<Wasteless Assignment no. 1>

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

Wasteless is a software application developed in Java , using the Eclipse IDE and Spring Boot tools. 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 calories value, expiration date , quantity*
* *Visualizing all the products that have been purchased*
* *Having the ability to donate excess foods to local charities*
* *Setting a goal regarding the number of calories the user intends to consume daily*
* *Getting notifications when food waste levels are too high*
* *Using Hibernate to interact with the database*

# Non-functional Requirements

*Some of the non-functional requirements that have to be mentioned here are data integrity (through the usage of validators ), extensibility, maintainability,scalability and testability. Of course, the usability can be also added to the list .*

*The application does not provide a lot of portability (since it is developed in Eclipse IDE) .*

2. Use-Case Model

*Use case: create new list & add products*

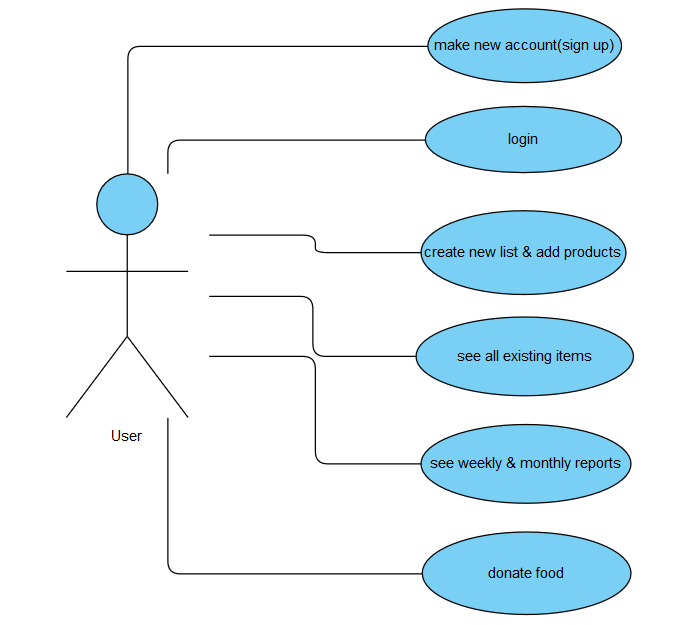
*Level: user-goal level*

*Primary actor: user*

*Main success scenario: The logged in user chooses from the main menu the button which directs him to adding a new list. The first thing that he will be asked to do is entering a suitable name for the newly created list. Then he will be able to add one product at a time , specifying for it the name , quantity, caloric value, purchase and expiration dates. While adding items to the list , he is able to see all the current products that are already there. Once the user is done , he can than save the newly created list with all the selected items.*

*Extensions: The product data that is entered is validated before it is added to the list. If quantities or caloric values smaller than 0 are entered, or if the purchase date of the item is not prior to the current date( the day in which the data is entered) the user will be asked to recheck the values and the product will not be yet added.*

*]*



3. System Architectural Design

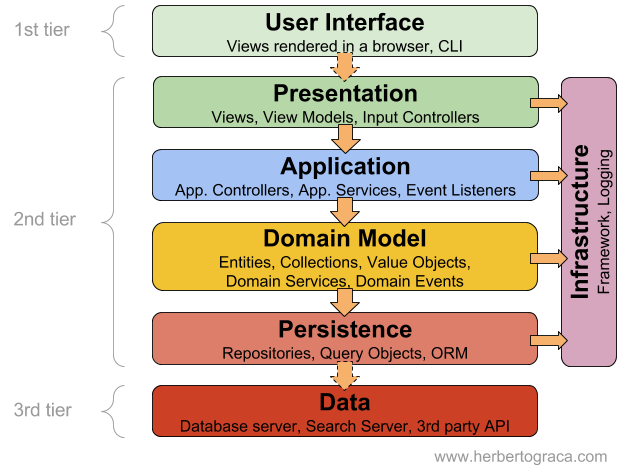
**3.1 Architectural Pattern Description**

*For the development of this application the layered architectural pattern has been used. This approach provides a level of abstraction to each group of subtasks that is developed. And so, each layer of the application is performing a specific role (components within a specific layer only deal with logic that depends on that layer).*

*The system is organized accordingly to the chosen pattern, as it consists of three layers: presentation ( for handling all user interface communication logic), business (validation of data and other specific business rules depending on the requests) ,as well as the data tier (where all the repositories and entities are defined ).*

**3.2 Diagrams**

*Conceptual architecture of the system:*

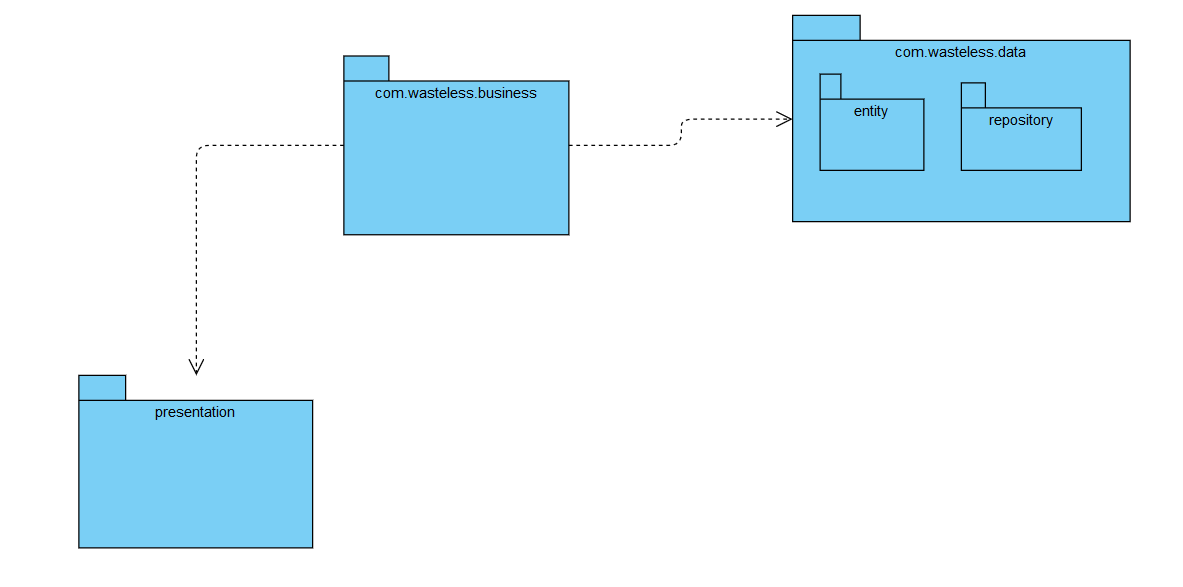


It can be observed how the layered architecture is presented in the previous image , as all the tiers are contained . The way in which they are applied has been briefly described previously. One thing to mention , anyway, is that the persistence is performed using an ORM approach (with Hibernate ) and , given that it was required to use a DI container, the Spring framework imposed a little structure of its own.

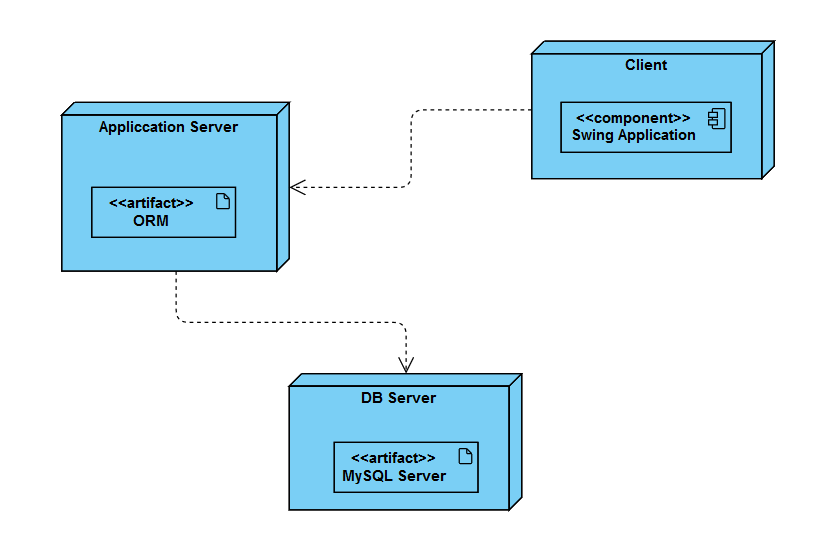
So, the presentation layer deals with the user interface . It contains the frames of the application , alongside controllers which are solving the requests given when the user is pressing a certain button. The business layer uses validators to check the integrity of the entered data and , moreover, it deals with certain business logic concerns ( burndown rate calculator , the service classes of the entities, the factory classes ).

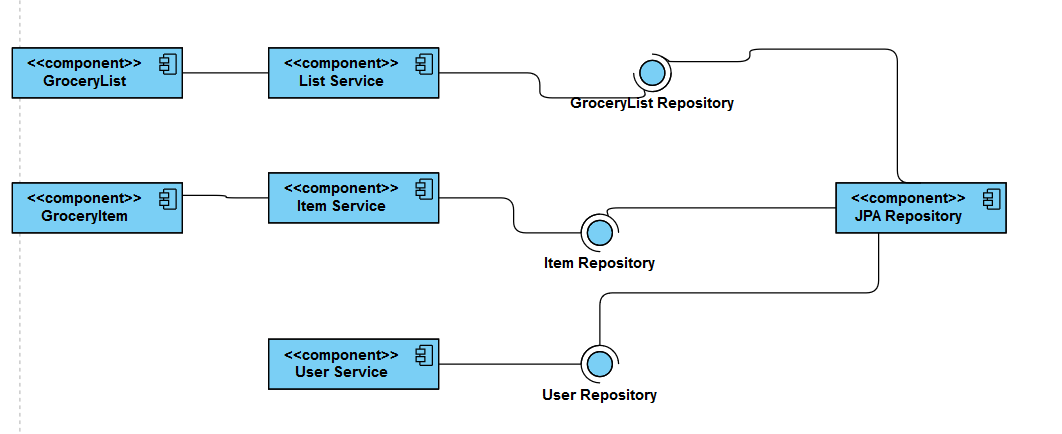
The data tier is basically split into two parts, as it holds both the entities that will make the migration of data from/to the database easier , as well as the repositories that hold the methods necessary for this communication. This approach has been used because I wanted to develop my application with the aid of Hibernate and Spring Boot.

Package :

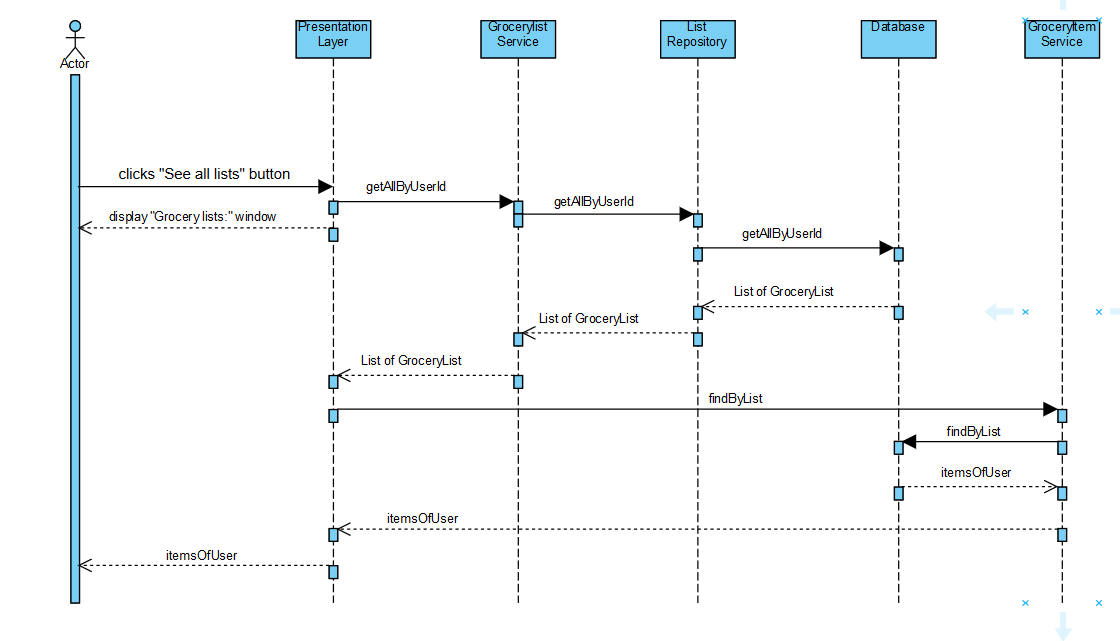


Deployment :

**



4. UML Sequence Diagrams

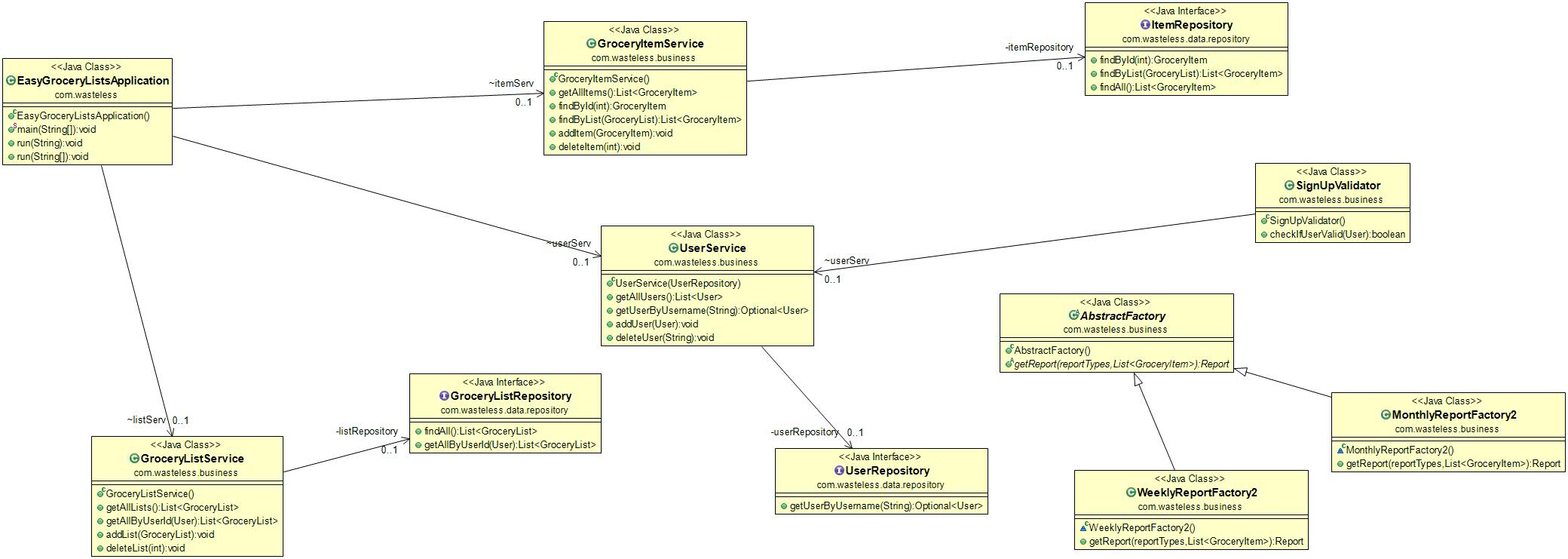


5. Class Design

**5.1 Design Patterns Description**

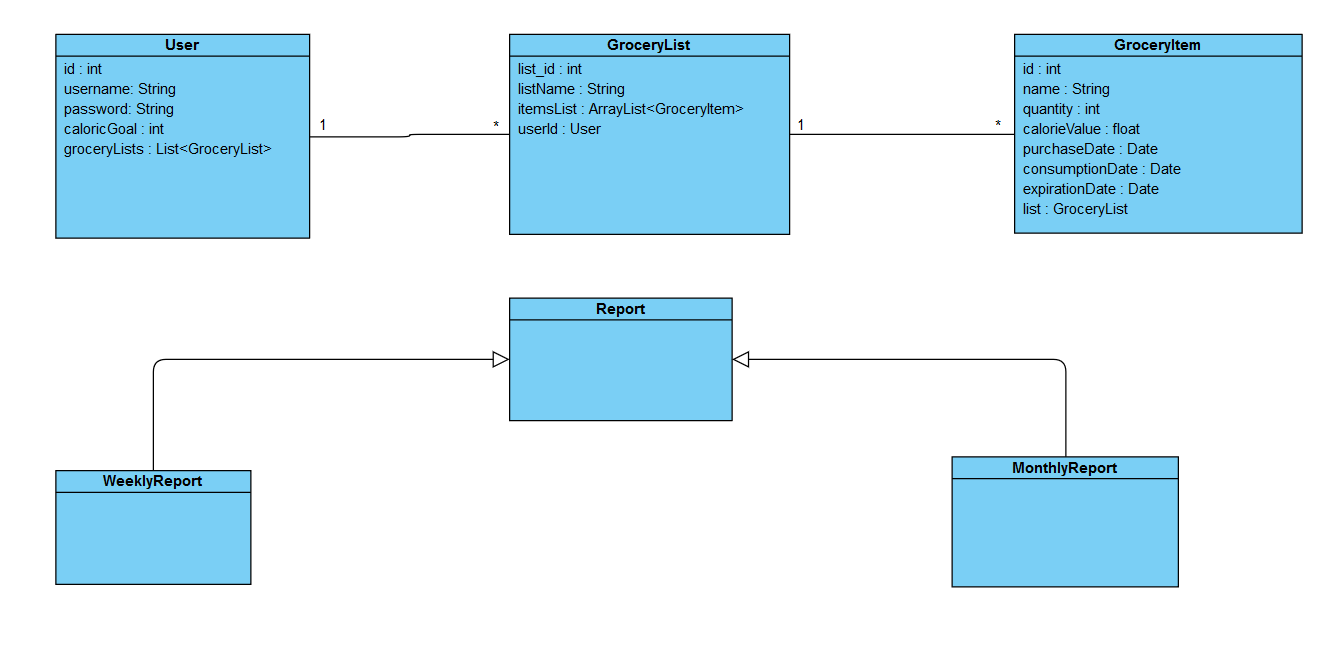
*For this application ,the Abstract Factory Pattern has been used in order to generate weekly and monthly reports regarding food waste. This is a creational pattern , as its main focus is providing an easier way of creating objects. An interface is responsible for creating a factory of related objects 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 patterns makes adding new types of objects easier.*

**5.2 UML Class Diagram**



Because the class diagram was to complex, I chose to put aside some of the classes( mainly the ones from the presentation layer and a few from the service). The most important ones can be still seen on the above diagram . The implementation of the abstract factory pattern can be observed from the image . A abstract class , Abstract Factory, is extended by two smaller factories (namely Weekly and Monthly Report factories) . Another class, called ReportCreator, has the task of choosing which factory must be created, making the differentiation between the weekly and the monthly reports based on an enumeration that is defined in the same package.

6. Data Model



*Above we can see the main entities present in the application. The user, which has an account , can add multiple GroceryList objects .Moreover, every grocery list will store grocery items . The fields of all the three models can be seen in their class representation . Apart from that, three more structures are useful , namely the Report , from which two types of ‘sub-reports’ are extended (weekly and monthly).*

7. System Testing

*The system has been tested and retested for each one of the operations . Invalid data has been entered in the login , new product , consumption and goal sections to make sure that the system performs accordingly . When it comes to operations which regard data handling , the database has been constantly checked to observe whether the requests have been performed successfully .*

8. Bibliography

<https://www.callicoder.com/spring-boot-rest-api-tutorial-with-mysql-jpa-hibernate/?fbclid=IwAR1qyY52KYOGP_FZScXEC_yEXKuqu6-qNHDcDk9l7yIhqLtQZwhGPpeyudY>

<https://www.baeldung.com/hibernate-one-to-many>

<https://medium.com/java-vault/layered-architecture-b2f4ebe8d587>