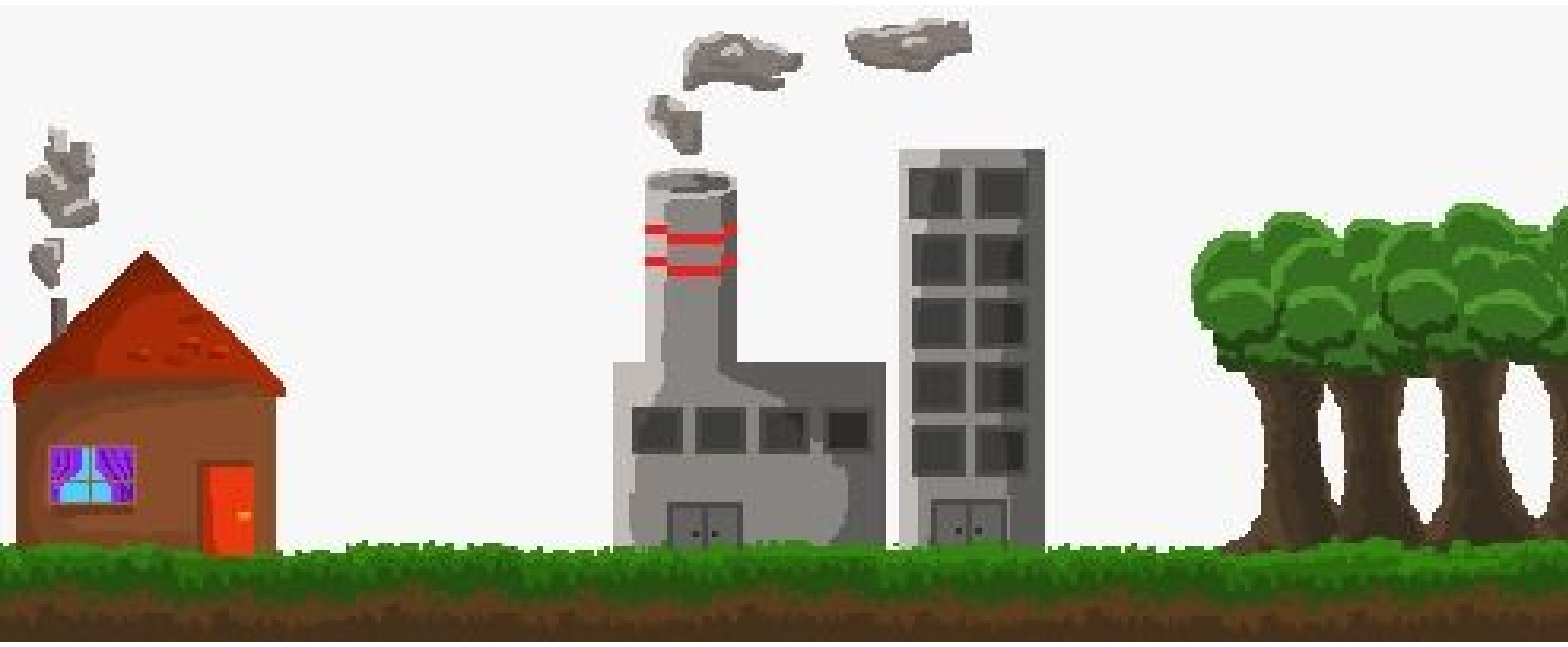
Go-Greeen

Application

OOP Group 4



Report for OOP Project

This is the report of Group 4 for the course CSE 1105 consisting of:

- Erwin van Dam (4928113)
- Elias Baha (4830547)
- Eyüp Vroegop (4929578)
- Gyum Cho (4840054)
- Marit Radder (4957334)
- Marko Matusovic (4908406)
- Shruti Arora (4770501)

Our teaching assistent is David Alderliesten.

General 3

Design decision for application 3

- Client 3
- Server 5
- GUI 5

Points of improvement 6

- Client 6
- Server 6
- GUI 6

Individual feedback 6

- Erwin 6
- Eyüp 7
- Gyum 7
- Marit 7
- Marko 8
- Shruti 8

Value Sensitive Design 8

Introduction

This report is about a project with as goal to create an application which should motivate people to go green. During this project we have created a client side application, in which the user can click on buttons and add small descriptions to describe the behaviour of the user. The client is connected to a server, which uses a database to validate and to keep track of every move of the user. During this project we had to work together as a team of 7.

This report will describe the general going of events during the development of the application, after which design decisions are explained, following are some points of improvement, then individual feedback to finally conclude with the value sensitive design.

General

We planned our tasks in sprint plans per week. In the beginning we didn't take problems into account, so our sprint plans contained way to much work to be done during the week. Over the course we got to know what we are capable of doing in one week, but still the sprint plan wasn't always exactly what we managed to do in one week. Next to that there were weeks where not everyone knew what to do exactly, because of a error in the planning.

Overall the team collaboration went well. Gladly, we all like each other, and have a common sense of humour. We communicated through WhatsApp. We tried to have a second meeting on a weekly basis, during which we would work together really efficient. We divided the tasks between the group members. This really worked well for the team efficiency. The effect was that we knew which person we had to go to if we needed a certain method or attribute for our own part.

We were dependent on each others work, which caused several problems. Often we needed someones code so badly that we approved the merge request nonetheless it had problems. A lot of those problems were solved soon after, but sometimes such a problem wasn't really solved at all, which would result in broken code in everyone's branch.

We used GitLab as version. In the end we managed to work with it, but during the project we have made several bad mistakes. In week 1 we had direct pushes into the master, in which we encountered problems like losing someones commit because of someone else's commit. After that we used branches. Our general idea was to create a development branch per week, everyone could branch off and merge into this branch during that week. A problem we encountered was that our branches were up too long, such a branch didn't have all the updates from other branches for like a week, after which a merge request had way to much conflicts. Next to that if the merge of a branch didn't come for like a week, others would have to wait 7 days before they could use this branch, or they ended up with broken code if they pulled from the branch earlier. Our solution was to create smaller branches, which were up for something like 1 day and 7 commits, such that we didn't have to wait for someone else's code for too long. In the weekend we would merge everything back to the development branch, after which we would add every mandatory file, and then merge it into the master. The biggest problem we encountered there was that we had to wait for someone's branch to be merged until saturday in the evening, before we could add all the needed screenshots.

Our commits improved a lot during the process, still it happened a lot that someone put up a merge request with non working tests, or code with checkstyle errors. Next to that we did review each others commits, but we did that in person, so we didn't really add a lot of comments to a merge request on

GitLab. As we reviewed the code, but didn't really test it to the bone per merge request, some small issues were not seen up till someone tried to do something silly in the application, after which it all of a sudden broke. Once CI was implemented the merges could no longer contain failing tests or broken code, which helped a lot!

As scrum board we used the issue system of GitLab. This worked really well, but we didn't always use it correctly. Some issues were made during the commit, as there was no issue for that commit before. Other issues were not moved to closed or testing once finished. Next to that the labels and milestones could be expanded.

Overall we learned a lot. One of the important things is that we learned how to work efficient with each other and how to use git. Gladly we were able to finish everything in time. Next to cooperation we all gained experience in java, especially in using external sources like Spring and Javafx in java.

Design decision for application

General

During the project we used the [1]Camel case convention (upper camel case for classes, lower camel case for methods and data names), we didn't use numbers or special characters.

Client

Our main plan was to create an application in which the user had to do as little as possible to fill in an action (being e.g. eating vegetarian meal). We didn't want to ask the user for a lot of information, we wanted to stick to username, password and email, the first two for identification purposes in our application, the last one for situations like password reset.

During the first stages of the projects we tried to make a good structural plan on how to structure the application. Our first design decision on the client side had to deal with this. We decided to create one application package, which included the categories, the GUI and the API, for those things do have nothing to do with the server or the database. As the GUI became bigger and bigger, we decided to move the GUI to another package as well. The categories all got a typical layout of having one method per action, in which each action did its own call to a method which is connected to the server with its own parameters.

After that we came upon user restrictions. The username has been restricted to a-z A-Z characters for simplicity, we decided that this should be more than enough for our current application, and it would be the easiest way to filter out special characters. Due to responsible software we implemented a check for offensive names. For this we use a google file with all banned names for Youtube etc. The password has to be at least like 8 characters long (for testing reasons we used 1 for a long time during development). Another restriction is about the actions: we restricted the user to a certain amount of kilometres for traveling actions, to a certain amount of food actions per day etc. This was because otherwise users could fill in stuff like trips of 10000000 kilometres.

The username is encoded using [2]base64 for privacy reasons. The password was first of all encrypted strongly, but as multiple times the same password will result in different encryptions, we chose

[1] https://en.wikipedia.org/wiki/Camel_case
[2] https://en.wikipedia.org/wiki/Base64, https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html

hashing for the password. This is also non reversible and very strong, but here we could just hash the password and sent it to the database to be checked, which wasn't possible for encryption.

We didn't want to send the hashed password with every request we make, this is because we think it's best that even a hashed password shouldn't be in the air for to long. So we came up with the system that the user logs in, after which the server sends back a token if the login is correct. The client side will send this token with every future request. We generate the token on server, it is a time (in milliseconds) hashed with [3]SHA256. If the user user opts in for "stay logged in" option, then the token is stored in local file and hence removes the requirement to log-in every time the program starts.

For general classes we decided to make every method static, to make the class final, and to include a private constructor. The reason for this is that those classes have a lot of methods in them the application uses, but the application doesn't need several instances of them, and when different instances of for instance the server connection class are used, one might miss the token for the connection, which could cause problems.

Testing started as simple Junit tests, but it quickly evolved into [4]PowerMockito as our main testing framework. This was because of our interconnected class structure full of static methods, and as much private methods and fields as possible. The downside of this was that not everyone was very skilled with PowerMockito, but as we didn't want to throw our class design decisions, we had to choose PowerMockito as our testing framework.

As an API we started with [5]CoolClimate, but the problem that occurred was that CoolClimate requires a lot of general information about the user. At first we tried to fill in as much as possible for the user using standard models, but once we discovered [6]Brighter Planet we managed to get our results using only information like distance for a car action.

Using the API we wanted to calculate the total amount of Carbon produced, using only necessary information like the distance. Next to that we have implemented a carbon reduction using standard models. Once someone takes a bike, we compare his action to someone taking a car with the same distance.

As we wanted the car actions and energy actions to be influenced by the user having an electric car or not, we decided to implement an onload method. This method fetches information from the database like if the user has clicked on electric car before. In the carbon production calculations and the carbon reduction calculations we consider those facts.

Server

For writing the server part we needed a lot of research as there were a lot of unanswered questions like how to make HTTP, POST or GET requests. When we were able to find out how to make the requests, the question was how to send objects in JSON format between client and server. We have used httpEntity and [7]RestTemplate for making the requests and transfers of objects between client

[3] https://en.wikipedia.org/wiki/SHA-2, https://google.github.io/guava/releases/23.0/api/docs/com/google/common/hash/Hashing.html
[4] https://www.baeldung.com/intro-to-powermock, https://mvnrepository.com/artifact/org.powermock/powermock-module-junit4/1.6.5
[5] https://coolclimate.berkeley.edu/api
[6] http://impact.brighterplanet.com/?r=
[7] https://www.baeldung.com/rest-template

and server. We're using [8]Spring framework as it was most comfortable to work with and had a lot of documentation available.

Next to that we had another search going on for Database, where we decided to use Postgres SQL and query it through the server. We have our Database deployed on [9]Heroku so it's super easy to insert and get data from it. We then had to decide on the design of database. We could've gone for one big table with everything in it which would have left us in a mess of tuples, but then we used our learnings from last quarter and created organised tables. The database tables have everything linked to the user through their encoded username. To access the details the user has to provide password and username or a token which are compared to the hashed password and the encoded username on the database.

To avoid the mess of thousand lines of code we organized different packages or folders for each side of communication that is between client and server and server and database. There are different classes on client and server to transfer the data as one object. There's a client side, server side and database side of the communication. Client side makes the requests to the server and then waits for the response from the server, server side of communication accepts the requests and then calls the database methods for values, database side has all the methods that can create a query to handle the data in the database. Then the data from database side goes to server side, from there it goes to client side. So basically server is in the middle and most of the calculations take place on client to reduce the load on server.

GUI

For the GUI we needed to make several design decisions, but it all began with choosing which libraries to use. We choose for JavaFX[10], the main reason for this decision was that it was recommended during the lectures and that we could find enough examples and tutorials for it online. As this library had everything in it we needed, we didn't look for another library. Above that we also used the JFoenix[11] library a lot. We used it to create the buttons and other components, and to make them more beautiful. Next to that it was fairly easy to use.

In the design decisions, the main decision was choosing our navigation system. After some trial and error with other components, for example the hamburger buttons in combination with drawers, we found the tabpane, which worked perfect for us. It was easy to use and even more easy to expand when we needed new tabs. The downside was that it made our code very messy, but we decided to keep it this way, because it was too late to turn back.

The second design decision we made was to keep the interface simplistic. The thought behind this was that it would be simple to use, without taking too much time of the user. Our thought was that a user doesn't want to fill out a whole form after the user ate a sandwich, pressing three buttons should be enough.

Then we had to choose between pixel art or drawing for the planet. After Marit made a quick sketch for both we found out that the pixel art had better results, so we choose for pixel art.

[8] https://spring.io/
[9] https://www.heroku.com/
[10] https://openjfx.io/#
[11] http://www.jfoenix.com/documentation.html

Then, lastly, we also choose to implement a dark theme. This has multiple reasons, mainly because dark themes are loved by developers but also by users. Secondly dark themes also save battery lives in some cases. This is of course better for the climate.

Points of improvement

Client

Our client programming was enough good. However, I think we improve the program quality by implement the better test file. since the testing is one of the main part of the making good program. we can make out program better with more test.

Server

The side of server which could be improved is better validation of communication with the app. This includes checking for correct request format, order and content. This could be achieved by adding sanitization checkpoints on server side, this would however decrease the performance of the server. Also decreasing the time between generating authentication token for user could increase the security. Furthermore the communication between the client and the server could be encrypted with asymmetric encryption based on private and public keys, however such level of security is not required for this application.

GUI

The GUI surely can be improved. The main class is too big and a little bit of a mess, so we could make separate classes for each separate tab. Otherwise we could also improved it so we can make use of the whole screen. This also means we can make the program resizable.

There could also be more improvement on the team work regarding the GUI. The communication wasn't always the best between the server and database. I think a lot of time was lost there. For example waiting for the other party to explain what methods to use. But when we sat together things went better, and we could do the same amount of work in less time.

Group work

The overall development of the application went great. We had the weekly meetings on Monday and always stayed there some time before and after the official meeting with the TA. The motivation was enhanced by the food rule, someone was always obliged to bring some snacks.

At Mondays' meetings we would divide the work amongst all of us for the rest of the week. We had 3 working groups which worked on server, gui and API + application mechanics. As a group we had an early low at week three, then we didn't do that much. This was due to lack of organized assessment of tasks for the week. After week three this problem didn't occur.

Overall everyone put in a great effort and concerning group work there weren't many things needing improvements.

Individual feedback

Erwin

My strong point was that I was dedicated to finishing this project successfully. I really wanted the screenshots and files to be uploaded weekly, or to get full points for the demos. Sometimes this became my weak point, when I was so focused on the demo, that it became more important to me than to deliver good code, but team members saved me from making that mistake.

Another strong point was that I really tried to help creating a structural set of conventions to cooperate as efficient as possible as a team. My share evolved from commentless, broken, non-tested pushes into commented, clear and tested merge requests, an improvement I hope to continue.

I have achieved my goal to give work away after working on it for some time, such that someone else could look at it from another perspective. This worked really well with Marko (Testing with PowerMockito) and Marit (GUI).

My weak point was in situations that the planning wasn't so good, when communication was unclear, or when someone had difficulty to stay tuned with the rest of the group. In those situations I most of the time didn't try to solve the problems, but instead just did my part of the job and nothing else. Later on in the project we started to involve everyone as much as we could to solve those problems. Thanks to David's explanations I got to understand why this is a very bad way of handling difficulties as a team, and even learned something about how to solve those problems as a team. All those problems were solved by talking to each other, explaining why we acted in a particular way, and by finishing with conventions on how to prevent such problems to occur again.

Eyüp

I feel that my skill in Java was improved, although not in the way I was hoping. The contributions I made to the project were mostly in relation to the server, so gained a lot of experience in coding the server side, more specifically the database. I made a database and made methods based on what the server gave me and what was needed, as a result I now have a pretty good grasp on that works now. All in all I can say that I have gotten better at Java and have improved enough that I can get a better idea on how to code some specific things but not really something that will help my original goal in the personal development plan.

In general the project itself went fine, we had some hiccups here and there but nothing that couldn't be solved without some communication and help from our TA. The process of working as a team was a bit rough in the beginning, nevertheless we found our footing after a couple of weeks. After those couple of weeks our productivity went up and it was smooth sailing from there. In comparison to other groups I am content with how we worked together as a team, and how we solved problems that came up.

Gyum

The project was doable with the help of a teammate. My programming skill was not enough good to finish every part with me. However, with the help of teammate, I could do my part of the job and it was a precious experience for me. The overall project was well through, but there were some problems. One of the main problems was the miscommunication between teammate. Since it is teamwork, we should have to work as a team. However, sometimes we misunderstand others and make the same work twice or produce a useless program. This miscommunication made some time lose in our project and I think we could do it more efficiently. Eventually, it all solved with teamwork!

I think my weak point was arrogance during this project. Once my teammate sent me the stuff to do or explain to me what I need to do when I feel I understand it, I didn't pay attention enough that made the miscommunication between teammate. This causes me to do some useless thing and it causes the time to lose. Of course, after I made the mistake, I fully admit it was my fault and change my attitude.

Marit

Overall I think the project and the teamwork went okay. Of course we had our discussions but in the end it all went well. One of the problems I and the team had that there was some miscommunication. Miscommunication in the distribution of the work. And some miscommunication in joking around or meaning certain things. But I personally think that you always have this till a certain extent if your work on a group project.

I also think I really tried to improve my weaker points. In my development plan I sad one of my weaker points is to plan realistically, and I think I did my best to improve it. I tried to be involved in making the backlogs and sprints. Which helped me a lot learning to plan more realistically.

I also tried to use my strong points in my advantage. One of those points was that I was creative, which I tried to use as much as possible in the GUI. Another point I used in my advantage for the GUI was that I am sometimes a little perfectionist. And I think those two points are one of the reasons why I liked make the GUI so much. And also think that if I hadn’t had those qualities the GUI would turned out different than it did now.

Marko

This course has challenged me in many areas. Starting with limited knowledge of using web-servers in Java and basically no knowledge of how to use git, it was easy for me to fill those knowledge gaps effectively. The learning process was made a lot easier by having a teammate (Shruti) who researched the same thing as me. This allowed for double the heureka moments, which moved the studying forward.

Another aspect of this project which was perhaps not obvious from the start was the coordination of the team required to work together effectively. We achieved a good teamwork by having more meetings, at which we had marathons of brainstorming and programming. I realized that it is very important to make exact schemes before the actual coding. This way the distribution of tasks was much easier.

My personal development plan was mostly focused on developing skills in teamwork and communication as I tend to like doing things my own way and find it hard to let go even in the smallest detail. The project taught me that it was impossible to do all the work myself and in the end I was glad for the ideas, input and work of others. I also found that having a team to work with gives you opportunities to learn new techniques and discover new tools. I learned to appreciate git, discovering its features by myself as well as from my teammates.

Shruti

This was a very fun course for me as it was very practical. I'm usually not good at working in teams but over here it was very different, probably because of the amazing teammates that I had. In the starting, it seemed difficult to coordinate with everyone and some misinterpreted ideas but then during the course of next few weeks, our team was doing great. I'm glad we had Erwin to keep us ducklings in check and I love how patient and believing he is. I loved the streaming hot discussions even for the smallest problems and how we all looked at them differently. One of my weakest points that I noticed in the start was that I come up with ideas that are not organised at all so I tried to work on it and now I try to have a map of almost everything I want to do not just in my head but physically present written or drawn.

I spent most of the time on backend side of this project because it seemed challenging to me, which it was in the start but after all the research which I put in everything seems fine now. I learned a lot about HTTP requests, GET and POST make more sense now. I spent the first week looking at the most comfortable API to work with and Spring was the best one with a lot of documentation available. It took me a lot of time to figure how to request and send objects between client and server. Now it takes almost no time to make new root for a new task to be performed for sending an object from client to server to database and reverse. Our server team made sure that everything was organised and kept properly in classes. One of the most important thing that I learnt was how helpful Javadoc and comments are as it would have been almost impossible to predict what needs to be done. I have always been scared of SQL but here we had to query our database through java. Planning how to implement something was the best part where everything that seemed almost impossible came down to a few lines of code.

Then comes the git part where I made a total mess in the starting and left a branch unmerged and totally apart from other branches, it took ages to resolve conflicts and merge it but then I learned my lesson so I try to keep my branches under check now.

In the beginning, I thought that there wasn't enough motivation present in our team but now I guess we broke the scale.

Value Sensitive Design

As a team we set out to design our application for users that wanted to improve their environmental friendliness. For that goal to succeed, the user must be able to see how much Co2 they emit, and based on that act accordingly. We first give the user the tools to record their daily activities and calculate the Co2 they reduce by partaking in those activities. The more the user is able to reduce their Co2, the more points it will receive. If the user continues to partake in activities that are not eco-friendly, they will not be rewarded with points. The user is also able to have friends have compare points, which further nudges the user in doing green activities.

We strive to design for the values effectiveness, environmental-friendliness and motivation. By nature our application is environmental-friendly, it strives to decrease the Co2 emission, which as a result improves the environment. It is also effective, because user can input their activities and get an accurate result back, and based on that change their behaviour to get the intended result. The value motivation also plays a big part in our application, because the element of gamification is interwoven in the application. Gamification in our application gives the user rewards for completing activities that are eco-friendly, which influences the user into doing those activities more. Our application also has a leaderboard and a friends list, which gives the user the motivation to get more points, because most people inherently want to be the best in something and one-up their peers.

There is a lot of academic literature on the topics of motivation your users and getting them the results that they desire. For example, studies or books that go over the topic of gamification and how to properly implement that will help us gain theoretical insight in our values. Also, there are plenty of books and studies which portray the best ways to decrease your ecological footprint, which can again be used to increase our theoretical insight and improve our application.

Although we rely heavily on motivating the user, it is important that environmental-friendliness stays a key aspect when using the application. We don't want to user to get caught up too much in the gamification aspects, and lose sight of the actual goal, which is being more eco-friendly. So as a result could say that there's some tension between the values environmental-friendliness and motivation. We can loosen this tension up by implementing a design, which brings more attention of the user to the environment aspect of the application. For instance, by giving the user a lot of informations and using recommendations, we can divert some of the attention allocated to the gamification aspect and influence the user to think more about the choices he makes in regards to environmental-friendliness.

In conclusion, when we apply these concepts and realize them, we will surely be able to incorporate environmental-friendliness and motivation, but effectiveness relies too much on the user to make a sure guess. If the user uses the feedback given, then the value will be incorporated in the application. Otherwise if the user doesn't have the displace to apply the feedback, then the value effectiveness is lost.