Wasteless 1

Analysis and Design Document

Student:Cioban Dumitru-Darius

**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

The application provides the option of a user to manage their food waste, by tracking it using this application, by tracking how food is wasted, and providing notifications when food waste is imminent, giving the user the option to donate.

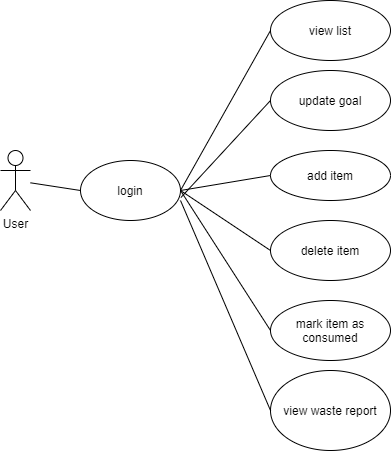
# Functional Requirements

*The functional requirements are adding and viewing the grocery list, viewing reports regarding the food waste, and notifications providing the donation option.*

# Non-functional Requirements

*The non-functional requirements are using layered architecture, using an abstract factory pattern and a database to store the data.*

2. Use-Case Model

**

*Use case: add item*

*Level: user-goal level*

*Primary actor: user*

*Main success scenario: The user inputs the data of the new grocery item they want to add. It passes through the validator, and if the data is valid, a new item is added to their list.*

*Extensions: The validator returns an invalid value, signifying that the data is not according to the standard. An error message is displayed, giving info on how the data should look like. Another possibility is the case where the item wasn’t successfully added to the database, in which case the update won’t take place and nothing will be changed.*

3. System Architectural Design

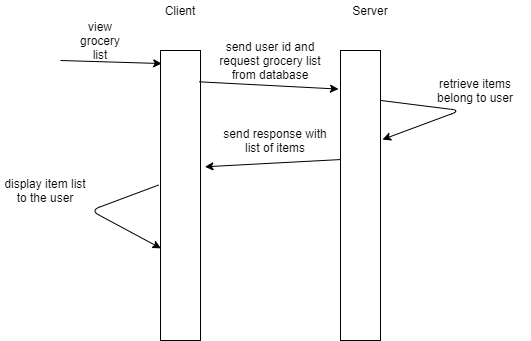
**3.1 Architectural Pattern Description**

*The architectural pattern used is the layered architecture. The layered architecture splits the application in 4 layers, the database layer, the persistence layer, the business layer and the presentation layer. The key feature is that each layer only has access to the layer below.*

**3.2 Diagrams**



*It can be seen from the above diagram (taken from the web) how the layered architecture looks like. The database has the database, while the persistence layer usually has data access objects (DAO). The business logic happens in the business layer and the user has access to the presentation layer (usually the UI). So if for example, the user wants to add an item, from the persistence layer the method from the business layer which adds the item is called, which will access the persistence layer, which in turn, accesses the database layer. No access can be made from let’s say the presentation layer to the persistence or database one.*

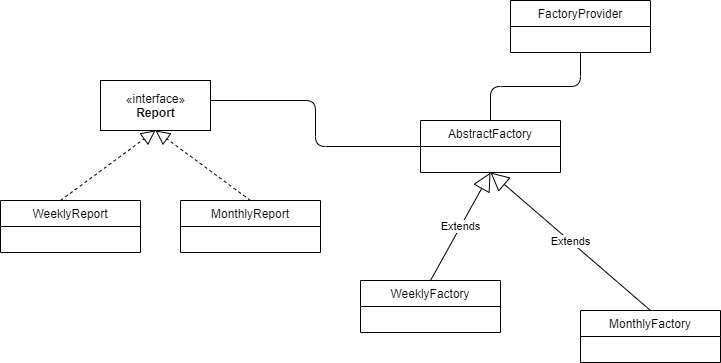
**

5. Class Design

**5.1 Design Patterns Description**

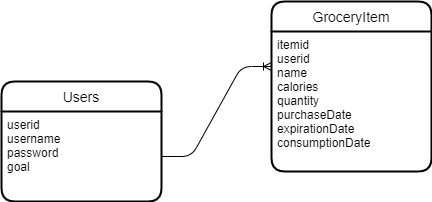
*The used designs pattern is the Abstract Factory one. In this application, there are two types of reports, monthly and weekly, which both implement the Report interface (they are both reports). Factories are used to create an object, so since we have two types of objects (weekly and monthly reports), we need two factories. But since those reports have in common that they are reports, so do the factories, both extending an abstract factory. Then, another component is needed, a factory provider, which provides a factory based on the type of report needed.*

**5.2 UML Class Diagram**

**

*It can be seen in this diagram that both WeeklyReport and MonthlyReport implement the Report interface, and WeeklyFactory and MonthlyFactory extend the AbstractFactory, which is an abstract class. Then, using FactoryProvider, a factory can be created, which will be used to create a report (MonthlyFactory can only create MonthlyReport objects, the same for weekly).*

6. Data Model

**

*The data is represented by 2 tables. One for the users, where their id is the primary key. Username and password are used for authentication and goal for functionality. The grocery item table has an item id as primary key, an user id (establishes a one to many relationship between users and items), a name, calorie count, quantity and 3 dates representing the purchase date, the expiration date and the consumption date.*

7. System Testing

*The testing was done by feeding the system both valid and invalid data, seeing how it reacts. However, there are some scenarios which haven’t been tested or have been tested but haven’t been updated (for example the need of inputing only numbers in number fields).*

8. Bibliography