<WasteLess A1>
Analysis and Design Document

Student: Andrei Rusu
Group: 30431

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The purpouse of this assignment was to create an application using any language that would fullfill a few requirements, detailed below.

Functional Requirements

The app needs to have a login capability, where a user can authenticate and add grocery lists with the items they want to consume. The system should, based on this lists, calculate how much food is being wasted by the user, based on their calorie burn rate, notify them about expiring items and offer options for the excess food to be donated to charities where possible.

The application needs to work with a database, and have an ORM (Object Relational Mapper) for interacting with the database. The choice I made was to use the Hibernate ORM with Java and a MySql database.
Also, a Dependency Injection container was required, and I managed to use PicoContainer for this task.

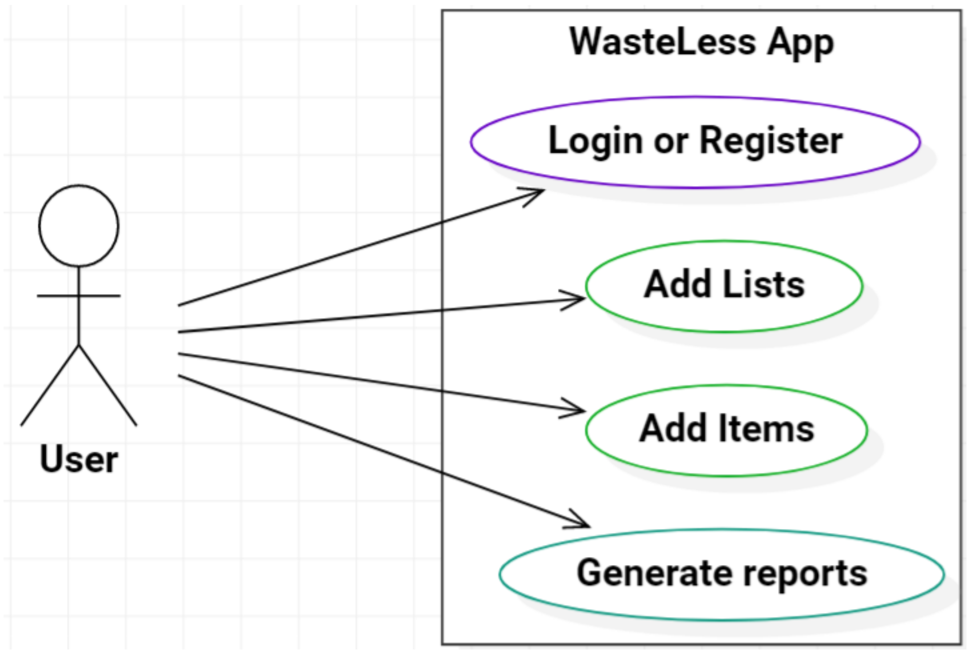
1.2 Non-functional Requirements

The application was required (if not explicitly, then by common sense) to be maintainable (hence the modularity), testable (point 7), have good usability, and also, from a further development standpoint, to be scalable and extensible further on.
Many of these requirements are fullfiled by using a good architecture, logic and coding style.

2. Use-Case Model

Use case: <Add an item to a list>
Level: <use-goal level>
Primary actor: <the user of the application>
Main success scenario: <Start the application, login or register, add a new list or select an existing one, fill out the new item details and add it to the current list>
Extensions: <failure if wrong data is input>
]

The use-case diagram is presented below:



3. System Architectural Design

3.1 Architectural Pattern Description

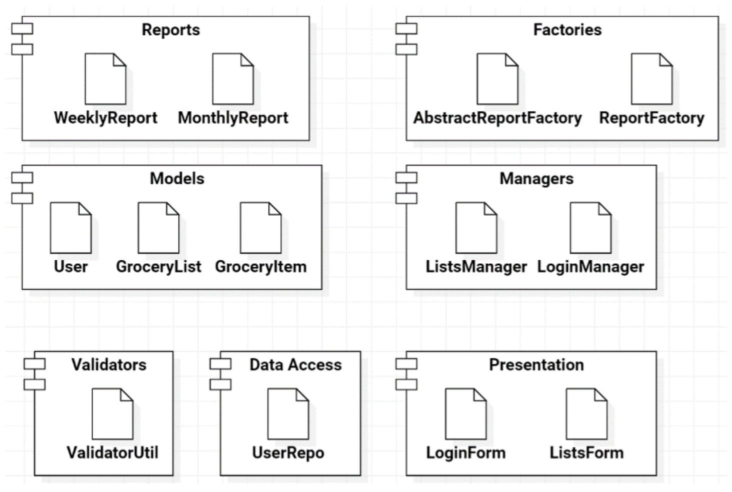
The system is based on a layered architecture, consisting of 4 layers:

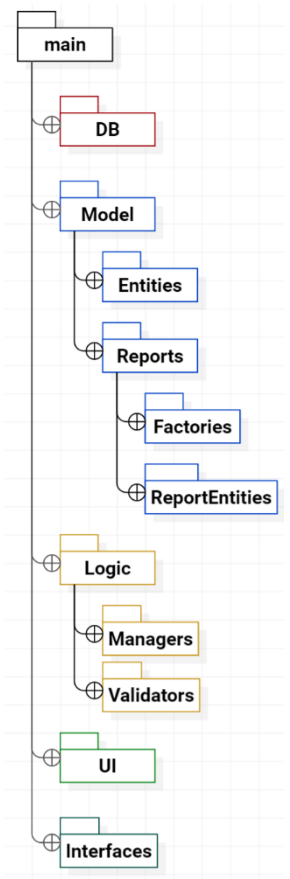
- The **presentation** layer, where the UI of the application is found;
- The **application/business logic** layer, where operations are performed using other components;
- The **persistence** layer, where the enitites of the application are found, like the User class;
- The **database** layer, consisting of the components that deal with the data source (DB) and persistence, like the ORM

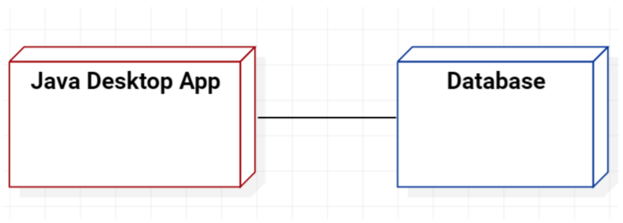
The dependencies go in a downward, meaning, for example, that the UI does not do any processing itself, it only calls methods in the business logic layer, which then goes on to read/write to the database thorugh dedicated methods in the database layer.

3.2 Diagrams

The component, package and deployment diagrams are shown below:





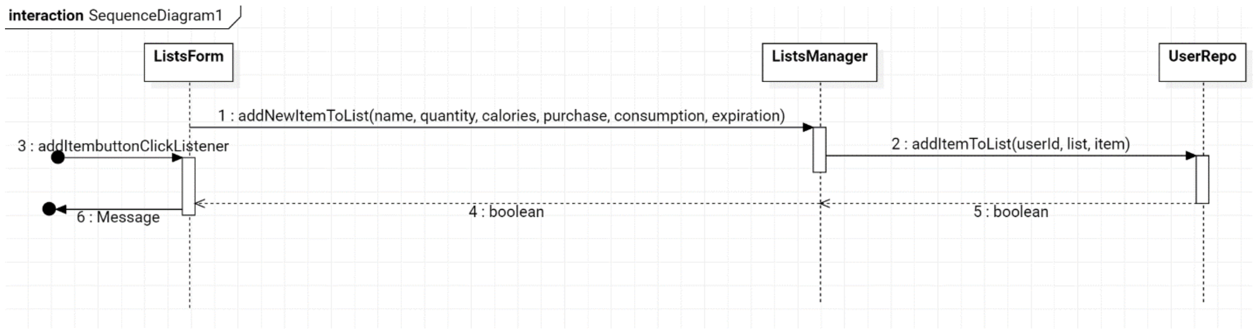


As described in the above section, the 4 layers interact with each other sequentially, usually starting with the UI, where the user presses or does something that triggers a listener to send a command to the manager, which then performs some taks and processes data, and then it sends a command to the data access components in order to read from or update the database.

In reality, the data access classes also have access to the system entities, due to the fact that the ORM software uses them, so there is a slight misaligment from the standard layered architecture pattern, but this is common in real applications.

4. UML Sequence Diagrams

The following diagram illustrates the scenario of a grocery item being added to a grocery list.


interaction SequenceDiagram1
ListsForm
ListsManager
UserRepo
1 : addNewItemToList(name, quantity, calories, purchase, consumption, expiration)
2 : addItemToList(userId, list, item)
3 : addItembuttonClickListener
4 : boolean
5 : boolean
6 : Message

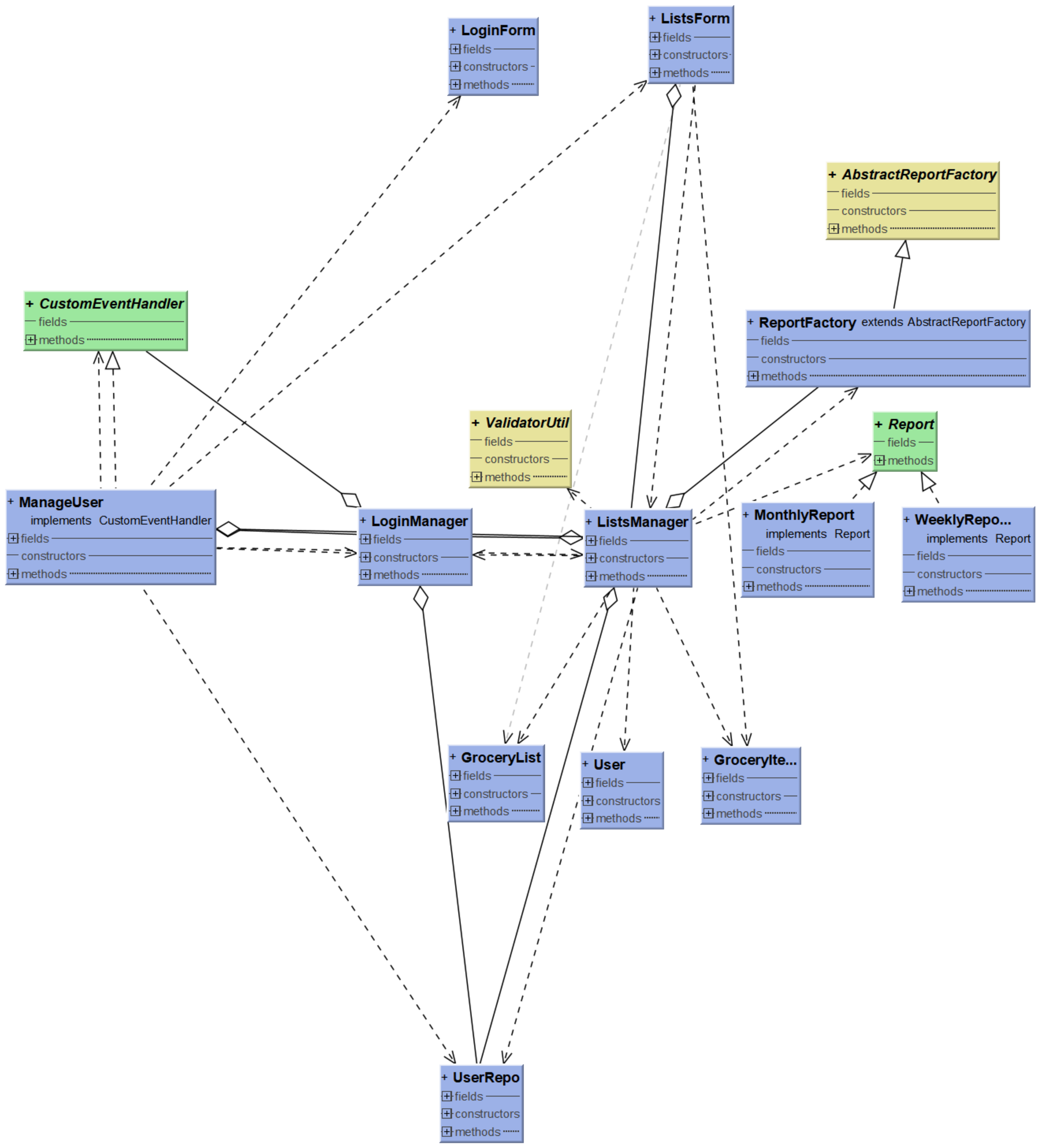
5. Class Design

5.1 Design Patterns Description

The abstract factory design pattern was used to generate the reports of wasted food, as per the requirements.
This basic factory method pattern creates a new object from a factory, which is an abstract class or interface that can create an object that another class requires.
On top of this, the abstract factory pattern takes this one level higher, by also being able to select a factory for object creation, and therefore have different factories for different types of objects, each one with its own types that can be created.
The exact implementation is detailed below.

5.2 UML Class Diagram

Below the UML class diagram of the system can be found. It includes all of the classes present in the project.
What is important to notice is how the abstract factory design pattern is used. The interface Report specifies a single method, used to compute wasted food. This gives the baseline that every report class will implement.
Next, the AbstractReportFactory abstract class is used to select the type of report to be created, using a single method with a parameter used to select a type of report. The class ReportFactory extends this class and implements the above mentioned method, which return a Report, either a weekly one or a monthly one. This Report is returned from the method to be used elsewhere in the program, mainly in the ListsManager class.
Also, from the diagram, the layer separation can be seen: the top is the UI, the next classes contain the application logic, the next layer is the model/persistence layer, and the final one includes the repository that deals with the data source through the Hibernate ORM.


+ LoginForm
fields
constructors
methods
+ ListsForm
fields
constructors
methods
+ AbstractReportFactory
fields
constructors
methods
+ CustomEventHandler
fields
methods
+ ReportFactory extends AbstractReportFactory
fields
constructors
methods
+ ValidatorUtil
fields
constructors
methods
+ Report
fields
methods
+ ManageUser implements CustomEventHandler
fields
constructors
methods
+ LoginManager
fields
constructors
methods
+ ListsManager
fields
constructors
methods
+ MonthlyReport implements Report
fields
constructors
methods
+ WeeklyRepo... implements Report
fields
constructors
methods
+ GroceryList
fields
constructors
methods
+ User
fields
constructors
methods
+ GroceryIte...
fields
constructors
methods
+ UserRepo
fields
constructors
methods

6. Data Model

The data is structured in the following manner:

The User

- has an ID, a first and last name, a password and a caloric intake goal
- the class also contains a Set of GroceryList items, which are the grocery lists of the user

The Grocery List

- has an ID, a name and a user index
- the user index establishes a 1-to-many relationship with the user
- the class also contains a Set of GroceryItem items, which are the individual items on the gorcery list (milk, bread etc)

The Grocery Item

- has an ID, a name and a list index
- the list index establishes a 1-to-many relationship with the containing list

7. System Testing

I tested the application myself, by adding new users, lists and item to each one, generating reports, testing the input data validation and the messages shown by the application.
The bugs that were found were resolved.

8. Bibliography

https://dzone.com/articles/layered-architecture-is-good
https://hibernate.org/
https://www.tutorialspoint.com/hibernate/index.htm
http://picocontainer.com/introduction.html
https://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm