Student:Manea Luca

**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

Nowadays, earth is facing a more and more dangerous situation regarding its resources, especially food. A main factor for the lack of food is its poor managing when buyed. Design and implement an application that helps users manage food waste.

# 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.

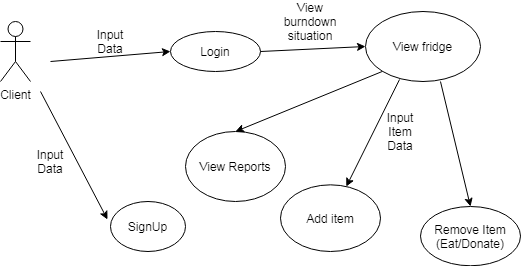
The ideal burndown rate for 100 calories worth of groceries due to expire in 5 days is 20 calories worth of groceries per day.

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

* The usage of an ORM and a DI Container
* Committing the work on the Git repository
* The usage of any OOP languag. Non-exhaustive: Python, C#, Java, Ruby, C/C++, JS+Typescript
* The usage of a layered architecture
* The usage of 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: Sign-up

Level: user-goal

Primary actor: client

Main success scenario:

* The user inputs a username and a password
* The username is checked to be unique and the password is re-entered by the user
* The user is now registered

Extensions:

* If the password is shorter than 8 characters/ username than 5 characters the operation fails

Use case: Login

Level: sub-function

Primary actor: client

Main success scenario:

* The user inputs a username and a password
* They are checked
* The user logs in

Extensions:

* The user views today’s optimal burndown rate

Use case: View fridge

Level: sub-function

Primary actor: client

Main success scenario:

* The user is shown current contents of the fridge

Use case: Add item

Level: user-goal

Primary actor: client

Main success scenario:

* The user inputs data related to a product
* The data is checked
* The object is added

Extensions:

* The user’s view of the fridge is updated with the addition of the new object

Use case: View Reports

Level: sub-function

Primary actor: client

Main success scenario:

* The user is displayed the items that expired in a past period of time

Use case: View Reports

Level: sub-function

Primary actor: client

Main success scenario:

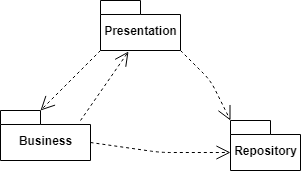
* The user is displayed the items that expired in a past period of time

3. System Architectural Design

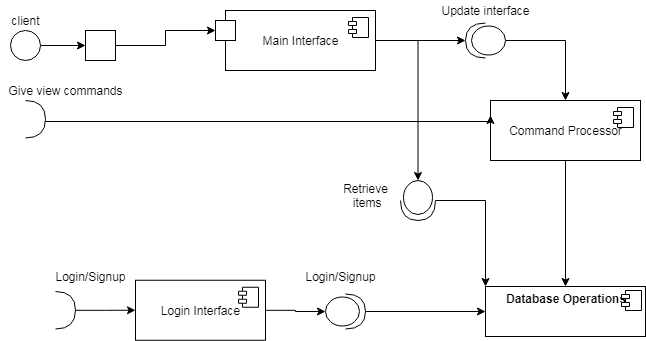
**3.1 Architectural Pattern Description**

The used architectural pattern is the layered one. Due to the simplicity of the application I used only three layers, which are reflected in the packages: Presentation layer, the business layer and the repository layer.

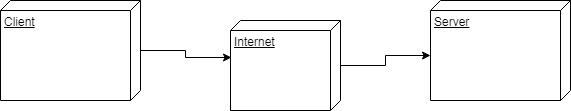
**3.2 Diagrams**



Package Diagram

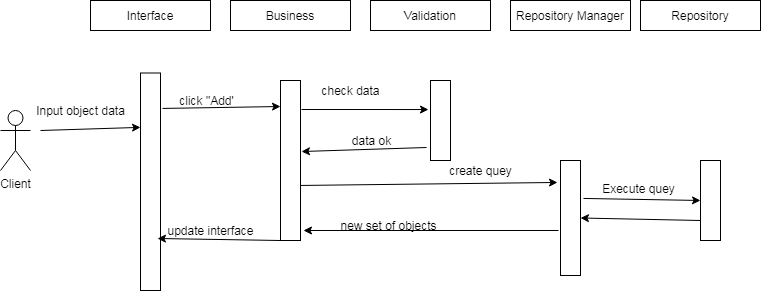


Component Diagram



Deployment diagram

4. UML Sequence Diagrams



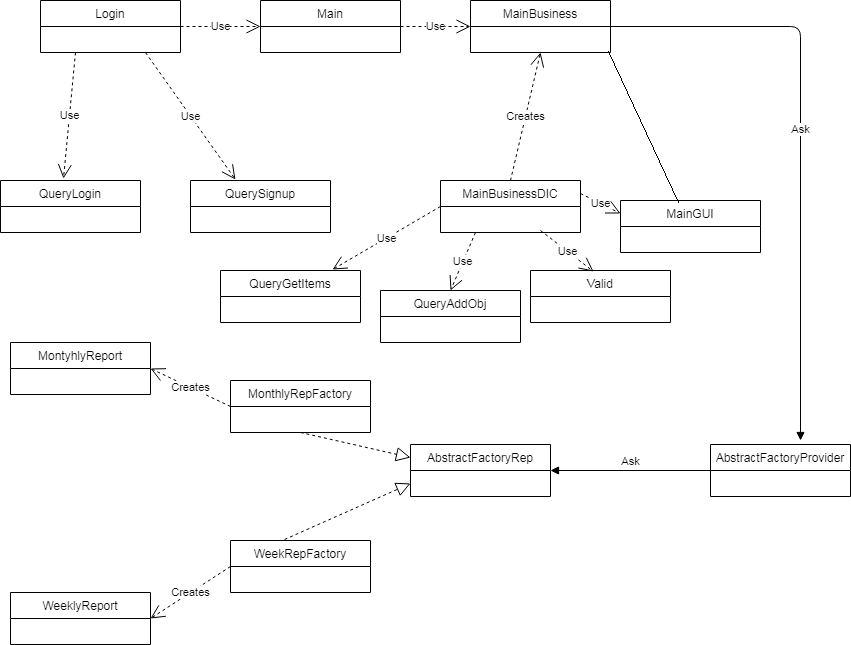
Add item to fridge scenario

5. Class Design

**5.1 Design Patterns Description**

I used the Abstract factory pattern to generate the reports. Usually there are more types of reports in each of the classes covered by each concrete factory (in our case weekly and monthly report generator factories), but I couldn’t think of any variation of reports.

**5.2 UML Class Diagram**



UML Class Diagram

The general design of the application follows the flow of the user experience. The app starts in the Login interface, which has as tools to communicate with the database the QueryLogin and the QuerySignup to verify/register the user. Later in the design, the creation of the MainBusiness class is done. This class is created using a DI Container to ease its intricate instantiation. MainBusiness works in close connection with MainGUI from there on to handle the user’s requests. These requests are represented by the respective repository classes used by the DI container. The abstract Factory pattern is used to create reports.

6. Data Model

The data models used for this project were:

* The client model:
  + ID
  + Username
  + Password
* The Grocery Item model:
  + ID
  + Id\_client
  + name
  + quantity
  + kcal
  + buy date
  + expiration date

7. System Testing

The main developing tool used in testing was printing to the system console. Classes and functions with specific tasks underwent unit testing . The testing also followed the flow of the user experience, as the application was developed following the same flow.

As the classes and methods regarding user login/register were completed I applied validation testing to check their correctness. The same was done with testing the second step of the application, the Fridge operations. Basically, two integration epochs took place: Login and Main one. After both proved successful, one validation step was done to check the whole app.

As the data sample used was not very large and it is not the scope of this application to achieve large data samples, partitioning and boundary analysis were not used.

8. Bibliography

1. <https://www.baeldung.com/java-abstract-factory-pattern>
2. <https://mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/>
3. <https://stackoverflow.com/questions/34174107/java-error-javax-swing-jtable1-cannot-be-cast-to-javax-swing-table-defaulttab>
4. <https://coderanch.com/t/602307/java/add-multiple-ActionListeners-multiple-Buttons>