WasteLess

Analysis and Design Document

Student: Roxana-Ioana Aldea

**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 client-server architecture based application designed in order to reduce food waste by allowing users to track their groceries and set goals based on the ideal burndown rate. First of all, the system is provided with a login system in order to allow only registered users to enter the application. In order to authenticate, the user has to introduce its credentials, meaning username and password. The user is able to create a grocery list by providing a name or to select an existing list and choose to edit it. Edit list means add items or donate them. 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 should provides the user with options to donate excess food to various local food charities and soup kitchens.

One important feature of this application is that each time the user logs in, he/she is notified about the items which expire soon, this way, those items can be consumed, reducing food waste.

# Functional Requirements

A user can authenticate by providing the right credentials (username and password) and after that it can:

* View, create, edit a new grocery list
* View, create, donate grocery items
* Set a goal to minimize food waste
* Generate reports

# Non-functional Requirements

The data should be stored in a database

The system architecture should be a layered one

In order to generate two types of reports, the abstract factory design pattern should be used

The data access should be implemented using ORM

The dependencies should be resolved using a DI containter

1. Use-Case Model

Use case: Add Item

Level: user-goal level

Primary actor: user

Main success scenario: The user is allowed to add new grocery items to a newly created list or to an existing list. It has to introduce some features as (name, quantity, calorie value, purchase date, expiration date and consumption date). If the user provides valid data, for example positive values for quantity and calories and valid consumption date, not before the purchase date or after expiration date, the new item will be added in the database.

Extensions: In case the user introduces invalid data, as I have said above, bad values for quantity, calories or consumption date, an error message will appear on the top of page.

*O imagine care conține desen

Descriere generată automat*

3. System Architectural Design

**3.1 Architectural Pattern Description**

**3.2 Diagrams**

**As required in the assignment statement, the application uses a client-server architecture. The server consists in a Java Application which uses Spring Boot framework. The client was implemented in TypeScript, using Angular. The requests sent by the client from browser are handled using the Controller classes from Java project which are linked to the Angular service files. The server used by the application is Tomcat, which is started automatically by Spring Boot. Because the client side has been developed in Angular, it consists in more components structured in 4 files: a template, HTML file, style files, CSS files and 2 Typescript files where the logic is written. Also, there are model classes such as grocery-item, grocery-list, goal etc. The server side, being developed with Spring has its specific structure of packages: controller, service, repository and configuration.**

**O imagine care conține captură de ecran

Descriere generată automat**

**O imagine care conține captură de ecran

Descriere generată automat**

O imagine care conține captură de ecran

Descriere generată automat

1. UML Sequence Diagrams

**Sequence Diagram for insert item use-case**

O imagine care conține captură de ecran

Descriere generată automat

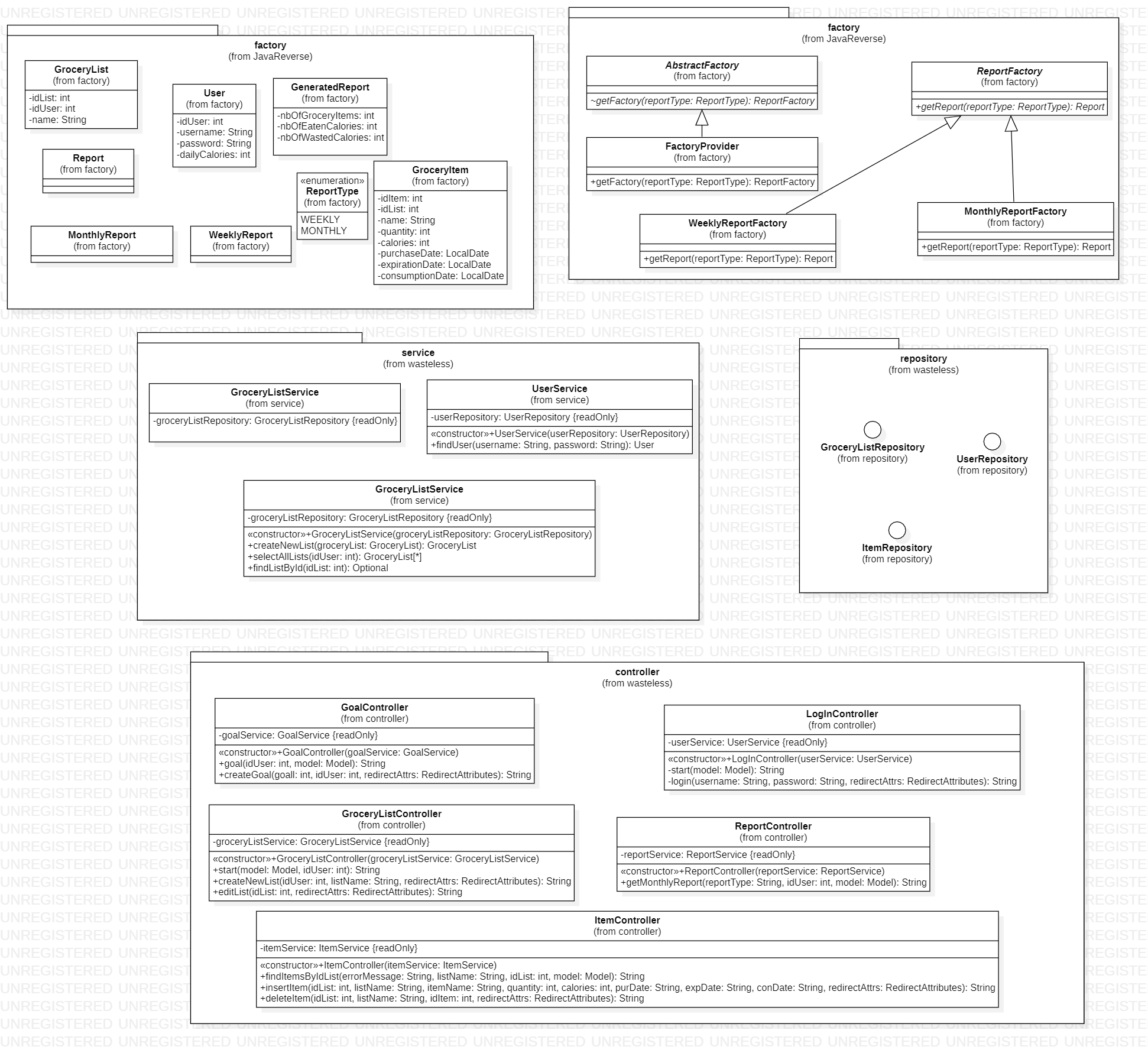
5. Class Design

**5.1 Design Patterns Description**

The application uses Abstract Factory Design Pattern in order to generate weekly and monthly reports. There are two types of Factories, a WeeklyReportFactory which instantiates WeeklyReport type, and a MonthlyReportFactory. Both Factory classes extend ReportFactory because have a common method. A FactoryProvider chooses which ReportFactory to instantiate depending on the report type, which is an Enum containing two values: WEEKLY and MONTHLY. Each ReportFactory subclass has a method, getReport which instantiates a Report type, depending on type, WeeklyReport or MonthlyReport, which extend the abstract class Report. Report class provides some methods for computing the features for GeneratedReport, one class which stores all the features which will appear in the presentation layer.

The Observer Design Pattern has been used in order to notify the user about the grocery items which will expire soon. Every time the user successfully logs in, on the main page, is able to see which items has to consume, in order to reduce the waste. The implementation of this DP consists in having 2 interfaces, Observer and Subject and 2 classes which implement them, Observer Handler and Subject Handler. The last 2 classes are instantiated in the Main Component, after the user logs in. The grocery items which have to expire are provided by Grocery Items Service.

**5.2 UML Class Diagram**



6. Data Model

O imagine care conține captură de ecran

Descriere generată automat

7. System Testing

The application was tested using Junit framework. For the components which use other components, I have been used Mockito to mock the functionality of that specific component. There are just some tests, for GroceryListController. The Service component has not been tested because uses Repository interface which extends Jpa Repository and as it is known, using Spring Boot eases very much the writing of queries and they are done by default using the model classes, mapped through ORM to the database entities.

8. Bibliography

* <https://stackoverflow.com/>
* <https://www.callicoder.com/spring-boot-rest-api-tutorial-with-mysql-jpa-hibernate/>
* <https://spring.io/docs>
* <https://angular.io/guide>