# Overview

SimpleGymTracker is a user-friendly app designed to track workouts and make use of past workouts or programs for easy planning and tracking. The app prioritizes simplicity and usability in a real gym setting, even if that means limiting versatility or features for the sake of ease of use.

The app uses both native Android and web browser interfaces. The native app maximizes usability at gym while the web interface can be used in a big display at home to comfortably set up a program or prepare the next workout.

# Motivation

I have been working out in gym for several decades even if not too seriously. I have tried to find a suitable gym tracker app to track my workouts but all have been some way flawed for my purposes. Usually, the problems lie in simplicity and ease of use at real gym environment; when you are trying to be versatile and everything for everybody you end up being not a great solution for anyone. As a workaround, I have ended up using excel sheets which work but are far from ideal. Now that my son has joined me in gym workouts, I see an even greater need for a simple, effective workout tracker. This course is my great opportunity to create a solution that meets my personal needs and priorities.

# Key features

* Easy, real-time workout tracking using the native Android app
* Simple initialization of workouts based on previous workouts or saved programs
* Convenient program setup and workout preparation at home on a large screen
* Dual interfaces: native Android app and web browser UI
* Offline functionality, allowing use in gyms without internet connectivity

# Technology stack

* **Front-end**: React Native with Expo, which will also be used to create the browser UI.
* **Back-end**: JavaScript & Node.js. As time permits, I may explore AWS Lambda for backend functions.
* **APIs**: REST API communication between the front-end and back-end.
* **Database**: MongoDB. If a relational database proves more suitable as the project progresses, I’ll consider switching.

# Development tools

* **IDE**: VSCode with necessary plugins
* **Emulators**: Android Studio
* **Database**: Dockerized MongoDB for the development environment

# Functionality

Initial functionality is divided into epics and corresponding user stories. A HTML prototype was used to refine ideas and plan the app structure but it’s not supposed to be used as a basis of the implementation as such. The following is the list refined on 28.10.2024

**Epic 0. Initialization**

*Story 0.1.* Create user

*Story 0.2.* Login

*Story 0.3.* Initialize data from server

**Epic 1. Track workout**

*Story 1.1.* Add new workout

*Story 1.2.* Add a new exercise to workout

*Story 1.3.* Delete an exercise

*Story 1.4.* Add a new set to an exercise

*Story 1.5.* Delete a set

*Story 1.6.* Save the workout

**Epic 2. Browse past workouts**

*Story 2.1.* Show past workouts

*Story 2.2.* Navigate to next / previous workouts

*Story 2.3.* Clone past workout to create new

*Story 2.4.* Delete past workout

**Epic 3. Plan program**

*Story 3.1.* Add a week

*Story 3.2.* Add a workout day to a week

*Story 3.3.* Add an exercise to a day

*Story 3.4.* Save the program

**Epic 4. Browse past programs**

*Story 4.1* Show past programs

*Story 4.2.* Navigate to next / previous programs

*Story 4.3.* Clone a program to create new

*Story 4.4.* Delete past program

# Design principles

* **Focus on value**: All functional and technical design decisions prioritize actual value to the user.
* **Decoupling Node.js**: Node will be decoupled in a way that allows easily forking an app-only version in the future.
* **Design for usability**: The primary goal is real-world usability in a gym environment.
* **Design for simplicity**: Both functional and technical design focus on simplicity over features, extensibility and versatility.

These principles imply that:

* **No localization**: The interface will not include internalization and localization. Terms like "program," "workout," "exercise," "set," and "weight" are universally understood. English will be the UI language, aligning with the course language.
* **Selective test automation**: Test automation will only be implemented if it adds clear value within the foreseeable lifecycle. No test automation will be implemented for the sake of having test automation.
* **Only necessary features**: Features implemented will be immediately useful, with no additions primarily for future-proofing.
* **Data persistence as a single object**: All data for an individual user will be stored as a single object, even if this may not be theoretically optimal. The very limited data volume and expected minimal impact on responsiveness should allow making this decision safely.
* **Minimized modals and messages**. Confirmation modals are avoided if possible and added only when found out to be necessary in usability testing. Design is done in a way that makes the app informative and safe to use as such.
* **No support for low-resolution screens.** Smaller premium phones like Pixel 5 and Galaxy S23 set the minimum bar for supported screen sizes and resolutions in design and testing.