UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Poosh App

Name: Olar Paula
Group: 30235

Table of Contents

Deliverable 1

Project Specification

The project is an inspirational Lifestyle Page for that helps people within daily choices regarding health and wellness, skincare, food plan and overall helping with the satisfaction of a good life.

There are two types of users: clients (regular users), and administrators. Both have to provide an email address and a password in order to register/login and have different requirements/permissions. The page incorporates different features as reading articles, buying products and food plans, subscription to newsletter and so on. There is only one administrator and any number of users.

The administrator can make all the operations that a user does and even more! The admin is able to add, update or delete the products, skin products and supplements, and can add/modify the nutrition profile of a user. He can view the activity of all users and can generate reports based on users details.

A user can make updates only on its own profile. He can complete the available profile and receive the corresponding nutrition profile based on the measurements. The user can add products to his wishlist and basket and make an order.

Functional Requirements

The clients (called regular users) and administrators have different requirements and features.

The administrator can perform the following:

- *CRUD operations on regular users*
- *CRUD operations on all products*
- *Generate pdf reports with the details of users/products*
- *Send emails to users*
- *View users activity on the site*

The regular user can perform the following:

- *View and modify personal account/View order history*
- *Lost password recovery by sending verification code via email*
- *Add products on wishlist*
- *Offer review for products*
- *Subscribe to newsletter*
- *Visualize products sort by category, price*

- *Complete user profile*
- *Visualize nutrition profile*
- *Receive notifications when new products are added*

Use Case Model

Use Cases Identification

Use-Case: Create user account
Level: User Level Goal
Primary Actor: User
Main success scenario: Sign in - add personal information - account created
Extensions: login – wrong sign in data
login – already used email address – user already in database

Use-Case: Add user
Level: Sub function
Primary Actor: Administrator
Main success scenario: Login - add user information - user added to database
Extensions: login – wrong login data
login – add user information – user already in database

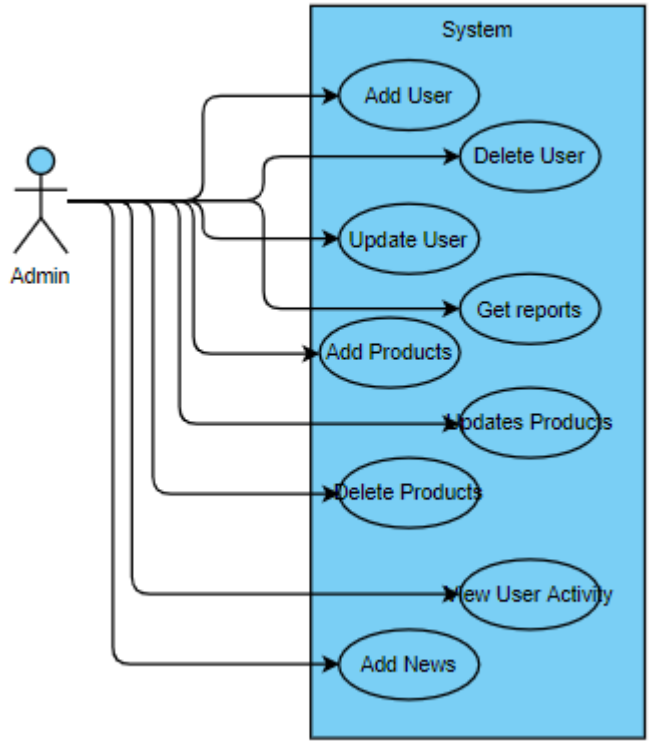
Use-Case: Buy product
Level: Sub function
Primary Actor: User
Main success scenario: Login - add product to basket - pay product
Extensions: login – wrong login data
login – add product to basket – product out of stock

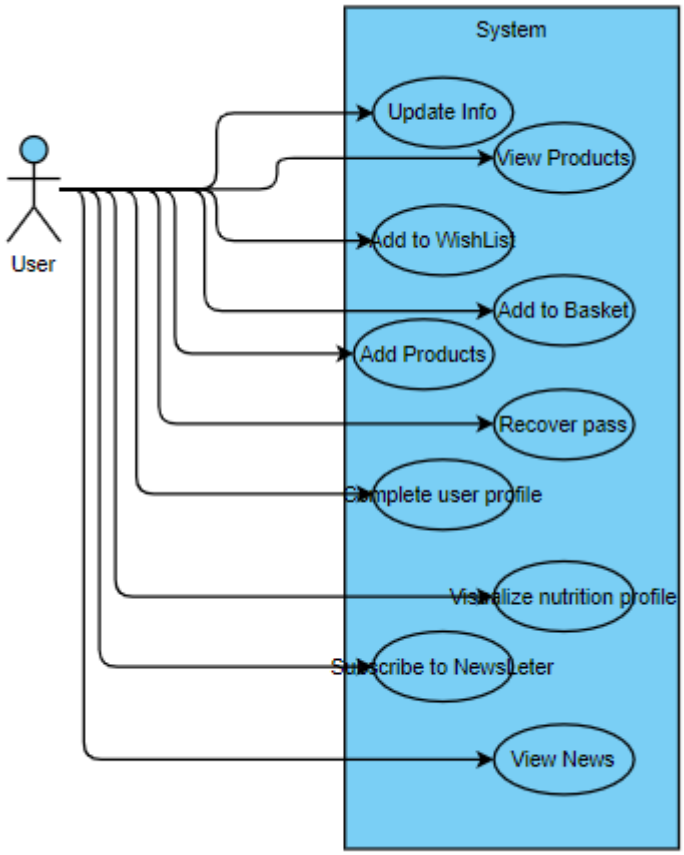
Use-Case: Delete user
Level: User Goal Level
Primary Actor: Administrator
Main success scenario: Login – delete user information - user deleted from database
Extensions: login – wrong login data
login – delete user information – trying to delete inexistent user

Use-Case: Subscribe to newsletter
Level: Sub function
Primary Actor: User
Main success scenario: Login – subscribe to newsletter
Extensions: login – wrong login data

UML Use Case Diagrams

The UML Use-Case Diagrams.





Supplementary Specification

Non-functional Requirements

Usability targets the aspect of how hard is to use the implemented product. The interface has to be user-friendly and easy to use based on the criteria: learnability, efficiency and satisfaction (is the design pleasant to use).

Access security tells how the system and its data is protected. Practicle examples are: passwords shall never be viewable at the point of entry or at any other time and users shall receive notification of profile changes via email (or maybe phone) of record when profile.

Performance describes how the implemented solution behaves when users interact with it in various scenarios. Idealy, the load time should not be more than one (few) second for users.

Confidentiality is the degree to witch the softare system protects sensitive data that allows only authorized access to the data. A good example for this application is no sensitive cardholder retention. The website will not retain customer credit or debit card information entered during the checkout payment processing.

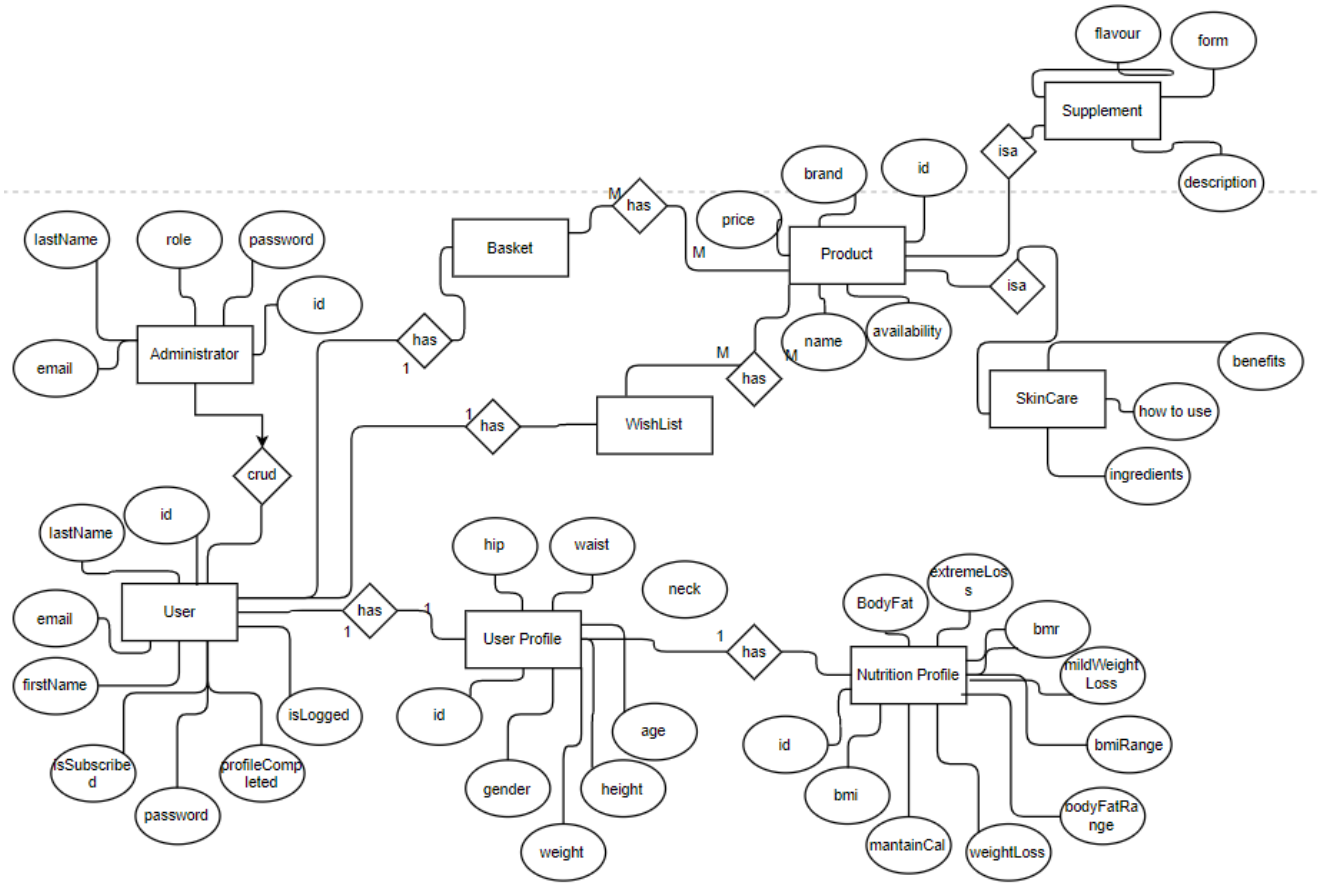
Design Constraints

Design constraints represent design decisions that have been mandated and must be adhered to. Some of the design constraints such as functional and non-functional requirements were previously presented, with the possibility(probability) of improvement. Technical contraints in the software architecture are the programming language - java, operating system or platform supported – windows, the use of a set of libraries and frameworks as Hibernate, Thymeleaf. The Data Store type is SQL, and it is used a low-level API in order to deal with it (JDBC). Between the Data Store and the application is an abstraction layer, ORM. ORM maps Java Class with Database Table. Aside from a computer and an internet connection, software process requirements are a text or HTML editor, java editor, web browsers, development tools.

Glossary

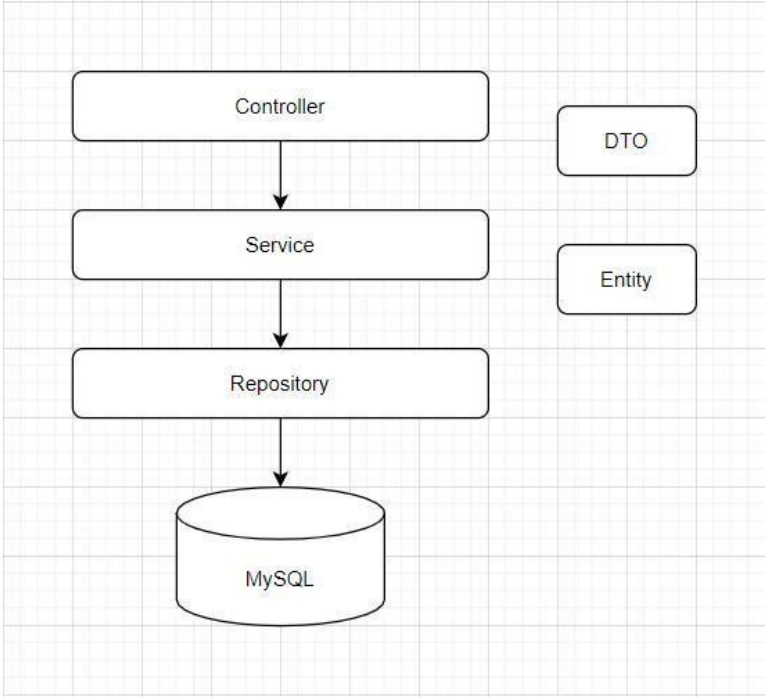
Deliverable 2

Domain Model



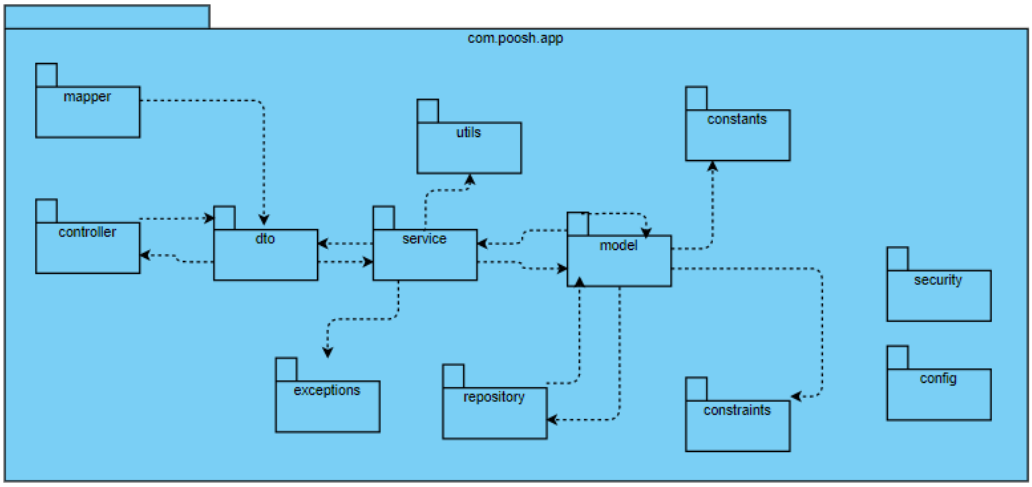
Architectural Design

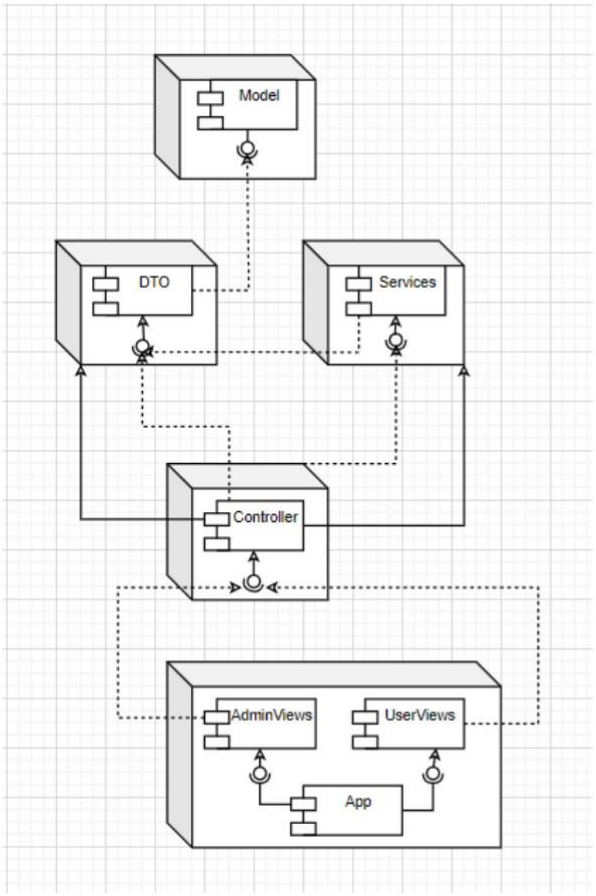
Conceptual Architecture

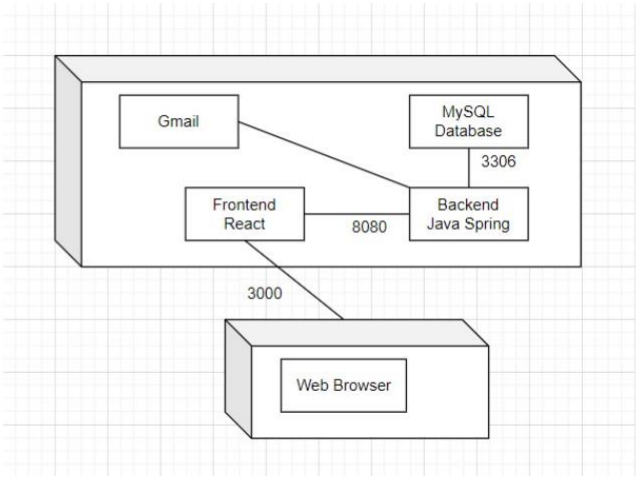


Package

Component and Deployment Diagram




Model
DTO
Services
Controller
AdminViews
UserViews
App


Gmail
MySQL
Database
3306
Frontend
React
8080
Backend
Java Spring
3000
Web Browser

Deliverable 3

Design Model

Dynamic Behavior

Diagrama de secventa pentru recuperarea parolei

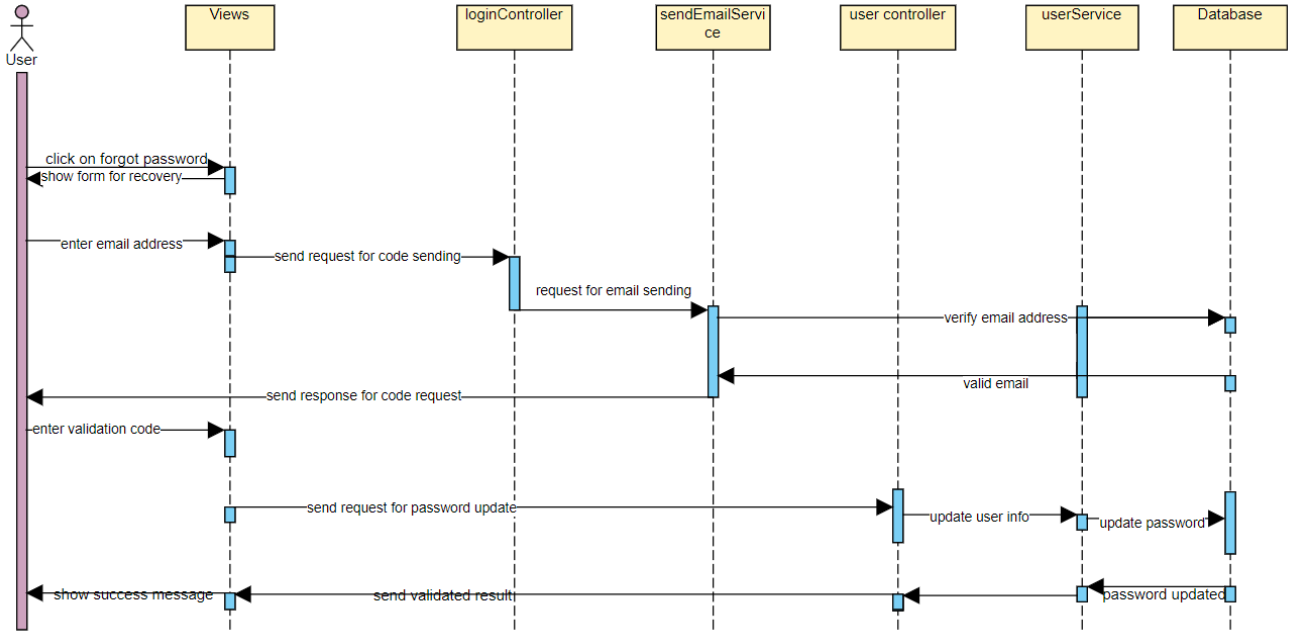
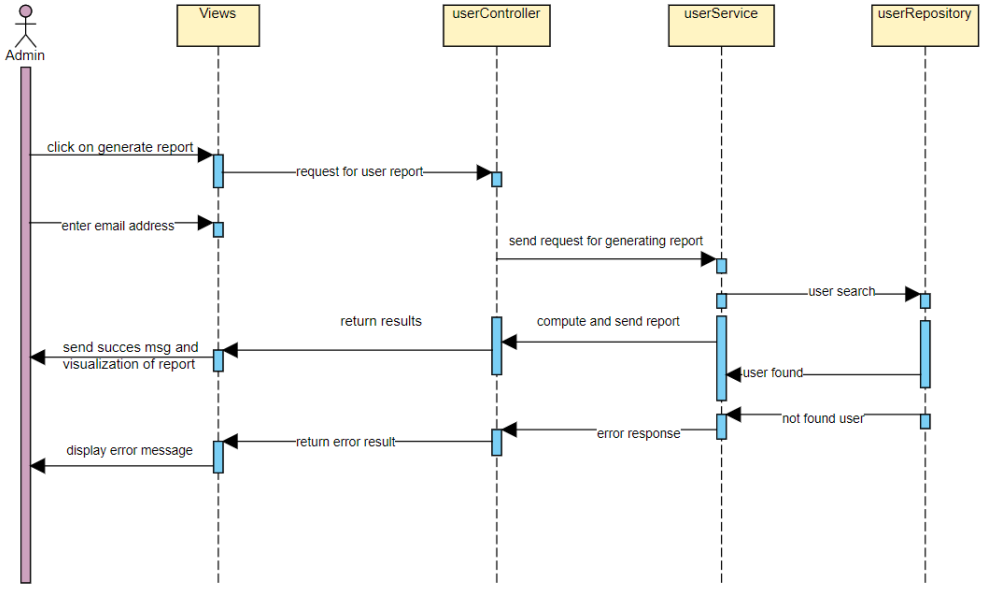
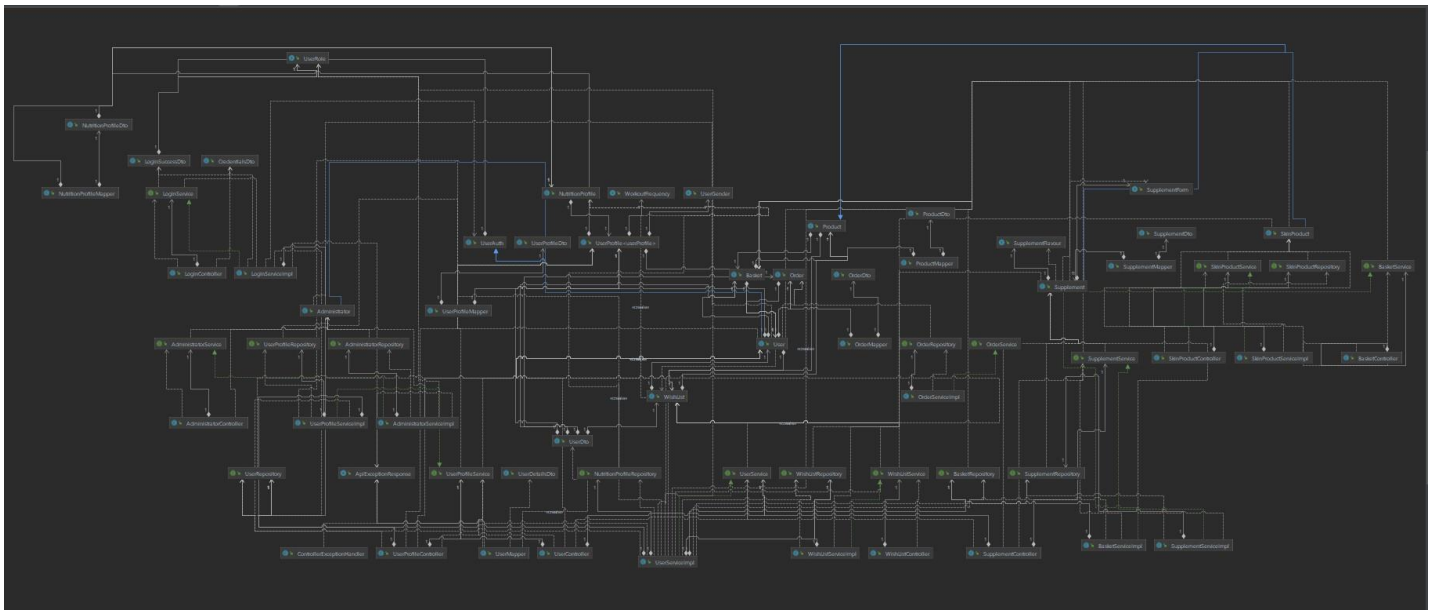


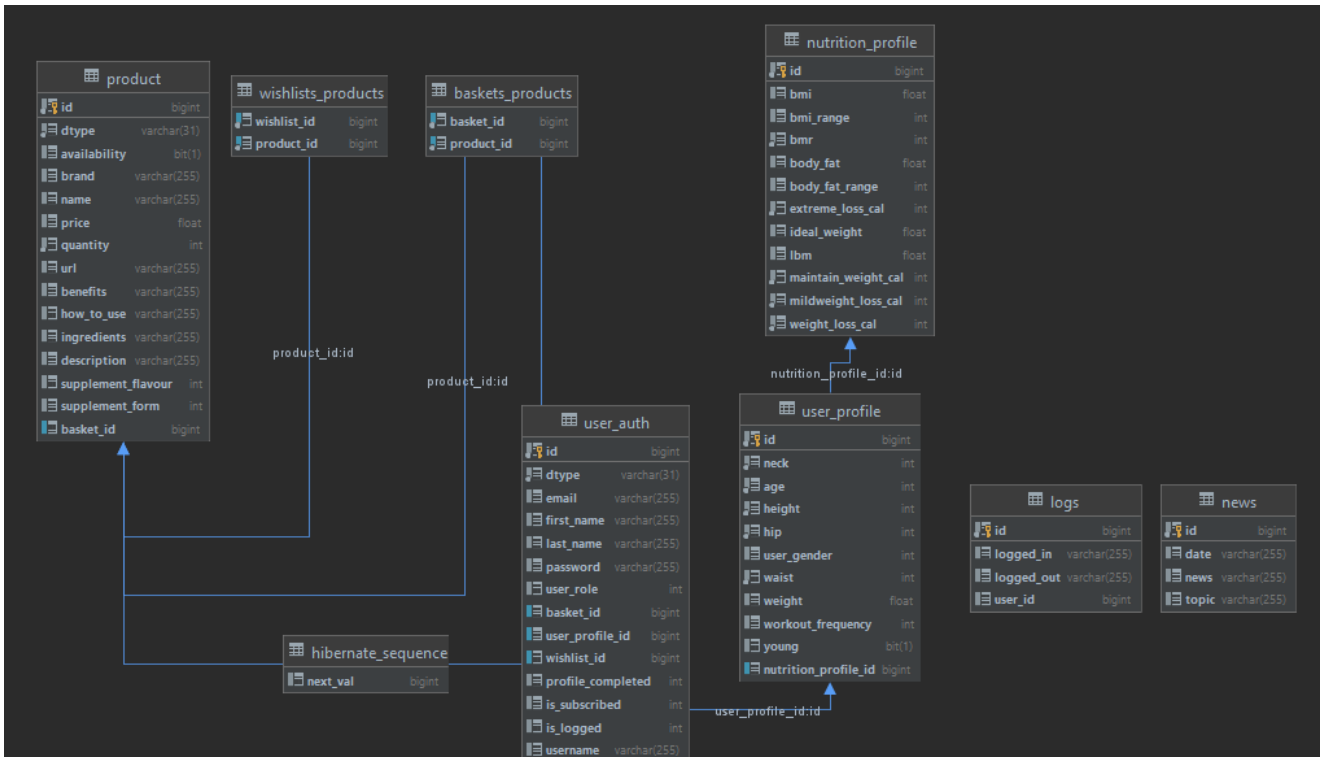
Diagrama de secventa pentru generare report



Class Diagram



Data Model



System Testing

Test cases implemented by me are for a user, more exactly, searching for a user by existing first name in database, searching for a user by non-existing name in database, searching user by existing first name and updating last name, and adding new user.

The first test case is searching for a user by an existing first name in the database. Signature is givenExistingFirstName_whenFindByFirstName_thenFindOne() and test that found user is not null and has the first name with the one provided. We search the user and we assert assertNotNull(user), assertEquals(FIRST_NAME, user.getFirstName()).

```
@Test
void givenExistingFirstName_whenFindByFirstName_thenFindOne() {
    //given
    userService = new UserServiceImpl(userRepository,null, null, null, null);

    //when
    User user2 = userService.findByFirstName(FIRST_NAME);

    //then
    assertNotNull(user2);
    assertEquals(FIRST_NAME, user.getFirstName());
}
```

The second test case is searching for a user by a non existing first name in the database. Signature is givenNonExistingFirstName_whenFindByFirstName_thenReturnNull() and test that user with providing name does not exist in database.. We search the user and we assert assertNull(user).

```
@Test
void givenNonExistingFirstName_whenFindByFirstName_thenReturnNull() {
    userService = new UserServiceImpl(userRepository,null, null, null, null);

    User user2 = userService.findByFirstName(NON_EXISTING_NAME);

    assertNull(user2);
}
```

The third test case is searching for a user by an existing first name in the database and updating its last name. Signature is givenExistingFirstName_whenFindByFirstName_thenUpdateLastName() and for the found we change the last name with the one provided and test if it has changed. We search the user, change name and assert assertNotNull(user), assertEquals(LAST_NAME, user2.getLastName()).

```
@Test
void givenExistingFirstName_whenFindByFirstName_thenUpdateLastName() {
    //given
    userService = new UserServiceImpl(userRepository,null, null, null, null);

    //when
    User user2 = userService.findByFirstName(FIRST_NAME);
    user2.setLastName(LAST_NAME);
    userService.addUser(user2);

    //then
    user2 = userService.findByFirstName(FIRST_NAME);
    assertNotNull(user2);
    assertEquals(LAST_NAME, user2.getLastName());
}
```

The last testcase consists in adding a new user to database and then searching to see if we find it. Signature is addUser() and we add the user with given name, we search it by that name and we assert assertNotNull(user) and assertEquals(NEW_NAME, user.getFirstName()).

```
//    @Test
//    void addUser() {
//
//        //given
//        userService = new UserServiceImpl(userRepository,null, null, null, null);
//
//        //when
//        User user = new User(null, null, "NEW_NAME", null,null, null, null, null, null, null);
//        userService.saveUser(user);
//
//        user = userService.findByFirstName("new_name");
//
//        assertNotNull(user);
//        assertEquals(NEW_NAME, user.getFirstName());
//    }
}
```

Future Improvements

Some future improvements for my application would be a wider range of products for users to choose from and the integration of meal plans to buy based on user preferences. It would be nice for users to have a section of chatting or article reading and maybe some interactive tutorials about health and wellness.

Conclusion

This project is a great opportunity to deepen my knowledge in java programming, spring boot framework and react. It incorporates both front-end and back-end and it challenges your practices and techniques and creativity..

Bibliography