A2 WasteLess

Student: Andrei Tosa

**Group: 30431**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

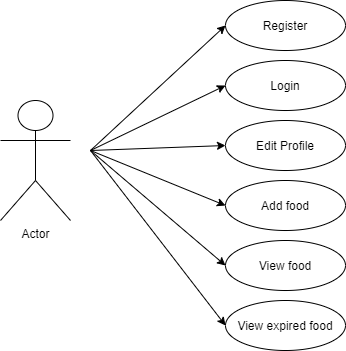
For this assignment we have to design and implement an application that helps users manage food waste. The user should be able to track all his food and see if he has an excess or if something went bad. He is presented with options to donate the food.

# Functional Requirements

Once a user is authenticated he can input grocery lists and see reports of how much food is wasted weekly and monthly. A grocery list item has a name and a quantity as well as a calorie value, purchase date, expiration date and consumption date. The system also allows users to track goals and minimize waste by sending reminders if waste levels are too high based on ideal burndown rates. The ideal burndown rate for 100 calories worth of groceries due to expire in 5 days is 20 calories worth of groceries per day. The system should provide you with options to donate excess food to various local food charities and soup kitchens and notify you of them prior to item expiration.

# Non-functional Requirements

* Implement and test the application
* Commit the work you do on your Git repository. Do it iteratively as you progress, not all at once (this will incur a penalty on your final mark)
* Use any OOP language you like. Non-exhaustive: Python, C#, Java, Ruby, C/C++, JS+Typescript
* Use a client-server architecture
* Use an observer for sending notifications to users about donation options when item
* expiration is due
* The data will be stored in a database
* All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.



2. Use-Case Model

Use case: **Show food**

Level: **user-goal level**

Primary actor: **User**

Main success scenario:

*1.Check if active\_user is logged in*

*2.Request server for food list*

*3.Receive food list*

*4.Display food list for the user*

Extensions: *[If no logged in user then show invalid command]*

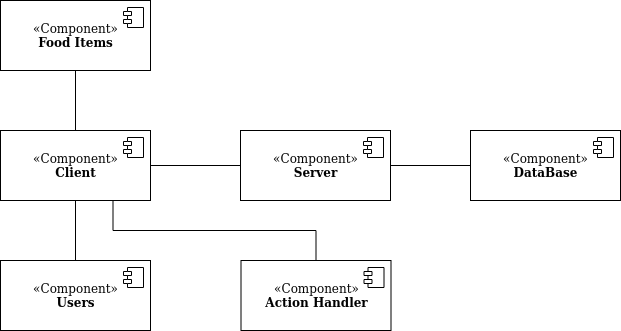
3. System Architectural Design

**3.1 Architectural Pattern Description**

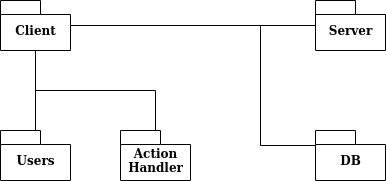
For this project I was required to use the Client Server Architecture. Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests.

**3.2 Diagrams**

**Component Diagram**



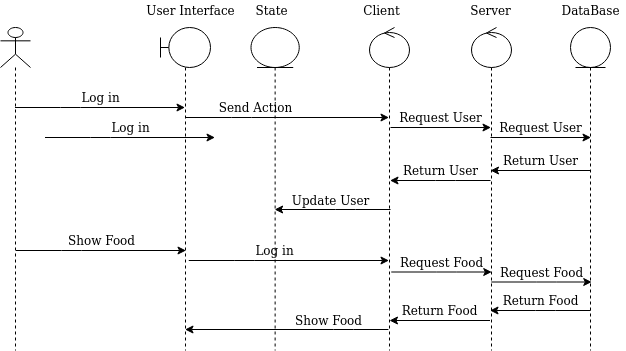
**Package Diagram**



**Deployment Diagram**



4. UML Sequence Diagrams

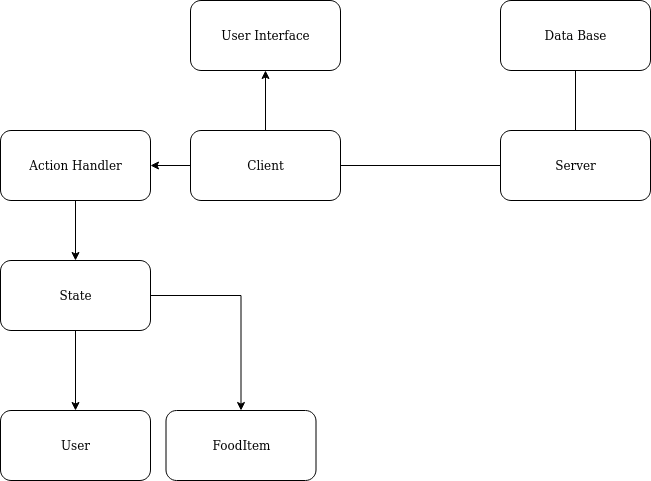


5. Class Design

**5.1 Design Patterns Description**

The Observer design pattern is one of the twenty-three well-known *"Gang of Four" design patterns* that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse. The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

**5.2 UML Class Diagram**



6. Data Model

**Profile:**

User: username, password, email

**FoodItem**:

name = String

quantity = PositiveInteger

calories = PositiveInteger

buy\_date = Date

exp\_date = Date

user\_id = ForeignKey

7. System Testing

Each component has been tested individually and after assuring functionality they have been put together. After that, data-flow testing was done to ensure that all components function as expected.

8. Bibliography

* https://www.youtube.com/watch?v=YXPyB4XeYLA&list=PL63UmddwapmmX9f25KIcnPYkk4dTrgKrV&index=2&t=14888s
* https://www.youtube.com/watch?v=SQj17D1Q\_6s&list=PL63UmddwapmmX9f25KIcnPYkk4dTrgKrV&index=3&t=0s
* https://www.youtube.com/watch?v=SQj17D1Q\_6s&list=PL63UmddwapmmX9f25KIcnPYkk4dTrgKrV&index=3&t=0s
* https://pythonprogramming.net/client-chatroom-sockets-tutorial-python-3/?completed=/server-chatroom-sockets-tutorial-python-3/
* https://https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/?completed=/pickle-objects-sockets-tutorial-python-3/www.journaldev.com/23365/python-string-to-datetime-strptime
* https://www.w3schools.com/sql/sql\_delete.asp