ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Implementierung einer Feature-rich Webplattform zur Unterstützung von Studien zur Stressreduzierung auf Basis des Mindful Walking Paradigmas

Masterarbeit an der Universität Ulm

Vorgelegt von:
Verena Pfaff
verena.pfaff@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Prof. Dr. Rüdiger Pryss

Betreuer:
Prof. Dr. Rüdiger Pryss

2020

Fassung 15. Oktober 2020

© 2020 Verena Pfaff

Kurzfassung

Immer häufiger werden im Gesundheitssektor mobile Anwendungen zur Gesundheitsvorsorge oder Behandlung von chronischen Krankheiten eingesetzt. Aufgrund ihrer ständigen Verfügbarkeit und Ausstattung mit diversen Sensoren eröffnen Smartphones Chancen für die Entwicklung von Anwendungen, mit deren Hilfe Nutzer gezielt bei der Reduzierung von Krankheitssymptomen unterstützt werden können.
Gerade im Zeitalter der Leistungsgesellschaft hat ständig auftretender Stress einen negativen Einfluss auf die Gesundheit der Menschen. Durch den Einsatz von Achtsamkeitsübungen kann das Stresslevel effektiv gesenkt und die damit einhergehenden Symptome abgeschwächt werden. Ein Beispiel für eine körperbezogene Achtsamkeitsübung stellt Mindful Walking dar, bei der Nutzer ihre Aufmerksamkeit vollständig auf das Gehen und die dazugehörigen Bewegungen lenken.
Durch den Einsatz von Smartphone-Apps ist es möglich, Nutzer adäquat anzuleiten, Achtsamkeitsübungen wie Mindful Walking korrekt durchzuführen. Darüber hinaus können Daten von Benutzern dieser Anwendungen gesammelt und von Forschern analysiert werden, um wertvolle Informationen über die Auswirkungen dieser Übungen auf die Gesundheit abzuleiten.

In der vorliegenden Arbeit wird eine Webplattform entwickelt, um Daten von Teilnehmern der Mindful Walking Studie des Instituts für Datenbanken und Informationssysteme der Universität Ulm, die mit einer bestehenden mobilen Anwendung gesammelt werden, zu verarbeiten. Dazu wird eine geeignete Schnittstelle und Webanwendung implementiert, um ausgefüllte Fragebögen und Übungsdaten von Nutzern dauerhaft zu speichern und für Forscher geeignet über das System auszuwerten bzw. in Form von Diagrammen zu visualisieren.

Inhaltsverzeichnis

1

Einleitung

Ständige Erreichbarkeit, Termindruck und Hektik - im Zeitalter der Leistungsgesellschaft ist Stress ein fester Bestandteil im Alltag zahlreicher Menschen in Deutschland. Dies lässt sich auch anhand von Zahlen belegen: Bei einer Studie der Techniker Krankenkasse gaben sechs von zehn Deutschen an, sich manchmal oder häufig gestresst zu fühlen, Tendenz steigend. Besonders die Generation der 30- bis 39-Jährigen ist regelmäßig von Stress betroffen. Dabei stellt der Beruf den Stressfaktor Nummer eins in Deutschland dar. Je nach Intensität und Dauer des Stresses erweist sich dieser als eine Gefahr für die physische und psychische Gesundheit. Krankenkassen konnten in den letzten 15 Jahren eine Zunahme von stressbedingten Krankschreibungen feststellen. Kopf- und Rückenschmerzen, Schlafstörungen, Tinnitus und Depressionen sind mögliche Beschwerden, die in Zusammenhang mit Stress gebracht werden können. Daher ist es in der heutigen Zeit umso wichtiger, Stress und auftretenden Symptomen gezielt entgegenzuwirken [1].

1.1 Motivation

Smartphones stellen aufgrund ihrer Eigenschaften eine bedeutende Chance für die Gesundheitsförderung und Prävention von Krankheiten dar. Mithilfe von Sensoren, die derzeit in mobilen Geräten eingebaut sind, können wertvolle Daten bezüglich der Gesundheit des Nutzers für Forschungszwecke gesammelt werden. Smartphones sind leicht im alltäglichen Leben einsetzbar und können die Anwender durch die Entwicklung geeigneter Applikationen gezielt bei der Stressreduzierung unterstützen [2, 3].

Eine solche mobile Anwendung wurde ihm Rahmen der *Mindful Walking* Studie von der Universität Ulm am Institut für Datenbanken und Informationssysteme in Zusammenarbeit mit der evangelischen Hochschule Nürnberg und der Donau-Universität Krems entwickelt. Sie unterstützt die Benutzer gezielt beim achtsamen Gehen, mit dessen Hilfe Stress und Depressionen gelindert werden sollen [3].

1.2 Zielsetzung

Zielsetzung der vorliegenden Arbeit ist die Entwicklung einer Webplattform zur Unterstützung von Studien zur Stressreduzierung auf Basis des Mindful Walking Paradigmas. Mithilfe der Plattform sollen die ausgefüllten Fragebögen und Übungsdaten von Teilnehmern der Mindful Walking Studie persistent gespeichert, verwaltet und visualisiert werden.
Die von der bestehenden mobilen Anwendung gesammelten Daten der Nutzer sollen dabei über eine Schnittstelle mit einer Webanwendung synchronisiert werden, um Forschern wertvolle Informationen bezüglich der Effektivität von Mindful Walking Übungen zu liefern. Darüber hinaus sollen Nutzer ihre Profildaten über die Plattform verwalten können und ein Feedback zum individuellen Studienverlauf erhalten. Daten und deren Auswertung sollen daher für Forscher und Nutzer mithilfe des Systems geeignet aufbereitet und in Diagrammen veranschaulicht werden.

1.3 Struktur der Arbeit

Die vorliegende Arbeit umfasst insgesamt acht Kapitel. Nach Einführung in die Motivation und Zielsetzung erfolgt in **Kapitel 2** die Erläuterung der fachlichen und technischen Grundlagen. Dabei wird näher auf die Mindful Walking Studie und verwandte Projekte eingegangen. Auch die verwendeten Technologien und Frameworks werden anschließend aufgeführt und beschrieben.
Kapitel 3 identifiziert die funktionalen und nicht-funktionalen Anforderungen an die zu implementierende Plattform und geht im Zuge dessen auch auf die verfügbaren Rollen

für Nutzer im System ein. Nachfolgend werden in **Kapitel 4** das Konzept und die Architektur des Systems vorgestellt. Hier soll der Aufbau und das Zusammenspiel der Client- und Serveranwendung ersichtlich werden.
Kapitel 5 beschäftigt sich mit der Implementierung ausgewählter Funktionen, die von der Anwendung bereitgestellt werden. Diese werden anhand von Code-Ausschnitten detailliert geschildert. In **Kapitel 6** wird schließlich die Webseite von Mindful Walking präsentiert, um die umgesetzten Anforderungen anhand der grafischen Benutzeroberfläche zu erläutern. Anschließend erfolgt in **Kapitel 7** ein Abgleich der funktionalen und nicht-funktionalen Anforderungen, die in Kapitel 3 beschrieben wurden. Zum Abschluss liefert **Kapitel 8** eine Zusammenfassung der vorliegenden Arbeit und der entstandenen Ergebnisse.

2

Grundlagen

In diesem Kapitel werden die fachlichen und technischen Grundlagen erläutert, die zum Verständnis der vorliegenden Arbeit erforderlich sind. Kapitel 2.1 vermittelt die Idee hinter Mindful Walking und der zugrundeliegenden Studie. Im Zuge dessen wird auf die bestehende mobile Anwendung eingegangen. Kapitel 2.2 stellt verwandte Arbeiten und deren Gemeinsamkeiten mit der zu entwickelnden Plattform vor. Zum Abschluss werden in Kapitel 2.3 die verwendeten Frameworks und Technologien, die zur Implementierung der Plattform ausgewählt wurden, beschrieben.

2.1 Mindful Walking

Mindful Walking basiert auf dem Prinzip der Achtsamkeit (engl. *Mindfulness*) [3, 4]. Daher soll dieser Begriff zunächst näher erläutert werden.
Achtsamkeit hat seinen Ursprung in Meditationsmethoden, wie sie primär im Buddhismus vorzufinden sind und ist eine besondere Form der Aufmerksamkeitslenkung. Genauer versteht man darunter die bewusste und nicht wertende Aufmerksamkeit auf Erlebnisse, die sich im gegenwärtigen Augenblick ereignen [3, 5].
Da es in der menschlichen Natur liegt, während alltäglichen Handlungen gedanklich abzuschweifen, ist es das Ziel von Achtsamkeitsübungen, das Bewusstsein in das Hier und Jetzt zurückzuholen, um sich wieder auf die aktuelle Tätigkeit zu fokussieren. Dabei sollen auftretende Bewusstseinsinhalte, wie beispielsweise Körperempfindungen oder Gedanken, nicht bewertet, sondern lediglich bewusst wahrgenommen werden [4, 5]. Viele Achtsamkeitsübungen nutzen dabei automatisierte körperbezogene Prozesse, wie Atmen oder Gehen und weisen den Übenden an, die Aufmerksamkeit absichtlich auf

diese Tätigkeiten zu richten [3].
Mindfulness hat nachweislich einen positiven Effekt auf die Gesundheit und wird zunehmend in Verhaltenstherapien integriert. Beispielsweise können Stress, Depressionen und Ängste durch Achtsamkeitsübungen gelindert werden [4, 6].
Bei Mindful Walking handelt es sich um eine körperbezogene Achtsamkeitsübung. Dabei soll der Übende sich des ansonsten automatisierten Gehens durch langsame und kleine Schritte bewusst werden. Bei diesem Ansatz wird nicht nur das Prinzip der Achtsamkeit angewandt, sondern auch die positive Wirkung der Bewegung auf den Körper. Da Anfänger oftmals Probleme haben, den Fokus während einer Übung aufrechtzuerhalten und achtsam zu bleiben, bietet die bestehende mobile Anwendung geeignete Feedback-Funktionen zur Unterstützung [3, 7].

2.1.1 Ablauf der Studie

In diesem Abschnitt soll der Ablauf der Mindful Walking Studie [8] anhand der bestehenden mobilen Anwendung vorgestellt werden. Um an der Studie teilnehmen zu können, müssen sich die Nutzer zunächst über die App registrieren, woraufhin der Nutzer zufällig in eine der zwei möglichen Gruppen eingeteilt wird: in die Interventions- oder Kontrollgruppe. Nach der Anmeldung ist es möglich, die Studie per Klick zu starten, um die nötigen Abläufe in Gang zu setzen (vgl. Abbildung 2.1).
Nutzer beider Gruppen müssen zunächst Fragebögen ausfüllen. Die Kontrollgruppe bekommt am fünften Tag nach Abgabe der Bögen eine Benachrichtigung, diese erneut auszufüllen. Danach ist die Studie beendet. Die Kontrollgruppe dient lediglich als Vergleichsgruppe, um die Effektivität von Mindful Walking Übungen überprüfen zu können. Im Unterschied dazu, werden die Teilnehmer der Interventionsgruppe nach erstmaliger Abgabe der Fragebögen für die Mindful Walking Übungen freigeschaltet. Die Übungen sollen fünf Tage lang einmal täglich von den Nutzern dieser Gruppe durchgeführt werden (vgl. Abbildung 2.1). Am fünften Tag nach Ausführung der letzten Übung, muss die Interventionsgruppe ebenfalls erneut die Fragebögen ausfüllen, woraufhin die Studie auch für diese Gruppe beendet ist.

Die Übungen stellen den relevantesten Teil der Studie dar und werden infolgedessen detailliert geschildert. Zunächst erhält der Teilnehmer über die App eine Beschreibung zur Übung, die im Freien für zehn Minuten durchgeführt werden soll.
Der genaue Ablauf einer Mindful Walking Übung ist dabei in vier Phasen unterteilt. In **Phase 1** wird dem Nutzer das Ziel des achtsamen Gehens erklärt. Wichtig dabei ist, dass der Übende seine Aufmerksamkeit vollständig auf das Gehen richtet. Er wird ebenfalls darauf hingewiesen, dass die App vibriert, sobald die Gehgeschwindigkeit zu hoch ist und angepasst werden muss. Die Zielgeschwindigkeit beläuft sich derzeit auf 4 km/h. Danach soll der Teilnehmer in **Phase 2** sein Befinden mithilfe eines sogenannten *Self-Assessment Manikin* Fragebogens (SAM) einschätzen. **Phase 3** stellt die Durchführung der 10-minütigen Übung dar. Dabei soll der Übende die Grundbewegungen des Gehens, wie beispielsweise das Anheben des Fußes, achtsam wahrnehmen. Den Abschluss der Übung kennzeichnet **Phase 4**. Hier muss der Teilnehmer wie in Phase 2 sein Befinden nach dem Gehen angeben. Zusätzlich soll ein Fragebogen zur Übung ausgefüllt werden.

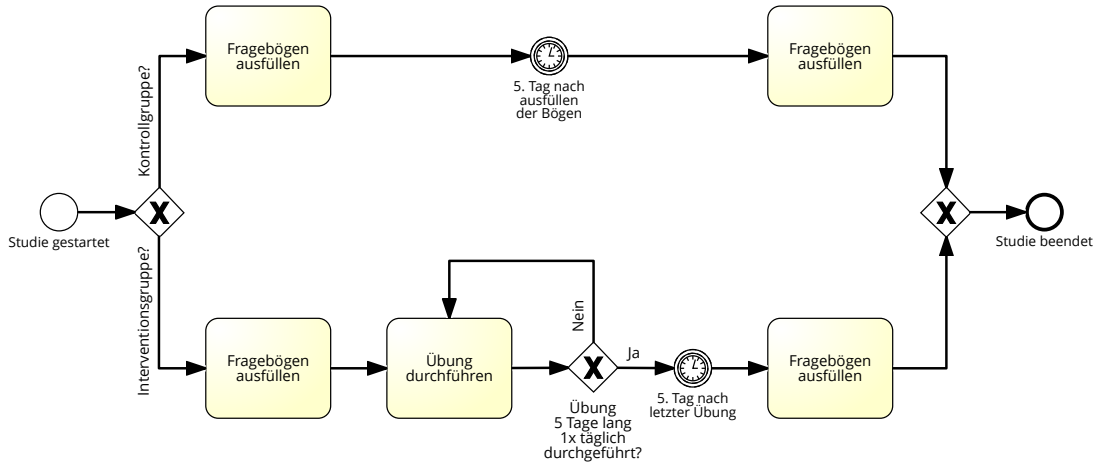

Abbildung 2.1: Ablauf der Mindful Walking Studie dargestellt als BPMN-Diagramm - Erstellt mit Signavio [9]

2.2 Verwandte Arbeiten

Dieses Kapitel stellt zwei weitere vergleichbare Projekte des Instituts für Datenbanken und Informationssysteme an der Universität Ulm vor.

2.2.1 Track Your Tinnitus

Tinnitus beschreibt ein Phänomen, bei dem Menschen ein Geräusch ohne dazugehörige akustische Reizquelle wahrnehmen. Jeder Mensch erlebt im Laufe seines Lebens einen Tinnitus, der jedoch nach wenigen Sekunden oder Minuten wieder verschwindet. Rund 10 % der Bevölkerung leiden hingegen an einer dauerhaften Form der schwer zu behandelnden Erkrankung [10, 11].
Für die Diagnose und Behandlung eines Tinnitus ist es notwendig, verschiedene Symptome wie Lautstärke und Variation der wahrgenommenen Geräusche, Stresspegel, depressive Verstimmungen und Ängste zu bewerten. Derzeit wird die Erkrankung bei Patienten mithilfe von Fragebögen, visuellen Analogskalen oder psychoakustischen Messungen beurteilt, die allerdings nicht die tägliche Variabilität der Lautstärke des Tinnitus oder den Stresspegel über einen Zeitraum erfassen. Meist werden die oben genannten Beurteilungen zu Hause oder in Kliniken vorgenommen, wodurch mögliche Umwelteinflüsse auf den Tinnitus nicht berücksichtigt werden können. Zusätzlich besteht das Problem, dass die auftretenden Symptome von Patienten retrospektiv beschrieben werden [10, 12].
Mithilfe der mobilen Crowdsensing-Plattform *TrackYourTinnitus* (TYT) können Schwankungen der Tinnituswahrnehmung und -belastung unter realen Bedingungen während des Tages gemessen werden [10]. Dazu stellt die Plattform eine Webseite, zwei mobile Anwendungen (iOS und Android) und eine relationale Datenbank zur Sammlung von Daten bereit. Nachdem Nutzer sich über die Webseite oder Smartphone-App registriert und die notwendigen Fragebögen ausgefüllt haben, können sie ihre tägliche Tinnituswahrnehmung und deren Schwankungen mithilfe der App über einen entsprechenden Fragebogen erfassen. Während dem Ausfüllen wird zusätzlich der Geräuschpegel der Umgebung gemessen, falls diese Option aktiviert wurde. Die Nutzer haben darüber

hinaus die Möglichkeit ihre Angaben zur Erkrankung auf der Webseite zu visualisieren. Die gesammelten Daten der Nutzer werden über eine REST-Schnittstelle (vgl. Kapitel 2.3.1) in der Datenbank gespeichert, die daraufhin von Forschern evaluiert werden können. So besteht die Möglichkeit, wertvolle Informationen über die Erkrankung aus den Patientendaten abzuleiten [10, 12].

2.2.2 Track Your Stress

Bei *TrackYourStress* (TYS) handelt es sich um eine mHealth-Crowdsensing-Plattform zur kontinuierlichen Erfassung des Stresslevels von registrierten Nutzern. Sie besteht wie *TrackYourTinnitus* aus einer Webseite, zwei mobilen Anwendungen (iOS und Android) und einer relationalen Datenbank zur Speicherung der Daten. Die Kommunikation zwischen den drei Komponenten erfolgt ebenfalls über eine REST-Schnittstelle (vgl. Kapitel 2.3.1) [13, 14].
Nach der Registrierung über die mobile Anwendung oder Webseite füllen Nutzer zunächst Fragebögen aus, die demographische Daten, die momentane Stresssituation und weitere stress-bezogene Parameter erfassen. Danach erfolgt die kontinuierliche Stressbewertung mithilfe von täglichen, wöchentlichen oder monatlichen Fragebögen. Während dem Ausfüllen werden nach dem Einverständnis des Nutzers die GPS-Position und der Geräuschpegel der Umgebung gemessen. Daraufhin werden die Daten übertragen und über die Schnittstelle in der Datenbank gespeichert. Auch hier ist es wieder möglich, wertvolle Informationen aus den gesammelten Daten abzuleiten [13, 14].

2.3 Verwendete Frameworks und Technologien

Im Fokus dieses Abschnitts stehen die verwendeten Frameworks und Technolgien, die für die Umsetzung der Mindful Walking Plattform verwendet wurden.

2.3.1 REST

REST steht für *Representational State Transfer* und beschreibt einen Architekturstil für die Kommunikation zwischen heterogenen verteilten Systemen im Internet. Der Begriff und dessen Konzepte wurden im Jahr 2000 von Roy Fielding in seiner Dissertation „*Architectural Styles and the Design of Network-based Software Architectures*“ [15] eingeführt und beschrieben [16, 17].
Schnittstellen (engl. ***A**pplication **P**rogramming **I**nterfaces*), die den REST-Architekturstil anwenden, werden als RESTful APIs bezeichnet [18]. REST nutzt vorhandene Prinzipen bzw. Protokolle des Webs und beruht unter anderem auf folgenden Bedingungen [17]:

Client-Server-Architektur: Sie birgt den Vorteil, dass Client und Server unabhängig voneinander entwickelt werden können. Dies steigert die Portabilität des Clients und verbessert die Skalierbarkeit, da der Server durch die Kapselung schlanker ausfällt [15, 19].

Zustandslosigkeit (engl. statelessness): Die Anfragen der Clients müssen alle erforderlichen Informationen enthalten, so dass der Server diese interpretieren und unabhängig voneinander bearbeiten kann. Der Zustand der Sitzung wird also beim Client gehalten, während der Server keinerlei Daten im Kontext der Anfragen speichert, also zustandslos agiert. Skalierbarkeit und Zuverlässigkeit der Anwendung nehmen dadurch zu [15, 17].

Caching: Durch den Einsatz von Caching wird die durchschnittliche Latenzzeit verbessert, da Interaktionen des Clients mit dem Server teilweise oder vollständig durch Zwischenspeichern der Daten eliminiert werden können. Damit ist es möglich, Effizienz und Skalierbarkeit des Systems zu erhöhen [15].

einheitliche Schnittstelle (engl. Uniform Interface): REST unterscheidet sich von anderen netzwerkbasierten Architekturstilen durch die Forderung nach einer einheitlichen Schnittstelle zwischen den Komponenten, die konsistent untereinander interagieren sollen. Dies sorgt für die Vereinfachung der Systemarchitektur [15, 20].

REST ist ressourcenzentriert, was bedeutet, dass alles, was von der Schnittstelle zugänglich gemacht wird, eine Ressource darstellt. Diese werden über einen eindeutigen *Uniform Resource Identifier* (URI) identifiziert bzw. adressiert. Jede vorhandene Ressource einer REST-Schnittstelle besitzt dabei mindestens eine Repräsentation. Im Web dominieren XML und JSON (***J**ava**S**cript **O**bject **N**otation*) als Datenaustausch-Format, die in erster Linie als Repräsentationen von Ressourcen verwendet werden [19, 21].
Für die Übertragung von Daten ist REST an kein bestimmtes Protokoll gebunden, jedoch wird in der Praxis ausschließlich HTTP als Transportprotokoll eingesetzt. Mithilfe der HTTP-Methoden `GET`, `POST`, `PUT` und `DELETE` ist es möglich, das geforderte *Uniform Interface* umzusetzen, da die gesamte Schnittstelle anhand dieser Basisoperationen entworfen wird. Durch die Kombination von URI und HTTP-Methode kann einheitlich auf die Repräsentation einer Ressource zugegriffen und Operationen, wie beispielsweise Löschen oder Anlegen, ausgeführt werden. Auf diese Weise kann der Client seinen Zustand wechseln [17, 19].
Listing 2.1 zeigt ein Beispiel für eine HTTP-GET-Anfrage, die an eine RESTful API gesendet wird, um den Nutzer mit der ID 1 abzufragen. Als Antwort erhält man die Daten im JSON-Format (vgl. Listing 2.2).

```
1 GET /users/1 HTTP/1.1
2 Host: localhost:3000
3 Accept: application/json
```

Listing 2.1: Beispiel für eine GET-Anfrage an eine RESTful API [22]

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
```

```
{
   "id": 1,
   "email": "user@example.org",
   "name": "Max",
   "age": 22,
   "links": [
       {
          "rel": "self",
          "uri": "/users/2"
       }
   ]
}
```

Listing 2.2: Antwort der GET-Anfrage [22]

Für die Einhaltung der einheitlichen Schnittstelle fordert Fielding nicht nur die Identifikation von Ressourcen und deren Manipulation durch Repräsentationen. Des Weiteren sollen die Nachrichten selbstbeschreibend sein, um von Client und Server vollständig interpretiert werden zu können. Über Metadaten innerhalb der Nachrichten können beispielsweise zusätzliche Informationen über den Ressourcenstatus, das Darstellungsformat (z.B. JSON) oder die Größe der Nachricht zwischen Client und Server vermittelt werden. Dies geschieht unter anderem über HTTP-Header [15, 17].
Die wohl kritischste Bedingung für REST und die Einhaltung der einheitlichen Schnittstelle stellt *Hypermedia as the Engine of Application State* (kurz HATEOAS) dar [17]. Damit wird gefordert, dass Repräsentationen von Ressourcen Links zu verwandten Ressourcen enthalten, mit denen die Nutzer intuitiv durch die Schnittstelle gelenkt werden. Listing 2.2 zeigt ein Beispiel für derartige Links. Der Nutzer weiß dadurch, dass ein User mit der ID 2 existiert, der über die angegebene URI abgefragt werden kann [19, 20].
Der Einsatz von HATEOAS findet in der Praxis wenig Beachtung, da vielmehr API-Dokumentationen eingesetzt werden, um die Schnittstelle und deren Nutzung zu beschreiben [19]. Aus diesem Grund wurde auf die Verwendung von Links innerhalb der implementierten Mindful Walking API verzichtet und eine ausführliche Dokumentation bevorzugt.

2.3.2 Angular

Angular [23] ist ein von Google entwickeltes clientseitiges Framework für die Erstellung von Single-Page-Applikationen [24].
Bei einer Single-Page-Applikation wird nur ein einziges HTML-Dokument und JavaScript-Code vom Server geladen. Jede Änderung der Ansicht durch Interaktion mit einem Nutzer wird im Browser gerendert. Aufgrund dessen sind diese Art von Anwendungen sehr performant, da es nicht nötig ist, bei jeder Änderung neue HTML-Seiten vom Server anzufordern [25, 26].
Bei Angular handelt sich um eine komplette Überarbeitung des beliebten Vorgängers AngularJS. Die erste Version von Angular wurde 2016 veröffentlicht und befindet sich aktuell in der Version 10, die im Juni 2020 erschienen ist [23, 27].
Für die Entwicklung von Angular-Anwendungen wird TypeScript genutzt. Dabei handelt es sich um eine von Microsoft entwickelte Programmiersprache, die den JavaScript-Sprachstandard um Konzepte wie beispielsweise Typsicherheit, Klassen oder Annotationen erweitert [28, 29].
Ein TypeScript-Feature, von dem Angular häufig gebraucht macht, sind *Decorators*. Sie erlauben es, Klassen um zusätzliche Informationen (Metadaten) zu erweitern. Dies nutzt Angular, um die Applikation lauffähig zu machen. Das Framework selbst wurde ebenfalls in TypeScript geschrieben. Vor der Auslieferung wird TypeScript in reines JavaScript umgewandelt, damit die Anwendung in jedem Browser lauffähig ist [24, 28, 29].

Für die vorliegende Arbeit wurde Angular in der Version 8 eingesetzt, die im Mai 2019 veröffentlicht wurde [23].

2.3.3 Bootstrap

Bootstrap [30] ist ein bekanntes CSS-Framework für die einfache Gestaltung responsiver Webseiten. Ursprünglich ist Bootstrap Mitte 2010 bei Twitter entstanden, wurde dann aber 2011 als Open-Source-Framework veröffentlicht. Bootstrap liefert zahlreiche vorgefertigte Komponenten wie Buttons, Dialoge oder Eingabefelder und ein responsives Grid-System, wodurch den Entwicklern viel Arbeit abgenommen wird. Das Grid-System

besteht aus zwölf Spalten und liefert vordefinierte Klassen, um Elemente innerhalb von Zeilen und Spalten platzieren zu können. Mithilfe von Breakpoints, die von Bootstrap definiert wurden, kann das Layout der Seite einfach an die Bildschirmgröße des Geräts angepasst werden [30, 31].
Es gibt zusätzlich eine Bibliothek namens *ng-bootstrap* [32], die mit Bootstrap gebaute Widgets für Angular bereitstellt. Dadurch wird die Nutzung des CSS-Frameworks innerhalb einer Angular-Anwendung erleichtert [24, 32].

Wegen der Popularität und der einfachen Einbindung in Angular wurde Bootstrap 4 für die Gestaltung der zu entwickelnden Webseite verwendet.

2.3.4 MariaDB

Bei MariaDB [33] handelt es sich um ein beliebtes relationales Datenbank-Managementsystem, das im Jahre 2009 erschienen ist. Es wurde von den ursprünglichen Entwicklern von MySQL entwickelt und ist als Open-Source-Software verfügbar [33, 34].
Bei relationalen Datenbanken werden die zu verarbeitenden Daten in Tabellen gespeichert, die in Beziehungen zueinander stehen. Die Manipulation erfolgt über SQL (***S**tructured **Q**uery **L**anguage*), der am häufigsten genutzten Sprache für relationale Datenbanksysteme [35, 36].

XAMPP

Bei XAMPP [37] handelt es sich um eine Apache-Distribution, die MariaDB, PHP und Perl beinhaltet [37]. Zusätzlich stellt das Open-Source-Paket weitere Werkzeuge wie die Datenbank-Administrationsanwendung *phpMyAdmin* [38] zur Verfügung, die über das Web genutzt wird [39].

Aufgrund der Beliebtheit von MariaDB und der Bereitstellung über XAMPP wurde diese Datenbank für die vorliegende Arbeit ausgesucht. Sie wird mithilfe von phpMyAdmin verwaltet.

2.3.5 Node.js

Node.js [40] ist eine Open-Source-Plattform zur Ausführung von JavaScript-Code auf dem Server, also außerhalb eines Browsers. Die Technologie wurde 2009 von Ryan Dahl ins Leben gerufen [41, 42].
Um eine serverseitige Laufzeitumgebung für JavaScript zu ermöglichen, nutzt Node.js die von Google entwickelte V8-JavaScript-Engine, die den Code zunächst in Maschinencode umwandelt und dann ausführt [42, 43].
Aufgrund der Schnelligkeit und Performanz von Node.js ist es möglich hoch skalierbare Server zu erstellen. Dies verdankt Node.js nicht nur der Nutzung der V8-JavaScript-Engine, sondern auch einer Technik namens *Event-Loop* [41, 44].
Der JavaScript-Code einer Node.js-Anwendung läuft auf einem einzigen Thread, wodurch nur eine Operation zu einem Zeitpunkt ausgeführt werden kann. Dieser Ansatz vereinfacht jedoch die Programmierung erheblich, da man sich nicht um die auftretenden Probleme bei Nebenläufigkeit kümmern muss. Mithilfe der Event-Loop können nicht-blockierende E/A-Operationen (z.B. Netzwerkanfragen oder Filesystemoperationen) ausgeführt werden, das heißt, dass die Anwendung nicht von langen Lese- oder Schreiboperationen blockiert wird. Eine eingehende Anfrage dieser Art wird daher an die Event-Loop weitergegeben und ein Callback registriert, der wiederum die Anfrage an das Betriebssystem weiterleitet. So kann Node.js die Ausführung der Anwendung fortsetzen. Ist die Operation der Anfrage beendet, tritt ein Event auf und die Event-Loop sorgt für die Ausführung der Callback-Funktion [42, 45].

Zur Organisation des Codes benutzt Node.js ein Modulsystem. Module werden über die Funktion `require` in ein File eingebunden (vgl. Listing 2.3) [43]. Dabei sind drei Arten von Modulen zu unterscheiden:

Built-in-Module: Diese Module werden mit Node.js selbst mitgeliefert und müssen nicht separat installiert werden. Ein Beispiel stellt das Dateisystemmodul *fs* dar, das den Zugriff auf das Dateisystem des Servers ermöglicht [42, 43].

Third-Party-Module: Dies sind Module, die über ein Package-Repository installiert werden. Mithilfe des *Node Package Managers* (npm) können Module für andere

Programmierer entwickelt und bereitgestellt oder in der eigenen Applikation genutzt werden [42, 43].

lokale Module: Dabei handelt es sich um Module, die innerhalb der eigenen Anwendung erstellt wurden. Über das `exports`-Objekt, das Node.js bereitstellt, kann die Funktionalität eines Moduls nach außen hin verfügbar gemacht werden [42, 43].

```
1  const http = require("http");
2
3  const hostname = "127.0.0.1";
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7     res.statusCode = 200;
8     res.setHeader("Content-Type", "text/plain");
9     res.end("Hello World");
10 });
11
12 server.listen(port, hostname, () => {
13    console.log("Server running");
14 });
```

Listing 2.3: Beispiel für einen Node.js-Webserver [40]

Express

Bei Express [46] handelt es sich um ein Web-Framework für Node.js, das als npm-Paket in der Anwendung installiert werden kann. Express ist im Kern ein Wrapper um das `http`-Modul von Node.js. Es nimmt Entwicklern viel Arbeit ab und liefert Unterstützung durch Features wie Routing, Error Handling oder das Rendern von Views [42, 46].

Sequelize

Sequelize [47] ist ein Node.js-ORM für relationale Datenbanken wie MariaDB oder MySQL. ORM steht für *Object-Relational Mapping* und bietet Entwicklern die Möglichkeit, Datenbankeinträge auf Objekte in objektorientierten Sprachen abzubilden. Dadurch können Entwickler auf Datenbankeinträge zugreifen, ohne sich um die Abfragedetails bei einem Datenbankzugriff kümmern zu müssen. Zusätzlich reduziert es den Overhead, der bei der Überbrückung von Anwendungslogik und Datenbank entsteht. Sequelize kann über npm in der Node.js-App installiert werden. Des Weiteren muss für die Verwendung des ORM ein entsprechender Driver für die eingesetzte Datenbank, in diesem Fall *mariadb*, heruntergeladen werden [47, 48].
Listing 2.4 zeigt, wie man Nutzer aus einer MariaDB-Datenbank mit Hilfe von Sequelize abfragen kann. Neben dem Verbindungsaufbau mit der Datenbank muss auch ein `Model` definiert werden, das alle Attribute beinhaltet, die auch in der entsprechenden Datenbanktabelle zu finden sind. Auf diese Weise können die Daten verarbeitet werden [47].

```
1  const Sequelize = require("sequelize");
2
3  // Verbindung mit der Datenbank aufbauen
4  const sequelize = new Sequelize("database", "username", "password", {
5     host: "localhost",
6     dialect: "mariadb"
7  });
8
9  // Modeldefinition eines Nutzers
10 const User = sequelize.define("user", {
11    // Attribute
12    firstName: {
13       type: Sequelize.STRING,
14       allowNull: false
15    },
16    lastName: {
17       type: Sequelize.STRING
```

```
18     }
19 }, {
20     // Optionen
21 });
22
23 // alle Nutzer abfragen
24 User.findAll().then(users => {
25     console.log("All users:", JSON.stringify(users, null, 4));
26 });
```

Listing 2.4: Beispiel für eine Datenbankabfrage mit Sequelize [47]

Da JavaScript durch Node.js nicht mehr nur für Client-, sondern auch für Serveranwendungen benutzt werden kann, erspart man sich das Erlernen einer weiteren Programmiersprache. Zusätzlich wird mithilfe von Sequelize die Arbeit mit der Datenbank erleichtert. Aus diesem Grund wurde Node.js mit Express als Framework und Sequelize als ORM für die vorliegende Arbeit verwendet.

2.3.6 Entwicklung und Testing

Git

Git [49] ist ein lizenzfreies verteiltes Versionsverwaltungssystem (engl. **V**ersion **C**ontrol **S**ystem) für die Aufzeichnung und Kontrolle von Änderungen an Softwareprojekten [49, 50]. Für das Mindful Walking Projekt wurde Git zusammen mit GitLab [51], einer Git- und webbasierten DevOps-Plattform, zur Speicherung und Verwaltung des Codes in einem Remote Repository verwendet [51].

Webstorm

Bei Webstorm [52] handelt es sich um eine JavaScript-IDE von JetBrains, die Entwicklern viele nützliche Features wie beispielsweise Codevervollständigung, Fehlererkennung in Echtzeit, VCS-Integration und Debugging liefert. Die Entwicklungsumgebung unterstützt

dabei zahlreiche JavaScript-Frameworks für Web und Server, wie Angular mit TypeScript und Node.js. Daher wurde Webstorm für die Entwicklung der Plattform ausgewählt [52].

Swagger

Die REST-Schnittstelle von Mindful Walking wurde mit Hilfe von Swagger [53], einer Suite von API-Entwicklertools, dokumentiert. Sie entstand 2010 und wurde 2015 von SmartBear Software übernommen. Die Suite bietet ein Tool namens *Swagger UI* an, mit dessen Hilfe Nutzer Ressourcen einer API visualisieren und mit ihnen interagieren können. Die Dokumentation wird automatisch anhand eines OpenAPI-Dokuments erstellt. Dabei handelt es sich um ein JSON-Objekt, das entweder in JSON- oder YAML-Format dargestellt werden kann. Das Dokument muss mit der OpenAPI-Spezifikation konform sein. Sie definiert eine standardisierte, sprachunabhängige Schnittstelle zu RESTful APIs, wodurch Nutzer ohne den Zugriff auf den Quellcode, die Schnittstelle verstehen können [53].
Um das OpenAPI-Dokument vom Server ausliefern und die Dokumentation im Browser darstellen zu können, wurde die Bibliothek *swagger-ui-express* [54] verwendet, die über npm heruntergeladen werden kann. Im Anhang A.2 ist ein Screenshot der Mindful Walking API Dokumentation, wie sie im Browser angezeigt wird, vorzufinden.

Postman

Bei Postman [55] handelt es sich um eine API-Entwicklungs-Plattform, die unter anderem einen Desktop-API-Client anbietet. Über den Client ist es möglich, REST-Anfragen zu senden und deren Antworten mit zusätzlichen Informationen wie HTTP-Status Code, Größe und Zeit zu inspizieren. Die Anfragen können darüber hinaus in einer Collection gespeichert, exportiert und mit anderen geteilt werden. Postman wurde zum Testen der Mindful Walking API verwendet, da Anfragen und die dazugehörigen Antworten mithilfe des Tools schnell und einfach überprüft werden können [55].

3

Anforderungsanalyse

Dieses Kapitel identifiziert und beschreibt die funktionalen und nicht-funktionalen Anforderungen an die zu implementierende Web- und Serveranwendung. Des Weiteren werden die unterschiedlichen Benutzerrollen, die im System verfügbar sein sollen, näher erläutert.

3.1 Rollen

Die zu entwickelnde Plattform soll zwei mögliche Rollen für Benutzer bereitstellen, die jeweils andere Funktionen und Rechte besitzen. Die genauen Anforderungen und Berechtigungen, die für die jeweilige Rolle gelten, können den Abschnitten 3.2.1 und 3.2.2 entnommen werden.

Admin: Die Rolle des `Admins` kann nur durch den Systemadministrator vergeben werden. Diese Accounts werden manuell in der Datenbank angelegt und können nicht direkt über die Webseite erstellt werden. Der `Admin` hat das Recht, die Daten der anderen Benutzer über die Webseite anzusehen und zu verwalten.

User: Die Rolle des `Users` wird dem Benutzer der Anwendung automatisch nach der Registrierung zugeteilt. Dabei handelt es sich um potenzielle Studienteilnehmer. Ein `User` hat grundsätzlich weniger Funktionen und Rechte als ein `Admin`. Dementsprechend besitzt er lediglich die Berechtigung, seine eigenen Daten zu verwalten. Es ist nicht vorgesehen, dass ein `User` durch den Systemadministrator auf die Rolle des `Admins` hochgestuft wird. Somit ist die Rolle des `Users` nicht änderbar.

Der Begriff `User` wird ab nun ausschließlich für die Bezeichnung der Rolle im System verwendet. Der Ausdruck *Nutzer* soll in diesem Zusammenhang für Benutzer der Webseite, also für die Rollen `User` und `Admin` stehen.

3.2 Funktionale Anforderungen

Funktionale Anforderungen spezifizieren die Funktionen, die unter festgelegten Bedingungen von einem Softwaresystem erfüllt werden sollen [56]. Diese Art von Anforderungen an die Mindful Walking Plattform sind nachfolgend aufgelistet.

3.2.1 Anforderungen an die Webanwendung

In diesem Abschnitt werden ausschließlich die zu implementierenden Funktionen der Webseite identifiziert und beschrieben.

FA1 An - und Abmeldung:
Nutzer sollen sich mit ihrer E-Mail-Adresse und ihrem Passwort auf der Webseite anmelden können. Erst danach sollen die bereitgestellten Funktionen verfügbar sein. Zusätzlich soll es die Möglichkeit geben, dass sich Nutzer von der Webseite wieder abmelden können.

FA2 Passwort ändern und zurücksetzen:
Nutzer sollen die Möglichkeit haben, ihr Passwort zurückzusetzen, falls sie dieses vergessen haben. Des Weiteren soll das Passwort innerhalb der Anwendung von Nutzern geändert werden können.

FA3 Profil bearbeiten:
Der `User` soll seine Kontodaten wie E-Mail-Adresse, Benutzername, Alter und Geschlecht einsehen und bearbeiten können.

FA4 Feedback:
Der `User` soll nach Beendigung der Studie ein Feedback in Form von Diagrammen

erhalten. Hier soll ersichtlich werden, inwiefern sich seine Angaben im Verlauf der Studie verbessert bzw. verschlechtert haben.

FA5 Informationen und Kontakt:

Dem `User` sollen Informationen über die Mindful Walking Studie bereitgestellt werden. Des Weiteren soll er die Möglichkeit haben, sich mit dem Administrator der Seite in Verbindung zu setzen. Hierfür sollen ihm alle notwendigen Kontaktdaten angezeigt werden.

FA6 Benachrichtigungen:

Der `Admin` soll Benachrichtigungen über neue Studienteilnehmer, ausgefüllte Fragebögen und Übungsabgaben erhalten. Diese sollen auf einem Dashboard angezeigt werden. Zusätzlich soll es möglich sein, Benachrichtigungen als gelesen zu markieren.

FA7 Fragebögen:

Eine Übersicht über alle verfügbaren Fragebögen zur Studie sollen dem `Admin` in Form einer Liste dargestellt werden. Er soll ebenfalls die Möglichkeit haben, einzelne Fragebögen zu bearbeiten. Name, Beschreibung und Antwort- bzw. Fragetext des Bogens sollen geändert werden können. Auch die Reihenfolge der Fragen soll veränderbar sein.

FA8 Nutzerdaten:

Der `Admin` soll die Möglichkeit haben, die Daten von angemeldeten Nutzern in Form eines Nutzerprofils ansehen zu können. Dabei soll er nicht nur deren Kontodaten, sondern auch ausgefüllte Fragebögen und Übungsdaten einsehen können. Zusätzlich soll er die Möglichkeit haben, einen Nutzer zu löschen.

FA9 Diagramme:

Dem `Admin` sollen Diagramme zu ausgewerteten Fragebögen und Übungsdaten der Nutzer bereitgestellt werden.

FA10 CSV Export:

Der `Admin` soll die Möglichkeit haben, die Daten der Nutzer im CSV-Format herunterladen zu können.

FA11 Statistiken:

Dem `Admin` sollen diverse Diagramme zur Anzahl der Nutzer nach Geschlecht bzw. Gruppe und Menge der Abgaben bereitgestellt werden.

3.2.2 Zusätzliche Anforderungen an die Serveranwendung

Die Serveranwendung muss nicht nur Funktionen für die Webseite bereitstellen, sondern auch Anforderungen umsetzen, die später von der mobilen Anwendung genutzt werden sollen. Diese sind nachfolgend aufgelistet.

FA12 Registrierung:

Die API soll die Möglichkeit bieten, dass sich `User` über die mobile Anwendung registrieren können. Dabei sollen Daten wie Benutzername, E-Mail-Adresse, Passwort, Alter und Geschlecht gespeichert werden.

FA13 E-Mail-Adresse verifizieren:

Die API soll nach der Registrierung einen Verifizierungslink an den `User` senden, damit dieser seine E-Mail-Adresse bestätigen kann. Nur nach Anklicken des Links und erfolgreicher Verifizierung soll die Anmeldung bei der App oder Webseite möglich sein.

FA14 ausgefüllte Fragebögen und Übungsabgaben:

Die Schnittstelle soll ausgefüllte Fragebögen und Übungsabgaben der einzelnen `User`, die über die mobile Anwendung gesendet werden, empfangen und persistent speichern.

FA15 Studie starten:

Hat ein `User` die Mindful Walking Studie über die mobile Anwendung gestartet, soll diese Aktion über die API gespeichert werden, so dass die nötigen Funktionen zum Ablauf der Studie bereitgestellt werden können.

FA16 Aufgaben:

Über die Schnittstelle soll es für den `User` möglich sein, Aufgaben abzufragen. Bei den Aufgaben handelt es sich um offene Fragebögen.

3.3 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen machen Aussagen über die geforderte Qualität des Softwareprodukts [57]. Das zu entwickelnde System soll ebenfalls qualitative Eigenschaften aufweisen, die im Folgenden zusammengefasst werden (vgl. [57, 58, Qualitätsmodell aus ISO 25010]).

NFA1 Benutzbarkeit:
Die Webanwendung soll benutzerfreundlich und intuitiv bedienbar sein. Der Nutzer soll ohne Probleme einzelne Funktionen der Webseite verstehen und durchführen können.

NFA2 Zuverlässigkeit:
Das System soll stets den Erwartungen der Nutzer entsprechend zuverlässig funktionieren. Fehler sollen zu keinem ungewöhnlichen Verhalten führen und als verständliche Fehlermeldung angezeigt werden.

NFA3 Sicherheit:
Die Serveranwendung und die gespeicherten Daten im System sollen vor unbefugten Zugriffen geschützt werden.

NFA4 Wartbarkeit:
Die Anwendung soll erweiterbar und aktualisierbar sein. Neue Funktionen oder Komponenten sollen leicht ausgetauscht oder hinzugefügt werden können.

NFA5 Effizienz:
Das System soll dem Nutzer bei Anfragen in akzeptabler Zeit antworten und dafür sorgen, dass keine langen Wartezeiten beim Laden der Anwendung oder der Daten entstehen.

NFA6 Responsiveness:
Die Webseite soll sich den unterschiedlichen Bildschirmgrößen auf Tablets und Smartphones automatisch anpassen und korrekt angezeigt werden.

4

Konzept und Architektur

In diesem Kapitel wird das Konzept und die Architektur der Mindful Walking Plattform erläutert. Der erste Abschnitt 4.1 beschreibt dabei den Aufbau des Systems in seiner Gesamtheit und geht detailliert auf die Kommunikation zwischen Client und Server ein. Im Fokus von Kapitel 4.2 steht die Architektur der Webanwendung und deren Bestandteile. Im Anschluss wird in Kapitel 4.3 der Aufbau der Serveranwendung mit der REST-Schnittstelle und der Datenbank vermittelt.

4.1 Gesamtübersicht

In Abbildung 4.1 wird der grobe Aufbau des Systems und die Zusammenarbeit der einzelnen Komponenten im Ganzen aufgezeigt. Das Zentrum der Architektur stellt die API dar. Sie verwendet den REST-Architekturstil und fungiert als Bindeglied zwischen Client und Server. Die Kommunikation und der Datenaustausch erfolgen ausnahmslos über diese Schnittstelle mittels HTTP. Als Datenaustauschformat wurde JSON gewählt, da es im Vergleich zur Alternative XML kompakter und leichter lesbar ist. Zusätzlich kann es innerhalb der Anwendung sofort als JavaScript-Objekt genutzt werden, was die Verarbeitung auf Client- und Serverseite erleichtert, da die Web- und Serveranwendung mit JavaScript umgesetzt wurden [59].
Der Vorteil der Client-Server-Architektur mit einer RESTful API stellt zum einen die Unabhängigkeit von Client und Server und zum anderen das einfache Kommunikationsmuster mittels HTTP dar. Die API erhält und sendet lediglich Daten im JSON-Format. So können mehrere unterschiedliche Clients unabhängig voneinander mit der API kommunizieren, in diesem Fall also die Webanwendung und die mobile Applikation (vgl. Kapitel 2.3.1).

Die Schnittstelle synchronisiert auf diese Weise die Daten zwischen den beiden Clients. Gibt ein Nutzer beispielsweise über die mobile Anwendung einen Fragebogen ab, wird dieser von der Webseite abgefragt und unmittelbar angezeigt.

Die Webanwendung wurde mit Angular umgesetzt. Sie wird über den Browser aufgerufen und liefert die Benutzeroberfläche, mit der die Nutzer interagieren können.
Die Serveranwendung wurde mithilfe von Node.js und Express implementiert. Über die API werden innerhalb der Anwendung die Daten verarbeitet und in der relationalen MariaDB-Datenbank persistent gespeichert.

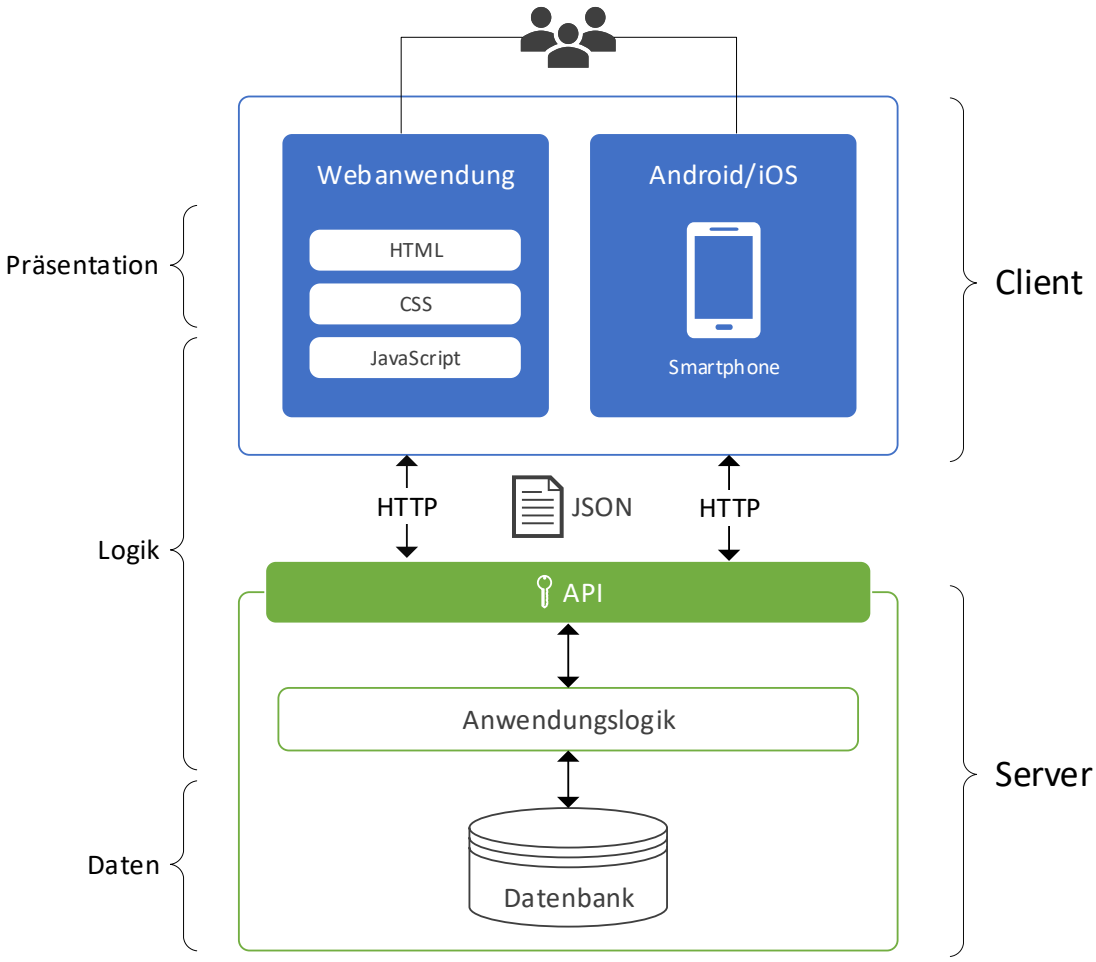

Abbildung 4.1: Gesamtarchitektur des Systems - Erstellt mit Microsoft Visio [60]

4.1.1 Authentifizierung

Der Zugriff auf die Daten im System soll nur nach Bestätigung der Identität des Nutzers möglich sein. Da eine RESTful API zustandslos ist, müssen die Sitzungsdaten im Client gespeichert und zum Server geschickt werden. Ein Weg dieses Problem zu lösen, sind *JSON Web Tokens* (JWT) [61].
Bei JSON Web Token [62] handelt es sich um einen Standard, der eine Methode zur sicheren Übertragung von Informationen zwischen zwei Parteien als JSON Objekt definiert. Die Informationen sind vertrauenswürdig, da sie digital signiert sind und verifiziert werden können. Die Tokens bestehen aus drei Teilen, die jeweils durch einen Punkt getrennt sind: Den `Header`, den `Payload` und die `Signatur`. Abbildung 4.2 zeigt ein JWT-Token, das von der Mindful Walking API generiert wurde. Der `Header` enthält typischerweise zwei Angaben. Zum einen den Typ des Tokens und zum anderen den verwendeten Signaturalgorithmus. Dieses JSON-Objekt wird daraufhin Base64Url codiert und bildet den ersten Teil des Tokens [62, 63].

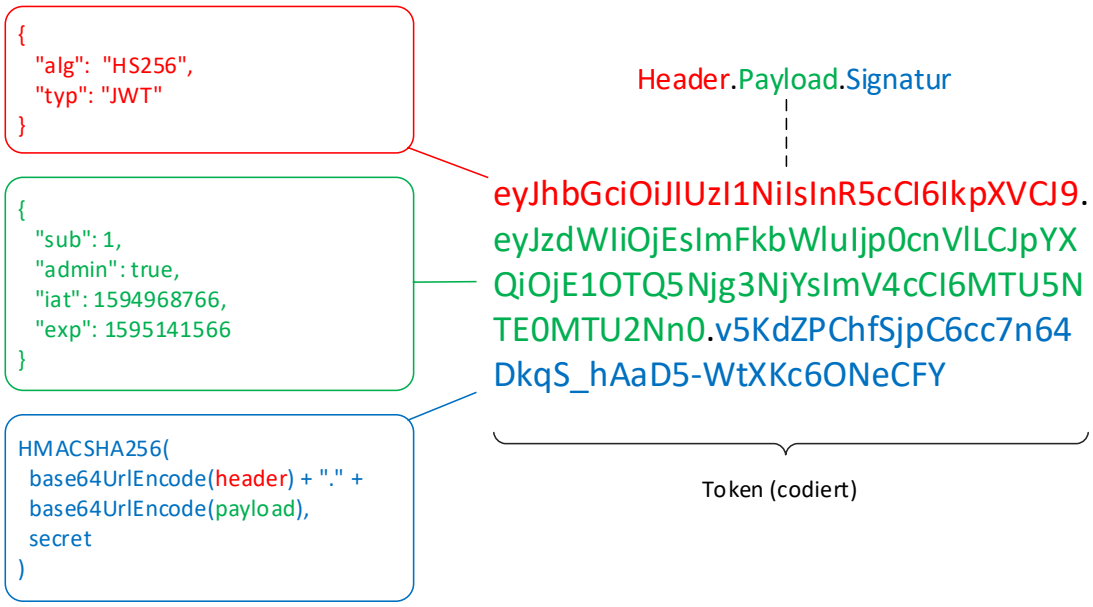

Abbildung 4.2: Aufbau eines JSON Web Tokens [62] - Erstellt mit Microsoft Visio [60]

Der zweite Teil besteht aus dem `Payload`. Er enthält die Claims, also Aussagen über eine Entität und zusätzliche Daten wie zum Beispiel die Ablaufzeit des Tokens. Bei der Entität handelt es sich typischerweise um den Benutzer, der sich über das Token authentifizieren möchte. Auch der `Payload` wird wieder Base64Url codiert. Die `Signatur` wird mithilfe des codierten `Headers`, dem codierten `Payload` und einem *Secret* signiert. Dazu wird der im Header angegebene Algorithmus verwendet, also in diesem Fall der HMAC SHA256-Algorithmus. Auch der letzte Teil des Tokens wird erneut Base64Url codiert [62, 63].
Mithilfe der `Signatur` kann das generierte Token verifiziert werden. Das bedeutet, dass überprüft werden kann, ob es auf dem Weg verändert worden ist und die Daten vom richtigen Absender kommen. Da der Inhalt eines Tokens eingesehen werden kann, sollte es keine vertraulichen Informationen beinhalten [62, 63].

Abbildung 4.3 veranschaulicht den Ablauf der Authentifizierung eines Nutzers über die Mindful Walking REST-Schnittstelle. Zunächst wird ein Request mit Passwort und E-Mail an die entsprechende URL der REST-Schnittstelle gesendet. Der Server sucht daraufhin in der Datenbank einen Nutzer mit dieser E-Mail-Adresse, um sein Passwort mit dem gesendeten Passwort vergleichen zu können. Gibt es einen solchen Nutzer und stimmen die Passwörter überein, wird ein JWT-Token generiert. Aus Sicherheitsgründen werden Passwörter mit einer Hashfunktion verschlüsselt. Für Node.js gibt es eine Bibliothek namens *bcrypt* [64], mit deren Hilfe Passwörter gehasht und verglichen werden können. Das Token wird daraufhin an die Clientanwendung gesendet, die es im Local Storage des Browsers speichert. Um auf die Daten des Systems zugreifen zu können, muss der Nutzer sich bei jedem weiteren Request mithilfe des Tokens authentifizieren. Dazu wird das Token an jeden Request gehängt und an den Server geschickt. Dies geschieht über den `Authorization-Header`. Das Token wird vom Server geprüft und anschließend eine entsprechende Antwort auf die Anfrage gesendet.
Für die Verifizierung und Verwaltung der Tokens gibt es ein geeignetes Node.js-Package namens *jsonwebtoken* [65], das in der vorliegenden Arbeit eingesetzt wurde. Mithilfe von JavaScript kann das Token auf Clientseite im Local Storage des Browsers gespeichert werden. Um Informationen, die im Token enthalten sind, zu decodieren, wurde die Bibliothek *jwt-decode* [66] verwendet. Auf diese Weise ist es möglich, clientseitig zu

überprüfen, ob es sich bei dem Nutzer um einen `Admin` handelt oder nicht.

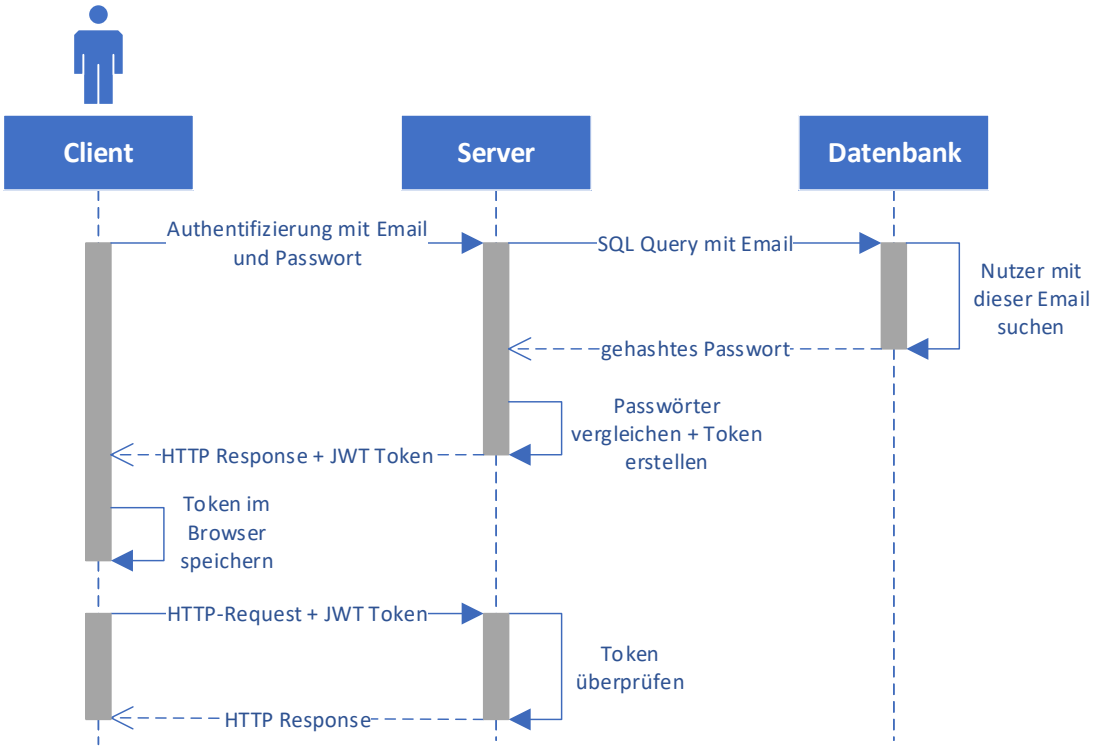

Abbildung 4.3: Authentifizierung mit JSON Web Token, basierend auf [67] - Erstellt mit Microsoft Visio [60]

4.1.2 Datendarstellung und -verarbeitung

Im Zentrum der Anwendung stehen die Fragebögen und Übungen, deren Daten zwischen Client und Server zur Anzeige, Bearbeitung und Auswertung ausgetauscht werden. In den folgenden Abschnitten wird die Darstellung der Daten und deren Verarbeitung erläutert.

Fragebögen

Alle Fragebögen des Systems besitzen die gleiche, fest definierte Struktur. Sie bestehen aus der Bezeichnung des Fragebogens, seiner Abkürzung, einer optionalen Anleitung und aus einer beliebigen Anzahl an Fragen und dazugehörigen Antwortmöglichkeiten. Bei den Fragen handelt es sich um Single-Choice-Fragen, das heißt, es kann nur eine von mehreren Antworten ausgewählt werden. Jede Frage eines Fragebogens hat dabei die gleichen Antwortmöglichkeiten. Im Anhang A.1 befindet sich ein Beispiel für einen Fragebogen (*Five-Facet Mindfulness Questionnaire*), der in der Studie verwendet wird. Er besteht aus 15 Fragen und 5 Antwortmöglichkeiten.
Listing 4.1 veranschaulicht anhand eines Beispiels, wie Fragebögen als JSON-Objekt zwischen Client und Server ausgetauscht werden. Jeder Fragebogen wird dabei über eine eindeutige `id` identifiziert. Diese wird mitgeschickt, damit der Client weitere Operationen über andere Endpunkte durchführen kann. Auch die Fragen und Antworten werden fragebogenübergreifend durch eine eindeutige `id` gekennzeichnet. Füllt ein User beispielsweise den Fragebogen über die Benutzeroberfläche aus, kann der ausgefüllte Fragebogen und die einzelnen Fragen mit der vom `User` ausgewählten Antwort identifiziert und später vom Server verarbeitet werden.

```
1  {
2     "data": {
3        "id": 1,
4        "name": "Fragebogen",
5        "instructions": "Anleitung",
6        "questions": [
7           {
8              "id": 1,
9              "questionText": "Frage eins",
10             "initialPosition": 1,
11             "position": 1
12          },
13          {
14             "id": 2,
```

```
15              "questionText": "Frage zwei",
16              "initialPosition": 2,
17              "position": 2
18          }
19      ],
20      "answers": [
21          {
22              "id": 1,
23              "answerText": "Antwort eins",
24              "position": 1
25          },
26          {
27              "id": 2,
28              "answerText": "Antwort zwei",
29              "position": 2
30          }
31      ]
32    }
33 }
```

Listing 4.1: Darstellung eines Fragebogens als JSON-Objekt

Fragebögen sollen aktualisiert und die Reihenfolge der Fragen verändert werden können. Über den Wert `position` wird die Anordnung der Fragen innerhalb eines Bogens angegeben. Je nach Änderung wird jeder `position`-Wert angepasst und die Fragen in dieser Reihenfolge an den Client gesendet. Der Wert `initialPosition` ist für die Auswertung des Fragebogens notwendig, da über bestimmte Fragen eines Bogens Skalen gebildet werden und dazu deren Reihenfolge in ihrem initialen Zustand benötigt wird. Für den Five-Facet Mindfulness Questionnaire gibt es fünf Skalen, die berechnet werden müssen (vgl. Anhang A.1, Scoring Information).
Jeder Fragebogen hat Antwortmöglichkeiten, denen ein Wert von 1 bis n oder 0 bis n, je nach Anzahl der Antworten, zugewiesen ist. Dies wird über den Wert `position` ausgedrückt. Er gibt dabei nicht nur die Reihenfolge der Antworten des Bogens an, sondern auch den Wert, der für die Berechnung der Fragebogenauswertung verwendet

wird. Um die Speicherung der Antworten in der Datenbank einheitlich zu gestalten, wurden nur Werte von 1 bis n für `position` vergeben. Bei der Berechnung müssen also Fragebögen, bei denen Werte von 0 bis n für die Antworten vergeben wurden, um eins korrigiert werden. Die Berechnung findet in der Angular-Anwendung statt. Der Server liefert dabei alle nötigen Angaben als JSON-Objekt. Für die Auswertung eines Fragebogens wird die Summe der Antwortwerte gebildet, die der Nutzer ausgewählt hat. Diese werden daraufhin in einem Diagramm dargestellt. Auf die genaue Implementierung der Fragebogenauswertung wird in Kapitel 5 eingegangen.

Übungen

Die Übungen werden wie die Fragebögen über die mobile Anwendung abgegeben, woraufhin die API ein JSON-Objekt mit den Übungsdaten verarbeitet. Listing 4.2 zeigt ein Beispiel für eine solche Übungsabgabe. Sie besteht aus verschiedenen Werten, die während der Übung gemessen wurden, wie beispielsweise die Geschwindigkeitsabweichung (`paceDeviation`), die angibt, wie oft der Nutzer die angegebene Zielgeschwindigkeit von 4 km/h überschritten hat.
Vor und nach der Übung füllen Studienteilnehmer Fragebögen aus (vgl. Kapitel 2.1.1), die mit den anderen Übungsdaten über die App verschickt und vom Server in der Datenbank gespeichert werden. Da der Self-Assessment Manikin Fragebogen zwei mal ausgefüllt wird, muss zur Unterscheidung der Abgaben über den Wert `metadata` angegeben werden, ob es sich um den ausgefüllten Fragebogen vor oder nach der Übung handelt. Eine Fragebogenabgabe besteht aus der `questionnaireId` und den `userAnswers`, also den Antworten, die der Nutzer für jede Frage eingegeben hat. Diese werden über die Kombination `questionId` und `answerId` identifiziert. Fragebogenabgaben, die nicht zu einer Übung gehören, werden ebenfalls in diesem Format übertragen.

```
1 {
2    "startedAt": "2020-10-03T09:31:00.403Z",
3    "endedAt": "2020-10-03T09:42:17.403Z",
4    "targetPace": 4.0,
5    "averagePace": 3.9,
```

```
  "paceDeviation": 2,
  "distance": 0.8,
  "heartRate": 86,
  "caloriesBurned": 123,
  "userSubmissions": [
     {
        "questionnaireId": 5,
        "metadata": "before",
        "userAnswers": [
           {
              "questionId": 45,
              "answerId": 23
           },
           {
              "questionId": 46,
              "answerId": 27
           }
        ]
     },
     {
        "questionnaireId": 5,
        "metadata": "after",
        "userAnswers": [
           {
              "questionId": 45,
              "answerId": 24
           },
           {
              "questionId": 46,
              "answerId": 28
           }
        ]
     },
     {
        "questionnaireId": 6,
```

```
41          "userAnswers": [
42              {
43                  "questionId": 47,
44                  "answerId": 32
45              },
46              {
47                  "questionId": 48,
48                  "answerId": 33
49              },
50              {
51                  "questionId": 49,
52                  "answerId": 32
53              },
54              {
55                  "questionId": 50,
56                  "answerId": 35
57              },
58              {
59                  "questionId": 51,
60                  "answerId": 34
61              },
62              {
63                  "questionId": 52,
64                  "answerId": 36
65              }
66          ]
67      }
68  ]
69 }
```

Listing 4.2: Darstellung einer Übungsabgabe als JSON-Objekt

4.2 Aufbau der Web-Applikation

In diesem Abschnitt wird der Aufbau der Webanwendung erläutert. Dabei werden zuerst die einzelnen Bestandteile einer Angular-Anwendung und deren Interaktion beschrieben. Im Anschluss wird der Aufbau der Mindful Walking Webanwendung anhand eines Architekturausschnitts veranschaulicht.

4.2.1 Bestandteile einer Angular-Anwendung

Jede Angular-Anwendung setzt sich aus sogenannten `Components` zusammen, die jeweils einen Teil der Anwendung darstellen. So kann es sich bei einer `Component` beispielsweise um eine ganze HTML-Seite oder nur ein kleines UI-Element handeln. Jede Angular-Anwendung besitzt mindestens eine `Component`, namens `AppComponent`, die den Startpunkt der Applikation darstellt. Ihr sind alle weiteren Komponenten untergeordnet [28, 29].
Jede `Component` besteht aus einer TypeScript-Klasse, die Anwendungsdaten und Logik enthält und einem dazugehörigen HTML-Template. Das Template ist der Bestandteil einer Komponente, den die Nutzer im Browser angezeigt bekommen. Die TypeScript-Klasse wird mit einem `Component-Decorator` versehen, dem die Metadaten für die Komponente übergeben werden (vgl. Listing 4.3). Hier wird das Template der `Component` festgelegt und ein CSS-Selektor angegeben. Es ist ebenfalls möglich ein oder mehrere CSS-Stylesheets mit der Komponente zu verbinden. Über den Selektor kann eine `Component` in andere Templates eingebunden werden. Angular fügt dabei überall dort eine Instanz der Komponente ein, wo es den entsprechenden CSS-Selektor findet. Bei dem Selektor handelt es sich typischerweise um einen HTML-Tag mit dem Namen der Komponente. Die in Listing 4.3 definierte `Component` wird beispielsweise über `<my-component></my-component>` in andere Templates eingebunden. Auf diese Weise können Komponenten verschachtelt werden, wodurch ein sogenannter Komponentenbaum entsteht [24, 29].

```
1 @Component({
2     selector:    "my-component",
3     templateUrl: "./my-component.component.html",
4     styleUrls: ["./my-component.component.css"]
5 })
6 export class MyComponent {
7     // Code
8 }
```

Listing 4.3: Beispiel für eine `Component` in Angular [24]

Eine Komponente und ihr dazugehöriges Template können miteinander kommunizieren. Dies geschieht über sogenanntes *Data Binding*. So ist es möglich, dass Daten aus der Komponente im Template angezeigt oder Ereignisse im Template von der Komponente verarbeitet werden können. Listing 4.4 zeigt, wie Template und `Component` in Angular Daten austauschen können [24, 29].

Um Werte, wie beispielsweise eine Komponenteneigenschaft, im HTML-Template anzuzeigen, wird der Name der Eigenschaft mittels *Interpolation* gebunden. Dies geschieht mithilfe von zwei geschweiften Klammern (vgl. Listing 4.4) [24, 29].

Bei *Property Binding* werden Eigenschaften eines DOM-Elements durch Übermittlung von Daten aus der Komponente gesetzt. Hier werden eckige Klammern verwendet. In Listing 4.4 wird beispielsweise das `src`-Property auf den Wert der Komponenteneigenschaft `imageUrl` gesetzt. Zusätzlich ist es möglich, Daten mithilfe von Property Binding in eine Komponente hinein zu übertragen. So können Eltern-Komponenten mit Kind-Komponenten kommunizieren [29].

Tritt ein Ereignis im Template einer Komponente ein, kann darauf mithilfe von *Event Binding* reagiert werden. Dabei kann es sich um Ereignisse eines DOM-Elements handeln wie beispielsweise einen Klick auf einen Button. Listing 4.4 zeigt, wie ein `click`-Event über die Methode `onClickMe()`, die in der TypeScript-Klasse definiert wurde, abgefangen wird [29].

Two-way Binding kombiniert Property und Event Binding in einer einzigen Notation. Dies wird häufig bei der Formularverarbeitung eingesetzt, da sich die Daten oft in beide

Richtungen aktualisieren sollen (vgl. Listing 4.4) [29].

```
1  <!-- Interpolation, Component -> DOM -->
2  <h1>{{ title }}</h1>
3
4  <!-- Property Binding, Component -> DOM -->
5  <img [src]="imageUrl">
6
7  <!-- Event Binding, DOM -> Component -->
8  <button (click)="onClickMe()">Click me!</button>
9
10 <!-- Two-way Binding, Component <-> DOM -->
11 <input [(ngModel)]="title">
```

Listing 4.4: Beispiele für Data Binding in Angular [24]

Ein weiterer wichtiger Bestandteil einer Angular-Anwendung stellen `Directives` dar. Direktiven sind Klassen mit einem `Directive-Decorator`, die mithilfe eines CSS-Selektors an ein DOM-Element gebunden werden. Sie steuern daraufhin das Verhalten des Elements. Komponenten stellen ebenfalls `Directives` dar, die allerdings ein eigenes Template besitzen. Angular bringt bereits einige `Directives` mit, die out of the box verwendet werden können. Dabei sind zwei weitere Arten von Direktiven zu unterscheiden: Attributdirektiven und Strukturdirektiven. Das Aussehen bzw. Verhalten eines DOM-Elements wird über Attributdirektiven, die wie normale HTML-Attribute wirken, geändert. Strukturdirektiven verändern das Layout durch Hinzufügen, Entfernen und Ersetzen von Elementen im DOM. Im Gegensatz zu Attributdirektiven werden Strukturdirektiven mit einem Stern-Zeichen versehen. Angular bietet zwei wichtige Strukturdirektiven an: `*ngIf` und `*ngFor`. Durch `*ngIf` kann gesteuert werden, ob ein DOM-Element, je nach Auswertung eines Ausdrucks, angezeigt wird oder nicht. Mithilfe von `*ngFor` kann ein Array durchlaufen werden, um für jedes Element des Arrays ein DOM-Element zu erzeugen. Listing 4.5 veranschaulicht die Verwendung dieser Direktiven [24, 29].

```
<!--
  Fuer jeden Namen im Array wird ein Listenelement angezeigt
-->
<li *ngFor="let name of names">{{name}}</li>

<!-- Dieser Paragraph wird immer angezeigt -->
<div *ngIf="true">Lorem ipsum</div>

<!-- Dieser Paragraph wird nie angezeigt -->
<div *ngIf="false">Lorem ipsum</div>
```

Listing 4.5: Strukturdirektiven in Angular [29]

Daten und Logik, die nicht mit einer spezifischen Ansicht verknüpft sind und von mehrere Komponenten gemeinsam genutzt werden, können in Angular in sogenannten `Service`-Klassen gekapselt werden. Um eine Klasse als `Service` zu definieren, verwendet man den `Injectable-Decorator`, um die Metadaten bereitzustellen, die es Angular ermöglichen, den `Service` als Abhängigkeit in eine Komponente zu injizieren. In `Services` finden typischerweise die HTTP-Requests zur Abfrage der Daten von einer API statt. Angular bietet dafür den `HttpClient-Service` an, der alle notwendigen Methoden bereitstellt. In Angular können zusätzlich `Interceptors` deklariert werden, die jedes HTTP-Request untersuchen und transformieren. Dies ist beispielsweise für die Authentifizierung nützlich, da im `Interceptor` das JWT-Token für jeden Request angehängt werden kann. Ohne einen entsprechenden `Interceptor`, müsste man dies für jeden einzelnen `HttpClient`-Methodenaufruf implementieren [24, 28, 29].
Bei der Interaktion mit der Anwendung muss es möglich sein, zwischen den verschieden Ansichten zu wechseln, die definiert wurden. Dieses Verhalten wird mithilfe des Angular `Routers` umgesetzt. Für jede definierte `Route`, die ein URL-Pfad darstellt, wird eine Komponente angegeben, die gerendert werden soll. Mit sogenannten `Guards` ist es möglich, `Routes` zu schützen. So kann man beispielsweise Nutzer daran hindern, Teile der Anwendung aufzurufen, wenn diese nicht dafür autorisiert sind (vgl. Listing 4.6) [24].

```
const routes: Routes = [
   { path: "first-component", component: FirstComponent },
   {
      path: "second-component",
      component: SecondComponent,
      canActivate: [AuthGuard]
   },
];
```

Listing 4.6: Definition von `Routes` in Angular [24]

Alle `Components`, `Services` und `Directives` einer Angular-Anwendung sind in Modulen organisiert. Jede Anwendung hat dabei mindestens ein Modul, namens `AppModule`, das als Einstiegspunkt dient. Hier werden auch die `Routes`, die in einem separaten Modul definiert wurden, importiert, um das Routing innerhalb der Anwendung zu aktivieren. Mit `AppModule` wird das Bootstrapping der Anwendung angestoßen. Es rendert dabei die erste `Component`, injiziert die `Services` und bereitet alle erforderlichen Abhängigkeiten vor [29, 68].

4.2.2 Architekturausschnitt der Angular-Anwendung

Abbildung 4.4 zeigt einen Ausschnitt der umgesetzten Webanwendung und verdeutlicht deren Aufbau. Die `index.html`-Datei bildet den Einstieg jeder Angular-Anwendung beim Aufruf im Browser. Über das `AppModule` werden, wie bereits erwähnt, alle Teile der Anwendung, wie Komponenten, Routing und Service-Klassen, gebündelt [29].
Eine zentrale Komponente der Anwendung stellt die `NavigationComponent` dar. Hinter ihr verbirgt sich eine Side- und Navigationbar. Sie dient als Wrapper für alle Komponenten, die durch einen `AuthGuard` geschützt sind, also nur angezeigt werden, wenn der Nutzer authentifiziert ist. Über die Side- und Navigationbar ist es möglich innerhalb der Anwendung zu navigieren. Manche Komponenten bzw. Ansichten, die über die `NavigationComponent` erreichbar sind, sollen nur für den Admin zugänglich sein und werden daher durch den `AdminGuard` geschützt.

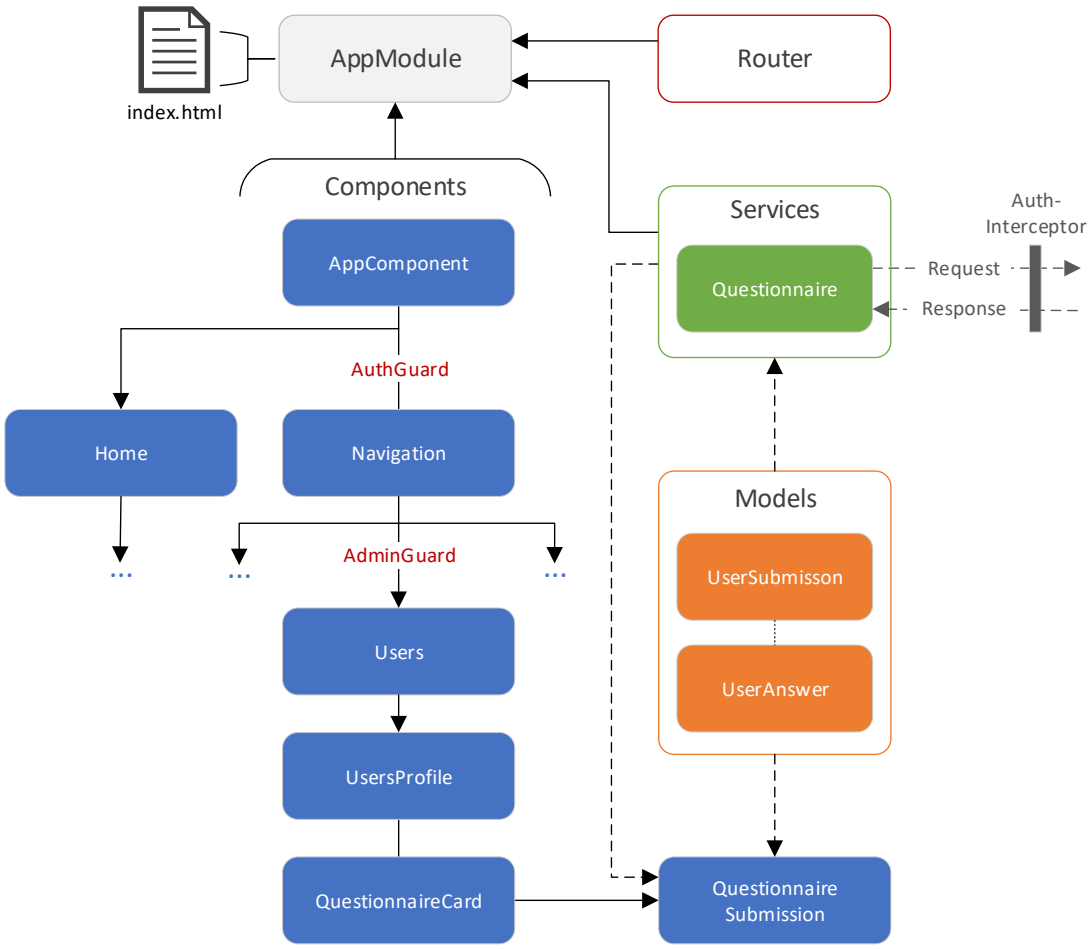

Abbildung 4.4: Auszug aus dem Aufbau der Webanwendung, basierend auf [68] - Erstellt mit Microsoft Visio [60]

Ein Beispiel dafür ist die `UsersComponent`. Hier werden die verfügbaren `User` im System aufgelistet. Mithilfe von Routing kann über die `UsersComponent` die `UsersProfileComponent` aufgerufen werden. Sie zeigt das entsprechende Profil eines `Users` an. Hier wird unter anderem auch die `QuestionnaireCardComponent` eingebunden, die alle abgegebenen Fragebögen eines Users in einer Liste anzeigt. Über sie kann zu dem jeweiligen ausgefüllten Fragebogen navigiert werden.
Die `QuestionnaireSubmissionComponent` beinhaltet das Template und die Logik Abgaben anzuzeigen. Diese Komponente bedient sich weiterer Klassen. Zum einen wird hier der `QuestionnaireService` injiziert. Er beinhaltet die HTTP-

Requests, um die Abgabedaten von der API zu erfragen. Diese gehen durch den `AuthInterceptor`, der das JWT-Token einfügt. Die Daten werden dann in der Komponente verarbeitet und gerendert. Zusätzlich nutzen der `QuestionnaireService` und die `QuestionnaireSubmissionComponent` zwei `Model`-Klassen, die als Blaupause für die Abgabedaten dienen und von der API geliefert werden. In Kapitel 5 und 6 wird näher auf die dazugehörige Implementierung und grafische Benutzeroberfläche eingegangen.

4.3 Aufbau der Serveranwendung

In diesem Abschnitt wird auf den Aufbau der Serveranwendung mit Node.js bzw. dem Express-Framework und der MariaDB-Datenbank eingegangen. Zuerst werden die Bestandteile einer Express-Anwendung und deren Zusammenspiel näher beleuchtet bevor im nächsten Abschnitt die Architektur der implementierten REST-Schnittstelle beschrieben wird. Den Abschluss bildet die Darstellung und Beschreibung des Datenmodells.

4.3.1 Bestandteile einer Express-Anwendung

Jede Express-Anwendung basiert auf sogenannten *Middlewares*. Dabei handelt es sich um Funktionen, die Zugriff auf das verfügbare `Request`-Objekt, `Response`-Objekt und die `next`-Funktion haben. Das `Request`-Objekt repräsentiert die HTTP-Anfrage. Es besitzt zahlreiche Properties, über die beispielsweise der Body oder Parameter der eingehenden Anfrage verarbeitet werden können. Das `Response`-Objekt stellt die HTTP-Antwort auf eine Anfrage dar. Die `next`-Funktion ist eine Funktion im Express-`Router`, die für den Aufruf der nächsten Middleware zuständig ist. Falls die aktuelle Middleware den Anfrage-Antwort-Zyklus nicht beendet, also keine Antwort auf eine Anfrage gesendet wurde, ist es nötig, explizit mit `next()` die nächste Middleware aufzurufen, da ansonsten die Anfrage still steht. So können Middleware-Funktionsaufrufe verkettet werden [46, 69, 70].
Die Kommunikation mit einer RESTful API findet über URIs in Kombination mit einer HTTP-Methode statt. Dies wird in Express mithilfe von Routing umgesetzt. Express

liefert ein `app`-Objekt, dessen Methoden den HTTP-Methoden entsprechen. Sie erhalten als Argumente den Route-Pfad und ein oder mehrere Handler-Funktionen (Middlewares), die ausgeführt werden, wenn diese Route von einem Client in Kombination mit dieser HTTP-Methode angefragt wird. Listing 4.7 zeigt ein Beispiel für eine Expressanwendung. Hier kann ein GET-Request an den Pfad `/greetings` gesendet werden, um den Text „Hello World“ auszugeben [46, 69].

```
1  const express = require("express");
2  const app = express();
3  const port = 3000;
4
5  app.get("/greetings", (req, res) => {
6     res.send("Hello World!")
7  });
8
9  app.listen(port, () => {
10    console.log("Server running!")
11 });
```

Listing 4.7: Beispiel für eine Expressanwendung [46]

Damit es möglich ist, Routing in unterschiedliche Dateien auszulagern, kann die Express-`Router`-Klasse verwendet werden. Eine `Router`-Instanz ist eine Middleware, das heißt sie kann als Argument in `app.get()` oder den anderen Methoden übergeben werden. Das `Router`-Objekt liefert ebenfalls HTTP-Methoden, die als Argumente Pfad und Handler-Funktionen erhalten [46, 70].

4.3.2 Architektur der Express-Anwendung

Abbildung 4.5 veranschaulicht den Aufbau der implementierten Serveranwendung. Einstiegspunkt der API stellt die Datei `app.js` dar, über die der Server gestartet wird. In dieser Datei werden auch die einzelnen `Routes` der Anwendung eingebunden, die in separaten Dateien, je nach Aufgabengebiet, ausgelagert wurden.

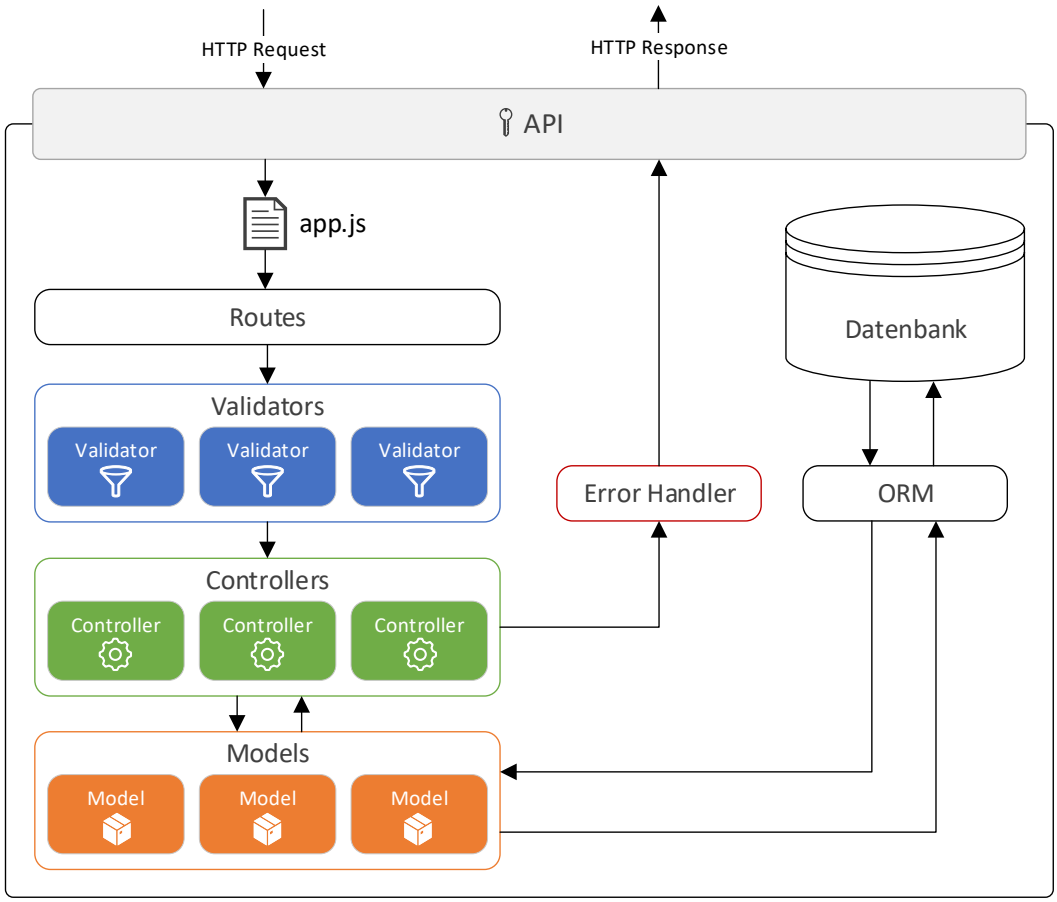

Abbildung 4.5: Aufbau der Serveranwendung, basierend auf [70] - Erstellt mit Microsoft Visio [60]

Die Eingaben der Nutzer müssen serverseitig validiert werden. Für Express existiert eine Bibliothek namens *express-validator* [71], die eine Menge von Middlewares anbietet, mit denen die Überprüfung der übergebenen Daten umgesetzt werden

kann. Die `Validators` werden als Middleware zu den Routes hinzugefügt. Listing 4.8 zeigt ein Beispiel für die Verwendung der Bibliothek. Hier wird die Email und das Passwort aus dem Body der Anfrage durch die mitgelieferten Methoden `isEmail()` und `isLength()` validiert [70, 71].

```
1  const express = require("express");
2  const {body} = require("express-validator");
3
4  const router = express.Router();
5
6  router.post("/register", [
7     body("email")
8        .isEmail(),
9     body("password")
10       .isLength({min: 8})
11 ], (req, res, next) => {
12    /* Anwendungslogik zur Registierung */
13 });
```

Listing 4.8: Beispiel für die Validierung von Nutzereingaben mit *express-validator* [71]

Die Anwendungslogik hinter den definierten `Routes` wurde in `Controller`-Files ausgelagert. Hier findet die Kommunikation mit der Datenbank über Sequelize statt. Die `Models`, die über das ORM definiert wurden (vgl.Kapitel 2.3.5), bilden die Datenbanktabellen ab. Sie sind separate Dateien, die von den `Controller`-Files importiert werden, um die Datenbankabfragen durchführen zu können.
Je nach Ergebnis der Anfragen werden in den `Controller`-Funktionen unterschiedliche Antworten mit unterschiedlichen HTTP-Status-Codes geschickt. Um auftretende Fehler abfangen und behandeln zu können, wurde eine Error-Handling-Middleware eingesetzt. Sie sendet den passenden HTTP-Status-Code mit der entsprechenden Fehlermeldung.

4.3.3 Datenmodell

Abbildung 4.6 zeigt einen Ausschnitt aus dem Datenbankmodell, der die Speicherung von Fragebögen, Übungen und Abgaben verdeutlicht. Da Fragebögen aus 1 bis n Fragen und Antworten bestehen (1:n-Beziehung), wurden dafür jeweils eigene Tabellen angelegt. Somit setzt sich ein Fragebogen insgesamt aus den Tabellen `questionnaires`, `questions` und `answers` zusammen.
Ausgefüllte Fragebögen sind immer einem Nutzer und einem Fragebogen zugeordnet. Dies spiegelt sich in der Tabelle `user_submissions` wider. Da für die Übungen ebenfalls Fragebögen beantwortet werden, besteht eine Beziehung zwischen den Tabellen `user_submissions` und `exercises`. Gehört eine Fragebogenabgabe nicht zu einer Übung, wird die `exercise_id` in der `user_submissions`-Tabelle auf `null` gesetzt. Fragebögen, die nach Starten der Studie und deren Abschluss ausgefüllt werden, sind zusätzlich einer Aufgabe zugeordnet. Für die Darstellung der Beziehung wird bei diesen Abgaben die entsprechende `task_id` in der `user_submissions`-Tabelle gesetzt.
Die Antworten einer Fragebogenabgabe werden in einer separaten Tabelle namens `user_answers` abgespeichert, die in Relation zu `user_submissions` steht. Hier werden die Antworten mithilfe der entsprechenden `question_id` und `answer_id` identifiziert.

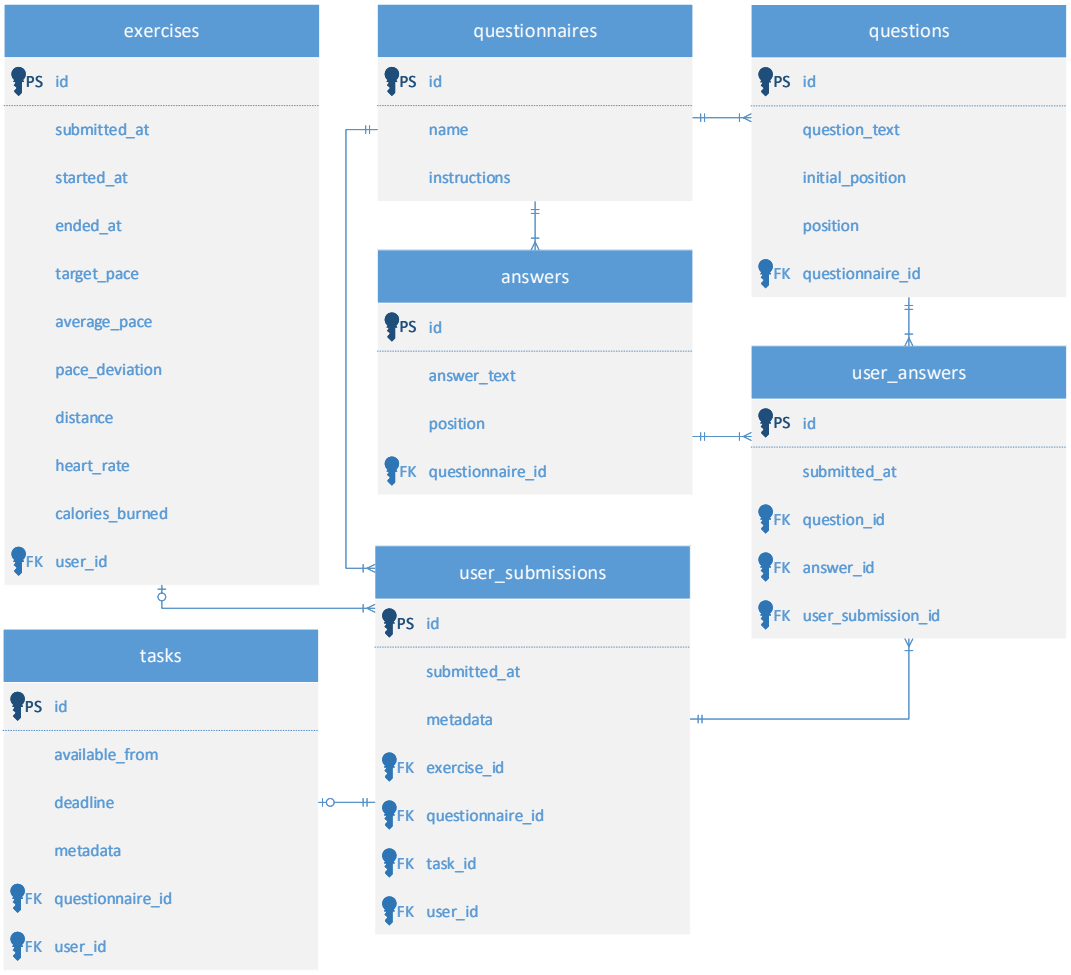

Abbildung 4.6: Ausschnitt aus dem Datenbankmodell - Erstellt mit Microsoft Visio [60]

5

Ausgewählte Aspekte der Implementierung

In diesem Kapitel wird zunächst eine Übersicht zur Umsetzung der Studie gegeben, um deren Implementierung aufzuzeigen. Danach erfolgt eine detaillierte Erläuterung ausgewählter Funktionen des Systems anhand von Code-Fragmenten.

5.1 Übersicht zur Umsetzung der Studie

Abbildung 5.1 veranschaulicht die Umsetzung der Studie und gibt einen Anhaltspunkt dafür, wie deren Ablauf aus Kapitel 2.1.1 im System abgebildet und implementiert wird. Aus Gründen der Übersichtlichkeit wurde lediglich der Studienverlauf von Teilnehmern der Interventionsgruppe dargestellt. Bei Personen, die der Kontrollgruppe zugeordnet wurden, bleibt der Übungsteil aus (siehe Abbildung 5.1).
Für die korrekte Umsetzung ist es nötig, bei den entsprechenden Endpunkten der REST-Schnittstelle zu prüfen, ob zu diesem Zeitpunk mit den gegebenen Daten und Voraussetzungen eine Aktion des `Users` durchgeführt wird oder nicht. Dies wird beispielsweise bei der Abgabe von Abschlussfragebögen deutlich: Wurde die Abgabefrist erreicht, können diese Fragebögen nicht mehr im System gespeichert werden. Für diesen `User` ist die Studie dementsprechend beendet und das Feedback freigegeben. Anderenfalls muss nach jeder Abgabe überprüft werden, ob alle Abschlussfragebögen gesendet und gespeichert wurden. Falls ja, wird die Studie daraufhin ebenfalls beendet und das Feedback freigegeben. Ansonsten ist es für den `User` weiterhin möglich, bis zur Deadline die Abschlussfragebögen auszufüllen. Wie auch bei den

Anfangsfragebögen wird jedes Mal kontrolliert, ob der Fragebogen bereits abgegeben wurde oder nicht, um doppelte Einträge zu vermeiden (vgl. Abbildung 5.1). Dies wurde im Diagramm aus Gründen der Übersichtlichkeit für diese Art von Fragebögen weggelassen.

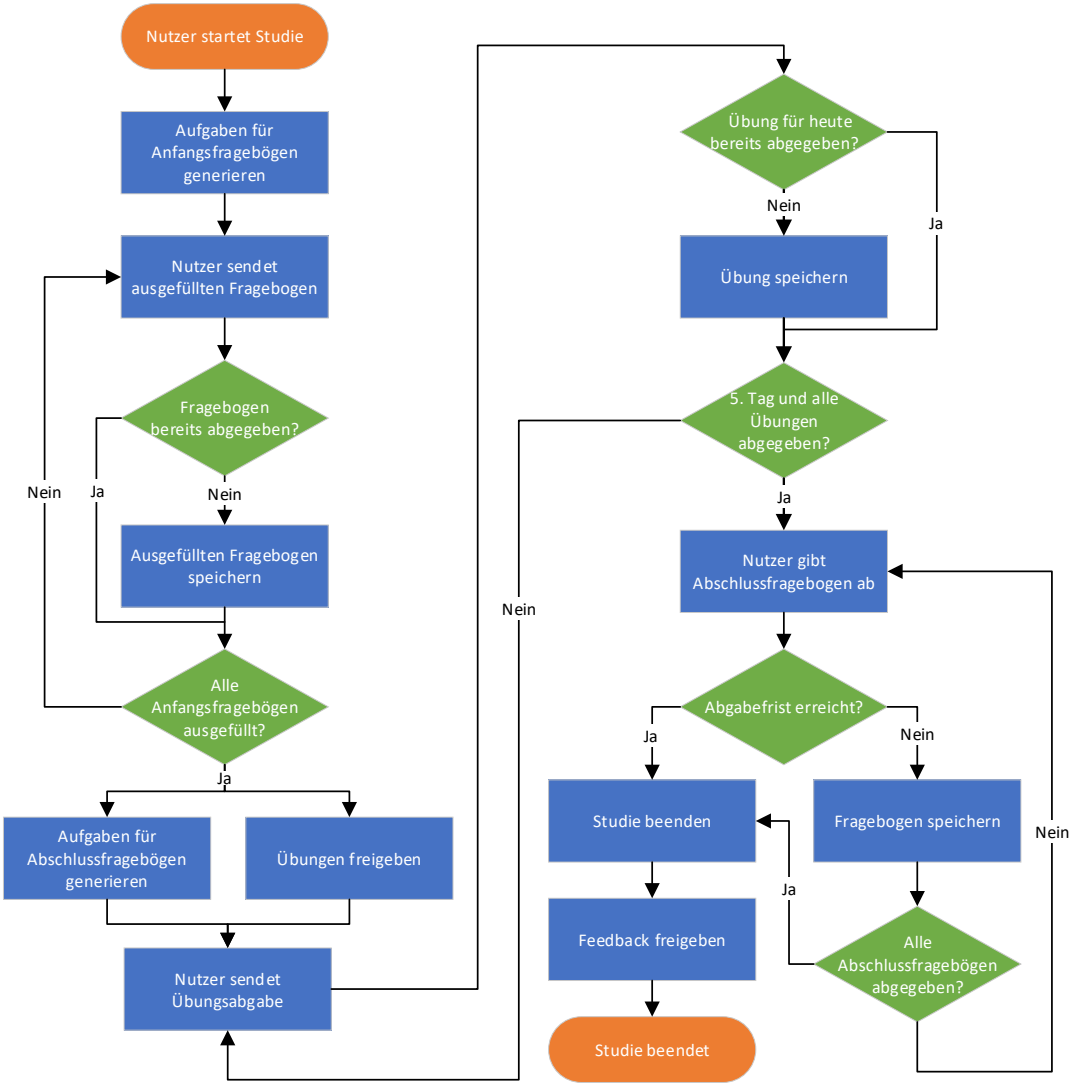

Abbildung 5.1: Übersicht zur Umsetzung der Studie als Flussdiagramm - Erstellt mit Microsoft Visio [60]

5.2 Anzeige von ausgefüllten Fragebögen

Dieser Abschnitt erklärt die Umsetzung von ausgefüllten Fragebögen im Client und Server. Dazu wird zunächst auf die clientseitige Implementierung zur Anzeige von Abgaben im Browser eingegangen. Anschließend erfolgt die Beschreibung der serverseitigen Implementierung, um die Verarbeitung und Anfrage von Daten an der REST-Schnittstelle zu verdeutlichen.

5.2.1 Clientseitige Implementierung

Die ausgefüllten Fragebögen eines Nutzers sollen lediglich dem `Admin` auf der Webseite angezeigt werden. Dazu ist es nötig, zunächst die Daten von der REST-Schnittstelle abzufragen. HTTP-Requests wurden in der Anwendung, aufgrund ihrer Verwendung über mehrere Klassen hinweg, in `Services` ausgelagert. Für Fragebögen und Abgaben wird der `QuestionnaireService` verwendet, der die Methode `getUserSubmissionDetail()` bereitstellt. Sie liefert die abgegebenen Fragebögen anhand von `userId` und `submissionId` (siehe Listing 5.1). Für die Anzeige benötigt die Anwendung nicht nur die Antworten der `User`, sondern auch den Aufbau des jeweiligen Fragebogens (vgl. Kapitel 4.1.2).

```
1  @Component({
2     selector: "app-questionnaire-submission",
3     templateUrl: "./questionnaire-submission.component.html",
4     styleUrls: ["./questionnaire-submission.component.scss"]
5  })
6  export class QuestionnaireSubmissionComponent implements OnInit {
7
8     userSubmission: UserSubmission;
9     userId: number;
10    submissionId: number;
11    loading = false;
12    alertOpen = false;
13    alertMessage = null;
```

```
14
15    constructor(private route: ActivatedRoute,
16                private questionnaireService: QuestionnaireService) {}
17
18    ngOnInit() {
19       this.route.paramMap.subscribe((params: Params) => {
20          this.userId = +params.get("id");
21          this.submissionId = +params.get("subId");
22          this.loading = true;
23          // Abfrage von Fragebogenabgaben
24          this.questionnaireService.getUserSubmissionDetail(this.
                userId, this.submissionId).subscribe((submission:
                UserSubmission) => {
25             this.userSubmission = submission;
26             this.loading = false;
27          }, error => {
28             this.alertOpen = true;
29             this.alertMessage = error.error.message;
30          });
31       });
32    }
33
34    getUserAnswerByQuestionId(id: number) {
35       for (const userAnswer of this.userSubmission.userAnswers) {
36          if (userAnswer.questionId === id) {
37             return userAnswer.answerId;
38          }
39       }
40    }
41 }
```

Listing 5.1: TypeScript-Code zur Anzeige von ausgefüllten Fragebögen

Im nächsten Schritt müssen die Daten in dem zugehörigen Template der Komponente angezeigt werden. Dazu wurden die Werte aus der `Component` mithilfe von Interpolation

gebunden. Der Name und die Instruktionen des Fragebogens können direkt aus dem `userSubmission`-Objekt verwendet werden (siehe Listing 5.2).
Für die Anzeige der Fragen und Antworten ist es nötig, mithilfe von `*ngFor` durch die einzelnen Elemente von `userSubmission.questionnaire.questions` bzw. `userSubmission.questionnaire.answers` zu iterieren.
Die Antworten werden über Radio-Buttons in Kombination mit dem `answerText` angezeigt. Je nachdem welche Antwort der `User` angegeben hat, soll nur dieser Radio-Button markiert sein. Gelöst wird dies durch das HTML-Attribut `checked`, das mithilfe von Property Binding in Kombination mit einem Ausdruck den Wert `true` oder `false` übermittelt bekommt. Bei jeder Frage über die iteriert wird, findet ebenfalls eine Iteration über die Antworten des Fragebogens statt. Dabei muss die `answer.id` der aktuellen Antwort mit der `answerId` der entsprechenden Antwort aus `userSubmission.userAnswers` übereinstimmen, die zur aktuellen Frage gehört. Dazu wird die Methode `getUserAnswerByQuestionId()` der Komponenten-Klasse verwendet. Ihr wird die `question.id` der Frage aus der aktuellen Iteration übergeben. Sie iteriert daraufhin über die `userSubmission.userAnswers` und sucht die Antwort zur der Frage, die mit der übergebenen übereinstimmt. Sie liefert danach die zugehörige `answerId`. Vergleicht man nun die `answer.id` der gegenwärtigen Iteration von `submission.questionnaire.answers` mit diesem Wert, wird nur die Antwort markiert, die der Nutzer angegeben hat.

```
1  <div class="card border-0">
2    <div class="card-header py-3 mx-0 border-bottom-0">
3      <div class="mb-3 font-weight-bold">
4        {{ userSubmission.questionnaire.name }}
5      </div>
6      <p class="mb-0">
7        {{ userSubmission.questionnaire.instructions }}
8      </p>
9    </div>
10   <div class="card-body px-0 py-0">
```

```
11        <div *ngFor="let question of userSubmission.questionnaire.
              questions">
12            <div class="question-container border-bottom border-top">
13                <p class="mb-0">
14                    {{ question.position }}.
15                    {{ question.questionText }}
16                </p>
17            </div>
18            <div class="container-fluid answer-container">
19                <div *ngFor="let answer of userSubmission.questionnaire.
                      answers">
20                    <input class="mr-2" type="radio"
21                        [checked]="answer.id === getUserAnswerByQuestionId(
                              question.id)"
22                        (click)="$event.preventDefault()">
23                    <label>{{ answer.answerText }}</label>
24                </div>
25            </div>
26        </div>
27    </div>
28 </div>
```

Listing 5.2: HTML zur Anzeige von ausgefüllten Fragebögen

5.2.2 Serverseitige Implementierung

Über die Route `/api/v1/users/:userId/submissions/:submissionId` können einzelne ausgefüllte Fragebögen von Nutzern abgefragt werden (siehe Listing 5.3). Bei `:userId` und `:submissionId` handelt es sich um dynamische Werte, über die Abgaben eindeutig identifiziert werden können. Alle Routes, die mit dem Pfad `/api/v1/users` beginnen, wurden in einem `user.routes.js`-File ausgelagert. Zusätzlich war es aus Gründen der Übersichtlichkeit nötig, die Route-Handlers ebenfalls in separaten Dateien zu kapseln.

Mithilfe der `checkAuth`-Middleware wird die Identität des Nutzers anhand des mitgelieferten JWT-Tokens überprüft. Danach erfolgt der Aufruf von `checkAdmin`, um zu verifizieren, dass es sich bei dem anfragenden Nutzer um einen `Admin` handelt. Ansonsten wird ein Fehler zurückgeliefert.
Die eigentliche Logik findet in der `user.controller.js`-Datei statt. In der Middleware-Funktion `getUserSubmissionById` wird mithilfe von Sequelize die entsprechende Abgabe von der Datenbank abgefragt. Das ORM bietet dabei alle nötigen Konfigurationsmöglichkeiten an, um zum einen die gewünschten Attribute und zum anderen die in Beziehung stehenden `Models` in die Anfrage miteinzubeziehen. Dadurch können auf Anhieb alle Daten geliefert werden. Bei erfolgreichem Request werden alle Daten mit dem HTTP-Statuscode `200` im JSON-Format an den Client gesendet. Treten Fehler auf, werden diese entsprechend abgefangen und an die Error-Middleware mit `next(err)` weitergeleitet.

```
1  // aus user.routes.js
2  const express = require("express");
3
4  const checkAuth = require("../middleware/check-auth");
5  const checkAdmin = require("../middleware/check-admin");
6  const userController = require("../controllers/user.controller");
7
8  const router = express.Router();
9
10 router.get("/:id/submissions/:subId", checkAuth, checkAdmin,
       userController.getUserSubmissionById);
11
12 module.exports = router;
13
14 // aus user.controller.js
15 exports.getUserSubmissionById = (req, res, next) => {
16    UserSubmission.findOne({
17       where: {
18          id: req.params.subId,
```

```
19          userId: req.params.id,
20          exerciseId: null
21       },
22       attributes: ["id", "userId", "submittedAt"],
23       include: [
24          {
25             model: Questionnaire,
26             as: "questionnaire",
27             attributes: ["id", "name", "description"],
28             include: [
29                {
30                   model: Question,
31                   as: "questions",
32                   attributes: [
33                      "id",
34                      "questionText",
35                      "initialPosition",
36                      "position"
37                   ]
38                },
39                {
40                   model: Answer,
41                   as: "answers",
42                   attributes: ["id", "answerText", "position"]
43                }
44             ]
45          },
46          {
47             model: UserAnswer,
48             as: "userAnswers",
49             attributes: ["questionId", "answerId"]
50          }
51       ]
52    }).then(userSubmission => {
53       if (!userSubmission) {
```

```
54          const error = new Error(
55              "Eine Abgabe mit dieser ID konnte nicht gefunden werden."
56          );
57          error.statusCode = 404;
58          throw error;
59        }
60        res.status(200).json({data: userSubmission});
61     }).catch(err => {
62        next(err);
63     });
64 };
```

Listing 5.3: Serverseitiger JavaScript-Code zur Abfrage von ausgefüllten Fragebögen

5.3 Auswertung von Fragebögen

In diesem Abschnitt wird die Implementierung von Fragebogenauswertungen detailliert beschrieben. Wie im Kapitel zuvor wird zunächst auf den clientseitigen Code eingegangen, während im Anschluss die serverseitige Implementierung erläutert wird.

5.3.1 Clientseitige Implementierung

Diagramme spielen in der Anwendung eine wichtige Rolle. Sie sollen die Ergebnisse der Fragebögen und Übungen veranschaulichen, um dem `Admin` und `User` ein sinnvolles Feedback zu liefern. Für die Umsetzung der Diagramme wurde die Bibliothek *Chart.js* [72] eingesetzt. Sie wird in die Angular-Anwendung per npm eingebunden und liefert viele hilfreiche Features.
Chart.js-Diagramme werden über ein `<canvas>`-HTML-Element eingefügt (siehe Listing 5.4). Im Code wird das Diagramm über dessen ID in der Methode `initializeChart()` initialisiert. Hier wird auch der Typ der Charts festgelegt, der z. B. dafür sorgt, dass Werte in einem Liniendiagramm angezeigt werden.

```
<div class="mb-3">
   <select class="custom-select mt-4 mb-3" id="chart-select"
   [(ngModel)]="selectedChart" (change)="getSelectedChart()">
      <option *ngFor="let questionnaire of questionnaires"
         [value]="questionnaire.id">
         {{ questionnaire.name }})
      </option>
   </select>
</div>
<div *ngIf="loading" class="d-flex justify-content-center m-4">
   <div class="spinner-border" role="status">
      <span class="sr-only">Loading...</span>
   </div>
</div>
<div>
   <canvas id="chart"></canvas>
</div>
```

Listing 5.4: HTML zur Anzeige von Diagrammen

Die Diagramme für die jeweiligen Fragebögen sollen dynamisch mit den entsprechenden Daten gefüllt werden. Über ein Select-Menü kann der Nutzer den gewünschten Bogen auswählen. Dabei wird zum einen der Wert `selectedChart` über Two-Way Binding mit der `Component` synchronisiert und zum anderen bei jeder Änderung die Methode `getSelectedChart()` ausgeführt (siehe Listing 5.5). Bei `selectedChart` handelt es sich um die ID des Fragebogens, der ausgewählt wurde. In der oben genannten Methode werden die Daten über den `QuestionnaireService` abgefragt. Die Methode `getUserSubmissionsOfQuestionnaire()` liefert dabei alle Abgaben eines `Users` zu diesem Fragebogen. Um die Daten im Diagramm anzuzeigen, wird die Methode `fillChartWithData()` aufgerufen. Hier kommen die nützlichen Features von Chart.js zum Einsatz. Zunächst werden die x-Achsen des Diagramms erstellt. Dazu wird die ID und der Abgabezeitpunkt in einem Array gespeichert, das später von Chart.js zum Rendern der x-Achse genutzt werden kann. Je nach dem welcher Fragebogen

ausgewählt wurde, sollen andere Farben, Bezeichnungen und Berechnungen für die Anzeige verwendet werden. Dies wird über ein Switch-Case gelöst. Listing 5.5 zeigt aus Gründen der Übersichtlichkeit nur einen Ausschnitt für die Berechnung und Anzeige eines Fragebogens. Mithilfe von Chart.js können Farbe und Aussehen des Diagramms leicht angepasst werden. Für diesen Bogen wird die Summe über alle vom `User` ausgewählten Antworten berechnet.

```
1  // aus chart-card.component.ts
2  initializeChart() {
3     this.chart = new Chart("chart", {
4        type: "line"
5     });
6  }
7
8  getSelectedChart() {
9     this.loading = true;
10    this.questionnaireService
11       .getUserSubmissionsOfQuestionnaire(this.userId, this.
             selectedChart)
12       .subscribe((submissions: UserSubmission[]) => {
13          this.fillChartWithData(+this.selectedChart, submissions);
14          this.loading = false;
15       }, error => {
16          this.alertOpen = true;
17          this.alertMessage = error.error.message;
18       });
19 }
20
21 fillChartWithData(selectedChart: number, submissions: UserSubmission
       []) {
22    const xAxisLabels = [];
23    for (const submission of submissions) {
24       xAxisLabels.push(`ID: ${submission.id},
25          ${this.convertTimestamp(submission.submittedAt)}`);
```

```
26      }
27
28      switch (selectedChart) {
29          /* Code */
30          case 3: {
31              this.chart.data.labels = xAxisLabels;
32              this.chart.data.datasets = [{
33                  label: "Perceived Stress Scale",
34                  fill: false,
35                  lineTension: 0,
36                  pointRadius: 3.5,
37                  backgroundColor: "#15C96F",
38                  borderColor: "#15C96F",
39                  data: this.diagramEvaluationService.calculateSum(
                        submissions, selectedChart, null, [4, 5, 7, 8])
40              }];
41              this.setChartOptions("Perceived Stress Scale");
42              this.chart.update();
43              break;
44          }
45          /* Code */
46      }
47  }
```

Listing 5.5: TypeScript-Code zur Anzeige von Diagrammen

Chart.js erwartet die Daten, die im Diagramm angezeigt werden sollen, in Form eines Arrays. Die Methode `calculateSum()`, die im `DiagramEvaluation-Service` implementiert wurde, berechnet dabei für jede Abgabe zu einem Fragebogen die Summe der Antwort-Werte und gibt im Anschluss ein Array mit den berechneten Werten zurück (siehe Listing 5.6). Die Methode benötigt neben den `userSubmissions` und der Fragebogen-ID noch zwei weitere Angaben: `questionsToFilter` und `answersToInvert`. Bei beiden Argumenten handelt es sich um Arrays. `questionsToFilter` enthält alle Fragen, die für die Berechnung verwendet werden

sollen, da viele Fragebögen nicht die Summe über alle Fragen berechnen, sondern diese auf verschiedene Skalen aufteilen. Diese werden dabei über den Wert `initialPosition` angegeben.
Für die Berechnung ist es oft nötig, die Werte der angegebenen Antworten zu invertieren. Dafür wird das Array `answersToInvert` verwendet, das die Antworten über den Wert `position` identifiziert. Mithilfe der Methode `invertAnswerValue()` wird die Invertierung durchgeführt. Für manche Fragebögen muss der Wert der Antwort zusätzlich um eins korrigiert werden, da den Antworten Werte von 0 bis n statt von 1 bis n zugeordnet wurden.

```
1  // aus diagram-evaluation.service.ts
2  calculateSum(userSubmissions: UserSubmission[],
3               selectedChart: number,
4               questionsToFilter: number[],
5               answersToInvert: number[]) {
6
7     const userAnswerData: UserAnswers[][] = [];
8     const sumData: number[] = [];
9
10    for (const userSubmission of userSubmissions) {
11       userAnswerData.push(userSubmissions.userAnswers);
12    }
13
14    for (let userAnswers of userAnswerData) {
15       let sum = 0;
16       // filter alle Fragen aus questionsToFilter-Array
17       if (questionsToFilter) {
18          userAnswers = userAnswers.filter(el => questionsToFilter.
               includes(el.question.initialPosition));
19       }
20
21       for (const userAnswer of userAnswers) {
22          let selectedAnswerValue: number;
23          // korrigiere selectedAnswerValue fuer PSS-10 und WHO-5
```

```
24          if (selectedChart === 3 || selectedChart === 4) {
25              selectedAnswerValue = userAnswer.answer.position - 1;
26          } else {
27              selectedAnswerValue = userAnswer.answer.position;
28          }
29          // invertiere alle Antworten aus answersToInvert-Array
30          if (answersToInvert) {
31              for (const value of answersToInvert) {
32                  if (value === userAnswer.question.initialPosition) {
33                      selectedAnswerValue = this.invertAnswerValue(
                            selectedAnswerValue, selectedChart);
34                  }
35              }
36          }
37          sum += selectedAnswerValue;
38      }
39      sumData.push(sum);
40  }
41  return sumData;
42 }
```

Listing 5.6: TypeScript-Code zur Berechnung von Diagrammen

5.3.2 Serverseitige Implementierung

Um alle notwendigen Daten zur Auswertung von Fragebogenabgaben zu erhalten, muss eine Anfrage an `/api/v1/users/:userId/submissions?questionnaire=:id` gestellt werden (siehe Listing 5.7). Ohne die Angabe des URL-Parameters `questionnaire` würden alle Abgaben eines `Users` gesendet werden. Zur Auswertung ist es allerdings nötig, zu einem bestimmten Fragebogen alle Abgaben des Nutzers zu liefern. Dies kann durch den Parameter gesteuert werden. Im `user.controller.js`-File wird zunächst in der Funktion `getUserSubmissions` geprüft, ob alle dynamischen Werte innerhalb der `Route` korrekt sind. Wenn nicht, wird ein Fehler geworfen.

Danach wird unterschieden, ob alle Abgaben eines `Users` geliefert werden sollen oder nur Abgaben zu einem bestimmten Fragebogen. Dementsprechend wird eine Datenbankabfrage mithilfe von Sequelize ausgeführt.

```
1  // in user.routes.js
2  router.get("/:id/submissions", checkAuth, checkAdmin, userController.
       getUserSubmissions);
3
4  // in user.controller.js
5  exports.getUserSubmissions = (req, res, next) => {
6     const questionnaireId = +req.query.questionnaire;
7     User.findByPk(req.params.id).then(user => {
8        if (!user) {
9           const error = new Error(
10             "Ein Nutzer mit dieser ID konnte nicht gefunden werden."
11          );
12          error.statusCode = 404;
13          throw error;
14       }
15       if (questionnaireId) {
16          Questionnaire.findByPk(questionnaireId)
17          .then(questionnaire => {
18             if (!questionnaire) {
19                const error = new Error(
20                   "Kein Fragebogen mit dieser ID verfuegbar."
21                );
22                error.statusCode = 404;
23                throw error;
24             }
25             UserSubmission.findAll({
26                where: {
27                   userId: req.params.id,
28                   questionnaireId: questionnaireId
29                },
30                attributes: ["id", "userId", "submittedAt"],
```

```
31                 include: [
32                     {
33                         model: Questionnaire,
34                         as: "questionnaire",
35                         attributes: ["id", "name", "instructions"]
36                     },
37                     {
38                         model: UserAnswer,
39                         as: "userAnswers",
40                         attributes: ["questionId", "answerId"],
41                         include: [
42                             {
43                                 model: Question,
44                                 as: "question",
45                                 attributes: [
46                                     "id",
47                                     "questionText",
48                                     "initialPosition",
49                                     "position"
50                                 ]
51                             },
52                             {
53                                 model: Answer,
54                                 as: "answer",
55                                 attributes: [
56                                     "id",
57                                     "answerText",
58                                     "position"
59                                 ]
60                             }
61                         ]
62                     }
63                 ]
64             }
65         }).then(userSubmissions => {
```

```
66              res.status(200).json({data: userSubmissions});
67          }).catch(err => {
68              next(err);
69          });
70      } else {
71          UserSubmission.findAll({
72              where: {
73                  userId: req.params.id,
74                  exerciseId: null
75              },
76              attributes: ["id", "userId", "submittedAt"],
77              include: [
78                  {
79                      model: Questionnaire,
80                      as: "questionnaire",
81                      attributes: ["id", "name", "instructions"]
82                  }
83              ]
84          }).then(userSubmissions => {
85              res.status(200).json({data: userSubmissions});
86          }).catch(err => {
87              next(err);
88          });
89      }
90  }).catch(err => {
91      next(err);
92  });
93 };
```

Listing 5.7: Serverseitiger JavaScript-Code zur Abfrage von allen `User`-Abgaben eines Fragebogens

6

Vorstellung der Webanwendung

Im Rahmen dieses Kapitels wird die Webseite von Mindful Walking vorgestellt. Anhand geeigneter Screenshots sollen nicht nur einzelne Funktionen, sondern auch die grafische Benutzeroberfläche näher beschrieben werden.

6.1 Startseite

Nach Aufrufen der Webanwendung im Browser erscheint zunächst die Startseite (siehe Abbildung 6.1). Hier werden dem Nutzer alle wichtigen Informationen über das Projekt mitgeteilt. Über die Navigationsleiste am oberen Rand oder über die Fußzeile am Ende der Seite können Funktionen wie Anmelden oder weitere Seiten wie Über uns, das Impressum oder die Datenschutzerklärung aufgerufen werden. Ebenfalls ist unten eine Verlinkung zur mobilen Anwendung im App- oder Google Play Store vorzufinden.

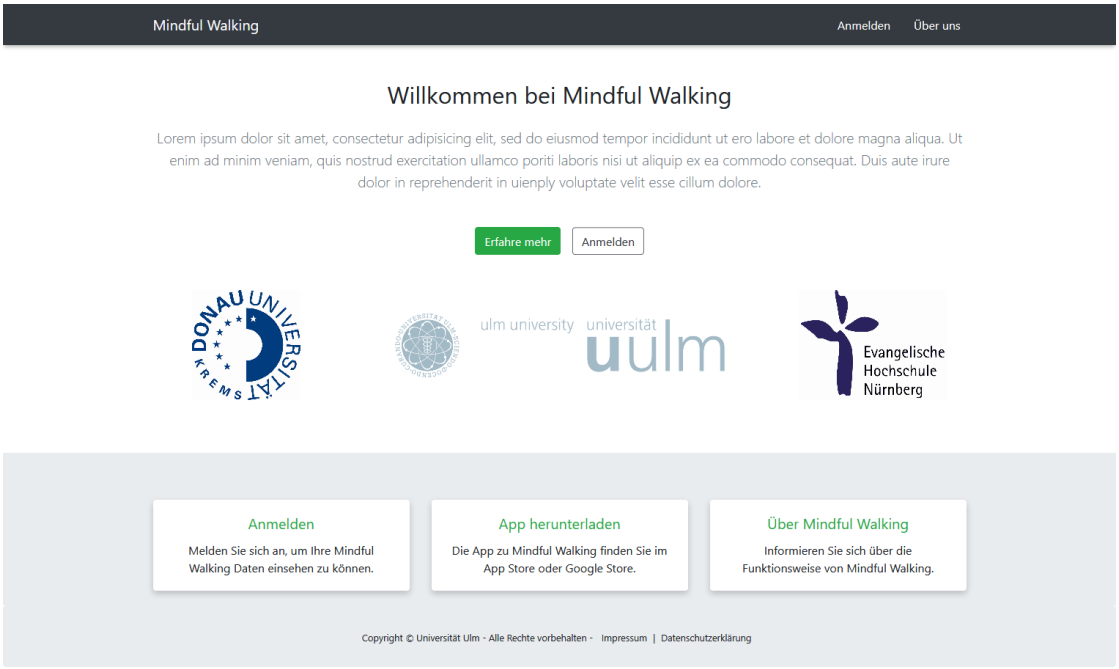

Abbildung 6.1: Startseite der Mindful Walking Webanwendung

6.2 Anmeldung

Nach Betätigung des Anmelde-Buttons über die Navigationsleiste, wird der Anmelde-Dialog geöffnet (siehe Abbildung 6.2). Hier kann der Nutzer seine Zugangsdaten wie E-Mail-Adresse und Passwort eingeben um sich einzuloggen. Zusätzlich ist es möglich, die Funktion *Passwort vergessen* über den entsprechenden Button aufzurufen, wodurch ein neuer Dialog geöffnet wird. Durch Eingabe der E-Mail-Adresse kann sich der Nutzer einen Link zum Zurücksetzen des Passworts senden lassen.
Hat ein Nutzer seine E-Mail-Adresse noch nicht verifiziert, erscheint eine entsprechende Fehlermeldung. Dabei wird ihm die Funktion *neuen Link anfordern* angeboten, die einen Dialog öffnet. Dort kann der Nutzer seine E-Mail-Adresse eintragen, um sich einen neuen Verifizierungslink senden zu lassen. Alle Dialoge können über das X am oberen rechten Rand oder durch den Abbrechen-Button geschlossen werden.

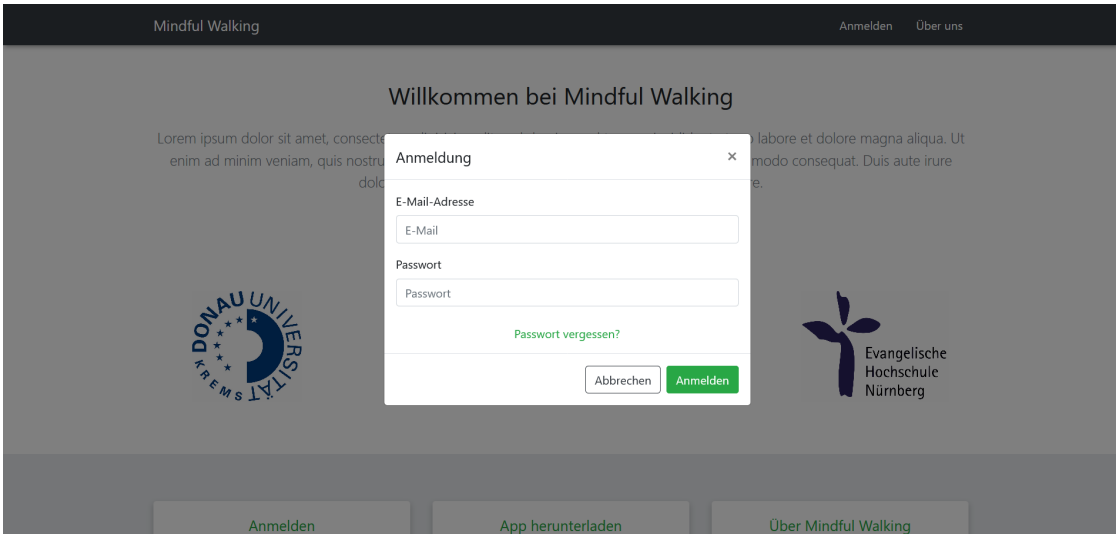

Abbildung 6.2: Dialog zur Anmeldung auf der Webseite

6.3 Dashboard

Nach erfolgreicher Anmeldung erscheint den Nutzern der Webseite zuerst ein Dashboard. Der `User` erhält hier lediglich Informationen über die Studie, während der `Admin` seine Benachrichtigungen in Form einer Liste vorfindet (siehe Abbildung 6.3 und 6.4). Durch Anklicken der Nachrichten wird der `Admin` auf das entsprechende Profil des `Users` oder der Abgabe navigiert.

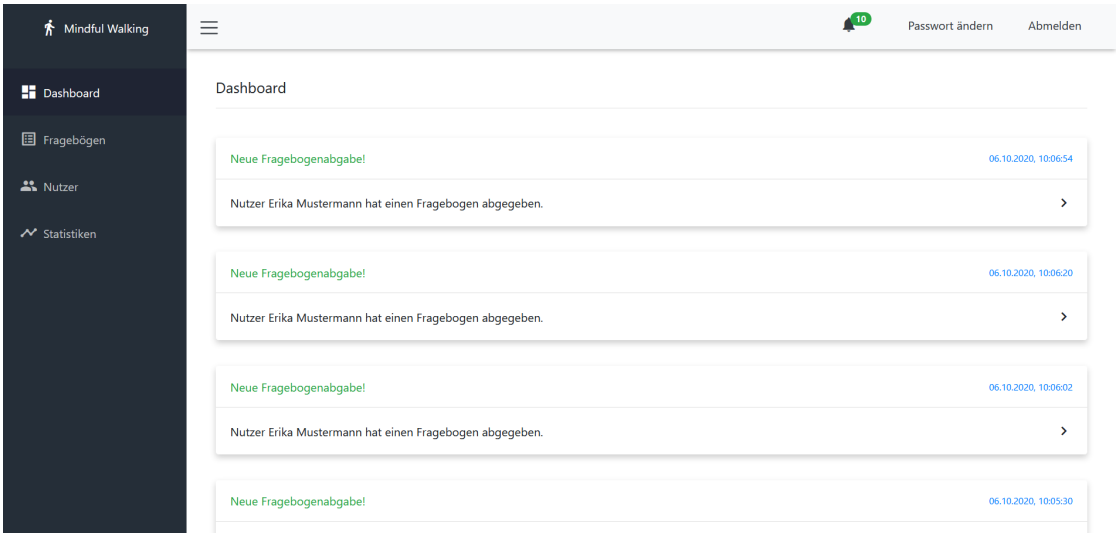

Abbildung 6.3: Dashboard und Navigationsansicht eines `Admins`

6.4 Navigation

Die Navigations- und Seitenleiste sehen für den `User` und den `Admin` jeweils unterschiedlich aus (siehe Abbildung 6.3 und 6.4). Während der `User` über die obere Leiste sein Profil aufrufen kann, steht dem `Admin` dort ein Benachrichtigungs-Icon zur Verfügung. Nach Öffnen des Menüs durch Anklicken des Icons kann der `Admin` dort seine Benachrichtigungen als gelesen markieren. Zusätzlich ist es für ihn möglich, über den Button *Passwort ändern* den entsprechenden Dialog mit der genannten Funktion zu öffnen. Beide haben die Möglichkeit sich über die Navigationsleiste abzumelden.
Die Seitenleiste ist standardmäßig auf der Webseite geöffnet. Sie kann jedoch über das Burger-Icon geschlossen und wieder angezeigt werden. Mithilfe der Seitenleiste kann der Nutzer zu jeglichen Seiten und Funktionen der Webseite navigieren.
Um mögliche Aktionen verständlicher darzustellen, wurden die *Material Icons* [73] von Google verwendet.

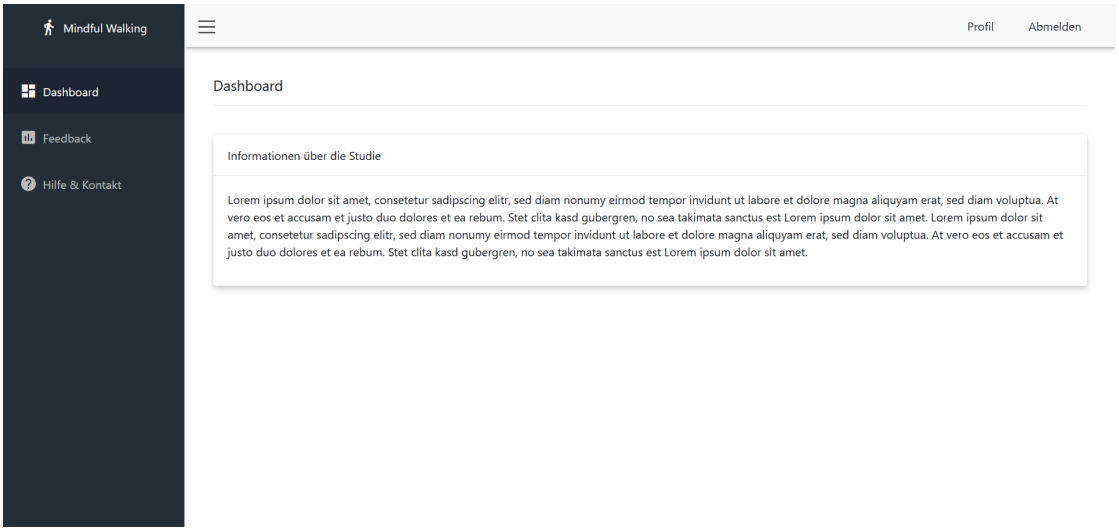

Abbildung 6.4: Dashboard und Navigationsansicht eines `Users`

6.5 Profil

Über die Profilseite kann der `User` seine Kontodaten einsehen und bearbeiten (siehe Abbildung 6.5). Auch die Funktion *Passwort ändern* ist dort vorzufinden. Sie kann über das Stift-Icon aufgerufen werden. Dadurch erscheint dem `User` ein Dialog zum Ändern des Passworts. Um dieses erfolgreich ersetzen zu können, muss er sein altes und neues Passwort eingeben. Sein neues muss er erneut bestätigen. Durch Speichern wird die Änderung übernommen, durch Abbrechen verworfen und der Dialog geschlossen. Betätigt der `User` den Button *Profil bearbeiten*, können die zuvor deaktivierten Eingabefelder benutzt werden, um die Daten direkt zu bearbeiten. Die Änderungen können durch den Button *Änderung speichern* übernommen oder über den Abbrechen-Button wieder gelöscht werden.

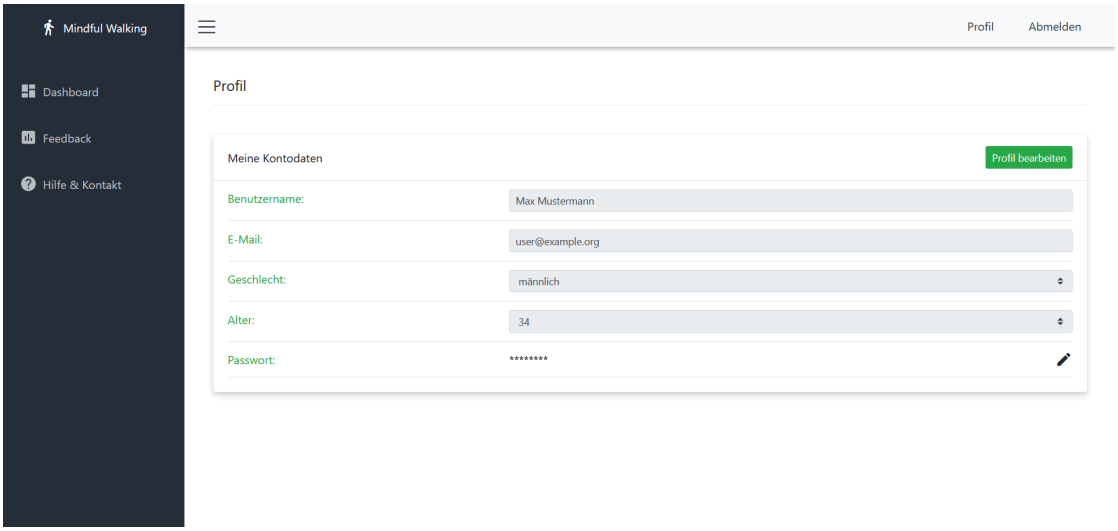

Abbildung 6.5: Profilansicht eines `Users`

6.6 Feedback

Auf der Feedbackseite, die über die Seitenleiste erreichbar ist, kann der `User` nach erfolgreichem Beenden der Studie sein Feedback einsehen. Über das Select-Menü kann er die verfügbaren Diagramme auswählen (siehe Abbildung 6.7). Dadurch erhält er einen Vorher-Nachher-Vergleich seiner Abgaben im Verlauf der Studie. Ist das Feedback noch nicht freigeschaltet, ist das Select-Menü des Diagramms deaktiviert (siehe Abbildung 6.6). Er erhält zudem einen Informationstext, der ihn darauf hinweist, dass sein Feedback noch nicht verfügbar ist.

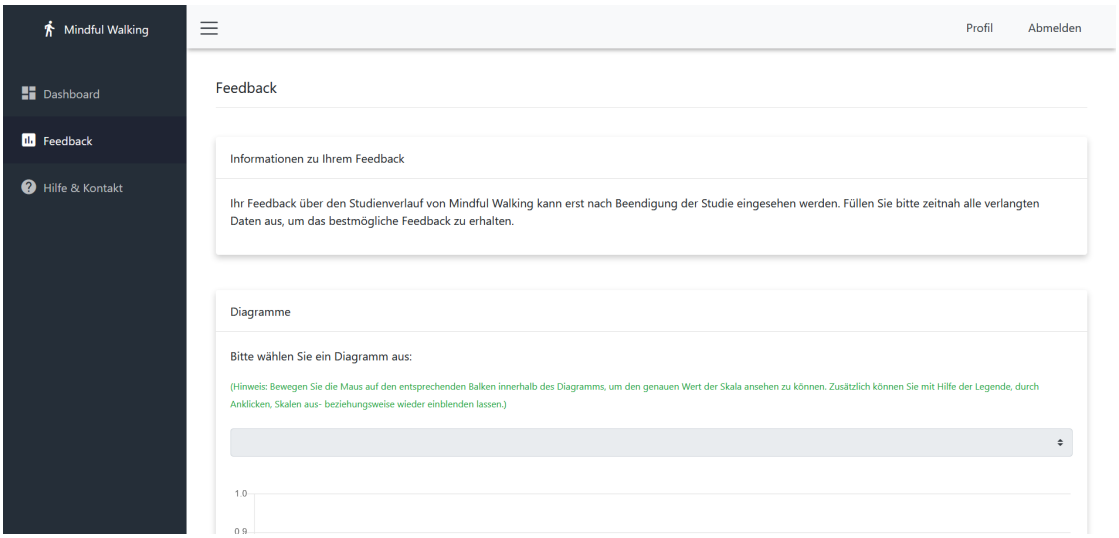

Abbildung 6.6: Feedback-Ansicht vor der Freischaltung

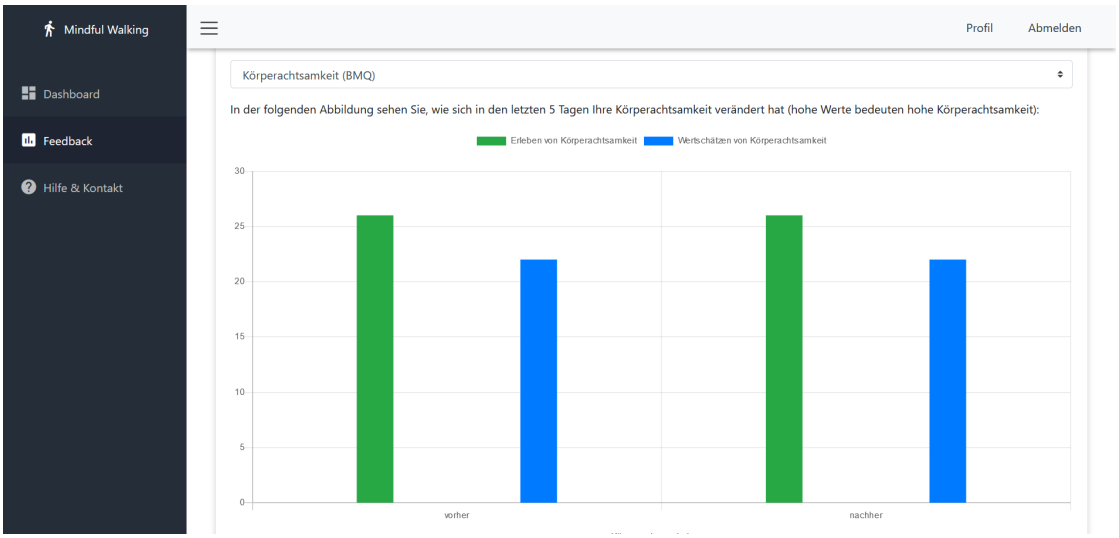

Abbildung 6.7: Feedback-Ansicht nach der Freischaltung

6.7 Hilfe und Kontakt

Über die Seitenleiste kann der `User` die Seite *Hilfe & Kontakt* erreichen (siehe Abbildung 6.4). Hier findet er alle Informationen, um sich mit dem Systemadministrator in Verbindung zu setzen, falls er Probleme oder Fragen zur Studie hat.

6.8 Fragebögen

Die Seite mit der Fragebogenliste steht nur dem `Admin` zur Verfügung. Hier werden alle vorhandenen Fragebögen der Studie aufgeführt (siehe Abbildung 6.8). Durch Anklicken eines Listeneintrags wird der `Admin` zur Bearbeitungsseite des jeweiligen Fragebogens navigiert (siehe Abbildung 6.9). Hier kann er über entsprechende Eingabefelder die Texte und Beschreibungen des Bogens anpassen. Die Felder werden über den Button *Fragebogen bearbeiten* aktiviert. Zusätzlich kann per Drag and Drop die Reihenfolge der Fragen geändert werden (siehe Abbildung 6.10). Über das Breadcrumb ist es möglich, zurück zur Fragebogenliste zu navigieren.

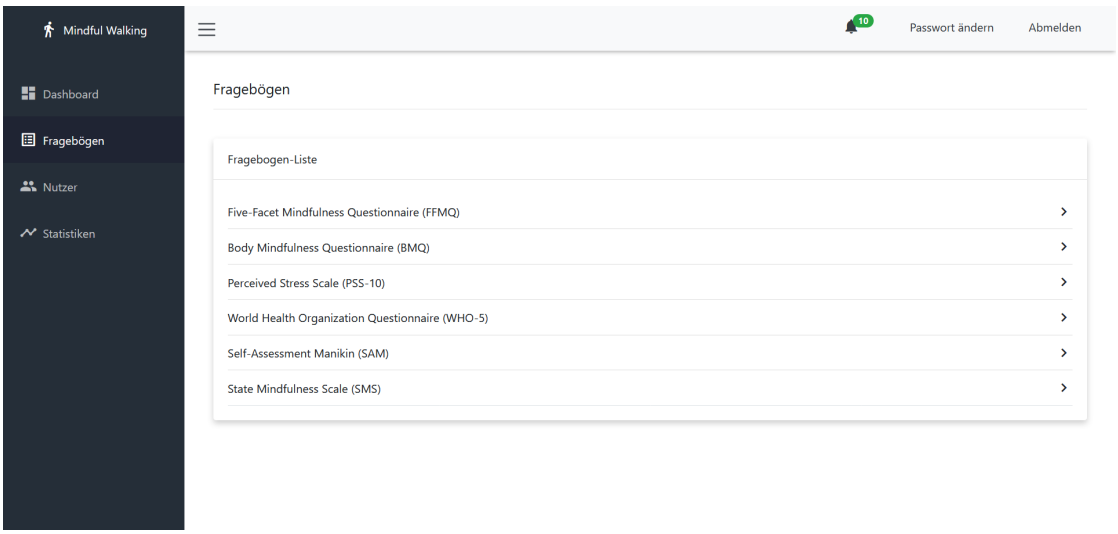

Abbildung 6.8: Liste aller Fragebögen im System

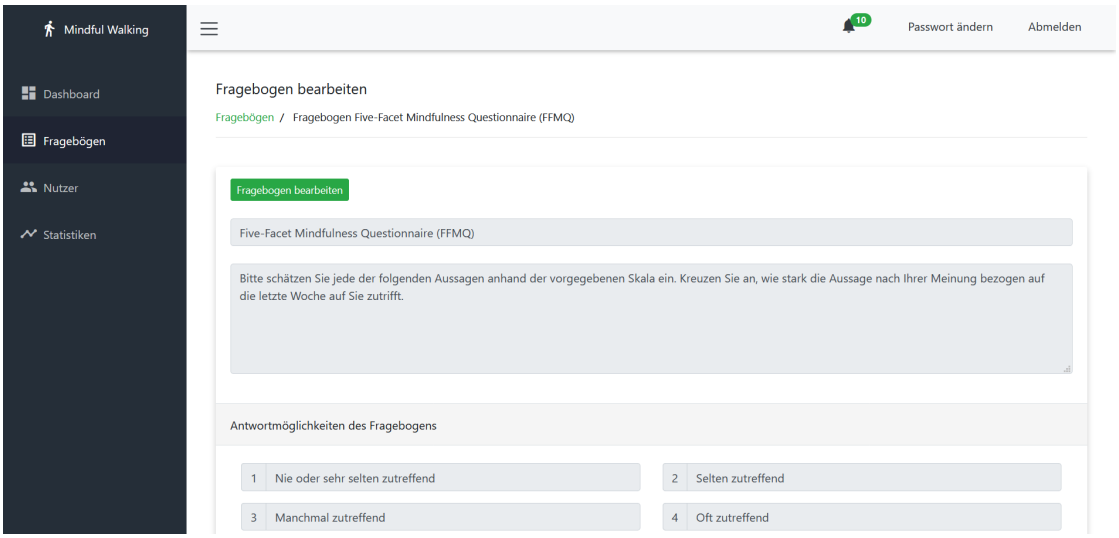

Abbildung 6.9: Bearbeitung von Fragebögen - Teil eins

Abbildung 6.10: Bearbeitung von Fragebögen - Teil zwei

6.9 Nutzer

Mithilfe der Nutzerseite erhält der `Admin` einen Überblick über alle angemeldeten `User` des Systems (siehe Abbildung 6.11). Zusätzlich kann er hier durch das Export-Icon rechts oben die Daten aller `User` im CSV-Format herunterladen. Durch Betätigen des Buttons *Daten ansehen* wird der `Admin` auf das jeweilige Nutzerprofil navigiert. Hier sind alle detaillierten Daten wie Profildaten, Abgaben und Diagramme des Users in Abschnitten aufgelistet (siehe Abbildung 6.12). Einzelne Abschnitte können über das Chevron-Icon auf- und wieder zugeklappt werden. Die Abgaben der `User`, die per Klick auf den jeweiligen Listeneintrag aufgerufen werden können, sind in den Abschnitten *Abgegebene Fragebögen* und *Abgegebene Übungen* aufgelistet. Über das Icon rechts oben kann der entsprechende Nutzer gelöscht werden. Dazu erscheint zunächst ein Dialog, der sicherstellt, dass der `Admin` diese Aktion auch wirklich durchführen möchte. Abbildung 6.13 zeigt die Ansicht für ausgefüllte Fragebögen.

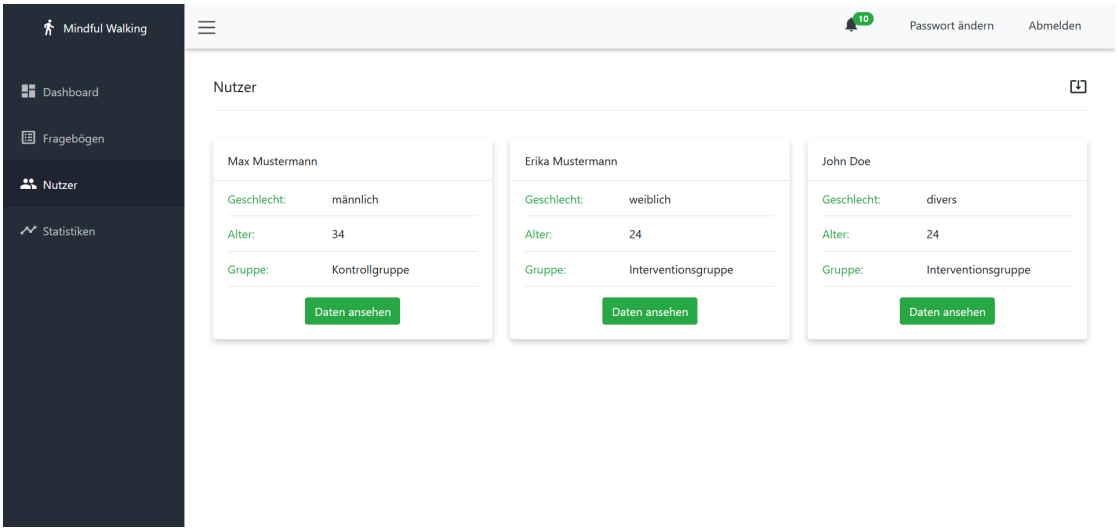

Abbildung 6.11: Nutzeransicht

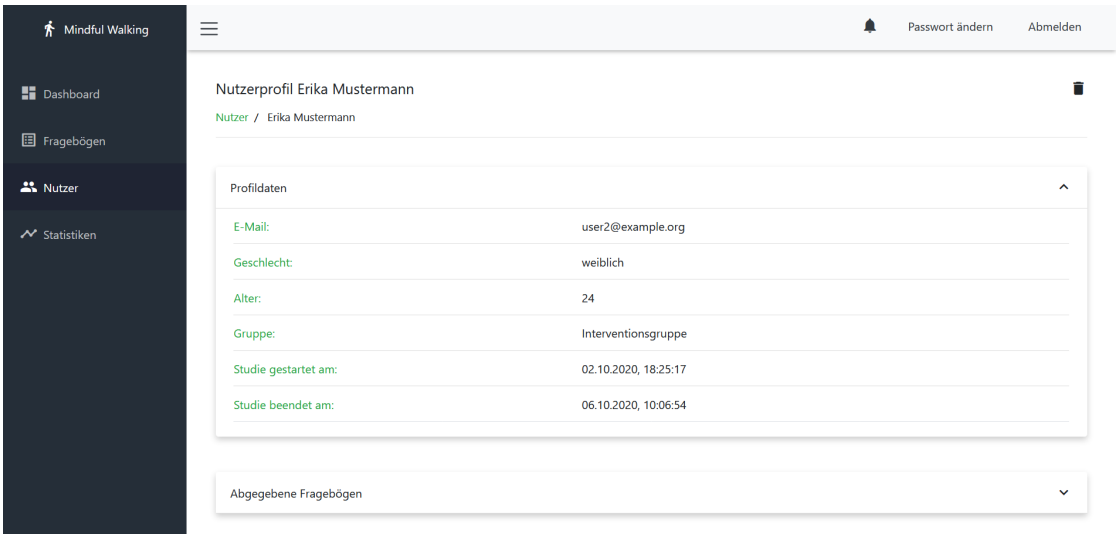

Abbildung 6.12: Nutzerprofil

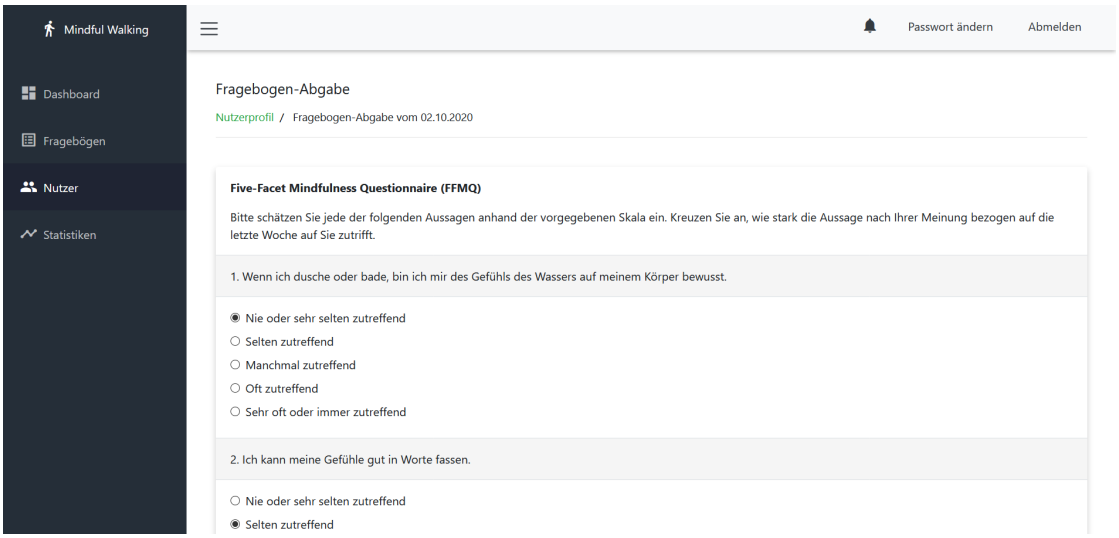

Abbildung 6.13: Ansicht für ausgefüllte Fragebögen

6.10 Statistiken

Die Statistiken werden dem `Admin` in Form von Diagrammen dargestellt. Er erhält eine Übersicht zur Anzahl der Nutzer und deren Verteilung nach Geschlecht bzw. Gruppe, sowie zur Anzahl der Abgaben im System (siehe Abbildung 6.14).

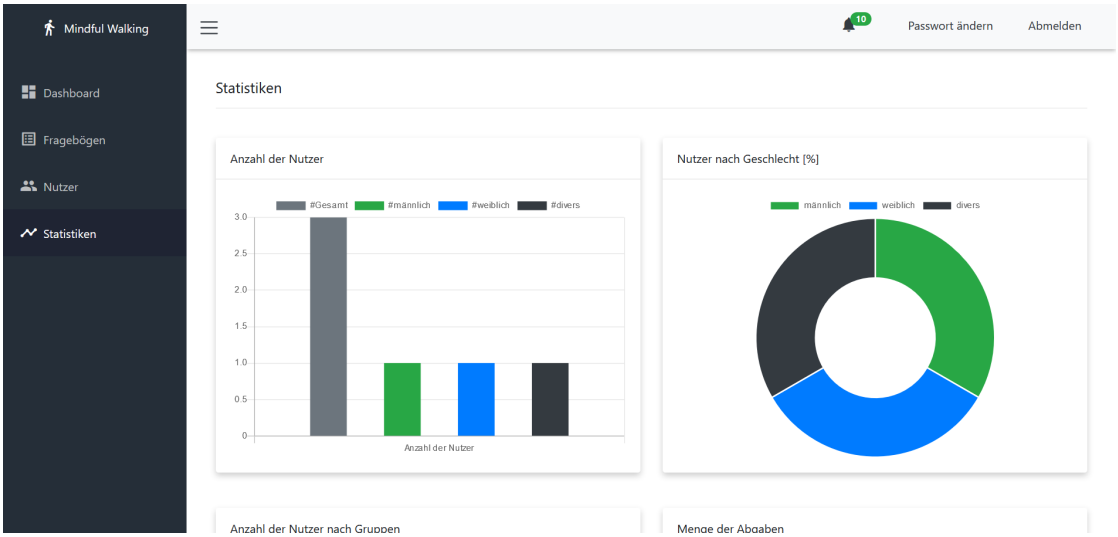

Abbildung 6.14: Statistiken über Systemdaten

7

Anforderungsabgleich

In diesem Kapitel werden alle Anforderungen an die Mindful Walking Plattform, die in Kapitel 3 beschrieben wurden, bezüglich ihres Erfüllungsgrads abgeglichen.

7.1 Abgleich der funktionalen Anforderungen

In diesem Abschnitt werden zunächst die funktionalen Anforderungen abgeglichen. FA1 bis FA11 stellen dabei die Anforderungen an die Webanwendung dar, während FA12 bis FA16 die zusätzlichen Anforderungen an die Serveranwendung repräsentieren (vgl. Kapitel 3).

FA1 An - und Abmeldung:
Erfüllt. Nutzer können sich mit ihrer E-Mail-Adresse und ihrem Passwort auf der Webseite anmelden und danach alle verfügbaren Funktionen der Anwendung nutzen. Die Abmeldung von der Webseite ist ebenfalls möglich.

FA2 Passwort ändern und zurücksetzen:
Erfüllt. Nutzer können ihr Passwort zurücksetzen, falls sie es vergessen haben. Innerhalb der Anwendung besteht zusätzlich die Möglichkeit, das Passwort zu ändern.

FA3 Profil bearbeiten:
Erfüllt. Der `User` kann seine Kontodaten wie E-Mail-Adresse, Benutzername, Alter und Geschlecht über sein Profil ansehen und bearbeiten.

FA4 Feedback:

Erfüllt. Dem `User` wird nach Beendigung der Studie sein Feedback freigeschaltet. Die Daten, die im Verlauf der Studie über ihn gesammelt wurden, werden in Form von Vorher-Nachher-Diagrammen veranschaulicht.

FA5 Informationen und Kontakt:

Erfüllt. Der `User` kann Informationen zur Studie ansehen und sich aufgrund der Angabe von Kontaktdaten im Falle von Problemen oder Fragen mit dem Administrator der Seite in Verbindung setzen.

FA6 Benachrichtigungen:

Erfüllt. Der `Admin` erhält Benachrichtigungen über neue Teilnehmer an der Studie, ausgefüllte Fragebögen und Übungsabgaben, die allesamt auf einem Dashboard angezeigt werden. Er kann die Benachrichtigungen einzeln oder alle mit einem Klick als gelesen markieren.

FA7 Fragebögen:

Erfüllt. Dem `Admin` werden alle verfügbaren Fragebögen in Form einer Liste angezeigt. Er kann diese einzeln aufrufen und bearbeiten. Im Bearbeitungsmodus hat er die Möglichkeit, Name, Beschreibung und alle Antwort- bzw. Fragetexte des Bogens zu ändern. Die Reihenfolge der Fragen kann ebenfalls geändert werden.

FA8 Nutzerdaten:

Erfüllt. Die Kontodaten der angemeldeten `User` kann der `Admin` auf den jeweiligen Profilen ansehen. Des Weiteren sind hier die ausgefüllten Fragebögen bzw. Übungsdaten der `User` übersichtlich aufgelistet und können dabei einzeln aufgerufen werden. Der `Admin` hat zusätzlich die Möglichkeit, einzelne `User` zu löschen.

FA9 Diagramme:

Erfüllt. Dem `Admin` werden Diagramme zu Auswertungen der Fragebögen und Übungsdaten der `User` auf dem entsprechenden Nutzerprofil bereitgestellt.

FA10 CSV Export:

Erfüllt. Der `Admin` hat die Möglichkeit, die Daten der Nutzer im CSV-Format

herunterzuladen. Dazu wurden die Bibliotheken *json2csv* [74] und *FileSaver.js* [75] eingesetzt.

FA11 Statistiken:

Erfüllt. Dem `Admin` werden Statistiken zur Anzahl der Nutzer nach Geschlecht bzw. Gruppe und Menge der Abgaben bereitgestellt.

FA12 Registrierung:

Erfüllt. `User` können sich mithilfe der API über die mobile Anwendung registrieren, indem Daten wie E-Mail-Adresse, Benutzername, Passwort, Alter und Geschlecht empfangen und abgespeichert werden.

FA13 E-Mail-Adresse verifizieren:

Erfüllt. Nach der Registrierung wird dem `User` ein Verifizierungslink gesendet, mit dem er seine E-Mail-Adresse bestätigen kann. Dazu wurden die Bibliotheken *nodemailer* [76] und *nodemailer-express-handlebars* [77] eingesetzt. Erst nach der erfolgreichen Bestätigung kann er sich bei der Webseite oder mobilen Anwendung anmelden.

FA14 ausgefüllte Fragebögen und Übungsabgaben:

Erfüllt. Über die Schnittstelle werden ausgefüllte Fragebögen und Übungsabgaben, die von `Usern` über die mobile Anwendung geschickt werden, empfangen und gespeichert.

FA15 Studie starten:

Erfüllt. Nachdem `User` die Mindful Walking Studie gestartet haben, wird diese Information über die API gespeichert. Danach werden alle Funktionen, die zum Ablauf der Studie notwendig sind, freigeschaltet.

FA16 Aufgaben:

Erfüllt. Über die Schnittstelle ist es möglich, Aufgaben für `User` abzufragen. So wird ihnen mitgeteilt, welche Fragebögen sie ausfüllen müssen.

7.2 Abgleich der nicht-funktionalen Anforderungen

Abschließend werden in diesem Abschnitt die nicht-funktionalen Anforderungen an das System abgeglichen. Diese wurden ebenfalls alle erfolgreich umgesetzt.

NFA1 Benutzbarkeit:

Erfüllt. Aufgrund der Verwendung von Bootstrap ist das Design der Webseite einheitlich und benutzerfreundlich, da das Framework intuitiv bedienbare Komponenten wie Radio-Buttons oder Navigationsleisten bereitstellt. Bei der Erstellung der Webanwendung wurde darauf geachtet, Tooltips und eindeutige Bezeichner für Funktionen einzusetzen, um Verständnisprobleme bei der Nutzung der Seite zu vermeiden.

NFA2 Zuverlässigkeit:

Erfüllt. Durch ausreichendes Testen der Plattform wurde sichergestellt, dass das System stets den Erwartungen entsprechend reagiert. Fehler werden nicht nur zuverlässig abgefangen und behandelt, sondern auch durch verständliche Fehlermeldungen beschrieben.

NFA3 Sicherheit:

Erfüllt. Die Daten der Nutzer werden aufgrund der erforderlichen Bestätigung der Identität über JSON Web Tokens (vgl. Kapitel 4.1.1) vor unbefugten Zugriffen geschützt.

NFA4 Wartbarkeit:

Erfüllt. Durch die Verwendung geeigneter Frameworks, wie Angular oder Node.js mit Express, die Anwendungen modular aufbauen, können Komponenten, Funktionen und Klassen leicht erweitert, hinzugefügt oder aktualisiert werden.

NFA5 Effizienz:

Erfüllt. Bei Angular-Anwendungen handelt es sich um Single-Page-Applikationen. Sie sind sehr performant, da sie bei Interaktionen mit dem Nutzer nur einzelne Teile der Ansicht ein- oder ausblenden und nicht jedes Mal eine neue HTML-Seite vom Server anfragen müssen [24]. Node.js sorgt ebenfalls aufgrund der Event-

Loop-Technologie für eine performante Serveranwendung, die schnelle Antworten liefert.

NFA6 Responsiveness:

Erfüllt. Das Grid-System von Bootstrap, das für die Implementierung der Webseite eingesetzt wurde, sorgt für die automatische Anpassung der Ansicht auf unterschiedlichen Bildschirmgrößen.

8

Fazit

In diesem Kapitel wird die erstellte Arbeit abschließend bewertet und die entstandenen Ergebnisse zusammengefasst.

8.1 Zusammenfassung

Die Zielsetzung der vorliegenden Arbeit war die Entwicklung einer Webplattform zur Unterstützung von Studien zur Stressreduzierung auf Basis des Mindful Walking Paradigmas. Durch die entstandene Plattform ist es möglich, ausgefüllte Fragebögen und Übungsabgaben von Teilnehmer der Mindful Walking Studie, die mithilfe der bereits existierenden mobilen Anwendung gesammelt werden, persistent in einer Datenbank zu speichern und für Forscher bzw. Nutzer geeignet in Diagrammen zu visualisieren. Des Weiteren können Benutzer ihre Profildaten bearbeiten und wichtige Informationen über die genannte Studie einsehen bzw. in Erfahrung bringen. Gerade für Forscher bietet die Plattform zahlreiche Features zur Verwaltung von Studienteilnehmern, Abgaben und Systemdaten, um wertvolle Informationen über die Effektivität der Mindful Walking Übungen abzuleiten.

Das umgesetzte System wurde dabei in eine Web- und Serveranwendung zerlegt, die über eine einheitliche REST-Schnittstelle miteinander kommunizieren. Mithilfe der Client-Serverarchitektur ist es möglich, weitere Clients einfach zu integrieren, da diese ebenfalls mit der Serveranwendung über die API interagieren können. Auf diese Weise erfolgt die Synchronisation der Nutzerdaten über die mobile Applikation mit der entstandenen Webanwendung.

Durch die Auswahl geeigneter Tools und Frameworks konnten die in Kapitel 3 beschriebenen funktionalen Anforderungen alle umgesetzt werden. Aufgrund der Implementierung einer Single-Page-Applikation mit Angular und einer Serveranwendung mit Node.js war es möglich, ein performantes System zu verwirklichen, das nicht nur benutzerfreundlich ist, sondern auch zuverlässig und robust arbeitet. Demzufolge konnten auch alle nicht-funktionalen Anforderungen an die Plattform erfolgreich umgesetzt werden.

8.2 Ausblick

Obwohl alle genannten Anforderungen umgesetzt wurden, besteht aufgrund des modularen Aufbaus der Plattform die Möglichkeit, weitere Funktionen und Komponenten problemlos hinzuzufügen. Dieser Abschnitt soll einen Ausblick auf mögliche Erweiterungen der Plattform liefern.

8.2.1 Sprache

Bisher wird die Plattform nur auf Deutsch angeboten. Je nach Einsatz könnte das System international zu Verfügung gestellt werden, indem es in weiteren Sprachen, insbesondere Englisch, angeboten wird.

8.2.2 Dynamische Fragebögen

Eine weitere denkbare Funktion wären dynamische Fragebögen. Bisher ist es lediglich möglich, fest definierte Bögen zu bearbeiten und die Reihenfolge der Fragen über die Webseite zu ändern. Durch die genannte Erweiterung könnten nach Belieben neue Fragen zu bestehenden Fragebögen hinzugefügt oder entfernt werden. Da dies eine Auswirkung auf die Auswertung der Bögen und die Diagramme hat, müsste dafür ebenfalls eine UI-basierte Lösung gefunden werden, die es erlaubt, Auswertungen dynamisch zu ändern.

8.2.3 Erfolge

Um Nutzer zu motivieren, täglich ihre Mindful Walking Übungen durchzuführen, wäre es denkbar, individuelle Nutzer-Erfolge anzuzeigen. Dazu könnten Parameter wie Häufigkeit oder Länge der Übung und Anzahl der Geschwindigkeitsüberschreitungen ausgewählt werden, um Erfolge zu generieren.

Literaturverzeichnis

[1] Wohlers, K., Hombrecher, M.: TK-Stressstudie 2016 - Entspann dich, Deutschland. Techniker Krankenkasse (2016)

[2] Albrecht, U.V.: Chancen und Risiken von Gesundheits-Apps (CHARISMHA). Medizinische Hochschule Hannover (2016) http://www.digibib.tu-bs.de/?docid=00060000.

[3] Pryss, R., Reichert, M., John, D., Frank, J., Schlee, W., Probst, T.: A Personalized Sensor Support Tool for the Training of Mindful Walking. In: 2018 IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN). (2018) 114–117

[4] Heidenreich, T., Michalak, J.: Achtsamkeit («Mindfulness») als Therapieprinzip in Verhaltenstherapie und Verhaltensmedizin. Verhaltenstherapie **13** (2003) 264–274

[5] Sonnenmoser, M.: Mindfulness-basierte Therapie: Richtungsweisende Impulse. Deutsches Ärzteblatt International **4** (2005) 415–417

[6] Gotink, R.A., Hermans, K.S.F.M., Geschwind, N., De Nooij, R., De Groot, W.T., Speckens, A.E.M.: Mindfulness and mood stimulate each other in an upward spiral: a mindful walking intervention using experience sampling. Mindfulness **7** (2016) 1114–1122

[7] Fricchione, G.L., Teut, M., Roesner, E.J., Ortiz, M., Reese, F., Binting, S., Roll, S., Fischer, H.F., Michalsen, A., Willich, S.N., Brinkhaus, B.: Mindful Walking in Psychologically Distressed Individuals: A Randomized Controlled Trial. Evidence-Based Complementary and Alternative Medicine **2013** (2013) 489856

[8] Probst, T., John, D.: Mindful Walking Studie. Universität Ulm in Zusammenarbeit mit der evangelischen Hochschule Nürnberg und der Donau-Universität Krems (2020) Stand: März 2020.

[9] Signavio GmbH: Signavio Process Manager. https://www.signavio.com/de/products/process-manager/ (Abgerufen am 11.10.2020)

[10] Pryss, R., Schlee, W., Langguth, B., Reichert, M.: Mobile Crowdsensing Services for Tinnitus Assessment and Patient Feedback. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS). (2017) 22–29

[11] Track Your Tinnitus: Project Description. Institut für Datenbanken und Informationssysteme. https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/trackyourtinnitus/ (Abgerufen am 01.08.2020)

[12] Pryss, R., Probst, T., Schlee, W., Schobel, J., Langguth, B., Neff, P., Spiliopoulou, M., Reichert, M.: Prospective crowdsensing versus retrospective ratings of tinnitus variability and tinnitus-stress associations based on the TrackYourTinnitus mobile platform. International Journal of Data Science and Analytics **8** (2019) 327–338

[13] Pryss, R., John, D., Schlee, W., Schlotz, W., Schobel, J., Kraft, R., Spiliopoulou, M., Langguth, B., Reichert, M., O'Rourke, T., Peters, H., Pieh, C., Lahmann, C., Probst, T.: Exploring the Time Trend of Stress Levels While Using the Crowdsensing Mobile Health Platform, TrackYourStress, and the Influence of Perceived Stress Reactivity: Ecological Momentary Assessment Pilot Study (2019)

[14] Pryss, R., John, D., Reichert, M., Hoppenstedt, B., Schmid, L., Schlee, W., Spiliopoulou, M., Schobel, J., Kraft, R., Schickler, M., Langguth, B., Probst, T.: Machine Learning Findings on Geospatial Data of Users from the TrackYourStress mHealth Crowdsensing Platform. In: 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI). (2019) 350–355

[15] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)

[16] Chen, X., Ji, Z., Fan, Y., Zhan, Y.: Restful API Architecture Based on Laravel Framework. Journal of Physics: Conference Series **910** (2017) 012016

[17] Subramanian, H., Raj, P.: Hands-On RESTful API Design Patterns and Best Practices. Packt Publishing (2019)

[18] Huang, X.W., Hsieh, C.Y., Wu, C.H., Cheng, Y.C.: A Token-Based User Authentication Mechanism for Data Exchange in RESTful API. In: 2015 18th International Conference on Network-Based Information Systems. (2015) 601–606

[19] Dazer, M.: RESTful APIs - Eine Übersicht. Technical University Berlin (2012)

[20] Masse, M.: REST API Design Rulebook. O'Reilly Media, Inc. (2011)

[21] Costa, B., Pires, P.F., Delicato, F.C., Merson, P.: Evaluating a Representational State Transfer (REST) Architecture: What is the impact of REST in my architecture? In: 2014 IEEE/IFIP Conference on Software Architecture. (2014) 105–114

[22] Sturgeon, P.: Build APIs You Won't Hate: Everyone and their dog wants an API,so you should probably learn how to build one. Leanpub (2015)

[23] Google Inc.: Angular. `https://angular.io` (Abgerufen am 01.09.2020)

[24] Google Inc.: Angular Docs. `https://angular.io/docs` (Abgerufen am 11.07.2020)

[25] MDN web docs: SPA (Single-page application). `https://developer.mozilla.org/en-US/docs/Glossary/SPA` (Abgerufen am 30.07.2020)

[26] Schwarzmüller, M.: Angular - The Complete Guide 2020, Packt Publishing (2020)

[27] Fain, Y., Moiseev, A.: Angular Development with Typescript. 2 edn. Manning Publications (2018)

[28] Höller, C.: Angular: Das umfassende Handbuch. Rheinwerk Verlag (2019)

[29] Woiwode, G., Malcher, F., Koppenhagen, D., Hoppe, J.: Angular - Grundlagen, fortgeschrittene Techniken und Best Practices mit TypeScript. dpunkt.verlag (2018)

[30] Twitter Inc.: Bootstrap. `https://getbootstrap.com/` (Abgerufen am 12.07.2020)

[31] Krause, J.: Bootstrap kurz & gut. dpunkt.verlag (2018)

[32] ng-bootstrap: Bootstrap widgets - The angular way. `https://ng-bootstrap.github.io/#/home` (Abgerufen am 12.07.2020)

[33] MariaDB Foundation: MariaDB Server: The open source relational database. `https://mariadb.org/` (Abgerufen am 03.10.2020)

[34] Digital Guide IONOS by 1&1: MariaDB vs. MySQL. https://www.ionos.de/digitalguide/hosting/hosting-technik/mariadb-vs-mysql/ (Abgerufen am 03.10.2020)

[35] Throll, M.: MySQL 4. Galileo Press (2003)

[36] Vanier, E., Shah, B., Malepati, T.: Advanced MySQL 8. Packt Publishing (2019)

[37] XAMPP: Was ist XAMPP? https://www.apachefriends.org/de/index.html (Abgerufen am 03.08.2020)

[38] phpMyAdmin: Bringing MySQL to the web. https://www.phpmyadmin.net/ (Abgerufen am 03.08.2020)

[39] Digital Guide IONOS by 1&1: XAMPP-Tutorial: Installation und erste Schritte. https://www.ionos.de/digitalguide/server/tools/xampp-tutorial-so-erstellen-sie-ihren-lokalen-testserver/ (Abgerufen am 03.08.2020)

[40] Node.js: Node.js Docs. https://nodejs.org/en/docs/ (Abgerufen am 30.07.2020)

[41] Hughes-Croucher, T., Wilson, M.: Einführung in Node.JS. O'Reilly Verlag (2012)

[42] Springer, S.: Node.js: Das umfassende Handbuch. Galileo Press (2013)

[43] Nandaa, A.: Beginning API Development with Node.js. Packt Publishing (2018)

[44] Rauch, G.: Smashing Node.js: JavaScript Everywhere. 2 edn. Wiley (2012)

[45] Node.js: Introduction to Node.js. https://nodejs.dev/learn (Abgerufen am 12.07.2020)

[46] Express: Express - Fast, unopinionated, minimalist web framework for Node.js. https://expressjs.com/ (Abgerufen am 13.07.2020)

[47] Sequelize: Sequelize v5. https://sequelize.org/v5/ (Abgerufen am 14.07.2020)

[48] Chen, T.H., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M., Flora, P.: Detecting Performance Anti-Patterns for Applications Developed Using Object-Relational Mapping. In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014, New York, NY, USA, Association for Computing Machinery (2014) 1001–1012

[49] Git. `https://git-scm.com/` (Abgerufen am 03.08.2020)

[50] Digital Guide IONOS by 1&1: Git-Tutorial: Die ersten Schritte mit dem Versionskontrollsystem. `https://www.ionos.de/digitalguide/websites/web-entwicklung/git-tutorial/` (Abgerufen am 03.08.2020)

[51] GitLab. `https://gitlab.com/` (Abgerufen am 04.08.2020)

[52] JetBrains: Webstorm - Die raffinierteste JavaScript-IDE. `https://www.jetbrains.com/de-de/webstorm/` (Abgerufen am 03.08.2020)

[53] SmartBear Software: Swagger - API Development for Everyone. `https://swagger.io/` (Abgerufen am 04.08.2020)

[54] Scott, S.: Swagger UI Express. `https://www.npmjs.com/package/swagger-ui-express` (Abgerufen am 01.09.2020)

[55] Postman Inc.: The Collaboration Platform for API Development. `https://www.postman.com/` (Abgerufen am 04.08.2020)

[56] Balzert, H.: Nichtfunktionale Anforderungen. In: Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb, Heidelberg, Spektrum Akademischer Verlag (2011) 109–133

[57] Braun, M.: Nicht-funktionale Anforderungen. Juristisches IT-Projektmanagement. Lehrstuhl für Programmierung und Softwaretechnik (2016) Ludwig-Maximilians-Universität München.

[58] Bergsmann, J.: Requirements Engineering für die agile Softwareentwicklung. 2 edn. dpunkt.verlag (2018)

[59] Gumm, H.P., Sommer, M.: Einführung in die Informatik. 10 edn. De Gruyter Oldenbourg (2013)

[60] Microsoft Corporation: Microsoft Visio. https://www.microsoft.com/de-de/microsoft-365/visio/flowchart-software (Abgerufen am 01.09.2020)

[61] van de Moere, J.: Learn Angular: Build a Todo App. SitePoint (2018)

[62] JWT: Introduction to JSON Web Tokens. https://jwt.io/introduction/ (Abgerufen am 16.07.2020)

[63] Digital Guide IONOS by 1&1: JSON Web Token (JWT) vorgestellt. https://www.ionos.de/digitalguide/websites/web-entwicklung/json-web-token-jwt-vorgestellt/ (Abgerufen am 17.07.2020)

[64] Kelektiv: bcrypt. https://www.npmjs.com/package/bcrypt (Abgerufen am 01.09.2020)

[65] Auth0: jsonwebtoken. https://www.npmjs.com/package/jsonwebtoken (Abgerufen am 01.09.2020)

[66] Auth0: jwt-decode. https://www.npmjs.com/package/jwt-decode (Abgerufen am 01.09.2020)

[67] Hoque, S.: Full-Stack React Projects. 2 edn. Packt Publishing (2020)

[68] Uluca, D.: Angular for Enterprise-Ready Web Applications. 2 edn. Packt Publishing (2020)

[69] Bojinov, V.: RESTful Web API Design with Node.js 10. 3 edn. Packt Publishing (2018)

[70] Schwarzmüller, M.: Node.js - The Complete Guide, Packt Publishing (2019)

[71] express-validator: Introduction. https://express-validator.github.io/docs/ (Abgerufen am 26.07.2020)

[72] Chart.js: Chart.js - Simple yet flexible JavaScript charting for designers & developers . https://www.chartjs.org/ (Abgerufen am 02.09.2020)

[73] Google Inc.: Material Design - Icons. https://material.io/resources/icons/?style=baseline (Abgerufen am 07.10.2020)

[74] Mirco Zeiss: json2csv. https://www.npmjs.com/package/json2csv (Abgerufen am 03.10.2020)

[75] Grey, E.: FileSaver.js. https://www.npmjs.com/package/file-saver (Abgerufen am 07.10.2020)

[76] Reinman, A.: Nodemailer. https://nodemailer.com/about/ (Abgerufen am 03.10.2020)

[77] Kazakov, V.: nodemailer-express-handlebars. https://www.npmjs.com/package/nodemailer-express-handlebars (Abgerufen am 07.10.2020)

[78] Sussex Partnership NHS Foundation Trust. Worthing in West Sussex, England (2016) https://www.sussexpartnership.nhs.uk/sites/default/files/documents/jenny_gus_short_ffmq-15_june_16.pdf.

[79] Chowdhury, M.R.: The Five Facet Mindfulness Questionnaire (FFMQ). https://positivepsychology.com/five-facet-mindfulness-questionnaire-ffmq/ (Abgerufen am 11.10.2020)

[80] Schickler, M., Pryss, R., Reichert, M., Schobel, J., Langguth, B., Schlee, W.: Using Mobile Serious Games in the Context of Chronic Disorders: A Mobile Game Concept for the Treatment of Tinnitus. In: 2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS). (2016) 343–348

[81] Baer, R.A., Smith, G.T., Allen, K.B.: Assessment of Mindfulness by Self-Report: The Kentucky Inventory of Mindfulness Skills. Assessment **11** (2004) 191–206

[82] Gao, L., Zhang, C., Sun, L.: RESTful Web of Things API in Sharing Sensor Data. In: 2011 International Conference on Internet Technology and Applications. (2011) 1–4

[83] Hapke, U., Maske, U.E., Scheidt-Nave, C., Bode, L., Schlack, R., Busch, M.A.: Chronischer Stress bei Erwachsenen in Deutschland. Bundesgesundheitsblatt - Gesundheitsforschung - Gesundheitsschutz **56** (2013) 749–754

A

Anhang

A.1 Five-Facet Mindfulness Questionnaire

Der nachfolgende Fragebogen *Five-Facet Mindfulness Questionnaire* [78, 79] wurde für die Mindful Walking Studie der Universität Ulm, die in der vorliegenden Arbeit beschrieben wurde, eingesetzt.

FFMQ-15: 15-item Five-Facet Mindfulness Questionnaire

Instructions

Please use the 1 (never or very rarely true) to 5 (very often or always true) scale provided to indicate how true the below statements are of you. Circle the number in the box to the right of each statement which represents your own opinion of what is generally true for you. For example, if you think that a statement is often true of you, circle '4' and if you think a statement is sometimes true of you, circle '3'.

	Never or very rarely true	Rarely true	Sometimes true	Often true	Very often or always true
1. When I take a shower or a bath, I stay alert to the sensations of water on my body.	1	2	3	4	5
2. I'm good at finding words to describe my feelings.	1	2	3	4	5
3. I don't pay attention to what I'm doing because I'm daydreaming, worrying, or otherwise distracted.	1	2	3	4	5
4. I believe some of my thoughts are abnormal or bad and I shouldn't think that way.	1	2	3	4	5
5. When I have distressing thoughts or images, I "step back" and am aware of the thought or image without getting taken over by it.	1	2	3	4	5
6. I notice how foods and drinks affect my thoughts, bodily sensations, and emotions.	1	2	3	4	5
7. I have trouble thinking of the right words to express how I feel about things.	1	2	3	4	5
8. I do jobs or tasks automatically without being aware of what I'm doing.	1	2	3	4	5
9. I think some of my emotions are bad or inappropriate and I shouldn't feel them.	1	2	3	4	5
10. When I have distressing thoughts or images I am able just to notice them without reacting.	1	2	3	4	5
11. I pay attention to sensations, such as the wind in my hair or sun on my face.	1	2	3	4	5
12. Even when I'm feeling terribly upset I can find a way to put it into words.	1	2	3	4	5
13. I find myself doing things without paying attention.	1	2	3	4	5
14. I tell myself I shouldn't be feeling the way I'm feeling.	1	2	3	4	5
15. When I have distressing thoughts or images I just notice them and let them go.	1	2	3	4	5

Baer, R. A., Carmody, J., & Hunsinger, M. (2012). Weekly change in mindfulness and perceived stress in a mindfulness-based stress reduction program. *Journal of Clinical Psychology, 68*(7), 755-765. doi: 10.1002/jclp.21865

Gu, J., Strauss, C., Crane, C., Barnhofer, T., Karl, A., Cavanagh, K., & Kuyken, W. (2016). Examining the factor structure of the 39-item and 15-item versions of the Five Facet Mindfulness Questionnaire before and after mindfulness-based cognitive therapy for people with recurrent depression. *Psychological assessment, 28*(7), 791. doi: 10.1037/pas0000263

Abbildung A.1: Five-Facet Mindfulness Questionnaire - Teil eins

Background

This measure is a short form of the 39-item FFMQ (Baer et al., 2006). It includes the same five facets as the long form: Observing, Describing, Acting with Awareness, Non-Judging of inner experience, and Non-Reactivity to inner experience. The 15-item FFMQ (FFMQ-15) was developed by Baer et al. (2012) and includes three items for each facet. Items were selected from the FFMQ-39 based on their loadings on each facet and to maintain the breadth of content for each facet. The factor structure and psychometric properties of the FFMQ-15 were tested by Gu et al. (2016). They found that the factor structure of the FFMQ-15 was consistent with that of the FFMQ-39 and there were large correlations between total facet scores of the short and long forms. This indicates that both versions of the FFMQ measured highly similar constructs. They also found that the two FFMQ versions did not differ significantly from each other in terms of convergent validity. Additionally, internal consistency was adequate for the FFMQ-15 and the measure was found to be sensitive to change over the course of Mindfulness-Based Cognitive Therapy (small/moderate to moderate/large and significant increases in total facet scores). Taken together, Gu et al.'s findings support the use of the FFMQ-15 as an alternative measure in research where briefer forms are needed.

In addition to the above findings, Gu et al. (2016) found that the factor structures of the FFMQ-39 and FFMQ-15 were not stable before and after MBCT; for both versions, before MBCT a four-factor structure without the observing subscale best fit the data but after MBCT a five-factor structure provided the best fit. This suggests that comparisons in FFMQ scores before and after mindfulness interventions may not be valid. They recommend that researchers consider excluding the observing facet score from comparisons of total scale/subscale scores before and after mindfulness interventions.

Scoring Information

*Observing items: 1, 6, 11.
Describe items: 2, 7R, 12.
Acting with awareness items: 3R, 8R, 13R.
Non-judging items: 4R, 9R, 14R.
Non-reactivity items: 5, 10, 15.

Reverse-phrased items are denoted by 'R' after the item number, e.g. 14R.

*Refer to the background information regarding recommendations for omitting the observing subscale score from comparisons of total scale/subscale scores before and after mindfulness interventions.

References

The original FFMQ was developed by Baer et al. (2006) and the FFMQ-15 was developed by Baer et al. (2012). The factor structure and psychometric properties of the FFMQ-15 was tested by Gu et al. (2016).

Baer, R. A., Smith, G. T., Hopkins, J., Krietemeyer, J., & Toney, L. (2006). Using self-report assessment methods to explore facets of mindfulness. *Assessment, 13*(1), 27–45. doi: /10.1177/1073191105283504

Baer, R. A., Carmody, J., & Hunsinger, M. (2012). Weekly change in mindfulness and perceived stress in a mindfulness-based stress reduction program. *Journal of Clinical Psychology, 68*(7), 755-765. doi: 10.1002/jclp.21865

Gu, J., Strauss, C., Crane, C., Barnhofer, T., Karl, A., Cavanagh, K., & Kuyken, W. (2016). Examining the factor structure of the 39-item and 15-item versions of the Five Facet Mindfulness Questionnaire before and after mindfulness-based cognitive therapy for people with recurrent depression. *Psychological assessment, 28*(7), 791. doi: 10.1037/pas0000263

Abbildung A.2: Five-Facet Mindfulness Questionnaire - Teil zwei

A.2 Dokumentation der Schnittstelle mit Swagger UI

Abbildung A.3: Mindful Walking API Dokumentation mit Swagger UI - Teil eins [53]

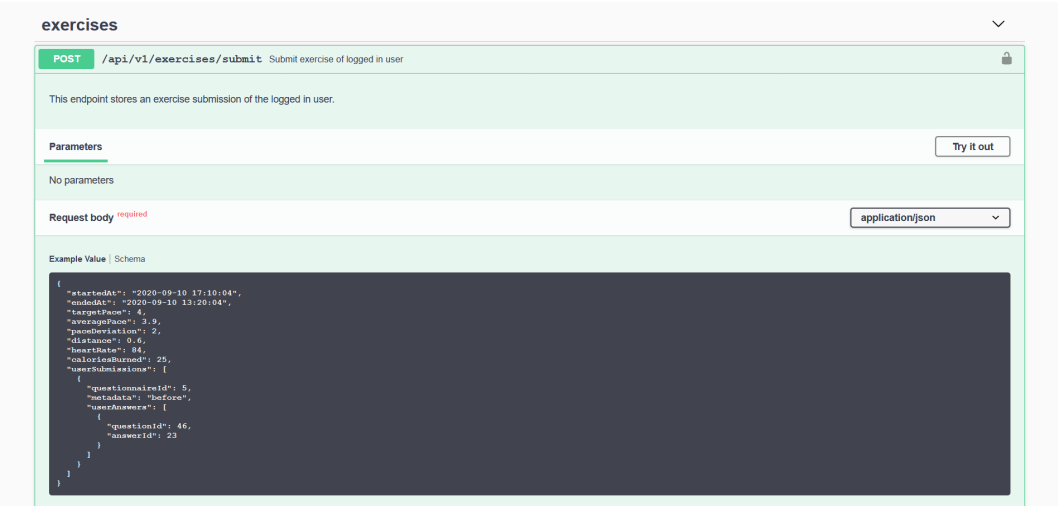

Abbildung A.4: Mindful Walking API Dokumentation mit Swagger UI - Teil zwei [53]

Abbildungsverzeichnis

Listings

Name: Verena Pfaff Matrikelnummer: 839225

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Verena Pfaff