Fridge Management Automation Software

Implementation Document 07.09.2023

—

Name -	Github ID -	NET ID-
Kyaw Min Thu	kyawminthu20	gb6499
Nayan Pai	painayan18	bp3398
Sarthak Adhikari	adhikarisarthak	bv9292
Yichao Hao	alcibiades7	ur5215

Preface

The Implementation Document serves as a roadmap for the development team and stakeholders involved in the implementation of a software project. It outlines the project's objectives, scope, and key features to provide a clear vision. It details the chosen technologies, architectural design, and integration considerations to ensure a successful implementation. The document also covers project management approaches, task allocation, and risk management strategies for effective coordination and progress monitoring. By following this comprehensive document, stakeholders can collaborate efficiently, ensuring a smooth and successful implementation of the software solution.

Introduction

The Django architecture for the Fridge Management System (FMS) follows a Model-View-Template (MVT) pattern. Models represent the database structure for storing fridge inventory data. Views handle the logic for processing user requests and rendering responses. Templates provide the presentation layer, rendering HTML pages for user interaction. URL routing maps user requests to specific views. This architecture enables efficient organization, tracking, and monitoring of fridge inventory while promoting separation of concerns and maintainability.

Architectural Design Change

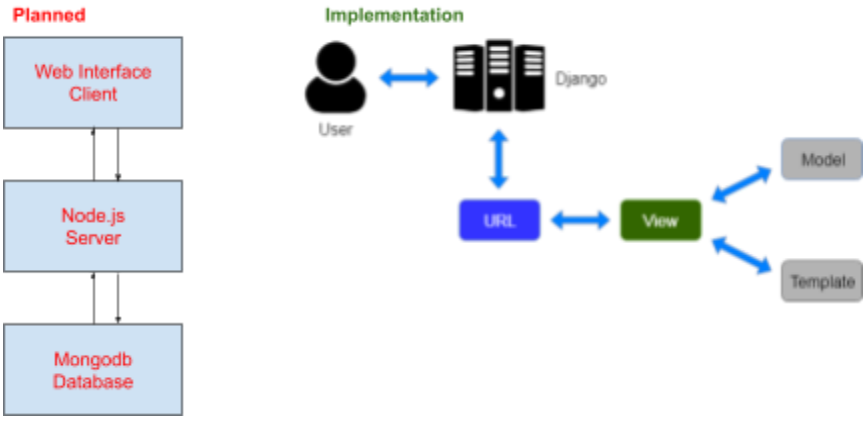
I. High-Level Architecture Change

A. The transition from a client-server architecture to the MVT (Model-View-Template) architecture in web application development offers several benefits. MVT promotes separation of concerns between the model, view, and template components, making the codebase more modular and easier to maintain. It enables reusability and modifiability of components, allowing for flexibility in development and reducing the impact of changes on other parts of the application. MVT architecture increases productivity by providing built-in tools and conventions, simplifies template rendering, and supports collaborative development by enabling teams to work on different components simultaneously. Overall, MVT architecture addresses the specific

needs of web development, promoting efficient and maintainable application development.

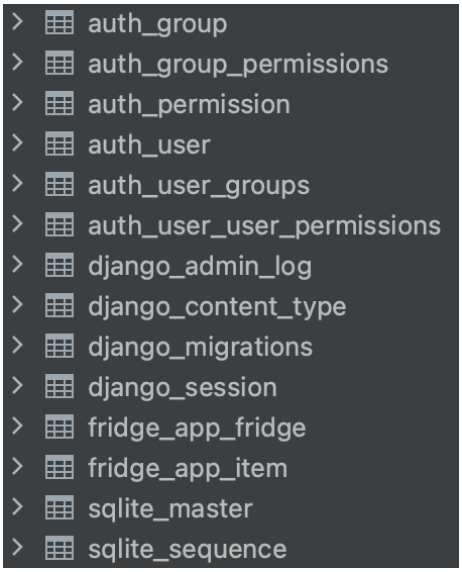
B. Client Server Architecture (Planned)
    1. User Interface - HTTP, CSS
    2. The controller in Server - Node.js
    3. Database - MongoDB

C. MVT Architecture (Implemented)
    1. Model: The model in MVT represents the data and business logic, similar to the model in MVC. It defines the structure of the data and handles database interactions.
    2. View: In MVT, the view is responsible for rendering the HTML templates and presenting data to the user. It handles how the data is displayed and interacts with the user interface.
    3. Template: The template component in MVT corresponds to the view in MVC. Templates define the structure and layout of the HTML pages. They contain placeholders for dynamic data that are filled in by the view.

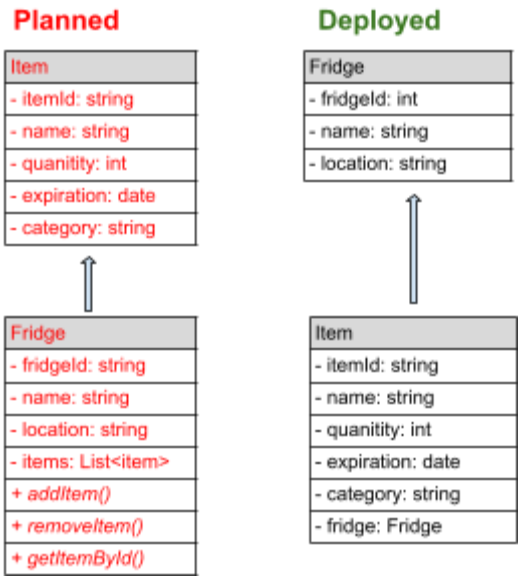


II. Detail Design Change

- Data Management
  - Separation of Data
    - Fridge and Item
  - Fridge model class:
    - It represents a fridge entity in the CRM system.
    - The Fridge class has attributes such as name (the name of the fridge), location (the location of the fridge), and user (the user who owns the fridge).
    - The user attribute is a foreign key field that establishes a relationship with the User model class. It means that each fridge belongs to a specific user.
    - When a user is deleted, the on_delete=models.CASCADE parameter ensures that all associated fridges will also be deleted.
  - Item model class:
    - It represents an item entity that can be stored in a fridge in the CRM system.
    - The Item class has attributes such as name (the name of the item), category (the category of the item), qty (the quantity of the item), fridge (the fridge in which the item is stored), and expiry_date (the expiration date of the item).
    - The fridge attribute is a foreign key field that establishes a relationship with the Fridge model class. It means that each item belongs to a specific fridge.
    - The expiry_date attribute has a default value of the current date and time, provided by the timezone.now function.
    - The get_absolute_url method returns the URL to the detail view of an item, which can be useful for redirecting users after creating or updating an item.

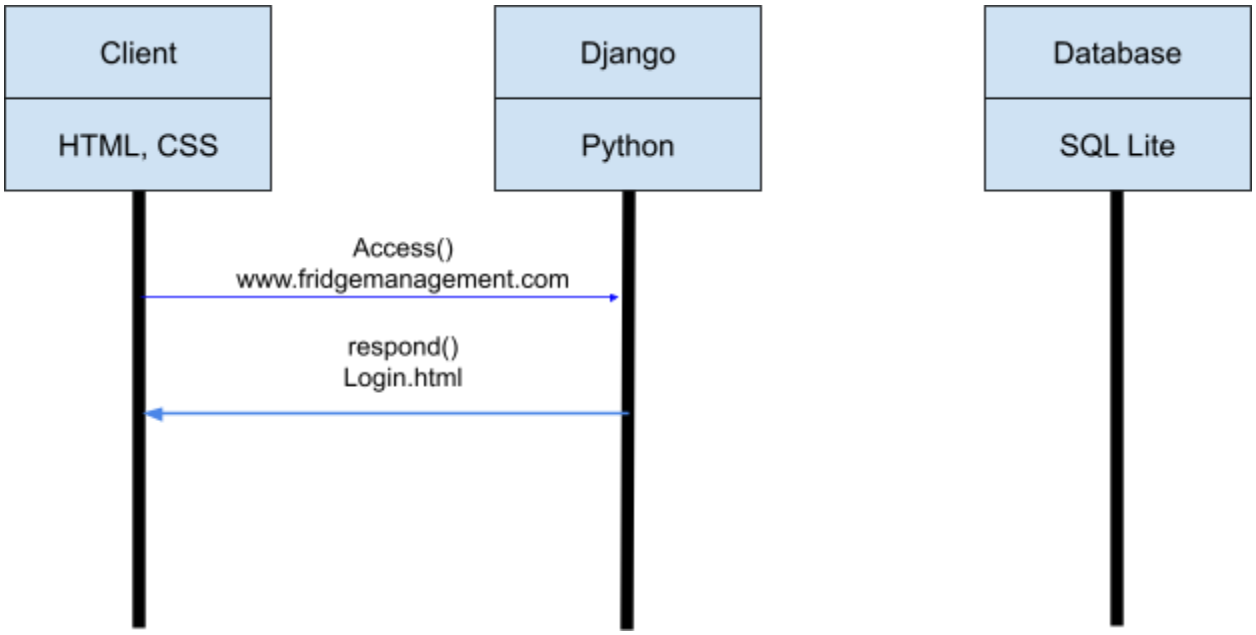
- Database:
  - SQL lite database is used as a development server
  - Tables are created with Django manage.py migration functions.
  - 

Class Diagram (a class diagram including class names, attributes, methods, and relationships with other classes, e.g., Generalization)

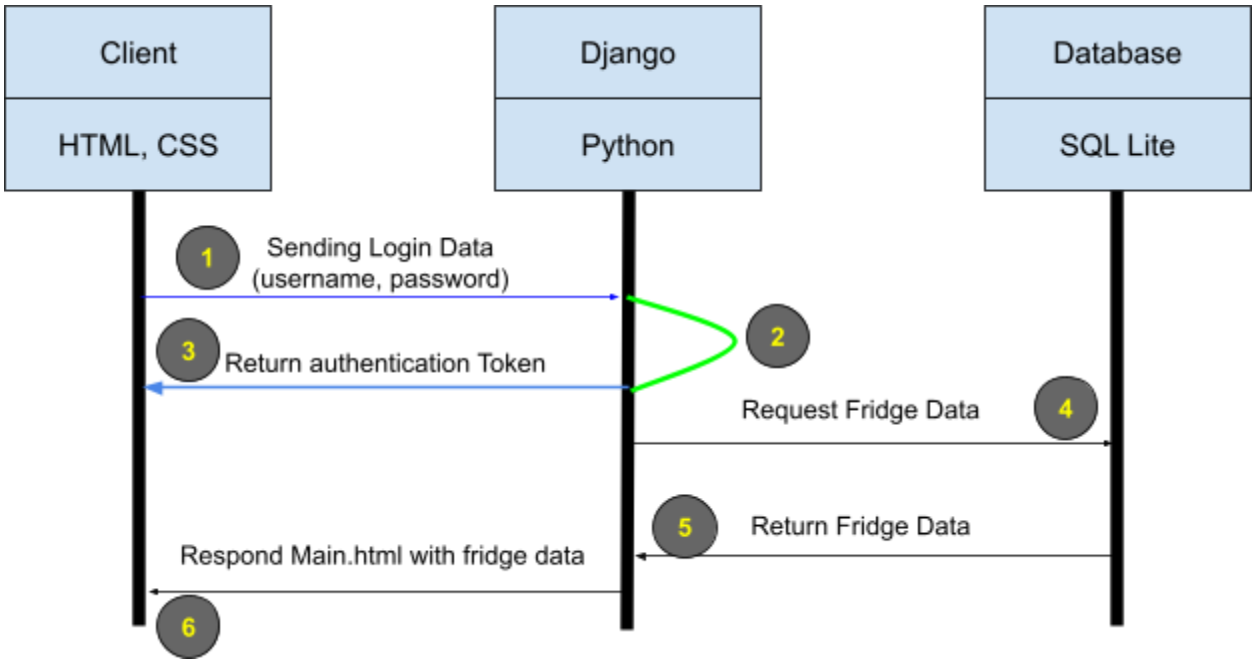


III. A set of sequence diagrams

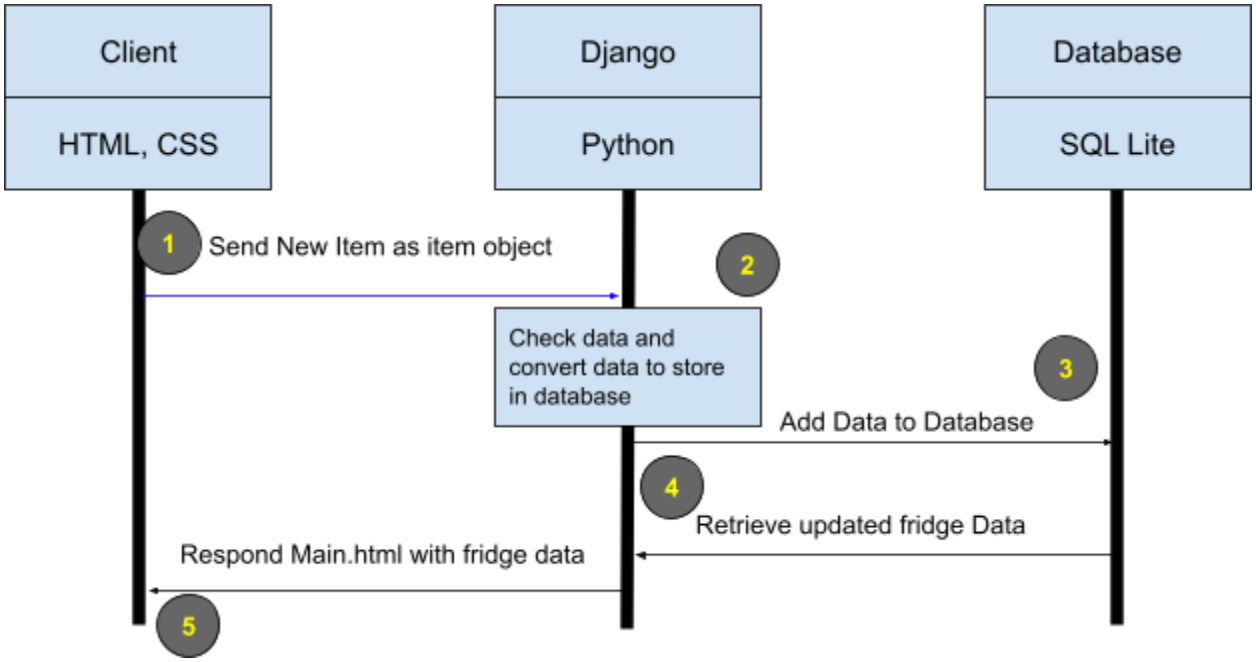
1. Accessing Main Page



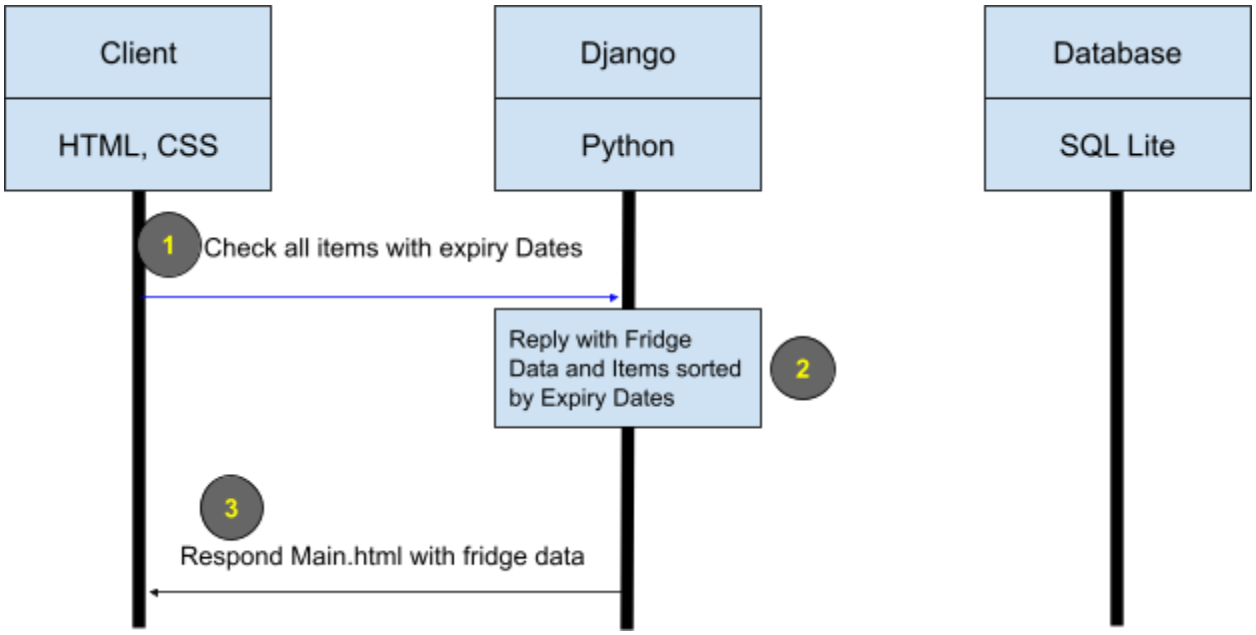
2. Logging In



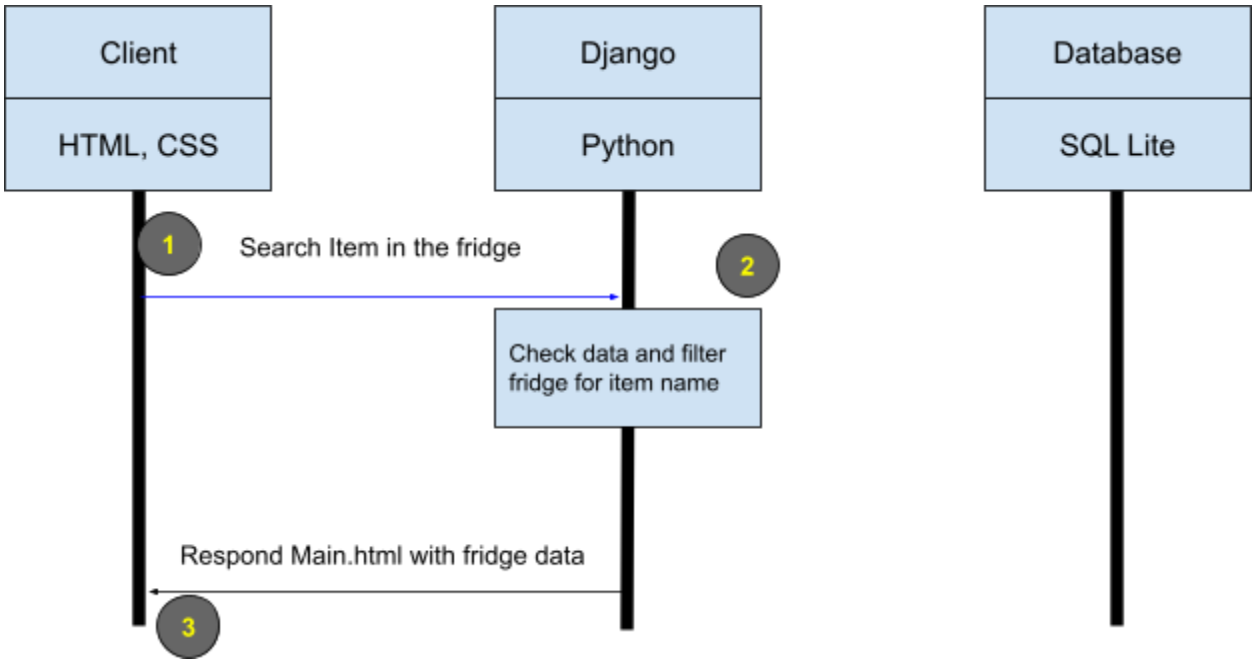
3. Adding Items



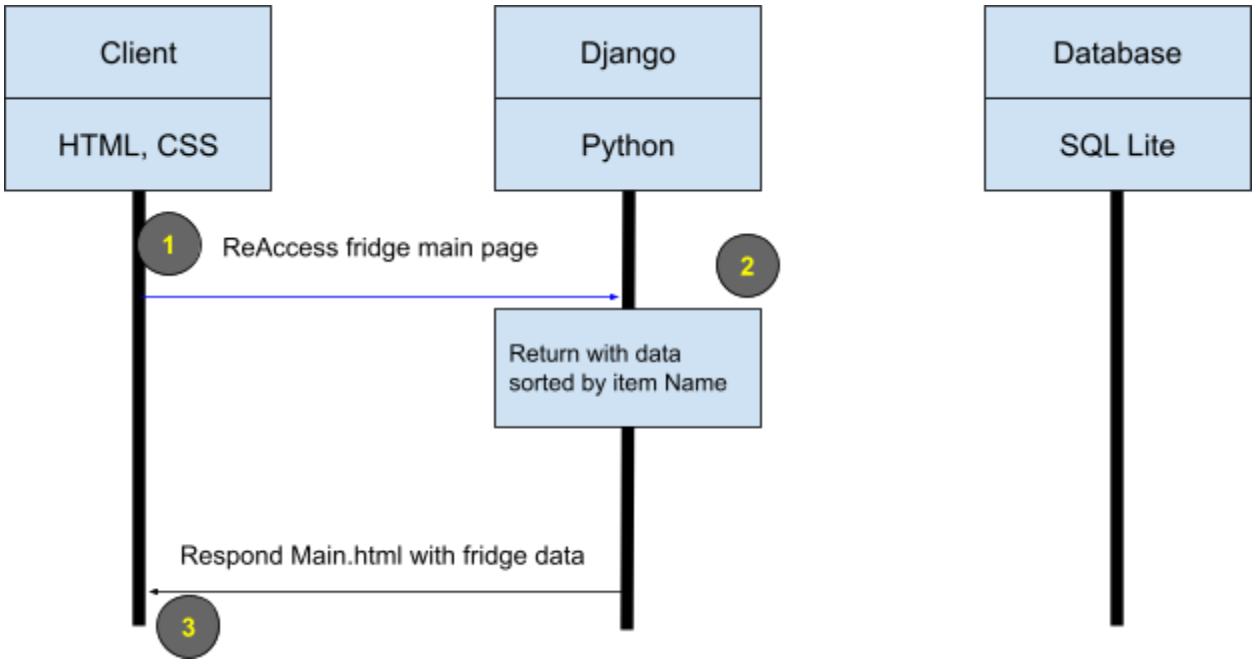
4. Checking Expiry Dates



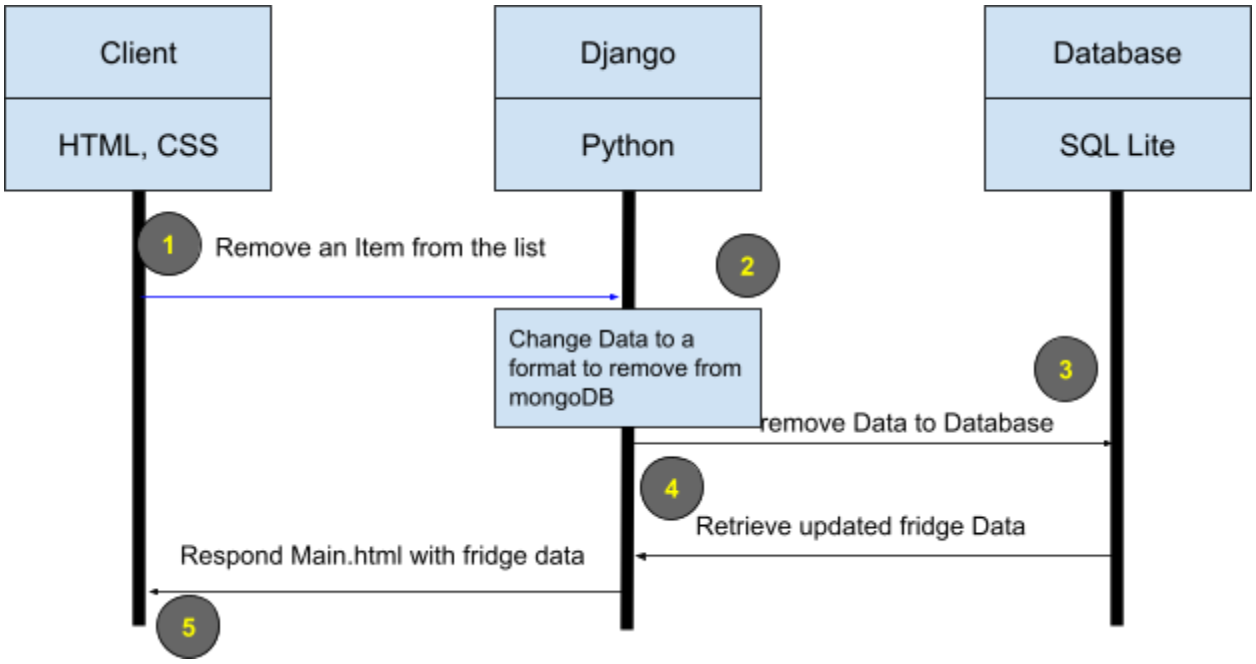
5. Searching Item



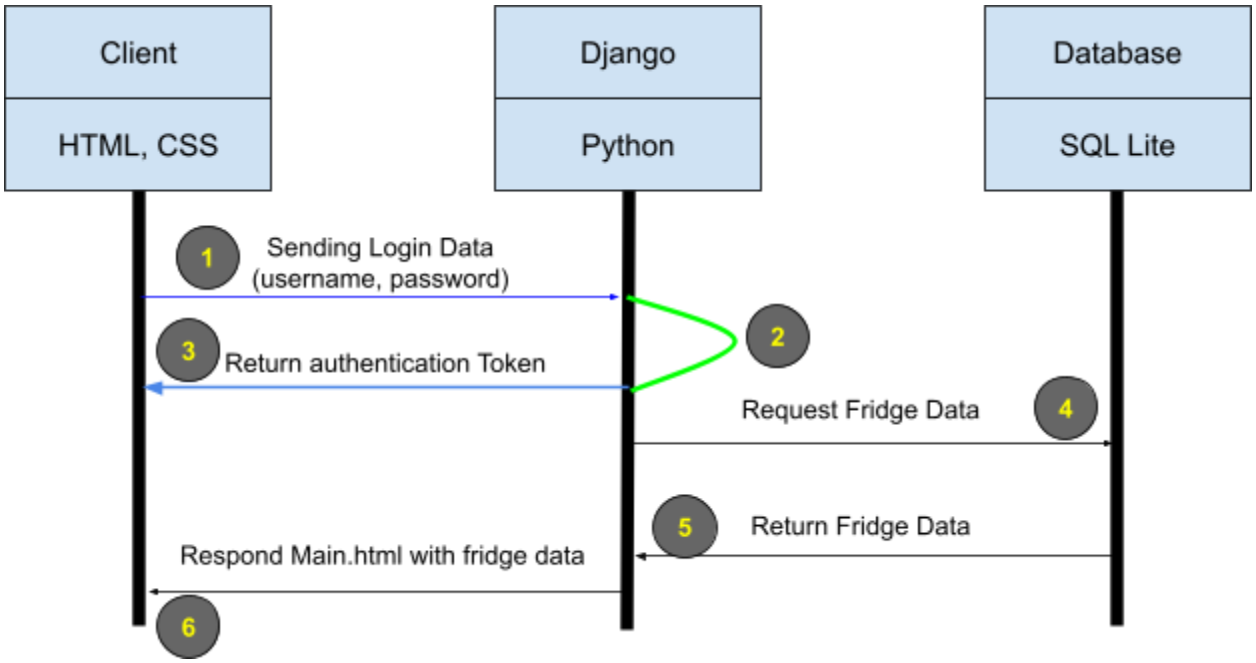
6. Monitoring Quantities



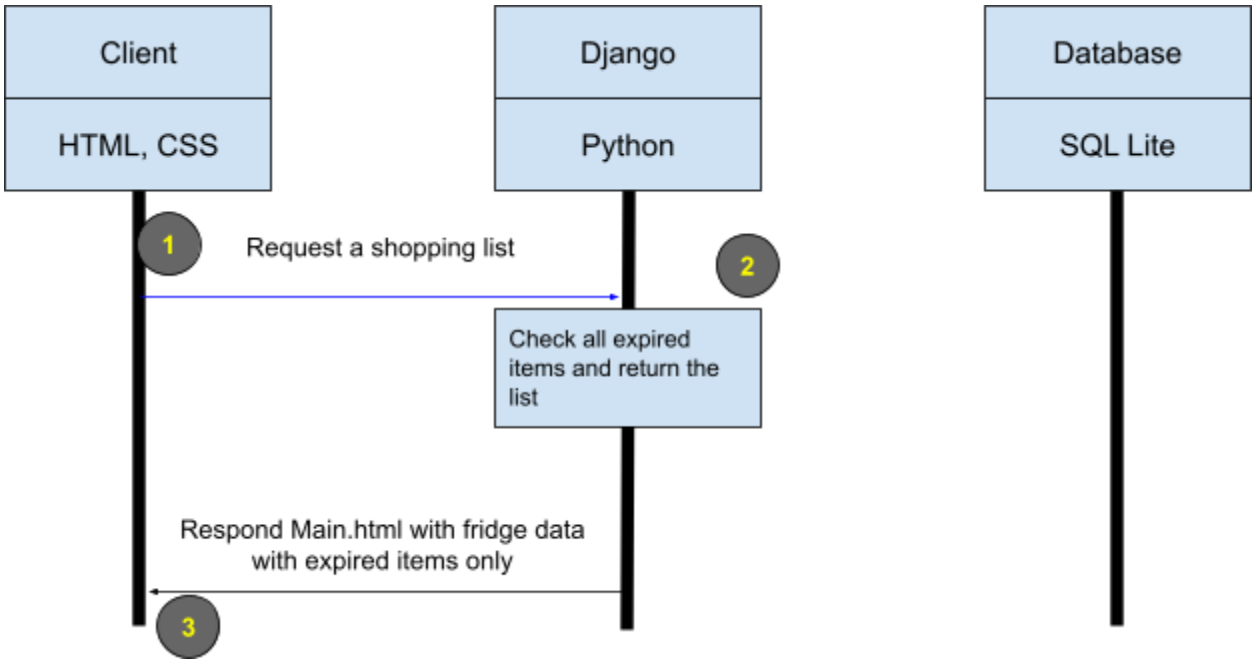
7. Removing an Expired Item - to add in sprint



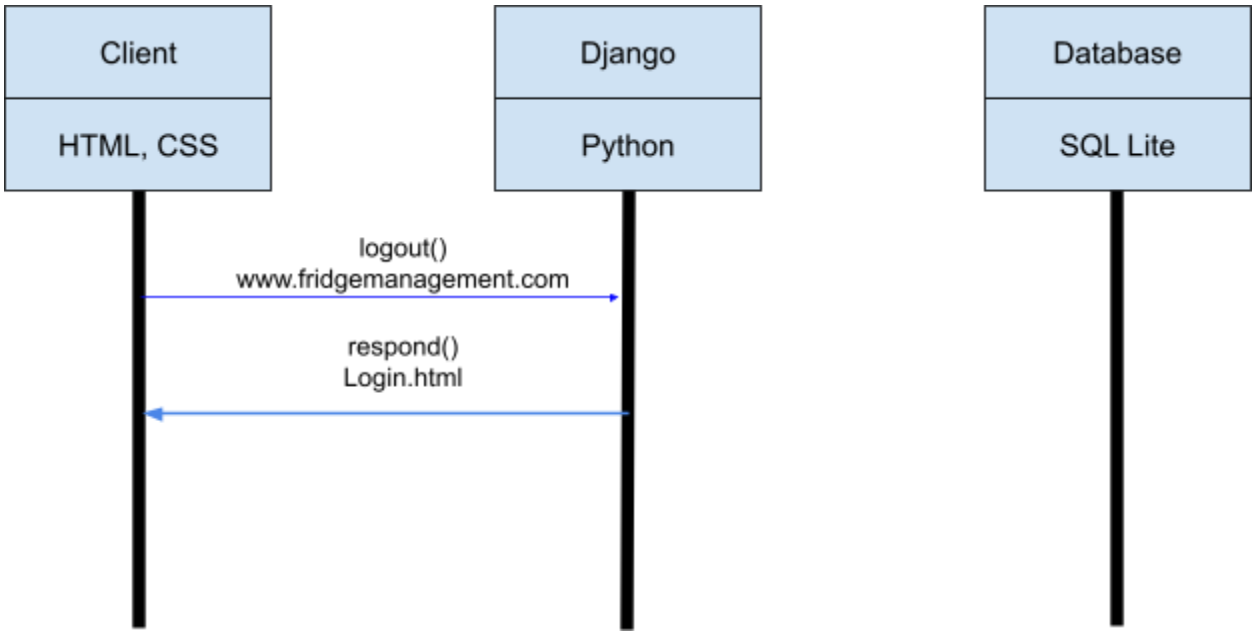
8. Accessing inventory on Multiple Devices



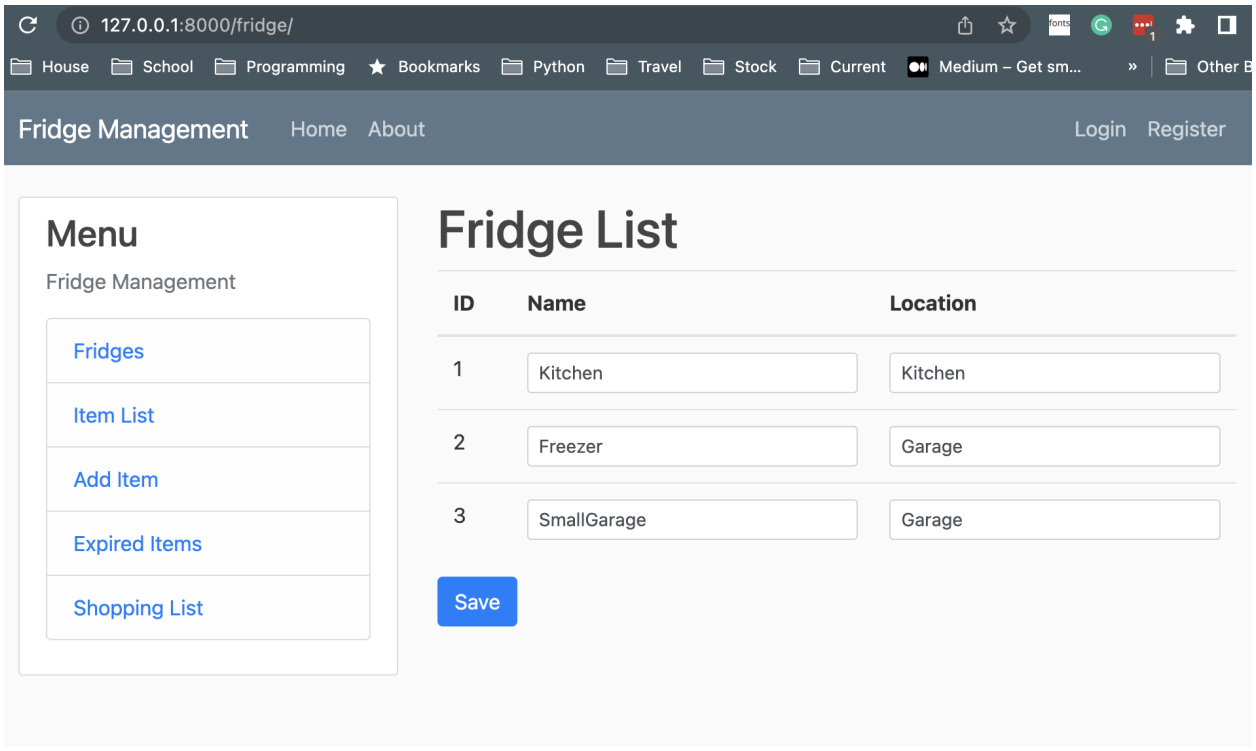
9. Generate a shopping list.

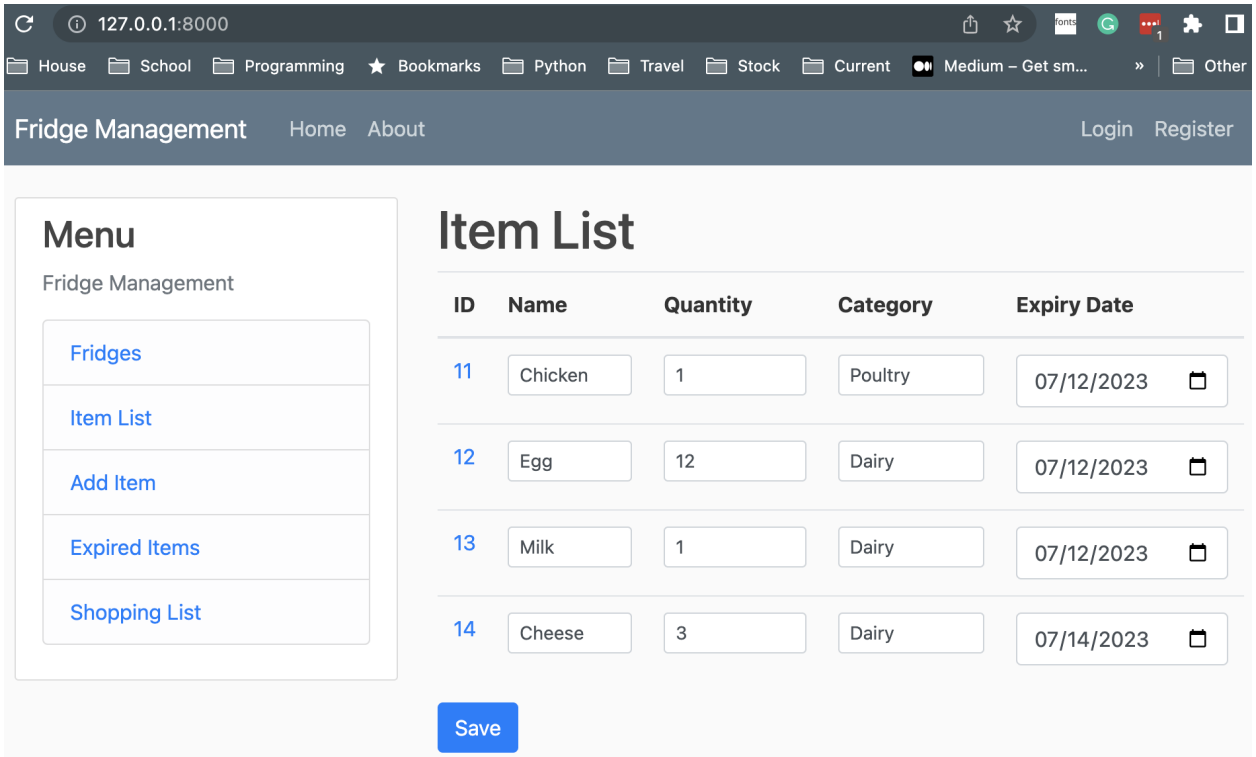


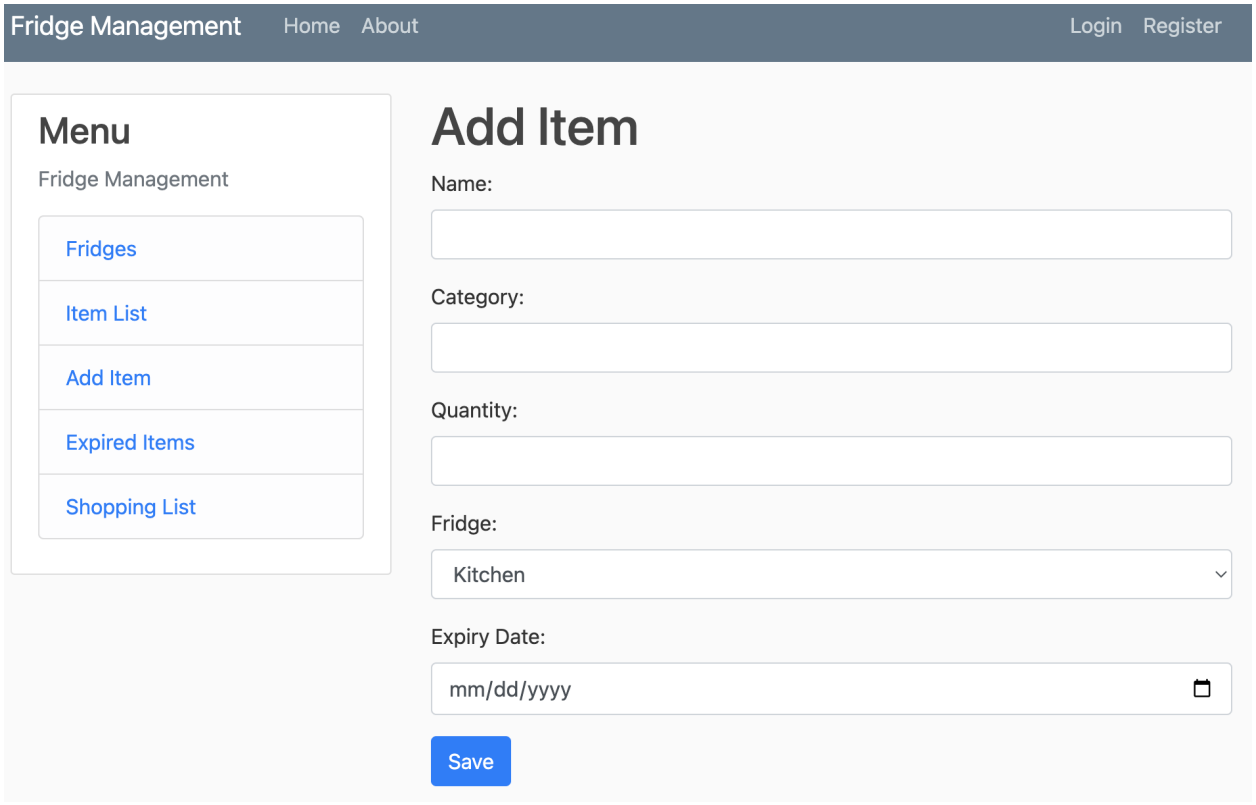
10. Logging Out



User Interface Design (Bootstrap)







Deployment and Infrastructure

- Deployment is performed by integrating GitHub Actions with AWS services to create a CI/CD pipeline for deploying applications.

  The process involves the following steps:

  a. Clone the project from the AWS code samples repository.

  b. Deploy an AWS CloudFormation template to set up the required services.

  c. Update the source code of the application.

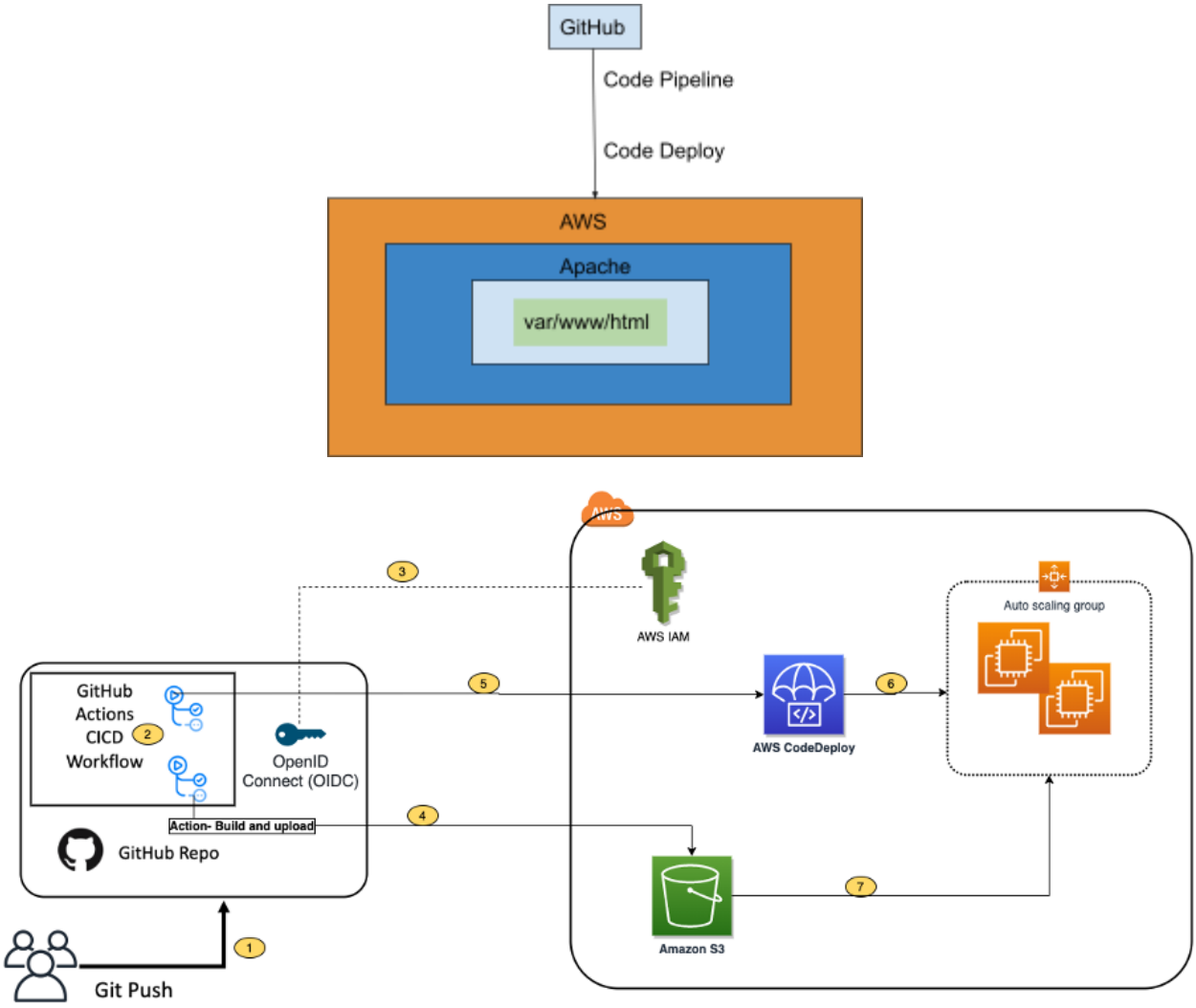
  d. Configure GitHub secrets to securely store sensitive information.

  e. Integrate CodeDeploy with GitHub to enable deployment automation.

  f. Trigger the GitHub Action to build and deploy the code.

g. Verify the deployment to ensure the application is successfully deployed to the Amazon EC2 instances.

By following these steps, we can leverage GitHub Actions and AWS CodeDeploy to streamline their CI/CD processes, automate deployments, and reduce the time it takes to release software updates.



Ref: AWS Github Interface

Deployment and Git Repository:

To deploy a Django project on AWS EC2 running Ubuntu, follow these general steps:

1. Launch an EC2 instance:

- Sign in to the AWS Management Console.
- Open the EC2 service and click on "Launch Instances".
- Select an Ubuntu Server AMI and choose an instance type.
- Configure the instance details, storage, security groups, and other settings.
- Create or select an existing key pair for SSH access.
- Launch the instance.

2. Connect to the EC2 instance:

- Retrieve the public IP or DNS of the EC2 instance from the EC2 dashboard.
- Open a terminal or command prompt on your local machine.
- Use SSH to connect to the EC2 instance using the key pair:

  ssh -i /path/to/key.pem ubuntu@<public_ip_or_dns>

3. Install system dependencies:

- Update the package manager:

  sudo apt update

  sudo apt upgrade

- Install Python and pip:

  sudo apt install python3 python3-pip

4. Set up a virtual environment (optional but recommended):

- Install virtualenv:

  sudo pip3 install virtualenv

- Create a virtual environment:

  virtualenv env

- Activate the virtual environment:

  source env/bin/activate

5. Clone your Django project:
- Install Git (if not already installed):

  sudo apt install git

- Clone your project repository:

  git clone <https://github.com/adhikarisarthak/fridgemanagement.git>

6. Install project dependencies:
- Navigate to your project directory:

  cd </home/ubuntu/fridgemanagement>

- Install required packages using pip and your requirements.txt file:

  pip install -r requirements.txt

7. Configure your Django project:
- Update the settings.py file in your Django project to set appropriate configurations for your EC2 instance, such as the allowed hosts and database settings.

8. Run migrations and collect static files:
- Apply migrations to create database tables:

  python manage.py migrate

- Collect static files:

  python manage.py collectstatic

9. Test your Django project locally:
- Start the development server:

  python manage.py runserver 0.0.0.0:80

- Open a web browser and access your Django project using the EC2 instance's public IP or DNS and the appropriate port (default is 8000):

  http://<public_ip_or_dns>

  AWS assigned an elastic ip

  Go to http://www.fridgemanagement.com or http://fridgemanagement.com

10. (Sprint)Configure a web server (e.g., Nginx or Apache):
- Install and configure the web server of your choice (e.g., Nginx).
- Configure Nginx to proxy requests to the Django development server or Gunicorn.

- Update your Nginx configuration to serve static files directly or proxy them to Django.
- Restart Nginx to apply the changes.

11. Test your Django project on the production server:
- Access your Django project using the EC2 instance's public IP or DNS (without the port) in a web browser.
- Ensure that everything is functioning as expected.

12. (Sprint)Set up a domain name and HTTPS:
- Register a domain name with a registrar of your choice.
- Configure DNS settings to point your domain to the EC2 instance's public IP or DNS.
- Set up an SSL/TLS certificate using a service like Let's Encrypt.

Configuring a WSGI server like Gunicorn is in the sprints.

Our Git Repository is:

https://github.com/adhikarisarthak/fridgemanagement.git

IV. Requirement Changes

Functional requirements including:

- remove item from fridge
- update quantity of item in fridge
- sort items by expiration date
- connection of fridges to user accounts

have been moved from core implemented features to planned features in future sprints. This modification does not entail any changes to our design architecture as the features will be implemented using the same MVT pattern in parallel with existing implemented features.