<WasteLess>
Analysis and Design Document

Student: Manea Luca
Group: 30431

Table of Contents

1. Requirements Analysis

1.1 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.

1.2 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.

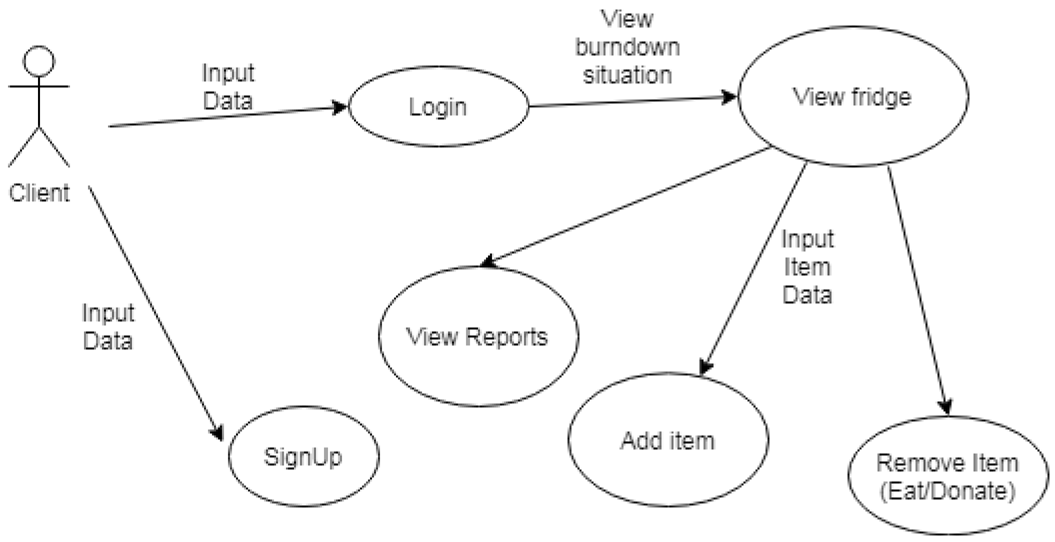
The ideal burndown rate for 100 calories worth of groceries due to expire in 5 days is 20 calories worth of groceries per day.

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.

1.3 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: Sign-up
Level: user-goal
Primary actor: client
Main success scenario:

- The user inputs a username and a password
- The username is checked against validation rules
- The username is sent to the database where it is checked to be unique
- The user is now registered

Extensions:

- If the password is shorter than 8 characters/ username than 5 characters the operation fails
- A number is sent back to the client which encodes the possible outcomes(registered, not unique, database fail)

Use case: Login
Level: sub-function
Primary actor: client
Main success scenario:

- The user inputs a username and a password
- They are sent to the server
- They are checked with the database
- 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:

- A request is sent to the server
- Data is retrieved
- 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
- Data is sent to the server
- The object is added
- The object is retrieved from the database and sent to the client, as it is stored in the database (ORM)

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:

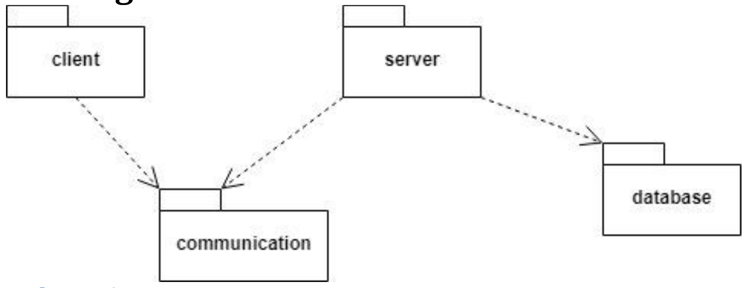
- The user is displayed the items that expired in a past period of time

3. System Architectural Design

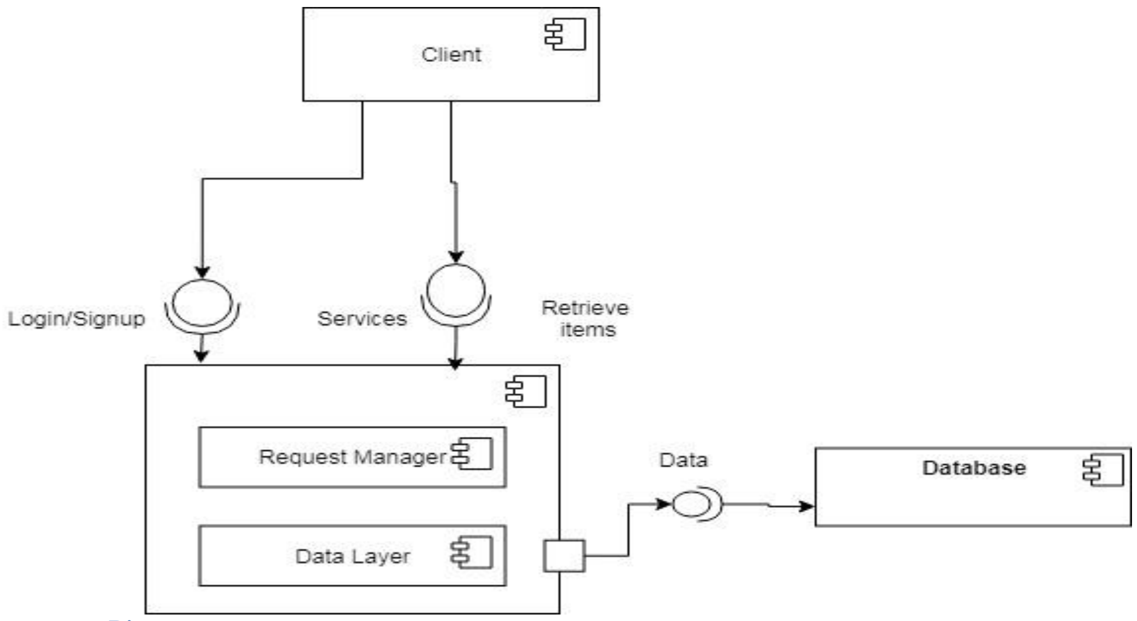
3.1 Architectural Pattern Description

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.

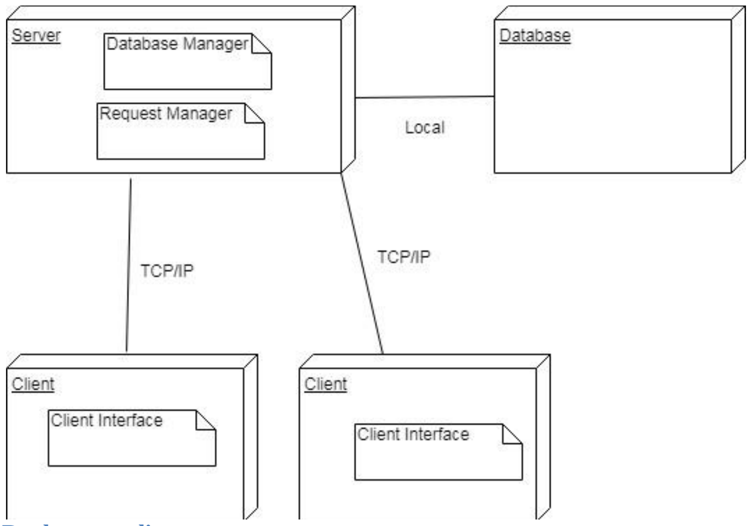
3.2 Diagrams



Package Diagram

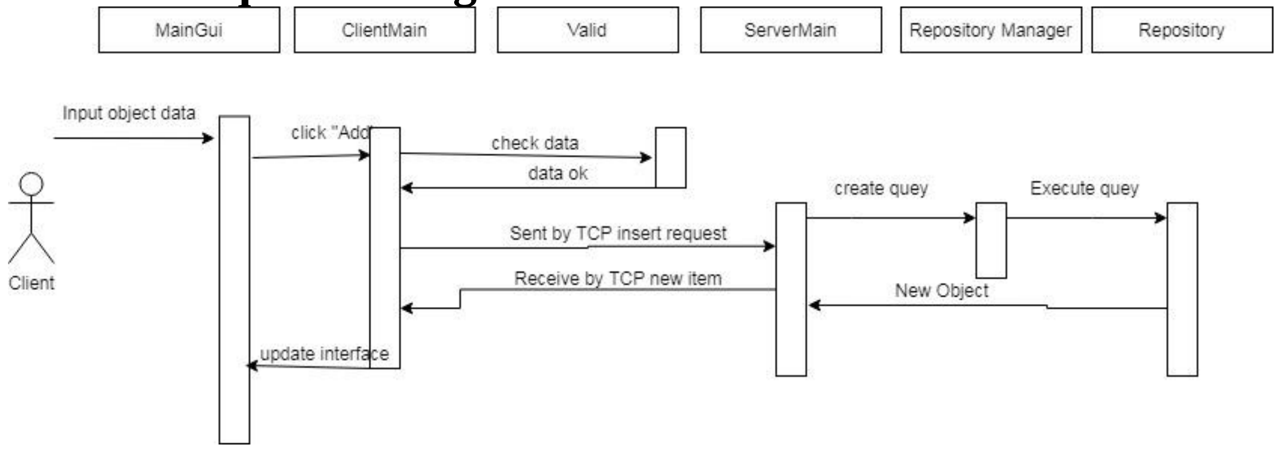


Component Diagram



Deployment diagram

4. UML Sequence Diagrams



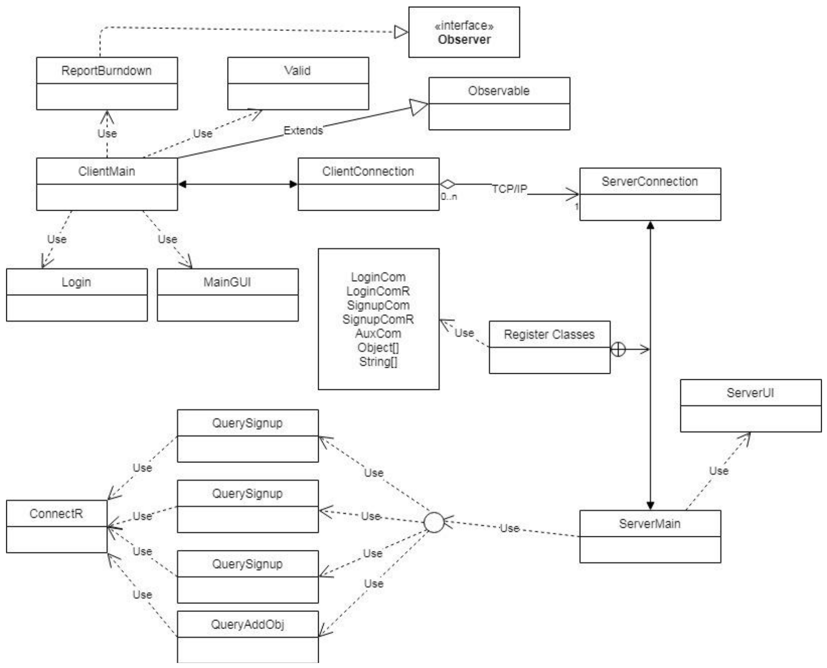
Add item to fridge scenario

5. Class Design

5.1 Design Patterns Description

I used the Observer pattern to notice clients when items would expire the next day. This was a bit out of place because I couldn't extend the notify operation over more classes due to the client-server architecture. Thus the notify is done between two classes where there already exists an association relation.

5.2 UML Class Diagram



UML Class Diagram

The general design of the application is based on the Client-Server architecture . This was made possible in virtue of the KryoNet library, which does the serialization and port handling. It requires the classes which are to be serialized and send by TCP or UDP to be "registered". Thus the existence of the RegisterClasses class.

The server and the client side use the ServerConnection respective the ClientConnection classes. They are used to create the server/client objects which are the actual endpoints of the connections. Also, the connection classes manage incoming messages and they decide what to do with the data received depending on the data class.

The Server connects to the database using the Mysql connector library.

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.

Also, as a very surprisingly efficient validation testing, I let my little sister to play around with the application after it was completed. She accidentally discovered an undesirable high number of bugs and helped me a lot. This is how the “Report Mode” was comprised, since you cannot delete items while viewing a report.

For more details about the general implementation of the application check the classes ClientMain and ServerMain.

8. Bibliography

1. https://www.geeksforgeeks.org/observer-pattern-set-2-implementation/?ref=rp
2. https://github.com/EsotericSoftware/kryonet/tree/master/lib