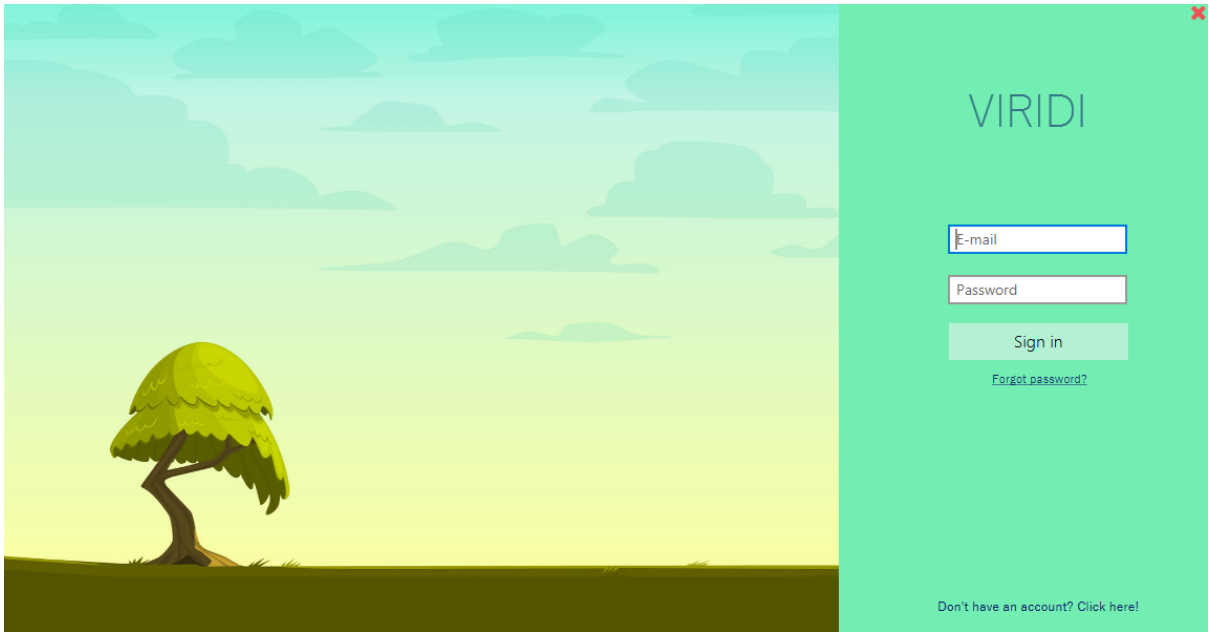
Object-Oriented Programming Project

VIRIDI

April 14, 2019



Ka Fui Yang - 4542088
Alexander Serraris - 4912330
Balys Morkūnas - 4909836
Nikola Ntasi - 4853237
Wessel Oosterbroek - 4961544
Luuk Haarman - 4931173

Delft University of Technology
Group 67

Contents

1. Introduction

People nowadays are starting to do everything they can do to revert the damage they have done to the earth. Electric cars, solar energy, wind energy, riding bikes etc. The amount of ecologically-conscious actions is huge nowadays. The users can also contribute to this cause of helping the earth. Want to do something positive for the earth and feel that you have left a mark after? This application encourages users to save some CO_2 in a fun and memorable way. Just a little CO_2 saving can go a long way. The user does not have to do something big like being the CEO of an electric car company worth billions to effectively contribute to the reduction of the emission of carbon dioxide. Even the smallest thing like eating vegan food can contribute to the cause. Not only are users motivated to save CO_2 but also their friends with the news feed, leaderboard, and the follower's list.

Chapter 2 will discuss the various design choices the group has made and what impact these had on the application. Chapter 3 will be about how the project went from start to finish. The little and big improvements that we as a group could have done will be discussed in chapter 4. Individual contribution and growth will be discussed in chapter 5. We end the report with a discussion about ethics and how our application violates or doesn't violate them.

2. Product

From the beginning, our main focus was to make tracking your CO_2 savings easy and enjoyable. Our primary goal was to make the user feel important when he saves CO_2 and to motivate other users to also save CO_2. The user should feel proud when he does something positive for the environment. The user just needs to enter his/her actions in the application and these actions are sent to their followers. These actions might also encourage their followers to also save CO_2. The leaderboard is for those competitive users who want to compare their CO_2 saving with other followers.

Technical Choices

One of the smallest choices but also a big part of our program was the storage system. Should we use a Database or a normal file to store all the necessary data? After doing some research on the web, we quickly came to the conclusion that using a Database would be the optimal solution for our application. Mainly because Databases contain a lot of useful features like recovering from a crash, the ability to use SQL and flexibility. It is also easier to scale up using Databases.

The very next step was to decide the Database Management System we were going to use for our application. We all agreed that a relational Database would fit our needs without making things too complicated. We decided to go with MySQL because it was easy to set up and some of our group members were already familiar with MySQL.

We used multiple libraries for our application. One of the most useful ones and the backbone of our application is Spring. Spring was used to connect to the Database. Spring JPA, a part of Spring, made sure that we didn't have to focus all our energy on the data access layer of our application [1]. This made it very easy for us to start working on our application and creating java classes without thinking too much about the database.

Images also play a big role in our application because an image can be uploaded for every action (eating a meal, going somewhere with the bike) and every user can also have a profile picture. So the storage system for storing an image was also a big concern. We thought about storing the pictures in the database but doing that significantly slows down your database which would have influenced our application negatively. It also wouldn't scale pretty well because it puts a heavy load on the database. Considering those two things and the fact that there are no real advantages of storing images in a database we decided to store them on disk and only store the path of the image in the database.

Our server uses Spring MVC for connections between the server and client[2, 3]. The load on the server might be greater when using sockets instead of Spring MVC. This is apparent when multiple clients try to make a socket connection to the server at the same time. Our application connects with the server via HTTPS request and reacts to the HTTPS response from the server. Spring MVC facilitates the connection between server and client via URL. The server sends java objects to the client in the JSON format when the client requests data from the server. The client then converts the JSON back to java object that it can use.

For our graphical user interface we used JavaFX, JavaFX was recommended to us by our TA and after working with it for some time we decided to use it for the project. It took us some time to fully understand how JavaFX worked but we never felt like we had to search for an alternative as there was a lot of information available online on how to use JavaFX. JavaFX uses .fxml files to create a scene, .fxml files are similar to html file where its a skeleton of what the user sees, and this can be enhanced with stylesheets and Controller java classes. We used a scene builder to make the designing process of the GUI much simpler, as normally we would have to edit .fxml ourselves in a text editor. With the help of scene builders we could very easily visualize the GUI and see what it would look like.

Architectural Choices

We were fixated on three key values when designing the GUI for our application. The first key value was to make the application more accessible and easy to use for everyone. That is the reason why we implemented many small Quality of Life features into our GUI such as using icons instead of text, password reset option and the ability to view your password when entering it. The second key value was design. We made our GUI more appealing for the eyes of the user. The green background colour of the main screen was to encourage the users to feel like they are in the forest and that their action matters for the environment. The third key value was equality. It doesn't matter which computer you have the GUI should be the same for everybody. The application is re-sizable so the application doesn't care if the computer screen of another person is bigger or smaller.

The application consists of three modules. The server module, the client module and a module called client_server. The server has the server application to run the server, repositories for easy communication between the server and database and a class for resetting passwords. The client module has the client application which has all the methods for connecting to the server, the GUI controllers, the GUI models, the login screen application and the main screen application. The client_server module has all the classes that both the application and server use. Because both the server and client sometimes use the same classes it was more efficient to create a new module and put all these classes in it.

CO_2 Savings

To calculate the amount of CO_2 savings we used an API that would calculate the amount of CO_2 saved in kilograms [4]. We also did some research to calculate the amount of CO_2 saved for some of the options that were not possible using the API such as the options for the meals. Since we reward the user in points for reducing their emissions, we made sure that the amount of points per kilogram saved was consistent for every single action, to make sure every action is fair compared to the others. For using transportation we looked at how much CO_2 you would save by taking a specific kind of transportation compared to using a car. We chose to use the car for comparison as it is the most used form of transport and most of the options in our applications come as an alternative to using the car. Since some of the research only gave us information about annual savings, we divided the savings from the research by the amount of days in a year to get to an estimated amount of daily savings.

Responsible Security Features

For those clumsy users who forget their password, the application has a password reset option. Resetting your password requires you to enter your email address in the application. A link to a web page where you can enter a new password will be sent to you. The link contains a secret token and stays valid for only 30 minutes. You can not change your password if the link is already used or if you try to use the link after 30 minutes.

The way we stored the password was a security concern so we did some research on best practice for storing passwords. The server hashes all the passwords with Bcrypt (using a salt) before storing them into the Database. This means it is impossible for anyone to retrieve your original password, even when having access to the Database without spending a lot of time brute-forcing. Because Bcrypt has an adjustable work factor we can also easily increase the time it takes to hash a password and thus increase the time it would take to brute-force a password [5].

When a user starts the application he/she is first asked to sign up or enter his/her login details. When the login details are entered an HTTPS request to the server is sent containing the password and email of

the user. If the password/email combination is valid the server will send an Authentication token to the client [6]. The client can authenticate itself for the next hour with the authentication token. After one hour the authentication token becomes invalid and the user has to login again. We use a JWT library to create and verify authentication tokens. JWT tokens are created by hashing a server secret. This means the server can validate a token without having to store anything in the database [7].

When taking privacy into consideration we decided to only store the essential information of the user, mainly the e-mail address which is used for logging in and for password recovery, name for the ability to recognize and be recognized by your friends and a password which is hashed and stored in the database upon account creation, meaning we have no way of accessing it or directly seeing what it is. Also, there is a feature that lets you hide your account from others, this means you can only be followed by people that have your e-mail. This ensures that no one you do not know or do not want to be followed by can track your activities.

3. Process

At the very beginning, we faced the challenge of not knowing where to start. We managed to get our vision together, explain what each of us wanted but we were overwhelmed by the number of paths we could choose from. After a lot more discussion and some tips from our TA, we decided to use Spring library for the server because it also includes tools to connect your project with a database, securely store passwords by providing hashing functions, RESTful services for setting up our server as required by the project. It also includes a substantial amount of documentation which always came in handy. For the Client/GUI part, we chose JavaFX as it was recommended and seemed suitable for beginners. Looking in retrospective, these choices seem to be valid and definitely worked well as we managed to really grasp their concepts and gain valuable experience.

For project management and structure we chose Apache Maven which is based on the concept of the project object model (POM). At first, we had a lot of issues with it. Random errors, not knowing how to keep the structure of the project correctly, an intense amount of plugins to use and to manage, compilation issues and so on. But in small steps at a time, everything fell into their corresponding places. We learned that we must abide by the rules of Maven and not our own. After we set up our projects structure everything sort of started fixing itself. We understood the idea of plugins and how to use, delete, update them. We managed to generate reports that we needed and to test our project, see the coverage percentage. Even though it was a real headache some times Maven really helped us to maintain our project, catch some inconsistencies and in the end was also a viable solution.

During the project, we used a technique called Scrum, which is basically a program you follow that helps you, and your team members stay on track and on par with each other. Every week, after a so-called Sprint, we would discuss the progress we achieved and make up a new plan for the upcoming days. During these reviews we all had the ability to share, question, discuss and brainstorm ideas, development plans and take further steps in the creation of our project. We would keep track of features that needed to be implemented in Scrum boards, point each other in the right direction or help with code structure when reviewing someones else work. In general, using version control and Scrum really shaped us into a better team while also boosting our efficiency and the quality of our work.

One major issue we ran into during the process was communication. In the beginning, we had divided the work across the team nicely and for some time focused only on our own things that needed to be done. So, for example, the person who did a part of the GUI, at the time, did not pay much attention to the server implementation or vice-versa. This led to some miss-communication because our visions differed across certain aspects. After realizing this we all agreed to pay more attention to each others work, frequently discuss how our project is being implemented and make decisions together, as a team. This small, but crucial incident most likely was a lesson for all of us, we saw how important it is to show interest and be active in those discussions because the repercussions of rewriting a chunk of your code are very time and energy consuming.

In the end, all of the problems we ran into were very useful and only positive results came from them. Each of us faced many difficulties and most definitely learned or took valuable insights and experience from them. We improved our problem-solving skills and developed new while working as a team. The process has been filled with excitement and frustration but all of us now know how it feels and what it takes to be a team.

4. Reflection

Product

Every time a user does something in our application information has to be sent and/or retrieved from the server. As a result, our application will not function properly when the user is not connected to the internet. An improvement could be to make the application work partly offline and partly online. If the user doesn't have internet then every action that the user performs could be saved in a local file and when an internet connection has established the data on the file can be sent to the server.

The application does not connect via the internet because the server is local. The reason is that during development of the application it was easier to test the process this way. The developers don't have to rely on the connection to a remote server. An improvement could be to switch to a remote server when the application is finalized and having a local server for testing purposes.

Not every user has a good intention when using the program. Users with malicious intent could try to get a hold on data that is not meant for any user to see. These users could try to SQL inject the database system. An improvement could be to check on the server side if the request of a client is malicious. Prepared query statements could be used to prevent an SQLi attack.

The point system of the application depends on the user input. A user could input a big number just for the sole reason of getting a huge amount of points under their name or by accident. A limit on the amount of distance could be implemented for preventing these uncommon situations.

The user has to log in every time the application is closed. This might be annoying for some users and these users might refrain from using the application again. An improvement could be to save a specific file with email and hashed password in the disk which gets deleted after a period of time. The application could use the data on the file when starting the program. If this feature is implemented, logging in again would not be necessary.

Process

Adding a new attribute in the classes of the Client_Server module is complicated. The reason is that these small changes might cause other parts of the program to fail. The amount of pipeline failures is a good indicator. Having separate classes for the client and the database might also be problematic. There will be a lot of copying code between server side and client side which is not a good programming practice. An improvement might be to just break Java object from the client into multiple Java objects in the server and saving these individual objects in separate relation.

Every method for client and server connection is in two classes. One resides on the client side and the other one on the server side. There are a lot of methods in these classes. To get a specific code a developer would have to scroll through all the code that is not relevant for this developer. An improvement could be to separate all these methods by function and putting them into their own classes.

Some of the code is not readable. Some code lines are worth multiple code lines. These codes might not be readable by GUI developers. An improvement could be to split hard to understand codes in multiple lines.

The majority of the methods in the client side are HTTPS requests to a specific URL. An improve-

ment is to have String URL with the green text below(Figure 4.1) and then use that string for all the methods. When the server URL is changed, the only thing to adjust is the String URL.

```
ResponseEntity<Boolean> response = restTemplate.postForEntity( url: "https://localhost:8080/postActionUpvotes", map, Boolean.class);
```

Figure 4.1: The code for the HTTPS request

Course

The OOPP course is one of the important courses in the computer science curriculum because it encourages students to work together in a project. The course has a little amount of lectures which is good because the students should be focusing on solving the problem in a team. The only gripe we have is that some stuff to start the project were not explained at all. The course lecture talked about a git but not maven which was kinda disappointing. The project started very slowly because nobody knew nothing about maven.

The TA of the project was very helpful. She didn't tell us how to solve a specific problem. The only thing she told us was to go look it up online because the problem you are having might have been already solved by someone else. If the problems ware more serious - she would kindly help us.

Our group had meetings twice a week. There were still some miss communications. The reason is that everybody was creating different methods or implementing something else in the GUI and when we actually merged everything together we would sometimes spot some inconsistencies. The developers were still figuring out how the application would look like and they had no experience. This might not be an issue in the future because these developers might have gained some experience in their field. An improvement could be to sit down and discuss the structure of the relevant class that both the GUI and the server uses in more detail. During the project, we recognized some miss-communication and we tackled it by giving every issue a more descriptive name, a thoughtful explanation and we tried to keep the SCRUM board updated during every sprint. Besides that, every week we would thoroughly discuss what had to be done and how will we be doing that. During the project each of us most definitely developed better descriptive skills and the gained the ability to express ourselves better.

6. Design For Values

In this section, we discuss the ethical aspects of the project.

Safety

A not obvious value could be safety, if more people take public transport this means that there will be fewer cars on the road. This means traffic will decrease and people will be able to get to their destination sooner during peak hours. Because of the decrease in the number of cars, the number of accidents will decrease too. Coincidentally, the number of accidents will be reduced even more because in general public transport is much safer than cars.[1]. A conclusion can be derived that there exists more benefits for taking public transport besides reducing your carbon emissions.

We would have to change our product in a few ways, since taking public transport adds additional unintended benefits we should make sure that the user understands that taking public transport does not only lower his/her CO_2 emissions but also adds to their own and other civilians safety. Because of the additional benefits, more points could be awarded for taking transportation besides the standard amount of points awarded for reducing CO_2 emissions. If the application takes off in popularity and a lot of users will start using public transport to score more points, public transport companies might have to deal with crowded buses during peak hours, which might cause them to need more busses and employees. Since more people will take public transport now instead of their busses, gas stations will see fewer customers and therefore their revenue will drop.

A possible way to research the effect of a large drop in car usage and a large increase in the use of public transport would be to pay a large group of people that use any kind of transport to be tracked everyday and note how many accidents occur within a period of time for that specific kind of transport. We could contact public transport businesses and gas stations to see if they're willing to share information regarding the increase or decrease in customers if they become aware of the benefits of taking public transport.

Physical well-being

Another not obvious although very important value is the physical well-being of the general population. Over the past decades, obesity rates in modern countries have skyrocketed. Over more than half of the Dutch adults are overweight[2], therefore it is important to keep the physical health of people in mind when designing an application. If more people take the bike for their commute or eat more healthy, their chances of becoming overweight decrease[3]. If we can make users of our application more health-conscious, we might lower the number of overweight adults, consequently lowering the amount of overweight-related diseases and potential deaths.

Changing the product in a number of ways could possibly make the user more aware of the health benefits of certain actions. We should make sure that the user understands that commuting by bike instead of car does not only lower his/her CO_2 emissions but also that it is advantageous to their health. Because of the additional benefits, more points could be awarded for taking the bike over the bus or even the car. Besides awarding extra points, in the information window we can provide extra information to the user about why it is healthier to ride your bike. Possible consequences of implementing that could be that bike lanes will be more crowded than before, thus congesting traffic. The local government would have to consider building extra bike lanes, which in return promotes bike usage again.

[1] http://www.nctr.usf.edu/wp-content/uploads/2014/12/JPT17.4_Litman.pdf
[2] https://www.volksgezondheidenzorg.info/onderwerp/overgewicht/cijfers-context/huidige-situatie
[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2872299/

A possible way to research the consequences of the implementation of the physical well-being value is to take a large sample group of people and split that in two, and for one year check every day their body composition, track how they commute and what they eat. One group (test group) uses our app, the other group (control group) doesn't. After one year you check the relative difference(between the start and end of the year) in the body fat percentage of the test group and control group. In that way, you could tell whether your app is designed for the value of physical health.

Bibliography

[1] Spring data jpa - reference documentation. https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

[2] Spring Boot documentation 2.0.5. https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/.

[3] Spring web services reference documentation. https://docs.spring.io/spring-ws/docs/3.0.7.RELEASE/reference/.

[4] Impact models[brighter planet cm1]. http://impact.brighterplanet.com/models/.

[5] Spring Security documentation. https://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity-single.html#new-3.1.

[6] Apache package documentation. https://hc.apache.org/httpcomponents-client-dev/httpclient/apidocs/.

[7] Package io.jsonwebtoken javadoc. https://jar-download.com/artifacts/io.jsonwebtoken/jjwt/0.6.0/documentation.