<WasteLess - assignment 1>
Analysis and Design Document

Student: Anda Papita
Group: 30431

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The specification for this assignment is to 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. 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 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.

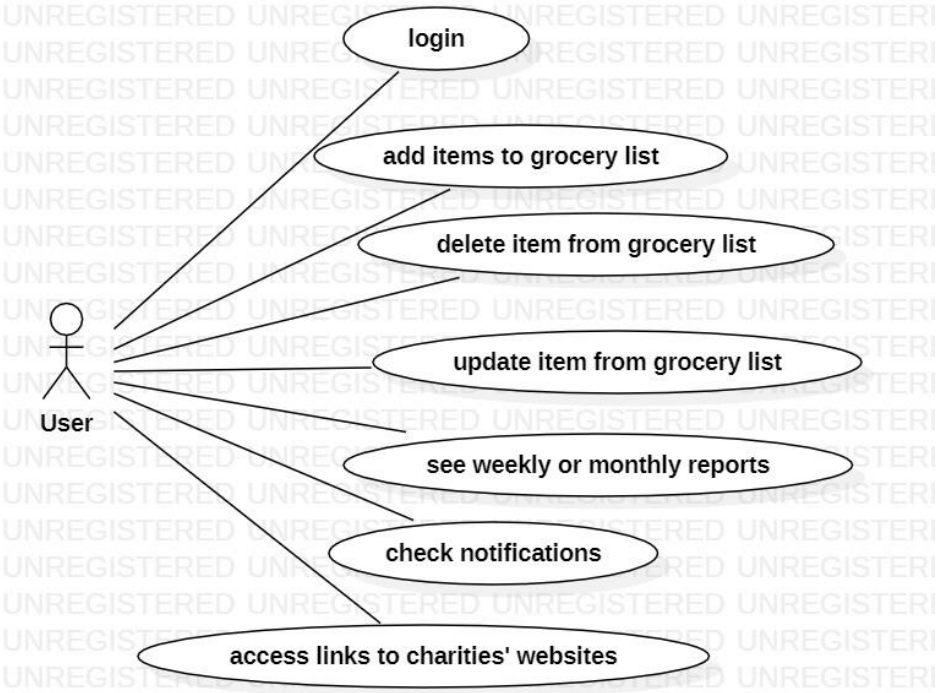
1.2 Functional Requirements

- The user is able to create a grocery list by adding new items
- The user is able to see reports of how much food is wasted weekly/monthly
- The user is able to track goals by following ideal burndown rates for each grocery on the list
- The user is able to see notifications and reminders about the best way to consume his/her groceries, reminders, waste alert etc.
- The user is able to access sites where he can donate his extra food

1.3 Non-functional Requirements

- Implement and test the application
- Use an ORM and a DI Container
- Commit the work you do on your Git repository. Do it iteratively as you progress, not all at once (this will incur a penalty on your final mark)
- Use any OOP language you like. Non-exhaustive: Python, C#, Java, Ruby, C/C++, JS+Typescript
- Use a layered architecture
- Use the abstract factory pattern for creating weekly/monthly reports
- The data will be stored in a database
- All the inputs of the application will be validated against invalid data before submitting the data and saving it in the database.

2. Use-Case Model



Use case: add new item to grocery list
Level: user-goal level
Primary actor: user
Main success scenario:

- Select option to add new item
- Input details of new item
- Save new item
- Redirection to main page after successfully adding the new item to the list

Extensions: failure scenario

- Select option to add new item
- Input details of new item
- Save new item
- Error message due to incorrect input details
- Possibility to try again

3. System Architectural Design

3.1 Architectural Pattern Description

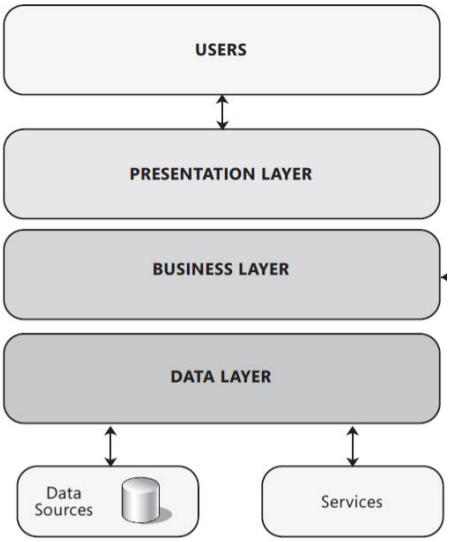
The architectural pattern which was used is the layered architecture pattern, the most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers, as does mine: presentation, business, persistence, and database. One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only

with logic that pertains to that layer.

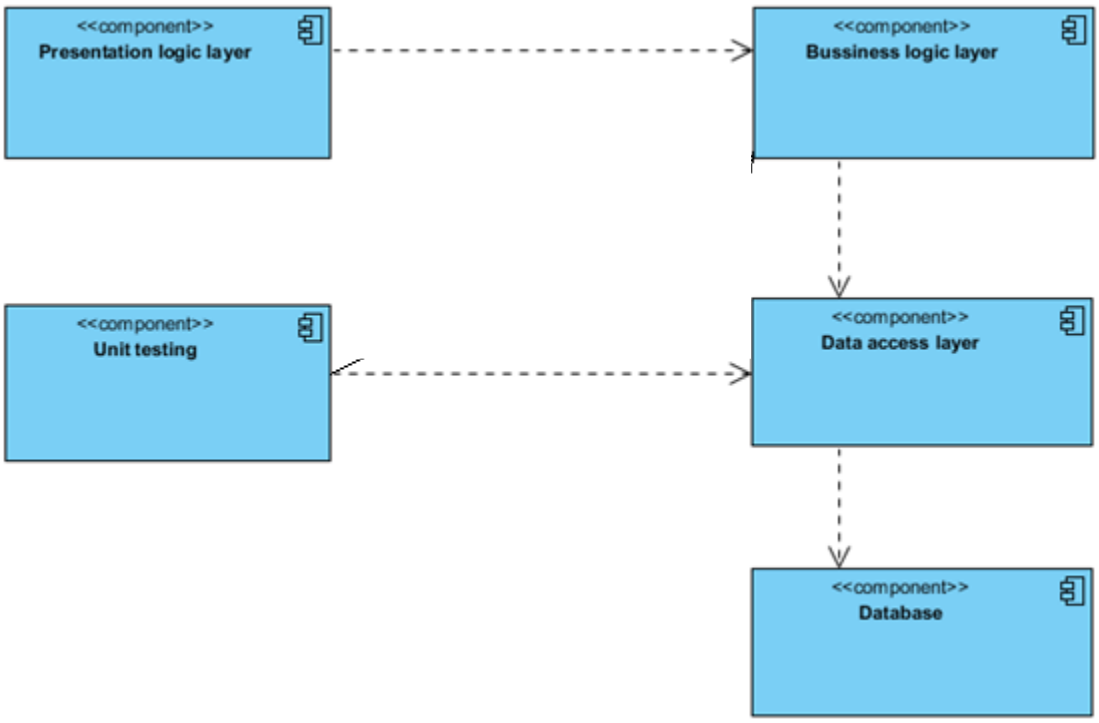
My application is a web application created with Java Servlet Pages (Jsp) and Tomcat Server, in the Eclipse framework. The database was created with MySql. I also used Hibernate for ORM.

3.2 Diagrams

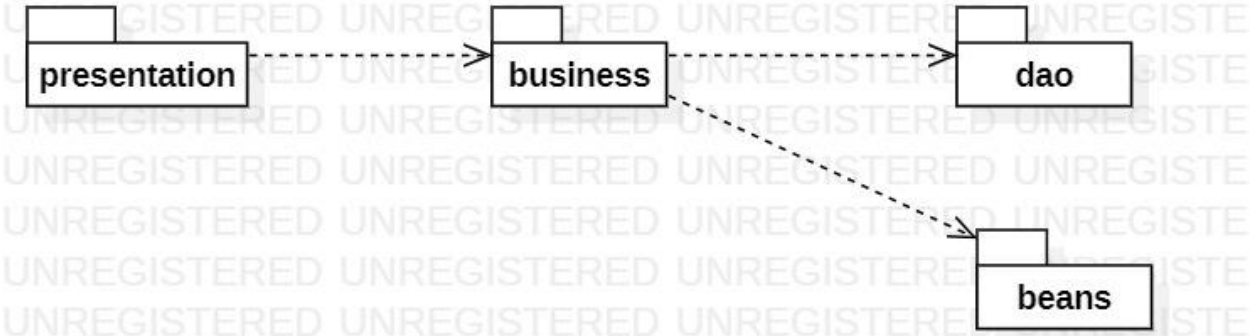
System's conceptual architecture:



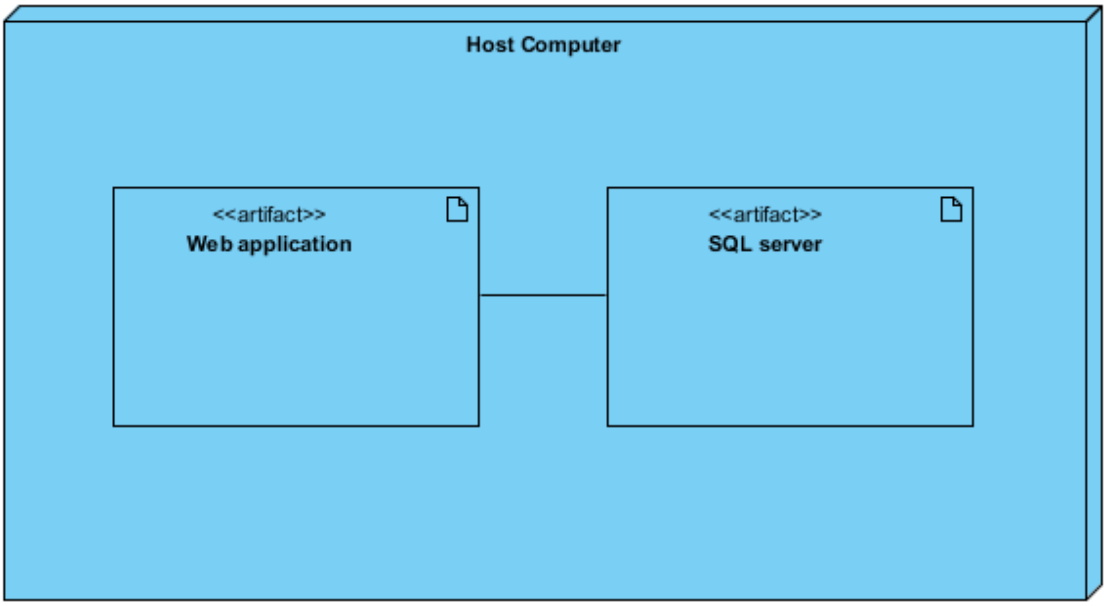
Component diagram:



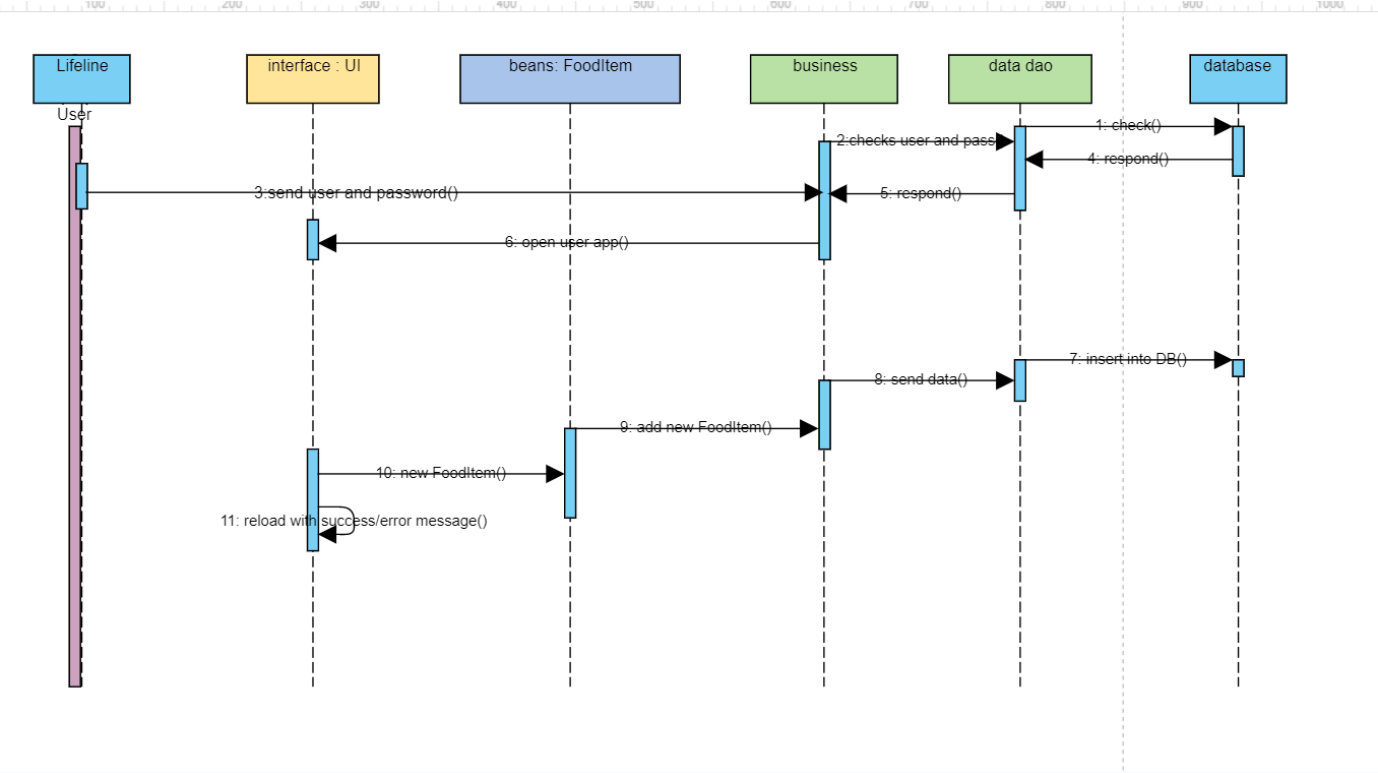
Package diagram:



Deployment diagram:



4. UML Sequence Diagrams



5. Class Design

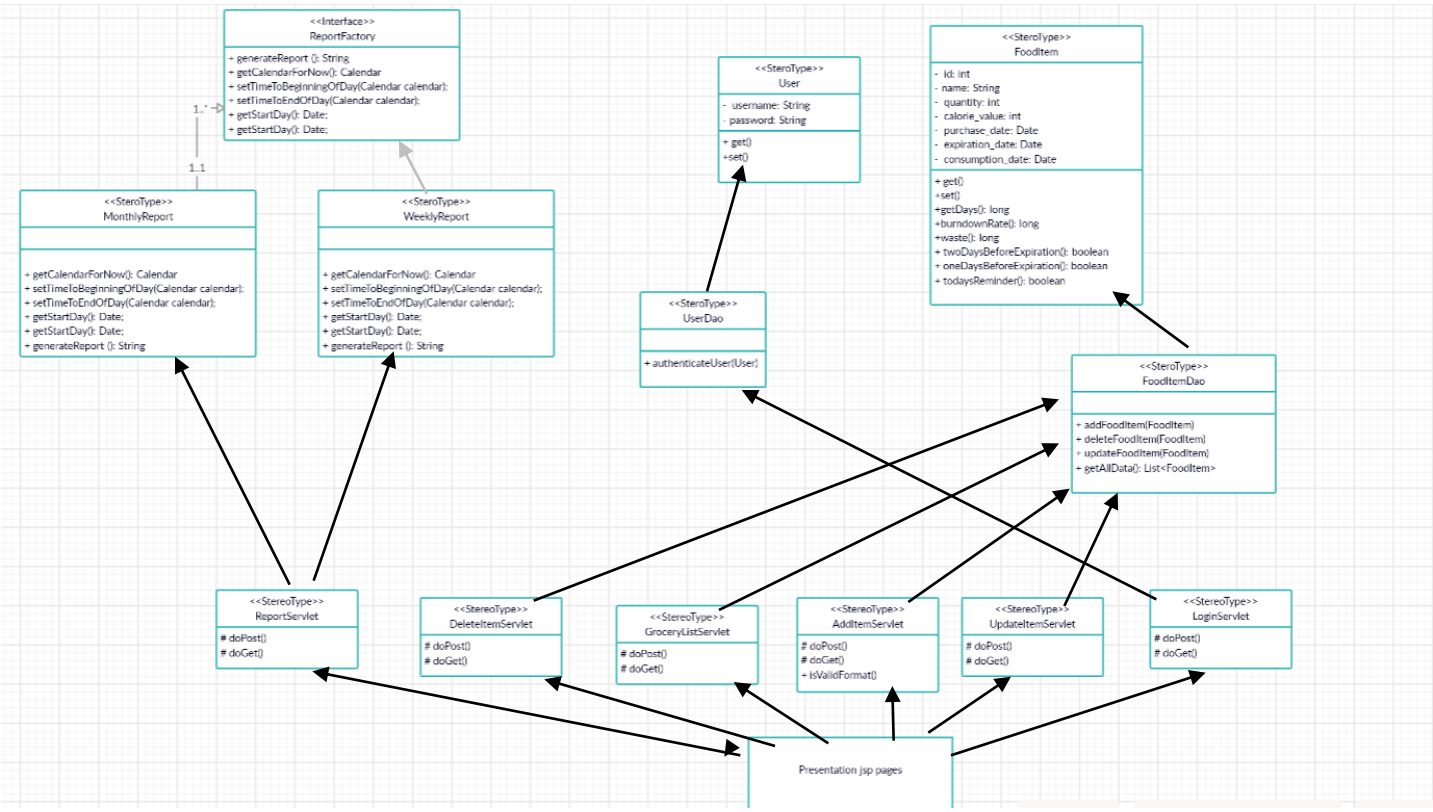
5.1 Design Patterns Description

The used design pattern is the **abstract factory pattern**, used for creating weekly and monthly reports. Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

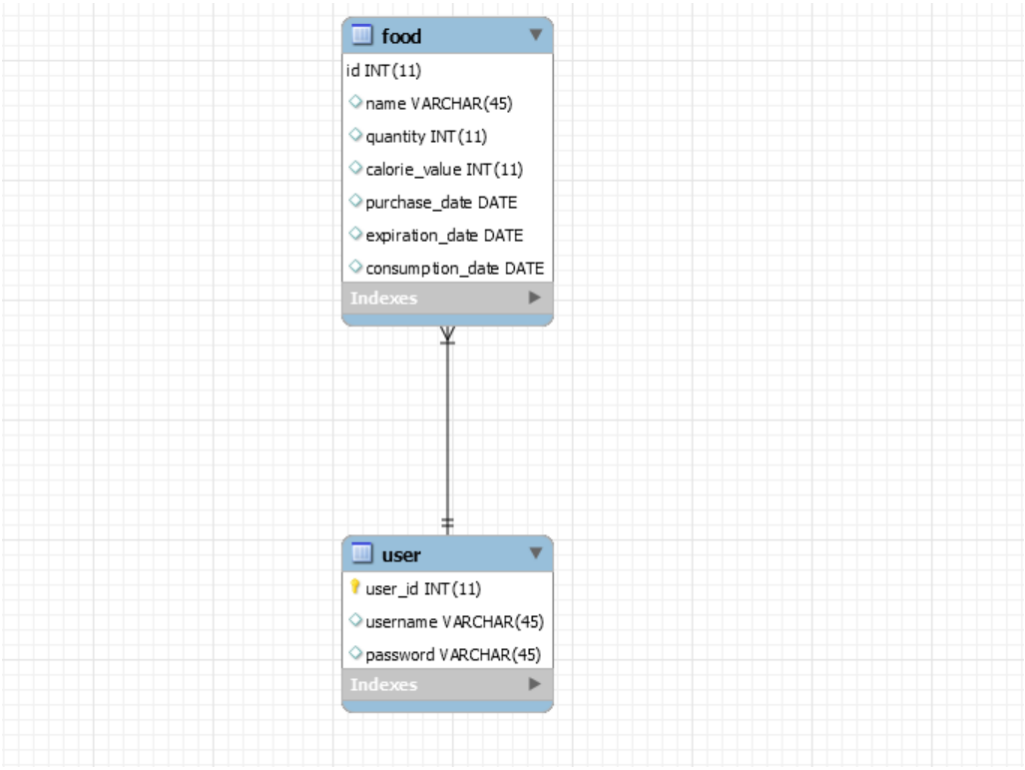
In Abstract Factory pattern an interface is responsible for creating a factory of related objects, in my case the class *ReportFactory*, without explicitly specifying their classes. Each generated factory (*MonthlyReport*, *WeeklyReport*) can give the objects as per the Factory pattern.

Moreover, for the design of this application I have used the **ORM**, a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. In this sense I have used **Hibernate**, is an object-relational mapping tool for the Java programming language. It provided a framework for mapping an object-oriented domain model (*FoodItem*) to a relational database (MySQL *waste* database). Out of the Hibernate provided operations, I have implemented CRUD (create, read, update, delete) on *FoodItem* objects.

5.2 UML Class Diagram


<<Interface>>
ReportFactory
+ generateReport (): String
+ getCalendarForNow(): Calendar
+ setTimeToBeginningOfDay(Calendar calendar);
+ setTimeToEndOfDay(Calendar calendar);
+ getStartDay(): Date;
+ getStartDay(): Date;
1..*
1..1
<<SteroType>>
MonthlyReport
+ getCalendarForNow(): Calendar
+ setTimeToBeginningOfDay(Calendar calendar);
+ setTimeToEndOfDay(Calendar calendar);
+ getStartDay(): Date;
+ getStartDay(): Date;
+ generateReport (): String
<<SteroType>>
WeeklyReport
+ getCalendarForNow(): Calendar
+ setTimeToBeginningOfDay(Calendar calendar);
+ setTimeToEndOfDay(Calendar calendar);
+ getStartDay(): Date;
+ getStartDay(): Date;
+ generateReport (): String
<<SteroType>>
User
- username: String
- password: String
+ get()
+set()
<<SteroType>>
FoodItem
- id: int
- name: String
- quantity: int
- calorie_value: int
- purchase_date: Date
- expiration_date: Date
- consumption_date: Date
+ get()
+set()
+getDays(): long
+burndownRate(): long
+waste(): long
+ twoDaysBeforeExpiration(): boolean
+ oneDaysBeforeExpiration(): boolean
+ todaysReminder(): boolean
<<SteroType>>
UserDao
+ authenticateUser(User)
<<SteroType>>
FoodItemDao
+ addFoodItem(FoodItem)
+ deleteFoodItem(FoodItem)
+ updateFoodItem(FoodItem)
+ getAllData(): List<FoodItem>
<<StereoType>>
ReportServlet
doPost()
doGet()
<<StereoType>>
DeleteItemServlet
doPost()
doGet()
<<StereoType>>
GroceryListServlet
doPost()
doGet()
<<StereoType>>
AddItemServlet
doPost()
doGet()
+ isValidFormat()
<<StereoType>>
UpdateItemServlet
doPost()
doGet()
<<StereoType>>
LoginServlet
doPost()
doGet()
Presentation jsp pages

6. Data Model


food
id INT(11)
name VARCHAR(45)
quantity INT(11)
calorie_value INT(11)
purchase_date DATE
expiration_date DATE
consumption_date DATE
Indexes
user
user_id INT(11)
username VARCHAR(45)
password VARCHAR(45)
Indexes

7. System Testing

To test methods, properties, classes I have used JUnit framework. JUnit is an open source Unit Testing Framework for JAVA. To simulate the interaction with the database I testes the four CRUD operations from Hibernate: read, create, update, delete, which all tested positive.

A few screenshots from my app:

List of groceries you have

ID	Name	Quantity (grams)	Calories	Purchase date	Expiration date	Consumption date
1	sparanghel	500	1000	2020-03-28	2020-04-04	2020-04-03
2	mere	1000	1500	2020-03-28	2020-05-01	2020-04-25
27	cascaval	250	800	2020-03-27	2020-04-18	
28	struguri	200	1800	2020-03-13	2020-03-18	
42	cirese	1000	800	2020-03-30	2020-04-02	
44	pizza	1000	700	2020-03-31	2020-04-03	
45	sparanghel	500	1000	2020-03-27	2020-04-03	2020-04-02

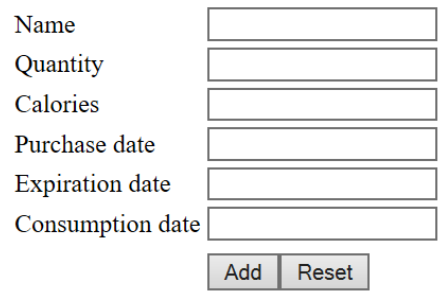
CHECK Notifications

28	struguri	200	1800	2020-03-13	2020-03-18	
42	cirese	1000	800	2020-03-30	2020-04-02	
44	pizza	1000	700	2020-03-31	2020-04-03	
45	sparanghel	500	1000	2020-03-27	2020-04-03	2020-04-02

CHECK Notifications

There are 2 days left until sparanghel expires, better eat it or give it away!
WASTE ALERT! There is waste for struguri!
ONE DAY LEFT before pizza expires, consume it before it goes to waste!
ONE DAY LEFT before sparanghel expires, consume it before it goes to waste!
-> To remain on track with the ideal burndown rate, today you should consume sparanghel, mere, cascaval, cirese, pizza, sparanghel,

Ideal burndown rates for groceries:



8. Bibliography

- http://hibernate.org/orm/
- https://mail.codejava.net/coding/java-servlet-and-jsp-hello-world-tutorial-with-eclipse-maven-and-apache-tomcat
- https://examples.javacodegeeks.com/core-java/junit/junit-test-case-example-for-web-application/