<Wasteless Assignment 2>

Analysis and Design Document

Student:Iacob Andrei Ștefan

**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

Wasteless2 is a software application built using the client-server architecture. Both parts are developed in Java, making use of GraphQL for querying . It provides users a way to better organize their grocery lists, entering all the products that have been bought into the application . The products from all the lists will then be available for visualization and the user will be able to perform multiple operations on them : check that a specific item has been consumed (and when) , set a goal regarding how many calories he/she intends to consume daily, see when the food that is available exceeds the values needed for his goal. Also, the user can see weekly and monthly reports showing how much food has been wasted in the last 7/30 days ( what products and how many calories). In order for the application to be complete, a person is able to donate any food that he/she would like to (the application will make suggestions in this sense) to local charities. The application can be used by multiple users , as it also provides a sign up system . So new accounts can be made and once you are logged in to the app, you are free to use any of its functionalities.

# Functional Requirements

Some of the main functional requirements of the application:

-New user registration

-Authentication

- New grocery list creation , adding products to the new list

-Specifying for each item the name ,caloric value, expiration date ,purchase date, quantity

-Visualizing all the products that have been purchased

-Having the ability to donate excess foods to local charities

-See reports of how much food is wasted weekly/monthly

-Setting a goal regarding the number of calories the user intends to consume daily

-Sending reminders when food waste levels are too high based on ideal burndown rates

# Non-functional Requirements

- Implement and test the application

- 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: donate food item

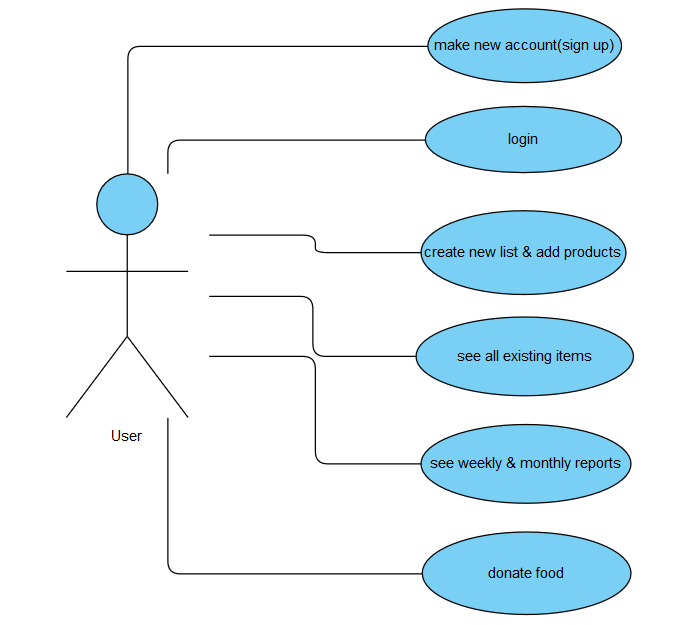
Level: user-goal level

Primary actor: user

Main success scenario: The logged in user chooses from the main menu the button which directs him to the Donation section. Now the user is notified which items are due to expire in the next 2 days . He is able to choose one of the items and donate it . This operation will remove the item from the database, as we consider that it is no longer in his possession.

Extensions: The user must own food items (and have them in the application , so say) to be able to donate. Otherwise the items list will be empty.

]



3. System Architectural Design

**3.1 Architectural Pattern Description**

For the development of this application the client-server architectural pattern has been used. On a ‘micro’ level , the client side is developed in Java and respects the layered architecture(presentation , business, data).

The client-server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection . Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them.

Also, the client-server architecture is a producer/consumer computing architecture where the server acts as the producer and the client as a consumer.

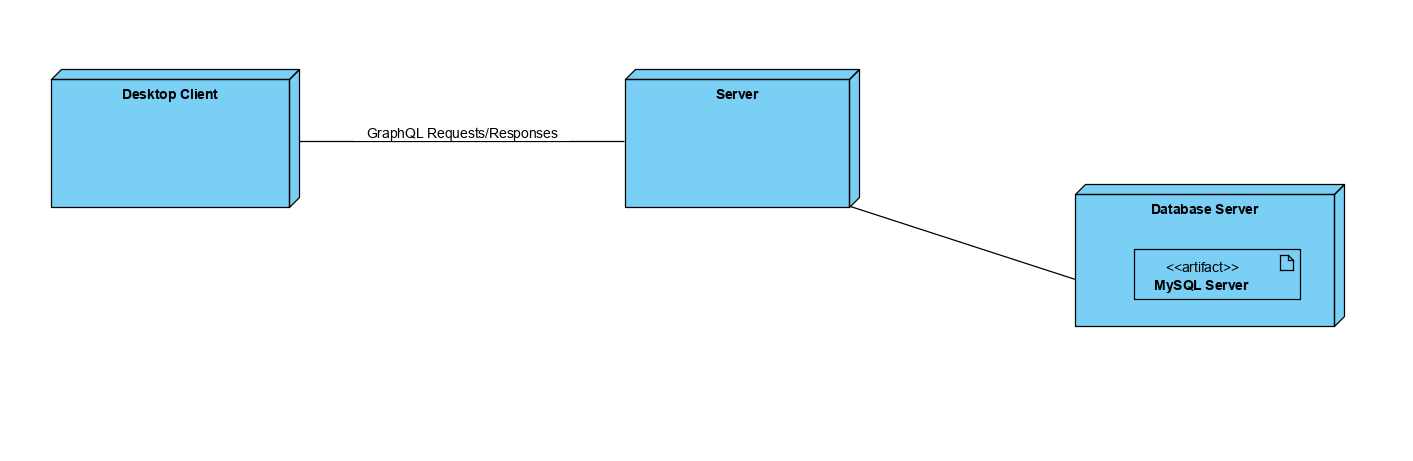
**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

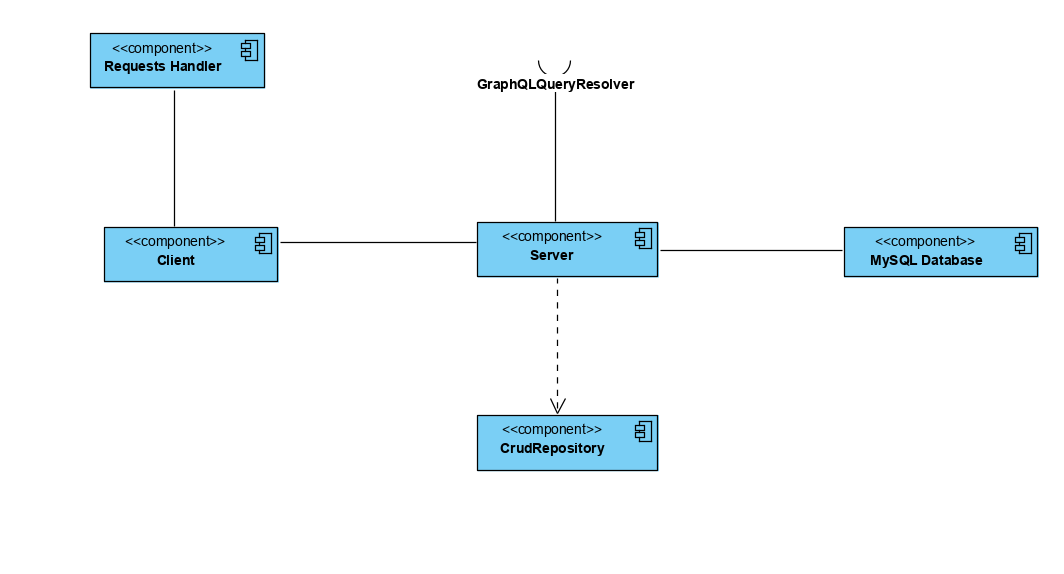
The system’s conceptual architecture :



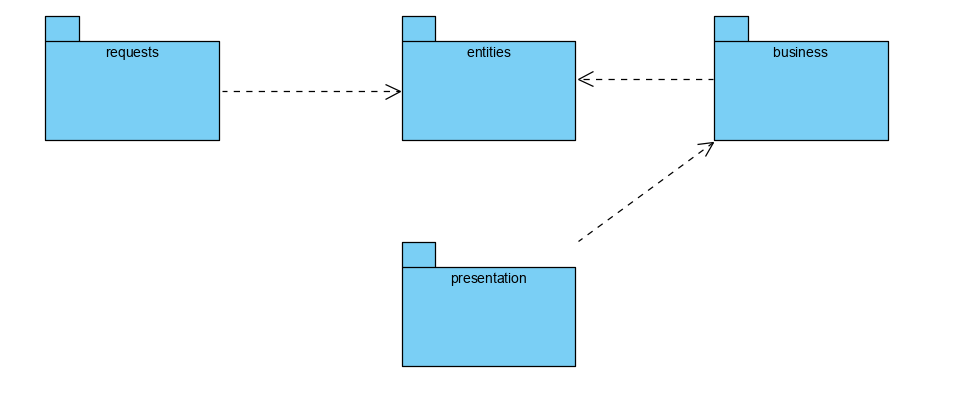
The clients are sending requests through the network, while the server is receiving and processing them . Once the server is done , he will provide a response that can be used by the client side afterwards( something to be printed on the UI etc.).

*Deployment:* 

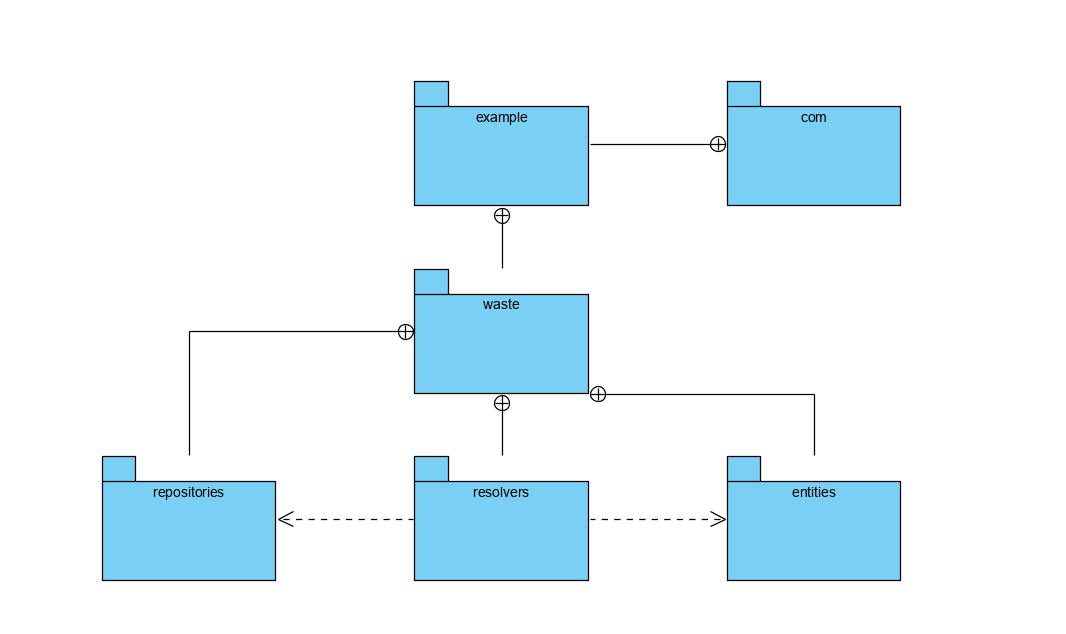
*Component*

**

*Package*

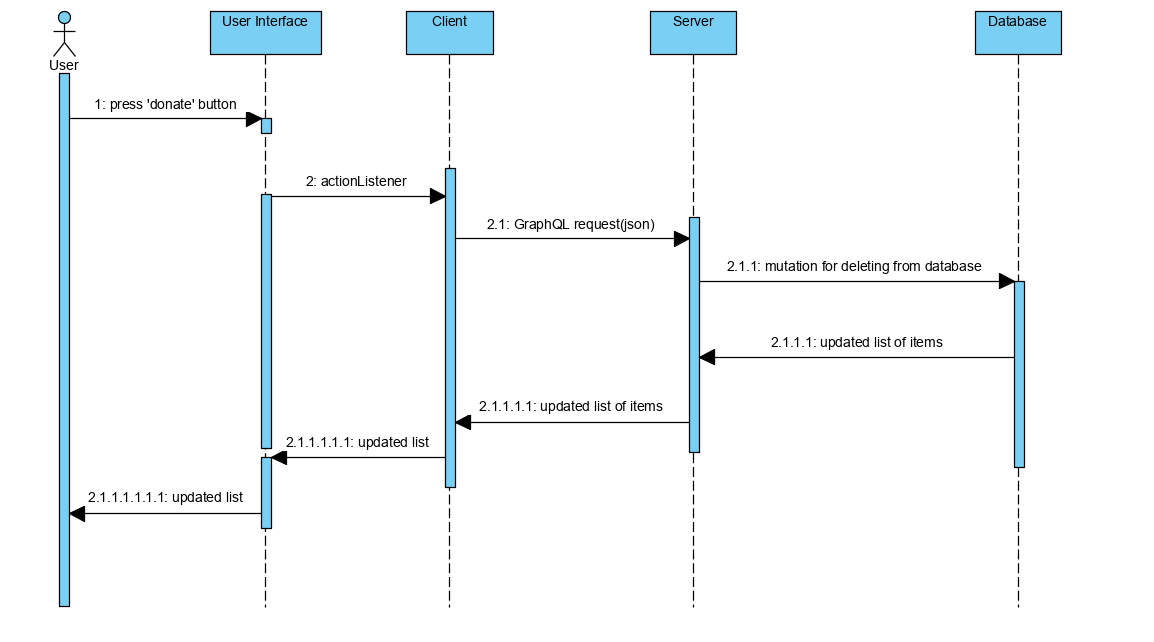
**

*Server:*

**

4. UML Sequence Diagrams

*I know that the response does not use the same type of line, but I couldn’t find any other in Paradigm.*



5. Class Design

**5.1 Design Patterns Description**

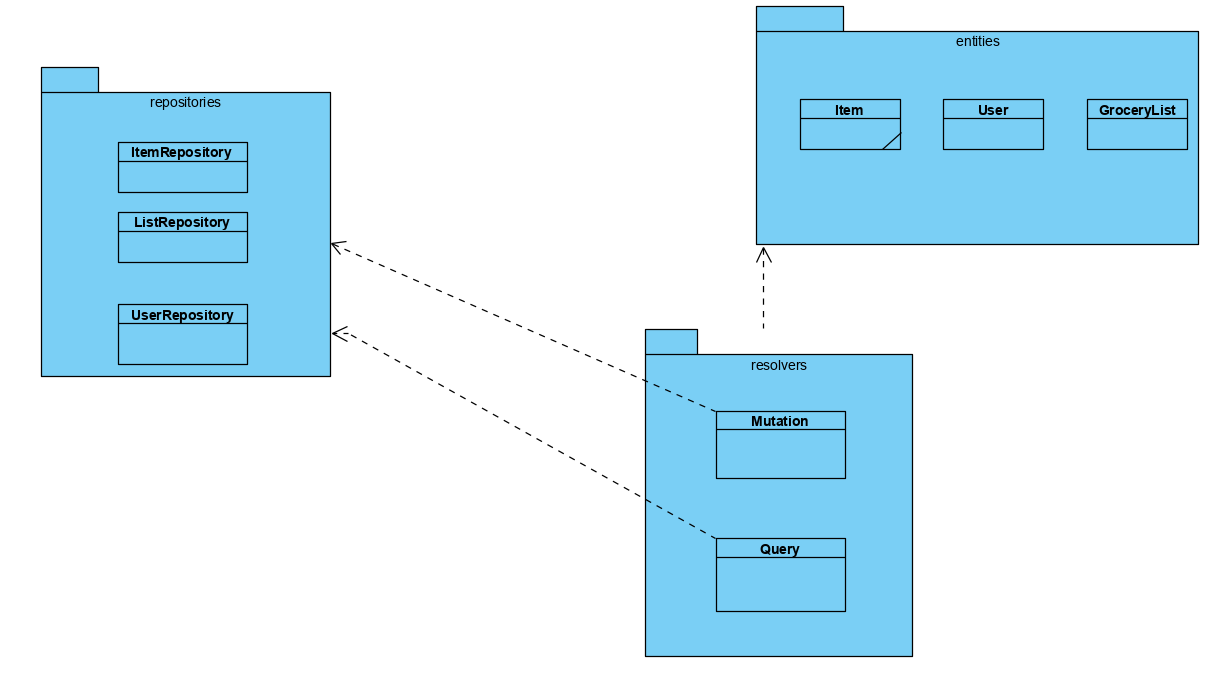
For this application ,the Abstract Factory Pattern has been used in order to generate weekly and monthly reports regarding food waste. This is a creational pattern , as its main focus is providing an easier way of creating objects. An interface is responsible for creating a factory of related objects without specifically specifying their classes. In our case, a report abstract factory interface was built. This was implemented by a report factory, from which two different sub-factories were “born”(monthly report factory and weekly report factory). This sub-factories were actually creating the products that we are interested in ,namely the two types of reports .The pattern makes adding new types of objects easier.

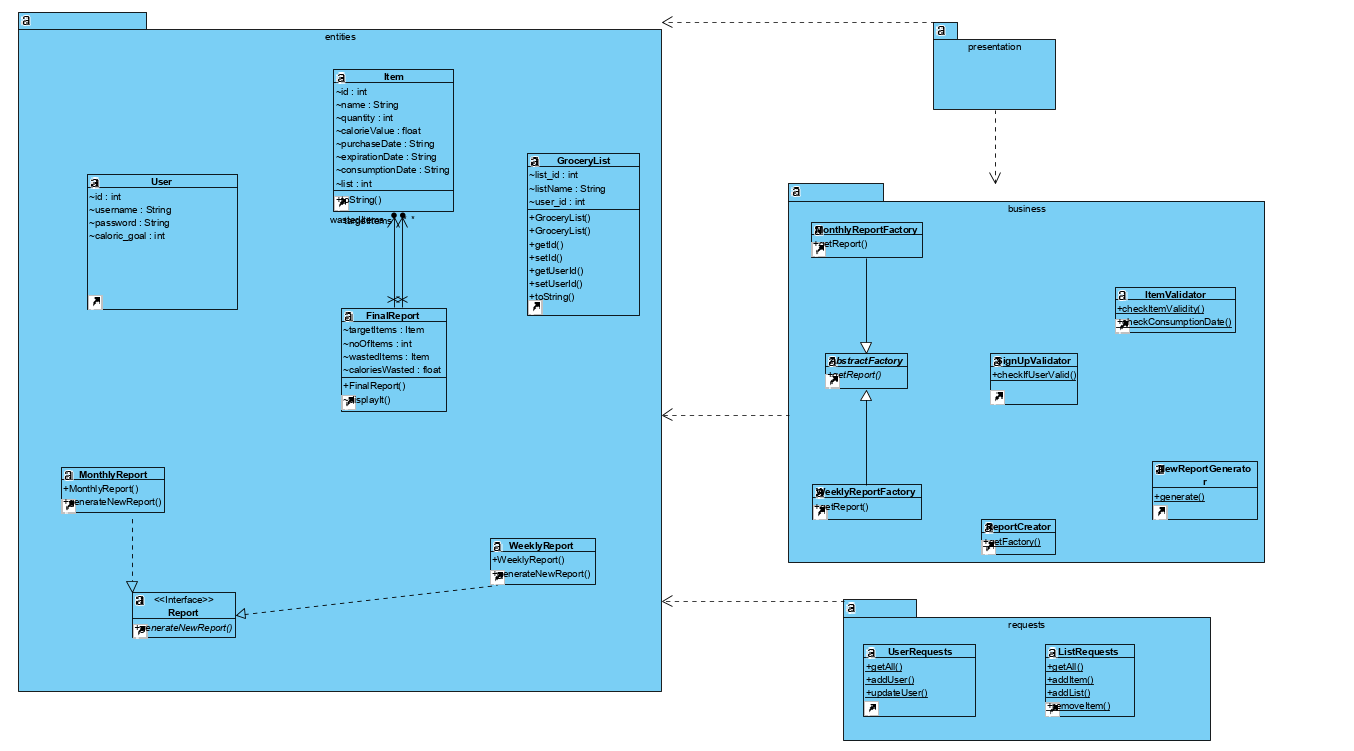
On the other hand , the Observer Design Pattern offers aid for the communication between objects . Here it is used on the Client side to send notifications whenever food items are due to expire soon (in the next 2 days). For implementing it, the Observer interface is implemented by the main frame. I used it more like a demo , and I thought that it is better to display it when the user is entering the application ( not every X seconds or each time he enters a specific window). This is because I do not like apps that are sending too many notifications and insist on solving them.

**5.2 UML Class Diagram**

I have already talked about the way in which the design patterns are used.

Like I said , entities are more separated here because I wanted a simpler way of performing requests on GraphQL (instead of having a list of items stored in every list, I would rather have a field ‘list\_id’ on each Item object which tells me what list it belongs to).





6. Data Model

For the second assignment , I have changed a bit the way in which entities are defined ( for simplicity , when working on the server side).

User: contains an id , calories goal , an username and a password.

Food Item: id, name , quantity, dates(expiration , purchase , consumption) represented as Strings , list id .

Grocery List: list\_id, list\_name , user\_id .

Even though the dates are represented as strings here , they are still checked on the validators. I implemented this way because it was easier to parse the response received from the server .

7. System Testing

Each component has been tested individually , immediately after being developed. Once the components were working as expected, they were put together.

Because I have worked with GraphQL , I have been able to build the server and run queries directly on the graphiql interface . So the server has been built priorly and tested with priority to ensure that the main functionalities( database connection , mutations) are met. After adding the client and providing requests-responses, the data-flow testing has been used.

8. Bibliography

<https://dev.to/sambenskin/howto-build-graphql-services-in-java-with-spring-boot---part-1-38b2?fbclid=IwAR2xCObpWczj45B8uAnrPJT2SpzeXpd13hBmSphwmytFjpbiGQyYVdPdNn4>

<https://stackoverflow.com/questions/51924820/anilist-api-v2-graphql?fbclid=IwAR1Am9iBx0UT3jG1ec_qf7n7y3KOvvsnbwZv8zpc-JcYReflSt8rk-w9UNo>

<https://dev.to/sambenskin/howto-integrate-a-mysql-database-into-your-java-spring-boot-graphql-service-26c>