Final Report

Looking back on the 'Go Green' project

Group Members

Jules van der Toorn	4952278	Yuxin Jiang	4771737
Dimitar Petrov	4865944	Rami Al-Obaidi	4770846
Aleks Bake	4872150	Kevin Nanhekhan	2959094
Jonathan van Oudheusden	4951522		

Introduction

The project started directly in the first week of the quarter. Everyone was curious as to who would be their team members. Everyone was present from the start and immediately it was clear that the team was very motivated to create an amazing product.

We decided on having a new chairman and secretary each week to keep the hierarchy balanced and give everyone from the team a chance to take a lead in the meeting.

The first 'real' meeting on Monday directly showed points of improvements. We gathered in the default computer room and didn't take any break during the chaotic 4 hour long discussion. The reason for this second point was plainly because we forgot about it due to all the ideas we had and our overall enthusiasm.

After the meeting everyone noticed that the discussion during the last half was very exhausting and wasn't very constructive. This needed improvement!

1. Scrum

We got introduced to Scrum and were immediately intrigued by it. We decided on fully dedicating to scrum for the first weeks and then later on reflecting if the methodology is a good fit for the team.

We made sprint plans via the milestones feature in GitLab[1], used the issue board[2] as our Scrum board and made a weekly sprint review markdown file. On the beginning all these things were quite overwhelming, but every week we got more used to maintaining them. There were definitely some doubts about it, mainly when we had to create the full backlog which took almost 2 hours. Despite this fact, after everything was in place, the team got more and more comfortable with the methodology.

We could make a weekly plan, called a 'sprint plan', in a few minutes due to the full backlog already being in place, which was sorted on priority. This gave us the mental space to focus on greater aspects of the weekly plan, like assigning tasks according to the unique skill of each member.

The sprint review always took the longest, as we always thought we had a lot of points to improve on. We also payed attention to personal development, with respective blocking issues, to get everyone back on track if something seemed to go in the wrong direction.

Reflecting on the sprint review itself, we sometimes noticed that it was hard to "see the forest through the trees", as the Dutch would say. Everyone had a lot of points of improvements to share with the group, which resulted in us not getting a clear overview of which points actually had the biggest priority.

To cope with this, we came up with the idea to unanimously write down three points of improvements spread out over three post-its. After this was done we would all stick them on the white board of the meeting room and then group them based on category. We would give each post-it group a name describing the category.

Terms that frequently reappeared were 'Communication', 'Task scheduling' or 'Task division'. The biggest groups immediately showed which points of improvements had the biggest priority and streamlined the sprint review process quite a lot!

2. Workflow

2.1 Communication

In the first week we used WhatsApp to communicate, but this showed it's limitations for a professional development team platform quite soon. You couldn't separate different topics clearly and there wasn't any support for sending code snippets.
We decided to try out Slack[3], which was a lot more suited for our needs. We could create separate channels for specific

1 Feature in GitLab to create a milestone for a certain date, see https://docs.gitlab.com/ee/user/project/milestones
2 Feature in GitLab to structure issues as a board, see https://about.gitlab.com/product/issueboard
3 Chat platform made for professional work environments, see https://slack.com/

subjects like ‘server’, ‘client’ and ‘database’. This way the team wouldn’t be bothered by unrelated subjects, but could still periodically check in to keep track of the weeks progress.

For code review, we used the discussion form on a merge request to write feedback about new code snippets. We agreed on never approving a merge request until all discussion was closed.

When we needed a team member to review a merge request, we asked this via private chat in Slack. The communication around this was always done in the form of a question, not as a command. This way code reviewing wasn’t seen as an annoying task, but rather as a sign of giving respect.

2.2 Continuous Integration[4]

Admittedly, because implementing Continuous Integration on the project gave extra points in the project, we looked into this workflow quite early. We never heard of it before so it gave quite the learning curve to implement. We created time for this by making implementing this the only task of a team member during the sprint.

After it had been correctly implemented that week, we immediately saw the benefits of it. Merge requests could immediately be declined if we saw that code wasn’t compiling or tests were failing.

Getting the hang of it, we also added support to the CI (short for Continuous Integration) for showing the code coverage percentage and the amount of CheckStyle warnings. This boosted the time for reviewing a merge request to a minimum, only having the need of checking the actual code quality.

As the project advanced and all the team members knew how to minimize CheckStyle warnings and improve the code coverage percentage, the need for more detailed CI feedback was needed.

Due to this, we added the feature to automatically upload the full code coverage/CheckStyle reports to GitLab by the CI.

This way all the team members could quickly look into the details of improvement, which improved the efficiency of code reviewing even further.

2.3 Pull-based development

As you could already read in the communication section, we used merge requests during the whole project. Because we didn’t quite well know the capabilities of the team we decided that this would be a good workflow to use to keep track of the code quality of each other. We of course also hoped that we could learn from each other by using this workflow.

This was really something to get used to, as we all worked alone on projects before this. Even small things like documentation needed to be reviewed for e.g. grammar mistakes, which felt unnecessary. Eventually this became and habit and a lot of bad code was prevented from entering the master branch.

An example of one of our merge requests can be found here: https://gitlab.ewi.tudelft.nl/cse1105/2018-2019/oopp-group-90/template/merge_requests/110

3. Design decisions

3.1 General

The template repository was already initialized as a Maven[5] project, which felt familiar because of using NPM[6] in the previous quarter.

One problem of using this structure was that there was only one Java folder to work with, which was quite limiting when you wanted to keep server and client separate. After some research we found out that it was possible to create nested modules inside the project, to easily create atomic sub-projects. We created a server module and a client module.

Looking into our repository you can also find a module called ‘tests’. This was created to handle everything around creating the aggregate JaCoCo report[7]. The major configuration change was setting the graphical user interface package to be ignored in the report.

3.2 Database

PostgreSQL[8] was an important asset for the development of your application. It is released under an open source license and meets most of the modern SQL standards. It has modern features that we believe we will help us develop a robust back-end for our project such as modern JDBC drivers, multiple specialized data types, concurrency controls, point in time recovery, robust authentication, procedural language support (PL/PGSQL), etc. It is also the relational database management system that all of the team members have experience with.

4 Workflow where new code is automatically tested on quality, see https://about.gitlab.com/product/continuous-integration

5 Among other things, a package manager for Java projects. See https://maven.apache.org

6 Package manager for Node projects, see https://www.npmjs.com

7 Library used to see branch coverage of Java tests, see https://github.com/jacoco/jacoco

8 Postgresqlorg. (2019). Postgresqlorg. Retrieved 9 April, 2019, from https://www.postgresql.org/about

We chose to use SQLite for testing because it is an excellent embedded database management system and can be used not just to test the Java JDBC methods, but also the SQL queries that these Java methods contain. In an earlier stage of development, we used DbUnit, it is a JUnit extension for testing database-driven projects but we faced an issue with the coverage report due to a bug within JaCoCo coverage provider, we decided to switch to testing with an embedded database.

The choice of JDBC over the JPA framework was due to its improved performance and direct communication with the database. JPA would have also caused issues with testing for the team since it is harder to debug. A drawback of JDBC is the maintainability of the code since SQL queries have to be stored in Java classes. We remedied this problem by storing the larger queries in an XML file so they are retrieved during runtime.

The database schema has been designed with care which ensured minimal data redundancy. All the proposed changes to the database schema are discussed during the weekly meetings, then an ER diagram and relational schema diagram are committed to our GitLab repository to be reviewed by the team before the changes are made to ensure maintainability.

3.3 Server

Choosing to use the Java Spring Framework[9] was a major decision made early in the project. The decision was made by evaluating the three major points when starting a project, stability, efficiency and privacy. For stability it was accepted after we saw that most of large companies on the market used the Spring Framework. For efficiency we discovered that it had all the libraries that the average developer needs for creating a standard application, and by sticking with it we avoided trying to connect different types of frameworks together, which would have been more prone to error. The Spring Boot library is the prime example as to why choosing this particular framework was extremely efficient. As for privacy, the Spring libraries contains most of the tools a person needs to mold a very secure application, with libraries such as 'BCryptpasswordencoder'.

The decision to use REST API was largely influenced by our decision to use the Java Spring Framework. The Spring framework does contain a lot of different libraries that help set the a developer on the right path of developing the desired application. The REST API just like the Spring Framework, supports a large variety of things and provides all the tools to deal with any issue that may come across. REST API unlike most other API's does not have a standard which makes it more flexible and easy to use for developers with not a lot of experience.

3.4 Client

The only real design decision on the client, excluding the graphical user interface, was which networking library to use. We decided on using the built in 'HttpsUrlConnection' library from the JDK. We decided on using this library because it is known to be mature and reliable and it included all the needed functionality.
We created a small wrapper around this, called 'NetworkManager', to pack all the basic needed methods and provide detailed feedback during runtime if something went wrong.

Due to this wrapper needing to inspect runtime exceptions, we chose to ignore the CheckStyle warning which discouraged the use of catching these. In this case, we preferred reliability of the product over having a perfect CheckStyle structure.

3.5 Testing

The reason we chose to use Mockito is that our project, as a fully functional program, has different parts which are dependent on each other. In the process of development, we could not guarantee that different parts could work at the same time. For example, when client side is done, but the server side is still under development, we can't test client side without server side code. As a result, we used Mockito which could mock the parts that haven't been done yet, in this case, all parts could be tested immediately after it is done and don't need to worry about the dependency.

We used MockMvc in our server controller, it mocked the client URL request, it is very convenient and easy to use because it help to start the in-memory servlet container. We also mocked the object in the test for DataAccess, which makes the tests look more organized.

3.6 Graphical user interface

Of course, the most design decisions were made on the graphical user interface. Beginning at the start, we chose to use the JavaFX[10] framework. The main competitors were Swing and JavaFX, as most other frameworks were not very mature.

We preferred JavaFX because it seemed more modern. Quoting from the 'Made for Dummies' website, "no work is being done anymore to enhance Swing, and Oracle has made it clear that JavaFX is the future"[11]. This choice was also motivated by the OOPP course.

Within JavaFX, we chose to obfuscate the ideas of 'Nodes', 'Panes' and 'Canvases' by making our own type called 'View'. It's actually a glorified so-called StackPane, but we chose this design to encapsulate all the unneeded functionality and to add our own useful methods, like adding the ability to easily add styling to the StackPane using Java code. This made it easier for other team members with less experience to easily work with the JavaFX library.

9 Framework to create server in Java, see https://spring.io/

10 Framework used to create graphical user interface, see https://openjfx.io

11 Dummies. (2018). 10 DIFFERENCES BETWEEN JAVAFX AND SWING. Retrieved April 9, 2019, from: https://www.dummies.com/programming/java/10-differences-between-javafx-and-swing/

We also further specialized on the ‘View’ concept, creating a ‘ListView’, ‘TimelineView’, etcetera. This made it very easy to later on put graphical user interface components on other new locations in the application (for example, our application has a timeline for the user and his friends, but also a personal timeline on the profile page).

For the actual visible design decisions, we based the overall theme of the application on ‘material design’. Android uses this style too currently, giving our project a modern and professional feel.
Laying out of the components was quite hard to decide on. To make sure the product won’t be obscure or unclear, we mainly looked at Slack for the navigation aspect. To make the product feel mature, we looked at Facebook for the social aspect.

The green theme of the project reflects the ecological friendly atmosphere around the application.

We added the feature for an account to have a profile picture to feel closer connected to in-application friends. The personal aspect of the picture will make users of the product feel more like there’s an actual human behind the contribution timeline, stimulating their own progress as well.

3.7 Gamification

Each of the activity that user can complete in the application help reduce the CO_2 emissions. Each activity awards the user a specific number of points. We researched about the CO_2 saved by these activities before deciding the green points for the activities. Most of the activities award 1 green point for each 0.1 kilogram of CO_2 saved. However, some activities don’t have proportional point representation as it will result in big amount of points awarded for that activity and it can result in users only focusing on certain activities and demotivated in saving the environment in other ways.

A PDF file about the points and amount of CO_2 saved can be found on our GitLab repository.

Users can compare the progress with their friends through the leader board so a sense of competition is provided to motivate the users.

Achievements are awarded to users who complete a set of activities in a specific category. They motivate users to try wide range activities as it provides sense of accomplishment that can be compared with their friends.

3.8 Privacy and security

We have things in place to protect the users information and secure our application. All requests to the server are with HTTPS with a self-signed certificate. This makes all our traffic from the client to the server encrypted and therefore protected from others. The certificate is on the client and the server. Because of this the client doesn’t accept a response that doesn’t have the same certificate and thus only accepts responses from the server, protecting it from man in the middle attacks.

We protect the users information in a couple ways. First all our passwords are hashed and salted, using B crypt. This is one of the best hashing algorithms, in terms of security. Making it almost impossible to get the users password, and use it to login. Server-database communication occur through prepared statements so SQL injection attack.

If the user would like to keep its activities private from others, we have that option. Then the user can set his/her account to private, this prevents its contributions from showing up in the time line of friends. The friends will only be able to see the total points, but not how the user got them. You can still see your own time line.

4. Value sensitive design

Our main design choice for our application is to ensure that it is simple for the end user and encourage our users to join in combating climate change by changing their daily habits to reduce their CO_2 footprint. The app is designed to be accessed on a global scale without any discrimination towards a specific ethnicity, religion, gender, etc. It is designed to be accessed affordably with just a computer and an internet connection. We hope we can give our users a sense of community action against climate change and reduce the CO_2 emissions worldwide and set a precedent that humanity needs to fight climate change in order to preserve planet Earth for future generations.

Research is still needed on how users can contribute to reducing CO_2 emissions. Environmental scientists can provide great input on the application design and how to improve it to make a better effect in reducing CO_2 emissions and how the activities that can be done by the users be refined to be more representative of an actual effort to reduce CO_2 emissions.

The user experience needs extensive feedback from sociologists, behavior scientists and psychologists to further enhance the user experience and refine the application features to make our applications more engaging and appeal to more users but to also make our users more motivated about reducing their CO_2 footprint and protecting the environment. Research is also needed about the possible social and psychological effects that can stem from the usage of our application such as excessive usage and how these effects can be mitigated.

When a user first uses our application, they have to agree to the terms of service, privacy policy, and license agreement. Due to time constraints and lack of expertise, we were not able to draw up those legally binding documents properly so we can provide the user with their rights regarding the application and their personal data. Recruiting legal experts to write these legal documents would have ensured that our users are fully aware of their rights and obligations when they first decide to use our service.

Since the application is being developed for a Dutch organization and intended to be used on European Union territory, we have to ensure our application complies with the European Union's General Data Protection Regulation (EU) 2016/679 ("GDPR")[12]. To ensure compliance with GDPR, we have taken multiple design choices[13] into considerations which are outlined below:

- **Privacy by design:** Throughout development, we put the user's privacy at the center of our design process. All proposed changes had to go through discussions to ensure that the user's privacy is protected. The application also complies with Article 3 of GDPR which restricts the collection of user information to only the necessary user information. Only first name and the email address of our users are stored in the database which is needed for greeting the user and retrieving a forgotten password or to communicate with our users.
- **Ask for explicit consent:** The user is asked explicitly to agree on privacy policy during sign up.
- **The right to be forgotten:** The user is able to delete their account which will result in all of their data to be permanently removed from our database. Explicit consent is needed by the user before account deletion.
- **Transparency:** The privacy policy contains information about data transparency and how user data might be used.
- **Encryption and data storage:** The users 'passwords are never stored in plain text in our database and authentication is needed before any private data can be accessed. More about this in section 3.7 of this report.

However, there is a long way before our application can be fully compliant with the EU's GDPR. We lack a communication platform that can enable us to communicate with our users in case of a data breach. Such a communication platform can also enable us to respond to users' requests about how we use their data or if they need a copy of their data (Subject access requests). Documentation about the data we use needs to be written for administrative purposes. The documentation should contain logs and justification for the data collection. Legal experts or a data protection officer would have guided us on how to make our application is fully compliant with the European Union's General Data Protection Regulation directive.

We chose an end user license agreement (EULA) to ensure that our third parties are held liable in case of misuse of our project for personal gains or to further any kind of agenda, such as using users' data for political motives such as to support environmentalists or anti-environmentalists groups. Third parties are also held liable for forging, reversing engineering the software to prevent misuse. Users must agree to the license agreement (along with the full terms and conditions) before sign up, otherwise, they won't be able to use our service. As this project was conducted for a Dutch institute, the law of the Netherlands applies. However, we believe EULA have might some drawbacks to our project as it limits its usage for the creative community. Extensive research is needed about license agreement before it can be can professionally be finalized.

PDF documents of the terms and conditions and the license agreement can be obtained from the registration page or the login page of our app or our GitLab repository.

Security and privacy were very important during development. Some of these choices are hashed passwords, encrypted connection, privacy settings. In-depth information can be found in section 3.7 of this report.

5. Conclusion

After working on this product for 10 weeks with a team of seven members, we can all conclude that we're very proud of ourselves. Our motivation never went away, it might even became bigger as the project continued. We worked really as a team and always helped each other when something went wrong.

Of course, there is always room for improvement. Development wise, it's hard to come up with new points of improvements after tweaking them for 10 weeks. We do find ourselves not balancing the work load well enough sometimes, due to differences in experience in the computer science field. This would improve if we just continued to work together, eventually evening out the amount of knowledge per team member.

Product wise, a lot more features can be thought of that are nice to add. One of these things is having the possibility to recover an account using the provided email address, where a 'reset password' email could be sent. This was hard to implement during the project because we didn't had our own email server available.
Tons of other small things can be added like support for setting a status, but these are easy enough to reflect on yourself. We decided to focus fully on creating the minimal viable product, and making it as streamlined as possible.

Thinking more wildly, porting the product to Android and iOS would greatly improve the amount of users, as almost everyone uses mobile apps nowadays.

12 Europaeu. (2019). Europaeu. Retrieved 10 April, 2019, from https://eur-lex.europa.eu/eli/reg/2016/679/oj
13 Mobiloudcom. (2018). The MobiLoud Blog. Retrieved 9 April, 2019, from https://www.mobiloud.com/blog/gdpr-compliant-mobile-app

6. Individual reflection

6. 1 Jules van der Toorn

Weaker points

My weakest point in the project was definitely taking too much responsibility for the team. I've remodeled the core Models of the project just before I was going away for the weekend because I thought that it had to be done. In hindsight this was just really stupid of myself. Only when you work alone you can do these kind of things, but for group work you either have to plan ahead better or thoroughly discuss changes like these.
The positive aspect of this is that I did learn a lot from these mistakes and got better each week in trying to making the best of the tools we had instead of trying to redo everything.

Stronger points

My strongest point during the project was my leadership role. I've helped a lot in the beginning to get all the important structural things in place, like writing guides on how to create the weekly reports. I also worked on general things to improve the workflow, like implementing Continuous Integration on GitLab and always positively approaching people on Slack or during the meetings. I tried to learn a lot on Scrum and agile development and immediately tried to apply it to the weekly meeting to see what worked and what didn't. Due to this search for improvement I also got better at dividing tasks and recognizing the strong and weak points of people.

6. 2 Rami Al-Obaidi

I worked on the back-end side of the project specializing in the database department. I was responsible in creating and maintaining the databases that we used for our application (PostgreSQL) and for testing (SQLite). I worked on making the server be able to send and receive data from the database and optimize the methods/queries work in efficient ways. Me and my team were able to communicate efficiently. We always stayed in touch when were away and had open discussions to solve any conflicts. Our weekly meetings helped us plan ahead. Our planning early in the week enabled us to meet deadlines on time. My weak points were mainly to do with my weakness in networking as I wasn't able to help with working on client-server communication. I feel this is a part where I need to improve on later as it is very critical in application development. I also felt that I lacked in contributing in other departments of the project, but I compensated this by perfecting the database department. My strong points was mainly related to my good background in databases and my passion for data related topics. I was able to contribute to my team with helping in ensuring that we have a stable and efficient database and guide them on testing database related methods in the project. Overall, I was able to improve my skills in back-end development during this project especially with regards to database connection.

6.3 Yuxin Jiang

It is my first time to work with others as a team to make a programming project, I was mainly in charge of server side, which is a brand-new area for me, I searched a lot of information on how the server side code should be structured and definitely grasped lots of new knowledge. Even though some of the task could not be done by me completely, I still learned a lot from looking into my team member's code.

My first weak point is that I still have difficulty solving a problem I encounter by myself, sometimes I can't connect the different parts of knowledge I learned before. When I got stuck on some point, I got a little bit impatient and not confident about my ability. Secondly, I think my strategy to search information online still need to be improved, at the beginning of the project, I spent quite a lot time on looking for the right information I need, but the situation is getting better as the project moves on.

As for my stronger points, Firstly, I will say I know better for myself how a complete java project should look like, how the different parts as GUI, client, server and database should combine together to make the whole program work. Besides, I feel more comfortable working in a team after this project, I am more clear about my role in a team and how to contribute to a team.

6.4 Dimitar Petrov

To start with, for me this entire project was an amazing experience. I had a lot of fun during the last two months and I'm really grateful that I had the opportunity to work with such inspired people.

Personal development

So, in the beginning I was a little lost, because I didn't know what exactly I should do. But as time passed, I began to get used to the environment . I chose to be in charge of the GUI. I had never programmed in JavaFX before, but I wanted to learn it. So I got what I wanted and managed to design the whole visual application..

Weaker points

During the learning process, I encountered a lot of issues. Luckily my team members were really helpful. Every time I got stuck they would try to figure out a way to solve my problem and help me with it. Also, at the beginning, I had mentioned that I struggle to motivate myself sometimes. However this was not an issue during this project, because I really enjoyed the process. My biggest weak point,, is that sometimes parts of my work remained not fully finished and people had to compensate for me.

Stronger points

I created the layout for the pages and designed them. I connected some of them to the actual data and actually made them work. During the meetings I came up with various ideas for the project itself:

In summary I managed to reach my goals for this project: I learned a lot of new things and gained some experience as in working as a team. Also, I met with some awesome people and saw their point of view for the project.

6.5 Jonathan van Oudheusden

For the application I started with working on the server part. Making the basic methods. I worked on sessions and how to make that work on both client and server. When more of the essential server methods were done, I focused more on the client and helped others to work on the server.

Weaker points

Sometime during this project when something had to be done, I would work alone to get it done without asking the help of team mates, which could have helped me. This would make it that I had to spend more time than was necessary to to get it done. I improved on this quite a lot, when I have something that I don't have/want the time for, I asked someone if they can help me, and every time someone could. I didn't have any programming experience outside of what I learned in OOP, so almost everything was new to me. I did not know how to do what needed to be done in this project. I learned a lot and now I have knowledge I didn't have before.

Stronger points

Like I said in my personal development plan, I was/am very interested to learn everything needed in making a working application. Therefore I worked hard and when problems came up I would spend a lot of effort to make it work. There were times when this would mean working after midnight and one time all night.

6.6 Kevin Nanhekhan

I only worked on the GUI, so I did not do any coding on the server/database/client side of things. But despite me doing only GUI side of coding, I think I did a good job together with Dimitar and Jules who also worked on it. This is because everything looks like how we as a team had discussed it.

Weaker points

There were some unexpected problems during the project such as a lot of tests failing where I as someone who works on the GUI could not help much with because I did not know how the testing exactly was done. But despite this I tried to help as much as I could for example for merge request look at the code and see if it works as intended. I also looked if it made sense for as much as I could comprehend it.

Stronger points

Even though I'm not much of a talker as said in my personal development plan, I tried to join in on the conversations, even though it wasn't much talking from my side. I also tried to learn from the others for example how client/server side is done and such. Even though I did not have much experience with programming, I know I could always depend on the others in case I was stuck on something. This also applies for the other way around, in case the others had questions on things I understood, I tried to explain it in the best way I could.

6.7 Aleks Bako

Reflection

The project was an amazing and unique experience. The joy of working on one thing with a group of people to reach a common goal was great, and I think that there was no better group of people to work with than the one I had the pleasure of working with.

Weaker points

I would say that the biggest weakness for me throughout this project was my lack of curiosity. I had specific ideas for a basic application and I searched for just a simple solution to solve the problems that were needed to be solved, not giving a second glance at the possibility that there might have been a more efficient way of dealing with it.

***S*tronger points**

One of my strongest points throughout the whole project was the control of the server and the awareness that if I were to not do my part then there would be a lot of issues that would follow in other parts of the project. Also developed organization skills, which helped shape our team work with how to improve our teamwork.