<WasteLess - assignment 3>

Analysis and Design Document

Student: Anda Papita

**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 4

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 8

7. System Testing 9

8. Bibliography 10

1. Requirements Analysis

# 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.

# Functional Requirements

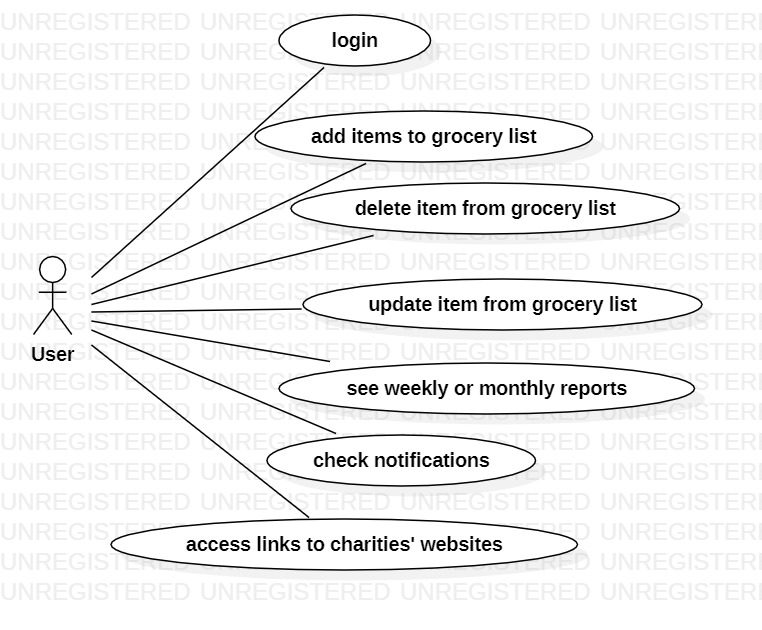
* The user is able to create a grocery list by adding new items
* The user is able to see reports of how much food is wasted weekly/monthly
* The user is able to track goals by following ideal burndown rates for each grocery on the list
* The user is able to see notifications and reminders about the 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

# 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 CQRS architecture, use a mediator pattern to handle requests
* Use a decorator pattern for changing the color of the report (green for above the ideal
* rate and red for under)
* 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

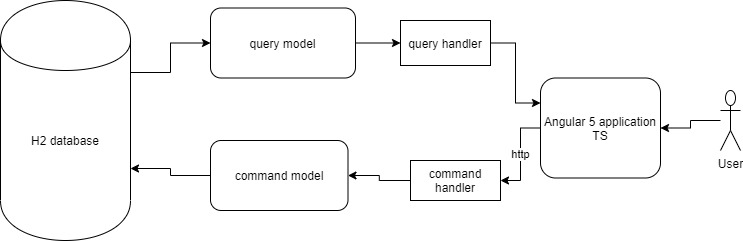
**3.1 Architectural Pattern Description**

The architectural pattern which was used is a client-server with CQRS architectural pattern, an **architectural pattern** that separates reading and writing into two different models. This means that every method should either be a Command that performs an action or a Query that returns data. A Command cannot return data and a Query cannot change the data. Moreover, a **client–server model** is a [distributed application](https://en.wikipedia.org/wiki/Distributed_application) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](https://en.wikipedia.org/wiki/Server_(computing)), and service requesters, called [clients](https://en.wikipedia.org/wiki/Client_(computing)). The *client-server* characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide.

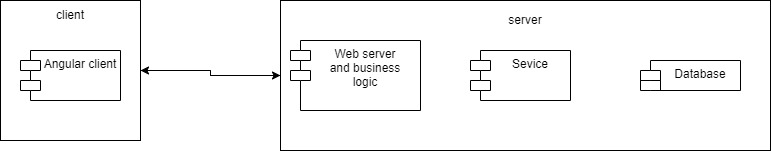
My application is a web application created with Spring boot and Angular 5, with Java language in the IntelliJ framework. The database was created with Spring’s H2 database dependency.

**3.2 Diagrams**

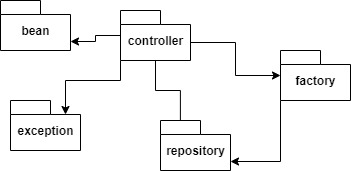
**System’s conceptual architecture:**

**

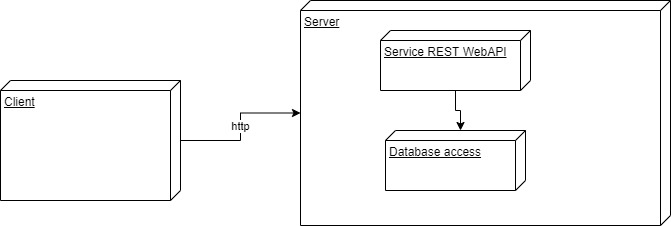
**Component diagram:**



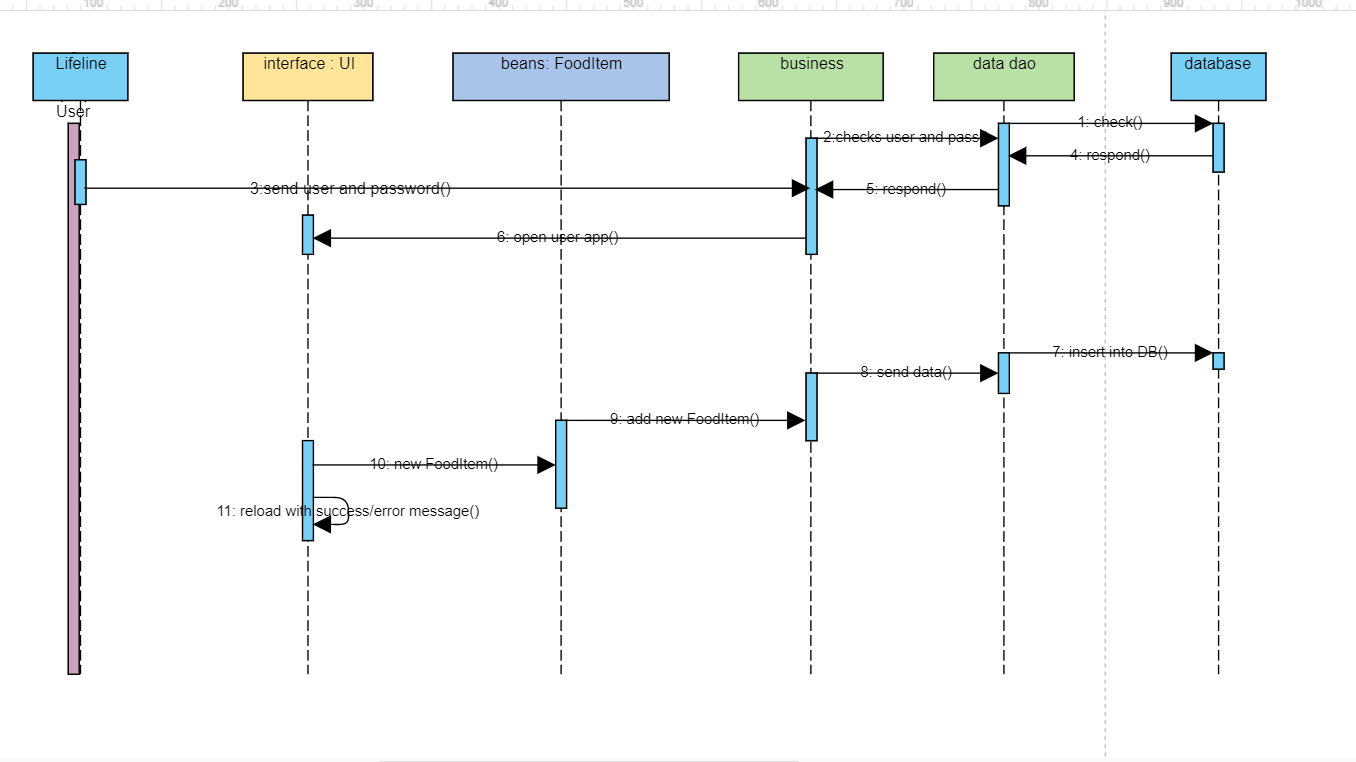
**Package diagram:**

****

**Deployment diagram:**

****

4. UML Sequence Diagrams



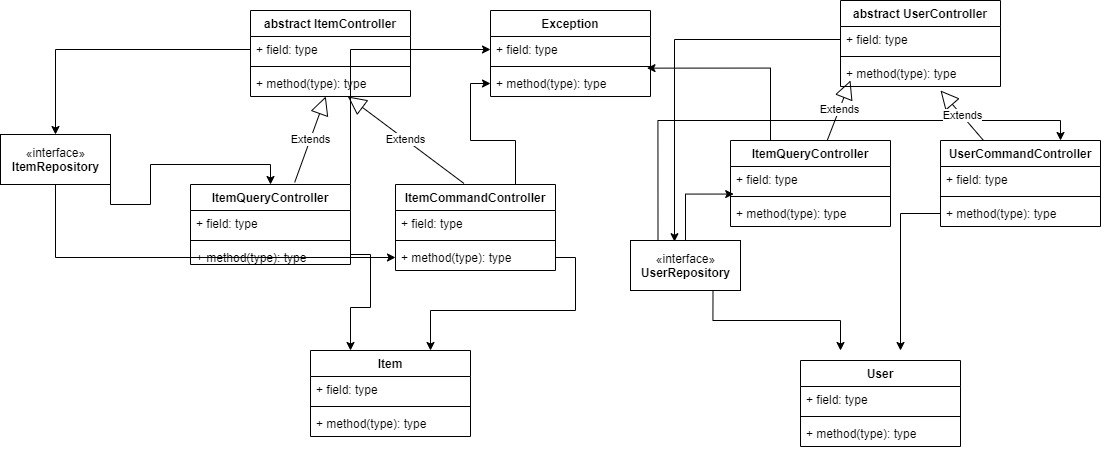
5. Class Design

**5.1 Design Patterns Description**

The used design pattern is themediator pattern, used to reduce communication complexity between multiple objects or classes. In object-oriented programming, we should always try to design the system in such a way that components are loosely coupled and reusable. This approach makes our code easier to maintain and test. The intent of the Mediator Pattern is to reduce the complexity and dependencies between tightly coupled objects communicating directly with one another. This is achieved by creating a mediator object that takes care of the interaction between dependent objects. Consequently, all the communication goes through the mediator.

In the Mediator pattern, the Mediator interface defines the interface ItemRepository and UserRepository which are used to communicate with the Colleague classes, which in my case are the Controller classes. There are two abstract Controller classed in my design, ItemController and UserController, whose jobs are only to define the mediator. They are extended by the ConcreteColleague classes, ItemQueryController, ItemCommandController and UserQueryController, User CommandController, respectively.

**5.2 UML Class Diagram**



From this diagram the ReportController (simple QueryController) and WeeklyReport, MonthlyReport classes are missing, I couldn’t fit them in the diagram.

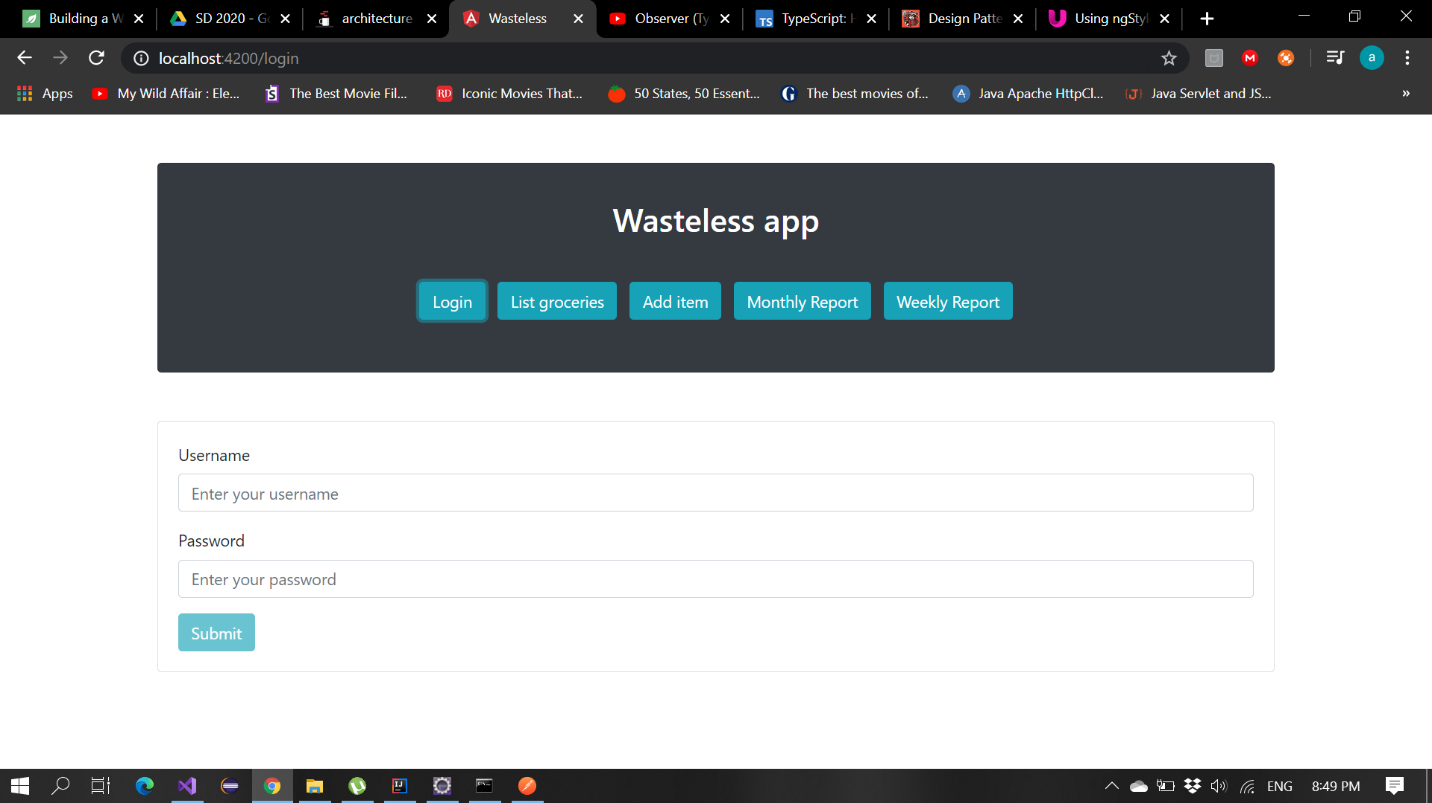
6. Data Model

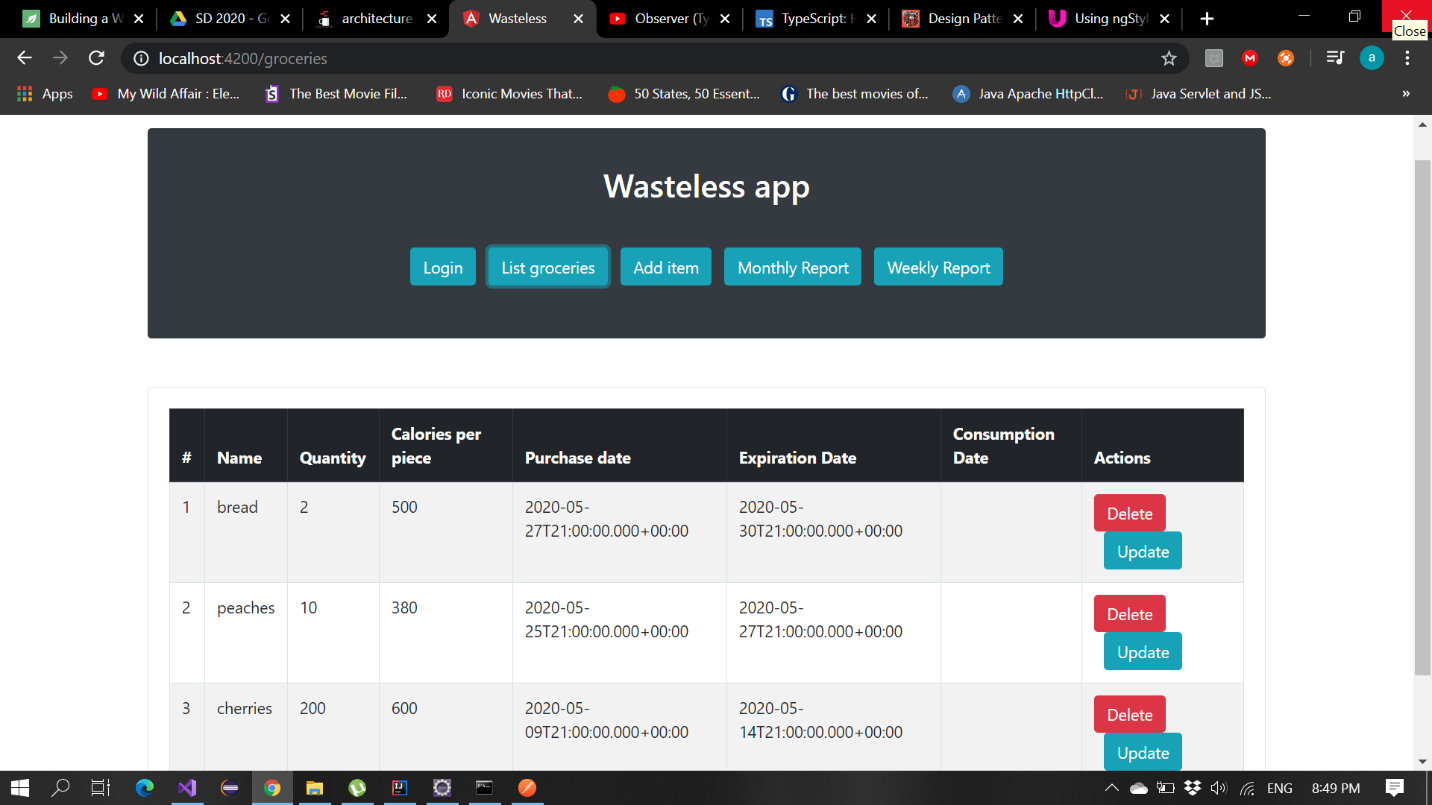
Since I use an auto-generated H2 database from Spring, there isn’t a database diagram. However, the H2 databse contains tables that hold food items’ and users’ information, correspondent with Item and User bean classes.

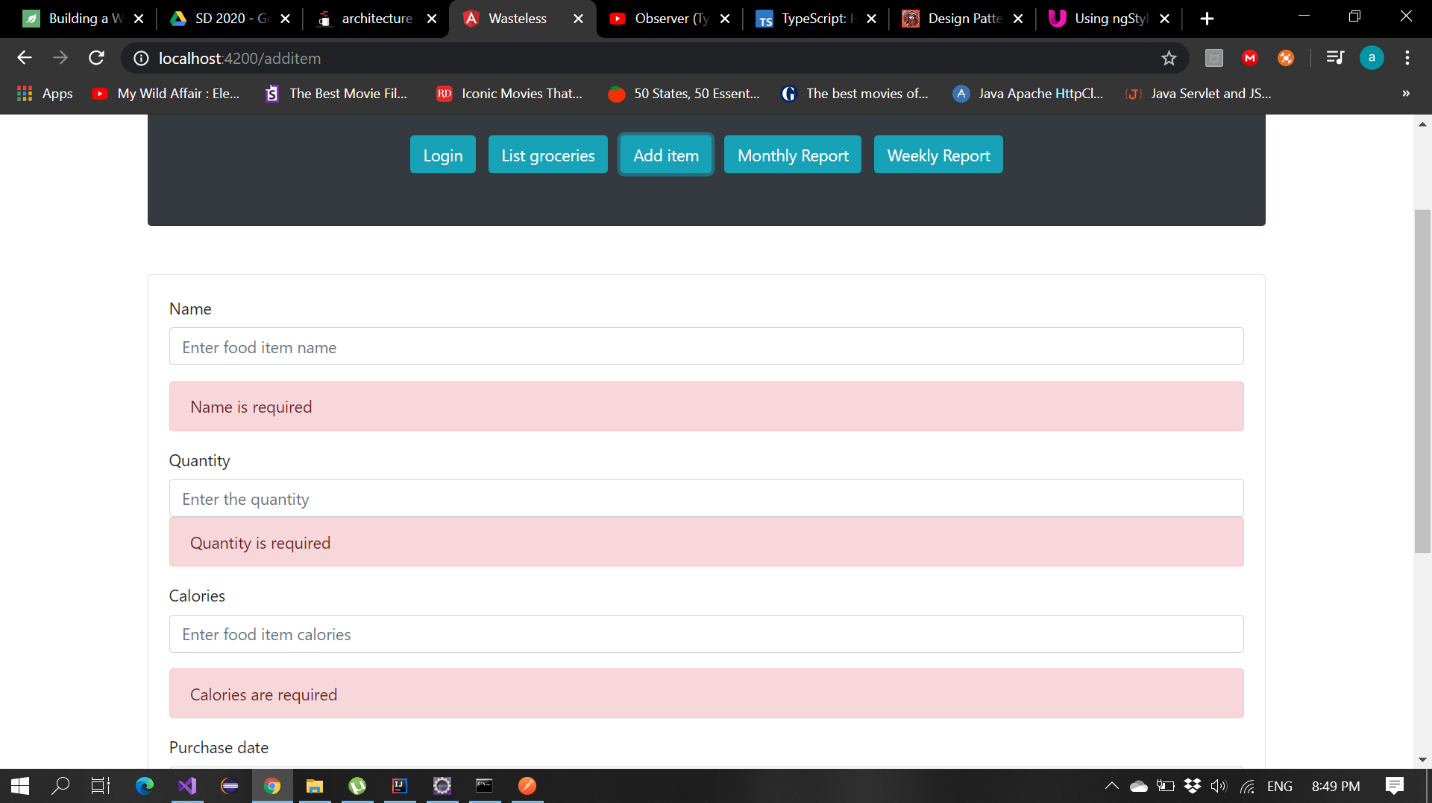
7. System Testing

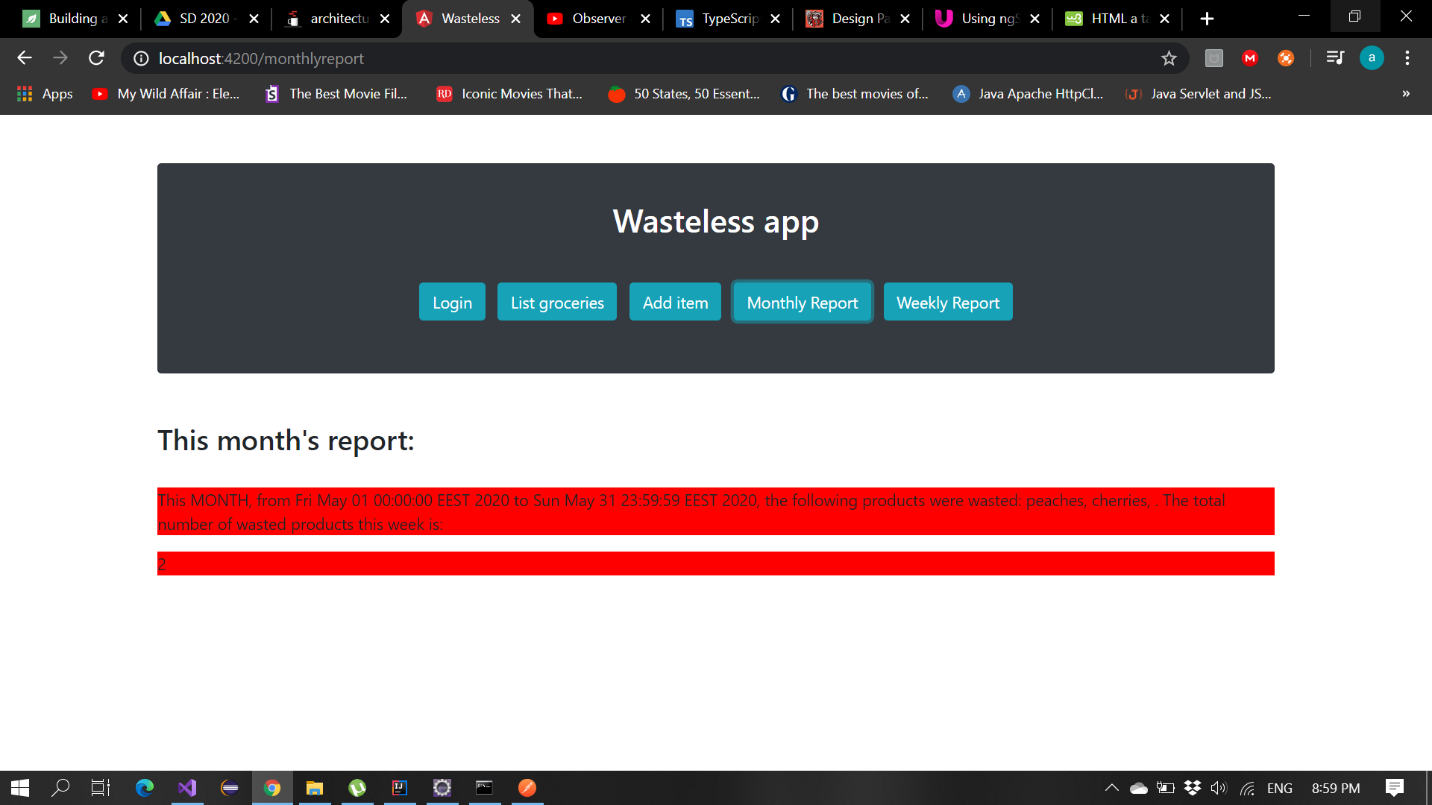
To test methods, properties, classes I have used JUnit framework. JUnit is an open source Unit[Testing](https://www.guru99.com/software-testing.html)Framework for JAVA. To simulate the interaction with the database I testes the four CRUD operations: read, create, update, delete, which all tested positive.

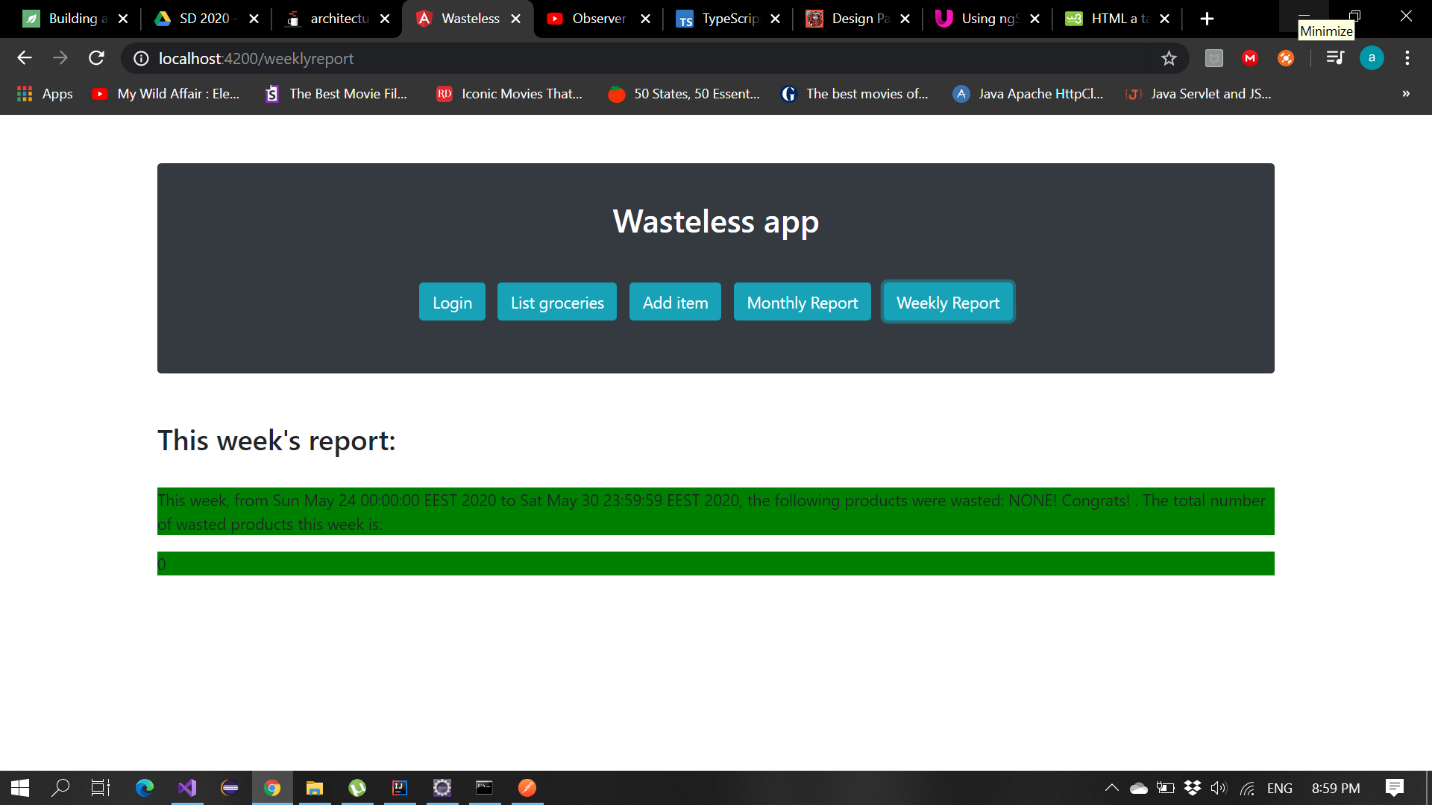
A few screenshots from my app:

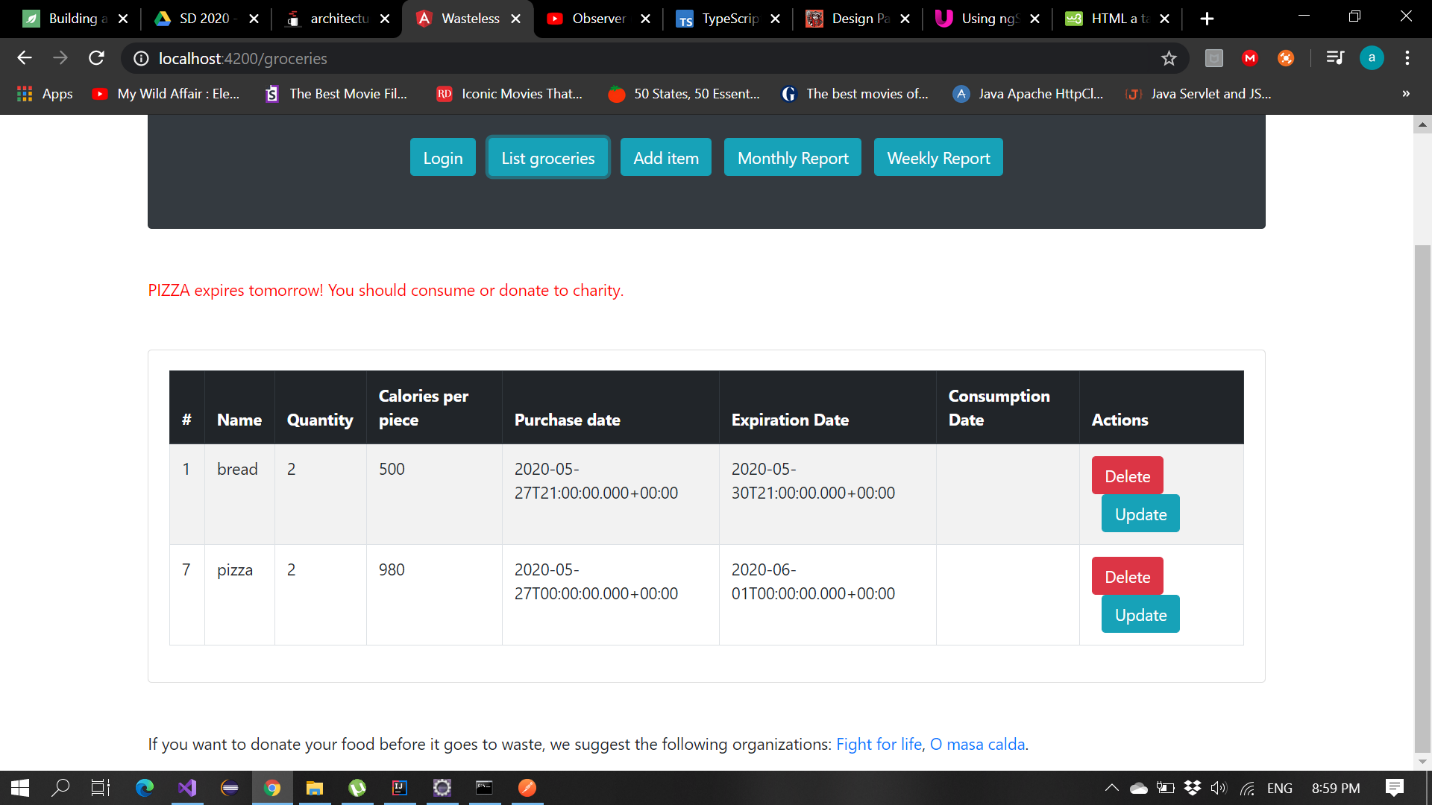












8. Bibliography

* <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/>