DurtleNet

A smart eHealth system coded in Java

Group G21:
Serkan Atay (1207077)
Viktor Benini (1298976)
Amalie-Margarete Wilke (1304925)

Frankurt University of Applied Science
Fachbereich II - Informatik
Modul: Java Object Oriented Programming
Lecturer: Luigi La Blunda
WS 21/22

Abstract

This technical documentation outlines the developement and implementation of the smart E-health consulting system DurtleNet which allows its users to search for doctors in a individual radius based on their health needs and schedule appointments as well as shift or cancel them. The application is coded in Java and uses a MySQL database for data storage. GUI implementation is done with JavaFX. The geocoding data necessary for a radius based doctor search is pulled from Google Geocoding as a Json file and parsed to extract the necesssary data points. Email reminders are sent to users via Java Mail.
The entirety of our source code can be viewed using following link:
Link: GitHub- eHealth Project DurtleNet
URL: `https://github.com/anurasalientia/eHealth_DurtleNet`

Contents

Chapter 1

Introduction: Amalie Wilke

1.1 Project Description

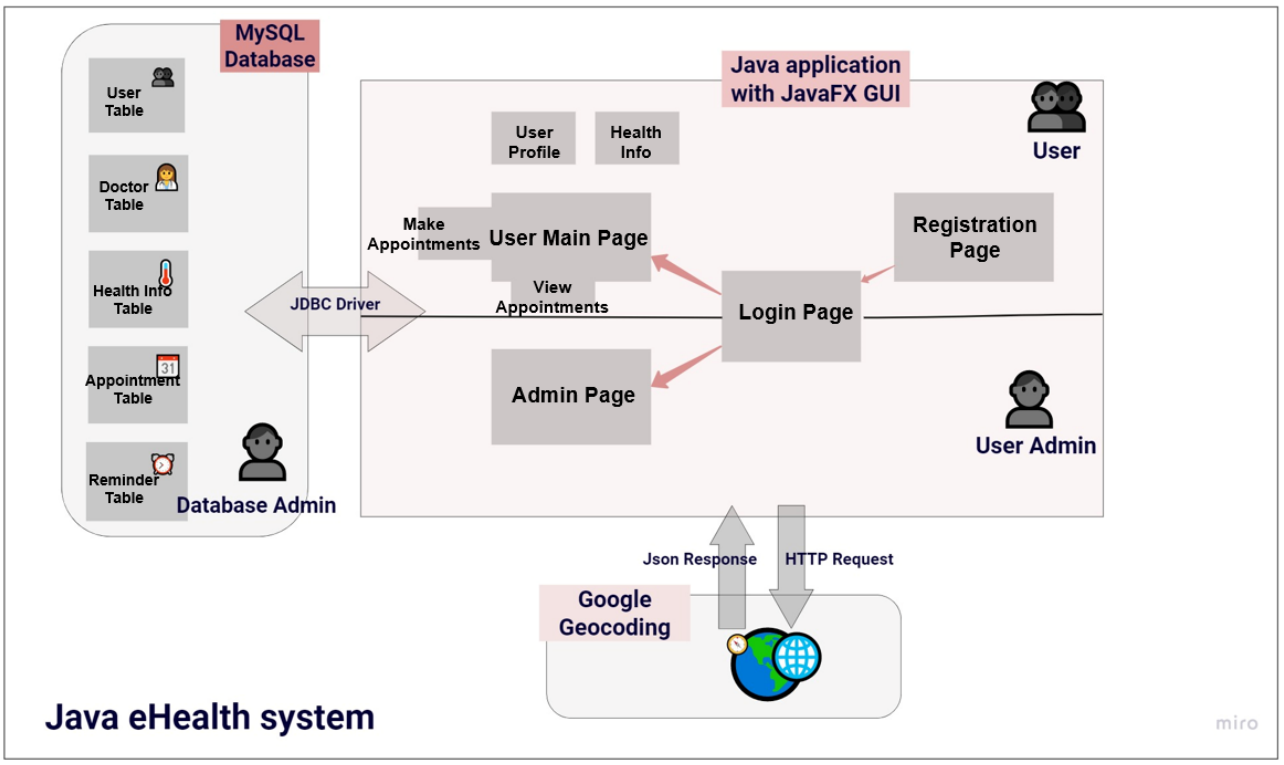

Figure 1.1: The eHealth system diagram

The smart E-health consulting system DurtleNet is a Java coded medical appointment scheduler connected to a MySQL database that stores user, doctor and appointment data. A user registers with the application with relevant personal and health information and uses an encrypted password and their email as unique user identifciation to log in to the application after this to schedule medical appointments with doctors based on their specified needs and desired radius. A registered user can view their profile as well as their main page on which they can schedule the appointments as well as view and edit them, cancelling or shifting them depending on their needs. Users will be reminded about their appointments via email and see appointments based on user selected time pending (ranging from one week to 1 hour). User registration and maintenance is supported by a user admin who has identifying credentials and can add, delete and edit users over a graphical admin page interface. There is also a database admin who works directly with the MySQL database and has full access to all user, doctor and appointment data and is the only one who can register user admins.

1.2 Project Motivation

During our third semester Java module class, we were asked to design a smart E-health consulting system using Java. The German medical system is still lagging behind in digital accessibility. While the trend in other countries has already been going towards quick digital appointment scheduling and communication, Germany is still far behind and most appointment scheduling with doctors is still done via phone or email. This is inefficient and demands time, German language skills as well as familiarity with the doctor being called. Our consulting system DurtleNet aims to make medical appointment scheduling accessible, efficient and comfortable for patients as well as doctors. Users can quickly search for doctors based on their medical and location needs and keep track of appointments and edit them easily. Doctor offices gain valuable time due to the digital appointment scheduling that frees up their front desks and efficiently schedules appointment slots. DurtleNet contributes to a modern, digital medical system and makes the German health care system approachable.

1.3 Project Challenges

JavaFX Integration Due to the GUI requirements we used JavaFX but it was highly difficult to integrate this smoothly into our overall project because class paths and dependencies refused to read properly when using jar files. It took us over two weeks of manual trying before we switched to Maven which streamlined dependency inclusion and restructured the entire Java project as a JavaFX project which in combination solved our issue.

MySQL Database local hosting We worked with a MySQL database which was easy and well connectable to our Java code via driver (JDBC) but resulted in issues how each team member would be accessing the same database path and information. In the end we each created a database independently but with the same file path, user name and password and used identical table names, values and information - using a Google Drive document for MySQL queries and table creation commands as well as cvs files for the data inside.

Java-MySQL variable value transference and reading It was not always an exact fit for variable values used and saved in our Java code to be transferred to the corresponding tables in our database. Especially enums and date formats were tricky and needed some adapting. We solved most of this by using strings for data in both our Java code and MySQL and created methods for transforming them into workable values based on needs.

GitHub pushing and pulling The shared repository system in GitHub with pushing and pulling our code worked well most of the time but we ran into three major glitches during pulling and comitting of code which each time deleted important code or shot the entire code structure. After the first time in the beginning stages of coding we began making independent backups and learned how to work with rollbacks to fix these issues.

Group Communication Due to our team member Serkan Atay not being available for any communication lines outside of WhatsApp, we worked with him over WhatsApp and GoogleDrive which was not always the most suitable and quickest way of discussing the work that had been done and needed doing. This resulted in miscommunication sometimes.

Chapter 2

Requirements: Amalie Wilke

2.1 Database

The application needs a well structured and adaptable database to store, access and edit user, doctor and appointment data. The database further needs to be connectable to the Java source code and reliably accessible and modifyable. A database table structure for user management, doctor management and appointment management as well as reminder and health information needs to be developed that ties data points (all appointments of a specific user, specific appointments, doctors connected to appointments) together tidily and logically. The database access needs to be protected as it is the vital information storage point of the system.

2.2 GUI

The application runs over a Graphical User Interface (GUI) for the user as well as the user admin (the database admin uses the MySQL provided database interface). The GUI needs to be intuitively usable, well sectioned and designed for user friendliness and enjoyable usage (calm color scheme due to medical nature of application, easy and decluttered interface). It should be self explainable without an introductional tutorial or user support while registering and using it. The GUI login will be used by both users and user admins.

2.3 Base Functionalities

2.3.1 User

The base functionalities of the application are the registration, login and logout of a user. First time registration asks the user for all information to be stored in the database. These are:
• Gender • First Name • Last Name • Date of birth • Health Information (Usual general practitioner, Pregnancy, Smoker, Allergies, Asthma, Medicine, family health issues)
• Adress Information • Insurance Type (private or public) • Insurance Name • Email • Password
Three non editable data columns will be autofilled in the database automatically when a new user registers, these are creation date of user entry, user ID as an automatic unique auto increment as well as a Boolean IsAdmin which automatically sets to 0. A method checks that the email entered is unique before allowing registration and if any of the needed text fields are not filled, there is a prompt. After registration the user is redirected to the login page where they can enter their email and password to login for the first time. The password will never be stored in the database, instead the password is encrypted and compared. After successful login the user lands on their main page. Here they can click to redirect to their profile to view or edit it or to the help page. From the main page there is also the possibility of switching to Health Information Page to view the user health information or edit it. The main page further has the option to view appointments or make a new appointment. An appointment can be made while specifying doctor type, date and time and radius of search based on user address

as well as specific reasons for making the appointments in a text field. In the appointment list, an appointment can be selected to shift or cancel. A user can log out of the main page to be redirected to the login page. Health Information is inputted by the user seperately after registration so that the registration process stays brief and approachable and the user later has more time for more detailed information such as insurance number etc. (probably not things one has "on hand").

2.3.2 User Admin

The user admin does not have the need (or ability) to register themselves, the database admin registers them directly in the database with the column IsAdmin as a Boolean 1. The database is the core of the application and as such is also sectioned from the user admin in certain parts such as doctor data and appointment data (this might potentially be expanded). A user admin has a unique email and password which they use to login from the GUI login page the user uses as well. Once successfully logged in they are directed to the admin page instead of the user main page though. There they can display a list of all registered users, click on a user to delet them (directly from the database), add a user to the database over the GUI as well as edit a user using the user ID as identifier. They can also logout of the admin page and are then redirected to the login page.

2.4 Expanded Functionalities

2.4.1 User

The expanded functionalities are also highly important but even without them the application would be usable. For the user these functionalities are a drop down reminder functionality for the appointment list that shows appointments based on 1 week, 3 days, 1 hour or 10 minutes until they will take place. The user can also select one or several boxes during the scheduling of an appointment to send them a reminder email for the appointment 1 week, 2 days, 1 hour or 10 minutes before it (it can also be chosen to omit all boxes/reminders). reminder will also be sent to the user via email. The user can also export their health information page as a PDF or text file to print or save.

2.4.2 User Admin

The user admin could be expanded to have control over user appointments and be able to cancel, make or shift them as well as view doctor information. There could also be an admin log that shows which admin edited what. These are extended functionalities that will not be able to be implemented due to time constraints but could be a useful extension that optimizes the system.

Chapter 3

Team Organization: Amalie Wilke

3.1 Organizational Aspects and Priority Determination

After we received the project requirements,the most pressing concerns were to create a working communication structure as well as a way to share code - the roles of project manager and technical manager respectively. Project management entailed the creation of a shared Google Drive Folder for correspondence, meeting protocols and detailed documentation of the different implementations as well as streamlining team communication, spearheading meetings and keeping a working overview of the different areas of coding and planning needed. Technical management needed to create a shared GitHub repository for the project and keep an eye on code integration and possible issues throughout the project duration. I (Amalie) took over project management and Viktor became technical management. Our third team member, Serkan Atay, informed us that he was unable to access a video feed or audio stream due to professional and personal reasons and that his only means of communication would be our WhatsApp group. Once the first questions about the project and its concept were out of the way, we sat down to the technical aspects of the application. Both Viktor and I were new to GUI implementations as well as integrated database work so we knew a larger portion of time would be used for research and restructuring of initial ideas and diagrams. To keep to our time plan and avoid over stressed crunch time shortly before the deadline, I sectioned the work into priority groups that already became clear while grouping requirements in the chapter before and are displayed in the table.

First Priority- needs to be finished first to build on	High Priority- builds on the first priority functionalities	Medium Priority- part of requirements but non vital to program functioning
Database	Registration Page	User Profile Editing
Java - DB connection	Appointment Table in DB	Appointment reminder, dropdown
GUI: Login, Logout and Main Page	Appointment Page with appointment making	Appointment reminder as email
Class Diagram	Admin Page with user editing, deletion and adding	Health Info as pdf/txt export
	Password Encryption - non negotiable but can be delayed	Smoothed out GUIs with icons and color scheme
	Radius search for doctors	
	Health Information Page with editing	

3.2 Task Distribution

The most important things were to get our DB up and running and have a solid application framework with working login, registration and admin functionalities in GUI form. This would test how good our DB connection (and the DB itself) were and also make the more specific functions such as password encryption, doctor searching and appointment viewing more developable. The table below details the final task distribution and the areas everyone worked on. Especially towards the end of the project, some of these became fluid, technical management had to be done cooperatively by Viktor and me due to some GitHub issues and Maven dependencies.

Amalie	Viktor	Serkan
Project Managment	Technical Management: GitHub, Maven	GUI design frontend
Technical Managment: MySQL DB, Maven		
eHealth Database	Code Skeleton	Class Diagram
DB Java Connectivity		Flow Diagram
User Profile Page+Functionalities	User Mainpage	Sequence Diagram
Login Page+Functionalities	Appointment View + Functionalities	
Registration Page + Functionalities	Encryption	
Admin Page + Functionalities		
Technical documentation - LaTex draft	JavaDoc	
Radius Search		
Email Reminder	Scheduler and Email Reminder	
Dropdown Option for appointments in certain time ranges (appointment view		
GUI - overall design and color scheme, image integration, overall integration/scene switching	GUI - image integration	

3.3 Application Timeline - developement stages

The Gantt chart below shows our final workflow and how the different sections of the application were distributed and worked on. Our active coding time for the application kicked off on the 18th of December 2021 and our hard deadline for a working and presentable application was the 13th of February 2022, in line with the handing in of the written documentation.
Database Deadline: 30.12.21
First Priority: 16.1.22
High Priroity: 26.1.22
Medium Priority: 15.2.22

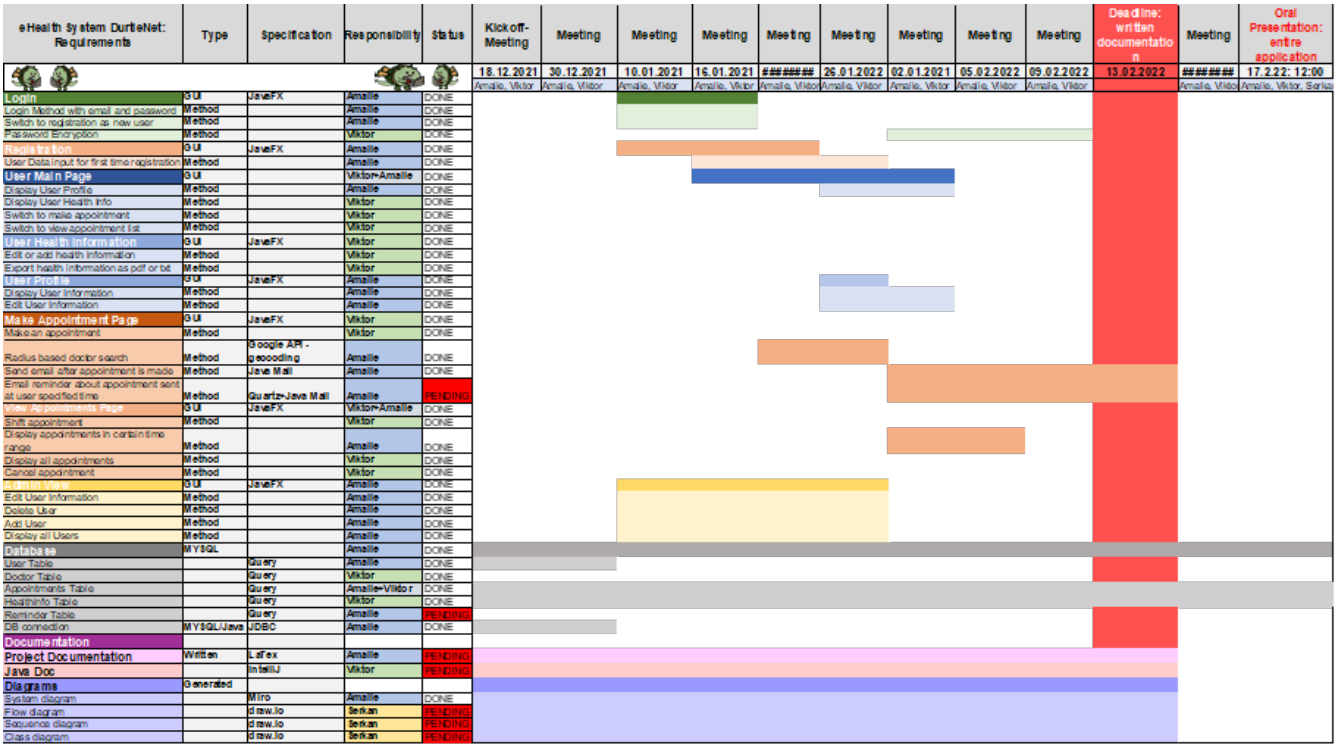

Figure 3.1: Gantt chart: eHealth System DurtleNet

Chapter 4

Technical description of solutions

4.1 Diagrams: Serkan Atay

Unfortunately, due to work (primarily Wednesdays and Fridays 12 hours-shift for each of those workdays) and private reasons in this current situation with the switch to the online mode and the change of plans regarding the dates of the exercise groups and changing of the days, I am not able to participate in a verbal manner consistently and spontaneously. My solution was to update each other when something important had to be discussed or to clarify things via the abovementioned methods. I used the three general objects (GUI, Server and Database) due to time reasons and to showcase it in a more general manner rather than using every object in a class which would make the diagram a lot more complicated and would not be fitting.

Class Diagram (Figure 4.1, p. 12 This diagram takes the most time out of all the diagram due to multiple changes and updates it must be maintained as such.

Sequence Diagram (Figure 4.2, p. 12 This diagram shows the process from the perspective of the User and the admin. Certain moves are made that create and update certain Backend functionalities like the Database when creating and editing a profile. By making certain decisions the process is continued in the therefore destined direction to complete the whole process.

Flow Diagram (Figure 4.3, p. 12 Sequence diagram displays the different objects (meaning can be interpreted as either the class files or more general GUI, Server, and Databases). It shows the relation and communication between those objects and the time in which the objects are active (lifeline boxes).

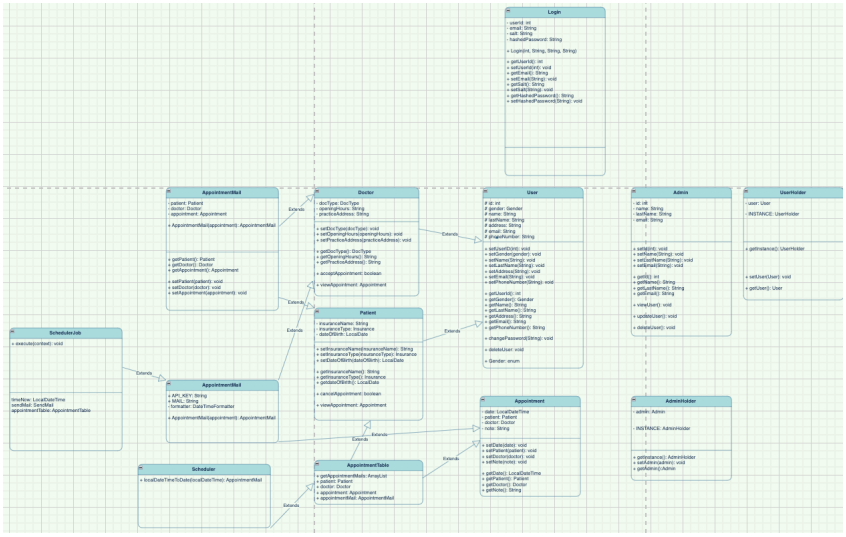

Figure 4.1: Class diagram - eHealth System

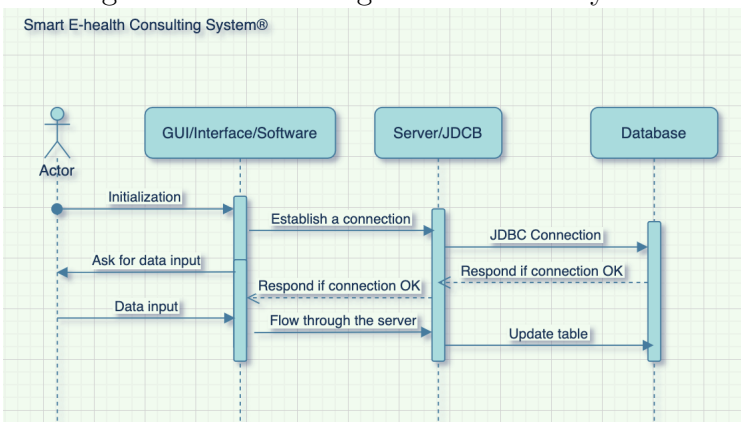

Figure 4.2: Sequence diagram - eHealth System

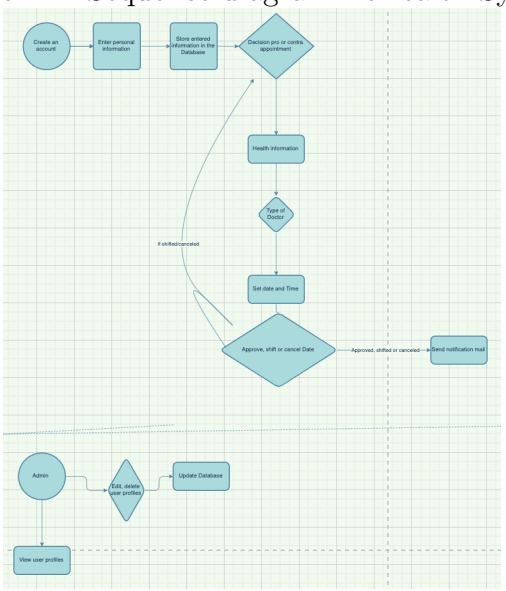

Figure 4.3: Flow diagram - eHealth System

4.2 Build Tool GitHub and Maven: Amalie Wilke

To store, integrate and track our seperate coding areas we chose the repository hosting system GitHub where a private repository held our source code as well as any additional files with the three of us as collaborators. Each of us had an instance of the source code on our IDE of choice (Viktor and I: IntelliJ, Serkan: Visual Studio) and depending on our progress we pulled, pushed or updated using the repository. Through the repository and the merging and branching of our different code we kept the main source protected and could track all changes and see who contributed what. GitHub was an efficient and secure service for collaborative coding but there were moments of accidental pushing or comitting of code that erased code. While GitHub solved the problem of collaborative coding efficiently, it did not solve the problem of the different dependencies and libraries we needed to run different areas of our application such as the GUI (JavaFX) or the MySQL connection (JDBC). For this we used Maven.
One of the biggest hold ups we experienced while coding the application was in the beginning, when we were still unfamiliar with the dependencies and libraries needed for our GUI. Especially JavaFX proved difficult to integrate using an indidivual approach of adding class paths to JAR files by hand in our IDEs and halted our initially well going indidvual progress. This was when Maven entered the picture. Only after we switched our project to a Maven project and began integrating dependencies using its project object model (POM) did our project compile across our individual IDEs. It took most of the stress of adding class paths and sped up work.

4.3 Database: Amalie Wilke

Due to previous familiarity and everyone having a Databases module with MySQL in their third semester, I chose to implement a relational MySQL Database which would be connected to the application via local host on each team member´s device. To connect the database and the source code itself I used the Java API JDBC[1] (Java Database Connectivity), which manages connecting to a database, querying and commands as well as receiving queried results from the database (as result sets). Connection to the database happened over the DriverManager where the MySQL path as well as database name, user and database password where passed. Once successfully connected (and error handled by a try catch clause), querying could be done with preparedStatements that directly translated MySQL queries and either received (executeQuery) or sent (updateQuery) data to and from the database. All database tables (user, doctor, health info, reminder and appointment) were created in MySQL workbench but user, health info and appointment were directly modifyable through the Java source code, doctor and reminder were usable but not modifyable using the source code (but of course changable in MySQL).

4.4 Login and Registration: Amalie Wilke

Login is the first GUI a user (and user admin) will encounter and is simple in its functions at first glance. There is the possibility to login using an email and a password or clicking on the "New User? Register here" button to be forwarded to the registration page. Behind the scenes there are imporant methods needed to validate login or registration though. As unique identifier for the user I chose their email address, creating a method to disallow a new user to register with an email already in the database. Further, if email or password are missing login

[1]Tyson, M.(April 11, 2019) What is JDBC? Introduction to Java Database Connectivity.Info World. Retrieved from: https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html (10.2.22)

is not possible. The login is also where user and user admin part ways. A user admin is stored in the same user table as a user but they have a Boolean value 1 in the IsAdmin row and a method checks the value on each login and directs accordingly. If the method validating the entered password returns a match, login is successful and redirection to the main page (user) or admin page (admin user) follows.

4.5 Appointment Scheduling: Viktor Benini

To store an appointment created by a user, we had to create a new table in our database. The important information that had to be included were the unique id of the appointment as well as the user id, doctor id, date and time of the appointment, the special note for the doctor and the created time of the appointment, for the reminder scheduler. The user and doctor id are foreign keys. This allows us to inner join the tables to get all information we need for the appointment, by simply storing the ids. By this method we don't have to add all the information to the table. This is used to show all details of an appointment by gaining, for example all the information of a doctor, to display them in the detailed appointment view.

Making the appointment By creating an appointment the doctor is selected by searching in the database for the input of the user and store his id. Date and time along with the note for the doctor are entered separately by the user. The data is than parsed into the database as a new appointment and gets it own unique id.

View the appointment The user has the ability to look up all of his appointments matching his UserID in the Appointment view that can be accessed on the main page. He can delete one appointment at a time. The appointment id will specify the selected appointment, to delete the appointment out of the database, to ensure only one appointment will be deleted. The user also can update his appointment which works in the same way by selecting an appointment and the selected appointment will be updated by its id. If the user wants to see all information about his appointment. An inner join is used to select the information of the user and the doctor, selected by their id, along the appointment data.

Cancel and Update We decided that the user only has the opportunity to change the date and time of the appointment, because changing the doctor is related to more changes and complications than creating a new appointment. But he has the option to cancel his appointment, if he doesn't wants to go to the selected doctor or the problems solved themselves.

4.6 User Profile and Health Information: Amalie Wilke, Viktor Benini

We decided to split user profile and health information into two parts (just as during the registration process), to not overwhelm the user with too much information in one go and seperate personal and (personal) medical information cleanly. The user profile displays all database columns of the user table save for creation date, IsAdmin and the User ID which are all only for internal purposes. A button redirects to an editing page if the user wishes to change their profile information. Here all information displayed in the profile can be edited. The health information can be set by the user after the login. He can specify his weight, height, Social Security Number, Medical Record Number, Health Insurance Number, Medicine use and Allergies. This information is saved in the database and outputted in a pdf file, which the user

can save on his device wherever he wants by selecting the path. The pdf will be sent to the doctor automatically after making an appointment.

4.7 Email reminder: Viktor Benini

The JavaMailTM API provides classes that model a mail system. The javax.mail package defines classes that are common to all mail systems. The javax.mail.internet package defines classes that are specific to mail systems based on internet standards such as MIME, SMTP, POP3, and IMAP.(defines Email standards)

The Message Class The Message class is an abstract class that defines a set of attributes and a content for a mail message. Attributes of the Message class specify addressing information and define the structure of the content, including the content type. The Message class adds From, To, Subject, Reply-To, and other attributes necessary for message routing via a message transport system.

The Session Class The Session class defines global and per-user mail-related properties that define the interface between a mail-enabled client and the network. JavaMail system components use the Session object to set and get specific properties. The Session class also provides a default authenticated session object that desktop applications can share. The Session class is a final concrete class.

4.8 Admin View: Amalie Wilke

An admin can use the admin view after a successful login to display all users currently registered in the database (in a list view). They can add a new user (bypassing the registration step) - but in the creation context a entered email will always be checked for uniqueness -, edit a user using their User ID (displayed in the list) and click on a displayed user to delete them from the database.

4.9 Code Structure -the Singleton Pattern

User Holder: Viktor Benini The Userholder is a singleton class that stores a user. The special on a singleton class is that the constructor cannot be accessed from outside the class. That means it is impossible to create a new instance of the class. We used this opportunity to store our user in the Userholder. By this we have access to the stored user at any time and point in the program. To get the correct user we set the user after the successful login, to ensure the correct user can be selected during the program.

Doctor Holder: Amalie Wilke Using JavaFX and scene switching as well as a database connection can result in difficulties transporting values needed from one scene to another. Here the Singleton pattern helped us creating a global access point for it with a one instance class[2]. We implemented the pattern for our User Holder as well as our Doctor Holder which was important for data being retrievable even after scene switches (for example from main page to appointment view page).

[2]Creational Patterns- Singleton. Refactoring.Guru. Retrieved from: https://refactoring.guru/design-patterns/singleton (10.02.22)

4.10 Graphical User Interfaces

4.10.1 GUI - backend: Viktor Benini, Amalie Wilke

The GUIs (Graphical User Interfaces) are created via an application called Scene Builder. The Scene Builder creates an xml file which contains the information about the look of the page and the method names that are called by a specific action when the xml is "executed" by the program. For the internal usage of the xml files, we decided to use JavaFX, because it is a modern and easy to implement API for GUIs. It allows us to create a controller to specify the previous, in Scene Builder, assigned methods. We can open a window in only a view line of code. But we noticed shortly that we need to switch between the same scene's multiple times. To avoid the rewriting of the same method repeatedly, we decided to implement a method that holds all of these switches, and we only have to call the needed switch method to save time and have a better overview. We also noticed after some time that we have the opportunity to call methods of a controller before opening the scene, to fill in images or tables. So, we added these to the scene controller too. In the beginning stages our teammate Serkan Atay worked on the frontend design of our GUIs and decided on a black color scheme. However, after a discussion over WhatsApp about how black might seem in a health care and medical context (perhaps to be perceived as a more sombre and disconcerting color) he handed over the GUI design to the other team members to complete so he could focus on the diagrams as we were about one and a half weeks to handin.

Figure 4.4: The DurtleNet mascot Durtle

4.10.2 GUI - frontend design prototype: Serkan Atay

I created a little prototype of the GUI by using the Swing Framework because it was also shown in the slides of the lectures. Implementation of DB and other classes and packages should be made onwards. Later the team decided to use the JavaFX Framework so I started to dig into it to get familiar with it so I can start implementing again from the scratch. The team initially decided to make me do the design aka front end part of these three Scenes with the following file names: AdminTableview.fxml, login.fxml and registration.fxml. But after some questions on my part it suddenly changed and AdminTableview.fxml was replaced with the following Scene: UserDBTableview.fxml. So, I had to make this one from the scratch again. For the layout and overall design my initial thought was to leave it simple with a simplistic design due to the limitations of the changes in the code that I should leave out according to the team. I decided to let the UserDBtableview.fxml as simple as possible due to the importance of functionality. A further adjustments and coloring would distract the user/admin unnecessarily from the primary work and would make things more complicated and not as efficient as it should be. At the end

the team decided to take over the designs of the Interfaces. The prototypes I designed can be seen below.

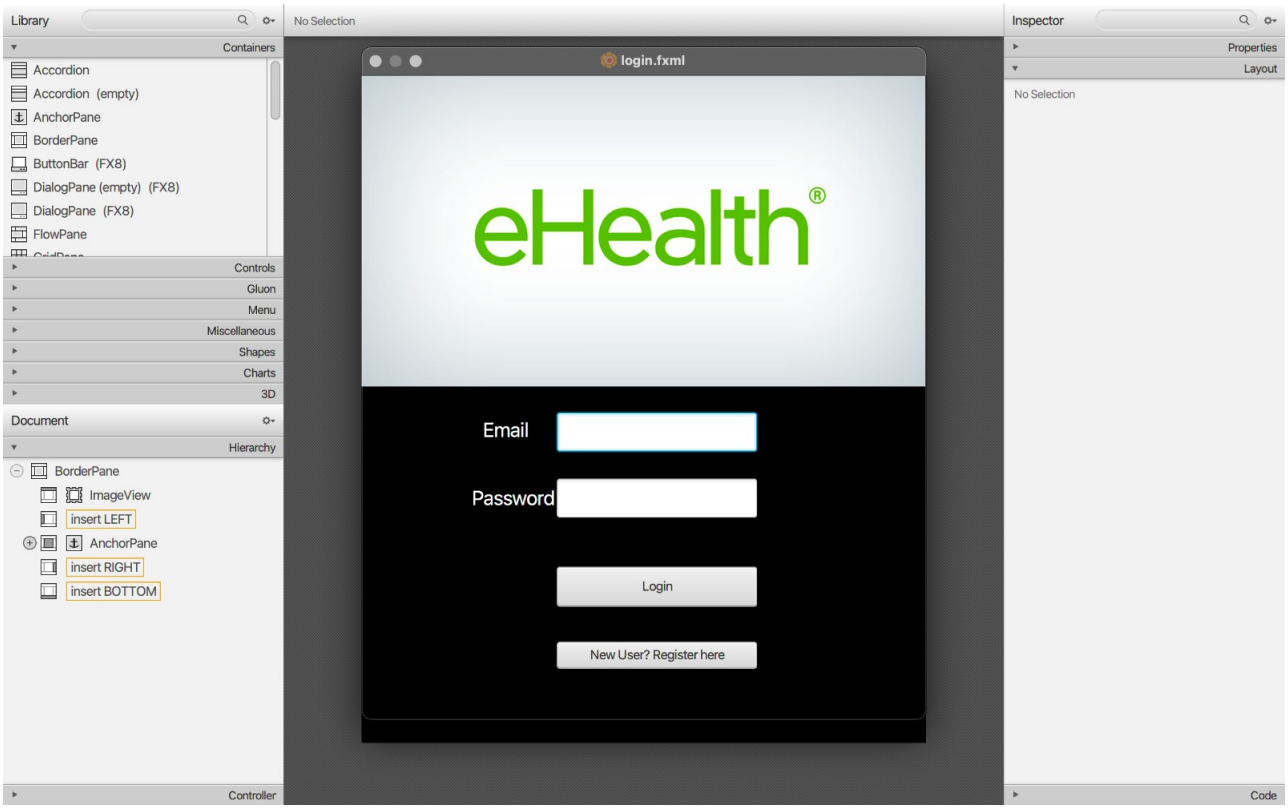

Figure 4.5: Login GUI prototype - Serkan Atay

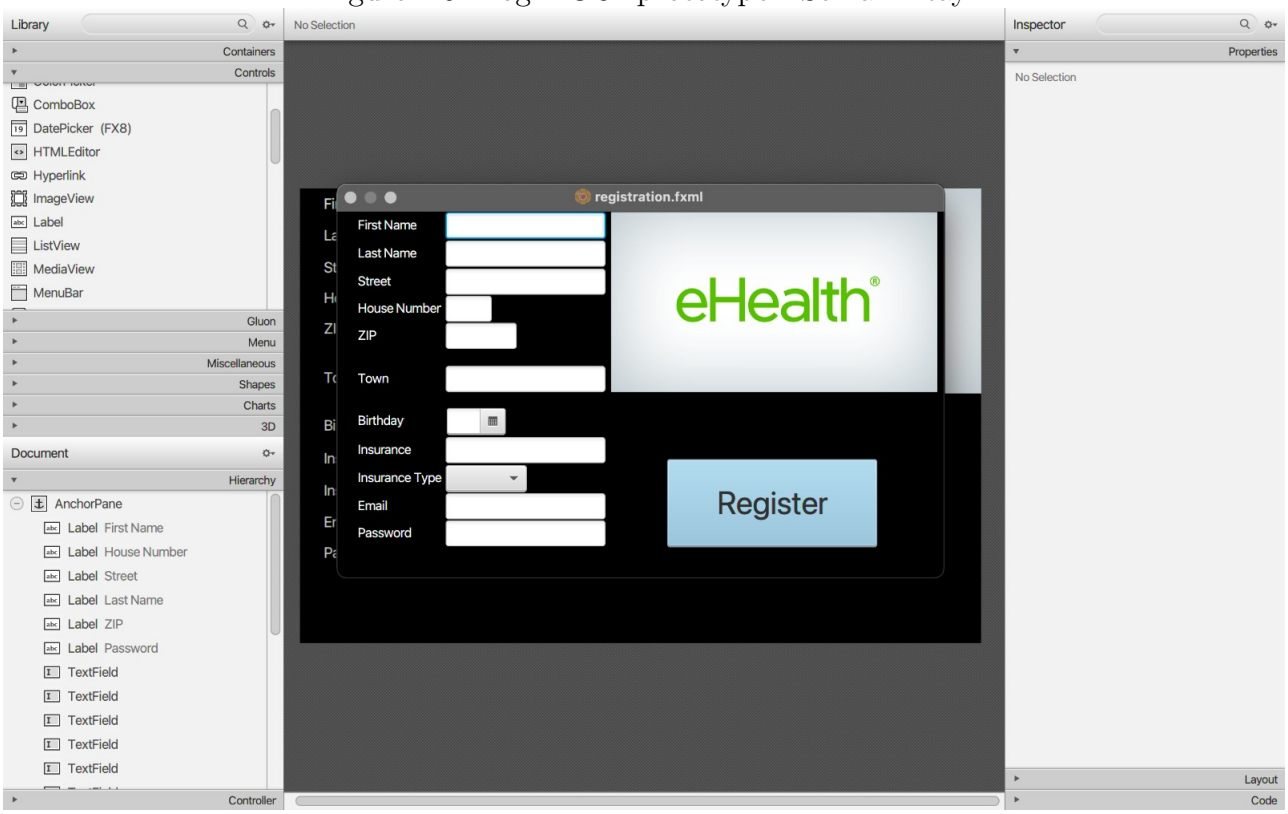

Figure 4.6: Registration GUI prototype - Serkan Atay

4.10.3 GUI - frontend, final implementation, Amalie Wilke

For the frontend design of our eHealth application, we went with a muted green-blue color scheme that suited our mascot Durtle and also conveyed a calming, sophisticated feeling. All designs were created using SceneBuilder and JavaFX, with more time we would have integrated css style sheets as well to adapt tables and menu bars as well as stage frames but due to scheduling we had to shelve this. The image shows our final GUI design:

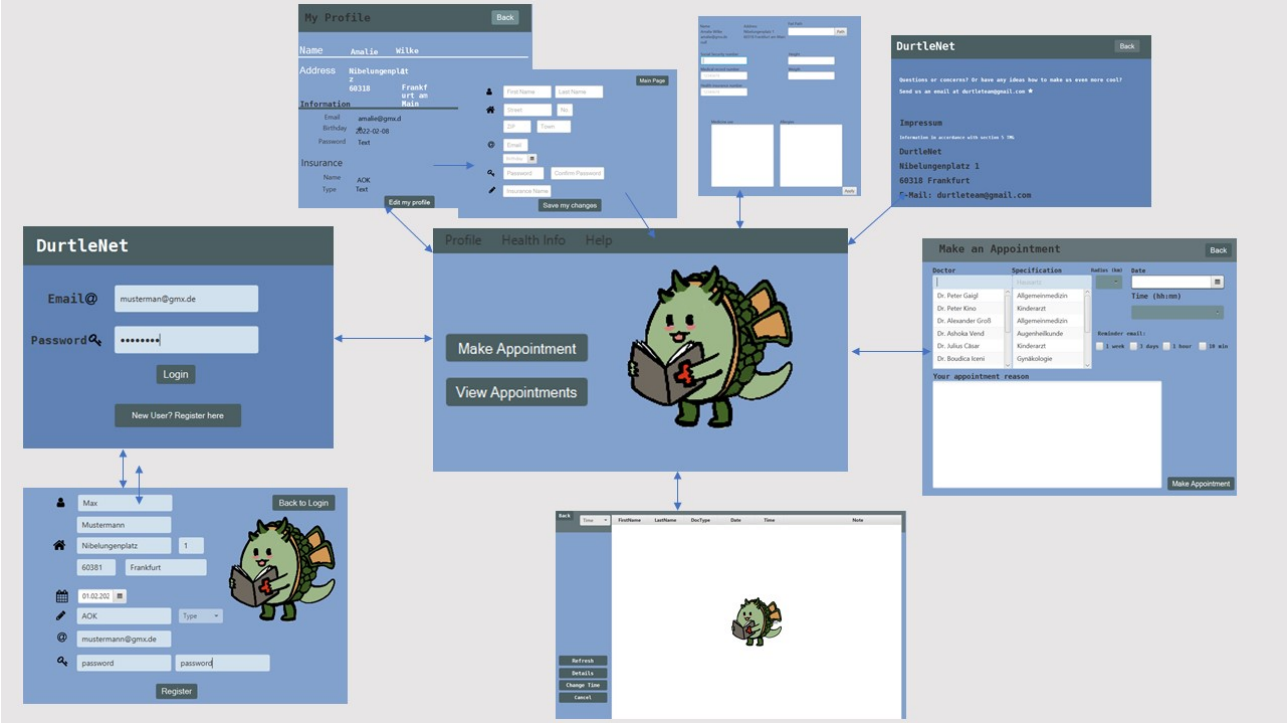

Figure 4.7: DurtleNet Frontend final design

4.11 Password Encryption: Viktor Benini

Our users have profiles, including their personal information. To ensure that this information is protected and cannot be accessed by any person fetching the passwords out of the database, we must encrypt their passwords. We used the java security API, because it has a decent number of security classes and methods. The MessageDigest method was the exact thing we were looking for.[3] It offers us the opportunity to select the hash algorithm we want. From the outdated MD2 to one of the most secure SHA-512. For the encryption we have decided to use a hash function designed by the NSA called SHA-256 (Secure Hash Algorithm). It is secure enough to not be decrypted easily and is fast at the same time. The algorithm converts the inputted data into a bit chain and extends it, if necessary, to a total length of 512 bits. After that, 64 bits will be appended to describe the original length of the data in bits. Then our data is split into blocks of even length. The blocks get calculated with some constant values like the square root of the 8 first primes. After each block is done with its calculation they are reassembled in the order they were split. This hash will be stored in our database instead of the original password. The advantage of this technic is that the stored hash value is useless for everyone, because it is impossible to restore the original password from the hash. But if the

[3]Wagner, L. (July 8, 2020) How SHA-256 works Step-by-Step. *Qvault.Io.* Retrieved from: https://qvault.io/cryptography/how-sha-2-works-step-by-step-sha-256 (13.2.22)

user enters his password the hash algorithm replicates the hash, and we can compare it to the stored value in the database.

4.12 Doctor Radius Search using geocoding: Amalie Wilke

Most sources online seemed satisfied using the spherical law of cosines[4] to determine whether a latitude-longitude determined point on earth lay in or on a radius and my own tests with addresses showed that it was accurate enough for my purposes of finding points within 1 and up kilometers range (for more detailed searching down to meters there might have been a worse error margin) of user specification. For the formula to be applicable I needed latitude and longitude of both my radius center as well as my points to be found. I decided to calculate and save latitude and longitude of each user address directly in the DB as DECIMAL (float in Java) with digits depending on accuracy needed parallel to them registering for the first time (the doctor coordinates were saved via database and calculated with Google Maps). For the data I registered with Google Cloud and used their Google Geocoding API via HTTPS request and Json response.

Google Geocoding data via personalized URL and HTTPS request Google Cloud has a vaste array of services and one of their most popular is their Google Maps API which in a subsection includes Google Geocoding which returns information on a normalized (e.g. format used by the national postal service of the country[5]) address including longitude and latitude. The Geocoding API accepts HTTPS requests using a formatted URL which must include the sender´s API key as well as the specified return format of the request (Json or XML).

```
https://maps.googleapis.com/maps/api/geocode
/json?place_id=ChIJd8BlQ2BZwokRAFUEcm_qrcA
&amp;key=YOUR_API_KEY
```

The API key is tied directly to the sender´s Google Cloud and keeps tracks of usage (and eventuel costs). Addresses may not have spaces so I coded a small snipet to transform all address strings by concatanating them before passing them as strings to my adapted URL so it would accept variables (%20 indicates seperation of the different variables to Google in line of a space):

```
      String s=street; String t=town; String k="My API key here";
      int hn=housenumber; int z=zip;
URL url=new URL("https://maps.googleapis.com/
maps/api/geocode/json?address=" +s+"%20"+hn+"%20"++"%20"+t+"&key="+k);
```

Code snippet to concatenate my strings before passing to the URL:

```
String str_nospaces=str.replaceAll("\\s","");
String to_nospaces=to.replaceAll("\\s","");
```

[4]Calculate distance, bearing and more between Latitude/Longitude points.(2022) Movable-type. Retrieved from: http://www.movable-type.co.uk/scripts/latlong.html#ellipsoid (10.2.22)

[5]Geocoding API requests and responses.(2022). Google. Retrieved from: https://developers.google.com/maps/documentation/geocoding/requests-geocoding (10.2.22)

Google response as Json file Google returns a Json response to a correct request which I read as a string and then parsed via a method (my own parsing method due to only certain data being needed) to extract the relevant latitude and longitude. While Json is platform independent and commonly used for online exchange of data, it needs some intricacy to extract data into a workable Java format. Json files have only two data structures, objects and arrays but the overall structure of a Json file needs to be understood to extract the values inside them as often several objects and arrays can be inside an overall array.

Saving latitude and longitude with accurate precision Latitude and longitude are large decimal values that can range from -90 to +90 degrees for latitude and -180 to +180 degrees for longitude but to keep storage space free in the database and along with Google Maps recommendations I chose to keep only 6 didigts of precision after the decimal and save them as DECIMAL(10, 8) and DECIMAL(11, 8).[6] While MySQL warned me that values were being trunctuated when being saved from my Java float variables to DECIMAL, it still gave accurate enough calculations and so I kept it for the broader radius calculations I was doing.

Calculating radius using the spherical law of cosine To determine whether or not the doctor latitude longitude coordinates were within the user specified radius centered around their address I adapted the spherical law of cosine. R in the below equation stands for the Earth´s radius which as mean radius in kilometers is 6,371 km.

$$distance = acos(sin\phi 1 \cdot sin\phi 2 + cos\phi 1 \cdot cos\phi 2 \cdot cos\Delta\lambda) \cdot R$$

MySQL allows more complex mathematical operations, so I queried the doctors using the above law in an adapted way, using the JDBC driver and a prepared statement. If the distance calculated was smaller or equal to the selected radius of the user, the doctors were within the radius.

```
PreparedStatement preparedStatement = connection.prepareStatement
("select * from doctors WHERE acos(sin(Latitude * 0.0175) *
sin( UserLatitude* 0.0175) * cos(Latitude * 0.0175) *
cos( UserLatitude* 0.0175) *cos(( UserLongitude* 0.0175) -
(Longitude * 0.0175)*6371<=radius);");
```

The above UserLongitude and UserLongitude as well as radius were set using preparedStatement.setValue with the paramter index and ? directly in the source code but were placed in above code snippet to allow for better readability.

[6] https://dba.stackexchange.com/questions/107089/decimal-or-point-data-type-for-storing-geo-location-data-in-mysql

Chapter 5

Conclusion: Viktor Benini, Amalie Wilke

Our first moment of pride as a team was felt as we ran through a first test of the application on the last day of January, an internal hard deadline to get as much as possible done before the stress of the exams started. The application worked - with some bugs and oversights here and there and some aha-moments of "we need to fix that" - and as we scrolled through our source code over Zoom for error fixing we realized that we had amassed over 2000 lines of code, give or take, the most any of us had ever coded. It was an interesting movement very far removed from the coding we had done before this for our university classes, more focused on concepts and algorithm structures meant to convey the basics of coding and logical thinking. But for the first time we had combined previous module knowledge into one, from database concepts to the to us completely unfamiliar world of GUI design. It was a rewarding feeling after a month of hard work and while our testing showed a long list of bugs and to-dos, we had a running application. There were things to work on but there was also confidence and pride that we had created and kept to our time schedule. Now, with the application ready for presentation, there is still pride but also the realisation that there is much more potential within our code and our initial concept. In our motivation we already touched on the benefits of a digital way to schedule medical appointments, the barriers it would tear down and the streamlining and increased accessiblity it would introduce to health care. DurtleNet needs more shaping here, from replacing of text based doctor searching to perhaps icon and audio usage here for disabled users as well as a doctor GUI to extend our users to doctors who can also shift, edit and cancel or create appointments and manage their own information. Security is also something we would like to extend on for our database and source code to make sure the sensitive health care data there is protected. We are aware this makes our application sound grander and more experienced than it is, DurtleNet is the result of a team only just starting software engineering but it was an immense learning experience which propelled all of us forward. For that we are grateful.

Figure 5.1: A resting Durtle

Chapter 6

Bibliography

[1] Tyson, M. (April 11, 2019). What is JDBC? Introduction to Java Database Connectivity. *Info World* Retrieved from: //www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html (10.2.22).

[2] Creational Patterns - Singleton.(2022) *Refactoring.Guru* Retrieved from: https://refactoring.guru/design-patterns/singleton (10.2.22).

[3] Calculate distance, bearing and more between Latitude/Longitude points.(2022) *Movable-Type* Retrieved from:http://www.movable-type.co.uk/scripts/latlong.html (10.2.22).

[4] Geocoding API requests and responses.(2022) *Google* Retrieved from: https://developers.google.com/maps/documentation/geocoding/requests-geocoding (10.2.22).

[5] Decimal or Point Data Type for storing Geo location data in MySQL. (July 15, 2015) *DBA StackExchange* Retrieved from: https://dba.stackexchange.com/questions/107089/decimal-or-point-data-type-for-storing-geo-location-data-in-mysql (10.2.22).

[6] Wagner, L. (July 8, 2020). How SHA-256 works Step-by-Step. *Qvault.io* Retrieved from: https://qvault.io/cryptography/how-sha-2-works-step-by-step-sha-256 (13.2.22).

Chapter 7

Declaration of Authorship

Hereby, we declare that we have composed the presented paper independently on our own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such

February 13, 2022
Serkan Atay
Viktor Benini
Amalie-Margarete Wilke

List of Figures