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

We didn’t always stick to the sprint we made for that week, but in the end we finished the task on time. That’s why we already put some slack in the planning, because finish the whole planning in a week is unrealistic. There are always some issue which appear during the week and we tried to take those problem into account.

Overall the team collaboration went good. We communicated through WhatsApp and that went well. We also tried to plan an extra meeting during most weeks, where we really worked efficient. We divided the tasks between the group and everybody knew what was expected from them. 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 used git as a version control, and this really went well. It took some time to understand how it worked, but in the end it really improved our workflow. We decided that we made a new branch called development every week. And then personally branch of the development branch for the issue you are working on. After you finished the issue the idea was to merge it back into the development branch. This didn’t always go well, but in the end it worked. When the week was finished we David merged the development back into the master, and we could start over for the coming week. This in combination with the CI was really nice to know at which feature the program stopped working. Which had the effect that we could solve the issue fast.

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. But we also learned a lot about making and querying a database with java, set up a server with Spring. And we also learned how to use API’s to get data from websites and how to make a user interface in Java.

Design decision for application

General

During the project we used the CamelCase convention (UpperCamelCase for classes, lowerCamelCase 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 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 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 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 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 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 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 then the question was how to send objects in JSON format between client and server. In the end we have used httpEntity and RestTemplate for making the requests and transfer of object between client and server. We're using Spring API as it was most comfortable to work with and had a lot of documentation available.

On the other side we had another search going on for Database, where we decided to use SQL and query it through the server. We have our Database deployed on 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 a one big table with everything 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 encrypted username. To access the details the user has to provide password and username or a token which are compared to the encrypted password and the encrypted 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 has the requests made to the server and it 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 which are queries made to 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 choices, but it all began with choosing which libraries to use. First we choose for JavaFX, 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. But we also used the library JFoenix a lot. We used it for make the buttons, and other components more beautiful. And the main reason we choose it because it was fairly easy to use.

Then we go to the design decisions, the main decision was choosing our navigation system. After some trial and error with other components I found the tabpane. Which worked perfect for us. It was easy to use and even more easy to add more tabs which became very useful.

The second design decision we made was to keep the interface simplistic. The thought behind this was that is would be simple and quick to use. Because you don't want to fill out a whole form after you ate a sandwich, pressing three buttons should be enough.

Then we had to choose between a pixel art as planet, or a drawing. After I made a quick sketch for both I found out that the pixel art had a much better result. So we choose for pixel art.

Then lastly we also choose to implement a dark theme. This has multiple reasons, mainly because dark themes are very 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 then using the light theme.

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.

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 help of teammate. My programming skill was not enough good to finish every part with myself. However, with help of teammate I could do my part of job and it was precious experience for me. Overall project was well through, but there was some problems. One of the main problem was the miscommunication between teammate. Since it is teamwork, we should had to word as team. However, sometimes we misunderstand others and make same work twice or produce 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 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 cause me to do some useless thing and it cause the time 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.