WasteLess

Student:Rus Tudor

**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

WasteLess is an app that helps users manage food waste.

# Functional Requirements

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 system provides you with options to donate excess food to various local food charities and soup kitchens and notifies you of them prior to item expiration.

# Non-functional Requirements

The non-functional requirements are:

* The user should not have access to another user’s data
* The user should not be allowed to have access to any data without logging in

2. Use-Case Model

Use case: User login

Level: user-goal level

Primary actor: user

Main success scenario: the user introduces the correct username and password and is redirected to the main app page

Extensions: user introduces a wrong username or password => error message is displayed



3. System Architectural Design

**3.1 Architectural Pattern Description**

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

The backend implements the Layered Architecture Pattern and CQRS Architecture Pattern.

**Layered Architecture Pattern**

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

**Client-Server Architecture Pattern**

A Client-Server Architecture consists of two types of components: clients and servers. A server component perpetually listens for requests from client components. When a request is received, the server processes the request, and then sends a response back to the client. Servers may be further classified as stateless or stateful. Clients of a stateful server may make composite requests that consist of multiple atomic requests. This enables a more conversational or transactional interactions between client and server. To accomplish this, a stateful server keeps a record of the requests from each current client. This record is called a session.

**CQRS** **Architecture** **Pattern**

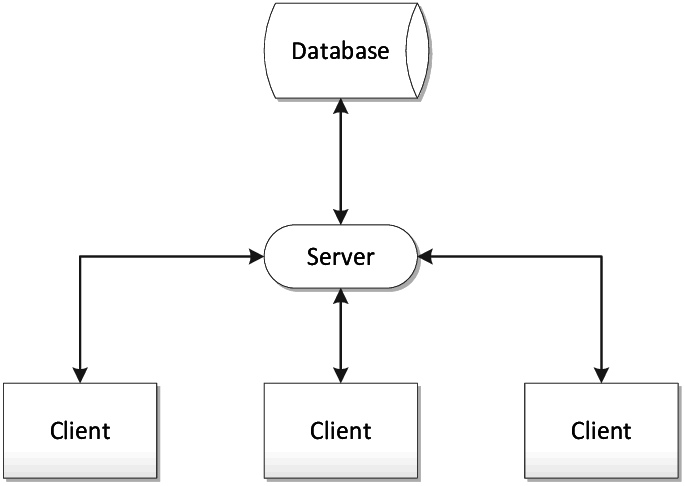
The Command and Query Responsibility Segregation (CQRS) pattern separates read and update operations for a data store. Implementing CQRS in your application can maximize its performance, scalability, and security. The flexibility created by migrating to CQRS allows a system to better evolve over time and prevents update commands from causing merge conflicts at the domain level.

**3.2 Diagrams**

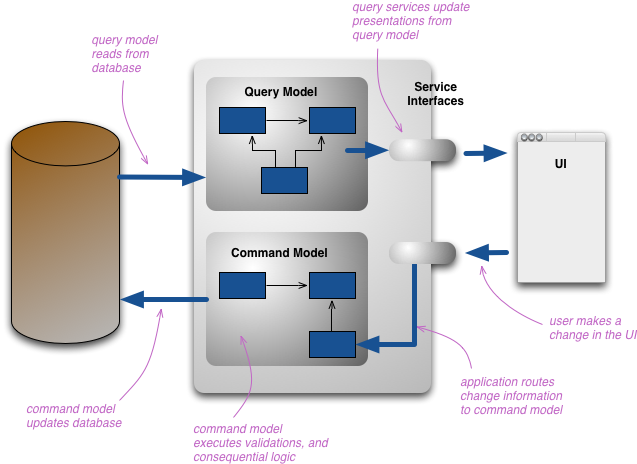
Layered architecture diagram:



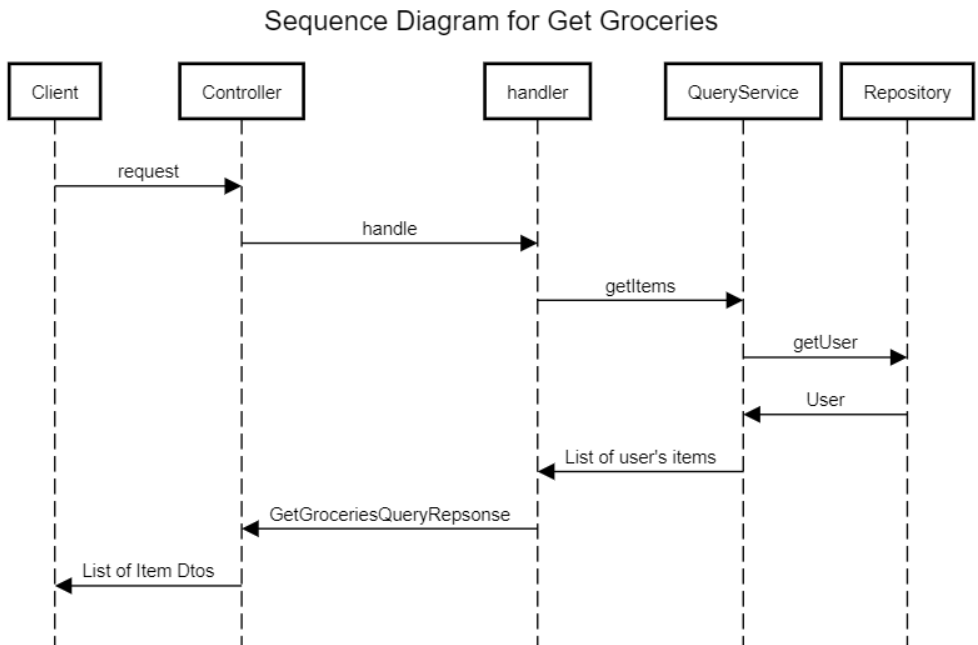
Client-Server diagram:



CQRS diagram:



4. UML Sequence Diagrams



**5 Class Design**

**5.1 Design Patterns Description**

Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

****

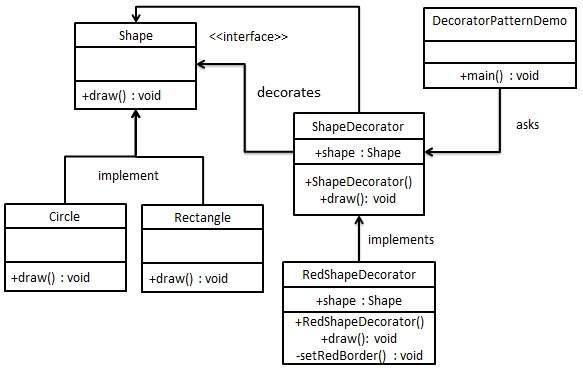
Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

****

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

We are demonstrating the use of decorator pattern via following example in which we will decorate a shape with some color without alter shape class.

****

5.2UML Class Diagram

On github

6. Data Model

Item:

Id, name, quantity, calories, purchaseDate, expirationDate, consumptionDate, userFK

Login:

Id, username, password, userFk, role

User:

Id, name, goal, email, groceryList, login

7. System Testing

Unit tests for simple use cases.

8. Bibliography

<https://www.tutorialspoint.com/>

<https://stackoverflow.com/>