ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Diplomarbeit
im Studiengang Informatik

Konzeption und technische Realisierung eines mobilen Frameworks zur Unterstützung tinnitusgeschädigter Patienten

vorgelegt von

Jochen Herrmann

März 2014

1. Gutachter	Prof. Dr. Manfred Reichert
2. Gutachter	Dr. Winfried Schlee
Betreuer:	Dipl. Inf. Rüdiger Pryss
Matrikelnummer	640062
Arbeit vorgelegt am:	19. März 2014

Kurzfassung

Tinnitus bezeichnet ein Symptom, bei dem der Betroffene Töne oder Geräusche ohne physikalische Ursache wahrnimmt. Bei 60% der Tinnituspatienten schwankt die subjektive Lautstärke des Tinnitus innerhalb und zwischen den Tagen erheblich. Der genaue Verlauf dieser Schwankungen sowie dessen Ursachen sind bislang nicht bekannt. Bisher werden diese Schwankungen in der Tinnitusforschung nur durch sogenannte Tinnitustagebücher erfasst, bei denen der Patienten handschriftlich den Verlauf des Tinnitus festhält. Dabei ist eine exakte Aufzeichnung des zeitlichen Verlaufs nicht möglich. Außerdem lassen sich so die Schwankungen nur schwer mit anderen Tätigkeiten, der Umgebungslautstärke oder stressigen Situationen im Alltag eines Tinnitusgeschädigten in Verbindung bringen. Um in der Forschung, aber auch den Betroffenen selbst, eine Möglichkeit zu bieten, diese Schwankungen zu überwachen, wurde das Track Your Tinnitus Projekt entwickelt.
Diese Arbeit zeigt, wie die Schwankungen der Tinnituswahrnehmung auf Smartphones überwacht und visualisiert werden können. Hierfür wurde eine Webseite, sowie eine App für iOS und Android entwickelt. Diese drei Bereiche werden im Detail vorgestellt, aber auch aus technischer Sicht betrachtet. Es wird gezeigt, wie ein solches Projekt architektonisch zu realisieren ist. Außerdem erklärt diese Arbeit spezielle Implementierungsthemen, wie das Senden von Benachrichtigungen in den Apps, sowie das Erstellen von eigenen User-Interface Elementen.
Am Ende entstand ein voll funktionsfähiges mobiles Framework zur Unterstützung tinnitusgeschädigter Patienten, das in dieser Form auch auf andere Bereiche (wie zum Beispiel der Migräneforschung oder chronischem Schmerz) übertragen werden kann. Es dient nicht nur der Forschung, sondern auch den Betroffenen die Schwankungen der Tinnituswahrnehmung besser zu verstehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Diese Kapitel beschreibt zuerst, was ein Tinnitus ist und wie hoch die Verbreitung in der Bevölkerung ist. Danach wird beschrieben, wie das Track Your Tinnitus Projekt [Sch13a] in diesem Zusammenhang sinnvoll ist. Am Ende wird noch ein kurzer Überblick über den Aufbau dieser Arbeit gegeben.

1.1. Tinnitus

Tinnitus bezeichnet ein Symptom, bei dem der Betroffene Töne oder Geräusche wahrnimmt, ohne dass in der unmittelbaren Umgebung eine physikalische Ursache für das akustische Signal zu finden ist. Diese Art von Tinnitus nennt man subjektiver Tinnitus. Im Gegensatz dazu nimmt der Betroffene beim objektiven Tinnitus Geräusche oder Töne war, die in seinem Körper entstehen und so von außen wahrnehmbar bzw. messbar sind. Im Vergleich zum subjektiven Tinnitus tritt der objektive Tinnitus nur sehr selten auf. Ein subjektiver Tinnitus tritt in den unterschiedlichsten Formen auf: Von kaum wahrnehmbaren Geräuschen oder Tönen, bis hin zu einer so starken Beeinflussung des Lebens eines Betroffenen, welche bis zum Selbstmord führen kann. Es gibt aktuell keine objektiven Methoden, um den subjektiven Tinnitus festzustellen oder zu messen. Nur ein Betroffener kann über den Tinnitus berichten. Dabei kann ein Tinnitus nur in einem Ohr, in beiden Ohren, oder im Inneren des Kopfes wahrgenommen werden [Lan07].
Subjektiver Tinnitus tritt häufig in Verbindung mit Gehörverlust, wie zum Beispiel nach einem Knalltrauma, aber auch mit zunehmenden Hörverlust im Alter auf. In den meisten Fällen kann allerdings keine Ursache gefunden werden [Mol07]. Leidet ein Betroffener länger als 6 Monate unter einem Tinnitus spricht man von einem chronischen Tinnitus. Nach verschiedenen Studien nimmt die Häufigkeit eines Tinnitus mit dem Alter zu und liegt zwischen 5% im Alter von 20 - 30 Jahren, bis zu 12% im Alter von 50 Jahren oder älter [Lan07]. In der Bundesrepublik Deutschland leiden knapp 10% an einem dauerhaften Tinnitus [Bun09].

1.2. Motivation

Die Wahrnehmung des Tinnitus kann selbst bei einem chronischen Tinnitus im Laufe des Tages oder auch zwischen den Tagen schwanken. Grund für diese Schwankungen können Umgebungsgeräusche, die Tageszeit, die aktuelle Tätigkeit, stressig empfundene Lebensereignisse oder vieles mehr sein. Da ein Tinnitus nur vom Betroffenen beschrieben werden kann, ist eine genaue Messung dieser Schwankungen nur schwer möglich. Viele Betroffene können allerdings eine gute Einschätzung dieser Schwankungen abgeben und den Zeitverlauf aus dem Gedächtnis rekonstruieren. Eine systematische Erfassung der Tinnituswahrnehmung und ihrer Schwankungen über mehrere Wochen war bislang noch nicht möglich [Sch13c].
Mit dem Track Your Tinnitus Projekt möchten wir es den Betroffenen ermöglichen, die individuellen Schwankungen ihrer Wahrnehmung des Tinnitus zu überwachen. Dadurch ist es möglich, die Schwankungen festzustellen und mit dem Tagesablauf und Tätigkeiten eines Betroffenen in Zusammenhang zu bringen. Außerdem kann ein Betroffener die erfassten Daten mit seinem behandelden Arzt oder Therapeuten besprechen [Sch13c]. Hierfür haben wir eine Webseite und eine App für iOS und Android entwickelt. Die Webseite dient dabei als Informationsquelle für neue Benutzer und bietet darüber hinaus noch weitere Funktionen (siehe Kapitel 4). Das Abfragen der Schwankungen der Tinnituswahrnehmung findet nur in den Apps statt. Ein Benutzer wird dabei in unregelmäßigen Abständen daran erinnert einen kurzen Fragebogen auszufüllen. Diese Methode nennt man auch “Experience Sampling” oder “Experience Sampling Method” und geht auf die Forscher Larson und Csikszentmihalyi zurück [Lar83].
Es gab bereits eine Studie, in der eine Anwendung auf einem Personal-Digital-Assistant (kurz PDA) entwickelt wurde. Diese benachrichtigte die Teilnehmer vier Mal am Tag und forderte diese auf, einen Fragebogen mit 19 Fragen zu beantworten. Dabei mussten den Teilnehmern die entsprechenden Geräte zur Verfügung gestellt werden. Auch die Auswertung der Daten erfolgte nach Rückgabe der Geräte. Die Benachrichtigungszeiten konnten vom Benutzer nicht selbst eingestellt werden. Das Gerät plante jeden Tag vier Benachrichtigungen in der Zeit von 8 - 20 Uhr [Hen12].

1.3. Aufbau dieser Arbeit

Dieser Abschnitt stellt eine Übersicht der Struktur dieser Arbeit dar. Diese Arbeit ist in sieben Kapitel unterteilt. Nach dieser Einleitung folgt zuerst die Darstellung der Anforderungen an das Track Your Tinnitus Projekt (Kapitel 2). Im Anschluss folgt ein Kapitel über die Architektur des Projekts (Kapitel 3). In diesem Kapitel wird eine generelle Übersicht der Architektur gegeben, die Datenstruktur erklärt und die Architektur und den Aufbau der Apps erläutert. Kapitel 4 stellt das Projekt vor und beschreibt die Webseite, sowie der Apps. Hierbei wird beschrieben, welche Funktionen die Webseite bzw. die Apps einem Benutzer bieten und wie ein Benutzer diese Funktionen bedienen kann. Danach folgt ein Kapitel über die Implementierung des Projekts (Kapitel 5). Dabei wird auf spezielle Probleme in der Entwicklung eingegangen. Der erste Abschnitt dieses Kapitels beschreibt einige Themen, die bei der Entwicklung der Webseite interessant waren. Der zweite Abschnitt befasst sich mit der Implementierung der Apps. Hier wird auch dargestellt, wie Benachrichtigungen auf den Plattformen iOS und Android gesendet werden und wie diese zeitlich verteilt werden. Außerdem zeigt dieses Kapitel, wie eigene User-Interface Elemente (einfache Darstellungen, sowie interaktive Elemente) auf

den Plattformen iOS und Android implementiert werden. Im Anschluss werden die Anforderungen mit dem Stand der Entwicklung abgeglichen (Kapitel 6). Das letzte Kapitel beinhaltet die Zusammenfassung und einen Ausblick auf die mögliche weiter Entwicklung des Projekts (Kapitel 7).

2 Anforderungen

Diese Kapitel definiert die Anforderungen an das Track Your Tinnitus Projekt. Dabei sind diese Anforderungen in funktionale und nichtfunktionale Anforderungen unterteilt.

2.1. Funktionale Anforderungen

Dieser Abschnitt zeigt die funktionalen Anforderungen an die Webseite bzw. die Apps. Dabei werden die wichtigsten Funktionen der Webseite bzw. der Apps gezeigt, die diese einem Benutzer bieten sollten. Die folgende Tabelle zeigt eine Aufstellung der funktionalen Anforderungen.

Nr.	**Beschreibung**	**Problembeschreibung**
1.	Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung	Die Apps sollten einen Fragebogen anzeigen, mit dem ein Benutzer in unregelmäßigen Abständen die Schwankungen der Tinnituswahrnehmung überwachen kann.
2.	Einstellungen der Benachrichtigungszeiten	Ein Benutzer sollte die Benachrichtigungszeiten in den Apps einstellen können, so dass die Benachrichtigungen nach den Wünschen eines Benutzers erscheinen.
3.	Einstellung des Klingeltons einer Benachrichtigung	Da der Klingelton einer Benachrichtigung in manchen Fällen vom Tinnitus überdeckt werden kann, sollte einem Benutzer die Möglichkeit gegeben werden, den Klingelton der Benachrichtigung zu ändern.

Nr.	Beschreibung	Problembeschreibung
4.	Anzeige der Ergebnisse in den Apps	Um die zeitliche Entwicklung in der App direkt anzeigen zu können, sollten die Ergebnisse aus dem Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung visualisiert werden.
5.	Registrierung in den Apps ermöglichen	Die Benutzung der Apps kann ohne Benutzerkonto nicht erfolgen. Es sollte allerdings möglich sein, direkt auf einem Gerät ein solches Benutzerkonto zu erstellen.
6.	Slider ohne initialen Wert	Ein Benutzer lässt sich beim Ausfüllen eines Fragebogens davon beeinflussen, welcher Wert voreingestellt ist (siehe dazu Kapitel 5.4.1). Daher darf der Slider in einem Fragebogen in den Track Your Tinnitus Apps keinen initialen Wert haben.
7.	Messung des Geräuschpegels	Um herauszufinden, ob der Tinnitus von den Umgebungsgeräuschen überdeckt oder beeinflusst wird, sollten die Apps während des Ausfüllens des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung den Pegel der Hintergrundgeräusche messen.
8.	Apps auch ohne Internetverbindung benutzbar.	Eine funktionierende Internetverbindung auf dem Smartphone sollte keine Voraussetzung für das Benutzen der Apps sein, da ein Benutzer evtl. nur schlechten oder gar keinen Empfang haben kann.
9.	Ausfüllen der statistischen Fragebögen auf der Webseite und in den Apps	Da das Ausfüllen der statistischen Fragebögen Voraussetzung für die Benutzung der Apps ist, sollte dies auf der Webseite und in den Apps möglich sein.
10.	Erweiterbarkeit der statistischen Fragebögen	Die statistischen Fragebögen sollten auf der Webseite von bestimmten Benutzern veränderbar sein. Es sollte möglich sein, neue statistische Fragebögen einzupflegen, oder auch bestehende zu bearbeiten.
11.	Synchronisierung der Antworten auf die statistischen Fragebögen	Es sollte möglich sein, auch während des Ausfüllens eines Fragebogens auf der Webseite zur App zu wechseln, um dort die Bearbeitung des Fragebogens zu beenden.

Nr.	Beschreibung	Problembeschreibung
12.	Synchronisierung der Ergebnisse	Zur Visualisierung der Ergebnisse aus dem Fragebogen zur Überwachung der Tinnituswahrnehmung und für Forschungszwecke, sollten diese Ergebnisse aus den Apps an den Server übertragen werden.
13.	Exportfunktion der Daten eines Benutzers	Ein Benutzer sollte die Möglichkeit haben, seine Daten (Antworten auf den Fragebogen zur Überwachung der Tinnituswahrnehmung) auf der Webseite und in den Apps zu exportieren, um sie zum Beispiel seinem Arzt zur Verfügung zu stellen.

2.2. Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen definieren die Anforderungen an das Projekt im Hinblick auf Aussehen, Handhabung und Datenschutz. Die folgende Tabelle zeigt die nichtfunktionalen Anforderungen an das Track Your Tinnitus Projekt.

Nr.	Beschreibung	Problembeschreibung
1.	Ausfüllen des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung sollte in weniger als einer Minute möglich sein.	Das Ausfüllen des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung wird vom Benutzer mehrmals am Tag ausgefüllt. Daher sollte dieser Vorgang möglichst schnell zu erledigen sein.
2.	Identisches Aussehen der Apps unter iOS und Android.	Die App sollte unter Android und iOS identisch aussehen und sich gleich bedienen lassen, so dass sich ein Benutzer auch bei einem Wechsel des Smartphones nicht erneut mit dem Bedienen der App auseinandersetzen muss.
3.	Anpassung des Designs der iOS App an iOS 6 und iOS 7	Die iOS Version der Track Your Tinnitus App unterstützt iOS in der Version 6 und 7. Da sich das Design der Benutzeroberfläche von iOS 6 zu iOS 7 grundlegend geändert hat, muss dies in der App ebenfalls berücksichtigt werden.

Nr.	Beschreibung	Problembeschreibung
4.	Keine Speicherung der E-Mail Adresse oder IP-Adresse	Aufgrund von rechtlichen Restriktionen muss es dem Benutzer überlassen sein, ob er bei der Registrierung seine E-Mail Adresse speichert. Generell darf die IP-Adresse eines Benutzers, weder beim Zugriff über die Webseite, noch beim Zugriff über die Apps, gespeichert werden.

High performance has never been so well defined.

Apple's Marketing Slogan 2013

3 Architektur

Dieses Kapitel beschreibt die Architektur des Track Your Tinnitus Projekts. Zuerst wird eine kurze Übersicht über die Architektur gegeben (Kapitel 3.1). In Kapitel 3.2 wird ein typischer Ablauf gezeigt, gefolgt von der detaillierten Beschreibung der Datenstruktur (Kapitel 3.3). Am Ende des Kapitels wird die Architektur der Apps erklärt (Kapitel 3.4).

3.1. Architekturübersicht

Track Your Tinnitus besteht aus zwei Komponenten: Server und Apps für Smartphones. Der Server verwendet Linux als Betriebssystem, Apache als Webserver, MySQL als Datenbank und PHP als Skriptsprache. Die Webanwendung, sowie die API, wurden mithilfe des Laravel Frameworks entwickelt [Otwa]. Die Apps wurden jeweils nativ entwickelt (ObjC auf iOS und Java auf Android). Abbildung 3.1 zeigt den Überblick und die Verknüpfungspunkte der einzelnen Komponenten. Die Apps greifen nicht direkt auf die Daten auf dem Server zu. Jegliche Kommunikation zwischen App und Webanwendung läuft über eine REST-ähnliche JSON-API [Fie00].

3.2. Genereller Ablauf

Die Überwachung der Schwankungen der Tinnituswahrnehmung in den Apps setzt ein Benutzerkonto voraus. Track Your Tinnitus bietet einem Benutzer die Möglichkeit, entweder ein Benutzerkonto auf der Webseite oder in der App zu erstellen. Abbildung 3.2 zeigt einen typischen Ablauf. Wenn ein Benutzer zuerst auf die Webseite gelangt, muss er sich zunächst registrieren. Nach der Eingabe aller erforderlichen Daten (Benutzername, E-Mail Adresse und Passwort) erhält ein Benutzer eine Bestätigung per E-Mail. In dieser E-Mail befindet sich ein Link, mit dem er sein Benutzerkonto aktivieren kann. Erst nach erfolgreicher Aktivierung ist eine Anmeldung möglich. Im Benutzerbereich findet ein Benutzer alle statistischen Fragebögen. Nachdem

Abbildung 3.1.: Übersicht der Architektur

alle statistischen Fragebögen ausgefüllt sind, werden die Links zu den Apps angezeigt. Weitere Literatur zu Fragebögen im psychologischen Kontext kann hier gefunden werden [Cro13] [Ise13] [Ruf13] [Schon].
Ein Benutzer kann allerdings auch direkt im Apple App Store, oder Google Play Store, die App herunterladen. Daher bietet die App ebenfalls die Möglichkeit, ein Benutzerkonto zu registrieren. Nach der Bestätigung der E-Mail Adresse kann sich ein Benutzer dann in der App anmelden. Wenn bereits ein Benutzerkonto vorhanden ist, ist ein direkter Login möglich. Die App überprüft, ob alle statistischen Fragebögen bereits beantwortet sind. Wenn nicht, müssen diese in der App zwingend ausgefüllt werden, da sonst eine Überwachung der Schwankungen der Tinnituswahrnehmung nicht möglich ist. Danach kann ein Benutzer auswählen, ob er die vorgegebenen Benachrichtigungseinstellungen (siehe Kapitel 4.2.6) überprüfen möchte. Wenn er diese überprüfen möchte, werden die Benachrichtigungseinstellungen direkt angezeigt, ansonsten wird der Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung

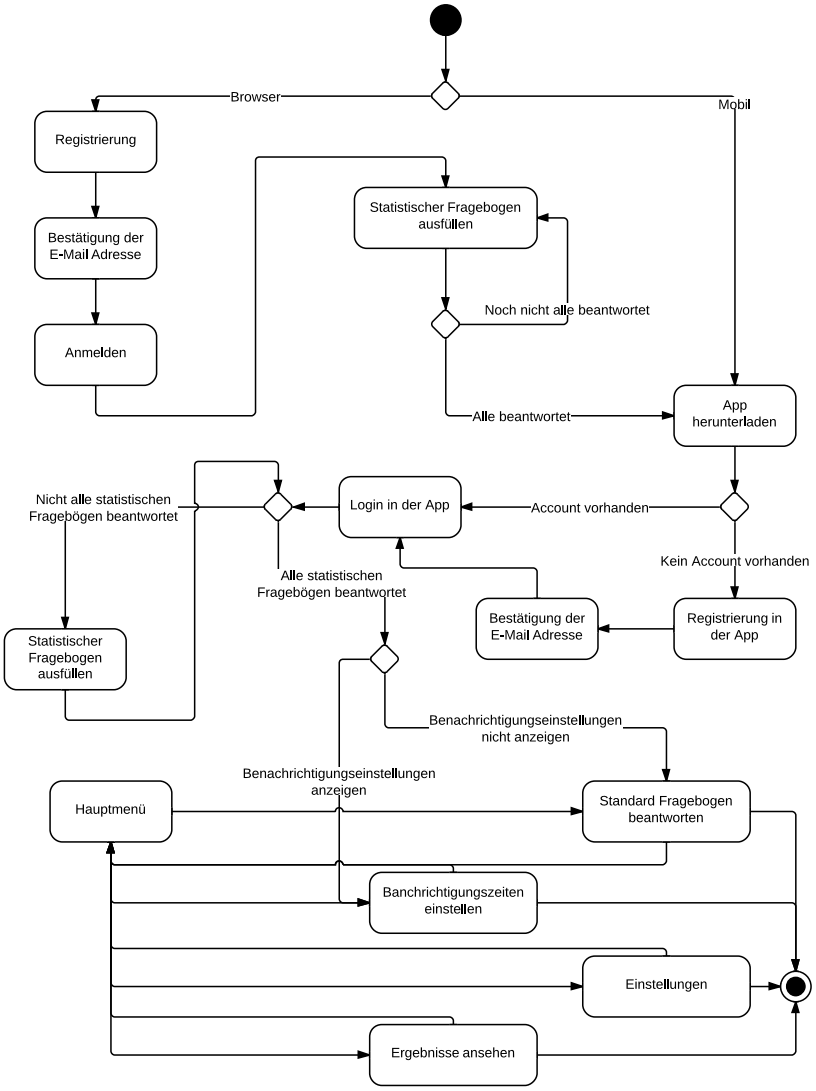

Abbildung 3.2.: Ablaufdiagramm

(siehe Kapitel 4.2.5) geöffnet. Nun lässt sich auch das Hauptmenü öffnen, mit dem ein Benutzer zu den weiteren Bereichen der App navigieren kann. So lassen sich in der App die Benachrichtigungszeiten einstellen, generelle Einstellungen (Klingelton der Benachrichtigung und Privatsphäreeinstellungen) vornehmen (siehe Kapitel 4.2.7), oder die Ergebnisse anzeigen (siehe Kapitel 4.2.8).

3.3. Datenstrukturen

Track Your Tinnitus basiert auf einem relationalen Datenmodell, dass auf dem Server in MySQL umgesetzt wurde. Die statistischen Fragebögen können im Administrationbereich der Webseite verwaltet werden. Daher müssen diese, mit den entsprechenden Antworten eines Benutzers, auch in der Datenstruktur abgebildet sein. Ebenso werden alle Daten zur Benutzerverwaltung und

Authentifizierung in der Datenbank gespeichert. Bei der Überwachung der Schwankungen der Tinnituswahrnehmung werden nur die Antworten eines Benutzers gespeichert, die Fragen selbst sind in den Apps vorgegeben. Abbildung 3.3 zeigt eine Übersicht der Datenstruktur, die in den folgenden Abschnitten näher beschrieben wird. In diesem Entity-Relationship Diagram wurden allerdings alle Felder entfernt, die keinen Primärschlüssel oder Fremdschlüssel darstellen.

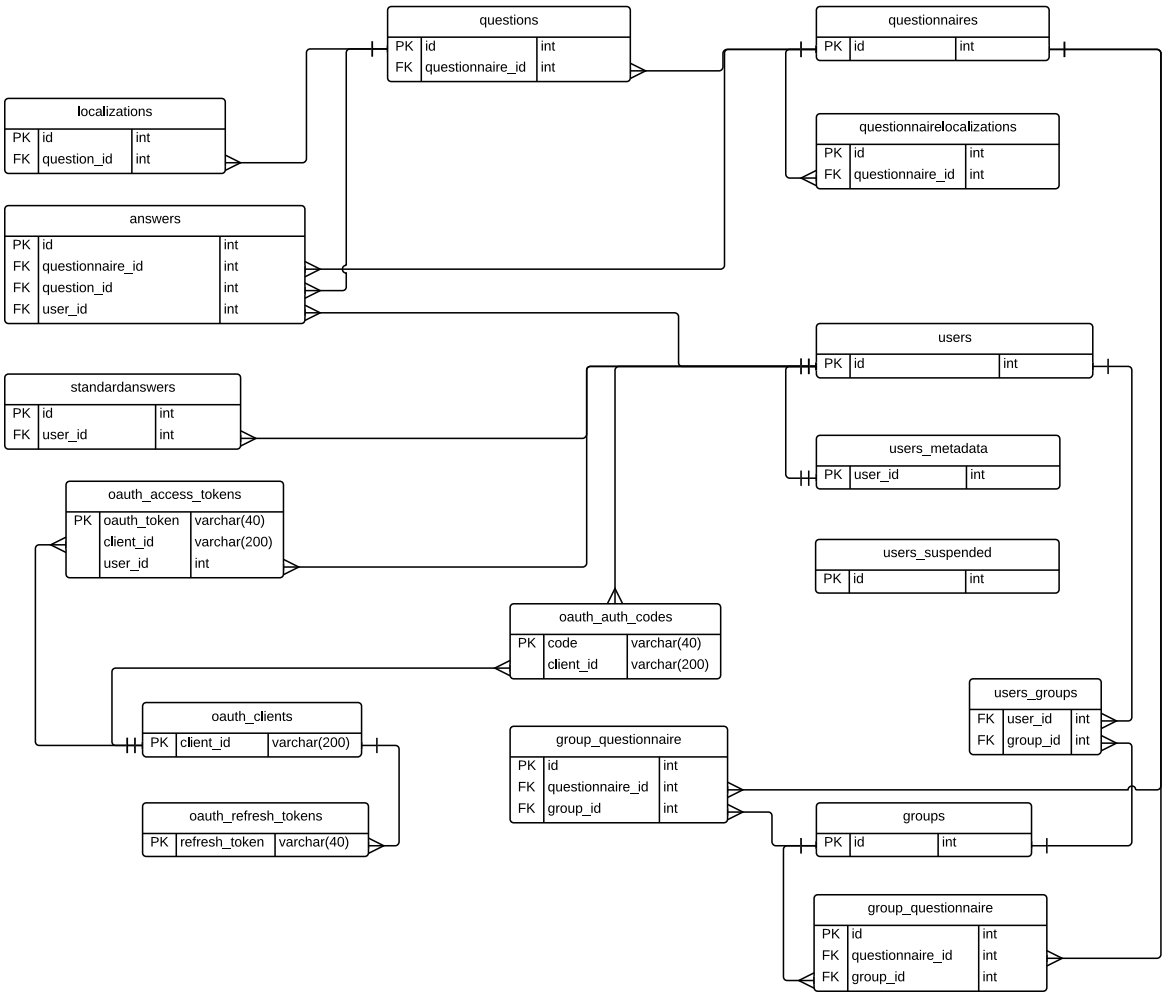

Abbildung 3.3.: Übersicht der Datenstruktur

3.3.1. Datenstruktur der Fragebögen, Fragen und Antworten

Abbildung 3.4 zeigt die Datenstruktur der Fragebögen, Fragen und Antworten mit der Referenz auf die Benutzertabelle. Jeder Fragebogen ist in zwei Tabellen aufgeteilt. Die Tabelle *questionnaires* enthält alle Daten, die einen Fragebogen beschreiben. Die Tabelle *questions* beinhaltet die eigentlichen Fragen, die einem Fragebogen zugeordnet sind. Ein Fragebogen hat immer einen Titel und eine Beschreibung, außerdem ein Flag, mit dem sich ein Fragebogen als öffentlich markieren lässt. Dieses Flag dient dazu, einen Fragebogen erst zu veröffentlichen, wenn alle zugehörigen Fragen eingepflegt sind. Das weitere Flag *one_ answer* gibt an, ob die

Fragen eines Fragebogens nur einen Antworttyp haben. So muss nicht bei jeder Frage einzeln der Antworttyp und die Antwortkonfiguration angegeben werden: *type* und *configuration* beschreiben diesen Antworttyp und diese Antwortkonfiguration. Wenn *one_ answer* gesetzt ist, muss der Antworttyp und die Antwortkonfiguration in dem Datensatz des Fragebogens angegeben werden, ansonsten zu jeder Frage einzeln. Neben den Zeitstempeln, wann ein Eintrag erstellt und zuletzt bearbeitet wurde, gibt das *repeat_ interval* an, nach welcher Zeit ein Fragebogen erneut beantwortet werden kann.

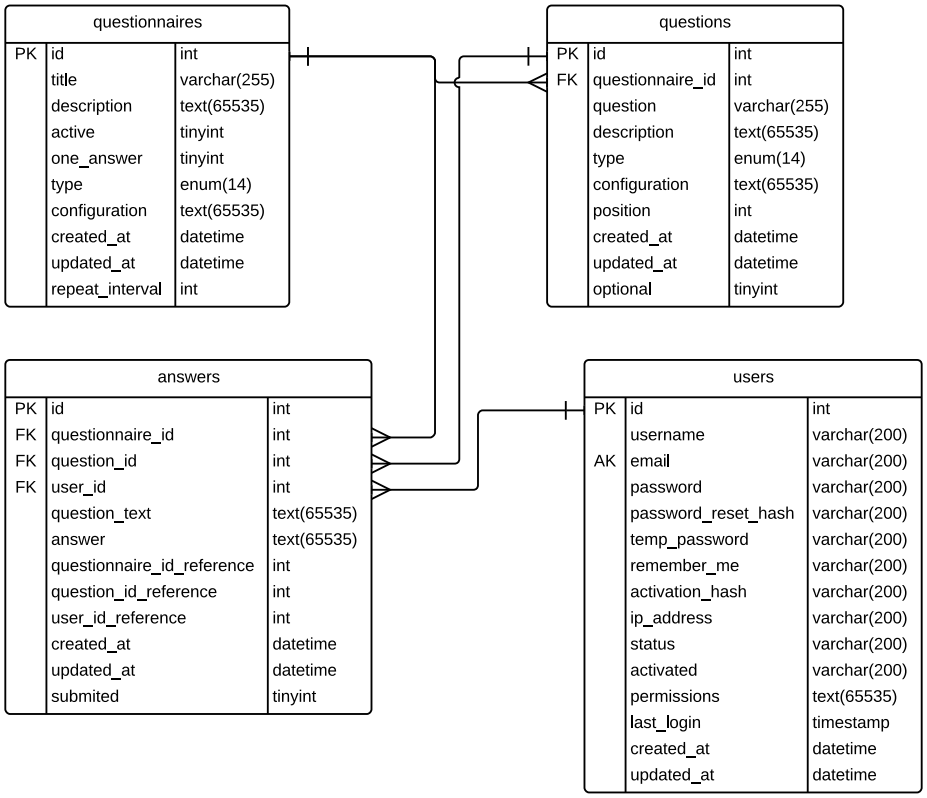

Abbildung 3.4.: Datenstruktur der Fragebögen

Die Fragen beinhalten immer eine Referenz auf den zugehörigen Fragebogen. Ein Datensatz besteht aus der Frage und eine optionalen Beschreibung. Analog zu den Feldern des Fragebogens definieren *type* und *configuration* den Antworttyp sowie die Antwortkonfiguration. Die Position innerhalb eines Fragebogens wird anhand des Feldes *position* angegeben.
Die Antworten, die ein Benutzer auf die Fragen eines Fragebogens gibt, werden in der Tabelle *answers* gespeichert. Jede Antwort hat eine Referenz auf den Fragebogen und die Frage, sowie auf den Benutzer, von dem diese Antwort stammt. Diese drei Referenzen werden zusätzlich ein weiteres Mal gespeichert, um diese auch zu erhalten, wenn ein zugehöriger Datensatz in den referenzierten Tabellen gelöscht wird. Neben der Antwort als Text wird zusätzlich auch der Wortlaut der Frage gespeichert. So lässt sich eine vollständige Versionierung der Fragebögen vermeiden.

3.3.2. Datenstruktur der Lokalisierung der Fragebögen

Wie in Abbildung 3.5 gezeigt, gibt es zu den Tabellen *questionnaires* und *questions* (siehe Kapitel 3.3.1) jeweils eine Tabelle für die lokalisierten Daten. Die Tabelle *questionnaireslocalizations* hält die übersetzten Daten eines Fragebogens. Jeder Eintrag wird mit dem *languagecode* nach dem ISO 639-1 Standard [Int11] einer Sprache zugeordnet. Des Weiteren beinhaltet die Tabelle ein Feld für den lokalisierten Titel und die lokalisierte Beschreibung. Auch eine lokalisierte Antwortkonfiguration wird gespeichert.

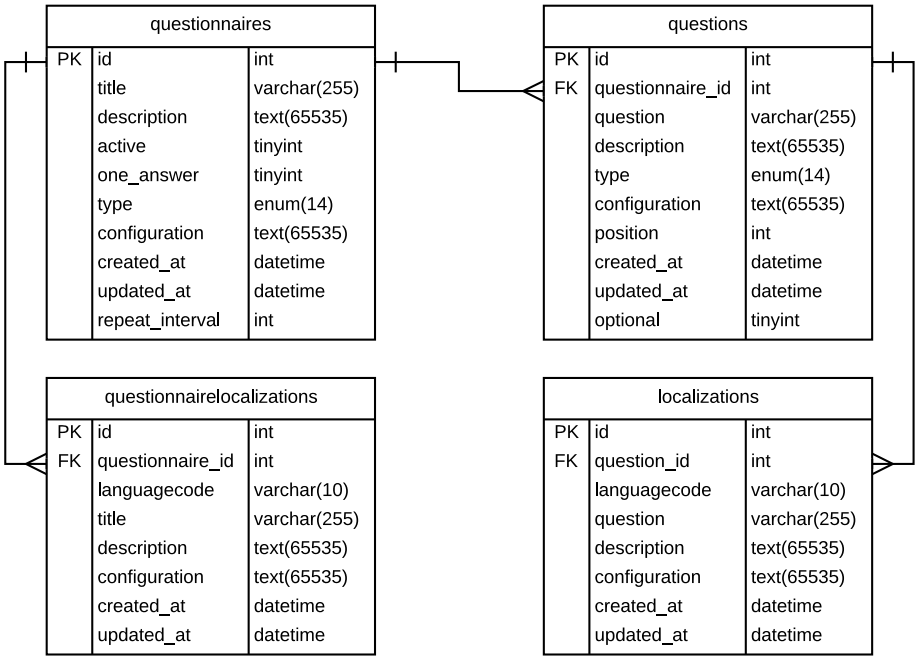

Abbildung 3.5.: Datenstruktur der Lokalisierung der Fragebögen

Analog dazu enthält die Tabelle für die lokalisierte Version einer Frage auch nur die Felder, die übersetzt werden müssen: die Frage selbst, die Beschreibung und die Antwortkonfiguration.

3.3.3. Datenstruktur der Antworten auf den Fragebogen zur Tinnitusüberwachung

Die Antworten auf den Fragebogen zur Überwachung der Tinnituswahrnehmung werden in der Tabelle *standardanswers* gespeichert. Abbildung 3.6 zeigt diese Tabelle. Die Fragen selbst sind in den Apps fest vorgegeben und können nicht dynamisch verändert werden. Jeder Eintrag wird mit einer Referenz auf die *users* Tabelle einem Benutzer zugeordnet. Diese Referenz wird zusätzlich ein zweites Mal eingetragen, falls das Benutzerkonto gelöscht wird. Die Antworten auf die acht Fragen (siehe Kapitel 4.2.5) werden in *question1* bis *question8* gespeichert. Die erste und letzte Frage ist eine Ja/Nein-Frage und wird daher als *tinyint* gespeichert. Die restlichen Antworten haben alle einen Wert zwischen 0 und 1 und werden daher als *float* gespeichert. Wenn ein Benutzer die Pegelerfassung der Hintergrundgeräusche erlaubt, wird dieser Pegel in *soundlevel* gespeichert. Neben den Standard-Zeitstempeln (*created_at* und *updated_at*) wird

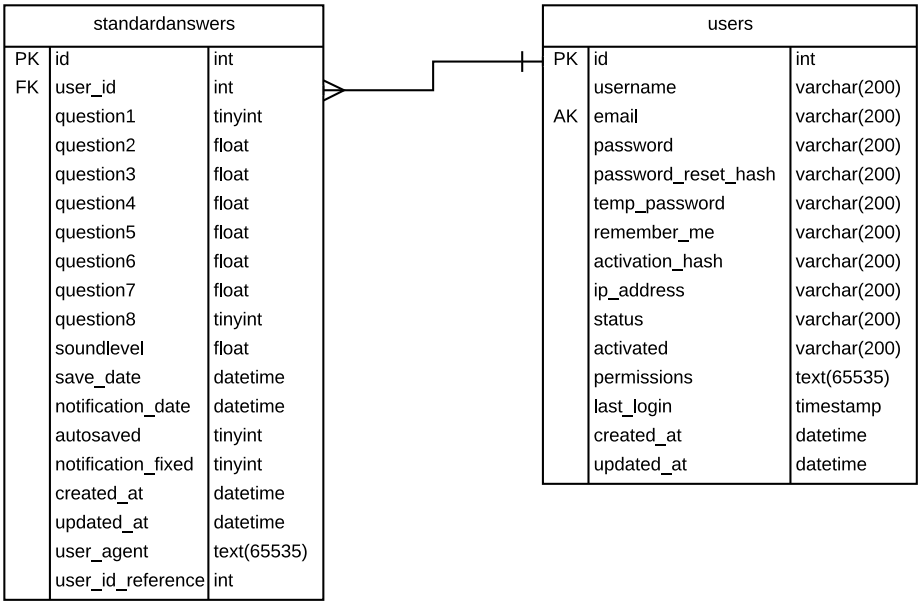

Abbildung 3.6.: Datenstruktur der Antworten auf den Fragebogen zur Tinnitusüberwachung

ein Zeitstempel gespeichert, wann ein Benutzer seine Antworten gespeichert hat. Außerdem wird ein Antwortdatensatz, wenn möglich, einer bestimmten Benachrichtigung zugeordnet und ein Zeitstempel dieser Benachrichtigung gespeichert. Wenn ein Benutzer seine Benachrichtigungszeiten selbst gewählt hat, wir dies in dem Feld *notification_fixed* gespeichert. Sollte ein Benutzer vergessen, seine Antworten explizit zu speichern, werden diese Antworten von den Apps selbstständig gespeichert. Ob der Benutzer seine Antworten speichert, oder die App dies selbstständig gemacht hat, wird mit dem Feld *autosaved* angegeben. Für den Fall, dass sich die Fragen in den Apps in einer neueren Version ändern, wird noch der HTTP-User-Agent im Feld *user_agent* erfasst. Dieser beinhaltet immer die Version der App-Plattform (Android oder iOS), die Versionsnummer der App und bei Android Geräten zusätzlich Informationen zum benutzten Gerät (Hersteller und Modellbezeichnung).

3.3.4. Datenstruktur der Benutzer und Gruppen

Für die Registrierung und Anmeldung eines Benutzers verwendet Track Your Tinnitus "Sentry", ein Package für das Laravel Framework [Car13]. Die Tabellen *users*, *user_groups*, *groups*, *users_metadata*, *users_suspended* und *rules* in Abbildung 3.7 wurden automatisch von diesem Package erstellt. Das Benutzerkonto wird in der Tabelle *users* gespeichert. Neben den Angaben, die ein Benutzer bei der Registrierung machen muss (Benutzername, E-Mail Adresse und Passwort), gibt es Felder für die Funktion zum Zurücksetzen des Passwortes, zum Aktivieren des Accounts und für die Berechtigungen. Jeder Eintrag in *users* hat genau einen Eintrag in der Tabelle *users_metadata*, in der eigene Felder definiert werden können. Track Your Tinnitus speichert hier ob ein Benutzer E-Mails erhalten möchte, ob ein Benutzer seine E-Mail Adresse speichern möchte, und einen Code, um das Passwort zurück zu setzten, sollte die E-Mail Adresse nicht gespeichert sein. Zu statistischen Zwecken wird hier die mobile Plattform

(Android oder iOS) erfasst. *users_suspended* erfasst alle blockierten Benutzer, wobei dies im Administrationsbereich der Webseite nicht implementiert ist. In der Tabelle *rules* lassen sich verschiedene Berechtigungen erstellen, die dann einer Gruppe oder einem Benutzer zugewiesen werden können. Eine Benutzergruppe wird in der Tabelle *groups* gespeichert. Dabei kann ein Benutzer mehreren Gruppen zugehören. Außerdem kann ein Fragebogen einer oder mehreren Gruppen zugeordnet werden. So können bestimmte Fragebögen auch nur für eine bestimmte Benutzergruppe sichtbar gemacht werden.

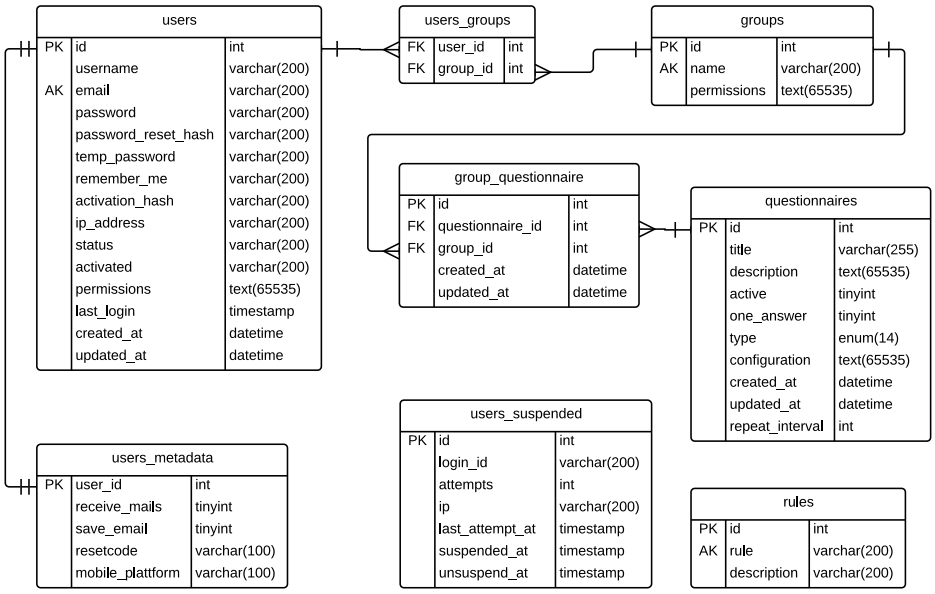

Abbildung 3.7.: Datenstruktur der Benutzer und Gruppen

3.3.5. Datenstruktur der OAuth2 API

Für die Authentifizierung eines Benutzers der Apps an den API-Endpunkten kommt das Package "PHP OAuth 2.0 Server for Laravel" zum Einsatz [Deg13]. Abbildung 3.8 zeigt die Datenstruktur des Authentifizierungsmechanissmus der API, die von diesem Package erstellt wurde. Da Track Your Tinnitus bisher nur die Authentifizierung mit Benutzername und Passwort unterstützt und ein *Access Token* nicht nach einer bestimmten Zeit ungültig wird, werden die Tabellen *oauth_auth_codes* und *oauth_refresh_token* bisher nicht benötigt. In *oauth_clients* werden die *Clients* anhand von *client_id* und *client_secret* gespeichert. Eine *redirect_uri* muss zwar angegeben werden, wird für diesen Authentifizierungsmechanismus aber bisher nicht benötigt. *client_id* und *client_secret* müssen bei der Authentifizierung eines Benutzers übertragen werden, um einen *Access Token* vom Server zu erhalten. Dieser *Access Token* wird in der Tabelle *oauth_access_token* gespeichert und ist direkt mit dem Client und einem Benutzer verknüpft. Diese *Access Tokens* sind bei Track Your Tinnitus unendlich lange gültig, wodurch das Feld *expires* keine Bedeutung hat. Diese Tabelle hält zusätzlich noch einen *scope*, der allerdings bisher ebenfalls keine Verwendung findet.

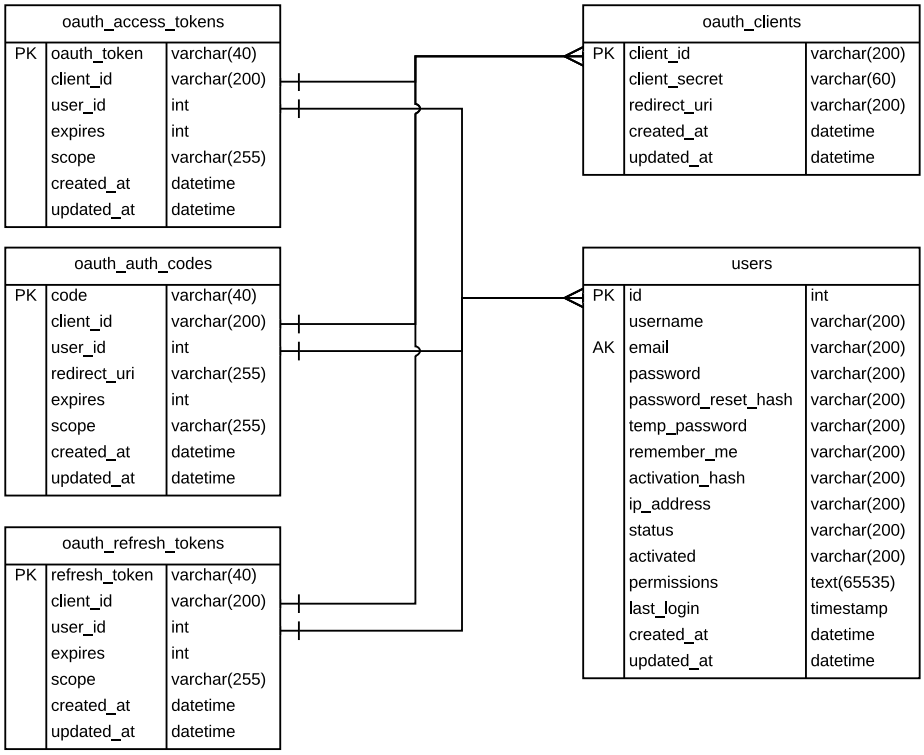

Abbildung 3.8.: Datenstruktur der OAuth2 API

3.4. Architektur der Apps

Die Track Your Tinnitus Apps für iOS und Android wurden jeweils nativ entwickelt. Weitere Informationen, wieso eine native Implementierung für komplexe mobile Anwendungen geeignet ist, ist zu finden unter [Geion] [Gei13] [Rob11] [Sch13d]. Beide Plattformen sind nach dem Model-View-Controller Prinzip aufgebaut. Die folgenden Abbildungen zeigen die entsprechenden Klassendiagramme, wobei immer die Vererbung zu den Super-Klassen der Plattformen dargestellt wird. Diese Super-Klassen sind jeweils grau hinterlegt. Alle Klassen, bei denen in den Diagrammen keine Vererbung angegeben ist, erben von *NSObject* bei iOS und *Object* bei Android.

3.4.1. iOS App

Die Track Your Tinnitus iOS App wurde für iOS6 und iOS7 entwickelt und unterstützt alle iPhone und iPod Touch Modelle, die diese Betriebssystemversionen unterstützen. Auf dem iPad wird eine skalierte Version der App angezeigt.
Die folgenden Kapitel beschreiben die Architektur der iOS App. Dabei werden zuerst die View Controller beschrieben. Danach folgen die *UITableViewCell* Subklassen, das Model und zuletzt die Architektur der selbst implementierten Views und Controls.

3.4.1.1. View Controller

Abbildung 3.9 zeigt die Architektur der View Controller und sonstiger Klassen. Jeder View Controller, dessen View keine Tabelle enthält, erbt direkt von *UIViewController*, alle anderen von *UITableViewController*. *IntroductionLoginViewController* zeigt das entsprechende View zur Einführung in Track Your Tinnitus und lädt auch das View des *LoginViewController*, der für den Login Mechanismus zuständig ist. Zusätzlich enthält der *LoginViewController* auch die Methoden, um alle erforderlichen Daten nach dem Login vom Server zu laden. Der *Regis-*

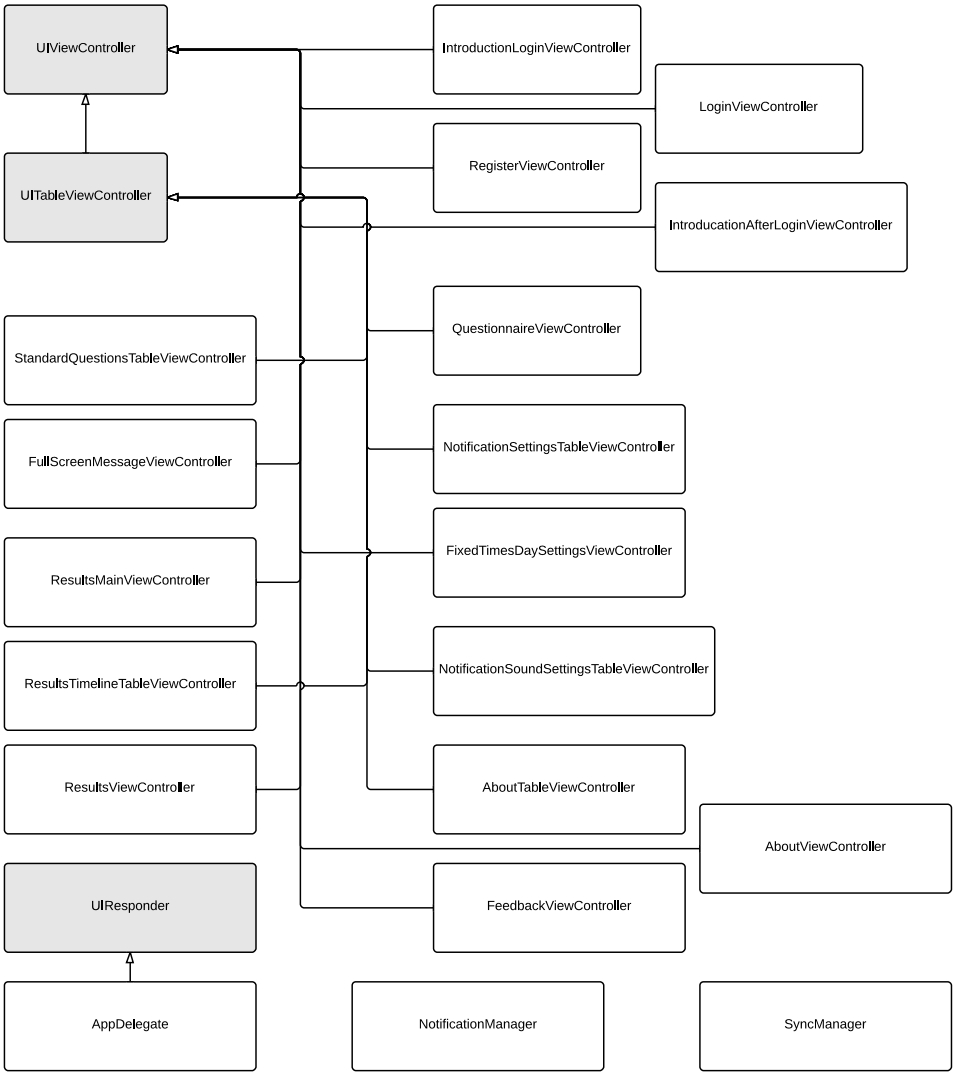

Abbildung 3.9.: iOS ViewController Diagramm

terViewController zeigt das Formular zur Registrierung an und enthält auch den Code, um ein neues Benutzerkonto auf dem Server zu registrieren. Falls ein Benutzer alle statistischen Fragebögen ausgefüllt hat, wird nach dem Login der *IntroductionAfterLoginViewController* geladen. Dieser zeigt nur eine kurze Meldung und gibt dem Benutzer die Auswahl, direkt zu den Benachrichtigungseinstellungen zu wechseln oder zum Fragebogen zur Überwachung der

Schwankungen der Tinnituswahrnehmung zu gelangen. Sollte ein Benutzer noch nicht alle statistischen Fragebögen beantwortet haben, wird der *QuestionnaireViewController* geladen, der einen statistischen Fragebogen anzeigt und die Antworten entsprechend speichert und an den Server sendet. Der *StandardQuestionsTableViewController* zeigt den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung. Nach dem Ausfüllen und Speichern dieses Fragebogens, wird der *FullScreenMessageViewController* geladen. Dieser zeigt eine kurze Meldung und beendet die App nach einem Countdown von drei Sekunden. *NotificationSettingsTableViewController* zeigt die Benachrichtigungseinstellungen und beinhaltet alle Funktionen für diese Einstellung außer die genaue Zeitangabe, wenn ein Benutzer die Benachrichtigungszeiten selbst festlegen will. Diese Einstellung wird mit einem Kalender realisiert, der von *FixedTimesDaySettingsViewController* angezeigt wird. Die generellen Einstellungen (z.B. des Klingeltons der Benachrichtigungen) zeigt der *NotificationSoundSettingsTableViewController*. Die Ergebnisse können vom Benutzer in zwei verschiedenen Darstellungen angezeigt werden. Einerseits als Diagramme, welche von *ResultsViewController* geladen und angezeigt werden, andererseits als Timeline, die von *ResultsTimelineTableViewController* geladen und angezeigt wird. Beide View Controller werden wiederum im *ResultsMainViewController* geladen, der auch das Umschalten zwischen den Ansichten beinhaltet. Die drei restlichen View Controller *AboutTableViewController*, *AboutViewController* und *FeedbackViewController* werden für den ´´About" Bereich benötigt. *AboutTableViewController* zeigt dabei die Übersicht und alle Unterpunkte (wie z.B. Über das Team) in einer Tabelle. *AboutViewController* zeigt die Texte, wenn ein Benutzer einen Unterpunkt wählt. Der *FeedbackViewController* zeigt ein Formular, mit dem ein Benutzer direkt eine Nachricht an das Team senden kann.
Zusätzlich zu den View Controllern zeigt Abbildung 3.9 die Klassen *AppDelegate*, *NotificationManager* und *SyncManager*. *AppDelegate* implementiert die Methoden, die von dem Delegate des Singelton-Objekts *UIApplication* vorgegeben sind. Diese Methoden bieten Informationen über die Schlüsselereignisse des Lebenszyklus einer App [App13i]. Der *NotificationManager* beinhaltet die Methoden, um automatisch die Local Notifications anhand der Benachrichtigungseinstellungen zu planen. Außerdem ist hier auch der Algorithmus zur zufälligen Verteilung der Benachrichtigungen bei Standard-Einstellungen implementiert. *SyncManager* ist für die Synchronisierung der Antworten auf den Fragebogen zur Überwachung der Tinnituswahrnehmung verantwortlich. Diese Synchronisierung wird nach jedem Ausfüllen des Fragebogens und zusätzlich noch beim Beenden der App gestartet. Sollte kein neuer Antwort-Datensatz vorhanden sein, oder die Übertragung fehlschlagen, wird der Datensatz beim nächsten Aufruf zum Server gesendet.

3.4.1.2. Table Cells

In iOS wird jede Zeile einer Liste, oder Tabelle, in einem TableViewController durch eine Object der Klasse UITableViewCell repräsentiert. Abbildung 3.10 zeigt die verwendeten Subklassen von UITableViewCell. Im oberen Teil von Abbildung 3.10 sind die Klassen dargestellt, die für die TableViewController verwendet werden, die die Fragebögen anzeigen. Diese Klassen werden sowohl bei den statistischen Fragebögen verwendet, als auch bei dem Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung. Jedem möglichen Antworttyp wurde eine eigene Subklasse zugeordnet. *BaseCell* beinhaltet alle Methoden, die alle Antwortmöglichkeiten gemeinsam haben. Alle zum jeweiligen Antworttyp spezifischen Methoden wurden

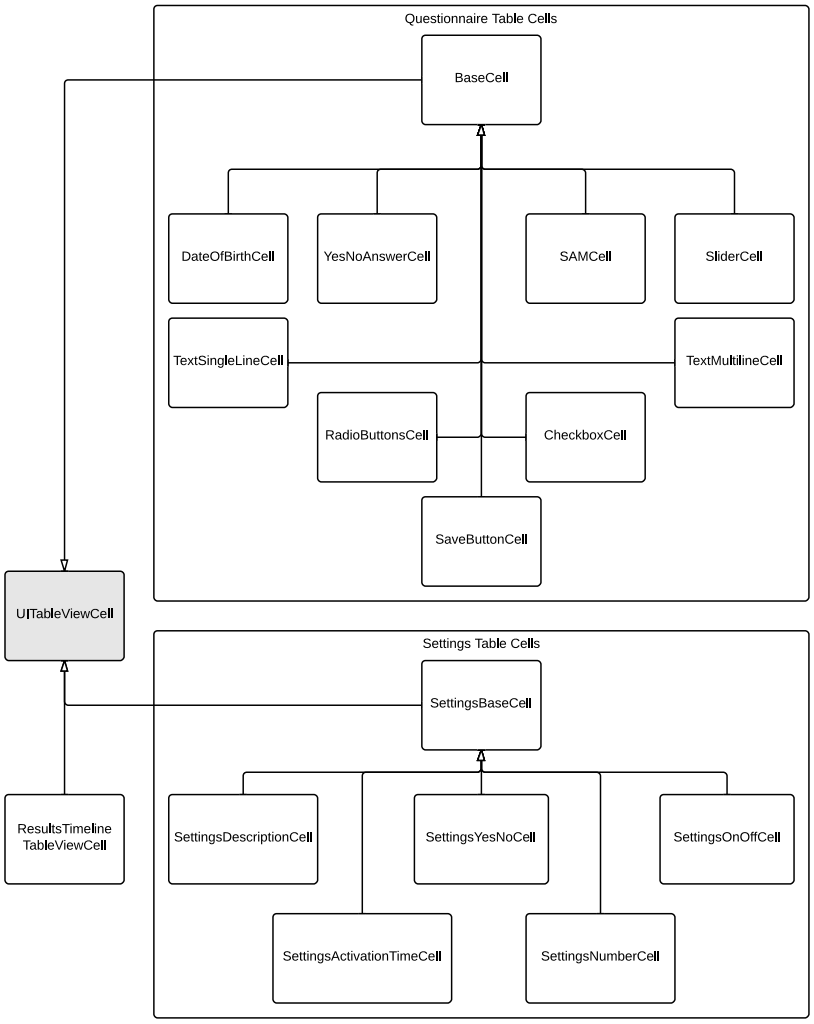

Abbildung 3.10.: iOS TableViewCell Diagramm

in den Subklassen implementiert. Folgende Klassen mit Ihrem zugehörigen Antworttyp sind implementiert:

- **DateOfBirthCell:** Zelle um ein Datum (z.B. ein Geburtsdatum) einzutragen
- **YesNoAnswerCell:** Zelle für eine Ja/Nein-Frage
- **SAMCell:** Zelle für eine Frage des Self-Assessment-Tests. Wird nur im Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung verwendet.
- **SliderCell:** Zelle für eine Frage, die anhand eines Sliders beantwortet wird. Der Slider zeigt allerdings initial keinen Knopf an.
- **TextSingleLineCell:** Zelle für eine Frage, die mit einem Text beantwortet wird. Es wird nur ein einzeiliges Textfeld angezeigt.
- **TextMultilineCell:** Analog zu TextSingleLineCell. Es wird allerdings ein Textfeld mit mehreren Zeilen angezeigt.

- **RadioButtonCell:** Zelle, die verschiedene Antwortmöglichkeiten vorgibt, von denen nur eine ausgewählt werden kann.
- **CheckboxCell:** Analog zu *RadioButtonCell*. Es kann allerdings mehr als eine Antwortmöglichkeit ausgewählt werden.
- **SaveButtonCell:** Zelle zeigt nur einen Button, mit dem der Fragebogen gespeichert werden kann.

Im unteren Teil von Abbildung 3.10 sind die Klassen dargestellt, die für die Benachrichtigungseinstellungen verwendet werden. Auch hier erben alle Zellen von *SettingsBaseCell*, die die gemeinsamen Methoden vereint. Im Folgenden sind die Subklassen von *SettingsBaseCell* mit ihrer jeweiligen Funktion aufgelistet:

- **SettingsDescriptionCell:** Zelle, die einen kurzen Erklärungstext zeigt.
- **SettingsYesNoCell:** Diese Zelle wird verwendet, um zwischen den Benachrichtigungsarten Standard und Benutzerdefiniert zu wechseln.
- **SettingsOnOffCell:** Eine Zelle, die einen Text und einen Ein-/Aus-Schalter anzeigt. Diese wird verwendet, um Benachrichtigungen im Gesamten ein- bzw. auszuschalten.
- **SettingsActivationTimeCell:** Diese Zelle wird bei der Benachrichtigungsart Standard verwendet, um Start- und Endzeitpunkt des Benachrichtigungszeitraumes einzustellen.
- **SettingsNumberCell:** Zelle, die die Anzahl an Benachrichtigungen zeigt und außerdem zwei Buttons beinhaltet, mit welchen diese Anzahl erhöht oder verringert werden kann.

Die Ergebnisdarstellung in einer Timeline wurde ebenfalls mit einem TableViewController realisiert. Die Zelle *ResultsTimelineTableViewCell* zeigt einen Eintrag in dieser Timeline.

3.4.1.3. Datenmodell

Das Datenmodell auf dem Server, wie in Kapitel 3.3 beschrieben, wird zum Teil als Klassen in der iOS App umgesetzt. Abbildung 3.11 zeigt das Klassendiagramm des Datenmodells auf einem iOS Gerät. Die Klasse *FixedNotificationTime* repräsentiert eine Uhrzeit/Tag Kombination im Speicher, wenn ein Benutzer eine Uhrzeit im Kalender der Benachrichtigungseinstellungen erstellt. Diese Objekte werden nicht direkt in die interne Datenbank gespeichert, sondern in den *NSUserDefaults* [App13e]. Analog dazu wird die Klasse *StandardQuestion* verwendet, um die Fragen des Fragebogens zur Überwachung der Tinnituswahrnehmung im Speicher zu verwalten. Diese werden ebenfalls nicht in der Datenbank gespeichert. Alle anderen Klassen, die direkt von *NSManagedObject* [App13d] erben, werden in die interne SQLite Datenbank des iOS Gerätes persistiert. Ein Object der Klasse *Notification* beinhaltet Felder für den Typ der Benachrichtigung (Standard oder Benutzerdefiniert), den Tag und die Uhrzeit, wann eine Benachrichtigung angezeigt werden soll. Der *NotificationManager* speichert so die geplanten Benachrichtigungszeiten in der Datenbank, um sie später einem Antwort-Datensatz zuordnen zu können. Um die Antworten auf den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung zu verwalten, wird die Klasse *StandardAnswer* verwendet. Die *SerializationAdditions* bieten zusätzlich Methoden, um einen Datensatz zu serialisieren und an den Server senden zu können. *Questionnaire*, *Question* und *Answer* wurden analog zum Model auf dem Server erstellt und werden verwendet, um einen statistischen Fragebogen, inklusive Fragen und Antworten eines

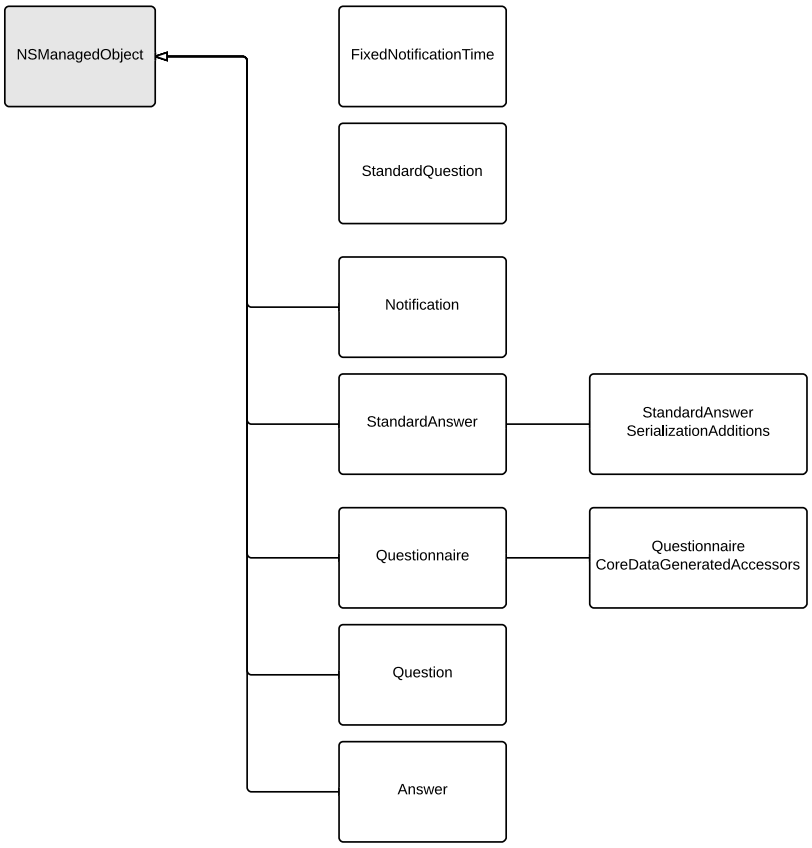

Abbildung 3.11.: iOS Modell Diagramm

Benutzers, im Speicher darzustellen. Die *CoreDateGeneratedAccessors* bieten zusätzlich Methoden, um einem Fragebogen-Objekt direkt ein Frage-Objekt zuzuordnen.

3.4.1.4. Eigene Views und Controls

Für Track Your Tinnitus konnten nicht für jede Funktion die Standard-Views und Standard-Controls von iOS verwendet werden. Eine Checkbox, oder ein Slider ohne initialen Wert sind in iOS nicht vorgesehen. Daher haben wir ein eigenes View und drei eigene Controls implementiert. Abbildung 3.12 zeigt die Klassen, mit denen diese Komponenten implementiert sind.
ResultsChartView bekommt die Daten der Antworten auf den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung übergeben und stellt für eine der Fragen die Werte in einem Diagramm grafisch dar. Da diese View keine weiteren Funktionen benötigt, erbt es direkt von *UIView*. *UIControl* ist die Basisklasse, um eigene Controls zu erstellen. Sie bietet, unter anderem, Methoden um Touches auf diesem Control zu überwachen [App11]. Die Klasse *JHCheckbox* ist für die horizontale Reihe von Checkboxes verantwortlich. In dieser Reihe kann ein Benutzer mehr als eine Möglichkeit auswählen. Die eigentlichen Checkboxes werden in eigenen Layer-Klassen gezeichnet. Da die oberen und unteren Ecken des Rahmens einer Checkbox-Reihe abgerundet sind, wird jeweils die erste und letzte Checkbox von *First-ButtonLayer* und *BottomButtonLayer* gezeichnet. Alle Checkboxes, die sich dazwischen finden,

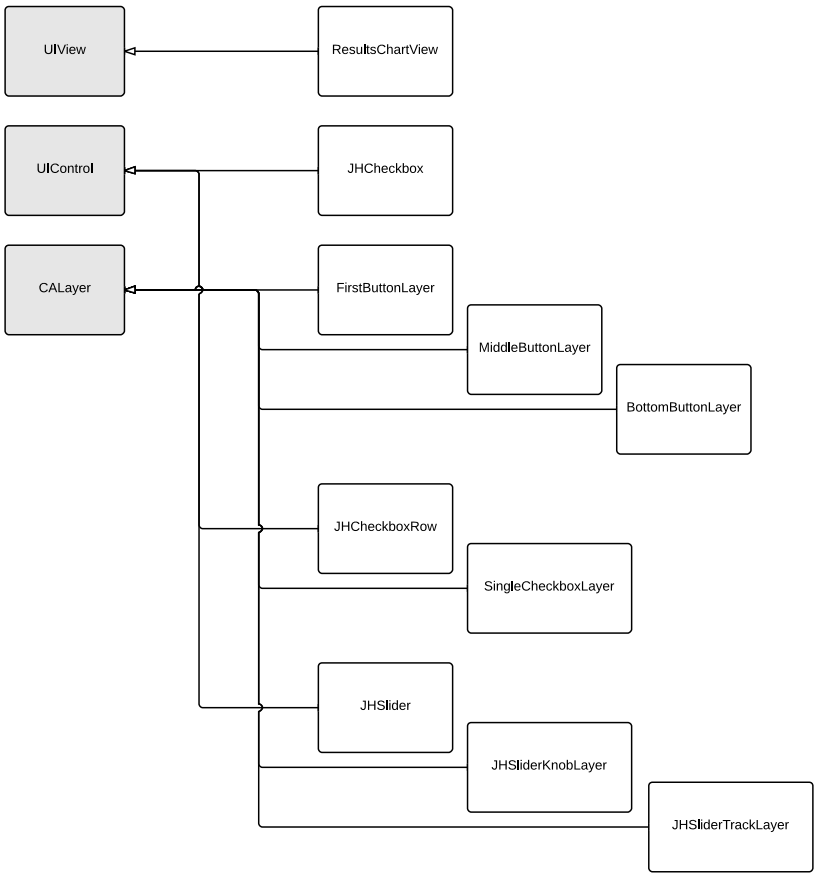

Abbildung 3.12.: iOS View und Control Diagramm

werden von *MiddleButtonLayer* gezeichnet. *JHCheckboxRow* bietet eine horizontale Anordnung von Radio Buttons, die für den Self-Assessment-Test verwendet werden. Die einzelnen Buttons werden mit der Klasse *SingleCheckboxLayer* gezeichnet. *JHSlider* ist die Implementierung des Sliders, der allerdings initial keinen Knopf hat. Erst nach der ersten Interaktion mit dem Control erscheint dieser Knopf. Der Rest der Funktionen ist an die des Sliders aus iOS angelehnt. Auch hier wird die Leiste, auf der sich der Knopf später bewegt, mit der Klasse *JHSliderTrackLayer* gezeichnet. Der Knopf wird von der *JHSliderKnobLayer* gezeichnet. Die Logik, um den Slider anhand der Touches zu bewegen, befindet sich, wie bei den anderen Controls auch, in der *JHSlider* Klasse.

3.4.2. Android App

Bei der Implementierung der Track Your Tinnitus Android App in Java wurden viele Inner-Classes verwendet. Daher sind die Klassendiagramme der Android App nach den Funktionen geordnet aufgebaut. In den folgenden Kapiteln wird zuerst die Architektur der Login- und Registrierungsfunktionen, sowie der Hauptfunktion der App beschrieben. Danach folgt das Datenmodel und die eigenen Views, Receiver und Services.

3.4.2.1. Login, Registrierung und Hauptmenü

Der Login, die Registrierung und das Hauptmenü wird, mit allen zugehörigen Klassen, in Abbildung 3.13 dargestellt. Die Klasse *LoginActivity* zeigt ein Formular, mit dem sich ein Benutzer in der App anmelden kann. Diese Klasse hat drei innere Klassen, die jeweils von *AsyncTask* erben. *DownloadQuestionnairesAndQuestions* lädt die Fragebögen und Fragen vom Server. *DownloadStandardAnswers* lädt die Antworten auf den Fragebogen zur Überwachung der

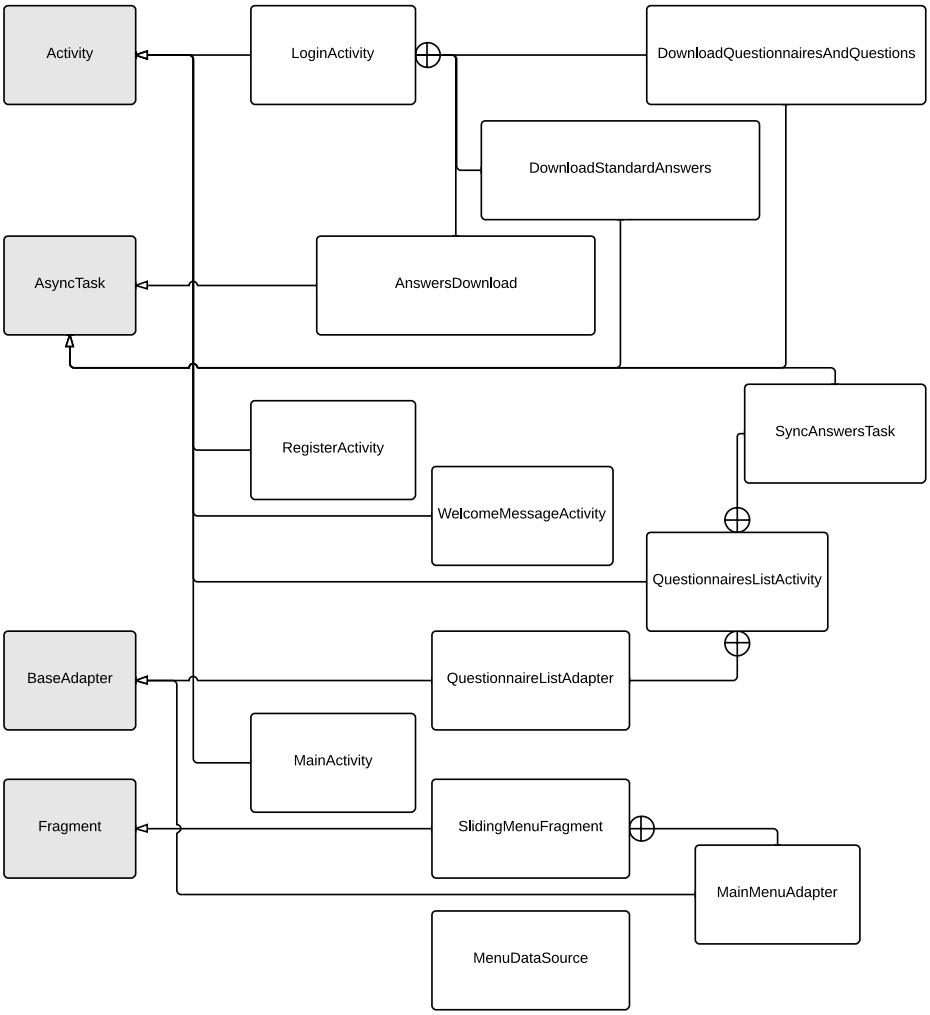

Abbildung 3.13.: Android Login, Registrierung und Hauptmenü Klassendiagramm

Schwankungen der Tinnituswahrnehmung, falls ein Benutzer sich nicht das erste Mal in der App anmeldet. *AnswersDownload* lädt die Antworten auf die statistischen Fragebögen, falls diese bereits auf der Webseite beantwortet wurden. Die *RegisterActivity* Klasse stellt das Formular zum Erstellen eines Benutzerkontos dar und bietet die entsprechende Methode, um diese Daten an den Server zu senden. Nach der erfolgreichen Anmeldung wird die Klasse *WelcomeMessageActivity* geladen, wenn bereits alle statistischen Fragebögen beantwortet wurden. Diese Klasse zeigt eine kurze Meldung und bietet einem Benutzer die Möglichkeit, entweder direkt zu den Benachrichtigungseinstellungen oder zum Fragebogen zur Überwachung der Schwankungen der

Tinnituswahrnehmung zu gelangen. Falls noch nicht alle statistischen Fragebögen beantwortet wurden, werden nach der Anmeldung diese statistischen Fragebögen nacheinander angezeigt. *QuestionnairesListActivity* lädt, mithilfe der Klasse *SyncAnswersTask*, immer zuerst die Antworten vom Server, falls sich in der Zwischenzeit etwas verändert hat, und stellt diese dar. In einer Liste dient ein Adapter immer als Verbindung zwischen dem View und den Daten [Gooa]. Die Klasse *QuestionnaireListAdapter* lädt die einzelnen Zeilen eines statistischen Fragebogens. Die Klasse *MainActivity* ist die Hauptklasse und beinhaltet die Methoden, um das Hauptmenü zu öffnen oder zwischen den einzelnen Bereichen der App zu wechseln. Das Hauptmenü wird dabei als Fragment von *SlidingMenuFragment* dargestellt. Die Einträge des Hauptmenüs werden von *MainMenuAdapter* bereitgestellt, wobei die Daten für diese Liste von der Klasse *MenuDataSource* geladen werden.

3.4.2.2. Hauptfunktionen

Die Hauptfunktionen der Track Your Tinnitus Android App wurden mithilfe von Fragments implementiert, die von der *MainActivity* entsprechend geladen werden. Abbildung 3.14 zeigt das Klassendiagramm der Hauptfunktionen. *StandardQuestionnaireFragment* lädt den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung. *StandardQuestionnaireAdapter* dient dieser Liste als Adapter und stellt so die Verbindung zwischen dem View und den Daten her, die von der Klasse *StandardQuestionnaireDataSource* zur Verfügung gestellt werden. *MessageFragment* stellt eine einfache Meldung dar und beendet die App nach einem Countdown von drei Sekunden. Die Benachrichtigungseinstellungen werden von der Klasse *NotifictionSettingsFragement* dargestellt. Die Daten dieser Einstellungen werden von *NotificationSettingsDataSource* verwaltet und durch die Klasse *EntryAdapter* geladen. Um den Kalender, mit dem ein Benutzer die genauen Benachrichtigungszeiten einstellen kann, deutlich von den restlichen Einstellungen zu trennen, ist dieser in einer separaten *Activity* implementiert. *CalendarActivity* stellt den Kalender in der Tag- und Wochenansicht dar und übernimmt ebenfalls die Datenverwaltung der Benachrichtigungszeiten. Wenn ein Benutzer eine dieser Benachrichtigungszeiten bearbeitet, wird ein entsprechender Dialog angezeigt, der in der Klasse *CalendarEditDialogFragment* implementiert ist. Die Einstellungen des Klingeltons einer Benachrichtigung und der Privatsphäre werden ebenfalls in einer Liste dargestellt. *GeneralSettingsFragment* stellt diese Liste dar und bietet Methoden, die die Einstellungen, die ein Benutzer vornimmt, verwaltet. *GeneralSettingsAdapter* stellt dabei die Daten zur Verfügung. Eine eigene Klasse für die Daten ist hier nicht nötig, da die Daten der Liste direkt im Adapter implementiert sind. Da die Anzahl der Antworten auf dem Fragebogen zur Überwachung der Schwankungen der Tinnituswahnehmung sehr hoch werden kann, werden diese Datensätze zur Ergebnisdarstellung in einem eigenen *AsyncTask* geladen. Die Klasse *LoadAnswersFromDatabaseTask* lädt die Datensätze in einem separaten Thread aus der Datenbank und gibt sie an das *ResultsFragment* zurück. Die Klasse *ResultsFragment* stellt diese Daten dann in Diagrammen dar. Die Timline-Ansicht der Ergebnisse ist, wie in iOS, mit einer Liste implementiert. *ResultsListFragment* verwaltet diese Liste. *ResultsCursorAdapter* stellt dabei die Verbindung zur internen Datenbank her. Der letzte Punkt des Hauptmenüs ist der About-Bereich. Die Übersicht der Unterpunkte wird durch die Klasse *AboutListFragment* dargestellt. Diese Liste ist statisch und wird, wie auch bei den allgemeinen Einstellungen, vom *StableArrayAdapter* geladen. Wenn ein Benutzer einen Eintrag des About-Menüs auswählt, wird eine neue *Activity* geladen. *AboutDetailActivity* stellt dann den ausgewählten Text dar.

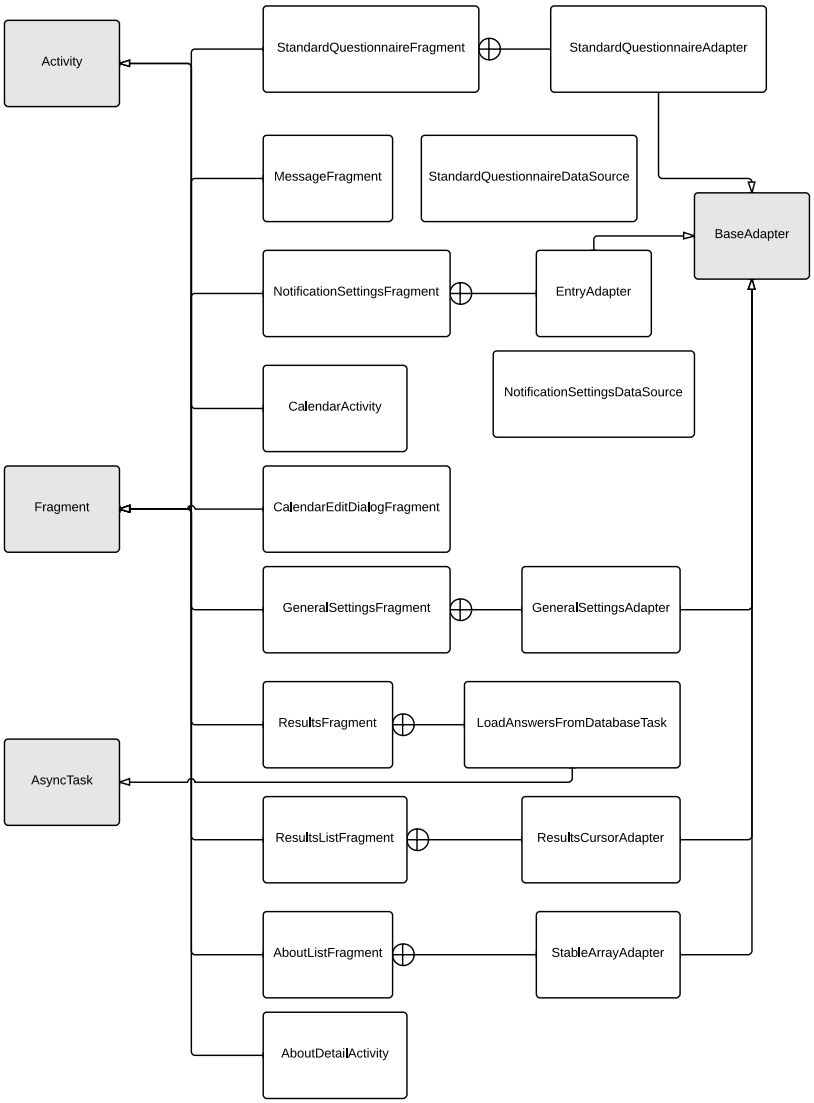

Abbildung 3.14.: Android Klassendiagramm der Hauptfunktionen

3.4.2.3. Model

Das Datenmodel wird, wie bei iOS, in Klassen umgesetzt. Abbildung 3.15 zeigt die Klassen des Datenmodels unter Android, sowie die *SQLiteOpenHelper* Subklassen, die benötigt werden, um die Daten mit der SQLite Datenbank zu synchronisieren [Gooo]. Für die Datenhaltung der statistischen Fragebögen, sind die Klassen *Questionnaire* für die Daten des Fragebogens, *Question* für die Fragen eines Fragebogens und *Answer* für die Antworten, verantwortlich. Jede dieser Klassen hat eine eigene *SQLiteOpenHelper* Subklasse, die die Daten in die Datenbank persistiert. *QuestionnaireSQLiteOpenHelper* verwaltet die Fragebögen, *QuestionsSQLiteOpenHelper* die Fragen und *AnswersSQLiteOpenHelper* die Antworten. Für den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung werden die Fragen mithilfe der Klasse *StandardQuestion* verwaltet. Die Antworten auf diese Fragen werden alle in ein Ob-

jekt der Klasse *StandardAnswer* gehalten. Dieses Objekt wird beim Speichern durch die Klasse

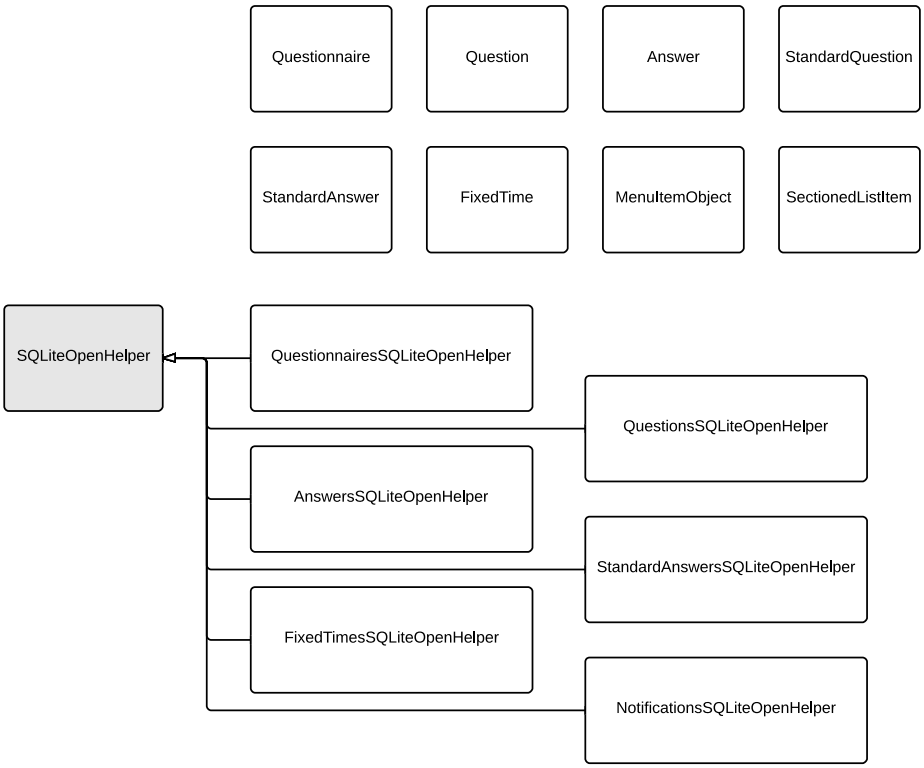

Abbildung 3.15.: Android Klassendiagramm des Datenmodels

StandardAnswersSQLiteOpenHelper in die Datenbank gesichert. Die Fragen diese Fragebogens sind fest einprogrammiert. Daher wird keine Subklasse von *SQLiteOpenHelper* für die Fragen benötigt. Die Klasse *FixedTime* hält die Daten eines Benachrichtigungszeitpunktes, den ein Benutzer selbst fest einstellen kann. Im Gegensatz zu iOS werden die Benachrichtigungszeiten unter Android in der internen SQLite Datenbank gespeichert. Für die Verwaltung dieser Benachrichtigungszeiten in der Datenbank wird die Klasse *FixedTimesSQLiteOpenHelper* benötigt. Ein Objekt der Klasse *MenuItemObject* hält die Daten eines Eintrages im Hauptmenü. *SectionedListItem* ist für die Einträge in den Listen der Benachrichtigungseinstellungen und der allgemeinen Einstellungen verantwortlich und kann entweder als Überschrift einer Section markiert werden, oder als üblicher Listeneintrag. Die Benachrichtigungen, die anhand der Benachrichtigungseinstellungen erstellt werden, werden in der Datenbank gespeichert. Dafür bietet die Klasse *NotificationsSQLiteOpenHelper* die entsprechenden Methoden. Eine eigene Klasse existiert für diese Benachrichtigungen nicht.

3.4.2.4. Custom Views, Receiver und Services

Ähnlich wie unter iOS bietet Android die Klasse *View* für die Implementierung eigener User-Interface-Elemente [Gooq]. Die Diagramme in der Ergebnisdarstellung werden von der Klasse *JHGraphView*, anhand der übergebenen Daten gezeichnet. Im Gegensatz zur iOS Version bietet Android in der Klasse *SeekBar* die entsprechenden Methoden, die benötigt werden, um

einen Slider ohne initialen Wert anzuzeigen. Dieser Slider wird von der Klasse *JHSeekBar* dargestellt und erbt direkt von *SeekBar* [Goom]. Die Benachrichtigungen werden mithilfe des *AlarmManagers* [Goob] geplant. Dieser *AlarmManager* sendet dann zu dem Zeitpunkt an dem die Benachrichtigung erscheinen soll ein Intent an eine Instanz der Klasse *NotificationReceiver*. Diese Klasse erstellt dann die Benachrichtigung (Titel, Text, Sound, LED-Farbe) und stellt diese auf dem Gerät des Benutzers dar. Da die Zeiten, an denen der *AlarmManager* den *NotificationReceiver* aufruft, einen Neustart des Gerätes nicht überstehen, wird nach dem Start des Gerätes immer ein Intent an *BootCompleteReceiver* gesendet. Dieser startet dann einen Service [Goon], den *CustomNotificationService*, der das Planen der Benachrichtigungen startet. Die Methoden, um Benachrichtigungen im *AlarmManager* zu planen, werden von der Klasse

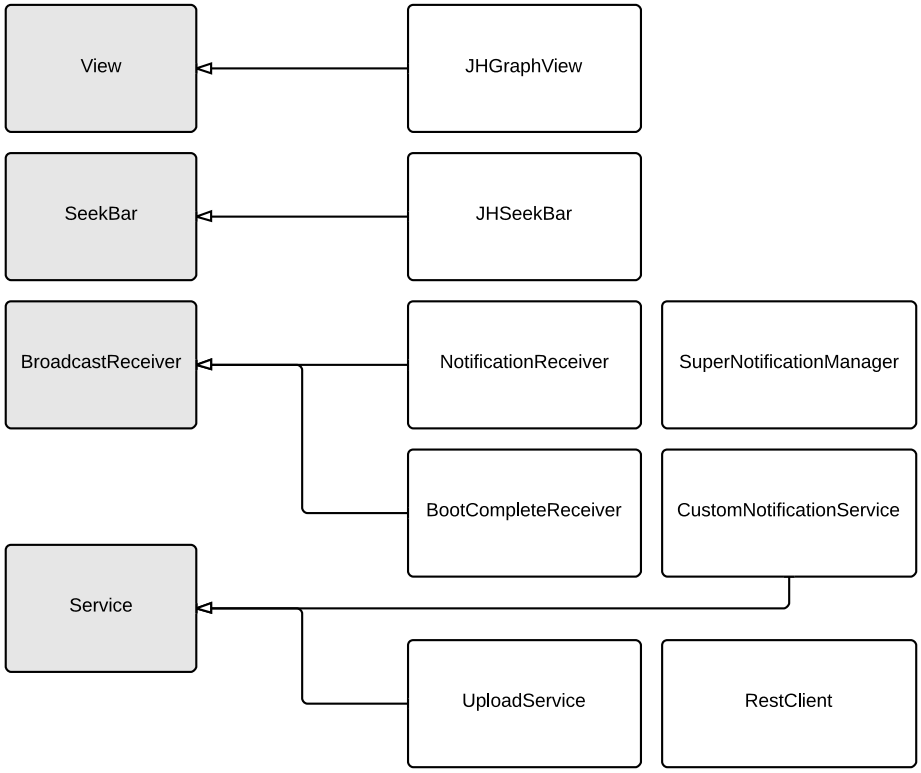

Abbildung 3.16.: Android Klassendiagramm Custom Views, Receiver und Services

SuperNotificationManager geboten. Hier ist auch der Algorithmus implementiert, der die Benachrichtigungszeiten nach dem Zufall verteilt. Die Klasse *UploadService* bietet alle Methoden, die benötigt werden, um die Antworten des Benutzers (auf statistische Fragebögen, sowie auf den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung) an den Server zu senden. Dieser Service wird immer beim Schließen der App gestartet. Die Klasse *RestClient* kapselt die Methodenaufrufe für eine Anfrage an die API.

There's an app for that. That's the iPhone. Solving life's dilemma one app at a time.

Apple's Marketing Slogan 2008

4

Vorstellung des Track Your Tinnitus Rahmenwerk

Dieses Kapitel stellt das Track Your Tinnitus Rahmenwerk aus Sicht eines Benutzers vor. Dabei werden die einzelnen Funktionen gezeigt. Dieses Kapitel ist in zwei Abschnitte unterteilt. Im ersten Abschnitt (Kapitel 4.1) wird die Webseite vorgestellt und gezeigt, welche Möglichkeiten ein Benutzer auf der Webseite hat. Der zweite Abschnitt beschreibt die Funktionen der Apps und zeigt dabei die Unterschiede zwischen der Version für iOS und Android (Kapitel 4.2).

4.1. Vorstellung der Webseite

Auf der Track Your Tinnitus Webseite kann sich ein Benutzer registrieren, statistische Fragebögen ausfüllen, Ergebnisse anzeigen und detaillierte Informationen über das Projekt erfahren. Administratoren bietet sie außerdem die Möglichkeit statistische Fragebögen, Benutzer und Gruppen zu verwalten. Die Überwachung der Schwankungen der Tinnituswahrnehmung ist allerdings nur in den Apps (siehe Kapitel 4.2) möglich.

4.1.1. Startseite

Die Startseite der Track Your Tinnitus Webseite ist die erste Seite, die ein Besucher sieht, wenn er das Projekt in seinem Browser öffnet. Unter dem Titelbild wird erklärt, welche Schritte ein Besucher ausführen muss, um die Schwankungen seiner Tinnituswahrnehmung überwachen zu können (1). Abbildung 4.1 zeigt die Startseite mit angemeldetem Benutzer, der auch Administrations-Rechte hat. Über die Navigationsleiste am oberen Rand der Seite kann der Benutzer zu den unterschiedlichen Bereichen der Seite navigieren (2). So hat er Zugang zu seinem Benutzerbereich (siehe Kapitel 4.1.2), der *About*-Seite, mit detaillierten Informationen über das Projekt, den Datenschutz und das Team, sowie der Administrationsoberfläche (siehe

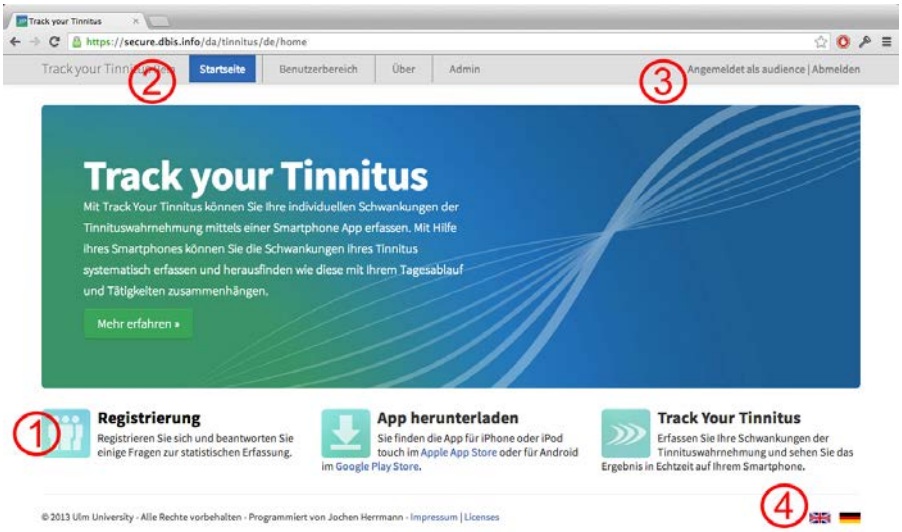

Abbildung 4.1.: Startseite

Kapitel 4.1.6). Auf der rechten Seite in der Navigationsleiste wird angezeigt, ob ein Besucher angemeldet ist. Zusätzlich gibt es hier die Möglichkeit, einen Benutzer abzumelden (3). In der rechten unteren Ecke befindet sich die Sprachauswahl (4). Diese Sprachauswahl wird, wie die restliche Fußzeile, auf jeder Seite des Projektes angezeigt. Wenn die Webseite in die Sprache des Benutzers übersetzt ist, wird anhand der Spracheinstellung des Browsers automatisch die entsprechende Sprache gewählt und eine lokalisierte Version der Webseite angezeigt. Ein Benutzer kann die Sprache aber auch manuell wählen. Track Your Tinnitus unterstützt zum Zeitpunkt dieser Arbeit die Sprachen Deutsch und Englisch.

4.1.2. Benutzerbereich

Nach der Anmeldung mit Benutzername und Kennwort wird ein Benutzer automatisch in seinen persönlichen Benutzerbereich (Abbildung 4.2) geleitet. Diese Seite beinhaltet eine Unternavigation (1), mit der der Benutzer zu seinen statistischen Fragebögen (siehe Kapitel 4.1.3), zur Ergebnisdarstellung seiner Daten aus der App (siehe Kapitel 4.1.5), sowie Einstellungen zu seinem Konto oder Benachrichtigungen gelangen kann. Diese Einstellungen beinhalten das Ändern des Benutzernames oder Kennworts, das Löschen des gesamten Kontos und der Einstellung, ob der Benutzer einen Newsletter per E-Mail erhalten möchte. Im Hauptteil des Bildschirms wird der Fortschritt des Benutzers beim Ausfüllen der statistischen Fragebögen visualisiert. Dieser Fortschritt wird einerseits je Fragebogen anhand eines Balkens dargestellt (2), andererseits gibt es einen Balken, der den Fortschritt über alle Fragebögen anzeigt (3). Diese Fortschrittsbalken haben je nach Status eines Fragebogens eine bestimmte Farbe. Ein unbeantworteter Fragebogen wird durch einen leeren Fortschrittsbalken dargestellt. Sollte ein Fragebogen zum Teil beantwortet sein, wird dies durch einen blauen Fortschrittsbalken dargestellt. Vollständig beantwortete Fragebögen werden mithilfe eines grünen, vollständig gefüllten, Fortschrittsbalken dargestellt. Abbildung 4.2 zeigt einen Benutzerbereich, bei dem alle Fragebögen vollständig beantwortet wurden. So kann direkt erkannt werden, welche Fragebögen noch beantwortet werden müssen und wieviele aller Fragebögen bereits beantwortet wurden.

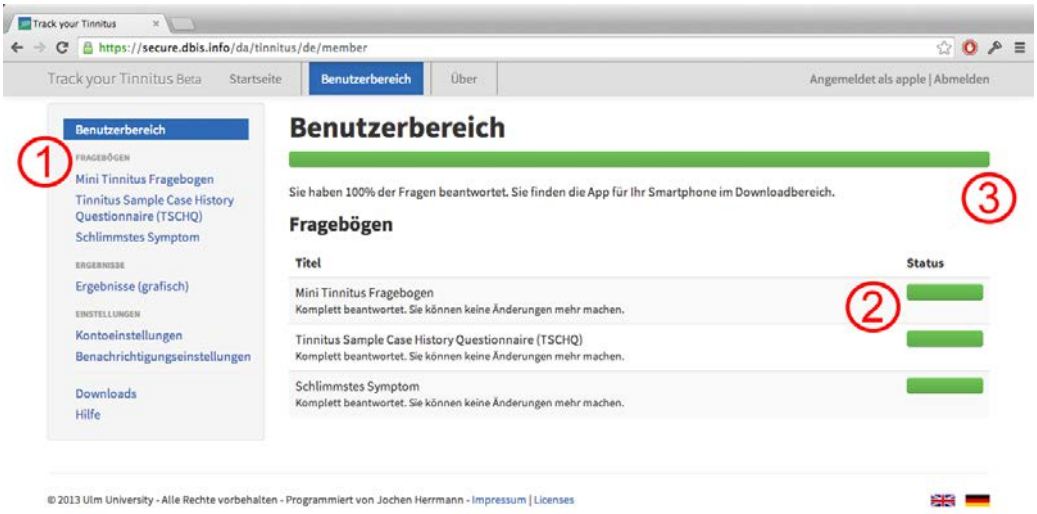

Abbildung 4.2.: Benutzerbereich

4.1.3. Ausfüllen eines statistischen Fragebogens

Die statistischen Fragebögen können sowohl in der App als auch im Browser auf der Webseite ausgefüllt werden. Um die Überwachung der Schwankungen der Tinnituswahrnehmung in der App benutzen zu können, müssen zuerst alle statistischen Fragebögen ausgefüllt werden. Folgende drei statistische Fragebögen sind zum Zeitpunkt dieser Arbeit eingepflegt:

- Mini Tinnitus Fragebogen (siehe Kapitel A.1)
- Tinnitus Sample Case History Questionnaire (siehe Kapitel A.2)
- Schlimmstes Symptom (siehe Kapitel A.3)

Ein statistischer Fragebogen besteht immer aus einem Titel (1), einer optionalen Beschreibung (2) und Fragen (3) mit entsprechenden Antwortmöglichkeiten. Abbildung 4.3 zeigt einen Fragebogen, der für alle Fragen einheitliche Antwortmöglichkeiten bietet.
Ein Benutzer hat beim Speichern des Fragebogens drei verschiedene Möglichkeiten:

- Wenn alle nicht-optionalen Fragen beantwortet sind, werden die Antworten gespeichert und der Fragebogen abgeschlossen. Eine Änderung dieser Antworten ist dann nicht mehr möglich (So wird verhindert, dass ein Nutzer die Antworten verändern kann, wenn sich zu einem späteren Zeitpunkt oder im Laufe einer Studie etwas verändert).
- Wenn mindestens eine nicht-optionale Frage nicht beantwortet wurde, kann der Benutzer den Fragebogen speichern, ohne ihn abzuschließen. Der Fragebogen kann dann später erneut geöffnet werden, um die Antworten zu bearbeiten und die noch unbeantworteten Fragen zu beantworten. Es ist dann auch möglich, den Fragebogen nach der Anmeldung in der App zu bearbeiten und die restlichen Fragen auf dem Smartphone zu beantworten.
- Wenn mindestens eine nicht-optionale Frage nicht beantwortet wurde, kann der Benutzer den Fragebogen speichern und abschließen. Da manche Benutzer nicht alle Fragen beantworten können oder möchten, gibt es so auch die Möglichkeit den Fragebogen abzuschließen, ohne alle nicht-optionalen Fragen beantwortet zu haben. Eine spätere Bearbei-

tung der Antworten ist dann nicht mehr möglich (So wird verhindert, dass ein Nutzer die Antworten verändern kann, wenn sich zu einem späteren Zeitpunkt oder im Laufe einer Studie etwas verändert).

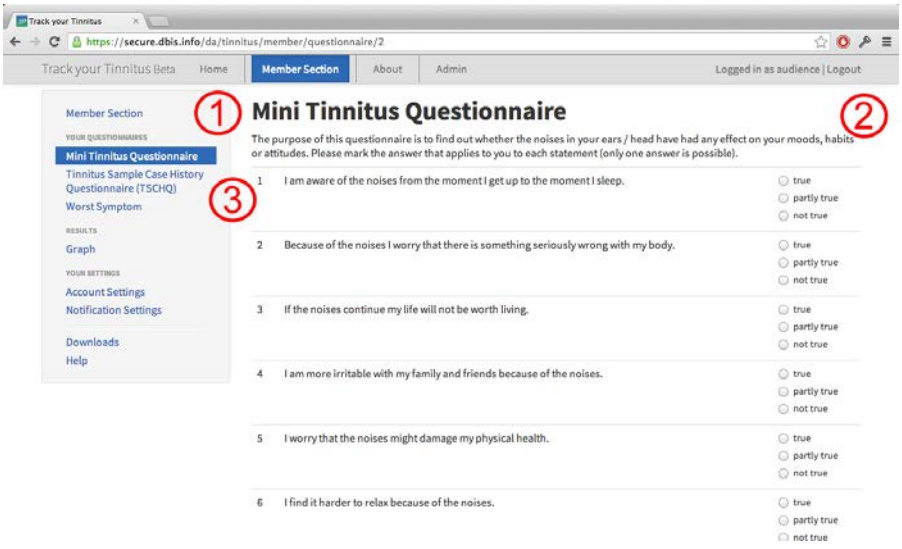

Abbildung 4.3.: Ausfüllen eines statistischen Fragebogens

4.1.4. Antwortmöglichkeiten bei statistischen Fragebögen

Ein statistischer Fragebogen kann entweder einheitliche Antwortmöglichkeiten für alle Fragen haben, oder für jede Frage, je eine unterschiedliche Antwortmöglichkeit. Track Your Tinnitus bietet sechs verschiedene Antwortmöglichkeiten, die im Administrationsbereich (siehe Kapitel 4.1.6) für jeden Fragebogen konfiguriert werden können:

1. **Text:** Als Antwort kann ein Text in eine Zeile eingegeben werden. Die Eingabe wird dabei nicht eingeschränkt oder validiert. Bei nicht-optionalen Fragen wird beim Speichern lediglich überprüft, ob ein Text eingegeben wurde.
2. **Text (mehrzeilig):** Unterscheidet sich von dem Antworttyp "Text" nur in der Darstellung des Benutzerelements auf der Webseite. Hier wird ein größeres Textfeld angezeigt, das die Eingabe von mehreren Zeilen Text erlaubt. In den Apps unterscheiden sich "Text" und "Text (mehrzeilig)" im User-Inferface nicht.
3. **Radio Buttons:** Es gibt mehrere vorgegebenen Antwortmöglichkeiten, von denen allerdings nur eine ausgewählt werden kann. Handelt es sich um eine nicht-optionale Frage, wird beim Speichern überprüft, ob ein Benutzer eine Antwort ausgewählt hat.
4. **Checkboxes:** Ähnlich wie bei der Antwortmöglichkeit "Radio Buttons" gibt es ebenfalls mehrere vorgegebenen Antworten, von denen allerdings mehrere ausgewählt werden können. Bei einer nicht-optionalen Frage wird überprüft, ob mindestens eine Antwort ausgewählt wurde.
5. **Slider:** Ein spezielles Element um Werte auf einer Skala anzugeben, z.B. von 0% bis 100%. Allerdings wird die Skala meist anhand von Beschreibungen des besten, bzw. schlechtesten

Falls definiert. In den Apps wird initial kein Button als Startposition angezeigt. Dieser erscheint, nachdem der Benutzer das erste mal mit dem Slider interagiert hat.

6. **Date Picker:** Als Antwort kann ein Datum angegeben werden, z.B. für die Angabe eines Geburtsdatums. Diese Antwortmöglichkeit wird immer durch ein Textfeld repräsentiert, in dem nur Datumsangaben in einem vorgegebenen Format eingetragen werden können. Auf der Webseite wird dies durch eine kalenderähnliche Ansicht dargestellt, mit der ein Benutzer nacheinander Jahr, Monat und Tag auswählen kann. In den Apps wird diese Eingabe durch einen *DatePicker* (Abbildung 4.4) realisiert [App13c][Gool].

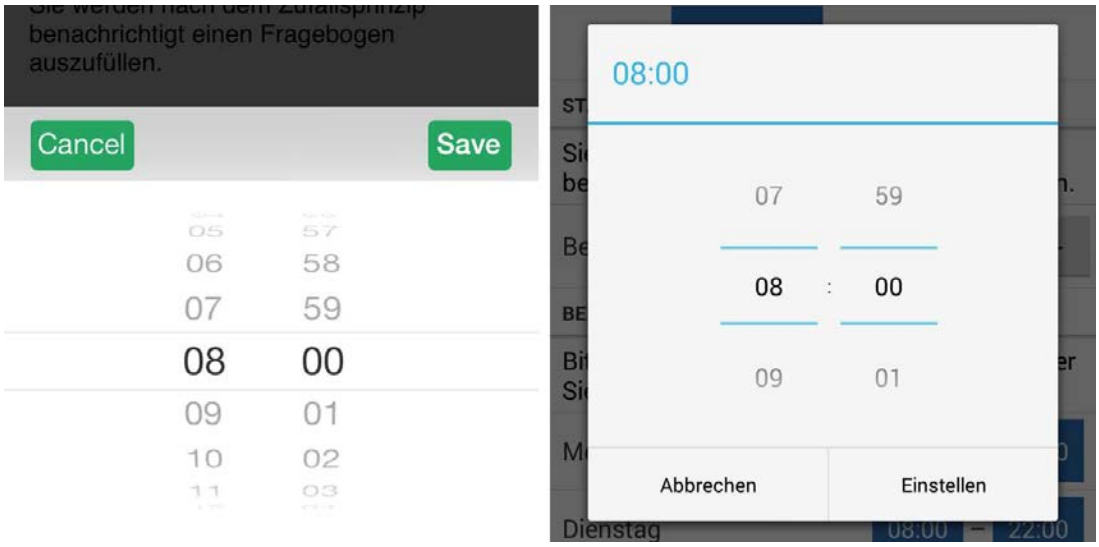

Abbildung 4.4.: DatePicker auf iOS und Android

4.1.5. Ergebnisse

Die Daten aus den Apps zur Überwachung der Schwankungen der Tinnituswahrnehmung werden automatisch auf den Server übertragen. Diese Daten werden in der Ergebnisdarstellung auf der Webseite visualisiert (Abbildung 4.5). Je Frage gibt es ein Diagramm, das die Daten auf einer Zeitlinie anzeigt. Abbildung 4.5 zeigt einen Ausschnitt dieser Visualisierung mit zwei Fragen, die in der App mit einem Slider eingegeben werden. Die genauen Zahlenwerte bleiben verborgen. Die Y-Achse hat einen Wertebereich von “Minimum” bis “Maximum”, die X-Achse zeigt den Datumsbereich der Datensätze. Auch die Antworten auf die zwei Fragen des Self-Assessment-Tests werden ebenfalls in einem Diagramm visualisiert. Eine genaue Zuordnung der Antworten zu den Self-Assessment-Manikins[1] ist hier nicht möglich. Um genauere Auswertungen erstellen zu können, kann ein Benutzer seine Daten als CSV-Datei (z.B. für den Excel-Import) im Downloadbereich herunterladen.

[1] Das Self-Assessment Manikin (SAM) ist ein sprachfreies Verfahren, um die Dimensionen Freude (Pleasure), Erregung (Arousal) und Dominanz (Dominance) affektiver Reaktionen zu erfassen [Ins09]. Track Your Tinnitus benutzt eine modifizierte 9-Punkte Skala für Freude und Erregung. Dominanz wird nicht abgefragt [Irt08].

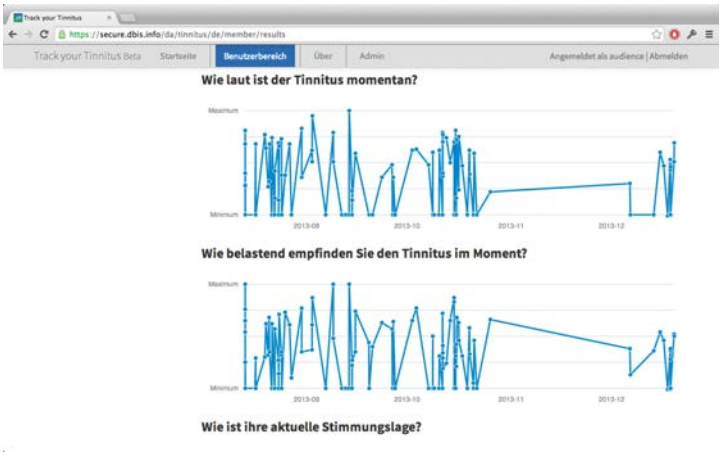

Abbildung 4.5.: Ergebnisdarstellung

4.1.6. Administrationsoberfläche

Die erste Seite der Administrationsoberfläche (Abbildung 4.6) zeigt eine Statistik zu Benutzung der Plattform, darunter die Anzahl der Benutzer aufgeschlüsselt nach Smartphone Betriebssystem, die Anzahl an beantworteten Fragen sowie die Anzahl der Datensätze aus dem Fragebogen zur Überwachung der Tinnituswahrnehmung aus der App.

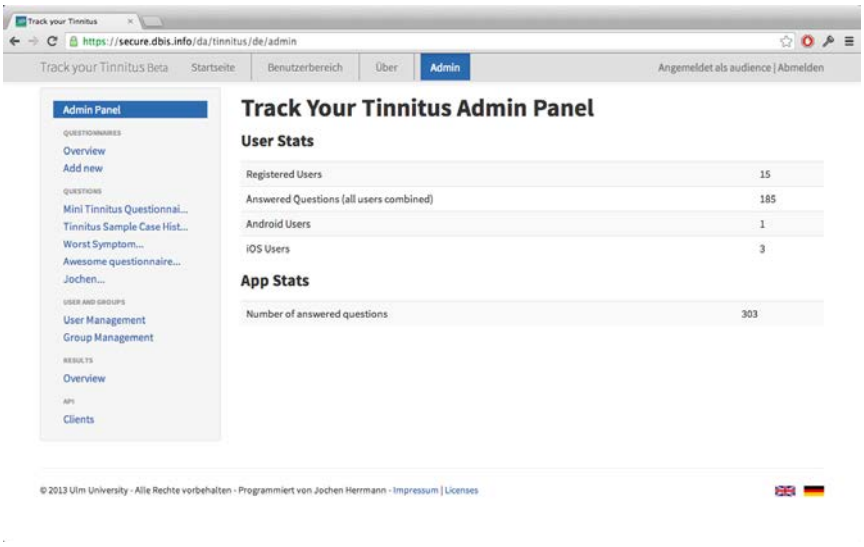

Abbildung 4.6.: Track Your Tinnitus Administrationsoberfläche

In der Track Your Tinnitus Administrationsoberfläche können statistische Fragebögen erstellt bzw. bearbeitet werden und Fragen innerhalb eines Fragebogens verwaltet werden. Außerdem gibt es eine Benutzer- und Gruppenverwaltung, sowie eine Ansicht um neue Clients für den API Zugang zu erstellen.

4.1.6.1. Verwaltung der statistischen Fragebögen

In der Verwaltung der statistischen Fragebögen können neue Fragebögen erstellt oder bestehende bearbeitet werden. Jeder Fragebogen hat einen Titel und eine Beschreibung. Zusätzlich kann ein Zeitintervall eingegeben werden, nach dem der Fragebogen ein weiteres Mal ausgefüllt werden kann. Um zu vermeiden, dass bei einem Fragebogen mit einheitlichem Antworttyp dieser bei jeder einzelnen Frage konfiguriert werden muss, kann zusätzlich ein Antworttyp mit entsprechender Konfiguration angegeben werden, der dann für alle Fragen dieses Fragebogens gilt. Da Track Your Tinnitus mehrsprachig ausgelegt ist, können die oben genannten Daten ebenfalls in weiteren Sprachen eingegeben werden. Ein Fragebogen hat zusätzlich eine Option, mit der er sich veröffentlichen lässt. Ist ein Fragebogen nicht als öffentlich markiert erscheint er weder auf der Webseite noch in den Apps. Außerdem lässt sich jeder Fragebogen einer bestimmten Benutzergruppe zuordnen. Dadurch kann ein Fragebogen auch nur für bestimmte Studien, oder für bestimmte Benutzergruppen freigegeben werden.

4.1.6.2. Verwaltung der Fragen eines Fragebogens

Die Fragen, die einem Fragebogen zugeordnet sind, können ebenfalls im Administrationsbereich erstellt, bearbeitet und entfernt werden. Jede Frage besteht aus der Frage selbst, sowie einer optionalen Beschreibung. Falls der Fragebogen keine einheitlichen Antwortmöglichkeiten hat, kann dieser je Frage konfiguriert werden. Zusätzlich kann jede Frage als optional markiert werden. Der Fragebogen wird beim Beantworten durch einen Benutzer abgeschlossen, auch wenn diese Frage nicht beantwortet wurde. In der Übersicht der Fragen eines Fragebogens können die Fragen zusätzlich in ihrer Reihenfolge per Drag-and-Drop verändert werden. Wie ein Fragebogen kann auch jede Frage lokalisiert werden.

4.1.6.3. Verwaltung von Benutzern und Benutzergruppen

Die Benutzerübersicht zeigt neben der E-Mail Adresse auch das Datum der Anmeldung sowie die Benutzergruppen, welcher ein Benutzer zugeordnet ist. Jeder Benutzer kann hier einer Gruppe zugeordnet, oder aus einer Gruppe entfernt werden. Es ist außerdem möglich, Benutzer aus der Datenbank zu entfernen. In dieser Ansicht kann auch nach bestimmten Benutzern gesucht werden.

4.1.6.4. Erstellung eines neuen API Clients

Die API von Track Your Tinnitus ist die Schnittstelle zu den Apps. Jegliche Daten der Apps werden über diese Schnittstelle an den Server übertragen. Zur Authentifizierung eines Benutzers wird das OAuth2 Verfahren eingesetzt [Har12]. Damit der Server eine Anfrage zum Login akzeptiert benötigt jede App nicht nur Benutzername und Kennwort, sondern auch eine Client-ID und einen geheimen Client-Secret. In diesem Bereich der Administrationsoberfläche können neue Clients erstellt werden, dabei können Client-ID und Client-Secret selbst eingegeben werden. Eine zusätzliche Callback-URL muss ebenfalls angegeben werden, wobei der Authentifizierungsmechanismus mit Benutzername und Passwort in den Apps verwendet wird und dafür eine Callback-URL nicht nötig ist.

4.2. Vorstellung der Apps für iOS und Android

Die Überwachung der Schwankungen der Tinnituswahrnehmung ist nur auf dem Smartphone möglich. Hierfür wurde eine App für iOS und Android entwickelt. Aus diesem Grund ist auch der Fragebogen zur Überwachung der Tinnituswahrnehmung (Kapitel 4.2.5) die Hauptfunktion der Apps. Die iOS Version unterstützt lediglich ein iPhone bzw. einen iPod Touch. Auf dem iPad wird eine vergrößerte Version im Stil einer iPhone App angezeigt. Es wird iOS6 und iOS7 unterstützt. Die Android Version unterstützt jegliche Android Geräte ab Android Version 2.3, wurde aber für die Darstellung auf Smartphones optimiert.

4.2.1. Anmeldung mit Benutzername und Kennwort

Ohne eine gültige Anmeldung mit Benutzername und Kennwort lässt sich die App nicht benutzen. Daher ist die Anmeldung (Abbildung 4.7) die erste Ansicht, mit der ein Benutzer interagieren kann. Ein Benutzer kann sich entweder direkt anmelden, oder erreicht über den Button "Registrierung" die Ansicht, in der er ein Benutzerkonto auf dem Server erstellen kann. Nach der Eingabe von Benutzername und Kennwort meldet sich der Benutzer mit dem Button "Login" am Server an. Sollte das Gerät aktuell keine funktionierende Internetverbindung haben, oder Benutzername und Kennwort falsch sein, wird eine entsprechende Fehlermeldung ausgegeben.

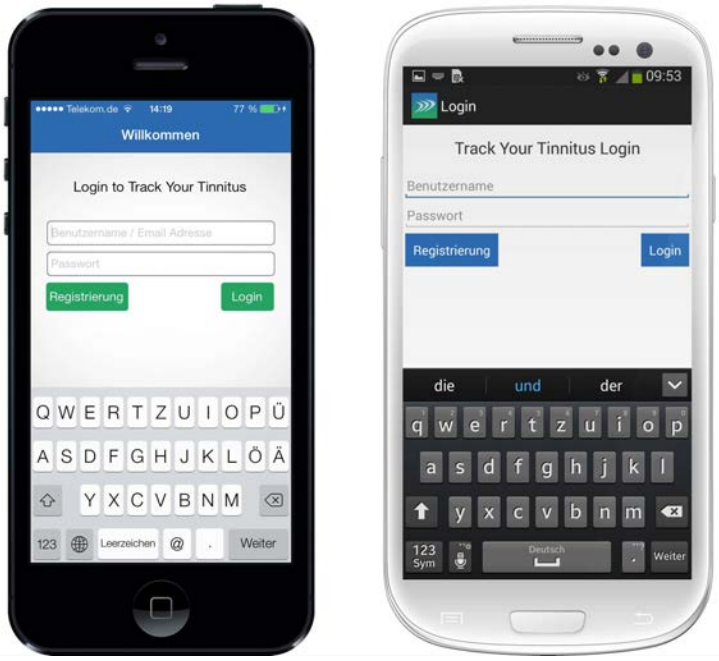

Abbildung 4.7.: Login Ansicht auf iOS und Android

Nach der erfolgreichen Anmeldung werden automatisch die aktuellen statistischen Fragebögen heruntergeladen. Falls der Benutzer bereits auf der Webseite diese Fragebögen beantwortet hat, werden auch die entsprechenden Antworten heruntergeladen. Da nicht davon ausgegangen werden kann, dass der Benutzer sich das erste Mal in der App anmeldet, werden zusätzlich eventuell vorhandene Antworten auf den Fragebogen zur Überwachung der Schwankungen der

Tinnituswahrnehmung heruntergeladen, um eine konsistente Darstellung der Ergebnisse in der App zu gewährleisten.
Unterschiede zwischen der iOS Version und der Android Version der App sind nur optisch vorhanden.

4.2.2. Registrierung in der App

Für die Registrierung bei Track Your Tinnitus muss ein Benutzer lediglich einen Benutzernamen, seine E-Mail Adresse und ein Passwort angeben. Um falsche Eingaben zu verhindern, müssen E-Mail Adresse und Passwort jeweils zwei Mal eingegeben werden. Aufgrund von Restriktionen der Universität Ulm wird die E-Mail Adresse nicht automatisch in der Datenbank gespeichert. Die E-Mail Adresse wird daher sofort nach der Bestätigung durch den Benutzer aus der Datenbank gelöscht. Ein Benutzer kann aber auch explizit zustimmen, dass seine E-Mail Adresse gespeichert wird. Der Button "Registrieren" überprüft, ob alle Daten eingegeben wurden und überträgt diese Daten an den Server. Im Erfolgsfall wird eine Meldung angezeigt, dass eine Bestätigung per E-Mail an den Benutzer gesendet wurde. Bei fehlenden Daten, wenn der Benutzername oder die E-Mail Adresse bereits registriert sind, oder bei fehlender Internetverbindung des Smartphones, wird eine entsprechende Fehlermeldung ausgegeben.
Wie in Abbildung 4.8 gezeigt, unterscheiden sich die iOS und Android Version in der Navigation zurück zur Anmeldung (1) und in der Art des User-Interface Elements für die Zustimmung zur Speicherung der E-Mail Adresse (2).

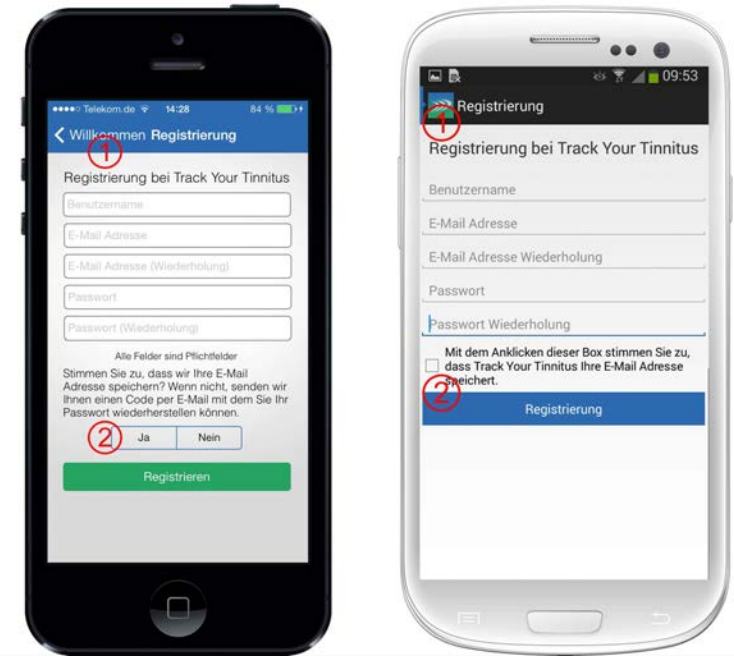

Abbildung 4.8.: Registrierung in der App auf iOS und Android

In der iOS Version wird ein *UINavigationController* verwendet, der automatisch einen Button am oberen Bildschirmrand anzeigt, mit dem ein Benutzer zurück zur Anmeldung navigieren kann. Da jedes Android Gerät eine physikalische "Zurück"-Taste hat, kann der Benutzer unter Android mit dieser zurück zur Anmeldung navigieren. Außerdem bietet Android ebenfalls einen

"Zurück"-Button, der unter dem App-Icon am oberen Rand liegt und durch einen schmalen Balken links neben dem Icon visualisiert ist.
Unter iOS wird ein *UISegmentedControl* verwendet, mit dem ein Benutzer angeben muss, ob seine E-Mail Adresse gespeichert werden darf. Android bietet hingegen eine *Checkbox*, mit der ein Benutzer zustimmen kann (2).

4.2.3. Statistische Fragebögen

Nach der Anmeldung wird der Benutzer zum ersten statistischen Fragebogen geleitet. Es wird eine Liste der Fragen präsentiert, wobei die Frage immer mit grau hinterlegt ist. Die Antwortmöglichkeiten, bei denen der Benutzer interagieren muss, sind weiß hinterlegt (Abbildung 4.9). Die Konfiguration der Fragebögen wird auf dem Server vorgenommen. Nach dem Ausfüllen des Fragebogens kann dieser mit dem Button "Speichern" gespeichert und die Antworten an den Server gesendet werden. Danach erscheint der nächste Fragebogen. Der Benutzer muss alle statistischen Fragebögen beantworten, bevor er die Schwankungen seiner Tinnituswahrnehmung überwachen kann. Sind nicht alle nicht-optionalen Fragen beantwortet, wird eine Fehlermeldung ausgegeben.

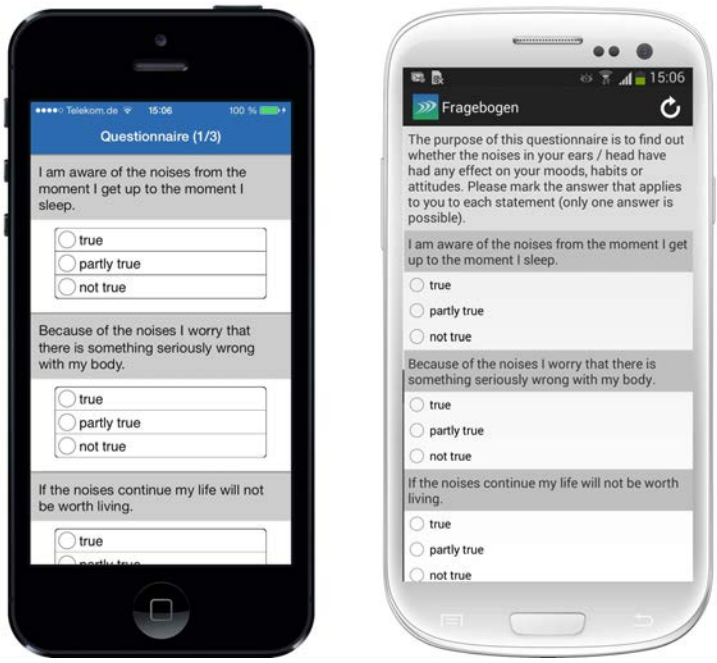

Abbildung 4.9.: Statistischer Fragebogen auf iOS und Android

Wie in Kapitel 4.1.3 beschrieben, kann ein Benutzer diese statistischen Fragebögen ebenfalls auf der Webseite beantworten. Ist ein Fragebogen auf der Webseite schon vollständig beantwortet, wird er in der App nicht mehr angezeigt. Wenn ein Fragebogen auf der Webseite unvollständig ausgefüllt und gespeichert wurde, werden die bereits beantworteten Fragen in der App entsprechen vorausgefüllt. Ein Wechsel zwischen App und Webseite, während ein Fragebogen ausgefüllt wird, ist möglich. Die Synchronisation muss allerdings manuell gestartet werden.
Um die Synchronisation zwischen App und Server zu starten, wird in der iOS Version die sogenannte "Pull-To-Refresh"-Geste verwendet. Dabei streicht der Benutzer so weit nach unten,

bis der obere Rand des Fragebogens überschritten wird. Ab einer bestimmten Überschreitung werden die Antworten neu geladen. In der Android Version befindet sich in der rechten oberen Ecke ein Button, um die Synchronisation manuell zu starten. Auf Geräten mit kleineren Displays wird dieser Button im Kontextmenü angezeigt.

4.2.4. Hauptmenü

Das Hauptmenü (Abbildung 4.10) bietet einem Benutzer die Möglichkeit, zu den verschiedenen Bereichen der App, wie z.B. den Benachrichtigungseinstellungen zu gelangen. Die Menüpunkte sind auf iOS und Android identisch. Die Apps unterscheiden sich in der Art, wie das Hauptmenü geöffnet wird.

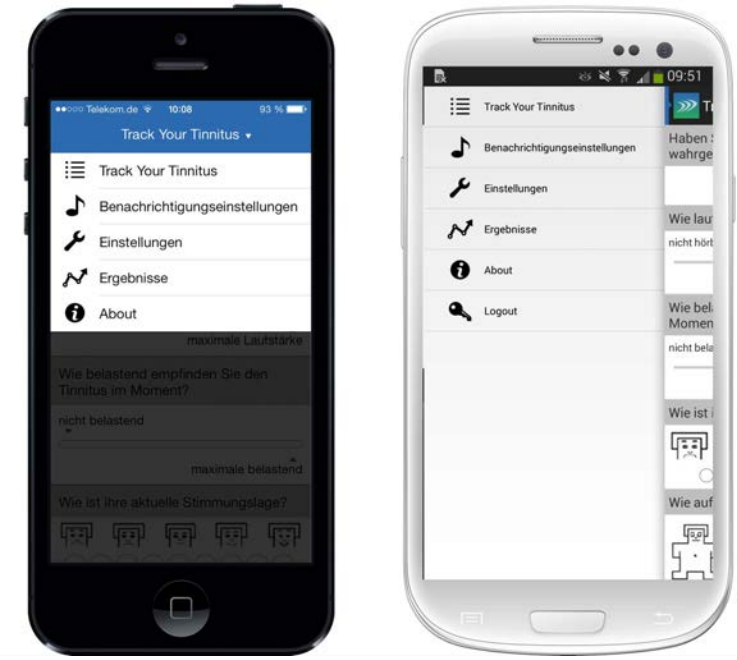

Abbildung 4.10.: Hauptmenü der App auf iOS und Android

In der iOS Version wurde die Titelleiste so verändert, dass der Titel einer Ansicht auf Berührungen reagiert. Dadurch öffnet sich das Menü in einer Animation von oben nach unten. Der Inhalt, der gerade angezeigt wird, wird dabei ausgeblendet. Nach dem Auswählen eines Menüpunktes wird das Hauptmenü mit der umgekehrten Animation wieder geschlossen und der entsprechende Inhalt angezeigt.
Auf Android Geräten ist das Hauptmenü einerseits über den Button in der oberen linken Ecke des Bildschirms, sowie über die physikalische Menü-Taste des Gerätes erreichbar. Das Öffnen wird zusätzlich animiert, indem der Inhalt nach rechts verschoben wird und das Menü links erscheint. So hat der Benutzer den Eindruck, das Menü wäre hinter dem eigentlichen Inhalt verborgen. Nach dem Auswählen eines Menüpunktes verschwindet das Menü wieder hinter dem Inhalt und die ausgewählte Ansicht wird geöffnet. Die Android App kann mit der physikalischen "Zurück"-Taste am Gerät beendet werden, allerdings nur, wenn der Benutzer den Fragebogen zur Überwachung der Tinnituswahrnehmung geöffnet hat. Wenn der Benutzer in einer anderen

Ansicht ist, bewirkt das Betätigen der "Zurück"-Taste den Wechsel des Inhalts zum Fragebogen zur Überwachung der Tinnituswahrnehmung.

4.2.5. Fragebogen zur Überwachung der Tinnituswahrnehmung

Der Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung ist der Hauptteil der Apps. Wenn sich ein Benutzer in der App bereits angemeldet hat, wird beim Start der App oder bei der Interaktion mit einer Benachrichtigung, dieser Fragebogen angezeigt. Mit diesem Fragebogen kann ein Benutzer zu unterschiedlichen Zeiten seinen aktuellen Status speichern. Abbildung 4.11 zeigt den oberen Teil des Fragebogens. Dieser Fragebogen besteht aus sieben Fragen, die immer enthalten sind:

1. Haben Sie gerade den Tinnitus bewusst wahrgenommen? - Ja/Nein Antworttyp
2. Wie laut ist der Tinnitus momentan? - Slider Antworttyp
3. Wie belastend empfinden Sie den Tinnitus im Moment? - Slider Antworttyp
4. Wie ist ihre aktuelle Stimmungslage? - Frage 1 des Self-Assessment-Test - Radio Buttons
5. Wie aufgeregt sind sie gerade? - Frage 2 des Self-Assessment-Test - Radio Buttons
6. Wie gestresst fühlen Sie sich gerade? - Slider Antworttyp
7. Wie sehr haben Sie sich auf das konzentriert, was Sie gerade tun? - Slider Antworttyp

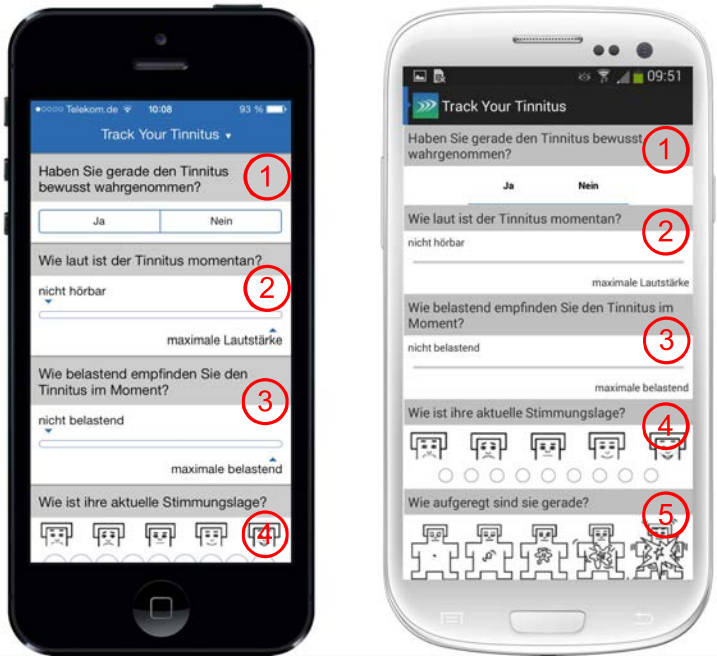

Abbildung 4.11.: Fragebogen zur Überwachung der Tinnituswahrnehmung auf iOS und Android

Zu den statistischen Fragebögen gehört auch eine Frage nach dem "Schlimmsten Symptom" (siehe Kapitel A.3). Wenn ein Benutzer diese Frage mit einem bestimmten Symptom beantwortet hat, wird eine zusätzliche Frage angezeigt, die genau dieses Symptom abfragt. Beispiel: Sollte ein Benutzer unter Schlafmangel aufgrund des Tinnitus leiden, lautet die letzte Frage:

“Fühlen Sie sich ausgeschlafen?”.
Ein Benutzer kann am Ende des Fragebogens seine Antworten speichern. Nachdem die Antworten gespeichert wurden, wird eine Meldung angezeigt, auf der ein Countdown von drei Sekunden läuft, bis die App automatisch geschlossen wird. Wenn ein Benutzer in dieser Zeit das Hauptmenü öffnet und einen Menüpunkt auswählt, wird der Countdown beendet und die App bleibt geöffnet. Falls das Speichern vom Benutzer versehentlich vergessen wird und er die App beendet, werden die Antworten automatisch gespeichert.
Unterschiede zwischen den verschiedenen Betriebssystemen der Apps sind nur optisch erkennbar. Die Funktionalität ist identisch.

4.2.6. Benachrichtigungseinstellungen

Die Track Your Tinnitus Apps senden dem Benutzer Benachrichtigungen, um ihn daran zu erinnern, den Fragebogen zur Tinnitusüberwachung neu auszufüllen. Die Zeiten, zu denen diese Benachrichtigungen auf dem Smartphone erscheinen, können in den Benachrichtigungseinstellungen konfiguriert werden.
Mit der ersten Option (1) in Abbildung 4.12 lassen sich diese Benachrichtigungen generell aus- bzw. einschalten. Bei der Benachrichtigungsart (2) hat ein Benutzer zwei Möglichkeiten:

1. Standard: Bei der Standardeinstellung wir ein Benutzer nach dem Zufallsprinzip in einem selbst definierbaren Zeitraum eines Tages benachrichtigt (Kapitel 4.2.6.1).
2. Benutzerdefiniert: Bei dieser Einstellung kann ein Benutzer die Benachrichtigungszeiten in einem Kalender fest vorgeben (Kapitel 4.2.6.2).

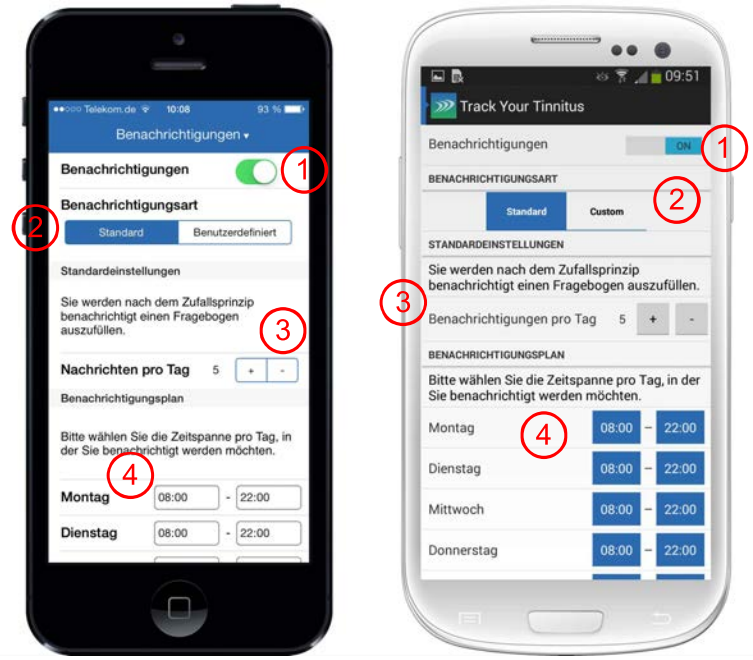

Abbildung 4.12.: Benachrichtigungungseinstellungen auf iOS und Android

Je nachdem welche Einstellung ein Benutzer bei der Benachrichtigungsart wählt, ändert sich der restliche Inhalt dieser Ansicht. Die Einstellungen werden, wie z.B. in den Telefoneinstellungen des iPhones, sofort bei einer Änderung wirksam. Ein "Speichern" Button ist nicht vorhanden.

4.2.6.1. Benachrichtigungsart Standard

Abbildung 4.12 zeigt die weiteren Optionen, wenn die Benachrichtigungsart "Standard" eingestellt ist. Zuerst muss der Benutzer die Anzahl der Benachrichtigungen einstellen (3), die an einem Tag der Woche gesendet werden. Danach kann für jeden Tag der Woche ein Zeitintervall anhand eines Start- und Endzeitpunktes angegeben werden (4). Innerhalb dieses Zeitraumes werden die Benachrichtigungen nach dem Zufallsprinzip gesendet. Diese Uhrzeiten werden direkt mit einem *DatePicker* (Abbildung 4.4) eingestellt. Auf die genaue Uhrzeit, zu welcher eine Benachrichtigung gesendet wird, hat ein Benutzer bei dieser Benachrichtigungsart keinen Einfluss.
Ein Unterschied zwischen der iOS- und Android-Version ist nur optisch erkennbar.

4.2.6.2. Benachrichtigungsart Benutzerdefiniert

Bei der Benachrichtigungsart "Benutzerdefiniert" erscheint in dieser Ansicht nur eine Option, mit der der Benutzer zu einem Kalender gelangt. Dieser Kalender öffnet sich zuerst in der Wochenansicht (Abbildung 4.13). So erhält der Benutzer eine Übersicht, wann er in einer Woche, an welchen Tagen, zu welchen Uhrzeiten, benachrichtigt wird. Durch einen Touch auf den Kalender wird ein erweiterter *DatePicker* geöffnet, mit dem sich der Tag und die Uhrzeit einstellen lässt, um so einen neuen Benachrichtigungszeitpunkt zu erstellen. Dieser *DatePicker* (Abbil-

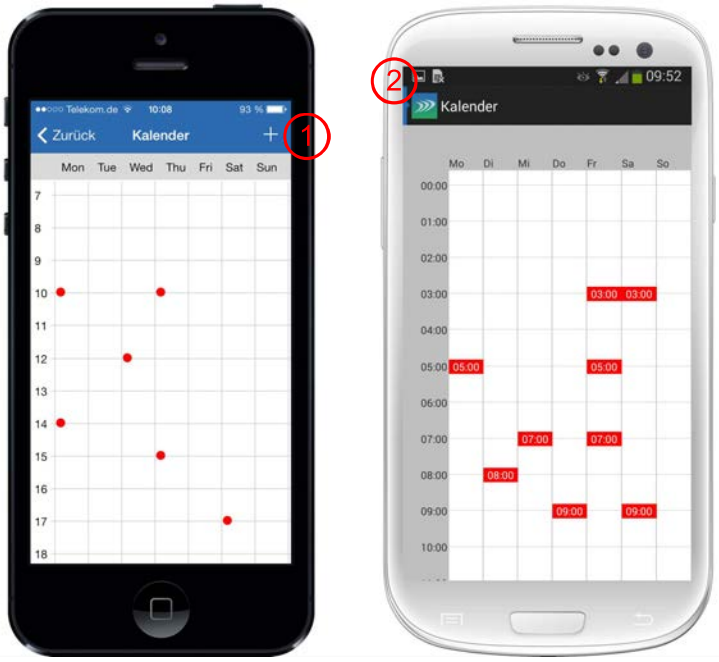

Abbildung 4.13.: Kalender Ansicht auf iOS und Android

dung 4.4) öffnet sich immer mit dem vom Benutzer berührten Tag und ungefährer Uhrzeit. Mit einem Long-Touch auf die Spalte eines Tages wird die Tagesansicht geöffnet. Auch hier lassen sich neue Zeitpunkte durch einen Touch auf eine bestimmte Uhrzeit erstellen.
Die iOS Version bietet zusätzlich zur Funktionalität eines Touches noch einen "+"-Button (1) in der Navigationsleiste, um neue Einträge zu erstellen. Außerdem lassen sich Zeitpunkte nur in der Tagesansicht mit einem Touch bearbeiten oder löschen. Da iOS Geräte keine physikalische "Zurück"-Taste haben, wird in der unteren rechten Ecke der Tagesansicht ein Button eingeblendet, der die Navigation zurück zur Wochenansicht erlaubt. Diese Navigation ist nicht mit dem "Zurück"-Button in der Navigationsleiste möglich.
Auf einem Android-Gerät lassen sich die Zeitpunkte in der Wochenansicht, sowie in der Tagesansicht, mit einem Touch auf einen Zeitpunkt bearbeiten oder löschen. Das Navigieren von der Tagesansicht zurück zur Wochenansicht ist über die physikalische "Zurück"-Taste des Android Gerätes sowie über den "Zurück"-Button (2) in der Navigationsleiste möglich.

4.2.7. Einstellungen

Die Einstellungen der Apps (Abbildung 4.14) bestehen aus zwei Bereichen: "Klingelton" und "Privatsphäre". Im ersten Bereich kann der Klingelton der Benachrichtigungen eingestellt werden. Zur Auswahl stehen sieben eigene Klingeltöne und der Standardklingelton, der in den Einstellungen des Gerätes allgemein festgelegt ist. Falls ein Benutzer aufgrund seines Tinnituses einen Klingelton nicht deutlich wahrnehmen kann, kann dieser hier umgestellt werden. Der Haken auf der rechten Seite eines Klingeltons visualisiert, dass dieser der aktuell ausgewählte ist.

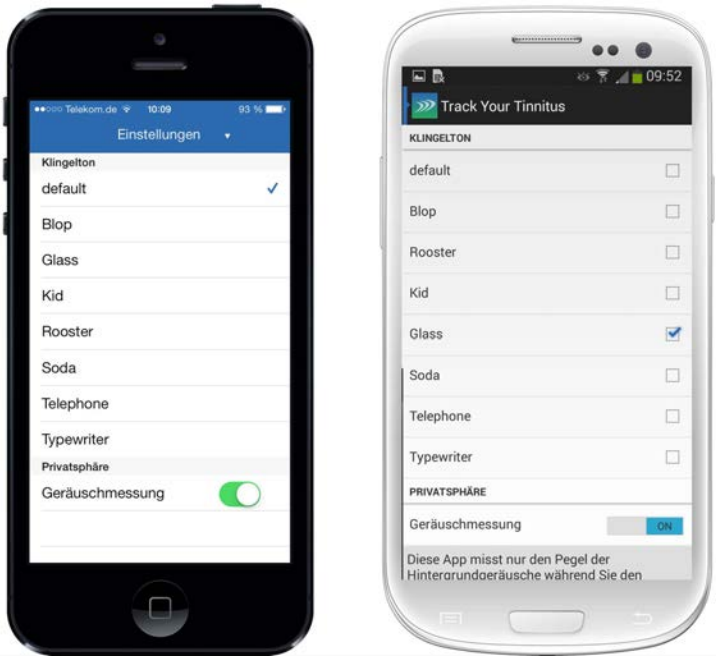

Abbildung 4.14.: Einstellungen auf iOS und Android

Im zweiten Bereich gibt es nur eine Option, um die Geräuschpegelmessung, die während des Ausfüllens des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung

stattfindet, auszuschalten. Darunter wird noch einmal deutlich gemacht, dass keine Aufzeichnung der Hintergrundgeräusche stattfindet, sondern nur der Geräuschpegel gemessen wird.

4.2.8. Ergebnisse

Die Antworten, die ein Benutzer in dem Fragebogen zur Überwachung der Tinnituswahrnehmung angibt, können sowohl in den Apps, wie auch auf der Webseite betrachtet werden. Die Apps bieten zwei verschiedene Ansichten der Antworten. Einerseits eine Diagrammdarstellung und andererseits eine Timeline-Ansicht, die an Zeitleisten aus sozialen Netzwerken erinnert [Fac].
In der iOS App erfolgt die Umschaltung zwischen diesen Ansichten im unteren Bereich über ein sogenanntes *SegmentedControl*. Die Android App bietet zum Umschalten ein Drop-Down-Menü in der Titelleiste.

4.2.8.1. Ergebnisse in Diagrammen

In der Ergebnisansicht mit Diagrammen (Abbildung 4.15) werden die Antworten für jede Frage in einem separaten Diagramm visualisiert. Dabei ist die X-Achse der zeitliche Verlauf, die Y-Achse gibt den eingetragenen Wert an. Bei Ja/Nein-Fragen gibt es nur zwei mögliche Werte,

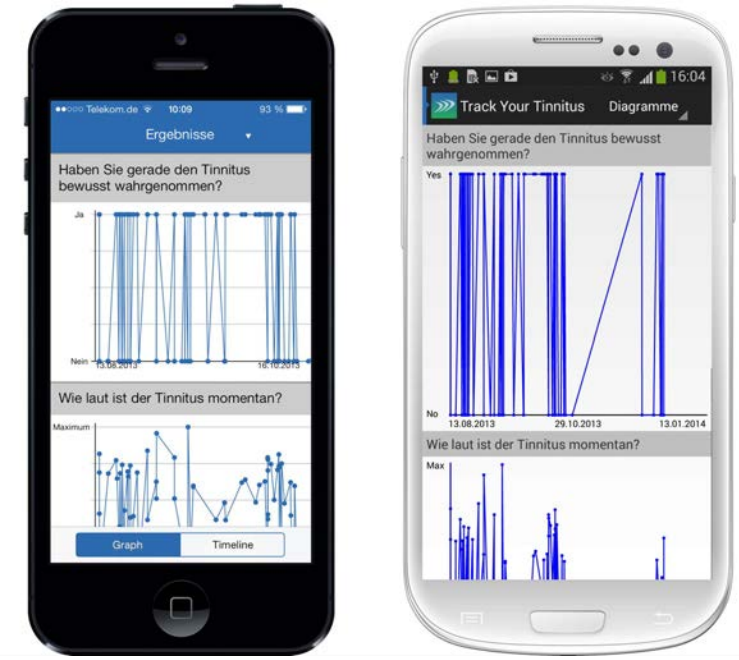

Abbildung 4.15.: Ergebnisse (Diagramme) auf iOS und Android

die dann den höchsten bzw. niedrigsten Wert im Diagramm annehmen. Fragen, die mit einem Slider beantwortet werden, können jegliche Werte annehmen. Für die Antworten auf den Self-Assessment-Test wird der Wertebereich der Y-Achse in acht Teile aufgeteilt, um so jede Antwort darstellen zu können. Dabei ist aber eine genaue Zuordnung der gegebenen Antwort zu einem Self-Assessment-Manikin nicht möglich. Das Soundlevel wird in dieser Ansicht nicht visualisiert.

Auf iOS Geräten sind die Diagramme breiter als das Display und lassen sich so horizontal scrollen. Auf Android Geräten ist dies aus Performance-Gründen nicht möglich.

4.2.8.2. Ergebnisse in einer Timeline

In der Ergebnisansicht in einer Timeline (Abbildung 4.16) wird jeder Datensatz in einem eigenen Rahmen dargestellt, der auf einer Zeitlinie erscheint. Diese Datensätze sind zeitlich absteigend sortiert, das heißt der neueste Datensatz erscheint in der ersten Zeile, der älteste in der letzten. Über einem Rahmen steht das Datum und die Uhrzeit eines Datensatzes. Im Rahmen werden die gegebenen Antworten einzeln aufgeführt. Da ein Slider optisch keinen definierten Wertebereich hat, werden diese Werte in Prozent angegeben. Je nach Wert wird der Hintergrund der

Abbildung 4.16.: Ergebnisse (Timeline) auf iOS und Android

entsprechenden Zeile eingefärbt, wobei eine Mischung aus Grün und Rot zum Einsatz kommt. Im Bereich von 0% bis 50% erscheinen je nach Wert Farbmischungen von Grün bis Gelb. Im Bereich von 50% bis 100% werden Farben von Gelb bis Rot verwendet. Das Soundlevel wird, im Gegensatz zur Diagramm-Ansicht, ebenfalls angezeigt. Eine Visualisierung, wie bei den Antworten, findet hier nicht statt. Der größte Balken am unteren Rand eines Rahmens gibt an, ob der Tinnitus wahrgenommen wurde. Die zwei Fragen des Self-Assessment-Tests werden auf der rechten Seite des Rahmens mit den ausgewählten Self-Assessment-Manikins visualisiert.
Einen Unterschied zwischen den App Versionen gibt es lediglich beim Schnell-Scrollen. Die iOS Version bietet einen Index am rechten Rand, über den ein Benutzer mit dem Finger streichen kann. Dadurch verschiebt sich der Inhalt in größeren Schritten nach oben oder unten. Die Android Version bietet ein sogenanntes “Fast-Scrolling”. Scrollt ein Benutzer einen kleineren Bereich nach oben oder unten, erscheint am rechten Rand ein Scrollbalken, der ähnlich wie im Web-Browser nach unten gezogen werden kann.

4.2.9. About

Die About-Ansicht bietet einem Benutzer weitere Informationen zum Track Your Tinnitus Projekt. Abbildung 4.17 zeigt die Übersicht der einzelnen Unterpunkte, die sich jeweils in einer neuen Ansicht öffnen. Über "Kontakt" erreicht der Benutzer ein Formular, mit dem er direkte

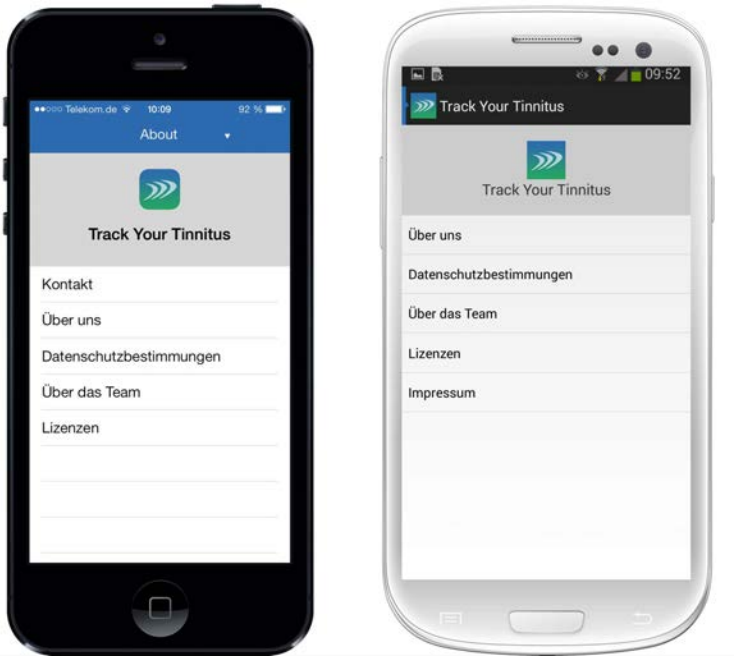

Abbildung 4.17.: About Submenü auf iOS und Android

Nachrichten an das Team von Track Your Tinnitus senden kann. Dabei ist die Angabe einer E-Mail Adresse optional. "Über uns" gibt nähere Information über das Track Your Tinnitus Projekt, "Datenschutzbestimmungen" enthält Informationen zur Verwendung der Benutzerdaten und "Über das Team" nennt die beteiligten Personen. Diese Texte, wie auch das Impressum, sind ebenfalls auf der Webseite im About-Bereich zu finden. Der Unterpunkt "Lizenzen" zeigt alle Open-Source Lizenzen der Komponenten, die in der jeweiligen App verwendet werden, wie z.B. *AFNetworking* in der iOS Version oder den *ActionBarSherlock* in der Android Version.

Wo ist eigentlich die P-Taste?

Rüdiger Pryss 2013

5 Implementierung

In diesem Kapitel werden spezielle Themen in der Implementierung von Track Your Tinnitus beschrieben. Zuerst werden einige Techniken in der Entwicklung der Webseite gezeigt (Kapitel 5.1). Im zweiten Teil wird dann auf spezielle Probleme in der Entwicklung der Apps eingegangen. Diese beinhalten das Planen von Benachrichtigungen (Kapitel 5.2), das Erstellen von eigenen User-Interface Elementen (Kapitel 5.3) und das Erstellen von eigenen Controls (Kapitel 5.4).

5.1. Entwicklung der Webseite

Die Webseite von Track Your Tinnitus und die API wurden mit dem Laravel Framework Version 3 entwickelt [Otwa]. Der folgende Abschnitt zeigt, welche Vorteile das Laravel Framework bietet, wie *Controller* erstellt werden können, und wie das *Object-Relationship-Mapping* in Laravel funktioniert.

5.1.1. Einführung in das Laravel Framework

Laravel ist ein Open-Source PHP-Framework, das dem Model-View-Controller Pattern folgt. Laravel unterscheidet sich unter anderem in den folgenden Punkten von anderen PHP-Frameworks [Otwa]:

- Mit *Bundles* lassen sich neue Komponenten in das modulare System von Laravel einbinden.
- *Eloquent ORM* als *Object-Relationship-Mapping*
- Mit *Reverse Routing* lassen sich Links zu benannten Routen erstellen. Das Framework setzt automatisch die richtige URI ein.
- *Restful Controller*, um zum Beispiel GET- und POST-Anfragen im Controller zu trennen.

- *Class Auto Loading*: Klassen werden automatisch geladen, eine Konfiguration des PHP-Autoloaders ist nicht erforderlich.
- *Migrations* als Versionskontrollsystem für Datenbankschemas.

5.1.2. Controller und Restful Controller

Nach dem Model-View-Controller Pattern stellt ein *Controller* die Verbindung zwischen dem *View* und dem Datenmodel her. Ein *Controller* wird im Laravel Framework im Verzeichnis *application/controller* gespeichert und erbt direkt von der Laravel-eigenen Klasse *Controller*. Da alle Controller auch gemeinsame Methoden bieten, ist eine *Base_ Controller* Klasse bereits beim Installieren des Frameworks vorhanden. In Track Your Tinnitus erbt jede *Controller*-Klasse direkt von der *Base_ Controller*-Klasse. Listing 5.1 zeigt einen *Controller* in Laravel anhand der *Home_ Controller*-Klasse. Jede Methode, die das *action*-Keyword vorangestellt hat, ist von der Webseite aus erreichbar. Alle anderen Methoden, egal welche Sichtbarkeit diese haben, können nur innerhalb des Controllers aufgerufen werden [Otwb]. Für die Methoden in der Klasse *Home_ Controller* wurde jeweils eine Route definiert, so dass zum Beispiel die Methode *action_ index()* (Zeile 10) aufgerufen wird, wenn ein Benutzer die Startseite aufruft. Durch das Keyword *action* lassen sich Routen definieren, die unabhängig von der HTTP-Request Methode die Methode im *Controller* aufrufen.

Listing 5.1: Controller im Laravel Framework

```
1  <?php
2
3  class Home_Controller extends Base_Controller {
4
5      public function __construct()
6      {
7          parent::__contruct();
8      }
9
10     public function action_index()
11     {
12         return View::make('home.index');
13     }
14
15     public function action_about()
16     {
17         return View::make('home.about');
18     }
19
20     public function action_licenses()
21     {
22         return View::make('home.licenses');
23     }
24
25     public function action_imprint()
```

```
26      {
27          return View::make('home.imprint');
28      }
29  }
```

Anstatt des Schlüsselwortes *action* kann einer Methode auch eine HTTP-Request-Methode (POST, GET, DELETE, PUT) vorangesetzt werden. Diese Methoden werden dann nur bei einem entsprechenden HTTP-Request aufgerufen. Hierfür muss eine *Controller*-Klasse aber die Variable *$restful* auf *true* setzen. Listing 5.2 zeigt einen *Restful Controller* am Beispiel des *Login_Controller*. In Zeile 4 wird zuerst die Variable auf *true* gesetzt, um den *Controller* als Restful zu definieren. Ein Aufruf der Login Seite im Browser wird als HTTP-GET Request gesendet. Daher ist eine Methode definiert, die bei einem GET-Request aufgerufen wird (Zeilen 10 - 13). Die Daten des Login Formulars werden per POST-Request an den Server gesendet und von der entsprechenden Methode verarbeitet (Zeilen 15 - 43). So ist es möglich, den Aufruf des Login Formulars und das Absenden über die gleiche URL zu realisieren.

Listing 5.2: Restful Controller im Laravel Framework

```
1  <?php
2  class Login_Controller extends Base_Controller {
3
4      public $restful = true;
5
6      public function __construct() {
7          parent::__contruct();
8      }
9
10     public function get_index() {
11         // Display the login form
12         return View::make('login.login');
13     }
14
15     public function post_index() {
16         // Validate input
17         if (!Input::has('username') || !Input::has('password')) {
18             return Redirect::to('login') -> with('login_errors', true);
19         }
20         $email = Input::get('username');
21         $password = Input::get('password');
22         $remember = Input::get('remember');
23         $rememberbool = false;
24         if ($remember == "remember") {
25             $rememberbool = true;
26         }
27         // Try to log a user in
28         try {
29             $valid_login = Sentry::login($email, $password, $rememberbool);
30
```

```
31              if ($valid_login) {
32                  // Loged a user in successfully
33                  return Redirect::to('member');
34              } else {
35                  // Login errors
36                  $data['errors'] = "Invalid login!";
37              }
38          } catch (Sentry\SentryException $e) {
39              $data['errors'] = $e -> getMessage();
40          }
41          // Redirect a user back to the login form
42          return Redirect::to('login') -> with_input() -> with('login_errors', true);
43      }
44      // ...
45  }
```

5.1.3. Eloquent ORM

Das Laravel Framework bietet als *Object-Relationship-Mapping* das sogenannte Eloquent ORM. Generell wird eine Modellklasse pro Datenbanktabelle erstellt, die von der Klasse *Eloquent* erbt. Damit das Mapping ohne weitere Einstellungen funktioniert, muss die Datenstruktur die folgenden Voraussetzungen erfüllen:

- Jede Tabelle muss einen Primärschlüssel haben, der den Namen *id* hat.
- Jeder Tabellenname muss die Pluralform des Namen der Modellklasse sein.

Den Tabellennamen und der Name des Primärschlüssels kann aber auch als Variable in der Modellklasse angegeben werden [Otwc]. Listing 5.3 zeigt die Implementierung der Modellklasse zu den Datensätzen der Antworten auf den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung. Die zugehörige Tabelle in der Datenbank hat den Namen *standardanswers* und einen Primärschlüssel mit dem Namen *id*. So ist keine weiter Angabe dieser Namen nötig.

Listing 5.3: Eloquent ORM Modellklasse im Laravel Framework

```
1 <?php
2 class Standardanswer extends Eloquent {
3
4     public function user() {
5         return $this->belongs_to('User');
6     }
7 }
```

5.1.3.1. Abfragen des Datenmodels

Eloquent bietet diverse Methoden, um Daten aus der Datenbank zu laden. Listing 5.4 zeigt diese Methoden. Mit der *find()*-Methode (Zeile 2) lässt sich ein Datensatz anhand des Pri-

märschlüssels laden. Dabei wird ein Objekt der Klasse *Standardanswer* zurück gegeben. Diese Abfrage wird direkt in ein SQL-Statement übersetzt, welches in Zeile 1 zu sehen ist. Alle Datensätze einer Tabelle werden mit der Methode *all()* geladen (Zeile 4). Der Rückgabewert dieser Methode ist ein Array mit *Standardanswer*-Objekten oder ein leeres Array, falls keine Datensätze vorhanden sind. Eine Abfrage kann aber auch aus mehreren Methodenaufrufen bestehen, die mit *first()* bzw. *get()* abgeschlossen werden. Die Methode *first()* liefert dabei den ersten Datensatz der Abfrage als Objekt zurück, *get()* liefert ein Array der Objekte, oder ein leeres Array, falls keine Datensätze gefunden wurden. Eloquent ORM bietet unter anderen folgende Methoden, um eine Abfrage zu definieren [Otwc]:

- *where()*: Diese Methode benötigt drei Parameter: Den Namen der Spalte auf den sich diese Operation bezieht, einen Operator und einen Wert (Zeile 6). Dieser Methodenaufruf kann aber auch in einen der Form *where_ {Name der Spalte}()* zusammengefasst werden (Zeile 7).
- *or_ where()*: Where-Abfrage, die allerdings mit *OR* verknüpft werden (Zeile 8).
- *order_ by()*: Sortieren der Ergebnisse (Zeile 9).
- *take()*: Einschränken der Anzahl der Datensätze (Zeile 9). Mit einem zusätzlichen Aufruf der Methode *skip()* lässt sich eine bestimmte Anzahl von Datensätzen überspringen.

Listing 5.4: Abfragen mit Eloquent ORM im Laravel Framework

```
1 // SELECT * FROM "standardanswers" WHERE "id" = 1;
2 $standardanswer = Standardanswer::find(1);
3
4 $standardanswers = Standardanswer::all();
5
6 $standardanswer = Standardanswer::where('question1', '=', 1)->first();
7 $standardanswer = Standardanswer::where_question1(1)->first();
8 $standardanswers = Standardanswer::where('question1', '=', 1)->or_where('question2', '
      =', 1)->get();
9 $standardanswers = Standardanswer::order_by('save_date', 'desc')->take(10)->get();
```

5.1.3.2. Erstellen und Bearbeiten von Datensätzen

Um einen neuen Datensatz zu erstellen, wird zuerst eine neue Instanz des entsprechenden Objektes erstellt. Listing 5.5 zeigt das Erstellen eines neuen Datensatzes in der Tabelle *standardanswers*. Nachdem eine neue Instanz des Objekts erstellt wurde (Zeile 3), werden die einzelnen Spalten direkt als Eigenschaft dieses Objekts gesetzt, wobei der Name einer Eigenschaft dem Namen der Spalte entspricht. Ein Aufruf der Methode *save()* speichert den Datensatz in der Datenbank (Zeile 8). Beim Bearbeiten eines Datensatzes wird das Objekt mit einer Abfrage aus der Datenbank geladen (siehe Kapitel 5.1.3.1). Nachdem die Eigenschaften neu gesetzt sind, wird ebenfalls die Methode *save()* des Objektes aufgerufen, welche die Änderungen in die Datenbank schreibt [Otwc].

Listing 5.5: Erstellen eines Datensatzes mit Eloquent ORM

```
1 <?php
2
```

```
3 $standardanswer = new Standardanswer;
4
5 $standardanswer.question1 = 1;
6 $standardanswer.question2 = 0.3;
7
8 $standardanswer->save();
```

5.1.3.3. Beziehungen zwischen Tabellen

Beziehungen zwischen Tabellen werden in Track Your Tinnitus in fast allen Tabellen verwendet. Zum Beispiel hat ein Antwortdatensatz auf den Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung immer eine Referenz auf den zugehörigen Benutzer, oder eine Frage ist immer einem Fragebogen zugeordnet. Laravel unterstütz drei Arten von Beziehungen:

- One-To-One
- One-To-Many
- Many-To-Many

Eine Beziehung zwischen Tabellen in einem Model wird definiert, indem eine Methode der Modellklasse hinzugefügt wird, die eine der folgenden Methoden aufruft und den Rückgabewert dieser Methode zurückliefert: *has_one(), has_many(), belongs_to()* oder *has_many_and_belongs_to()*.

One-To-One In Track Your Tinnitus findet die One-To-One Beziehung keine Anwendung. Listing 5.6 zeigt daher ein Beispiel, in dem jedem Benutzer ein Datensatz in der Tabelle *Settings* zugeordnet ist. Beim Aufruf der Methode *has_one()* wird der Name der zugehörigen Tabelle übergeben (Zeile 6).

Listing 5.6: One-To-One Beziehung mit Eloquent ORM

```
1 <?php
2
3 class User extends Eloquent {
4
5     public function settings(){
6         return $this->has_one('Settings');
7         // return $this->has_one('Settings', 'settings_foreign_key_name');
8     }
9
10 }
```

Listing 5.7 zeigt eine einfach Abfrage. Bei dieser Abfrage kann nach dem Abfragen des User-Datensatzes direkt mit dieser Methode auf den verknüpften Settings-Datensatz zugegriffen werden (Zeile 2). Eloquent ORM wandelt diese in die SQL-Statements in Zeile 3 und 4 um. Dabei wird automatisch angenommen, dass der Fremdschlüssel in der *Settings* Tabelle den Namen *user_id* hat. Ist dies nicht der Fall, kann der Name des Fremdschlüssels auch beim

Aufruf der Methode *has_ one()* angegeben werden (siehe Listing 5.6, Zeile 7). Ohne den Aufruf der *first()* Methode kann auch direkt mithilfe einer dynamischen Eigenschaft auf einen Settings Datensatz zugegriffen werden (Zeile 6).

Listing 5.7: One-To-One Beziehung mit Eloquent ORM

```
1 <?php
2 $settings = User::find(1)->settings()->first();
3 // SELECT * FROM "users" WHERE "id" = 1;
4 // SELECT * FROM "settings" WHERE "user_id" = 1;
5
6 $settings = User::find(1)->settings;
```

Um nun aus einem *Settings*-Datensatz den verknüpften Benutzer zu erhalten, muss die *Settings* Klasse um die Methode *user()* erweitert werden. Listing 5.10 zeigt die Implementierung dieser Methode. Da der Fremdschlüssel in der Tabelle Settings steht, wird die Methode *belongs_ to()* aufgerufen (Zeile 6). Der verknüpfte Benutzer kann analog zur Implementierung in Listing 5.7 abgefragt werden.

Listing 5.8: One-To-One Beziehung mit Eloquent ORM

```
1 <?php
2
3 class Setting extends Eloquent {
4
5     public function user() {
6         return $this->belongs_to('User');
7     }
8 }
```

One-To-Many Eine One-To-Many Beziehung besteht in Track Your Tinnitus zum Beispiel zwischen den Fragebögen und den Fragen. Jedem Fragebogen sind die entsprechenden Fragen zugeordnet. Jede Frage kann allerdings nur einem Fragebogen zugeordnet sein. Listing 5.9 zeigt einen Ausschnitt der Implementierung der Klasse *Question*. Um die Beziehung im Model darzustellen, wird die Methode *has_ many()* aufgerufen (Zeile 5).

Listing 5.9: One-To-Many Beziehung mit Eloquent ORM

```
1 <?php
2 class Question extends Eloquent {
3
4     public function answers() {
5         return $this->has_many('Answer');
6     }
7
8     // ...
9 }
```

Many-To-Many Bei einer Many-To-Many Beziehung wird eine zusätzliche Tabelle benötigt, die die Fremdschlüssel der verknüpften Tabellen beinhaltet. Beim Interagieren mit dem Datenmodell muss dies allerdings nicht berücksichtigt werden. In Track Your Tinnitus kann ein Fragebogen einer oder mehreren Gruppen zugeordnet werden. Jeder Gruppe kann aber auch mehreren Fragebögen zugeordnet werden. Dadurch entsteht eine Many-To-Many Beziehung. Listing 5.10 zeigt die Implementierung der Klasse *Group*. Durch die Methode *has_many_and_belongs_to()* wird diese Beziehung in der Modellklasse definiert. Ohne eine Angabe des Namens der Verknüpfungstabelle geht Eloquent ORM von einer Kombination aus den Namen der verknüpften Klassen aus (zum Beispiel *group_questionnaire*).

Listing 5.10: Many-To-Many Beziehung mit Eloquent ORM

```
1 <?php
2 class Group extends Eloquent {
3
4     public function questionnaires() {
5         return $this->has_many_and_belongs_to('Questionnaire');
6     }
7
8 }
```

Das Abfragen der verknüpften Datensätze ist in Listing 5.11 dargestellt. Analog zur Abfrage eines One-To-One verknüpften Datensatzes kann ein Array der verknüpften Datensätze in einer Many-To-Many Beziehung mit dem Aufruf der entsprechenden Methode gefolgt von *get()* erfolgen (Zeile 3). Es wird entweder ein Array der Objekte oder ein leeres Array zurückgeliefert, wenn keine Datensätze vorhanden sind. Die Abfrage kann auch kürzer mithilfe der dynamischen Eigenschaft erfolgen (Zeile 4).

Listing 5.11: Abfrage einer Many-To-Many Beziehung mit Eloquent ORM

```
1 <?php
2
3 $questionnaires = Group::find(1)->questionnaires()->get();
4 $questionnaires = Group::find(1)->questionnaires;
```

Einfügen verknüpfter Datensätze Unabhängig von der Art der Beziehung zwischen den Tabellen bietet Eloquent ORM einen einheitlichen Weg neue verknüpfte Datensätze einzufügen. Listing 5.12 zeigt die Verknüpfung eines Fragebogens mit einer Gruppe. Wenn ein neuer Fragebogen erstellt wurde (Zeile 3), kann dieser direkt mit *insert()* einer Gruppe zugeordnet werden (Zeile 8). Dabei wird zuerst der Fragebogen-Datensatz erstellt und dann ein Eintrag in der Verknüpfungstabelle erstellt. Wenn beide Datensätze bereits vorhanden sind, kann mit *attach()* auch nur ein Eintrag in der Verknüpfungstabelle erstellt werden (Zeile 10).

Listing 5.12: Einfügen einer Many-To-Many Beziehung mit Eloquent ORM

```
1 <?php
2
3 $questionnaire = new Questionnaire();
4 // set the data of the questionnaire
```

```
5
6 $group = Group::find(1);
7
8 $group->questionnaires()->insert($questionnaire);
9
10 $group->questionnaires()->attach($questionnaire_id);
```

5.2. Notifications in den Apps

Die Hauptfunktion der Track Your Tinnitus Apps ist das Ausfüllen des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung. Ein Benutzer soll in möglichst unregelmäßigen Abständen daran erinnert werden, diesen Fragebogen erneut auszufüllen. So können die Schwankungen über Wochen verteilt aufgezeichnet werden. Um einen Benutzer zu benachrichtigen, wenn die App nicht im Vordergrund ausgeführt wird, bietet iOS und Android jeweils eigene Mechanismen, die in diesem Kapitel näher erläutert werden.

5.2.1. Remote Notifications oder Local Notifications

iOS und Android bieten jeweils unterschiedliche Möglichkeiten, einem Benutzer eine Benachrichtigung anzuzeigen. Man kann generell zwischen Remote Notifications und Local Notifications unterscheiden. Remote Notifications werden von einem Server über das Internet an ein Gerät gesendet. Dabei kann der Inhalt einer Benachrichtigung auf dem Server bestimmt werden. Eine funktionierende Internetverbindung ist dabei zwingend notwendig, damit die Benachrichtigung ausgeliefert werden kann. iOS bietet hierfür den *Apple Push Notification Service* [App13a], Android biete das *Google Cloud Messaging* [Goof]. Local Notifications werden auf dem Gerät geplant, auf dem sie später auch dargestellt werden. Dabei muss allerdings schon vorher feststehen, zu welchem Zeitpunkt eine Benachrichtigung erscheinen soll. Da eine funktionierende Internetverbindung auf einem Smartphone nicht garantiert werden kann und der Benachrichtigungszeitpunkt nur von den Einstellungen auf dem Gerät abhängig ist, haben wir uns bei Track Your Tinnitus für Local Notfications entschieden.

5.2.2. Implementierung von Local Notifications

Local Notifications unter iOS und Android unterscheiden sich in der Implementierung grundlegend. iOS bietet direkt Methoden, mit denen sich eine Local Notification erstellen und planen lässt. Da Android keine direkte Methode bietet, um einen Notification zu einem bestimmten Zeitpunkt in der Zukunft zu planen, muss hier ein AlarmManager implementiert werden, der einen Broadcast an einen entsprechenden Receiver sendet. Erst dieser erstellt die Notification und lässt sie sofort auf dem Display des Benutzers erscheinen. Kapitel 5.2.2.1 beschreibt die Implementierung in iOS und Kapitel 5.2.2.2 die Implementierung unter Android.

5.2.2.1. Local Notifications unter iOS

iOS bietet Methoden, mit denen sich eine Local Notification erstellen und zu einem bestimmten Zeitpunkt planen lässt. Listing 5.13 zeigt die Implementierung in der Track Your Tinnitus App. Generell müssen fünf Schritte ausgeführt werden, um eine Notification zu planen [App13g]:

1. Ein *UILocalNotification* Objekt alloziieren und initialisieren (Zeile 3).
2. Das Datum und die Zeit der Auslieferung der Notification setzten (*fireDate*, Zeile 5). Wenn die Zeitzone (*timeZone*, Zeile 6) gesetzt wird, passt das iOS System automatisch die Auslieferungszeit an, wenn ein Benutzer die Zeitzone wechselt. Optional kann noch ein Zeitintervall (*repeatInterval*) definiert werden, wenn es sich um eine regelmäßige Notification handelt.
3. Konfigurieren des Erscheinungsbildes der Notification.
   - Die Eigenschaft *alertBody* beschreibt dabei den Text einer Notification, der dem Benutzer angezeigt wird (Zeile 8) und *alertAction* den Titel des Buttons, um direkt auf die Notification zu reagieren (Zeile 9). Für beide Eigenschaften werden mit *NSLocalizedString* die übersetzten Texte gesetzt.
   - Mit der Eigenschaft *soundName* kann ein eigener Klingelton für die Notification gesetzt werden. In der Track Your Tinnitus App kann dieser Klingelton in den Einstellungen verändert werden. Daher wird der Name der Sounddatei aus den *NSUserDefaults* geladen und gesetzt (Zeilen 10 - 22).
   - Die Klasse *UILocalNotification* bietet noch zwei weitere Eigenschaften: Mit der Eigenschaft *applicationIconBadgeNumber* kann auf dem Icon der App eine Nummer angezeigt werden. Mit der Eigenschaft *alertLaunchImage* kann ein spezieller Splash-Screen angezeigt werden, wenn der Benutzer direkt mit der Notification interagiert [App10]. Beide Eigenschaften finden in Track Your Tinnitus keine Verwendung.
4. Optional können eigene Daten übergeben werden. Dafür wird mit der Eigenschaft *userInfo* ein *NSDictionary*, das Datum und die Uhrzeit der Notification übergeben (Zeile 24, Zeile 25).
5. Übergeben der Notification an das System zur Auslieferung. Zeile 28 zeigt, wie eine *Notification* direkt an das System übergeben werden kann. Track Your Tinnitus fügt alle *UILocalNotification* Objekte einem Array hinzu (Zeile 29) und übergibt dann das Array an das System.

Die Implementierung in der Track Your Tinnitus App bietet noch zwei Besonderheiten: Da die Benachrichtigungen bei der Benachrichtigungsart "Standard" mit einem Algorithmus zufällig verteilt werden, wird in Zeile 26 überprüft, ob der Benachrichtigungszeitpunkt mindestens 15min in der Zukunft liegt. Nur wenn dies der Fall ist, wird die Benachrichtigung an das System übergeben. Zusätzlich wird der Benachrichtigungszeitpunkt und die Benachrichtigungsart in der internen Datenbank gespeichert. Dazu wird in Zeile 31 ein neues Notification Objekt im *Context* erstellt. Dieses Objekt bekommt den Benachrichtigungszeitpunkt und die Benachrichtigungsart übergeben (Zeile 32 und 33) und wird in der Datenbank gespeichert (Zeile 34).
Bei der Benachrichtigungsart "Benutzerdefiniert" kann ein Benutzer die Zeitpunkte selbst festlegen. Diese Benachrichtigungen werden nur ein Mal an das System übergeben. Zusätzlich wird das *repeatInterval* der Notification gesetzt. Für das Zeitintervall einer wöchentlichen Wieder-

holung steht unter iOS die Konstante *NSWeekCalendarUnit* zur Verfügung. Eine Überprüfung, ob der Benachrichtigungszeitpunkt einen Abstand von mindestens 15min zum aktuellen Zeitpunkt hat, ist nicht nötig, da die Berechnung der Zeitpunkte dies bereits berücksichtigt. Beim Speichern dieser Benachrichtigungszeit in der Datenbank wird das Flag für *isFixedNotification* auf *TRUE* gesetzt.

Listing 5.13: Local Notifications

```
1  - (BOOL)scheduleLocalNotification:(NSDate *)fireDate {
2      // 1
3      UILocalNotification *localNotif = [[UILocalNotification alloc] init];
4      // 2
5      localNotif.fireDate = fireDate;
6      localNotif.timeZone = [NSTimeZone defaultTimeZone];
7      // 3
8      localNotif.alertBody = [NSString stringWithFormat:NSLocalizedString(@"Please fill
       in a new Questionnaire.", nil)];
9      localNotif.alertAction = NSLocalizedString(@"Open", nil);
10     NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
11     NSString *soundName = [defaults stringForKey:@"NotificationSoundName"];
12     if (soundName) {
13         if ([soundName isEqualToString:@"default"]) {
14             localNotif.soundName = UILocalNotificationDefaultSoundName;
15         }
16         else {
17             localNotif.soundName = [NSString stringWithFormat:@"%@.wav",
       soundName];
18         }
19     }
20     else {
21         localNotif.soundName = UILocalNotificationDefaultSoundName;
22     }
23     // 4
24     NSDictionary *infoDict = [NSDictionary dictionaryWithObject:fireDate forKey:@"
       FireDate"];
25     localNotif.userInfo = infoDict;
26     if ([fireDate timeIntervalSinceNow] > 900) {
27         // 5
28         //[[UIApplication sharedApplication] scheduleLocalNotification:localNotif];
29         [_localNotifications addObject:localNotif];
30
31         Notification *newFixedNotification = [NSEntityDescription
       insertNewObjectForEntityForName:@"Notification" inManagedObjectContext:
       _managedObjectContext];
32         newFixedNotification.date = fireDate;
33         newFixedNotification.isFixedTime = [NSNumber numberWithBool:FALSE];
34         [self saveContext];
35         return YES;
```

```
36      }
37      return NO;
38  }
```

5.2.2.2. Local Notifications unter Android

Das Android Betriebssystem bietet, im Gegensatz zu iOS, keine Möglichkeit Local Notifications direkt an das System zur Auslieferung in der Zukunft zu geben. Abbildung 5.1 zeigt die Schritte, die implementiert werden, um einem Benutzer eine Notification anzuzeigen. Android bietet den *AlarmManager*, um eine App zu einem bestimmten Zeitpunkt auszuführen. In der Track Your Tinnitus App werden diese Alarme vom *SuperNotificationManager* erstellt. Zusätzlich muss einem Alarm ein entsprechendes *Intent* übergeben werden, welches gesendet wird, wenn der geplante Zeitpunkt eintritt. Geplante Alarme bleiben erhalten, solange das Gerät inaktiv ist, werden aber gelöscht, wenn das Gerät neu gestartet wird [Goob]. Das *Intent*, welches beim Erstellen eines Alarms übergeben wird, ist bei Track Your Tinnitus ein *Broadcast*. Der *AlarmManager* sendet zum eingestellten Zeitpunkt diesen *Broadcast* an das Betriebssystem, welches die App startet und die Methode *onReceive* (siehe Listing 5.14) der Klasse *NotificationReceiver* aufruft. *NotificationReceiver* ist daher eine direkte Sub-Klasse von *BroadcastReceiver* [Gooc]. Erst diese erstellt und konfiguriert die Notification, die dann direkt mithilfe des *NotificationManager* einem Benutzer angezeigt wird.

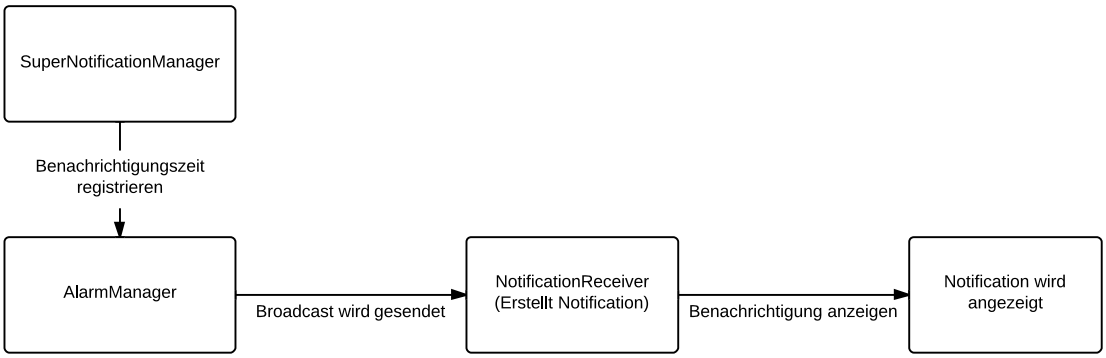

Abbildung 5.1.: Senden einer Benachrichtigung in Android

Listing 5.14 zeigt den Code, der *onReceive*-Methode, welche aufgerufen wird, wenn ein Broadcast vom System gesendet wurde. Generell müssen folgende Schritte implementiert werden, um eine Notification zu erstellen und anzuzeigen:

1. Für das Erstellen eins Objekts der Klasse *Notification* bietet Android die Builder-Klasse *Notification.Builder*. Da die Track Your Tinnitus App Android API Version ab Version 4 unterstützen soll, wird die Klasse *NotificationCompat.Builder* [Gooh] aus der *Android Support* Library verwendet [Goop]. Nach der Initialisierung eines Objekts dieser Klasse wird der Inhalt der Notification gesetzt, welcher im Folgenden näher beschrieben ist:
   - Icon, welches neben dem Text der Notification erscheinen soll (Zeile 8).
   - Titel der Notification (Zeile 9).

- Text einer Notification (Zeile 10).
- Text, des Tickers, der in der Statusleiste eines Gerätes erscheint, wenn die Notification eintrifft und das Notification Center nicht geöffnet ist (Zeile 11).
- Klingelton (Zeile 12).
- Farbe und Blinkintervall der LED am Gerät, falls eine solche vorhanden ist (Zeile 13).
- Interval des Vibrationsalarms (Zeile 14).

2. Erstellen eines *Intents* für eine *Activity*, welches gesendet wird, wenn ein Benutzer mit der Notification interagiert (Zeile 16). Zusätzlich wird noch ein *TaskStackBuilder* verwendet, damit das Drücken des "Zurück"-Buttons am Gerät die App wieder beendet [Gooj]. Dieses Intent wird dem *NotificationCompat.Builder* übergeben (Zeile 18).
3. Eine Instanz des *NotificationManager* vom System holen (Zeile 28) [Gooi].
4. Mit Hilfe des *NotificationManager* die Notification ausliefern (Zeile 32).

Das Speichern eines Benachrichtigungszeitpunktes in der Datenbank, sowie das Planen einer wiederkehrenden Benachrichtigung sind in der Klasse *SuperNotificationManager* implementiert.

Listing 5.14: Notification Receiver

```
1  @Override
2  public void onReceive(Context context, Intent intent) {
3      Uri alarmSound = RingtoneManager.getDefaultUri(RingtoneManager.
       TYPE_NOTIFICATION);
4      long[] pattern = {500,500,500,500};
5      // 1
6      NotificationCompat.Builder mBuilder =
7              new NotificationCompat.Builder(context)
8              .setSmallIcon(R.drawable.ic_launcher)
9              .setContentTitle(context.getResources().getString(R.string.
       notification_track_now))
10             .setContentText(context.getResources().getString(R.string.notification_text))
11             .setTicker(context.getResources().getString(R.string.notification_ticker_text
       ))
12             .setSound(alarmSound)
13             .setLights(Color.BLUE, 500, 500);
14     mBuilder.setVibrate(pattern);
15     // 2
16     Intent resultIntent = new Intent(context, MainActivity.class);
17
18     TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);
19     stackBuilder.addParentStack(MainActivity.class);
20
21     stackBuilder.addNextIntent(resultIntent);
22     PendingIntent resultPendingIntent =
23             stackBuilder.getPendingIntent(
```

```
24                  0,
25                  PendingIntent.FLAG_UPDATE_CURRENT
26              );
27      mBuilder.setContentIntent(resultPendingIntent);
28      NotificationManager mNotificationManager =
29          (NotificationManager) context.getSystemService(Context.
        NOTIFICATION_SERVICE);
30
31      Calendar cal = Calendar.getInstance();
32      mNotificationManager.notify((int)cal.getTimeInMillis(), mBuilder.build());
33  }
```

5.2.3. Algorithmus für die Benachrichtigungsverteilung

Die Track Your Tinnitus Apps unterstützen zwei verschiedene Benachrichtigungsarten: "Standard" und "Benutzerdefiniert". Bei der Benachrichtigungsart "Standard" kann ein Benutzer für jeden Tag einer Woche einen Start- und Endzeitpunkt bestimmten. Die genauen Benachrichtigungszeiten werden dann mit einem Algorithmus bestimmt. Den generellen Ablauf zu Bestimmung der Benachrichtigungszeiten zeigt Abbildung 5.2 und lässt sich in drei Schritte unterteilen:

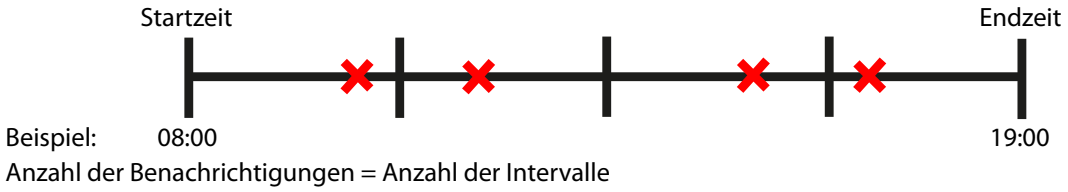

Abbildung 5.2.: Algorithmus für die Benachrichtigungsverteilung

1. Der Zeitraum zwischen Start- und Endzeit wird in so viele Intervalle unterteilt, wie ein Benutzer Benachrichtigungen erhalten will.
2. Innerhalb dieser Intervalle werden die Benachrichtigungszeitpunkte zufällig verteilt
3. Eine Überprüfung stellt sicher, dass zwei Benachrichtigungen jeweils mindestens 15 Minuten voneinander entfernt sind.

Sowie unter iOS, als auch unter Android werden Benachrichtigungen sofort ausgeliefert, wenn der Benachrichtigungszeitpunkt in der Vergangenheit liegt. Dies wird verhindert, indem der Algorithmus immer vom aktuellen Zeitpunkt in die Zukunft rechnet.

5.2.3.1. Implementierung unter iOS

Listing 5.15 zeigt die Implementierung des Algorithmus unter iOS. iOS bietet keine Möglichkeit, bereits geplante Benachrichtigungen zu löschen. Daher werden bei jedem Aufruf alle bereits

geplanten Benachrichtigungen gelöscht (Zeile 3). Die Methode *deleteAllNotificationsFromQueue* löscht dabei auch alle Einträge aus der internen Datenbank. Um einzelne Komponenten eines Datums zu verändern, bietet iOS die Klasse *NSDateComponents*. Diese wird in den Zeilen 5 und 6 mit dem aktuellen Datum initialisiert. Außerdem wird noch ein *NSDateFormatter* Objekt benötig, welches als Format den Index des Tages in der Woche angibt (Zeile 8 und 9). Die Anzahl der Benachrichtigungen kann vom Benutzer in den Einstellungen der App verändert werden und wird in Zeile 11 aus den *NSUserDefaults* geladen. Das iOS System behält immer 64 geplante Benachrichtigungen, die als nächstes ausgeliefert werden sollen [App10]. Daher wird das Planen der Benachrichtigungen in einer Schleife ausgeführt, die überprüft, ob diese Anzahl bereits erreicht ist (Zeile 14). In dieser Schleife werden zuerst die Variablen berechnet, die zum Verteilen der Benachrichtigungen benötigt werden:

- Name des Tages in einer Woche, an dem die Benachrichtigungen verteilt werden sollen (Zeilen 14 bis 17).
- Datum des Startzeitpunktes des Benachrichtigungsintervals an diesem Tag. Hierfür wird zuerst die Uhrzeit als *NSDate*-Objekt aus den Einstellungen geladen (Zeile 20) und das Jahr, der Monat und der Tag dieses Datums auf den aktuellen Tag gesetzt (Zeilen 21 bis 24). Schließlich werden die Komponenten wieder zu einem *NSDate*-Objekt zusammengefügt (Zeile 25).
- Datum des Endzeitunktes des Benachrichtigungsintervals. Wird analog zum Startzeitpunkt berechnet (Zeilen 26 bis 31).
- Für die Verteilung der Benachrichtigungen wird das Interval zwischen Start- und Endzeitpunkt in Sekunden berechnet (Zeile 33), sowie die Länge eines Unterintervals in dem eine Benachrichtigung geplant werden muss (Zeile 34).

Nachdem diese Variablen für den aktuellen Tag berechnet wurden, werden die Benachrichtigungszeitpunkte berechnet und die Benachrichtigungen im System geplant. Dazu wird zuerst eine Zufallszahl, zwischen 0 und der Länge eines Unterintervals, bestimmt (Zeile 39). Nachdem die Anzahl der Sekunden seit dem Startzeitpunkt berechnet wurde (Zeile 41), wird überprüft, ob dieser Zeitpunkt mindestens 15 Minuten vom letzten Benachrichtigungszeitpunkt entfernt ist (Zeile 43 und 44). Ist dies nicht der Fall, wird dieser Abstand auf 15 Minuten angepasst (Zeile 46). Durch die Verschiebung des Benachrichtigungszeitpunktes kann es vorkommen, dass der Benachrichtigungszeitpunkt nach dem Endzeitpunkt der Benutzereinstellung liegt. Daher wird dies überprüft (Zeile 50) bevor die Benachrichtigung geplant wird (Zeile 51). Diese Methode ist in Kapitel 5.2.2.1 beschrieben.

Listing 5.15: Algorithmus zur Benachrichtigungsverteilung unter iOS

```
1 - (void)setUpRandomNotifications {
2     NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];
3     [self deleteAllNotificationsFromQueue];
4
5     NSCalendar *cal = [[NSCalendar alloc] initWithCalendarIdentifier:
      NSGregorianCalendar];
6     NSDateComponents *mincomp = [cal components:NSYearCalendarUnit|
      NSMonthCalendarUnit|NSDayCalendarUnit|NSWeekCalendarUnit|
      NSWeekdayCalendarUnit|NSHourCalendarUnit|NSMinuteCalendarUnit fromDate:[
      NSDate date]];
```

```
7
8       NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
9       [formatter setDateFormat:@"e"];
10
11      int numberOfNotificationsPerDay = [userDefaults integerForKey:@"
        RandomNumberOfNotifications"];
12
13      int numberOfSchedueledNotification = 0;
14      while (numberOfSchedueledNotification < 65) {
15          int dayInWeekNumber = [[formatter stringFromDate:[cal dateFromComponents:
        mincomp]] intValue];
16          NSArray *daysOfWeek = [NSArray arrayWithObjects:@"", @"Sunday", @"
        Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday", @"Saturday", nil];
17          NSString *day = [daysOfWeek objectAtIndex:dayInWeekNumber];
18
19          NSCalendar *calendar = [[NSCalendar alloc] initWithCalendarIdentifier:
        NSGregorianCalendar];
20          NSDate *startDate = [userDefaults objectForKey:[NSString stringWithFormat:@"
        RandomNotificationTimesStart%@", day]];
21          NSDateComponents *startComponents = [calendar components:
        NSYearCalendarUnit|NSMonthCalendarUnit|NSDayCalendarUnit|
        NSHourCalendarUnit|NSMinuteCalendarUnit fromDate:startDate];
22          [startComponents setYear:mincomp.year];
23          [startComponents setMonth:mincomp.month];
24          [startComponents setDay:mincomp.day];
25          startDate = [calendar dateFromComponents:startComponents];
26          NSDate *endDate = [userDefaults objectForKey:[NSString stringWithFormat:@"
        RandomNotificationTimesEnd%@", day]];
27          NSDateComponents *endComponents = [calendar components:
        NSYearCalendarUnit|NSMonthCalendarUnit|NSDayCalendarUnit|
        NSHourCalendarUnit|NSMinuteCalendarUnit fromDate:endDate];
28          [endComponents setYear:mincomp.year];
29          [endComponents setMonth:mincomp.month];
30          [endComponents setDay:mincomp.day];
31          endDate = [calendar dateFromComponents:endComponents];
32
33          NSTimeInterval difference = [endDate timeIntervalSinceDate:startDate];
34          float lengthOfIntervall = difference/numberOfNotificationsPerDay;
35
36          int lastNotification = -900;
37          for (int i = 0; i < numberOfNotificationsPerDay; i++) {
38              // Random time in part-intervall
39              int secondsSinceStartOfInterval = arc4random_uniform(lengthOfIntervall);
40              // Timestamp in absolute intervall
41              int absoluteInterval = secondsSinceStartOfInterval+(i*lengthOfIntervall);
42              // Check for 15min difference
```

```
43              int differenceSinceLast = absoluteInterval - lastNotification;
44              if (differenceSinceLast < 900) {
45                  // difference between notification < 15min, push next to a difference of
        15mins.
46                  absoluteInterval = absoluteInterval + differenceSinceLast;
47              }
48              lastNotification = absoluteInterval;
49              // Check if notification is in absolut intervalll
50              if (absoluteInterval < difference) {
51                  BOOL success = [self scheduleLocalNotification:[NSDate
        dateWithTimeInterval:absoluteInterval sinceDate:startDate]];
52                  if (success)
53                      numberOfSchedueledNotification++;
54              }
55          }
56          mincomp.day++;
57      }
58  }
```

5.2.3.2. Implementierung unter Android

Der Algorithmus unter Android ist nach dem gleichen Prinzip wie unter iOS implementiert. Zuerst werden alle bisher geplanten Benachrichtigungen mit der Methode *removeAllNotifications()* aus dem *AlarmManager* und der Datenbank gelöscht (Zeile 2). Für die Berechnung einer Zufallszahl wird ein Objekt der Klasse *Random* benötigt (Zeile 3). Außerdem werden noch 2 Hilfsvariablen definiert (Zeile 4 und 5) und die Anzahl der Benachrichtigungen aus den *SharedPreferences* geladen (Zeile 6). Da der *AlarmManager* keine Beschränkung der Anzahl der geplanten Zeitpunkte hat, werden 100 Benachrichtigungszeitpunkte in einer Schleife geplant (Zeile 7). Für die weitere Berechnung wird zuerst der Index des aktuellen Tages (Zeile 8) und der Name dieses Tages mit der Methode *getDayNameFromDayOfWeek()* bestimmt (Zeile 9). Im Gegensatz zur Implementierung unter iOS ist es in Android nicht möglich, *Date*-Objekte in den *SharedPreferences* zu speichern. Daher wird der Startzeitpunkt in Millisekunden aus den *SharedPreferences* geladen (Zeile 11). Dieser Zeitpunkt wird mithilfe der Klasse *GregorianCalendar* auf den aktuellen Tag angepasst (Zeilen 12 bis 17) und wiederum in Millisekunden in der Variablen *startTime* erfasst (Zeile 18). Derselbe Ablauf wird bei der Bestimmung des Endzeitpunktes verwendet (Zeilen 20 bis 27). In den Zeilen 29 und 30 wird die Anzahl der Millisekunden des Intervalls von Start- bis Endzeitpunkt berechnet, sowie das Unterintervall, in dem eine Benachrichtigung geplant werden soll. Nun können die Benachrichtigungen in einer Schleife verteilt werden (Zeile 32). Hierzu wird zuerst eine zufällige Anzahl von Millisekunden berechnet und zum Startzeitpunkt addiert (Zeile 33). Dies stellt den Benachrichtigungszeitpunkt dar. Dieser wird in den Zeilen 35 bis 38 zuerst darauf überprüft, ob er in der Vergangenheit liegt, oder der Benachrichtigungszeitpunkt einen geringeren Abstand als 15 Minuten zur aktuellen Uhrzeit hat. Danach folgt die Überprüfung, ob der Abstand zum vorher berechneten Benachrichtigungszeitpunkt kleiner ist als 15 Minuten und wird in diesem Fall entsprechend angepasst (Zeilen 40 bis 42). Zuletzt wird noch überprüft, ob der Benachrichtigungszeitpunkt nach dem Endzeitpunkt liegt (Zeilen 44 bis 47). Sollten diese Überprüfungen erfolgreich verlaufen, wird die Benachrich-

tigung mit der Methode *scheduleNotification()* (siehe Kapitel 5.2.2.2) geplant (Zeile 49). Bevor die nächste Benachrichtigung berechnet wird, werden noch die Hilfsvariablen entsprechend angepasst (Zeilen 51 bis 53). Wenn die Berechnung der Benachrichtigungszeitpunkte eines Tages abgeschlossen ist, wird der aktuelle Tag um eins erhöht (Zeile 55).

Listing 5.16: Algorithmus zur Benachrichtigungsverteilung unter Android

```
1  private void setUpRandomNotifications() {
2          removeAllNotifications();
3          Random r = new Random();
4          int numberOfNotificationsSchedueld = 0;
5          long lastNotificationTime = 0L;
6          int numberOfNotifications = sharedPreferences.getInt("
       numberOfRandomNotifications", 10);
7          while(numberOfNotificationsSchedueld < 100) {
8              int today = calendar.get(GregorianCalendar.DAY_OF_WEEK);
9              String day = getDayNameFromDayOfWeek(today);
10
11             long startTime = sharedPreferences.getLong("randomNotificationTimeStart"
       +day, 0);
12             GregorianCalendar startCalendar = new GregorianCalendar();
13             startCalendar.setTimeInMillis(startTime);
14             startCalendar.set(GregorianCalendar.SECOND, 0);
15             startCalendar.set(GregorianCalendar.DAY_OF_MONTH, calendar.get(
       GregorianCalendar.DAY_OF_MONTH));
16             startCalendar.set(GregorianCalendar.MONTH, calendar.get(
       GregorianCalendar.MONTH));
17             startCalendar.set(GregorianCalendar.YEAR, calendar.get(GregorianCalendar.
       YEAR));
18             startTime = startCalendar.getTimeInMillis();
19
20             long endTime = sharedPreferences.getLong("randomNotificationTimeEnd"+
       day, 0);
21             GregorianCalendar endCalendar = new GregorianCalendar();
22             endCalendar.setTimeInMillis(endTime);
23             endCalendar.set(GregorianCalendar.SECOND, 0);
24             endCalendar.set(GregorianCalendar.DAY_OF_MONTH, calendar.get(
       GregorianCalendar.DAY_OF_MONTH));
25             endCalendar.set(GregorianCalendar.MONTH, calendar.get(GregorianCalendar
       .MONTH));
26             endCalendar.set(GregorianCalendar.YEAR, calendar.get(GregorianCalendar.
       YEAR));
27             endTime = endCalendar.getTimeInMillis();
28
29             long timeSpan = endTime-startTime;
30             double timeInterval = (double)timeSpan/(double)numberOfNotifications;
31
32             for(int i = 0; i < numberOfNotifications; i++) {
```

```
33                  long randomMillis = startTime + (long)(r.nextDouble()*timeInterval);
34                  // Check if calculated time is before (now + 15min)
35                  if (randomMillis <= Calendar.getInstance().getTimeInMillis() + 900000L)
    {
36                      startTime = startTime + (long)timeInterval;
37                      continue;
38                  }
39                  // Check if calculated time is in a 15min window to the last scheduled
    notification
40                  if ((randomMillis - lastNotificationTime) < 900000L) {
41                      randomMillis = randomMillis + (randomMillis - lastNotificationTime
    );
42                  }
43                  // if calculated time is after endTime continue;
44                  if (randomMillis > endTime) {
45                      startTime = startTime + (long)timeInterval;
46                      continue;
47                  }
48
49                  scheduleNotification(randomMillis, notificationsDatabase, false);
50
51                  lastNotificationTime = randomMillis;
52                  numberOfNotificationsScheduld++;
53                  startTime = startTime + (long)timeInterval;
54              }
55              calendar.setTimeInMillis(calendar.getTimeInMillis() + 86400000L);
56          }
57      }
```

5.3. Custom Views

Die Diagramme zur Visualisierung der Ergebnisse aus dem Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung in den Apps wurden als eigene *UIView* bzw. *View* Klassen implementiert. Gegenüber einer vorgefertigten Third-Party-Library hat dies mehrere Vorteile:

- Mehr Flexibilität in der Gestaltung der Diagramme
- Bessere Optimierung der Performance
- Die Dateigröße der App wird nicht durch eine externe Library vergrößert, obwohl aus dieser Library nicht alle Funktionen benötigt werden.

Dieser Abschnitt beschreibt die Erstellung eigener *View*-Subklassen am Beispiel der Diagramme. Außerdem wird gezeigt, wie eigene Darstellungen gezeichnet werden.

5.3.1. Custom Views in iOS

Wie in Kapitel 5.4.1.1 beschrieben, lässt sich ein View unter iOS mit verschiedenen Layer-Klassen erstellen. Bei den Diagrammen wurde allerdings ein anderer Ansatz gewählt. Der Code um das *UIView* zu zeichnen, wurde direkt in der *UIView*-Subklasse implementiert. Die *UIView* Klasse definiert einen rechteckigen Bereich auf dem Display und das Erscheinungsbild. Während der Ausführung der App rendert ein *UIView*-Objekt den Inhalt in diesem Bereich. Das Zeichnen des Inhalts erfolgt je nach Bedarf. Wenn das *UIView*-Objekt das erste Mal auf dem Bildschirm angezeigt wird, oder wenn alles oder Teile erscheinen, ruft das System die entsprechende Methode des *UIView*-Objekts auf. Um eigenen Inhalt zu Zeichnen, wird dazu die Methode *drawRect:* aufgerufen [App13k]. Die Implementierung dieser Methode muss in den aktuellen grafischen Context zeichnen, der vom System vor dem Methodenaufruf erstellt wird. Die Texte an den Achsen werden zunächst als *UILabel* Objekte erstellt und in der *init* Methode dem *UIView* hinzugefügt. So müssen diese nicht selbst gezeichnet werden und können auch verändert werden, ohne das gesamte *UIView* neu zu laden. Das Diagramm wird in iOS mit Hilfe von *Quartz 2D* gezeichnet. *Quartz 2D* ist eine zweidimensionale Drawing Engine für das Zeichnen in iOS Apps und Max OS X Apps. *Quartz 2D* ist auflösungs- und geräteunabhängig. So lässt sich mit *Quartz 2D* nicht nur in *UIViews*, sondern auch in PDFs oder auf Seiten, die an einen Drucker gesendet werden, zeichnen [App13f]. Das Zeichnen mit *Quartz 2D* ist generell in zwei Schritte unterteilt: Zeichenpfad erstellen und Zeichenpfad im *Context* zeichnen. Ein Pfad besteht aus einer oder mehreren Formen oder Unterpfaden. Ein Unterpfad kann dabei aus Linien und Kurven bestehen. *Quartz 2D* bietet darüber hinaus noch Methoden, um Rechtecke oder Kreise direkt mit einem Methodenaufruf zu zeichnen [App13f].
Beim Erstellen eines Pfades müssen folgende Punkte beachtet werden [App13f]:

- Linien, Kreise oder Kreisbögen werden immer an den aktuellen Punkt gezeichnet. Ein neuer Pfad hat keinen aktuellen Punkt. Dieser muss zuerst mit *CGContextMoveToPoint*, oder einem Methodenaufruf, der dies implizit macht, gesetzt werden.
- Ein Pfad kann durch *CGContextClosePath* geschlossen werden. Dabei wird allerdings eine Linie zwischen dem aktuellen Punkt und dem Startpunkt des Pfades gezeichnet.
- Wenn ein Kreisbogen gezeichnet wird, fügt *Quartz 2D* automatisch eine Linie zwischen dem aktuellen Punkt und dem Startpunkt des Kreisbogens hinzu.
- Methoden, die ein Rechteck oder einen Kreis direkt zeichnen, fügen dem Pfad einen geschlossenen Unterpfad hinzu.
- Ein Pfad wird erst im *Context* gezeichnet, wenn die Methoden zum Zeichnen des Rahmens des Pfades, bzw. zum Füllen des Pfades aufgerufen werden.
- Nach dem Zeichnen des Pfades wird dieser aus dem aktuellen *Context* entfernt.

Listing 5.17 zeigt die *drawRect:* Methode, die das Diagramm in der Ergebnisdarstellung zeichnet. Zuerst wird in Zeile 3 eine Referenz auf den aktuellen *Context* einer Variablen zugewiesen. Jede Methode, die einen Pfad erstellt, oder diesen zeichnet, benötigt diese Referenz. In den Zeilen 4 bis 6 werden die drei benötigten Farben (schwarz, hellgrau und blau) ebenfalls einer Variablen zugewiesen. Vor dem Zeichnen müssen diese mit *CGContextSetStrokeColorWithColor()* als Linienfarbe (Zeile 8) und mit *CGContextSetFillColorWithColor(()* als Füllfarbe (Zeile 7) gesetzt werden. Um eine einfache Linie zu zeichnen, muss der aktuelle Punkt zuerst mit *CGContextMoveToPoint()* an den Startpunkt verschoben werden, um mit *CGContextAddLine-*

ToPoint() eine Linie dem Pfad hinzuzufügen. In den Zeilen 9 -16 wird zunächst ein Pfad für die grauen Hilfslinien im Hintergrund des Diagramms erstellt und in Zeile 17 mit *CGContextStrokePath()* gezeichnet. Nachdem die Farben auf schwarz gesetzt wurden (Zeile 18 und Zeile 19) werden die X- und Y-Achse des Diagramms nach dem gleichen Prinzip gezeichnet (Zeile 20 - 24).
Das Zeichnen der Werte ist in sieben Schritte unterteilt:

1. Bestimmen des ersten und letzten Datensatzes der Antworten auf die Fragen des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung, die als Array an die Klasse übergeben wurde (Zeile 29 und Zeile 30).
2. Errechnen der Anzahl Punkte im *UIView*, die einer Sekunde im Zeitraum zwischen dem ersten und letzten Antwortdatensatz entsprechen (Zeile 31).
3. Wert der Antwort auf die Frage bestimmen. Der Klasse wird die Nummer der Frage als *_index* übergeben. Wenn zu diesem Antwortdatensatz eine Antwort auf diese Frage gegeben wird, wird dieser Wert der Variablen *answer_float* zugewiesen. Wenn keine Antwort auf diese Frage gegeben wird, beginne wieder bei 2 mit dem nächsten Datensatz (Zeilen 34 - 85).
4. Errechnen des Punktes auf der Zeichenfläche (Zeile 86).
5. Zeichnen eines Kreises auf diesem Punkt mit der Methode *CGContextFillEllipseInRect()* (Zeile 87)
6. Falls es einen Datensatz vor dem aktuellen gab, wird eine Linie zwischen diesen zwei Datensätzen gezeichnet (Zeilen 89 - 98).
7. Beginne wieder mit 2., bis keine Datensätze mehr vorhanden sind.

Listing 5.17: Drawing Methode der ResultsChartView Klasse

```
1 - (void)drawRect:(CGRect)rect
2 {
3     CGContextRef context = UIGraphicsGetCurrentContext();
4     UIColor *blackColor = [UIColor colorWithRed:0.0/255.0 green:0.0/255.0 blue:0.
      0/255.0 alpha:1.0];
5     UIColor *lightgreyColor = [UIColor colorWithRed:200.0/255.0 green:200/255.0 blue
      :200.0/255.0 alpha:1.0];
6     UIColor *blueColor = [UIColor colorWithRed:44.0/255 green:107.0/255 blue:177.0/255
       alpha:1.0];
7     CGContextSetFillColorWithColor(context, lightgreyColor.CGColor);
8     CGContextSetStrokeColorWithColor(context, lightgreyColor.CGColor);
9     CGContextMoveToPoint(context, 50, (rect.size.height-20)/4+10);
10    CGContextAddLineToPoint(context, rect.size.width, (rect.size.height-20)/4+10);
11    CGContextMoveToPoint(context, 50, (rect.size.height-20)/4*2+10);
12    CGContextAddLineToPoint(context, rect.size.width, (rect.size.height-20)/4*2+10);
13    CGContextMoveToPoint(context, 50, (rect.size.height-20)/4*3+10);
14    CGContextAddLineToPoint(context, rect.size.width, (rect.size.height-20)/4*3+10);
15    CGContextMoveToPoint(context, 50, 10);
16    CGContextAddLineToPoint(context, rect.size.width, 10);
17    CGContextStrokePath(context);
```

```
18      CGContextSetFillColorWithColor(context, blackColor.CGColor);
19      CGContextSetStrokeColorWithColor(context, blackColor.CGColor);
20      CGContextMoveToPoint(context, 55, 5);
21      CGContextAddLineToPoint(context, 55, rect.size.height-10);
22      CGContextMoveToPoint(context, 50, rect.size.height-10);
23      CGContextAddLineToPoint(context, rect.size.width, rect.size.height-10);
24      CGContextStrokePath(context);
25      CGContextSetFillColorWithColor(context, blueColor.CGColor);
26      CGContextSetStrokeColorWithColor(context, blueColor.CGColor);
27      CGPoint lastPoint = CGPointZero;
28      CGContextSetLineWidth(context, 2.0f);
29      StandardAnswer *firstAnswer = [_resultsArray objectAtIndex:0];
30      StandardAnswer *lastAnswer = [_resultsArray objectAtIndex:[_resultsArray count
        ]-1];
31      float stepValue = (rect.size.width-70)/[lastAnswer.date_save timeIntervalSinceDate
        :firstAnswer.date_save];
32      CGContextSetLineWidth(context, 1.0f);
33      for (int i = 0; i < [_resultsArray count]; i++) {
34          StandardAnswer *answer = [_resultsArray objectAtIndex:i];
35          float answer_float;
36          switch (_index) {
37              case 2:
38                  if (!answer.question2) {
39                      continue;
40                  }
41                  answer_float = [answer.question2 floatValue];
42                  break;
43              case 3:
44                  if (!answer.question3) {
45                      continue;
46                  }
47                  answer_float = [answer.question3 floatValue];
48                  break;
49              case 4:
50                  if (!answer.question4) {
51                      continue;
52                  }
53                  answer_float = [answer.question4 floatValue];
54                  break;
55              case 5:
56                  if (!answer.question5) {
57                      continue;
58                  }
59                  answer_float = [answer.question5 floatValue];
60                  break;
61              case 6:
```

```
62                  if (!answer.question6) {
63                      continue;
64                  }
65                  answer_float = [answer.question6 floatValue];
66                  break;
67              case 7:
68                  if (!answer.question7) {
69                      continue;
70                  }
71                  answer_float = [answer.question7 floatValue];
72                  break;
73              case 8:
74                  if (!answer.question8) {
75                      continue;
76                  }
77                  answer_float = [answer.question8 floatValue];
78                  break;
79              default:
80                  if (!answer.question1) {
81                      continue;
82                  }
83                  answer_float = [answer.question1 floatValue];
84                  break;
85          }
86          CGPoint point = CGPointMake(60+[answer.date_save timeIntervalSinceDate:
        firstAnswer.date_save]*stepValue, (rect.size.height-(rect.size.height-20)*answer_float
        )-10);
87          CGContextFillEllipseInRect(context, CGRectMake(point.x-3, point.y-3, 6, 6));
88          //CGContextAddEllipseInRect(context, CGRectMake(point.x-2, point.y-2, 4, 4)
        );
89          if (lastPoint.x == 0 && lastPoint.y == 0) {
90              lastPoint = point;
91          }
92          else {
93              CGContextMoveToPoint(context, lastPoint.x, lastPoint.y);
94              CGContextAddLineToPoint(context, point.x, point.y);
95              CGContextStrokePath(context);
96              lastPoint = point;
97          }
98      }
99  }
```

5.3.2. Custom Views in Android

Jedes User-Interface Element in Android ist eine Subklasse der Klasse *View*. Custom Views müssen daher eine Subklasse von *View* sein, oder von einer der vorhandenen Klassen, wie z.B.

Button, erben. Die Klasse, die ein Diagramm in der Ergebnisdarstellung der Track Your Tinnitus App darstellt, erbt direkt von *View*. Für das Zeichnen unter Android wird das *android.graphics*-Framework verwendet. Dieses unterteilt den Zeichenvorgang in zwei Bereiche:

- Was gezeichnet wird, wird von der Klasse *Canvas* übernommen.
- Wie dies gezeichnet wird, wird von der Klasse *Paint* übernommen.

Zum Beispiel bietet *Canvas* eine Methode, um eine Linie zu zeichnen, *Paint* bietet die Methoden um die Farbe dieser Linie zu definieren [Gooe]. Bevor etwas gezeichnet werden kann, muss mindestens ein *Paint*-Objekt erstellt werden. Listing 5.18 zeigt den Konstruktor der *JHGraphView* Klasse. In dieser Methode werden die vier benötigten *Paint*-Objekte erstellt:

1. *axisPaint*: Definiert den Stil der Zeichnung der Achsen des Diagramms (Zeile 22 bis 23).
2. *textPaint*: Definiert den Stil des Textes, der an den Achsen gezeichnet wird (Zeile 28 und 29).
3. *graphPaint:* Definiert den Stil der Linie zwischen den einzelnen Werten (Zeile 31 bis 35).
4. *circlePaint:* Definiert den Stil der Kreise, die um jeden Wert gezeichnet werden (Zeile 37 - 41).

Für jedes dieser *Paint*-Objekte wird das Anti Aliasing aktiviert (Zeile 22), die Linienbreite festgelegt (Zeile 23) und die Farbe der Zeichnung bestimmt (Zeile 24). Außerdem wird mit *setStyle()* der Stil der Linie festgelegt (Zeile 25). Mit dieser Methode kann eine Linie unter anderem als durchgehende oder gestrichelte Linie festgelegt werden. Mit der Methode *setStrokeJoin()* wird festgelegt, wie Linien und Formen miteinander verbunden werden [Gook].
In den Zeilen 43 bis 45 wird der Index der Frage aus dem Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung bestimmt. Dieser Index wird in der XML-Layout-Datei eingestellt und als *AttributeSet* an den Konstruktor übergeben. Danach wird noch die Höhe des Displays bestimmt (Zeile 47 bis 55). Diese wird benötigt, um die Größe des Diagramms in der *onMeasure()*-Methode zu bestimmen.

Listing 5.18: Konstruktor der JHGraphView Klasse

```
1  private static final float LEFT_SPACING = 60f;
2  private static final float BOTTOM_SPACING = 25f;
3  private static final float LINE_WIDTH = 3f;
4
5  private Paint axisPaint = new Paint();
6  private Paint textPaint = new Paint();
7  private Paint graphPaint = new Paint();
8  private Paint circlePaint  = new Paint();
9  private Path axisPath = new Path();
10 private Path circlePath = new Path();
11 private Path graphPath = new Path();
12 private DateFormat dateFormat = SimpleDateFormat.getDateInstance();
13
14 private int question = 1;
15 private StandardAnswer[] answers;
16 private int displayHeight;
17
```

```
18 @SuppressLint("NewApi")
19 @SuppressWarnings("deprecation")
20 public JHGraphView(Context context, AttributeSet attrs) {
21     super(context, attrs);
22     axisPaint.setAntiAlias(true);
23     axisPaint.setStrokeWidth(LINE_WIDTH);
24     axisPaint.setColor(Color.BLACK);
25     axisPaint.setStyle(Paint.Style.STROKE);
26     axisPaint.setStrokeJoin(Paint.Join.ROUND);
27
28     textPaint.setAntiAlias(true);
29     textPaint.setTextSize(23f);
30
31     graphPaint.setAntiAlias(true);
32     graphPaint.setStrokeWidth(LINE_WIDTH);
33     graphPaint.setColor(Color.BLUE);
34     graphPaint.setStyle(Paint.Style.STROKE);
35     graphPaint.setStrokeJoin(Paint.Join.ROUND);
36
37     circlePaint.setAntiAlias(true);
38     circlePaint.setStrokeWidth(LINE_WIDTH);
39     circlePaint.setColor(Color.BLUE);
40     circlePaint.setStyle(Paint.Style.FILL);
41     circlePaint.setStrokeJoin(Paint.Join.ROUND);
42
43     TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.Options);
44     question = a.getInt(R.styleable.Options_question, 1);
45     a.recycle();
46
47     WindowManager wm = (WindowManager) context.getSystemService(Context.
       WINDOW_SERVICE);
48     Display display = wm.getDefaultDisplay();
49     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2
       ) {
50         Point outSize = new Point();
51         display.getSize(outSize);
52         displayHeight = outSize.y;
53     } else {
54         displayHeight = display.getHeight();
55     }
56 }
```

Das Zeichnen des Diagramms ist in der Methode *onDraw()* implementiert, die in Listing 5.19 dargestellt ist. Wie beim Zeichnen in iOS müssen in Android mehrere Pfade erstellt werden, die dann vom übergebenen *Canvas* gezeichnet werden. Zuerst wird der Pfad für die Achsen des Diagramms erstellt (Zeile 3 bis 5). Dafür muss der Startpunkt festgelegt werden (Zeile 3). Von diesem aus wird zuerst die Y-Achse (Zeile 4) und dann die X-Achse (Zeile 5) dem

Pfad hinzugefügt. Die Methode *drawPath()* der *Canvas*-Klasse zeichnet einen Pfad (*axisPath*) mithilfe eines *Paint*-Objekts. Text kann direkt mit der Methode *drawText()* der Klasse *Canvas* gezeichnet werden. In den Zeilen 7 und 8 wird der Text am oberen bzw. unteren Ende der Y-Achse gezeichnet. Für den Text an der X-Achse werden zunächst die Zeitpunkte für den Anfang, die Mitte und das Ende bestimmt (Zeile 9 - 13). Außerdem wird die Anzahl der Punkte pro Sekunde in diesem Zeitraum bestimmt (Zeile 14). Mit diesen Werten können nur die Texte unter der X-Achse gezeichnet werden (Zeilen 16 - 18). Das Zeichnen der einzelnen Punkte der Datensätze ist in einer Schleife implementiert (Zeile 19). Falls ein Datensatz keine Antwort auf die Frage hat, wird kein Punkt gezeichnet (Zeilen 20 - 22). Falls ein Datensatz eine Antwort auf die Frage hat, wird der X- und Y-Wert im *Canvas* berechnet (Zeile 23 und 24). Zu diesem Punkt wird dann eine Linie dem Pfad *graphPath* hinzugefügt (Zeile 25) und ein Kreis, um diesen Punkt dem *circlePath* (Zeile 26). Nachdem alle Datensätze berücksichtig wurden, werden die Pfade für die Linie des Diagramms und die Kreise um die einzelnen Werte gezeichnet (Zeile 28 und 29).

Listing 5.19: Drawing Methode der JHGraphView Klasse

```
1  @Override
2  protected void onDraw (Canvas canvas) {
3      axisPath.moveTo(LEFT_SPACING, 5f);
4      axisPath.lineTo(LEFT_SPACING, canvas.getHeight()-5f-BOTTOM_SPACING);
5      axisPath.lineTo(canvas.getWidth(), canvas.getHeight()-5f-BOTTOM_SPACING);
6      canvas.drawPath(axisPath, axisPaint);
7      canvas.drawText(this.getAxisCaption(true), 0, 20, textPaint);
8      canvas.drawText(this.getAxisCaption(false), 0, canvas.getHeight()-
       BOTTOM_SPACING, textPaint);
9      if (this.answers != null) {
10         long firstDate = answers[0].save_date.getTime();
11         long lastDate = answers[answers.length-1].save_date.getTime();
12         long timeSpan = lastDate - firstDate;
13         long middleDate = lastDate - timeSpan / 2;
14         double pixelsPerSecond = ((double)canvas.getWidth()-LEFT_SPACING-5f)/((
       double)timeSpan);
15         graphPath.moveTo(LEFT_SPACING, 5f);
16         canvas.drawText(dateFormat.format(firstDate), LEFT_SPACING-5f, canvas.
       getHeight(), textPaint);
17         canvas.drawText(dateFormat.format(lastDate), canvas.getWidth()-
       LEFT_SPACING-60f, canvas.getHeight(), textPaint);
18         canvas.drawText(dateFormat.format(middleDate), canvas.getWidth()/2+
       LEFT_SPACING-90f, canvas.getHeight(), textPaint);
19         for (int i = 0; i < answers.length; i++) {
20             if(answers[i].getAnswer(question) == -1) {
21                 continue;
22             }
23             double xvalue = (answers[i].save_date.getTime()-firstDate)*pixelsPerSecond;
24             float yvalue = canvas.getHeight()-5f-BOTTOM_SPACING-((canvas.
       getHeight()-5f-BOTTOM_SPACING-10f)*answers[i].getAnswer(question));
25             graphPath.lineTo((float)xvalue + LEFT_SPACING, yvalue);
```

```
26                  circlePath.addCircle((float)xvalue + LEFT_SPACING, yvalue, 4f, Direction.
        CCW);
27              }
28              canvas.drawPath(graphPath, graphPaint);
29              canvas.drawPath(circlePath, circlePaint);
30          }
31      }
```

Die *View*-Klasse bietet noch eine weitere Methode, um die Größe des *View* auf dem Bildschirm zu berechnen. Da Android verschiedene Bildschirmauflösungen unterstützt, kann für das Berechnen der Größe die Methode *onMeasure()* überschrieben werden. Listing 5.20 zeigt die Implementierung dieser Methode in der *JHGraphView* Klasse. Das System übergibt zwei *View.MeasureSpec* Werte. Diese Werte beinhalten Informationen darüber, wie groß das Parent-View möchte, dass das aktuelle View gezeichnet wird. Außerdem enthalten diese Werte noch Informationen darüber, ob diese Größe festgelegt ist, oder nur einen Vorschlag darstellt.
Zuerst wird in den Zeilen 4 und 5 die bevorzugte Breite und Höhe festgelegt. In den Zeilen 7 bis 10 wird für die Breite und Höhe je der Wert bestimmt, den das System übergibt, sowie den *Mode* dieser Angabe. Dieser Mode kann drei verschiedene Werte annehmen [Goor]:

1. *MeasureSpec.EXACTLY*: Das *View* muss exakt die Breite und Höhe der übergebenen Werte annehmen.
2. *MeasureSpec.AT_MOST*: Das *View* darf höchsten die übergebenen Werte für die Breite und Höhe annehmen, darf allerdings auch kleiner sein.
3. *MeasureSpec.UNSPECIFIED*: Es gibt keine Vorgaben für die Breite und Höhe des *View*.

Je nachdem, welcher Wert für *Mode* übergeben wird, wird in den Zeilen 16 bis 25 die Breite und in den Zeilen 28 bis 38 die Höhe berechnet. Wird als *Mode EXACTLY* übergeben, wird die vorgeschlagene Breite (Zeile 16 und 18) bzw. Höhe (Zeile 28 und 30) übernommen. Wenn als *Mode AT_MOST* übergeben wird, wird das Minimum aus übergebener und gewünschter Breite bzw. Höhe berechnet (Zeilen 29 und 21 bzw. 31 und 33). Sollte als *Mode UNSPECIFIED* übergeben werden, wird die gewünschte Breite bzw. Höhe als berechnete übernommen (Zeile 22 und 24 bzw. 35 und 37). Die Methode *onMeasure()* hat keinen Rückgabewert. Die berechneten Werte werden mit der Methode *setMeasuredDimension()* (Zeile 41) an das System zurückgegebenen. Wird diese Methode nicht aufgerufen, wirft die *View*-Klasse eine Runtime Exception [Gooq].

Listing 5.20: *onMeasure* Methode der JHGraphView Klasse

```
1  @Override
2  protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
3
4      int desiredWidth = 100;
5      int desiredHeight = displayHeight/2;
6
7      int widthMode = MeasureSpec.getMode(widthMeasureSpec);
8      int widthSize = MeasureSpec.getSize(widthMeasureSpec);
9      int heightMode = MeasureSpec.getMode(heightMeasureSpec);
10     int heightSize = MeasureSpec.getSize(heightMeasureSpec);
```

```
11
12      int width;
13      int height;
14
15      //Measure Width
16      if (widthMode == MeasureSpec.EXACTLY) {
17          //Must be this size
18          width = widthSize;
19      } else if (widthMode == MeasureSpec.AT_MOST) {
20          //Can't be bigger than...
21          width = Math.min(desiredWidth, widthSize);
22      } else {
23          //Be whatever you want
24          width = desiredWidth;
25      }
26
27      //Measure Height
28      if (heightMode == MeasureSpec.EXACTLY) {
29          //Must be this size
30          height = heightSize;
31      } else if (heightMode == MeasureSpec.AT_MOST) {
32          //Can't be bigger than...
33          height = Math.min(desiredHeight, heightSize);
34          //height = heightSize/2;
35      } else {
36          //Be whatever you want
37          height = desiredHeight;
38      }
39
40      //MUST CALL THIS
41      setMeasuredDimension(width, height);
42  }
```

5.4. Custom Controls

Controls, die interaktiven User-Interface Elemente, sind ein zentraler Bestandteil jeder App. Sie bieten einem Benutzer eine grafische Möglichkeit, mit den Daten einer App zu interagieren. iOS und Android bieten einige vorgegebenen User-Interface Elemente, die für die meisten Funktionen ausreichen. Bei der Entwicklung von Track Your Tinnitus waren diese vorgegebenen Elemente allerdings nicht ausreichend. Beide Plattformen bieten unter anderem keine Möglichkeit den vorgegebenen Slider so einzustellen, dass initial kein Knopf angezeigt wird.

5.4.1. Slider ohne initialen Wert

Bei der Beantwortung von Fragebögen lässt sich ein Benutzer durch Vorinformationen (relevant oder irrelevant) beeinflussen. Diesen Effekt nennt man in der Psychologie "Ankereffekt". Angewandt auf die visuelle Analogskala bedeutet dies, dass die Voreinstellung der Skala die Entscheidung eines Benutzers in die Richtung der Voreinstellung beeinflusst. Die Voreinstellung bietet so eine "Anker" in dessen Nähe dann auch die Entscheidung des Benutzers landen wird [Kah11]. Daher haben wir uns bei Track Your Tinnitus dazu entschlossen, einen Slider ohne initialen Wert zu implementieren. Im Folgenden wird anhand des Sliders ohne initialen Wert beschrieben, wie eigene Controls unter iOS und Android erstellt werden können.

5.4.1.1. Slider ohne initialen Wert in iOS

In iOS muss jedes User-Interface Element eine Subklasse von UIView sein, damit es auf dem Display angezeigt werden kann. Abbildung 5.3 zeigt die direkten Subklassen von UIView [App13j]. Alle vorgegebenen User-Interface Elemente erben von *UIView*, wie zum Beispiel *UILabel* oder *UIImageView*. Ein Teil der Elemente, wie zum Beispiel *UIButton* oder *UISlider* erben allerdings direkt von *UIControl*. *UIControl* ist die Basisklasse für User-Interface Elemente, mit denen ein Benutzer direkt interagieren kann. Diese Klasse kann nicht direkt benutzt werden, um Controls zu erstellen. Sie definiert allerdings die gemeinsame Schnittstellen und Strukturen für die Subklassen. Außerdem implementiert *UIControl* den *Target-Action* Mechanismus, welcher eine Nachricht an andere Objekte sendet, wenn ein Event auftritt [App13h]. Um den Slider ohne

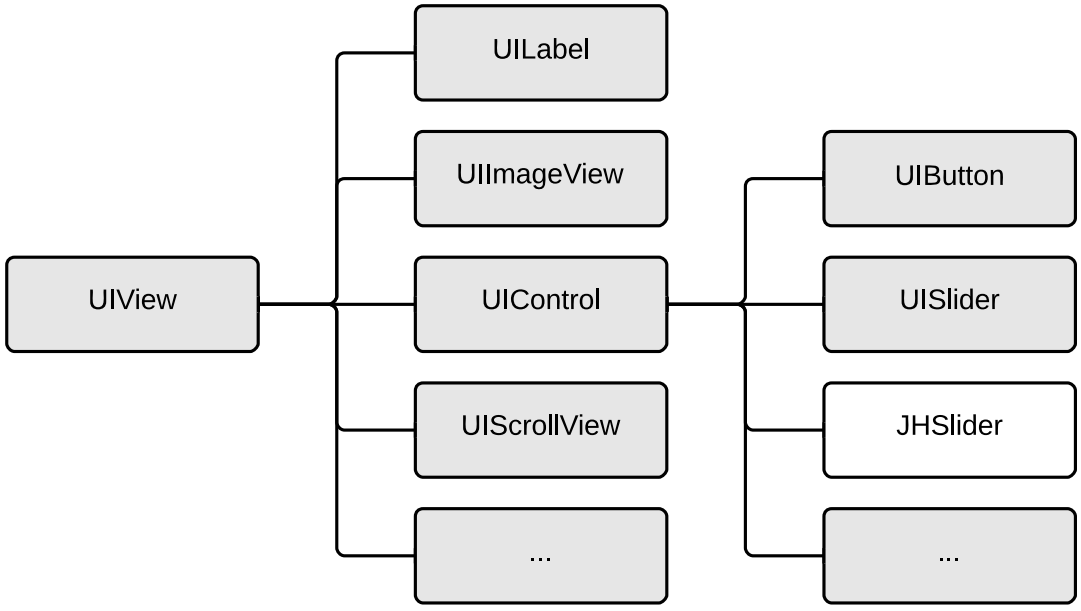

Abbildung 5.3.: UIKit Class Hirarchie [App13j]

initialen Wert bei Track Your Tinnitus zu erstellen, haben wir eine direkte Subklasse von *UIControl* erstellt. Wie in Abbildung 5.3 gezeigt, ist *JHSlider* die direkte Subklasse von *UIControl* und daher die Hauptklasse des Sliders.

Der aktuelle Zustand des Sliders muss zu jedem Zeitpunkt gespeichert werden. Listing 5.21 zeigt den Header des *JHSlider*, in dem drei Felder definiert sind: *maximumValue* speichert den maximalen Wert und *minimumValue* den minimalen Wert, den der Slider annehmen kann, *value* den aktuell eingestellten Wert. - *(void) resetSlider* und - *(void)setCustomValue:(float)value* definieren öffentliche Methoden, um den Slider auf den Zustand ohne initialen Wert zurückzusetzen und einen übergebenen Wert zu setzen.

Listing 5.21: Header von JHSlider

```
1 @interface JHSlider : UIControl
2
3 @property (nonatomic) float minimumValue;
4 @property (nonatomic) float maximumValue;
5 @property (nonatomic) float value;
6
7 - (void) resetSlider;
8 - (void)setCustomValue:(float)value;
9
10 @end
```

Listing 5.22 zeigt die Methode - *(id)initWithCoder:(NSCoder *)aDecoder* der *JHSlider* Klasse. Die - *(id)initWithCoder:(NSCoder *)aDecoder* Methode initialisiert einen Slider ohne initialen Wert. Hier werden zuerst die Standard-Werte für den maximalen Wert (Zeile 4), den minimalen Wert (Zeile 5) und den aktuellen Wert (Zeile 6) gesetzt. Zusätzlich werden zwei Hilfsvariablen gesetzt: *_useableTrackLength* (Zeile 7) beinhaltet die Länge des Bereiches, in dem der Knopf des Sliders bewegt werden kann und *_initialyTouched* (Zeile 8) ist ein Flag, das angibt, ob ein Benutzer bereits mit dem Slider interagiert hat.

Listing 5.22: Initialisierung eines JHSlider

```
1 - (id)initWithCoder:(NSCoder *)aDecoder {
2     self = [super initWithCoder:aDecoder];
3     if (self) {
4         _maximumValue = 1.0;
5         _minimumValue = 0.0;
6         _value = 0.0;
7         _useableTrackLength = self.bounds.size.width - self.bounds.size.height;
8         _initialyTouched = NO;
9         // ...
10    }
11    return self;
12 }
```

Erscheinungsbild Um das Erscheinungsbild des Controls zu bestimmen, bietet iOS zwei Möglichkeiten:

1. **Images:** Für die unterschiedlichen Elemente eines Controls werden jeweils Bilder erstellt. Dadurch kann das Aussehen eines Controls in einem Zeichenprogramm genau bestimmt werden. Um das Erscheinungsbild zu verändern müssen nur die entsprechenden Bilder

neu gezeichnet werden. Bei dieser Möglichkeit kann das Erscheinungsbild allerdings nicht direkt im Code beeinflusst werden.

2. **Core Graphics:** Die unterschiedlichen Elemente eines Controls werden in Layer unterteilt und jeweils mithilfe des *Core Graphics* Frameworks gezeichnet. Dabei muss der Code zum Zeichnen eines Layers implementiert werden. Allerdings ist es so auch möglich, das Erscheinungsbild direkt im Code zu verändern. Das Erscheinungsbild ist unabhängig von der Auflösung des Displays und wirkt auch dann nicht unscharf, falls in Zukunft ein Display mit höherer Auflösung in iOS Geräten eingesetzt wird.

Bei der Implementierung des Sliders ohne initialen Wert bei Track Your Tinnitus haben wir uns für die zweite Möglichkeit entschieden. Dabei wird der Slider mit zwei *CALayer* Subklassen realisiert. Wie in Abbildung 5.4 zu sehen, wird der Knopf des Sliders mit der Klasse *JHSliderKnobLayer* gezeichnet und die Leiste, auf der sich dieser Knopf bewegen lässt mit der Klasse *JHSliderTrackLayer*. Außerdem zeigt die Abbildung 5.4 die drei unterschiedlichen Zustände, die ein Slider einnehmen kann:

1. **Initialized:** Der Slider wurde gerade initialisiert und ein Benutzer hat noch nicht mit ihm interagiert.
2. **Pressed:** Der Benutzer interagiert gerade aktiv mit dem Slider, das heißt der Knopf wird verschoben und so ein neuer Wert eingestellt.
3. **Normal:** Ein Benutzer hat einen Wert eingestellt, interagiert aber aktuell nicht mit dem Slider.

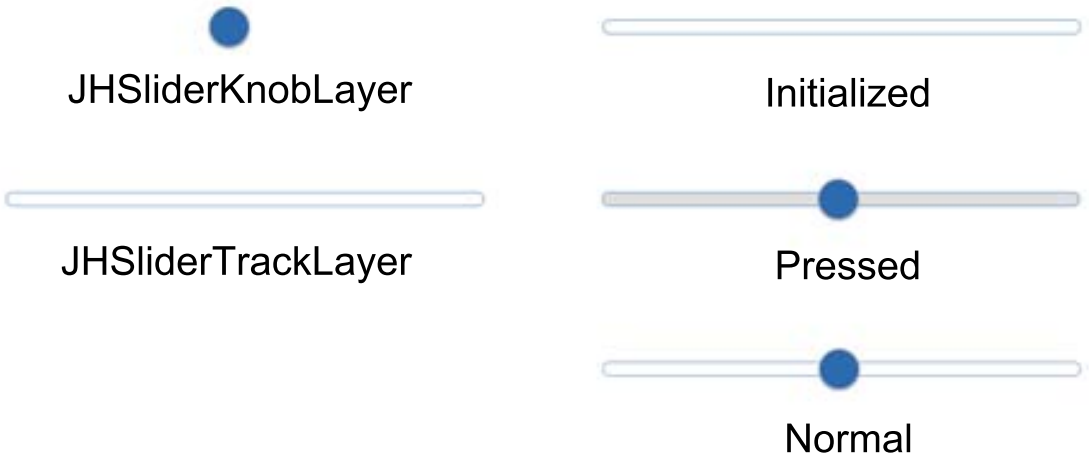

Abbildung 5.4.: Layer und Zustände des iOS-Sliders

Implementierung der Leiste des Sliders Listing 5.23 zeigt die Implementierung der Leiste, auf der sich der Knopf bewegen lässt. Die Klasse *JHSliderTrackLayer* ist eine Subklasse von *CALayer*. Die Leiste hat zwei Zustände: Normal und Pressed. Diese Zustände unterscheiden sich nur in der Füllfarbe der Leiste. Daher hat die Klasse *JHSliderTrackLayer* eine öffentliche Eigenschaft *_isTracking*, die angibt, ob ein Benutzer aktuell mit dem Slider interagiert. Der Inhalt eines *CALayer* wird in der Methode - *(void)drawInContext:(CGContextRef)context* ge-

zeichnet [App12a]. Dieser Methode wird eine Referenz auf den grafischen Context übergeben. Der grafische Context enthält alle Parameter, die benötigt werden, um das Gezeichnete auf dem Bildschirm zu rendern [App13b]. Zuerst wird der übergebene grafische Context als der aktuelle Context gesetzt, in dem gezeichnet werden soll (Zeile 16). Dieser Aufruf von *UIGraphicsPushContext* muss immer mit einem Aufruf der Methode *UIGraphicsPopContext* (Zeile 29) beendet werden. Zwischen diesen Aufrufen kann im Context gezeichnet werden. In den Variablen *primaryColor* (Zeile 17) und *secondaryColor* (Zeile 18) wird jeweils eine Referenz auf die Farbe des Rahmens und die Füllfarbe gehalten. Wenn ein Benutzer aktuell mit dem Slider interagiert, und so *_isTracking true* ist, wird eine andere Farbe als Füllfarbe festgelegt (Zeile 19 - 21). In den Zeilen 22 - 24 werden die Zeichenparameter des Context gesetzt: Die Füllfarbe (Zeile 22), die Farbe des Rahmens (Zeile 23) und die Linienbreite des Rahmens (Zeile 24). Wie in Abbildung 5.4 zu sehen, ist die Leiste des Sliders ein Rechteck, mit runden Enden. Diese wird in den Zeilen 25 und 26 mit Hilfe eines *UIBezierPath* erstellt. Der einfachste Weg ein Rechteck mit abgerundeten Enden zu erstellen, ist ein Rechteck mit abgerundeten Ecken zu erstellen, wobei der Radius der Ecken genau der Hälfte der Höhe des Rechtecks entspricht. Der Rahmen dieses Pfades wird in Zeile 27 gezeichnet und der Pfad in Zeile 28 gefüllt.

Listing 5.23: Implementierung der JHSliderTrackLayer Klasse

```
1  @implementation JHSliderTrackLayer
2
3  - (id)init {
4      self = [super init];
5      if (self) {
6          // Set the scale of the content to the scale of the screen.
7          if ([self respondsToSelector:@selector(setContentsScale:)])
8          {
9              self.contentsScale = [[UIScreen mainScreen] scale];
10         }
11     }
12     return self;
13 }
14
15 - (void)drawInContext:(CGContextRef)context {
16     UIGraphicsPushContext(context);
17     CGColorRef primaryColor = [UIColor colorWithRed:45.0/255.0 green:107.0/255.0 blue
       :177.0/255.0 alpha:1.0f].CGColor;
18     CGColorRef secondaryColor = [UIColor whiteColor].CGColor;
19     if (_isTracking) {
20         secondaryColor = [UIColor colorWithRed:223.0/255.0 green:223.0/255.0 blue:223.
       0/255.0 alpha:1.0f].CGColor;
21     }
22     CGContextSetFillColorWithColor(context, secondaryColor);
23     CGContextSetStrokeColorWithColor(context, primaryColor);
24     CGContextSetLineWidth(context, 1.0);
25     CGRect borderRect = CGRectInset(self.bounds, 1, 1);
26     UIBezierPath *borderPath = [UIBezierPath bezierPathWithRoundedRect:borderRect
       cornerRadius:self.bounds.size.height/2];
```

```
27      [borderPath stroke];
28      [borderPath fill];
29      UIGraphicsPopContext();
30  }
31
32  @end
```

Implementierung des Knopfes des Sliders Listing 5.24 zeigt die Implementierung des Knopfes des Sliders. Der Knopf hat, wie in Abbildung 5.4 zu sehen, ebenfalls zwei Zustände: Initialized und Normal/Pressed. Nach der Initialisierung ist der Knopf nicht zu sehen. Nachdem ein Benutzer das erste Mal mit dem Slider interagiert hat, wird der Knopf angezeigt. Ob ein Benutzer bereits mit dem Slider interagiert hat, wird durch die Variable *_initialyTouched* repräsentiert. Ist diese Variable *false* wird in der Methode *-(void)drawInContext:(CGContextRef)context* kein Inhalt gezeichnet. Ist die Variable *true* wird der Knopf in den Zeilen 18 bis 25 gezeichnet. Wie beim Zeichnen der Leiste werden zuerst die Farben bestimmt (Zeile 18 und 19) und als Füllfarbe (Zeile 20) bzw. Rahmenfarbe (Zeile21) gesetzt. Nachdem die Linienbreite gesetzt wurde (Zeile 22) wird der Knopf mithilfe eines *UIBezierPath* gezeichnet. Dieser Pfad wird allerdings mit einem Kreis initialisiert (Zeile 23) und mit der Füllfarbe gefüllt (Zeile 25). Zusätzlich wird mit *CGContextSetShadowWithColor* ein Schatten des Knopfes erstellt. Dieser Methode wird neben dem Context als ersten Parameter, der Offset des Schattenwurfs, der Unschärfegrad des Schattens und die Farbe des Schattens übergeben.

Listing 5.24: Implementierung der JHSliderKnobLayer Klasse

```
1  @implementation JHSliderKnobLayer
2
3  - (id)init {
4      self = [super init];
5      if (self) {
6          // Set the scale of the content to the scale of the screen.
7          if ([self respondsToSelector:@selector(setContentsScale:)])
8          {
9              self.contentsScale = [[UIScreen mainScreen] scale];
10         }
11     }
12     return self;
13 }
14
15 - (void)drawInContext:(CGContextRef)context {
16     UIGraphicsPushContext(context);
17     if (_initialyTouched) {
18         CGColorRef primaryColor = [UIColor colorWithRed:108.0/255.0 green:108.
       0/255.0 blue:108.0/255.0 alpha:1.0f].CGColor;
19         CGColorRef whiteColor = [UIColor colorWithRed:45.0/255.0 green:107.0/255.0
       blue:177.0/255.0 alpha:1.0f].CGColor;
20         CGContextSetFillColorWithColor(context, whiteColor);
```

```
21          CGContextSetStrokeColorWithColor(context, primaryColor);
22          CGContextSetLineWidth(context, 1.0);
23          UIBezierPath *borderPath = [UIBezierPath bezierPathWithOvalInRect:
        CGRectInset(self.bounds, 2, 2)];
24          CGContextSetShadowWithColor(context, CGSizeMake(0, 1.0), 2.0, [UIColor
        lightGrayColor].CGColor);
25          [borderPath fill];
26      }
27      UIGraphicsPopContext();
28  }
29  @end
```

Interaktive Logik des Sliders Listing 5.25 zeigt die Methoden, die benötigt werden, um den Slider interaktiv zu machen. Die Methode zur Initialisierung des Sliders wird so erweitert, dass die Layer für die Leiste und den Knopf des Sliders jeweils initialisiert und zum Layer des Sliders hinzugefügt werden (Zeile 5 - 10). Gefolgt von einem Aufruf der Methode - *(void)setLayerFrames*. Diese Methode bestimmt die Position des Knopfes auf der Leiste. Es wird zuerst die Hilfsmethode - *(float) positionForValue:(float)value* (Zeilen 48 - 52) aufgerufen, die für einen bestimmten Wert des Sliders die Position des Knopfes bestimmt (Zeile 18). Anhand dieses Wertes wird der Frame (X- und Y-Koordinate, Breite und Höhe) des Knopf-Layers gesetzt (Zeile 19). Zusätzlich wird der Wert der Variablen *_initialyTouched* übergeben. Durch einen Aufruf der Methode *setNeedsDisplay* eines Layers wird dieses Layer neu gezeichnet [App12a]. Die Methode - *(void) resetSlider* setzt den Slider auf den initialisierten Zustand zurück. - *(void) setCustomValue:(float)value* setzt den Slider in den normalen Zustand auf den übergebenen Wert. Diese zwei Methoden werden nur verwendet, wenn der Slider in einer *UITableViewCell* verwendet wird. Jede Veränderung der Layer müssen Teil einer *CATransaction* sein [App12b]. Daher wird in beiden Methoden der Aufruf der *setLayerFrames* Methode in eine *CATransaction* gekapselt (Zeilen 28 - 33 und Zeilen 40 - 45).

Listing 5.25: Interaktive Logik von JHSlider

```
1  - (id)initWithCoder:(NSCoder *)aDecoder {
2      self = [super initWithCoder:aDecoder];
3      if (self) {
4          // ...
5          _trackLayer = [JHSliderTrackLayer layer];
6          _trackLayer.isTracking = NO;
7          [self.layer addSublayer:_trackLayer];
8
9          _knobLayer = [JHSliderKnobLayer layer];
10         [self.layer addSublayer:_knobLayer];
11
12         [self setLayerFrames];
13     }
14     return self;
15 }
16
```

```
17 - (void)setLayerFrames {
18     float  upperKnobCentre = [self positionForValue:_value];
19     _knobLayer.frame = CGRectMake(upperKnobCentre - self.bounds.size.height / 2, 0,
       self.bounds.size.height, self.bounds.size.height);
20     _knobLayer.initialyTouched = _initialyTouched;
21     [_knobLayer setNeedsDisplay];
22 }
23
24 - (void) resetSlider  {
25     _initialyTouched = NO;
26     _value = 0.0f;
27     _previousTouchPoint = CGPointZero;
28     [CATransaction begin];
29     [CATransaction setDisableActions:YES] ;
30
31     [ self  setLayerFrames];
32
33     [CATransaction commit];
34 }
35
36 - (void)setCustomValue:(float)value {
37     _value = value;
38     _initialyTouched = YES;
39     _previousTouchPoint = CGPointMake([self positionForValue:_value], 0);
40     [CATransaction begin];
41     [CATransaction setDisableActions:YES] ;
42
43     [ self  setLayerFrames];
44
45     [CATransaction commit];
46 }
47
48 - (float )  positionForValue:( float )value
49 {
50     return  _useableTrackLength * (value - _minimumValue) /
51     (_maximumValue - _minimumValue) + (self.bounds.size.height / 2);
52 }
```

Implementierung der Touch-Handler Die Klasse *UIControl* bietet vier Methoden um Touches auf dem Control zu überwachen [App11]:

1. ***- (BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event*** wird aufgerufen, wenn ein Touch des Benutzers innerhalb der Grenzen des Controls registriert wird.

2. ***- (BOOL)continueTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event*** wird durchgehend aufgerufen um einen Touch innerhalb der Grenzen des Controls zu überwachen.
3. ***- (void)endTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event*** wird aufgerufen, wenn der letzte Touch endet.
4. ***- (void)cancelTrackingWithEvent:(UIEvent *)event*** wird aufgerufen, wenn das Control die Überwachung eines Touches beenden soll.

Beim Slider ohne initialen Wert unter iOS sind die ersten drei Methoden implementiert, die in Listing 5.26, Listing 5.27 und Listing 5.28 gezeigt werden.
*- (BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event* ist die erste Methode, die aufgerufen wird, wenn ein Touch innerhalb der Grenzen eines Controls registriert wird. Listing 5.26 zeigt die Implementierung dieser Methode. Da ein Benutzer beim Aufruf dieser Methode mit dem Slider interagiert hat, wird das Flag *_initialyTouched* auf *YES* gesetzt (Zeile 5). In den Zeilen 8 bis 12 wird der neue Wert berechnet, der durch diesen Touch verändert wird. Dazu wird zuerst der Unterschied zwischen dem vorherigen Touch-Punkt und dem aktuell registrierten Touch-Punkt in Punkten berechnet (Zeile 8). Dieser Unterschied der Touch-Punkte wird dann in den Unterschied innerhalb des Wertebereichs des Sliders umgerechnet (Zeile 9). Der Wert (*_value*) des Sliders wird dann um diesen Unterschied verändert (Zeile 10). Anschließend wird dieser neue Wert entsprechend den Grenzen des Wertebereichs mit Hilfe des Makros *BOUND* (Zeile 1) angepasst, falls der neue Wert nicht mehr im Wertebereich liegt. Der aktuelle Touch-Punkt wird in der Variablen *_previousTouchPoint* gespeichert, um beim nächsten Aufruf den Unterschied der Touches korrekt berechnen zu können (Zeile 12). Da diese Methode aufgerufen wird, wenn ein Touch beginnt, wird das *isTracking* Flag der Leiste des Sliders auf *YES* gesetzt, so dass sich die Füllfarbe ändert (Zeile 13). Danach werden die Layer neu gezeichnet (Zeile 17), wobei dies wieder in eine *CATransaction* gekapselt wird. Da der Slider dem *Target-Action* Mechanismus entspricht, wird noch ein *UIControlEventValueChanged* Event gesendet, um Objekte, die den Slider überwachen zu informieren, dass sich der Wert verändert hat.

Listing 5.26: Touch Handler des JHSlider - Begin Touch

```
1  #define BOUND(VALUE, UPPER, LOWER)  MIN(MAX(VALUE, LOWER), UPPER)
2
3  - (BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event
4  {
5      _initialyTouched = YES;
6      CGPoint touchPoint = [touch locationInView:self];
7
8      float  delta = touchPoint.x - _previousTouchPoint.x;
9      float  valueDelta = (_maximumValue - _minimumValue) * delta /
       _useableTrackLength;
10     _value += valueDelta;
11     _value = BOUND(_value, _maximumValue, _minimumValue);
12     _previousTouchPoint = [touch locationInView:self];
13     _trackLayer.isTracking = YES;
14     [CATransaction begin];
15     [CATransaction setDisableActions:YES] ;
```

```
16
17      [self setLayerFrames];
18
19      [CATransaction commit];
20      [self sendActionsForControlEvents:UIControlEventValueChanged];
21      return YES;
22  }
```

Listing 5.27 zeigt die Implementierung der Methode - *(BOOL)continueTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event*. Diese wird bei jeder Veränderung des Touches erneut aufgerufen. Wie bereits in der - *(BOOL)beginTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event* Methode wird in dieser Methode der Wert des Sliders neu gesetzt (Zeilen 3 - 10) und die Layer neu gezeichnet (Zeilen 12 - 17). Zusätzlich wird ebenfalls ein *UIControlEventValueChanged* Event gesendet. Während eines Touches ändern sich keine Flags, da der Slider immer im Zustand Pressed verbleibt. Nur der Knopf wird verschoben.

Listing 5.27: Touch Handler des JHSlider - Continue Touch

```
1  - (BOOL)continueTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event
2  {
3      CGPoint touchPoint = [touch locationInView:self];
4
5      float delta = touchPoint.x - _previousTouchPoint.x;
6      float valueDelta = (_maximumValue - _minimumValue) * delta /
       _useableTrackLength;
7
8      _previousTouchPoint = touchPoint;
9      _value += valueDelta;
10     _value = BOUND(_value, _maximumValue, _minimumValue);
11
12     [CATransaction begin];
13     [CATransaction setDisableActions:YES] ;
14
15     [self setLayerFrames];
16
17     [CATransaction commit];
18     [self sendActionsForControlEvents:UIControlEventValueChanged];
19     return YES;
20 }
```

Die letzte Methode, die aufgerufen wird, wenn ein Touch endet, zeigt Listing 5.28. Diese Methode ändert den Zustand des Sliders von Pressed zurück auf Normal. Der Knopf des Sliders bleibt bestehen, allerdings wird das Flag der Leiste *isTracking* auf *NO* gesetzt, so dass sich die Füllfarbe der Leiste auf ihre Ursprungsfarbe ändert (Zeile 2). Da nach jeder Änderung die Layer neu gezeichnet werden müssen, folgt auch in dieser Methode ein Aufruf von *setLayerFrames* (Zeile 6).

Listing 5.28: Touch Handler des JHSlider - End Touch

```
1  - (void)endTrackingWithTouch:(UITouch *)touch withEvent:(UIEvent *)event {
```

```
2       _trackLayer.isTracking = NO;
3       [CATransaction begin];
4       [CATransaction setDisableActions:YES] ;
5
6       [ self  setLayerFrames];
7
8       [CATransaction commit];
9       [ self  sendActionsForControlEvents:UIControlEventValueChanged];
10  }
```

Nach der Implementierung des Erscheinungsbildes, der interaktiven Logik und der Touch-Handler, ist der Slider ohne initialen Wert komplett.

5.4.1.2. Slider ohne initialen Wert in Android

Android bietet, wie iOS, vorgefertigte User-Interface Elemente, wie zum Beispiel *Button*, *TextView*, oder *SeekBar*. Android bietet ebenfalls aber keine direkte Möglichkeit, einen Slider so einzustellen, dass nach der Initialisierung der Knopf nicht angezeigt wird. Daher haben wir bei der Android Version der Track Your Tinnitus App den Slider selbst implementiert.
Die Basisklasse für alle User-Interface Elemente unter Android ist *View* [Good]. Für einen normalen Slider bietet Android die Klasse *SeekBar*, von der die Klasse des Sliders ohne initialen Wert direkt erbt. So müssen, unter anderen, die Methoden *getProgress()* oder *onMeasure()* nicht neu implementiert werden, da sich an ihnen, gegenüber der Standard-Implementierung nichts ändert. Listing 5.29 zeigt die Variablen und den Konstruktor der Klasse *JHSeekBar*. Zunächst werden die benötigten Variablen definiert. Für das Zeichnen der Leiste wird ein Objekt der Klasse *Paint* benötigt (Zeile 1). Außerdem wird der Pfad, der später gezeichnet wird, in einem Objekt der Klasse *Path* gespeichert (Zeile 2). Die Variable *trackPosition* hält die Position des Knopfes auf dem Slider (Zeile 3). Um zu überwachen, ob ein Benutzer bereits mit dem Slider interagiert hat, wird das Flag *initialyTouched* verwendet. Zeile 5 bis Zeile 7 sind die Variablen für das Bild des Knopfes. Im Konstruktor werden die Parameter des Objekts der Klasse *Paint* gesetzt, die das Zeichnen beeinflussen [Gook]:

- Anti-Aliasing (Zeile 16)
- Linienbreite (Zeile 17)
- Zeichenfarbe (Zeile 18)
- Linienstil (Zeile 19)

Listing 5.29: Implementierung der JHSeekBar - Initialisierung

```
1 private Paint paint = new Paint();
2 Path trackerPath = new Path();
3 float  trackPosition = -1f;
4 boolean initialyTouched = false;
5 Bitmap thumbBitmap;
6 Bitmap thumbBitmapNormal = BitmapFactory.decodeResource(getResources(), R.
      drawable.scrubber_control_normal_holo);
```

```
7  Bitmap thumbBitmapTracking = BitmapFactory.decodeResource(getResources(), R.
       drawable.scrubber_control_pressed_holo);
8
9  public JHSeekBar(Context context) {
10     super(context);
11 }
12
13 // Called when initialized from xml
14 public JHSeekBar (Context context, AttributeSet attrs) {
15     super(context, attrs);
16     paint.setAntiAlias(true);
17     paint.setStrokeWidth(4f);
18     paint.setColor(Color.LTGRAY);
19     paint.setStyle(Paint.Style.STROKE);
20 }
```

Erscheinungsbild Im Gegensatz zur iOS-Version des Sliders ist die Leiste des Slider unter Android im Code gezeichnet, der Knopf wird allerdings als Bild geladen und angezeigt. Abbildung 5.5 zeigt die einzelnen Teile und Zustände des Android Sliders. Der Knopf wird als Bilddatei geladen und je nach Zustand verändert. Die Leiste wird hingegen direkt im Code gezeichnet. Da die Logik für den Slider unter Android in einer Klasse zusammengefasst ist, gibt es keine einzelnen Klassen für den Knopf und die Leiste. Genau, wie der Slider unter iOS, hat auch der Slider unter Android 3 Zustände: “Initialized”, “Pressed” und “Normal” (siehe Kapitel 5.4.1.1).

Abbildung 5.5.: Teile und Zustände des Android-Sliders

Listing 5.30 zeigt die Implementierung der *onDraw()* Methode. Diese Methode ist für das Zeichnen auf dem Hintergrund verantwortlich. Zunächst wird der Pfad für die Leiste erstellt (Zeile 3 und Zeile 4). Dieser Pfad wird anschließend auf das *Canvas* gezeichnet (Zeile 5). Wenn ein Benutzer noch nicht mit dem Slider interagiert hat und so die Variable *initialyTouched false*

ist, wird der Knopf nicht gezeichnet. Ansonsten wird zur Sicherheit zuerst überprüft, ob sich der Wert des Sliders bereits verändert hat (Zeilen 7 - 9). Danach folgt die Umrechnung des Wertes des Sliders in Punkte im *Canvas* auf der X-Achse (Zeile 10). Da der Mittelpunkt des Bildes des Knopfes genau an der Position des Wertes des Sliders sein soll, wird dieser X-Achsen Wert in den X-Wert des linken Randes des Bildes umgerechnet (Zeile 12). In den Zeilen 13 bis 16 wird überprüft, ob dieser X-Wert außerhalb der Leiste liegen würde und wird entsprechend angepasst. So wird verhindert, dass ein Wert außerhalb des Wertebereichs des Sliders eingestellt werden kann. Zuletzt wird das Bild für den Knopf an die entsprechende Position im *Canvas* gezeichnet (Zeile 18).

Listing 5.30: Implementierung der JHSeekBar - *onDraw* Methode

```
1  @Override
2  protected void onDraw(Canvas canvas) {
3      trackerPath.moveTo(thumbBitmap.getWidth()/2, canvas.getHeight()/2);
4      trackerPath.lineTo(canvas.getWidth()−thumbBitmap.getWidth()/2, canvas.getHeight
       ()/2);
5      canvas.drawPath(trackerPath, paint);
6      if (initialyTouched) {
7          if (getProgress() == −1){
8              return;
9          }
10         trackPosition = ((canvas.getWidth())/100)∗getProgress();
11
12         float leftPosition = trackPosition−thumbBitmap.getWidth()/2;
13         if ( leftPosition < 0)
14             leftPosition = 0;
15         if (( leftPosition +thumbBitmap.getWidth()∗2) > canvas.getWidth())
16             leftPosition = canvas.getWidth()−thumbBitmap.getWidth();
17
18         canvas.drawBitmap(thumbBitmap, leftPosition, 0, paint);
19         }
20     }
21 }
```

Da die Klasse *JHSeekBar* direkt von *SeekBar* erbt, sind keine Methoden für die interaktive Logik notwendig. Für das Einstellen des Wertes des Sliders werden die Standard-Implementierungen der System Methoden verwendet.

Implementierung des Touch-Handlers Die Klasse *View* bietet eine Methode, um einen Touch auf dem Control zu überwachen: *onTouchEvent (MotionEvent event)* [Gooq]. Listing 5.31 zeigt die Implementierung dieser Methode in der Klasse des Sliders ohne initialen Wert. Anhand des übergebenen *MotionEvent* kann ermittelt werden, um welche Art von Touch-Event es sich handelt. Unter anderen bietet diese Klasse folgende drei Konstanten für eine ausgeführte *Action*, die in der Implementierung verwendet werden [Goog]:

1. ***MotionEvent.ACTION_ MOVE*** Eine Änderung einer gedrückten Geste ist aufgetreten.

2. ***MotionEvent.ACTION_ UP*** Eine Geste wurde beendet.
3. ***MotionEvent.ACTION_ CANCEL*** Die aktuelle Geste wurde abgebrochen.

Da ein Benutzer mit dem Slider interagiert hat, sobald die Methode *onTouchEvent()* aufgerufen wurde, wird das Flag *initialyTouched* auf *true* gesetzt, damit der Knopf beim nächsten Zeichenvorgang angezeigt wird (Zeile 2). Die Position des Touches wird in der Variablen *trackPosition* gespeichert (Zeile 3). Erst jetzt findet die Unterscheidung in die verschiedenen *Actions* statt (Zeile 5). Handelt es sich bei dem *MotionEvent* um eine Änderung einer gedrückten Geste (Zeilen 6ff), wird der Zustand des Sliders auf "Pressed" gesetzt, d.h. die Farbe der Leiste verändert (Zeile 7) und das Bild des Knopfes verändert (Zeile 8). Wurde die Geste beendet (Zeilen 10ff) oder abgebrochen (Zeilen 14ff), wird der Slider auf den Zustand "Normal" gesetzt. Dabei wird sowohl die Farbe der Leiste, wie auch das Bild des Knopfes wieder auf seine initiale Farbe zurückgesetzt (Zeile 11 und Zeile 12, bzw. Zeile 15 und Zeile 16). Mit der folgenden Abfrage (Zeilen 21 - 23) wird verhindert, dass der Knopf des Sliders erscheint, wenn ein Benutzer beim Scrollen ein Touch Event auf dem Slider auslöst. Danach wird die *onTouchEvent()* Methode der Superklasse aufgerufen, die unter anderem den Wert des Sliders einstellt und den Slider neu zeichnet.

Listing 5.31: Implementierung der JHSeekBar - *onTouchEvent* Methode

```
1  public boolean onTouchEvent (MotionEvent event) {
2      initialyTouched = true;
3      trackPosition = event.getX();
4
5      switch (event.getAction()) {
6          case MotionEvent.ACTION_MOVE:
7                  paint.setColor(Color.GRAY);
8                  thumbBitmap = thumbBitmapTracking;
9                  break;
10         case MotionEvent.ACTION_UP:
11             paint.setColor(Color.LTGRAY);
12             thumbBitmap = thumbBitmapNormal;
13             break;
14         case MotionEvent.ACTION_CANCEL:
15             paint.setColor(Color.LTGRAY);
16             thumbBitmap = thumbBitmapNormal;
17             break;
18         default:
19     }
20
21     if(event.getAction() == MotionEvent.ACTION_CANCEL && getProgress() == 0)
       {
22         initialyTouched = false;
23     }
24     return super.onTouchEvent(event);
25 }
```

Durch das direkte Erben von der Klasse *SeekBar* ist die Implementierung unter Android mit deutlich weniger Code zu erreichen.

6 Anforderungsabgleich

In diesem Kapitel werden die Anforderungen, die in Kapitel 2 definiert wurden, mit der Implementierung und den Funktionen der Webseite bzw. der Apps abgeglichen. Wie schon bei der Definition der Anforderungen ist dieses Kapitel in funktionale und nichtfunktionale Anforderungen unterteilt.

6.1. Funktionale Anforderungen

Dieser Abschnitt zeigt den Abgleich der funktionalen Anforderungen an die Webseite bzw. die Apps. Dabei wird überprüft, ob in der aktuellen Implementierung alle funktionalen Anforderungen erfüllt sind.

Nr.	**Beschreibung**	**Anforderungsanalyse**
1.	Fragebogen zur Überwachung der Schwankungen der Tinnituswahrnehmung	Anforderung erfüllt (siehe Kapitel 4.2.5)
2.	Einstellungen der Benachrichtigungszeiten	Anforderung erfüllt (siehe Kapitel 4.2.6)
3.	Einstellung des Klingeltons einer Benachrichtigung	Anforderung erfüllt (siehe Kapitel 4.2.7)
4.	Anzeige der Ergebnisse in den Apps	Anforderung erfüllt (siehe Kapitel 4.2.8)
5.	Registrierung in den Apps ermöglichen	Anforderung erfüllt (siehe Kapitel 4.2.2)
6.	Slider ohne initialen Wert	Anforderung erfüllt (siehe dazu Kapitel 5.4.1).

Nr.	Beschreibung	Anforderungsanalyse
7.	Messung des Geräuschpegels	Anforderung erfüllt. Der Geräuschpegel wird auf beiden Plattformen gemessen, wenn ein Benutzer dies nicht explizit abschaltet.
8.	Apps auch ohne Internetverbindung benutzbar.	Anforderung erfüllt. Alle Daten werden in der lokalen Datenbank auf dem Gerät gespeichert. So ist eine volle Funktion auch ohne funktionierende Internetverbindung garantiert.
9.	Ausfüllen der statistischen Fragebögen auf der Webseite und in den Apps	Anforderung erfüllt (siehe Kapitel 4.1.3 und Kapitel 4.2.3)
10.	Erweiterbarkeit der statistischen Fragebögen	Anforderung erfüllt (siehe Kapitel 4.1.6)
11.	Synchronisierung der Antworten auf die statistischen Fragebögen	Anforderung erfüllt (siehe Kapitel 4.2.3)
12.	Synchronisierung der Ergebnisse	Anforderung erfüllt. Nach dem Ausfüllen des Fragebogens zur Überwachung der Tinnituswahnehmung (siehe Kapitel 4.2.5) wird dieser Datensatz an den Server übertragen.
13.	Exportfunktion der Daten eines Benutzers	Anforderung erfüllt. Ein Benutzer kann auf der Webseite seine Daten im Bereich “Download” exportieren.

6.2. Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen werden in der folgenden Tabelle abgeglichen.

Nr.	Beschreibung	Anforderungsanalyse
1.	Ausfüllen des Fragebogens zur Überwachung der Schwankungen der Tinnituswahrnehmung sollte in weniger als einer Minute möglich sein.	Anforderung erfüllt. Nach persönlichen Messungen ist der Fragebogen zur Überwachung der Tinnituswahrnehmung in ca. 40 Sekunden beantwortet.
2.	Identisches Aussehen der Apps unter iOS und Android.	Anforderung teilweise erfüllt. Es wurde darauf geachtet, dass die Apps so weit wie möglich identisch aussehen und sich gleich bedienen lassen. Dies konnte nicht vollständig erreicht werden. Zum Beispiel unterscheiden sich die Apps in der Art, wie sich das Hauptmenü öffnen lässt.

Nr.	Beschreibung	Anforderungsanalyse
3.	Anpassung des Designs der iOS App an iOS 6 und iOS 7	Anforderung erfüllt. Das User-Interface der iOS Version wurde an beide Versionen angepasst.
4.	Keine Speicherung der E-Mail Adresse oder IP-Adresse	Anforderung erfüllt. Bei der Registrierung kann ein Benutzer angeben, ob seine E-Mail Adresse gespeichert werden soll, oder nicht. Wenn er dies nicht möchte, wird ein Code per E-Mail zugesendet, mit dem er sein Passwort zurücksetzen kann. Danach wird die E-Mail Adresse in der Datenbank gelöscht. Auch eine Speicherung der IP-Adresse findet weder beim Zugriff auf die Webseite noch beim Zugriff über die Apps statt.

7 Fazit

Die erste Phase der Entwicklung des Track Your Tinnitus Projekts ist zur Zeit dieser Arbeit abgeschlossen. Die Funktionalität der Webseite und der Apps ist derart umgesetzt, dass ein Benutzer die Schwankungen seiner Tinnituswahrnehmung überwachen und auswerten kann. Der erste Teil dieses Kapitels (Kapitel 7.1) fasst die Erkenntnisse aus dieser Arbeit zusammen. Kapitel 7.2 beschreibt weiter Ideen, die denkbar wären, um die Funktion des Frameworks weiter zu verbessern. Der letzte Teil beinhaltet meine persönliche Erfahrung beim Entwickeln dieses Projekts (Kapitel 7.3).

7.1. Zusammenfassung

Im Rahmen dieser Diplomarbeit ist ein mobiles Framework zur Überwachung der Schwankungen der Tinnituswahnehmung entstanden.
Die Anforderungen wurden in mehreren Meetings definiert. Auch während der Arbeit kamen einige neue Anforderungen und Ideen hinzu, die in dieser Arbeit berücksichtig wurden.
Aufgrund dieser Anforderungen reichte es in der Implementierung der Apps nicht aus, auf die vorgegebenen User-Interface Elemente der zwei Plattformen iOS und Android zurückzugreifen. So musste der Slider ohne initialen Wert selbst erstellt werden. Im Gegensatz zu Android bietet iOS keine Checkboxen oder Radio Buttons. Daher mussten auch diese implementiert werden. Diese Arbeit beschreibt daher allgemein, wie eigene User-Interface Elemente auf beiden Plattformen entwickelt werden können. Dabei wurde gezeigt, wie einfach Elemente (zum Beispiel Diagramme) gezeichnet werden können, aber auch, wie eigene interaktive Elemente implementiert werden können (zum Beispiel ein Slider ohne initialen Wert). Außerdem wurde gezeigt, wie Benachrichtigungen auf den jeweiligen Plattformen an einen Benutzer gesendet werden, ohne dass eine funktionierende Internetverbindung notwendig ist. Ebenso wurde beschrieben, wie Benachrichtigungen in einem vom Benutzer definierten Zeitraum möglichst zufällig verteilt werden können.
Die Webseite wurde mit dem Laravel Framework entwickelt. Dies biete mit *Eloquent ORM*

einen komfortablen Weg mit einer relationalen Datenbank zu interagieren. Da sich das Laravel Framework an das *Model-View-Controller Patter* hält, ist eine gut strukturierte Webentwicklung möglich.
Am Ende dieser Arbeit ist die Entwicklung des Frameworks in einem Status, in dem eine Veröffentlichung ohne Probleme möglich wäre. Die iOS App wurde bereits von Apple für die Veröffentlichung im Apple App Store freigegeben.

7.2. Ausblick

Dieser Abschnitt beschreibt die Ideen, die während der Entwicklung von Track Your Tinnitus aufgekommen sind, um das Projekt zu verbessern. Dabei werden einige Verbesserungen am User-Interface (zum Beispiel die Verbesserung der Visualisierung der Ergebnisse) aber auch grundlegende Änderungen (zum Beispiel eine Version für Tablets) dargestellt.

7.2.1. Verbesserung der Visualisierung der Ergebnisse

Die Visualisierung der Ergebnisse auf der Webseite und in den Apps bietet bisher keine Möglichkeit, sich nur bestimmte Bereiche im Detail anzusehen, oder auch verschiedene Fragen zu vergleichen. Die Visualisierung kann, sowohl in den Apps, als auch auf der Webseite in ein Diagramm zusammengefasst werden. In diesem Diagramm lassen sich dann die Kurven einzelner Fragen separat hinzu- oder wegschalten. So kann auch ein Vergleich unterschiedlicher Fragen erfolgen. Zusätzlich kann das Diagramm gezoomt werden, um sich Teilbereiche genauer ansehen zu können.
Bisher gibt es außerdem noch keine Visualisierung der Ergebnisse aller Benutzer. Ein Forscher hat bisher nur die Möglichkeit, die Daten von allen Benutzern direkt aus der Datenbank zu laden und in eigenen Programmen Auswertungen dieser Daten zu erstellen. Eine Integration von bestimmten Auswertungen über den kompletten Datenbestand ist daher denkbar.

7.2.2. Tablet Version

Für die Überwachung der Schwankungen der Tinnituswahrnehmung ist ein Tablet nur bedingt geeignet. In der Regel hat ein Benutzer sein Smartphone immer bei sich, wo hingegen ein Tablet meist nicht immer mitgeführt wird. Eine App kann auch nicht erkennen, ob ein Benutzer aktuell sein Smartphone oder sein Tablet benutzt, falls die App auf beiden Geräten installiert ist. Eine bessere Möglichkeit, um eine Version der App für Tablet zu entwickeln, ist die Auswertung der Ergebnisse. Durch ein deutlich größeres Display lassen sich die Daten genauer visualisieren.
In diesem Zusammenhang ist auch eine Tablet Version für Ärzte und Therapeuten von Tinnituspatienten denkbar. Nachdem ein Benutzer auf seinem Smartphone seinem Arzt entsprechende Rechte eingeräumt hat, kann dieser auf die Ergebnisse des Benutzers zugreifen. Dabei ist allerdings der Schutz der Daten des Patienten deutlich schwieriger. Eine Möglichkeit diese Problematik zu bewältigen, wäre die Übergabe von vom Server generierten Schlüsseln. So könnte der Patient zuerst einen Schlüssel eingeben, um das Gerät des Arztes identifizieren zu können. Danach wird dem Patient ein Schlüssel angezeigt, welchen wiederum der Arzt eingeben muss, um auf die Daten zugreifen zu können. Ein Übergabe dieser Schlüssel mittels QR-Codes wäre denkbar.

7.2.3. Reward System

Bisher wird die App nach dem Ausfüllen des Fragebogens zur Überwachung der Tinnituswahrnehmung beendet. Ein Benutzer kann zwar in der Ergebnisdarstellung sehen, dass die Daten aufgezeichnet werden, eine Motivation den Fragebogen öfters oder regelmäßiger auszufüllen findet nicht statt. Daher wäre ein Reward System denkbar, welches zum Beispiel pro ausgefüllten Fragebogen Punkte vergibt. Angelehnt an die Highscore eines Spiels, kann auch ein Vergleich mit anderen Benutzern motivierend wirken (wobei keine Benutzernamen oder ähnliches erkennbar sein dürfen).
Ein anderer Ansatz, um einen Benutzer zu motivieren, wäre nach jedem Ausfüllen einen Motivationssatz anzuzeigen, der dynamisch generiert wird. Zum Beispiel: "Sie haben gerade Ihren 10. Fragebogen beantwortet. Weiter so!". Eine weiter Möglichkeit wäre auch positive Fakten zum Tinnitus einzublenden. Zum Beispiel: "Heute ist Ihr Tinnitus leiser als gestern.".

7.2.4. Unterstützung für verschiedene Versionen von Fragebögen

Die aktuelle Implementierung unterstützt keine Versionierung der statistischen Fragebögen. Wenn eine Frage vom Inhalt oder der Formulierung geändert wird, kann diese Änderung in der Datenbank nicht nachverfolgt werden. Zum Beispiel wird eine Frage von "Wie gestresst fühlen Sie sich gerade?" in "Wie entspannt fühlen Sie sich gerade?" geändert, wird ein Benutzer eine unterschiedliche Antwort auf die Frage geben. Bisher wird nur der Wortlaut der Frage gespeichert und ein Administrator wird darauf hingewiesen, eine Frage nicht zu verändern. Eine vollständige Versionierung der statistischen Fragebögen würde dieses Problem beseitigen.

7.2.5. Verbesserung der Unterstützung für Forschungsgruppen oder Projekte

Bisher können einzelne Benutzergruppen erstellt und Fragebögen diesen Gruppen zugeordnet werden. Auch eine direkte Registrierung eines Benutzers in einer dieser Gruppen ist möglich, allerdings nur auf der Webseite. Einerseits kann das Projekt so erweitert werden, dass in einer Gruppe auch eigene Fragen zur Überwachung der Schwankungen der Tinnituswahrnehmung definiert werden können. Diese werden dann in den Apps heruntergeladen und müssen auch in der Ergebnisdarstellung visualisiert werden. Andererseits ist es möglich, eine App auf dem Smartphone mit eigenen Parametern (zum Beispiel der Gruppen-Id) zu starten, um so auch eine direkte Registrierung in einer Gruppe in den Apps zu ermöglichen. In iOS und Android ist es möglich, eigene URLs zu definieren, mit dem ein Benutzer die App über einen Link in einer E-Mail oder dem Browser öffnen kann. Eine andere Möglichkeit, um zum Beispiel von einem Flyer eine *Id* an die App zu übergeben, wäre ein QR-Code.

7.2.6. Dynamische Einstellung des Benachrichtigungsplanes

Ein Benutzer hat die Möglichkeit, die Benachrichtigungszeiten in der App selbst zu bestimmen. Es ist aber auch möglich, dass für einzelne Studien die Benachrichtigungszeiten vorgegeben werden. Im Zuge der besseren Unterstützung von Forschergruppen oder Projekten könnte zusätzlich noch die Möglichkeit für Administratoren geschaffen werden, die Benachrichtigungszeiten auf

der Webseite für eine Studie vorzugeben. Auch eine Option, ob ein Benutzer diese noch selbst verändern kann, oder diese fest vorgegeben wird, ist denkbar.

7.2.7. Verbesserungen am User-Interface und Hilfe

Bisher geht aus der Darstellung der statistischen Fragebögen in den Apps nicht hervor, welche Fragen optional sind, also vom Benutzer nicht zwingend beantwortet werden müssen. Da das Speichern eines statistischen Fragebogens in den Apps nicht möglich ist, ohne alle nicht-optionalen Fragen zu beantworten, kann ein Benutzer nicht erkennen, welche Frage er noch beantworten muss, um den Fragebogen zu speichern. Eine farbliche Hervorhebung oder ähnliche Markierung würde dieses Problem beheben.
Die Webseite und die Apps beinhalten außer dem About-Bereich bisher keinerlei Hilfetexte. Auch werden in den Apps die einzelnen Funktionen nur kurz erklärt. Falls ein Benutzer weitere Hilfe benötigt, oder spezielle Fragen hat, findet er keinerlei Texte. Die Erstellung eines "FAQ" bzw. einer ausführlichen Hilfe ist daher unumgehbar.

7.2.8. Prozess-Management-Technologie

Prozess-Management-Technologie [Rei12], vor allem mobile Prozesse [Pry14] [Pry10a] [Pry10b] [Pry12] [Pry13], sind weitere vielversprechende Techniken, die man in diesem Kontext einsetzen könnte.

7.3. Persönliche Erfahrungen

In diesem Abschnitt beschreibe ich meine persönlichen Erfahrungen bei der Entwicklung von Track Your Tinnitus.
Am Anfang der Entwicklung der Webseite suchte ich ein passendes PHP-Framework, um die Webseite und die API zu implementieren. Dabei haben mich die Vorteile des Laravel Frameworks, vor allem das einfache Interagieren mit der Datenbank, überzeugt. Obwohl ich bisher noch keine Erfahrung mit diesem Framework gemacht habe, konnte ich schon nach kurzer Zeit erste Ergebnisse erzielen. Während der Entwicklung des Projekts wurde allerdings Laravel Version 4 veröffentlich. Da sich einige grundlegenden Funktionen von Version 3 zu Version 4 verändert haben, war ein einfacher Umstieg leider nicht möglich.
Bei der Entwicklung der Apps habe ich zuerst die Version für iOS entwickelt. Dies hatte zwei Gründe: Ich habe persönlich mehr Erfahrung in der Entwicklung von Apps für iOS und die Restriktionen bei einer Veröffentlichung im Apple App Store müssen von Anfang an in der Entwicklung beachtet werden. Da die Menüstruktur, die Funktionen und die Architektur in iOS bereits implementiert waren, konnte ich die Android App in wenigen Wochen fertigstellen. Im Vergleich gefällt mir die Implementierung von Apps für iOS besser, was aber zum größten Teil daran liegt, dass die Entwicklertools deutlich besser funktionieren.
Während der Entwicklung der iOS App wurde außerdem iOS 7 veröffentlich. Da die Veränderungen am Design des User-Interface des iPhones sehr deutlich sind, musste auch die App für iOS 7 optimiert werden. Diese Optimierung stellte sich allerdings als erstaunlich schwierig heraus, da jeweils ein Design des User-Interface für iOS 6 und iOS 7 implementiert werden musste.

Diese Arbeit deckt die drei Bereiche ab, in denen ich mich am liebsten bewege: PHP-Entwicklung, iOS-Entwicklung und Android-Entwicklung. Daher hatte ich zu jedem Zeitpunkt dieser Arbeit viel Spaß.

A

Statistische Fragebögen

A.1. Mini Tinnitus Fragebogen

Mit dem Mini Tinnitus Fragebogen (Mini TF12) lässt sich herausfinden, ob der Tinnitus Einflüsse auf die Gefühle, Verhaltensweisen oder Einstellungen eines Patienten hat [Hil04]. Jede Frage hat drei Antwortmöglichkeiten (“stimmt”, “teilweise”, “stimmt nicht”), von denen jeweils nur eine ausgewählt werden kann. Folgende zwölf Fragen sind enthalten:

1. Ich bin mir der Ohrgeräusche vom Aufwachen bis zum Schlafengehen bewußt.
2. Ich mache mir wegen der Ohrgeräusche Sorgen, ob mit meinem Körper ernstlich etwas nicht in Ordnung ist.
3. Wenn die Ohrgeräusche andauern, wird mein Leben nicht mehr lebenswert sein.
4. Auf Grund der Ohrgeräusche bin ich mit meiner Familie und meinen Freunden gereizter.
5. Ich sorge mich, dass die Ohrgeräusche meine körperliche Gesundheit schädigen könnten.
6. Wegen der Ohrgeräusche fällt es mir schwer, mich zu entspannen.
7. Oft sind meine Ohrgeräusche so schlimm, dass ich sie nicht ignorieren kann.
8. Wegen der Ohrgeräusche brauche ich länger zum Einschlafen.
9. Wegen der Ohrgeräusche bin ich leichter niedergeschlagen.
10. Ich denke oft darüber nach, ob die Ohrgeräusche jemals weggehen werden.
11. Ich bin Opfer meiner Ohrgeräusche.
12. Die Ohrgeräusche haben meine Konzentration beeinträchtigt.

A.2. Tinnitus Sample Case History Questionnaire (TSCHQ)

Tinnitus ist ein subjektives Symptom, welches für Forscher nur sehr schwer zu messen oder zu bewerten ist. Um einen einheitlichen Standard zu schaffen, wurde der Tinnitus Sample Case History Questionnaire entwickelt [Lan06]. Tabelle A.1 zeigt alle Fragen mit den entsprechenden Antwortmöglichkeiten und dem Antworttyp (siehe Kapitel 4.1.4).

Nr.	**Frage**	**Antwortmöglichkeiten**	**Antworttyp**
1.	Geburtsdatum	-	Date Picker
2.	Geschlecht	männlich; weiblich	Radio Buttons
3.	Händigkeit	rechts; links; beidseitig	Radio Buttons
4.	Tinnitus-Beschwerden in der Familie	Ja; Nein; Wenn Ja: Eltern; Geschwister; Kinder	Checkboxes
5.	Beginn des Tinnitus: Wann haben Sie den Tinnitus zum ersten Mal wahrgenommen?	-	Text
6.	Wie haben Sie den Beginn wahrgenommen?	allmählich; unvermittelt	Radio Buttons
7.	Stand der Beginn Ihres Tinnitus in Verbindung mit:	Knalltrauma; Verletzung der Halswirbelsäule; Veränderung des Hörvermögens; Stress; Kopfverletzung; Sonstiges	Radio Buttons
8.	Haben Sie das Gefühl, dass Ihr Tinnitus pulsiert?	Ja, im Rhythmus meines Herzschlags; Ja, anders als mein Herzschlag; Nein	Radio Buttons
9.	Wo nehmen Sie Ihren Tinnitus wahr?	im rechten Ohr; im linken Ohr; in beiden Ohren, stärker im linken Ohr; in beiden Ohren, stärker im rechten Ohr; in beiden Ohren gleich stark, im Inneren des Kopfes; an anderer Stelle	Radio Buttons
10.	Wie ist der Verlauf?	Phasen mit und ohne Tinnitus wechseln sich ab; Der Tinnitus ist ständig vorhanden	Radio Buttons
11.	Verändert sich die LAUTSTÄRKE Ihres Tinnitus von Tag zu Tag?	Ja; Nein	Radio Buttons
12.	Bitte beschreiben Sie die durchschnittliche LAUTSTÄRKE Ihres Tinnitus.	-	Slider
13.	Bitte beschreiben Sie mit eigenen Worten, wie Ihr Tinnitus normaler Weise klingt:	-	Text

Nr.	Frage	Antwortmöglichkeiten	Antworttyp
14.	Hört sich Ihr Tinnitus eher wie ein Ton an oder eher wie Lärm?	Ton; Lärm; Grillen; andere Empfindung	Radio Buttons
15.	Bitte beschreiben Sie die Frequenz Ihres Tinnitus.	sehr hohe Frequenz; hohe Frequenz; mittlere Frequenz; niedrige Frequenz	Radio Buttons
16.	Wieviel Prozent der Zeit waren Sie sich im letzten Monat Ihres Tinnitus bewußt?	-	Slider
17.	Wieviel Prozent der Zeit im letzten Monat haben Sie sich über Ihren Tinnitus geärgert, bzw. waren Sie wegen des Tinnitus unglücklich oder genervt?	-	Slider
18.	Wie vielen verschiedenen Behandlungen haben Sie sich unterzogen aufgrund Ihres Tinnitus?	keiner; einer; mehreren; vielen	Radio Buttons
19.	Wird die Lautstärke Ihres Tinnitus durch bestimmte Arten von Umgebungsgeräuschen reduziert bzw. überdeckt, wie zum Beispiel durch das Rauschen eines Wasserfalls oder das Geräusch fließenden Wassers, wenn Sie unter der Dusche stehen?	Ja; Nein; Ich weiß es nicht.	Radio Buttons
20.	Verschlechtert starker Lärm Ihren Tinnitus?	Ja; Nein; Ich weiß es nicht.	Radio Buttons
21.	Beeinflusst eine Bewegung Ihres Kopfes und/oder Ihres Nackens (zum Beispiel das Vorschieben des Kiefers oder das Zusammenbeißen der Zähne) oder eine Berührung Ihrer Arme, Hände oder Ihres Kopfes Ihren Tinnitus?	Ja; Nein	Radio Buttons
22.	Beeinflusst ein kurzer Schlaf während des Tages (z.B. Mittagsschlaf) Ihren Tinnitus?	verstärkt meinen Tinnitus; vermindert meinen Tinnitus; hat keine Auswirkung	Radio Buttons
23.	Besteht eine Verbindung zwischen Ihrem Nachtschlaf und Ihrem Tinnitus während des Tages?	Ja; Nein; Ich weiß es nicht.	Radio Buttons
24.	Hat Stress Einfluss auf Ihren Tinnitus?	verstärkt meinen Tinnitus; vermindert meinen Tinnitus; hat keine Auswirkung	Radio Buttons

Nr.	Frage	Antwortmöglichkeiten	Antworttyp
25.	Beeinflusst die Behandlung mit Medikamenten Ihren Tinnitus?	-	Text
26.	Haben Sie ein Problem mit Ihrem Hörvermögen?	Ja; Nein; Ich weiß es nicht.	Radio Buttons
27.	Benützen Sie Hörgeräte?	rechts; links; auf beiden Seiten; auf keiner Seite	Radio Buttons
28.	Fühlen Sie sich besonders geräuschempfindlich? Fühlen Sie sich beispielsweise gestört durch Geräusche, die anderen Menschen in Ihrer Umgebung nicht störend laut vorkommen?	niemals; selten; manchmal; gewöhnlich; immer	Radio Buttons
29.	Führen laute Geräusche bei Ihnen zu Schmerz-ähnlichem Empfinden oder zu körperlichem Unwohlsein?	Ja; Nein; Ich weiß es nicht.	Radio Buttons
30.	Leiden Sie unter Kopfschmerzen?	Ja; Nein	Radio Buttons
31.	Leiden Sie unter Schwindel?	Ja; Nein	Radio Buttons
32.	Haben Sie Beschwerden im Bereich Ihres Kiefergelenkes oder Ihrer Kaumuskulatur?	Ja; Nein	Radio Buttons
33.	Leiden Sie unter Nackenschmerzen?	Ja; Nein	Radio Buttons
34.	Leiden Sie unter anderen Schmerzen?	Ja; Nein	Radio Buttons
35.	Befinden Sie Sich momentan in psychiatrischer Behandlung?	Ja; Nein	Radio Buttons

Tabelle A.1.: Tinnitus Sample Case History Questionnaire

A.3. Schlimmstes Symptom

Es gibt eine Reihe von Symptomen (zum Beispiel: Schlaflosigkeit, Konzentrationsschwierigkeiten, Angst, der Tinnitus könnte schlimmer werden), die von vielen Patienten berichtet werden. Davon hat jedoch fast jeder Patient ein Symptom, das für ihn oder sie am schlimmsten ist. Dieser Fragebogen fragt nach diesem Symptom und hat folgende Antwortmöglichkeiten, von denen nur eine ausgewählt werden kann [Sch13b]:

- Aufgrund des Tinnitus bin ich mit meiner Familie, Freunde und Arbeitskollegen gereizter.
- Wegen des Tinnitus fällt es mir schwer mich zu entspannten.
- Ich mache mir starke Sorgen wegen meines Tinnitus.
- Wegen des Tinnitus fällt es mir schwer abends einzuschlafen.
- Aufgrund des Tinnitus kann ich mich schlechter konzentrieren.

- Wegen des Tinnitus bin ich geräuschempfindlicher als früher.
- Mir fällt es wegen des Tinnitus schwer Gesprächen, Musik oder Filmen zuzuhören.
- Wegen des Tinnitus fühle ich mich niedergeschlagen.
- Ich habe keines dieser Symptome.

Literaturverzeichnis

[App10] Apple Inc. UILocalNotification Class Reference. https://developer.apple.com/library/ios/documentation/iPhone/Reference/UILocalNotification_Class/Reference/Reference.html, Mai 2010. [Online; zuletzt besucht am 04. Februar 2014].

[App11] Apple Inc. UIControl Class Reference. https://developer.apple.com/library/IOs/documentation/UIKit/Reference/UIControl_Class/Reference/Reference.html, September 2011. [Online; zuletzt besucht am 30. Januar 2014].

[App12a] Apple Inc. CALayer Class Reference. https://developer.apple.com/library/mac/documentation/graphicsimaging/reference/CALayer_class/Introduction/Introduction.html, Dezember 2012. [Online; zuletzt besucht am 23. Februar 2014].

[App12b] Apple Inc. CATransaction Reference. https://developer.apple.com/library/mac/documentation/GraphicsImaging/Reference/CATransaction_class/Introduction/Introduction.html, Dezember 2012. [Online; zuletzt besucht am 23. Februar 2014].

[App13a] Apple Inc. Apple Push Notification Service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html, September 2013. [Online; zuletzt besucht am 04. Februar 2014].

[App13b] Apple Inc. CGContext Reference. https://developer.apple.com/library/mac/documentation/graphicsimaging/reference/CGContext/Reference/reference.html, September 2013. [Online; zuletzt besucht am 23. Februar 2014].

[App13c] Apple Inc. Date Picker. https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/UIDatePicker.html, Dezember 2013. [Online; zuletzt besucht am 19. Januar 2014].

[App13d] Apple Inc. NSManagedObject Class Reference. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSManagedObject_Class/Reference/NSManagedObject.html, Oktober 2013. [Online; zuletzt besucht am 30. Januar 2014].

[App13e] Apple Inc. NSUserDefaults Class Reference. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults_Class/Reference/Reference.html, Oktober 2013. [Online; zuletzt besucht am 01. Februar 2014].

[App13f] Apple Inc. Quartz 2D Programming Guide. https://developer.

apple.com/library/mac/documentation/graphicsimaging/Conceptual/drawingwithquartz2d/Introduction/Introduction.html, Dezember 2013. [Online; zuletzt besucht am 27. Februar 2014].

[App13g] Apple Inc. Scheduling, Registering, and Handling Notifications - Local and Push Notification Programming Guide. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/IPhoneOSClientImp.html, September 2013. [Online; zuletzt besucht am 04. Februar 2014].

[App13h] Apple Inc. Target-Action Design Pattern. https://developer.apple.com/library/ios/documentation/general/conceptual/Devpedia-CocoaApp/TargetAction.html, September 2013. [Online; zuletzt besucht am 22. Februar 2014].

[App13i] Apple Inc. UIApplicationDelegate Protocol Reference. https://developer.apple.com/library/ios/documentation/uikit/reference/UIApplicationDelegate_Protocol/Reference/Reference.html, September 2013. [Online; zuletzt besucht am 30. Januar 2014].

[App13j] Apple Inc. UIKit Framework Reference. https://developer.apple.com/library/ios/documentation/uikit/reference/UIKit_Framework/Introduction/Introduction.html, September 2013. [Online; zuletzt besucht am 21. Februar 2014].

[App13k] Apple Inc. UIView Class Reference. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/UIView/UIView.html, Dezember 2013. [Online; zuletzt besucht am 24. Februar 2014].

[Bun09] Bundesministerium für Ernährung und Landwirtschaft, Forst. Häufige gesundheitliche Beeinträchtigungen. http://de.statista.com/statistik/daten/studie/6800/umfrage/haeufige-gesundheitliche-beeintraechtigungen/, Januar 2009. [Online; zuletzt besucht am 06. März 2014].

[Car13] Cartalyst LLC. Sentry. https://github.com/cartalyst/sentry/tree/1.1/master, 2013. [Online; zuletzt besucht am 26. Januar 2014].

[Cro13] Crombach, A., Nandi, C., Bambonye, M., Liebrecht, M., Pryss, R., Reichert, M., Elbert, T. and Weierstall, R. (2013). Screening for mental disorders in post-conflict regions using computer apps - a feasibility study from burundi. In *XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conf*, pages 70–70, June 2013.

[Deg13] Degasperi, L. (2013). PHP OAuth 2.0 Server for Laravel. https://github.com/lucadegasperi/oauth2-server-laravel, November 2013. [Online; zuletzt besucht am 26. Januar 2014].

[Fac] Facebook Inc. Einführung der Chronik. https://www.facebook.com/about/timeline. [Online; zuletzt besucht am 19. Januar 2014].

[Fie00] Fielding, R. T. (2000). Representational State Transfer (REST). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. [Online; zuletzt besucht am 23. Januar 2014].

[Gei13] Geiger, P., Pryss, R., Schickler, M. and Reichert, M. (2013). Engineering an advanced location-based augmented reality engine for smart mobile devices. Technical Report UIB-2013-09, University of Ulm, Ulm, October 2013.

[Geion] Geiger, P., Schickler, M., Pryss, R., Schobel, J. and Reichert, M. (2014). Location-based mobile augmented reality applications: Challenges, examples, lessons learned. In *10th Int'l Conf on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, April 2014 (accepted for publication).

[Gooa] Google Inc. Adapter Class Reference. `http://developer.android.com/reference/android/widget/Adapter.html`. [Online; zuletzt besucht am 31. Januar 2014].

[Goob] Google Inc. AlarmManager Class Reference. `http://developer.android.com/reference/android/app/AlarmManager.html`. [Online; zuletzt besucht am 01. Februar 2014].

[Gooc] Google Inc. BroadcastReceiver Class Reference. `http://developer.android.com/reference/android/content/BroadcastReceiver.html`. [Online; zuletzt besucht am 04. Februar 2014].

[Good] Google Inc. Custom Components Guide. `http://developer.android.com/guide/topics/ui/custom-components.html`. [Online; zuletzt besucht am 24. Februar 2014].

[Gooe] Google Inc. Custom Drawing. `http://developer.android.com/reference/android/view/MotionEvent.html`. [Online; zuletzt besucht am 27. Februar 2014].

[Goof] Google Inc. Google Cloud Messaging for Android. `http://developer.android.com/google/gcm/index.html`. [Online; zuletzt besucht am 04. Februar 2014].

[Goog] Google Inc. MotionEvent Class Reference. `http://developer.android.com/reference/android/view/MotionEvent.html`. [Online; zuletzt besucht am 24. Februar 2014].

[Gooh] Google Inc. NotificationCompat.Builder Class Reference. `http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html`. [Online; zuletzt besucht am 07. Februar 2014].

[Gooi] Google Inc. NotificationManager Class Reference. `http://developer.android.com/reference/android/app/NotificationManager.html`. [Online; zuletzt besucht am 04. Februar 2014].

[Gooj] Google Inc. Notifications Guide. `http://developer.android.com/guide/topics/ui/notifiers/notifications.html`. [Online; zuletzt besucht am 04. Februar 2014].

[Gook] Google Inc. Paint Class Reference. `http://developer.android.com/reference/android/graphics/Paint.html`. [Online; zuletzt besucht am 24. Februar 2014].

[Gool] Google Inc. Pickers. `http://developer.android.com/guide/topics/ui/controls/pickers.html`. [Online; zuletzt besucht am 19. Januar 2014].

[Goom] Google Inc. SeekBar Class Reference. `http://developer.android.com/reference/android/widget/SeekBar.html`. [Online; zuletzt besucht am 01. Februar 2014].

[Goon] Google Inc. Service Class Reference. `http://developer.android.com/reference/android/app/Service.html`. [Online; zuletzt besucht am 01. Februar 2014].

[Gooo] Google Inc. SQLiteOpenHelper Class Reference. `http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html`. [Online; zuletzt besucht am 01. Februar 2014].

[Goop] Google Inc. Support Library for Android. `http://developer.android.com/tools/support-library/index.html`. [Online; zuletzt besucht am 07. Februar 2014].

[Gooq] Google Inc. View Class Reference. `http://developer.android.com/reference/android/view/View.html`. [Online; zuletzt besucht am 01. Februar 2014].

[Goor] Google Inc. ViewMeasure.Spec Reference. `http://developer.android.com/reference/android/view/View.MeasureSpec.html`. [Online; zuletzt besucht am 01. März 2014].

[Har12] Hardt, D. (2012). OAuth2 Authorization Framework. `http://tools.ietf.org/html/rfc6749`, Oktober 2012. [Online; zuletzt besucht am 16. Januar 2014].

[Hen12] Henry, J. A, Galvez, G, Turbin, M. B., Thielman, E. J., McMillan, G. P. and Istvan, J. A. (2012). Pilot Study to Evaluate Ecological Momentary Assessment of Tinnitus. Ear and Hearing, 2012. 33(2), 179–290. doi:10.1097/AUD.0b013e31822f6740.

[Hil04] Hiller, W. and Goebel, G. (2004). Mini Tinnitus Fragebogen (Mini TF12). `http://www.eutinnitus.com/Dateien%20MiniTF12/MiTiTe_de.php`, 2004. [Online; zuletzt besucht am 19. Januar 2014].

[Ins09] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin (2009). Self-Assessment Manikin (SAM). `http://www.qu.tu-berlin.de/menue/forschung/laufende_projekte/joyofuse/joy_of_use/joy_of_use/measurement_methods/sam/`, Juni 2009. [Online; zuletzt besucht am 19. Januar 2014].

[Int11] International Information Centre for Terminology - Infoterm. ISO 639 Language Codes. `http://www.infoterm.info/standardization/iso_639_1_2002.php`, Oktober 2011. [Online; zuletzt besucht am 25. Januar 2014].

[Irt08] Irtel, H. (2008). Self-Assessment Manikin (SAM). `http://irtel.uni-mannheim.de/pxlab/demos/index_SAM.html`, 2008. [Online; zuletzt besucht am 19. Januar 2014].

[Ise13] Isele, D., Ruf-Leuschner, M., Pryss, R., Schauer, M., Reichert, M., Schobel, J., Schindler, A. and Elbert, T. (2013). Detecting adverse childhood experiences with a little help from tablet computers. In *XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conf*, pages 69–70, June 2013.

[Kah11] Kahnemann, D. (2011). *Thinking, fast and slow.* Allen Lane Paperback, 2011. ISBN 978-1-846-14606-0, Kapitel 11: Anchors, S. 119-128.

[Lan06] Langguth, B., Goodey, R., Azevedo A. et al. (2006). Consensus for Tinnitus Patient Assessment and Treatment Outcome Measurement (Tinnitus Research Initiative Meeting. Regensburg. July 2006). Progress in Brain Research, 2007, Vol. 166: Appendix. `http://www.tinnitusresearch.org/en/consensus/consensusdocuments/de/Case_History_Questionnaire_de.pdf`, Juli 2006. Übersetzt von Berthold Langguth. [Online; zuletzt besucht am 19. Januar 2014].

[Lan07] Langguth, B., Hajak, G., Kleinjung, T., Cacace, A., Moller, A. (2007). *Tinnitus: Pathophysiology and Treatment.* Elsevier Science Ltd., Mai 2007. ISBN 978-0444531674, Preface, S. XVII, XVIII.

[Lar83] Larson, R., Csikszentmihalyi, M. (1983). *New Directions for Methodology of Social and Behavioral Science*. Jossey-Bass, 1983. "The experience sampling method.", S. 41-56.

[Mol07] Moller, A. (2007). *Tinnitus: presence and future*. Elsevier Science Ltd., Mai 2007. In Tinnitus: Pathophysiology and Treatment; ISBN 978-0444531674, Chapter 1, S. 3ff.

[Otwa] Otwell, T. Laravel 3 Dokumentation. `http://three.laravel.com/docs`. [Online; zuletzt besucht am 23. Januar 2014].

[Otwb] Otwell, T. Laravel 3 Dokumentation - Controllers. `http://three.laravel.com/docs/controllers`. [Online; zuletzt besucht am 01. März 2014].

[Otwc] Otwell, T. Laravel 3 Dokumentation - Eloquent ORM. `http://three.laravel.com/docs/database/eloquent`. [Online; zuletzt besucht am 01. März 2014].

[Pry10a] Pryss, R., Tiedeken J. and Reichert, M. (2010). Managing processes on mobile devices: The MARPLE approach. In *CAiSE'10 Demos*, June 2010.

[Pry10b] Pryss, R., Tiedeken, J., Kreher, U. and Reichert, M. (2010). Towards flexible process support on mobile devices. In *Proc. CAiSE'10 Forum - Information Systems Evolution*, number 72 in LNBIP, pages 150–165. Springer, 2010.

[Pry12] Pryss, R., Langer, D., Reichert, M. and Hallerbach, A. (2012). Mobile task management for medical ward rounds - the MEDo approach. In *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, number 132 in LNBIP, pages 43–54. Springer, September 2012.

[Pry13] Pryss, R., Musiol, S. and Reichert, M. (2013). Collaboration support through mobile processes and entailment constraints. In *9th IEEE Int'l Conf on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'13)*. IEEE Computer Society Press, October 2013.

[Pry14] Pryss, R., Mundbrod, N., Langer, D. and Reichert, M. (2014). *Supporting medical ward rounds through mobile task and process management*. Springer Berlin Heidelberg, 2014.

[Rei12] Reichert, Manfred and Weber, Barbara. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, Berlin-Heidelberg, 2012.

[Rob11] Robecke, A., Pryss, R. and Reichert, M. (2011). Dbischolar: An iphone application for performing citation analyses. In *CAiSE Forum-2011*, number Vol-73 in Proceedings of the CAiSE'11 Forum at the 23rd Int'l Conf on Advanced Information Systems Engineering. CEUR Workshop Proceedings, June 2011.

[Ruf13] Ruf-Leuschner, M., Pryss, R., Liebrecht, M., Schobel, J., Spyridou, A., Reichert, M. and Schauer, M. (2013). Preventing further trauma: KINDEX mum screen - assessing and reacting towards psychosocial risk factors in pregnant women with the help of smartphone technologies. In *XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conf*, pages 70–70, June 2013.

[Sch13a] Schlee, W. (2013). Track Your Tinnitus. [Unveröffentlicht, Idee und Name des Projekts], März 2013.

[Sch13b] Schlee, W. (2013). Worst Symptom Questionnaire. [Unveröffentlich, per E-Mail], September 2013.

[Sch13c] Schlee, W. (2013). Über das Track Your Tinnitus Projekt. `https://secure.dbis.info/da/tinnitus/de/about`, November 2013. [Online; zuletzt besucht am 12. März 2014].

[Sch13d] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H. and Reichert, M. (2013). Using vital sensors in mobile healthcare business applications: Challenges, examples, lessons learned. In *9th Int'l Conf on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, pages 509–518, May 2013.

[Schon] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M. (2014). Towards process-driven mobile data collection applications: Requirements, challenges, lessons learned. In *10th Int'l Conf on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, Aachen, Germany, 2014 (accepted for publication).

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 19. März 2013 Jochen Herrmann