WasteLess

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

Design and implement an application that helps users manage food waste.

Once a user is authenticated, he can input grocery lists and see reports of how much food is wasted weekly and monthly. 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.

# 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 layered architecture. To be more specific, a 3-tier layered architecture, split in data access, business logic and presentation layer, on large scale. Due to the fact that it is a Web Application and uses Spring Framework, the application is also split in Spring’s specific components: Repository, Service and Controller. The Repository package, together with the Model package form the data access layer, the layer which interacts with the database. The business layer consists in the Service annotated classes and in the Factory package which handles the generating of reports.**

**The presentation layer consists in the Controller package, which contains Controller annotated classes which handle the requests coming from the user.**

**Also, in the presentation layer could be considered the JSP files but due to the Spring Boot imposed organization of packages, the .jps files are found in the webapp/WEB-INF/jsp package. The requests sent by the client, from browser are handled using the Controller classes which are linked to the jsp files which make the result visible to the user. The server used by the application is Tomcat, which is started automatically by Spring Boot.**

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

**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

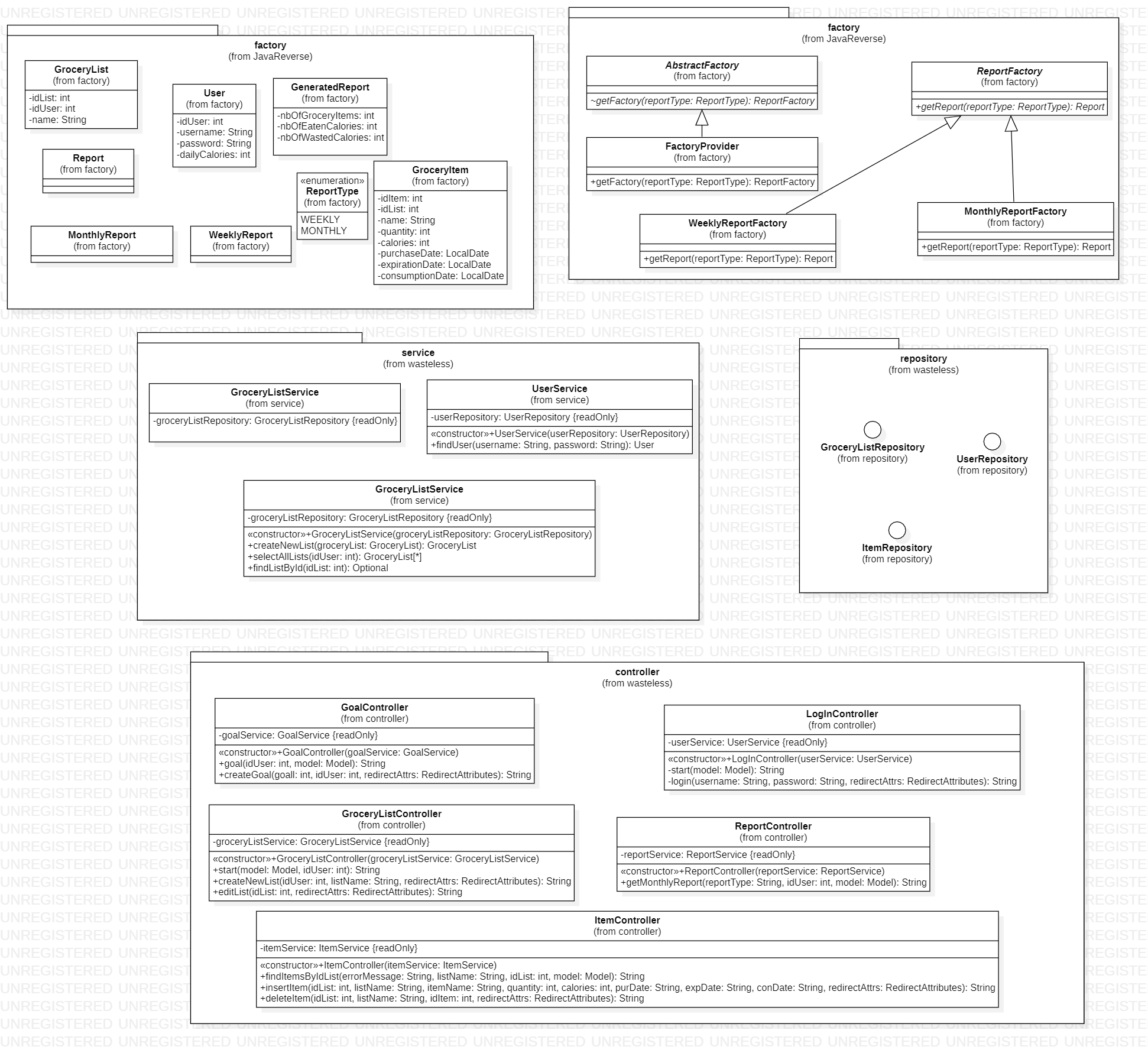
4. UML Sequence Diagrams

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.

**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. Because of low coupling and of DI container, each component could have been tested easily. For the components which use other components, it has 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>