WasteLess 2

Analysis and Design Document

Student: Petrariu 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 4

3. System Architectural Design 4

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 8

7. System Testing 8

8. Bibliography 9

1. Requirements Analysis

# Assignment Specification

WasteLess is a food organizer application which stores information about the user’s groceries and keeps records of items that have expired or been used. The app calculates the user’s average daily calories consumed and the ideal daily calorie intake based on the items’ expiration dates. Expired groceries are kept in weekly/monthly reports.

# Functional Requirements

These are the main functional necessities of this application:

* Stores data about groceries (name, calories, quantity purchased, expiration date, purchase date, consumption date). Considering the items are stored in a fridge, they must have realistic longevity.
* Stores purchased items, from which the user may remove.
* Keeps track of expired items in report files.
* Calculates the ideal burndown rate of unexpired, unconsumed items and the user’s average daily calorie intake.
* Gives information about nearby charitable organizations which accept food.

# Non-functional Requirements

WasteLess is a small-sized project which does not have performance or usability issues. The maintainability and testability are guaranteed by the simple design of the application. Other non-functional requirements are:

* Using an ORM - The data access layer was implemented using the Hibernate libraries.
* Using a DI Container – The big-sized objects of this application are initialized and interact with each other using the Spring dependency injection container.
* Using an OOP language – Source code is written in Java.
* Having a client-server architecture – This application’s database is on the server side, the GUI on the client-side, and the business layer has been split and implemented in both.

2. Use-Case Model

Use-case diagram

A close up of a logo

Description automatically generated

Use case: Buy Groceries

Level: Summary

Primary actor: User

Main success scenario: After logging-in, the user goes to the ‘groceries’ panel and enter a quantity for each item that must be purchased. Then the user clicks the ‘purchase’ button and every item is mapped to a BoughtItem object and saved in the database.

Extensions: -

3. System Architectural Design

**3.1 Architectural Pattern Description**

The architectural pattern used in this project is the 3-layer architecture. The login and main GUIs represent the presentation layer, the logic and model classes represent the business layer and the data access class is the data management layer.

**3.2 Diagrams**

**Architecture diagram**

**A picture containing clock

Description automatically generated**

**Package diagram**

**A screenshot of a cell phone

Description automatically generated**

**Component diagram**

**A screenshot of a social media post

Description automatically generated**

**Deployment diagram**

**A screenshot of a cell phone

Description automatically generated**

4. UML Sequence Diagrams

Sequence diagram of the ‘Buy Groceries’ use case

A screenshot of a social media post

Description automatically generated

5. Class Design

**5.1 Design Patterns Description**

* Abstract Factory Pattern: used in the client of the application for generating objects of classes that implement ReportFactory. There’s two Report implementations: WeeklyReport and MonthlyReport. These classes generate files using a list of expired BoughtItem retrieved from the database.
* Observer Pattern: used in the client for sending a notification to the user when purchased items will expire in a day. The observer is the ItemObserver class and the observable is the MainGUI class. When the purchased items panel is initialized, the expiring items are counted and the observer is updated: if it already notified the user or there are no items expiring soon, it doesn’t send notifications. Otherwise, a small message box appears telling the user to donate the excess food.

**Class diagram
Description automatically generated5.2 UML Class Diagram**

6. Data Model

There are three model classes:

* BoughtItem: represents an item after it was purchased from the list of groceries and is an entity in the bought\_items table of the database.
* ShopItem: represents an item on the shelf in the store.
* User: keeps a user’s credentials and is an entity in the users table.

7. System Testing

Since the application was developed starting from the visuals, and ending with the algorithms, with no components added from other projects, all testing was done from the graphical interface of the app.

First, the login and main GUI classes were tested using fake data and a very simple business layer. Then, the data model was deduced from the interface and the project requirements: the Item class(split into two: BoughtItem, or items which have an owner, a purchase date and possibly a consumption date and ShopItem, or groceries) and the User class, used for authentication and retrieving items.

The database was designed using the data model. After inserting sample data in the tables and updating the business class, the application was tested again.

With the data and presentation layers tested successfully, the reports and rate algorithms inside the business layer remained to be implemented and checked. This required some updates to the data and presentation layers, mostly to the data access class.

The final tests were done to verify the ‘donate’ tab.

8. Bibliography