Group 46

Presented by:

Mirijam Zhang, 4660129

Qingna Zheng - 4834127

Tim Anema - 4953940

Stephen van der Kruk - 4832868

Daniela Toader - 4904648

Shah Farooq - 4792394

Momchil Bozhkov - 4806816

Table of contents

OOP Project - GoGreen
Final Report

GoGreen represents a Java application that is going to encourage people to engage in a more sustainable lifestyle. It consists of two main components: the application part, where the users can submit environmentally-friendly activities, as well as keep track of their journey in becoming "greener", and the gamification part, which enables the players to compete with their friends in order to determine who is going to have "the greenest" way of life.

The application offers them the opportunity to change their habits when it comes to food, transportation and use of energy and it determines them to embrace a fun approach towards a better future.

In order to make all of this happen, we have been conducting an elaborate process spread over multiple weeks of work.

1. Process

1.1 Plan management

We predominantly managed to stick to the planning as we were in time for the demos with the required functionalities. Sometimes we prolonged the self-made deadlines for some of the tasks as not everyone was finished in time but overall we had a good workflow.

Meetings were where most of the planning was accomplished. We reflected upon each sprint review to better evaluate the difficulty of each task and how much time each feature would require, which gradually improved our efficiency and accuracy laying out a thorough and detailed course of action for each point addressed.

We followed this planning model throughout the entire project, including weeks in-between deliveries. While we mostly stuck to the decisions we made at the beginning of each week, some improvising was required as we encountered problems we either had not expected or obstacles that required more effort to overcome, mostly on an individual level, therefore requiring slight on-the-fly deviations from the plan, but ultimately following the deadlines.

All in all, despite minor difficulties in the planning process, all deadlines were successfully met and we believe we managed to do a good job of delivering the results we desired, thanks to great team coordination.

1.2 Team collaboration

The collaboration went nicely as everyone tried to help the other teammates as well as they could when they asked questions or had different types of problems. We were very receptive to new ideas while showing a lot of interest in improving our project.

1.3 Communication

Although we lacked communication in the beginning, we managed to adapt in order to provide enough discussion topics so we could make everything more clear. We used messaging applications and extra meetings with the purpose of boosting this process.

1.4 Version control

Version control played an essential part in developing our application as it ensured that the entire process went smoothly, enhancing collaboration, working in parallel and preventing code conflicts, as well as being able to revert and compare between previous versions of our product.
An extraordinarily useful feature provided by Git is the ability to work on multiple branches in parallel, and have a protected master branch, where only solid and fully tested, code reviewed features are implemented. The !3, !44, !45, !47, !50 discussions in our merge requests properly reflect all of these aspects and provide an insight to our usual task completing procedure. We took full advantage of the provided tools, which ensured that everyone involved in a part of the project understood exactly what the others have implemented and how.

1.5 What did you learn?

We have greatly improved our ability to cohesively cooperate as a team instead of trying to piece together individual work. Moreover, we have gained invaluable experience in using widespread and extremely useful tools such as git and in taking advantage of their specialised features, CI for instance.

2. Product & design decisions

We divided our team in a backend and a frontend team. After that we used controllers to implement functionality and interaction with the database. This took the most effort since most of us needed explanation on the API's and endpoints and much more.

3. Server

The server uses Spring Boot. The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications on any kind of deployment platform. We made this decision based on several factors, the most important of which were available documentation, previous experience and versatility of the framework.

The server has several layers: controller layer, service layer, repository layer.

- *Controller layer*: Handles the ‘initial contact’ with the request. This is where a request first ‘reaches’ our code. The controller layer handles things like checking authentication, verifying input, checking if a given target (user/profile/category/activity) exist, and finally calling the service layer.

- *Service layer:* This is where most of the logic of the actual request is handled. The service layer can be a simple call to the repository layer (retrieving profiles) or it can call other services and trigger events and then return a response (submitting activity / registering).

- *Repository layer:* This layer is responsible for retrieving data from the database. This may sound like a lot of work, but spring & jpa & hibernate handle most of it.

3.1 Architecture

Both of our services (main API and CO2 API) are containerized applications. This way we can easily deploy our services to our server(s). Since we have several containers and dependencies we use docker-compose to handle creating, pushing, pulling and running our containers. This enables us to configure a YAML file to configure our application’s services, which can then be simply started with a simple command to execute them.

Our architecture is relatively simple, and easily maintainable, as can be seen in the appendix. We make heavy use of Digital Oceans’ services, which is our main cloud service infrastructure provider.

3.2 Security

For this project, we chose BCrypt as our default password encoder. There are a lot of hashing algorithms, but BCrypt is an industry-standard, and was already shipped with Spring Security. The people working on the server would have prefered something like Argon, but unfortunately this algorithm is relatively new and therefore not implemented by a trustworthy source in Java. Most of our authentication is handled by Spring Security. However, email verification and 2FA is handled by our own custom filters.

Security was very important to us since we wanted to ensure the safety of our users' data. All forms of authentications were heavily tested to make sure only user with the high rank privileges had access to restricted server endpoints and other users would be barred from entry other than what was granted to them. Also to prevent something like the recent Facebook debacle[1].

Session management was also introduced to keep track for logged in users so they wouldn't need to login every time they have send requests, which would also expire after a determined amount of time of inactivity.

To ensure our user data is safe, we have deployed several security measures. The first thing a request to our server(s) will reach is a hardware firewall, owned by Digital Ocean[2]. This hardware firewall is configured to block all traffic except on ports 22, 80, 443. Once the request has made it to our server(s), it is 'greeted' by a software firewall, UFW to be exact, which has the exact same settings as the hardware firewall. This way our server is always behind some firewall, if the hardware firewall fails for whatever reason and the traffic is still let through (very rare!).

After that, all traffic is routed through a nginx server, which serves as a reverse proxy. Nginx then proxies the request to the internal containerized service, to make sure all traffic is routed over a secure connection.

The internal service makes request to the other containers. The other containers (SQL DB and CO2 API) are not accessible from outside. The SQL database can only be accessed from other local containers or by using something like a SSH tunnel.

3.3 Scalability

As can be seen in the appendix we use Digital Oceans' floating IPs. This means we can route all traffic to a floating IP, to a random server in the datacenter, without any downtime. We can effectively switch server -as long as the server is in the same datacenter- without downtime and changing DNS records.

However, in its current state the application can only be vertically scaled, not horizontally. There are two major reasons for this.

The first reason is our database. Right now the database is a container on the same machine as the API. This means each instance of our server would have its own database. Luckily this can easily be solved by separating the database from our docker-compose config and running a seperate PostgreSQL cluster.

At the moment our server is a stateful application. In our case, the server has session data stored in memory. This means separate instances of our server would not be able to serve clients who authenticated on another server. The easy fix would be to use a Redis cache, which stores the session data.

---

[1] Facebook Newsroom. *Keeping passwords secure.* 2019. Accessed on April 10, 2019 from: https://newsroom.fb.com/news/2019/03/keeping-passwords-secure/

[2] Digital Ocean. *Cloud Firewalls.* 2019. Accessed on April 10, 2019 from: https://www.digitalocean.com/products/cloud-firewalls/

A better way would be to switch from sessions to something like JWT. That would make our servers truly stateless and, combined with a PostgreSQL cluster, both vertically and horizontally scalable.[3]

3.4 Co-API

We wanted to implement an accurate CO2 tracker that collects data real-time from different sources that are GDPR compliant and makes heuristics about the user and their consumptions. Initially, we got access for the “CoolClimate” API from UCBerkeley however it was too complex with over 300 variables in every response in XML format. We decided it wasn’t worth it since a lot of time was consumed in parsing and mapping all these data items to and from, an estimate of 3 seconds per response was unacceptable. We acquired another source “BrighterPlanet” API which had an exceptionally simple requests and swift response times, which we optimized further using our caching mechanism.

We tweaked our request parameters according to the user’s activities and got precise reading for the CO2 output matched with its reductions, which we can converted into points using our complex algorithm.

3.5 Modules

The Maven build management tool was used to structure our large project retrieving a number of dependencies from a central Maven Repository and ensuring the project would have the same contained environment for every group member by specifying the same configuration, plugins and profiles. We ran the project on a wrapped jar and it also integrated very well on GitLab.

We separated our modules according to the MVC design pattern of keeping the server, client and gui all separated from each other and avoid unnecessary cluttering therefore each module could be standalone and future-proof.

3.6 Data Storage

For data storage, we used H2 Database for the development and testing phase and then ported it to the industry-standard PostgreSQL. The data was structured in a manner that every data item in a table was functionally dependent in 3NF; this was handled very well by Hibernate and JPA integrating well with Spring Boot’s Framework. Using these frameworks made sure that our data was persistent and followed the ACID principles.

The database was hosted on a server with security measures in place preventing any promiscuous unauthorized access by granting only the required privileges. We made use of springboot’s autoconfigure security library that added an extra boost to our security.

We applied the concepts learnt in the Web & Database Technologies course in Q2 of implementing normalized tables, foreign keys (showing the use of relational data models), triggers – for example: on completing an activity, a trigger would be sent to the badges repository and a badge with a UUID would be generated for the user relevant to the category of the activity submitted.

4. Client

On the frontend we worked with JavaFX and SceneBuilder. We decided on using JavaFX thanks to its more modern and broadly supported features and tools compared to alternatives such as Swing.
We chose SceneBuilder since it was much more visual and allowed us to spend less time on creating a design. We first divided the wireframes and made the outline with SceneBuilder. During the

[3] Wikipedia. *JSON Web Token.* 2019. Accessed on April 10, 2019 from: https://en.wikipedia.org/wiki/JSON_Web_Token#Use

meetings we decided on material design and made changes we all agreed on. We went for a dark, minimalistic theme with lots of green.

4.1 GUI Layout and Features

The layout of our application is a basic sidebar with next to it a pane to display the page.
After a successful authentication, the user is sent to a page that contains a sidebar and an empty pane to the right. The controller initializes this pane to the homepage. We used SceneBuilder to create the structure and positioning of the static attributes within the pages.

Minimal styling was done in the fxml pages, because the static attributes needed minimal styling. Most of the styling was done in Javafx, because most dynamic attributes needed the most styling. CSS was used to add styling to groups of attributes and if there was no other option. An example is to have an transparent background for the scrollpanes. We chose for a minimalistic design. So features of this that we used are: simple icons and font, hiding information which only gets shown when the user needs wants it. Furthermore we used an only slightly contrasting green color scheme and a stylized background to not draw much attention from the user. Therefore the user can focus on the information on the pages.

4.2 Validation

We have several points of validation. First of all to log in, the user has to sign in with his/her username and password. If the credentials are incorrect a message will be shown telling the user to try again. For the sign up we decided that the username has to have at least 3 characters and the password at least 5. If the user enters an invalid username or password, the user will be shown a message prompting him or her to create a valid username/password. Lastly we have our submit page, where the user can submit new activities for points. Fields cannot be left empty, if left empty an error popup will be shown.

To make API requests to our server we used a widely-popular library called UniRest that was extremely valuable. We built our own generic version of the unirest library making different requests that would map the server responses to our server models, confirming that the right classes were being used and each data item was mapped to its respective variable.

We ensured to implement a caching mechanism that would return frequently accessed static data from cache instead of making new requests to the server. We kept our client-side relatively stress-free and did a lot of the heavy lifting server side with fetching, storing and manipulating data mostly done server-side. The client had stripped down version of all the data keeping our architecture clean and client requests relevant. As for all our modules, testing was our prime priority to make sure our code was functional was returning the correct responses designed for client interaction through the GUI.

5. Reflection

5.1 Points for improvement

There is always room for improvement and as far as the communication between peers is concerned, we all had different backgrounds and varying levels of experience, therefore evenly distributed collaboration and effective dividing of labor was a definite challenge. Although we had split tasks according to our strengths and interests, we could have taken the time to more thoroughly explain the approaches we had chosen for our individual piece of work and the rationale behind it.

While we encountered certain inconveniences in the communication department, conflicts have been avoided, thanks to the willingness to collaborate of all team members. A more serious issue we encountered was the difference in experience and expectations. While for some members, this course represented the first project of any importance they have ever worked on, for others it was one of many such projects in their career. Less experienced members had at times difficulties keeping up

with the expertise and knowledge of others who had an extensive previous background. Thanks to thorough planning and a helpful and understanding attitude from peers, the damage caused by this issue was minimized. However, improvement could be present on both sides, as the less experienced members could have put more time into studying the technologies used by the ones with a professional background and the practices they employed, and a stronger sense of empathy from them could have encouraged those newer to the industry to do so more effectively.

Overall, the performance of our team was not decisively negatively influenced by any of those factors, and each of them have been individually dealt with through various means, such as meticulous labor division, thorough planning, extensive sprint reviews and scrum usage, and although there were setbacks, we believe we managed to achieve what we aimed for and that the cooperation between us was great. Moreover, we consider that some points of improvement when it comes to the course could be providing more study materials, extensive lectures and constant feedback.

5.2 Personal feedback

Tim. The project went about as expected. We used some libraries I never worked with before, some of which I hope I never have to work with again. It was speculated this would be a big project, but I felt that wasn't really the case. It was more of a standard 'administration application' with a climate twist thrown in. So I didn't think the actual project itself was anything special.

Enough about the course, more about my personal development. At the start of this project I set two goals for myself: not being a control freak and pacing myself. I feel I succeeded in not being a control freak; I tried to only work on the server as much as possible, and seeing as some of my feedback from the group was "*it would have been nice if you took the leadership*". However, I do think I did not achieve my second goal of pacing myself, and giving clear explanations.

Besides a few performance issues in the last two or three weeks of the project, I feel there were no real problems in our group. Of course, there always is something to say about communication (or lack thereof), but I feel that we did all we could in ten weeks.

Mirijam. I have learned so much during our project. From working in a group I've learned that communication is so important. I have become less shy to ask for help, and more confident to speak up. I have learned a lot from the more experienced team members while still trying my best to help others that wanted my help. I still find it hard to give negative feedback, but I have done that during the project and that's a huge personal improvement for me. Working in this project also taught me how to work with git, which I still find really interesting and still do not fully grasp.

Furthermore I feel like I know where to begin when creating a program, at the beginning I was lost, but after 1/2 weeks I knew what was happening and what still had to be done. Seeing how everything came together with trial and error was really exciting and fun. It was frustrating at times but I always had my team members who were willing to help me. I had no experience using SceneBuilder, JavaFX and making connections with databases, I learned that as well which expanded my programming knowledge. The project was a great experience and I have developed personally and I have also gained more knowledge about computer science in general.

Qingna. In this project, I was working on the Gui part. As working in a team, I believe that I still have a lot more to improve in the future. As I mentioned before in my Personal Development Plan, I want to improve my English skills and programming skill during the project. In fact, I think my English has improved but not too much. Every week, we have meetings together, everyone talked a lot with each other which can improve the skill of communicating.

Also, my programming skill is better than before. We decided to use JavaFX in the first meeting. We can easily design the user interface by using Scene Builder and focus on the controller part since the view and controller are separate. For the JavaFX, I watched a few videos online and read one book about it to learn it. My stronger point in the team is that I will try to finish my task by myself but since it is teamwork, it's better to ask help from others and finish everything on time. My weaker point is I

cannot explain things clearly and sometimes I would forget what I am going to say due to the poor English skill. Another weaker point of me is that I am not good at coding, usually, I took a lot of time on searching things online for the same thing and leave the rest until I finish the previous task.

Stephen. During the project I kept myself busy with the GUI. I got more comfortable with programming in JavaFX, but also with Java in general. I'm satisfied with what I achieved. I didn't get to learn more about backend though. I worked on my own tasks instead of being a control freak. It turned out better because I didn't have much time management issues that way. There is enough documentation to learn about the structure of our backend, so I will look at that to learn more about our backend.

During the meetings I was trying to organize the team by dividing the tasks for the week. It was tough getting everyone to think together because everyone was busy with their own stuff. I tried my best to be assertive and make choices. I believe that it helped getting the team ready for the sprint, so I was being a bit of a control freak in this case, but in the end, it did help the team.

The biggest issue we as a team experienced in my opinion was the lack of communication and sometimes miscommunication. To overcome this issue, I was trying to keep the team updated with the deadlines. That was sometimes without success. I'll have work on keeping the team updated such that everybody knows what's going on without being intrusive with deadlines.

Daniela. This course has been an excellent opportunity for me to broaden my knowledge of both Java development and widespread industry practices. Of my strong points, communication and persistence have helped me overcome most of the pitfalls that I encountered, with the help of my team-mates. While at points during the project I struggled to get a good grasp on the overview of how all the technologies tie together in the project and that hindered my ability to accomplish everything I wanted to, I managed efficiently communicate with my team, and together we succeeded to both understand each other's contribution to the project and to contribute our part before each delivery.

As far as my weaker points are concerned, although the lack of experience and predisposition to over-stress I mentioned in the personal development plan have had a part in my contribution to the project, I believe that my increased awareness of their existence has helped me minimize the effect they had on the team as a whole. Developing this application has not only provided me with valuable experience and expertise which will help me build a better understanding of programming in a team, but also made me evaluate the difficulty and importance of tasks more accurately, therefore significantly reducing the negative consequences of my weak points.

All in all, GoGreen was challenging and necessary first step in my hands-on career, answering many of the questions that have arose about the practical appliance of all the theory-based concepts we have learned in previous course.

Shah. At the start of the course, we were told this project was more about teamwork rather than just programming. This was crucial at so many steps along the way since we'd often have people working on the same thing or having nothing to do. This led us to focus on team management skills and progress started to boom.

This experience has led me to acquire new skills like problem solving when handling unexpected exceptions, future-proofing for scalability & failures, adapting to unfamiliar situations since there were numerous libraries used throughout the project. This new skillset was enhanced and captured in my work, a new found respect for testing was also developed. In retrospect often I would've preferred things to have gone perfectly, for example the API - I obtained credentials for earlier had a lot of complexity and bugs wrapped into it therefore a lot of time was spent resolving them and, in the end, I used another simpler, cleaner and faster API - perfect for use case. If things would've been different, I wouldn't have learnt as much about web/server optimizations and clean coding standards. Looking back to my PDM plan, which was ambitious and optimistic to say the least, I was most satisfied with the end-product after several man-hours spent on StackOverflow, implementing functions, communicating ideas, debugging and testing. I had a great team that was passionate about the project and I really enjoyed my experience – which will without doubt be crucial for future projects.

Momchil. During the past 9 weeks I achieved a lot of progress in what was one of my bigger initial goals, which was to acquire a bit more knowledge on what it is like to actually develop a full-fledged software app. The process was very long and at times frustrating but nonetheless I believe that not just me but all of my team learned a lot of new skills. It was very hard for me because the project was big but towards week 2-3 I started getting up to speed on what it actually meant to be in a real-time developing software team.

I obtained a lot of new communication, teamwork and software developing skills which in no doubt will be useful during my future career as a software developer. I had a fantastic team and though we may have had some issues along the way they were very minor so it was easy to work through them and all the technical problems were easily resolved. In conclusion I do believe that I extracted as much as possible from CSE 1105 and would love to do it all over again and even on a bigger scale with my new acquired skills.

6. Value Sensitive Design

The value which we thought was the cornerstone of our project is privacy. That is reflected in the measures we took to protect our users' data, from properly encrypting passwords, to adding both physical and software firewalls in order to ensure the data the user entrusted us with is safe and private to only themselves.

Stakeholders for our current product include the demographic of people who are concerned with the wellbeing of the environment and the sustainability of the future. Moreover, due to the gamification factors involved in our application, as well as the social aspect of it, stakeholders could also include a younger audience that is interested in such features. Therefore, it can be said that our application is designed for the environmentally aware person who enjoys competition and socializing. If the stakeholders for which we were designing the application would change, that would result in fundamental alterations to core concepts of our product. One of those could be the gamification dimension we integrated into the functionality: had we targeted an older demographic we might've removed implementations such as achievements and badges, which appeal to a person who is more familiar with the world of computer games, and instead we could've spent the time on minimizing the difficulty of accessing and using the application, making the learning curve more tolerable to people who are not as tech-savvy. Moreover, another significant change we might've made had we designed for different stakeholders would be the amount of emphasis we put on privacy. As younger people tend to be more aware of the dangers of online data sharing, that makes it a certain amount easier to design a properly secured application for them as they have more experience and a better understanding of the online environment than people who are either very new to the internet or people who struggle to grasp many of its concepts.

To gain a better understanding of our stakeholders and their needs, we could consider consulting social scientists and environmental scientists who could provide us with a better understanding of what our clients would be interested in and how that would correlate with the goal of our application and the values behind it.

There could of course be clashes of interests between our stakeholders. Some of the users of our application might be more interested in the game-like aspects of it and the social structure it provides, whilst others might argue that the core of the application lies in the promotion of sustainability and the aforementioned aspects might in fact be harmful to our end goal as they take focus away from it. We believe a certain compromise can be found between those opposing views, as although they both have valid perspectives, the elimination of one of those aspects would lead to an entirely different end product, which would appeal to a much smaller audience, and therefore be less effective at reaching the desired outcome, which is to lower the carbon footprint of its users in an enjoyable manner. Therefore, we believe a sweetspot can be met through experimentation and research that would satisfy both parties and still retain the quality of the product.

Appendix

Used libraries

Project-wide dependencies:

- hamcrest library (1.3)
  - Matchers for flexible testing
- lombok (1.18.4)
  - Saves a lot of basic code
  - No longer need to test trivial code
- commons-logging (1.1.1)
  - Package to use loggers
- commons-io (2.4)
  - Utilities for IO functionality
- javassist (3.20.0-GA)
  - Bytecode engineering toolkit
  - We don't use it ourselves, but it overrides a broken dependency for mocking
- powermock-api-mockito2 (2.0.0-beta.5)
  - PowerMock API for Mockito2.x
  - Used for mocking/stubbing/spying
    - Basically everything Mockito cannot do and more
- powermock-module-junit4 (2.0.0-beta.5)
  - PowerMock support module for JUnit 4.x
- mockito-core (2.23.4)
  - Mockito library
  - Used for mocking
- commons-lang3 (3.0)
  - Provides some methods that should have been in Java
- commons-collections4 (4.1)
  - Adds some extra data structures
- junit (4.12)
  - Used for writing unit tests

Client-specific dependencies:

- gogreen:shared (LATEST)
  - Our shared module
- unirest-java (1.4.9)
  - Library to make HTTPS calls to our server
- httpclient (4.3.6)
  - Dependency of Unirest
- httpasyncclient (4.0.2)
  - Dependency of Unirest
- httmime (4.3.6)
  - Dependency of Unirest
- json (20180813)
  - Dependency of Unirest
- jackson-databind (2.9.8)
  - We use jackson for object mapping
- objenesis (3.0.1)
  - A very handy and small library that allows us to initiate a new object of a class without needing a default constructor.
  - We used this in our client side API to mock real requests as a placeholder for non-existing endpoints.

CO2API-specific dependencies:

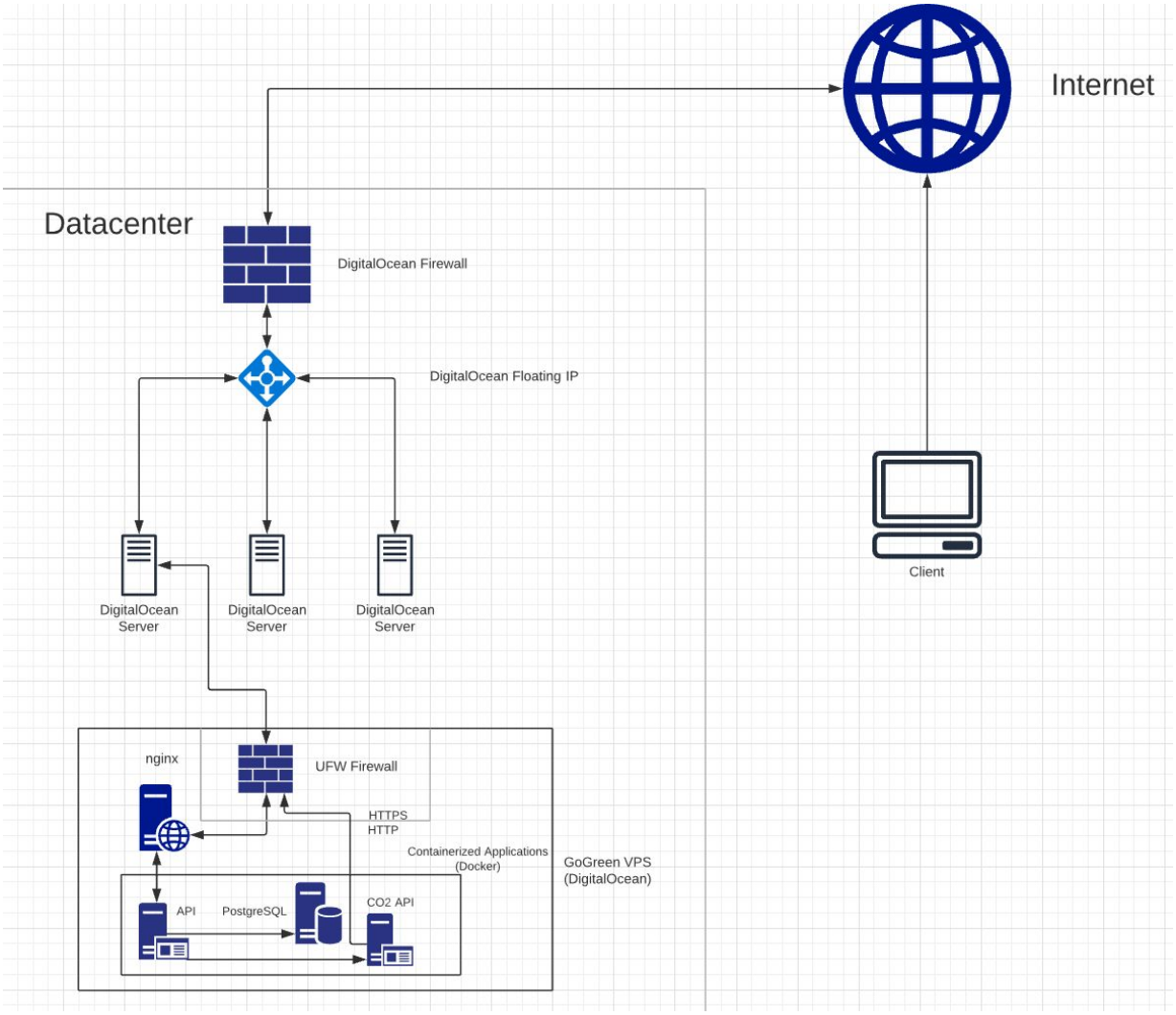
- spring-boot-starter-web
    - Enabled Tomcat
- spring-boot-devtools
    - A useful library for development
        - Live reload
        - Properties
        - Remote applications
- rest-assured (3.3.0)
    - Helps us testing our controllers
- spring-boot-starter-test
    - Helps us testing our controllers
- unirest-java (1.4.9)
    - Library to make calls to third-party APIs

GUI-specific dependencies:

- gogreen:client (LATEST)
    - A module we made which includes a cache, server callbacks and a way to communicate with the server
- jfoenix (8.0.8)
    - Adds materialistic design objects to our java GUI

Server-specific dependencies:

- spring-boot-starter-web
    - Enabled Tomcat
- spring-boot-devtools
    - A useful library for development
        - Live reload
        - Properties
        - Remote applications
- rest-assured (3.3.0)
    - Helps us testing our controllers
- spring-boot-starter-test
    - Helps us testing our controllers
- unirest-java (1.4.9)
    - Library to make calls to third-party APIs
- spring-boot-starter-data-jpa
    - Makes using the JPA in Spring a breeze
- spring-boot-starter-security
    - Adds spring security
- spring-security-test
    - Helps us tes spring security
- h2
    - Adds a driver to access a H2 database for development and testing
- postgresql (42.2.5)
    - Adds a driver to access a PostgreSQL database for the production environment
- gogreen:shared (LATEST)
    - Our shared module
- commons-validator (1.6)
    - Adds a way to easily validate strings
- hibernate-search-orm (5.10.5.Final)
    - Adds search engine
- aerogear-otp-java (1.0.0)
    - Generates OTPs for 2FA



Server stack diagram