# ZensoMotion

## Computer Science
## Subject Module Project

Group number: V2026631-4

Supervisor:  Sune Thomas Bernth Nielsen

Semester: 5

Group members:

Ali Isac Showiki #66494

Bahar Sadik #66700

Anis Aziz #66422

Hassan Zalaf #66463

Date: 18/12-2020

Characters: 55875

# 1 Abstract

This report will first and foremost be centered around the creation of our Fitness oriented application *ZensoMotion.* The report will both explain the overall functionality of the Android application, and essential parts of the code. There will also be a rather in depth focus on a variety of the choices that were taken throughout the development process of the program. This includes crucial design choices, delimitations and theories. Design patterns, such as MVC and Singleton will be used and discussed in the report. This, combined with a use of sensors, sounds and a database, will be the interactive and social experience that is, *ZensoMotion*.

The report will furthermore focus on the performance of four usability tests, and the choices that were made in relation to their execution. There will also be accounted for the changes that were made as a result of these tests, as well as future changes that are yet to be implemented.

# 2 Preface

Link to the github repository, where our current version of the app, lies under the DatabaseClass branch:
https://github.com/UNImakeup/Firebase/tree/DataBaseClass

## Primary research question

How can we build an app, whilst using object oriented programming principles and various design patterns, that uses android phones' sensors to create an interactive and social workout experience?

## Support questions

1. How can OOP principles and design patterns be used to create manageable code in an app that uses data across activities?

2. How can we use firebase to create a user system, where users can compete with each other?
3. How can we use sensors and sounds to create an interactive experience?
4. How can we perform a usability test of our app?

# 3 Motivation

We are four people with one common interest in making exercise fun. Android Studio gave us the opportunity to choose between two coding languages: Kotlin or Java. We chose Java as the programming language since we were more familiar with Java as a programming language. This gave us an increased desire to develop an application, because we see it as an opportunity to get more familiar with java as a coding language. In addition to having a desire to work with app development, working with databases such as the Realtime Database, on Google's Firebase platform gave us the opportunity to learn how to make scalable applications (in terms of adding more users/data).

## 4.3 Project Design

We will be developing an Android app, using the Android Studio IDE and Java as the programming language. Android Studio is the official development kit from Android, and is used in the creation of mobile applications for Android devices. The development kit offers a visual look into the developed application, a blueprint-tab that eases visual customizations.

**4.4 Delimitation**

This section will account for the delimitations that were done in relation to the project.

At the start of the semester, shortly after establishing our interest in creating a mobile application, we chose to limit ourselves by exclusively developing for Android. After thorough research on the matter, we concluded that it made more sense for us to choose Android as the operating system for our application. This delimitation was mainly influenced by the fact that we already had some experience with Java due to our previous courses, while having no experience with Swift, the development language for iOS.

It could also have proved to be beneficial to use Flutter as a framework when programming. The framework would have resulted in an application that would have been applicable to both iOS and Android devices (Flutter.dev). This could have benefited both our future work with the application, and the previously performed usability tests, as the group of potential users would not exclusively have been individuals with Android devices. However, we first learned about the existence of this framework rather late in the project, at a point where we had worked quite a bit and established multiple functionalities in Android Studio. We therefore had to proceed with our current work.

We furthermore had to delimit ourselves in the choice of functionalities that were to be available upon hand-in. There was initially a focus on including exercises that focused on cardio in addition to the currently established resistance training exercises. We were however forced to momentarily pause our work on a map-focused running exercise, due to the prioritization of resistance exercises being available when testing the prototype.

Lastly, we chose to limit ourselves in the number of usability tests that were performed. There were a total number of four tests, with individuals from our own network. Our choice of informants, and limitation in regards to the number of performed tests, was both influenced by our limited range of options due to COVID-19, and our time-limit.

# 4 Introduction - Description of the project

This project is centered around the creation of a fitness application that makes use of the mobile device's sensors in its variety of exercises. The application is made for Android devices, and seeks to make exercising more accessible as well as more interactive. This is accomplished by including gamification aspects in the form of starting and taking part in competitions with users from one's network. A user is furthermore able to exercise alone, by completing sets of chosen resistance exercises put together as a routine. The current exercise routine consists of push-ups, squats, sit-ups and back-bends.

The application is also centered around the use of a login system that forces the user to register or log in, in order to make use of the application.

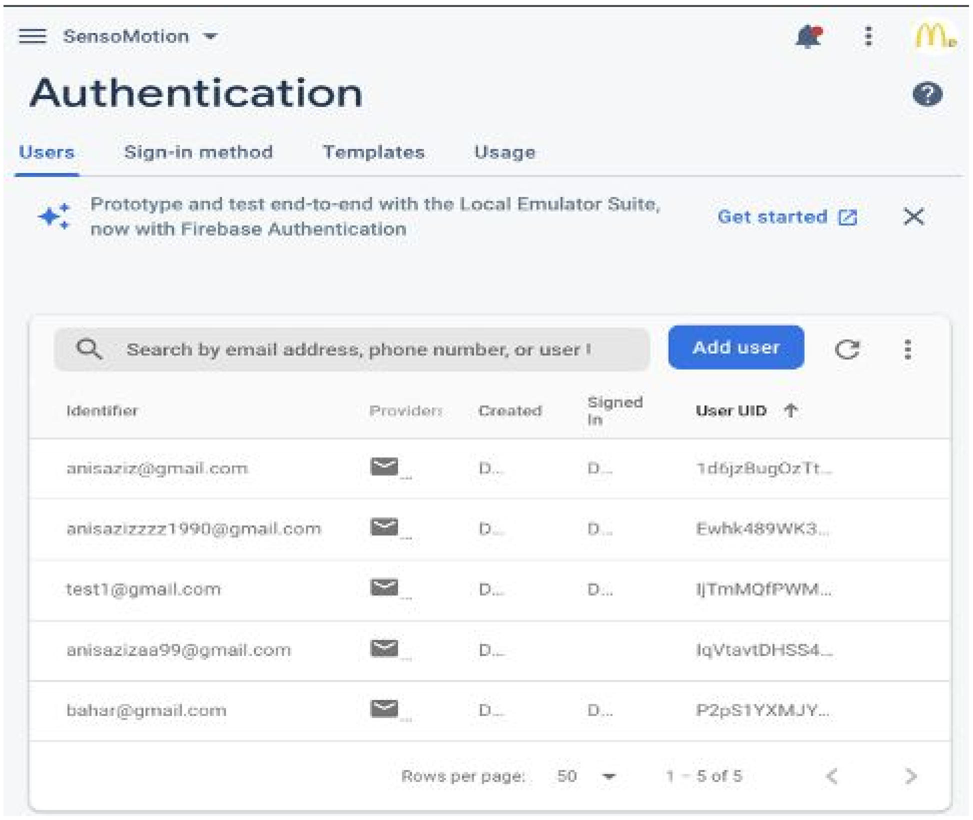
Zensomotion also includes facets that are not directly linked to exercising, but were still deemed to be essential for the user. This includes one's ability to have the application calculate a Body Mass Index, by inserting a height and a weight. The BMI is then stored, and displayed in the application. The zen of the application really shows in the “magic place”. Here, you can come and relax after a training session, and absorb the good vibes.

## Firebase

It is possible to try and implement functionality from scratch, when using a database. However, in this project we decided to work with and base the application on the Firebase platform. Firebase is a toolkit and infrastructure that aims at supporting the process of building applications. Firebase includes an SDK console, which provides a backend as a service to the developers. The services we decided to work with are Authentication and Realtime databases (Firebase, 2020)

Firebase provides other types of backend service that we could use. One of the services we considered using was the cloud storage to save users’ profile pictures. However, due to time limitations we prioritized more crucial functionalities.

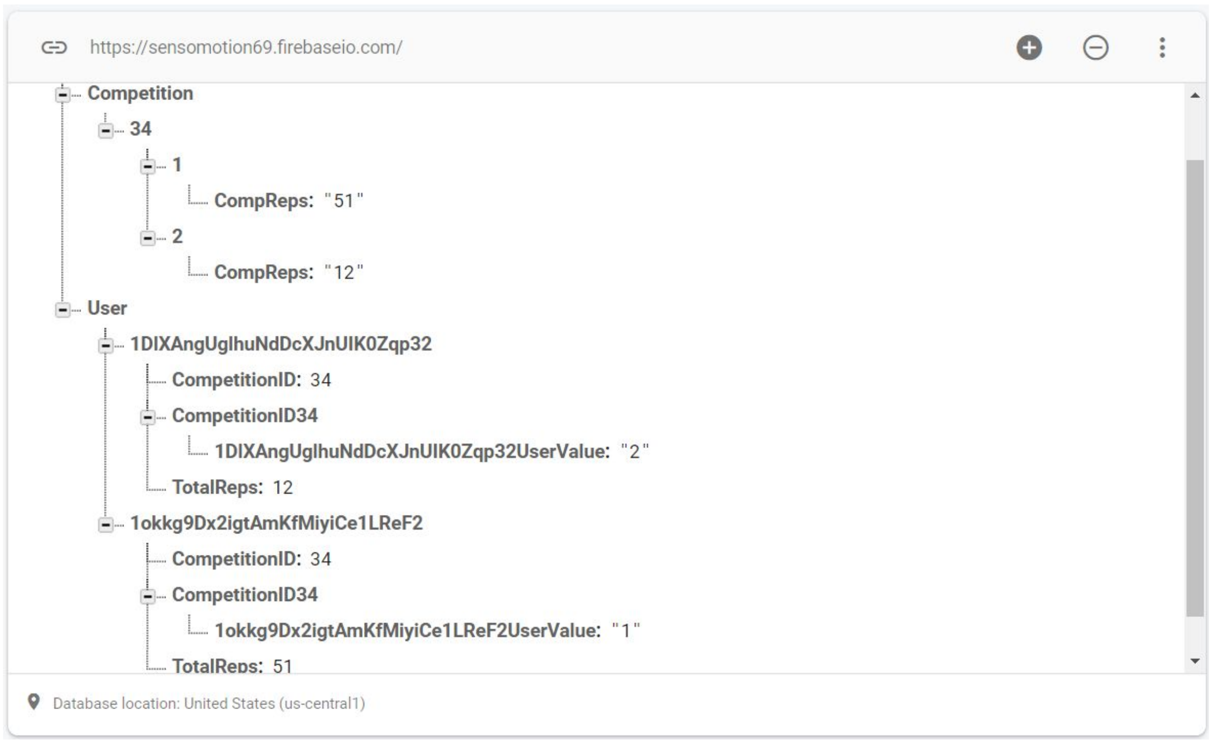
We have used Firebase authentication as our backend service to take care of getting users logged in and identified. That is something essential to our application especially in regards to restricting access to data, only allowing logged in users to access the database. This includes giving the users a way of tracking their progress. However, in the case of authentication, the main purpose is to make the user create their own individual account, registering with an email and password. Firebase authentication functions store the data (email and password) in the database. Firebase’s authentication service will store the user by having an identifier (email) when it is created, as well as their own unique User UID (Firebase Userid). This is also shown below (Firebase, 2020).


SensoMotion
Authentication
Users
Sign-in method
Templates
Usage
Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication
Get started
Search by email address, phone number, or user I
Add user

| Identifier | Providers | Created | Signed In | User UID |
|---|---|---|---|---|
| anisaziz@gmail.com | ✉ | D... | D... | 1d6jzBugOzTt... |
| anisazizzzz1990@gmail.com | ✉ | D... | D... | Ewhk489WK3... |
| test1@gmail.com | ✉ | D... | D... | IjTmMQfPWM... |
| anisazizaa99@gmail.com | ✉ | D... | | IqVtavtDHSS4... |
| bahar@gmail.com | ✉ | D... | D... | P2pS1YXMJY... |

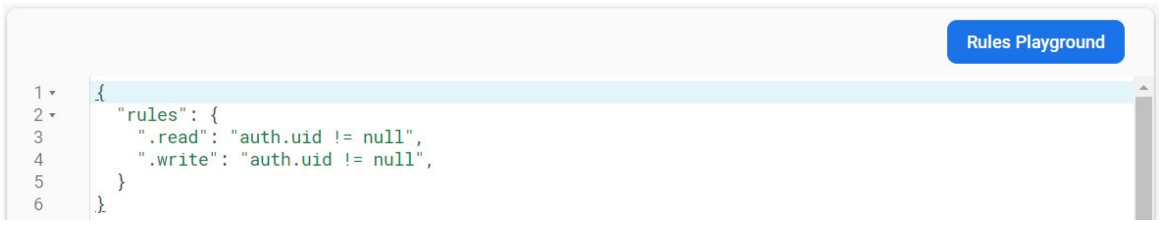
Rows per page: 50 1 – 5 of 5

We have used Firebase Realtime database to store and sync data with their NoSQL cloud database. This results in having the data synced across all clients in real-time. The Realtime database works by using a cloud hosted database, which is stored as JSON and synchronized in real-time to every connected client. Our intentions with the use of the Realtime Database by Firebase was to create a tree structure. With two branches, one for users (and userdata) and another for the in app competitions. This is shown below in our Firebase database (Firebase 2020).


```
https://sensomotion69.firebaseio.com/
Competition
  34
    1
      CompReps: "51"
    2
      CompReps: "12"
User
  1DIXAngUglhuNdDcXJnUIK0Zqp32
    CompetitionID: 34
    CompetitionID34
      1DIXAngUglhuNdDcXJnUIK0Zqp32UserValue: "2"
    TotalReps: 12
  1okkg9Dx2igtAmKfMiyiCe1LReF2
    CompetitionID: 34
    CompetitionID34
      1okkg9Dx2igtAmKfMiyiCe1LReF2UserValue: "1"
    TotalReps: 51
```
Database location: United States (us-central1)

The competition branch includes; The ID of the competition (competitionID), the two participants of the competition (userCompetitionID's, the 1 or 2) and how many reps they have done after they have taken part of the competition.
The user branch includes: The userID, the number of repetitions when doing exercises, competition id, storing which user they are in the competition and their individual BMI.

## Security

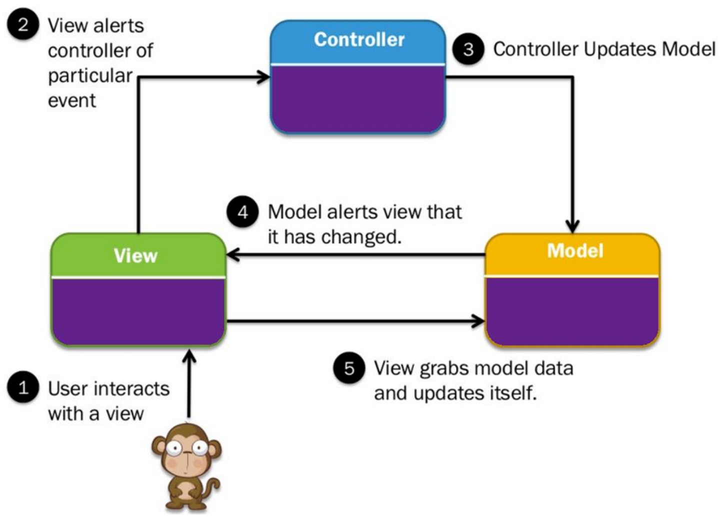


```
{
  "rules": {
    ".read": "auth.uid != null",
    ".write": "auth.uid != null",
  }
}
```

We set the rules of the database, so that only authorized users can read and write data on the database. In an actual use case, further rules should be established so certain users can access certain information. This would make sure that sensitive information, such as a user's BMI would only be accessible to the user itself.

## Model View Controller

One of the architectural patterns we strived to adhere to in our project was the Model View Controller design pattern. The following section describes the architecture of the Model View Controller (MVC) design pattern and at last the implementation of this model in our program. MVC is used to decouple user-interface(view), data (model), and application logic (controller).


2 View alerts controller of particular event
Controller
3 Controller Updates Model
4 Model alerts view that it has changed.
View
Model
1 User interacts with a view
5 View grabs model data and updates itself.

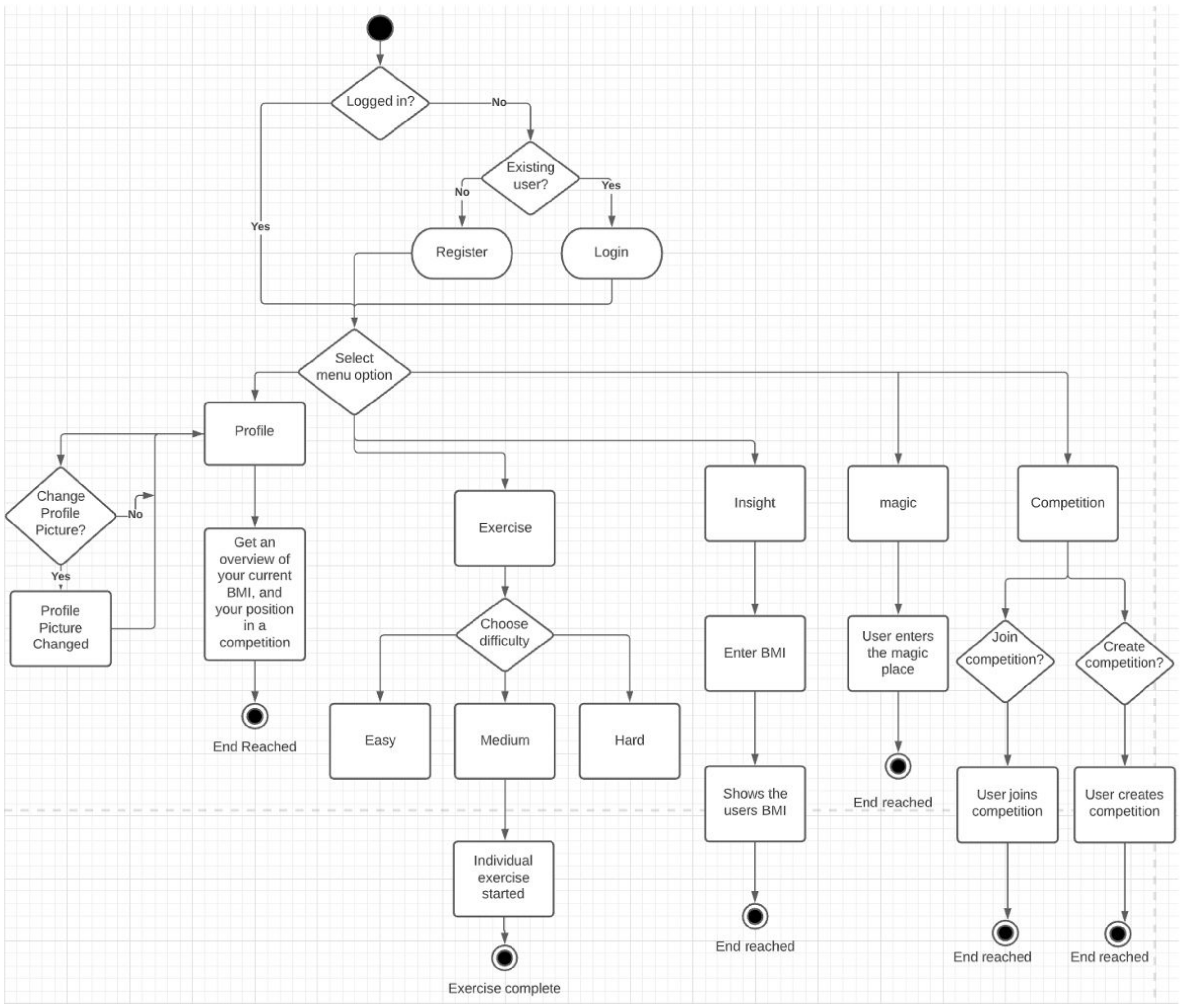
The **Model** is responsible for managing the data of the application, which it receives from the user input from the view.
**View** represents the UI that the user interacts with.
**Controller** acts as an interface between the Model and the View components, interpreting the users requests, manipulating data from the Model. This can result in changes in the view.

In our case it makes it easier to navigate through the system and make changes when needed. The pattern also makes it easier to identify bugs. This kind of architecture encapsulates functionalities and supports rapid and parallel development, since one programmer can work on the view while another programmer works on the controller. It is additionally more difficult to debug and update code that has dependencies from another of these components. Both the view and the controller depend on the model. The model however depends on neither the view or the controller. This is one of the major benefits, since it allows the model to be built and tested independent of the visual presentation (Microsoft, 2020). We will, throughout the explanation of the code, discuss this design pattern further.

# 5. Walkthrough of the application

This section will provide an overview of the intended usage of the application, from the very beginning where the user is obligated to create a profile, to the completion of an exercise routine. The section seeks to eliminate any possible misunderstandings of the application's functionality by walking the reader through the application in all its aspects.

The written walkthrough will be supplemented by screenshots of the different sections of the application. We will start the walkthrough, by showing an activity diagram we made specify the intended user-behaviour. This is illustrated through activities that define the behaviour of the use case (Seidl et. al, 2015:142).



**This activity diagram showcases the actions, outcomes and flow within the process that the application Zensomotion contains.**

The starting point of our application checks whether the user is logged in.

If yes: The user is taken to a new decision "select menu option" (the user is transferred to the home screen).

If no: The user is taken to a new decision; ("existing user?"), which represents if the user has an account. If the user does have an account, they will be transferred to a login activity, where they can login. If not, one has to register. After succesfully registering the user will be sent to "select menu option" (home screen).

The "select menu option" involves five alternate decisions with their own activity:

Profile:

The profile activity displays an overview of the users bmi result (given from insights) and their current position in the competition. The endpoint is reached. One has to either logout or go back to move on from this screen.

Exercise:

This activity involves a decision where the user can choose the difficulty between easy, medium and hard. The exercises consisting of pushup, squat, situps and backbends, will start after the chosen difficulty. The endpoint is reached after completed exercises. Here, you can go back to the home screen.

Insights:

This activity gives the user the opportunity to enter their height (in cm) and weight (in kg) to calculate their overall BMI. The endpoint is reached. You can go back to go to the home screen.

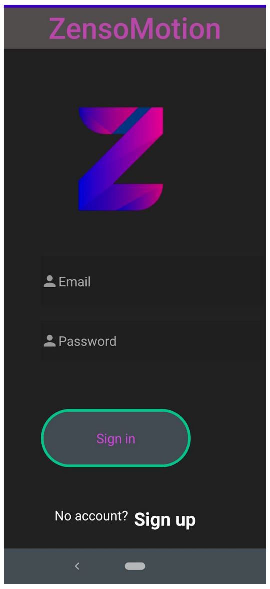
Magic:

This activity takes the user to a place of good vibes. The goal with this activity is to create a relaxing environment. You can use the bottom navigation menu, to move to either the home screen or the competition screen.

Competition:

This activity involves two decisions, where the user can join an exercise competition against another user or to create one giving other users an opportunity to join their competition.

## The Login Screen

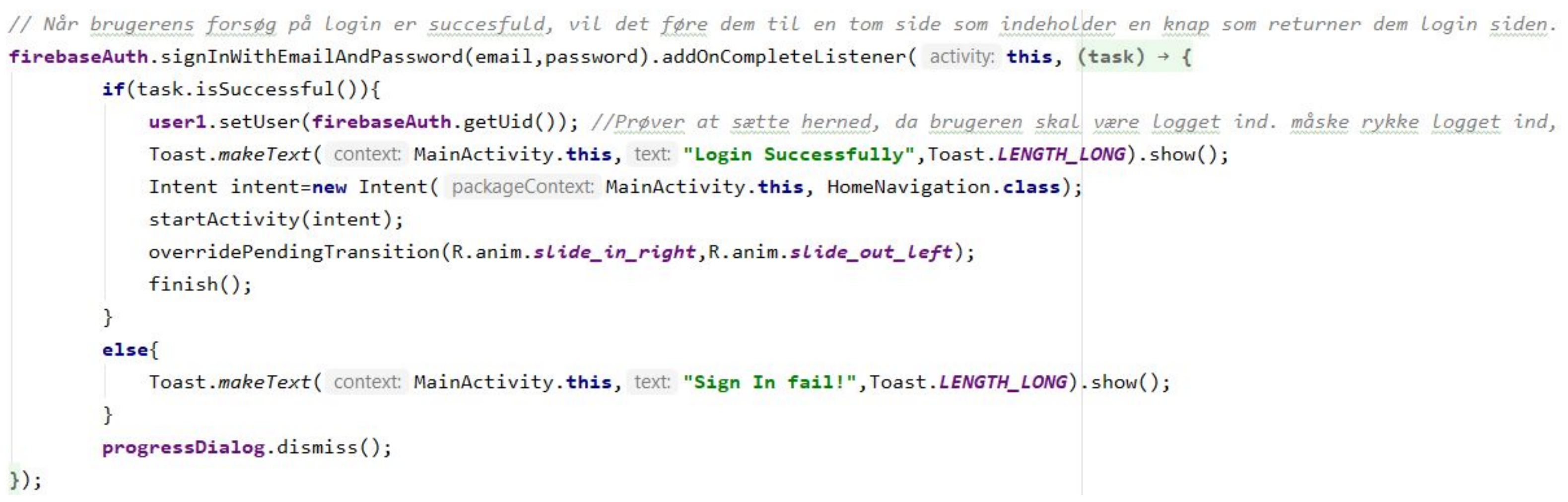


Upon installation, the user is introduced to a login page, that gives the user the option to register, and thereby create an account, or simply log in with an existing account.

On this screen, we start by checking whether the user is logged in. We do this by checking the user object, in which we have implemented the singleton design pattern. This allows us to get the same instance of it, in different activities, so that we can use the same information across multiple activities.

```
//Hvis brugeren ikke er logget ind.
if(user1.getUser().isEmpty()){ //Har gjort det med brugerobjektet og det virker.
    System.out.println("user not logged in");
} else { //Hvis brugeren er logget ind, kommer vi til hjemmeskærmen.
    Intent loggedIn = new Intent( packageContext: MainActivity.this, HomeNavigation.class);
    startActivity(loggedIn);
}
```

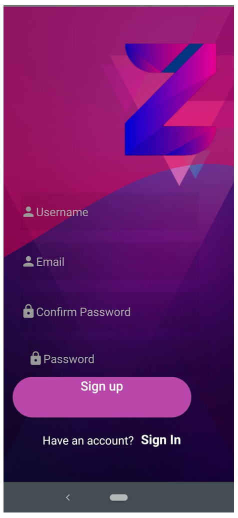
We use an if statement, where the user is transferred to the home screen, if they are already logged in. To check whether the user is logged in, we make use of shared preferences in the User class. When the user logs out, we empty the “username” String in the sharedPreferences of the app, which is where we also store the username, when the user logs in.



```
// Når brugerens forsøg på login er succesfuld, vil det føre dem til en tom side som indeholder en knap som returner dem login siden.
firebaseAuth.signInWithEmailAndPassword(email,password).addOnCompleteListener( activity: this, (task) → {
        if(task.isSuccessful()){
            user1.setUser(firebaseAuth.getUid()); //Prøver at sætte herned, da brugeren skal være logget ind. måske rykke logget ind,
            Toast.makeText( context: MainActivity.this, text: "Login Successfully",Toast.LENGTH_LONG).show();
            Intent intent=new Intent( packageContext: MainActivity.this, HomeNavigation.class);
            startActivity(intent);
            overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
            finish();
        }
        else{
            Toast.makeText( context: MainActivity.this, text: "Sign In fail!",Toast.LENGTH_LONG).show();
        }
        progressDialog.dismiss();
});
```

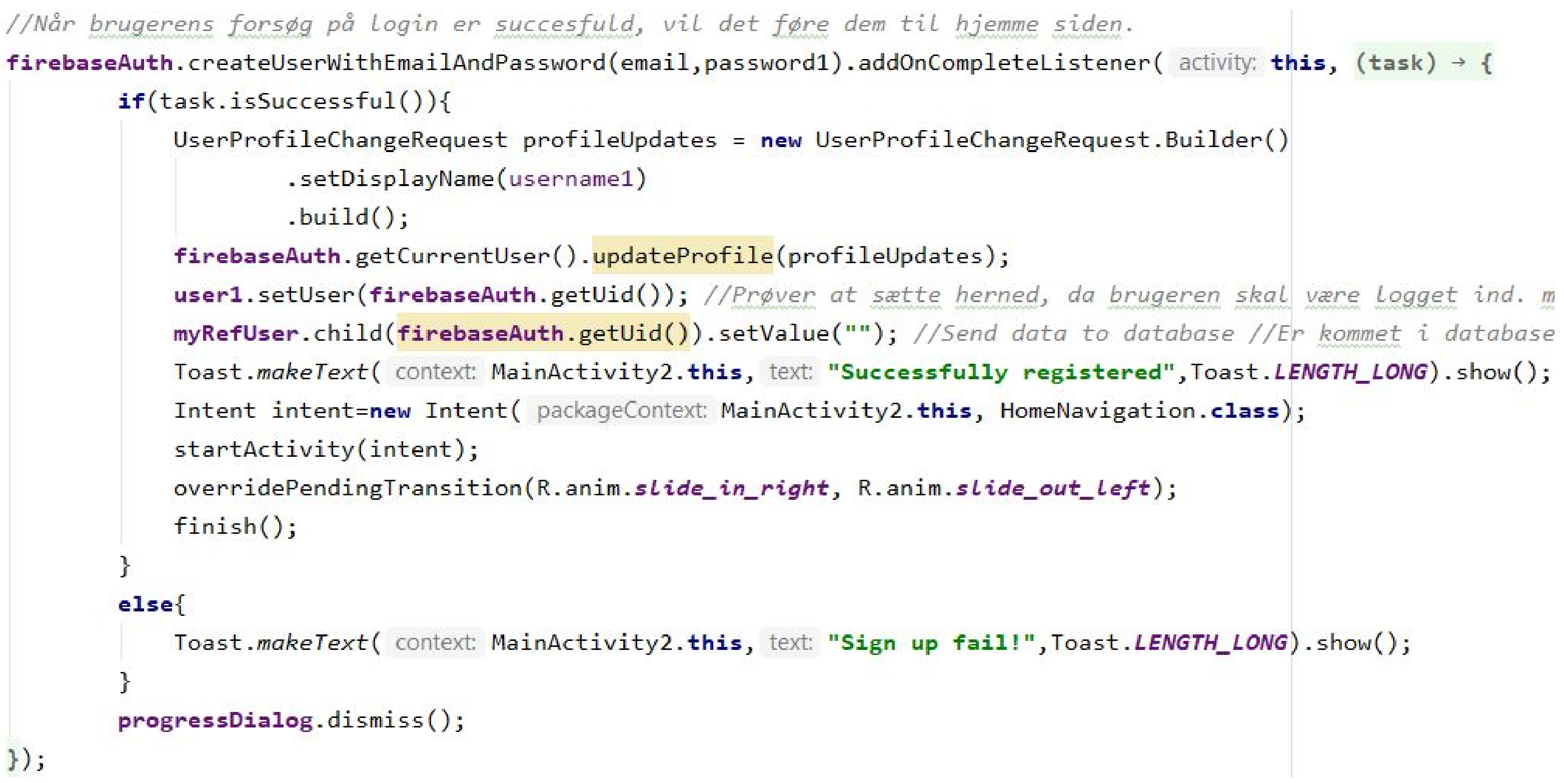
To log the user in, we use Firebase authentication with the function that signs a user in, with their email and password. Furthermore, if the login is successful, we save the UserID in the User singleton object. We can then use it later, without having to get it from the database and check whether the user is logged in, if you close and open the app again. Finally, the user is sent through to the home screen.

## The Registration Screen



If the user clicks on the “register” button, they are directed to a screen, asking for the creation of a username and password, with the help of the user's email. When the user successfully enters their chosen username, email and password, they are redirected to the home page.

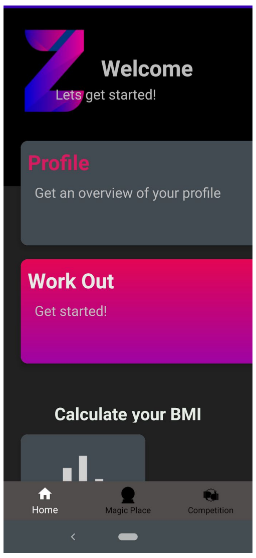
The register screen runs the register() method, when the signUp button is clicked. The method runs the createUserWithEmailandPassword() method, if no fields are empty.



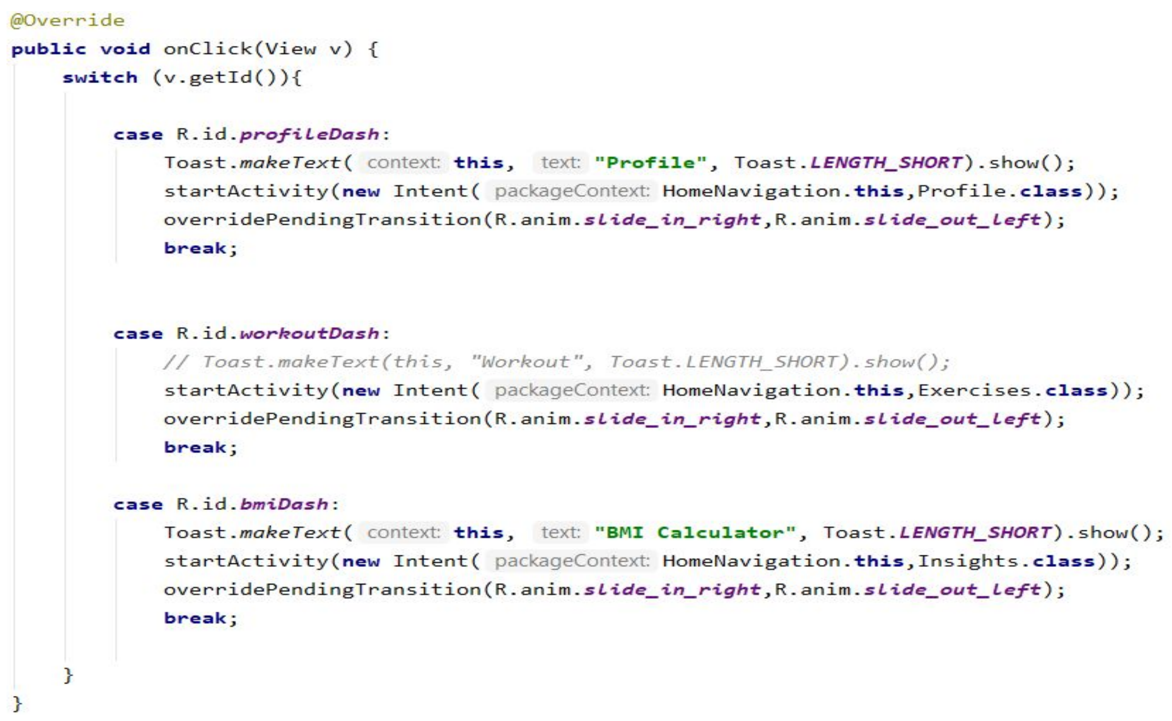
```
//Når brugerens forsøg på login er succesfuld, vil det føre dem til hjemme siden.
firebaseAuth.createUserWithEmailAndPassword(email,password1).addOnCompleteListener( activity: this, (task) → {
        if(task.isSuccessful()){
            UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                    .setDisplayName(username1)
                    .build();
            firebaseAuth.getCurrentUser().updateProfile(profileUpdates);
            user1.setUser(firebaseAuth.getUid()); //Prøver at sætte herned, da brugeren skal være logget ind. m
            myRefUser.child(firebaseAuth.getUid()).setValue(""); //Send data to database //Er kommet i database
            Toast.makeText( context: MainActivity2.this, text: "Successfully registered",Toast.LENGTH_LONG).show();
            Intent intent=new Intent( packageContext: MainActivity2.this, HomeNavigation.class);
            startActivity(intent);
            overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_left);
            finish();
        }
        else{
            Toast.makeText( context: MainActivity2.this, text: "Sign up fail!",Toast.LENGTH_LONG).show();
        }
        progressDialog.dismiss();
});
```

If the user is successfully created, we set their displayname to the username they wrote in the EditText field, by calling the updateProfile() method on the current User. We then set the User to the current Uid, in the User singleton object, so that we can use it when the user is logged in. We then create a child of the User branch in our Realtime Database, that we created in our firebase console. This child is set to the Uid of the current user. Since we saved this Uid in our User singleton object, we can save information on the database, under this User, and later retrieve the information using the Uid of the current user.

## The Main Menu/The Homepage



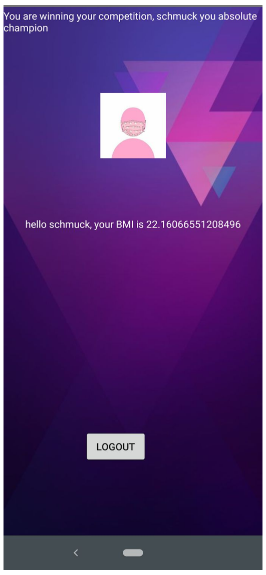
After successfully logging in, the user is directed to a home screen, with three key features, and an additional bar at the bottom that allows the user to engage with the “magic place”, as well as the competition screen.



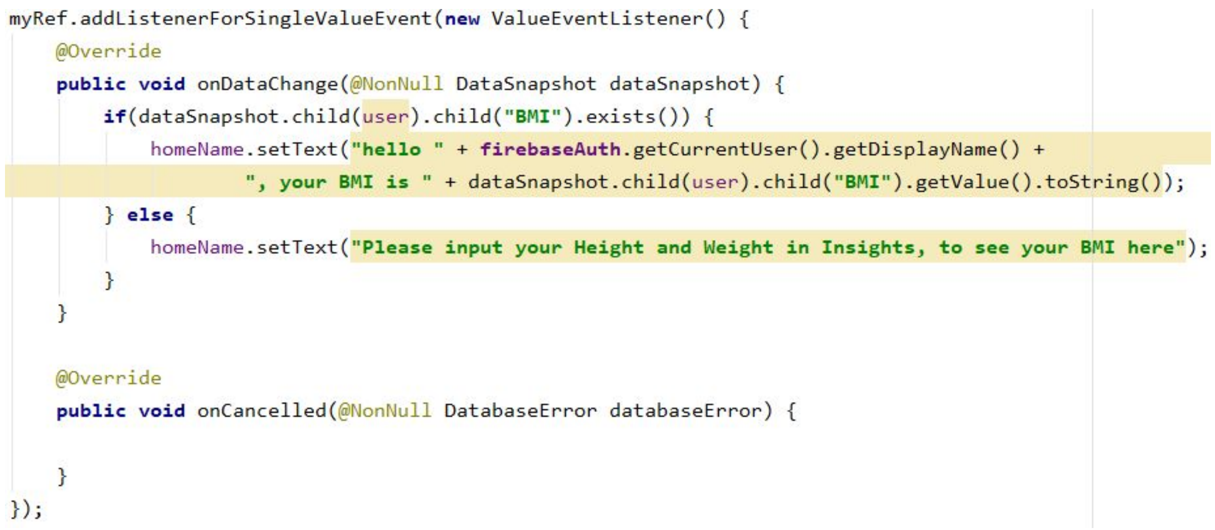
```
@Override
public void onClick(View v) {
    switch (v.getId()){

        case R.id.profileDash:
            Toast.makeText( context: this,  text: "Profile", Toast.LENGTH_SHORT).show();
            startActivity(new Intent( packageContext: HomeNavigation.this,Profile.class));
            overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
            break;


        case R.id.workoutDash:
            // Toast.makeText(this, "Workout", Toast.LENGTH_SHORT).show();
            startActivity(new Intent( packageContext: HomeNavigation.this,Exercises.class));
            overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
            break;

        case R.id.bmiDash:
            Toast.makeText( context: this,  text: "BMI Calculator", Toast.LENGTH_SHORT).show();
            startActivity(new Intent( packageContext: HomeNavigation.this,Insights.class));
            overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
            break;

    }
}
```

On this page, you can navigate through the app. This functionality is made with two switch statements which decide what happens, based on what element you select. By clicking on the elements you are sent through to the chosen activity.

## The Profile Screen

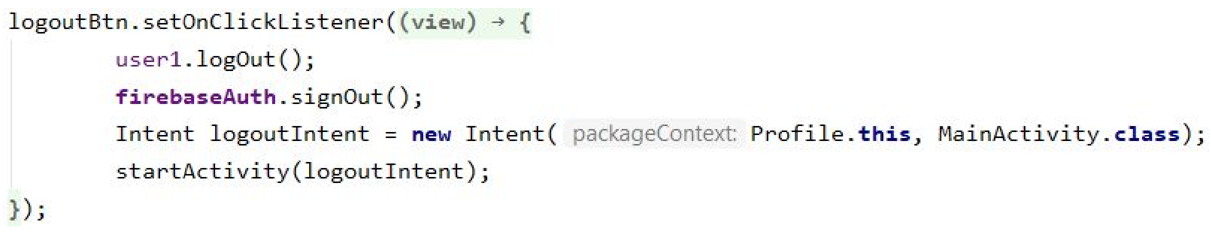

You are winning your competition, schmuck you absolute champion
hello schmuck, your BMI is 22.16066551208496
LOGOUT

On the profile screen, the user can check their current Body Mass Index. An initial overview of a user's position in a competition is furthermore portrayed here. The user will more specifically be informed if they are ahead, or behind their competitors. Lastly the user is able to log out, and be redirected to the initial log in page, by clicking the "Log Out" button.



```
myRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if(dataSnapshot.child(user).child("BMI").exists()) {
            homeName.setText("hello " + firebaseAuth.getCurrentUser().getDisplayName() +
                    ", your BMI is " + dataSnapshot.child(user).child("BMI").getValue().toString());
        } else {
            homeName.setText("Please input your Height and Weight in Insights, to see your BMI here");
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

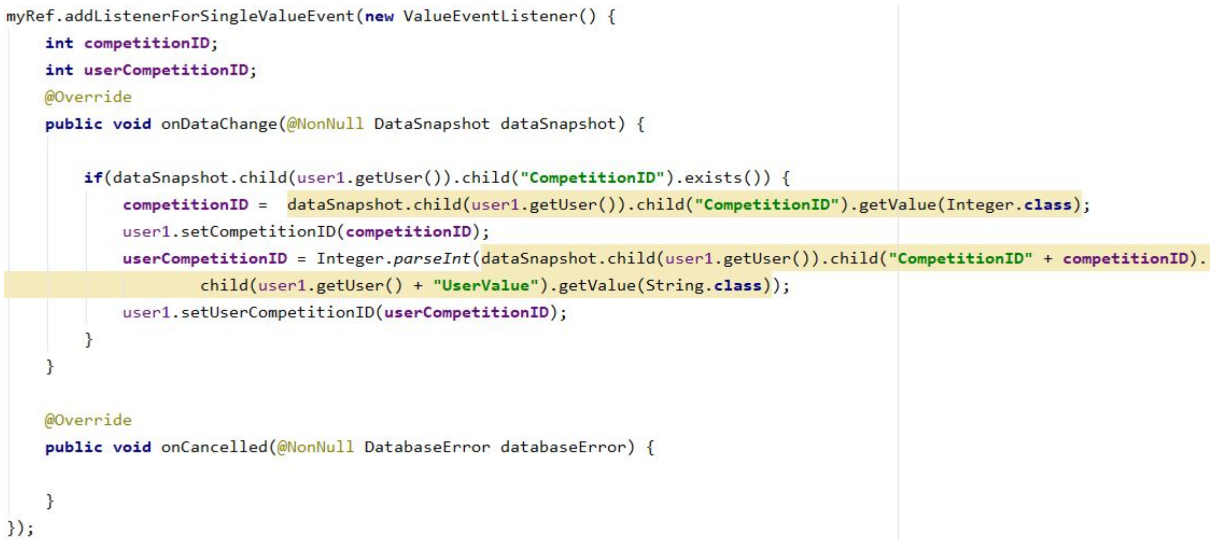
Firstly we connect to the database, to print out the BMI, if the user has put in their height and weight on the insights screen (and it has been saved on the database). We then get the current user's display name, to show it when we talk to the user. This could have been done in a separate singleton class (such as the User class), when the app starts (on the login screen, before sending the user on to the HomeNavigation activity) to have it readily available when you have to show it. Not doing this gives us an issue, because the text sometimes takes time to load which means that the TextView is empty for a short period in the beginning. Due to time limitations we had to prioritize different tasks. If the user does not have a BMI saved, we tell them to go to the insights screen.



```
logoutBtn.setOnClickListener((view) → {
        user1.logOut();
        firebaseAuth.signOut();
        Intent logoutIntent = new Intent( packageContext: Profile.this, MainActivity.class);
        startActivity(logoutIntent);
});
```

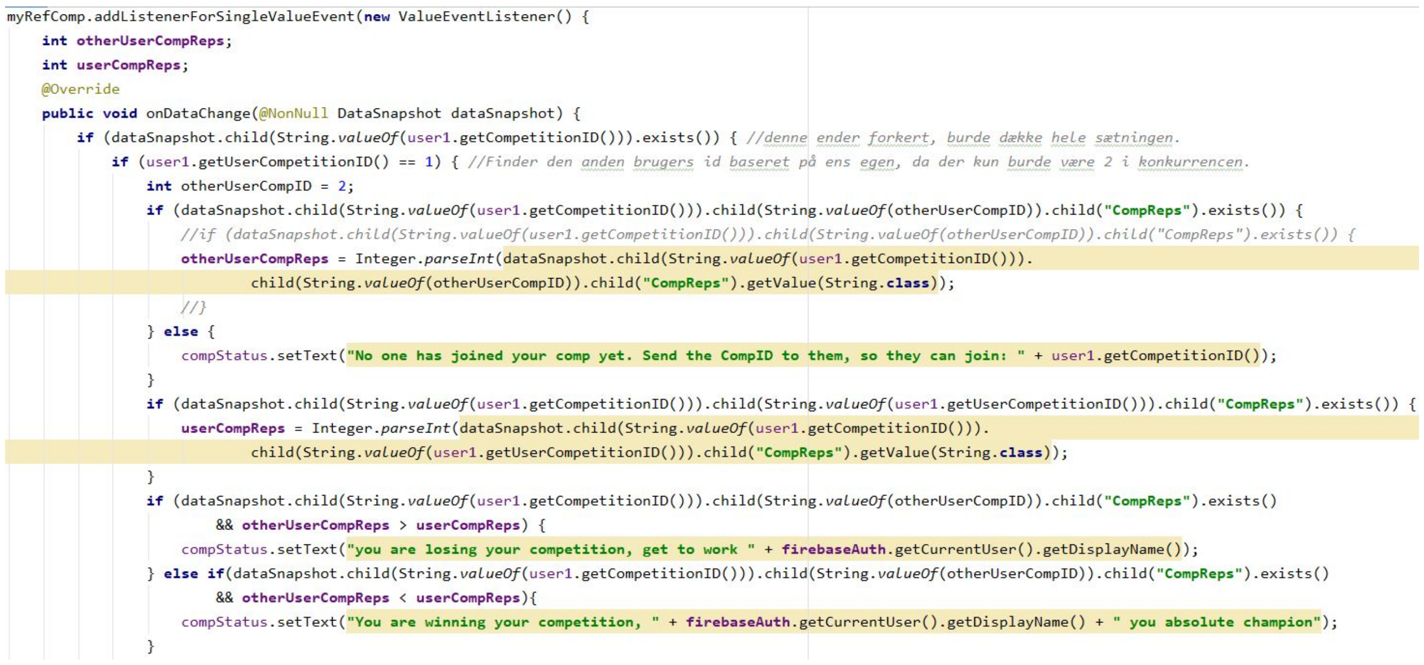
To log the user out, we call the logOut method on the User object, so that the Uid is removed from the shared preferences of the app. Furthermore, we use Firebase Authentication's signOut() method and send the user to the login screen.

To show the competition status, we first save the competitionID and the userCompetitionID to the User singleton class. The userCompetitionID is the ID of the user, in the competition branch, as a child of the competitionID.



```
myRef.addListenerForSingleValueEvent(new ValueEventListener() {
    int competitionID;
    int userCompetitionID;
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        if(dataSnapshot.child(user1.getUser()).child("CompetitionID").exists()) {
            competitionID =  dataSnapshot.child(user1.getUser()).child("CompetitionID").getValue(Integer.class);
            user1.setCompetitionID(competitionID);
            userCompetitionID = Integer.parseInt(dataSnapshot.child(user1.getUser()).child("CompetitionID" + competitionID).
                    child(user1.getUser() + "UserValue").getValue(String.class));
            user1.setUserCompetitionID(userCompetitionID);
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

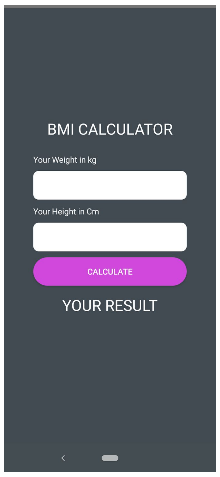
We then use these values to show the current status of the competition, whether the user is winning or losing.



```java
myRefComp.addListenerForSingleValueEvent(new ValueEventListener() {
    int otherUserCompReps;
    int userCompReps;
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (dataSnapshot.child(String.valueOf(user1.getCompetitionID())).exists()) { //denne ender forkert, burde dække hele sætningen.
            if (user1.getUserCompetitionID() == 1) { //Finder den anden brugers id baseret på ens egen, da der kun burde være 2 i konkurrencen.
                int otherUserCompID = 2;
                if (dataSnapshot.child(String.valueOf(user1.getCompetitionID())).child(String.valueOf(otherUserCompID)).child("CompReps").exists()) {
                    //if (dataSnapshot.child(String.valueOf(user1.getCompetitionID())).child(String.valueOf(otherUserCompID)).child("CompReps").exists()) {
                    otherUserCompReps = Integer.parseInt(dataSnapshot.child(String.valueOf(user1.getCompetitionID())).
                            child(String.valueOf(otherUserCompID)).child("CompReps").getValue(String.class));
                    //}
                } else {
                    compStatus.setText("No one has joined your comp yet. Send the CompID to them, so they can join: " + user1.getCompetitionID());
                }
                if (dataSnapshot.child(String.valueOf(user1.getCompetitionID())).child(String.valueOf(user1.getUserCompetitionID())).child("CompReps").exists()) {
                    userCompReps = Integer.parseInt(dataSnapshot.child(String.valueOf(user1.getCompetitionID())).
                            child(String.valueOf(user1.getUserCompetitionID())).child("CompReps").getValue(String.class));
                }
                if (dataSnapshot.child(String.valueOf(user1.getCompetitionID())).child(String.valueOf(otherUserCompID)).child("CompReps").exists()
                        && otherUserCompReps > userCompReps) {
                    compStatus.setText("you are losing your competition, get to work " + firebaseAuth.getCurrentUser().getDisplayName());
                } else if(dataSnapshot.child(String.valueOf(user1.getCompetitionID())).child(String.valueOf(otherUserCompID)).child("CompReps").exists()
                        && otherUserCompReps < userCompReps){
                    compStatus.setText("You are winning your competition, " + firebaseAuth.getCurrentUser().getDisplayName() + " you absolute champion");
                }
```

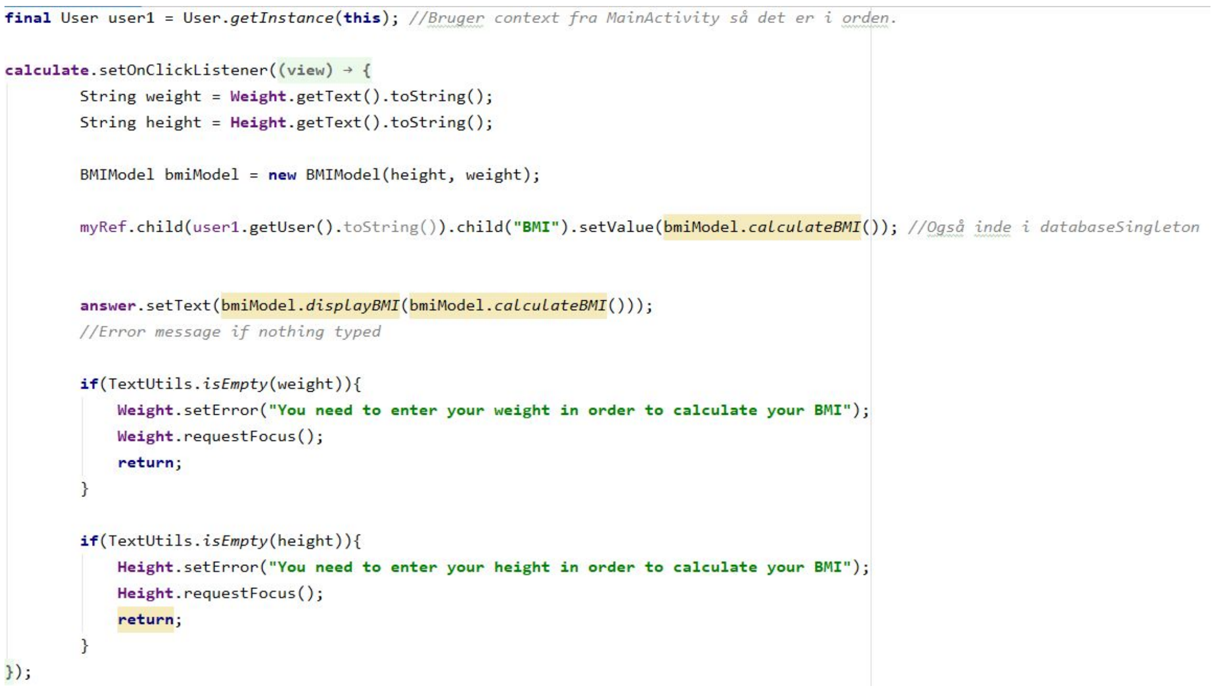
To see which user is winning, we have to get the amount of reps they have taken after joining or creating the competition. The amount of repetitions is saved in the Competition branch under the competitionID, that the user is partaking in. We first check whether the user is in a competition. If the user's userCompetitionID is 1, they started the competition, and therefore the other User would have 2 as their ID, as there can only be two users in the competition.
We can then save the current user and the competing users' amount of reps in two integer variables. We check whether the values exist before getting them, as the app crashes if it tries to fetch data that does not exist.
To print out the messages, we compare the two strings. If the other user (competing user) has a higher amount of reps, a message signalling that you are losing the competition is shown. If the current user has higher reps, it is signalled. Finally, if the other users' reps don't exist, it is shown that they have not joined the competition yet.
If the current user has 2 as their ID, we go through the same steps, without notifying if the other user exists, which they must do. The other user would be user 1, as they created the competition.

## The Insights Screen



On this screen, the user can input their height and weight, to calculate their BMI, and save it on their profile.

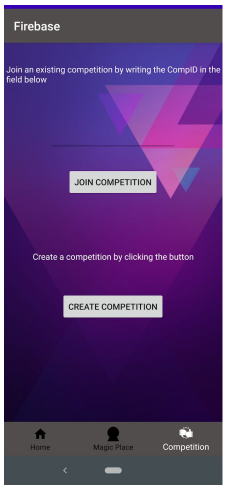
To calculate and display the BMI we use the BMIModel class. This is one of the instances where we have attempted to implement a MVC inspired structure, to adhere to the encapsulation principle of OOP. In this way, we have tried to make a more clear distinction of the responsibilities of the classes.



```
final User user1 = User.getInstance(this); //Bruger context fra MainActivity så det er i orden.

calculate.setOnClickListener((view) → {
        String weight = Weight.getText().toString();
        String height = Height.getText().toString();

        BMIModel bmiModel = new BMIModel(height, weight);

        myRef.child(user1.getUser().toString()).child("BMI").setValue(bmiModel.calculateBMI()); //Også inde i databaseSingleton


        answer.setText(bmiModel.displayBMI(bmiModel.calculateBMI()));
        //Error message if nothing typed

        if(TextUtils.isEmpty(weight)){
            Weight.setError("You need to enter your weight in order to calculate your BMI");
            Weight.requestFocus();
            return;
        }

        if(TextUtils.isEmpty(height)){
            Height.setError("You need to enter your height in order to calculate your BMI");
            Height.requestFocus();
            return;
        }
});
```

When the user clicks the calculate button we calculate the bmi using the calculateBMI method, from the BMIModel class. Furthermore we display it, with a message depending on the result, using the displayBMI() method. On the database, we save the BMI, under the user’s ID in the User branch of the database.

## The Competition feature

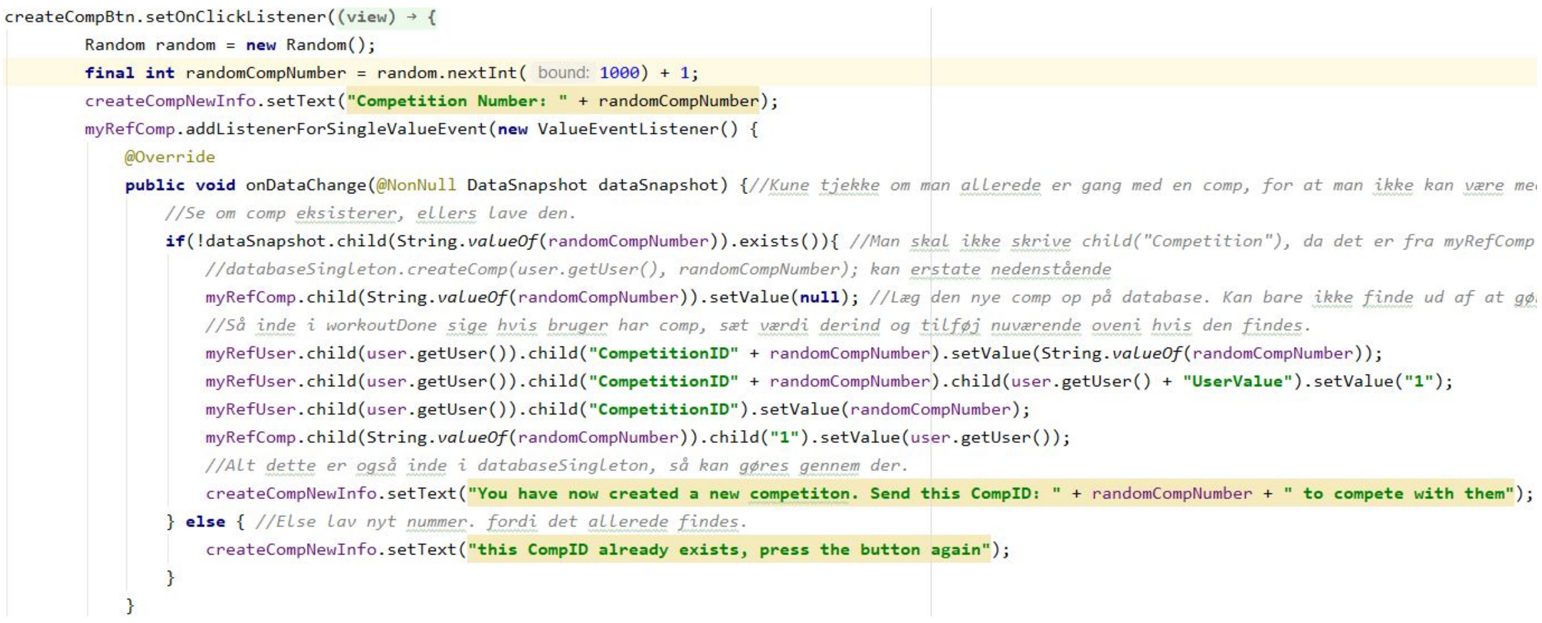


Join an existing competition by writing the CompID in the field below

Create a competition by clicking the button

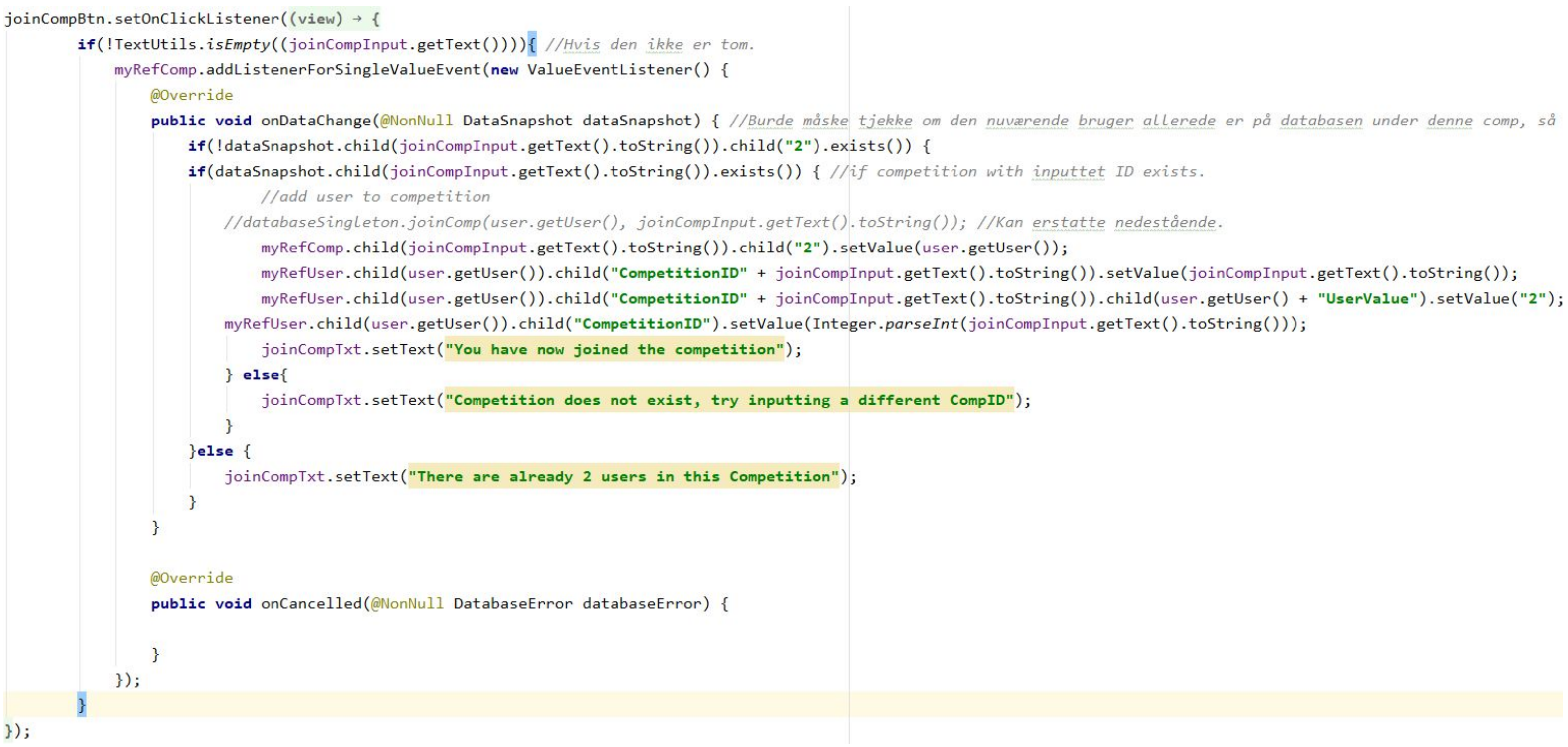
We have furthermore implemented a competition feature where the user is first able to compete with other individuals from their network. The competition feature keeps track of each of the participants' total reps, and orders them by this criteria.

The competition feature has two major functionalities. Joining a competition and creating a competition.



```
createCompBtn.setOnClickListener((view) → {
        Random random = new Random();
        final int randomCompNumber = random.nextInt( bound: 1000) + 1;
        createCompNewInfo.setText("Competition Number: " + randomCompNumber);
        myRefComp.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {//Kune tjekke om man allerede er gang med en comp, for at man ikke kan være me
                //Se om comp eksisterer, ellers lave den.
                if(!dataSnapshot.child(String.valueOf(randomCompNumber)).exists()){ //Man skal ikke skrive child("Competition"), da det er fra myRefComp
                    //databaseSingleton.createComp(user.getUser(), randomCompNumber); kan erstate nedenstående
                    myRefComp.child(String.valueOf(randomCompNumber)).setValue(null); //Læg den nye comp op på database. Kan bare ikke finde ud af at gø
                    //Så inde i workoutDone sige hvis bruger har comp, sæt værdi derind og tilføj nuværende oveni hvis den findes.
                    myRefUser.child(user.getUser()).child("CompetitionID" + randomCompNumber).setValue(String.valueOf(randomCompNumber));
                    myRefUser.child(user.getUser()).child("CompetitionID" + randomCompNumber).child(user.getUser() + "UserValue").setValue("1");
                    myRefUser.child(user.getUser()).child("CompetitionID").setValue(randomCompNumber);
                    myRefComp.child(String.valueOf(randomCompNumber)).child("1").setValue(user.getUser());
                    //Alt dette er også inde i databaseSingleton, så kan gøres gennem der.
                    createCompNewInfo.setText("You have now created a new competiton. Send this CompID: " + randomCompNumber + " to compete with them");
                } else { //Else lav nyt nummer. fordi det allerede findes.
                    createCompNewInfo.setText("this CompID already exists, press the button again");
                }
            }
```

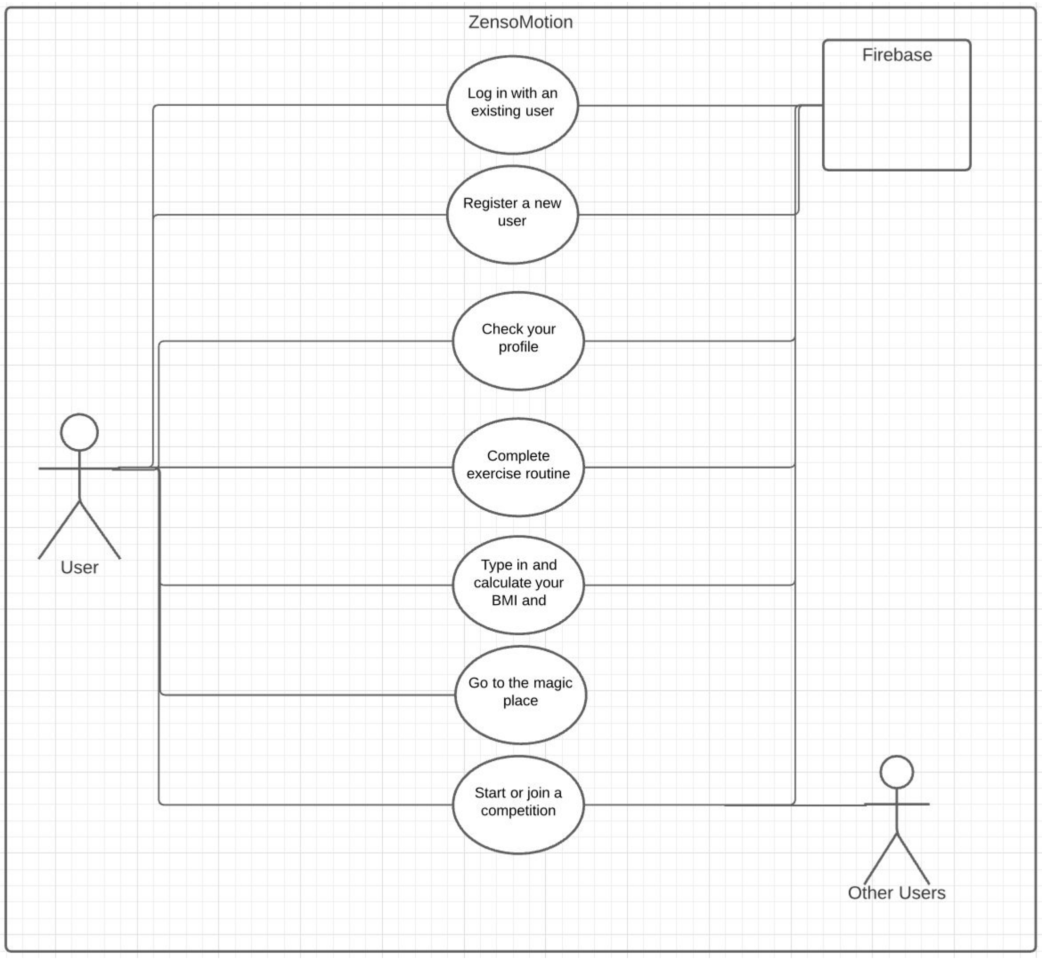
Above is the code that takes care of creating competitions. It starts off by creating a random number, between 1-1001. We add one to make sure the ID is not zero. We then check whether the competition has already been created, and give an error message if it is. If the competitionID has not been created/saved, we create the competition. This is done by adding a child to the Competition branch in our database, with the new competitionID. We then save the competitionID under the user, in the User branch of the database, whilst saving their ID in the competition, which is 1 because they started the competition. So under the newly created competition, they are user 1 whereas someone who joins the competition would be user 2. This could have been done differently, because we right now overwrite the user’s current competition when creating a new one. This could either be done by creating more competitions for one user or having the ability to sign out of a competition. Due to time limitations we have not added this feature.



```
joinCompBtn.setOnClickListener((view) → {
        if(!TextUtils.isEmpty((joinCompInput.getText()))){ //Hvis den ikke er tom.
            myRefComp.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) { //Burde måske tjekke om den nuværende bruger allerede er på databasen under denne comp, så
                    if(!dataSnapshot.child(joinCompInput.getText().toString()).child("2").exists()) {
                    if(dataSnapshot.child(joinCompInput.getText().toString()).exists()) { //if competition with inputtet ID exists.
                            //add user to competition
                        //databaseSingleton.joinComp(user.getUser(), joinCompInput.getText().toString()); //Kan erstatte nedestående.
                            myRefComp.child(joinCompInput.getText().toString()).child("2").setValue(user.getUser());
                            myRefUser.child(user.getUser()).child("CompetitionID" + joinCompInput.getText().toString()).setValue(joinCompInput.getText().toString());
                            myRefUser.child(user.getUser()).child("CompetitionID" + joinCompInput.getText().toString()).child(user.getUser() + "UserValue").setValue("2");
                        myRefUser.child(user.getUser()).child("CompetitionID").setValue(Integer.parseInt(joinCompInput.getText().toString()));
                            joinCompTxt.setText("You have now joined the competition");
                        } else{
                            joinCompTxt.setText("Competition does not exist, try inputting a different CompID");
                        }
                    }else {
                        joinCompTxt.setText("There are already 2 users in this Competition");
                    }
                }

                @Override
                public void onCancelled(@NonNull DatabaseError databaseError) {

                }
            });
        }
});
```

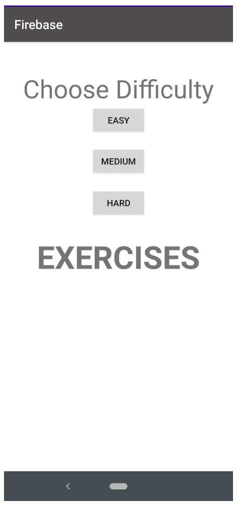
To join a competition we first check whether the EditText field is empty, as this would cause issues when fetching from the database. We then, similarly to the createComp functionality, save the competitionID under the user as this will make it easier to fetch the competition data later. We then add the user to the competition under the Competition branch, with the input ID, if it exists. If not we ask them to write a different compID. If there are already 2 users in the competition we do not add them to the competition.

Because the competition feature is a bit complex, we have decided to create a model illustrating the different actors' roles in the use case.


ZensoMotion
Firebase
Log in with an existing user
Register a new user
Check your profile
Complete exercise routine
Type in and calculate your BMI and
Go to the magic place
Start or join a competition
User
Other Users

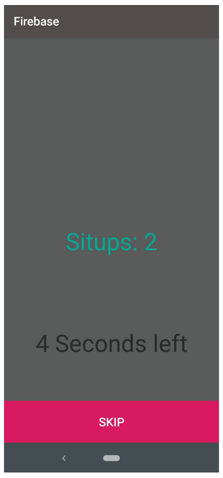
The three actors are illustrated above, showcasing the relationship between the current user, the other user and Firebase. They all play an imperative role in creating and sustaining a competition.

**Exercise Feature**



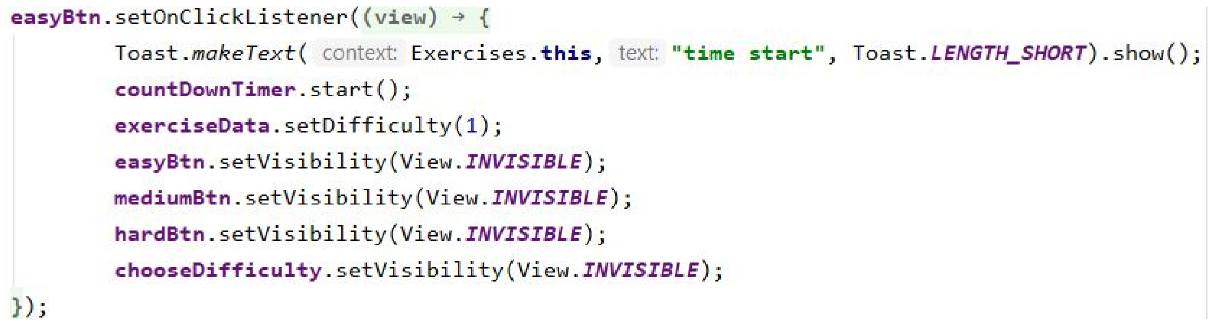
When clicking the exercise feature, you are transferred to the Exercises activity where you can choose your difficulty. This screen has a background that animates between three gradient backgrounds, whilst playing a soundbite that sets the mood for an intense workout.

You are then brought through the different exercises, where your repetitions are counted using the phone's sensors with our Zenso™ features (the code that counts repetitions). To really solidify the interactivity of the app, encouraging sounds are played depending on the amount of repetitions the user has done.



On the end screen of the workout, the repetitions in each exercise and the total amount of repetitions are shown.

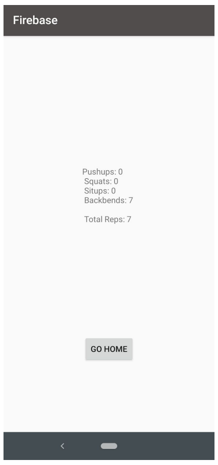
Our exercise feature is a line of activities followed by each other. It starts with the Exercises activity. You can decide the difficulty of your workout by clicking one of the three buttons on the screen.



```
easyBtn.setOnClickListener((view) → {
        Toast.makeText( context: Exercises.this, text: "time start", Toast.LENGTH_SHORT).show();
        countDownTimer.start();
        exerciseData.setDifficulty(1);
        easyBtn.setVisibility(View.INVISIBLE);
        mediumBtn.setVisibility(View.INVISIBLE);
        hardBtn.setVisibility(View.INVISIBLE);
        chooseDifficulty.setVisibility(View.INVISIBLE);
});
```

When one of the buttons is clicked, the countDownTimer is started. When that timer ends a ten second timer starts, that upon completion commences the next activity. We then set the difficulty in the exerciseData object, which is singleton, so that we can use it in other activities. Furthermore we make the buttons invisible, so they can not be clicked again. The difficulty is 1 (easy), 2 (medium) or 3 (hard) which is used in the exercises to set the duration of the exercises.

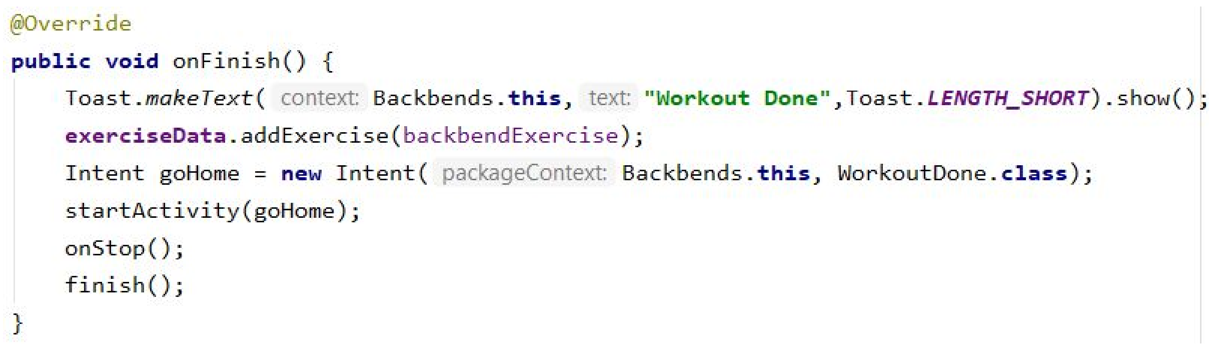
We then go through all of the exercises where we count reps. The activities are quite similar, so we will explain the overall concept, and where they differ.
Then we will explain the model that supports what happens on the screen.
Lastly, the WourkoutDone (workout endscreen) activity will be explained.



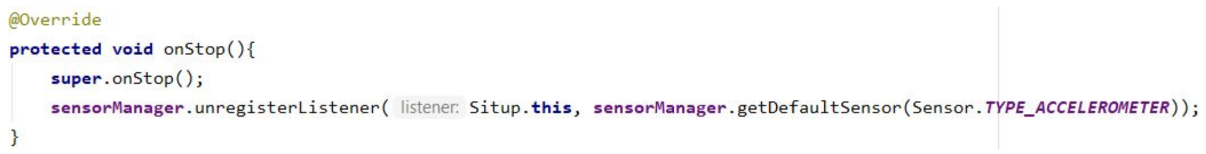
Pushups: 0
Squats: 0
Situps: 0
Backbends: 7

Total Reps: 7

Each activity has a timer that starts when the onCreate() method runs. This timer is meant for giving the user time to prepare for doing the upcoming exercise. When this timer stops the timer for the exercise starts. We also register the sensor here, so it can start to register the repetitions taken by the user. The duration of the new timer depends on the difficulty chosen on the Exercises activity. We have implemented this by having a switch case, that sets some variables that we use for the timer. The higher the difficulty, the more time one has to do the exercises.

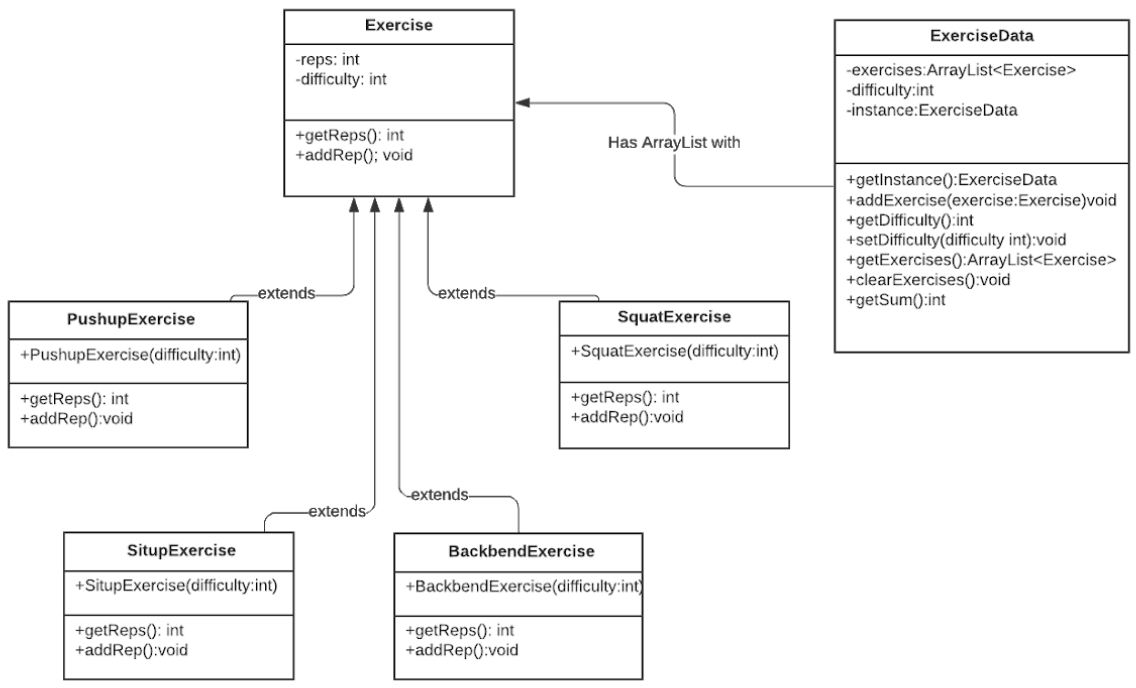


```
@Override
public void onFinish() {
    Toast.makeText( context: Backbends.this, text: "Workout Done",Toast.LENGTH_SHORT).show();
    exerciseData.addExercise(backbendExercise);
    Intent goHome = new Intent( packageContext: Backbends.this, WorkoutDone.class);
    startActivity(goHome);
    onStop();
    finish();
}
```



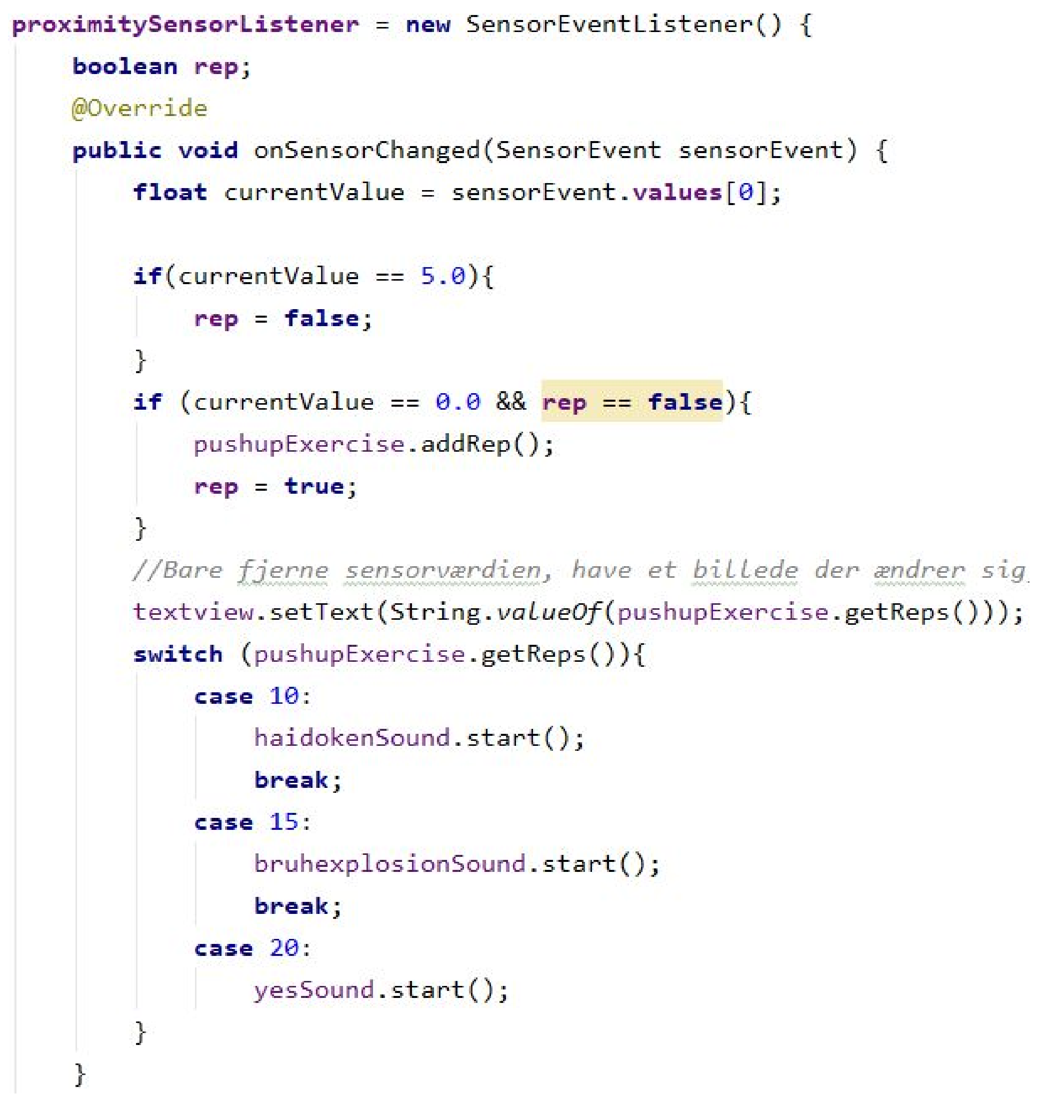
```
@Override
protected void onStop(){
    super.onStop();
    sensorManager.unregisterListener( listener: Situp.this, sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
}
```

When the last timer is done, we move on to the next activity. Furthermore, we unregister the listener (sensor) in the onStop() method. Lastly, we add the exercise (backbend, pushup, situp or squat), where we have added reps during the activity's lifetime, to the ExerciseData object's ArrayList over exercises.



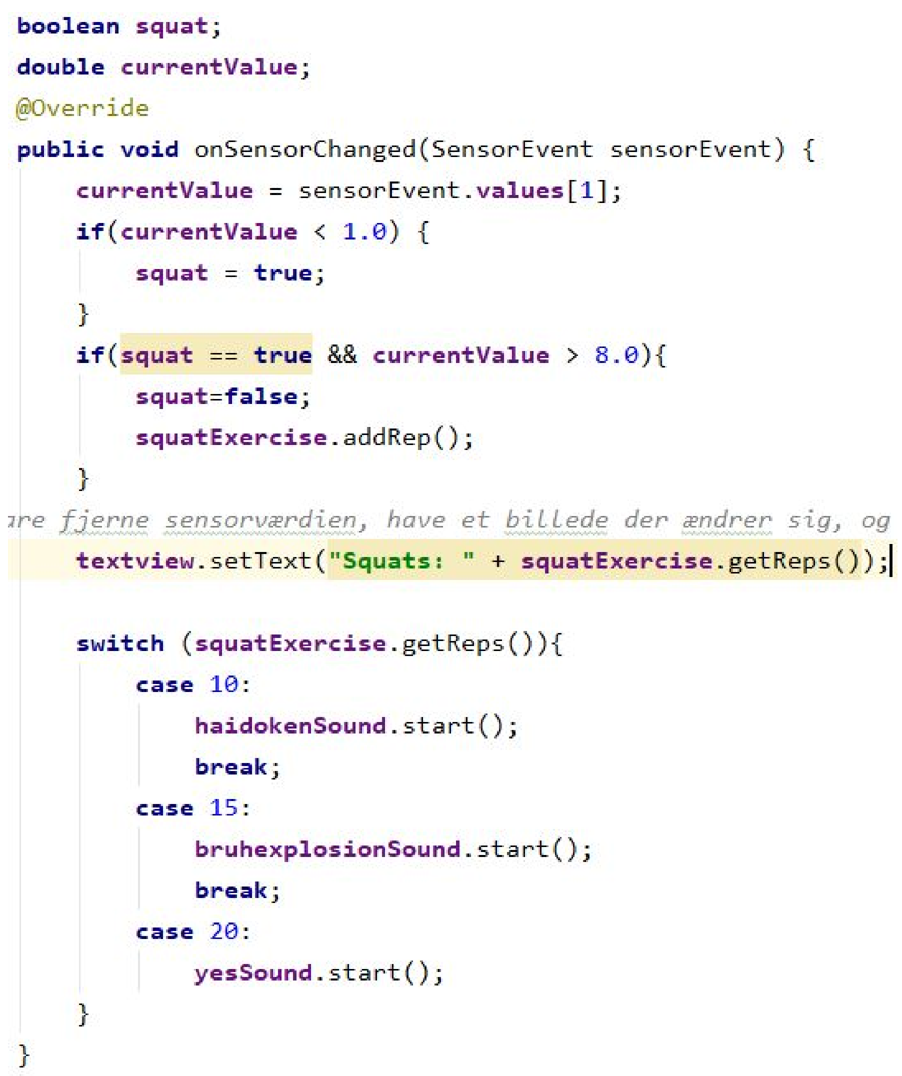
Each of the exercise activities has its own class. Each of these classes inherit from the Exercise class. We use polymorphism here, because the classes have similar traits, and for later use, when we make an ArrayList full of Exercises. Because they are all subclasses of the Exercise class, we can put them in an ArrayList with the type Exercise.

The major differences in the different exercise activities is the way we count repetitions. Firstly there is a difference in which sensors we use. Secondly, there is a difference in how we register the reps. Situp and Squat uses the accelerometer, while Pushup and Backbend uses the proximity sensor.
With both of the sensors, we register them when the first timer ends (preparation for exercises) and unregister it when we move on to a new activity.



```java
proximitySensorListener = new SensorEventListener() {
    boolean rep;
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        float currentValue = sensorEvent.values[0];

        if(currentValue == 5.0){
            rep = false;
        }
        if (currentValue == 0.0 && rep == false){
            pushupExercise.addRep();
            rep = true;
        }
        //Bare fjerne sensorværdien, have et billede der ændrer sig
        textview.setText(String.valueOf(pushupExercise.getReps()));
        switch (pushupExercise.getReps()){
            case 10:
                haidokenSound.start();
                break;
            case 15:
                bruhexplosionSound.start();
                break;
            case 20:
                yesSound.start();
        }
    }
```

To register the reps in the pushups activity, we use the proximity sensor values. If one is close to the proximity sensor, which is placed on the front of the phone (near the top of the screen), the value is equal to 0.0. We use the boolean value, to check whether the user has been away from the screen (extending their arms during the pushup), before we add a rep. Without this boolean value the reps would keep going up, whilst the user is close to the screen. In the Backbend activity we add reps when value is 5.0, as we want to register a new rep when the user is away from the screen.

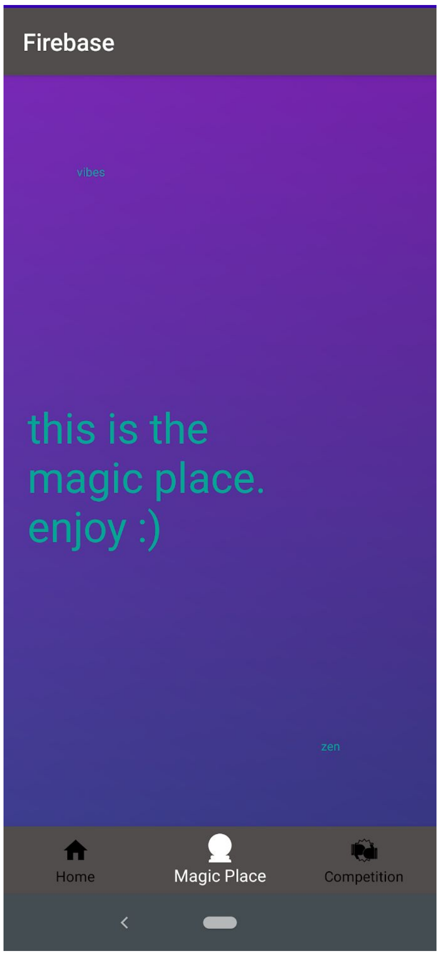


```
boolean squat;
double currentValue;
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    currentValue = sensorEvent.values[1];
    if(currentValue < 1.0) {
        squat = true;
    }
    if(squat == true && currentValue > 8.0){
        squat=false;
        squatExercise.addRep();
    }
re fjerne sensorværdien, have et billede der ændrer sig, og
    textview.setText("Squats: " + squatExercise.getReps());

    switch (squatExercise.getReps()){
        case 10:
            haidokenSound.start();
            break;
        case 15:
            bruhexplosionSound.start();
            break;
        case 20:
            yesSound.start();
    }
}
```

Similarly in the Squat and Situp activities, we use a boolean value. This is done to make sure that the user has moved from the position that adds a rep, when we add a new one. We use the accelerometer’s value on the y-axis. This can be used to check the way the phone tilts. When squatting, the phone should be in the users front pocket so we can see if they fully extend their hips when in the top position and if they go low enough on the squat. The same sensor is used, but just with a smaller tilt required for Situps.

Also we play audio when one hits certain amounts of reps, to encourage the user.

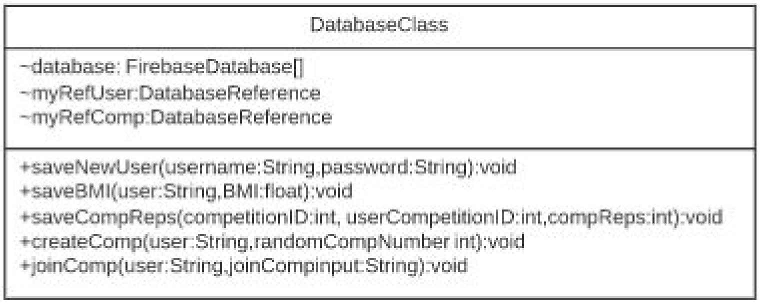
## Magic Place



this is the magic place. enjoy :)

The magic place brings the “zen” to the ZensoMotion®  app. It is a screen that implements an animation that switches between three gradient backgrounds, whilst playing Swell’s “i’m sorry”. This really brings out the vibes and zen, which is the idea behind the magic place. The sound loops, and the animation keeps going, making the magic place the perfect place to spend several hours. The audio loop stops, either when you leave the app or change activity.

### Optional Class - DatabaseSingleton



```
DatabaseClass
-database: FirebaseDatabase[]
-myRefUser:DatabaseReference
-myRefComp:DatabaseReference
+saveNewUser(username:String,password:String):void
+saveBMI(user:String,BMI:float):void
+saveCompReps(competitionID:int, userCompetitionID:int,compReps:int):void
+createComp(user:String,randomCompNumber int):void
+joinComp(user:String,joinCompinput:String):void
```

We had created the design for a class that we did not implement, due to us prioritizing getting functional features, rather than focusing on the architecture surrounding it. This would be used to save data on the database. It would also encapsulate a lot of similar blocks of code into a class and allow us to have less code in the activities, thus making the activities easier to read. Furthermore it would adhere to MVC's decoupling of the controller and the model, making the code more manageable. Also, when changing methods or pieces of code this approach would make it less error prone/help avoid bugs, as we would only have to change it in the model.

**The View of our App**

The view of our project is mostly defined in the xml files. **We can manipulate these items in our activities, that serve as a mix between a controller and a model, as they contain some logic/data.** We have, however, tried to implement some model-like classes, inspired by the MVC design pattern, that we instantiate and use in our activities. Some of the view parts of our code will now be explained.

In the **res**ource folder of our project we have a lot of the visual/audio elements of our app.
In the **raw** folder, all of the sound files are kept.
The **menu** folder holds our bottomNavigation menu that we implement in multiple activities, as a tool to navigate through the screens. There are three items, each with an icon that describes what they contain (for instance a house for the home item).
In the **anim** folder, we have different animations that we use when switching between the screens. In the drawable folder, we have the backgrounds and pictures that are featured in the app.
Lastly, in the **layout** folder we have the layouts for the different screens. As this is mostly repetitive coding with variating color -'s background choices, we will not go into great detail with this, as the functionality of the app was prioritized over the visual side.
We will, however, go through the overall concepts and choices made.
We used constraintLayout on most of the screens, as it offers better performance due to replacing nesting elements with a flat hierarchy with constraints, as opposed to other layouts (Hagikura, 2017). Furthermore, the constraintLayout makes the app easily adaptable to different screen sizes (Developer Android, 2020). A few times we used different layouts, for the flexibility they could provide when designing the interface.
We implemented a scrollview on our homeNavigation screen, to allow the user to scroll through the different menu items.

## Object Oriented Programming Principles In our Code

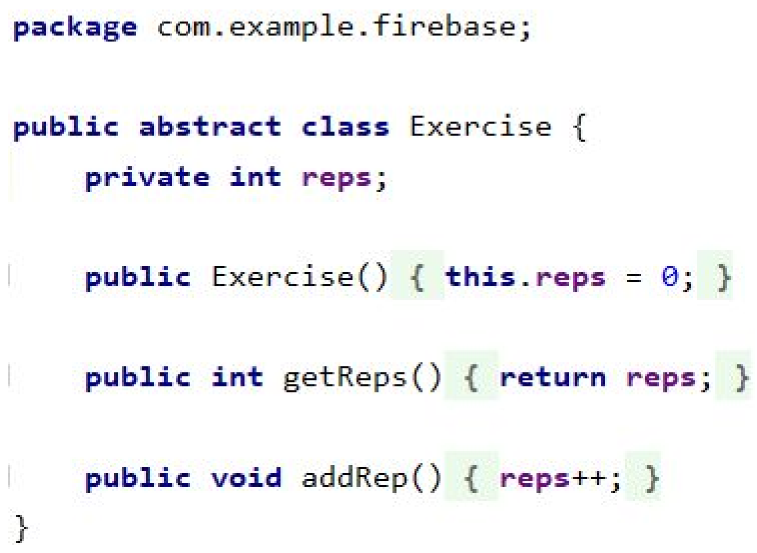
Before we started to program our application, we discussed how we could make the code as clean as possible. Since we have chosen Java as our programming language in Android Studio, it made sense for us to implement the principles of OOP. Java is an object oriented programming language, which provides features that assist in implementing an object oriented model. These features are abstraction, inheritance, encapsulation and polymorphism (Javatpoint).

### Abstraction

One of the fundamentals of Object Oriented Programming is abstraction. The idea behind abstraction is to only show the relevant data and hide unnecessary details of an object from the user. Data Abstraction involves the process of only identifying the required characteristics of an object and ignoring the irrelevant details. The properties and behaviours related to the corresponding object differentiate it from other objects. The properties and behaviours of an object can furthermore help in classifying the objects. We used abstraction throughout the code, by creating classes, where we hid unnecessary details, so that the activities had less code (tutorialspoint).

### Abstract Classes

Furthermore, abstract classes made a big impact on making our code organized, manageable and easy to understand. The abstract class can not be instantiated, and has to have at least a method. This is why we use subclasses, so that the abstraction can be useful. These classes contain different forms that are related to each other by inheritance, also known as polymorphism. This will result in some advantages, such as that the code can be reused through inheritance. Abstract classes can contain concrete methods as well and these methods can be inherited by subclasses. We can take a look down below at our code (Javatpoint).

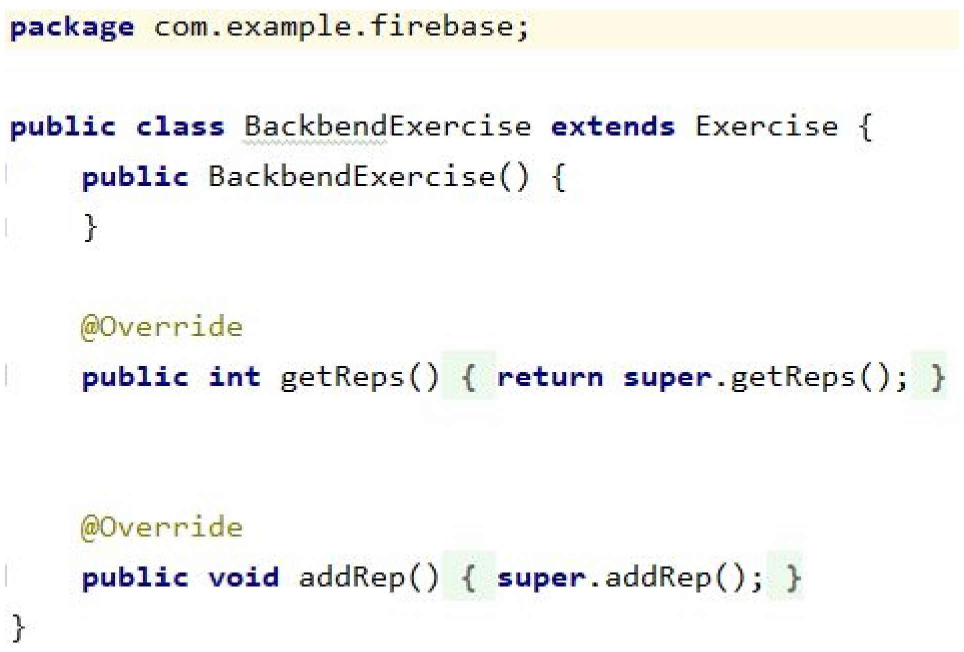


```
package com.example.firebase;

public abstract class Exercise {
    private int reps;

    public Exercise() { this.reps = 0; }

    public int getReps() { return reps; }

    public void addRep() { reps++; }
}
```

As previously mentioned the advantage of an abstract class, was that the code could be reusable. The way we have been using an abstract class was to reuse the code through inheritance, with the “Exercise” class being our parent class. As it can be seen in our code, the abstract class “Exercise” contains the attribute “reps".

The methods and attributes from the parent class have been inherited into the different forms of our subclasses “PushupExercise”, “SitupExercise”, “BackbendExercise" and “SquatExercise”.

Down below the code showcases one of our subclasses (BackbendExercise) that extends the parent class. Here we can see how the method “getReps” has been inherited by overriding the method from our parent class.



```
package com.example.firebase;

public class BackbendExercise extends Exercise {
    public BackbendExercise() {
    }

    @Override
    public int getReps() { return super.getReps(); }


    @Override
    public void addRep() { super.addRep(); }
}
```

Before the implementation of the abstract class we had to write the code from our parent class in each one of our subclasses/exercises. We only have four exercises, but imagine if we were having many more different exercises, which need the same attributes and methods. This can be really time consuming and unnecessary to do. This is why we wanted to implement inheritance into our code, it allows us to make the code reusable by extensibility through polymorphism and inheritance. Inheritance lets us inherit the attributes and methods from our Exercise class. Polymorphism uses those methods to perform different tasks. This results in performing a single action in different ways, hereby making our code reusable, manageable and less time consuming in the long run.

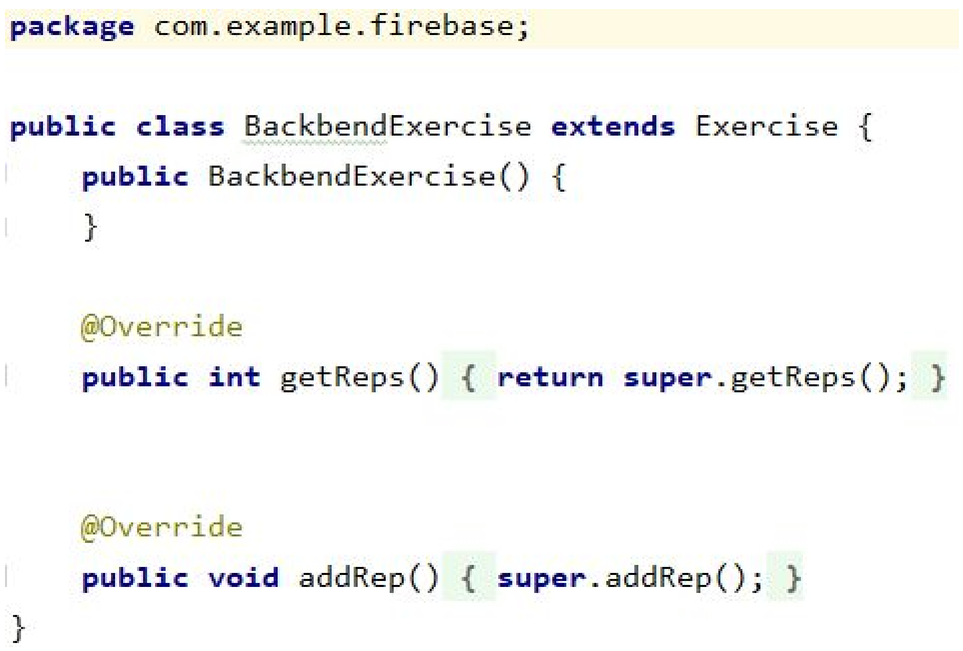
**Encapsulation**

The encapsulation principle refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of an object inside a class, preventing unauthorized parties from getting direct access to them.

It is the idea that data inside the object should only be accessed through a public interface, that is the object's methods. In order to use the stored data in an object to perform an action, we need to define a method associated with the corresponding object. This action can be performed whenever we use the method on the object.

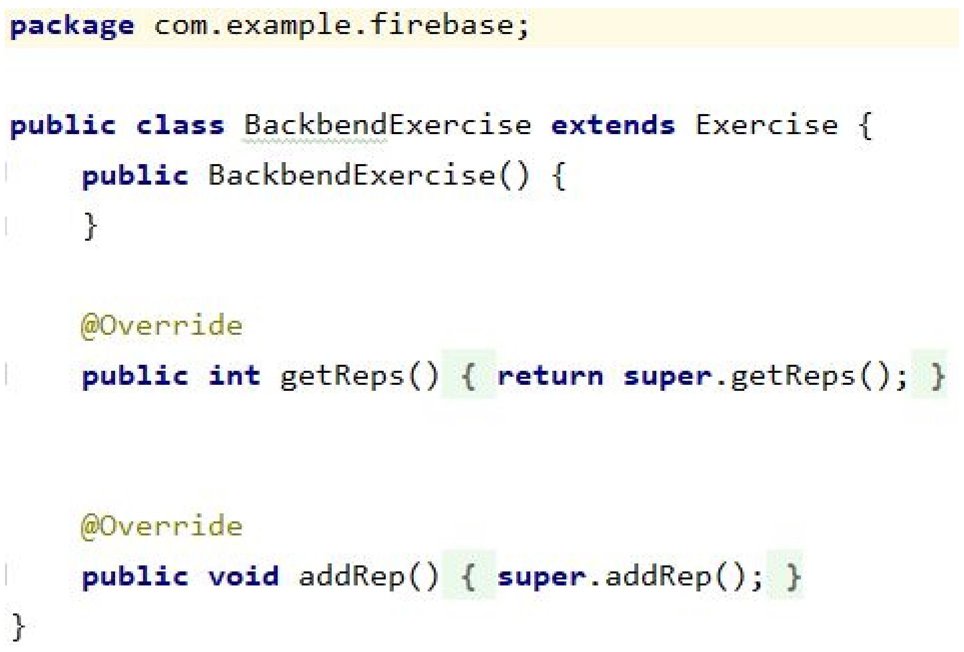
We found the use of encapsulation beneficial for several reasons. One of the reasons is that the functionality in the code is defined in one place and not in multiple places. This happens in a logical place, where we have stored all the data. Encapsulation also promotes maintenance because code changes can be made independently without affecting other classes (mrbool).

This can be seen here in our code as well:



```
package com.example.firebase;

public class BackbendExercise extends Exercise {
    public BackbendExercise() {
    }

    @Override
    public int getReps() { return super.getReps(); }


    @Override
    public void addRep() { super.addRep(); }
}
```

We can see our exercise class stores our data with the access level of a private modifier, meaning the accessibility is only within the class. This makes the data encapsed, the data of our exercise class is hidden from any other class and can be accessed only through objects of the same class in which they are declared. This can be seen in the following code associated with the object which does this.



```
package com.example.firebase;

public class BackbendExercise extends Exercise {
    public BackbendExercise() {
    }

    @Override
    public int getReps() { return super.getReps(); }


    @Override
    public void addRep() { super.addRep(); }
}
```

We can see in the code above how we have accessed the encapsulated data with the use of a public method. The use of encapsulation helps us to make the code more organized, since the function of the exercises is only defined in one place (exercise class) instead of multiple places, such as every single training activity. Also when we use a method, we only need the information of what result the method will produce, in other words, we do not need to know details about the parent object internals in order to use it on our subclasses.

**Fragments - How They Could Have Been Used**

Android also offers a piece of an activity, that enables modularity for the activities, which is called Fragments. As mentioned before, we have in our program primarily been working with Activities. You can look at Fragments as layers that are being put on activities, which give reusability and modularity into the activities user interface, which are some of the advantages among others. The disadvantages using Fragments is that it adds complexity to the code. Since it is able to more or less achieve the same things with using activities, we chose primarily to work with activities and dropped working with Fragments, due to the time we had of learning to work with fragments. So lets as an example say, that we would have used Fragments in our project. We would have implemented  our bottom navigation in our HomeNavigation activity (as it is now) which is in the started lifecycle state, and then set up the rest of the classes as fragments, instead of activities, which can be picked from the bottom Navigation and the other classes that are attached to the HomeNavigation activity. Dividing our UI into fragments would have made it easier for our activities' appearance at runtime.

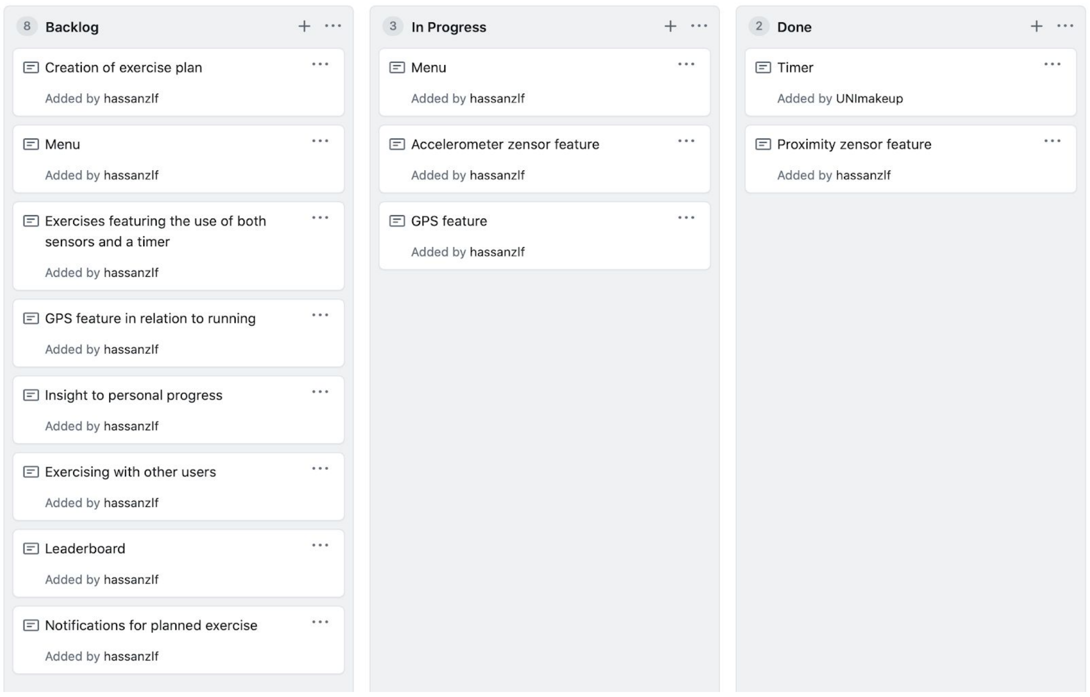
Though also keeping in mind to only provide a fragment with the necessary logic it needs in order to manage on its own, to keep the fragments from being dependent on each other (Developer Android Fragment, 2020).

# 5. Development Environment

Since we were working on a computer science project, we decided to make use of various tools in order to improve the work process and efficiency. The tools that were used were the Kanban System for the management of our project and VCS (Version Control System) - Git.

## 5.1 Github

GitHub is a service that contains access controls as well as a number of collaboration features such as basic task management. The service hosts the users source code projects in a variety of different programming languages and keeps track of the various changes that have been made by each iteration.
We used Git as our version control system on the platform GitHub. We used GitHub, as every group member had gained experience with it from our courses.
The code branches changed throughout the development process. Each member of the group worked on their own branches, where we combined useful code throughout the project.


8 Backlog

Creation of exercise plan
Added by hassanzlf

Menu
Added by hassanzlf

Exercises featuring the use of both sensors and a timer
Added by hassanzlf

GPS feature in relation to running
Added by hassanzlf

Insight to personal progress
Added by hassanzlf

Exercising with other users
Added by hassanzlf

Leaderboard
Added by hassanzlf

Notifications for planned exercise
Added by hassanzlf

3 In Progress

Menu
Added by hassanzlf

Accelerometer zensor feature
Added by hassanzlf

GPS feature
Added by hassanzlf

2 Done

Timer
Added by UNImakeup

Proximity zensor feature
Added by hassanzlf

A group member took a task from the Kanban board.

Kanban gives a more sufficient overview of the completed, ongoing and future tasks that are yet to be concluded (Github Project Boards).

## Network Theory

Everett M. Rogers developed a theory defining a variety of factors that have a greater influence in a potential change of behaviour. Rogers sees the individual's behaviour being more adaptable to change, if it is likewise adapted by entities from their network (Dalum et. al 2000:34-35). The theorist furthermore distinguishes between two different network types. First, he defines a "close network", consisting of the entities the individual has a rather close affiliation with. This primarily includes family members and close friends (Dalum et. al 2000:36).
Lastly, he defines a "far network", consisting of entities with whom the individual feels no strong bond (Dalum et. al 2000:36).
Rogers states that both networks have the ability to influence an individual, but that the near network is more likely to cause a change in behaviour (Dalum et. al 2000:37).

This theory has greatly influenced our application, and the choice to include a competition feature that focuses on implementing a social aspect that allows the user to interact with individuals from their near network. The inclusion of this aspect should increase the chance for users that seek to exercise more often to achieve their goal, as they should be more motivated to continuously use the application. Users that already exercise should furthermore be more motivated to continuously make use of the application as a result of their interactions with individuals from their near network.

## 6. Usability Testing

In usability tests, potential users complete a variety of tasks that represent the intended usage of the app. It is furthermore preferable to perform these tests in an environment that matches the intended usage (Lazar, 2017:263).
The test seeks to improve the interface's overall quality by locating possible flaws, and improvable areas.

The number of included users in these tests can vary. A usability test can involve thousands of users, and can likewise also be narrowed down to a handful of users (Lazar, 2017:265).

A moderator should decide upon the overall representative audience, the tests should focus on. At this point it is also crucial to establish the needed number of users, in order to gain a sufficient amount of feedback from the tests. It is often believed that five users manage to locate about 80% of the overall issues with the interface (Lazar, 2017:275).

Lastly, one of the most essential **stages** in preparing a usability test is the creation of a set of task lists. A task list should consist of a number of goal-oriented tasks, that are centered around the user exploring multiple facets of the interface, as well as exploring the essential elements of the software. The main rule in regards to these tasks, is that they stay clear, with no need of further explanation. The user should simply be able to understand the task, with no additional information. (Lazar, 2017:286)

**Our Usability Tests**

We performed a total of four usability tests, each with an individual from our personal network. The tests were performed two weeks before hand-in, and the program was therefore not finished. It was a prototype that was used to test the overall concept and usability of the app.

The users were first of all given a number of tasks that we deemed to be self-explanatory. These tasks were furthermore not supplemented with a guide, because we wanted to analyze the user's interaction with our application.

**In the creation of the tasks, the choice of participants, and the determination of the location, we chose to focus heavily on Lazar's guidelines regarding the performance of a usability test.** We first concluded that the location of the tests had to represent the intended usage of the program. It was therefore decided that we should perform these tests within a closed environment, representing the user's homes. This proved to be essential, as the main focus of the application is being able to complete a larger variety of exercises at home.

We furthermore chose to exclusively include users from our intended audience. Some of our users are therefore individuals that focus heavily on exercising in their daily life. We however also chose to include users that do not exercise on a weekly basis, as our demographic also includes individuals that are seeking to train more lightly.

Lastly, we focused heavily on implementing Lazar's ideals in the creation of our task list. There were therefore created a number of goal-oriented tasks, that first and foremost did not need any additional explanation, and secondly, were centered around the essential aspects of our application.

The informants' interactions with our prototype were documented, whereafter they were asked about various aspects of the application.

**The Execution of the Tests**

These tests were performed in the homes of the users, as it represented the intended usage of our application. We installed the prototype on the mobile devices of the informants, and gave them a set of tasks to perform. The tasks were as previously mentioned self-explanatory, and were as follow:

1. Create a profile and login
2. Enter your height and weight under Insights
3. Complete an exercise
4. Check your personal profile

(Bilag 5)

We furthermore followed multiple self-tailored guidelines that primarily drew inspiration from Lazar. These guidelines were implemented to ensure the quality of the tests. The guidelines were as follow:

1. Observe and write down potential problems the user faced while making use of the application

2. Keep the description of the tasks, and additional help to a minimum. The user should work individually with the application

3. The questions are quite broad, so there's a possibility to ask the user supplementary questions. These supplementary questions are based on the user's answers.

4. The tests should be documented through written notes.

(Bilag 5)

After completing the tests, we performed in-depth qualitative interviews with the informants, and asked a number of questions in relation to the prototype, a final product and their experiences with already existing fitness applications (Bilag 5).

**The Prototype**

When creating the prototype, we found it imperative to narrow down the functionality, and exclusively focus on the elements with utmost importance. These elements were to represent our vision for the application, as well as give the user an overview of the intended use-case. These exclusions were primarily made due to the time-limitation of the project. We found the performance of the tests, and their timing crucial, as it allows changes if deemed necessary.

After an analysis of the intended functionalities, we concluded that the application should first and foremost be centered around the user creating a personal profile. This is introduced to the user at the very start of the application, and is necessary for further interactions with the program. The user is then, after successfully creating a profile and logging in, introduced to a menu with the options. Firstly, the user is able to check their personal profile. Secondly, there is an option to exercise. Upon clicking on this option, the user is guided through a routine containing a variety of exercises that are centered around resistance training. These exercises include push-ups, sit-ups, squats and backbends. Finally, the user is able to click on a menu option called "insight". This menu option allows the user to input their height and weight, in order to calculate their Body Mass Index (BMI).

These were as previously mentioned the functions we deemed essential for the understanding of the application. The previously mentioned tasks are therefore centered around these very functionalities, and ensure that the user explores every aspect of the prototype.

**The results of the usability tests**

We performed a total of four usability tests, with individuals whose experience with similar applications, and training in general varied. After performing these tests, and further analysing the results, we were able to form multiple conclusions in relation to the prototype.

We first concluded that the overall ease of use of our application was quite high, as the users not only encountered no problems while working with the application, but also elaborated that the level of usability, specifically in relation to the main menu, was rather high. Only one user commented that they would find a simple tutorial beneficial, so this could have been implemented as one of the features of the app. However, the social aspect of the app still needed a lot of work, and was therefore prioritized.
We were furthermore also able to conclude that there were underlying issues with our visual design. Every informant commented on our visual design, elaborating that it would be beneficial if it saw some changes. One informant even explained that the application looked more like a mobile game, than an application that focused on exercising (Bilag 1).

We were furthermore able to see a pattern in relation to the different informants' opinion on the usage of the application. All informants agreed that the application could use a wider variety of exercises. However, the informants that exercised more in their daily life, focused more on this point. One informant even concluded that the application was in no way usable, due to the lacking intensity of the exercise plan.

"The training is not sufficient in any way. It needs to change as I previously specified. If there were concrete routines that were first of all a lot more intense, and routines that were approved by specialists I might reconsider, but even then I might be a little opposed." (Bilag 4)

The user was here asked to specify why they would not make use of the application. It was furthermore previously stated by the informant, that they not only heavily focused on their training, but also that they specifically made use of multiple fitness applications, including 7-minute workout, the Fitness World application and Apple's own application on the Apple Watch (Bilag 4).
The users that were more open to using the application, were furthermore currently not very active in their training. These very informants also tended to see the application being used as a part of their training. Quite a few of them were however hesitant on the application being the main source of training, and saw more potential in the application as a supplement that could be used on days where a user could not find time to train. This informant even defined the application's time-flexibility as one of its biggest strengths.

*“There’s actually multiple aspects I really like. The thing I like the most about the application is the fact that a training exercise is actually not too time consuming. Currently I have to set time aside to go train, but this application would make it quite easier.“* (Bilag 1)

Most of the informants furthermore wished that the training routines differentiated between different areas of the body, and allowed the user to choose after personal preference. The training routines were defined as being too wide, and not allowing the user to adjust the routines after personal needs.

*“In order to be used as the main training gadget, it needs a deeper focus on different kinds of training, and exercises that focus on different areas of the body. For instance exercises that are specifically for your arms.”* (Bilag 3)

Three of the four informants furthermore explained that the addition of a social aspect where the user would be able to get an overview of the routines other users were focusing on, would greatly benefit the app, and even motivate users to actively make use of it.

*“Maybe if one could suggest a routine or a few exercises to friends, or see how strangers and fitness experts are using the app. “* (Bilag 3)

The informant sees the potential in the user being able to suggest routines to individuals from their personal network, or gain inspiration from users that are experts on the field.

Multiple informants also complimented the application’s ability to calculate and keep track of the user’s Body Mass Index. The first two informants both complemented the feature, although commenting on the lack of the ability to insert age in relation to it.

It is worth noting that the final prototype that was handed in, in relation to this report, was impacted by the achieved results from the first four tests. One noticeable difference is the overall layout of the application, which was initially heavily criticised. Additionally, we added a competitive feature as a social aspect.

**Sources of error**

In regards to the execution of our usability tests, it can be discussed whether or not it would have resulted in a different outcome if the chosen users were not individuals from our personal network. The users having personal relations with the creators of the application, could potentially lead to a less critical view, and a more optimistic look on the overall product. We however, chose to exclusively include individuals from our network, as a result of Covid-19 only allowing us to interact with a small circle.

Another factor that could have potentially led to a different conclusion, was the number of informants that were included in our usability tests. It can be discussed whether including a wider number of users would have led to a higher level of representation, that could eventually have influenced our final prototype. We were however to a certain degree limited in regards to these tests, and were forced to limit our tests, due to the lack of time and Covid-19.

We also excluded all quantitative data in the performance of our tests. It could be discussed whether performing a series of quantitative tests at the start of the semester, focusing on the intended consumer's interest in regards to a fitness application, would have led to a different conclusion.

# 7. Conclusion

From the beginning of this project we sought out to make an interactive fitness application that made use of Android mobile devices' sensors. Our choice of sensors was decided by the variety of resistance training exercises that were adapted into the application. These prioritized exercises were pushups, squats, sit-ups and backbends.

In the development of the application we made use of multiple OOP principles in order to make a more manageable codebase. This was decided with the thought of extension of a larger variety of exercises in mind. This is the reason we implemented an abstract class, because an abstract class is like a template, so we could extend and build on it. Inheritance allowed us to use properties and methods of the already existing class, while polymorphism helped us perform the task in multiple ways in regard to having different forms of exercises.

One of the essential aspects of this fitness application is the social workout experience, where the users can enter a competition against one another to see who the better performer is. But we managed to resolve the small tasks we faced in the process and ended up creating the interactive part with Firebase authentication and Realtime database. This enabled us to create a user system, where the users can compete with each other.

Certain parts of our code has been inspired by the MVC design pattern. We could have adhered more to the concept, as it did provide us with models that could be reused.

We used sensors to register the repetitions of the user, whilst sounds were used as encouragement when hitting certain amounts of reps, creating an interactive experience.

The feedback from the usability tests made us reevaluate the layout. Furthermore we decided to include social features, in the form of competitions. The test was done in an environment resembling that of an actual use case, whilst using informants of varied experience levels of exercise. This gave us a representative result, with feedback that could help us refine our design

# 8. Literature

Anderson J. D & Carmichael A. (2016). *Essential Kanban Condensed.* Seattle, Washington: Lean Kanban University.

Britton, Carol & Doake, Jill, 2005 A Student Guide to Object-Oriented Development; O'Riley

Chacon and Straub (2014). *Pro Git.* published by Apress

Dalum, P. Sonne, T. F. Davidsen, M., Sundhedsstyrelsen (2000), *At tale om forandring,* http://sundhedsstyrelsen.dk/~/media/E9AAB1FA544749E292495FEE444034D9.ashx

Developer.Android (2020), *Fragments,* Android Developer, https://developer.android.com/guide/fragments (accessed 18-12-2020)

Developer.Android: 2020, *Support different screensizes,* Android Developer, https://developer.android.com/training/multiscreen/screensizes (accessed 17-12-2020)

Firebase (2020), *Realtime Database,* Firebase, *https://firebase.google.com/docs/database/ios/structure-data* (accessed 18/12-20)

Flutter, https://flutter.dev (accessed 13-12-2020)

Github Project Boards (2020), *About project boards*, Github docs, https://docs.github.com/en/free-pro-team@latest/github/managing-your-work-on-github/about-project-boards (accessed 17-12-2020)

Hagikura, Takeshi: 2017,*Understanding the performance benefits of ConstraintLayout* Understanding the performance benefits of ConstraintLayout (accessed 17-12-2020)

Javatpoint, *Java OOPs Concepts,* https://www.javatpoint.com/java-oops-concepts (accessed 14/12-20)

Lacey, Matt (2017) *3 reasons to use the MVVM pattern* (05-12-2020) https://www.mrlacey.com/2017/04/3-reasons-to-use-mvvm-pattern.html

Lazar Jonathan., et al.; 2017; *Research Methods in Human-Computer Interaction 2nd Edition* https://books.google.dk/books?hl=da&lr=&id=0fx_CwAAQBAJ&oi=fnd&pg=PP1&dq=what+is+android+studio&ots=zFzBKE5KAE&sig=JXdaXAZeWlJeRGt7Yv4Vp4cnRXU&redir_esc=y#v=onepage&q=what%20is%20android%20studio&f=false

Microsoft; 2020, Overview of ASP.NET Core MVC https://docs.microsoft.com/da-dk/aspnet/core/mvc/overview?view=aspnetcore-3.1 (accessed 17-10-2020)

Tutorialspoint, *Java abstraction,* https://www.tutorialspoint.com/java/java_abstraction.htm (accessed 10/12-20)