Student:Andreea Monea

**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

# Functional Requirements

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

# Non-functional Requirements

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.

2. Use-Case Model

*Use case: register*

*Primary actor: user*

*Main success scenario: register successfully*

*Extensions: if the process fails, the user must follow the instruction and have the possibility to do the registration process again*

*Use case: login*

*Level: <one of: summary level, user-goal level, sub-function>*

*Primary actor: user*

*Main success scenario: login successfully*

*Use case: add food*

*Primary actor: user*

*Main success scenario: added item into the account*

*Extensions: if the process fails, the user must follow the instruction and have the possibility to do the process again*

*Use case: add goal*

*Primary actor: user*

*Main success scenario: added goal into the account*

*Extensions: if the process fails, the user must follow the instruction and have the possibility to do the process again*

*Use case: donate*

*Primary actor: user*

*Main success scenario: donation completed*

*Extensions: if the process fails, the user must follow the instruction and have the possibility to do the process again*

3. System Architectural Design

**3.1 Architectural Pattern Description**

*The Layered Architecture has been used. The application has the following layers:*

*Models (Data Layer)– base classes for the application*

*DataAccess Layer – for DB connection*

*Business Layer – for verifying the data received, in order to insert it into the data base*

*Forms – the forms for each action the user can take*

*Templates – html files for all the existing froms of the app*

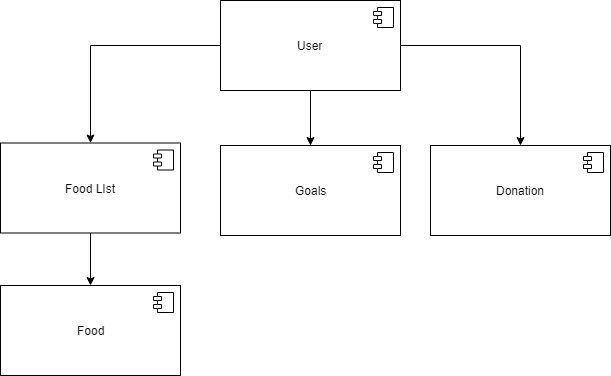
**3.2 Diagrams**

O imagine care conține captură de ecran

Descriere generată automat

O imagine care conține captură de ecran

Descriere generată automat



4. UML Sequence Diagrams

O imagine care conține captură de ecran

Descriere generată automat

5. Class Design

**5.1 Design Patterns Description**

The application does not use the required Abstract Factory pattern for weekly reports

**5.2 UML Class Diagram**

O imagine care conține text

Descriere generată automat

6. Data Model

The data models used are the user, list of foods, food, donations and goals.

The center of all is the user which has a username and a password also a list of food assigned automatically through its id. The user can insert food which has a name, a quantity, calories and expire date which is measured in days until expire date(implementation decision). After the inserting a pop-up notification will show with the amount of calories that need to be consumed in order to not waste food, also the alternative with grams/day will show also as a pop-up notification. This numbers are computed by dividing the number of calories/quantity to the number of days. The user can also view its items in the list.

The user can set goals that can also have pop-up notifications implemented in the Flask App. He has donate options: for donating food waste to orphans or homeless organizations.

7. System Testing

The system testing is found in test.py where login, register and logout methods are being unit tested. Because of the database I have chosen(found too late it was a bad choice) I have not found configuration for MySQL Database just for SQL lite. I gave it a try even though I am new to the WEB app testing.

8. Bibliography

Flask tutorials on Youtube, ex:

[https://www.google.com/search?q=flask+tutorial&oq=flask+tur&aqs=chrome.1.69i57j0l7.3259j0j1&sourc eid=chrome&ie=UTF-8#kpvalbx=\_0uuRXum9G-TmrgTMtJ-4Bg72](https://www.google.com/search?q=flask+tutorial&oq=flask+tur&aqs=chrome.1.69i57j0l7.3259j0j1&sourc%20%20%20eid=chrome&ie=UTF-8#kpvalbx=_0uuRXum9G-TmrgTMtJ-4Bg72)

9.Important info

*The application is a WEB Application made with Flask for the front-end part and python for backend.*

*The migration of the database is actually an export, I have a problem with migration in MySQL, do not know why, as I said bad decision to chose MySQL.*

*Even though all the CRUD operation are implemented just the insert into database is used(jus adding action where specified)*

*The app does not have weekly reports so the Abstract Factory Pattern is not implemented.*