

How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: “**Capstone_Stage1**”
3. Replace the text in green

Submission Instructions

1. After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
3. Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”

GitHub Username: lalok

FitSim

Description

FitSim is a fitness simulation that lets you build up your virtual athlete. As in real life you let him train different muscles and give him food to eat. Your athletes muscles will grow and you will become stronger. But be carefull. When he will get to much food he might become fat. FitSim tries to illustrate all the effects that sport and nutrion as good as possible.

Intended User

This app is primary intended for young people that are interested in fitness in general or just enjoy doing some kind of simulation applications. This might be students, pupil or also older people like employees.

Features

Main features:

- Train your virtual Athlete
- Take food and observe your calories
- Calculates highscore based on your stats
- Do some usefull calculations: BMI, FFMI, Metabolic Energy

Implementation notes

The app was planned for release in Playstore this summer but recently my company communicated that the nanodegree certificate is absolutely required until 10th of march. Because of lack of time I had to recude the functionality of the app primary to present the obtained knowledge and skill and to meet all the required specifications.

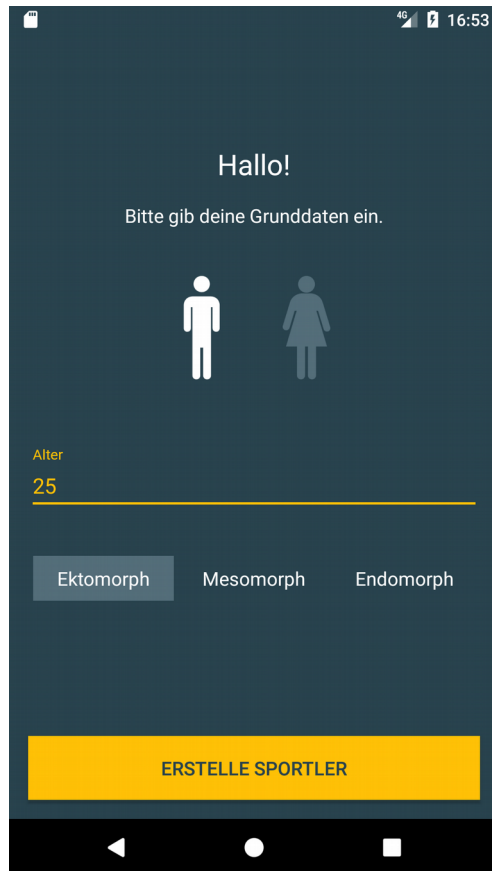
At the moment there are only a few activities and nutrition available to demonstrate the workflow. All available resources are only stored locally in assets folder as json files. Of course it would be better to provide them via a REST-API.

Actually activities and nutrition will not block the athlete for specified time. Later, when doing an exercise, the athlete will be blocked until finished and only afterwards the values will be changed. This behavior also applies to nutrition. Like in reality the athlete will than take some time to digest.

Because there is no backend available right now I had to simplify the data requests that are displayed on the home widget. Instead doing some queries against an api I only request some highscore values from the firebase real time database. This might of course be absurd because they could already be updated when running the app.

User Interface Mocks

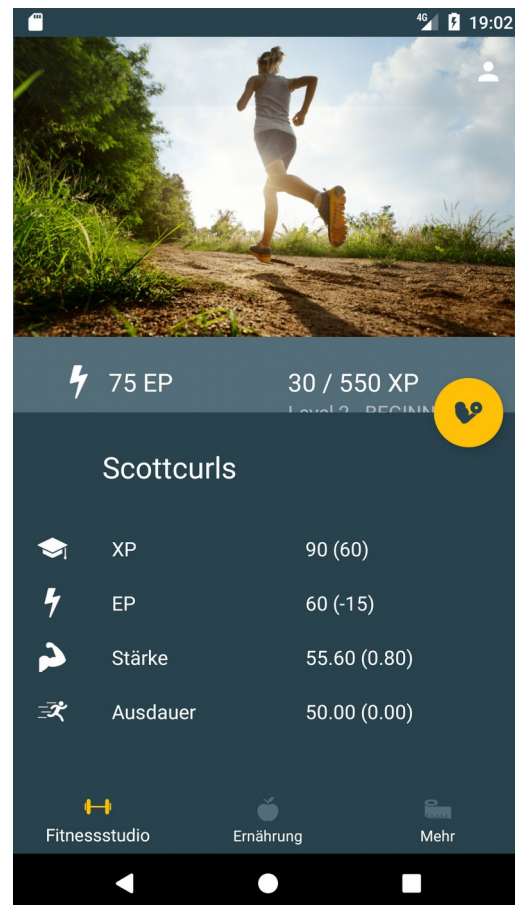
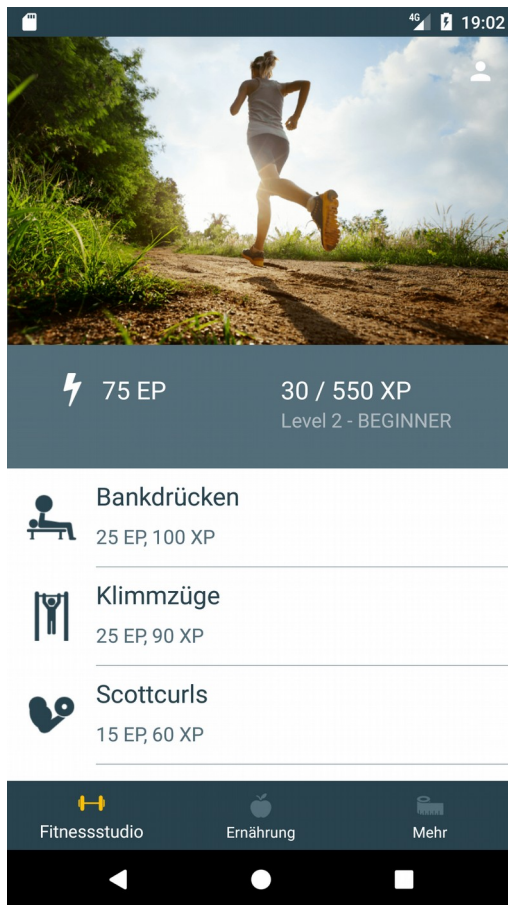
Onboarding

A mobile app onboarding screen with a dark blue background. At the top, it says "Hallo!" and "Bitte gib deine Grunddaten ein." Below this are icons for a male and female user. A yellow label "Alter" is above a text input field containing "25". Below the input field are three radio button options: "Ektomorph", "Mesomorph", and "Endomorph". At the bottom is a large yellow button labeled "ERSTELLE SPORTLER". The screen is framed by a black border with a white status bar at the top showing "4G" and "16:53", and a white Android navigation bar at the bottom.

This screen is displayed when there is no athlete created. The user has to set some basic data like gender, age and its related body type. For now only the body type has effects on the users athlete like the amount of food he can eat and how fast the strength and/or stamina improves after doing exercises.

The button on bottom leads to the next screen.

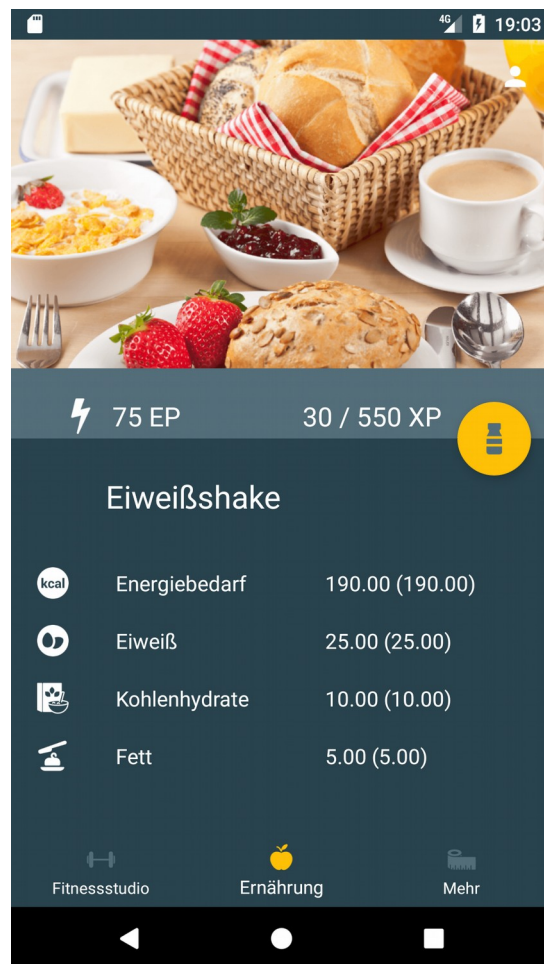
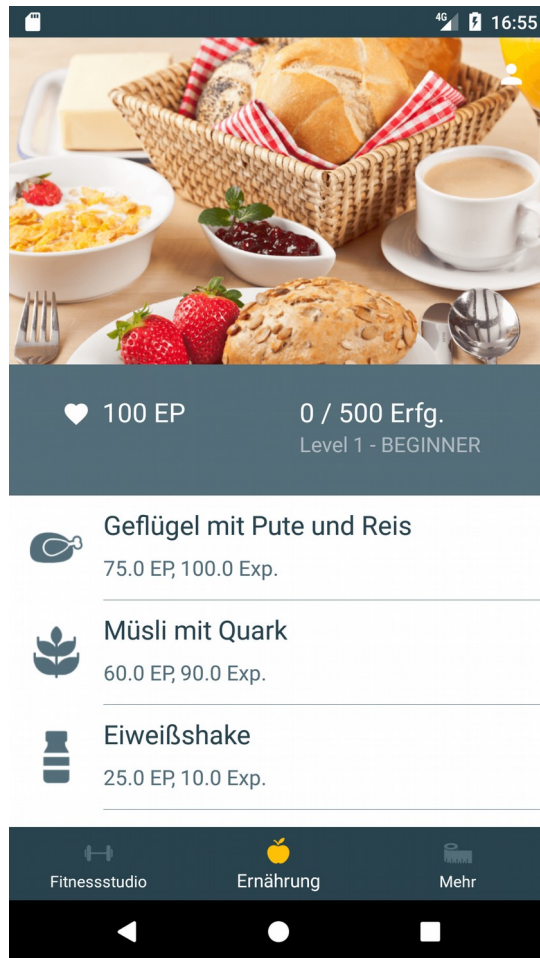
Gym-Screen



Here the user can do some exercises to improve the athletes muscles, strength and stamina. You can select an exercise and start the selected one. After selecting an activity, a bottom sheet appears which displays all changed values based on the values of the exercise. After starting an activity, all influenced values will immediately change as displayed in preview. When required experience points are reached, an interstitial ad will be displayed and the athlete will increase its level.

When there is not enough energy remaining for performing the activity, a toast will be displayed. Exercises that cannot be executed are marked grey.

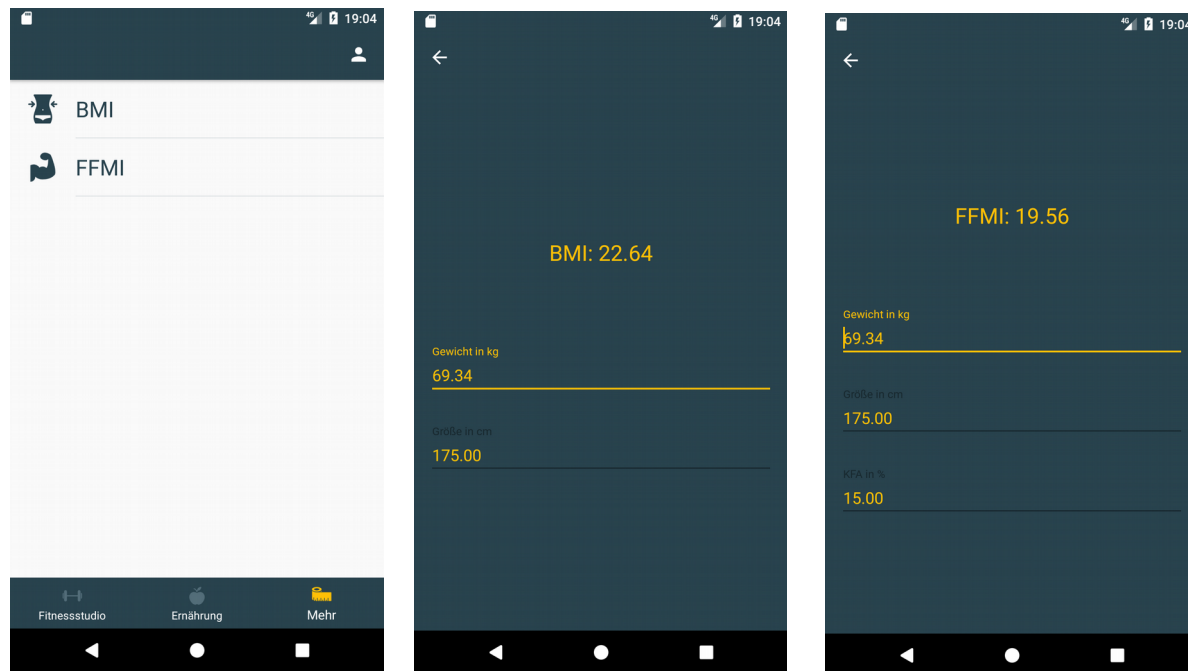
Nutrition-Screen



The second screen allows the user to take some food. This is primary required that the athletes weight will not reduce. Eating to much will also rise the weight. Some food like energy or coffee will even increases the energy points.

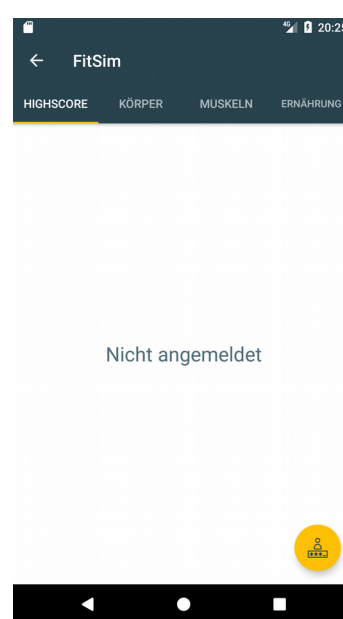
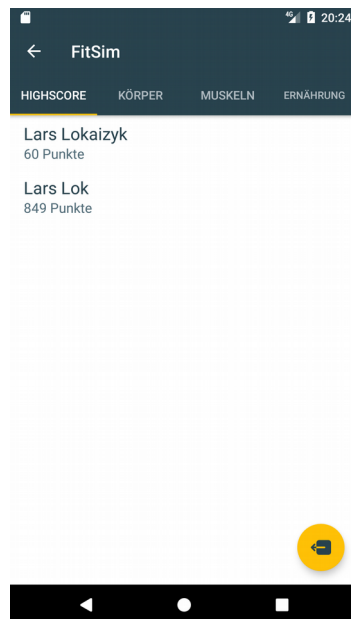
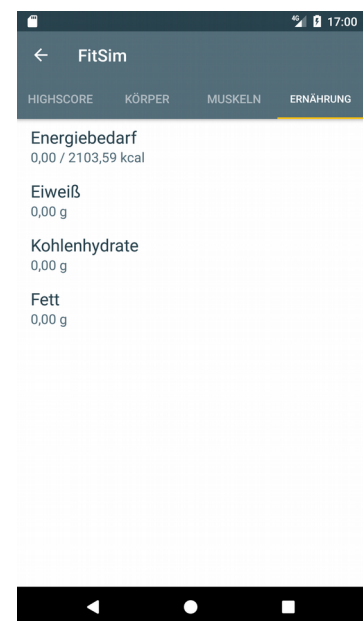
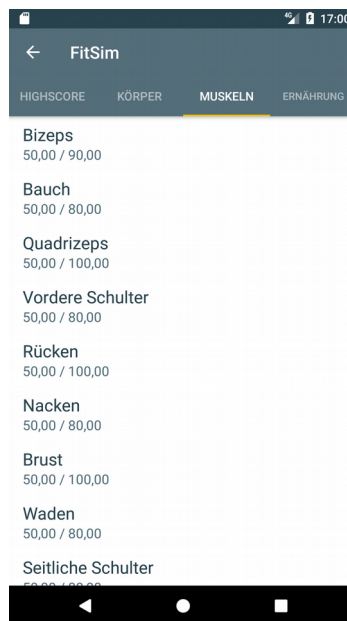
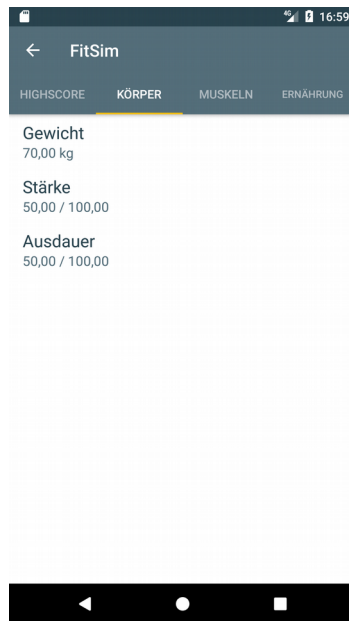
Like on gym screen, all influenced values are displayed as preview on a bottom sheet after a nutrition is selected.

More-Screen



This is just a screen that has some useful utility functions like calculating the BMI and the FFMI. For every utility function, a new screen is launched that allow to enter some values and directly watch the calculated result.

Profile-Screen



The profile screen can be accessed by pressing the profile icon which is displayed on gym screen and nutrition screen in the top right corner.

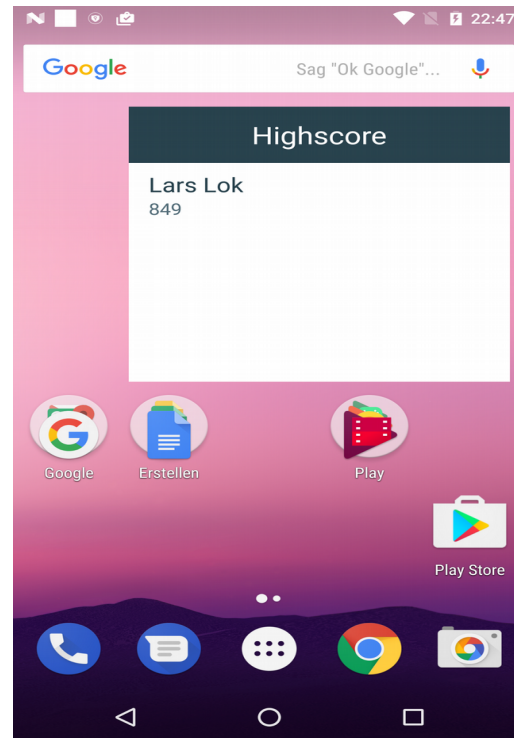
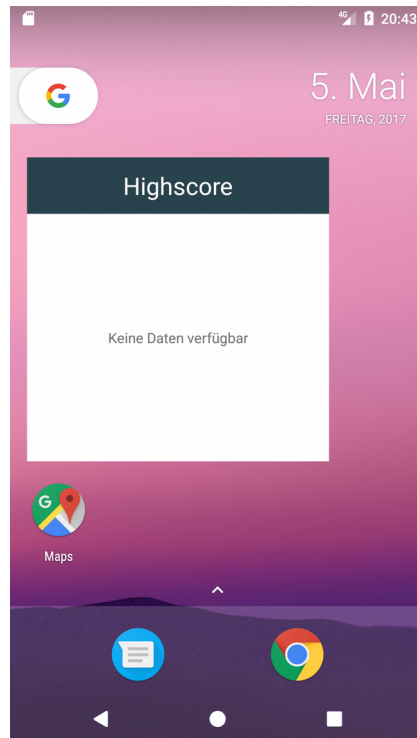
This screen displays several info about the athlete. The user can swipe between the available info screens.

The first screen displays the highscore but only when the user is logged. If not logged in, the user can login via firebase authentication.

The other sections only display some info about the athlete

- muscles
- body stats (weight, strength, stamina)
- calories (required/consumed calories, consumed proteine, carbs and fat)

Gym-Screen



The widget displays the high score list which is also shown in high score screen. When the user is not logged, an info view is visible.

Key Considerations

How will your app handle data persistence?

To initially demonstrate the apps functions there will be some data provided by locally stored assets as json. Later they can be replaced by online resources and the parsing of json would be similar.

All athlete related values will be stored in a local SQLite database. The access to this data is simplified by using an object relational mapping library. In addition the app will calculate a high score based on the current stats of the athlete. To keep them up to date they will be synced immediately via firebase real time database. This sync is only done when the user is logged in.

Furthermore I use the shared preferences to simply store some key value pairs that indicate the execution of dispatched jobs or if there has been an athlete created.

Summarized persistence methods:

- Filesystem: Available ressources (Activities, Nutrition)
- Shared-Preferences: Simple key-value pairs
- Firebase Real-Time DB: Users highscore points
- SQLite DB: Values of the athlete

Describe any corner cases in the UX.

There are no unusual cases in the UX. The navigation hierarchy is kept simple and flat. There are only two levels. The first level is the main screen that uses the bottom navigation pattern. As proposed in material design there is no back stack established when switching between the entries. From main screen the user can access a second hierarchy either by entering the profile screen or when selecting a calculation. From the second level the user can go a step back by pressing back button or the back navigation icon, displayed in the toolbar.

Describe any libraries you'll be using and share your reasoning for including them.

Mvp-lib:

This is a custom library that was developed by me and some other developers at our company. Its main purpose is to implement the application structure based on the MVP-Pattern. This makes all Activities and Fragments much more manageable. All user interactions are now moved from the app components to a related presenter. The presenter is directly connected to the components lifecycle.

Also the library has a simple implementation to run some background tasks. This implementation uses asyc task loaders to support running request while the user rotates the screen.

Another feature is to simplify the handling of saved instance state. This can simply be done by annotating parcelable fields and automatically whose state should be saved.

And finally there are some useful things that make development easier and save some boilerplate code.

Dagger 2:

This is a dependency injection library that allows to separate the creation and instantiation of objects. This also simplifies the injection of depending objects and makes components more clear.

Firebase:

The app uses several firebase libraries (Analytics, Real-Time DB, AdMob, Auth/Auth-UI and Jobdispatcher. (Described in Google Play Services section)

Greendao:

This is an object relational mapping library which simplifies the usage of SQLite databases. Stored data can just mapped to related objects and vice versa.

Timber:

A simple logging library. Primary used to remove the manual writing of log tags and to ensure that log statements are not logged in release builds.

Gson:

Gson is used to map structured json data to POJOs.

Describe how you will implement Google Play Services.

All used google play services are implemented by using firebase.

Analytics:

It is used to track some basic events like app usage time, installations or when users remove the app. Furthermore I added events for screen tracking and for activities/nutrition items used by the users.

Real-Time DB:

This is implemented to store all high score points and display changes instantly.

AdMob:

To not overload user with ads I only show an interstitial ad when an athlete reaches a new level.

Auth/Auth-UI:

I use this this to allow user to register with an account. This account is a requirement for storing its high score points.

Firebase-Jobdispatcher:

FitSim requires some periodic tasks that must be executed e.g. to refresh the athletes calories. These tasks are scheduled by the firebase-jobdispatcher because it also supports older devices, the app gets notified when a job was not scheduled and it is quit easier to implement compared to a sync adapter.

Furthermore it is used to schedule the updating of the widget.

Next Steps: Required Tasks

Task 1: OOA/OOD

1. Analysis of the Business Logic

The simplification of doing sport and nutrition out of real life is a core function of FitSim. This absolutely requires to do an object oriented analysis of the interaction of all components before starting with implementation. This is probably a different approach because nowadays most apps only use simple data structures. But only using data structures would complicate the interaction a lot. E.g. only when an athlete is doing some sports there are a lot of dependencies between the Objects: Athlete gains experience and loses energy. Eventually also a new level was reached. Than the activity strains some muscles, improves strength, increases the required energy and so on.

2. Application Structure

Defining the access to the business layer which here is provided by the repository pattern.

Task 2: Define all features and screens

Task 3: Create Project

- initialize local git repository
- create android studio project
- create keystore and signing configuration
- add third party libraries

Task 4: Implement Business-Logic

- Athlete components (Body, Stats, Fitness,...)
- Athlete itself as facade
- Activities/Nutrition as command-pattern
- Write unit tests for business objects

Task 5: Implement Data-Access-Layer and Repositories

- Abstract implementation for all DAOs
- Single implementation for Athlete, Activities, Nutrition and Muscles
- Add mock implementation for first usage and tests
- Db object models and Dagger injections

Task 6: Main Screen and Fragments

- create simple functional layout
- Bottom navigation
- Selection of menu items
- Loading of Athlete for all Extended screen fragments
- create firebase console project and connect to app

Task 7: Gym Screen

- Load list of activities and display in list
- Persist changed athlete after performing an activity

Task 8: Nutrition Screen

- Load list of nutrition
- Persist Athlete after nutrition was taken

Task 9: Profile Screen

- Firebase authentication
- High score section
- Login and logout via firebase auth-ui
- use custom style for firebase auth-ui
- Display all athletes in remaining sections

Task 10: Widget

- add all implementation required classes (widget provider, provider xml, item factory and service
- add scheduled job for updating the widget
- run intent service to load data in background and send broadcast when finished

Task 11: Onboarding

- add repository for creating new athletes
- validate user input
- create athlete and finish screen

Task 12: More Screen and Utility Screens

- add list of utility features
- add screen for every features

Task 13: Finalize

- add custom events to firebase console
- track analytics event
- check correct state handling
- check strings and context descriptions

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"