

Hochschule Flensburg

BACHELOR-THESIS

Thema: Erstellung von einem Trainingsplan basierend auf Zielen, Grundfähigkeiten und Trainingsverlauf des Nutzers

von: Piet Carstensen

Matrikel-Nr.: 630310

Studiengang: Angewandte Informatik

Betreuer/in und
Erstbewerter/in: Dipl.-Math. Jochen Stamp

Zweitbewerter/in: B. Sc. Oliver Prekszas

Ausgabedatum: 07.12.2020

Abgabedatum: 07.02.2021

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Thesis ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen benutzt habe.

Inhaltsverzeichnis

1 Einleitung	4
1.1 Motivation	4
1.2 Zielsetzung	5
1.3 Aufgabenstellung	5
2 Grundlagen	6
2.1 Back4App	6
2.2 Parse-Server	6
2.3 Datenbank	6
2.4 Datenbankstruktur	7
2.5 Cloud-Funktionen	10
2.6 NodeJS	10
2.7 Unterschied zwischen Technik-/ Sprint und Ausdauer-Übungen	11
3 Konzeption	12
3.1 Problemanalyse	12
3.1.1 Nutzerziele	12
3.1.2 Grundfähigkeiten des Nutzers	12
3.1.3 Trainingsverlauf des Nutzers	13
3.1.4 Dauer eines Trainings	13
3.1.5 Integration in das bestehende System	14
3.2 Bedingungen	15
3.3 Methodik	16
3.3.1 Ziele	16
3.3.2 Grundfähigkeiten	17
3.3.3 Trainingsverlauf	19
3.3.4 Dauer eines Trainings	20
3.4 Trainingsplan-Algorithmus Theorie	20
3.4.1 Parameter	20
3.4.2 Nutzerdaten	20
3.4.3 Trainingsdauer und Distanz	21
3.4.4 Bereichseinteilung	21
3.4.5 Boost (prozentuale Erhöhung der Bereiche)	21
3.4.6 Trainingsverlauf	22
3.4.7 Sortieren der Übungen	22

3.4.8	Hinzufügen der Übungen	23
3.4.9	Speichern des Trainings	27
3.5	Aufbau der Klassen und Methoden im Code	27
4	Implementierung und Ergebnis	29
4.1	Abfrage der Nutzerinformationen	29
4.2	Berechnung der Trainingsdaten	31
4.3	Abfrage der Post-Übungen	36
4.4	Übungen hinzufügen	38
4.4.1	Ausdauer- und Sprintbereich	39
4.4.2	Technikbereich	46
5	Zusammenfassung und Ausblick	48
5.1	Zusammenfassung	48
5.2	Ausblick	49
5.2.1	Erweiterung der Ergebnisse dieser Arbeit	49
5.2.2	Neuer Ansatz - Machine Learning	49
6	Abkürzungsverzeichnis	50
7	Glossar	51
	Abbildungsverzeichnis	52
	Quellcodeverzeichnis	53
	Literatur	55
8	Anhang	56

1 Einleitung

Fast 11 % aller Deutschen gehen regelmäßig im Freibad, in der Schwimmhalle oder im See schwimmen. Damit belegt das Schwimmen den dritten Platz der beliebtesten Sportarten [12]. Schwimmen korrigiert Fehlhaltungen, baut Stress ab, trainiert Kraft und Ausdauer - und das bei einem sehr geringen Verletzungsrisiko. Beim Schwimmen wirkt der Körper schwerelos, das schont Knorpel und Knochen, während fast jeder Muskel trainiert wird [11].

Schwimmen und dabei seine Technik effizient verbessern?

Das macht die App H2Coach der Fjordev UG möglich [9]. Dabei soll dem Nutzer die Problematik der selbstständigen Dokumentation von seinem Training abgenommen werden. Bisher war dafür entweder viel Vorbereitungsaufwand oder die Abhängigkeit von einem Trainer oder einer Trainingsgruppe bei gleichzeitig hohen Kosten und geringer Flexibilität notwendig.

1.1 Motivation

Wenn sich ein Nutzer dazu entscheidet, etwas tiefer in die Sporttheorie vorzudringen, ohne sich selber alles Wissen aneignen zu müssen, kommt er normalerweise nicht an einem Trainer vorbei. Ein eigenen Trainer bedeutet zuerst einmal Kosten für den Nutzer. Für wöchentliche Trainingseinheiten oder die Zusammenstellung individueller Trainingspläne wird der Trainer entlohnt. Der Nutzer muss dann weniger Anstrengung in die Planung seines Trainings investieren. Es entsteht für den Nutzer eine zeitliche und örtliche Bindung. Der Trainer kann zum Beispiel nur zu ausgewählten Zeiten in dem Schwimmbad Vorort sein. Zusätzlich wird auch das Tempo des Trainings vom Trainer stark bestimmt. Durch Kommunikationsfehler kann es dort zu Missverständnissen kommen, und der Nutzer bekommt beispielsweise ein viel zu leichtes oder schwieriges Trainingsprogramm.

Falls der Nutzer sich entscheidet, keinen eigenen Trainer zu engagieren, entfallen die Kosten, doch es können andere Probleme entstehen. Ohne fremde Hilfe muss der Nutzer viel Zeit aufbringen einen Trainingsplan selbst zu erstellen. Öffentliche- und/oder kommerzielle Seiten zum Erstellen von Schwimmtrainingsplänen, die dem Nutzer dabei helfen könnten, sind nicht

individuell genug und benötigen für jedes Training erneut Datensätze (persönliche Daten, Zeiten,...) [8, 7].

Um dieses Problem zu lösen, soll in dieser Arbeit ein Programmteil (Feature) der App H2Coach [9] für die automatische Erstellung von individuellen Trainingsplänen für Nutzer integriert werden. Der zu programmierende Algorithmus soll den Nutzer und seine Leistungen einschätzen und ihm dementsprechend individuell auf ihn abgestimmte Übungen zuweisen. Das Erstellen der Trainingspläne soll im Hintergrund ablaufen und der Nutzer soll nur bei Bedarf das fertige Training auf seinem Endgerät angezeigt bekommen und er muss sich nicht mit den Sport-theoretischen Ansätzen auseinandersetzen. Das Feature übernimmt die Arbeit eines eigenen Trainers.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Konzeption und die Realisierung einer ersten Version eines Features zur Erstellung eines Trainingsplans auf der Basis der Grundfähigkeiten, der Ziele und des Trainingsverlaufes eines Nutzers. Im Anschluss dieser Arbeit und bis Mitte 2021 sollte das Feature für eine Veröffentlichung in der App [9] fertiggestellt werden.

1.3 Aufgabenstellung

Im Rahmen dieser Arbeit sollen Funktionen und Methoden konzipiert und implementiert werden, die zusammen integrale Bestandteile des Features sind. Der Programmteil soll zur Erstellung von Trainingsplänen dienen und damit die Vorlage für die App H2Coach [9] sein. Die Sport-theoretischen Hintergründe wie beispielsweise die Dauer einer Übung werden vorausgesetzt und in dieser Arbeit durch angenommene Zahlenwerte ersetzt. Es geht vielmehr nicht um die Sporttheorie sondern um die Konzeption und Implementierung mit Sport-theoretischem Ansatz zur Erstellung von individualisierten Trainingsplänen anhand eines Trainingsalgorithmus in der Sportart Schwimmen.

2 Grundlagen

Um jeden Aspekt dieser Arbeit, jeden Quellcodeausschnitt und jede Idee der Konzeption genauestens nachvollziehen zu können, sind einige Grundlagen zu dieser Arbeit in folgendem Abschnitt erklärt.

2.1 Back4App

Die App H2Coach die dem Nutzer eine grafische Übersicht über seine Trainingpläne und einzelne Übungen liefert, benutzt Back4App. Back4App [2] ist ein Open-Source Backend-Host-Anbieter, der Entwicklern dabei hilft, erweiterbare und skalierbare Web- und Mobil-Anwendungen zu erstellen. Dafür stellt Back4App Parse-Server-basierte Anwendungen, die automatisch skaliert werden, zur Verfügung. Die Schlüsselfunktionen dabei sind, eine auf MongoDB basierte NoSQL-Datenbank und die Bereitstellung einer übersichtlichen UI für Rest-Endpoints und SDKs [4, 21].

2.2 Parse-Server

Ein Parse-Server [3] ist ein Backend-as-a-Service (BaaS) Framework [5], das initial von Facebook entwickelt wurde. Es stellt den Web- und Mobil-Entwicklern eine umfassende Technik zur Verfügung, um die Anwendung mit dem Backend-Cloud-Speicher und den APIs zu verbinden und Funktionen wie Authentifizierung, Push-Benachrichtigungen, Integration von sozialen Netzwerken und Datenanalysen zu bieten. BaaS überbrückt die Lücke zwischen dem Frontend der App und verschiedenen Cloud-basierten Backends [14] über eine einzige API und eine SDK.

2.3 Datenbank

Die SDK des Parse-Servers nennt sich *Parse SDK for Javascript*. Sie bietet viele Methoden, um beispielsweise auf die Datenbank zuzugreifen, aber auch vorgefertigte Funktionen wie *Login* oder *Register*. Bei einer Abfrage (Query) einer Klasse in der Datenbank, wird die Klasse als *ParseObject* zurückgegeben. Ein *ParseObject* hat neben einem Klassennamen (className) auch Attribute (attributes). Alle möglichen Abfragen (Queries) der Datenbank und die Funktionen und Attribute eines ParseObjects können

unter der Dokumentation (<https://parseplatform.org/Parse-SDK-JS/api/master/index.html>) gefunden und eingesehen werden.

2.4 Datenbankstruktur

Für den Trainingsplan-Algorithmus ist es wichtig, die Datenbankstruktur und die Beziehungen zwischen den bereits bestehenden Klassen zu kennen.

Eine Übung (ExerciseStep) besitzt das Attribut Bewegungsbereich (MovementArea). Ein Beispiel für MovementArea ist das Ein -oder Aus-schwimmen als Bewegungsbereich. Die Klasse MovementArea hat außerdem im Attribut Übungsart (ExerciseStepType) die Information über die Art der Übung. Beispiele für Arten sind Technik, Ausdauer oder Sprint. Des weiteren besitzt jeder ExerciseStep ein Attribut Übungsgruppe (ExerciseGroup), das mehrere Übungen zusammenfasst und näher beschreibt. Jede ExerciseGroup besitzt zuletzt ein Attribut Schwimmstil (SwimStyle), den Schwimmstil der Übungsgruppe.

Damit aus den Übungen dann ein Training (Workout) erstellt werden kann, hat ein Workout neben einer Gesamtdistanz (Distanz), mehrere Trainingsblöcke (WorkoutBlocks). WorkoutBlocks fassen inhaltlich mehrere Trainingsübungen (WorkoutSteps) zusammen. WorkoutSteps haben einen Pointer auf ExerciseStep und können mit den Attributen Distanz in Metern für eine Übung (distance) und Wiederholungen (repetitions) angeben, auf welche Distanz und mit wie vielen Wiederholungen ein ExerciseStep geschwommen werden soll.

Jeder Nutzer (User) kann somit beliebig viele Workouts besitzen. Außerdem hat jeder Nutzer eine Klasse Benutzerinformationen (UserData). In dieser sind auch Nutzer-spezifische Daten gespeichert. Neben Bahnlänge und dem Profilfoto des Nutzers, ist beispielsweise dort auch das nächste Workout gespeichert. Abbildung 1 zeigt einen Ausschnitt der beschriebenen Klassen und Beziehungen in der Datenbank.

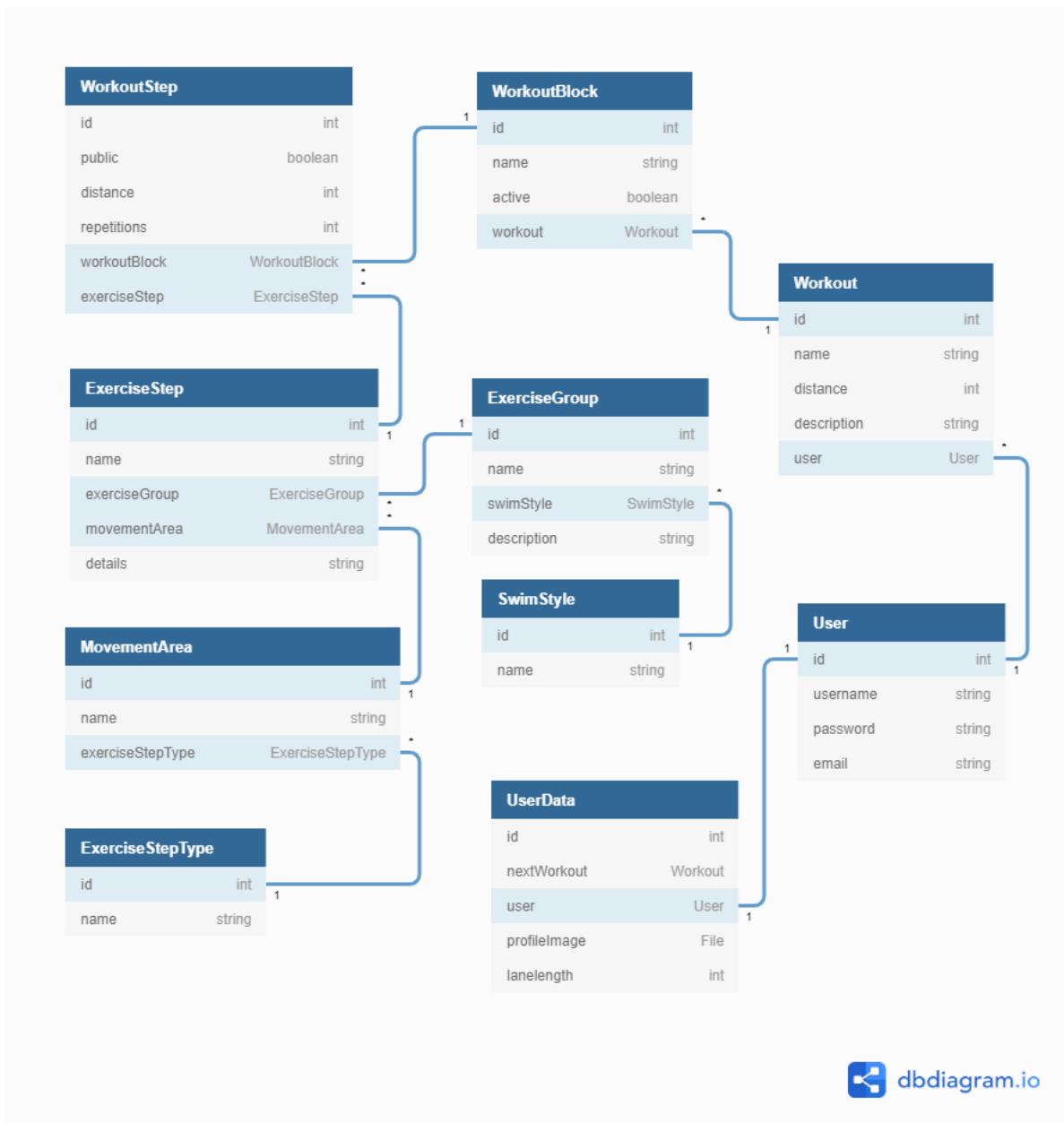


Abbildung 1: Datenbankstruktur

In Abbildungen 2 und 3 sind die Klassen aus der Datenbank in der H2Coach App grafisch dargestellt. Abbildung 2 zeigt beispielhaft das Training (Workout) *Eine Meile*. Dieses besteht aus Attributen wie Name, Beschreibung und Distanz, die im Datenbankmodell (vgl. Abbildung 1) als

Klassen dargestellt sind. In der Abbildung 3 ist der Teil *Sprinteinheit* des Workouts für *Eine Meile* dargestellt.

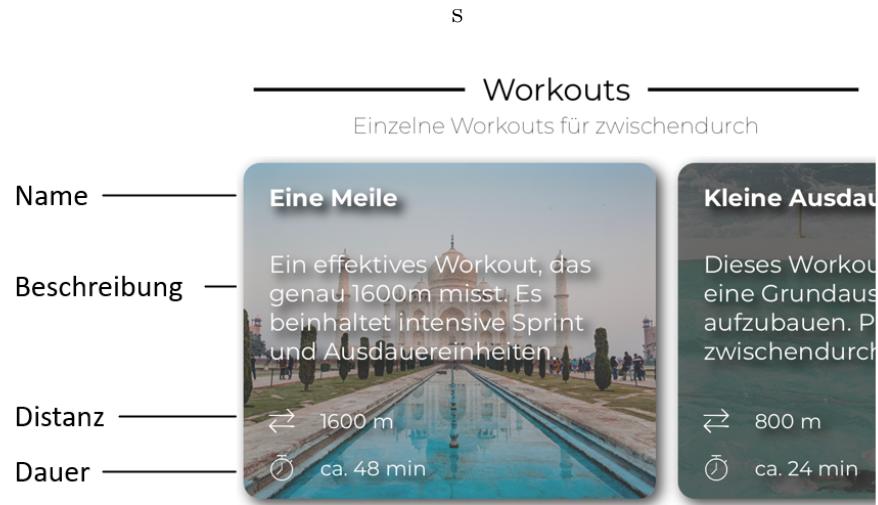


Abbildung 2: Workout Struktur

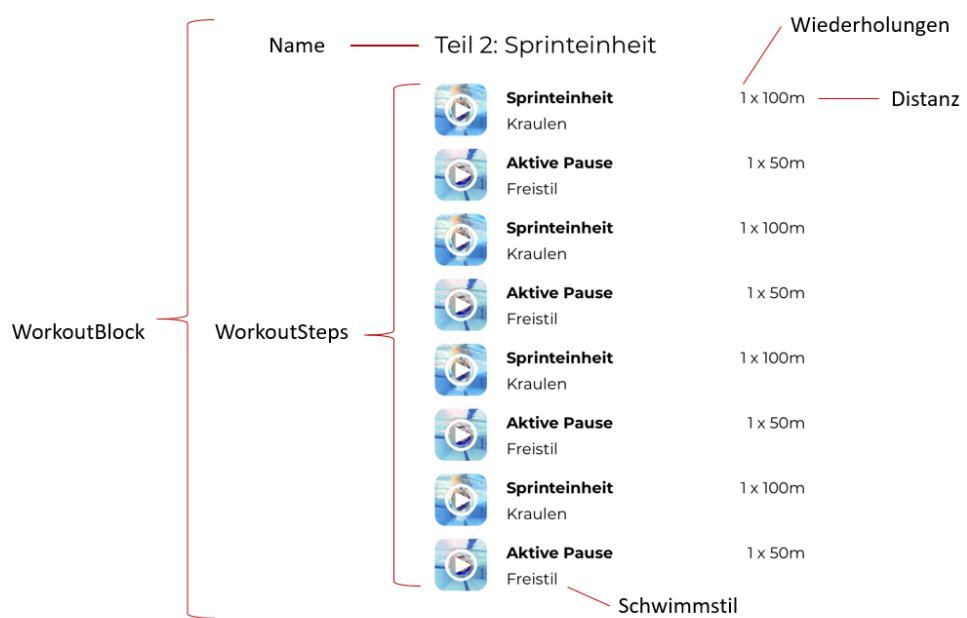


Abbildung 3: Struktur eines WorkoutBlocks am Beispiel Sprinteinheit

2.5 Cloud-Funktionen

Die Cloud-Funktion ist ein Tool des Parse-Servers, mit dem eine NodeJS-Funktion in der Back4App-Cloud ausführt werden kann [13]. Back4App führt die Funktion nur aus, wenn die Funktion über die API oder über die SDK aufgerufen wird. Mit Cloud-Funktionen ist es auch möglich, Funktionen zu erstellen, die durch App-Events ausgelöst werden.

Ein Beispiel für eine klassische Cloud-Funktion ist, das Einloggen eines Nutzers in der App. Wenn der Nutzer den Login-Button einer App betätigt und Nutzernname und Passwort angegeben hat, würde die Cloud-Funktion, in der Datenbank abfragen, ob Nutzername und Passwort übereinstimmen und einen Fehler melden, falls die Kombination falsch ist. Die Cloud-Funktion wird aus einem Event (Login-Button wird gedrückt) gestartet.

Der Aufbau einer Cloud-Funktion wird in dem Codeausschnitt 1 gezeigt [1].

```
1 Parse.Cloud.define("test", (request) => {
2     // returns jsonObject with the property text = "Hello World"
3     var text = "hello world";
4     var jsonObject = {
5         "answer": text
6     };
7     return jsonObject
8 });
```

Codeausschnitt 1: Beispielcode Cloud-Funktion

2.6 NodeJS

NodeJS ist eine Open-Source Server-Umgebung. NodeJS läuft auf einer Vielzahl von Betriebssystemen wie beispielsweise Windows, Linux, Unix und Mac OS X. Außerdem verwendet NodeJS serverseitig die aus Web-Anwendungen bekannte Programmiersprache Javascript. Javascript ist eine Skriptsprache, die anders wie z.B. Java, objektorientiert strukturiert ist [6].

2.7 Unterschied zwischen Technik-/ Sprint und Ausdauer-Übungen

In der Implementierung muss für die Entscheidung welche Übung in ein Training hinzugefügt wird, zwischen Sprint-/ Ausdauer- und Technikübungen unterschieden werden. Technikübungen können aufeinander aufbauen. Das bedeutet, für viele Technikübungen existiert eine Folgeübung. Aus diesem Grund wird bei der Auswahl der Technikübungen die Liste der Folgeübungen mit betrachtet. Sprint-/ Ausdauerübungen werden zufällig aus einem Array an möglichen Übungen gewählt, sofern der Trainingsalgorithmus diese als erforderlich einschätzt.

3 Konzeption

3.1 Problemanalyse

Bei der Erstellung eines Trainingsplan für einen Nutzer soll speziell auf folgende Faktoren eingegangen werden:

- Ziele - Warum möchte der Nutzer trainieren bzw. was möchte dieser verbessern?
- Grundfähigkeiten - Wie objektiv beherrscht der Nutzer die verschiedenen Schwimmstile?
- Trainingsverlauf - Welche Trainingseinheiten hat der Nutzer in der Vergangenheit ausgeübt?

Die Frage, die sich in erster Linie in diesem Zusammenhang stellt ist, wie genau können diese Faktoren in Programmcode umgesetzt werden?

3.1.1 Nutzerziele

Eine grundlegende Voraussetzung ist es, dem Nutzer die Möglichkeit zu bieten, seine Ziele einzutragen und zu priorisieren. Er hat die Auswahl zwischen Verbesserung des Sprints, Verbesserung der Ausdauer und Verbesserung der Technik. Wählt der Nutzer beispielsweise Sprint (Verbesserung von Sprint), soll sich dies in seinem erzeugten Trainingsplan widerspiegeln, indem er überwiegend Sprintübungen im Training wiederfindet. Dies meint allerdings nicht, dass die anderen Ziele keine Rolle im Training spielen oder vernachlässigbar sind. Auch beispielsweise Ausdauer-/ Technikübungen sollen dann mit einem geringeren prozentualen Anteil im Training zu finden sein.

3.1.2 Grundfähigkeiten des Nutzers

Die Grundfähigkeiten des Nutzers sind ebenfalls wichtig für die späteren Entscheidungen, welche Übungen dem Nutzer zugetraut werden können. Die Eignung der Übungen soll von der Erfahrung des Nutzer abhängig sein. Wenn ein Nutzer beispielsweise noch nie Kraul geschwommen ist, könnten im ersten Training einfache Kraul-Technikübungen hauptsächlich vertreten sein. Für die Ermittlung der Grundfähigkeiten in einem Stil sind zwei Varianten möglich. Einerseits könnten diese mit einer Eigeneinschätzung des

Nutzers im Schwimmstil ermittelt werden. Dies sollte dann in Form einer Abfrage des Nutzers in der App geschehen.

Andererseits könnten die Daten auch durch ein Benchmark-System gewonnen werden. Das bedeutet, nachdem der Nutzer sich dafür entschieden hat, den Trainingsplan-Algorithmus zu nutzen, muss er ein Benchmark-Training schwimmen. Mit den Übungen in diesem Training und den aufgezeichneten Zeiten soll dann eine ungefähre Einschätzung der Leistung in den verschiedenen Schwimmstilen getroffen werden.

3.1.3 Trainingsverlauf des Nutzers

Das bei der Erstellung eines neuen Trainingsplans ältere Trainingseinheiten untersucht werden, stellt eine weitere wichtige Kernfunktion dar. Untersucht wird, welche Übungen bei dem Nutzer gut und welche schlecht funktionierten, um möglicherweise Alternativübungen bei schlecht bewerteten Übungen bereitzustellen oder auf der gut bewerteten Übung aufbauende Folgeübungen zu finden und in das nächste Training einzubauen.

Übungen sollen außerdem durch einen zeitlichen Stempel sortiert werden können, sodass dem Nutzer auch neue Übungen vorgeschlagen werden und er nicht immer die gleichen Übungen absolvieren muss.

3.1.4 Dauer eines Trainings

Neben den Übungen besteht ein Training auch aus einer Gesamtdauer. Das ist die Summe der Dauer jeder Übung in einem Training. Der Nutzer kann die Dauer des Trainings zugreifen (vgl. Abbildung 2), damit er sein Training zeitlich vorausplanen kann. Vielmehr soll der Nutzer auch angeben können, wie lange er minimal und maximal trainieren möchte. Anhand dieser Zeiten (in Sekunden), wird der Algorithmus die Gesamtdauer des Trainings bestimmen.

Ein Problem in diesem Zusammenhang ist, dass die Dauer einer Übung eine wesentliche Rolle dabei spielt, wie viele Übungen in ein Training passen. Das bedeutet, man müsste von jeder Übung die ungefähre Dauer wissen, um entscheiden zu können, ob eine Übung in ein Training passt oder diese Übung den zeitlichen Rahmen (Gesamtdauer) überschreitet.

3.1.5 Integration in das bestehende System

Der Trainingsplan-Algorithmus ist als Feature für das bestehende System der H2Coach Schwimm-App geplant. Deswegen ist die Server-seitige Umgebung bereits fest vorgegeben (vgl. Abschnitt 2.1).

Die Abbildung 4 zeigt einen Ausschnitt der bestehenden Server-seitigen Struktur auf dem Parse-Server des Projektes. Die Dateien mit der Endung *.js* sind, Javascript Dateien von speziellen Cloud-Funktionen. Damit der Algorithmus bestehende Cloud-Funktionen (z.B. *saveWorkout*) nutzen kann, werden alle Funktionen, die für den Trainingsplan-Algorithmus wichtig sind, ebenfalls auf dem gleichen Server angelegt. Dies soll Code-Redundanz vorbeugen.

Name	Size
..	
helperFunctions	
internal	
jobs	
views	
DebugLogger.js	292 B
app.js	2.43 KB
cancelSubscription.js	2.62 KB
deactivateWorkout.js	598 B
editWorkout.js	8.26 KB
editWorkoutDescription.js	790 B
editWorkoutName.js	741 B
getAllContentData.js	1.58 KB
getAllUserData.js	1.49 KB
getBraintreeToken.js	1.04 KB
getCountries.js	129 B
getExecutions.js	305 B

Abbildung 4: Ausschnitt aus dem Aufbau des Dateiensystems auf dem Parse-Server

3.2 Bedingungen

Aus der Problemanalyse ergeben sich folgende Bedingungen, die außerhalb dieser Arbeit erfüllt werden müssen.

- Grundfähigkeiten - Ein Nutzer muss vor der Ausführung des Algorithmus seine Grundfähigkeit im Schwimmstil in der App festgelegt haben.
- Ziele - Ein Nutzer muss vor der Ausführung des Algorithmus sein priorisiertes Ziel in der App festgelegt haben.
- Intensität einer Übung - Für jede Übung muss in der Datenbank eine Intensität festgelegt worden sein - Wie anstrengend sind 100m dieser Übung?
- Dauer einer Übung - Für jede Übung muss in der Datenbank eine Dauer festgelegt worden sein - Wie lange dauern 100m dieser Übung?
- Dauer eines Trainings - Ein Nutzer muss vor der Ausführung des Algorithmus seine präferierte Trainingslänge (Minimum und Maximum) in der App festgelegt haben.
- Technikübungen - Jede Technikübung muss ein Array von Alternativübungen und Folgeübungen besitzen.

3.3 Methodik

Da der Sport-theoretische Teil nicht der zentrale Teil der Arbeit ist, steht die Umsetzung des Trainingsplan-Algorithmus im Vordergrund. Aus diesem Grund wurden die Sport-theoretischen Probleme zunächst ohne Experten auf dem Gebiet des Schwimmtrainings gelöst. Das heißt, zum Beispiel einige Zahlenwerte sind realitätsnahe Annahmen, die sich aus Sicht der Programmierung eignen.

3.3.1 Ziele

Abbildung 5 zeigt die Klasse ExerciseStep in der Datenbank. Die Klasse beschreibt für diese Arbeit die Ziele, zwischen denen ein Nutzer in der App wählen kann. In der Klasse UserData (vgl. Abbildung 1) wird daher ein weiteres Attribut *aim* vom Typ Nutzerziele (ExerciseStepLevel) angelegt, das dieses Ziel einem Nutzer zuweist.

CLASS | 5 OBJECTS

ExerciseStepType API Reference Video Tutorial

	objectId String	position Number	ACL ACL	name String
<input type="checkbox"/>	P5eLKFcIKV	5	Public Read	Tauchen
<input type="checkbox"/>	oQ4PDCAwX0	1	Public Read	Technik
<input type="checkbox"/>	8uvl9Arapv	2	Public Read	Ausdauer
<input type="checkbox"/>	jCiN5848jd	3	Public Read	Sprint
<input type="checkbox"/>	GAkiIHAKxu	4	Public Read	Lockeres Schwimmen

Abbildung 5: Ausschnitt aus der Klasse ExerciseStep

3.3.2 Grundfähigkeiten

Die Grundfähigkeiten des Nutzers sollen dazu beitragen, dass jedes Training den Nutzer nicht überdurchschnittlich unter- oder überfordert. Dafür wird aus jeder Grundfähigkeit in einem Schwimmstil eine Instanz der Klasse *Technique* erzeugt. Die Klasse *Technique* in der Datenbank besitzt als Attribut einen Pointer auf den Nutzer, einen Pointer auf den Schwimmstil und einen Wert zwischen 0 und 4, der beschreibt, welche Erfahrung im Schwimmstil vorliegt. Dabei steht 0 für wenig Erfahrung und 4 für mehr Erfahrungen. Abbildung 6 zeigt die Klasse *Technique*.

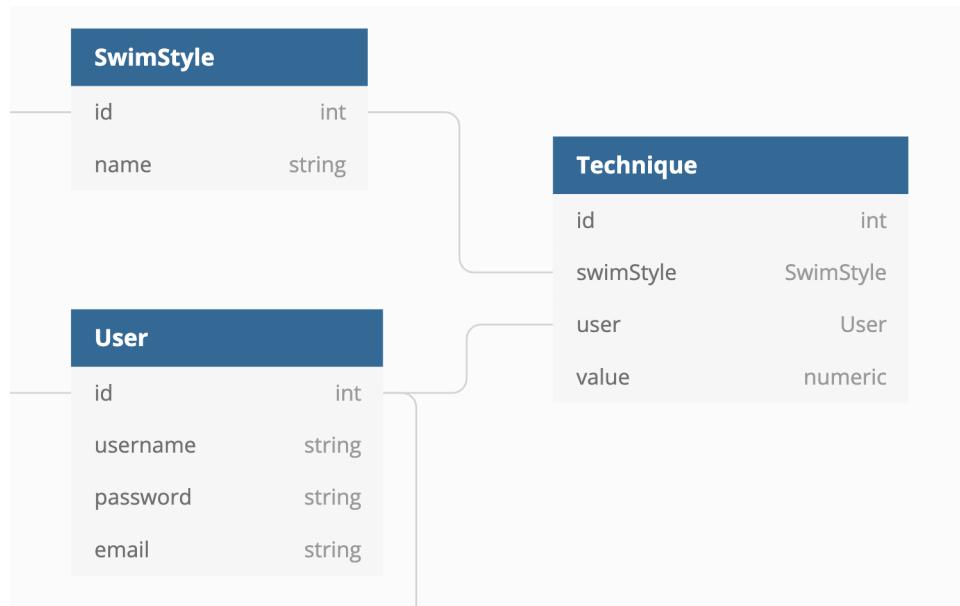


Abbildung 6: Datenbankklassen Technique, SwimStyle und User in Beziehung zueinander

Aus den Grundfähigkeiten wird für jeden Stil beim Ausführen des Trainingsplan-Algorithmus ein Intensitätsmaximum erzeugt. Dieses Intensitätsmaximum dient als Abbruchbedingung beim Hinzufügen von Übungen. Ist das Intensitätsmaximum überschritten, ist der Nutzer ab jeder zusätzlichen Übung, die hinzugefügt wird, überfordert. Das Intensitätsmaximum ergibt sich aus der Grundfähigkeit multipliziert mit einem Faktor.

$$intensity = \{0, 1, 2, 3, 4\} * 4 \text{ (Faktor)}$$

Damit überhaupt das Intensitätsmaximum gefüllt werden kann, muss jede Übung über eine Intensität verfügen (vgl. Abschnitt 3.2). Ein weitere Abbruchbedingung ist die Zeit des Trainings. Wenn diese überschritten wird, wird auch das Hinzufügen der Übungen angehalten. Damit auch diese Funktion implementiert werden kann, muss die Klasse ExerciseStep auch über ein Attribut *duration* verfügen. Die Abbildung 7 zeigt die Klasse ExerciseStep mit dem Attribut Intensität (*intensity*) und Dauer (*duration*).

ExerciseStep	
id	int
name	string
exerciseGroup	ExerciseGroup
movementArea	MovementArea
details	string
intensity	numeric
duration	numeric

Abbildung 7: Datenbankklasse ExerciseStep mit Intensität und Dauer

3.3.3 Trainingsverlauf

In der App gibt es nach jeder Technikübung die Möglichkeit, die Übung zu bewerten. Der Nutzer wird gefragt, wie anstrengend die Übung empfunden wurde und wie er diese bewerten würde. In der Datenbank wird dann für den Nutzer eine Klasse WorkoutStepExecution angelegt, in der die Bewertungen gespeichert werden. Abbildung 8 zeigt einen Ausschnitt der Struktur der Klasse WorkoutStepExecution der Datenbank.

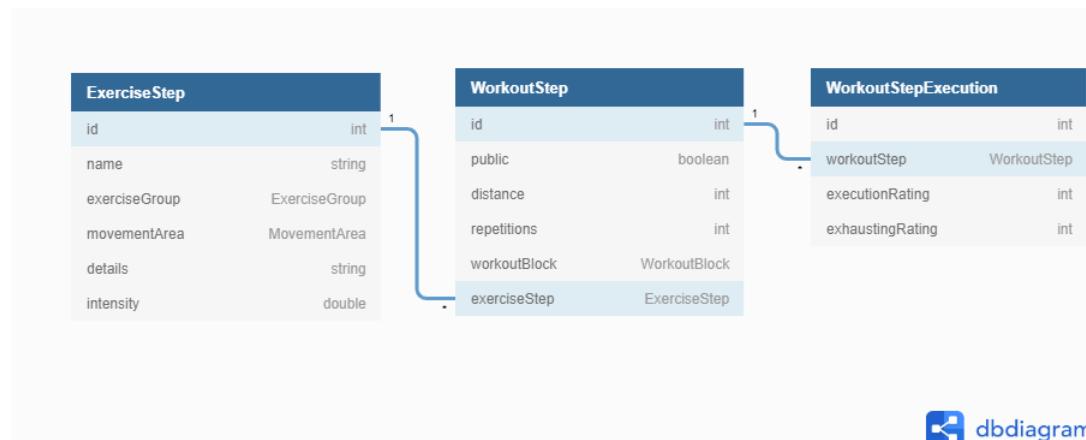


Abbildung 8: Ausschnitt der Datenbank für Trainingsverlauf

Nach der Bewertung des Nutzers liegen Daten vor, ob eine Übung in seinem Sinne gut oder schlecht verlief. Dieser bewertet die Übung mit einer Zahl zwischen 0 und 4. Bei einer schlechten Bewertung werden für ihn Alternativübungen bereitgestellt und bei guter Bewertung Folgeübungen. Die Entscheidungen, ob eine Folgeübung oder eine Alternativübung hinzugefügt wird, sind in einem späteren Abschnitt in dem Aktivitäts-Diagramm in Abbildung 13 dargestellt [16].

3.3.4 Dauer eines Trainings

Um das Problem der Dauer zu lösen (siehe Abschnitt 3.1.4) muss jeder Nutzer eine Übungsart auf mindestens 100 Meter schwimmen. Dadurch bekommt man eine ungefähre zeitliche Angabe, wie lange der Nutzer für diese Art der Übung braucht. Wenn eine Übung in ein Training eingesetzt wird, soll darauf geachtet werden, dass die Übung zeitlich in die Gesamtzeit des Trainings passt. Andernfalls muss eine andere Übung gefunden werden. In dieser Arbeit wird davon ausgegangen, dass diese Zeiten in der Datenbank existieren und ermittelt wurden.

3.4 Trainingsplan-Algorithmus Theorie

In diesem Abschnitt wird der Ablauf des Trainingsplan-Algorithmus beschrieben. Durch Grafiken und Beschreibungen soll die Idee dahinter vermittelt werden. Alle Funktionen und Klassen, die umgesetzt werden sollen, sind in der Abbildung 14 zu sehen.

3.4.1 Parameter

Für den Start der Erstellung eines Training muss zunächst eine Art von Pointer oder eine UserID (einheitliche Identifikationsnummer des Nutzers) an die Funktion übergeben werden. Die Nutzerinformationen werden aus der Datenbank abgefragt.

3.4.2 Nutzerdaten

Um das Intensitätsmaximum des Nutzers auszurechnen, müssen zunächst die Grundfähigkeiten in den Schwimmstilen abgefragt werden. Außerdem sind für die Trainingslänge, die vom Nutzer angegebene minimale und maximale Trainingsdauer notwendig. Diese befinden sich in der Klasse *UserData*

der Datenbank und werden durch die Funktion `getUserData` per Query abgefragt.

3.4.3 Trainingsdauer und Distanz

Zunächst wird die Trainingsdauer bestimmt. Dies passiert durch die Funktion `getDuration`. Diese bekommt als Parameter die minimale und maximale Zeit in Sekunden und bestimmt zufällig eine Trainingsdauer. Die Distanz in Metern wird aus dem Intensitätsmaximum und der Trainingsdauer in der Funktion `getDistance` berechnet.

3.4.4 Bereichseinteilung

Um zu entscheiden, wie viel Prozent jedes Ziel (siehe Abbildung 5) im Training einnimmt, wurde in der bereits abgefragten Klasse `UserData` das priorisierte Ziel des Nutzers unter `prefAim` gespeichert. Die Funktion `setProportions` setzt diese Bereiche abhängig des priorisierten Ziels. Wenn beispielsweise der Sprint im Fokus des Nutzers liegt, wird das nächste Training zu 60% aus Sprintübungen, 20% aus Ausdauerübungen und 20% aus Technikübungen zusammengesetzt.

Dabei sind allerdings die Aufwärm- und Cooldown-Übungen ausgeschlossen, denn diese werden bei jedem Training fest vorgegeben. Abbildung 9 zeigt eine mögliche Bereichsverteilung.

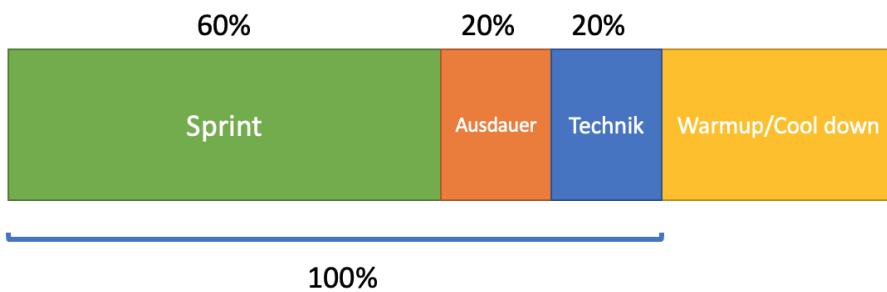


Abbildung 9: Bereichsverteilung bei Priorität Sprint

3.4.5 Boost (prozentuale Erhöhung der Bereiche)

Damit nicht bei jedem Training, die gleiche Verteilung der Bereiche vorliegt, werden Bereiche zufällig erhöht. In diesem Zusammenhang wird vom

einem Boost gesprochen. Abbildung 10 zeigt zwei mögliche Verteilungen, der Abbildung 9, entstanden durch die Funktion *boostProportions*.

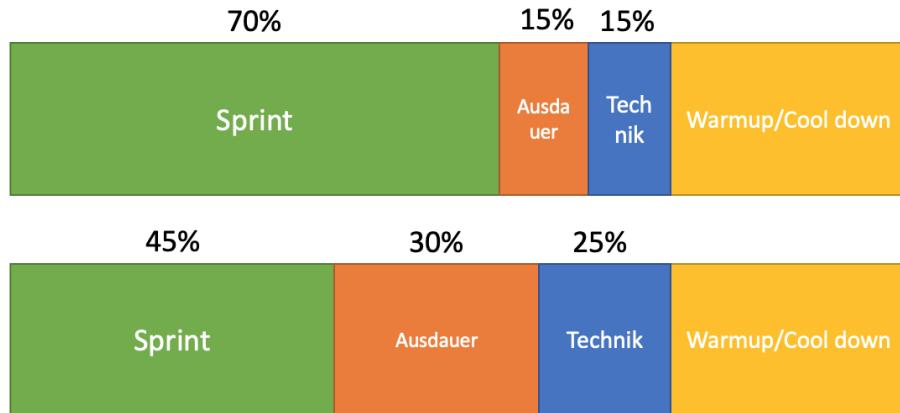


Abbildung 10: Bereichsverteilung bei Priorität Sprint (Boosted)

Durch die zufällig addierten Prozente in einem Bereich, kann es einerseits sein, dass alle Bereiche ungefähr gleich verteilt sind. Andererseits kann auch der priorisierte Bereich, der schon mehr Anteile besitzt, erhöht werden. Insgesamt bleibt der Gesamtanteil aber gleich, d.h. zu einem Bereich addierte Prozente werden von den anderen Bereichen wieder abgezogen.

3.4.6 Trainingsverlauf

Nachdem die Verteilung der Bereiche prozentual festgelegt wurden, wird der Trainingsverlauf des Nutzers abgefragt. Die Funktion *getWorkoutExecutions* fragt alle *WorkoutStepExecution* Einträge der Datenbank ab. Dort befinden sich die Bewertungen der Übungen. Anschließend werden in dem Bereich Sprint-/ Ausdauer und Technik die Übungen anhand des Attributes *createdAt* zeitlich sortiert.

3.4.7 Sortieren der Übungen

Aus dem präferierten Schwimmstil und einem Bereich (Sprint, Ausdauer, Technik), werden in der Funktion *getExerciseSteps* alle passenden Übungen aus der Datenbank abgefragt und gespeichert. Dann wird das Array der

Übungen aus dem Trainingsverlauf mit der möglichen Übungen verglichen und zeitlich anhand des Attributes `createdAt` sortiert.

Abbildung 11 zeigt ein Beispiel für zwei Arrays an Übungen, aus denen dann ein Ergebnis-Array entsteht.

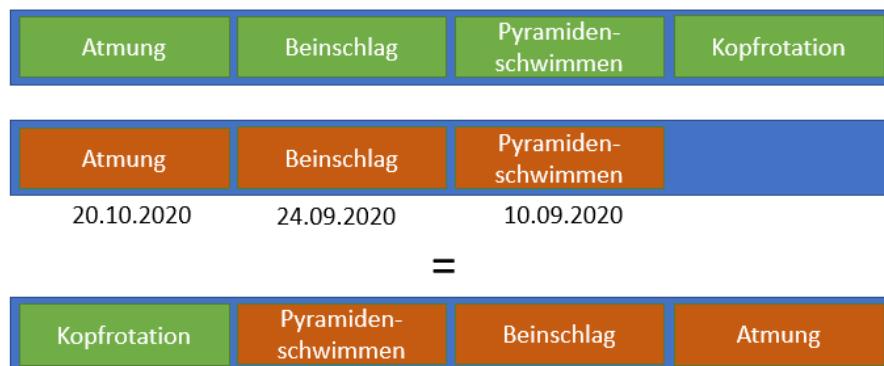


Abbildung 11: Sortierung der Post-Übungen mit möglichen Übungen

Das erste Array (nur grüne Elemente) zeigt die möglichen Übungen, die durch die Funktion `getExerciseSteps` abgefragt wurden.

Das zweite Array (nur orangene Elemente) beinhaltet alle in der Vergangenheit ausgeführte Übungen. Durch das Sortieren durch die Funktion `sortExercises`, wird ein neues Array mit grünen und orangenen Elementen gefüllt. Die Übung Kopfrotation steht in dem neu entstandenen Array an erster Stelle, da sie nicht im orangenen Array zu finden ist und somit für den Nutzer unbekannt ist. Die Übung Pyramiden-Schwimmen steht an zweiter Stelle des Ergebnis-Arrays. Zwar wurde sie in der Vergangenheit bereits vom Nutzer in einem Training ausgeführt, jedoch gibt es erstens keine weitere Übung, die für den Nutzer unbekannt ist und zweitens steht diese an letzter Stelle in dem orangenen Array und ist mit dem ältesten Datum versehen. Die Sortierung der anderen Übungen ergeben sich nach dem gleichen Schema.

3.4.8 Hinzufügen der Übungen

Beim Hinzufügen einer Übung in einen Bereich, wird zwischen Technik- / Sprint und Ausdauer-Übungen unterschieden (vgl. Abschnitt 2.7). Abbildung 12 zeigt das Aktivitäts-Diagramm der Logik für das Hinzufügen einer

Sprint- und Ausdauerübung.

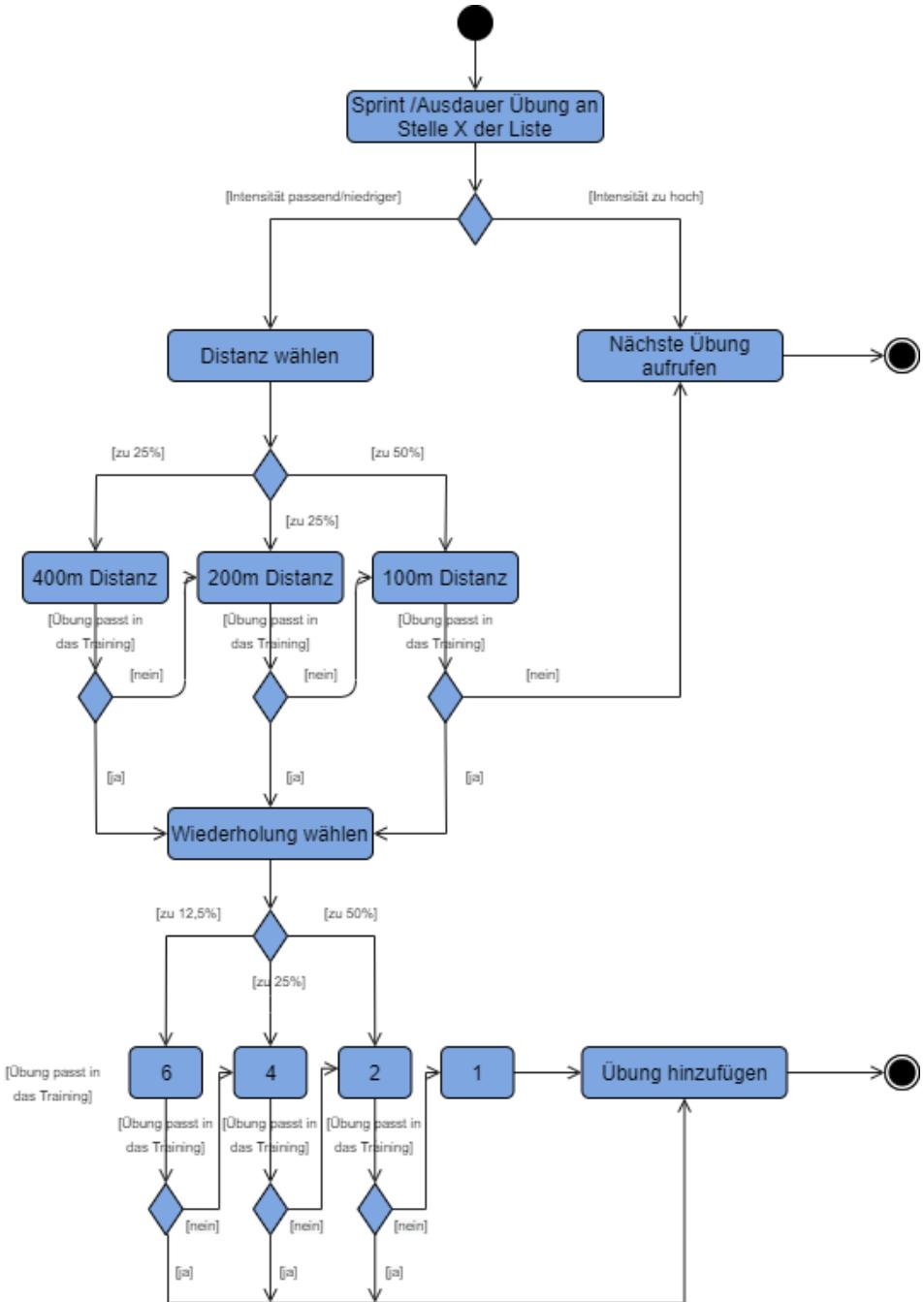


Abbildung 12: Aktivitätsdiagramm bei Sprint-/ Ausdauer-Übungen

Der Ausgangspunkt ist ein ExerciseStep aus einem Array an möglichen Übungen. Da für jedes Training ein Intensitätsmaximum definiert ist und jeder ExerciseStep eine Intensität besitzt, wird zunächst abgefragt, ob das Aufsummieren der Übungsintensitäten das Intensitätsmaximum überschreitet. Falls dies der Fall ist, wird die Funktion an dieser Stelle beendet und es wird nach einer neuen Übung mit weniger Intensität in dem Array an möglichen Übungen gesucht.

Falls das Intensitätsmaximum nicht überschritten ist, wird im nächsten Schritt die Distanz der Übung gewählt. Zu 50% wird dabei eine Distanz von 100m, zu 25% eine Distanz von 200m und zu ebenfalls 25% eine Distanz von 400m gewählt. Danach wird abgefragt, ob die Übung mit dieser Distanz und dementsprechender Dauer in das Training passt. Wenn die Distanz bei 200m und 400m nicht ins Training passt, wird sie auf die nächst kleinere Einheit reduziert. Sollten auch 100m dieser Übung nicht in das Training passen, wird nach einer anderen Übung gesucht. Passt die zufällig gewählte Distanz in das Training, wird die Anzahl an Wiederholungen bestimmt.

Nach dem gleichem Schema wird bei der Wiederholungsanzahl verfahren. Es gibt 3 Möglichkeiten (2x,4x,6x), die zufällig ausgewählt werden. Anschließend wird geprüft, ob die Übung mit gewählter Distanz multipliziert mit der Anzahl an Wiederholungen in das Training passt. Falls die Anzahl an Wiederholungen nicht passt, wird diese reduziert. Sollte die zufällig bestimmte Anzahl an Wiederholungen ohne Reduzierung bereits passen, wird die Übung mit der Distanz und den Wiederholungen dem Training hinzugefügt.

Bei Technikübungen muss vor dem Durchlaufen des Aktivitäts-Diagramms, das in Abbildung 12 beschrieben ist, eine weitere Entscheidung getroffen werden. Die Abbildung 13 zeigt das Aktivitäts-Diagramm, um zu entscheiden ob die nächste Übung eine bereits Angefangene-, Folge- oder Alternativ-Übung sein soll.

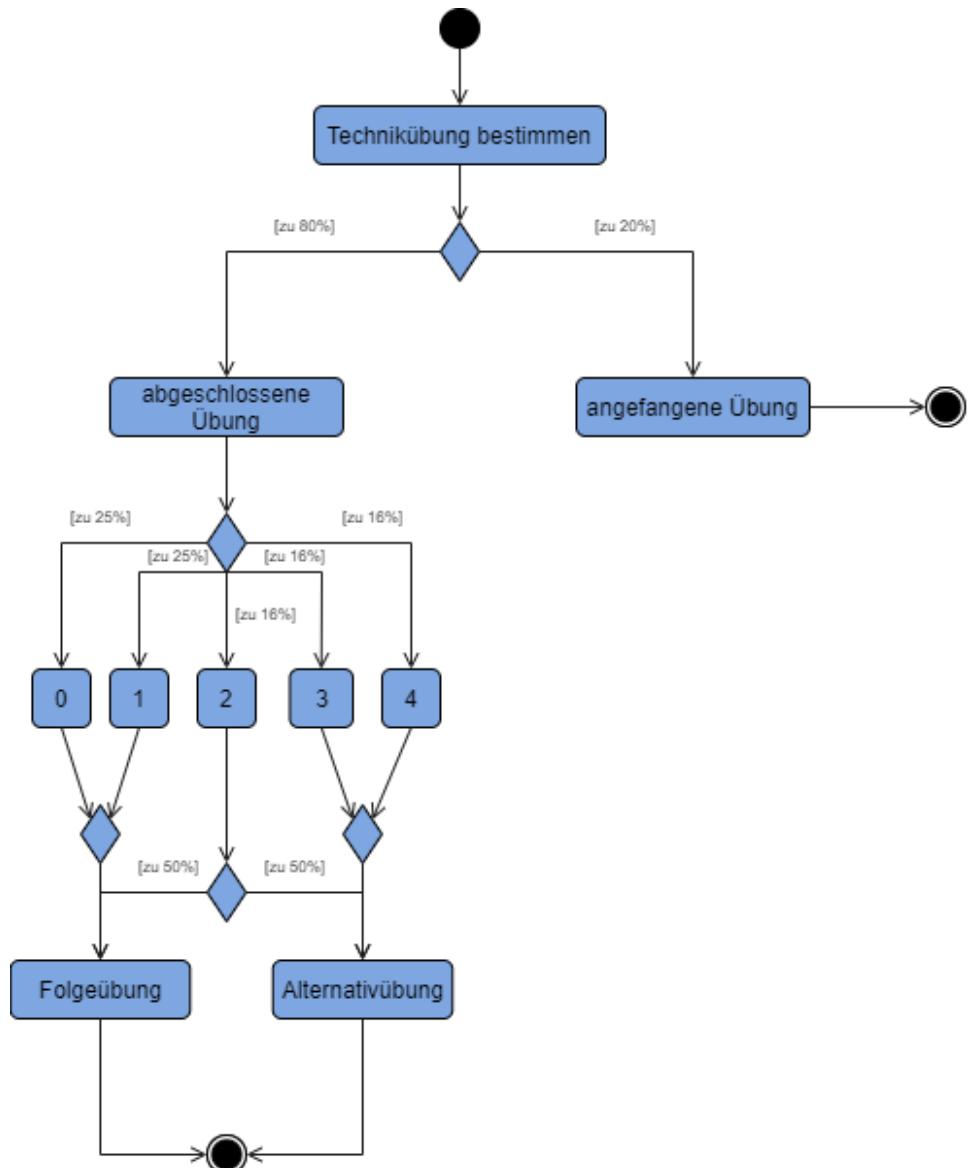


Abbildung 13: Aktivitätsdiagramm bei Technikübungen

Nach dem Ausgangspunkt werden zu 20% vom Nutzer bereits angefangene, aber nicht beendete Übungen für das nächste Training ausgewählt. Dies ist sinnvoll, da der Nutzer die angefangene Übung nach dem gleichen Ablauf im vorherigen Training zugewiesen bekommen hat. Zu 80% wird eine neue Technikübung anhand alt Bewerteten ausgewählt. Der Nutzer kann jeder Technikübung eine Bewertung von 0 bis 4 Sternen geben. Mit einer Wahr-

scheinlichkeit von 25% werden die Entscheidungswege Null -und Ein Stern abgelaufen. Der Nutzer soll bei schlecht bewerteten Übungen auf Alternativen zugreifen können. Zu jeweils 16% werden die Technikübungen mit 3 bis 4 Sternen ausgewählt und dann nach einer passenden Folgeübung gesucht. Die 2 Sterne Übungen werden mit einer Wahrscheinlichkeit von 16% ausgewählt.

Folgt man diesen Entscheidungsknoten, gibt es zwei weitere Zustandsknoten, mit jeweils 50% Wahrscheinlichkeit. Es wird per Zufallsprinzip entschieden, ob der Übung eine Folgeübung oder eine Alternativübung folgt, da bei 2 Sternen nicht eindeutig ist, was diese Bewertung vom Nutzer bedeutet. Außerdem können anhand des Trainingsverlaufes Übungen sortiert werden. So befinden sich vor kurzem ausgeführte Übungen eher am Ende des Übungs-Arrays und für den Nutzer unbekannte Übungen weiter vorne.

3.4.9 Speichern des Trainings

Nachdem für jeden Bereich des Trainings passende Übungen gefunden wurden, ist entweder das Intensitätsmaximum des Nutzers, die maximale Trainingsdauer oder die maximale Distanz erreicht. Im letzten Schritt wird das gefüllte Objekt *Workout* an die Cloud-Funktion *saveWorkout* gegeben, die dieses in der Datenbank speichert.

3.5 Aufbau der Klassen und Methoden im Code

Abbildung 14 beschreibt den genannten Aufbau der Klassen und Funktionen als Klassen-Diagramm [18].

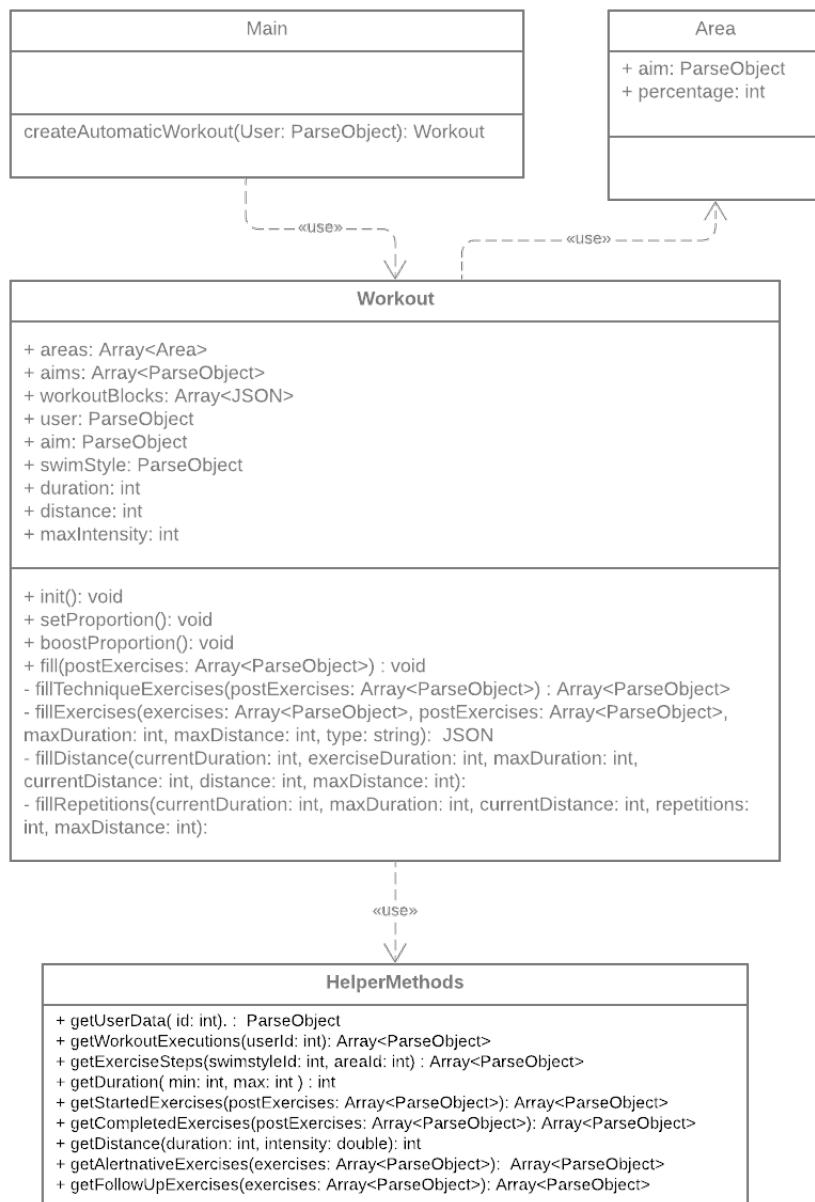


Abbildung 14: Klassendiagramm auf dem Parse-Server

4 Implementierung und Ergebnis

Alle in diesem Abschnitt gezeigten Ausschnitte des Quellcodes wurden in Englisch kommentiert.

4.1 Abfrage der Nutzerinformationen

Beim Aufruf der Funktion *createAutomaticWorkout* (vgl. Abbildung 14) wird das Feature der automatischen Erstellung eines Trainingsplans gestartet. Zuerst wird überprüft, ob ein Nutzer, der in die Funktion übergeben wird, existiert. Das Feature soll nur angemeldeten Nutzern zur Verfügung stehen. Danach müssen weitere Informationen über den Nutzer aus der Datenbank abgefragt werden. In der Klasse *UserData* der Datenbank befinden sich der präferierte Schwimmstil, das Ziel und die ausgewählte Trainingsdauer des Nutzers. Durch die Cloud-Funktion *getUserData* wird der UserData-Eintrag des Nutzers gefunden. Codeausschnitt 2 zeigt den kommentierten Quellcode für das Abfragen der Nutzerinformationen aus der Klasse *UserData*. Die Funktion *getUserData* ist im Anhang als Quellcode zu finden.

```
1 // User is sent as a parameter in the Parse-Cloud-Function
2 let user = request.params.user;
3
4 // Queries whether the user is null or undefined
5 if (contentCheck.isEmpty(user)) {
6     throw new Error("Error in createAutomaticWorkout:
7     A user must exist to create a training")
8 }
9
10 // Queries specific data for the user
11 const params = { user: user };
12 let userData = await Parse.Cloud.run("getUserData",params);
13
14 // Queries whether the userData is null or undefined
15 if (contentCheck.isEmpty(userData)) {
16     throw new Error("Error in createAutomaticWorkout:
```

```

17     UserData must exist in order to create a training");
18 }
19
20 // Dissolves the UserData data package
21 // and saves the data in its own variables
22 const userDuration = userData.get('workoutDuration');
23 const userAim = userData.get('aim');
24 const userSwimStyle = userData.get('prefSwimStyle');
25
26 // Queries whether the userDuration, userAim or
27 // userSwimStyle is null or undefined
28
29 if (!userDuration || !userAim || !userSwimStyle) {
30     throw new Error("Error in createAutomaticWorkout:
31         userDuration or userAim or prefSwimStyle is empty");
32 }
```

Codeausschnitt 2: Abfrage der Nutzerinformationen

Im nächsten Schritt werden die Grundfähigkeiten des Nutzers abgefragt. Diese befinden sich in der Klasse *Technique* der Datenbank und werden durch die Cloud-Funktion *getTechniques* abgerufen. Aus allen Grundfähigkeiten für die verschiedenen Stile wird dann der Eintrag für den präferierten Schwimmstil herausgefiltert (siehe Codeausschnitt 3). Die Funktion *getTechniques* ist im Anhang vollständig als Quellcode zu finden.

```

1 // Queries the basic skills of the user
2 // in the Technique table of the database
3 const params = { user: user };
4 const techniques = await Parse.Cloud.run("getTechniques",
5 params);
6
7 // Queries whether the technique is empty or undefined
8 if (contentCheck.isEmpty(techniques)) {
```

```

9   throw new Error("Error in createAutomaticWorkout:
10  The user did not provide any
11    information about his or her basic skills");
12 }
13 // Finds the preferred style in the array
14 // of basic skills per style
15 const technique = techniques.find(technique =>
16 technique.get("swimStyle").id === swimStyle.id);

```

Codeausschnitt 3: Abfrage der Grundfähigkeiten

4.2 Berechnung der Trainingsdaten

Aus der Grundfähigkeit des präferierten Schwimmstils wird das Intensitätsmaximum errechnet. Anhand der präferierten Trainingsdauer wird die Gesamtdauer zufällig bestimmt und aus dem Intensitätsmaximum und der Dauer, dann die Distanz für einen Nutzer errechnet.

Codeausschnitt 4 zeigt den Aufruf der Methoden und die Berechnung.

```

1 // Calculates the maximum intensity for the user
2 const intensityFactor = 4;
3 const maxIntensity = (technique.get("value") * intensityFactor);
4
5
6 // Determines the duration of the training in seconds
7 const duration = helperFunctions.getDuration(userDuration);
8
9 // Determines training distance in meter
10 const distance = helperFunctions.getDistance(duration,
11 intensity);

```

Codeausschnitt 4: Aufruf der Hilfsfunktionen und Berechnung der maximalen Intensität

Im Codeausschnitt 5 ist die Funktion *getDistance* im Detail dargestellt. Übergeben werden die Parameter Dauer und Intensitätsmaximum. Anhand des Intensitätsmaximum wird eingeschätzt, ob es sich bei dem Nutzer um einen Anfänger oder Fortgeschrittenen handelt. Wenn der Nutzer ein Intensitätsmaximum unter 10 (selbst erzeugte Einheit) aufweist, dann wird für ihn eine Zeit von 33,3 Minuten (1998 Sekunden) pro Kilometer angesetzt und er wird als Anfänger klassifiziert.

Andernfalls wird er als Fortgeschrittenen identifiziert und seine Zeit wird auf 20 Minuten (1200 Sekunden) pro Kilometer angenommen. Folgende Formel wird für Berechnung der Gesamtdistanz des Trainings verwendet: $x = b*c/a$ [17]

x = Gesamtdistanz in Metern

a = 1998 bzw. 1200 Sekunden

b = 1000 Meter

c = Gesamtdauer in Sekunden

```

1 getDistance: function (duration, intensity) {
2     // Determines whether the user is a beginner
3     // or an advanced user based on the duration
4     // of the workout and the technique of the style
5
6     let distance;
7
8     if (intensity < 10) {
9         // 1998 seconds = 1 km
10        distance = this.ruleOfThree(1998, 1000, duration);
11    } else {
12        // 1200 seconds = 1 km
13        distance = this.ruleOfThree(1200, 1000, duration);
14    }
15    return distance;
16 }
```

Codeausschnitt 5: Funktion *getDistance*

Nachdem alle Parameter für das Training erstellt worden sind, wird eine Instanz der Klasse *Workout* erzeugt und die Parameter in den Konstruktor übergeben. Darauf folgt der Aufruf der Funktion *init*, *setProportion* und *boostProportion* der Klasse *Workout* (vgl. siehe Abbildung 14) (siehe Codeausschnitt 6).

```

1 const workout = new Workout(user, userAim, userSwimStyle,
2 duration, distance, intensity);
3
4 workout.init();
5 workout.setProportion();
6 workout.boostProportion();
7
8
9 // Class Workout (contructor)
10 constructor(user, aim, swimStyle, duration, distance,
11 maxIntensity) {
12     this.user = user;
13     this.aim = aim;
14     this.swimStyle = swimStyle;
15     this.duration = duration;
16     this.distance = distance;
17     this.maxIntensity = intensity;
18 }
```

Codeausschnitt 6: Aufruf der *Workout* Methoden und Konstruktor der *Workout* Klasse

Die Funktion *init* ist eine asynchrone Funktion, die durch die Cloud-Funktion *getAims* (Quellcode siehe Anhang) alle auswählbaren Ziele in der Datenbank abfragt und diese bereitstellt. Die Funktion *setProportion* setzt die prozentualen Werte für die Ziele. Beim Iterieren über dem Array aus Zielen wird überprüft, ob es sich um das Hauptziel handelt. Falls es das Hauptziel ist, wird der Wert 60% in die *Area* Instanz gegeben. Andernfalls der Wert *percentage*, also der Prozentanteil, der übrig ist, durch die Anzahl

an restlichen Bereichen geteilt. Danach wird das Objekt *area* in das Array der *areas* eingefügt. Es wird später benötigt, um die richtigen Übungen für einen Bereich zu finden. Codeausschnitt 7 zeigt den Ablauf der Funktion *setProportion*.

```

1 setProportion() {
2
3     const mainPercentage = 60;
4     const secondPercentage = 40;
5
6     // Calculate the percentage of the area
7     const count = this.aims.length;
8         const percentage = secondPercentage / count;
9
10    // If it is the main goal, the area gets assigned 60%.
11    // Otherwise the respective percentage
12    // depending on the total number of goals
13
14    for (let aim of this.aims) {
15        let area;
16        if (aim.id === this.aim.id) {
17            area = new Area(aim, mainPercentage);
18        } else {
19            area = new Area(aim, percentage);
20        }
21        this.areas.push(area);
22    }
23 }
```

Codeausschnitt 7: Funktion *setProportion*

Die Funktion *boostProportion* (vgl. Abschnitt 3.4.5) erhöht zufällig die prozentuale Anteile der Bereiche. Zuerst wird ein Wert zwischen 5% und 10% gewählt. Dann wird eine Stelle des Array *areas*, das die *Area* Objekte beinhaltet, zufällig bestimmt, die diesen Wert addiert bekommt. Damit dadurch

nicht mehr Prozente als vorher im Array *areas* zu finden sind, wird in *balanceNumber* der abzuziehende Teil für die anderen Bereiche berechnet und durch eine Iteration über das Array pro Bereich abgezogen. Codeausschnitt 8 zeigt den kommentierten Quellcode der Funktion *boostProportion*.

```
1 boostProportion() {
2     // the proportions of the areas
3     // are randomly changed by 5-10%
4     // randomly chooses a percentage between 5 and 10
5     const percentageBoost = helperFunctions
6         .getRandomIntInclusive(5, 10);
7
8     const length = this.areas.length;
9
10    // Increases the portion of the area
11    // at one point in the array
12    const indexBoost = helperFunctions
13        .getRandomIntInclusive(0, length - 1);
14
15    this.areas[indexBoost].percentage += percentageBoost;
16
17    // Calculates the factor by which
18    // all other areas must be reduced again
19    const balanceNumber = percentageBoost / count;
20    // subtracts the factor for each area
21    for (let i = 0; i < this.areas.length; i++) {
22        if (i !== indexBoost) {
23            this.areas[i].percentage -= balanceNumber;
24        }
25    }
26}
```

Codeausschnitt 8: Funktion *boostProportion*

4.3 Abfrage der Post-Übungen

Nach Festlegung der prozentualen Bereiche des Trainings, folgt der Vergleich der möglichen Übungen mit denen aus vergangenen Trainingseinheiten (vgl. Elemente grün mit Elementen orange in Abbildung 11). Die Cloud-Funktion *getWorkoutStepExecutions* fragt in der Datenbank alle *WorkoutStepExecution* Objekte für den Nutzer ab. Die Funktion bekommt als Parameter den Nutzer übergeben und prüft zuerst ob dieser existiert. Ein *WorkoutStepExecution* Eintrag besitzt einen Pointer für den Nutzer und kann deswegen für jeden Nutzer einfach abgefragt werden. Damit später auch der *ExerciseStep* des *WorkoutSteps* des *WorkoutStepExecution* Eintrages (vgl. Abbildung 8) mit dem der möglichen Übungen für das Training verglichen werden kann, sind insgesamt 3 Abfragen notwendig, die mit dem Parse-Befehl *matchQuery* verbunden werden. Der Codeausschnitt 9 zeigt die Cloud-Funktion *getWorkoutStepExecution*.

```
1 Parse.Cloud.define('getWorkoutStepExecutions',
2   async (request, response) => {
3     let userPointer;
4
5     const userId = request.params.userId;
6     /**
7      * Queries whether the user is null or undefined
8      */
9     if (contentCheck.isEmpty(userId)) {
10       throw new Error("Error in getUserData:
11         A user id must exist to query UserData");
12     }
13     /**
14      * Creates a pointer to get all the attributes
15      * of the UserData class for the correct
16      * user in the database
17      */
18     const userObject = Parse.Object.extend("User");
19     const userPointer = userObject.createWithoutData(userId);
20 }
```

```

21
22
23 const WorkoutStepExecution = Parse.Object.
24 extend("WorkoutStepExecution");
25
26 const ExerciseStep = Parse.Object.extend("ExerciseStep");
27 const WorkoutStep = Parse.Object.extend("WorkoutStep");
28
29 const executionQuery = new Parse.
30 Query(WorkoutStepExecution);
31
32 executionQuery.equalTo('user', userPointer);
33 executionQuery ascending("createdAt");
34 executionQuery.include('workoutStep');
35 executionQuery.include('workoutStep.exerciseStep');
36
37
38 const exerciseStepQuery = new Parse.Query(ExerciseStep);
39 exerciseStepQuery.exists('intensity');
40 exerciseStepQuery.exists('duration');
41
42 const workoutStepQuery = new Parse.Query(WorkoutStep);
43 workoutStepQuery.matchesQuery('exerciseStep',
44 exerciseStepQuery)
45
46 executionQuery.matchesQuery('workoutStep',
47 workoutStepQuery);
48
49 return await executionQuery.find({useMasterKey: true});
50 });

```

Codeausschnitt 9: Cloud-Funktion getWorkoutStepExecutions

4.4 Übungen hinzufügen

Nachdem alle Post-Übungen abgefragt wurden, wird als nächstes die Funktion *fill* der Klasse *Workout* ausgeführt. Übergeben werden die abgefragten Post-Übungen. Codeausschnitt 10 zeigt den kommentierten Quellcode der Funktion *fill*.

```
1  async fill(postExercises) {
2
3      // Iterates across all areas
4      for (let area of this.areas) {
5
6          let workoutBlock;
7
8          // Calculate percentages for distance and
9          // duration for each range using
10         const areaDuration = this.duration * (area.percentage / 100);
11         const areaDistance = this.distance * (area.percentage / 100);
12
13         // Query possible exercises that
14         // correspond to the swimming style
15         // and the area to be tested.
16         let exercises = await Parse.Cloud.run("getExerciseSteps",
17             { swimStyleId:this.swimStyle.id, areaId: area.info.id });
18
19
20         // Distinguish between technique, sprint,
21         // endurance and loose swimming
22         if (area.info.id === "oQ4PDCAwx0") {
23             // technique
24             exercises = this.fillTechniqueExercises(postExercises);
25
26             workoutBlock = this.fillExercises(exercises,
27                 postExercises, areaDuration, areaDistance, "Technik");
28         }
29 }
```

```

30     if (area.info.id === "jCiN5848jd") {
31         // sprint
32         workoutBlock = this.fillExercises(exercises,
33             postExercises, areaDuration, areaDistance, "Sprint");
34     }
35
36     if (area.info.id === "8uvl9Arapv") {
37         // endurance
38         workoutBlock = this.fillExercises(exercises,
39             postExercises, areaDuration, areaDistance, "Ausdauer");
40     }
41     this.workoutBlocks.push(workoutBlock);
42 }
43 }
```

Codeausschnitt 10: Funktion *fill* der Klasse *Workout*

Bei dem Iterieren über das Array *areas* um den Sprint oder Ausdauer Bereich, wird die Funktion *fillExercises* aufgerufen. Beim Technikbereich wird die Funktion *fillTechniqueExercises* aufgerufen und danach die Funktion *fillExercises*.

4.4.1 Ausdauer- und Sprintbereich

Nach dem Schema, das in der Abbildung 12 durch ein Aktivitätsdiagramm dargestellt ist, werden durch die Funktion *fillExercises* Übungen, Distanz und Wiederholungen bestimmt. Codeausschnitt 11 zeigt den kommentierten Quellcode der Funktion *fillExercises*.

```

1 fillExercises(exercises, postExercises, maxDuration,
2 maxDistance, type) {
3
4     // All exercises, the post exercises,
5     // the duration and distance are handed over
```

```

7 let workoutBlock = { "name": type, "workoutSteps": [] };
8
9 // A training history is available,
10 // the exercises will be from old to new
11 if (postExercises.length > 0) {
12     exercises = helperFunctions
13         .sortExercises(exercises, postExercises);
14 }
15
16 let count = 0; // Count up the Exercises
17
18 // Fits becomes false when either maxIntensity,
19 // the maximum duration or maximum distance are reached
20 let fits = true;
21
22 // Variables that are increased when an exercise is added
23 let intensity = 0;
24 let currentDuration = 0;
25 let currentDistance = 0;
26
27 while (fits) {
28     const exercise = exercises[count];
29
30     let exerciseDistance;
31     let exerciseRepetitions;
32
33     // If the maxIntensity is exceeded,
34     // no more exercises are added
35     // and fits becomes false.
36     const exerciseIntensity = exercise.get('intensity');
37     const exerciseDuration = exercise.get('duration');
38
39     if (intensity + exerciseIntensity
40         > this.maxIntensity) {
41
42         // Condition that applies with a selected chance

```

```

43     const r1 = Math.random();
44     if (r1 < .5) {
45         exerciseDistance = this.fitsDistance(
46             currentDuration,
47             maxDuration,
48             currentDistance,
49             100,
50             maxDistance);
51     } else if (r1 < .75) {
52         exerciseDistance = this.fitsDistance(
53             currentDuration,
54             maxDuration,
55             currentDistance,
56             200,
57             maxDistance);
58     } else {
59         exerciseDistance = this.fitsDistance(
60             currentDuration,
61             maxDuration,
62             currentDistance,
63             400,
64             maxDistance);
65     }
66
67     // If the function fitsDistance returns null,
68     // the exercise does not fit into the course
69     if (exerciseDistance != null) {
70
71
72         // If the exercise fits in, the variables are increased
73
74         currentDistance += exerciseDistance;
75         intensity += exerciseIntensity;
76         currentDuration += exerciseDuration * (currentDistance / 100);
77
78

```

```

79   // Secondly, the repetitions are determined
80   const r2 = Math.random();
81   if (r2 < .5) {
82     exerciseRepetitions = this.fitsRepetitions(
83       currentDuration,
84       maxDuration,
85       currentDistance,
86       2,
87       maxDistance);
88   } else if (r2 < .75) {
89     exerciseRepetitions = this.fitsRepetitions(
90       currentDuration,
91       maxDuration,
92       currentDistance,
93       4,
94       maxDistance);
95   } else {
96     exerciseRepetitions = this.fitsRepetitions(
97       currentDuration,
98       maxDuration,
99       currentDistance,
100      6,
101      maxDistance);
102   }
103
104   // The function fitsRepetitions returns at least
105   // one repetition and can return up to 6 repetitions
106   // if there is enough space in the workout
107
108   // Multiplies the repetition factor by
109   // the current duration and distance of the training
110   distance *= exerciseRepetitions;
111   duration *= exerciseRepetitions;
112
113   // Creates a JSON object that is later
114   // passed into the saveWorkout function

```

```

115     workoutBlock.workoutSteps.push(
116     {
117         "exerciseStepId": exercise.id,
118         "distance": currentDistance,
119         "repetitions": exerciseRepetitions
120     });
121 }
122 } else {
123     fits = false;
124 }
125 // When all exercises have been iterated over,
126 // no exercise fits into the training
127 // and fits are set false
128 if (count + 1 > exercises.length) {
129     fits = false;
130 }
131 count++;
132 }
133 return workoutBlock;
134 }
```

Codeauschnitt 11: Funktion fillExercises

Die Hilfsfunktionen *fitsDistance* und *fitsRepetitions* sind in Codeauschnitt 12 dargestellt. Beide Funktion sind rekursiv. Falls eine Übung aufgrund zu hoher Wiederholungsanzahl oder zu hoher Distanz (in Metern) nicht in das Training passt, wird der jeweilige Parameter durch einen rekursiven Aufruf reduziert bis die Übung passend für das Training ist, oder es wird nach einer passenden Übung gesucht. Die Funktionen werden nur für den Ausdauer-/ Sprintbereich angewendet.

```

1 fitsDistance(currentDuration, exerciseDuration, maxDuration,
2 currentDistance, distance, maxDistance) {
3 }
```

```

4   // The function returns either a distance or null
5   // Recursive function, that calls itself until
6   // the distance fits into the training or null is returned
7   switch (distance) {
8     case 100:
9       if ((currentDuration + exerciseDuration > maxDuration)
10          || (currentDistance + distance > maxDistance)) {
11         return null;
12       } else {
13         return 100;
14       }
15     case 200:
16       if ((currentDuration + exerciseDuration > maxDuration)
17          || (currentDistance + distance > maxDistance)) {
18         return fitsDistance(
19           currentDuration,
20           exerciseDuration,
21           maxDuration,
22           currentDistance,
23           100,
24           maxDistance);
25       } else {
26         return 200;
27       }
28     case 400:
29       if ((currentDuration + exerciseDuration > maxDuration)
30          || (currentDistance + distance > maxDistance)) {
31         return fitsDistance(
32           currentDuration,
33           exerciseDuration,
34           maxDuration,
35           currentDistance,
36           200,
37           maxDistance);
38       } else {
39         return 400;

```

```

40         }
41     }
42 }
43 fitsRepetitions(currentDuration, maxDuration, currentDistance,
44 repetitions, maxDistance) {
45     // The function returns either a repetition or 1
46     // Recursive function that calls itself until
47     // the repetition fits into the training or 1 is returned
48     switch (repetitions) {
49         case 2:
50             if ((currentDistance * repetitions > maxDistance)
51                 || (currentDuration * repetitions > maxDuration)) {
52                 return 1;
53             } else {
54                 return 2;
55             }
56         case 4:
57             if ((currentDistance * repetitions > maxDistance)
58                 || (currentDuration * repetitions > maxDuration)) {
59                 return this.fitsRepetitions(currentDuration,
60                     maxDuration, currentDistance, 2, maxDistance);
61             } else {
62                 return 4;
63             }
64         case 6:
65             if ((currentDistance * repetitions > maxDistance)
66                 || (currentDuration * repetitions > maxDuration)) {
67                 return this.fitsRepetitions(currentDuration,
68                     maxDuration, currentDistance, 4, maxDistance);
69             } else {
70                 return 6;
71             }
72     }
73 }

```

Codeauschnitt 12: Hilfsfunktion fitsDistance und fitsRepetitions

4.4.2 Technikbereich

Anders als bei den Sprint-/ Ausdauer-Übungen, wird in der Funktion *fill* beim Technikbereich zuerst die Funktion *fillTechniqueExercises* aufgerufen. Diese Funktion bekommt als Parameter alle Post-Übungen des Technikbereiches bereitgestellt und liefert entweder ein Array von Technikübungen, die vom Nutzer in der Vergangenheit angefangen wurden, ein Array von Folgeübungen oder ein Array von Alternativübungen. Das Aktivitätsdiagramm in Abbildung 13 beschreibt die Logik der Funktion *fillTechniqueExercises*, die im Codeausschnitt 13 als kommentierter Quellcode dargestellt ist. In dieser Funktion werden 4 verschiedene Hilfsfunktionen aufgerufen. Die Funktion *getCompletedExercises* filtert alle übergebenen Übungen nach vollständig abgeschlossenen Übungen und gibt dieses Array an Übungen zurück. Die Funktion *getStartedExercises* gibt alle nicht abgeschlossenen Übungen zurück. Die Funktion *getAlternativeExercises* bekommt ein Array von Übungen übergeben, mit dessen Hilfe in der Datenbank passende Alternativübungen gesucht und zurückgeliefert werden sollen. Die Funktion *getFollowUpExercises* bekommt ebenfalls ein Array von Übungen übergeben und liefert ein Array von Folgeübungen für die übergebenen Übungen zurück.

```
1 fillTechniqueExercises(postExercises) {  
2  
3     // Decides whether a follow-up exercise,  
4     // alternative exercise or started exercise  
5     // will be packed next in the training  
6  
7     // Condition that applies with a selected chance  
8     let temp;  
9     const r1 = Math.random();  
10    if (r1 < .2) {  
11        // completed exercises  
12        temp = helperFunctions  
13        . getCompletedExercises(postExercises);  
14  
15        const r2 = Math.random();  
16        if (r2 < .25) {
```

```

17         // 0 stars
18         return helperFunctions
19             .getAlternativeExercises(temp);
20     } else if (r2 < 0.5) {
21         // 1 stars
22         return helperFunctions
23             .getAlternativeExercises(temp);
24     } else if (r2 < 0.66) {
25         // 2 stars
26         const r3 = Math.random();
27         if (r3 < .5) {
28             return helperFunctions
29                 .getAlternativeExercises(temp);
30         } else {
31             return helperFunctions
32                 .getFollowUpExercises(temp);
33         }
34     } else if (r2 < 0.82) {
35         // 3 stars
36         return helperFunctions
37             .getFollowUpExercises(temp);
38     } else if (r2 < 1) {
39         // 4 stars
40         return helperFunctions
41             .getFollowUpExercises(temp);
42     }
43 } else {
44     // Started exercises
45     return helperFunctions
46         .getStartedExercises(postExercises);
47 }
48 }
```

Codeauschnitt 13: Funktion fillTechniqueExercises

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Entsprechend der Zielsetzung dieser Arbeit (vgl. Abschnitt 1.2) wurde ein Algorithmus zur Erstellung von eines Trainingsplans anhand der Ziele, Grundfähigkeiten und des Trainingsverlaufes eines Nutzers entwickelt. Dabei ermöglicht der implementierte Algorithmus durch alleinige Übergabe des Nutzers, das Erstellen eines Workout Objektes in der Datenbank, sofern die Bedingungen (vgl. Abschnitt 3.2) erfüllt sind.

Das Ergebnis dieser Arbeit sollte ein Bestandteil der vorhandenen Schwimm-App sein. Das bestehende System basiert auf Back4App und benutzt das Prinzip Parse als Backend-Server. Durch Cloud-Funktionen können Abfragen auf Datenbankeinträge (Query) gemacht werden.

Im ersten Schritt wurden weitere Parameter für Nutzer und Übungen hinzugefügt. Die Parameter für den Nutzer sind z.B. Grundfähigkeiten, präferierter Schwimmstil, die präferierte Trainingsdauer und das Ziel des Nutzers. Die Parameter für Übungen sind die Intensität und Dauer. Aus dem Ziel des Nutzers werden Bereiche gebildet, die angeben wie viel Prozent jeder Bereich in dem zu erzeugenden Training, einnimmt.

Der Trainingsverlauf des Nutzers beschreibt wie gut oder schlecht Übungen individuell bei dem Nutzer verliefen. Deswegen zählt auch dieser zu den Parametern und ist eine Kernbestandteil beim Hinzufügen von Übungen in das Training.

Die Logik für das Hinzufügen von Übungen wurde zuerst in verschiedenen Aktivitäts-Diagrammen modelliert und danach auf dem Parse-Server implementiert. Der entwickelte Trainingsplan-Algorithmus konnte in das bestehende System integriert werden.

Eine Aussage über den Sport-Theoretischen Hintergrund war nicht Bestandteil dieser Arbeit und sollte zukünftig erarbeitet werden.

5.2 Ausblick

5.2.1 Erweiterung der Ergebnisse dieser Arbeit

Im Rahmen einer zukünftigen Arbeit könnte der Algorithmus um die im Folgenden beschriebenen Funktionen erweitert werden.

Damit die Sport-Theoretischen Aspekte des Schwimmtrainings in den Trainingsplan-Algorithmus einfließen können, bedarf es einiger Änderungen. Diese Änderungen werden durch das H2Coach-Team in wöchentlichen Meetings ermittelt. Eine Änderung orientiert sich beispielsweise an der Frage, ob ein Benchmark, das der Nutzer absolvieren muss, verpflichtend ist, um den Trainingsplan-Algorithmus nutzen zu können. Darauf aufbauend ist außerdem die Ausgestaltung des Benchmark-Systems wichtig. Welche Übungen sollen geschwommen werden und warum?

Da die Implementierung des Features in einer NodeJS Umgebung weniger praktikabel ist (Javascript ist keine Objekt-orientierte Programmiersprache), könnte ein Wechsel von der Kombination Node und Javascript auf Node und Typescript [15] sinnvoll sein. Die Programmiersprache Typescript basiert auf Javascript, ist aber keine Skriptsprache sondern Objekt-orientiert.

Ein weiterer Ansatz könnte die Erweiterung des bestehenden H2Coach-Backends um einen Rest-API-Server in Java oder C# sein. Dadurch würde das Feature ausgelagert werden und unabhängig der restlichen Kernfunktionen der App laufen (verteilte Systeme).

5.2.2 Neuer Ansatz - Machine Learning

Statt einer Erweiterung des bestehenden Systems wäre es auch denkbar, dass Thema Machine Learning (ML) [19] bei der Trainingsplanerstellung aufzugreifen. Dafür müsste das ML-Netz allerdings eine große Menge an Eingabedaten in Form von Trainingsplänen in Abhängigkeit der Nutzer, zum Trainieren übergeben bekommen.

Da die Trainingspläne, die in der bestehenden H2Coach-App sind, nicht für Nutzer individuell abgestimmt sind, müssten Schwimmtrainer diese Informationen als Erfahrungswerte zur Verfügung stellen oder der in dieser Arbeit erzeugte Trainingsplan-Algorithmus wird zur Erzeugung der Trainingsdaten benutzt, sofern der Sport-Theoretische Aspekt komplett in diesen integriert wurde.

6 Abkürzungsverzeichnis

Workout Training

WorkoutBlocks Trainingsblöcke

WorkoutSteps Trainingsübungen

ExerciseStep Übung

ExerciseGroup Übungsgruppe

ExerciseStepType Übungsart

MovementArea Bewegungsbereich

ExerciseStepLevel Nutzerziele

SwimStyle Schwimmstil

distance Distanz in Metern für eine Übung

Distanz Gesamtdistanz

User Nutzer

UserData Benutzerinformationen

UI Benutzeroberfläche

repetitions Wiederholungen

7 Glossar

H2Coach

H2Coach eine Mobile-App für Schwimmanfänger aber auch Fortgeschrittene

REST

REST steht für Representational State Transfer und beschreibt die Regeln nach der eine API aufgebaut sein sollte.

SDK

SDK steht für Software Development Kit und beschreibt eine Gruppe an Werkzeugen, die das Programmieren von mobilen Anwendungen ermöglichen [21].

NoSQL

NoSQL steht für non-relational Data Management System, welches anders wie eine SQL-Datenbank kein genaues Schema benötigt [10].

MongoDB

SDK steht für Software Development Kit und ist ein Dokument-orientierte Datenbank, welche Daten in dem Format JSON speichert [10].

BaaS

Backend-as-a-Service (BaaS) ist ein Cloud-Service-Modell, bei dem Entwickler alle Hintergrundaspekte einer Web- oder Mobilanwendung auslagern, so dass sie nur das Frontend schreiben und warten müssen [5].

Pointer

Ein Pointer speichert die Referenz auf ein Objekt [20].

Post-Übungen

Eine Übung, die der Nutzer bereits absolviert hat.

Abbildungsverzeichnis

1	Datenbankstruktur	8
2	Workout Struktur	9
3	Struktur eines WorkoutBlocks am Beispiel Sprinteinheit . . .	9
4	Ausschnitt aus dem Aufbau des Dateiensystems auf dem Parse- Server	15
5	Ausschnitt aus der Klasse ExerciseStep	17
6	Datenbankklassen Technique, SwimStyle und User in Bezie- hung zueinander	18
7	Datenbankklasse ExerciseStep mit Intensität und Dauer . . .	19
8	Ausschnitt der Datenbank für Trainingsverlauf	19
9	Bereichsverteilung bei Priorität Sprint	21
10	Bereichsverteilung bei Priorität Sprint (Boosted)	22
11	Sortierung der Post-Übungen mit möglichen Übungen	23
12	Aktivitätsdiagramm bei Sprint-/ Ausdauer-Übungen	24
13	Aktivitätsdiagramm bei Technikübungen	26
14	Klassendiagramm auf dem Parse-Server	28

Quellcodeverzeichnis

1	Beispielcode Cloud-Funktion	10
2	Abfrage der Nutzerinformationen	29
3	Abfrage der Grundfähigkeiten	30
4	Aufruf der Hilfsfunktionen und Berechnung der maximalen Intensität	31
5	Funktion getDistance	32
6	Aufruf der Workout Methoden und Konstruktor der Workout Klasse	33
7	Funktion setProportion	34
8	Funktion boostProportion	35
9	Cloud-Funktion getWorkoutStepExecutions	36
10	Funktion fill der Klasse Workout	38
11	Funktion fillExercises	39
12	Hilfsfunktion fitsDistance und fitsRepetitions	43
13	Funktion fillTechniqueExercises	46
14	Cloud-Funktion getTechniques	56
15	Cloud-Funktion getUserData	57

Literatur

- [1] Back4App cloud-code dummy. <https://www.back4app.com/docs/android/parse-cloud-code>. Accessed: 2021-01-31.
- [2] Back4App what is back4app. <https://blog.back4app.com/what-is-back4app/>. Accessed: 2021-01-31.
- [3] Back4App what is parse server. <https://ariosoft.com/what-is-parse-server/>. Accessed: 2021-01-31.
- [4] Code Academy rest. <https://www.codecademy.com/articles/what-is-rest>. Accessed: 2021-01-31.
- [5] Coudfare what is baas? <https://www.cloudflare.com/de-de/learning/serverless/glossary/backend-as-a-service-baas/>. Accessed: 2021-01-31.
- [6] Dev-Insider was ist javascript? <https://www.dev-insider.de/was-ist-javascript-a-586580/>. Accessed: 2021-01-31.
- [7] docplayer trainingsplan. <https://docplayer.org/46259969-Schwimmen-trainingsplan.html>. Accessed: 2021-01-31.
- [8] fitforfun trainingsplan. <https://www.fitforfun.de/sport/trainingsplan/trainingsplan-schwimmen-so-schwimmen-sie-sich->. Accessed: 2021-01-31.
- [9] H2Coach swim app. <https://h2coach.com/>. Accessed: 2021-01-31.
- [10] MongoDB what is nosql? <https://www.mongodb.com/nosql-explained>. Accessed: 2021-01-31.
- [11] NDR schwimmen. <https://www.ndr.de/ratgeber/gesundheit/Schwimmen-Den-richtigen-Stil-trainieren,schwimmen1408.html#:~:text=Schwimmen%20trainiert%20die%20Muskeln%2C%20ohne,fast%20jeder%20Muskel%20trainiert%20wird.> Accessed: 2021-01-31.
- [12] Neumeyer-Abzeichen beliebteste sportarten. <https://www.neumeyer-abzeichen.de/blog/>

[die-fuenf-beliebtesten-sportarten-in-deutschland/#:~:text=Fast%2011%25%20aller%20Deutschen%20gehen,ein%20paar%20Runden%20zu%20drehen](https://de.wikipedia.org/w/index.php?title=Die_fuenf beliebtesten_Sportarten_in_Deutschland&oldid=112520000). Accessed: 2021-01-31.

- [13] Parse what is cloud-code. <https://docs.parseplatform.org/cloudcode/guide/>. Accessed: 2021-01-31.
- [14] ParsePlatform cloudcode funktionen. <https://docs.parseplatform.org/cloudcode/guide/>. Accessed: 2021-01-31.
- [15] Typescript typescript page. <https://www.typescriptlang.org/>. Accessed: 2021-01-31.
- [16] Visual-Paradigm aktivitätsdiagramm. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>. Accessed: 2021-01-31.
- [17] Wikipedia dreisatz. <https://de.wikipedia.org/wiki/Dreisatz>. Accessed: 2021-01-31.
- [18] Wikipedia klassen-diagramm. <https://de.wikipedia.org/wiki/Klassendiagramm#:~:text=Ein%20Klassendiagramm%20ist%20ein%20Strukturdiagramm,Klassen%2C%20Schnittstellen%20sowie%20deren%20Beziehungen>. Accessed: 2021-01-31.
- [19] Wikipedia machine learning. https://en.wikipedia.org/wiki/Machine_learning. Accessed: 2021-01-31.
- [20] Wikipedia pointer. [https://en.wikipedia.org/wiki/Pointer_\(computer_programming\)](https://en.wikipedia.org/wiki/Pointer_(computer_programming)). Accessed: 2021-01-31.
- [21] Wikipedia software development kit. https://en.wikipedia.org/wiki/Software_development_kit. Accessed: 2021-01-31.

8 Anhang

```
1 Parse.Cloud.define('getTechniques', async (request) => {
2     const contentCheck =
3         require('../helperFunctions/ContentCheck');
4
5     const userId = request.params.userId;
6     /**
7      * Queries whether the user is null or undefined
8      */
9     if (contentCheck.isEmpty(userId)) {
10         throw new Error("Error in getUserData:
11             A user id must exist to query UserData");
12     }
13
14     /**
15      * Creates a pointer to get all the attributes
16      * of the UserData class for the correct
17      * user in the database
18      */
19     const userObject = Parse.Object.extend("User");
20     const userPointer = userObject
21         .createWithoutData(userId);
22
23     /**
24      * Multiple Technique classes exists
25      * for each user. Therefore find();
26      */
27     return new Parse.Query('Technique')
28         .equalTo('user', userPointer)
29         .include('swimStyle')
30         .include('value')
31         .find({useMasterKey: true});
32 });
}
```

Codeausschnitt 14: Cloud-Funktion getTechniques

```

1 Parse.Cloud.define('getUserData', async (request) => {
2     const contentCheck =
3         require('../helperFunctions/ContentCheck');
4
5     const userId = request.params.userId;
6     /**
7      * Queries whether the user is null or undefined
8      */
9     if (contentCheck.isEmpty(userId)) {
10         throw new Error("Error in getUserData:
11             A user id must exist to query UserData");
12     }
13
14     /**
15      * Creates a pointer to get all the attributes
16      * of the UserData class for the correct
17      * user in the database
18      */
19     const userObject = Parse.Object.extend("User");
20     const userPointer = userObject.
21     createWithoutData(userId);
22
23     /**
24      * One UserData class exists for
25      * each user. Therefore first();
26      */
27     return new Parse.Query('UserData')
28         .equalTo('user', userPointer)
29         .include('aim')
30         .include('workoutDuration')
31         .include('prefSwimStyle')
32         .first({ useMasterKey: true });
33 });

```

Codeauschnitt 15: Cloud-Funktion getUserData