Student:Dragoteanu Bogdan
Group:30431

Table of Contents

1. Requirements Analysis

1. Assignment Specification

The assignment is an application that helps users manage food waste.

Authenticated users can input grocery lists and see reports of how much food is wasted weekly and monthly. An item in the grocery list has the following data:

Name,quantity, calorie value, purchase date, expiration date and consumption date

The system allows the users to track the goals and minimize waste by reminding them if the waste levels are too high based on ideal burn down rates.

The system gives the users options to donate excess food to various local food charities and soup kitchens.

If an item consumption date is close to the current date the user is notified.

2. Functional Requirements

- Search → The user is able to find data about the grocery item by searching explicitly for it
- Reports → The system will generate a weekly and monthly report for the user
- Modify → The user is able to change a grocery item's information
- AddItem → Insert a new item

2.1 Non-functional Requirements

- Security → The system shall ensure that data is protected from unauthorized access
  → System data may be accessed only by users authenticated by means of a username and password
- Portability → The application works on both Windows and Linux machines
- Extendibility → Features can be further enriched

2. Use-Case Model

Use case: Log In

Level: user-goal

Primary actor: Registered User

Main success scenario: User log into the system

Extensions: On fail it notifies the user that something is wrong

Use case: Sign Up

Level: user-goal

Primary actor: User

Main success scenario:

Extensions:

Use case: Search

Level: user-goal

Primary actor: Registered User

Main success scenario: The system finds the item in the list and shows its data to the user

Extensions: On fail it notifies the user that the item doesn't exist or that they may have misspelled its name

Use case: Refresh

Level: user-goal

Primary actor: Registered User

Main success scenario: Refreshes the Grocery List

Extensions: -

Use case: Modify

Level: user-goal

Primary actor: Registered User

Main success scenario: Changes the data of a selected item in the List

Extensions: If some parameters are bad then the user is notified

Use case: Add Item to Grocery List

Level: user-goal

Primary actor: Registered User

Main success scenario: Adds the data of a new item to the List

Extensions: If some parameters are bad then the user is notified

Use case: Create Report

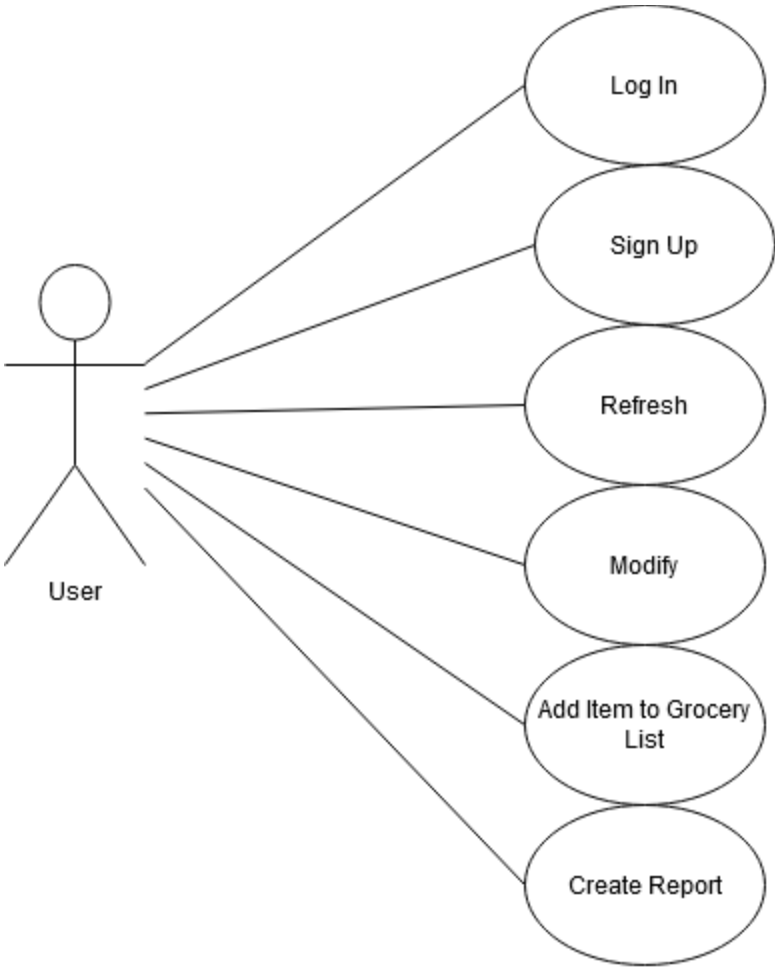
Level: user-goal

Primary actor: Registered User

Main success scenario: Creates report with the data from thegrocery list and shows it to the user.

Type depends on the chosen one (Weekly/Monthly).

Extensions: If the list is empty the user is notified that nothing can be shown.



3. System Architectural Design

The system will be made using a client server architecture.

The client will be made using a layered architecture

3.1 Architectural Pattern Description

Components in a layered architecture are organized into horizontal layers, each performing a specific role within the application. In our case we have four layers:

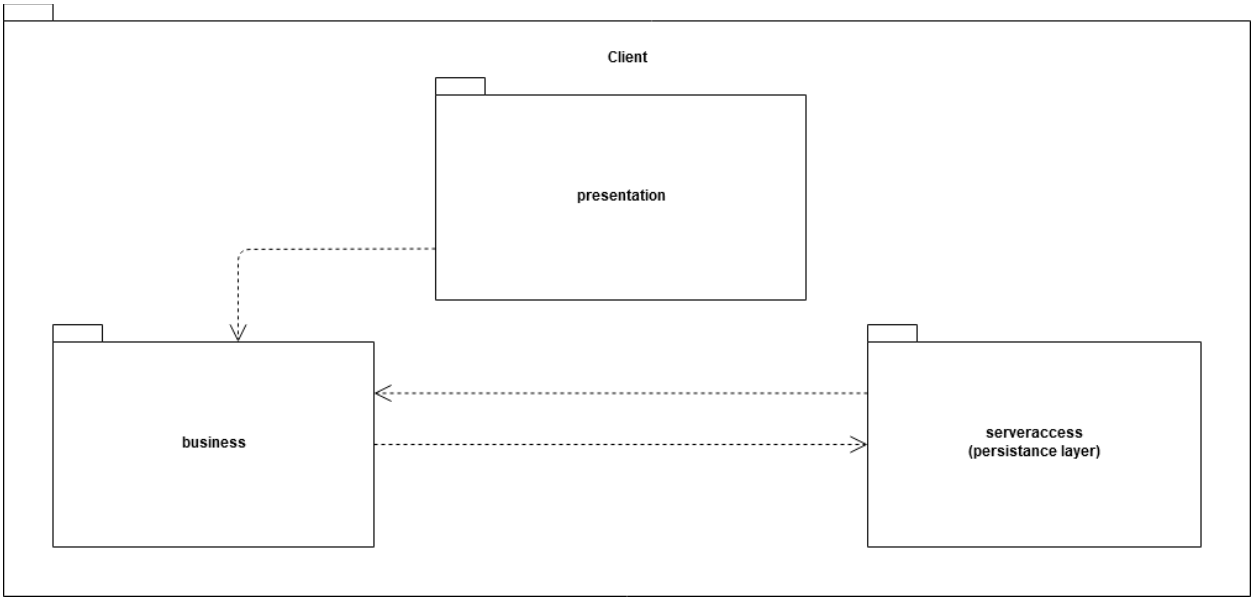
- Presentation → handles the user interface
- Business → handles requests
- ServerAccess

The layers are closed → requests move from layer to layer:
presentation → business → serveraccess

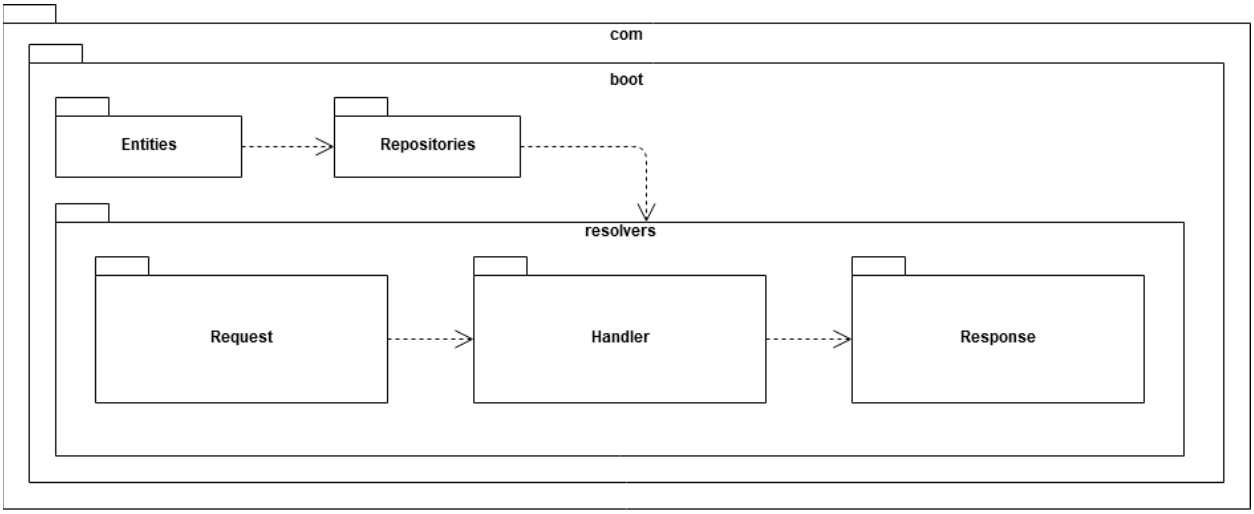
3.2 Diagrams

Package Diagram

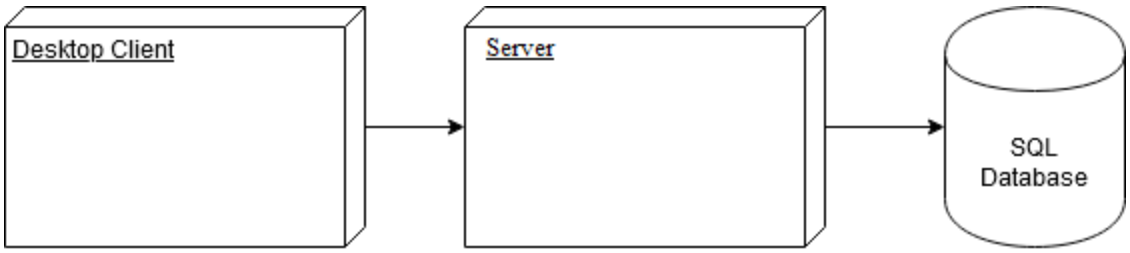
CLIENT



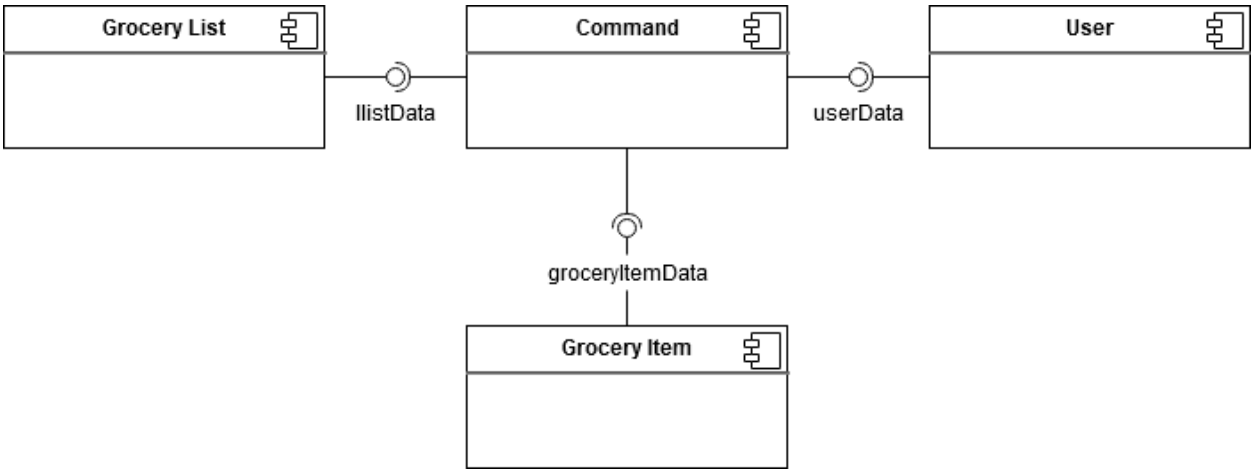
SERVER



Deployment Diagram



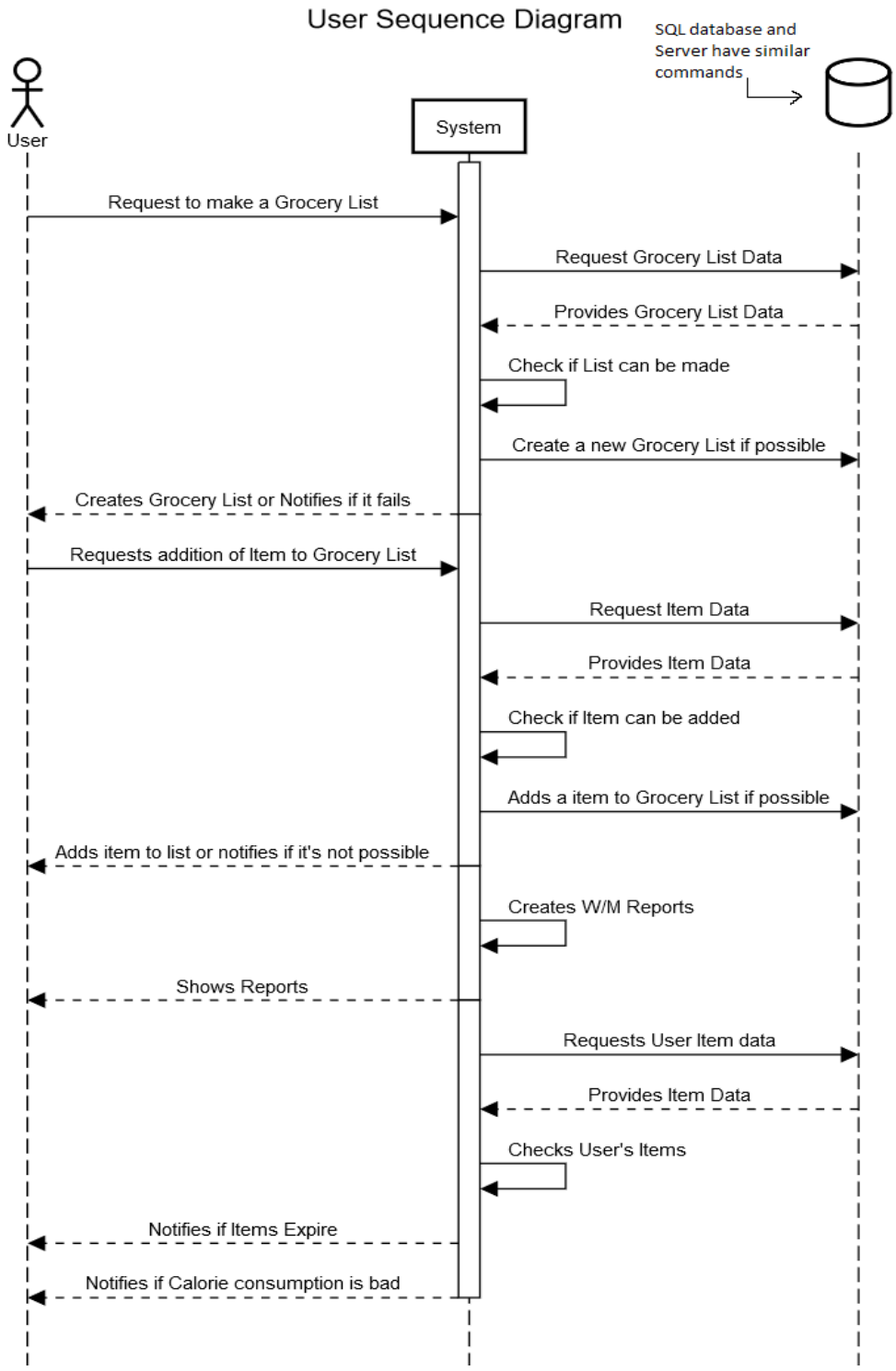
Component Diagram



Patterns used:

- layered → used to structure the program into groups of subtasks
  → each layer provides services to the next higher layer
- Server will be using and ORM and dependecy injection that come from the libraries used:
  - Spring
  - Graphql
- Observer → Used to notify the user if today is the final day set for certain item:
  - Notified with a list that contains the name and list of the item
- Decorator Pattern → separate the result writing of a bad or good calorie intake
- Mediator Pattern → separate requests from responses (reduce coupling)
  → makes implementing and modifying functions easier

4. UML Sequence Diagrams



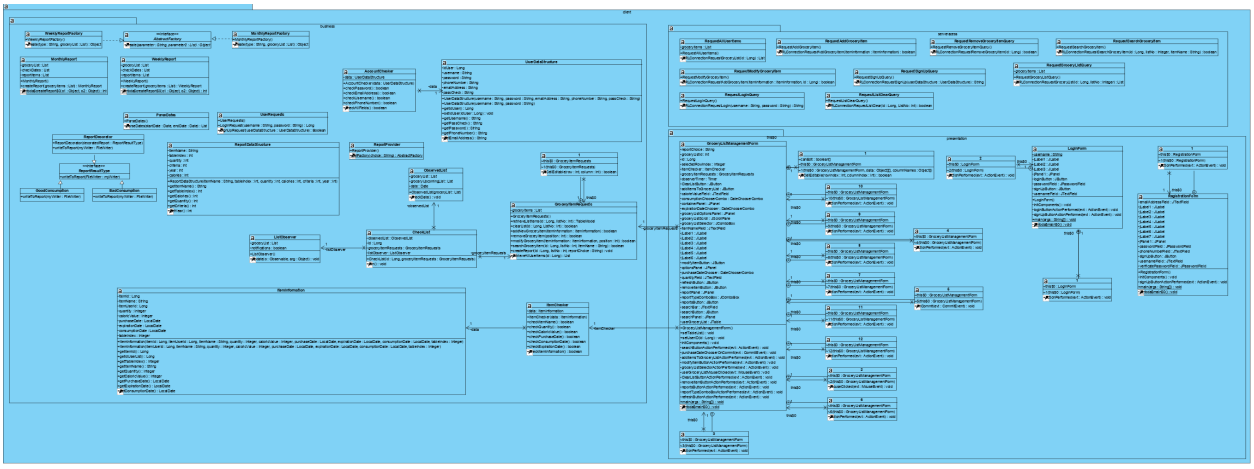
5. Class Design

5.1 Design Patterns Description

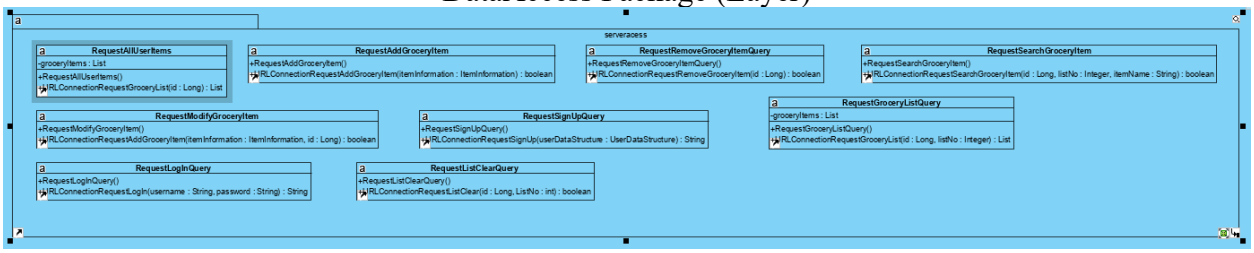
- Abstract Factory → Used in the creation of Weekly/Monthly Reports
  → Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- Layered → used to structure the program into groups of subtasks
  → each layer provides services to the next higher layer
- Observer Pattern → Notify the user if some item is close to consumption date
- Mediator Pattern → separate requests from responses (reduce coupling)
  → makes implementing and modifying functions easier
- Decorator Pattern → separate the result writing of a bad or good calorie intake

5.2 UML Class Diagram

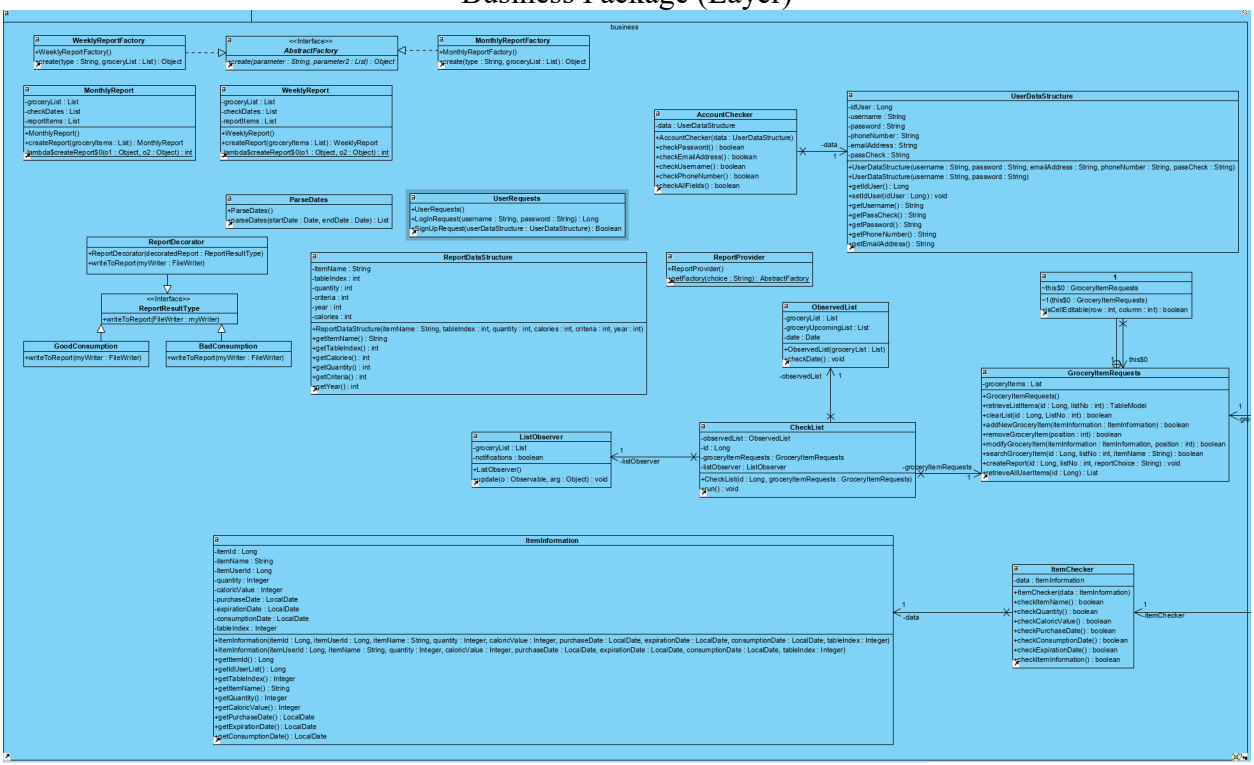
CLIENT



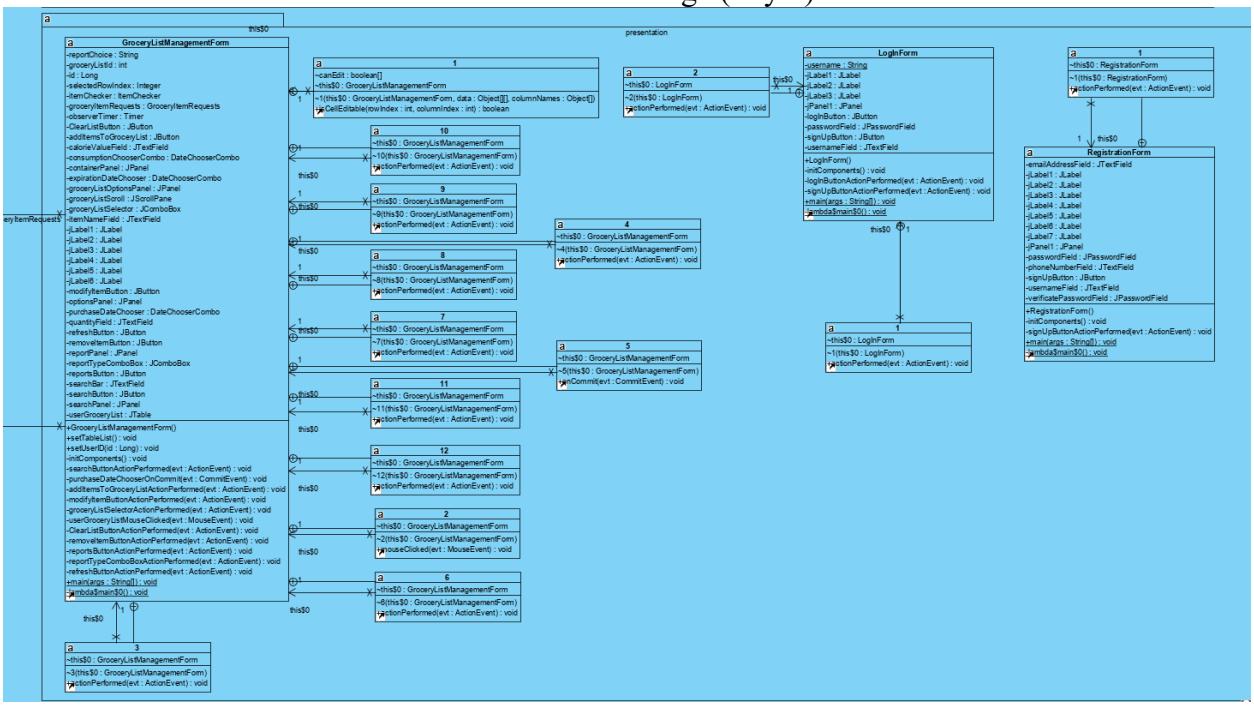
DataAccess Package (Layer)



Business Package (Layer)

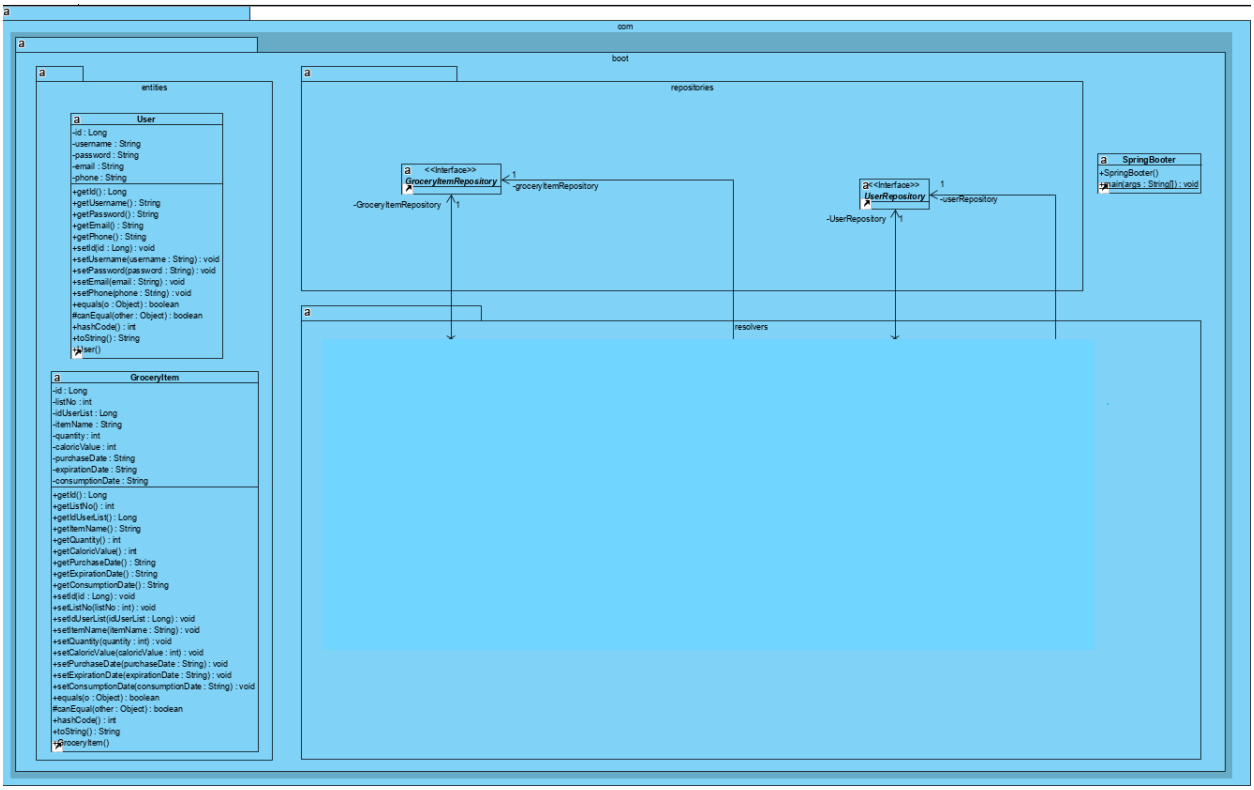


Presentation Package (Layer)



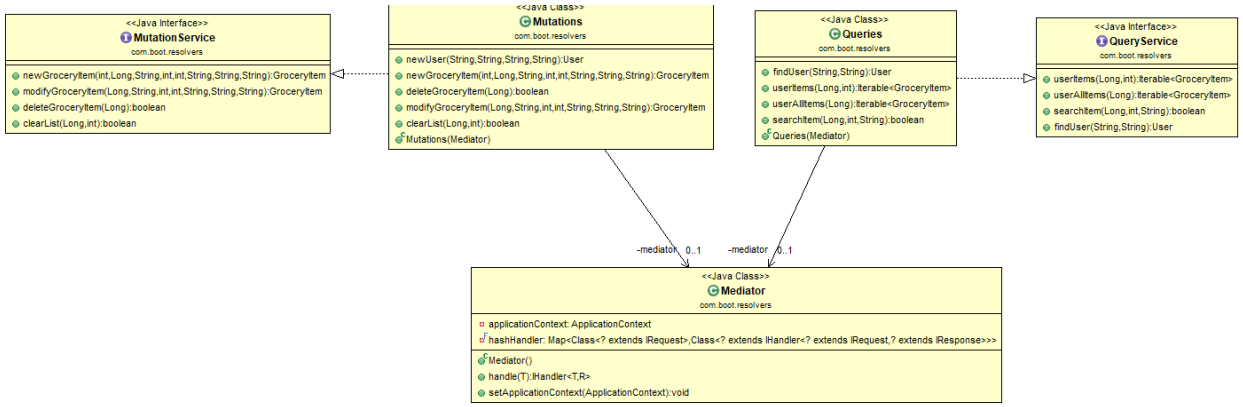
As it can be seen from the 4 images the structure of the program is devided into the three parts mentioned at the architectural pattern. This separates the data flow into 3 sections which makes it easier to troubleshoot and modify.
Abstract Factory is used to generate Factories to create reports based on the user's preference. This is a grouping of individual factories that have a common theme but no concrete classes. It separates the details of implementation the sets of objects WeeklyReport and Monthly Reports from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface .
Observer Pattern is used to notify the user if there are any items that are closed to the consumption date in his lists.
The Decorator pattern is used to differentiate how the program writes to the report how is the user's calorie intake.

SERVER

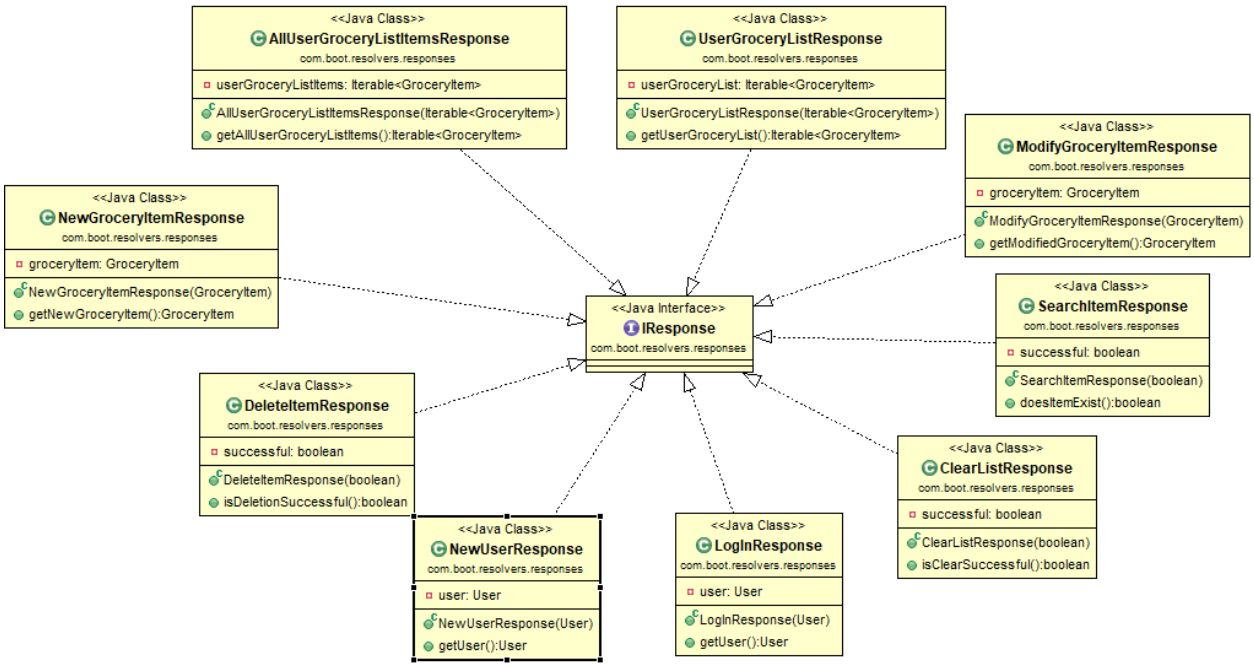


(Empty space in resolver package stems from some issues regarding Visual Paradigm. The following diagrams are what should've been there.)

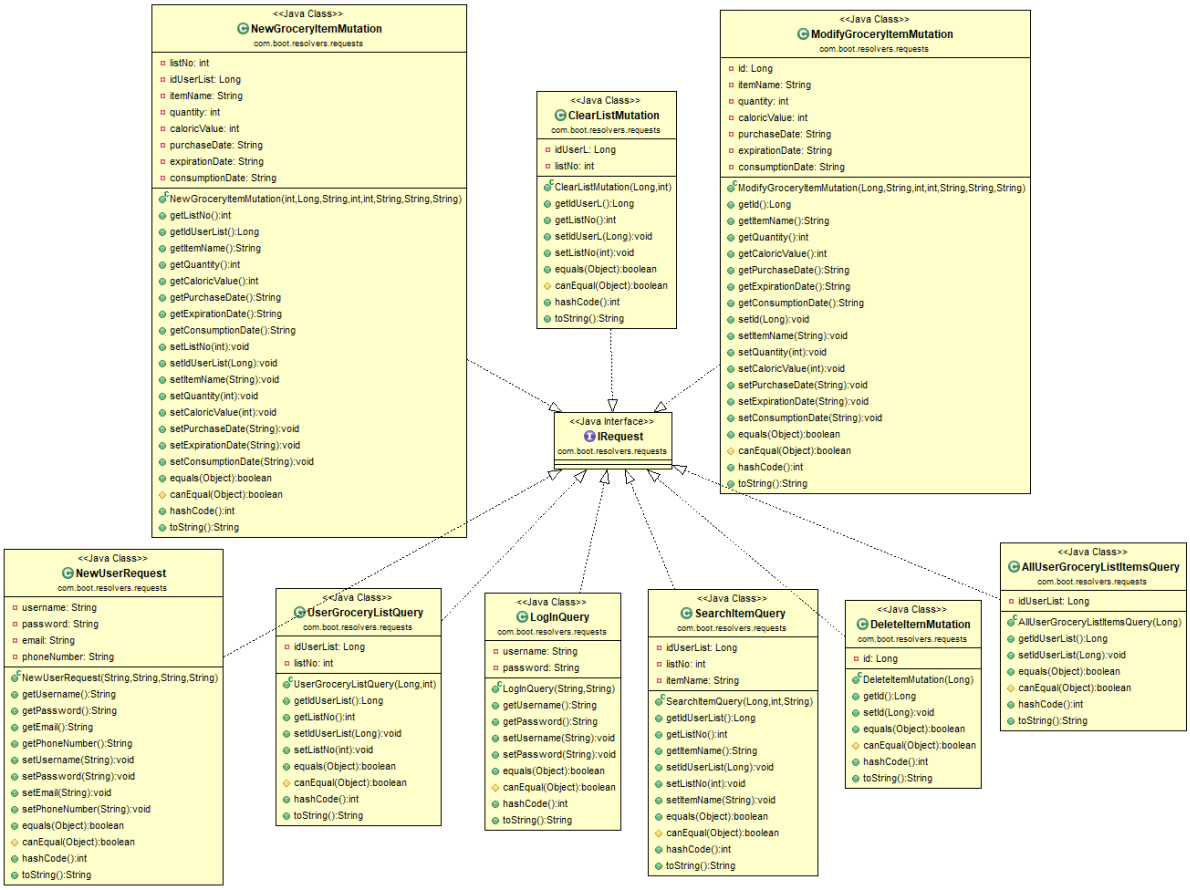
Server resolvers (Package)



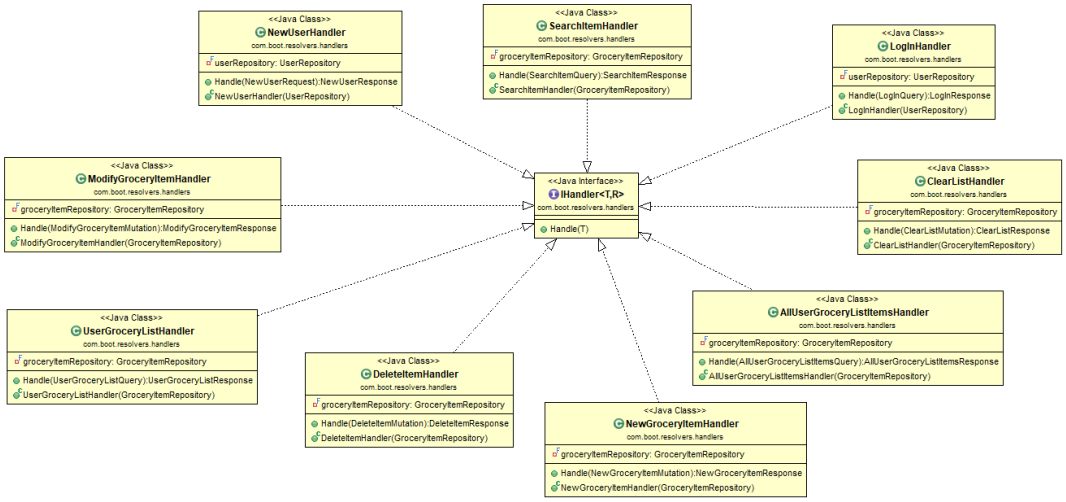
Server resolvers.responses (Packge)



Server resolvers.requests (Package)



Server resolvers.handlers (Package)



The server is devided into 3 main parts:

- entities – Contains the structures of the items (Contains the columns of the sql table)
- resolvers - Contains the functions that work with the database
  - This was modified to work though a mediator pattern
    - Used to separate requests and responses
    - The system receives the request (creates one for the mediator)
    - The mediator find it and activates the necessary handler
    - The handler works with the request and creates a response
    - The response is then send to the user
- repositories – Contains the list of items based on the entity data an id type

The server knows how to connect to the database based on the parameters in the application.properties file. Connection to the database is established by the spring framework + graphql API through hibernate.

6. Data Model

Relational Model for Database information:

- This model is based on first-order predicate logic and defines a table as an n-ary relation.
- Data is stored in tables called relations.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Entity-Relationship Model in DatabaseMS:

- Entity – is a real-word entity having attributes
  - attributes are defined by a domain
- Relationship – Mapped entities (in our case one to one (ex: PrimaryUserKey to PrimaryGroceryListKey))

7. System Testing

The testing method used was Unit Testing:

- Program was tested step by step through snippets of code
- Input data is verified by means of two classes AccountChecker and ItemChecker which look for bad inputs by seeing if they are in certain intervals (boundary analysis)
- Data Flow testing has been used in several areas to check if there's a proper flow (report making, retrieving Grocery List data). Here the data was checked to be:

  → properly defined in place

  → used properly (completely) for their usage
- The functions that were implemented in the server were tested using the graphiql library

- using the link: http://localhost:8080/graphiql you are sent to an interface
- in the interface graphql queries can be tested

8. Bibliography

[1] www. stackoverflow.com → Used for Explanation of:

- Email checking
- Abstract factory (?)
- Parsing dates

[2] https://www.baeldung.com → Used for Explanation of:

- Spring
- HTTPConnection Explanation

[3] https://github.com → Used for:

- GraphQL Documentation
- HTTPConnection Example Explanation

[4] Design Patterns - Decorator Pattern
https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm