04 and is hard wired to my router with an ethernet cable, and a static ip has been set for the server.

Port Forwarding

My router has been configured to forward traffic from ports 80 and 443 (http and https) to the server to provide external access to the server while maintaining network security.

Domain

I have purchased a domain through GoDaddy to get an easier method of access externally. The name “fridgechamp” was chosen from a randomly generated list and it fits the website well.

SSL

In order to use HTTPS an SSL certificate is necessary to prove to the user’s browser that the website is what it is claiming to be. I obtained a certificate from ZeroSSL and installed it into my server. The certificate expires every so often and needs to be redone.

Django

"Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source."

Though the documentation Django was easy to learn, but immensely complex due to the sheer amount of functionality it provides. Django handles the database interactions by wrapping relations into "models", which are basically tables that provide a very straightforward way to access the database, search for specific items, and everything else that would be done by SQL queries. In addition to the database management Django makes it easy to structure web pages by having each be a "view" that can supply an HTML "template" with contexts, or basically variables, to have very easy to create dynamic web pages. More info on templates will be broken down later.

PostgreSQL

"PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance."

Like Django, PostgreSQL is open-source and has very nice documentation. It also has a multitude of 3rd party management programs that when combined with Django make for a very comprehensive way to manage data.

Database relations

I went through a few iterations while trying to figure out the best way to store data regarding the fridge tracker, recipes, meal planner, and users themselves. I ended on the following diagram, which was made in a chart tool called draw.io. (See below)

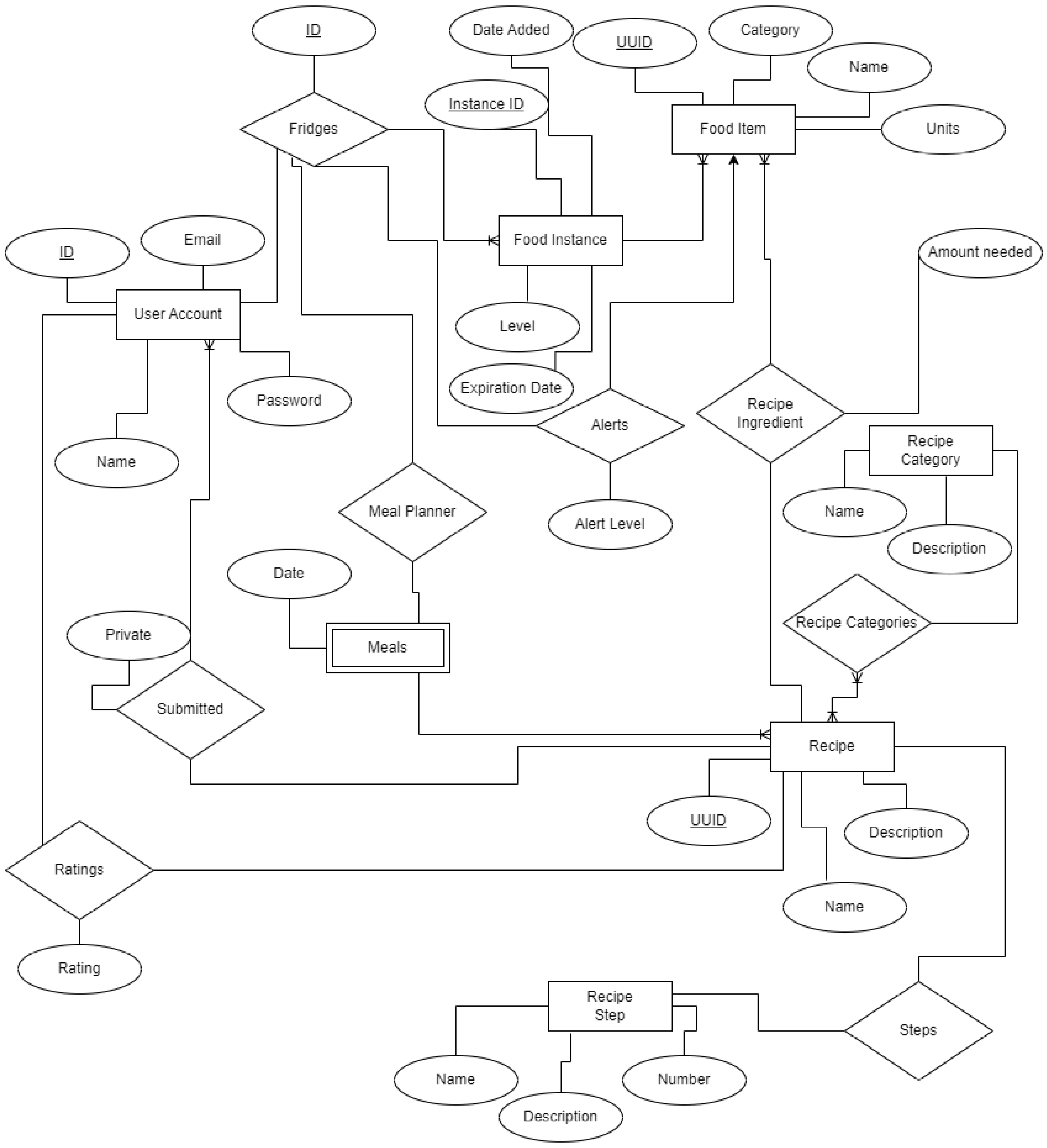

ID
Date Added
UUID
Category
Name
Instance ID
Fridges
Food Item
Units
Food Instance
ID
Email
Amount needed
User Account
Level
Password
Expiration Date
Alerts
Recipe Ingredient
Name
Recipe Category
Meal Planner
Alert Level
Name
Description
Date
Private
Recipe Categories
Meals
Submitted
Recipe
UUID
Description
Ratings
Name
Rating
Recipe Step
Steps
Name
Number
Description

Figure 1

- User:
  - The primary key is the user’s id, which is an automatically generated integer.
  - An email field (fancy string with email validation built in) has to be unique and will eventually be used for password recovery.
  - Another string field for a user’s name.
  - A field for a password, which boils down to a string field with password hashing, salting, and validation in it.
  - And lastly a foreign key field which just points to the id of the user’s fridge.
- Fridge:
  - The primary key is the fridge’s id, which is an automatically generated integer.
- FoodItem:
  - The primary key is a UUID which is an automatically generated hex string guaranteed to be unique.
  - A charfield to represent the items food category as a sequence of two characters:
    - ("FR", "Fruits"),
    - ("VG", "Vegetables"),
    - ("GR", "Grains"),
    - ("PR", "Proteins"),
    - ("DR", "Dairy"),
    - ("OT", "Other")
  - Another charfield for the name of the food item, capped to 64 characters

- A Positive Small Integer field for the units of the item, the units are represented as:
    - TEASPOON = 1
    - TABLESPOON = 2
    - FLUID_OZ = 3
    - CUP = 4
    - PINT = 5
    - QUART = 6
    - GALLON = 7
    - LB = 8
    - OUNCE = 9
    - EACH = 10

- FoodInstance:
    - The primary key is the instance id, which is an automatically generated integer.
    - A Foreign key is the fridge id.
    - Another Foreign key is the item id.
    - There is a date field for the date the food was added, and another for the date the food is set to expire.
- Alerts:
    - The primary key is the alert id, which is an automatically generated integer.
    - A Foreign key is the fridge id.
    - Another Foreign key is the item id.

- There is an integer field for the level to alert the user if the total non-expired food instances are less than the alert level.

- RecipeCategory:
  - The primary key is the id, which is an automatically generated integer.
  - A charfield for the name of the category.
  - A textfield for the description of the category.
- Recipe:
  - The primary key is the id, which is an automatically generated UUID like the food item’s id.
  - A many to many relation matching with the id’s of the recipe categories.
  - A charfield with the name of the recipe.
  - A textfield with the description of the recipe.
  - A boolean field setting whether the recipe is private to the user.
  - A Foreign key with the id of the user the recipe belongs to.
- RecipeStep:
  - The primary key is the id, which is an automatically generated integer.
  - A charfield with the name of the step.
  - A textfield with the description of the step.
  - A foreign key with the id of the recipe the step belongs to.
  - A small positive integer field with the step number.
- RecipeIngredient:
  - The primary key is the id, which is an automatically generated integer.
  - A foreign key with the id of the item the ingredient is.

  - A foreign key with the id of the recipe the ingredient belongs to.
  - A Positive Small Integer field for the units of the item, which are the same as the food item units defined above.
- Rating:
  - The primary key is the id, which is an automatically generated integer.
  - A foreign key with the id of the recipe the rating belongs to.
  - A Foreign key with the id of the user the rating belongs to.

Views/Templates

CSS

- The CSS is fairly basic and has been applied to only the “fridge” part. It also is only somewhat responsive, as it is still very janky to view the website on a phone. No external libraries are used.

JS

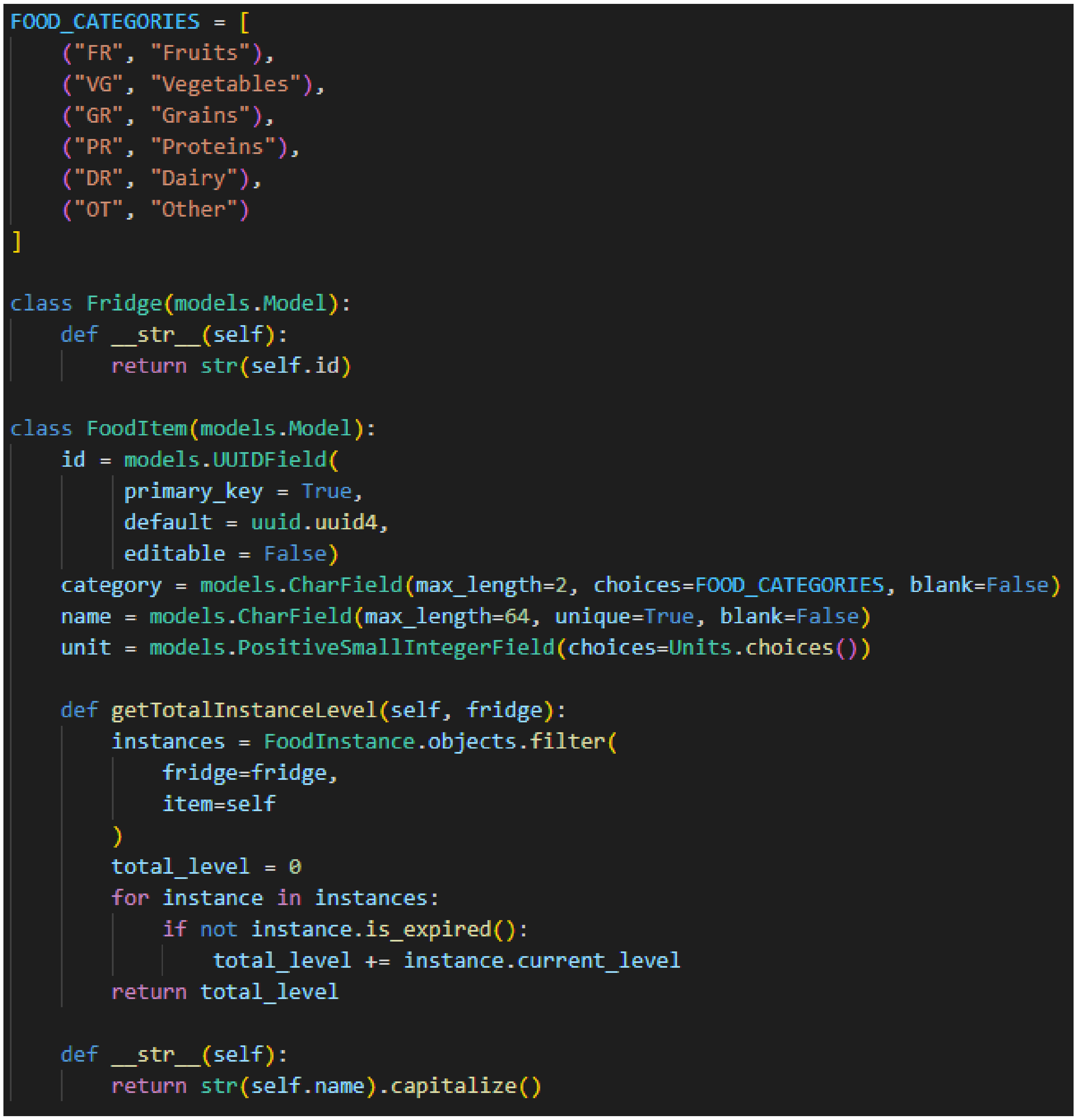
- The only javascript in this project is also used in the “fridge” part and is used to change which category of food is shown or switching between the combined and separate views.

Django Templates

- Every page is saved as a template in Django which allows for dynamic data to be displayed and allows for some inheritance. For instance the navbar of links is always displayed because it is in the base template which every other template incorporates.

Sample Code

Fridge Models:

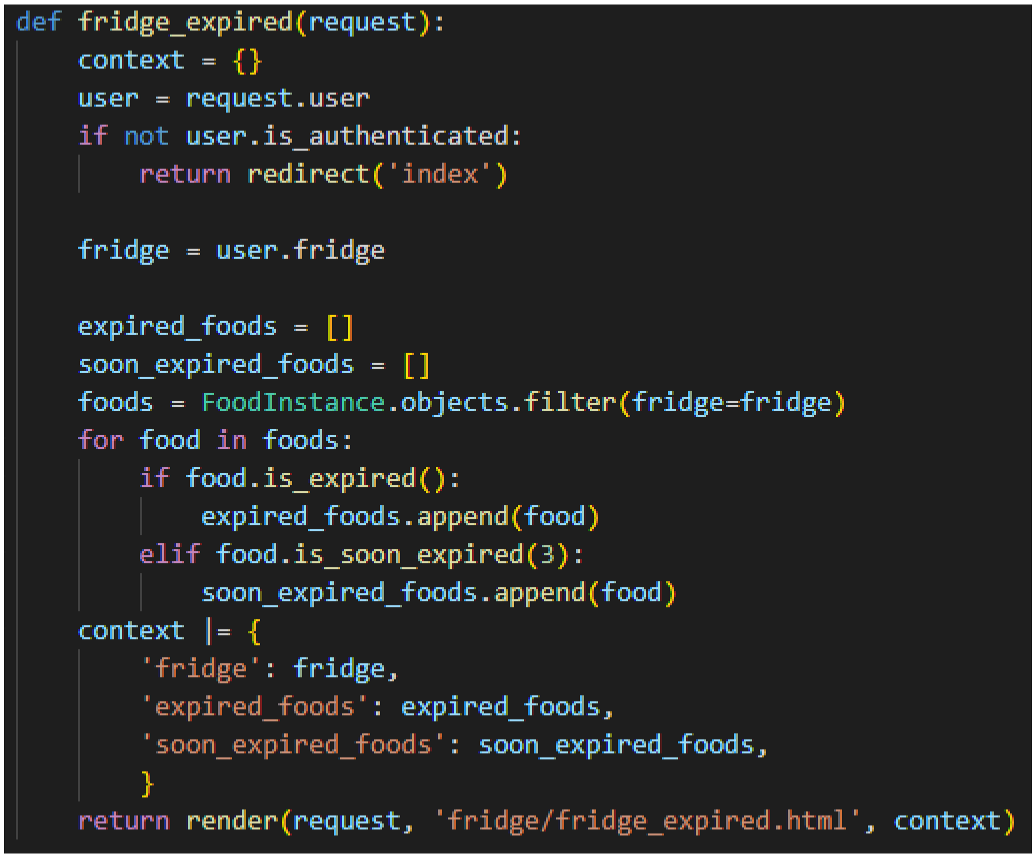


```
FOOD_CATEGORIES = [
    ("FR", "Fruits"),
    ("VG", "Vegetables"),
    ("GR", "Grains"),
    ("PR", "Proteins"),
    ("DR", "Dairy"),
    ("OT", "Other")
]

class Fridge(models.Model):
    def __str__(self):
        return str(self.id)

class FoodItem(models.Model):
    id = models.UUIDField(
        primary_key = True,
        default = uuid.uuid4,
        editable = False)
    category = models.CharField(max_length=2, choices=FOOD_CATEGORIES, blank=False)
    name = models.CharField(max_length=64, unique=True, blank=False)
    unit = models.PositiveSmallIntegerField(choices=Units.choices())

    def getTotalInstanceLevel(self, fridge):
        instances = FoodInstance.objects.filter(
            fridge=fridge,
            item=self
        )
        total_level = 0
        for instance in instances:
            if not instance.is_expired():
                total_level += instance.current_level
        return total_level

    def __str__(self):
        return str(self.name).capitalize()
```

Figure 2

Django translates these classes and their attributes into the corresponding database tables. For example the Fridge class has its own table in the database that holds the information for the fridge, in this case it's "empty" but Django autocreates an id for it. Django also allows methods to be defined that can be used on the objects retrieved from the database (which Django also handles).

Fridge Expired View



```python
def fridge_expired(request):
    context = {}
    user = request.user
    if not user.is_authenticated:
        return redirect('index')

    fridge = user.fridge

    expired_foods = []
    soon_expired_foods = []
    foods = FoodInstance.objects.filter(fridge=fridge)
    for food in foods:
        if food.is_expired():
            expired_foods.append(food)
        elif food.is_soon_expired(3):
            soon_expired_foods.append(food)
    context |= {
        'fridge': fridge,
        'expired_foods': expired_foods,
        'soon_expired_foods': soon_expired_foods,
        }
    return render(request, 'fridge/fridge_expired.html', context)
```

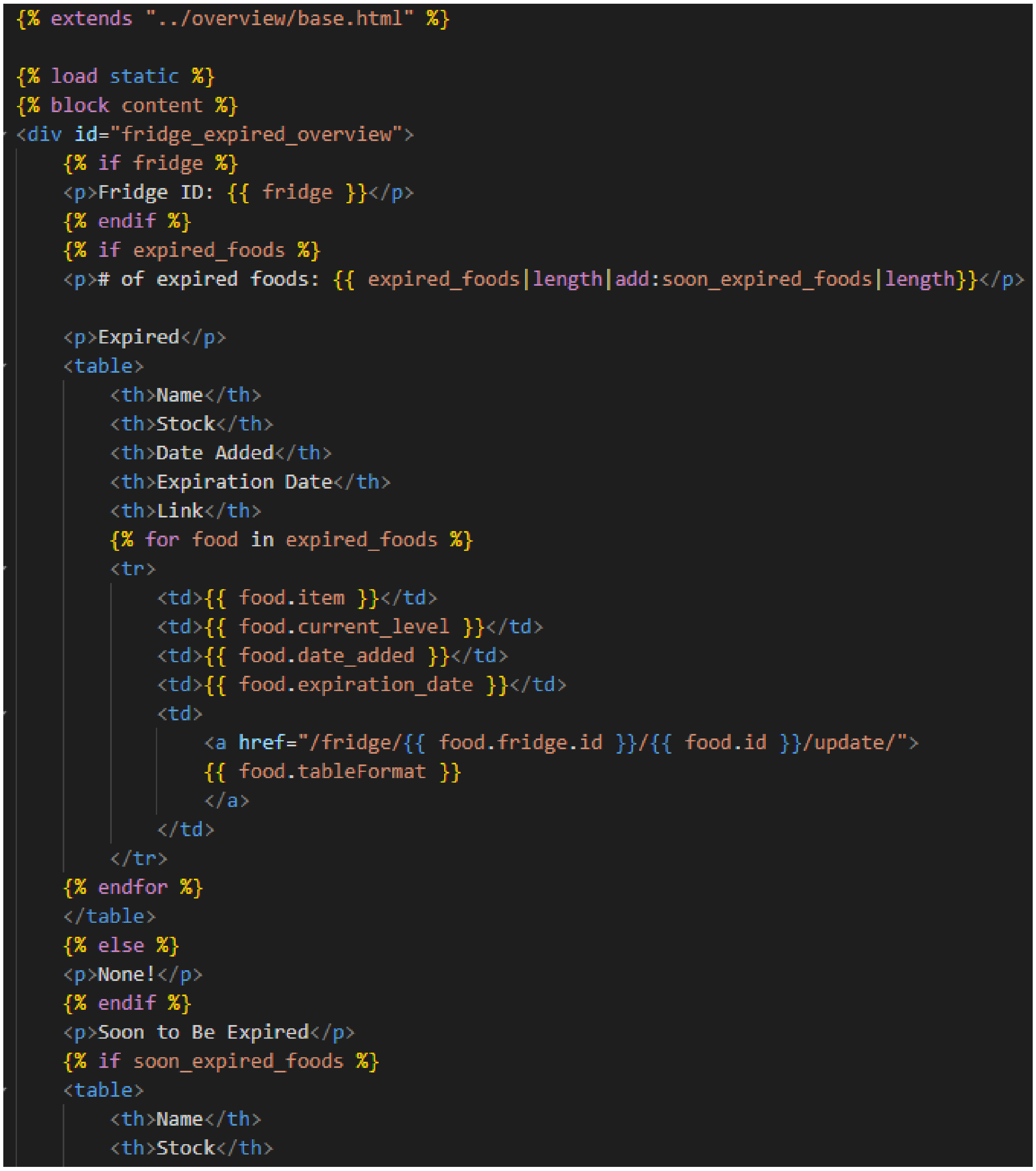
Figure 3

Django represents each view as a function that gets called when the page is loaded. This particular view:

- Checks whether the user is logged in

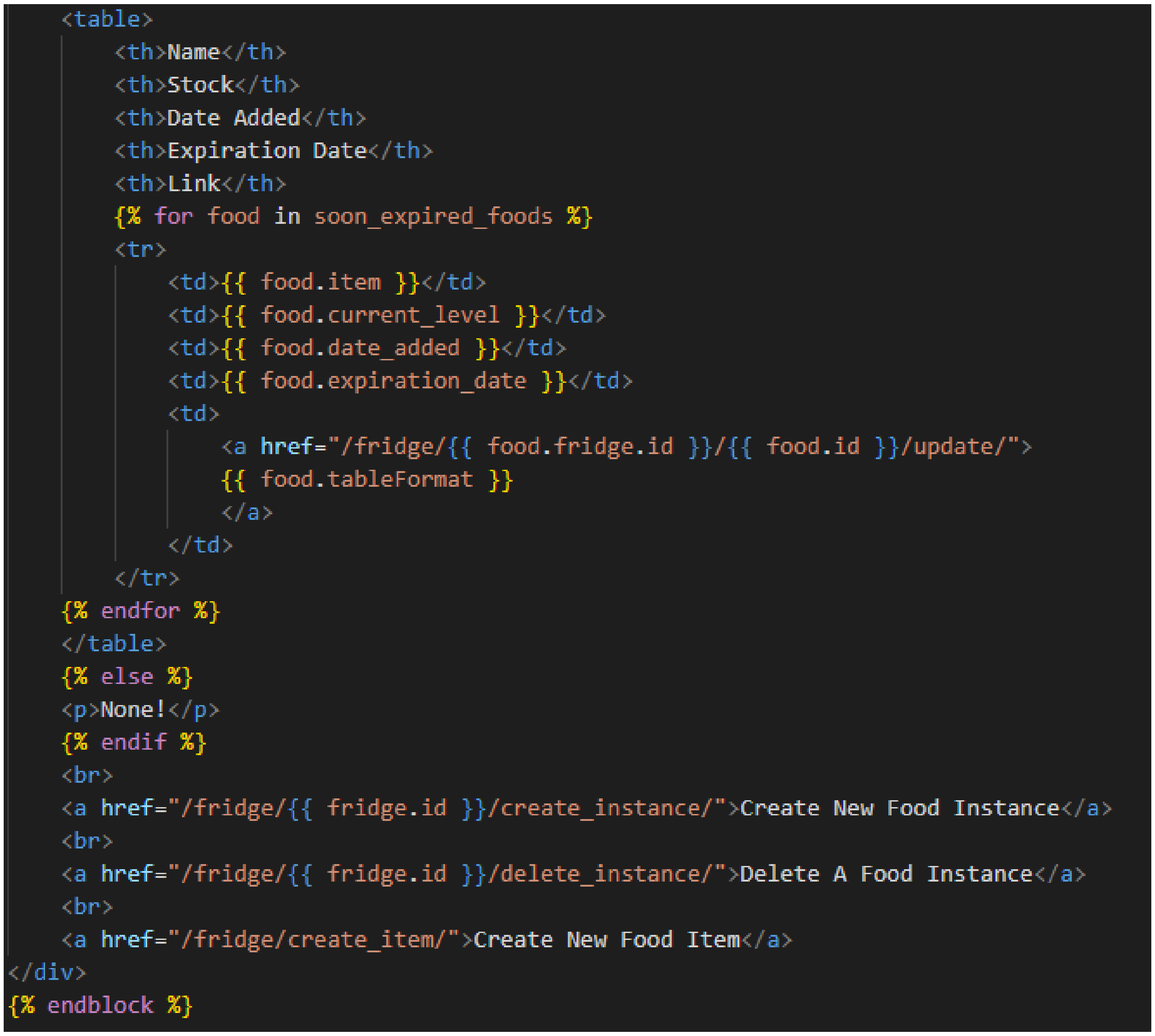
- Gets all the food instances from the user’s fridge
- Checks each instance to see if its expired or if it will expire soon
- Passes the expired and soon to be expired foods to the template to display to the user

Fridge Expired Template



```
{% extends "../overview/base.html" %}

{% load static %}
{% block content %}
<div id="fridge_expired_overview">
    {% if fridge %}
    <p>Fridge ID: {{ fridge }}</p>
    {% endif %}
    {% if expired_foods %}
    <p># of expired foods: {{ expired_foods|length|add:soon_expired_foods|length}}</p>

    <p>Expired</p>
    <table>
        <th>Name</th>
        <th>Stock</th>
        <th>Date Added</th>
        <th>Expiration Date</th>
        <th>Link</th>
        {% for food in expired_foods %}
        <tr>
            <td>{{ food.item }}</td>
            <td>{{ food.current_level }}</td>
            <td>{{ food.date_added }}</td>
            <td>{{ food.expiration_date }}</td>
            <td>
                <a href="/fridge/{{ food.fridge.id }}/{{ food.id }}/update/">
                {{ food.tableFormat }}
                </a>
            </td>
        </tr>
    {% endfor %}
    </table>
    {% else %}
    <p>None!</p>
    {% endif %}
    <p>Soon to Be Expired</p>
    {% if soon_expired_foods %}
    <table>
        <th>Name</th>
        <th>Stock</th>
```

Figure 4

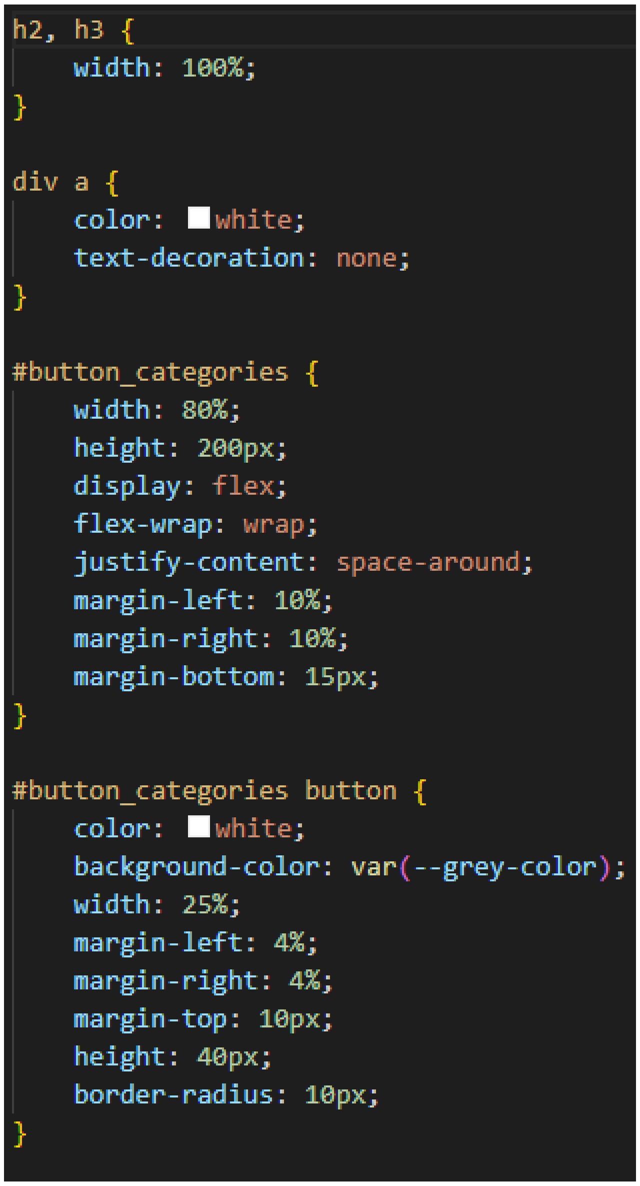


```
    <table>
        <th>Name</th>
        <th>Stock</th>
        <th>Date Added</th>
        <th>Expiration Date</th>
        <th>Link</th>
        {% for food in soon_expired_foods %}
        <tr>
            <td>{{ food.item }}</td>
            <td>{{ food.current_level }}</td>
            <td>{{ food.date_added }}</td>
            <td>{{ food.expiration_date }}</td>
            <td>
                <a href="/fridge/{{ food.fridge.id }}/{{ food.id }}/update/">
                {{ food.tableFormat }}
                </a>
            </td>
        </tr>
    {% endfor %}
    </table>
    {% else %}
    <p>None!</p>
    {% endif %}
    <br>
    <a href="/fridge/{{ fridge.id }}/create_instance/">Create New Food Instance</a>
    <br>
    <a href="/fridge/{{ fridge.id }}/delete_instance/">Delete A Food Instance</a>
    <br>
    <a href="/fridge/create_item/">Create New Food Item</a>
</div>
{% endblock %}
```

Figure 5

The template is able to be pre-loaded with variables in order to provide a more dynamic page. This template loads the base template, which has the navbar, and a table showing the user the expired and soon to be expired foods, or none if there are none. After that it gives a few helpful links to the CRUD (Create, Read, Update, Delete) of food instances.

Fridge CSS



```css
h2, h3 {
    width: 100%;
}

div a {
    color: white;
    text-decoration: none;
}

#button_categories {
    width: 80%;
    height: 200px;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    margin-left: 10%;
    margin-right: 10%;
    margin-bottom: 15px;
}

#button_categories button {
    color: white;
    background-color: var(--grey-color);
    width: 25%;
    margin-left: 4%;
    margin-right: 4%;
    margin-top: 10px;
    height: 40px;
    border-radius: 10px;
}
```

Figure 6

The CSS is standard CSS. A majority of this snippet doesn't correspond to the earlier examples because this css file is shared between all of the fridge specific templates.

Results

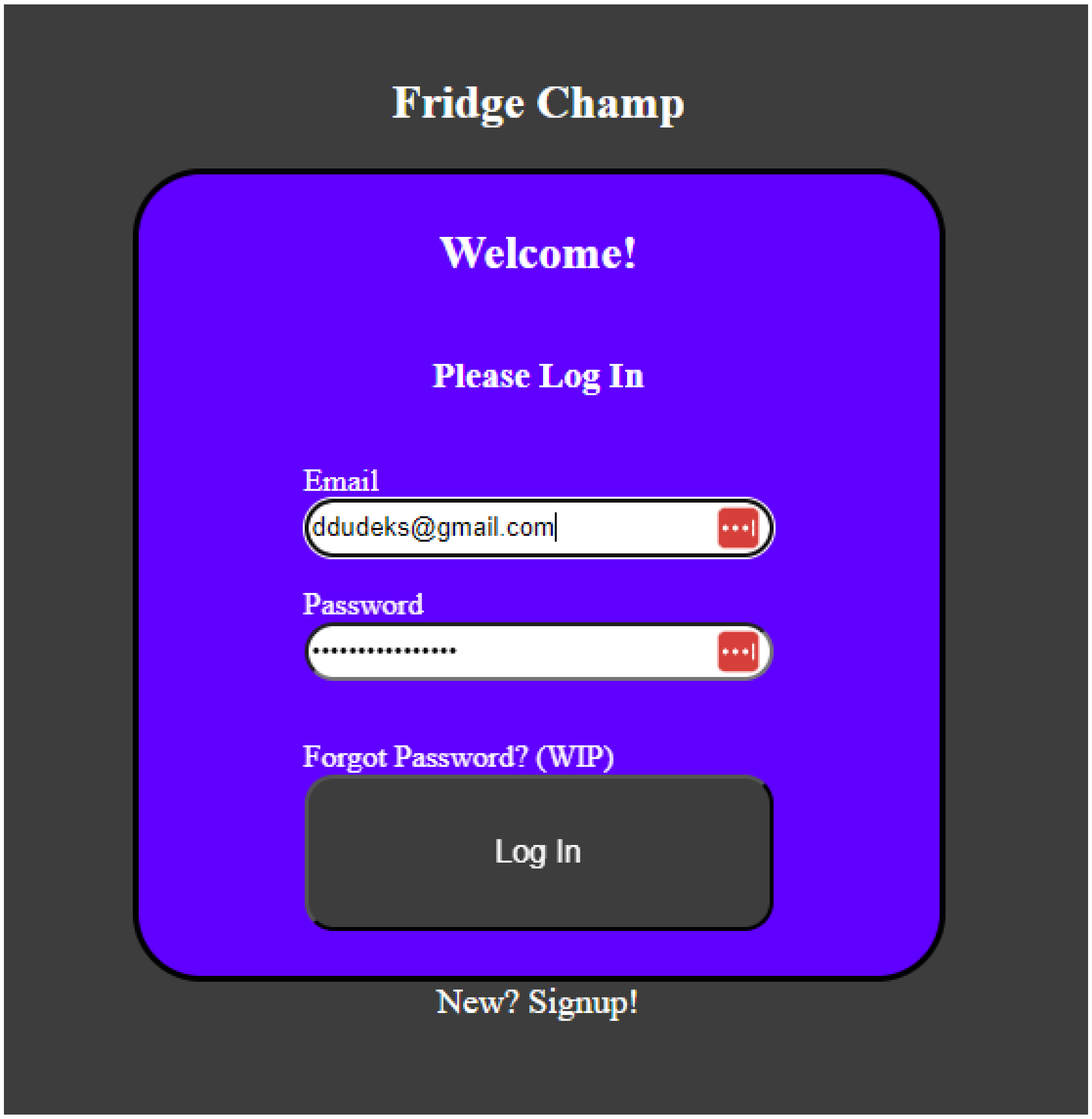
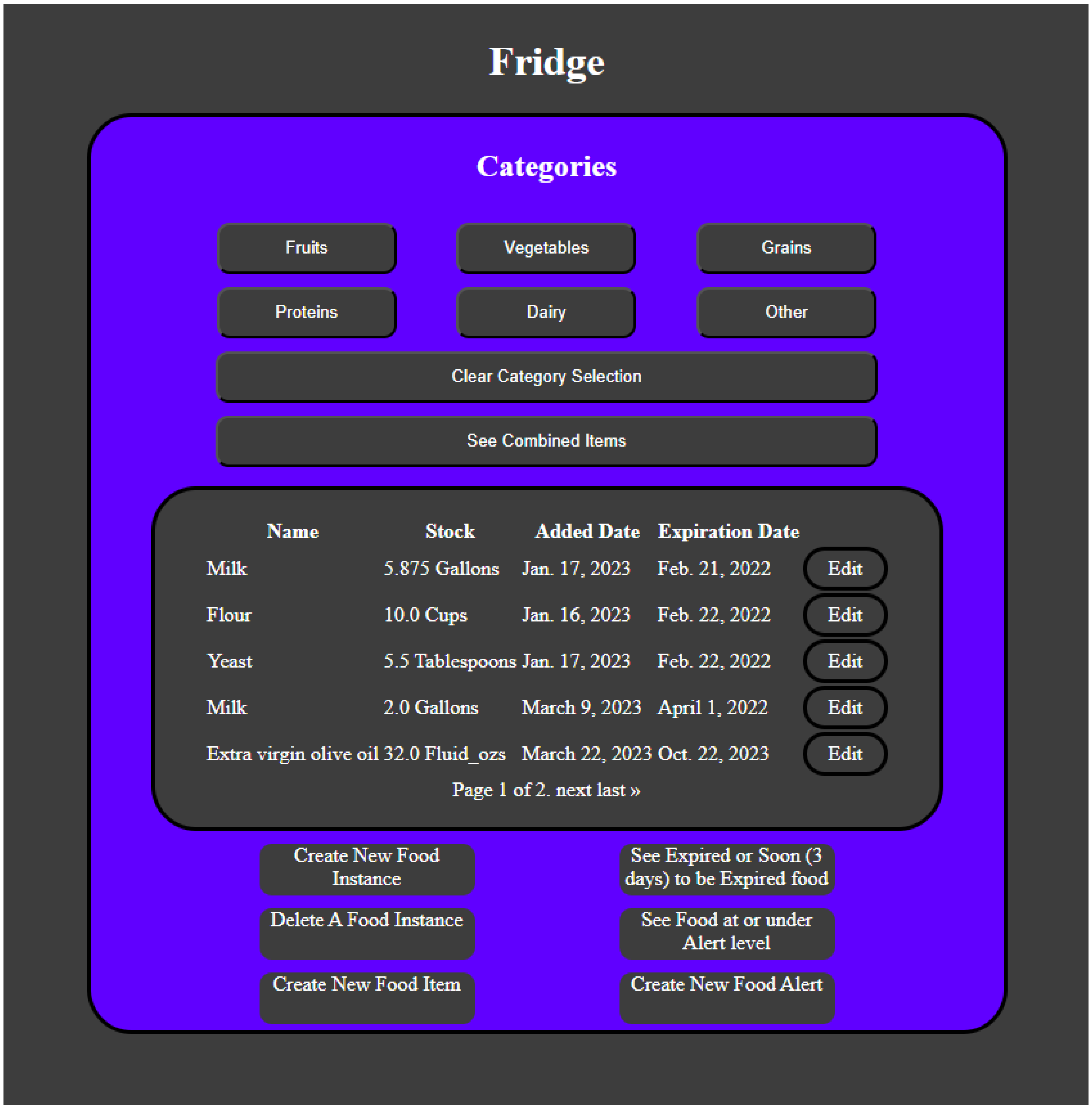


Figure 7

The Login Screen



Fridge

Categories

Name	Stock	Added Date	Expiration Date	
Milk	5.875 Gallons	Jan. 17, 2023	Feb. 21, 2022	Edit
Flour	10.0 Cups	Jan. 16, 2023	Feb. 22, 2022	Edit
Yeast	5.5 Tablespoons	Jan. 17, 2023	Feb. 22, 2022	Edit
Milk	2.0 Gallons	March 9, 2023	April 1, 2022	Edit
Extra virgin olive oil	32.0 Fluid_ozs	March 22, 2023	Oct. 22, 2023	Edit

Page 1 of 2. next last »

Figure 8

The Fridge Overview screen

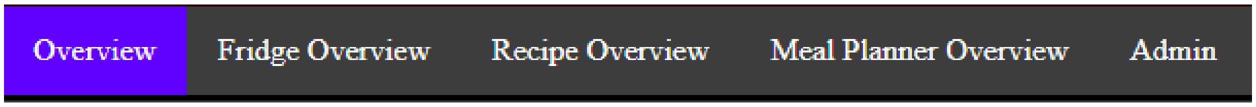


Figure 9

The Navbar between the main “modules”

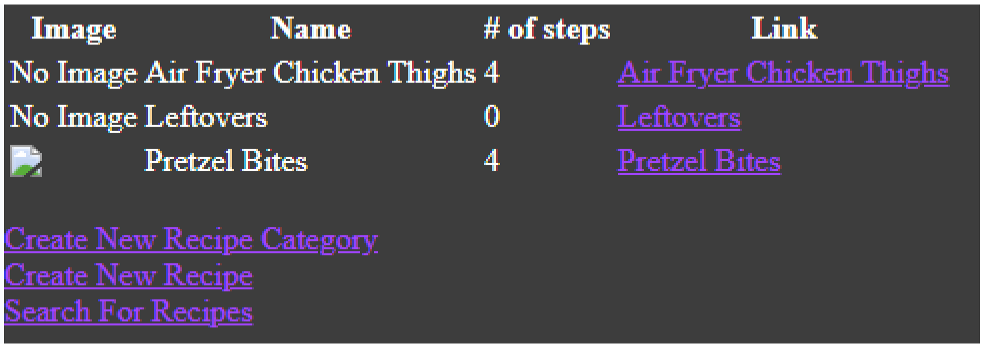
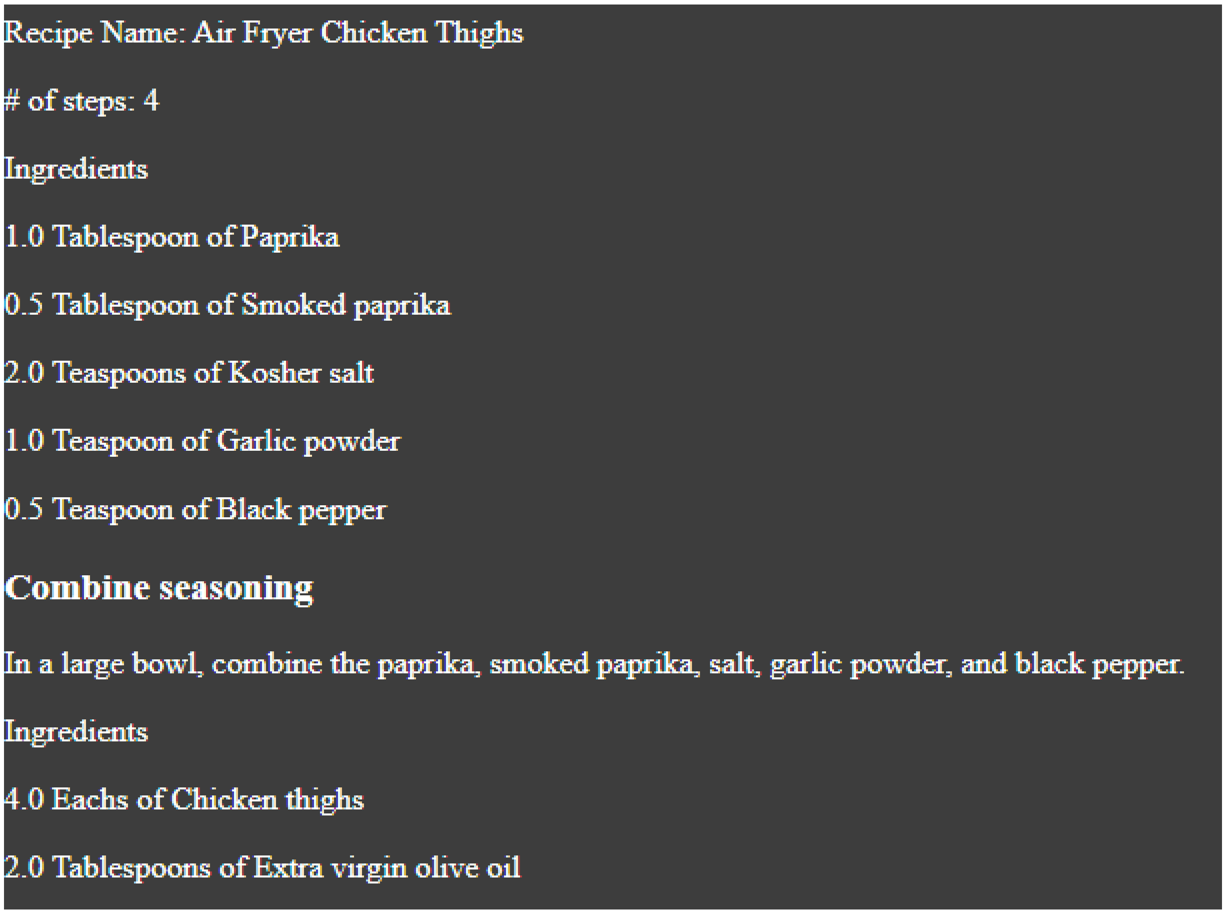


Image	Name	# of steps	Link
No Image	Air Fryer Chicken Thighs	4	Air Fryer Chicken Thighs
No Image	Leftovers	0	Leftovers
	Pretzel Bites	4	Pretzel Bites

Create New Recipe Category
Create New Recipe
Search For Recipes

Figure 10

The current Recipe Overview page.



Recipe Name: Air Fryer Chicken Thighs

# of steps: 4

Ingredients

1.0 Tablespoon of Paprika

0.5 Tablespoon of Smoked paprika

2.0 Teaspoons of Kosher salt

1.0 Teaspoon of Garlic powder

0.5 Teaspoon of Black pepper

Combine seasoning

In a large bowl, combine the paprika, smoked paprika, salt, garlic powder, and black pepper.

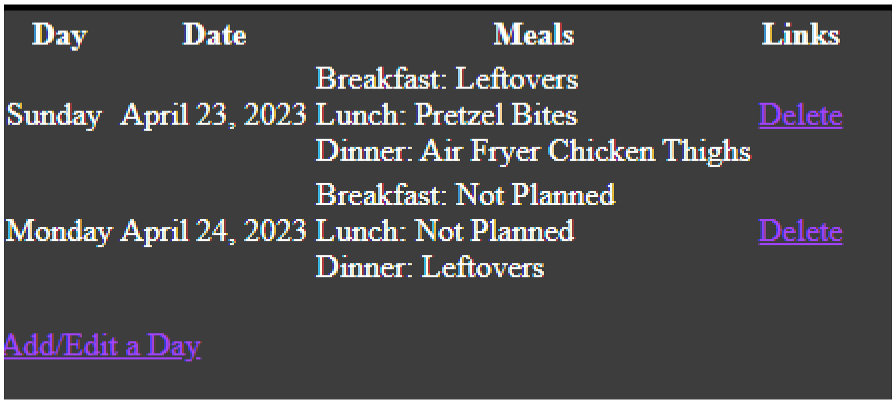
Ingredients

4.0 Eachs of Chicken thighs

2.0 Tablespoons of Extra virgin olive oil

Figure 11

An example of a recipe, currently just text. Trying to add images eventually.

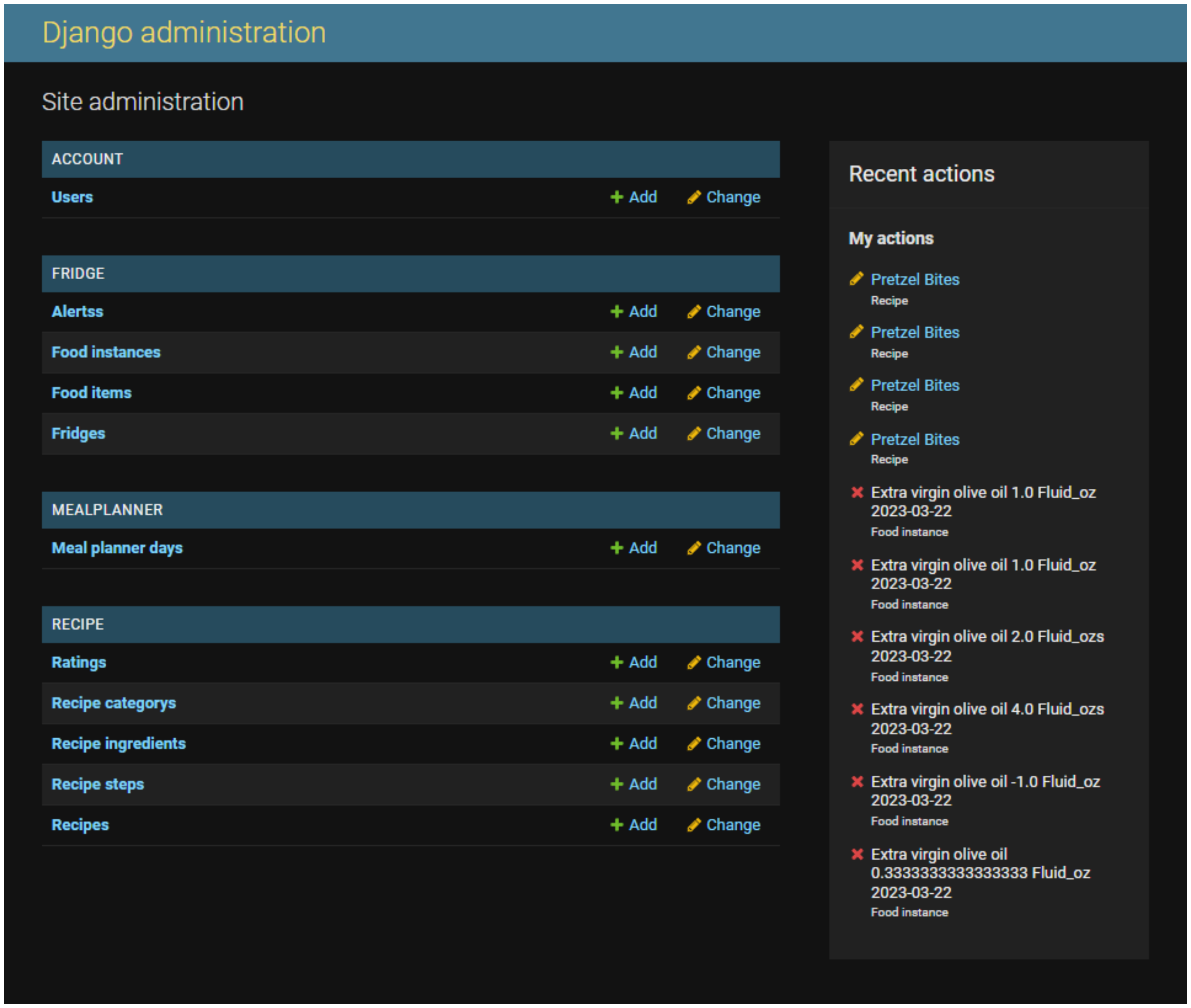


Day	Date	Meals	Links
Sunday	April 23, 2023	Breakfast: Leftovers Lunch: Pretzel Bites Dinner: Air Fryer Chicken Thighs	Delete
Monday	April 24, 2023	Breakfast: Not Planned Lunch: Not Planned Dinner: Leftovers	Delete

Add/Edit a Day

Figure 12

An example of the MealPlanner, allows users to plan breakfast, lunch, or dinner using the recipes in the database.



Django administration

Site administration

ACCOUNT		
Users	+ Add	Change

FRIDGE		
Alertss	+ Add	Change
Food instances	+ Add	Change
Food items	+ Add	Change
Fridges	+ Add	Change

MEALPLANNER		
Meal planner days	+ Add	Change

RECIPE		
Ratings	+ Add	Change
Recipe categorys	+ Add	Change
Recipe ingredients	+ Add	Change
Recipe steps	+ Add	Change
Recipes	+ Add	Change

Recent actions

My actions

- Pretzel Bites
  Recipe
- Pretzel Bites
  Recipe
- Pretzel Bites
  Recipe
- Pretzel Bites
  Recipe
- Extra virgin olive oil 1.0 Fluid_oz 2023-03-22
  Food instance
- Extra virgin olive oil 1.0 Fluid_oz 2023-03-22
  Food instance
- Extra virgin olive oil 2.0 Fluid_ozs 2023-03-22
  Food instance
- Extra virgin olive oil 4.0 Fluid_ozs 2023-03-22
  Food instance
- Extra virgin olive oil -1.0 Fluid_oz 2023-03-22
  Food instance
- Extra virgin olive oil 0.3333333333333333 Fluid_oz 2023-03-22
  Food instance

Figure 13

The built in Django admin page, very useful for manipulating the database objects without manually accessing the database.

Discussions and Conclusions

FridgeChamp is a web-based tool that enables home cooks to monitor their food inventory. Users can keep track of their recipes and choose whether to keep them private or share them publicly. The system also allows users to set expiration dates for their food items and receive notifications when items are about to expire. Moreover, users can set "alert" levels for each item to receive a warning when the stock goes below a certain threshold. Each user has their own account and fridge, and the planner feature enables users to plan their meals up to a year in advance, making it a useful tool for grocery shopping and managing home cooking. FridgeChamp also offers a recipe database where users can store all of their recipes in one convenient location without any excess information.

References

Django Software Foundation. (2022). Django. Retrieved from https://djangoproject.com

The Postgres95. User Manual. A. Yu and J. Chen. University of California. Berkeley, California. Sept. 5, 1995.

Enhancement of the ANSI SQL Implementation of PostgreSQL. Stefan Simkovics. Department of Information Systems, Vienna University of Technology. Vienna, Austria. November 29, 1998.

Python Software Foundation. Python Language Reference, version 3.10. Available at http://www.python.org