VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS

Software Engineering project

Food E-shop for healthy people

Done by:
Linas Ramanauskas
Nojus Makulavičius
Agnė Jankauskaitė
Miglė Morta Pumputytė
Matas Daunoras

Vilnius
2020

Contents

1 Business Case

This is a short explanation of our team and system

1.1 Team

Team name: WellFed

Member	area of interest
Linas Ramanauskas	leader/ security and front end development
Nojus Makutavičius	Database management and its implementation
Agnė Jankauskaitė	Using test-driven development (TDD).
Miglė Morta Pumputytė	Front end
Matas Daunoras	Communication implementation

1.2 Product or service

The software we are building is a simulation of a food E-shop.

1.2.1 Purpose and mission

There are a lot of people in the world that try to get fit. However, to do so you also need a proper diet. That is why our system will include calorie calculations for the sum of food people will order according to the number of calories the user has in his/her “cart”.

1.2.2 System description: user’s perspective

The result of this system for users will be the reduced research time for calorie amount in food and calculations as well a good way to order food.

1.2.3 System description: provider’s perspective

This is another perk that this system will have in comparison to other food delivery service programs. Since this system is meant for fitness and maintenance of a diet, in theory this should attract a lot healthy people or people that wish to live a healthier lifestyle.

1.2.4 Interface prototypes

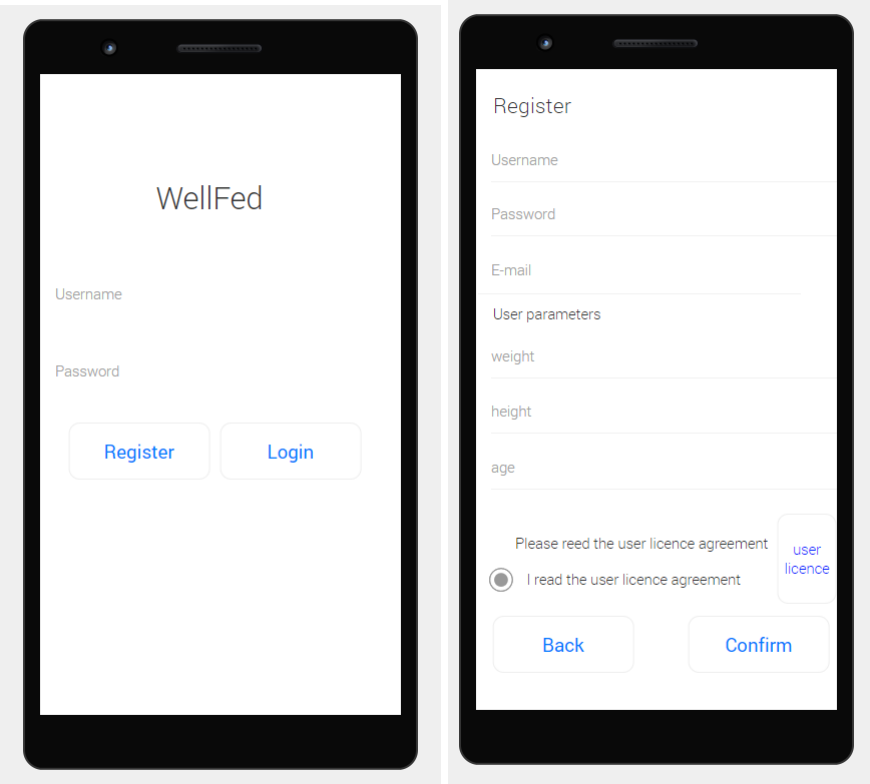

Figure 1: Login/Register screens

In the first screen you will see the login screen where the user will need to enter the username or password. If the user does not have an account created, he/she will need to create one by pressing the register button. There you will be required to to enter a unique username, password, e-mail address and user parameters. Hence we will use these parameters for calculations and store them in a database, the user must agree with the end-user licence agreement.

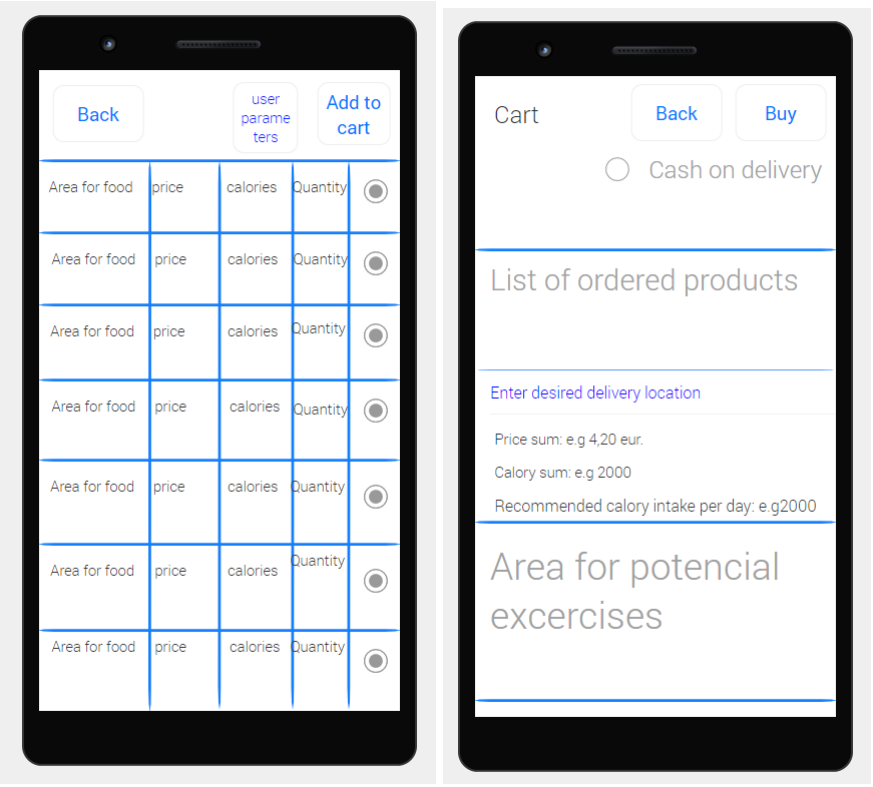

Figure 2: Products list and Cart screens

After the login or register process the user will have the option to order food. Hence our focus is food and health there will be sections for food, price, calories and quantity which the user can specify. The user will be able to choose multiple products and add them to the 'cart'. In the cart screen there will be a list of chosen products displayed on screen. Calculated price sum, calorie sum and a recommended calorie intake that is based on the users parameters. In the future we will add potential recommended exercises for the user to burn out the consumed calories, the cash on delivery option must be checked.

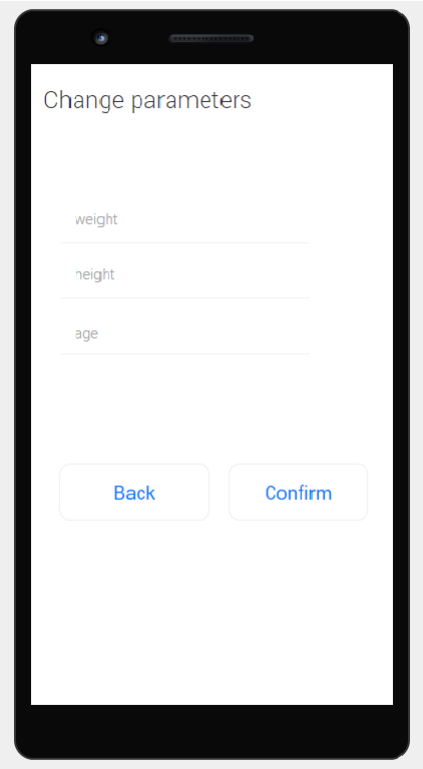

Figure 3: User parameters screen

The user will have the option to update his/her parameters. If pressed the user parameters button on the product menu screen the user will have the option to change the user parameters (weight, height, age).

2 System Functionality

This area describes the functionality of how the system should work and interact with the user.

2.1 Functional requirements

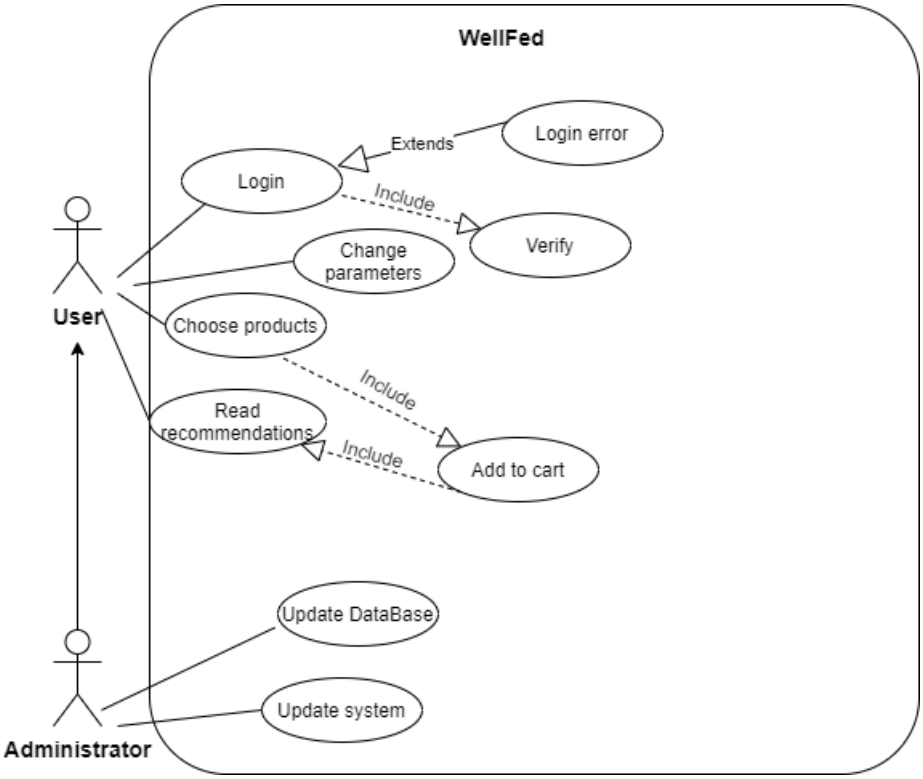

Figure 1: Use Case diagram

The user will be able to login, change parameters, chose food to order and reed recommendations. The user when choosing food will have the option change his/her parameters (Weight, Height, Age...). The user can choose multiple foods and add them to his/her cart. From the cart section the user can reed recommendations and buy the listed products. The admin can update the database and system.

Actor Role/Use Case	User	Administrator
Update system		
Update Database		
Login		
Register		
Confirm registration		
Change user parameters		
Add to cart		
Quantity		
Buy		

Figure 2: Use-case matrix

1. **Update system** –System administrator duties include regularly applying software updates in a timely manner to ensure organizational safety against crippling attacks. They also add new features to the program and remove outdated content to make it more productive. Only an administrator can make any changes in the application. Ordinary app users can not do any of them.

2. **Update Database** – Only an administrator can update system Database. By updating it he fixes various kinds of bugs and helps to avoid performance degradation.

3. **Login** – The user can login to our application „WellFed“ by entering his username and password.

4. **Register** – If a person does not have an account yet, he can click on an icon “register“ and become one by filling out requested information.

5. **Confirm registration** – After filling out the needed personal information about himself in the “Register“ window, the user has to read “User licence“ and agree to our terms. When he can confirm his account.

6. **Change user parameters** – Every account has “User parameters“, when you create an account you have to enter them. However, the users have an ability to change them whenever they want to. The parameters include: weight, height and age.

7. **Add to chart** – After login, the window with our products, their prices and calories they contain, will be displayed. The user has an ability to choose whenever they want and add these goods to their shopping cart.

8. **Quantity** – Next to every product an option to choose the quantity is displayed. It lets the user to select the amount he wants to add into his cart.

9. **Buy** – The user can buy all his selected products, which he has in his cart.

2.2 Non-functional requirements

The system should be easy for the users to navigate, the system should work well on most android devices. For reliability and confirmation the system should provide the user a screen with user end license agreement which states that we will not use the user data for ill will and the data will be purely used for calculations and user authentication.

3 Technologies

This section shortly describes the technical part of our system

3.1 Technology stack

We will use the Java programming language per Android studio. The new libraries and tools that will be needed are annotations, junit, kotlin-stdlib, kotlin-stdlib-common, okhttp, okio, PostgREST, Postgres, API database, android API, Docker, Docker-compose. The virtual server will be created in OpenNebula.

3.2 Operational environment

Operational environment	Work
Android	
Linux	
Windows	
Mac	

Figure: Operational environment table

The system is specifically made fore Android systems and will run on the minimum SDK Android 4.1 (JellyBean). This is a system for a mobile device, OS that do not provide mobile support will not work.

3.3 Deployment

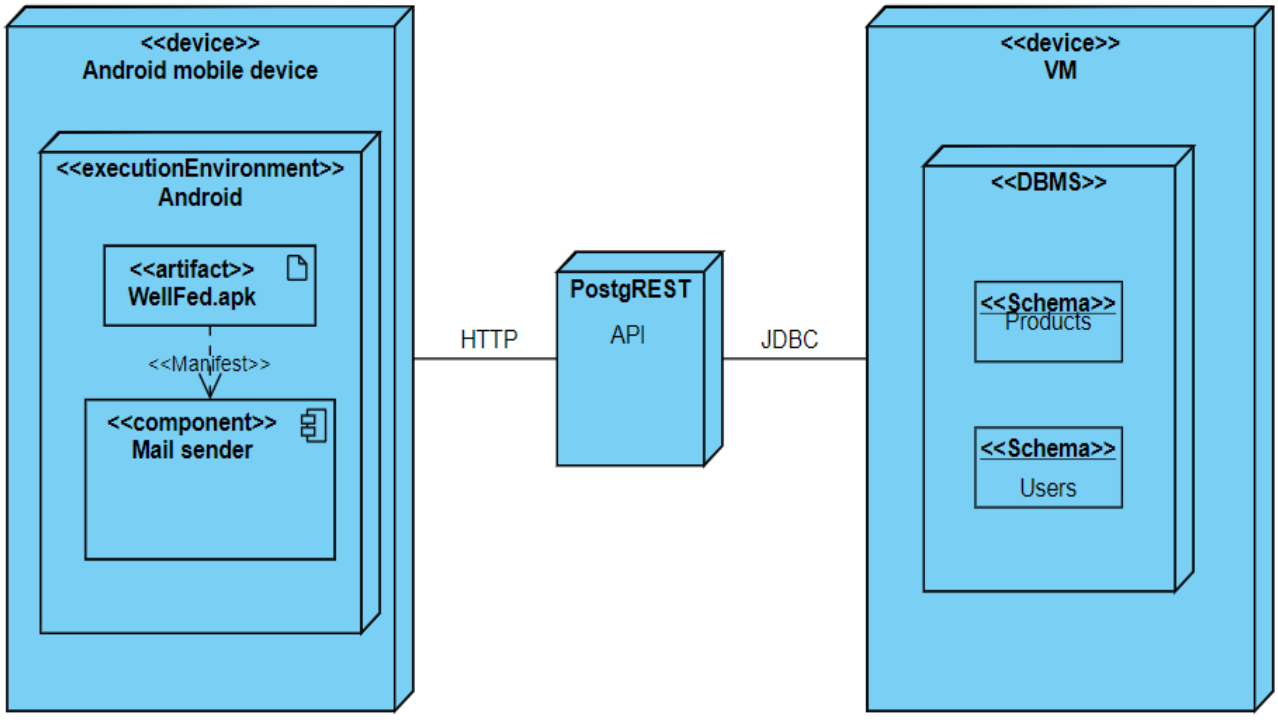

Figure: Deployment diagram

The system will take and insert the users data via a virtual server and take product data which the user can choose from later on in the system. When the user buys the products the system will send a message to the admins Gmail account.

4 Zero Feature Release

This section is about how we will review and test our project and which VCS we have choosen.

4.1 Version control system

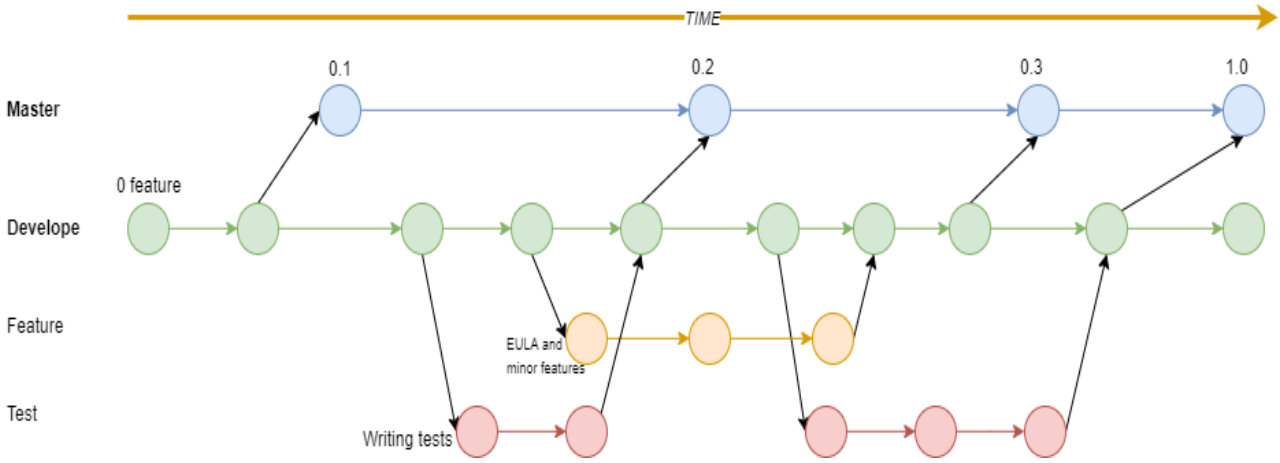

Figure: Git branching model

We will use GitHub for our git repository hosting. Distributed version control system is the one we have decided to use for our project. It means, that every single developer will have their own project copy. They will be able to work locally and disconnected. When they start working on the project they will clone the code from the master repository to their own hard drive. Each of them will make the changes locally, after making them, a request will be sent to the master repository for approval. Finally, they will be able to push their local repository code to the master repository.

Our team has choosen to use Gitflow strategy. This particular branching strategy has two main branches: master and develop. The rest are supporting branches. All main development happens in develop branch. In one of the supporting branches called feature we create minor features, for e.g. implementing EULA(end-user license agreement). From development branch we have test branch, where we test our code, after testing is completed, this branch merges back to develop. When fixing is done in the development branch, we release a new version which is added to the main branch.

4.2 Code reviews

For our project reviews our team will use GitHub. First, a project member creates a branch, when he works at it, creates commits and commit messages, which are mandatory, about their work. Commits will create a history of his work that others can follow to understand what he has done and why. After finishing his work, he will open a Pull Request By using GitHub's @mention system. In his Pull Request message, he will ask for feedback from other colleagues. When all teammates will finish discussing and reviewing his code, he will change and correct it according to the feedback. Before merging his code to the main branch, he will deploy his code into android studio and only after checking that everything works correctly, he will add these commits to the main branch.

4.3 Test automation

```
public class ExampleUnitTest {

    @Test
    public void getSHA_and_toHexString_test() throws NoSuchAlgorithmException
    {
        UActions uactions =new UActions();

        byte [] bit =uactions.getSHA( input: "test123");
        String hex = uactions.toHexString(bit);

        Assert.assertEquals( expected: "ecd71870d1963316a97e3ac3408c9835ad8cf0f3c1bc703527c30265534f75ae",hex);
    }

}
```

Figure: Test example of password hashing

Our team will write code test in android studio "wellfed" project, package test. We will write short programs that exercise our system under test and check if our expectations are met. To run our test code we use Gradle wrapper commands (e.g. gradlew test). This Gradle command will also be used to automate our tests. Currently we have one minor test for correct hashing of the users password. At the end of the project our main goal is to test login/register functionality and if we will have time we will also test other minor functions.

4.4 Build and CI/CD pipeline

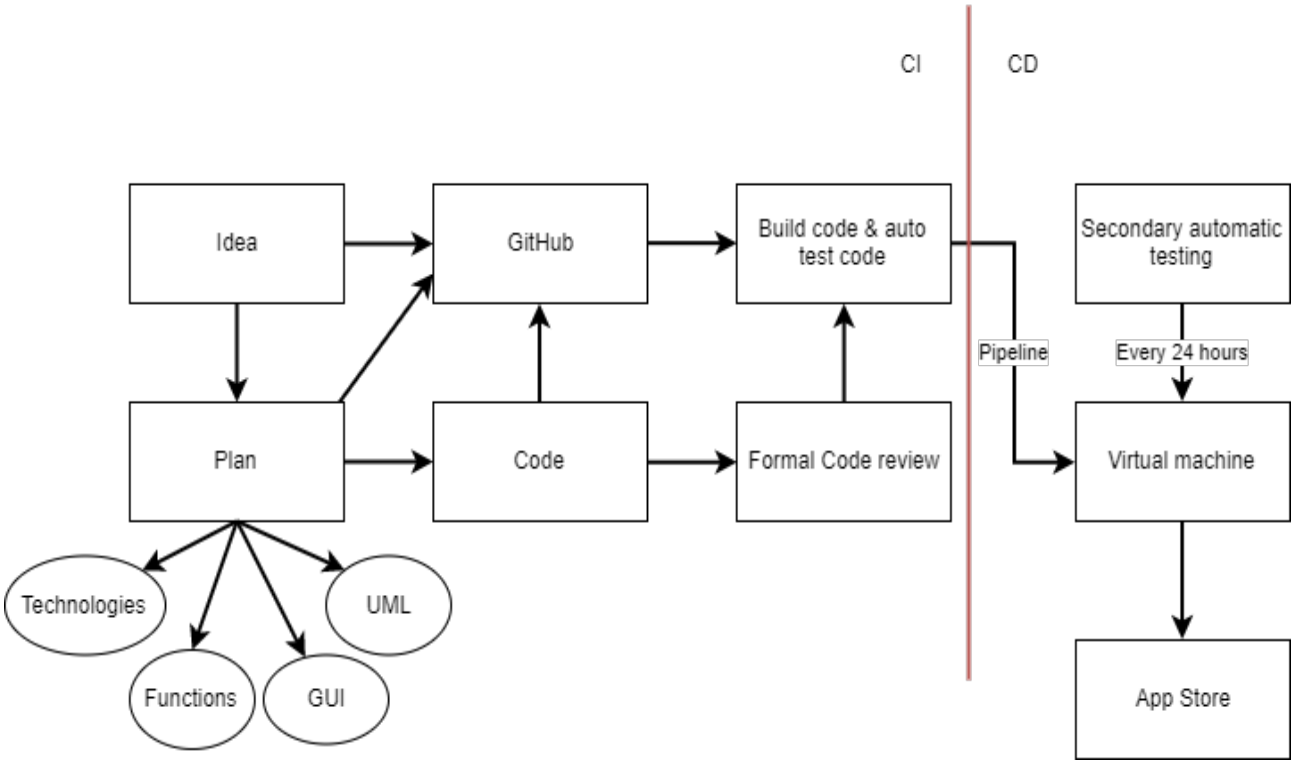

Figure: CI/CD Pipeline diagram

The CI/CD pipeline works when someone pushes the code in to the developement branch, then the team reviews the code to check if the changes matched the system idea. If agreed upon the changes the developer is allowed to merge the updated code into the master branch. The code then gets auto tested and built. Then every 24 hours our development test virtual machine, using the taskscheduler from windows, will take the code from our git repozitory, update the existing code and do secondary automatic tests and we will observe the outcomes. In theory, by using this dual test method we can further guarantee the sucessfull finall version and launch of our app.

5 Project plan

In this section we describe the project plan for our team, work schedule and risks which may occur along the process

5.1 Issue tracking / Planning tools

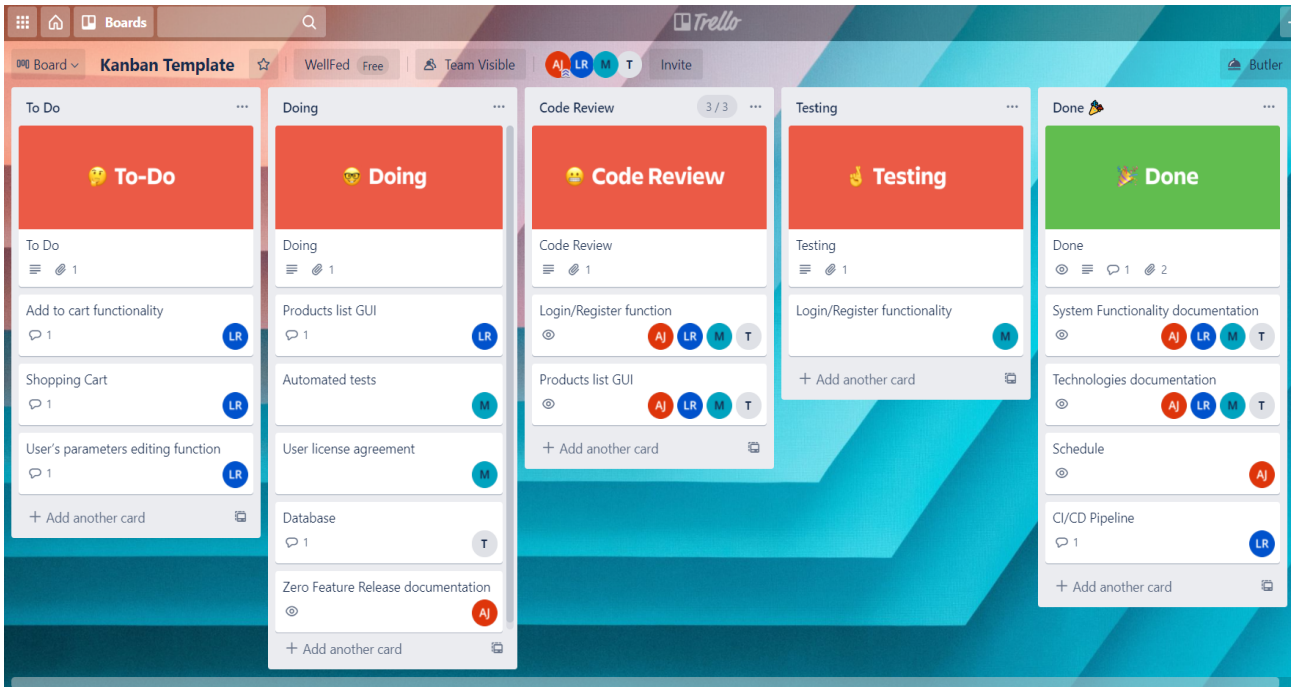

Figure: An example of Trello usage

For our project tasks and defects tracking we will use trello visual tool. We have chosen Kanban template. This particular template has diferent workflow stages. All of these stages are in separate columns. Our team members will have an ability to see, which task are available, which are in progress and what is completed. They will also be able to add comments.

5.2 Work breakdown

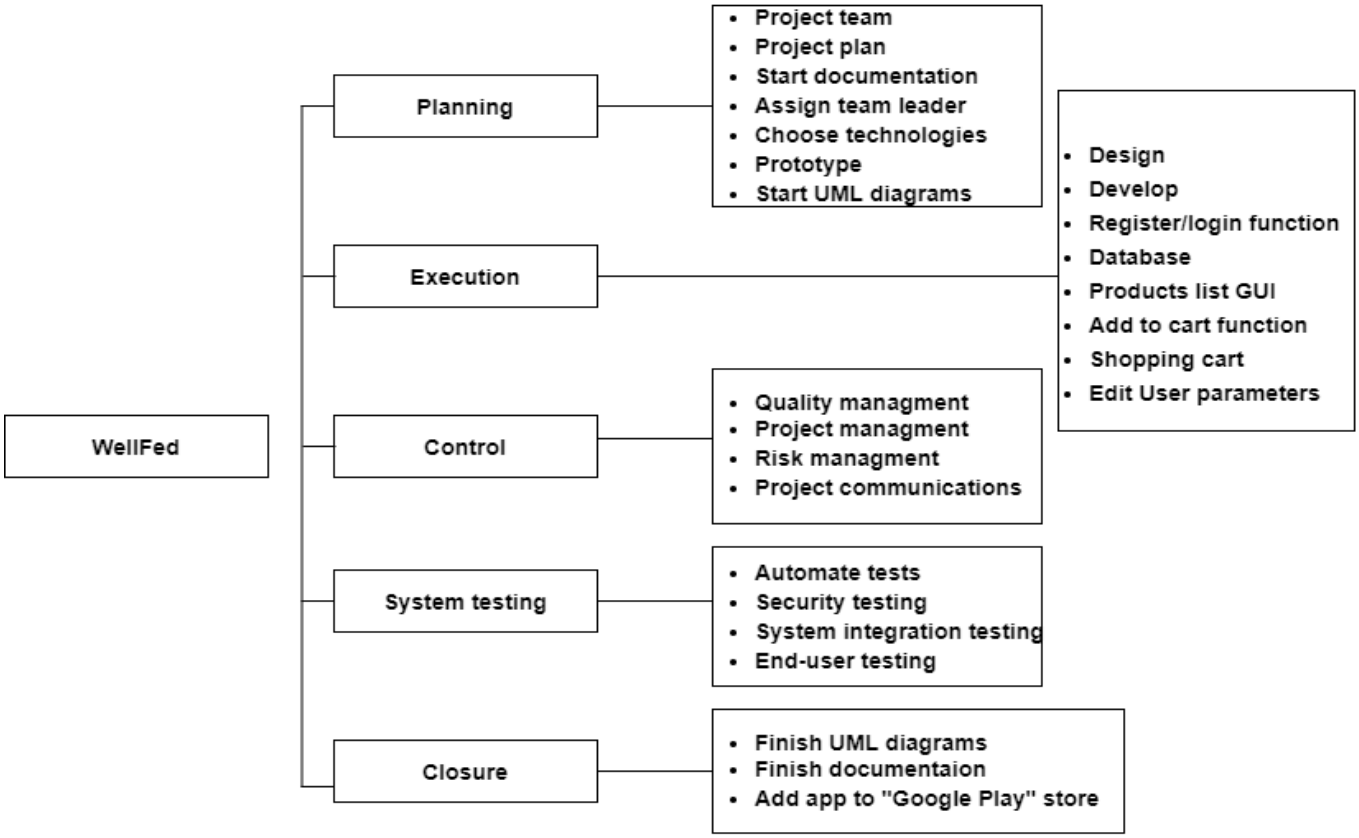

Figure: WBS diagram

This is our work breakdown structure. We use it to reduce more complicated activities to a collection of tasks.

5.3 Schedule

The estimated time for our project development is 4 months. We have created a schedule for each month that we must do, to finish this project according to deadline.

SEPTEMBER 2020						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
30	31	**1**	**2**	**3**	**4**	5
6	**7**	**8**	**9**	**10**	**11**	12
13	**14** Introduction and administrative matters	**15**	**16**	**17**	**18**	19
20	**21**	**22**	**23**	**24**	**25**	26
27	**28** Project team creation	**29** Generate ideas and choose topic (everyone)	**30**	1	2	3

Figure 1: September

OCTOBER 2020						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
27	28	29	30	**1**	**2** Present our team members and project	3
4	**5** Business Case documentation (Linas)	**6** Business Case documentation (Linas)	**7** System Functionality documentation (everyone)	**8** Technologies documentation (everyone)	**9** Technologies documentation (everyone)	10
11	**12** Review 1 delivery (everyone)	**13** Upload 1 delivery (Linas)	**14** **1 Delivery deadline**	**15**	**16**	17
18	**19** Download and install android studio, create work plan (everyone)	**20** Add all necessary libraries to android studio (everyone)	**21**	**22**	**23** Zero Feature Release documentation (Agne)	24
25	**26** Create main application page (Linas)	**27** Create products list GUI (Linas)	**28** Connect to GitLab and upload project (everyone)	**29** Create Database (Nojus)	**30** Add register and login function (Linas)	**31** Project plan documentation (Agne)

Figure 2: October

NOVEMBER 2020						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
1 Create products list GUI (Linas)	2 Create CI/CD Pipeline (Linas)	**3** Create CI/CD Pipeline (Linas)	**4** Create User license agreement (Miglė)	**5** Create Database (Nojus)	**6** Create Database (Nojus)	7 Correct documentation (Agne) Program testing (Miglė)
8	**9** Review 2 delivery (everyone)	**10** Upload 2 delivery (Linas)	**11** **2 Delivery deadline**	**12**	**13** Create products list GUI (Linas)	14
15	**16** Create add to cart function (Linas)	**17** Create shopping cart (Linas)	**18** Create shopping cart (Linas)	**19** Building the System documentation (Agne)	**20** Creating UML diagrams (Agne)	21 Creating UML diagrams (Agne) Program testing (Miglė)
22	**23** Create user's parameters editing function (Linas)	**24**	**25** Correct documentation (Agne)	**26** Add user's information to Database (Nojus)	**27** Add user's information to Database (Nojus)	28 Program testing (Miglė)
29	**30** Fix UML diagrams, correct documentation (Agne)	**1** Upload 3 delivery (Linas)	**2** **3 Delivery deadline**	**3** Add products list to Database (Nojus)	**4** Add products list to Database (Nojus)	5 Program testing (Miglė)

Figure 3: November

DECEMBER 2020						
Sunday	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**
29	30 Add database to an app (Nojus)	1 Add database to an app (Nojus)	2 Fix UML diagrams (Agne)	3	4	5 Program testing (Miglė)
6	7 Prepare the app for uploading to "Google Play" store (everyone)	8 Prepare the app for uploading to "Google Play" store (everyone)	9	10	11	12 Program testing (Miglė)
13	14	15	16 Correct documentation (Agne)	17	18 Upload an app to "Google Play" store (Linas)	19 Program testing (Miglė)
20	21	22 **Project presentation**	23	24	25	26
27	28	29	30	31	1	2

Figure 4: December

5.4 Main risks

Main risks	Solution
Lack of Database and Password security	Our team will create advise tips for users in the register window. We will ask our users to make their passwords stronger by using at least 8 characters, uppercase/lowercase letters, numbers and at least one special character
The application might not work on all android versions	We will try to test our app on as many versions as we can before we release our app
Not all functionality is implemented	The most necessary functions to use the app correctly will be done first and later we will try to implement all additional functionality
Not enough time to complete testing	We will test the most vulnerable parts of the program first
Unable to add our app to "Google Play" store	To avoid this issue, our team will prepare the app for uploading to "Google Play" store several days or a week before deadline. As a result, we will have time to fix this problem
The server might crash if too many customers will use it at once	For this particular problem, we will update our software regularly, run various tests

Figure: Main risks

These are the main risks, which our team anticipates to happen to our software development. We came up with solutions, which could help if they will occur to us in the future.

6 Building the System

6.1 Class Diagram

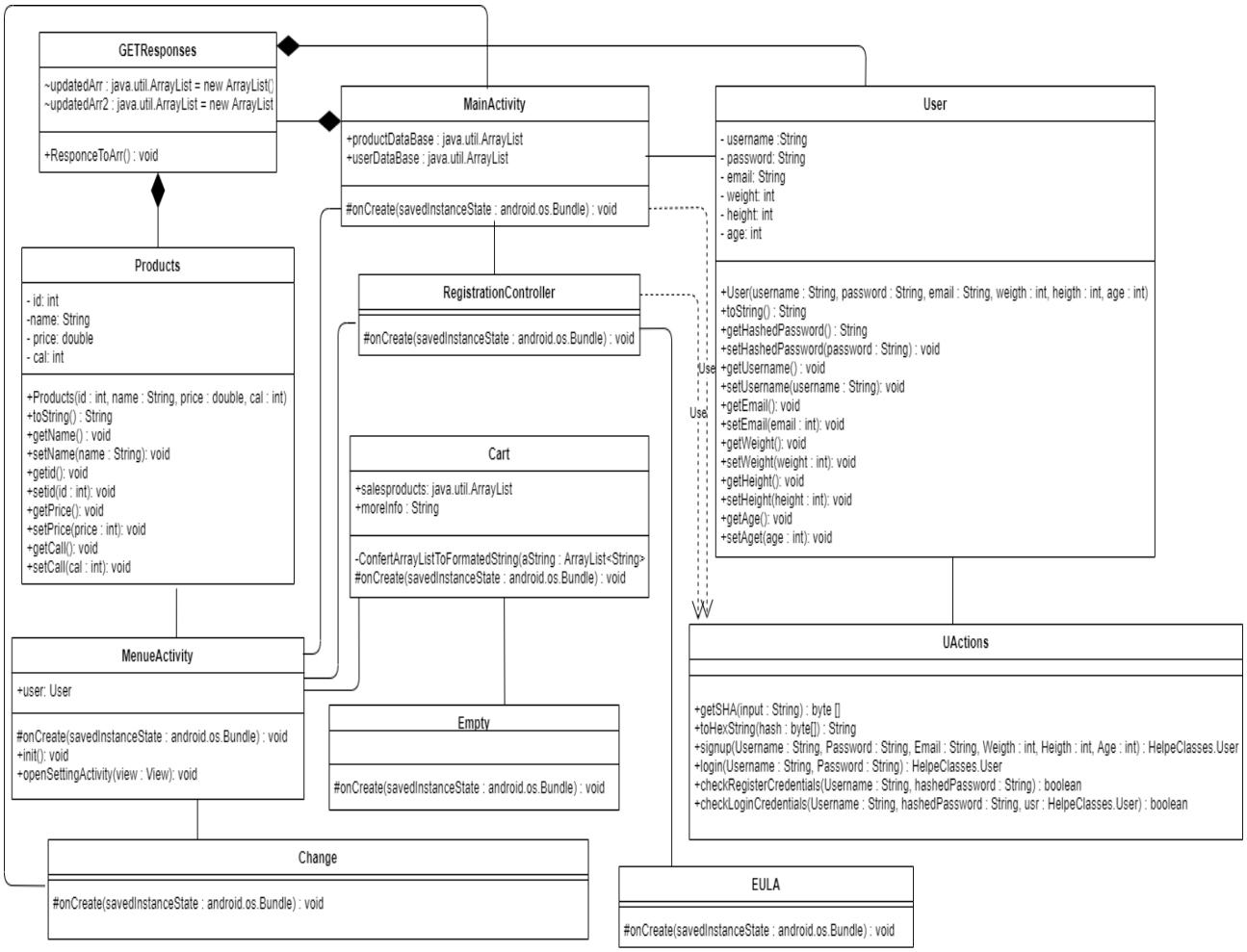

Figure : UML Class diagram diagram

This is our UML Class diagram. Class diagram is made out of three parts. The first part has the class name, the middle part stores variables and the end is for methods. Simple lines without arrows or dashes indicate association-type relationships. Association it denotes the semantic relationship that can occur between two instances. Dotted lines with The Usage above them denotes the Usage-type relationship between classes, which is a Dependency-type case. Dependency-type relationships indicate that another class is required for one class to function elements for its specification or implementation. Another type is composition. composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class. User and Products classes define objects, that we will store from DataBase in the ArrayLists(which are stored in the MainActivity class) by using GetResponse class. Users and Products will not exist without the GetResponse class. The UAction class is one of the main classes that is used to register and identify users. The methods in this class are used by MainActivity class and RegistrationController class. We also associate classes that are responsible for passing parameters between classes. The MainActivity class and RegistrationController are user interface type classes were the user can register(RegistrationController) and login(MainActivity). MenueActivity stores the static variable which will define the logged in user.

The Change class is responsible for updating users parameters(weight, height and age). The Cart class displays the products list added by the user. The Emty class is just responsible to inform the user about successful purchase. Lastly, EULA class displays only End User License Agreement ("EULA").

6.2 Sequence Diagram

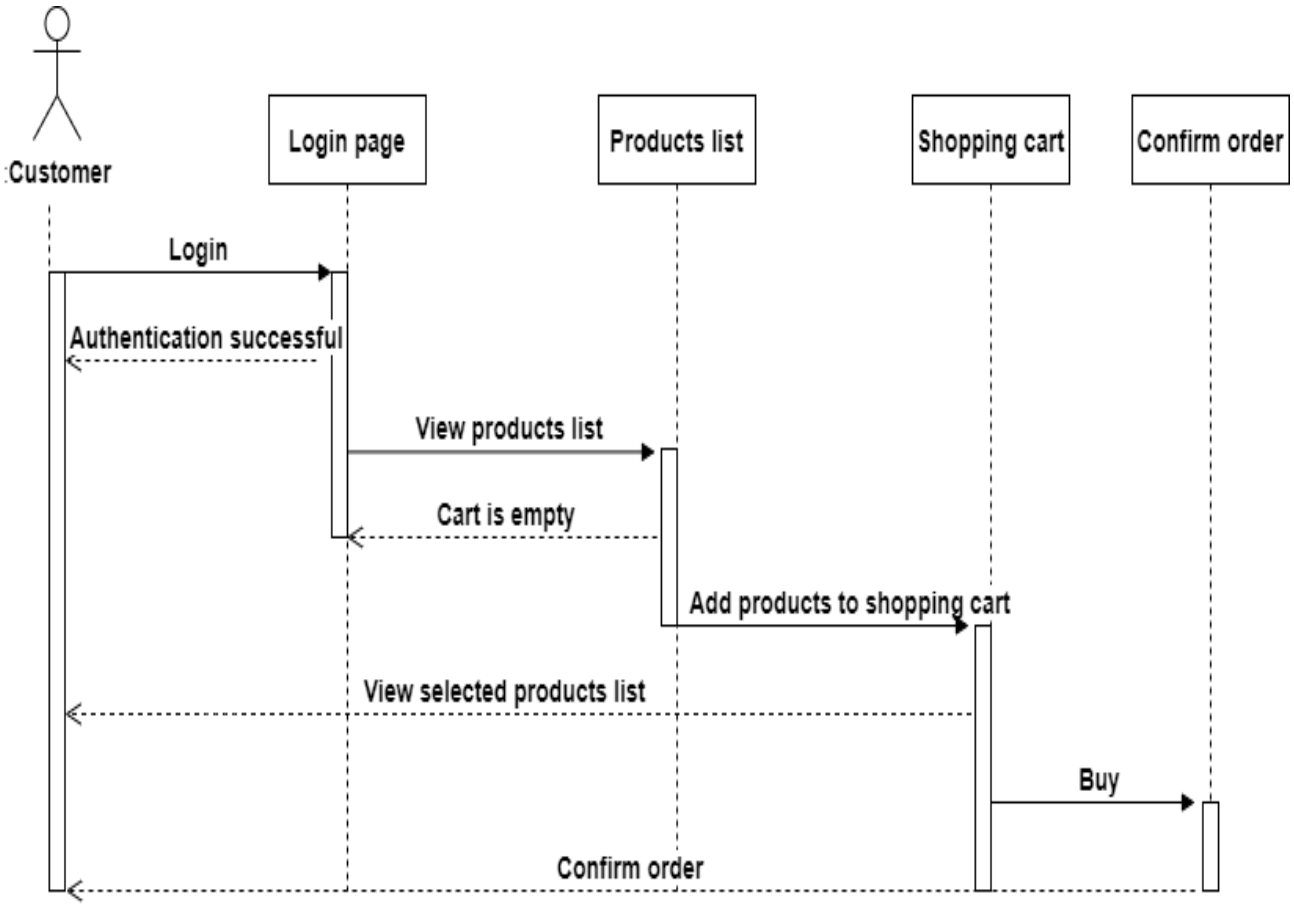

Figure : Sequence diagram

This is the Sequence diagram of our application. It has five lifelines: Customer, Login page, Products list, Shopping cart and Confirm order. First, the customer decides what should happen. He sends login message to Login page entity if everything works correctly the successful authentication is returned. After logging in the customer can view products list in the Products page. He can choose his desired items and add them to the Shopping cart. Also, he can view his Shopping cart. From the shopping cart the customer can select buy function Later, the order confirmation message is sent back to customer.

6.3 Activity Diagram

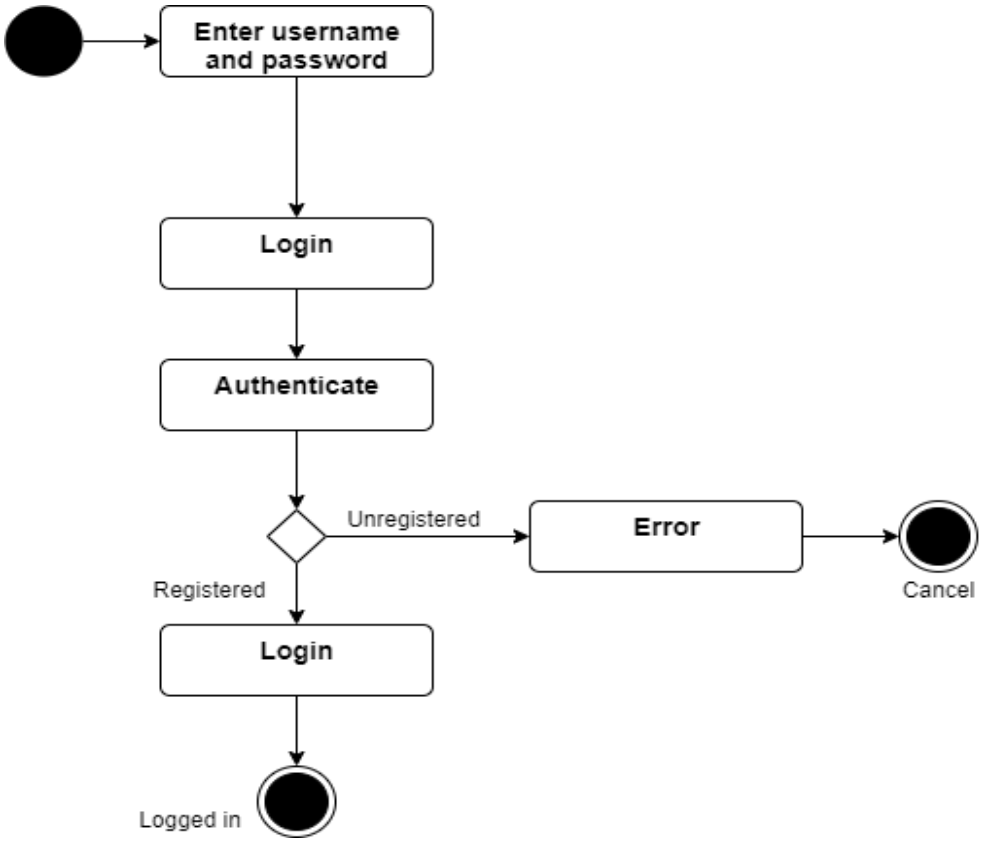

Figure : Activity diagram

This is our Login activity diagram. The workflow starts from when the user enters his username, password and clicks button Login. After this, his personal information is being authenticated. If his information is found in the database, he will be recognized as a registered user and will be logged in. Otherwise, he will be recognized as an unregistered user and will see an error message, his Login process will be finished with cancellation.

Conclusions and Recommendations

References

Appendices