WasteLess

Analysis and Design Document

Student: Kando Edina

**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

Design and implement an application that helps users manage food waste.

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.

# Functional Requirements

Implement the assignment specification.

# Non-functional Requirements

* Use an ORM and a DI Container
* Use any OOP language you like. Non-exhaustive: Python, C#, Java, Ruby, C/C++, JS+Typescript
* Use a layered architecture
* Use the abstract factory pattern for creating weekly/monthly reports
* 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

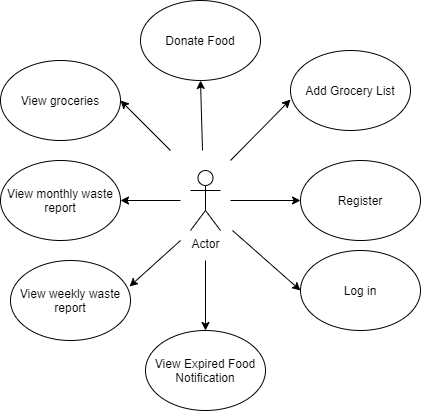
*Use case: View Waste Report*

*Level: user-goal level*

*Primary actor: regular user*

*Main success scenario: A table with the monthly/weekly wastes is shown to the user if he is logged in*

*Extensions: An alert box tells the user if an error occurred.*

s

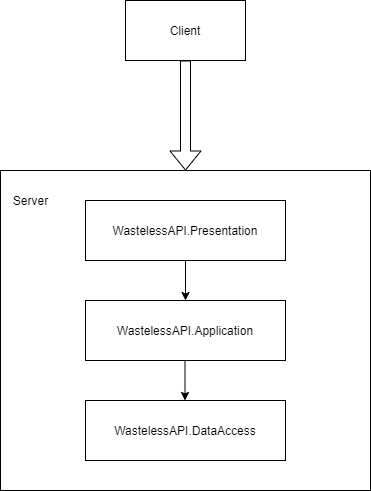
3. System Architectural Design

**3.1 Architectural Pattern Description**

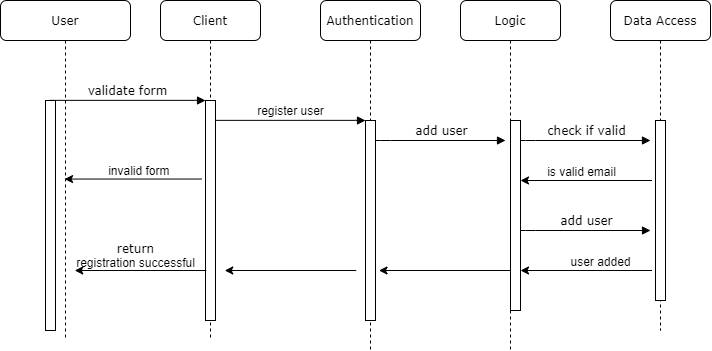
I used a client-server architecture with the client written in Angular/Typescript while the server is written in .Net Core.

The server is organized in a layered architecture with 3 layers: Presentation, Application and Data Access.

**3.2 Diagrams**

**

4. UML Sequence Diagrams



5. Class Design

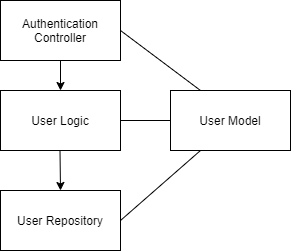
**5.1 Design Patterns Description**

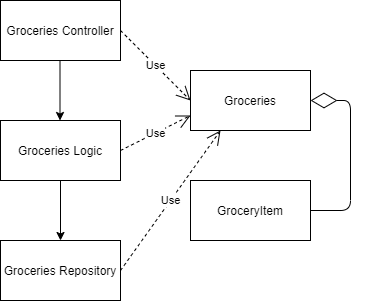
Firstly, for the report generation, I used the abstract factory pattern. For this, I created a factory interface that is implemented by the two concrete factories. I also added an interface for reports and created two concrete classes that implement it (MonthlyReport and WeeklyReport).

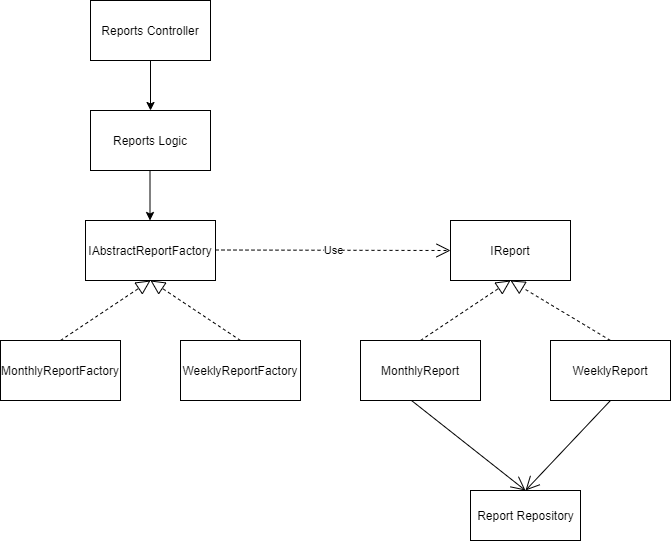
For the donation possibility notification, I used the observer design pattern. The observer is the PushNotification and the subject is the item expiration (if an item is bound to expire in the following 5 days, the user gets a notification that they should donate).

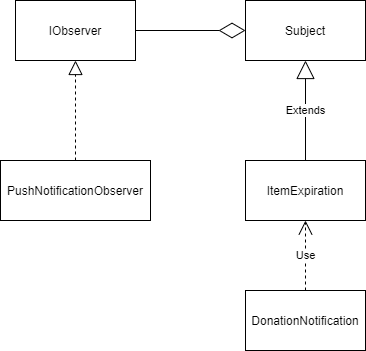
I added the Coravel library which I use as a scheduler. This schedule calls every day the CheckExpirationDates method from the subject which then notifies the PushNotificationObserver if any groceries satisfy the above mentioned rule.

**5.2 UML Class Diagram**









6. Data Model

The data models used in the application are the User, the GroceryList and the GroceryItem.

The User contains the email and encrypted password of the user, the grocery item contains name and a quantity as well as a calorie value, purchase date, expiration date and consumption date, while the GroceryList has a name and a list of GroceryItems.

7. System Testing

During the development process I did validation testing for my application, by providing input and analysing the results. For the client I used console logs while for the back-end I used debugging.

I also did integration testing, verifying if the entire flow of the use case is executed correctly.

I did not have yet the time to write unit tests.

8. Bibliography