WasteLess

Analysis and Design Document

Student: Socaci Radu Andrei

**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

The application should authentication (basic in this case). In order to access the application, the user needs to provide valid credentials (username/password).

Once authenticated, the user can perform the following operations:

* Perform CRUD on grocery lists (view, create, delete, update)
* Perform CRUD on specific items inside a grocery list
* Generate reports (monthly and weekly reports)
* Donate excess food to charitable organizations
* Set a goal and get notified whenever the number of calories is exceeded.

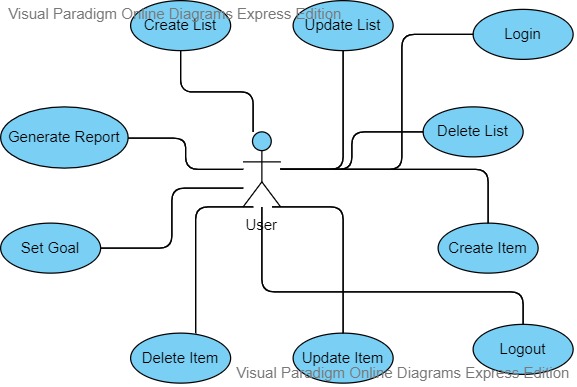
# Non-functional Requirements

The app did not impose any development restrictions, from a framework/technology point of view. However, a few non-functional requirements still aroused:

* The data should be persisted in a database
* The application should use a Layered architecture
* The data access level will be should be implemented using an ORM
* Perform input validation before saving the data into the database
* Use the abstract factory pattern for creating weekly/monthly reports

Moreover, the app has other typical non-functional requirements, such as: availability, performance, ease of use, etc.

2. Use-Case Model



Use case: Login

Level: User-goal level

Primary actor: The user

Main success scenario:

1. The user provides a username/password combination when prompted to log in
2. If the credentials are not valid, the user will be notified, and the flow will stop
3. If the credentials are valid, the user will be redirected to the home page

Extensions: In case of failure, the user will be able to log in again, with different credentials

3. System Architectural Design

**3.1 Architectural Pattern Description**

The application uses a 3-tier Layered architecture. It is 3 layers, data access, business logic and view. However, Spring Framework was used, and taking into considerations Spring best practices, I decided to further split the application, adding a Service layer, splitting the data access layer into Repository and Model packages. Also, part of the business logic layer is the Factory package, which hold all the logic related to creating reports.

The version of the application presented here does not use a client-server architecture, which means that static HTML is served directly by the server running the back-end application (Tomcat) and the content of the HTML files is not dynamically modified. An extra feature provided by spring was used, called Thymleaf, which was used to pass the context inside HTML files. For this reason, the view layer is split between the static HTML files and the controllers, which map a request to the correct static resources.

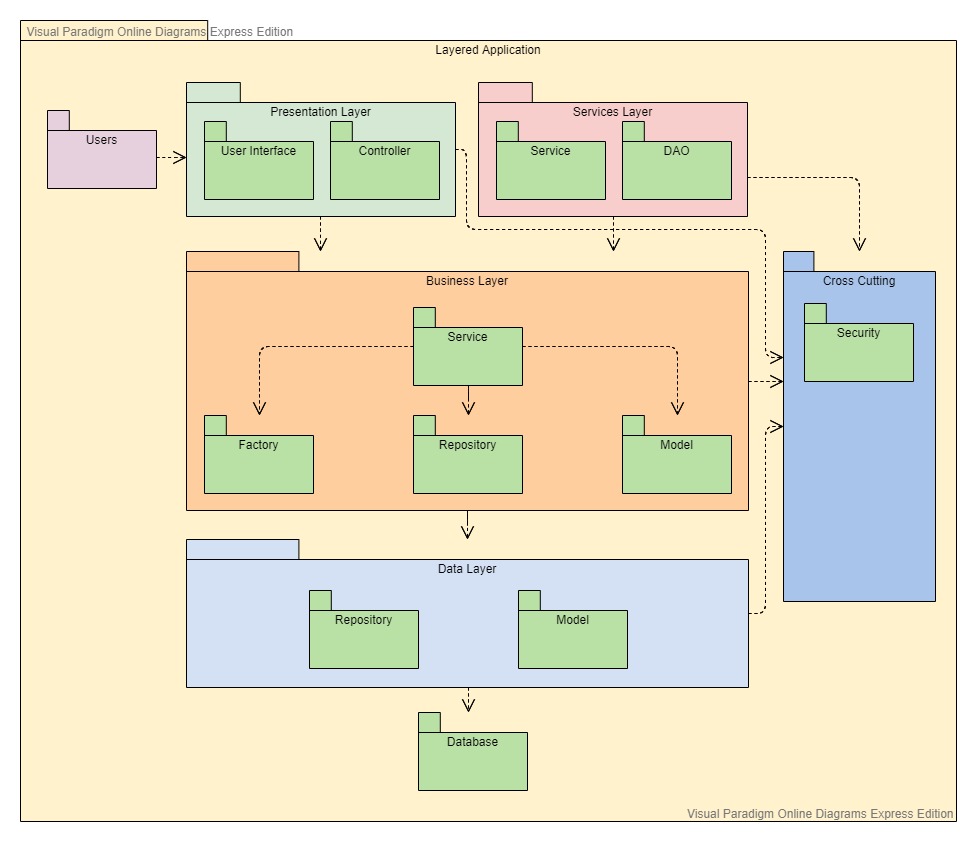
Other design patterns used were embedded into the data access layer, since they closely relate to the domain model classes (Builder and Observer).

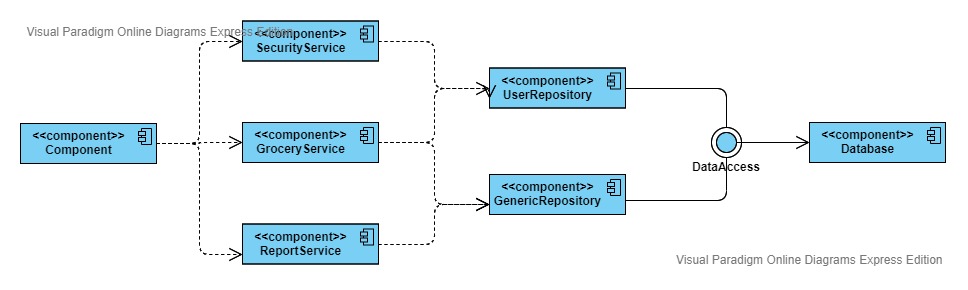
**3.2 Diagrams**

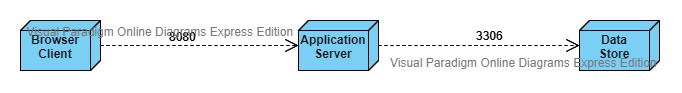
The presentation layer will contain a package for the controllers of the application and a package for other useful classes for the views. The presentation layer depends on the DTO (Data Transfer Object) classes and on the business layer.

The business layer package contains packages for the services, where logic and validations of data are implemented, and another package where the factory design pattern is implemented for generating Reports. The business package also depends on the DTO package, and as well on the model package, where there are classes for each model (of each table in the database).

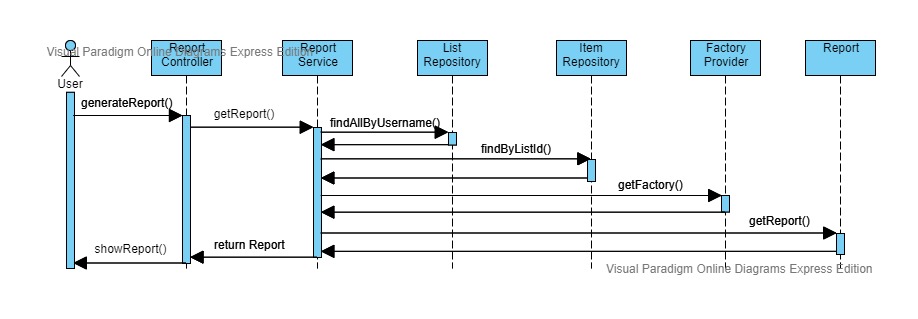
The data access layer contains the repositories used for obtaining data from the database and performing CRUD operations on them. It depends as well on the model package, with its respective classes, since these directly map to tables in the database.







4. UML Sequence Diagrams



5. Class Design

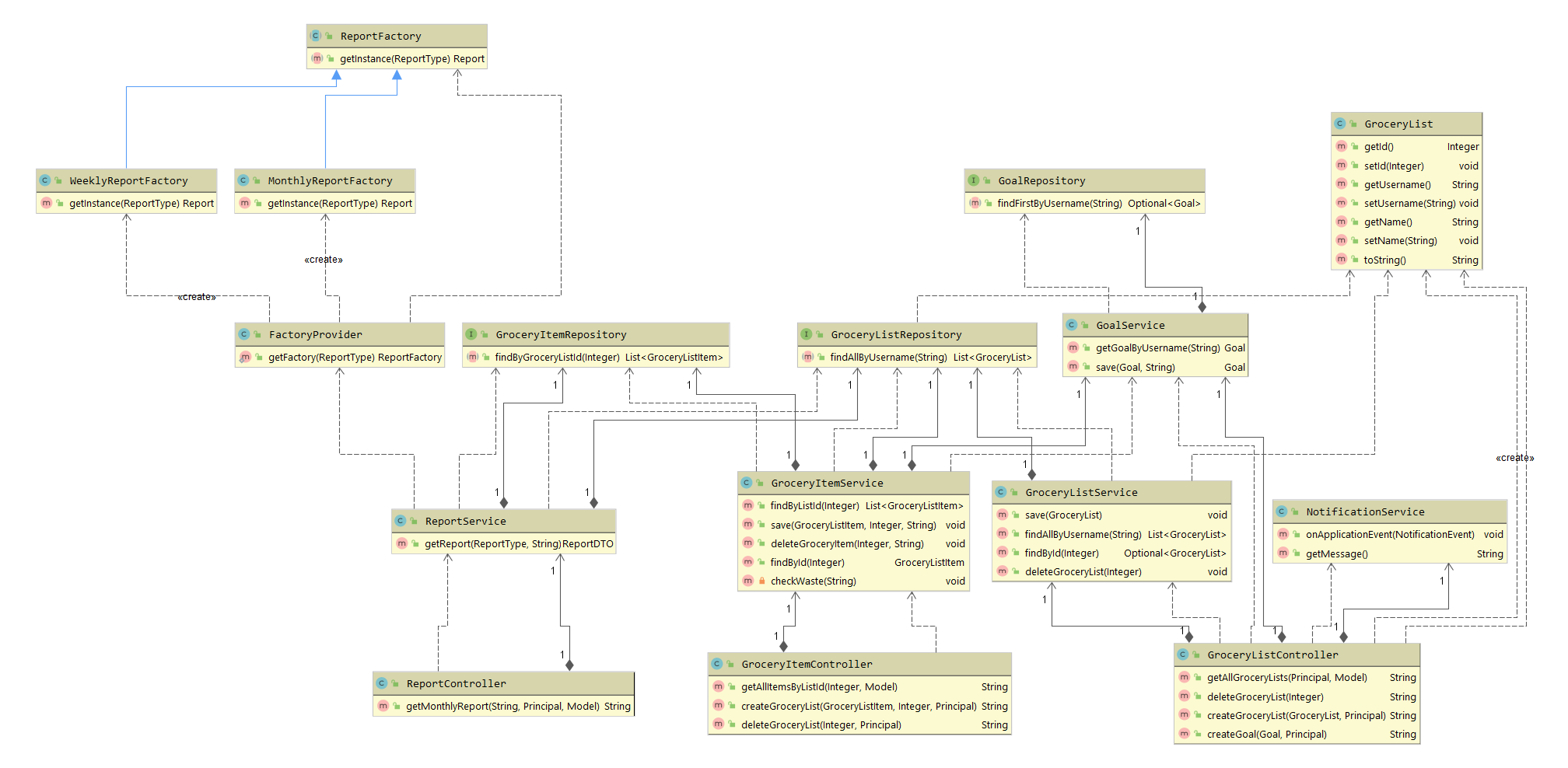
**5.1 Design Patterns Description**

First, the Abstract Factory design pattern was used for report generation. There is a factory provider, which instantiates a factory, based on the request parameter given. After the correct factory was instantiated, the factory returns an instance of type Report, which can either be a MonthlyReport or a WeeklyReport. These in term extend from an abstract class, Report, which was used to encapsulate the common functionality, to avoid code duplication. A ReportDTO is used for communication with the view layer.

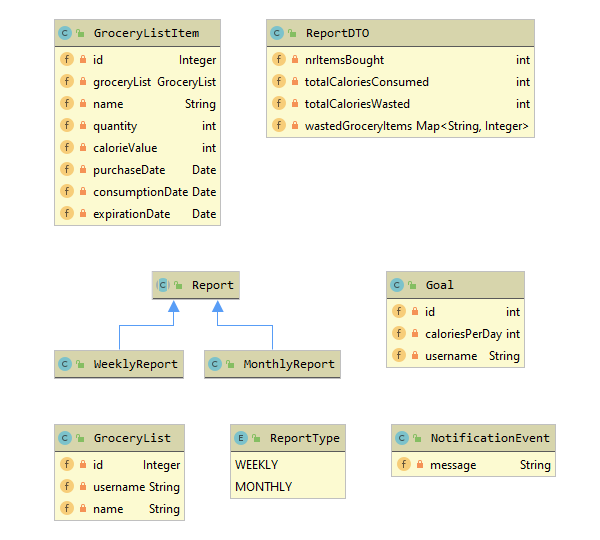
Second, the observer design pattern was used for reminders (called notification in the app). Spring does not allow for a classic implementation of the Observer design pattern. It is possible, by predefining the beans (subscriber) and subscribing to the observable. However, this approach is both error prone and not recommended. A better implementation can be achieved by the use of Spring Events, which were used in this case.

To complement the report generation functionality of the application, the Builder design pattern was also implemented to facilitate the creation of ReportDTOs’.

**5.2 UML Class Diagram**



6. Data Model



7. System Testing

Testing was mainly conducted on service classes. The Unit Testing was implemented using tools like JUnit and Mockito. JUnit was used for implementing the unit tests and Mockito for creating mocks of the repositories.

Some of the test cases include:

* Save goal test: The goal repository is mocked. Any goal class given to it will return a predefined mock. We assert that the value of the username of the actual instance that was saved in the database is the same as the one returned by the service.
* Get goal by username: Given a username, the repository returns an optional of goal. We assert that the returned instance’s username is the same as the one given to the service in the initial query.

8. Bibliography

* <https://stackoverflow.com/>
* <https://spring.io/docs>
* <https://docs.spring.io/spring-security/site/docs/current/reference/html5/>
* <https://www.baeldung.com/>