

**Fakultät**

**Ingenieurwissenschaften und Informatik**

**Hausarbeit im Fach**

**Webanwendungen**

**Entwicklung einer nativen Webapplikation mittels Ionic und Capacitor**

|  |  |
| --- | --- |
| **Autoren:** | Manuel Arling  965366  Thomas Meese  965321 |
| **Abgabedatum:** | 13.09.2022 |

Inhaltsverzeichnis

[1 Einleitung 5](#_Toc113933562)

[1.1 eingesetzte Technik 5](#_Toc113933563)

[1.2 Zielgruppe 6](#_Toc113933564)

[2 Was ist Train-Smarter 6](#_Toc113933565)

[3 Nutzerauthentifizierung und Datenhaltung 6](#_Toc113933566)

[3.1 Login 7](#_Toc113933567)

[3.2 Verwaltung der Personenbezogenen Daten 8](#_Toc113933568)

[4 ExerciseDB als Übungsdatenbank 9](#_Toc113933569)

[5 Technische Probleme und deren Lösungen 9](#_Toc113933570)

[5.1 Datenbankanfragen 9](#_Toc113933571)

[5.2 Page Animation 10](#_Toc113933572)

[5.3 Native App Login Lade Schleife 12](#_Toc113933573)

[6 Funktionalitäten der App 13](#_Toc113933574)

[6.1 Home-Ansicht 13](#_Toc113933575)

[6.2 Trainingspläne 14](#_Toc113933576)

[6.2.1 Trainingsplan-Kalender-Eintrag 15](#_Toc113933577)

[6.2.2 Detailansicht der Trainingspläne 16](#_Toc113933578)

[6.3 Laufrouten-tracker 16](#_Toc113933579)

[7 Abschlussbetrachtung 18](#_Toc113933580)

[8 Anhang 19](#_Toc113933581)

Abbildungsverzeichnis

[Abb. 1 Login und Registrierungsansicht 8](#_Toc113933634)

[Abb. 2 Ladeanimation 11](#_Toc113933635)

[Abb. 3 Login Lade Schleife 13](#_Toc113933636)

[Abb. 4 Ansicht: Home 14](#_Toc113933637)

[Abb. 5 Ansicht: Trainingspläne 16](file:////Users/manuel/Desktop/GitLab/webanwendungen-projekt/Projektbericht%20Webanwendungen%202.docx#_Toc113933638)

[Abb. 6 Kalender-Event 17](#_Toc113933639)

[Abb. 7 Ansicht Trainingsplan-Detail 18](#_Toc113933640)

[Abb. 8 Ansicht: Lauf-Routen 19](#_Toc113933641)

Snippet-Verzeichnis

[Snippet 1 Login Page: SignUp() 8](#_Toc113919403)

[Snippet 2 Interface: UserData 9](#_Toc113919404)

[Snippet 3ExerciseDBService 9](#_Toc113919405)

[Snippet 4 Nav-Animation 12](#_Toc113919406)

[Snippet 5 Einbinung der Animation in der Appmodule 12](#_Toc113919407)

[Snippet 6 Methode: provideFirebase() 13](#_Toc113919408)

# Einleitung

Webanwendungen stellen heutzutage eine wichtige Rolle dar, denn sie ermöglichen die Bereitstellung eines Programms, ohne dass dieses auf dem Rechner des Benutzers installiert werden muss. Dabei wird von einem Benutzer nur ein installierter Browser vorausgesetzt.

Zusätzlich lassen sich mit Cross-Side-Plattformen Webanwendungen zu nativen Applikationen verpacken. Der Vorteil besteht darin, dass für die Entwicklung einer Applikation diese nicht mehr gerätespezifisch erfolgt, sondern nur einmal und so auf alle Betriebssysteme gebracht werden kann. Somit lässt sich beispielweise mit einer Webanwendungen eine native App für iOS und Android erstellen.

Diese Facharbeit beschäftigt sich daher mit der Umsetzung von Train-Smarter (einer Webanwendungen), welche zusätzlich als native App auf mobilen Endgeräten gebracht wird.

## eingesetzte Technik

Den Grundbaustein für die Umsetzung dieser Anwendung stellt das Framework Ionic in Verbindung mit Angular dar. Zusätzlich verwendet Ionic Capacitor, womit sich native Apps erstellen lassen. Train-Smarter wird dabei eine native App für iOS und Android besitzen.

Durch die Verwendung von Capacitor lassen sich Handydatennutzen, wie den Geokoordinaten oder dem nativen Kalender.

Zusätzlich werden zwei APIs verwendet. Firebase stellt dabei das Halten der Nutzerdaten und der Authentifizierung der Nutzer dar. Wohingegen ExerciseDB eine Datenbank mit den rund 1300 verschiedenen Übungen darstellt, welche in die App eingebunden werden.

Während der Entwicklungsphase wird das Ionic Plugin „Ionic lab“ verwendet. Durch die Eingabe des gleichnamigen Befehls startet sich ein lokaler Webserver. Dieser ist standardmäßig unter den Ports 8200 und 8100 erreichbar. Beim Aufrufen des Ports 8100 erhält man die gewohnte Ansicht einer Internetseite. Die Besonderheit zeigt sich bei der Verwendung des Ports 8200. Hier erscheinen vorkonfigurierbare Smartphone-Ansichten, bei denen drei zur Auswahl stehen (iOS, Android, Windows), welche auch parallel angezeigt werden können. Der große Vorteil ist, dass das Aussehen der auf dem Smartphone von Anfang an in den Entwicklungsprozess einfließt. Jedoch ist hier zu beachten, dass Cordova Plugins in der Webansicht nicht angezeigt bzw. verwendet werden können, hierfür werden dann Emulatoren in Xcode oder Android-Studio benötigt. Welche sich ebenfalls nach dem Erstellen der jeweiligen Plattform App mit dem Befehl *ionic capacitor run [ios/android] --l -external* starten lassen.

## Zielgruppe

Die Zielgruppe von Train-Smarter richtet sich an sportinteressierte Personen jeglichem Alter. Dabei liegt der Fokus beim Krafttraining, jedoch lassen sich etliche Übungen auch für die Mobilität und der Schmerzbehandlung verwenden. Die App stellt eine breite Palette an unterschiedlichen Übungsformen zur Verfügung, wodurch es insgesamt mehr als 1300 Übungen. gibt

## Was ist Train-Smarter

Train-Smarter ist eine Anwendung, welche dem Nutzer ermöglicht sein Fitnesstraining effizienter und geleitet durchzuführen. Dabei stellt Train-Smarter dem Nutzer die Möglichkeit individuelle Trainingspläne zu erstellen, zu bearbeiten und die dazugehörigen Übungen hinzuzufügen. Dazu lassen sich die erstellten Trainingspläne „ausführen/starten“, wodurch der Nutzer die Möglichkeit hat für jede Übung die absolvierten Sätze mit den dazugehörigen Wiederholungen und dem Gewicht einzutragen. Somit verfügt der Nutzer über die Möglichkeit, seine absolvierten Trainingsverläufe zu sichern und diese zu analysieren. Außerdem kann der Nutzer seine Laufstrecke tracken und sich im Nachhinein anzeigen lassen.

Alle Funktionen im Überblick:

* SignIn, SigOut, SignUp
* Trainingspläne erstellen, ändern und löschen
* Übungen hinzufügen, löschen und bearbeiten
* Auflistung aller Übungen oder gefiltert nach Keywords
* Abarbeiten eines Trainingsplans
* Analysen der gesammelten Trainingsdaten
* Möglichkeit beim Joggen/Laufen die gelaufene Strecke zu tracken
* Getrackte Strecken anzeigen

# Nutzerauthentifizierung und Datenhaltung

Train-Smarter arbeitet zu einem großen Teil mit personenbezogenen Daten, um einen ständigen und sicheren Zugriff auf die Daten zu gewährleisten verwendet die App, die API Firebase für die Authentifizierung und als cloudbasierte Datenbank. Als Schnittstelle für den Zugriff werden zwei Services realisiert. Für die Services werden, die von Angular Fire 7 bereitgestellten Methoden verwendet. Bei den Services handelt es sich um den *DatabaseService* und den AutenticationService. Diese werden im Folgenden näher betrachtet.

## Login

Firebase bietet diverse Möglichkeiten, um eine Anmeldung an einen Dienst zu ermöglichen. In dieser App erfolgt die Anmeldung über die E-Mail-Authentifizierung. Jedoch kann die Anmeldung um weitere Verfahren erweiterter werden, so dass beispielsweise vorhandene Google, Facebook oder Github Konten verwendet werden können.

Graphical user interface, application, website

Description automatically generated

Abb. Login und Registrierungsansicht

Im *AuthentificationService* sind die Methoden *signIn(…), signUp(…)* und *signOut(…)* realisiert. Diese benutzen die Authentifizierungs-Methoden des Firebase-Authentification-Service. Der Service kommt beispielsweise in der Abb. 1 zu sehenden Login- bzw. Registeransicht zum Einsatz. Die im Formular eingetragenen Daten werden an die API übermittelt, deren Antwort wird dann ebenfalls auf der Ansicht visualisiert. Bei einer negativen Antwort erscheint ein Popup, dass die Anmeldung oder die Registrierung, fehlgeschlagen ist. Im positiven Fall wird der Nutzer zur Home-Seite weitergeleitet. Sollte es sich um einen Registrierungsvorgang handeln, wird bei einer erfolgreichen Registrierung ein neuer Benutzer in der *Userdata*-Collection von Train-Smarter angelegt.



Snippet Login Page: SignUp()

## Verwaltung der Personenbezogenen Daten

Für das Persistieren der Daten wird der NoSql-Datenbank-Dienst Could Firestore der Firebase API verwendet. Die Daten sind in vier Collections aufgeteilt, Nutzerdaten (*userData*), Trainingspläne (*trainingPlan*), aktives Training (*setLogging*) und den Laufrouten (walkData). Der Zugriff auf die Daten innerhalb der App ist durch den realisierten *DatabaseService*, der die Methode der API verwendet, gewährleistet. Für jede Collection sind dort Getter, Update und Add Methoden realisiert. Für alle Collections werden Interfaces verwendet werden, mit der die Struktur der Daten innerhalb der Datenbank eindeutig festgelegt ist.



Snippet Interface: UserData

Das Verwenden eines solchen, wie in Snippet zwei beispielhaft zu sehenden Schemas, bietet einige Vorteile. Neben dem strukturellen Vorteil können bei der Verwendung der Daten in der HTML-Seite oder in der .ts-Datei direkt auf die Variablenamen Zugriffen werden, wodurch Tippfehler vermieden werden.

# ExerciseDB als Übungsdatenbank

ExerciseDB stellt eine Datenbank API dar, welche mehr als 1300 Übungen zur Verfügung stellt. Dabei lassen sich Anfragen nachfolgenden Kategorien filtern: Übungsnummer, Körperteil, Muskelgruppe, Equipment und alle Übungen. Die API-Anfragen laufen über RapidAPI, welche eine zentrale Verwaltung von mehreren Programmierschnittstellen darstellt. 

Snippet ExerciseDBService

Snippet 3 zeigt den in der Anwendung umgesetzten Service. Eine Anfrage an die Datenbank erfolgt durch einen einfachen http-Request, mit den benötigten Key und der Host Adresse im Header. Die Antwort der API beinhaltet nach einer erfolgreichen Anfrage, eine Liste von allen Übungen die auf den Suchfilter passen, siehe *getData(type, target?)*. Dabei wird der *Filtertyp* und das *Target*, falls der Filtertyp dies fordert, übergeben. Enumerationen dienen dabei zur Unterstützung.

# Technische Probleme und deren Lösungen

Der Entwicklungsprozess von Train-Smarter beinhaltete mehrere technische Probleme, welche die Nutzung der App verhindert haben. Jedoch ließen sich diese im Verlauf des Entwicklungsprozesses lösen. Drei größere Probleme stellten dabei die Datenbankanfragen, die Page Animation und die Ladedauerschleife beim Login in der nativen App.

## Datenbankanfragen

Bei der Nutzung der Firestore-Datenbank hat sich folgendes Problem ergeben. Für den Inhalt der Seiten werden spezifische Daten aus der Datenbank benötigt (z.B. die Trainingspläne eines Nutzers). Diese werden via Anfragen geliefert. Da die Daten zeitversetzt ankommen, stehen sie der Seite nicht unmittelbar beim Laden zur Verfügung. Werden Attribute verwendet, die solche Daten verwenden, kann die Seite nicht mehr geladen werden. Der Fehler kann umgangen werden, in dem die Daten in der ngOnInit() Methode nachgeladen werden. Innerhalb des HTML-Quelltextes muss dann per \*ngIf geprüft werden, ob die Daten bereits vorhanden sind. Ist dies der Fall, kann der Quellcode generiert werden. Zusätzlich wird während der kurzen Wartezeit eine Ladeanimation angezeigt, was in folgender Abb. 2 zu sehen ist.

Graphical user interface, text, application, chat or text message

Description automatically generated

Abb. Ladeanimation

## Page Animation

Bei der Animation von einer Page zu einer anderen Page, kommt es zu einem kurzen Flackern. Bei näherer Betrachtung ist zu erkennen, dass bei der Animation die neue Page mit der Standardanimation erscheint. Nach dem die neue Page komplett zu sehen ist, wird für eine kurze Zeit (Millisekunde bis Sekunde) die vorherige Page angezeigt. Also als würde die neue Page kurz unsichtbar und wieder sichtbar werden. Das Problem ergab sich auch bei einem neuerstelltem Ionic Projekt, bei dem ebenfalls eine Verlinkung stattgefunden hat. Versuche wie das Nutzen einer extra Funktion welche die Methode *navigateByUrl()* aufruft, anstatt von *routerLinks* haben ebenfalls nicht geholfen.

Bei der Fehlersuche hat sich jedoch herausgestellt, dass das Problem durch das Ändern der *routerDirection* von *forward* zu *root* nicht auftritt. Jedoch ist dies keine mögliche Lösung für die App, da das Setzen der *routerDirection* auf *forward* für das Zurücknavigieren durch den *ion-back-button* verwendet wird. Zusätzlich tritt das Problem auch nicht auf, wenn man die Page Animationen komplett deaktiviert. Dadurch wird die Benutzerfreundlichkeit und das Design leider negativ beeinflusst und ist ebenfalls keine zufriedenstellende Lösung.

Die darauffolgende Idee eine eigene Page Animation zu erstellen, erwies sich als passende Lösung für die App. Die Datei src/app/animations/nav-animaton.ts beinhaltet die selbsterstellte Animation.



Snippet Nav-Animation

Diese realisiert eine Standardanimation und jeweils eine Animation für die eintretende und verlassende Page, sodass bei einer *forward*-Direktive, die Seite von rechts nach links erscheint und beim Verlassen genau andersherum. Dies ähnelt der Standardanimation, jedoch ist das Verhalten des Headers z.B. unterschiedlich.



Snippet Einbinung der Animation in der Appmodule

Danach muss die Animation nur noch für die App gesetzt werden, was innerhalb der *IonicModule.forRoot()* durch das Setzen der neuen Animation auf *navAnimation* erfolgt. Somit konnte das Problem mit der Animation behoben werden.

## Native App Login Lade Schleife

Graphical user interface, text, application, chat or text message

Description automatically generated

Abb. Login Lade Schleife

Bei der Entwicklung der App wurden zusätzlich Emulatoren von Xcode verwendet. Dabei stellt sich raus, dass die App beim Loginversuch in einer Lade Dauerschleife stecken bleibt, welches in Abb. 3 dargestellt ist. Dies geschieht auch bei dem Absenden einer Registrierung. Jedoch ist eine Verbindung zu Firebase gegeben, denn bei einer Registrierung werden die eingegebenen Daten in Firebase gespeichert. Anfangs wurde ein Fehler mit dem *LoadingController* vermutet, jedoch lad dort kein Fehler vor.



Snippet Methode: provideFirebase()

Es hat sich herausgestellt, dass wenn die App auf einem nativen Gerät ausgeführt wird, die Methode *initializeAuth()* von Firebase in der *app.module.ts* aufgerufen werden muss. Somit erfolgte der Login auf der nativen App ohne Probleme.

# Funktionalitäten der App

Das Folgende Kapitel behandelt die Funktionalitäten der Main-Pages, auf die der eingeloggte Nutzer Zugriff über das Burger Menü hat.

## Home-Ansicht

Graphical user interface, application

Description automatically generated

Abb. Ansicht: Home

Nach dem erfolgreichen Einloggen gelangt der Nutzer zur Home-Ansicht, diese begrüßt den Nutzer mit seinem hinterlegten Namen. Dazu lässt sich in Abb. 4 erkennen, dass sich darunter ein Slide aus Cards besteht. Diese stellen die vorhandenen Trainingspläne des Nutzers dar. Durch das swipen nach links oder rechts werden die Karten verschoben und ein anderer Trainingsplan gelangt in die Mitte. Durch das Klicken auf einer der Karten, gelangt man zu der Trainingsplan-Detailansicht des jeweiligen Trainingsplans.

Unterhalb der Trainingspläne befindet sich ein ähnlich aufgebauter Bereich, der die Trainingstermine der nächsten 14 Tage anzeigt. Diese werden ebenfalls durch das *Slides*-Element und darin enthaltenen *Cards* dargestellt. Dabei beinhaltet der *Card-Header* den Namen des Trainingplans. In dem *Card-Content* befindet sich einerseits den Tag des Termins, wie auch die Beschreibung des Plans. Das letzte Element der Home-Ansicht stellt der in Abb. 4 zu erkennender Button dar. Dieser dient dazu, um zum Live-Training weitergeleitet zu werden, wo automatisch das Training gestartet wird, welches als nächstes im Kalender stattfindet.

Wird die Home-Ansicht nicht in einer nativen App aufgerufen, so lässt sich die Auflistung der nächsten Trainingstermine nicht darstellen, da hierfür die Daten des nativen Kalenders verwendet werden. Dies ist im Browser jedoch nicht möglich. Deswegen wird der Nutzer durch das Betätigen des Buttons zur Live-Training Seite weitergeleitet, aber es erfolgt kein automatisches Starten eines Trainings-

## Trainingspläne

Abb. Ansicht: Trainingspläne

Graphical user interface, application

Description automatically generated

In der Trainingsplan-Seite gelangt der Nutzer zu einer Auflistung aller erstellten Trainingspläne (siehe Abb. 5 links). Die Pläne sind als Cards aufgelistet, welche aus dem Namen des Plans als Header und einer kurzen Beschreibung des Plans als Body besteht. Wenn eine Card nach rechts geswiped wird erscheint ein Button, mit dem der Trainingsplan gelöscht werden kann. Neue Trainingspläne können erstellt werden, indem der Plus-Button betätigt wird. Hierdurch öffnet sich ein Formular, in dem der Nutzer erste Angaben zum Trainingsplan tätigt (siehe Abb. 5 rechts). Bei den Angaben handelt es sich um den Namen des Plans, was er bewirken soll (eine Kurzbeschreibung) und wie häufig trainiert werden soll. Der Plan kann durch den Button „Trainingsplan speichern“ hinzugefügt werden. Solange die Formfelder nicht ausgefüllt sind der Button disabled, welches ebenfalls in Abb. 5 auf der rechten Seite zu erkennen ist. Der Erstellvorgang kann durch den „X“ Button beendet werden. Mit dem Speichern des Plans, wird dieser in der Datenbank hinzugefügt und das Formular schließt sich. Der neue Trainingsplan taucht anschließend in der List auf. Mit dem Antippen einer Card gelangt man in das Detail – Ansicht.

### Trainingsplan-Kalender-Eintrag

Graphical user interface, application

Description automatically generated

Abb. Kalender-Event

Die App verfügt über die Möglichkeit einen neuen erstellen Trainingsplan zu den lokalem Kalender hinzuzufügen. Wenn die Checkbox in dem vorherigem Form gesetzt ist, wird nach dem Erstellen des Trainingsplans ein neuer Kalender Eintrag erstellt, welcher dem Nutzer angezeigt wird. Dort sind alle wichtigen Daten voreingetragen, wie der Titel, Beschreibung, Kalender-Typ (Train-Smarter) und die Wiederholrate, siehe Abb. 6. Durch das Drücken auf „Add“ wird der Trainingsplan zum Kalender hinzugefügt und der Nutzer erhält 30 Minuten vor einem Training eine Benachrichtigung. Jedoch ist dabei zu beachten, dass diese Funktion nur für iOS-Geräte nutzbar ist, da dort als einzige Plattform ein eigener Kalender erstellt werden kann. Für Android wäre es möglich den Nutzer vorher zu fragen, zu welchen vorhandenen Kalender die Trainingspläne hinzugefügt werden sollen.

### Detailansicht der Trainingspläne

In der Detailansicht erhält der Nutzer alle Informationen zu dem Trainingsplan. Die neuen Informationen in dieser Ansicht sind die einzelnen Übungen, die der Plan beinhaltet. Diese sind als Cards dargestellt. Im Header der Card befindet sich der Name der Übung. Im Body befinden sich die Angaben zu den geplanten Sätzen, den Wiederholungen innerhalb der Sätze sowie das geplante Gewicht (z.B. vom Trainer empfohlen). Die einzelnen Übungen können durch das Swipe-Menü bearbeitet werden. Bei einem Swipe nach rechts kann die Übung entfernt werden. Während beim Swipe nach links die Übung bearbeitet werden kann. Dies ist in Abb. 6 zuerkennen. Bei der Betätigung öffnet sich ein Modal in der die einzelnen Zeilen der Liste ebenfalls durch einen Swipe bearbeitet werden können oder neue Sätze ergänzt werden können. Jeder der Bearbeitungsvorgänge kann abgebrochen werden, wodurch der Nutzer wieder zurück zur Detailansicht gelangt. Neue Übungen können über den Plus-Button hinzugefügt werden. Dies geschieht über die Übungsansicht. Auch dieser Vorgang kann abgebrochen werden. Durch den Back-Button gelangt der Nutzer wieder zurück zur Trainingsplan Übersicht.

Graphical user interface, application

Description automatically generated

Abb. Ansicht Trainingsplan-Detail

## Laufrouten-tracker

Train-Smarter bietet dem Nutzer die Möglichkeit Laufrouten Tracking zu betreiben. Sobald dieser mit dem Laufen oder Joggen beginnt, kann er das Tracking in der App starten. Für das Tracking wird das Cordova-Plugin Geolocation verwendet. Die Daten werden durch die dort enthaltene Methode *watchPostion()* gesammelt. Damit dies funktioniert, muss der Nutzer die Nutzung der Standorterfassung freigeben. Für die anschließende Visualisierung der so erstandenen Route wird Google Maps verwendet, wofür ein Api-Key eingebunden ist. In der Ansicht „Deine bisherigen Routen“ können die bereits aufgezeichneten Routen eingesehen werden. Die Auflistung erfolgt als Liste von Cards. Im Header der Card ist der Name des Laufs zu finden und im Body die benötigte Zeit für die Strecke. Lauf-Routen können durch ein Swipe-Menü bei den Cards gelöscht werden. Durch Anklicken der Card gelangt man in die Detailansicht. Dort ist die visualisierte Route einsehbar. Die Route wird hierfür als Polylinie in eine Google Maps Karte gezeichnet. Verlassen werden kann diese Ansicht durch den Backbutton. In der Hauptansicht können neue Routen durch den Plus Button hinzugefügt werden. Durch das Betätigen des Buttons gelangt man in die Erstellansicht. Das Tracking wird durch den Button „Lauf Starten“ aktiviert. Auf der dort enthaltenden Karte erscheint eine dynamisch wachsende Polylinie, die dem Laufweg entspricht (Hierbei können Ungenauigkeiten durch die Standorterfassung entstehen). Durch das Betätigen des „Lauf Beenden“-Button wird das Tracking gestoppt. Der Nutzer erhält die Möglichkeit die Lauf-Route zu benennen und diese unter dem Namen zu speichern. Falls kein Name angegeben wird, wird Datum und Uhrzeit des Laufs als Name verwendet. Alternativ kann die Lauf-Route auch verworfen werden. In beiden Fällen gelangt der Nutzer zur Hauptansicht der Lauf-Routen.

Da für die Nutzung der Google Maps API ein Google Konto mit hinterlegten Abrechnungsinformationen benötigt wird, steht die Darstellung des Laufens auf den nativen Apps nur eingeschränkt zur Verfügung. Im Web-Browser lassen sich jedoch die Strecken ohne Probleme anzeigen.

Graphical user interface, application, chat or text message

Description automatically generated

Abb. Ansicht: Lauf-Routen

# Abschlussbetrachtung

Auch nach Abschluss des Projekts bietet die App Möglichkeiten zur Umsetzung weiterer Features. So könnten weitere Funktionalitäten die Anzeige einer detaillierteren Analyse der gesammelten Trainingsdaten oder das Einbinden des Pulses durch eine Smartwatch sein. Bei der Analyse wären Graphen denkbar, beispielsweise wie sich das verwendete Gewicht bei einer Übung über die Trainingstage verändert hat. Außerdem wäre es sinnvoll Features zu verbessern. Bei der Lauf-Routen Aufzeichnung treten aktuell, bedingt durch die Verwendung von Geolocation, teils starke Ungenauigkeiten bei den Längen- und Breitengraden auf. Diese werden nicht akkurat genug erfasst. Hierfür könnte versucht werden die Genauigkeit der Daten während des Trackings durch Geolocation zu verbessern. Jedoch setzt Train-Smarter alle gewünschten Funktionen um und bietet dem Nutzer eine ideale Plattform, um das Training zu optimieren. Durch die Verwendung von Firebase lassen sich die Nutzerdaten cloudbasiert speichern, wodurch der Nutzer die Möglichkeit hat auf mehreren Geräten sein Konto zu verwenden. Dazu stellt die Einbindung der ExerciseDB die ideale Darstellung der einzelnen Übungen, da diese ihren einzelnen Kategorien zugeordnet werden. Die enthaltenen GIFs der ExerciseDB bieten einem Nutzer zugleich die Möglichkeit eine neue Übung einfach und schnell zu verstehen, damit dieser sie richtig und schonend ausführen kann.

# Anhang

Struktur der Abgabe

|  |  |
| --- | --- |
| Verzeichnis | Inhalt |
| /Ausarbeitung | Pdf Dokument der Ausarbeitung |
| /Projekt | Webstorm-Projekt |

Eigenständigkeitserklärung

Hiermit erklären wir an Eides statt, dass wir die vorliegende Arbeit selbst-  
ständig und ohne fremde Hilfe angefertigt haben. Die aus fremden Quellen direkt  
oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde  
bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Melle, 12.09.2022 ……………………. …………………



Unterschrift