FRANKFURT UNIVERSITY OF APPLIED SCIENCES

# Smart E-health Consulting System

*Project Documentation*

Faculty 2: Computer Science
Course: Java – Object Oriented Programming
Lecturer: Luigi La Blunda
Submitted: 11. February 2022

**Prepared by**

**Group 18:**
*Nabil Akir*
*Jan-Louis Schneider*
*Hoang Nguyen Duc*
*Ibtehal Al-Omari*
*Petrenco Veaceslav*

**Source Code**

# AUTHOR'S DECLARATION

We, the undersigned, hereby declare that this submission is entirely our own work, in our own words, and that all sources used in researching it are fully acknowledged and all quotations properly identified. It has not been submitted, in whole or in part, by us or another person, for the purpose of obtaining any other credit / grade. We understand the ethical implications of our research, and this work meets the requirements of the Faculty of Applied of Sciences, Faculty Computer Science and Engineering, Project Java „Smart E-health Consulting System", Professor La Blunda.

**Student Name:** Nabil Akir
**Student Number:** 1347636

Signature

[illegible]

**Student Name:** Jan-Louis Schneider
**Student Number:** 1345256

Signature

[illegible]

**Student Name:** Hoang Nguyen Duc
**Student Number:** 1342589

Signature

[illegible]

**Student Name:** Ibtehal Al-Omari
**Student Number:** 1323139

Signature

Ibtehal A.

**Student Name:** Petrenco Veaceslav
**Student Number:** 1309108

Signature

Petrenco

# TABLE OF CONTENTS

# 1. Introduction

## 1.1 Description

The goal of this project is to develop a Graphical User Interface called E-Health Consulting System. The system is made for people who need to go to the doctor, before that they must plan the dates into system, and so can the doctor organize his day to consult more patients.
The system will be able to synchronize all appointments and keep the patient and the doctor up to date.

The app will have a modest and intuitive graphical user interface, that will make all functions accessible and friendly for users.

To be able to use this app you must create an account, inputting your name, surname, telephone number etc., so you can create, delete, or replan your appointment(s).
Finally, if E-Health system crashes or has errors, you can export your health information in PDF format on your device.

## 1.2 Motivation

From the first day of our life, we are in the hands of a doctor. Very often the doctors become one important person in our live if we want or not. To visit a doctor in our days it becomes more complicated because the number of sick people has been increased.
We as a team, we want to help patients and doctors to be more flexible and organized during the illness or by simply visiting the doctor. All information about the replanning, shifting, or deleting an appointment will be accessible for patients and for doctors without calling or contacting by post. All this information you will be able find in our App. With the possibility of an automatic reminder via email about changes and add-ons, you don't have to worry about that because you will be immediately informed and always up to date.

## 1.3 Requirements

The requirements are very important for the success of a software project, they provide a clear picture of the work that needs to be done. When the requirements are not respected, the project can be in the end very expensive and can lead to fail.

Our project, called *"Smart E-health Consulting System"*, fulfils all requirements that were expected.

The GUI allows the registration of many users with the required information and then to login with username and password. All data are stored in database whereby the passwords will be hashed following modern standards. The admin can access all user profiles, edit or delete them.

The user can make an appointment for specialized medical doctor based on the health problem that he/she is having. For creating an appointment, the patient must give the following information: first and last name, address, health information, insurance type, insurance name, health problem and distance. To select a reminder for: 1 week, 3 days, 1 hour, 10 minutes and then the reminder will send the patient an email.

The patient can cancel or shift the appointment, after that the patient will receive an email as confirmation with the changes that were made.

Finally, the patient can export the health information as PDF.

## 1.4 Task Distribution

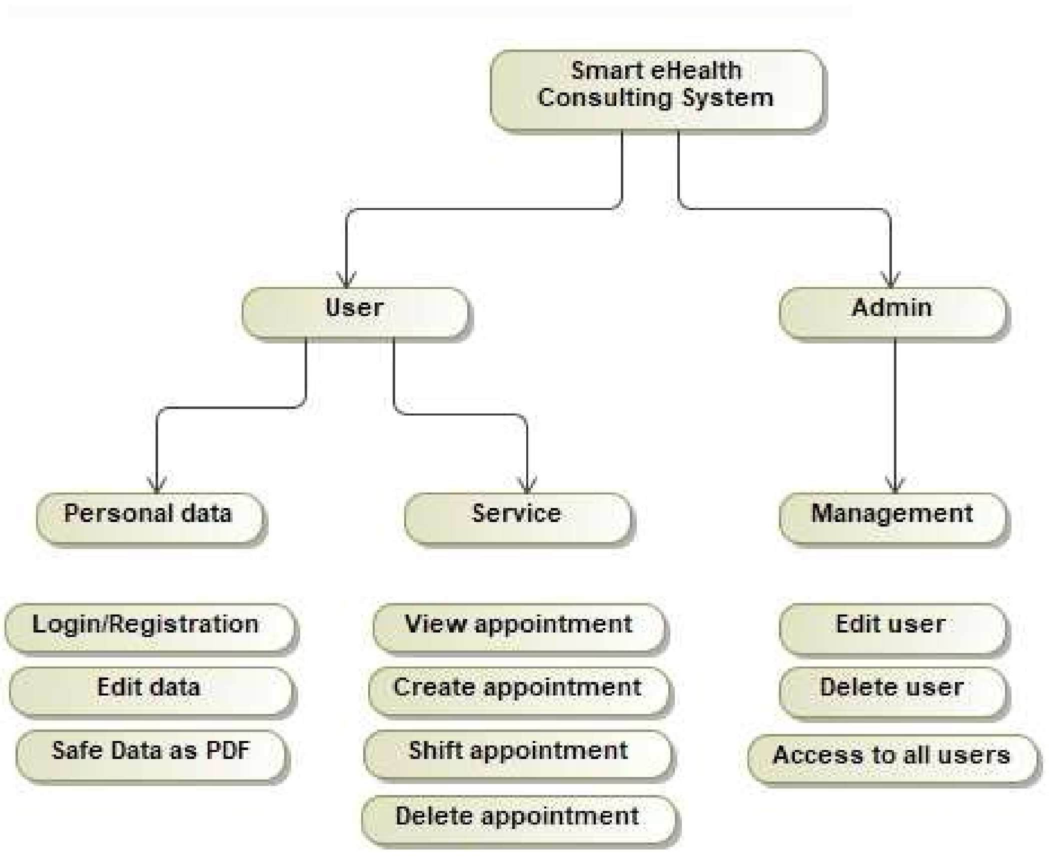
Because the today's situation does not allow us to meet in person, we decided to hold one meeting per week to discuss what was done, if someone needs help and what must be done in the next week. We didn't use other Platform to organize our work, but we had a leader *Nabil Akir*, who was responsible to check every member if weekly work was done.

We also divided the whole project into several parts, so that each team member was responsible for one or more aspects of the application. *Nabil Akir* was responsible for the GUI-Design, Login(Authentication), Dashboard, to export the information as PDF, Doctor suggestions and the database storage/encrypt login in database. *Jan-Louis Schneider* did the display user data, edit user data panel and Admin permission to access, edit/delete users.

*Hoang Nguyen Duc* was responsible for the creation of appointments, entry contains information about event, store appointment entries in database, prepare to display stored meetings on dashboards. *Ibtehal Al-Omari* did the reminder functionality which include drop down menu to select time, and reminder notification by email. The part for *Petrenco Veaceslav* was to delete appointments, to edit them and that the user gets email notification on deleted or edited appointments. All team members took part in preparing the documentation.

# 2. Technical Descriptions and Solutions

## 2.1 System Architecture

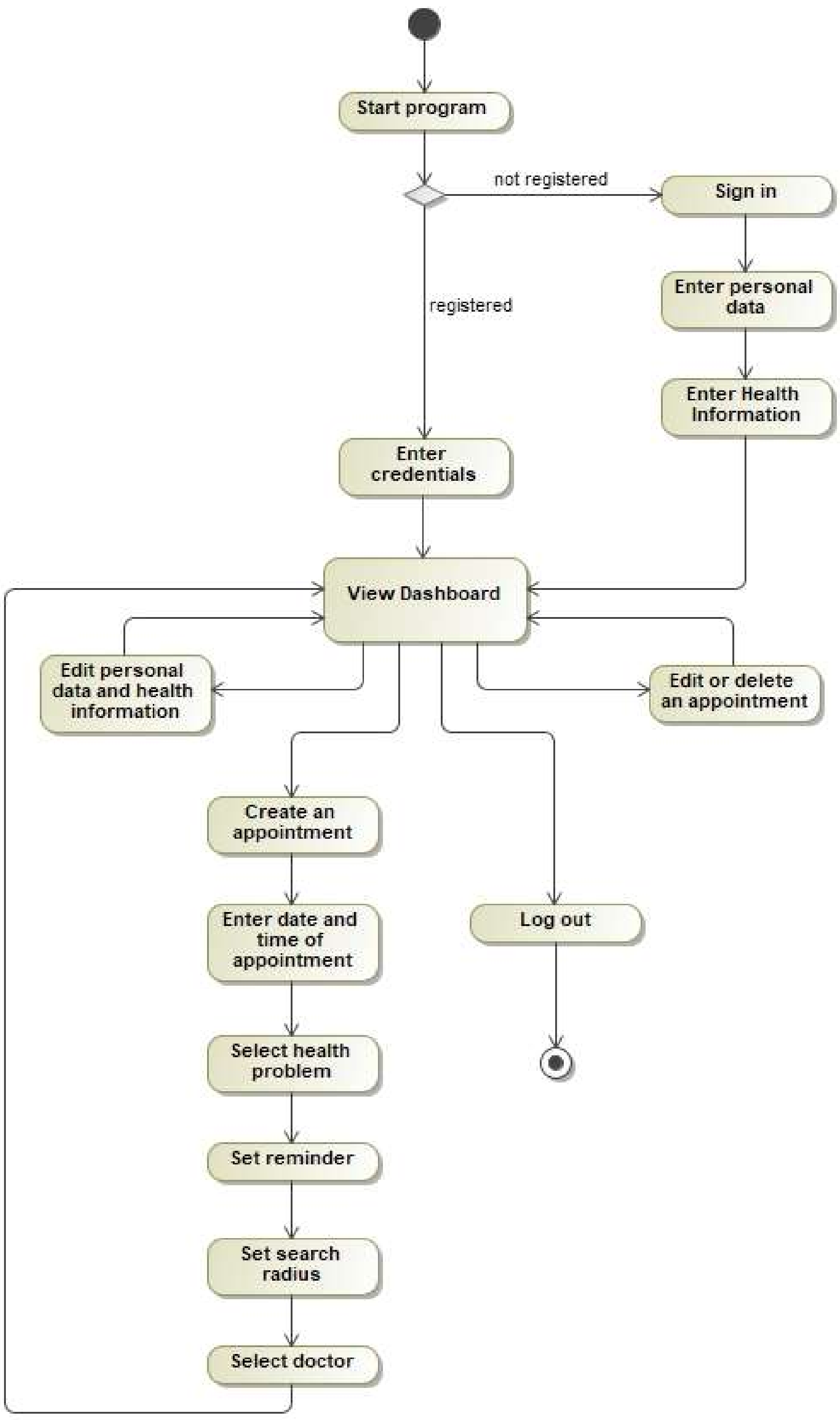


This Diagram illustrates the overall structure of the "*Smart eHealth Consulting System*". The System is distributed in two actors which have different specializations.

Starting off with the "*User*" which itself is divided into "*Personal data*" and "*Service*". On the one hand in order to use this system, a user account is required where personal information regarding the patient and his/her health information is needed. In case if any changes occur, the user is able to edit his/her data afterwards. Furthermore, it is possible that a patient can save all his/her information as a PDF file. Moving on to the service of our system which is provided for the user, he/she has different options after creating an appointment for a specialized doctor. It must be possible to have an overview of all upcoming appointments, which a user has, and in case if the appointment does not fit in the user's schedule, he/she is able to shift or delete the appointment.

Then there is the "*Admin*" who has the privilege of having access to all users, registered in the system. He/she is able to edit or delete user profiles.

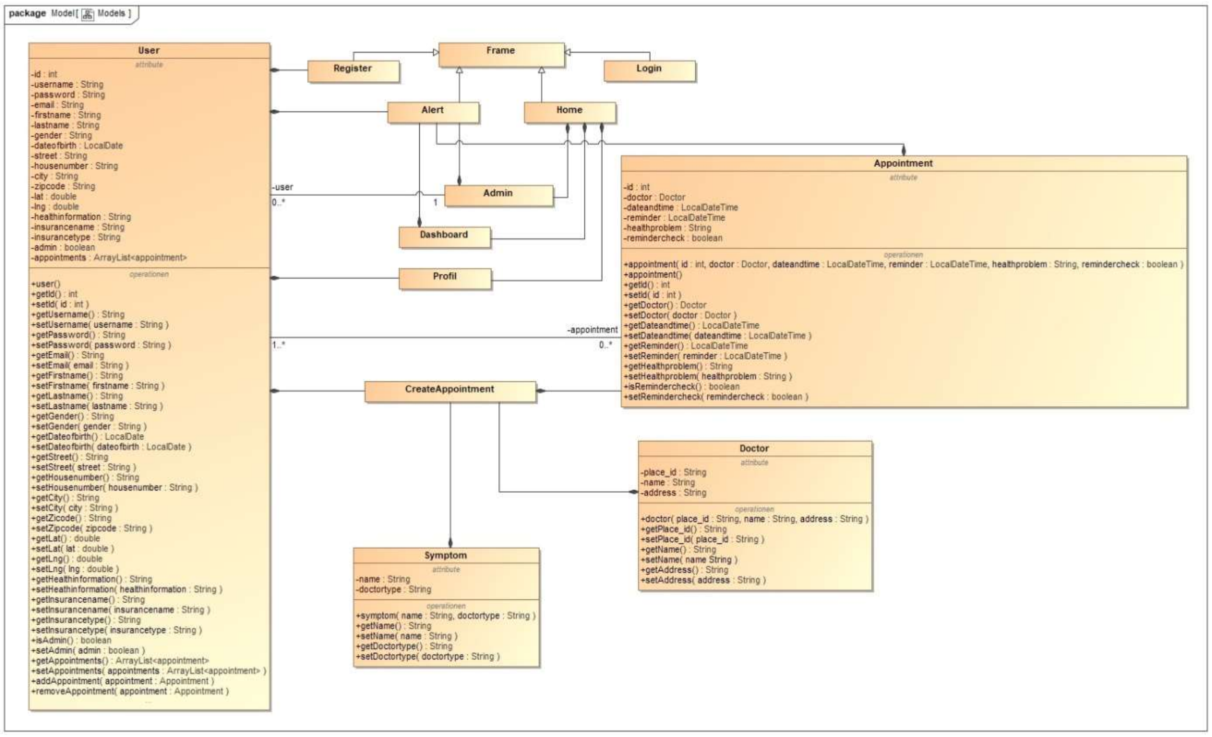
## 2.2 Activity Diagram



The activity diagram shows the flow of all functionalities which the "*Smart eHealth Consulting System*" provides. In the beginning the user starts the program and needs to enter his/her credentials in case he/she is already registered in the system. If the user is not already registered, he/she is able to create an account where personal data and the health information is required. After the login screen follows the dashboard where the user can create an appointment where he/she needs to enter date and time, select his/her health problem, set a reminder, adjust the search radius, and lastly select a doctor. If the appointment is created successfully the user gets redirected to the dashboard where editing and deleting the appointment is possible. In addition to that if any personal changes occur e.g., the user moves the residence or changes the insurance etc. he/she can edit the personal data and health information too. In the end if the user logs out the program terminates.

## 2.3 Class Diagram

The following class diagrams describe the structure of the system by showing the graphical representation of the classes, the implemented attributes, methods, and the relationship between the objects.


package Model[ Models ]
User
attribute
-id : int
-username : String
-password : String
-email : String
-firstname : String
-lastname : String
-gender : String
-dateofbirth : LocalDate
-street : String
-housenumber : String
-city : String
-zipcode : String
-lat : double
-lng : double
-healthinformation : String
-insurancename : String
-insurancetype : String
-admin : boolean
-appointments : ArrayList<appointment>
operationen
+user()
+getId() : int
+setId( id : int )
+getUsername() : String
+setUsername( username : String )
+getPassword() : String
+setPassword( password : String )
+getEmail() : String
+setEmail( email : String )
+getFirstname() : String
+setFirstname( firstname : String )
+getLastname() : String
+setLastname( lastname : String )
+getGender() : String
+setGender( gender : String )
+getDateofbirth() : LocalDate
+setDateofbirth( dateofbirth : LocalDate )
+getStreet() : String
+setStreet( street : String )
+getHousenumber() : String
+setHousenumber( housenumber : String )
+getCity() : String
+setCity( city : String )
+getZicode() : String
+setZipcode( zipcode : String )
+getLat() : double
+setLat( lat : double )
+getLng() : double
+setLng( lng : double )
+getHealthinformation() : String
+setHeathinformation( healthinformation : String )
+getInsurancename() : String
+setInsurancename( insurancename : String )
+getInsurancetype() : String
+setInsurancetype( insurancetype : String )
+isAdmin() : boolean
+setAdmin( admin : boolean )
+getAppointments() : ArrayList<appointment>
+setAppointments( appointments : ArrayList<appointment> )
+addAppointment( appointment : Appointment )
+removeAppointment( appointment : Appointment )
Register
Frame
Login
Alert
Home
-user
0..*
1
Admin
Dashboard
Profil
-appointment
1..*
0..*
Appointment
attribute
-id : int
-doctor : Doctor
-dateandtime : LocalDateTime
-reminder : LocalDateTime
-healthproblem : String
-remindercheck : boolean
operationen
+appointment( id : int, doctor : Doctor, dateandtime : LocalDateTime, reminder : LocalDateTime, healthproblem : String, remindercheck : boolean )
+appointment()
+getId() : int
+setId( id : int )
+getDoctor() : Doctor
+setDoctor( doctor : Doctor )
+getDateandtime() : LocalDateTime
+setDateandtime( dateandtime : LocalDateTime )
+getReminder() : LocalDateTime
+setReminder( reminder : LocalDateTime )
+getHealthproblem() : String
+setHealthproblem( healthproblem : String )
+isRemindercheck() : boolean
+setRemindercheck( remindercheck : boolean )
CreateAppointment
Doctor
attribute
-place_id : String
-name : String
-address : String
operationen
+doctor( place_id : String, name : String, address : String )
+getPlace_id() : String
+setPlace_id( place_id : String )
+getName() : String
+setName( name String )
+getAddress() : String
+setAddress( address : String )
Symptom
attribute
-name : String
-doctortype : String
operationen
+symptom( name : String, doctortype : String )
+getName() : String
+setName( name : String )
+getDoctortype() : String
+setDoctortype( doctortype : String )

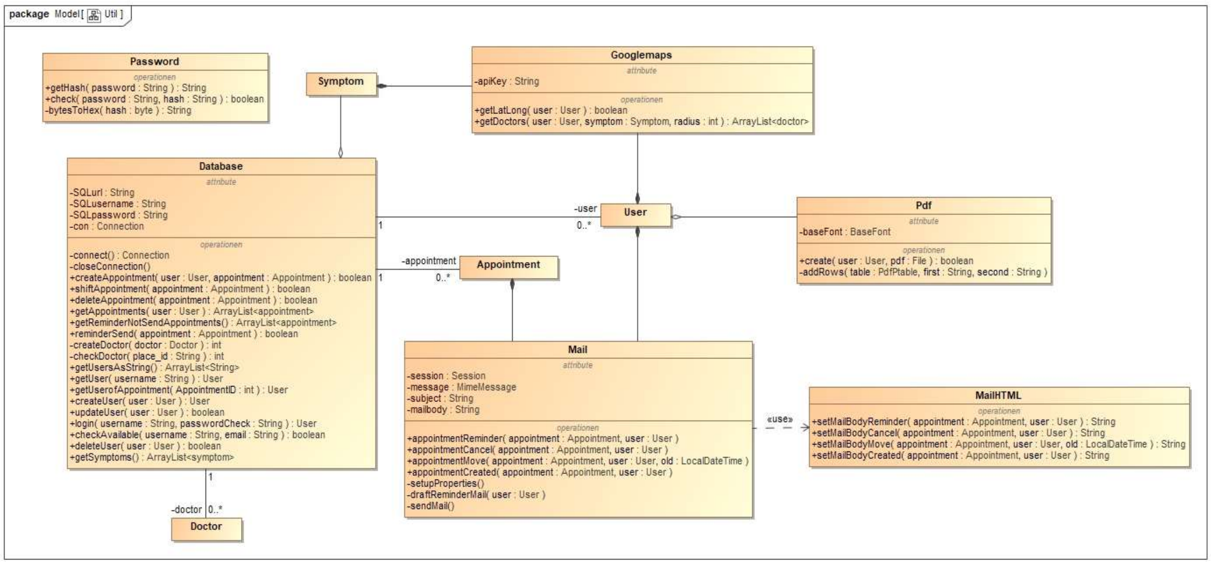
This class diagram represents the structure of the classes from the *Models* package and their relationship to each other and to the classes that are responsible for the Graphical User Interface. The classes of the *Models* package handle the data while the frame classes (*Register*, *Alert*, *Home* and *Login*) and the panel classes (*Admin*, *Dashboard*, *Profil* and *createAppointment*) serve as input functionality of required information and as presentation of the data.

**Generalization**: Classes “Register”, “Alert”, “Home” and “Login” are inheriting from “Frame” for the advantage of method overriding and to reuse the right fonts, and sizes.

**Composition Relationships:** After the registration of a new user with “Register” the data gets stored in the database through “User” therefore the class “Register” is dependent on “User”. The panels “Admin”, “Dashboard” and “Profil” are displayed through the frame “Home” that’s why they are depending on “Home”.

For the confirmation and the error message when deleting an appointment or a user the frame “Alert” needs the classes “Dashboard”, “Appointment” and “User”, so a composition relationship is needed between these elements and “Alert”. The data of the class “User” is needed to display the user’s profile in the “Profil” tab, which explains the composition relationships between “User” and “Profil”. To create an appointment “createAppointment” needs data from the database that it gets through the classes “User”, “Doctor” and “Symptoms”, which makes “createAppointment” dependent on these classes. “Appointment” requires the creation of an appointment with “createAppointment”; hence the class “Appointment” is dependent.

**Association Relationships:** “Admin” has access to edit and delete users which explains the association to “User”. The association relationship between “User” and “Appointment” implies that a user can have an unlimited number of appointments.


package Model[ Util ]
Password
operationen
+getHash( password : String ) : String
+check( password : String, hash : String ) : boolean
-bytesToHex( hash : byte ) : String
Symptom
Googlemaps
attribute
-apiKey : String
operationen
+getLatLong( user : User ) : boolean
+getDoctors( user : User, symptom : Symptom, radius : int ) : ArrayList<doctor>
Database
attribute
-SQLurl : String
-SQLusername : String
-SQLpassword : String
-con : Connection
operationen
-connect() : Connection
-closeConnection()
+createAppointment( user : User, appointment : Appointment ) : boolean
+shiftAppointment( appointment : Appointment ) : boolean
+deleteAppointment( appointment : Appointment ) : boolean
+getAppointments( user : User ) : ArrayList<appointment>
+getReminderNotSendAppointments() : ArrayList<appointment>
+reminderSend( appointment : Appointment ) : boolean
-createDoctor( doctor : Doctor ) : int
-checkDoctor( place_id : String ) : int
+getUsersAsString() : ArrayList<String>
+getUser( username : String ) : User
+getUserofAppointment( AppointmentID : int ) : User
+createUser( user : User ) : User
+updateUser( user : User ) : boolean
+login( username : String, passwordCheck : String ) : User
+checkAvailable( username : String, email : String ) : boolean
+deleteUser( user : User ) : boolean
+getSymptoms() : ArrayList<symptom>
1
-user
0..*
User
Pdf
attribute
-baseFont : BaseFont
operationen
+create( user : User, pdf : File ) : boolean
-addRows( table : PdfPtable, first : String, second : String )
-appointment
1
0..*
Appointment
Mail
attribute
-session : Session
-message : MimeMessage
-subject : String
-mailbody : String
operationen
+appointmentReminder( appointment : Appointment, user : User )
+appointmentCancel( appointment : Appointment, user : User )
+appointmentMove( appointment : Appointment, user : User, old : LocalDateTime )
+appointmentCreated( appointment : Appointment, user : User )
-setupProperties()
-draftReminderMail( user : User )
-sendMail()
«use»
MailHTML
operationen
+setMailBodyReminder( appointment : Appointment, user : User ) : String
+setMailBodyCancel( appointment : Appointment, user : User ) : String
+setMailBodyMove( appointment : Appointment, user : User, old : LocalDateTime ) : String
+setMailBodyCreated( appointment : Appointment, user : User ) : String
1
-doctor
0..*
Doctor

This diagram represents the classes of the *Util* package and the relationships to the classes *User*, *Appointment*, *Doctor* and *Symptoms*.

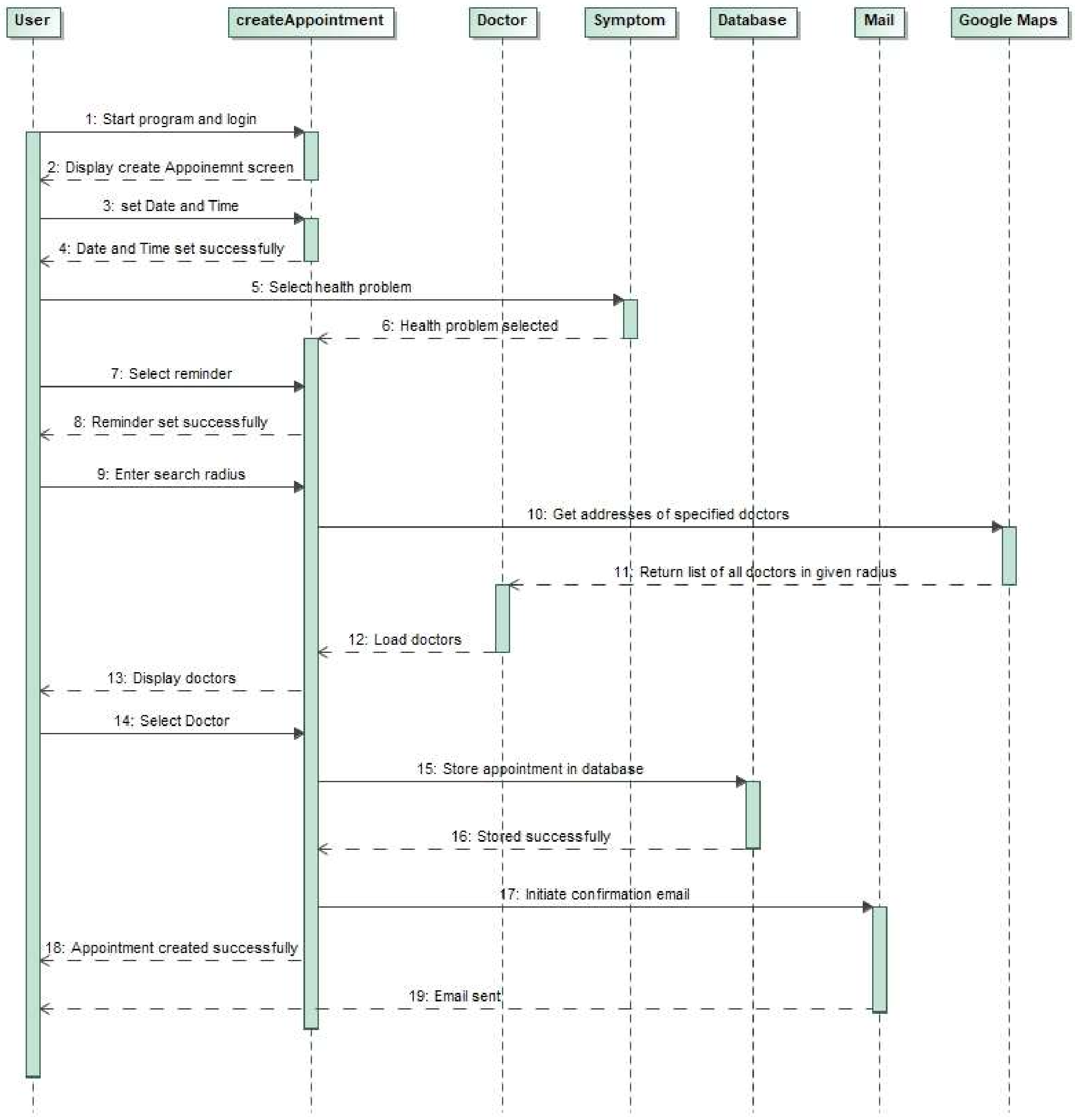
**Association Relationships:** To store users, appointments and doctors, the database uses the classes “User”, “Appointment” and “Doctor”.

**Composition Relationships:** “Googlemaps” needs the “User” class to get the coordinates of the users address and the doctors in the user’s area. It also needs “Symptom” to get the type of the doctor. For the email notifications the class “Mail” requires data from “Appointment” and “User”. “ Mail” also uses “MailHTML” to get the right body for the different operations.

**Aggregation Relationship**: The class “Symptom” is subordinate to “Database” for the reason that “Database” gets the symptoms and the associate doctor types from the database.

## 2.4 Sequence Diagram

This sequence diagram provides an insight of the main functionality of the "*Smart eHealth Consulting System*" which is creating an appointment with a specialized doctor based on the health problem a customer is having.


User
createAppointment
Doctor
Symptom
Database
Mail
Google Maps
1: Start program and login
2: Display create Appoinemnt screen
3: set Date and Time
4: Date and Time set successfully
5: Select health problem
6: Health problem selected
7: Select reminder
8: Reminder set successfully
9: Enter search radius
10: Get addresses of specified doctors
11: Return list of all doctors in given radius
12: Load doctors
13: Display doctors
14: Select Doctor
15: Store appointment in database
16: Stored successfully
17: Initiate confirmation email
18: Appointment created successfully
19: Email sent

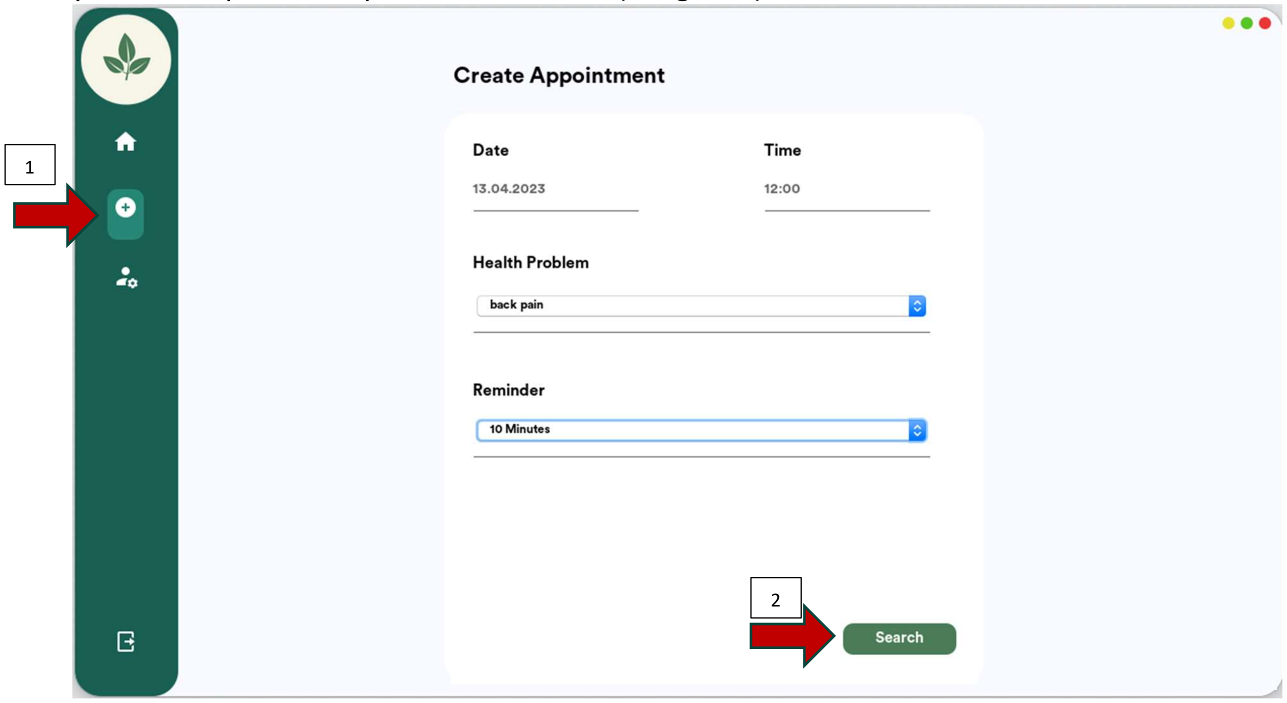
The whole process gets initiated by starting the program where the user needs to enter his/her credentials to get logged in into the system. In the home menu a new appointment can be created by clicking the create appointment button on the side bar.

After that the user gets forwarded to the “Create Appointment” tab where information like date and time of the appointment and the user’s health problem are necessary. Additionally, a reminder can be set so that the user gets notified when an appointment is coming up. In these first steps three classes are involved, which are the classes “User”, “createAppointment” and “Symptom”. The user then has the possibility to regulate the search radius for the specialized doctors nearby the residence of the user and subsequently “Google Maps” will fetch all locations that lay in the given search radius and get prepared to return a list of all doctors that match the user’s health problem. The “Doctor” class stores the name and address of the given doctors and returns it to the “createAppointment” class where it gets ready to be displayed in a table, so that the user has an overview of all matching doctors. Now he/she is able to select a preferred doctor and create a new appointment by submitting all information.

Lastly the appointment gets stored in the “Database” and the “createAppointment” class initiates a confirmation mail that gets sent to the users registered email by the “Mail” class.

## 2.5 How to Create, Edit and Delete an Appointment

To be able to create an appointment you have to be registered in our system. After you have logged in, you have to click on the “Add” button, which will guide you to another tab where you must input the required information. (Image 1.1)

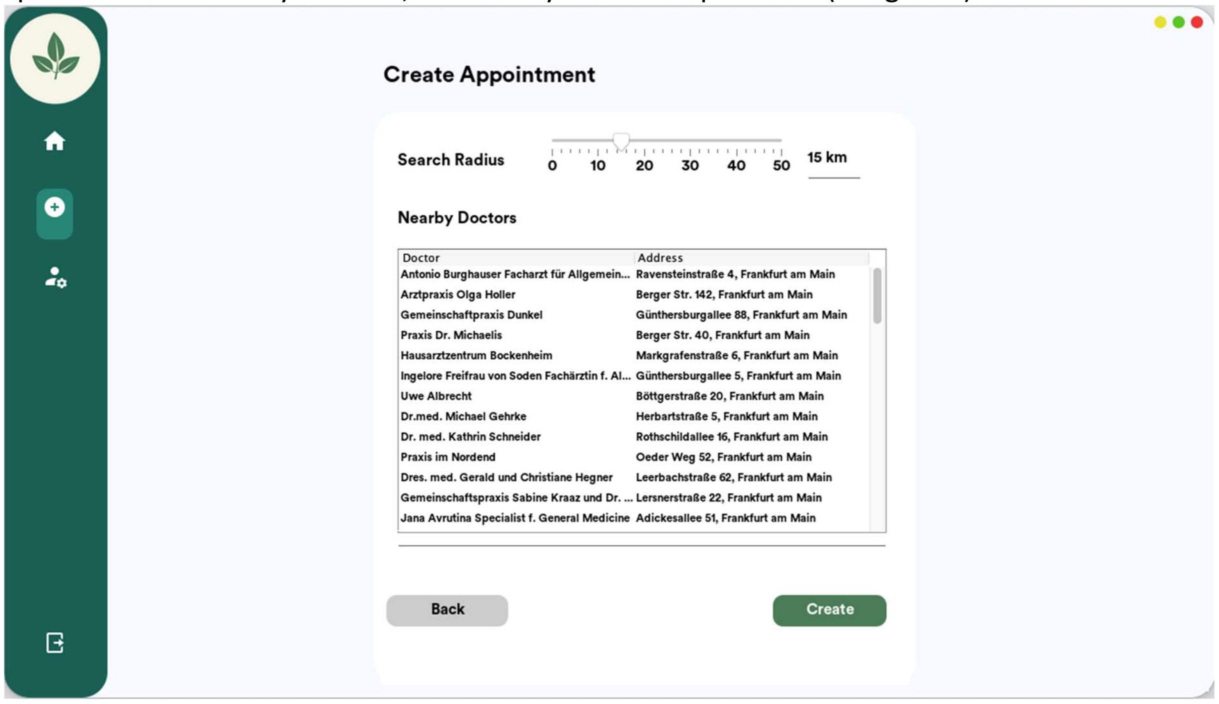


Create Appointment

Date: 13.04.2023
Time: 12:00
Health Problem: back pain
Reminder: 10 Minutes

*Image 1.1*

By clicking on „Search“ (Image 1.1) there will appear a window which displays a list with every specialized doctor in your area, based on your health problem. (Image 1.2)



Create Appointment

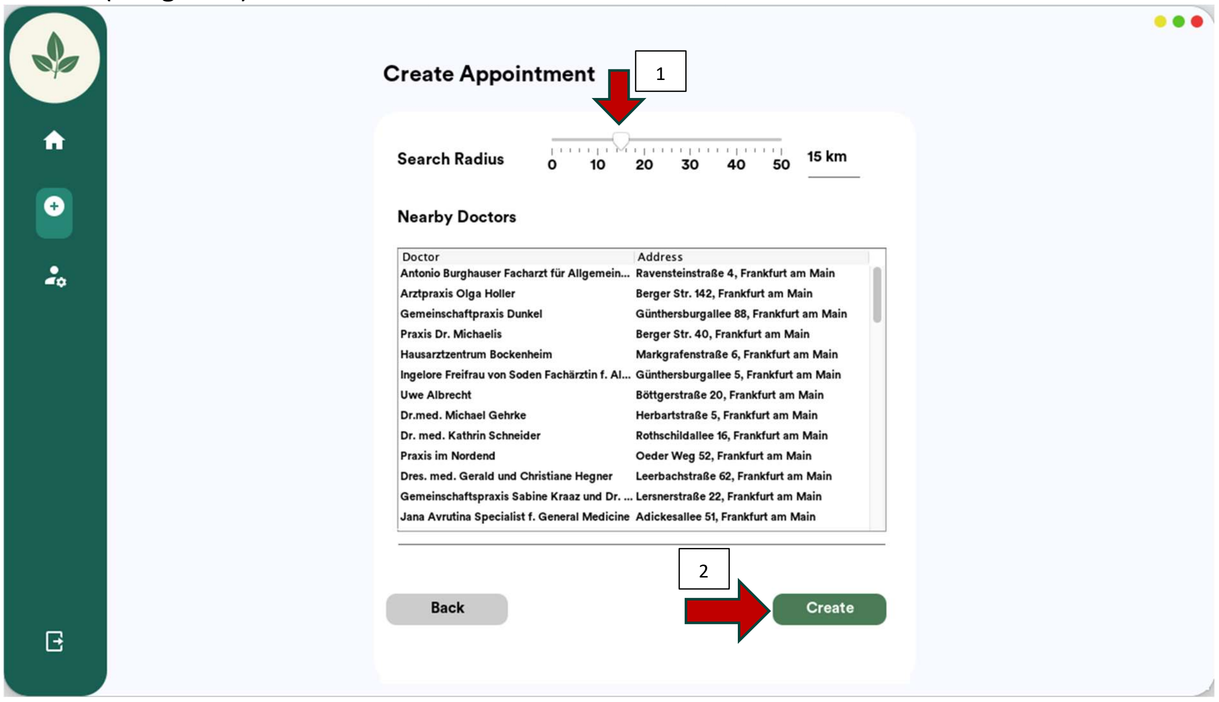
Search Radius: 15 km

Nearby Doctors

| Doctor | Address |
|---|---|
| Antonio Burghauser Facharzt für Allgemein... | Ravensteinstraße 4, Frankfurt am Main |
| Arztpraxis Olga Holler | Berger Str. 142, Frankfurt am Main |
| Gemeinschaftpraxis Dunkel | Günthersburgallee 88, Frankfurt am Main |
| Praxis Dr. Michaelis | Berger Str. 40, Frankfurt am Main |
| Hausarztzentrum Bockenheim | Markgrafenstraße 6, Frankfurt am Main |
| Ingelore Freifrau von Soden Fachärztin f. Al... | Günthersburgallee 5, Frankfurt am Main |
| Uwe Albrecht | Böttgerstraße 20, Frankfurt am Main |
| Dr.med. Michael Gehrke | Herbartstraße 5, Frankfurt am Main |
| Dr. med. Kathrin Schneider | Rothschildallee 16, Frankfurt am Main |
| Praxis im Nordend | Oeder Weg 52, Frankfurt am Main |
| Dres. med. Gerald und Christiane Hegner | Leerbachstraße 62, Frankfurt am Main |
| Gemeinschaftspraxis Sabine Kraaz und Dr. ... | Lersnerstraße 22, Frankfurt am Main |
| Jana Avrutina Specialist f. General Medicine | Adickesallee 51, Frankfurt am Main |

*Image 1.2*

If you do not have a car or you cannot travel with train or other means of transport, you can adjust your area from 0 km to a maximum of 50 km. Once you found a doctor, to finish the process you must click on "Create" and after 1-3 minutes you will receive the confirmation by E-Mail. (Image 1.3)



**Create Appointment**

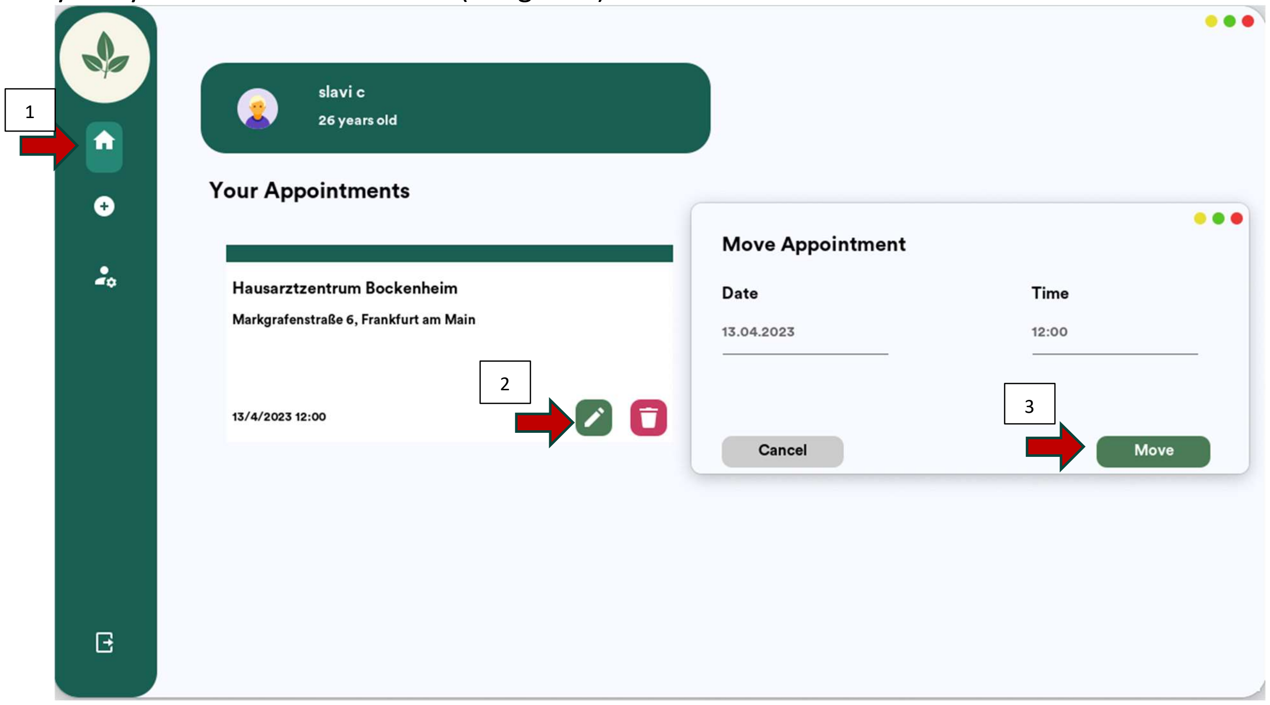
Search Radius: 15 km

**Nearby Doctors**

| Doctor | Address |
|---|---|
| Antonio Burghauser Facharzt für Allgemein... | Ravensteinstraße 4, Frankfurt am Main |
| Arztpraxis Olga Holler | Berger Str. 142, Frankfurt am Main |
| Gemeinschaftpraxis Dunkel | Günthersburgallee 88, Frankfurt am Main |
| Praxis Dr. Michaelis | Berger Str. 40, Frankfurt am Main |
| Hausarztzentrum Bockenheim | Markgrafenstraße 6, Frankfurt am Main |
| Ingelore Freifrau von Soden Fachärztin f. Al... | Günthersburgallee 5, Frankfurt am Main |
| Uwe Albrecht | Böttgerstraße 20, Frankfurt am Main |
| Dr.med. Michael Gehrke | Herbartstraße 5, Frankfurt am Main |
| Dr. med. Kathrin Schneider | Rothschildallee 16, Frankfurt am Main |
| Praxis im Nordend | Oeder Weg 52, Frankfurt am Main |
| Dres. med. Gerald und Christiane Hegner | Leerbachstraße 62, Frankfurt am Main |
| Gemeinschaftspraxis Sabine Kraaz und Dr. ... | Lersnerstraße 22, Frankfurt am Main |
| Jana Avrutina Specialist f. General Medicine | Adickesallee 51, Frankfurt am Main |

Image1.3

If you are unable to attend your booked appointment, you can move it by clicking on "Home" and there you will see all your created appointments. Click on the "Edit" button of your selected appointment and you will be able to choose a preferred date and time. To confirm you must click on "Move". The confirmation with your moved appointment will be send to you by e-mail within 1 minute. (Image 1.4)



slavi c
26 years old

**Your Appointments**

Hausarztzentrum Bockenheim
Markgrafenstraße 6, Frankfurt am Main
13/4/2023 12:00

**Move Appointment**

Date: 13.04.2023
Time: 12:00

Image 1.4

If you are feeling better and you want to cancel your appointment you can do so by clicking *Home>> delete icon>> confirm on "delete".* The confirmation of your deleted appointment will be sent to you by E-Mail within 1 minute. (Image 1.5)

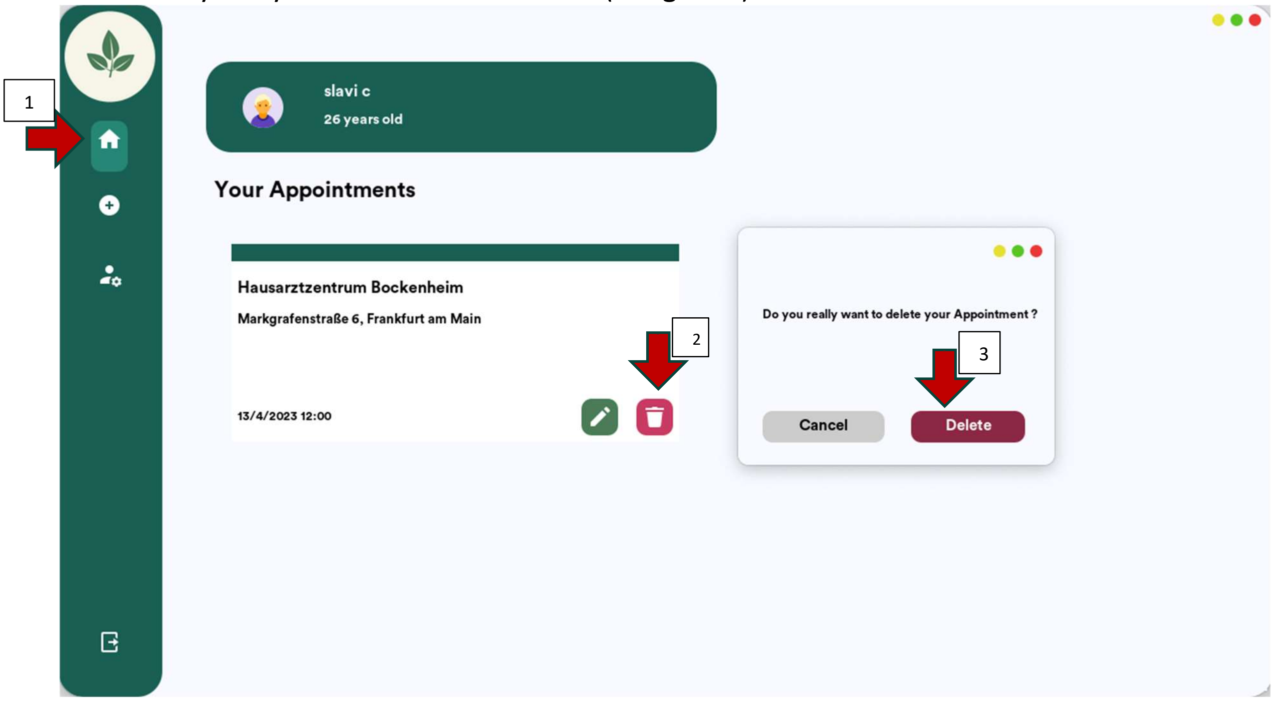
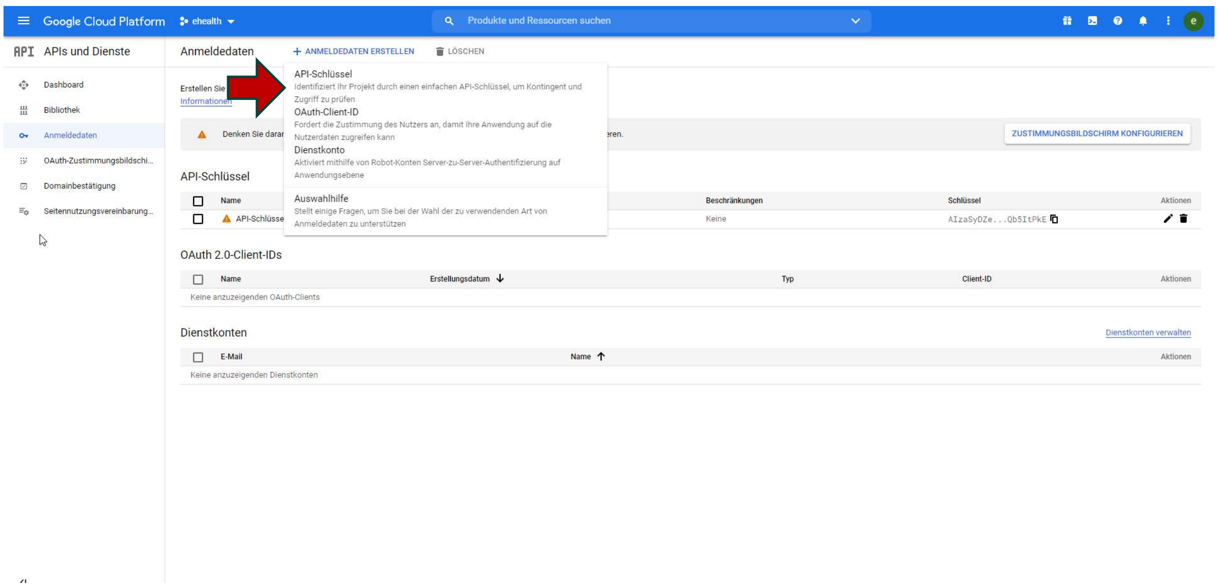

1
slavi c
26 years old
Your Appointments
Hausarztzentrum Bockenheim
Markgrafenstraße 6, Frankfurt am Main
13/4/2023 12:00
2
Do you really want to delete your Appointment ?
3
Cancel
Delete

Image 1.5

## 2.6 Doctor Suggestion

One of our biggest challenges was the Doctor Search. We had to search for doctors of a specific doctor type within a given radius. The best solution is to use an API to fetch real doctors.

There are many APIs to choose from like Google Maps API, HERE API or the OpenStreetMap API. We decided to use the Google Maps API because it has the highest number of doctors in its database. Google Maps has an endpoint called Nearby Search that is perfect for our doctor search. To use this API, you have to create an API-Key in the Google Console. (Image 2.1)



API-Schlüssel
Identifiziert Ihr Projekt durch einen einfachen API-Schlüssel, um Kontingent und Zugriff zu prüfen
OAuth-Client-ID
Fordert die Zustimmung des Nutzers an, damit Ihre Anwendung auf die Nutzerdaten zugreifen kann
Dienstkonto
Aktiviert mithilfe von Robot-Konten Server-zu-Server-Authentifizierung auf Anwendungsebene
Auswahlhilfe
Stellt einige Fragen, um Sie bei der Wahl der zu verwendenden Art von Anmeldedaten zu unterstützen

Erstellen Sie
Informationen

Denken Sie darar ... eren.

API-Schlüssel

| Name | Beschränkungen | Schlüssel | Aktionen |
|---|---|---|---|
| API-Schlüsse | Keine | AIzaSyDZe...Qb5ItPkE | |

OAuth 2.0-Client-IDs

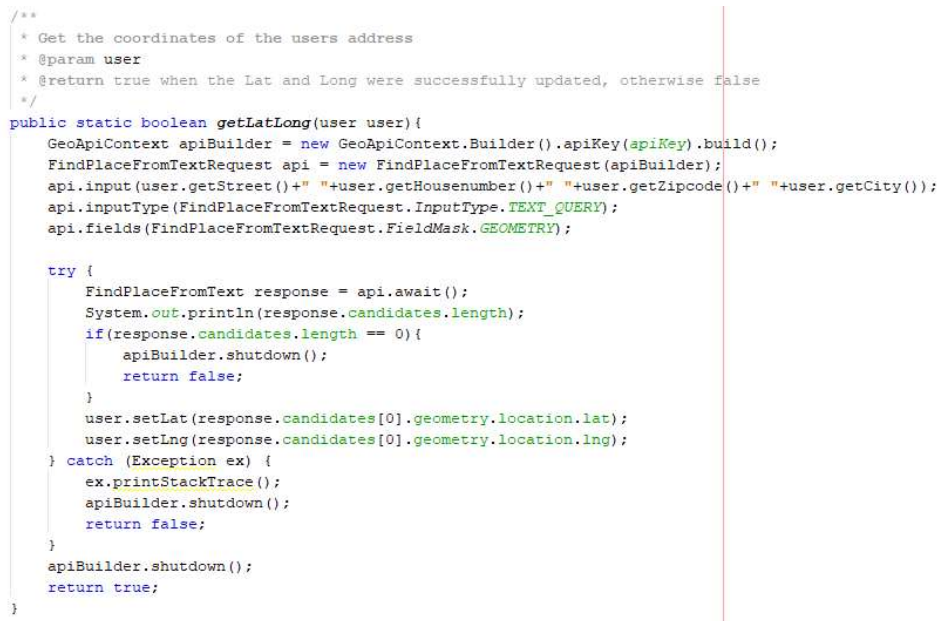
| Name | Erstellungsdatum | Typ | Client-ID | Aktionen |
|---|---|---|---|---|
| Keine anzuzeigenden OAuth-Clients | | | | |

Dienstkonten

| E-Mail | Name | Aktionen |
|---|---|---|
| Keine anzuzeigenden Dienstkonten | | |

*Image 2.1*

The Problem with the API is that it costs money but luckily Google gives you 300$ start credits to test its services. Google also provides you with an API Wrapper for Java, so you do not have to code your own HTTP requests. To fetch data from the Nearby Search endpoint you have to provide a keyword, radius and the location in coordinates. Our next problem was to find a way to convert the address of the patient to coordinates. To do this we decided to use another endpoint of the Google Maps API, called Find Place. You need to send your address to the endpoint, and it will return the coordinates. We thought that the best way to handle this is to also save the coordinates in the database when the patient creates/updates his account. This way we can send requests to the Nearby Search endpoint right away without fetching the coordinates with every doctor search, to save time and "money" in the Google Maps API.

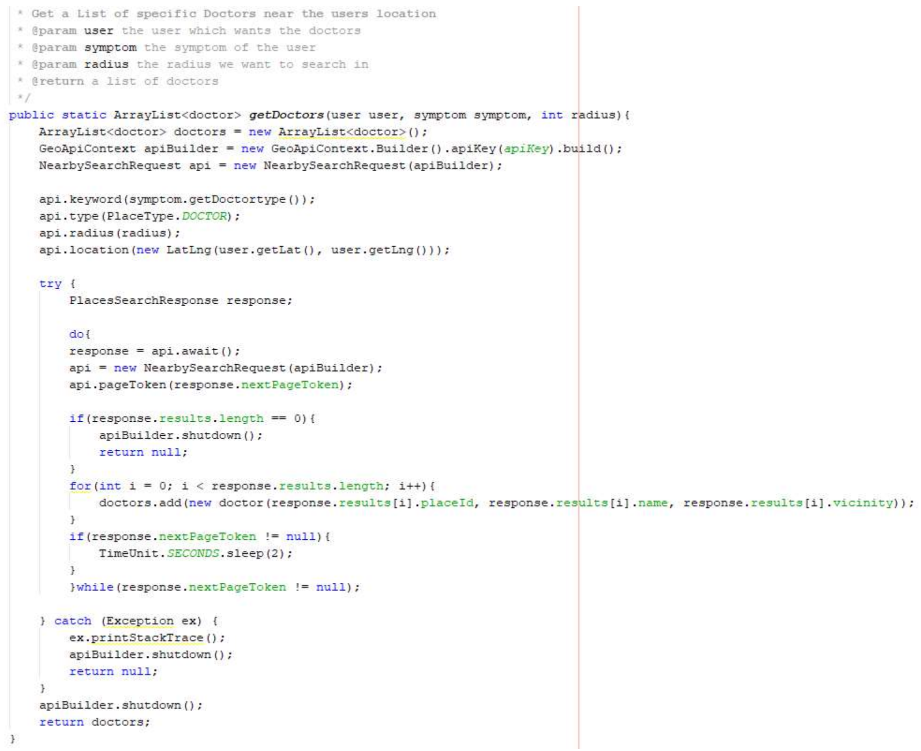


```
/**
 * Get the coordinates of the users address
 * @param user
 * @return true when the Lat and Long were successfully updated, otherwise false
 */
public static boolean getLatLong(user user){
    GeoApiContext apiBuilder = new GeoApiContext.Builder().apiKey(apiKey).build();
    FindPlaceFromTextRequest api = new FindPlaceFromTextRequest(apiBuilder);
    api.input(user.getStreet()+" "+user.getHousenumber()+" "+user.getZipcode()+" "+user.getCity());
    api.inputType(FindPlaceFromTextRequest.InputType.TEXT_QUERY);
    api.fields(FindPlaceFromTextRequest.FieldMask.GEOMETRY);

    try {
        FindPlaceFromText response = api.await();
        System.out.println(response.candidates.length);
        if(response.candidates.length == 0){
            apiBuilder.shutdown();
            return false;
        }
        user.setLat(response.candidates[0].geometry.location.lat);
        user.setLng(response.candidates[0].geometry.location.lng);
    } catch (Exception ex) {
        ex.printStackTrace();
        apiBuilder.shutdown();
        return false;
    }
    apiBuilder.shutdown();
    return true;
}
```

*Image 2.2*

In Image 2.2 you can see the implementation of the convert.



```
 * Get a List of specific Doctors near the users location
 * @param user the user which wants the doctors
 * @param symptom the symptom of the user
 * @param radius the radius we want to search in
 * @return a list of doctors
 */
public static ArrayList<doctor> getDoctors(user user, symptom symptom, int radius){
    ArrayList<doctor> doctors = new ArrayList<doctor>();
    GeoApiContext apiBuilder = new GeoApiContext.Builder().apiKey(apiKey).build();
    NearbySearchRequest api = new NearbySearchRequest(apiBuilder);

    api.keyword(symptom.getDoctortype());
    api.type(PlaceType.DOCTOR);
    api.radius(radius);
    api.location(new LatLng(user.getLat(), user.getLng()));

    try {
        PlacesSearchResponse response;

        do{
        response = api.await();
        api = new NearbySearchRequest(apiBuilder);
        api.pageToken(response.nextPageToken);

        if(response.results.length == 0){
            apiBuilder.shutdown();
            return null;
        }
        for(int i = 0; i < response.results.length; i++){
            doctors.add(new doctor(response.results[i].placeId, response.results[i].name, response.results[i].vicinity));
        }
        if(response.nextPageToken != null){
            TimeUnit.SECONDS.sleep(2);
        }
        }while(response.nextPageToken != null);

    } catch (Exception ex) {
        ex.printStackTrace();
        apiBuilder.shutdown();
        return null;
    }
    apiBuilder.shutdown();
    return doctors;
}
```

*Image 2.3*

In Image 2.3 you can see the implementation of the doctor search. In every request to the endpoint, it will return a maximum of 20 doctors and a nextPageToken if the API found more than 20 doctors. Then you can send the same request just with the nextPageToken set and it will return another 20 doctors and a another nextPageToken if the API found more than 20. This can happen up to 3 times so you can fetch at most 60 doctors from the API at once.

# 3. Conclusion

This project was a very unique experience for our group. None of us has worked on a bigger project like this before. Even though we divided different tasks between us, it felt like we never worked completely alone on something. Often two or more of us worked together on the same task or helped each other when necessary.
In the beginning it was not easy for us because it was our first time. Nobody really knew how we could start or how we will work on the project. But after getting the start done by sharing some ideas how we want our program to work, what functions are essential for us and some rough ideas how we could implement them, we could finally start working.
Due to our regular meetings, all of us were always up to date and everyone knew what was happening right now. Of course, not everything went as easy as we might have wished, there were problems that at first seemed almost impossible to solve, but someone of our group always had an idea and, in the end, we managed to resolve the problems.

Looking back at everything, we learned a lot about coding in Java, using the NetBeans IDE and Github to gather our work. Learning basic code in the lectures is one thing, but in a bigger project this basic code only helps you to a certain extent, you have to improvise and do research by yourself way more as you do not get clear exercises that show you exactly what you need to do.
This is a unique experience in a project that lectures cannot teach you the same way. Things like these made this project so valuable for us.

So, coming to a conclusion, we are very happy with our project and had a lot of fun working on it, and we are sure that we learned many things, about programming, but especially about working in a team together on the same project.

# Bookmark Links

Autor’s Declaration : https://ulsites.ul.ie › law › sites › default › files

Diagrams: https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/

Google Console: https://console.cloud.google.com/