Wasteless Application

Analysis and Design Document

Student: Sabou Cezara

**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

Requirements Analysis

# Assignment Specification

*This application is meant to help users manage food waste. Once the user is authenticated he can input grocery lists and see reports of how much food is wasted on a certain day, picked by the user. Every grocery list item is supposed to have the following things : name, quantity, calorie value, a purchase date, a consumption date and an expiration date. The system also provides the users to donate food to the local food charities when the users have exceeding waste levels.*

# Functional Requirements

*The requirements are the following:*

* *The user should be able to log into the page*
* *If the user does not yet have an account, there should be a registration page which will allow the user to create one*
* *After logging in, the user should be able to access the grocery lists previously created and add new lists*
* *A user should be able to open a grocery list item and add, delete or see the details of a certain grocery item*
* *Notifications should be sent to the user if the waste levels are too high*
* *The user should receive suggestions with options to donate the excess food*
* *The user should be able to see monthly/weekly reports of the waste details, as well as seeing the color representing the waste level(if the waste level exceeds limits, the color should be red, if not the color should be green)*

# Non-functional Requirements

*The non-functional requirements are:*

* *The user should not be allowed to alter the ideal burndown rate, nor the way the actual burndown rate is calculated*
* *The user should not be allowed to see the other user’s lists*
* *The user should not be allowed to access the grocery lists if the correct email and password was not provided*

2. Use-Case Model

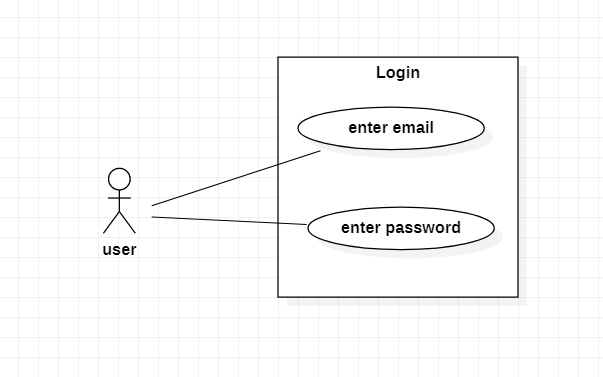
*Use case: User login*

*Level: user-goal level*

*Primary actor: regular user*

*Main success* *scenario: the user successfully introduces the email and the password, and the main page opens*

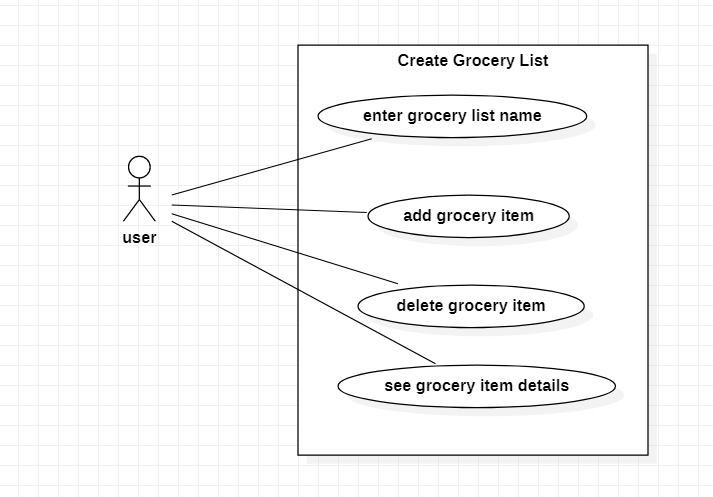
*Extensions: the user fails to introduce the correct email and password and an error message is shown*



*Use case: Create Grocery list*

*Level: user-goal level*

*Primary actor: regular user*

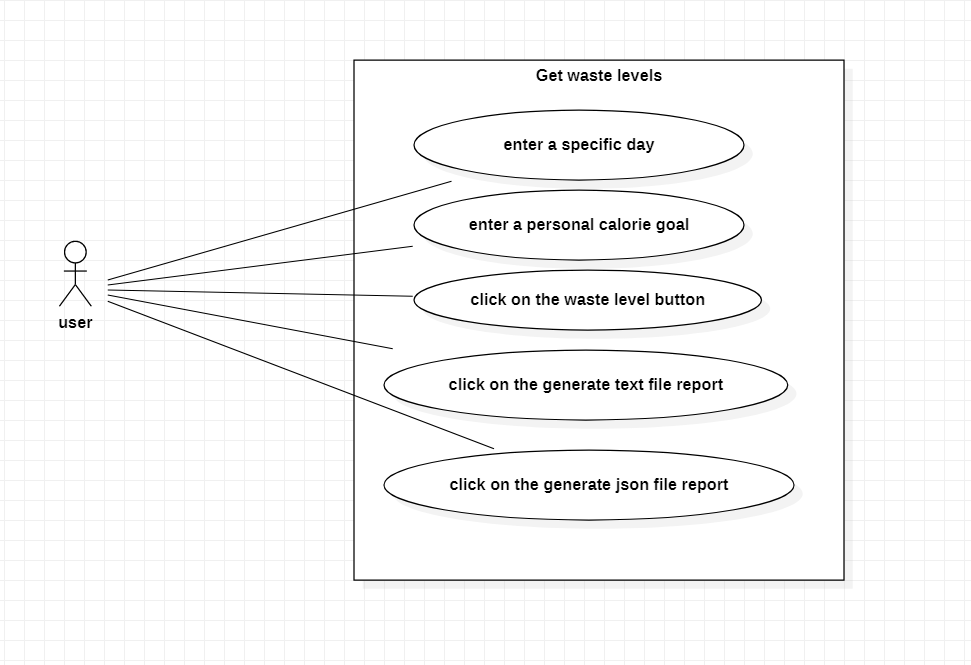
*Main success scenario: the user successfully introduces the name of the grocery list and the new list appears, where the user can add and delete items.* 

*Use case: Get waste levels*

*Level: user-goal level*

*Primary actor: regular user*

*Main success scenario: the user successfully introduces the day and the calorie goal, which is also called the ideal burndown rate, and then successfully presses the button “Waste level” of a specific list from the main page. A message appears on the screen with the current calorie number and also, if the waste level is too high, if it exceeds the goal, then the user is notified through an alert message.*



3. System Architectural Design

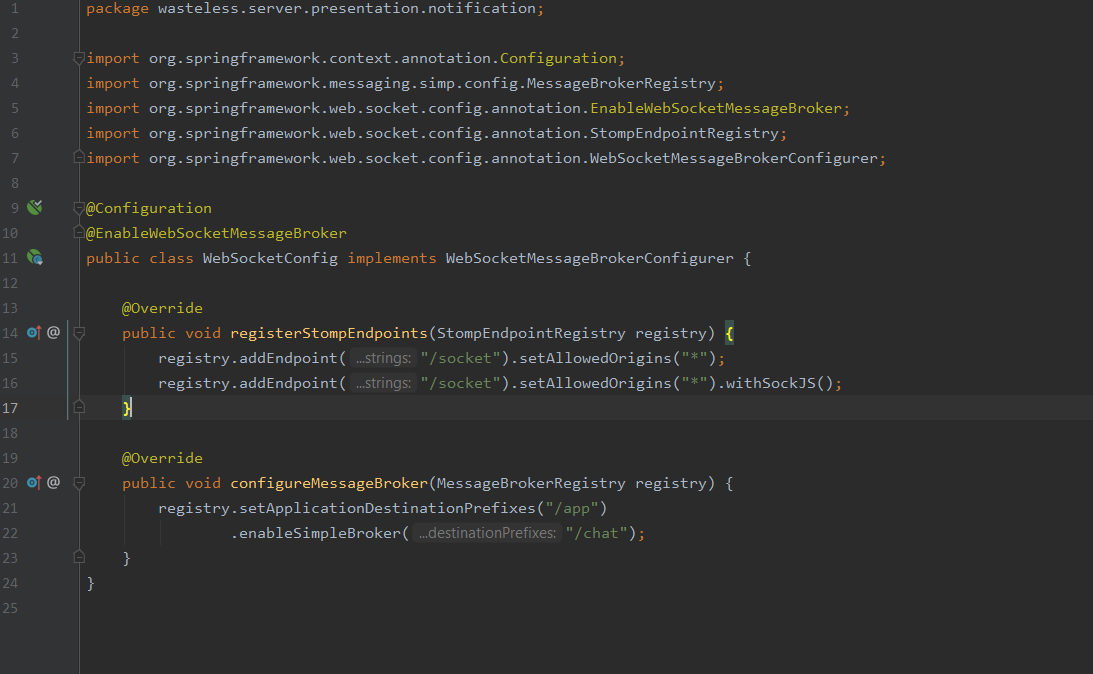
**3.1 Architectural Pattern Description**

*The application is split into two parts: the frontend and the backend, and it represents a client-server architecture.*

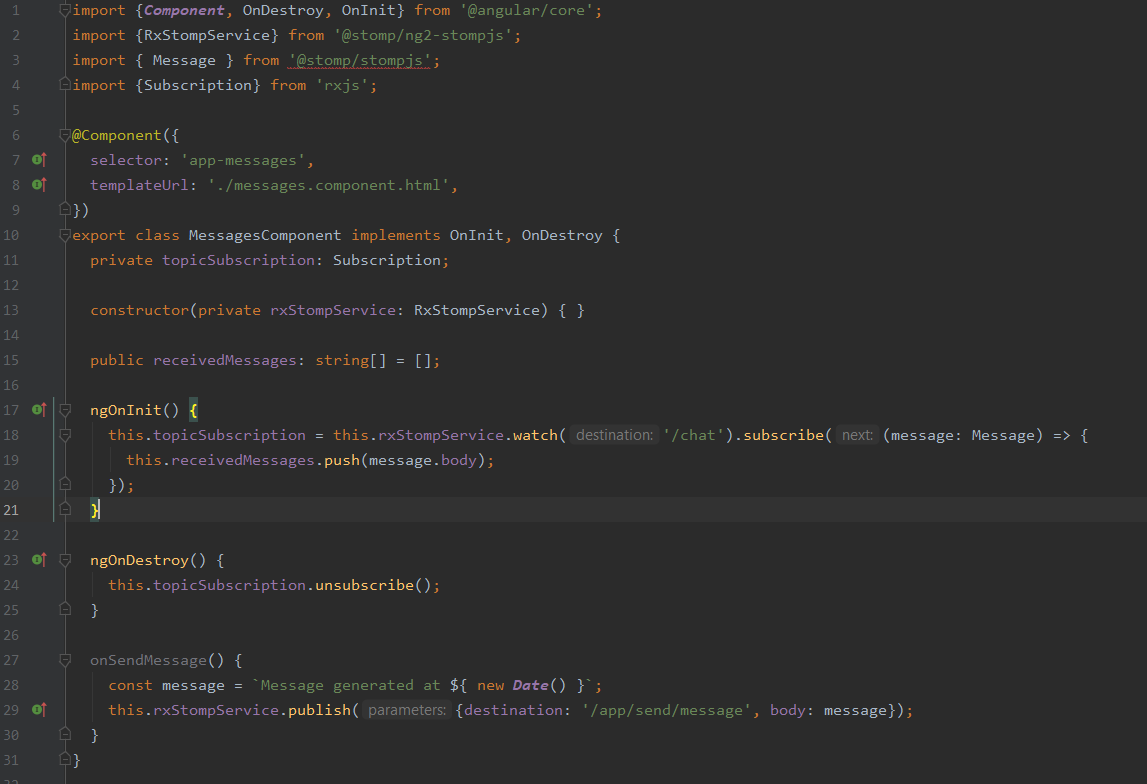
*In the frontend, I used Angular to design the application, and it represents the client side.*

*The architecture used in this project is called CQRS, which stands for Command Query Responsibility Segregation, and the goal is to clearly separate the both the service and the controller layers in order to deal with reads and commands, and they should be treated by the system separately. For this project I tried to apply the CQRS architecture to a rest API, this means that both the service and the controller layer were supposed to be separated both in command and query components. Because the mediator pattern was needed, it was used to redirect the requests from the client to the query handler or the command handler, depending on the request type.*

*The method which I used to send notifications to the user is through web sockets. Spring offers a very friendly method to do that, therefore I needed a single class which had two methods :*



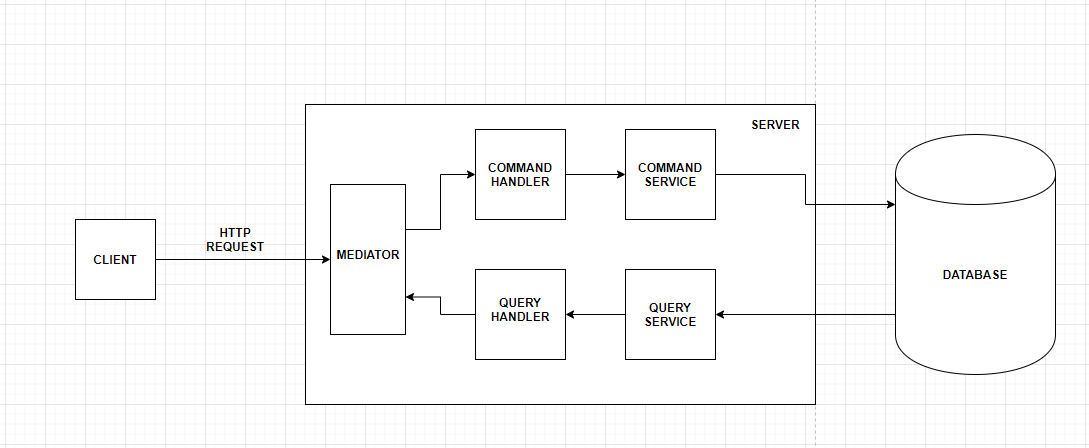
*For the client part, the code was a little bit more complex, but it was still not that difficult to implement:*



**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

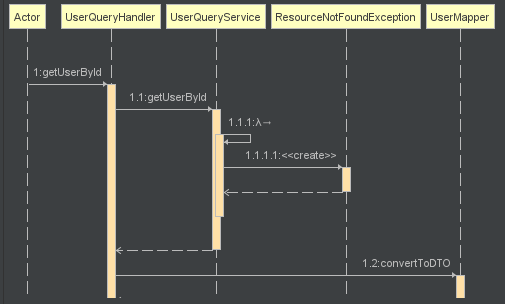
*The system’s conceptual architecture is the same as the one described in the course, which is layered:*



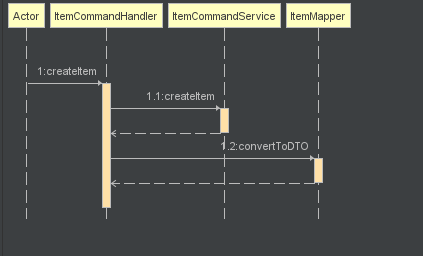
4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*

*This is the sequence diagram for getting a user by its id*



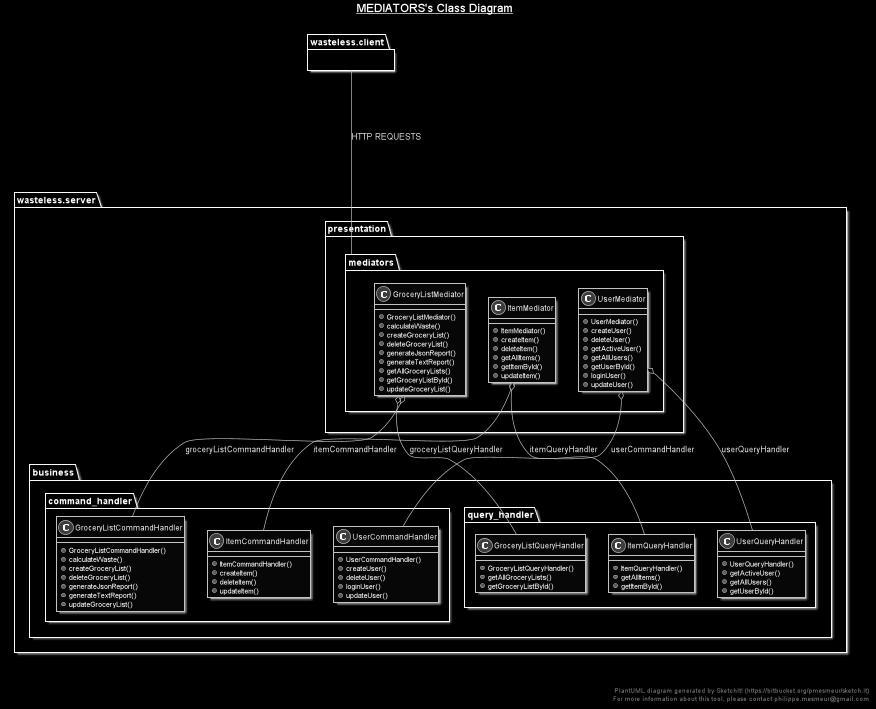
*And this is another sequence diagram, which represents creating an item :*



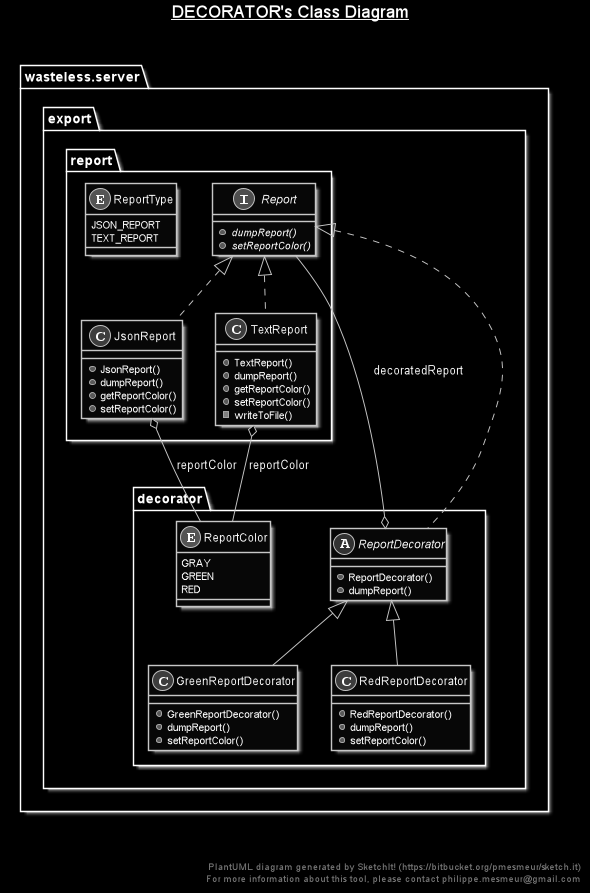
5. Class Design

**5.1 Design Patterns Description**

*The design pattern used in this project is the Mediator Pattern, which was used to handle client requests. I did not use generics to generate the type of mediator because I tried and I did not manage to make it work, therefore I chose to create a separate mediator for each data type, and map a specific handler for each type of request, just as requested in the project assignment.*

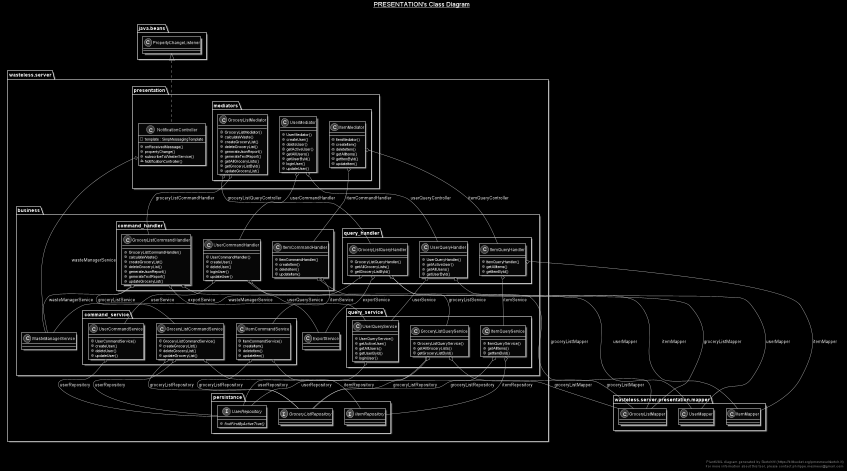
**

*The other pattern used in this project is the Decorator Pattern, which was used along with report generation. Its purpose is to simply add another property to an existing object. In our case the report was supposed to be green when the waste level was normal, and if it exceeded the limit it should be red.*

**

**5.2 UML Class Diagram**

*In this image there is the actual representation of the architecture chosen, which is the layered architecture. And the classes were designed to fit the layered architecture. Therefore, whenever the client issues a request, the data goes through the presentation layer, where for each data type, Item, User or Grocery list, there is a specific controller which*

**

6. Data Model

*This is the data model. I chose to have 3 data models:*

1. *The Item class.*

*This class holds all the attributes about the item object:*

* 1. *The item id*
  2. *The quantity*
  3. *The calorie value*
  4. *The purchase date*
  5. *The consumption date*
  6. *The expiration date*

1. *The User class.*

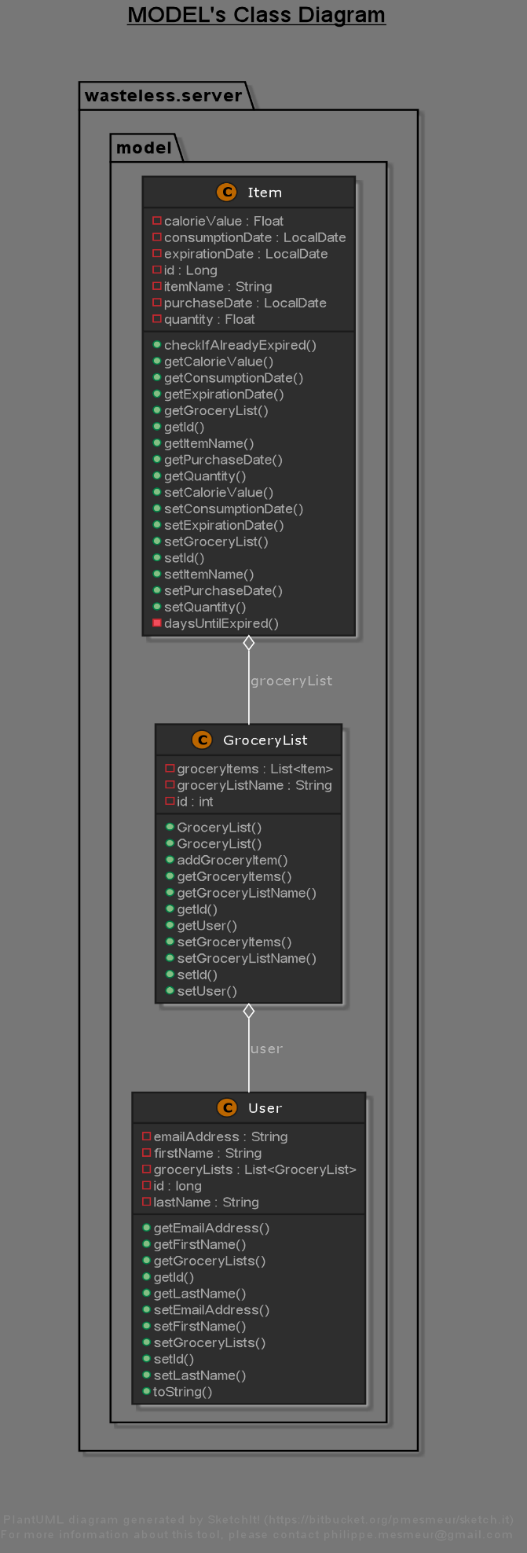
*This class contains information about the user:*

* 1. *The user id*
  2. *First name*
  3. *Last name*
  4. *Email address*
  5. *Password*
  6. *A personal list of grocery lists*

1. *The Grocery List class:*

*This class holds the information about a grocery list:*

* 1. *The grocery list id*
  2. *The grocery list name*
  3. *The list of items*

**

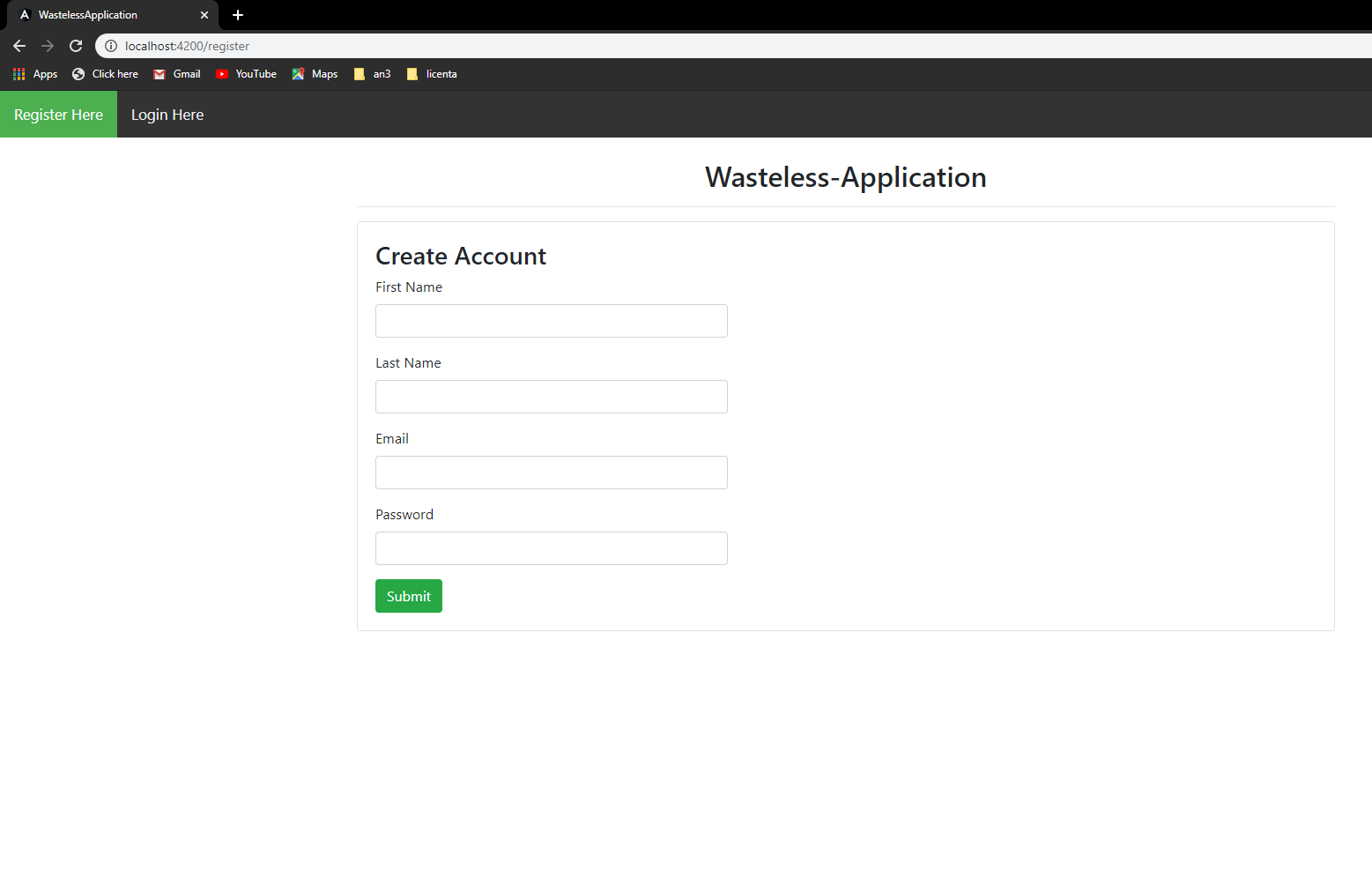
7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

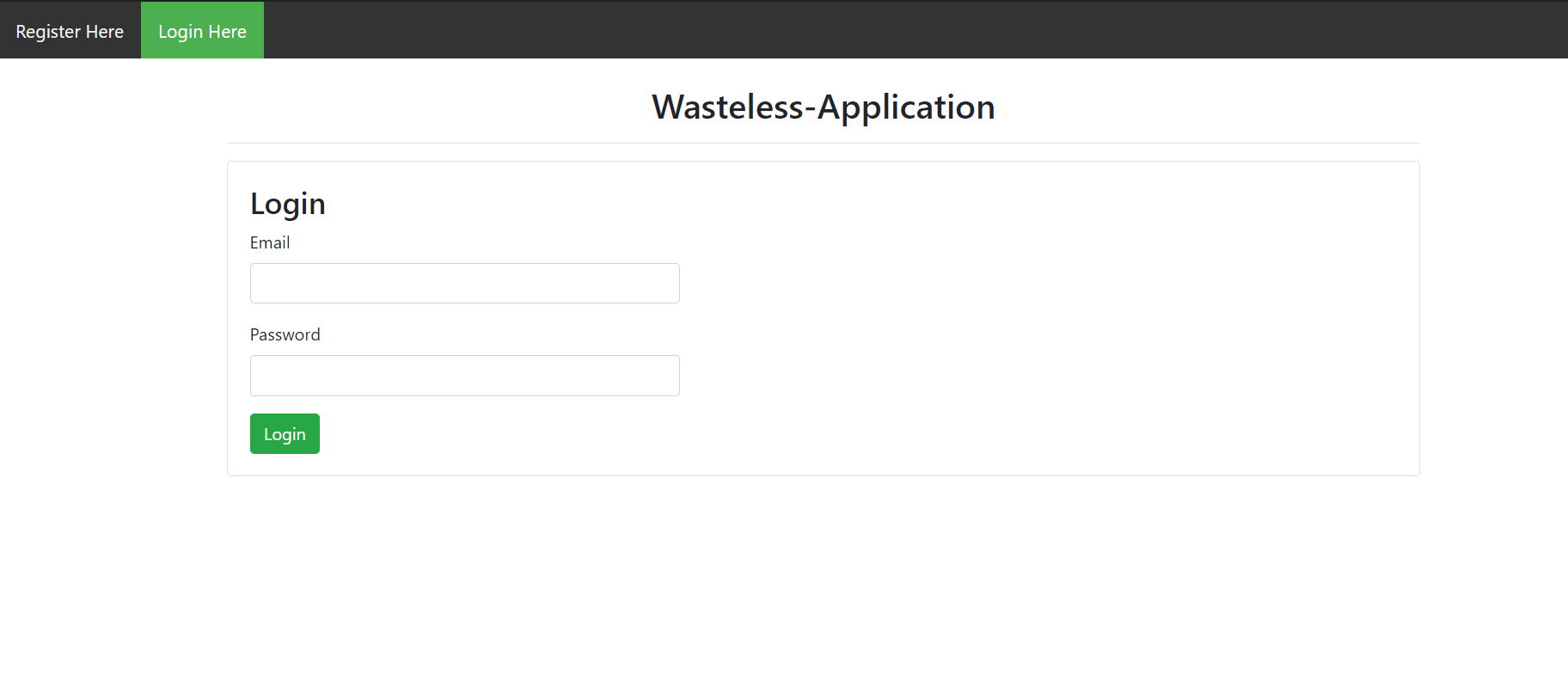
*For the testing part, I chose to use unit testing.*

*I will show the functionality of the application here:*

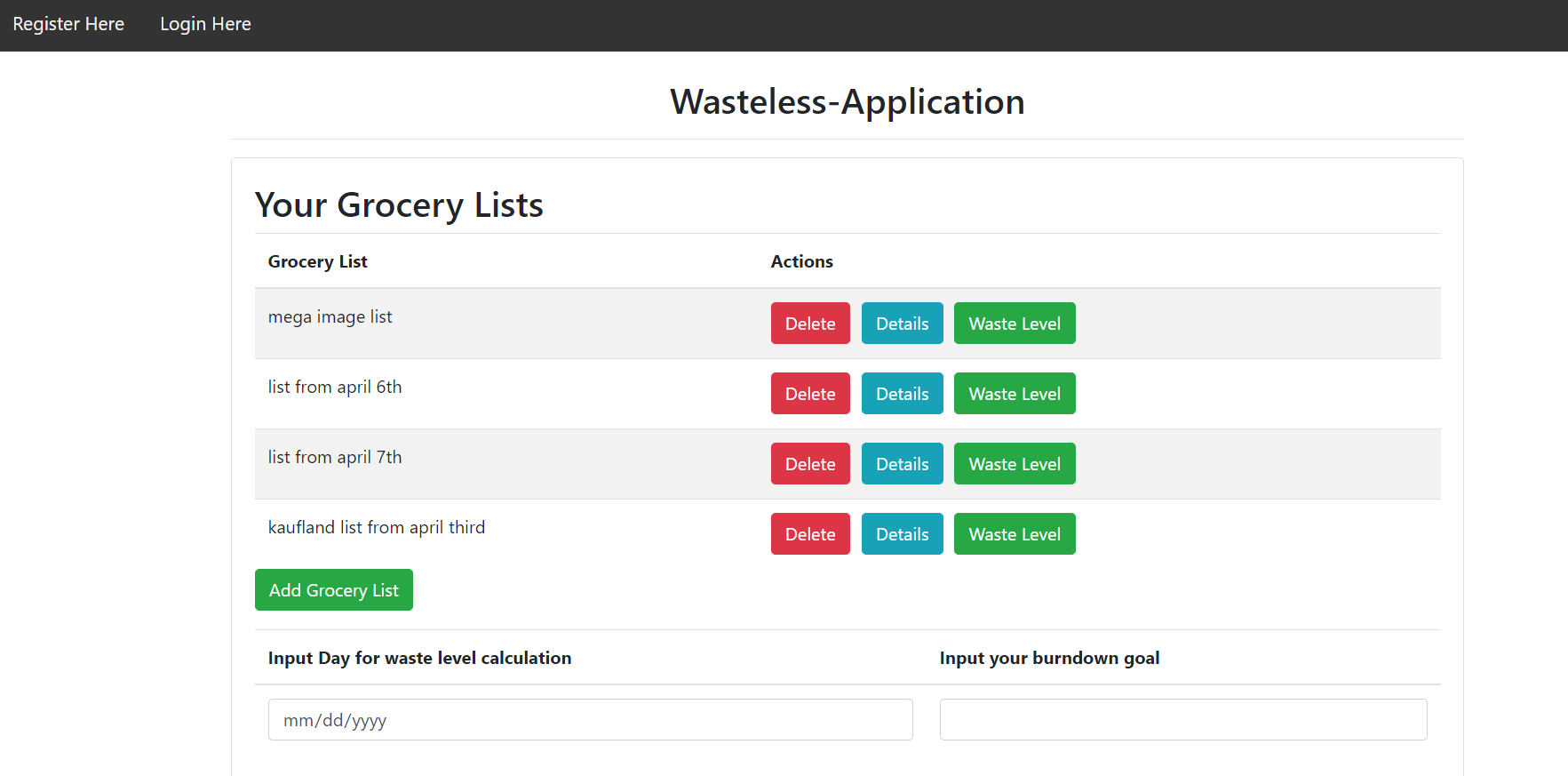
*The Register page, where the user inputs their data and it is saved into the database:*



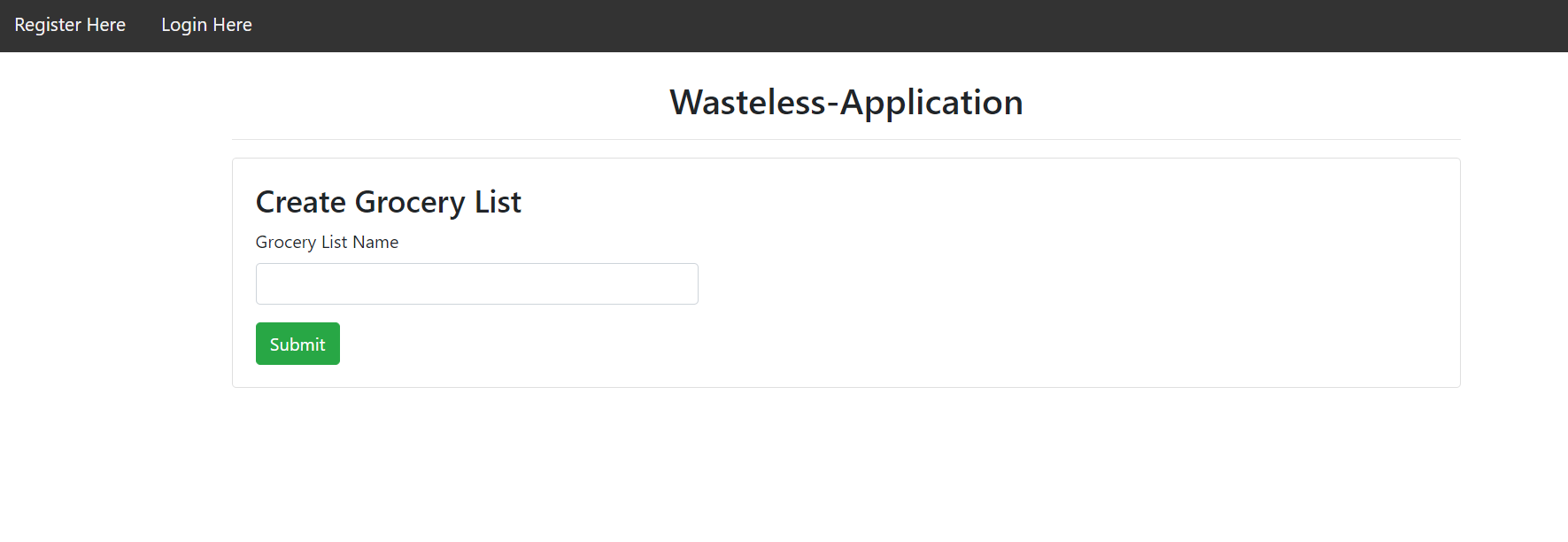
*The Login page, where the user can input their email and password, and if the credentials do not match the ones in the database, then they cannot enter to see their grocery lists:*



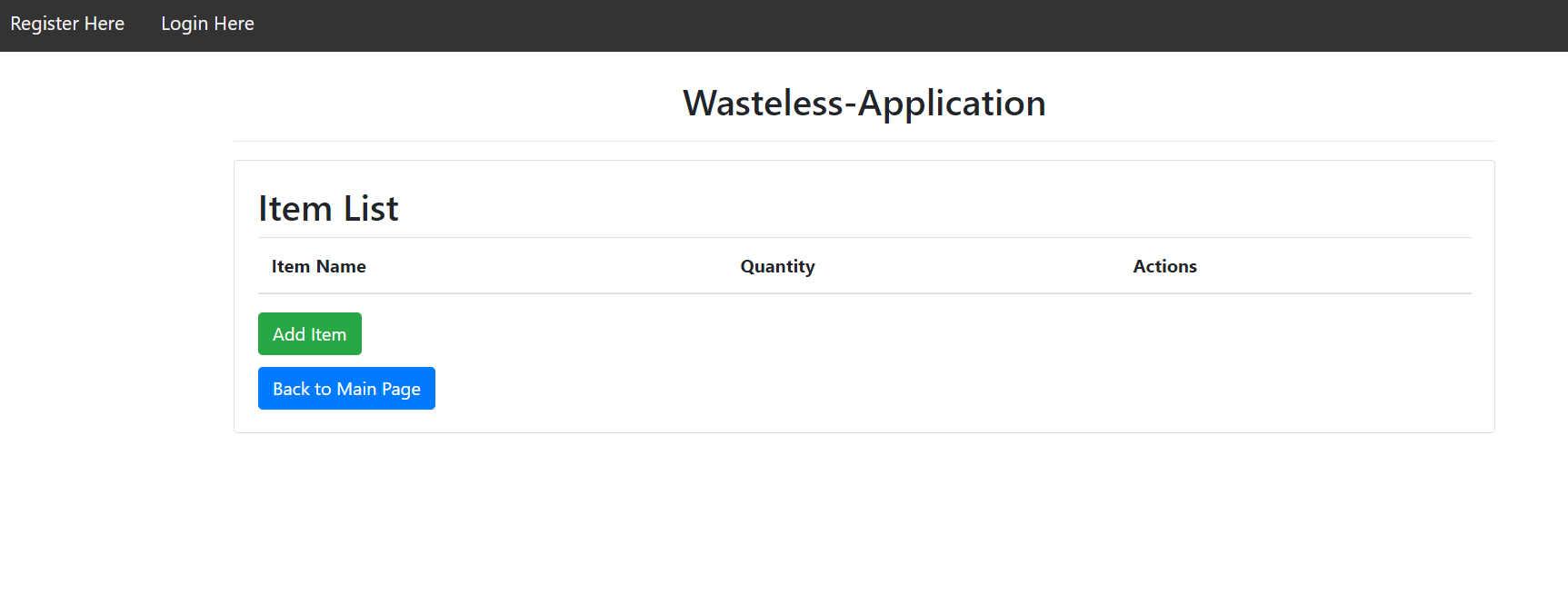
*The main page, where the user can see all the grocery lists previously entered.*



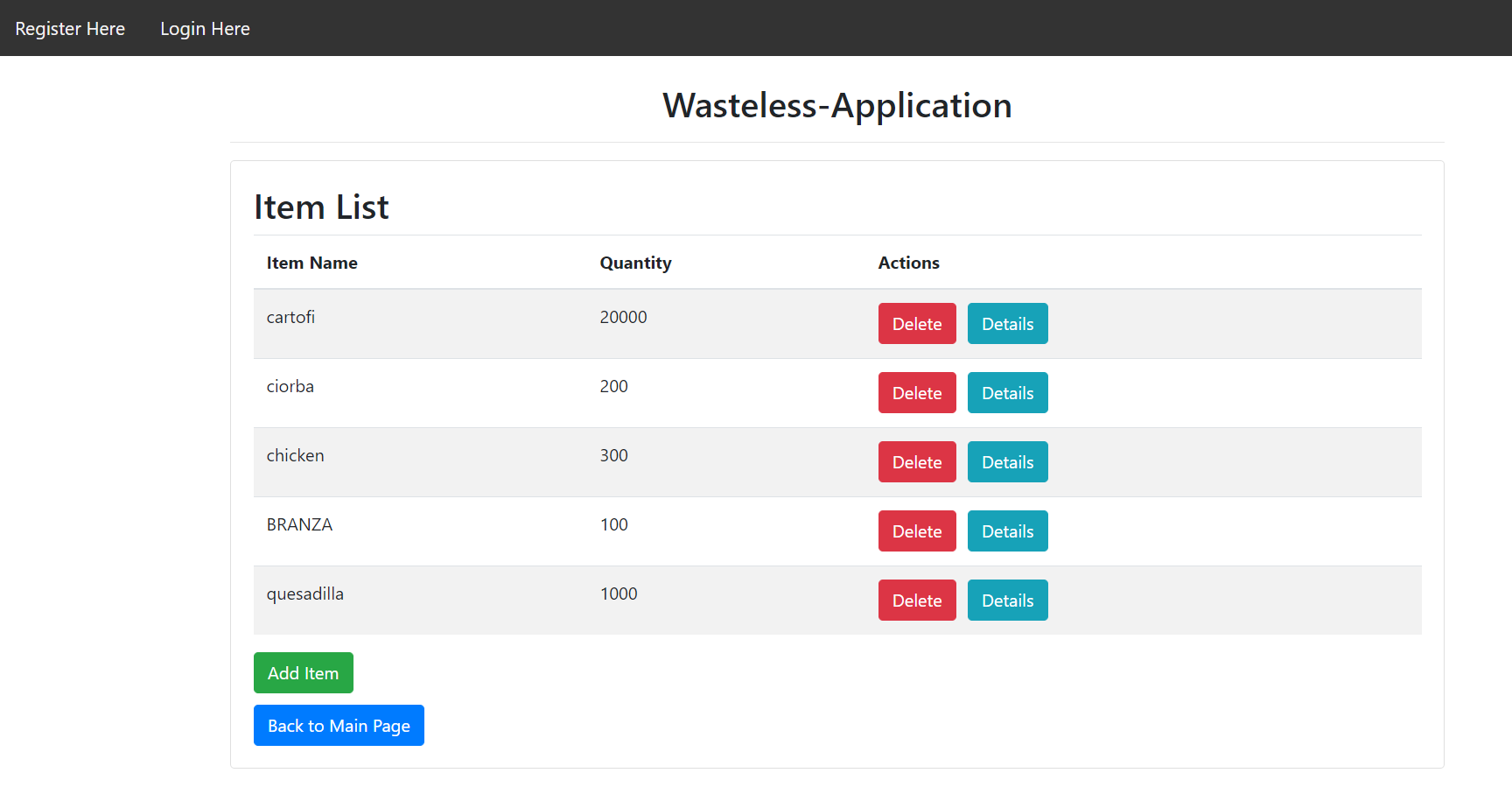
*And he can do the following operations:*

* *Create a new list, where they can enter the name of the list and then they are taken to the page list* 

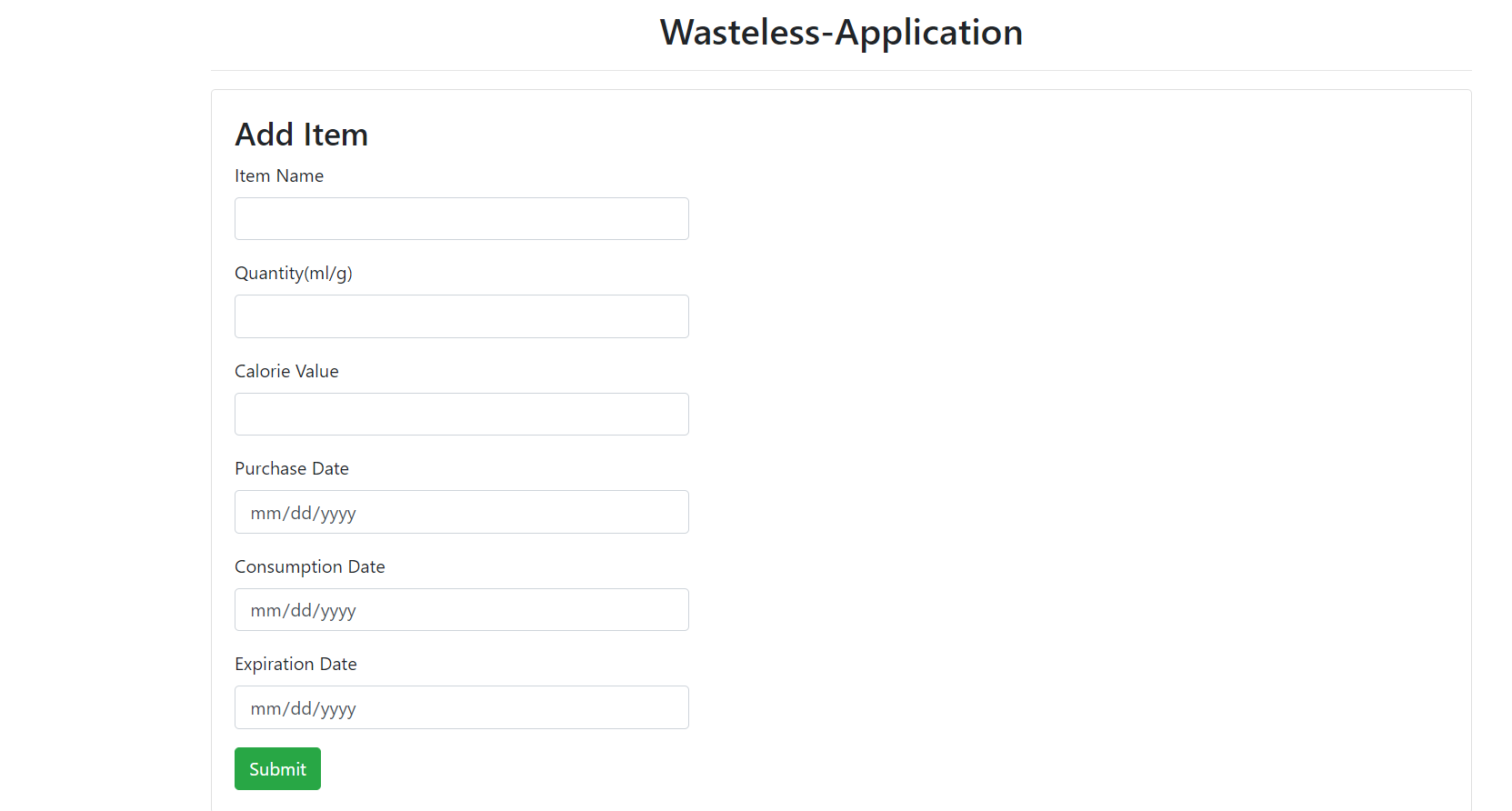
*After entering the name, the user can keep adding items to the list or go to the main page*



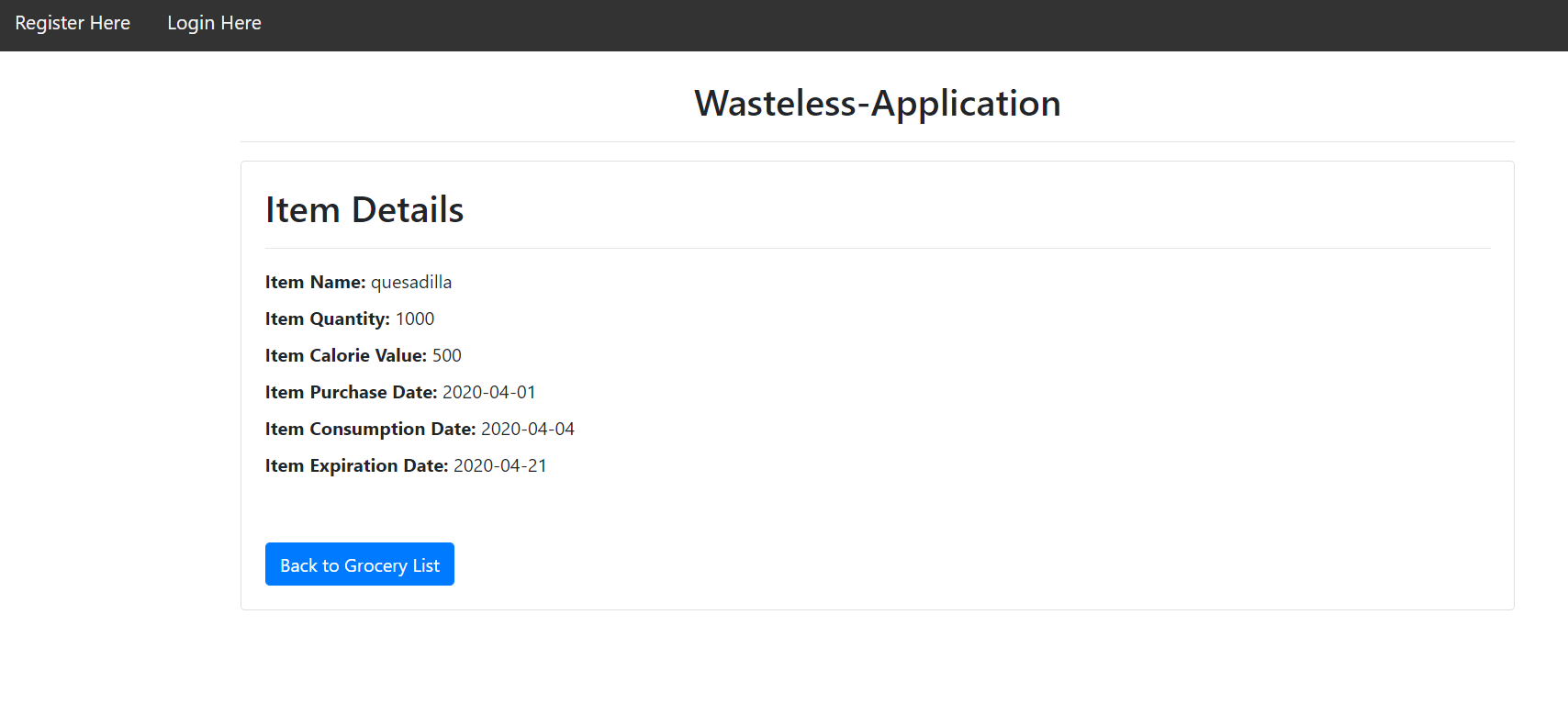
*A completed list looks like this:*

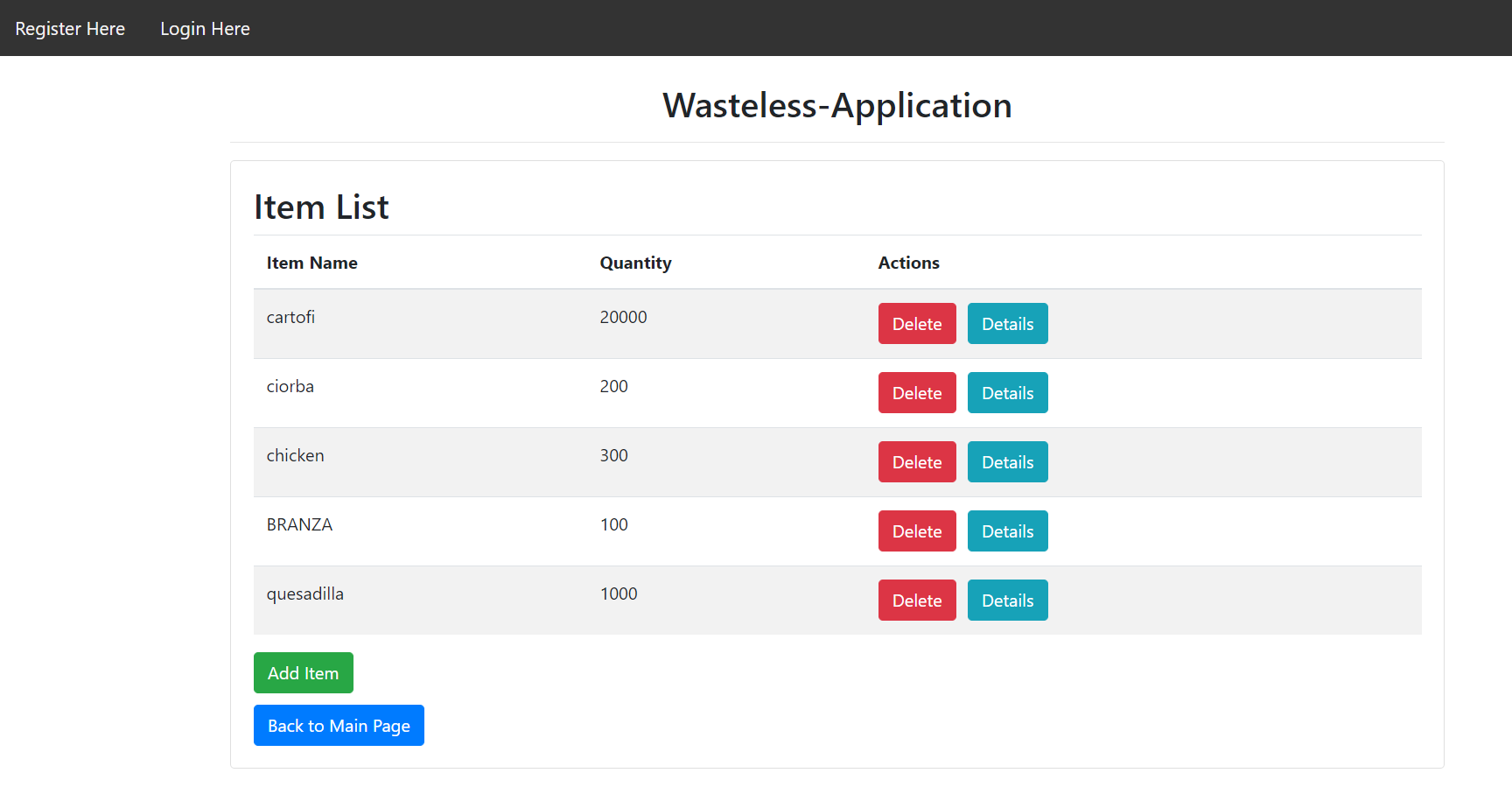


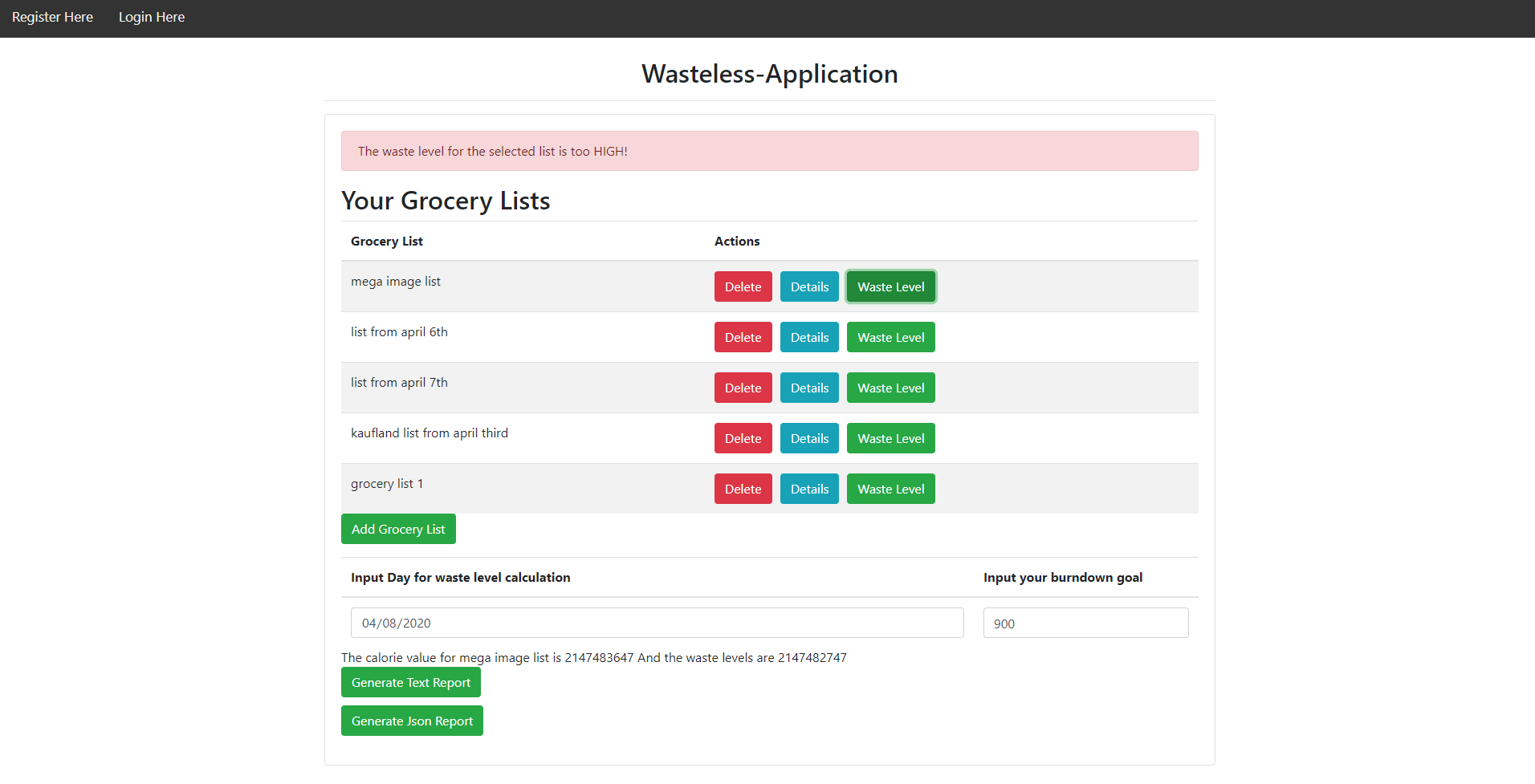
*If you click on Add Item, you are taken to the Add Item page:*



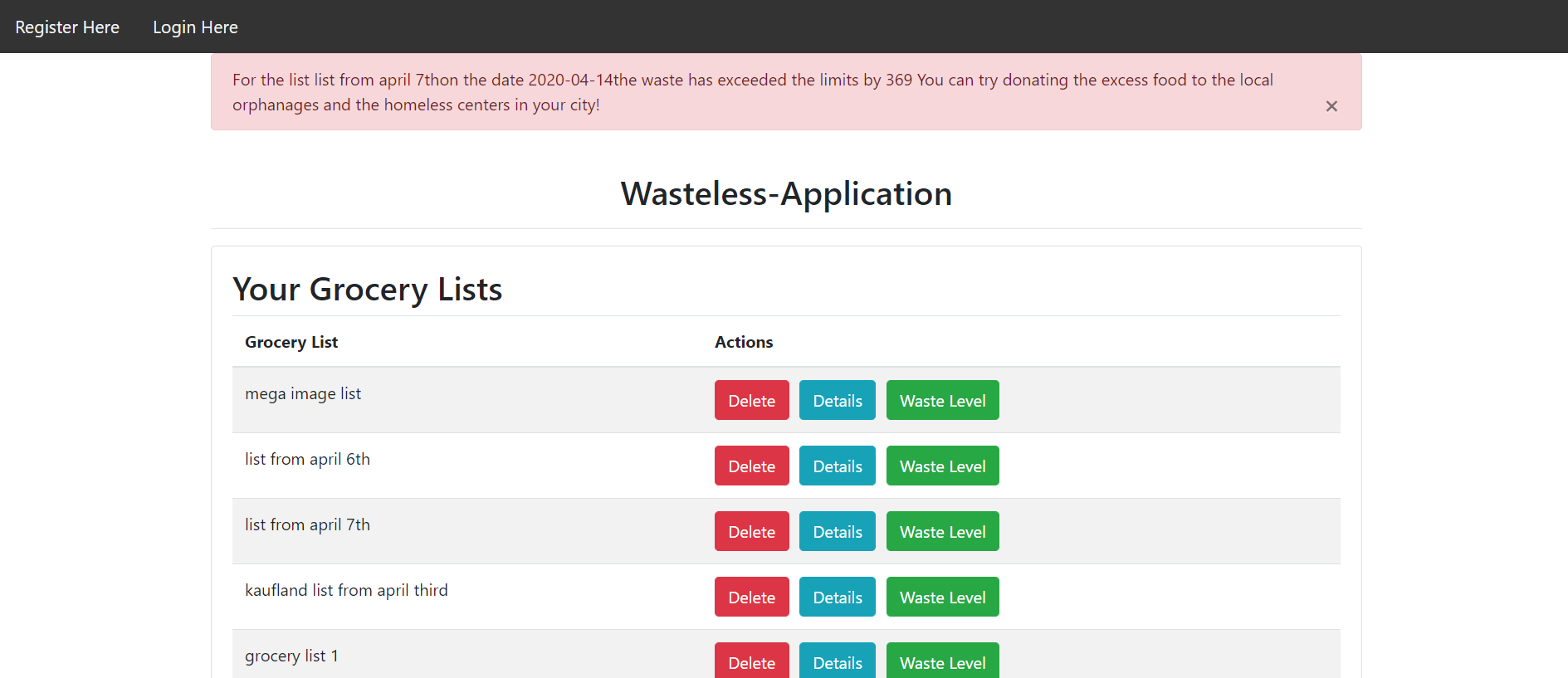
*If you click on the Details button, then you are taken to the Item Details page, where you can see all the information about a certain item:*



* *Delete a list*
* *See the details of a list*
* 
* *Input the day and the burndown goal and press the waste level of a certain list. If the waste level is too high, then, the user will get an error message. If the waste levels were correctly calculated, then the user can choose if they want to generate a text or json report by pressing on the newly appeared buttons: Generate Text Report and Generate Json report.*



*Now, if the waste levels are too high and if the user inputs the goal and the calculation day, donation suggestions on the top of the main page will appear, offering local food donation suggestions:*



8. Bibliography

<https://www.baeldung.com/java-observer-pattern>

<https://angular.io/tutorial/>

<https://www.javatpoint.com/angular-spring-crud-example>

<https://www.baeldung.com/spring-websockets-sendtouser>

<https://www.baeldung.com/cqrs-for-a-spring-rest-api>

<https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm>