


UNIVERSITY OF PIRAEUS - DEPARTMENT OF INFORMATICS

Πανεπιστήμιο Πειραιώς - Τμήμα Πληροφορικής

MSc «Informatics»

ΠΜΣ «Πληροφορική»

MSc Thesis
Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	Android application for detecting and recording Covid19 cases using Spring Boot Back-End. Android εφαρμογή για εντοπισμό και καταγραφή κρουσμάτων Covid19 με χρήση Spring Boot Back-End.
Student's name-surname: Ονοματεπώνυμο Φοιτητή:	Georgia Karagianni Καραγιάννη Γεωργία
Father's Name: Πατρώνυμο:	Dimitrios Δημήτριος
Student's ID No: Αριθμός Μητρώου:	ΜΠΠΛ18028
Supervisor: Επιβλέπων:	Efthimios Alepis, Associate Professor Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής

 Ημερομηνία Παράδοσης **Δεκέμβριος 2022**

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)

(υπογραφή)

(υπογραφή)

Efthimios Alepis
Associate Professor

Ευθύμιος Αλέπης
Αναπληρωτής Καθηγητής

Maria Virvou
Professor

Μαρία Βίρβου
Καθηγήτρια

Evangelos
Sakkopoulos
Associate Professor

Ευάγγελος Σακκόπουλος
Αναπληρωτής Καθηγητής

Table of Contents

Abstract	4
English.....	4
Ελληνικά.....	4
Introduction.....	5
Review Overview	6
User's Manual	8
System's Architecture	26
Spring Framework.....	26
Spring Boot	26
Spring Initializr.....	27
Spring Security OAuth.....	30
Spring Data JPA.....	31
IntelliJIDEA.....	32
PostgreSQL.....	32
Volley.....	39
Annotations	42
Future Expansions	48
Conclusions.....	49
Acknowledgements	49
References	50

Abstract

English

This Thesis focuses on the implementation of an Android application, which detects and records Covid19 cases, using the Spring Boot tool as back end and a PostgreSQL database.

The application includes a registration form and a user login form, as well as a map form that displays the user's location. There is also a form that presents statistical data of the users of the application by using graphical representations. All data are stored in a server using a PostgreSQL database. Finally, spring security is used to securely encrypt passwords.

The purpose of the application is for the user to be able to enter his personal data and to declare if he is infected with the Covid19 virus and to export statistics based on the registered data of the users, as well as to present an efficient way of connection of the aforementioned technologies.

Ελληνικά

Η παρούσα πτυχιακή επικεντρώνεται στην υλοποίηση μιας Android εφαρμογής, η οποία εντοπίζει και καταγράφει τα κρούσματα Covid19, με τη χρήση του εργαλείου Spring Boot ως back-end και μιας PostgreSQL βάσης.

Η εφαρμογή περιλαμβάνει μια φόρμα εγγραφής και μια φόρμα σύνδεσης του χρήστη καθώς και μια φόρμα με χάρτες που εμφανίζεται η τοποθεσία του. Επίσης, υπάρχει μια φόρμα που παρουσιάζει στατιστικά δεδομένα των χρηστών της εφαρμογής με χρήση γραφικής αναπαράστασης. Όλα τα δεδομένα αποθηκεύονται σε έναν server με τη χρήση μιας βάσης δεδομένων PostgreSQL. Τέλος, χρησιμοποιείται spring security για την ασφαλή κρυπτογράφηση των κωδικών.

Σκοπός της εφαρμογής είναι να μπορεί ο χρήστης να καταχωρεί τα προσωπικά του στοιχεία και να δηλώνει εάν νοσεί από τον ιό Covid19 προκειμένου να εξαχθούν στατιστικά στοιχεία με βάση τα καταχωρημένα στοιχεία των χρηστών, καθώς και να παρουσιάσει έναν αποτελεσματικό τρόπο σύνδεσης των προαναφερθέντων τεχνολογιών.

Introduction

Due to the Corona virus 2019 pandemic, governments all over the world decided to take measures for the prevention of further spread of the virus.

They provided tests to check if you are infected with the virus. Also, they banned mass gatherings in public and in private places. Further, they strongly recommended the use of protective masks in order to contain the spread of the virus. Even in some cases they restricted every external activity for some period of time.

Quickly the governments realized that another method to track and prevent the spread of the virus is with applications compatible with our smartphones. So, everyone can have access and live update their status of whether they are infected or not.

The object of the present thesis is the development of an android application, which will give the user the following features: Firstly, he can create an account and then log in, so he can share his covid status and location through the application and secondly, he will be able to stay updated through statistic diagrams for the number and percentage of people infected or not with covid, about their age or country they live in, etc. The second one is achieved by using a server with which the application will communicate with, and which has been implemented by using the Spring Boot tool.

So, every time a user makes use of the application to state whether he has covid or not and show his location, the data will be sent to the server and will be stored in a PostgreSQL database. By collecting these results from each user, it will be possible to extract the statistics about the percentages of infected and uninfected people. Let's note at this point, that each user will have a unique UserID, which will be automatically given to him with the completion of his registration to the application. During the registry, the user will register his name, surname, gender, covid state, date of birth, email and set a password to enter the application. The data stored in the database will be all the above mentioned, which are acquired during the registration, plus the coordinates (latitude, longitude) of user's location at that moment (from GPS).

The difference with this application is that uses a new technology to do all these functions, Spring Boot, in accordance with IntelliJ as for the back-end, Android Studio for front-end and PostgreSQL as the database. We will further describe all those technologies next.

At this point, it is worth mentioning the benefits of this application that will arise to the user himself. The application could be used for a psychological aid of the user as it creates a feeling of security, where through real data the user can see the number of people who are sick in the same period of time as him through a wealth of information such as their geographical location, their age and their gender.

Review Overview

It is a fact that since the beginning of the Covid-19 pandemic, the main priority of both the medical community and the experts, as well as the political community, has been to stop the rate of transmission of the virus and furthermore to make an early detection of the covid cases.

First of all, in order to stop the spread of the virus, the government decided that the appropriate solution was social distancing in order to maintain needless contacts among citizens. As a result, technology played a major part towards this direction, with the development of new applications in order to prevent the consequences of the Corona virus. When the development of the current application begun, the applications that already existed did not have the ability to address key issues of corresponding data, thus this interest quickly turned to creating new applications. For this reason, the interest quickly turned towards the creation of new applications.

Below we will present some similar illustrative applications.

Aarogya Setu App

Aarogya Setu is a governmental smartphone application that the Indian government created to track the COVID-19 epidemic and control its growth. Both GPS and Bluetooth are used by the application for tracking. The user's location is tracked via GPS in connection to other smartphone users who have registered as infected and are running the same application. This software employs Bluetooth technology to communicate with neighbouring devices. Additionally, this software offers extra features like self-assessment tests, test reports, electronic travel permits, information on preventative measures, online consultations, etc. The software has informed over 1.4 million users about the potential for contamination and has assisted in the creation of 697 coronavirus hotspots across the nation. Some privacy issues, such as information collecting, purpose limitation, data storage, institutional divergence, a lack of law, openness, and auditability, have been brought up in respect to the usage of the data. Following the government's response to these issues, the majority of people are now using the app.

COVID Symptom Study App

Researchers from King's College London Guys and St Thomas' Hospitals and the health technology firm Zoe Global Limited created the COVID Symptom Study application. This application's goal is to analyse the virus's transmission by detecting high-risk places in the UK, rating socially vulnerable groups, and comprehending how symptoms relate to underlying medical concerns. The purpose of this app is to gather information from users in order to support COVID-19 research in the UK and improve preventive measures. It does not offer any information or medical advice. People can willingly take part in the study and everyday share two different sorts of information. The first section is dealing with general information, including age, health, and any underlying illnesses; the second section deals with symptoms. More than 4 million people in the UK are freely providing data to the application, which has contributed to the development of an effective database for the analysis of COVID-19-related data.

Tawakkalna App

Tawakkalna is the official smartphone app authorized by the Saudi Arabian Ministry of Health. This application's goal is to make it easier for people to move around in an emergency during lockdown and prohibition. The tool also offers details about the COVID-19, such as the total number of infections throughout various regions. In the event of an emergency during lockdown, Saudi citizens may use the program to acquire travel authorizations. The program also alerts users when they are near hazardous or infectious areas. It is a mock-up of a Chinese app that employs a color-coded QR to represent the user's status. Green indicates that the person is in good health and is allowed to

travel. When someone's status is yellow, it means they are immobile because to COVID-19 suspicion. The colour red denotes an infectious person who must remain in quarantine and is not permitted to travel.

BeAware Bahrain

The Information & eGovernment Authority (iGA) and the National Taskforce for Combating the Coronavirus worked together to create the official mobile app for Android and iOS, named BeAware Bahrain (COVID-19). By using contact tracking initiatives to locate and stay updated on all active instances and their contacts, the program seeks to limit the spread of COVID-19. Additionally, it tracks quarantine cases' movements for a period of 14 days using location information provided by the public to warn people if they come too close to an active case or an area where an active case has been. Additionally, it provides daily updates on COVID-19 developments throughout the world and health advice. According to iGA Chief Executive Mohammed Ali Al Qaed, the program uses a GPS Tracking Bracelet that cannot be tampered with to provide real-time tracking data to medical professionals. Health professionals are alerted when quarantine cases depart their predetermined radius by 15 meters, at which point the team will remind people of the significance of adhering to protocols to protect the wellbeing of citizens and residents.

User's Manual

Opening the application, we see the Login screen (**Figure 1**). It consists of an edit text where the user must enter the email and a second one to enter the password. It also has two buttons, one to proceed to login and one to transfer to the Registration screen.

If you already have an account you can login with your credentials, either you can create a new one by clicking on the Register button.

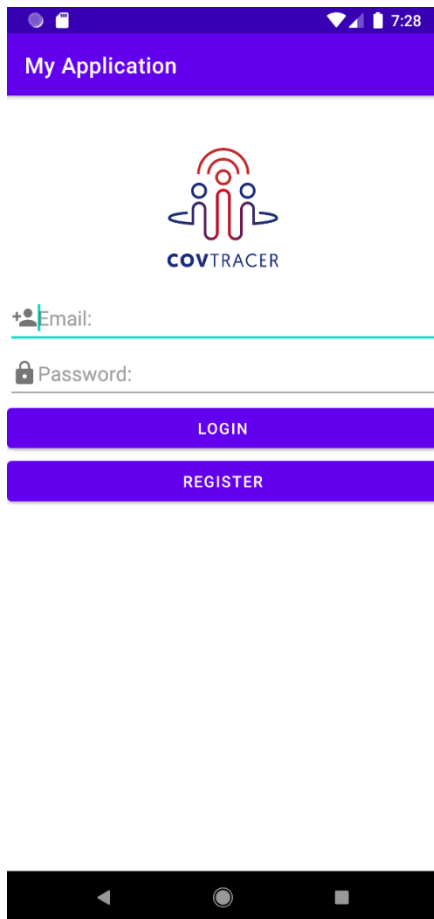


Figure 1: Login Form

In case the user tries to login without entering any credentials, an error message pop-up saying: "Email or password can not be empty" as we can see in **Figure 2**.

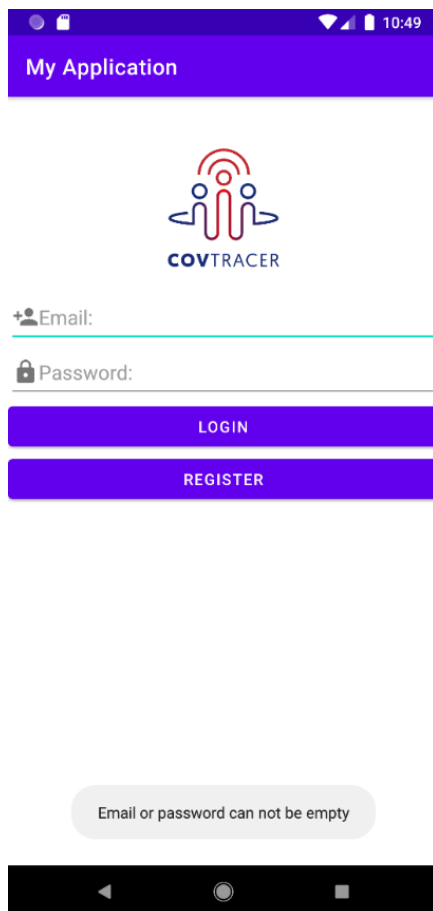
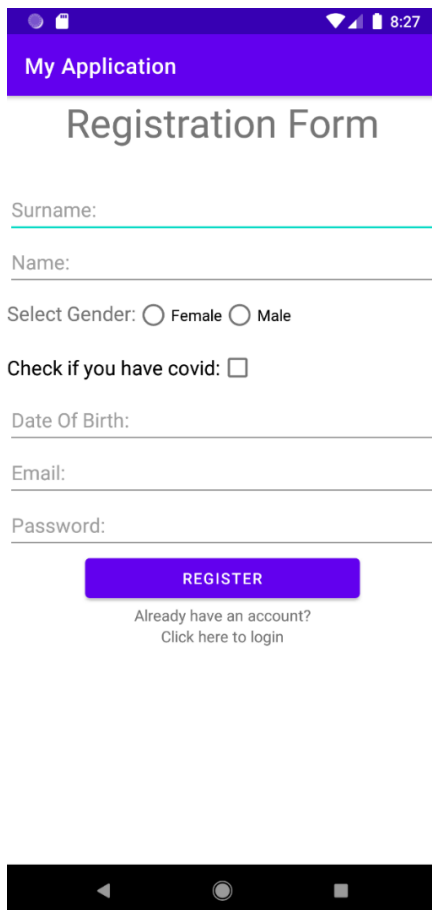


Figure 2: Login Form – error pop up message

On the Registration Form (**Figure 3**) there are textboxes for the user to enter the surname, name, date of birth, email and password, a radio button to select the gender and a checkbox to state whether he is infected with the covid-19 virus or not.

Also, there is a Register button, which performs the registration of the new user to the database and then redirects you to the Login form.

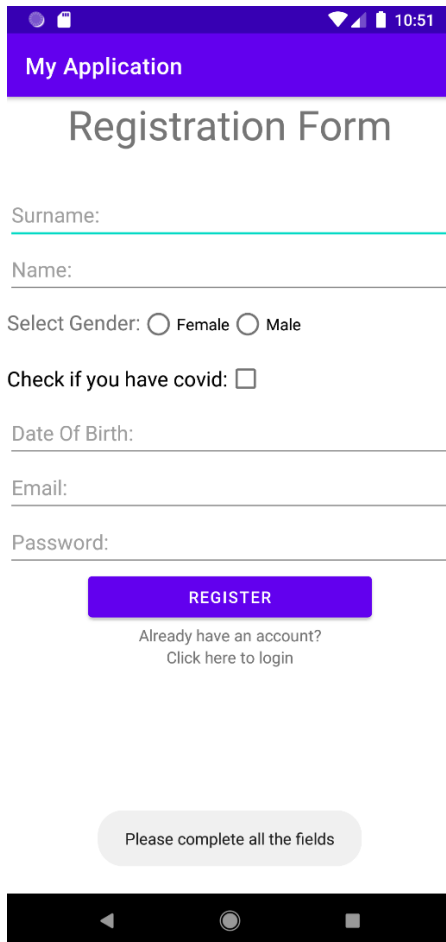
Finally, there is a TextView which indicates to "Click here to login" if you already have an account and it redirects you to the Login form.



The screenshot shows an Android application interface. At the top, there is a purple header bar with the text "My Application". Below this, the title "Registration Form" is displayed in a large, bold, grey font. The form consists of several input fields: "Surname:", "Name:", "Date Of Birth:", "Email:", and "Password:". Below the "Name" field, there is a "Select Gender:" label with two radio buttons labeled "Female" and "Male". Below the "Date Of Birth" field, there is a "Check if you have covid:" label with an unchecked checkbox. At the bottom of the form, there is a blue button labeled "REGISTER". Below the button, there is a link that says "Already have an account? Click here to login". The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

Figure 3: Registration Form

In case the user press the register button without entering any values in the fields of the Registration form a pop-up message appears at the bottom of the screen indicating to complete all the fields ("Please complete all the fields") as we can see in **Figure 4**.

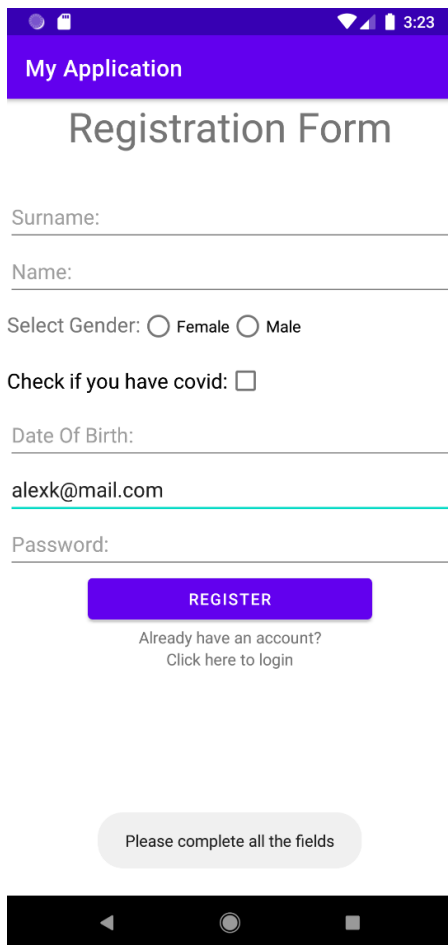


The screenshot shows a mobile application interface for a registration form. At the top, there is a purple header with the text "My Application". Below the header, the title "Registration Form" is displayed in a large, bold font. The form consists of several input fields: "Surname:", "Name:", "Date Of Birth:", "Email:", and "Password:". Below the "Name" field, there is a "Select Gender:" section with radio buttons for "Female" and "Male". Below the "Date Of Birth" field, there is a checkbox labeled "Check if you have covid:". A prominent purple "REGISTER" button is located below the form fields. Underneath the button, there is a link that says "Already have an account? Click here to login". At the bottom of the screen, a grey rounded rectangle contains the error message "Please complete all the fields". The Android navigation bar is visible at the very bottom.

Figure 4: Registration Form – error pop up message 1

In the same way as above, if the user only fills some of the required fields and press the Register button, again a pop-up message appears at the bottom of the screen indicating to complete all the

fields ("Please complete all the fields") as shown in Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε..



The screenshot shows a mobile application interface for a registration form. At the top, there is a purple header with the text "My Application". Below the header, the title "Registration Form" is displayed in a large, bold font. The form contains several input fields: "Surname:", "Name:", "Date Of Birth:", and "Password:". There are also radio buttons for "Select Gender: Female Male" and a checkbox for "Check if you have covid:". The email address "alexk@mail.com" is entered in the email field. A purple "REGISTER" button is located below the form fields. Below the button, there is a link that says "Already have an account? Click here to login". At the bottom of the form, there is a grey rounded rectangle containing the text "Please complete all the fields". The bottom of the screen shows the Android navigation bar.

Figure 5: Registration Form – error pop up message 2

While the user fills out the required fields, when comes the time to complete the “Date of birth” field, a calendar window pops-out, which gives you the ability to select a specific date . Please refer to **Figure 6**.

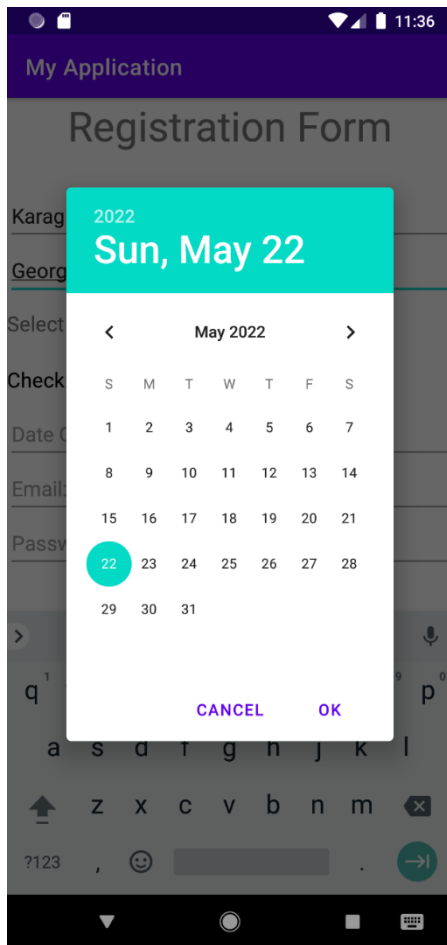


Figure 6: Calendar

In this pop-out calendar (**Figure 7**), if you click on the date on the top of the window, it leads you on a spinner mode to select the year of your choice. This is helpful to reach a date in a quicker way.

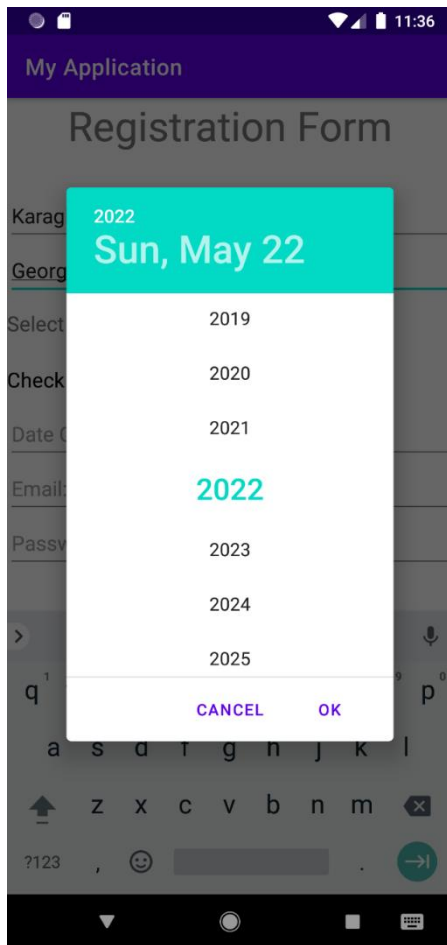


Figure 7: Calendar – Spinner mode

So, for our example we are going to pick 8 October 1992 (**Figure 8**), in the way explained above and then press OK. If we press on Cancel the calendar window will close without saving our choice.

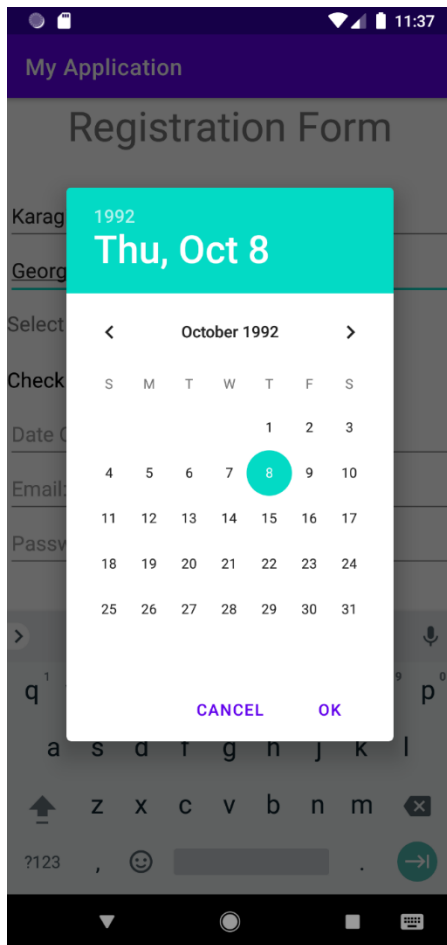
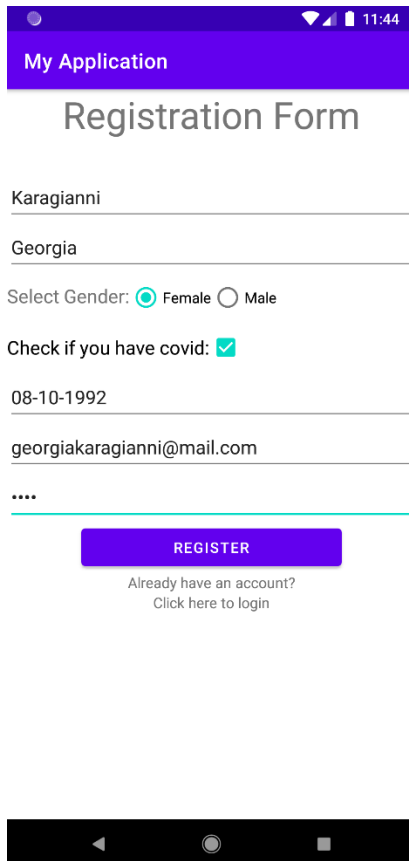


Figure 8: Calendar – example

For our example, we also complete the rest of the required fields. In the field "Surname" we typed "Karagianni" (**Figure 9**). In the field "Name" we typed "Georgia". In the field "Select Gender" we chose "Female". We checked the field "Check if you have covid". In the field "Date Of Birth" we chose "8 October 1992". In the field "Email" we typed "georgiakaragianni@mail.com". Finally, in the field "Password" we typed a 4-digit password which appears encrypted with bullets for higher password security.

After answering all the required fields we can click on the Register button.



My Application

Registration Form

Karagianni

Georgia

Select Gender: Female Male

Check if you have covid:

08-10-1992

georgiakaragianni@mail.com

....

REGISTER

Already have an account?
Click here to login

Figure 9: Registration Form - example

As we already mentioned, if we answer all the required fields and press the Register button, a new entry is created in the database and the user gets redirected to the Login form (**Figure 10**).

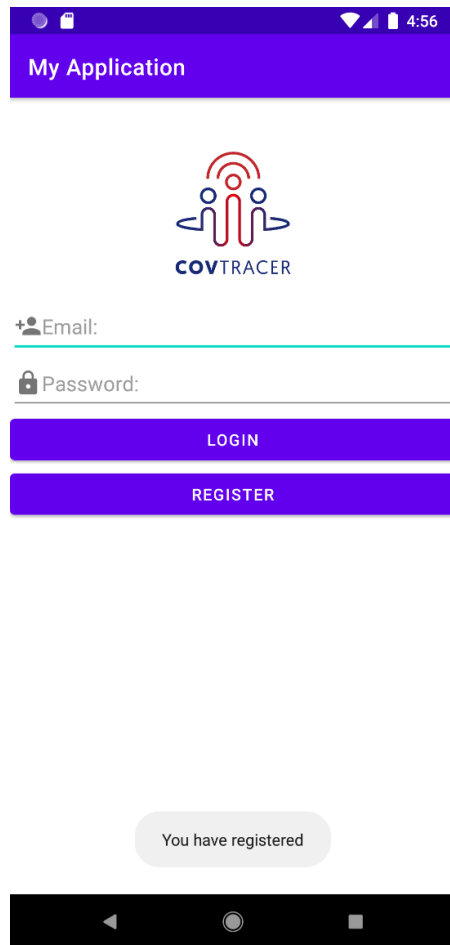


Figure 10: Successful Registration pop up

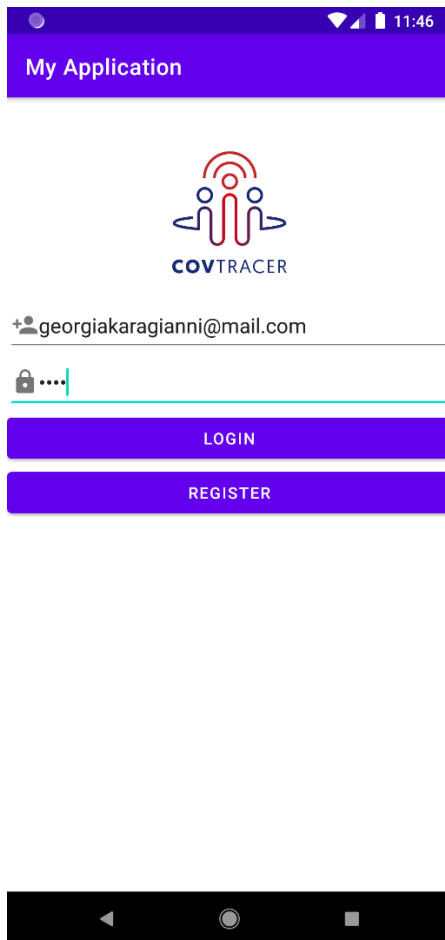
In **Figure 11** we can see a screenshot from the database after the addition of the new user. As we can see, the registry of the new user appears in the last line of the table with all the details we provided in the previous step and it was automatically assigned with the person id number 12.

```

person=# table person;
id | covid | dob | gender | name | password | surname | username
-----+-----+-----+-----+-----+-----+-----+-----
 3 | t | 1995-02-18 | Male | Alex | $2a$10$1rTfK1PEj6ZNOGMkvDjDR.N3Uid5tJgtkVgm0nGC8WnG15wiRIG. | K | alexk@mail.com
 4 | f | 1991-07-28 | Female | Maria | $2a$10$pDYD1AP37MJ0eGt5zpM.NHimYF3BezSSKcnCV0IcasMPC2FUfq | K | mariak@mail.com
 5 | t | 1992-10-08 | Female | Georgia | $2a$10$Bu98oUifA5qA3Uo3NC5RR0hnrnAc.gyHVxHM3uG5xp/0QwQhVF.g. | K | gk@mail.com
 6 | t | 1966-02-14 | Female | Katerina | $2a$10$CRsTNZmQ164pHs1yyLXLD.MtETFQpDq.0AqtVzUjIoT/OfXHIu4aW | K | katerinak@mail.com
 7 | f | 1961-01-05 | Male | Dimitris | $2a$10$ESU0t439yzukbZp08x1t/.ZnR5kRASHvFKZ12J2LI2N1pHd1.v1.e | K | dimitrisk@mail.com
 8 | t | 1992-06-30 | Male | Antonis | $2a$10$kb2VaA5gH.DjQg2ZUVqqvuHktk9lpTrpIoxuTbtcd2rNxoLLMx8u | N | antonism@mail.com
 9 | t | 1992-10-16 | Female | Areti | $2a$10$EnB3YcIyfaSTotBPL4RtF.R/Q1EE0Xsy0GVVTs64TF0yhrDjW/CjW | K | aretik@mail.com
10 | t | 1992-01-08 | Male | Kostas | $2a$10$20o6QNsCFbKvuduGnrDDt0HmZzNBZLYFNrQL1t8k2vqU/GpRVNYra | K | kostask@mail.com
11 | t | 1993-05-31 | Male | George | $2a$10$LKCXIDjccSTea1GrJYhV.eLtf02o5C2Qf..XhgyHAXhSY1QdFvZwK | G | georgeg@mail.com
12 | t | 1992-10-08 | Female | Georgia | $2a$10$EUISfZb86sNg1.r2ojfPmeUkZnf87BNsiTcngVxJCEu.UPnglW592 | Karagianni | gewrgiakarayianni@gmail.com
(10 rows)
    
```

Figure 11: Database – new record

After the registration of the new user, we complete the required fields "Email" and "Password" with the correct credentials and press the button Login. Please refer to **Figure 12**.



The screenshot shows the login form of the COVTRACER application. At the top, there is a status bar with the time 11:46 and icons for signal, Wi-Fi, and battery. Below the status bar is a purple header with the text "My Application". The main content area features the COVTRACER logo, which consists of a red and blue icon of two people with a signal wave above them, and the text "COVTRACER" below it. Below the logo is an email input field containing the text "georgiakaragianni@mail.com". Below the email field is a password input field with a lock icon and a red line indicating a password strength indicator. Below the password field are two purple buttons: "LOGIN" and "REGISTER". At the bottom of the screen is a black navigation bar with three icons: a back arrow, a home circle, and a recent apps square.

Figure 12: Login Form – entering crede

As the credentials that were entered were correct, the user gets redirected to the Welcome page, and a pop-up message informs him for the successful login. Please refer to *Figure 13*.

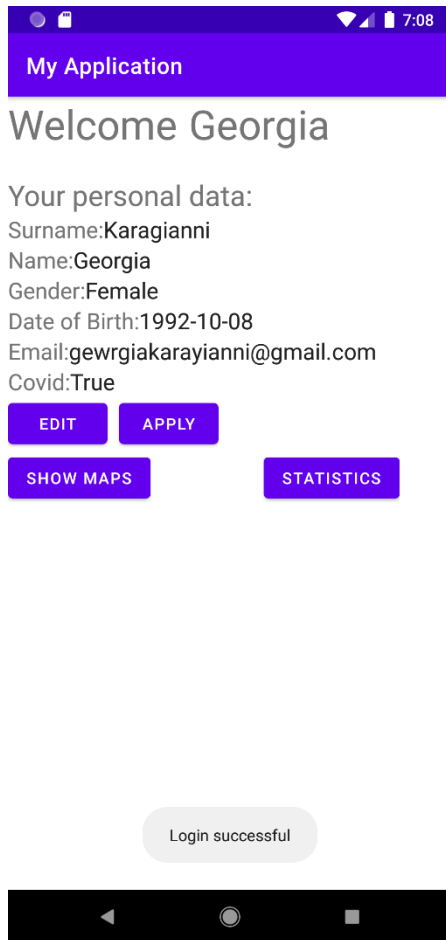


Figure 13: Welcome Form – successful login

As the application redirects the user to the Welcome page, he gets a greeting with his name and he can see the personal information he entered during the registration phase (**Figure 14**). Also, there are 4 buttons: Edit, Apply, Show Maps and Statistics. We will go through those buttons next.

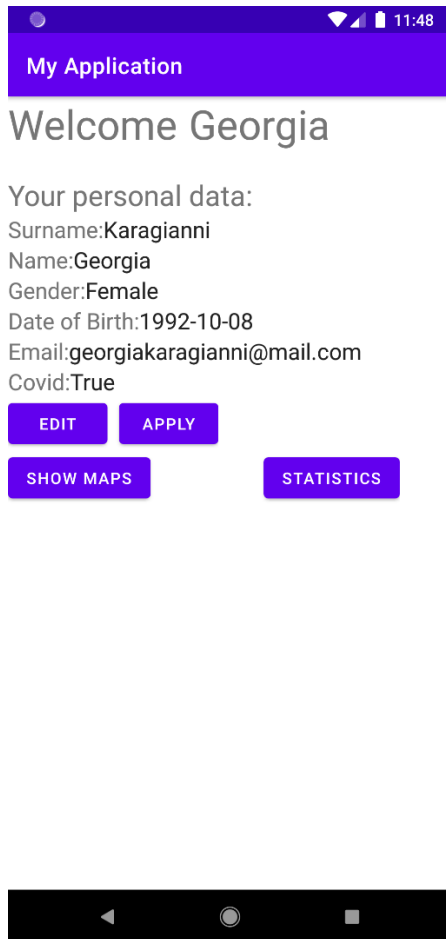


Figure 14: Welcome Form

The Edit button gives the ability to the user to change the state of the Covid field (**Figure 15**). This is useful because maybe the user was infected with Covid but now has recovered. Or maybe he was not infected but now has a positive result. So, the user can change the answer for True to False and vice versa.

Potentially, all the fields would be edited same as the user's covid status, but it was not deemed necessary for the needs of the present Thesis.

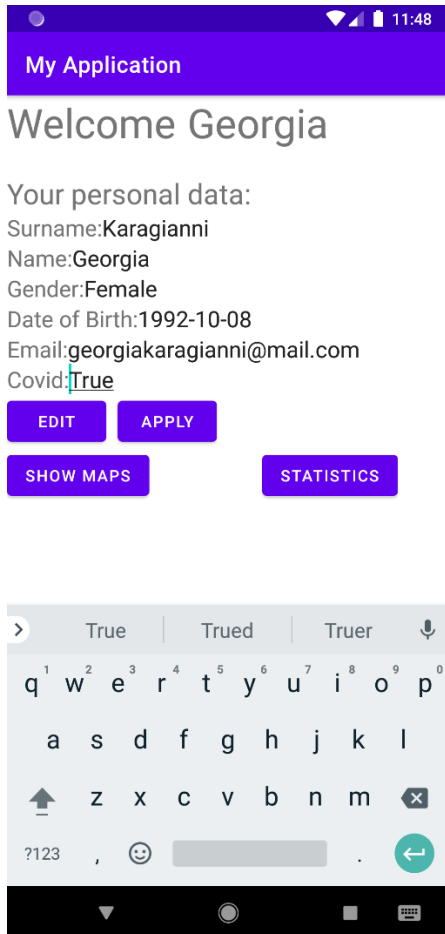


Figure 15: Edit button

In our example, the initiative answer was “True” and we changed it to “False” as we can see in **Figure 16**. Then we press the button Apply to save our answer in the database.

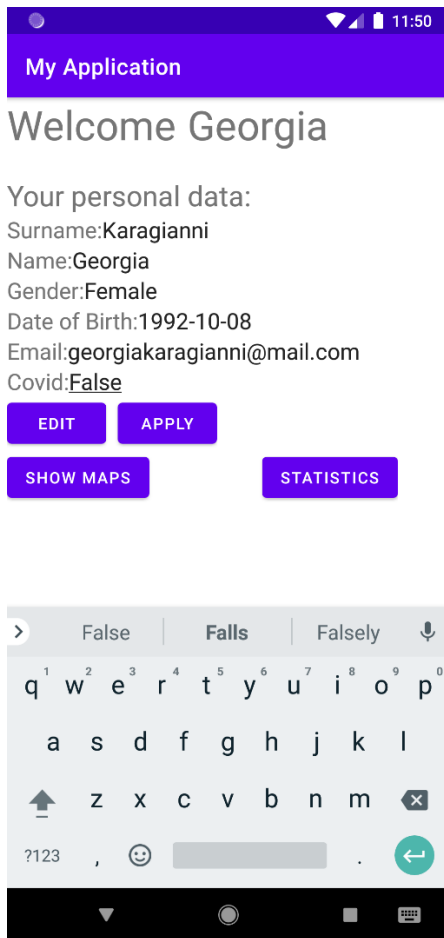


Figure 16: Covid status change

In **Figure 17** we notice that the change we made is properly stored in the database. The change made is in the last line of the covid column, with the letter “f” for false. (See the database state before this change in **Figure 11**).

```
person=# table person;
id | covid | dob      | gender | name      | password                                                                 | surname | username
-----+-----+-----+-----+-----+-----+-----+-----
 3 | t     | 1995-02-18 | Male   | Alex      | $2a$10$daL_B/RV2fSd1460SnaJUwPirA0/OXiuLwB4nuyFnK/iid.D4PRCO | K       | alex@mail.com
 4 | f     | 1991-07-28 | Female | Maria     | $2a$10$tNEVP9UqjBY1YiRcr4bxAekdP1QtGELsbTsB60inUPNIznnfNnuSy | K       | mariak@mail.com
 5 | t     | 1992-10-08 | Female | Georgia   | $2a$10$EKndbvDd6JezJqVyleF1v.7XLR8nyJ9kluIzfIcygZRhcZiWk.WA1 | K       | gk@mail.com
 6 | t     | 1966-02-14 | Female | Katerina  | $2a$10$Gbao3sBT0ailrFogaIXum.ROMjCdHHUrarfx.hicuiPdF13mcXVGO | K       | katerinak@mail.com
 7 | f     | 1961-01-05 | Male   | Dimitris  | $2a$10$Z/u56rJusNFQPS.11RRkW.C2uA87aOetrm05VlFfNubj.eIcm.v4. | K       | dimitrisk@mail.com
 8 | t     | 1992-06-30 | Male   | Antonis   | $2a$10$4b7rxbB/7E/XoVLvI92zPu3kwXY4Q7ZF1JUF1ktv/sGUG01q84i1i | N       | antonisn@mail.com
 9 | t     | 1992-10-16 | Female | Areti     | $2a$10$bjiMhotFCd3GXVsIhy1L5w.6yGdtG4Ys/5cZgRv0MZrM.vHLFVdyGi | K       | aretik@mail.com
10 | t     | 1992-01-08 | Male   | Kostas    | $2a$10$BQSmyprodSEn9nkMxrsKKeZQTMo0hIvH5XBjvZBy1hvuZ1YnlW1P1G | K       | kostask@mail.com
11 | t     | 1993-05-31 | Male   | George    | $2a$10$gvJ5wmlNadv1sLGS5u6e0uwHIRmcPaA/mGGYT8CFmz.aDbHDn6T6u | G       | georgeg@mail.com
12 | f     | 1992-10-08 | Female | Georgia   | $2a$10$w3SVIU.77Rocl7gKhi/0NOYHAL7heHKD4QalRqTG7hFJwH9Bsc0PS | Karagianni | georgiakaragianni@mail.com
(10 rows)
```

Figure 17: Covid status – database change

By pressing the button “Show Maps” the user gets redirected to a Google Maps page. Here, if we press the target button under the search button, the map will show our location. In our example, our location appears to be Googleplex, because we run the application through a PC emulator. If you run the application in a smartphone, it will show the user's current location. See below **Figure 18**.

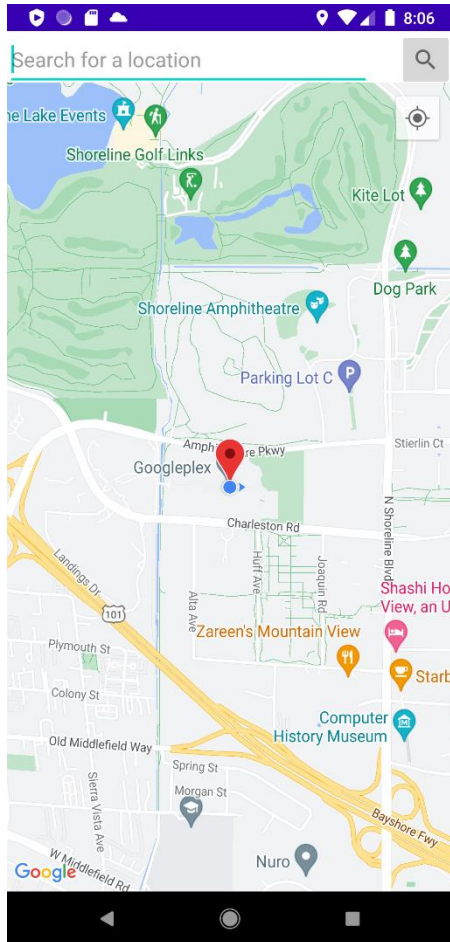


Figure 18: Maps Form

Now, if we type in the search bar a location, Google Maps will show this location and will demonstrate a location mark. Here, we searched for Athens and we can see the result in **Figure 19**.



Figure 19: Maps Form – location search

By clicking the button Statistics, it appears the page with various statistical schemes. The developer can add any sort of statistical schemes and with any population selection depending on the application needs.

Here we have chosen randomly two cases to study. The first one is a chart pie which depicts the percentage of Covid cases per gender. The second one is a horizontal bar chart which indicates the cases of male persons, over the age of 20, infected with Covid, divided per country. Please refer to **Figure 20**.

Every time a change is being made to the database, the graphs adjust the results automatically. Of course, the graphs are indicative for the purposes of this Thesis and you can add any case you might find intriguing or interesting in order to derive a result.

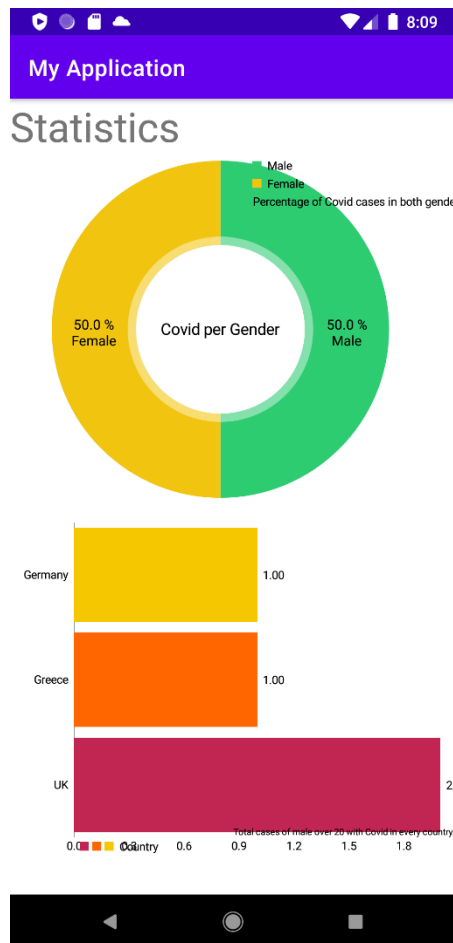


Figure 20: Statistics Form

System's Architecture

Spring Framework

The Java platform has an application framework and inversion of control container called the Spring Framework. Any Java application can use the framework's core functionality, however there are extensions available for creating web applications on top of the Java EE (Enterprise Edition) platform. The framework has gained popularity in the Java community as an extension to the Enterprise JavaBeans (EJB) architecture, despite the fact that it does not impose any particular programming model. The Spring Framework is an open source app.

Spring Boot

Spring Boot is a framework for building easily stand-alone applications. Whether you want to build back-end applications or full-stack applications using Java or Kotlin, Spring Boot is the application to go. Spring Boot is by far the most popular framework right now. If you need security, you can use security modules available. If you need logging, you can use the login integration connecting to databases, whether you want to connect to MongoDB, PostgreSQL, MySQL, they made it super easy for you to connect to any database. It also includes metrics for checking how your application is behaving in production. Also, it is a production-ready framework, which means that you can build microservices. Furthermore, it has dependency injection build-in, configuration, great community, and lots of other features.

Features

- Develop independent Spring applications.
- Directly embed Tomcat, Jetty, or Undertow with no need to deploy WAR files
- To make your build configuration simpler, provide opinionated "starting" dependencies.
- Whenever possible, configure Spring and third-party libraries automatically.
- Offer features that are ready for production including metrics, health checks, and externalized configuration
- Neither code generation nor XML configuration are necessary.

Last but not least, in order to help you get started quickly, we take an opinionated stance on the Spring platform and third-party libraries. It is also worth mentioning that most of the Spring Boot applications require minimal Spring configuration.

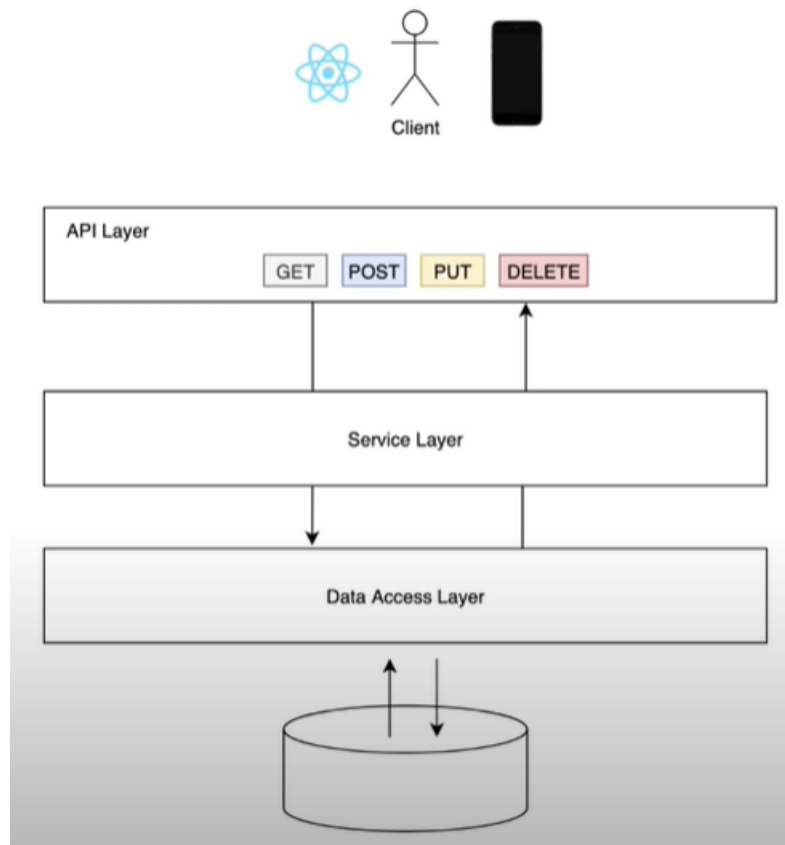


Figure 21: Spring Boot Architecture

We are going to build this entire application, excluding the front-end part, where we are going to have the API. So, the API will receive GET request, POST, PUT and DELETE. And then we will have a service layer, this is mainly for business logic, then we will have a data access layer, and this layer is responsible for connecting to any database.

So, we have our API Layer, which should talk to the Service Layer to get some data, and that Service Layer should also talk to the Data Access Layer to get the data. So, it does a round trip, from the client, API, Service, Data and then all the way back. This flow of data can be seen in **Figure 21**.

Spring Initializr

Spring Initializr generates Spring Boot projects with just what you need to start quickly.

The Pivotal Web Service offers a web-based tool called Spring Initializr. The framework of the Spring Boot Project can be readily generated with the aid of Spring Initializr. It provides a flexible API to build JVM-based programs.

Additionally, it gives the project a number of options that are defined in a metadata model. The JVM's supported dependencies list, platform versions, and other settings can all be configured via the metadata model. It offers third-party clients the necessary support by serving its metadata in a well-known manner.

The link for the site of Spring Initializr is the following: <https://start.spring.io/>

In this website is where we can bootstrap any given Spring Boot application.

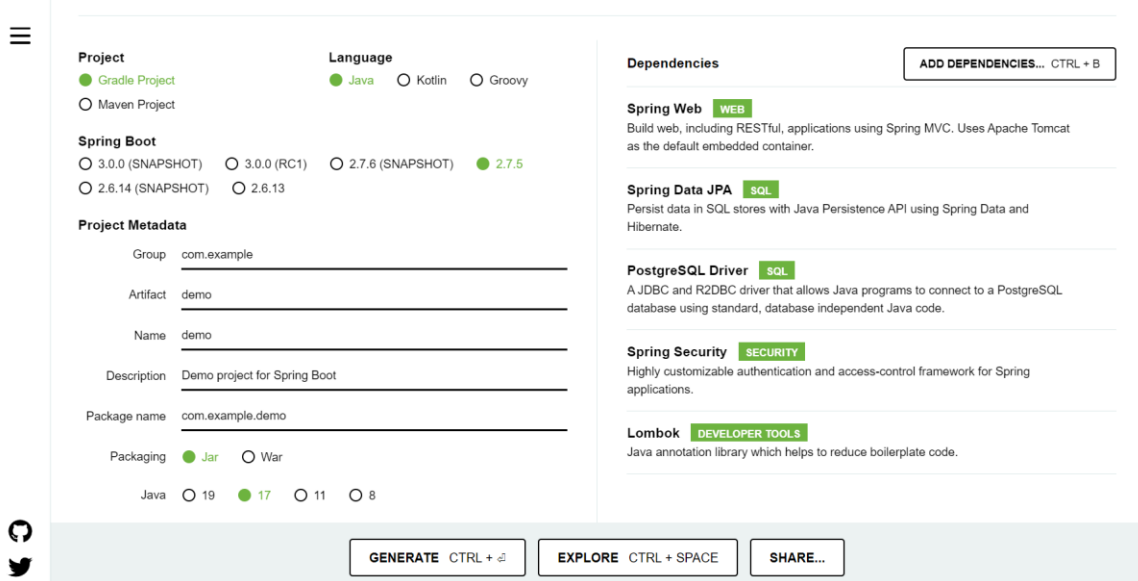
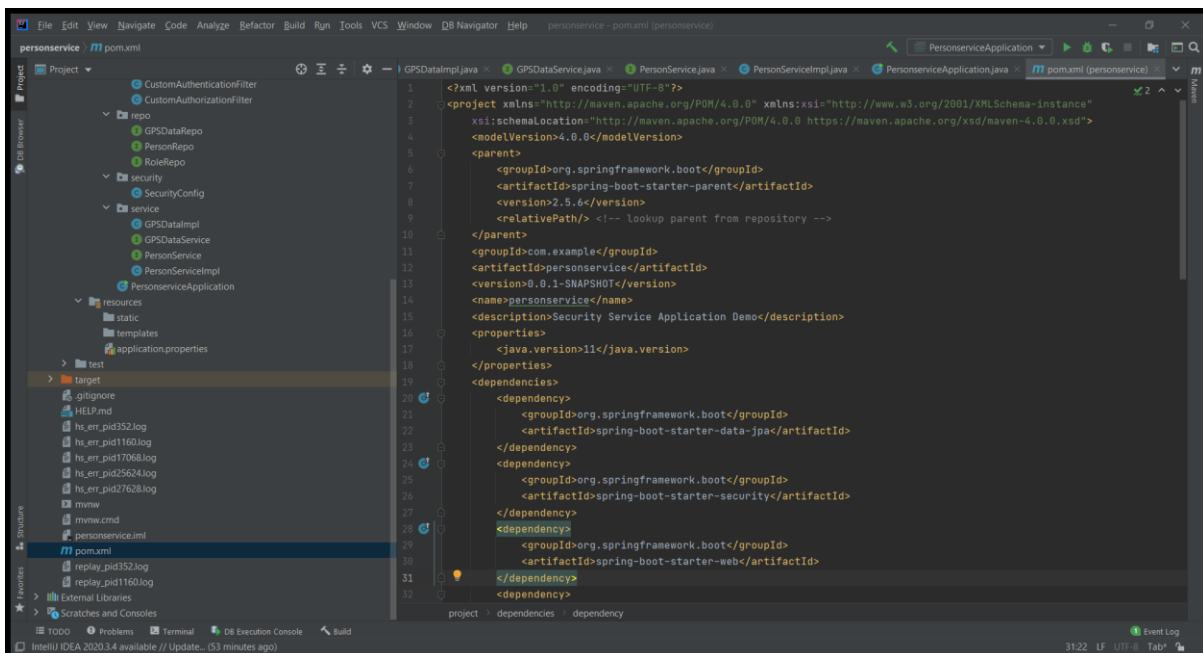


Figure 22: Spring Initializr

As we can see in **Figure 22**, we can pick between Maven and Gradle, so we are going to keep Maven. We can also choose the language: Java, Kotlin or Groovy. We are going to stick with Java. And then we can pick any version. We picked version 2.4.4 which was one of the options by the time we started building the application. Also, as we can see, we can customize the project metadata. For packaging we are going to select Jar, which is the most common packaging type for Java application. For Java version we have 15 installed, so we chose version 16 as it was one of the options by the time we started building the application.

After that we are going to select Dependencies. In here we can pick dependencies that our project needs. There is a quite huge list. For this application we are going to select Spring Web, Spring Data JPA, PostgreSQL Driver, Spring Security and Lombok. Finally, we are going to click on Generate button. We are going to open the generated extract with IntelliJ.

After creating the project, we can spot those dependencies at the pom.xml file (**Figure 23**) in IntelliJ.



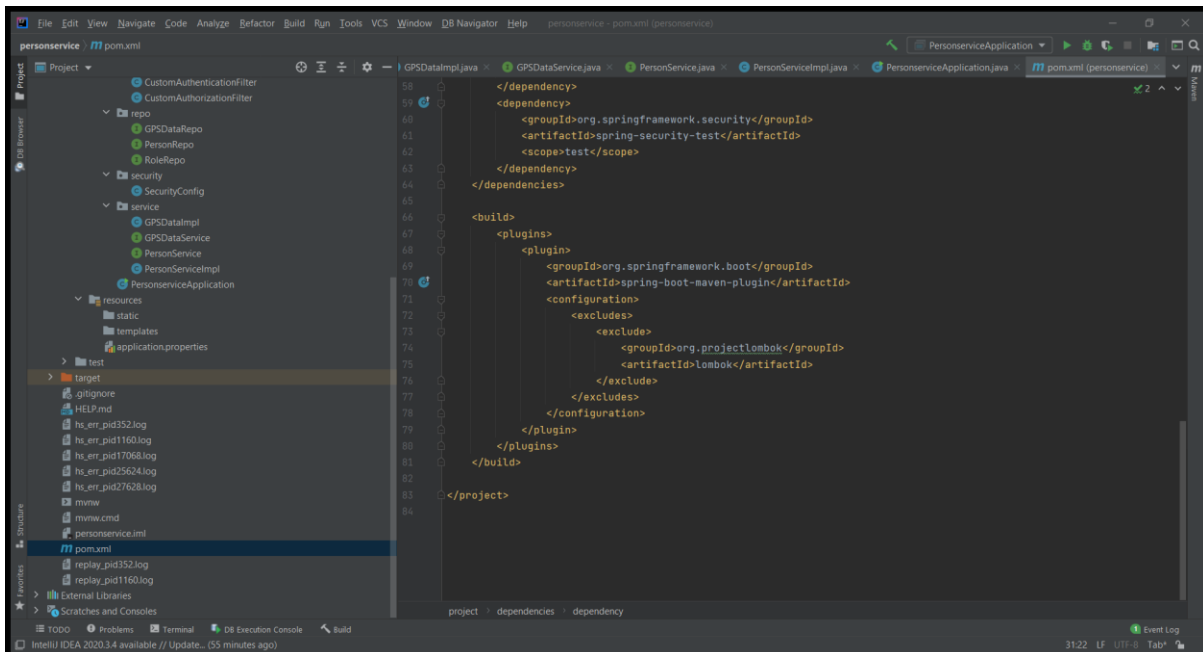
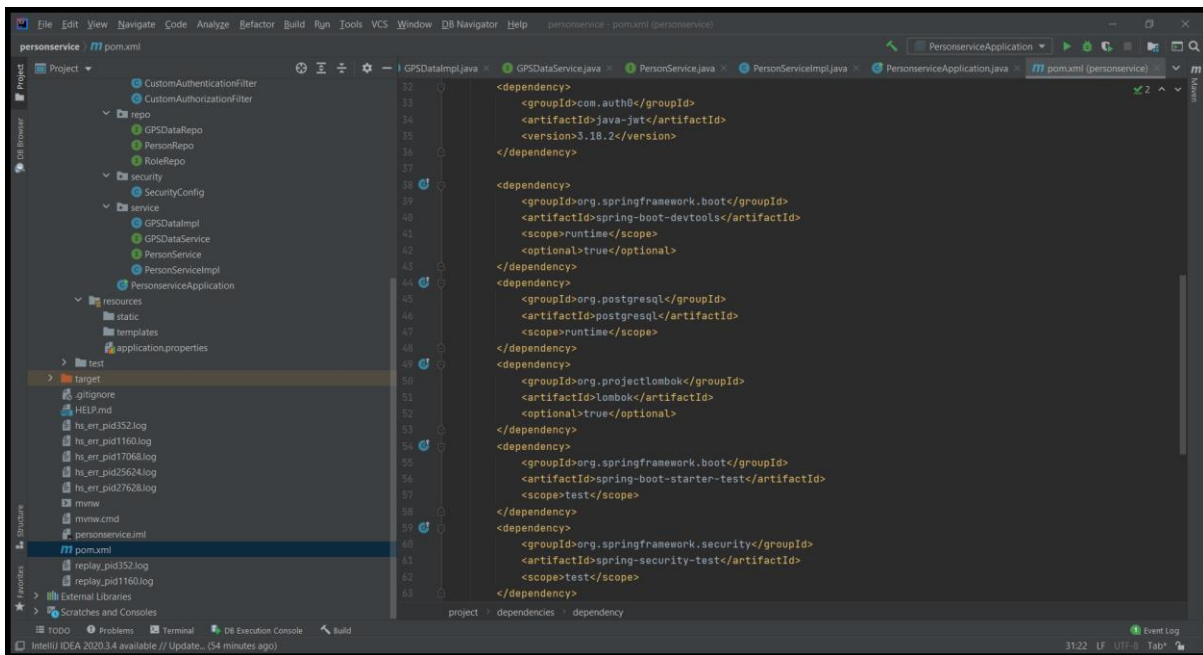


Figure 23: pom.xml

Below in **Figure 24** we can see the classes and packages that were created for the needs of the specific project:

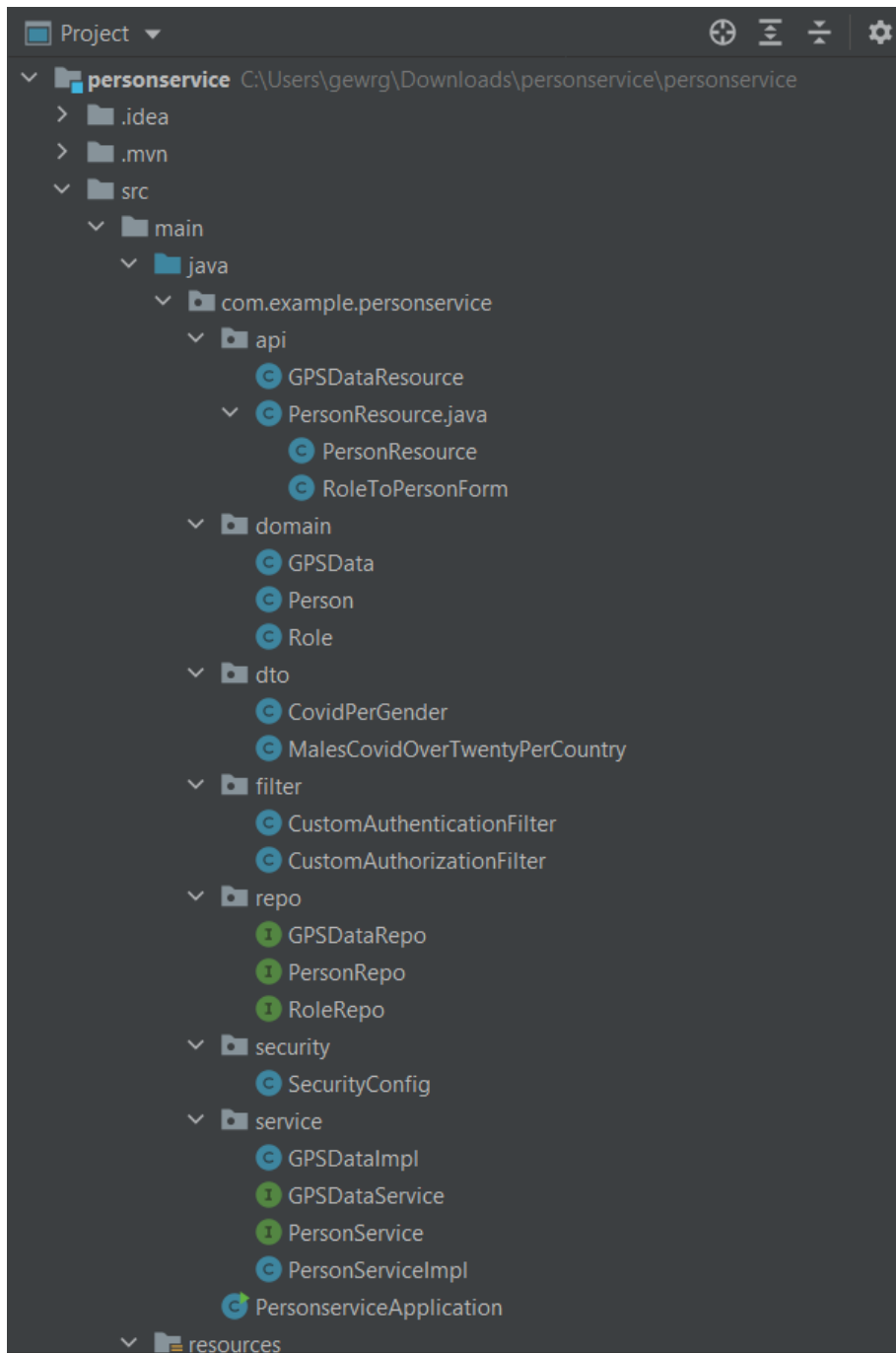


Figure 24: Project structure

As we can see in the image above, the packages and the corresponding, within them, classes have been created in such way that the use of all three levels of a Spring application is clear. These levels are: the Controller Layer corresponding to the package named *api* and the classes *GPSDataResource* and *PersonResource*, the Service Layer corresponding to the package named *service*, the interfaces *GPSDataImpl*, *GPSDataService*, *PersonService* and *PersonServiceImpl* and finally the Data Access Layer corresponding to the package named *repo* and the interfaces *GPSDataRepo*, *PersonRepo*, *RoleRepo*.

Spring Security OAuth

Standard Spring and Spring Security programming models and configuration idioms are supported when utilizing Spring Security with OAuth (1a) and OAuth2.

Features

- Support for OAuth providers and OAuth consumers
- Oauth 1(a) (including two-legged OAuth, a.k.a. "Signed Fetch")
- OAuth 2.0

OAuth is not the exception to the rule that applying security to applications needs a lot of effort and the respective attention. You should make sure you comprehend OAuth and the issue it is intended to solve before you get started. The OAuth website has helpful documentation, which will make sure you comprehend how Spring and Spring Security function as well.

To get the most out of the developer's guide, you should be very familiar with both OAuth (and/or OAuth2) and Spring Security. Both technologies are closely related to OAuth for Spring Security, thus the more familiar you are with them, the more likely you will be to understand the terminology and design patterns.

Spring Data JPA

Implementing JPA-based repositories is simple thanks to Spring Data JPA, which is a member of the larger Spring Data family. The topic of this module is improved support for JPA-based data access layers. It makes it simpler to develop applications that leverage data access technologies and are driven by Spring.

An application's data access layer has long been difficult to implement. To implement basic operations like pagination, auditing, and simple query execution, too much boilerplate code must be created. By focusing on the real effort required, Spring Data JPA seeks to greatly enhance the implementation of data access layers. Developers create their repository interfaces, including unique finder methods, and Spring takes care of the implementation for them.

Features

Pagination support, dynamic query execution, and the ability to integrate custom data access code are some of the sophisticated features that help developers create repositories based on Spring and JPA. Furthermore, type-safe JPA queries are made possible via support for Querydsl predicates, and domain class auditing is transparent. Other features include bootstrapping validation of @Query annotated queries, support for XML-based entity mapping, and JavaConfig-based repository configuration by introducing @EnableJpaRepositories. Below are listed some of these repositories.

Spring Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

PostgreSQL Driver

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Spring Security

Highly customizable authentication and access-control framework for Spring applications.

Spring Boot DevTools

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok

Java annotation library which helps to reduce boilerplate code.

IntelliJ IDEA

A Java-based integrated development environment (IDE) called **IntelliJ IDEA** is used to create applications in JAR-based languages including Groovy, Kotlin, and Java. It is created by JetBrains (formerly known as IntelliJ), and it comes in both a proprietary commercial edition and an Apache 2 Licensed community edition. Both can be applied to the development of businesses.

One of the earliest Java IDEs with built-in advanced code navigation and code refactoring features was IntelliJ IDEA, which launched its initial edition in January 2001.

Out of the four leading Java programming tools—Eclipse, IntelliJ IDEA, NetBeans, and JDeveloper—InformationWorld reported in 2010 that IntelliJ obtained the highest test center score.

Version 1.0 of Android Studio, an open-source IDE for Android apps built on the open source community edition of IntelliJ IDEA, was released by Google in December 2014. AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm, and MPS are additional programming environments that use IntelliJ's framework.

The IDE offers capabilities including context-sensitive code completion, code navigation that allows moving to a class or declaration in the code directly, code refactoring, code debugging, linting, and options to repair inconsistencies via suggestions.

Integration with build/packaging technologies like grunt, bower, Gradle, and SBT is achieved by the IDE. Git, Mercurial, Perforce, and SVN are just a few of the version control systems it supports. The Ultimate edition's incorporated DataGrip allows users to connect directly to databases including Microsoft SQL Server, Oracle, PostgreSQL, SQLite, and MySQL from within the IDE.

PostgreSQL

A free and open-source relational database management system (RDBMS) with an emphasis on flexibility and SQL conformance is PostgreSQL, also referred to as Postgres. As a replacement for the Ingres database created at the University of California, Berkeley, it was initially known as POSTGRES. Because it supported SQL, the project's name was changed to PostgreSQL in 1996. The development team opted to preserve the names PostgreSQL and Postgres following a review in 2007.

Requirement Analysis

The creation of the database for this work was based on the following specifications:

- There should be a "Person" table, in which the users who register in the application will be stored. This table will contain the username, password, name, surname, date of birth, gender and covid status of each user.
- There should be a table "Role", in which the role of the users will be stored.
- There should be a table named "GPSData", in which the location of the users will be stored. This table will contain the latitude, longitude, city and country of each user.

Entity – Relationship Model

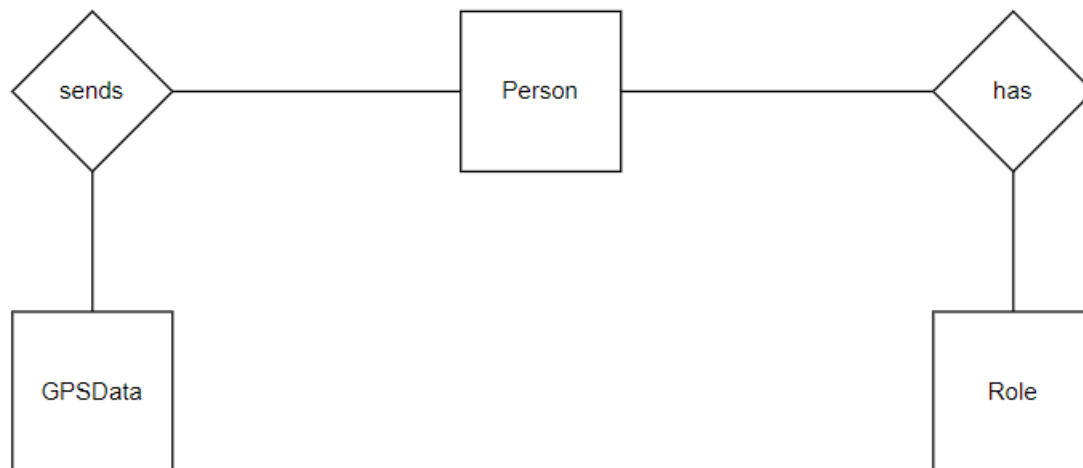


Figure 25: Entity - Relationship Model

From the diagram above (**Figure 25**), as we can see, there are three entities and two associations. One association is M:N (many – to – many) and the other one is 1:N (one – to - many).

More in detail:

Person_has_Role: the association between the Person table and the Role table is many – to - many, since a user can have one or more roles, and a role can be assigned to one or more users.

Person_sends_GPSData: the association between the Person and GPSData tables will be one – to - many, since a user can send data one or more times while a record of the GPSData table can belong to only one user.

Entity – Relationship Diagram

In the diagram below (**Figure 26**), are illustrated how the entities of the Entity – Relationship Model are related to each other within the system:

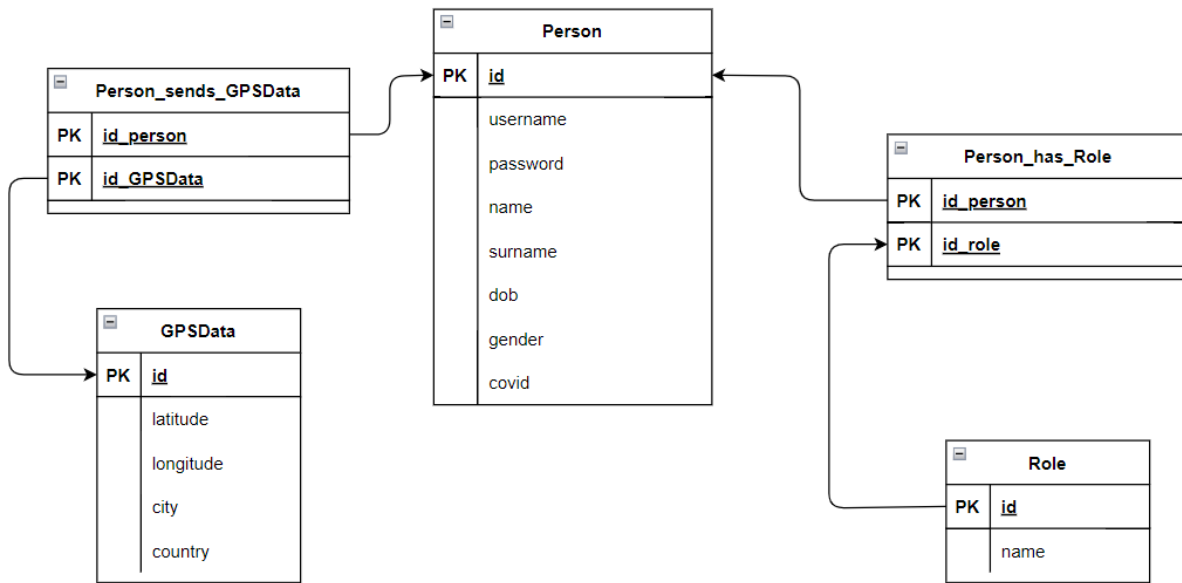


Figure 26: Entity – Relationship Diagram

Below (Figure 27) we can see a screenshot taken from the SQL Shell, which shows the list of relations of our database.

```

    List of relations
    Schema | Name      | Type  | Owner
    -----+-----+-----+-----
    public | gpsdata  | table | postgres
    public | person   | table | postgres
    public | person_roles | table | postgres
    public | role     | table | postgres
    (4 rows)
    
```

Figure 27: SQL – list of relations

The package domain with the GPSData, Role, Person classes implements and creates the database tables. The users of the application will be stored in the Person table. The users' role will be stored in the Role table. Finally, in the GPSData table will be stored the data that will be sent by each user to the server. The implementation of the Person table is shown below, in Figure 28.

```

package com.example.personservice.domain;

import ...

@Entity
public class Person implements Serializable {
    @Id @GeneratedValue(strategy = AUTO) //IDENTITY
    private Long id;
    private String name;
    private String surname;
    private String gender;
    private LocalDate dob;
    @Transient
    private Integer age;
    private Boolean covid;
    private String username; //email
    private String password;
    @ManyToMany(fetch = EAGER)
    private Collection<Role> roles = new ArrayList<>();
    @{...}
    private List<GPSData> gpsData = new ArrayList<>();

    public Person() {}

    public Person(Long id, String name, String surname, String gender, LocalDate dob, Boolean covid, String username, String password, Collection<Role> roles, List<GPSData> g

    public Person(String name, String surname, String gender, LocalDate dob, Boolean covid, String username, String password, Collection<Role> roles, List<GPSData> gpsData) {}

    public Long getId() {return id;}

    public void setId(Long id) {this.id = id;}

    public String getName() {return name;}
    
```

```

    public void setName(String name) {this.name = name;}

    public String getSurname() {return surname;}

    public void setSurname(String surname) {this.surname = surname;}

    public String getGender() {return gender;}

    public void setGender(String gender) {this.gender = gender;}

    public LocalDate getDob() {return dob;}

    public void setDob(LocalDate dob) {this.dob = dob;}

    public Integer getAge() {return Period.between(this.dob, LocalDate.now()).getYears();}

    public void setAge(Integer age) {this.age = age;}

    public Boolean getCovid() {return covid;}

    public void setCovid(Boolean covid) {this.covid = covid;}

    public String getUsername() {return username;}

    public void setUsername(String username) {this.username = username;}

    public String getPassword() {return password;}

    public void setPassword(String password) {this.password = password;}

    public Collection<Role> getRoles() {return roles;}

    public void setRoles(Collection<Role> roles) {this.roles = roles;}
    
```

Figure 28: Implementation of Person table

During the registration of each user into the application, they are prompted to enter the following data: username, password, name, surname, date of birth, gender and covid status.

Below we can see a screenshot taken from the database (Figure 29).

id	covid	dob	gender	name	password	surname	username
3	t	1995-02-18	Male	Alex	\$2a\$10\$xQEVcT.K6cmCtLBN79f21u/1mdNsNV4Vg55X5jI/guoBxS9UV1cf1	K	alexk@mail.com
4	f	1991-07-28	Female	Maria	\$2a\$10\$Q4TRaGti6E/FOHRfeCHsy0xXquDe40nhBknXfzERvE5Yt3HG0rqY2	K	mariak@mail.com
5	t	1992-10-08	Female	Georgia	\$2a\$10\$mb78hrWNFBaTtB4WJsChE.HWQV7fupnmX58m3Mpp7xQIYJGa5oJQ6	K	gk@mail.com
6	t	1966-02-14	Female	Katerina	\$2a\$10\$SKKV/2xsHx/PluZV1y1zLeKf0fiqXqgAICEfgfcyoY.kYZox/1s5u	K	katerinak@mail.com
7	f	1961-01-05	Male	Dimitris	\$2a\$10\$Qr1CX1hhJB0gvixCXcNSXeH5AnYQmE8FPHK/08J0t9o9mLfngDHS	K	dimitrisk@mail.com
8	t	1992-06-30	Male	Antonis	\$2a\$10\$CAhCIgGfwc.glo1nzgKFHe4LPwzSF5/uPAGX32NmbSy9kLF0haC0	N	antonis@mail.com
9	t	1992-10-16	Female	Areti	\$2a\$10\$hrUARfddzcVxdHf1bBgzI.R/4VX3H996ChpVvUtyPCW24oU6AWUF.	K	aretik@mail.com
10	t	1992-01-08	Male	Kostas	\$2a\$10\$qAZ9X1P10gInYhwOfg5N9uYk80Co9ZrdxLSHCevWuLcbHO.AowUTW	K	kostask@mail.com
11	t	1993-05-31	Male	George	\$2a\$10\$hvMWN72Quyu807Zq5Bh00qk0tWcZvnpSuscHScYNN8c92K1d5QrSS	G	george@mail.com

Figure 29: Database records

Through observation we detect 8 columns. The first column, named “id”, which is created automatically, is the primary key of the table. The second column, named “covid”, returns the user’s covid status in true or false values. The third column, named “dob”, represents the date of birth of the user. From this column we can extract the user’s age as well. The fourth column, named “gender”, shows the user’s gender and returns the values Male and Female as for the time present. The fifth column, named “name”, shows the user’s name. The sixth column, named “password”, is where the password selected by the user is stored in encrypted form. The seventh column, named “surname”, shows the user’s surname. The eighth and last column, named “username”, is where the selected by the user username is stored.

Next, we can see the implementation of the GPSData table (Figure 30).

```

package com.example.personservice.domain;

import java.io.Serializable;

public class GPSData implements Serializable {
    private Long id;
    private double longitude;
    private double latitude;
    private String city;
    private String country;
    private Person person;

    public GPSData(double longitude, double latitude, String city, String country, Person person) {
        this.longitude = longitude;
        this.latitude = latitude;
        this.city = city;
        this.country = country;
        this.person = person;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public double getLongitude() { return longitude; }
    public void setLongitude(double longitude) { this.longitude = longitude; }

    public double getLatitude() { return latitude; }
    public void setLatitude(double latitude) { this.latitude = latitude; }

    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
    }
    
```

Figure 30: Implementation of GPSData table

When the user clicks on the “SHOW MAPS” button, the GPS gets enabled and detects the user’s location. The GPS communicates with the database and the database stores the following data for each user: latitude, longitude, city and country.

Below we can see a screenshot taken from the database (Figure 31).

id	city	country	latitude	longitude	person_id
1	Volos	Greece	22.935295	39.36213	4
2	Athens	Greece	23.735777	37.973567	5
3	Kharkov	Bulgaria	25.558222	41.932371	7
4	Glasgow	UK	-4.247003	55.858917	8
5	Athens	Greece	23.74942	38.005388	3
6	Haskovo	Bulgaria	25.550045	41.932499	6
7	Edinburgh	UK	-3.197109	55.944727	9
8	Hamburg	Germany	10.101449	53.580401	10
9	London	UK	-0.118092	51.509865	11

(9 rows)

Figure 31: Database – GPSData table

Through observation we detect 6 columns. The first column, named “id”, which is created automatically, is the primary key of the table. The second column, named “city”, shows the city that the user is at the time he uses the application. The third column, named “country”, shows the country.

The fourth column, named “latitude”, shows the latitude. The fifth column, named “longitude”, shows the longitude. The sixth column, named “person_id”, is the foreign key, which shows that these data belong to the user with id equal to this number.

It should also be noted that with the use of “Spring Security” dependency, when connecting the user to the application, authentication takes place. If the authentication is successful, an access token and a refresh token are given, which are stored in the shared preferences of the android application. The user’s name and email are also stored in the shared preferences. Please refer to **Figure 32** for the implementation of these operations.

```
//methode to restore access_token and refresh_token in shared preferences
public void saveTokens(Token token){
    SharedPreferences sharedPreferences = ctx.getSharedPreferences(SHARED_PREF_NAME_TOKEN, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString(KEY_ACCESS_TOKEN, token.getAccess_token());
    editor.putString(KEY_REFRESH_TOKEN, token.getRefresh_token());
    editor.apply();
}

//methode to get access_token and refresh_token
public Token getTokens(){
    SharedPreferences sharedPreferences = ctx.getSharedPreferences(SHARED_PREF_NAME_TOKEN, Context.MODE_PRIVATE);
    return new Token(sharedPreferences.getString(KEY_ACCESS_TOKEN, s1: null), sharedPreferences.getString(KEY_REFRESH_TOKEN, s1: null));
}

//store the username in shared preferences
public void saveName(String name){
    SharedPreferences sharedPreferencesUser = ctx.getSharedPreferences(SHARED_PREF_NAME_USER, Context.MODE_PRIVATE);
    SharedPreferences.Editor editorUser = sharedPreferencesUser.edit();
    editorUser.putString(s: "name", name);
    editorUser.apply();
}

public String getEmail(){
    SharedPreferences sharedPreferencesUser = ctx.getSharedPreferences(SHARED_PREF_NAME_USER, Context.MODE_PRIVATE);
    return sharedPreferencesUser.getString(s: "name", s1: null);
}
```

Figure 32: Storage of access and refresh token

So, when the user makes a request to connect to the application through the server, by pressing the “Login” button the following code will be executed (**Figure 33**). After first checking that the editTexts “editTextEmail” and “editTextPassword” are not empty, their content will be stored on the String variables named email and password. Then a String Request will be executed through the Volley library with the Post method, with which the user’s credentials (email, password) will be sent to the server, which will check if they are correct (authentication). Then, if the check is successful, the corresponding response will be sent to the application, which will include the access token and the refresh token. So, the piece of code inside *public void onResponse(String response){...}* will be called. In this part, the access token and refresh token sent by the server, as well as the user’s name, will be stored in the shared preferences, so that they can be reused immediately whenever needed.

```

login.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (!editTextEmail.getText().toString().isEmpty() && !editTextPassword.getText().toString().isEmpty()){
            final String email = editTextEmail.getText().toString().trim();
            final String password = editTextPassword.getText().toString().trim();

            RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());

            StringRequest stringRequest = new StringRequest(Request.Method.POST, LOGIN_URL, new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    try {
                        JSONObject obj = new JSONObject(response);
                        Token token = new Token(obj.getString( name: "access_token"), obj.getString( name: "refresh_token"));
                        SharedPreferences.getInstance(getApplicationContext()).saveTokens(token);
                        SharedPreferences.getInstance(getApplicationContext()).saveName(email);
                    }catch (JSONException exception){
                        exception.printStackTrace();
                        Toast.makeText(getApplicationContext(), exception.getMessage(), Toast.LENGTH_LONG).show();
                    }finally {
                        Intent intent = new Intent(getApplicationContext(), WelcomeActivity.class);
                        startActivity(intent);
                    }
                    System.out.println(response);
                    Toast.makeText( context: MainActivity.this, text: "Login successful", Toast.LENGTH_LONG).show();
                }
            }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    error.printStackTrace();
                    //403: wrong username or password
                    if (error.networkResponse.statusCode == 403){
                        Toast.makeText(getApplicationContext(), text: "Wrong email or password", Toast.LENGTH_LONG).show();
                    }
                    else {
                        Toast.makeText(getApplicationContext(), text: "An error occurred " + error, Toast.LENGTH_LONG).show();
                    }
                }
            }){
                @Override
                protected Map<String,String> getParams() throws AuthFailureError{
                    Map<String,String> params = new HashMap<>();
                    params.put( k: "username", email);
                    params.put( k: "password", password);
                    return params;
                }
            };

            //add request to the requestQueue
            requestQueue.add(stringRequest);
        }
        else {
            Toast.makeText(getApplicationContext(), text: "Email or password can not be empty", Toast.LENGTH_LONG).show();
        }
    }
});
}
}

```

Figure 33: Login with Volley library

Thus, for every further communication between the server and the application, it will not be necessary to re-enter the email and password. Along with each request for communication, the application will send the access token that has received, which the server will check if it is active and hasn't expired. If the check is successful, there will be given a response to the request. If the access token has expired, the server will inform the application and the application will ask to receive a new one, using the refresh token.

Volley

With the help of the HTTP library Volley, networking for Android apps is incredibly simple and quick. Google created it, and it was presented at Google I/O 2013. It was created since the Android SDK lacks a networking class that can function without impairing the user experience. Google declared in January 2017 that Volley would become an independent library even though it is now a component of the Android Open Source Project (AOSP). It controls the handling and caching of network requests and spares developers precious time by preventing them from having to write the same network call/cache code repeatedly. Due to the fact that Volley keeps all responses in memory while parsing, it is not ideal for big download or streaming operations.

Some of the main features of Volley are: a) the prioritization and queuing of requests, b) efficient memory and request cache management, c) the library's adaptability and personalization to meet our demands, and d) revocation of the requests

Below we will see an example of using the access token and refresh token. The piece of code that will be presented will be for the "Apply" button (**Figure 34**).

```
apply.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    public void onClick(View view) {
        changedData();
    }
});
```

Figure 34: Apply button

Pressing the "Apply" button calls the `changedData()` method. This method sends the data that the user has changed regarding the covid status in the `WelcomeActivity`, to Spring. This is achieved by constructing a `JSONObject`.

After the `JSONObject` is constructed, a `JsonObjectRequest` is executed through Volley with the `PUT` method, which sends the `JSONObject` to the server. At the same time, as seen in the piece of code in the `public Map<String, String> getHeaders() throws AuthFailureError{...}` the access token is also sent. Then Spring communicates with the PostgreSQL database and applies the changed data.

If everything goes well, the message of the successful saving of the data will be displayed to the user, through a toast message ("Data changed") in `public void onResponse(JSONObject response){...}`.

The content of `changedData()` is shown below in **Figure 35**.

```
private void changedData() {
    emailPerson = SharedPreferences.getInstance(getApplicationContext()).getEmail();

    JSONObject jsonObject = new JSONObject();
    requestQueue = Volley.newRequestQueue(context: WelcomeActivity.this);

    String checked = covid.getText().toString();

    try{
        jsonObject.put( name: "covid", checked);
    }catch (JSONException exception){
        exception.printStackTrace();
    }

    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.PUT, url: UPDATE_URL+emailPerson, jsonObject, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            System.out.println(response);
            Toast.makeText(context: WelcomeActivity.this, text: "Data changed", Toast.LENGTH_LONG).show();
            getData();
        }
    });
}
```

```

    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            error.printStackTrace();
            // displaying toast message on response failure
            Toast.makeText(context: WelcomeActivity.this, text: "Fail to update data" + error.getMessage(), Toast.LENGTH_SHORT).show();
            //403: access token is expired, request for new with refresh token method
            if (error.networkResponse.statusCode == 403){
                //String loadDataFromServer = "LoadDataFromServer";
                refreshToken();
            }else{
            }
        }
    }
}) {
    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        Token token = SharedPrefManager.getInstance(getApplicationContext()).getTokens();
        String access_token = token.getAccess_token();
        Map<String, String> headers = new HashMap<>();
        headers.put("Authorization", "Bearer " + access_token);
        return headers;
    }
};

requestQueue.add(jsonObjectRequest);
}

```

Figure 35: *changedData()* method

Provided that the change of the data in the database is successful, the `getData()` method is called, so that the change that was made can be seen in the user interface `WelcomeActivity`. The content of `getData()` is shown below in **Figure 36**.

```

private void getData() {
    emailPerson = SharedPrefManager.getInstance(getApplicationContext()).getEmail();

    requestQueue = Volley.newRequestQueue(context: WelcomeActivity.this);

    StringRequest stringRequest = new StringRequest(Request.Method.GET, url: GET_PERSON_URL + emailPerson, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            try {
                JSONObject jsonObject = new JSONObject(response);
                surnamePerson = jsonObject.getString(name: "surname");
                surname.setText(surnamePerson);
                namePerson = jsonObject.getString(name: "name");
                name.setText(namePerson);
                welcome.setText(namePerson);
                genderPerson = jsonObject.getString(name: "gender");
                gender.setText(genderPerson);
                emailPerson = jsonObject.getString(name: "username");
                email.setText(emailPerson);
                covidPerson = jsonObject.getBoolean(name: "covid");
                if (covidPerson){
                    covid.setText("True");
                }else {
                    covid.setText("False");
                }
                dobPerson = jsonObject.getString(name: "dob");
                dob.setText(dobPerson);
            } catch (JSONException e) {
            }
        }
    });
}
}

```



```

    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            //403: access token is expired, request for new with refresh token method
            if (error.networkResponse.statusCode == 403){
                //String loadDataFromServer = "LoadDataFromServer";
                refreshToken();
            }else{
            }

        }
    }
}
}

@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    Token token = SharedPrefManager.getInstance(getApplicationContext()).getTokens();
    String access_token = token.getAccess_token();
    Map<String, String> headers = new HashMap<>();
    headers.put("Authorization", "Bearer " + access_token);
    return headers;
}

};
requestQueue.add(stringRequest);
}
}

```

Figure 36: `getData()` method

If the access token has expired, then an error indication with *status code 403* will be sent from the server, which will be received by the application inside Volley's `public void onErrorResponse(VolleyError error){...}`. So, if it receives an error with this status code then the `refreshToken()` method will be called to request a new access token. Below we see the implementation of this method in **Figure 37**:

```

public void refreshToken(){
    JSONObjectRequest refreshTokenRequest = new JSONObjectRequest(
        Request.Method.GET,
        REFRESH_TOKEN_REQUEST_URL,
        jsonRequest: null,
        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                try {
                    Token token = new Token(
                        response.getString("access_token"),
                        response.getString("refresh_token")
                    );

                    SharedPrefManager.getInstance(getApplicationContext()).saveTokens(token);
                } catch (JSONException exception) {
                    exception.printStackTrace();
                    Toast.makeText(getApplicationContext(), exception.getMessage(), Toast.LENGTH_LONG).show();
                } finally {
                    //make again the get request with the new access_token
                    //makeGetRequest();
                }
            }
        }
    );
}
}

```

```

        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                error.printStackTrace();
                Toast.makeText(getApplicationContext(),
                    text: "Error: " + error.getMessage().toString(), Toast.LENGTH_LONG).show();
            }
        });
        @Override
        public Map<String,String> getHeaders() throws AuthFailureError{
            Token token = SharedPrefManager.getInstance(getApplicationContext()).getTokens();
            refresh_token = token.getRefresh_token();
            Map<String,String> headers = new HashMap<>();
            headers.put( k: "Authorization", v: "Bearer " + refresh_token);
            return headers;
        }
    };
    requestQueue.add(refreshTokenRequest);
}

```

Figure 37: refreshToken() method

Annotations

Below are listed the annotations used in this application, along with a brief explanation of their function.

@SpringBootApplication

This component is made up by @Configuration, @ComponentScan, and @EnableAutoConfiguration. The base package retains the class with the annotation @SpringBootApplication. Performing the component scan is this annotation. Only the sub-packages are scanned, though.

@RestController

The annotations @Controller and @ResponseBody can be combined to form this. The @ResponseBody annotation is attached to the @RestController annotation. There is no longer a requirement to annotate @ResponseBody on every method.

@RequestMapping

Using @RequestMapping, the HTTP request is mapped. Both the class and the method can use it. There are numerous additional optional components, including consumes, name, method, request, path, etc.

@RequiredArgsConstructor

It helps a constructor to be created with the necessary inputs. Arguments with limitations like `@NonNull` and final fields must be provided.

`@NoArgsConstructor`

It creates a constructor with no arguments. If such a constructor is impossible to write because final fields exist, an error message will be produced.

`@AllArgsConstructor`

It produces a constructor with all arguments. Every field in the class must have one parameter when using an all-args constructor.

`@GetMapping`

It is an annotation for assigning particular handler methods to HTTP GET requests. In particular, `@GetMapping` is a constructed annotation that replaces `@RequestMapping(method = RequestMethod.GET)` when needed.

`@PostMapping`

The particular handler technique is used to map HTTP POST requests. It is utilized to create a web service endpoint. Use this in place of `@RequestMapping(method = RequestMethod.POST)`.

`@PutMapping`

The particular handler technique is used to map HTTP PUT requests. It is utilized to create or update a web service endpoint. Use this in place of `@RequestMapping(method = RequestMethod.PUT)`.

`@RequestBody`

It is utilized to link a method parameter with an object in an HTTP request. The request body is converted internally using HTTP MessageConverters. The Spring framework links the body of the incoming HTTP request to a method parameter when it sees the annotation `@RequestBody`.

`@PathVariable`

The values from the URI are extracted using it. Where the URL has a path variable, it is most appropriate for RESTful web services. A method can define a number of `@PathVariables`.

`@Data`

The Lombok data annotation (`@data`) produces getters for every field, a practical toString function, and hashCode and equals implementations that verify every non-transient field. Additionally, it will produce a constructor and setters for all non-final fields. The equivalent of a `@data` annotation is a mix of `@getter`, `@setter`, `requiredArgsConstructor`, `@toString`, `@equals`, and `hashCode`.

`@Entity`

The `@Entity` annotation is required by the JPA specification. A class is designated as an entity class.

`@ToString`

The non-static field names and values obtained by calling the getter are printed by the `@ToString` annotation along with the class name (if declared). Additionally, the fields are arranged in the source class according to the declaration order.

`@Id`

The member field below is the primary key for the current entity, as indicated by the `@Id` annotation.

`@GeneratedValue`

The persistent implementation can automatically give your identity fields a unique value by using the `@GeneratedValue` annotation. It won't generate any value; it is simply utilized to obtain the generated value. It is used to define the process for obtaining the value for the ID field. Along with the annotation of `Id`, `IT` is required.

`@OneToMany`

The Java persistent API specification has the annotation `@OneToMany`. Indicated by this annotation, a group of objects are the current entity's direct children.

`@ManyToOne`

In Spring Data JPA, a many-to-one relationship between two entities is defined using the `@ManyToOne` annotation. The owner of the relationship defined by the `@ManyToOne` annotation is the child object that contains the join column.

`@ManyToMany`

A many-to-many relationship is established between two entities using the `@ManyToMany` annotation. When there is a bi-directional association, both entities have the `@ManyToMany` annotation, but only one of them can be the relationship's owner.

`@JoinColumn`

When merging an entity association or element collection, a column is specified using the `@JoinColumn` syntax. According to this annotation, the enclosing entity is the relationship's owner and the accompanying table contains a foreign key column that points to the table of the non-owning side.

`@OnDelete`

When there are joined sub classes, `@OnDelete` in hibernate is used. The `@OnDelete` function determines whether or not deleting an entry from the database will also delete the rows represented by the joined sub class.

`@JsonIgnore`

This annotation enables us to ignore specific Java class properties during serialization to a JSON object.

`@JsonBackReference`

A backreference that is omitted during the serialization process is an annotation called `@JsonBackReference`.

`@Transient`

When a field is annotated with the `@Transient` flag, the mapping framework disregards that field and does not map it to any database columns (in RDBMS) or document properties (in NOSQL).

@Slf4j

Lombok creates a logger field in response to an annotation of type `Slf4j`. Classes and enumerations can both use this annotation.

@Override

The `@Override` annotation notifies the compiler that an element declared in a superclass is intended to be overridden by the current element. It is not necessary to use this annotation when overriding a method, however doing so helps to avoid mistakes.

@Query

Only repository interface methods can be annotated with the `@Query` annotation. The usage of the annotated methods is fairly simple; upon calling them, the statement contained there will be executed. Both JPQL and native SQL are supported by the `@Query` annotation.

@Configuration

`@Configuration` is a class-level annotation which is used as a source of bean definitions.

@EnableWebSecurity

In order to activate Spring Security's web security support and provide the Spring MVC integration, the `WebSecurityConfig` class is annotated with `@EnableWebSecurity`. In order to modify a few of `WebSecurityConfigurerAdapter`'s methods and set additional web security configuration parameters, it also expands that adapter's functionality.

@Service

It is applied on a class level. It indicates that the annotated class is a service class that uses external APIs and business core logic.

@Transactional

A proxy that implements the same interface(s) as the class you're annotating is automatically created by Spring when you annotate a method with the `@Transactional` annotation. So, the calls that clients make to your object are intercepted and the behaviors are injected via the proxy method.

@Bean

At the method level, the @Bean annotations are used to declare that a method creates a bean that the Spring container should handle. It serves as a substitute for the XML <bean> tag.

Future Expansions

The current application is at a very early stage, so in the future it is expected to be expanded and outgrowth in order to benefit even more the citizens. Surely, there are numerous perspectives for such an application, but let's focus on the solutions that seem to be the more prominent.

A good addition to this application would be to show to the users, according to their current position the nearest hospitals, drugstores or clinics, so in a case of emergency would be easy to find the quickest place to go and generally inform a patient about his choices. Although it was initially planned for this feature to be performed in the application, a change made by Google Maps made it executable only under payment, so it was decided not to perform it in the case of this Thesis. If it was about to be part of a real application, then it surely be included in the project.

Another useful addition could be the use of the Bluetooth, which surely is more precise when it comes to locating distances. It also operates better than any other mean of signal transmission, meaning that even if the GPS signal is weak or lost, the Bluetooth signal will remain strong and operative. It is noted that the Bluetooth signal has a range of 10 meters. With this feature, the overcrowded places will be detected and a warning will be sent to the users of the application to make them aware of the number of people that are gathered in a specific location. This serves another government's plan for the limitation of the corona virus spreading regarding avoiding overcrowded areas (the number of persons allowance differs from country to country).

Furthermore, we could add to the application a function that warns a user when they are within a certain distance of an infected person. These persons could then self-restraint with the guidance of the application and the competent authorities. This comes in accordance with the government's regulations about the covid-19 restriction meters.

Finally, it would be very useful to add a field for the user's vaccination status. With this feature, it will come in handy for every user to see how many shots has taken and see the duration of the vaccination certificate. Also, a good add-on would be for the users to be able to download the vaccination certificate or to book, change and reschedule their appointments for vaccination.

Conclusions

Unfortunately, the Covid-19 outbreak has caused us all to make changes to our regular routines because we were abruptly instructed to reduce our social interactions and activities. Therefore, the development of the work provided was focused on the context of this crisis, with the goal of providing users with a tool to register their covid status that would assist them avoid areas with high levels of crowding. It also provides the users with statistical insights about the virus spread. This would help limit the pandemic's impact and hasten the return to normalcy. With a straightforward and welcoming user interface, it is an easy software to use and comprehend. Spring Boot provides an easier and more responsive way of developing the web server with which the application will communicate. Regarding the Spring Boot, it constitutes the latest technology being used as far as it concerns the development of a web server.

Acknowledgments

I would like to thank my advisor on this Thesis, Efthymios Alepis, for his valuable insight and knowledge he was willing to share.

References

www.stackoverflow.com

<https://spring.io/projects/spring-boot>

<https://github.com/PhilJay/MPAndroidChart>

<https://www.postgresql.org/>

<https://google.github.io/volley/>

<https://developer.android.com/>

<https://pubmed.ncbi.nlm.nih.gov/33469298/>

https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

<https://app.diagrams.net/>

<https://www.lucidchart.com/pages/er-diagrams>