ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Konzeption und Realisierung eines Mobile Serious Game für akustische Lokalisation

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Martin Randelzofer
martin.randelzofer@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2017

Fassung 30. Juli 2017

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Die räumliche Zuordnung von Geräuschen ist eine wichtige Fähigkeit des menschlichen Hörsystems. Vor allem ältere Personen leiden häufig an Krankheiten, die diese Fähigkeit einschränken. Tinnitus, ein mögliches Symptom solcher Krankheiten, ist in Industriestaaten weit verbreitet. Durch die älter werdende Bevölkerung wird es immer wichtiger, neue Methoden für die Behandlung zu finden. Da innerhalb der letzten Jahre ein enormer Anstieg der Nutzung von mobilen Geräten stattfand, erforscht die Medizin, welche neuen Möglichkeiten durch diese Entwicklung entstehen.
Ziel dieser Arbeit ist die Realisierung einer mobilen Applikation, mit der die akustische Lokalisationsfähigkeit von Menschen getestet werden kann. Die dafür zu erstellende Applikation besteht aus drei verschiedenen Modi, die dreidimensionale Töne simulieren. Dadurch soll die Fähigkeit des Richtungshörens in unterschiedlichen Situationen getestet werden können.
Nach einer Erklärung des Symptoms Tinnitus erfolgt die Vorstellung eingesetzter Technologien (Sensoren, HRTF) und Frameworks (Web Audio API, Ionic 2, Angular, Cordova). Daraufhin wird die zu entwickelnde Applikation mit ihrer Benutzeroberfläche konzipiert und anschließend deren Umsetzung in Ionic 2 dokumentiert.

Inhaltsverzeichnis

1

Einleitung

Die auditive Wahrnehmung ist ein wichtiges Werkzeug, um sich im alltäglichen Leben zurechtzufinden. Im Gegensatz zu anderen Sinnen ist das Hören omnipräsent und lässt sich kaum bewusst unterdrücken. Eine wichtige Aufgabe unseres Hörsystems ist die räumliche Lokalisation von Geräuschen, wodurch wir beispielsweise frühzeitig Gefahren wie vorbeifahrende Autos erkennen. Einschränkungen dieser Fähigkeit würden in vielen Bereichen des Lebens fatale Auswirkungen haben.
Ein Symptom solcher Probleme stellt der Tinnitus dar, welcher in Deutschland weit verbreitet ist. Vor allem bei älteren Menschen tritt er häufiger auf und führt unter Umständen zu einem verminderten Richtungshören [1]. Abbildung 1.1 veranschaulicht die Anzahl der an Tinnitus Leidenden in Deutschland.

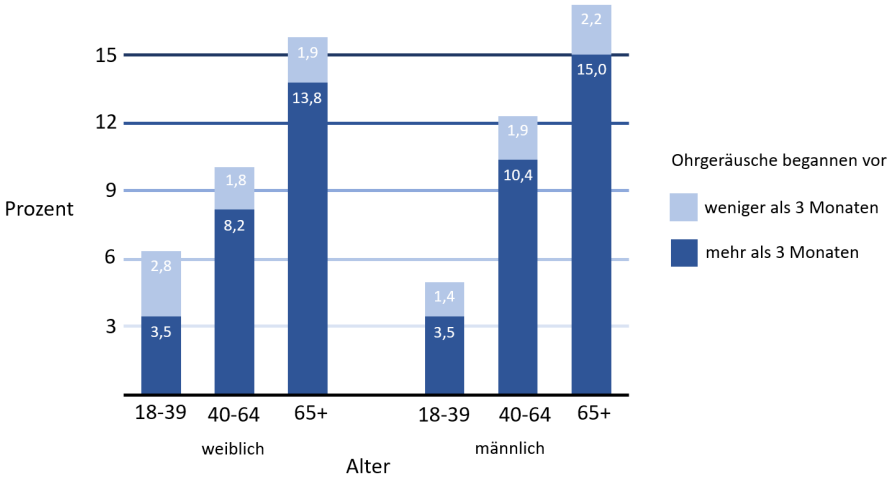

Abbildung 1.1: Anzahl an Tinnitus Leidenden [1]

In anderen Industriestaaten ist ein ähnlicher Trend festzustellen. Zudem ist durch die älter werdende Bevölkerung auch mit einem Anstieg der Leidenden zu rechnen [1]. Es wird folglich immer wichtiger, erfolgreiche Behandlungsmethoden zu finden.
Die Medizin erforscht daher wie viele andere Geschäftsbereiche die Einsatzmöglichkeiten neuer Technologien. Durch die in den letzten Jahren stark steigende Verbreitung mobiler Geräte, wie z.B. Smartphones ist das Interesse an mobilen Anwendungen gewachsen. Durch die immer größer werdende Nachfrage an mobilen Applikationen ist ein neuer Wirtschaftszweig der IT entstanden. Die Entwicklung mobiler Anwendungen ist aufwändig, da mehrere Plattformen unterstützt werden müssen, um die größtmögliche Anzahl an Benutzern zu erreichen. Da für jede Plattform die Anwendung separat entwickelt werden muss, steigen die Entwicklungskosten enorm. Vor allem für Auftraggeber mit wenig Budget stellt dies oft ein Problem dar. Aus diesem Grund sind in den letzten Jahren mehrere plattformübergreifende Entwicklungsansätze entstanden, die Dank des geringeren Aufwands für die Unterstützung mehrerer Plattformen große Beliebtheit erlangten.

1.1 Problemstellung

Die in Abbildung 1.1 veranschaulichte weite Verbreitung von Tinnitus verdeutlicht die Erfordernis neuartiger Untersuchungs- und Behandlungsmethoden für das Symptom. Deshalb werden auch in der Medizin Methoden untersucht, mit denen Patienten und Ärzte Vorteile aus der aufstrebenden Entwicklung mit mobilen Applikationen ziehen können. Dabei spielt die Reduzierung des Aufwands eine wichtige Rolle. Einerseits sollte der Entwicklungsaufwand minimiert werden, wozu plattformunabhängige Entwicklungsansätze beitragen würden. Andererseits soll der Behandlungsaufwands des Arztes vermindert werden [2]. Dies kann beispielsweise erreicht werden, indem spezielle Aufgaben in Abwesenheit des Arztes durch die Applikation angeleitet werden [3] [4]. Dabei muss immer darauf geachtet werden, dass beim Patienten die nötige Motivation geschaffen wird, was beispielsweise durch eine individuelle Anpassung der zu lokalisierenden Geräusche erreicht werden kann.

1.2 Zielsetzung

In dieser Arbeit soll eine mobile Applikation für akustische Lokalisation mithilfe des plattformübergreifenden Frameworks Ionic 2 konzipiert und entwickelt werden. Mit dieser Anwendung soll untersucht werden können, wie stark Personen im räumlichen Hören eingeschränkt sind und ob sie ihr Richtungshören mit wiederholtem Training verbessern können. Ebenfalls soll untersucht werden können, ob durch die Anwendung eine Linderung des Leidens der Tinnituspatienten erreicht werden kann. Um die Untersuchungen und Trainingseinheiten abwechslungsreicher zu gestalten und damit für eine höhere Motivation der Patienten zu sorgen, wird die Applikation spielerisch aufgebaut. Dazu werden drei verschiedene Spielmodi entwickelt, die jeweils andere Anforderungen an die Hörfähigkeiten des Nutzers stellen.

1.3 Struktur der Arbeit

Zunächst wird auf das Symptom Tinnitus sowie dessen Ursachen und Behandlungsmöglichkeiten eingegangen. Das dritte Kapitel befasst sich mit Grundlagen des räumlichen Hörens sowie mit den für diese Arbeit relevanten technischen Aspekten. Darunter fallen Einführungen in die verwendeten Frameworks *Web Audio API* und *Ionic 2*. Anschließend werden die funktionalen- und nichtfunktionalen Anforderungen an die Applikation definiert. Im fünften Kapitel wird das Konzept der Anwendung erläutert. Es erfolgt eine Erklärung der Funktionsweise und der verschiedenen Spielmodi. Zudem werden Mockups aufgezeigt und das Datenmodell vorgestellt. Daraufhin befasst sich das sechste Kapitel mit der Realisierung der Applikation. Hinterher erfolgt eine Evaluation, worin die Anwendung gegen die in Kapitel vier gestellten Anforderungen abgeglichen wird. Abschließend wird ein Fazit gezogen, wobei mögliche Erweiterungen der Anwendung im Vordergrund stehen.

2

Tinnitus

In diesem Kapitel werden grundlegende Aspekte des Symptoms Tinnitus vorgestellt. Insbesondere wird auf die Einteilung unterschiedlicher Ausprägungen des Symptoms, Ursachen sowie Behandlungsmöglichkeiten eingegangen.

2.1 Definition

Der medizinische Begriff *Tinnitus aurium* (lat.: „das Klingeln der Ohren“) bezeichnet ein Geräusch, welches zusätzlich zu den akustischen Reizen der Umgebung wahrgenommen wird [5]. Die daran Leidenden beschreiben dieses Geräusch oft als Pfeifen, Summen oder Knacken. Aus medizinischer Sicht wird Tinnitus nicht als eigenständiges Krankheitsbild, sondern als eine Begleiterscheinung anderer Krankheiten gesehen und fällt somit unter die *Symptome* [6].

2.2 Klassifikation

Ausprägungen von Tinnitus können wie im Folgenden zu sehen vielfältig sein. Vor allem für therapeutische Ansätze ist es daher wichtig, das Symptom zu klassifizieren, um eine effektive Behandlungen (siehe Abschnitt 2.3) vorzunehmen [7]. Die nachfolgenden Abschnitte geben einen kurzen Überblick über Kategorien, mit welchen eine solche Klassifikation ermöglicht wird.

2.2.1 Subjektiver Tinnitus

Ein subjektiver Tinnitus herrscht dann vor, wenn die vom Patienten wahrgenommenen Töne nicht aufgrund eines akustischen Reizes entstehen. Der Grund für die Entstehung der Töne wird durch Fehler des auditorischen Systems, welches für das Hören verantwortliche neuronale und rezeptive Strukturen zusammenfasst, hervorgerufen. Diese Art von Tinnitus ist viel häufiger vorzufinden als ein objektiver Tinnitus [8].

2.2.2 Objektiver Tinnitus

Bei einem objektiven Tinnitus entstehen die wahrgenommenen Töne durch körpereigene Störgeräusche [9]. Beispielsweise können Fehlbildungen von Blutgefäßen oder Muskelzuckungen in Nähe des Ohres solche Geräusche produzieren. Im Gegensatz zum subjektiven Tinnitus können daher die Geräusche bei Untersuchungen erfasst werden [8].

2.2.3 Weitere Klassifikationsaspekte

Neben der grundlegenden Einteilung in subjektiven und objektiven Tinnitus lässt sich das Symptom weiter nach seiner Dauer und Schwere einteilen.

Dauer

Bei der Dauer ist zu bestimmen, wie lange der Patient schon an Tinnitus leidet. Sind seit Beginn des Symptoms weniger als drei Monate vergangen spricht man von akutem Tinnitus, im anderen Fall von chronischem Tinnitus [1].

Schwere

Zudem kann der Tinnitus nach seinem Schweregrad eingeteilt werden. Hierbei wird eine vierstufige Skala verwendet.

Unter den ersten Grad fallen Patienten, welche durch ihren Tinnitus keine Beeinträchtigung im Alltag haben. Der zweite Grad herrscht dann vor, wenn der Patient den Tinnitus nur in speziellen Situationen wahrnimmt, z.B. unter bestimmten Belastungen, Stress oder absoluter Stille. Von einem dritten Grad ist dann die Rede, wenn der Patient durch seinen Tinnitus in allen Umfelden beeinträchtigt wird. Sobald der Tinnitus den Patienten so stark belastet, dass er privaten Tätigkeiten nicht mehr nachkommen kann oder berufsunfähig wird, ist der Tinnitus unter dem Schwergrad vier einzuordnen.
Grad eins und zwei werden als *kompensierter Tinnitus*, Grad drei und vier als *dekompensierter Tinnitus* zusammengefasst [1].

2.3 Ursachen und Behandlungsmöglichkeiten

Die folgenden Abschnitte befassen sich mit einer Auswahl an Ursachen von Tinnitus und deren Behandlungsansätze. Da die Ursachen, wie in den nächsten Absätzen zu sehen, vielfältig sind, ist es für eine erfolgreiche Behandlung ausschlaggebend, wie exakt die Diagnose ausfällt [10].

2.3.1 Subjektiver Tinnitus

Da sich die Ursachen und Behandlungen bei subjektivem und objektivem Tinnitus grundlegend unterscheiden, ist es sinnvoll diese auch getrennt zu betrachten. Zuerst werden die genannten Aspekte anhand des subjektiven Tinnitus beleuchtet.

Ursachen

Viele Ursachen des subjektiven Tinnitus stammen aus der *Otologie*, dem Gebiet der Medizin, das sich unter anderem mit Erkrankungen des Ohres befasst. Darunter fallen zum Beispiel Schall- und Lärmtraumata, welche aufgrund kurzzeitiger, übermäßig hoher Schalleinwirkung auf das Ohr entstehen [1].

Krankheiten des Nervensystems können ebenfalls ein Grund für Tinnitus sein. Zu den sogenannten *neurologischen* Ursachen zählen beispielsweise Multiple Sklerose, Schädel-Hirn-Traumata und unterschiedlichste Verletzungen bzw. Erkrankungen der Halswirbelsäule [8].
Hirnhautentzündungen, Mittelohrentzündungen und generell Infektionskrankheiten des Ohres können ebenfalls das Symptom Tinnitus aufweisen. Diese Krankheiten lassen sich unter den *infektiösen* Ursachen zusammenfassen [8].

Behandlungsmöglichkeiten

Bei der Behandlung des subjektiven Tinnitus kann auf verschiedene Methoden zurückgegriffen werden, die je nach Patient und Diagnose ausgewählt werden müssen. Bei den meisten davon ist die Wirkung allerdings nicht ausreichend nachgewiesen. Folglich ist bisher keine Therapie bekannt, die zuverlässig eine Verbesserung des Tinnitus herbeiführen kann [9].
Ein möglicher Ansatz ist die *Kognitive Verhaltenstherapie*. Hierbei wird dem Patient geholfen Muster, die das Erleben des Tinnitus beeinflussen, zu erkennen und diese zu verändern [9]. Eine weitere Möglichkeit besteht in der Stimulation des auditorischen Systems. Beispielsweise kann durch ein dauerhaftes Hintergrundgeräusch versucht werden, eine Maskierung des Tinnitus zu erreichen. Zudem können auf den Patienten angepasste Frequenzen eine Veränderung im zentralen Nervensystem hervorrufen, wodurch eine Hemmung des Tinnitus erzielt werden kann [11]. Neuere Methoden finden sich in den sogenannten *neuromodulatorischen* Therapieansätzen. Darunter fallen z.B. Methoden, die durch magnetische Impulse Gehirnaktivitäten modulieren [9].

2.3.2 Objektiver Tinnitus

Zuletzt werden die Ursachen und Behandlungsmöglichkeiten bei einem objektivem Tinnitus vorgestellt.

Ursachen

Wie in Abschnitt 2.2.2 angedeutet, ist ein häufiger Grund des objektiven Tinnitus eine Fehlbildung (z.B. Verengung) von Blut- oder Lymphgefäßen. Erkrankung der Herzklappen, Bluthochdruck sowie Spasmen der Muskeln in direkter Umgebung des Ohres, beispielsweise die Mittelohrmuskeln, können ebenfalls zu objektivem Tinnitus führen [8].

Behandlungsmöglichkeiten

Liegt der Grund des Tinnitus in Verengungen von Gefäßen, werden diese meist operativ beseitigt. Falls ein solcher Eingriff nicht möglich ist, kann eine Implantation einer Gefäßstütze Abhilfe schaffen. Bei Spasmen der Muskeln wird versucht durch die Einnahme von Medikamenten eine Reduzierung der Zuckungen herbeizuführen. Sollte dies nicht helfen, muss ebenfalls operiert werden. Auch Bluthochdruck wird mit Medikamenten behandelt, sofern Versuche, ihn natürlich zu senken, nicht zur Verbesserung beitragen. Es ist hierbei erkennbar, dass eine sehr spezifische Behandlung möglich ist, da ein objektiver Tinnitus eine eindeutige Ursache hat. Ferner ist es für die Behandlung vorteilhaft, dass das Störgeräusch auch tatsächlich gemessen werden kann, wodurch sich beispielsweise der Leidensdruck des Patienten besser einschätzen lässt und somit eine individuellere Behandlung möglich ist [12].

3

Grundlagen

Dieses Kapitel befasst sich mit Grundlagen, die für das Verständnis der im Rahmen dieser Arbeit behandelten Thematik nötig sind. Zuerst werden Aspekte vorgestellt, mit denen eine akustische Lokalisation von Geräuschen durch das menschliche Hörsystem erreicht wird. Anschließend verschiebt sich der Fokus auf die technische Seite. So wird auf Sensoren der Lagebestimmung in mobilen Geräten eingegangen sowie ein Überblick über ein verwendetes Framework, die *Web Audio API* gegeben. Abschließend werden Grundlagen zu Ionic 2 besprochen, dem Framework, mit welchem die mobile hybride Anwendung dieser Arbeit entwickelt wird.

3.1 Akustische Lokalisation

Wie der Begriff *akustische Lokalisation* vermuten lässt, handelt es sich hierbei um die **räumliche Zuordnung akustischer Informationen**. Diese wird anhand mehrerer Faktoren, auf die im Folgenden genauer eingegangen wird, durch das menschliche Hörsystem bewerkstelligt [13].
Zuerst werden die zwei wichtigsten Faktoren, die *interaurale Zeitdifferenz* und *interaurale Intensitätsdifferenz* vorgestellt. Daraufhin werden zusätzliche Faktoren angeführt, durch welche auch in speziellen Situationen eine möglichst genaue und richtige Ortung der Geräusche ermöglicht wird. Abschließend wird zur technischen Simulation des räumlichen Hörens übergeleitet.

3.1.1 Interaurale Zeitdifferenz

Der Begriff *interaural* bedeutet „zwischen den Ohren“. Folglich beschreiben interaurale Zeitdifferenzen im vorliegenden Kontext der akustischen Lokalisation die zeitlichen Unterschiede der Ankunft des akustischen Signals am jeweiligen Ohr, wie in Abbildung 3.1 veranschaulicht.
Dieser Zeitunterschied ist vor allem für die Lokalisation von Frequenzen unter 1500Hz wichtig, da in niedrigeren Frequenzbereichen der zweite wichtige Faktor, die interaurale Intensitätsdifferenz, auf Probleme stößt (siehe Abschnitt 3.1.2). Bei hohen Frequenzen hingegen, genauer bei Frequenzen mit einer kürzeren Wellenlänge als der Ohrabstand (Kopfdurchmesser) des Hörers, ist eine genaue interaurale Zeitdifferenz für den Menschen nicht erkennbar und somit dieser Faktor für die Lokalisation nicht ausschlaggebend [14]. An dieser Stelle sei noch angemerkt, dass eine interaurale Zeitdifferenz von 11 Mikrosekunden ($\sim 10^{-5}s$) einen Winkelunterschied von einem Grad darstellt [15].

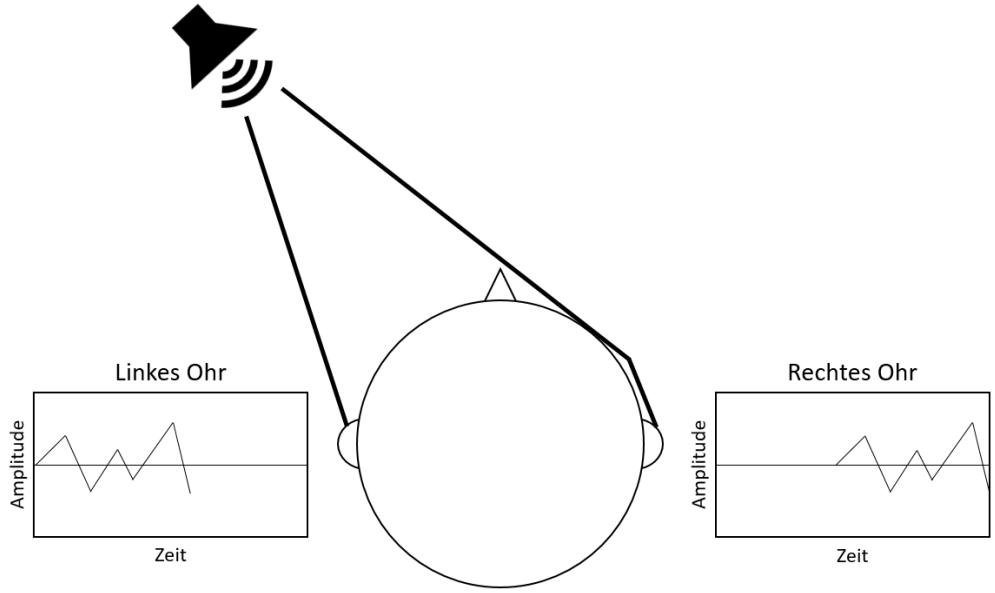

Abbildung 3.1: Interaurale Zeitdifferenz [16]

3.1.2 Interaurale Intensitätsdifferenz

Neben dem Faktor Zeit spielt bei der akustischen Lokalisation die Intensität des Geräusches eine entscheidende Rolle. Durch *interaurale Intensitätsdifferenzen* ist es dem menschlichen Hörsystem möglich, höhere Frequenzen zu lokalisieren. Der Intensitätsunterschied beruht, wie in Abbildung 3.2 veranschaulicht, auf einem sog. *akustischen Schatten*. Dieser entsteht um das von der Schallquelle abgewandte Ohr, aufgrund von Dämpfungen und Reflektionen der Schallwellen durch den Kopf des Hörers. Im Bereich dieses akustischen Schattens ist die Schallintensität geringer, wodurch die Richtung der Schallquelle lokalisiert werden kann.

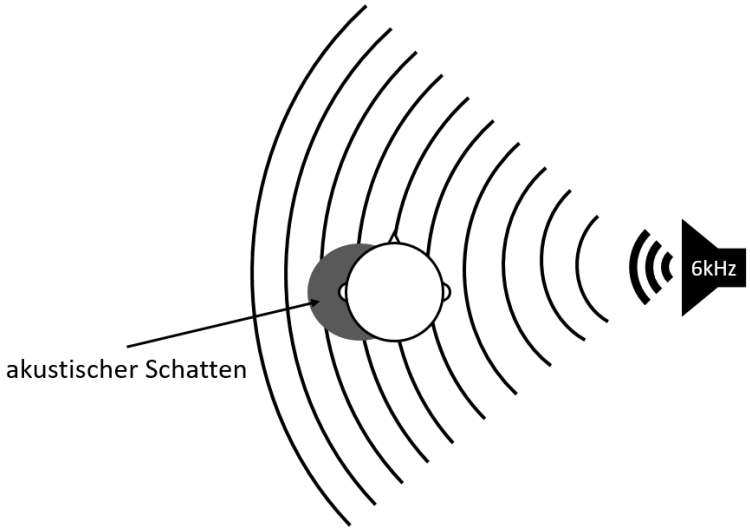

Abbildung 3.2: Interaurale Intensitätsdifferenz [17]

Niedrigere Frequenzen werden weniger reflektiert, wodurch ein solcher Schallschatten bei Frequenzen unter 1500Hz kaum vorhanden ist, und somit der interaurale Intensitätsunterschied nicht zur Lokalisation verwendet werden kann [17].
Dieser Sachverhalt ist in Abbildung 3.3 veranschaulicht. Zudem sind darin die Zusammenhänge zwischen dem Einfallswinkel des Schalls und der daraus resultierenden interauralen Intensitätsdifferenz (in dB) erkennbar.

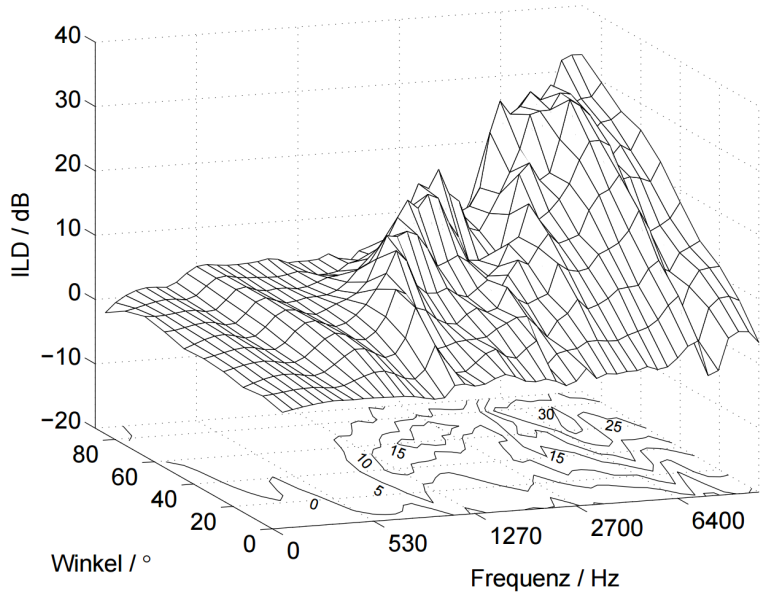

Abbildung 3.3: Interaurale Intensitätsdifferenz [15]

3.1.3 Weitere Faktoren

Nur mithilfe der interauralen Intensitäts- und Zeitdifferenzen lassen sich allerdings noch nicht alle Positionen bestimmen. Sei z.B. Schallquelle A genau vor dem Hörer (Winkel 0 Grad), dann existieren keine Zeit- und Intensitätsdifferenzen. Bei Schallquelle B, welche genau hinter dem Hörer (Winkel 180 Grad) positioniert ist, herrschen allerdings die gleichen Gegebenheiten vor. Des Weiteren bezog sich die Lokalisation bisher lediglich auf den Einfallswinkel des Schalls in der horizontalen Ebene. Für eine dreidimensionale Lokalisation werden zusätzlich die Aspekte Entfernung sowie die Höhe der Schallquelle benötigt.

Die Entfernung lässt sich für das menschliche Gehör am schwierigsten bestimmen. Hauptsächlich ist die Lautstärke des Schalls dafür ausschlaggebend. Wenn das Geräusch bereits bekannt ist, beruht die Bestimmung zusätzlich auf Erfahrungswerten. Bei der horizontalen Lokalisation spielt die Form des Kopfes und der Ohrmuschel eine entscheidende Rolle. Durch diese Form werden je nach Richtung des eintreffenden Geräusches die Intensitäten der Frequenzen verändert, welche im gesamten Spektrum

des Geräusches vorhanden sind. Anhand dieser Veränderungen kann auf die horizontale Position geschlossen werden [18].
Generell ist es durch das Bewegen des Kopfes und der daraus resultierenden Veränderung der Winkel leichter, Schallquellen zu lokalisieren [19]. Damit lässt sich das anfangs angesprochene Problem, ob das Geräusch von vorne oder hinten kommt, leicht bewältigen.

3.1.4 Head Related Transfer Function

Die in Abschnitt 3.1.3 beschriebene Intensitätsänderung von Frequenzen durch die Ohrmuschel ist die Grundlage der *Head Related Transfer Function* (HRTF). Anhand von Messergebnissen dieser Intensitätsänderungen für vorgegebene Positionen der Schallquelle im dreidimensionalen Raum, kann eine HRTF abgeleitet werden [19]. Einsatz finden HRTFs überall da, wo dreidimensionale Geräusche simuliert werden sollen. Da beispielsweise beim Tragen von Kopfhörern die Schallquelle immer an derselben Position ist, kann kein dreidimensionaler Effekt wie bei natürlichen Geräuschen durch die Veränderungen der Ohrmuschel entstehen. Soll dennoch ein räumlicher Ton hörbar sein, liefert die HRTF das Signal für beide Ohren, welches den gewünschten dreidimensionalen Eindruck vermittelt.

3.2 Web Audio API

Eine HRTF kann innerhalb der Web Audio API verwendet werden, um dreidimensionale Töne in Javascript zu simulieren. In diesem Abschnitt werden die grundlegenden Konzepte der Web Audio API vorgestellt. Im Folgenden wird besonders auf Aspekte eingegangen, welche im Rahmen dieser Arbeit Verwendung finden. Damit soll ein grundlegendes Verständnis für die in Kapitel 6 vorgestellte Implementierung geschaffen werden.

3.2.1 Knoten

Elementare Bestandteile der *Web Audio API* sind Knoten. Jeder Knoten wird für einen bestimmten Zweck verwendet. So gibt es beispielsweise Knoten, welche Töne erzeugen und andere, welche diese Töne durch Effekte verändern (z.B. Lautstärke erhöhen). Die API erlaubt es, beliebig viele solcher Knoten zu erzeugen. Letztendlich werden die erzeugten Knoten in einer bestimmten Reihenfolge zu einer Verarbeitungskette verbunden, die das gewünschte Geräusch liefert.
Die nächsten Abschnitte befassen sich mit einer Auswahl der für dieses Projekt wichtigsten Knoten.

Audio Kontext

Der *Audio Kontext* nimmt eine spezielle Rolle der Web Audio API ein. Streng genommen fällt er nicht unter die Kategorie „Knoten“, sondern ist vielmehr für die Erzeugung der Knoten und Prozesse der Audioverarbeitung sowie des Dekodierens zuständig. Jeder erzeugte Knoten läuft innerhalb genau eines Audio Kontexts.

Erzeuger

Unter die *Erzeugerknoten* fallen alle Knoten, welche eine Tonquelle repräsentieren. Sie stellen somit den Beginn einer Verarbeitungskette dar. Die erzeugten Töne können einerseits externen Audio Dateien entstammen, die durch den Audio Kontext dekodiert und in einem *AudioBufferSourceNode* repräsentiert werden. Andererseits bietet die Web Audio API einen *OscillatorNode* an, welcher zu einer bestimmten Frequenz einen Sinuston erzeugt. Des Weiteren gibt es die Möglichkeit, Töne durch *HTML5 <audio>* und *<video>* Elemente, sowie durch *WebRTC Streams* (z.B. Headsetmikrofone), zu erhalten.

Effekte

Effektknoten können innerhalb der Verarbeitungskette nach einem *Erzeugerknoten* oder einem anderen *Effektknoten* vorkommen. Jeder Knoten dieser Kategorie erfüllt den Zweck, eingehende Signale zu modifizieren. Die Web Audio API bietet hier viele Möglichkeiten. Neben Filtern und Kompressoren gibt es auch speziellere Knoten, z.B. den *DelayNode*, welcher das ankommende Signal mit einer Verzögerung ausgibt.

Ausgabe

Am Ende der Verarbeitungskette ermöglichen es *Ausgabeknoten* die erzeugten Geräusche auf ein Medium zu übertragen. Im Normalfall wird die Ausgabe auf Lautsprecher oder Kopfhörer mithilfe des *AudioDestinationNode* gelenkt. Wie bei den Erzeugerknoten ermöglicht es die Web Audio API zudem mit *WebRTC Streams* zu kommunizieren und folglich auch die Ausgabe auf diese zu leiten.

3.2.2 Panner-Listener Konzept

Durch das *Panner-Listener Konzept* können mithilfe der Web Audio API dreidimensionale Töne simuliert werden.
Panner ist dabei der Begriff für einen Effektknoten, durch welchen Schallquellen im dreidimensionalen Raum platziert werden können. Neben Positions- und Richtungsvektoren der Quelle enthält der Panner das sog. *panningModel*. Dieses gibt den Algorithmus an, der für die Simulation des dreidimensionalen Eindrucks verantwortlich ist. Im Rahmen dieser Arbeit wird dafür die HRTF (siehe Abschnitt 3.1.4) verwendet.

Die andere Komponente, der *Listener*, repräsentiert den Hörer im dreidimensionalen Raum. Auch seine Ausrichtung wird durch Vektoren beschrieben. Der erste Vektor gibt den Standpunkt des Hörers an. Der zweite beschreibt die horizontale Ausrichtung. Bei der Simulation eines menschlichen Hörers wäre dies die Richtung zum Kopf. Zuletzt wird die Blickrichtung mit einbezogen, wobei darauf zu achten ist, dass diese orthogonal zum Vektor der horizontalen Ausrichtung stehen muss.

3.3 Hardware / Sensoren

Da in diesem Projekt die Benutzerinteraktion hauptsächlich über Sensoren erfolgt, sollte ein grundlegendes Verständnis dieser Thematik vorhanden sein. Dazu werden sich die folgenden Abschnitte mit fundamentalen Aspekten der Funktionsweise ausgewählter Sensoren und der dafür benötigten Hardware befassen. Die Auswahl umfasst Sensoren, die durch Ionic 2 bereitgestellt sowie in diesem Projekt eingesetzt werden.

3.3.1 Magnetometer

Ein in einem mobilen Gerät verbautes Magnetometer wird für die Bestimmung der Stärke des umliegenden Magnetfelds verwendet. Sofern keine künstlich erzeugten lokalen Magnetfelder das Gerät umgeben, kann damit das Erdmagnetfeld gemessen und somit die Ausrichtung zum magnetischen Nordpol der Erde bestimmt werden [20]. Daher kann dem Magnetometer die Rolle eines digitalen Kompasses zugeschrieben werden.
Für die Bestimmung des Magnetfeldes wird in mobilen Geräten meist auf Sensoren zurückgegriffen, welche den *Halleffekt* ausnützen [21]. Dieser Effekt beschreibt eine elektrische Spannung, welche an einem unter Strom stehenden physikalischen Leiter, durch ein umgebendes Magnetfeld, entsteht. Die Halleffekt-Sensoren sind in der Lage, die hervorgerufene Spannung anhand ihrer Stärke und Polarität zu analysieren und dadurch die Ausrichtung des Geräts zum magnetischen Nordpol zu bestimmen [22].

3.3.2 Beschleunigungssensor

Ein Beschleunigungssensor misst die auf das Gerät einwirkende Beschleunigungskraft [23]. Das Grundprinzip der Messungen beruht auf der Trägheit der Masse. Es wird eine Masse an einer elastischen Vorrichtung (z.B. Feder) angebracht, wobei die Bewegungsrichtung eingeschränkt ist, sodass sich die Masse nur entlang einer Richtung vor und zurück bewegen kann. Wirkt nun eine Beschleunigungskraft auf die Masse, verschiebt sie sich entlang der vorgegebenen Achse entsprechend gegen die Richtung der Krafteinwirkung. Anschaulich wird die an der Masse angebrachte Feder gestaucht

bzw. gestreckt. Die Positionsänderung der Masse ist propotional zur einwirkenden Kraft und kann durch verschiedene Methoden erfasst werden. Anhand dieser Auslenkung kann folglich die Beschleunigungskraft bestimmt werden [24] [25].
In Abbildung 3.4 sind die Achsen der Beschleunigungsmessung an einem mobilen Gerät veraunschaulicht, wie sie in Ionic 2 Verwendung finden. Liegt das Smartphone mit dem Display nach oben auf einer ebenen Fläche, wirkt lediglich in positiver Z-Richtung eine Kraft. Wenn keine zusätzliche Kraft auf das Gerät ausgeübt wird, ist diese äquivalent zur Erdanziehung. Da durch die Erdanziehung folglich immer Beschleunigungskräfte auf das Gerät wirken, lassen sich auch in ruhiger Lage Neigungen des Smartphones messen. Diese bilden die Grundlage der in Absatz 6.3.4 beschriebenen Navigation.

Um mögliche Neigungen im weiteren Verlauf dieser Arbeit effizienter beschreiben zu können erfolgt an dieser Stelle noch eine kurze Definition der Begriffe *rollen, nicken* und *gieren*, welche den Begriffen der englischen *roll, pitch* und *yaw* Beschreibung entsprechen [20]. *Nach links rollen* beschreibt die Drehung um die Y-Achse gegen den Uhrzeigersinn, *nach rechts rollen* entsprechend im Uhrzeigersinn. *Nach vorne nicken* bedeutet eine Drehung um die X-Achse im Uhrzeigersinn, *nach hinten nicken* gegen den Uhrzeigersinn. *Rollen* und *Nicken* lassen sich mithilfe des eben beschriebenen Beschleunigungssensors messen. Für eine exakte Beschreibung der Lage des mobilen Gerätes im dreidimensionalen Raum ist zusätzlich die Drehung um die Z-Achse nötig. Diese wird durch das Magnetometer bestimmt (siehe 3.3.1) und bei einer Drehung im Uhrzeigersinn *rechtes gieren*, bei einer Drehung gegen den Uhrzeigersinn *linkes gieren* genannt.

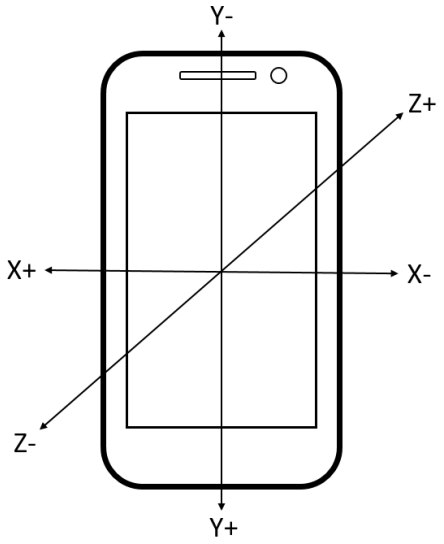

Abbildung 3.4: Achsen des Beschleunigungssensors

3.4 Ionic 2

Der folgende Abschnitt befasst sich mit dem Framework *Ionic 2*, das die Entwicklung mobiler hybrider Applikationen ermöglicht. Zuerst wird der Begriff *hybride Applikation* erläutert, wobei eine Unterscheidung von *nativen*, *cross-platform* sowie *Web-Applikationen* vorgenommen wird. Daraufhin werden wichtige Konzepte von Ionic 2 vorgestellt, wozu unter anderem auf die Frameworks *Cordova* und *Angular* eingegangen wird, auf denen Ionic 2 aufbaut.

3.4.1 Einordnung hybrider Applikationen

Bei der Entwicklung mobiler Anwendungen kann grundlegend zwischen vier Ansätzen unterschieden werden: *native, webbasierte, cross-platform* und *hybride*.
Native Anwendungen werden speziell für **eine** Plattform (z.B. Android, IOS) entwickelt. Hierbei kann, sofern es keine Versionskonflikte gibt, auf alle Programmierschnittstellen der Plattform direkt zugegriffen werden, wodurch die bestmöglichste Leistungsfähigkeit erreicht wird [26]. Da die Anwendung jedoch nur auf der entsprechenden Plattform

verwendet werden kann, entsteht ein hoher Zusatzaufwand, sofern die Unterstützung anderer Plattformen gewünscht ist [27]. An dieser Stelle setzen die *webbasierten, cross-platform* und *hybriden* Applikationen an. Sie alle versuchen, Applikationen für mehrere Platformen zu erstellen, die aus **einem** Quellcode hervorgehen und mit möglichst wenigen plattformspezifischen Anpassungen auskommen.
Webbbasierte Anwendungen (kurz: *web apps*) werden direkt durch den Browser des Gerätes angesprochen. Sie sind Webseiten, die gestalterisch auf mobile Geräte optimiert sind. Da sie im geräteeigenen Browser ablaufen, können sie von jedem mobilen Gerät, das einen solchen besitzt, ausgeführt werden. Sie müssen nicht installiert werden und sind sehr leicht wartbar. Dennoch eignen sie sich nur für bestimmte Anwendungen, da *web apps* auf wenige Programmierschnittstellen des Gerätes, lediglich die, die durch den Browser bereitgestellt werden, zugreifen können.
Der Ansatz der *cross-platform Applikationen* ist sehr ähnlich zu dem der *hybriden Applikationen*. Die Applikationen werden mithilfe von Webtechnologien (HTML, Javascript, CSS) erstellt und können für mehrere Plattformen kompiliert werden. Der Unterschied zwischen *cross-platform* und *hybrid* ist dabei, dass bei cross-platform Anwendungen die Benutzeroberfläche für jede Platform separat entwickelt wird, bei hybriden Applikationen wird diese nur einmal ausgearbeitet. Xamarin als ein cross-platform Framework verwendet bis zu 75% des geschriebenen Codes platformunabhängig [1]. Dies wird durch neue Konzepte wie z.B. Xamarin.Forms sogar auf nahezu 100% gesteigert.
Auf dem Endgerät verhalten sich beide wie eine native Anwendung, d.h. sie können beispielsweise im jeweiligen App-Store angeboten werden. Dies wird erreicht, indem die Applikation in einem *WebView* abläuft. Dieser ermöglicht, Webinhalte innerhalb der Anwendung darzustellen. Wie in Abbildung 3.5 zu sehen, wird eine hybride Anwendung durch eine native Applikation realisiert, welche lediglich aus einem WebView besteht, der die mit Webtechnologien entwickelte Oberfläche und Logik enthält. Der große Vorteil im Vergleich zu web apps ist, dass hybride Anwendungen auf geräteeigene Hardware (z.B. Sensoren) zugreifen können. Für die Verbindung zwischen nativen Bibliotheken und Javascript wird in Ionic 2 auf das Framework *Cordova* zurückgegriffen.

[1] Xamarin Homepage: `https://www.xamarin.com/platform`

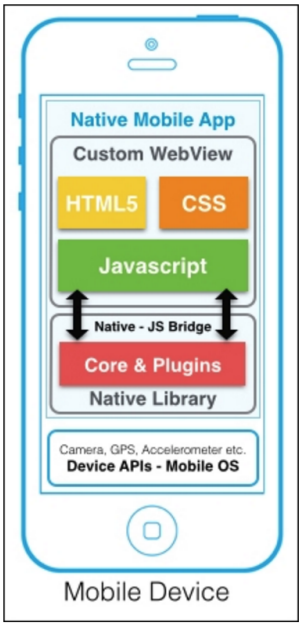

Abbildung 3.5: Aufbau hybrider Applikationen [28]

3.4.2 Cordova

Cordova ist ein Framework zur Erstellung von cross-platform Applikationen. An dieser Stelle sei nochmal hervorgehoben, dass Ionic 2 auf Cordova aufbaut, es durch einige Erweiterungen ergänzt, jedoch grundlegende Aspekte gleich sind. Cordova Applikationen werden beispielsweise ebenfalls in einem WebView ausgeführt und mit Webtechnologien entwickelt. Eine der wichtigsten Aufgaben von Cordova innerhalb Ionic 2 ist die in Abschnitt 3.4.1 angesprochene Verbindung zwischen nativen Bibliotheken und Javascript. Dazu stellt Cordova APIs bereit, die der Cordova-WebView eine Kommunikation mit sog. *Cordova Plugins* ermöglicht [29]. Ein Cordova Plugin ist in der Lage, geräteeigene Hardwarekomponenten (Sensoren, Filesystem, Kontakte, ...) anzusprechen. Für viele dieser Komponenten stehen bereits Cordova Plugins zur Verfügung, es ist jedoch auch möglich selbst ein solches Plugin zu schreiben.

3.4.3 Architektur

Die zweite große Komponente, auf welcher Ionic 2 basiert ist Angular, ein Javascript-Framework, das innerhalb Ionic 2 in der zweiten Version (Angular2) verwendet wird. Die Architektur von Ionic 2 ist maßgeblich von Angular beeinflusst, weshalb sich die folgenden Abschnitte zu großen Teilen mit Konzepten von Angular befassen werden.

Module

Module sind grundlegende Strukturelemente von Angular. Es gibt von ihnen in jeder Angular Anwendung mindestens eines, das sog. *root module*. Codebeispiel 3.1 zeigt ein solches root module, wie es in Ionic 2 verwendet wird. Die exportierte Klasse, mit dem konventionellen Namen eines root modules, *AppModule* wird durch einen *@NgModule decorator* als ein Modul gekennzeichnet. Der decorator nimmt ein Metadaten-Objekt entgegen, das das Modul genauer spezifiziert.
Im Array *declarations* müssen alle verwendeten Components des Moduls angegeben werden. Components werden im nächsten Abschnitt behandelt.
Sobald andere Module innerhalb eines Moduls verwendet werden, müssen diese importierten Module im *imports* Array dem Modul bekannt gemacht werden. In Ionic Anwendungen ist mindestens das IonicModule importiert, dessen statische Methode *forRoot()* den root component setzt. Zusätzlich enthält das IonicModule alle Components, Directives und Services, die das Ionic 2 Framework bereitstellt. Directives und Services werden im weiteren Verlauf dieses Abschnitts genauer betrachtet.

```
1 import { NgModule} from '@angular/core';
2 import { IonicApp, IonicModule} from 'ionic-angular';
3 import { IonicStorageModule } from '@ionic/storage';
4 import { MyApp } from './app.component';
5
6 @NgModule({
7   declarations: [MyApp],
```

```
8   imports: [
9     IonicModule.forRoot(MyApp),
10    IonicStorageModule.forRoot()
11  ],
12  bootstrap: [IonicApp],
13  entryComponents: [MyApp],
14  providers: []
15 })
16 export class AppModule {}
```

Listing 3.1: Grundlegendes Modul in Ionic 2

Inhalte im *bootstrap* Array sind Components, die von Angular beim Start der Anwendung innerhalb der index.html Datei angelegt werden. Im <body> der index.html Datei ist der Tag *<ion-app>* zu finden, in welchen diese Components geladen werden.
Um sicherzustellen, dass der Angular Compiler auch für dynamisch erzeugte Components eine *ComponentFactory* erstellt, müssen diese innerhalb der *entryComponents* gekennzeichnet werden [2]. Da alle Seiten in Ionic 2 durch den *NavigationController* geladen werden, müssen sie auch in den *entry components* angegeben werden.
Services, die in allen Components der Anwendung verfügbar seien sollen, müssen im *providers* Array enthalten sein.

Component

Components definieren bestimmte Teile der Benutzeroberfläche inklusive ihre Logik. Ionic 2 stellt bereits viele vorgefertigte Components bereit, die übliche GUI-Elemente in mobilen Anwendungen, wie z.B Floating Action Buttons repräsentieren. Es ist zudem möglich, Components selbst zu erstellen, wie in Codebeispiel 3.2 zu sehen.
Mithilfe eines *@Component decorator* wird eine Klasse als Component deklariert. Wie der *@NgModule decorator* nimmt auch der *@Component decorator* ein Metadaten-Objekt entgegen. Dort können direkt Informationen der GUI als template angegeben

[2] Angular Dynamic Component Loader: `https://angular.io/docs/ts/latest/cookbook/dynamic-component-loader.html`

werden. Da dies schnell unübersichtlich werden kann, ist es möglich, als *templateUrl* den Pfad einer HTML Datei anzugeben, in die diese Angaben ausgelagert werden. Der Wert des *selector* Attributs stellt einen CSS-Selektor dar. Aus einer CSS Datei kann damit der gesamte Component referenziert werden.
Eine komplette Seite einer Ionic 2 Applikation ist ebenfalls als Component zu realisieren. Dazu wird ein Ordner innerhalb des *pages*-Verzeichnis erstellt, der eine Javascript- (bzw. Typescript-) Datei, sowie zugehörige HTML und CSS Dateien beinhaltet. Listing 3.2 stellt den Inhalt einer solchen Typescript Datei dar, worin mithilfe der eben genannten Metadaten des component-decorator die Verknüpfung zu HTML und CSS erzielt wird. Das Ionic 2 CLI erstellt mit dem Befehl `>ionic g page pageName` einen Ordner im pages-Verzeichnis mit allen benötigten Dateien.

```
1  import { Component } from '@angular/core';
2  import { NavController } from 'ionic-angular';
3  import { UserDataService } from ./userDataService
4  import { OtherPage } from '../pages/other-page/other-page';
5
6  @Component({
7    selector: 'page-home',
8    templateUrl: 'home.html'
9    //template: '<h1>Hello World!</h1>'
10 })
11 export class MyApp {
12     constructor(navCtrl:NavController, uDS:UserDataService){...}
13
14     goToOtherPage(){
15         this.navCtrl.push(OtherPage, {comingFrom: 'MyApp'});
16     }
17
18     //page is no longer active page
19     //use userDataService to store changes
```

```
20      ionViewDidLeave(){
21          this.uDS.setData({name:'Martin'});
22      }
23  }
```

Listing 3.2: Component in Ionic 2

Navigation zwischen Seiten

Die Navigation zwischen Seiten wird in Ionic 2 durch einen *Pagestack* realisiert, der von einem sog. *NavController* verwaltet wird. Die oberste Seite des Stacks wird innerhalb der Anwendung angezeigt. Der *NavController* bietet wie in Codebeispiel 3.2 zu sehen unter anderem die Möglichkeit, Seiten auf den Stack zu pushen. Dabei können der neuen Seite Informationen in Form eines Objektes mitgegeben werden. Die aktive Seite lässt sich mit der *pop()* Funktion aus dem Stack entfernen. Weitere typische Operationen eines Stacks (insert, remove, ...) sind ebenfalls im *NavController* enthalten.
In jedem Component, der von dem *NavController* gepusht oder gepoppt wird, können *Lifecycle events* genutzt werden. In Codeauszug 3.2 wird dazu der Event verwendet, der gefeuert wird, sobald die Seite nicht mehr die aktive Seite ist. Dort könnten beispielsweise temporäre Daten der Seite lokal gespeichert werden.

Speicher

Um Daten persistent lokal zu speichern, stellt Ionic 2 ein eigenes Modul (*IonicStorageModule*) zur Verfügung. Das Modul verwendet im nativen Kontext priorisiert SQLite, da es im Vergleich zu *localstorage* und *IndexedDB* stabiler und mit weniger Problemen läuft [3]. Es werden immer key-value Paare verwendet, und da alle Speicherzugriffe asynchron ablaufen, gibt jede Methode des storage-Moduls ein *Promise* zurück. Wie in Codebeispiel 3.3 zu sehen, ermöglicht die Methode *set(key,value)* das Speichern eines solchen key-value Paares. Mit der Methode *get(key)* wird ein Promise zurückgegeben, das den zugehörigen value-Wert auswerten lässt. Wird ein key-value Paar nicht mehr benötigt, kann es mit *remove(key)* aus dem lokalen Speicher entfernt werden.

Services

Services ermöglichen die Auslagerung von Logik. Vor allem für wiederkehrende Aufgaben ist deren Verwendung sinnvoll und sie tragen zu einem übersichtlicheren und leichter wartbaren Code bei.
Bei *Services* wird die exportierte Klasse mit einem *@Injectable decorator* versehen. Die Logik des Services ist wie bei *Components* innerhalb der Klasse definiert. In Listing 3.2 wurde ein *UserDataService* importiert und dessen Methode *setData(userData)* aufgerufen. Der folgende Code (siehe Listing 3.3) zeigt diesen importierten Service, der beispielhaft Nutzerdaten verwaltet.

```
1  import { Injectable } from '@angular/core';
2  import{ Storage } from '@ionic/storage';
3
4  @Injectable()
5  export class UserDataService {
6      constructor(storage: Storage){ ... }
7
8      getData(){
9          this.storage.get('userData').then((_userData)=>{
10             return _userData;
11         )};
12     }
13
14     setData(_userData){
15         this.storage.set('userData', JSON.stringify(_userData));
16     }
17 }
```

Listing 3.3: Service in Ionic 2

Würde man Codebeispiel 3.2 und 3.3 nun ausführen, würde eine *"No provider for UserDataService Exception"* auftreten. Der Grund dafür liegt darin, dass der Service dem Modul nicht bekannt gemacht wurde, und somit auch nicht den Components in diesem Modul. Es muss also wie in der Erklärung für Listing 3.1 der UserDataService innerhalb des provider-Arrays eingetragen werden. Wenn der Service nicht in allen Components des Moduls verwendet werden soll, kann er auch im Metadaten-Objekt des *@Component decorator* innerhalb des providers-Array angegeben werden.

Directives

Directives ermöglichen es, DOM-Elemente zu manipulieren. Sie kommen in zwei verschiedenen Arten vor: *Attribute Directives* und *Structural Directives*.
Attribute Directives verändern das Aussehen und Verhalten (z.B. bei mouseover) von Elementen. Die in Codebeispiel 3.4 verwendete Attribute Directive *text-center* ist identisch zur CSS-Eigenschaft *text-align: center*. Hierzu liefert Ionic 2 viele vorgefertigte Attribute Directives.
Structural Directives formen das HTML-Layout der Seite durch Hinzufügen oder Entfernen von DOM-Elementen. Vorausgesetzt es existiert ein Array *products*, wird in Listing 3.4 durch die **ngFor Directive* über das Array iteriert und für jeden Eintrag ein Listenelement erstellt. Zusätzlich wird der Index der Iteration bestimmt. Dieser findet in der zweiten Directive **ngIf* Verwendung. Die **ngIf* Directive ist dafür verantwortlich, dass nur Elemente erstellt werden, bei denen die angegebene Bedingung zutrifft. In diesem Beispiel würde folglich eine Liste erzeugt werden, die alle (bis auf den ersten) Produktnamen des Arrays products beinhaltet.
Eine weitere häufig eingesetzte Directive ist *ngSwitch*. Sie ist keine Structural Directive, daher wird sie auch nicht mit einem * eingeleitet. *ngSwitch* fällt unter die Attribute Directives, wobei die Structural Directives **ngSwitchCase* und **ngSwitchDefault* von ihr gesteuert werden.

```
<div text-center>
    Centered Text
</div>

<ul *ngFor='let product of products; let i=index'>
    <li *ngIf='i != 0'>{{product.name}}</li>
</ul>

<div [ngSwitch]="products[0]?.categorie">
  <p *ngSwitchCase='food'>First product from food</p>
  <p *ngSwitchCase='garden'>First product from garden</p>
  <p *ngSwitchDefault>First product from unknown</p>
</div>
```

Listing 3.4: Directives in Ionic 2

Zusätzlich zu den von Angular und Ionic 2 bereitgestellten Directives ist es möglich, selbst Attribute- und Structural Directives zu schreiben. Das folgende Beispiel (siehe Listing 3.5) zeigt eine Attribute Directive mit dem Namen *myOwnDirective*, die Elementen eine rote Hintergundfarbe gibt. Die exportierte Klasse wird mithilfe eines *@Directive decorator* als Directive gekennzeichnet. Das selector-Attribut im Metadaten-Objekt des decorators gibt die Bezeichnung an, die im HTML-Code einem DOM-Element gegeben werden muss, um die Attribute Directive darauf anzuwenden.

```
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({
    selector: '[myOwnDirective]'
})

```

```
7  export class OwnDirective {
8      constructor(el: ElementRef) {
9          el.nativeElement.style.backgroundColor = 'red';
10     }
11 }
```

Listing 3.5: Eigene Directives in Ionic 2, entnommen aus https://angular.io/docs/ts/latest/guide/attribute-directives.html

4

Anforderungen

In diesem Kapitel werden die funktionalen- und nichtfunktionalen Anforderungen an die zu entwickelnde Anwendung definiert.

4.1 Funktionale Anforderungen

Funktionale Anforderungen definieren Funktionen, welche die Applikation erfüllen soll. Die nachfolgende Auflistung (siehe Tabelle 4.1) skizziert diese Anforderungen einschließlich ihrer Prioritäten. Die Skala der Prioritäten umfasst die Werte von 1 (unwichtig) bis 5 (unbedingt notwendig). Anschließend werden die funktionalen Anforderungen genauer beschrieben und die Entscheidung über die Prioritäten dargelegt.

Abkürzung	Anforderung	Priorität
FA#01	Räumliche Töne simulieren	5
FA#02	Töne verwalten	2
FA#03	Trainingseinheiten erstellen	5
FA#04	Trainingseinheit bearbeiten	1
FA#05	Geräuschquelle erstellen	5
FA#06	Geräuschquelle bearbeiten	3
FA#07	Trainingseinheit starten	5
FA#08	Trainingseinheit verlassen	4
FA#09	Steuerung mithilfe der Sensorik	4
FA#10	Bestimmung der Genauigkeit	5
FA#11	Rückmeldung der erzielten Leistung	5

Tabelle 4.1: Funktionale Anforderungen

FA#01 Räumliche Töne simulieren

Die Anwendung soll Töne im dreidimensionalen Raum simulieren und über die Audioschnittstelle des Endgerätes ausgeben können.

FA#02 Töne verwalten

Es soll dem Nutzer möglich sein, eine Übersicht der lokal gespeicherten Töne einzusehen und Töne aus der Sammlung zu löschen. Ferner soll die Sammlung durch neue Töne aus lokalen Audiodateien im Format *mp3, wav* und *ogg* erweitert werden können.

FA#03 Trainingseinheiten erstellen

Trainingseinheiten beinhaltet beliebig viele Geräuschquellen, die je nach Spielmodus (siehe 5.2) auf unterschiedliche Art und Weise lokalisiert werden müssen. Der Nutzer soll unter Angabe eines Namens und eines Spielmodus eine Trainingseinheit erstellen können. Es soll dabei keine Beschränkung der Anzahl geben.

FA#04 Trainingseinheit bearbeiten

Bereits erstellte Trainingseinheiten sollen gelöscht werden können. Bei Trainingseinheiten mit Spielmodus eins oder zwei soll der Spielmodus auf eins bzw. zwei geändert werden können. Für jede vorhandene Trainingseinheit soll es möglich sein, den Namen zu ändern.

FA#05 Geräuschquelle erstellen

Innerhalb einer Trainingseinheit soll es möglich sein, unter Angabe der dreidimensionalen Koordinate und eines Tons eine Geräuschquelle zu erzeugen.

FA#06 Geräuschquelle bearbeiten

Bereits erstellte Geräuschquellen sollen gelöscht werden können. Des Weiteren soll die Möglichkeit bestehen, die dreidimensionalen Koordinaten und den Ton existierender Geräuschquellen zu ändern.

FA#07 Trainingseinheit starten
Die Anwendung soll das Starten einer Trainingseinheit ermöglichen. Nach dem Start soll der Nutzer die entsprechende Aufgabe der gestarteten Trainingseinheit ausführen können.

FA#08 Trainingseinheit verlassen
Es soll zu jeden Zeitpunkt innerhalb einer Trainingseinheit möglich sein, diese zu verlassen. Nach erfolgreichem Verlassen wird über keine erzielten (Zwischen-) Leistungen informiert.

FA#09 Steuerung mithilfe der Sensorik
Die Anwendung soll Veränderungen der dreidimensionalen Lage des Endgeräts erkennen und daraus die Intention der Nutzereingabe umsetzen können.

FA#10 Bestimmung der Genauigkeit
Durch einen Abgleich der von den Sensoren gelieferten Daten und der Koordinaten der Geräuschquellen soll die Anwendung die erzielte Genauigkeit des Nutzers bestimmen können.

FA#11 Rückmeldung der erzielten Leistung
Nach Abschluss einer Trainingseinheit soll der Nutzer über seine erzielte Leistung informiert werden.

Erläuterung der Prioritäten

FA#01, FA#03, FA#05, FA#07, FA#08, FA#10, FA#11 sind für die Grundfunktionalität der Anwendung verantwortlich und daher unbedingt notwendig. **FA#02** wurde als weniger wichtig eingestuft, da die Applikation mit einem vorgefertigten Set an Tönen ausgeliefert wird, welche für viele Einsatzgebiete ausreichen sollte. **FA#04** hat die Priorität 1, da ein gleiches Level mit den gewünschten Änderungen mit geringem Zusatzaufwand neu erstellt werden kann. **FA#06** wurde als wichtig eingestuft, da das Löschen und

Ändern einer Geräuschquelle deutliche Auswirkungen auf eine Trainingseinheit hat. **FA#09** ist sehr wichtig, da ohne funktionierende Sensorik lediglich Trainingseinheiten mit Spielmodus eins vernünftig ausgeführt werden könnten.

4.2 Nichtfunktionale Anforderungen

Durch die nichtfunktionalen Anforderungen wird beschrieben, wie und mit welcher Qualität die funktionalen Anforderungen umgesetzt werden sollen. Zudem werden Qualitätsaspekte, die die gesamte Anwendung und ihre Entwicklung betreffen, definiert. In Tabelle 4.2 werden die nichtfunktionalen Anforderungen aufgelistet und daraufhin analog zu den funktionalen Anforderungen erläutert.

Abkürzung	Anforderung	Priorität
NFA#01	Verwendete Technologien	4
NFA#02	Leistung & Effizienz	3
NFA#03	Benutzbarkeit	5
NFA#04	Zuverlässigkeit	5
NFA#05	Nachvollziehbarkeit	5
NFA#06	Wartbarkeit & Erweiterbarkeit	4
NFA#07	Motivation	4

Tabelle 4.2: Nichtfunktionale Anforderungen

NFA#01 Verwendete Technologien
Die Anwendung soll mithilfe des hybriden Frameworks Ionic 2 in der Version 2.2.1 unter Verwendung von Typescript erstellt werden. Für die Simulation räumlicher Töne soll auf die lizenzfreie Web Audio API zurückgegriffen werden.

NFA#02 Leistung & Effizienz
Die Anwendung soll für mobile Endgeräte optimiert werden. Vor allem eine Reduzierung des Ressourcenbedarfs ist daher wünschenswert.

NFA#03 Benutzbarkeit
Die Anwendung soll leicht bedienbar sein. Um eine reibungslose Benutzbarkeit zu gewährleisten werde kurze Antwortzeiten seitens der Applikation gefordert.

NFA#04 Zuverlässigkeit
Die Anwendung darf nicht abstürzen oder stehen bleiben. Durch syntaktische Prüfung soll jede mögliche Eingabe des Nutzers verarbeitet werden können.

NFA#05 Nachvollziehbarkeit
Es soll dem Nutzer zu jedem Zeitpunkt bewusst sein welche Folgen seine Eingaben haben. Die Anwendung soll daher den Benutzer unterstützen, ein geeignetes mentales Modell zu entwickeln.

NFA#06 Wartbarkeit & Erweiterbarkeit
Es soll möglich sein, die Anwendung im Nachhinein ohne großen Zusatzaufwand zu verändern. Damit soll gewährleistet werden, dass mögliche Erweiterungen und Problembehebungen problemlos durchgeführt werden können.

NFA#07 Motivation
Die Anwendung soll die Benutzer motivieren. Dadurch soll unter anderem erreicht werden, dass sie zu einer erneuten Durchführung von Trainingseinheiten nicht abgeneigt sind.

Erläuterung der Prioritäten

NFA#01 ist als sehr wichtig eingestuft, da die in dieser Arbeit beschriebenen Technologien eingesetzt werden sollen. Lediglich die Versionen könnten sich im Laufe der Bearbeitung ändern, werden jedoch zu Beginn der Realisierung festgelegt um eine definierte Entwicklungsumgebung zu gewährleisten. **NFA#02** besitzt die Priorität 3, da die Anwendung keine anspruchsvollen Berechnungen leisten muss und keine Internetkonnektivität besitzt. Dennoch sollte auf eine effiziente Implementierung geachtet werden,

um NFA#03 gerecht zu werden. **NFA#03, NFA#04, NFA#05** wurden als unbedingt notwendig klassifiziert, weil sie die Zufriedenheit der Nutzer beeinflussen. Da für diese Anwendung die mitgebrachte Motivation der Nutzer eine entscheidende Rolle spielt, sind diese nichtfunktionalen Anforderungen besonders zu berücksichtigen. **NFA#06** ist sehr wichtig, da auch im Rahmen dieser Arbeit Probleme genannt und Verbesserungs- bzw. Erweiterungsvorschläge gegeben werden. Um diese umzusetzen ist eine gute Wartbarkeit und Erweiterbarkeit von Bedeutung. **NFA#07** ist sehr wichtig.Um möglichst aussagekräftige Ergebnisse zu erhalten muss sichergestellt werden, dass die Nutzer ausreichend motiviert werden.

5

Konzeption & Entwurf

Auf Basis der in Kapitel vier dargestellten Anforderungen befasst sich dieses Kapitel mit der Konzeption und dem Entwurf der zu entwickelnden Applikation. Dabei wird zuerst auf das Konzept eingegangen, worunter auch eine kurze Erklärung der drei Spielmodi fällt. Daraufhin folgen Mockups der Anwendung, anhand welcher einige Designentscheidungen begründet werden. Zuletzt wird das zu verwendende Datenmodell definiert.

5.1 Konzept der Applikation

Die Nutzergruppe der zu entwickelnden Applikation sind Personen, die hinsichtlich ihrer Fähigkeit des räumlichen Hörens untersucht werden sollen. Der Fokus liegt dabei vor allem auf Tinnituspatienten. Bei ihnen könnte die Anwendung zudem eine kurzfristige Linderung der Beschwerden hervorrufen. Dies wird durch akustische Stimulation versucht herbeizuführen.
Die grundlegende Aufgabe der Nutzer liegt in der Lokalisation von Geräuschen im dreidimensionalen Raum. Sog. *Trainingseinheiten* beinhalten Erzeuger solcher Geräusche. An die Nutzer werden über den Audio Ausgang des Gerätes die Geräusche weitergeleitet, die eine virtuelle Person innerhalb einer Trainingseinheit hört. Für eine bestmögliche Genauigkeit sollte der Nutzer dabei Stereokopfhörer tragen. Die virtuelle Person wird im Folgenden *Hörer* genannt.
Um eine Trainingseinheit möglichst gut auf den Nutzer abzustimmen, ist es empfehlenswert, vorerst die zu verwendenden Töne zu bestimmen. Dazu stellt die Anwendung eine Seite für die Verwaltung der Töne bereit. Eine auf den Nutzer abgestimmte Auswahl an

Tönen sollte sich positiv auf die Motivation auswirken (NFA#07).
Daraufhin kann eine Trainingseinheit erstellt werden, wobei unter anderem der gewünschte Spielmodus angegeben wird. Abschnitt 5.2 beschäftigt sich mit den verschiedenen Spielmodi. Zudem können innerhalb einer Trainingseinheit Tonquellen erstellt werden. Für jede Tonquelle kann die Position einerseits über den Winkel und die Distanz zum Hörer angegeben werden. Andererseits ist es auch möglich mit Angabe von Koordinaten eine genaue Positionierung der Tonquellen im dreidimensionalen Raum zu erreichen. Abbildung 5.1 veranschaulicht die zu verwendenden Angaben im benutzten Koordinatensystem. Der blaue Pfeil gibt dabei immer die initale Ausrichtung des Hörers an. Zum Beispiel entspricht ein Ton an Position 0|0|1 einem Winkel von 90 Grad bei einer Distanz von 1. Dieser Ton würde dem Nutzer vorkommen als käme er von rechts.

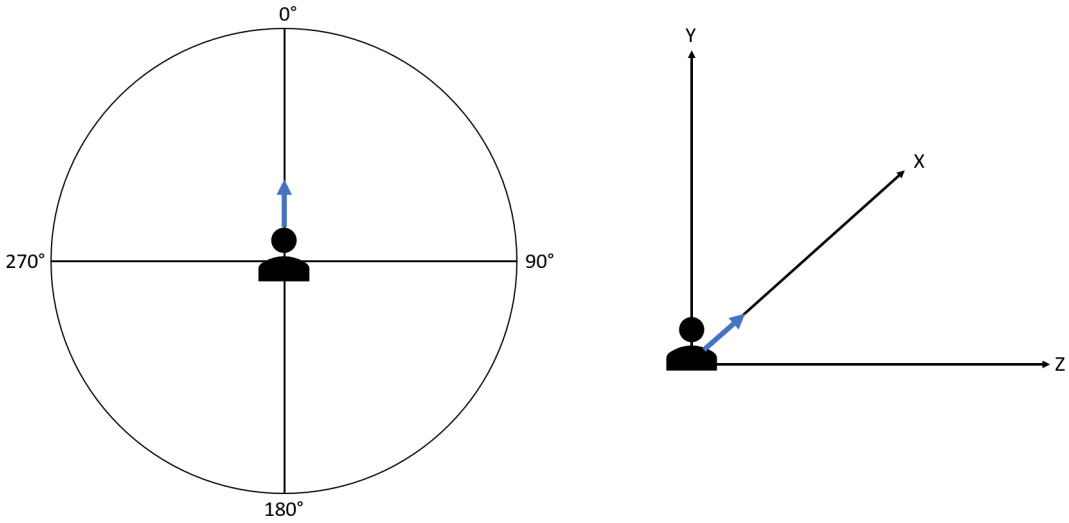

Abbildung 5.1: Verwendetes Winkel- und Koordinatensystem

5.2 Spielmodi

Ist eine Trainingseinheit mit mindestens einer Tonquelle erstellt, kann sie gestartet werden. Dabei unterscheidet sich die Art und Weise, auf die der der Nutzer die Ge-

räusche lokalisieren muss je nach Spielmodus. Im Folgenden werden die drei möglichen Spielmodi erläutert. Es werden die in Abschnitt 3.3.2 definierten Begriffe der Ausrichtung des mobilen Gerätes verwendet.

Spielmodus 1 (starr)

In Spielmodus 1 bleibt die Position und Ausrichtung des Hörers fest. Für jeden hörbaren Ton muss der Winkel angegeben werden, aus dem der Nutzer den Ton vermutet.

Spielmodus 2 (drehen)

In diesem Spielmodus ist es möglich die Ausrichtung des Hörers durch Gieren des mobilen Gerätes zu verändern. Die Ausrichtung des Hörers ist direkt an die Drehung des Gerätes gekoppelt. Giert man nach rechts wird der Winkel der Blickrichtung des Hörers erhöht, bei linkem Gieren verringert. Mit diesen Drehungen soll der Nutzer nun den Hörer so platzieren, dass er möglichst genau auf eine Tonquelle blickt. Die Drehung kann über die Benutzeroberfläche bestätigt werden. Sofern mehrere Tonquellen innerhalb der Trainingseinheit vorhanden sind, wird nach einer Bestätigung der Ton der getroffenen Tonquelle ausgeblendet und der Nutzer muss den nächsten Ton auf die gleiche Art und Weise lokalisieren. Diese Art von Spielmodus ist angelehnt an ein bereits entwickeltes Konzept eines mobile Serious Games für Tinnitus Patienten [30].

Spielmodus 3 (laufen)

Im dritten Spielmodus kann die Position des Hörers verändert werden. Je nach Einstellungen ist es zudem möglich, den Hörer wie in Spielmodus 2 zu drehen. Für die Veränderung der Position muss das mobile Gerät gerollt, bzw. genickt werden. Nach links rollen lässt den Hörer auf der X-Z-Ebene nach links laufen (Z-Koordinate wird verringert), rechtes Rollen bewegt den Hörer nach rechts. Nach vorne Gieren erhöht die X-Koordinate des Hörers, er bewegt sich nach vorne. Um nach hinten zu laufen muss das Gerät nach hinten genickt werden. Die Aufgabe des Nutzers besteht darin, zu den

vorhandenen Tonquellen möglichst genau zu navigieren und sobald die gewünschte Zielposition erreicht ist zu entscheiden, ob die Tonquelle über oder unter dem Hörer liegt.

5.3 Mockups

Mockups veranschaulichen die grundlegenden Auslegung der zu entwickelnden Benutzeroberfläche. Bevor auf seitenspezifische Designentscheidung eingegangen wird, soll ein Überblick der vorhandenen Seiten gegeben sein. Abbildung 5.2 beinhaltet alle Seiten, die innerhalb der Anwendung vorhanden sind.

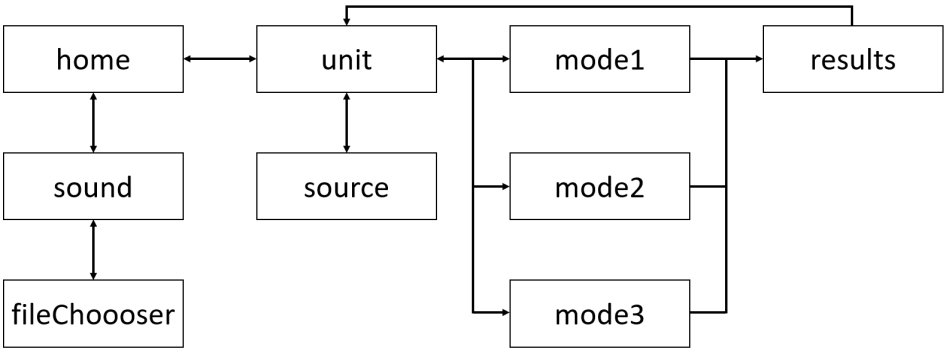

Abbildung 5.2: Überblick: alle Seiten der Applikation

Im Folgenden werden Mockups für jede dieser Seiten vorgestellt und relevante Designentwürfe begründet. Dabei werden die Seiten in Abbildung 5.2 primär von oben nach unten und sekundär von links nach rechts durchlaufen.

home

Als erste Seite wird dem Nutzer die *home* Seite präsentiert (Abbildung 5.3). Diese zeigt alle vorhandenen Trainingseinheiten in einer nach Spielmodus geordneten Liste an. Durch den Button oben rechts gelangt man zur Seite *sound*, welche die vorhandenen Töne verwaltet. Der Button unten in der Mitte ermöglicht es, eine neue Trainingseinheit zu erstellen. Dazu wird zur Seite *unit* weitergeleitet. Beim Klicken auf eine Trainingseinheit wird ebenfalls zur Seite *unit* weitergeleitet.

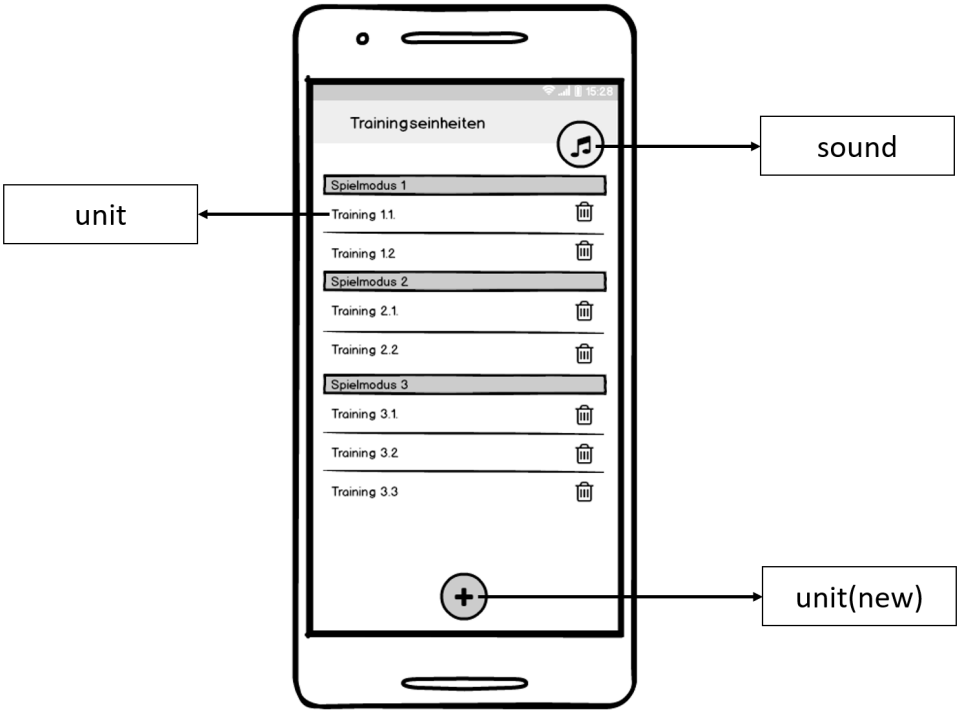

Abbildung 5.3: Seite: home

sound

In Abbildung 5.4 ist links die Seite *sound* zu sehen, rechts die Seite *fileChooser*. Mit ihr können die Töne der Anwendung verwaltet werden. Die Liste enthält alle Töne, die zur Zeit für die Applikation zur Verfügung stehen. Durch den Button unten in der Mitte wird die Seite *fileChooser* geöffnet. Diese ermöglicht es, durch lokal verfügbare Dateien des Gerätes zu navigieren. Damit kann zu neuen Audiodateien navigiert und diese

ausgewählt werden, um sie der Anwendung hinzuzufügen. Ist dies geschehen, wird dem Nutzer erneut die Seite *sound* präsentiert, die nun den ausgewählten Ton beinhaltet, sofern der Dateityp unterstützt ist.

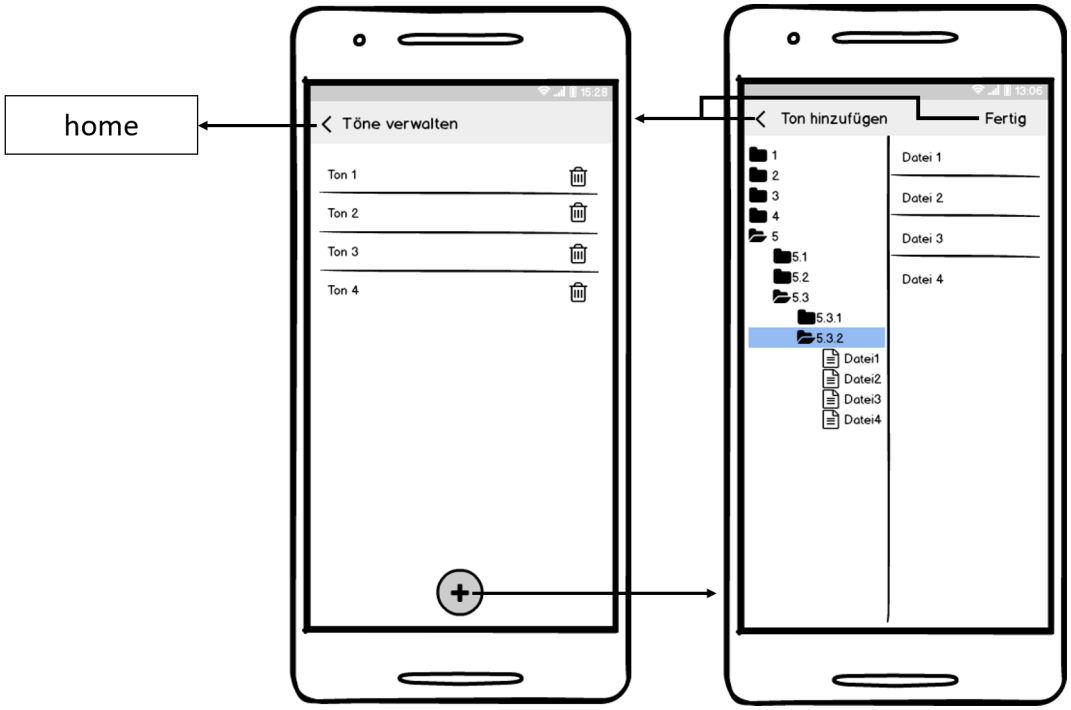

Abbildung 5.4: Seite: sound & fileChooser

unit

Abbildung 5.5 zeigt die Seite *unit*. Links ist sie zu sehen, wenn eine neue Trainingseinheit erstellt werden soll und rechts, wenn eine vorhandene Trainingseinheit geöffnet wurde. Bei neu zu erstellenden Trainingseinheiten wird der Name der Trainingseinheit in einem Eingabefeld erwartet. Darunter ist eine Scroll-View zu sehen, welche die zugehörigen Tonquellen enthält. Der Plus-Button leitet hierbei auf die Seite *source* weiter, in welcher neue Tonquellen erzeugt werden können. Bereits vorhandene Tonquellen können geklickt, und nach einer Weiterleitung auf die Seite *source* verändert werden. Unter der Scroll-View befindet sich die Auswahl des Spielmodus. Bei vorhandenen Trainingseinheiten kann darüber der Spielmodus gemäß FA#04 geändert werden. Sobald eine

Trainingseinheit erstellt wurde ist es ebenfalls auf dieser Seite möglich über den Button *Start* die Trainingseinheit zu starten. Dazu wird je nach Spielmodus auf die Seiten *mode 1, mode2* oder *mode3* navigiert.
Da unter Umständen viele Tonquellen innerhalb einer Trainingseinheit vorkommen können, wurde für ihre Repräsentation eine Scroll-View verwendet. Diese bietet die beste Nutzung bei variabel langen Listen. Um auf der Seite schnell zu erkennen, ob es sich um die Änderung oder Erstellung einer Trainingseinheit handelt, variiert der Name der App-Bar sowie die zu klickenden Buttons am unteren Bildschirmrand.

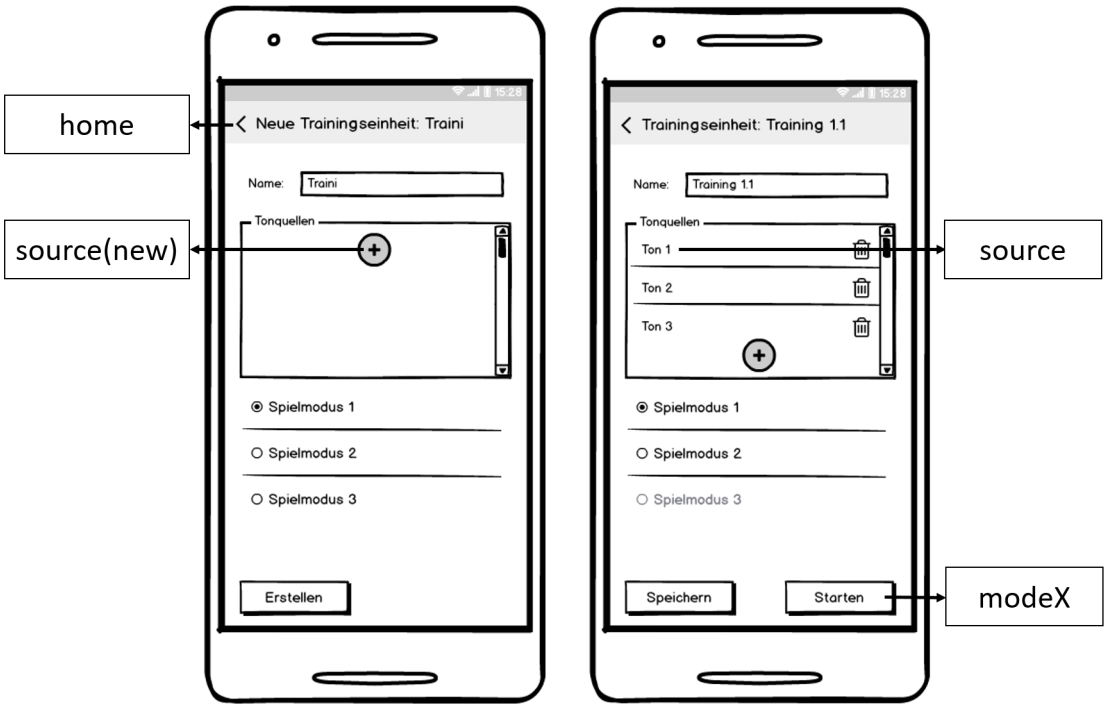

Abbildung 5.5: Seite: unit

source

Die Seite *source* (Abbildung 5.6) unterscheidet sich ebenfalls für neu zu erstellende bzw. vorhandene, zu ändernde Tonquellen. Auf der linken Seite ist das Layout für eine neue Tonquelle zu sehen, rechts das für eine vorhandene. Wie Trainingseinheiten besitzt eine Tonquelle einen Namen, der über ein Eingabefeld eingegeben werden kann. Darunter

befindet sich der Abschnitt, mit dem der zu erzeugende Ton bestimmt werden kann. Nach einem Klick auf das Datei-Icon öffnet sich, wie links zu sehen, ein Pop-Up mit einer Liste innerhalb einer Scroll-View, woraus der gewünschte Ton ausgewählt werden soll. Darunter sind die Eingaben für die Position der Tonquelle zu finden. Wie am Anfang dieses Kapitels beschrieben stehen dazu zwei Möglichkeiten bereit. Einerseits über den Winkel und die Distanz zum Hörer, andererseits durch absolute Positionierung unter Angabe der dreidimensionalen Koordinaten. Vorgenommene Änderungen sollen direkt in den jeweils anderen Slidern sichtbar sein. Wird beispielsweise der Winkel geändert, sollen sich die Slider der Koordinaten entsprechend ändern.
Hier wird für die variabel lange Liste der Töne ebenfalls eine Scoll-View verwendet um die Auswahl möglichst benutzerfreundlich zu gestalten. Für die Eingabe der Position werden zwei Methoden angeboten, sodass durch einen Winkel und Abstand eine schnelle, intuitive Positionsbestimmung möglich ist. Durch Eingabe der Koordinaten sollte es dennoch möglich sein, genauere, besser auf den Nutzer abgestimmte Tonquellen zu erzeugen.

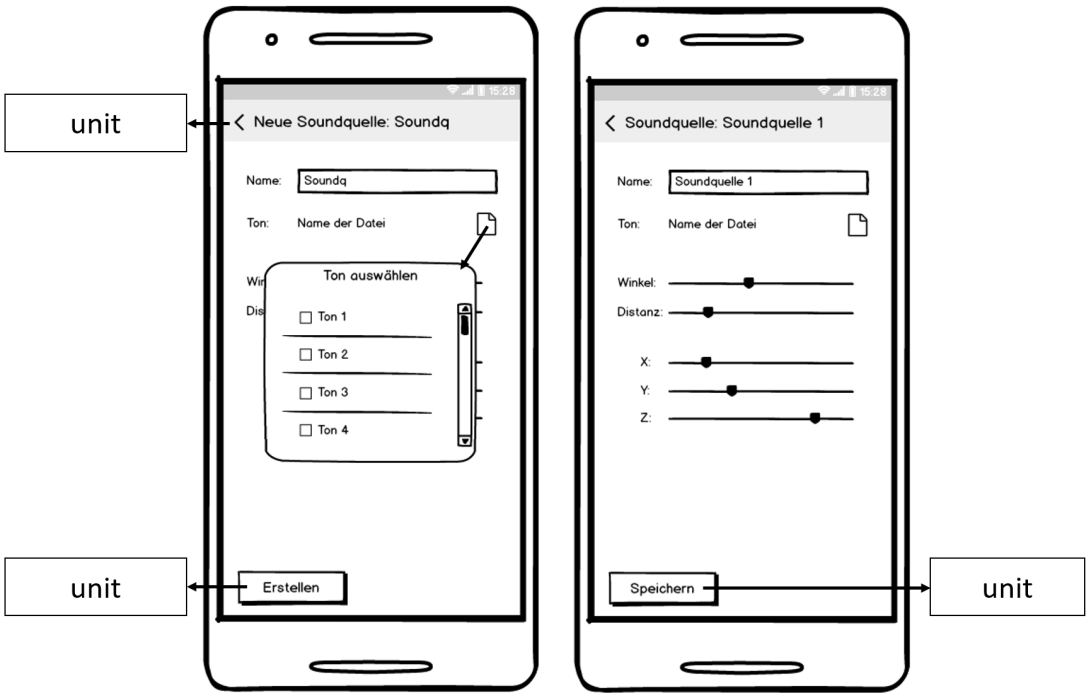

Abbildung 5.6: Seite: source

mode1

Abbildung 5.7 zeigt links die Seite *mode1*. Diese ist zu sehen sobald ein Spiel mit Modus 1 gestartet wurde. Über die Slider im unteren Bildschirmbereich kann die Ausrichtung der Pfeile innerhalb des Kreises verändert werden. Damit soll, wie in Abschnitt 5.2 erklärt, die Eingabe der vermuteten Positionen der Tonquellen vorgenommen werden. Die Unterscheidung der Tonquellen wird durch den Namen der Tonquellen in Verbindung mit Farben umgesetzt. Sind alle Pfeile wie gewünscht platziert, kann über den Button links oben die Ausrichtung bestätigt werden. Das erzielte Resultat wird in einem Pop-Up bekannt gegeben. Pop-Ups werden generell in Abbildung 5.9 beschrieben.

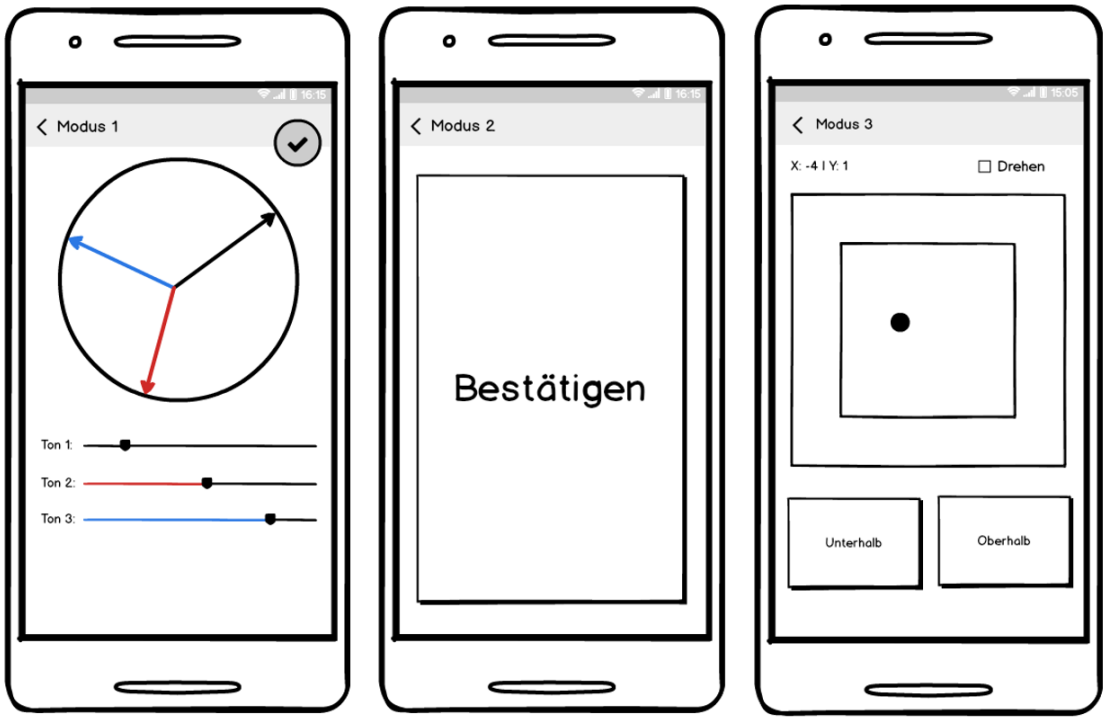

Abbildung 5.7: Seite: mode1, mode2 & mode3

mode2

Die mittlere Seite in Abbildung 5.7 zeigt die Oberfläche während einer Trainingseinheit mit Spielmodus 2. Da in diesem Spielmodus die relevante Interaktion aus Sensorik und auditivem Feedback besteht, ist die Oberfläche minimal gestaltet. Zudem soll dadurch

sichergestellt werden, dass der Nutzer keine zusätzliche mentale Belastung durch die Eingabe erfährt [31]. Die Oberfläche besteht lediglich aus einem Button, der gedrückt werden soll, sobald vermutet wird, direkt auf eine Tonquelle zu blicken (vgl. Abschnitt 5.2). Um die Konzentration des Nutzers möglichst auf das Gieren und Hören zu lenken, nimmt der Bestätigen Button einen Großteil der Bildschirmfläche ein, sodass nicht auf das Gerät geblickt werden muss, um ihn zu betätigen.

mode3

Nach dem Start einer Trainingseinheit mit Spielmodus 3 wird dem Nutzer die Seite *mode3* präsentiert (siehe Abbildung 5.7 rechts). Auch hier ist der wichtigste Teil der Interaktion durch die Sensorik zu bewerkstelligen. Um sich besser in der X-Z-Ebene zurechtzufinden wird dem Nutzer allerdings eine Karte vorgelegt. Diese besteht aus zwei Rechtecken. Innerhalb des äußeren Rechtecks kann sich der Hörer, welcher durch einen Kreis visualisiert ist, bewegen. Das innere Rechteck begrenzt die Fläche, in der Tonquellen zu finden sind. Des Weiteren werden die aktuellen Koordinaten des Hörers überhalb der Karte angezeigt, sodass feine Justierungen sichtbar werden. Die Checkbox *Drehen* ermöglicht es, zusätzlich den Hörer wie in Spielmodus 2 zu drehen. In der unteren Hälfte sind die Buttons zu finden, welche für die Bestimmung der Höhe einer Tonquelle verwendet werden (vgl. Abschnitt 5.2).

results

Die erzielten Resultate werden dem Nutzer wie in Abbildung 5.8 präsentiert. Die linke Spalte enthält immer den Namen der Tonquelle. Rechts daneben ist die erzielte Genauigkeit zu finden, die je nach Spielmodus unterschiedlich ermittelt wird (siehe Abschnitt 6.3). Für Ergebnisse des dritten Spielmodus wird eine weitere Spalte benötigt, die angibt, ob der Nutzer die horizontale Position der Tonquelle richtig bestimmt hat.

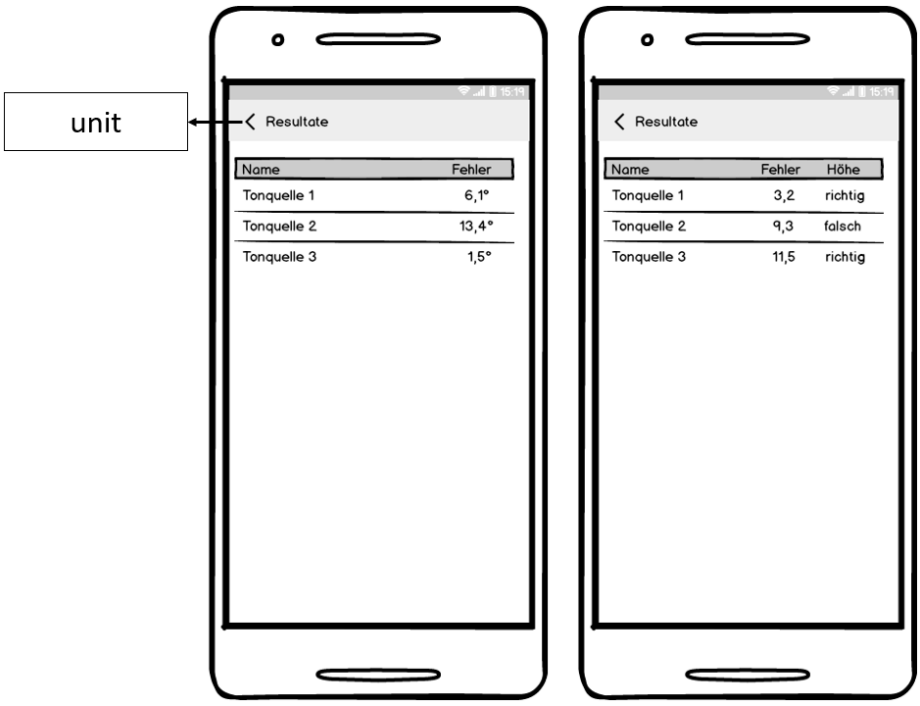

Abbildung 5.8: Seite: results

pop-ups

In den Mockups 5.3, 5.4 und 5.5 sind jeweils Löschen-Icons zu sehen. Diese Aktion erfordert eine Sicherheitsabfrage, welche durch Pop-Ups gelöst wird. Abbildung 5.9 zeigt links die Umsetzung eines solchen Pop-Ups anhand der *sound* Seite. Nach Bestätigung der Sicherheitsabfrage soll, wie rechts zu sehen, eine Rückmeldung in Form eines *Toasts* dem Nutzer präsentiert werden.

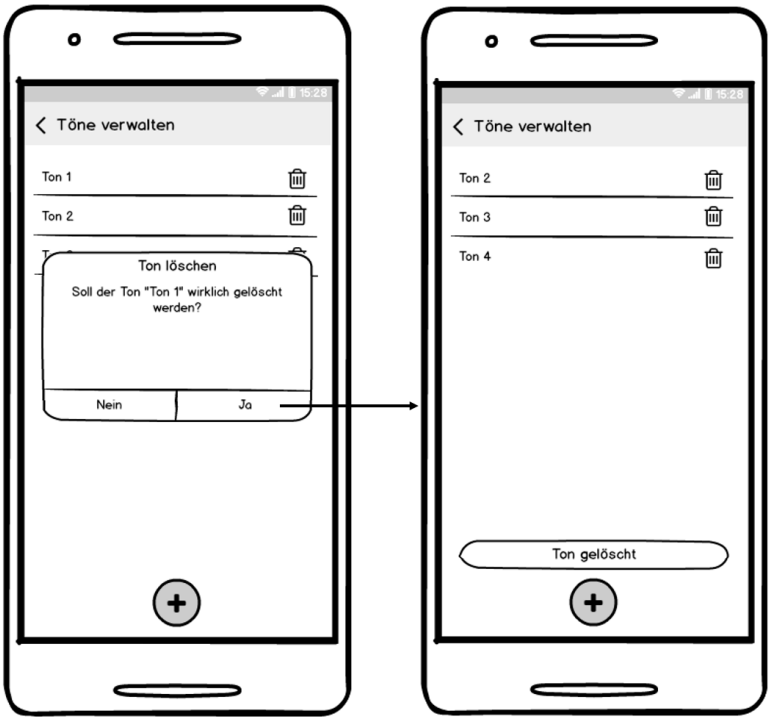

Abbildung 5.9: Beispiel eines Pop-ups

5.4 Datenmodell

Wichtige Aspekte des zu entwickelnden Datenmodells stellen die Leistung & Effizienz (NFA#02) sowie die Erweiterbarkeit (NFA#06) dar. Als eine mögliche Erweiterung wäre das Beziehen einer Trainingseinheit von einem Server denkbar. Ebenso könnten lokal erstellte Traininseinheiten hochgeladen und damit anderen Nutzern zur Verfügung gestellt werden.
Um solche Erweiterungen zu vereinfachen werden alle relevanten Daten einer Trainingseinheit innerhalb eines JSON-Objekts gespeichert. Darunter fallen, wie in Listing 5.1 zu sehen, unter anderem auch die Angaben aller in der Trainingseinheit vorhandener Tonquellen.

```
{
    id:int,
    name:String,
    mode:enum(1|2|3),
    sources:[
        {
           name:String,
           x:int,
           y:int,
           z:int,
           soundID:int,
           audioSource:AudioSource,
           isPlaying:bool
         }
    ]
}
```

Listing 5.1: Datenmodell einer Trainingseinheit

Dabei ist zu beachten, dass die Werte der Attribute *audioSource* und *isPlaying* erst beim Start einer Trainingseinheit initialisiert werden. Da die Datenmenge einer Tondatei und somit auch einer AudioSource der Web Audio API verhältnismäßig groß ist, würden andernfalls unnötige Ladezeiten geschaffen werden. Da auch die Angaben der Töne lediglich beim Start einer Trainingseinheit nötig sind, sollten sie ausgelagert werden. Sie werden daher über das Attribut *soundID* referenziert, welches auf ein Ton-Objekt (Listing 5.2) verweist.

```
{
    id:int,
    name:String,
    path:{
        rootPath:String,
        relativePath:String
    },
    type:enum(mp3|ogg|wav)
}
```

Listing 5.2: Datenmodell eines Tons

Beim Datenmodell eines Tons wurde das Attribut *path* im Hinblick auf die Verwendung innerhalb des Ionic 2 File-Plugins bereits aufgeteilt. Der erste Teil, *rootPath*, gibt dabei den Pfad des zu verwendenden Verzeichnisses an. Beispielsweise existieren unterschiedliche Verzeichnisse für App-spezifische Daten, Cache oder das Installationsverzeichnis der Anwendung. Alle vorhandenen Verzeichnisse können in der Dokumentation des File-Plugins [1] nachgelesen werden.

[1] Ionic 2 File Plugin: `http://ionicframework.com/docs/native/file/#instance-members`

6

Realisierung

Dieses Kapitel befasst sich mit der Realisierung der eben dargestellten Konzeption. Es werden Ausschnitte der Implementierung betrachtet, welche für die in Kapitel 4 definierten Anforderungen essentiell sind. Dabei wird durchgehend die in Abschnitt 5.3 konzipierte Benutzeroberfläche berücksichtigt. Zuerst widmet sich dieses Kapitel der Verwaltung der Töne. Daraufhin wird auf Trainingseinheiten und Tonquellen eingegangen, wobei der Fokus auf der Erstellung und Modifikation liegt. Danach wird die Implementierung vorgestellt, die für den Start und die Durchführung einer Trainingseinheit nötig ist. Erst werden dafür allgemeine Aspekte gezeigt, die unabhängig des gewählten Spielmodus ausgeführt werden müssen. Anschließend wird auf Besonderheiten für jeden der drei Spielmodi eingegangen.

6.1 Töne verwalten

Bei der Verwaltung der Töne wird grundsätzlich zwischen mitgelieferten und importierten Tönen unterschieden. Im Folgenden wird erst auf den Import und die Speicherung von Tönen eingegangen, welche bereits nach Start durch die Anwendung bereitgestellt werden. Daraufhin werden diese Aspekte anhand zu importierender Tondateien beleuchtet. Zuletzt wird beschrieben wie Töne wieder gelöscht werden können.

6.1.1 Mitgelieferte Töne

Beim ersten Start der Anwendung werden alle mitgelieferten Töne lokal gespeichert. Um herauszufinden, ob es sich um den ersten Start handelt, wird eine Variable lokal

gespeichert, die erst beim ersten Start initialisiert wird. Diese Prozedur ist in Listing 6.1 zu sehen und wird im Konstruktor der Seite *home* ausgeführt.

```
this.storage.get(keys.firstStart).then(_firstStart =>{
  if(!_firstStart){
    this.storage.set(keys.firstStart, "alreadyStartedOnce");
    this.loadGivenSounds();
  }
})
```

Listing 6.1: Bestimmung erster Start

Ist der erste Start erkannt, werden zuerst in der Methode `loadGivenSounds()` (Listing 6.2) die Dateinamen der mitgelieferten Töne bestimmt. Das Ionic 2 File-Plugin [1] bietet dafür die Methode `listDir()`, welche für jede Datei innerhalb des angegebenen Verzeichnisses einen *Cordova File Entry* [2] liefert. Das angegebene Verzeichnis setzt sich wie in Abschnitt 5.4 angekündigt aus einem *Base Filesystem* (im Folgenden *basePath* genannt) und einem dazu relativen Pfad zusammen.

```
private loadGivenSounds(){
  let allGivenSoundNames = [];
  let basePath = this.file.applicationDirectory;
  let relPath = "www/assets/sounds";
  this.file.listDir(basePath, relPath).then(_fileEntries =>{
    for(let fileEntry of _fileEntries){
      allGivenSoundNames.push(fileEntry.name);
    }
    this.saveGivenSounds(allGivenSoundNames);
  });
}
```

Listing 6.2: Erfassung mitgelieferter Töne

[1] Ionic 2 File Plugin: `http://ionicframework.com/docs/native/file/`
[2] Cordova File Entry: `https://cordova.apache.org/docs/en/2.4.0/cordova/file/fileentry/fileentry.html`

Nun wird für jeden der gefundenen Dateinamen in der Methode `saveGivenSounds()` (Listing 6.3) ein Tonobjekt erstellt. Das Datenmodell eines Tons ist in Listing 5.2 zu finden. Die ID für das Tonobjekt liefert die Methode `makeSoundID()`. Sie ist in Anhang A.1 zu finden. Um sicherzustellen, dass die erzeugte ID eindeutig ist, muss ein Vergleich der bisher vorhandenen IDs stattfinden. Da diese Vergleiche mit einem Speicherzugriff verbunden sind und daher vergleichsweise viel Zeit benötigen, kann der Speichervorgang der Töne aufgrund möglicher Schreib-Lese-Konflikte nicht asynchron ablaufen. Deshalb wird in der Methode lediglich der erste Eintrag der übergebenen Namen betrachtet. Ist für diesen Eintrag das Tonobjekt erzeugt und lokal gespeichert worden, wird die Prozedur erneut, ohne den ersten Eintrag, aufgerufen.
Die erzeugten Tonobjekte sind danach unter ihrer ID mit dem Präfix „Sound_ “ im lokalen Speicher zu finden.

```
1  private saveGivenSounds(_aGSN){
2    if(_aGSN.length == 0){
3      return;
4    }else{
5      this.makeSoundID(_soundID => {
6        let newSound = {
7          id: _soundID,
8          name: _aGSN[0],
9          path: {
10           "rootPath": constants.rootPathGS,
11           //"file:///android_asset/"
12           "relativePath": constants.relPathGS+_aGSN[0]
13           //"www/assets/sounds/*name.type*"
14         },
15         type: _aGSN[0].substring(_aGSN[0].lastIndexOf(".")+1)
16       };
17       this.storage.set(keys.prefixSound + _soundID,
18                        JSON.stringify(newSound))
19       .then(()=>{
```

```
20          _aGSN.splice(0,1); //Ersten Eintrag loeschen
21          this.saveGivenSounds(_aGSN);
22        });
23      });
24    }
25  }
```

Listing 6.3: Speicherung mitgelieferter Töne

6.1.2 Töne importieren

Um Töne aus dem internen oder externen Speicher des mobilen Geräts zu verwenden, können diese von dort importiert werden. Das Ionic 2 File Chooser Plugin [3] ermöglicht es, durch die Verzeichnisstruktur des Geräts zu navigieren und eine Datei auszuwählen. Um den base- und relativen Pfad der ausgewählten Datei zu bestimmen, sind Transformationen nötig. Das File Chooser Plugin liefert für die ausgewählte Datei einen *FileURI*. Dieser kann mithilfe eines weiteren Plugins, FilePath [4], in einen nativen Pfad umgewandelt werden. Das File Plugin kann schließlich aus dem nativen Pfad ein *File Entry* erstellen, der alle benötigten Informationen enthält.
Die ausgewählte Datei darf, wie in FA#02 definiert, nur dem Format .mp3, .wav oder .ogg angehören. Erfüllt die ausgewählte Datei diese Bedingung nicht, wird der Importvorgang abgebrochen und der Nutzer in Form eines *Toasts* darüber informiert. Andernfalls wird eine Kopie der ausgewählten Datei innerhalb des für die Anwendung vorgesehenen Verzeichnisses erstellt. Dadurch bleibt die Datei auch nach Änderungen im Ausgangsverzeichnis der Anwendung erhalten. Nach erfolgreichem Kopieren kann nun, wie bereits in Listing 6.3 beschrieben, das Tonobjekt erzeugt und lokal gespeichert werden. Die Abfolge des Imports ist verkürzt in Codeauszug 6.4 zu sehen.

[3] Ionic 2 File Chooser Plugin: `http://ionicframework.com/docs/native/file-chooser/`
[4] Ionic 2 File Path Plugin: `http://ionicframework.com/docs/native/file-path/`

```
1  //nFP:nativeFilePath; fE:fileEntry
2  //check PERMISSION.READ_EXTERNAL_STORAGE
3  this.fileChooser.open().then(_fileURI => {
4   this.filePath.resolveNativePath(_fileURI).then(_nFP => {
5    this.file.resolveLocalFilesystemUrl(_nFP).then(_fE =>{
6      let type = _fE.name.substring(_fE.name.lastIndexOf(".")+1);
7      if(type == "mp3" || type == "wav" || type == "ogg"){
8        this.file.copyFile(_fE.filesystem.root.nativeURL,
9        _fE.fullPath.substring(1, _fE.fullPath.length),
10       this.file.dataDirectory,
11       _fE.name).then(_res => {
12         //create and save soundObject
13       });
14     }else{
15       let toast = this.toastCtrl.create({
16         message: "Dateiformat wird nicht unterstuetzt, " +
17         "bitte nur .mp3, .wav oder .ogg Dateien auswaehlen",
18         duration: constants.longToast
19       });
20       toast.present();
21     }
22   });
23  });
24 });
```

Listing 6.4: Importieren von Tönen

Normalerweise sollte bereits eine Berechtigung für den Speicherzugriff gegeben sein, da diese direkt beim ersten Start der Anwendung für das Laden der mitgelieferten Töne eingefordert wird. Ist das Betriebssystem des mobilen Gerätes nicht Android in der Version 6.0 oder höher, so werden alle Rechte bereits bei der Installation gefordert. Dennoch könnte der Nutzer zwischenzeitlich die Berechtigungen geändert haben. Das File Choo-

ser Plugin fordert diese Berechtigung nicht ein, wodurch während der Transformationen ein Fehler aufgrund von fehlenden Berechtigungen auftreten könnte. Aus diesem Grund wird jedes Mal bevor der fileChooser geöffnet wird die Berechtigung geprüft und ggfs. erneut verlangt. Dafür kann das Plugin *Android Permissions* [5] verwendet werden.

6.1.3 Töne löschen

Ein Ton kann nur dann gelöscht werden, wenn er von keiner Tonquelle genutzt wird. Wird er in einer Tonquelle verwendet, so wird der Nutzer in Form eines Toasts darüber informiert.
Das Löschen von Tönen umfasst einerseits, den Eintrag des zugehörigen Tonobjekts aus dem lokalen Speicher zu entfernen, andererseits sollte die durch das Tonobjekt referenzierte Datei im Verzeichnis gelöscht werden. Wie in Codeauszug 6.5 zu sehen, ist dies mithilfe der entsprechenden Plugins leicht durchzuführen.
Anhand dieses Auszugs wird ferner die Realisierung von Pop-Ups gezeigt. Diese werden für Aktionen eingesetzt, die vor der Ausführung eine Sicherheitsabfrage benötigen. Ionic 2 liefert dafür einen *AlertController* Component [6]. Mit diesem können unter Angabe eines Titels, einer Nachricht sowie Buttons, Pop-Ups erstellt werden. Jedem Button kann dabei ein Text sowie ein handler zugeordnet werden, der je nach Benutzereingabe den entsprechenden Code ausführt.

```
1 private deleteSound(_sound){
2   //check if sound used
3   let confirm = this.alertCtrl.create({
4     title: 'Ton loeschen?',
5     message: 'Soll der Ton "'+_sound.name +
6     '" wirklich geloescht werden?',
7     buttons: [
8       //no Button
```

[5] Ionic 2 Android Permissions Plugin: http://ionicframework.com/docs/native/android-permissions/
[6] Ionic 2 Alert Controller: http://ionicframework.com/docs/api/components/alert/AlertController/

```
9         {
10          text: 'Ja',
11          handler: () => {
12            this.storage.remove(keys.prefixSound+_sound.id);
13            this.file.removeFile(_sound.path.rootPath,
14                                 _sound.path.relativePath);
15            //remove from View and confirmation Toast
16          }
17        }
18      ]
19    });
20    confirm.present();
21  }
```

Listing 6.5: Löschen von Tönen

6.2 Trainingseinheiten und Tonquellen

Dieser Abschnitt behandelt die Realisierung von Trainingseinheiten und Tonquellen. Es wird beschrieben, wie diese erstellt und verändert werden können. Da der Nutzer dafür viel mit der Oberfläche interagieren muss, wird in diesem Absatz unter anderem der Gebrauch einiger von Ionic 2 bereitgestellter Components erläutert.

6.2.1 Trainingseinheiten

Wie im Datenmodell festgelegt (siehe Listing 5.1) besteht eine Trainingseinheit aus einer ID, einem Namen und Modus sowie beliebig vieler Tonquellen. Für die Erstellung und Änderung von Trainingseinheiten ist die Seite *unit* zuständig. Sie wird immer mit zwei Parametern auf den NavigationStack gepusht (siehe Listing 6.6). Diese Parameter werden von der *unit* Seite ausgelesen und als Klassenvariable initialisiert. Deren Verwendung als Klassenvariable hat den Vorteil, dass nun Eingabeelemente aus der View direkt

auf ihnen operieren können. Beispielsweise kann zwischen einem Text-Eingabefeld und einer Variable dadurch ein two-way-databinding erzeugt werden. Dafür wird die von Angular bereitgestellte Directive **ngModel* verwendet. Auf diese Weise werden die Eingaben für den Namen und Modus der Trainingseinheit umgesetzt.

```
1  private goToUnitPage(_unit){
2    this.navCtrl.push(UnitPage, {"unit" : _unit, "new":false});
3  }
4
5  //in unitPage constructor
6  this.unit = navParams.get("unit");
7  this.createNew = navParams.get("new");
8
9  //example of two-way-databinding
10 <ion-input type="text" [(ngModel)]="unit.name"></ion-input>
```

Listing 6.6: Beispiel NavigationStack mit Parametern

Da keine Beschränkung der Anzahl an Tonquellen pro Trainingseinheit besteht, wurde für ihre Darstellung in Abschnitt 5.3 die Verwendung einer Scroll-View empfohlen. Das grundlegende Layout von Ionic 2 Seiten ist bereits als ScrollView implementiert [7]. Es soll darauf jedoch eine weitere ScrollView gelegt werden, die nur einen bestimmten Bereich der gesamten Seite einnimmt. Ionic 2 stellt dafür einen *ion-scroll* Component bereit. Innerhalb davon wird in einer Liste mit einer weiteren Angular Directive **ngFor* für jede vorhandene Tonquelle ein Listenelement erzeugt. Dies ermöglicht, wie in Codeauszug 6.7 zu sehen, eine kompakte Implementierung der geforderten Oberfläche.

```
1  <ion-scroll scrollY="true">
2    <ion-list>
3      <ion-item *ngFor="let source of unit.sources;let i=index">
4        <ion-icon (click)="deleteSrc(i)" name="trash"></ion-icon>
5        <p (click)="editSrc(i)">{{source.name}}</p>
6      </ion-item>
```

[7] Ion-Content: https://ionicframework.com/docs/api/components/content/Content/

```
7       <ion-item (click)="newSrc()">
8         <ion-icon name="add-circle"></ion-icon>
9       </ion-item>
10    </ion-list>
11  </ion-scroll>
```

Listing 6.7: Scroll View der Tonquellen innerhalb einer Trainingseinheit

6.2.2 Tonquellen

Bei Tonquellen ist die Eingabe für den Namen analog zu dem einer Trainingseinheit implementiert. Auch die Navigation zur *source* Seite, welche für die Erstellung und Modifikation von Tonquellen zuständig ist, läuft nach dem gleichen Schema ab. Für die Auswahl des zu verwendenden Geräusches wird erneut der *AlertController* verwendet. Diesem werden jedoch, anders als in Listing 6.5 zu sehen, nicht nur Buttons und Texte hinzugefügt. Er wird für jedes lokal gespeicherte Geräusch um eine neue Eingabe in Form von Radio Buttons erweitert. Codeauszug 6.8 zeigt dies anhand eines verkürzten Beispiels.

```
1  let chooser = this.alertCtrl.create();
2  chooser.setTitle('Ton fuer Tonquelle');
3
4  //foreach sound of local saved sounds
5  chooser.addInput({
6    type: 'radio',
7    label: sound.name,
8    value: sound.id,
9    checked: false //is true for exactly one input if in editing
10 });
```

Listing 6.8: Auswahl des Geräusches einer Tonquelle

Für die Bestimmung der Position einer Tonquelle werden dem Nutzer zwei Möglichkeiten angeboten. Wie in Abschnitt 5.3 beschrieben kann die Eingabe dabei einerseits über einen Winkel und einen Abstand zum Hörer erfolgen. Andererseits können die exakten dreidimensionalen Koordinaten der Tonquelle angegeben werden. Die Eingaben erfolgen durch einen Slider, der von Ionic 2 als *ion-range* Component geliefert wird (siehe Listing 6.9). Diesem kann ein Minimum und Maxium zugeteilt werden. Im Falle des Winkels ist dies z.B. 0 und 359. Der ausgewählte Wert wird erneut per **ngModel* mit Klassenvariablen verknüpft. Um zu ermöglichen, dass ein „Live Update " der Werte bei Änderung eines Sliders zu sehen ist, wird bei jedem Slider auf das Angular Event *ngModelChange* gehört. Dieses feuert, sobald der Nutzer den Slider verschiebt und triggert den Start der angegebenen Methode. Bei Events des Winkels oder Distanz Sliders wird die Methode `inputRangeChanged()` aufgerufen. Diese berechnet neue X und Z Positionen und ändert die entsprechenden Werte darauf. Da die anderen Slider durch die *ngModel* Directive an die Werte gekoppelt sind, ändern sie sich entsprechend auch in der View. In umgekehrter Reihenfolge ist die Methode `inputCoordChanged()` für die Berechnung des neuen Abstands und Winkels nach einer Änderung der X,Y oder Z- Koordinate zuständig. Für die Berechnungen werden einfache mathematische Funktionen benutzt. Lediglich die Bestimmung des Winkels, welchen zwei gegebene Punkte (in diesem Fall die x und z Koordinate der Tonquelle) zum Ursprung bilden, muss aufwändiger berechnet werden. Die dafür benötigten Fallunterscheidungen je nach zu betrachtendem Quadrant werden durch die, von Javascript gestellte, `Math.atan2()` Funktion abgenommen.

```
1 <ion-range (ngModelChange)="inputRangeChanged()"
2            [(ngModel)]="angle" min="0" max="359">
3 </ion-range>
4
5 private inputRangeChanged(){
6   this.src.x = Math.cos(this.angle*Math.PI/180)*this.distance;
7   this.src.z = Math.sin(this.angle*Math.PI/180)*this.distance;
8 }
9
```

```
10 private inputCoordChanged(){
11     let angleRadians = Math.atan2(this.src.z,this.src.x);
12     this.angle = (360+(angleRadians*180/Math.PI))%360;
13     this.distance = Math.sqrt(Math.pow(this.src.x,2)+
14                               Math.pow(1-this.src.y,2)+
15                               Math.pow(this.src.z,2));
16 }
```

Listing 6.9: Positionierung einer Tonquelle

6.3 Start einer Trainingseinheit

Um eine Trainingseinheit zu starten, müssen mehrere Schritte durchlaufen werden. Dieser Absatz erklärt vorerst die Implementierung der relevanten Aspekte, welche unabhängig vom gewählten Spielmodus immer durchgeführt werden müssen. Anschließend werden Besonderheiten der einzelnen Spielmodi aufgezeigt.

6.3.1 Allgemeines

Der erste Schritt beim Start einer Trainingseinheit besteht darin, einen *AudioContext* (siehe Abschnitt 3.2.1) mithilfe der Web Audio API zu erstellen. Wie in Codeauszug 6.10 zu sehen, kann der Hörer dem AudioContext entnommen werden. Wichtig zu beachten ist, dass der erstellte AudioContext wieder geschlossen wird, sobald die Seite nicht mehr aktiv ist. Ionic 2 bietet dafür eine Lifecycle-Methode `ionViewWillLeave()` an, in der dies vorgenommen werden kann. Das Schließen des AudioContext ist unter anderem deshalb notwendig, da maximal sechs AudioContext gleichzeitig geöffnet sein können.

```
1 this.audioContext = new ((<any>window).AudioContext ||
2                          (<any>window).webkitAudioContext)();
3 this.audioListener = this.audioContext.listener;
4
```

```
ionViewWillLeave(){
  this.audioContext.close();
}

for(let i=0;i<this.unit.sources.length;i++){
  this.loadSound(i);
}
```

Listing 6.10: Positionierung einer Tonquelle

Ist der AudioContext initialisiert, kann für jede vorhandene Tonquelle der Trainingseinheit das entsprechende Geräusch geladen werden. Dies geschieht innerhalb der Methode `loadSound()` (Listing 6.11). Dazu wird, nachdem die benötigten Daten des entsprechenden Tonobjekts geladen wurden, mithilfe des File Plugins der Inhalt der Datei in einen ArrayBuffer geschrieben. Wie später zu sehen sein wird, kann die WebAudioAPI die Inhalte des ArrayBuffers dekodieren und in ein abspielbares Format umwandeln.

```
private loadSound(_indexSource){
  //load soundObject from storage
  this.storage.get(keys.prefixSound+
                   this.unit.sources[_indexSource].soundID)
  .then(_sound =>{
     let sound = JSON.parse(_sound);
     //get content of file as arrayBuffer
     this.file.readAsArrayBuffer(sound.path.rootPath,
                                 sound.path.relativePath)
     .then(_arrayBuffer =>{
       this.positionSource(_indexSource, _arrayBuffer);
     });
  });
}
```

Listing 6.11: Laden des Geräusches einer Tonquelle

Die Methode `positionSource` (Listing 6.12) wird für den Vorgang der Erstellung und Spezifizierung eines Panners (siehe Abschnitt 3.2.2) verwendet. Vorerst muss der Inhalt des mitgegebenen ArrayBuffers dekodiert werden. Daraufhin kann mit der `createPanner()` Methode des AudioContext ein neuer Panner-Knoten erstellt werden. Dieser kann durch viele Einstellungen[8] genauer spezifiziert werden. Für diese Applikation ist vor allem die Angabe des *panningModel* von Bedeutung. Dieses wird im Hinblick auf die in Abschnitt 3.1.4 erklärte Funktion auf „HRTF“ gesetzt. Des Weiteren wird die Position des panners auf die bei der Erstellung der Tonquelle angegebenen Koordinaten gesetzt. Die *coneInnerAngle* Eigenschaft gibt an, in welchem Winkel der Tonquelle keine Reduktion der Lautstärke stattfindet. Diese wird auf 360 gesetzt, da der Ton in jede Richtung gleich laut ausgegeben werden soll. Jedoch soll der Ton je nach Distanz zum Hörer lauter bzw. leiser werden. Dafür kann das *distanceModel* verändert werden. Dieses berechnet die Lautstärke abhängig von der Distanz und weiteren Einstellungen des Panners. Diese wurden in dieser Applikation alle auf den Standardeinstellungen gelassen und werden daher nicht aufgelistet.
Nachdem der Panner erstellt und spezifiziert wurde, wird ein Knoten benötigt, der den Inhalt der Tondatei enthält. Mit `createBufferSource()` kann durch den AudioContext ein solcher Erzeugerknoten generiert werden. Diesem wird der dekodierte Inhalt über das *buffer* Attribut übergeben. Daraufhin findet die Verknüpfung gemäß der in Abschnitt 3.2.1 beschriebenen Verarbeitungskette statt. Der erstellte Erzeugerknoten wird schließlich innerhalb der Tonquelle gespeichert. Dadurch kann er von dort aus erreicht werden, um beispielsweise den Ton abzuspielen oder zu pausieren.

```
1 private positionSource(_idx, _buffer){
2   let posX = this.unit.sources[_idx].x;
3   let posY = this.unit.sources[_idx].y;
4   let posZ = this.unit.sources[_idx].z;
5   this.audioContext.decodeAudioData(_buffer)
6   .then( _soundContent => {
7     let panner = this.audioContext.createPanner();
8     panner.panningModel = "HRTF";
```

[8] Web Audio API Panner: `https://developer.mozilla.org/de/docs/Web/API/PannerNode`

```
9     panner.setPosition(posX,posY,posZ);
10    panner.coneInnerAngle = 360;
11    panner.distanceModel = 'inverse';
12    let audioSource = this.audioContext.createBufferSource();
13    audioSource.buffer = _soundContent;
14    audioSource.connect(panner);
15    panner.connect(this.audioContext.destination);
16    this.unit.sources[_idx].audioSource = audioSource;
17    this.checkForAllSourcesLoaded();
18  });
19 }
20
21 //View example
22 <span *ngIf="!readyToStart">
23   Trainingseinheit wird geladen...
24 </span>
25 <ion-icon *ngIf="!isRunning && readyToStart" name="play"
26           (click)="start()">
27 </ion-icon>
```

Listing 6.12: Positionieren des Geräusches einer Tonquelle

Nachdem die Töne aller Tonquellen geladen sind, ist `checkForAllSourcesLoaded()` dafür verantwortlich, eine Flag *readyToStart* zu setzen. Darüber kann in der View dem Nutzer der Status des Ladevorgangs mitgeteilt werden, indem man sich der Directive **ngIf* bedient. Auf diese Weise wird dem Nutzer auch der Start Button der Trainingseinheit präsentiert, welcher die Methode `start()` triggert. Sie iteriert unter anderem über alle Tonquellen und startet die Wiedergabe der Töne in Dauerschleife. Je nach Spielmodus werden zusätzliche Schritte innerhalb der `start()` Methode vorgenommen, wie in den folgenden Abschnitten zu sehen sein wird.

Die letzte Gemeinsamkeit besteht in der Präsentation der erzielten Ergebnisse. Nach Berechnung der Genauigkeit werden die Abweichungen an die Seite *results* übergeben, die für deren Darstellung zuständig ist.

6.3.2 Modus 1

Da in Spielmodus 1 der Hörer nicht verändert werden kann (siehe 5.2), muss lediglich gewährleistet werden, dass dieser anfangs richtig ausgerichtet ist. Standardmäßig würde er, laut dem vorgestellten Koordinatensystem (Abbildung 5.1), nach „hinten“, also im Winkel von 180 blicken. Er wird daher beim Start neu positioniert, indem seine Ausrichtung wie folgt verändert wird: `audioListener.setOrientation(1,0,0,0,1,0);` (siehe Absatz 3.2.2).
Die Darstellung der Auswahl wurde mithilfe von svg realisiert. Innerhalb des *svg-Containers* ist für jede Tonquelle eine Linie erzeugt worden. Zudem wird für jede Tonquelle ein Slider bereitgestellt, der die Eingabe des jeweiligen vermuteten Winkels zulässt. Analog zu den Slidern in Codebeispiel 6.9 wird bei Veränderung des Sliders durch ein event eine Methode aufgerufen, die die entsprechende Linie innerhalb des *svg-Containers* rotiert. Dadurch bekommt der Nutzer sofort visuelles Feedback seiner Eingaben. Um die Verbindung zwischen Slider und Linie eindeutig zu gestalten, wird auf unterschiedliche Farben zurückgegriffen, weshalb ein vorgefertigtes Farbschema implementiert wurde. Dies besteht, um eine möglichst gute Unterscheidung zu gewährleisten, aus fünf ausgewählten Farben. Daher ist bei diesem Spielmodus die Anzahl der maximal möglichen Tonquellen auf fünf begrenzt.

```
1 <svg width="250" height="250">
2   <circle cx="125" cy="125" r="120"/>
3   <line x1="125" y1="125" x2="125" y2="5" style="..."/>
4 </svg
5 //at change event from slider
6 line.setAttribute("transform", "rotate("+ _angle +"125 125)");
```

Listing 6.13: Spielmodus 1 Eingabe

Hat der Nutzer alle Linien wie gewünscht platziert, kann er seine Eingabe über die Betätigung eines Buttons auswerten lassen. Daraufhin wird die Abweichung für jede Tonquelle berechnet. Zudem wird die Wiedergabe aller Töne gestoppt und der Nutzer auf die Seite *results* weitergeleitet, auf der er seine Ergebnisse einsehen kann. Die Berechnung der Genauigkeit für den Spielmodus 1 ist in Anhang A.2 zu finden.

6.3.3 Modus 2

In Modus 2 wird der Listener durch Drehungen des mobilen Geräts entsprechend ausgerichtet. Um die Drehung des Geräts zu bestimmen wird das Ionic 2 Device Orientation Plugin[9] verwendet. Da während der gesamten Durchführung die Ausrichtung benötigt wird, kann die Ausrichtung durch eine subscription des Plugins in angegebenen Zeitintervallen erhalten werden (Listing 6.14). Dieses Intervall wird der subscription als *frequency* Parameter übergeben. Nach jedem erhaltenen Update der Ausrichtung wird `orientationChanged()` aufgerufen. Die Methode berechnet anhand des erhaltenen Winkels die neue Ausrichtung des Hörers und updatet diese entsprechend. Ebenso wie der AudioContext sollte die subscription abgebrochen werden, sobald die Seite verlassen wird.

```
1  //init subscription
2  this.orientationSub = this.deviceOrientation.watchHeading(
3    {frequency:constants.frequencyDeviceOrientation})
4    .subscribe((_orientation)=> {
5      this.trueHeading = _orientation.trueHeading;
6      this.orientationChanged();
7    });
8
9  //handle subscription data
10 private orientationChanged(){
11   let radiant = (this.trueHeading/180*Math.PI);
```

[9] Ionic 2 Device Orientation Plugin: http://ionicframework.com/docs/native/device-orientation/

```
12    let v1 = Math.cos(radiant);
13    let v3 = Math.sin(radiant);
14    this.audioListener.setOrientation(v1, 0, v3, 0, 1, 0);
15  }
16
17  ionViewWillLeave(){
18    this.orientationSubscription.unsubscribe();
19  }
```

Listing 6.14: Spielmodus 2 Realisierung Kompass

Die Berechnung der Genauigkeit erfolgt in ähnlicher Weise wie in Modus 1. Lediglich der Bezugspunkt muss geändert werden. Statt Linien gibt in diesem Modus der Winkel, welcher durch den Kompass bestimmt wird, die vermutete Position wieder.
Um die schon bestimmten und noch übrigen Tonquellen unterscheiden zu können, soll bei bereits lokalisierten Tonquellen die Wiedergabe gestoppt werden. Sobald sich der Nutzer in seiner gewünschte Ausrichtung befindet und eine Tonquelle durch Klick auf den „Bestätigen“ Button auswerten lassen will, wird für jede noch aktive Tonquelle die Genauigkeit bestimmt. Die Tonquelle mit minimalem Fehler wird daraufhin als ausgewählte vermutet, dessen Wiedergabe gestoppt und das Ergebnis gespeichert. Sind alle Tonquellen lokalisiert worden, wird automatisch auf die *results* Seite geleitet.

6.3.4 Modus 3

Für die in Modus 3 vorgesehene Navigation durch Neigung wird das Ionic 2 Device Motion Plugin[10] verwendet. Bei diesem wird innerhalb der `start()` Methode analog zum Device Orientation Plugin eine subscription eingerichtet. Für die erhaltenen Daten wurde, wie in Listing 6.15 zu sehen, die Methode `motionChanged()` implementiert. Diese extrahiert zuerst die gelieferten Beschleunigungen für die X- und Y-Achse. Damit der Nutzer auch an einer Position stehen bleiben kann, wurde eine Schwelle implementiert, die überschritten werden muss, um eine Positionsänderung des Hörers zu erreichen. Ist

[10] Ionic 2 Device Motion Plugin: `http://ionicframework.com/docs/native/device-motion/`

diese Schwelle überschritten wird je nach Stärke der Neigung eine Geschwindigkeit für die jeweilige Richtung berechnet. Diese wird zudem mit der Frequenz der erhaltenen Daten verrechnet. Daraufhin kann die Position des Hörers neu bestimmt und gesetzt werden. Falls die Checkbox „Drehen“ aktiv ist, wird zudem, analog zu Modus 2, der Hörer entsprechend der Ausrichtung des mobilen Geräts gedreht. Für das visuelle Feedback wurde ein svg-Container verwendet, der wie in Abbildung 5.7 gezeigt aus zwei Rechtecken besteht, die eine Begrenzung aller Tonquellen aufzeigen. Innerhalb des Containers befindet sich zudem ein Kreis, der die Position des Hörers veranschaulicht. Die Position dieses Kreises wird ebenfalls nach jeder Positionsbestimmung aktualisiert.

```
1  //handle subscription data
2  private motionChanged(_acceleration){
3    let accX = _acceleration.x;
4    let accY = _acceleration.y;
5    let speedX, speedY = 0;
6
7    if(Math.abs(accX)>constants.thresholdX){
8      if(accX < 0){
9        speedX = constants.facSpeed*(accX+constants.thresholdX);
10     }else{
11       speedX = constants.facSpeed*(accX-constants.thresholdX);
12     }
13    }
14    //same for accY
15
16   speedX = speedX/constants.frequencyDeviceMotion;
17   speedY = speedY/constants.frequencyDeviceMotion;
18   this.playerPosition.x += speedX;
19   this.playerPosition.y += speedY;
20   this.audioListener.setPosition(this.playerPosition.x,
21                                  0,
22                                  this.playerPosition.y);
```

```
23   if(this.turning){
24     //also turn listener like in mode 2
25   }
26   let player = document.getElementById("playerCircle");
27   player.setAttribute("transform", "translate("
28                       +this.playerPosition.x*2+","
29                       +this.playerPosition.y*2+")");
```

Listing 6.15: Spielmodus 3 Realisierung Neigungssensor

Für die Bestimmung der Genauigkeit müssen in diesem Modus die Position der Tonquellen und des Hörers verglichen werden. Dies wird durch Anwendung der Formel $sqrt((playerPosition.x - source.x)^2) + ((playerPosition.y - source.z)^2))$ erreicht.
Da sich der Hörer immer auf der gleichen Höhe befindet, ist ein Vergleich der Y-Koordinate der Tonquelle und die des Hörers ausreichend, um herauszufinden, ob der Nutzer die Höhe richtig erkannt hat.

7

Anforderungsabgleich

Die realisierte Anwendung wird im Folgenden hinsichtlich der gestellten Anforderungen aus Kapitel 4 evaluiert. Zuerst werden funktionale, daraufhin nichtfunktionale Anforderungen betrachtet.

7.1 Funktionale Anforderungen

FA#01 Räumliche Töne simulieren
Die Anwendung kann mithilfe des Panner-Listener Konzepts der Web Audio API Töne im dreidimensionalen Raum simulieren. Diese werden durch einen Ausgabeknoten der API an die Audioschnittstelle des mobilen Gerätes übergeben.

FA#02 Töne verwalten
Durch die Seite *sound* kann der Nutzer alle vorhandenen Töne einsehen und löschen. Zudem wird durch einen *fileChooser* ermöglicht, auf dem Gerät gespeicherte Tondateien zu importieren.

FA#03 Trainingseinheiten erstellen
Der Nutzer kann unter Angabe eines Namens und eines Spielmodus eine Trainingseinheit erstellen. Diese kann in Spielmodus zwei und drei beliebig viele Tonquellen enthalten. Für Spielmodus eins wurde die Anzahl der maximal vorhandenen Tonquellen auf fünf beschränkt um eine übersichtliche Oberfläche zu behalten (siehe Absatz 6.3.2).

FA#04 Trainingseinheit bearbeiten
Jede Trainingseinheit kann umbenannt und gelöscht werden. Trainingseinheiten mit Spielmodus eins können immer auf Modus zwei geändert werden. Die Rückrichtung ist allerdings, bedingt durch die eben genannte Beschränkung, nur bei Trainingseinheiten mit maximal fünf Tonquellen möglich.

FA#05 Geräuschquelle erstellen
Mit der Seite *source* können Tonquellen erzeugt werden. Die Eingabe der gewünschten Position kann unter anderem durch eine dreidimensionale Koordinate erfolgen. Des weiteren kann das Geräusch der Tonquelle aus allen vorhandenen Tönen gewählt werden.

FA#06 Geräuschquelle bearbeiten
Innerhalb der *unit* Seite können Tonquellen aus einer Trainingseinheit entfernt werden. Die Bearbeitung ist mithilfe der *source* Seite möglich.

FA#07 Trainingseinheit starten
Sofern die Trainingseinheit mindestens eine Tonquelle besitzt, kann diese gestartet werden. Danach sind die jeweiligen *mode* Seiten für den Ablauf verantwortlich.

FA#08 Trainingseinheit verlassen
Jede *mode* Seite kann über den „Zurück“ Button der Navigationsleiste verlassen werden.

FA#09 Steuerung mithilfe der Sensorik
Die Anwendung kann mithilfe von nativen Ionic 2 Plugins die benötigte Sensorik (Kompass & Neigungssensor) ansprechen, deren Daten auswerten und entsprechend auf die Anwendung übertragen.

FA#10 Bestimmung der Genauigkeit
Für jeden Modus ist es durch verschiedene Berechnungen möglich, die Genauigkeit der Nutzereingaben zu ermitteln.

FA#11 Rückmeldung der erzielten Leistung
Für die Einsicht der erzielten Ergebnisse wurde eine separate Seite implementiert. Diese zeigt in Tabellenform die Resultate einer abgeschlossenen Trainingseinheit.

7.2 Nichtfunktionale Anforderungen

NFA#01 Verwendete Technologien
Die Anwendung wurde mit den geforderten Frameworks erstellt.

NFA#02 Leistung & Effizienz
Es wurde stets darauf geachtet, lediglich die benötigten Daten in den Speicher zu laden. Beispielsweise werden Töne erst beim Start einer Trainingseinheit geladen sowie nicht benötigte Daten wieder freigegeben.

NFA#03 Benutzbarkeit
Durch den Einsatz teils großer Bedienelemente wird eine unbeschwerte Benutzbarkeit gewährleistet. Die einzige, vergleichsweise längere Wartezeit entsteht bei der Vorbereitung einer Trainingseinheit für den Start. Dabei ist die Größe der zu ladenden Audiodateien ausschlaggebend.

NFA#04 Zuverlässigkeit
Für den Aspekt der Zuverlässigkeit wurde während der Entwicklung auf stetige begleitende Tests geachtet. Nutzereingaben werden immer überprüft und Fehler durch geeignete Bedienelemente, wie z.B. Slider oder ButtonGroups im Vorhinein vermieden.

NFA#05 Nachvollziehbarkeit
Falls der Nutzer falsche Eingaben getätigt oder Angaben vergessen hat, erhält er sofortige Rückmeldung. Ebenso ist beispielsweise bei der Eingabe während Spielmodus eins oder bei Veränderung der Slider für die Positionierung einer Tonquelle darauf geachtet worden, dem Nutzer visuell seine Änderungen zu präsentieren.

NFA#06 Wartbarkeit & Erweiterbarkeit

Die Lösung funktional gleicher Aufgaben wurde so implementiert, dass sie an anderen Stellen leicht wiederverwendbar sind. Auch für die Benutzeroberfläche wurde beispielsweise die Erstellung und Änderung von Trainingseinheiten sowie Tonquellen durch eine Seite gehandhabt, da beide Aktionen nahezu die gleichen Eingaben benötigen. Im Hinblick auf die Erweiterbarkeit wurden bereits bei Erstellung des Datenmodells mögliche Erweiterungen mit einbezogen.

NFA#07 Motivation

Diese Anforderung ist seitens der Anwendung durch die Erfüllung von **NFA#03**, **NFA#04**, **NFA#05** erreicht. Es wurde zudem darauf geachtet, dass Inhalte speziell auf einen Nutzer abgestimmt werden können.

8

Zusammenfassung & Ausblick

Im Rahmen dieser Arbeit wurde eine mobile Applikation erstellt, die es ermöglicht, die auditive Lokalisationsfähigkeit eines Menschen zu messen. Besonders im Fokus standen dabei Tinnitus Patienten, welche durch diese Anwendung eine kurzzeitige Milderung der Beschwerden erfahren könnten. Um Tinnitus richtig einordnen zu können, wurde zu Beginn ein kurzer Überblick des Symptoms gegeben. Daraufhin wurde der Bezug zu räumlichem Hören hergestellt und technische Grundlagen gelegt. Für die Umsetzung der Anwendung wurden erst genaue Anforderungen definiert. Daraufhin wurde aus den Anforderungen ein Konzept entwickelt, das unter anderem geeignete Oberflächen enthält. Anschließend wurden die vorgehenden theoretischen Überlegungen genutzt, um die Anwendung mithilfe des hybriden Frameworks Ionic 2 zu erstellen. Zum Schluss fand eine Evaluation der realisierten Anwendung statt, indem sie mit den gestellten Anforderungen verglichen wurde.

8.1 Ausblick

Für die entwickelte Anwendung sind mehrere Erweiterungen denkbar. Es könnten beispielsweise die erzielten Leistungen umfangreicher erfasst werden indem die benötigte Zeit für eine Trainingseinheit oder sogar für eine spezielle Tonquelle erfasst werden würde. Dies wäre sehr leicht umzusetzen, da jede Trainingseinheit über Methoden verfügt, die für den Start und das Ende verantwortlich sind. Neben ausführlicheren Resultaten ist es ebenfalls denkbar, innerhalb der Anwendung eine Komponente zu realisieren, die die erzielten Resultate über einen bestimmten Zeitraum protokolliert.

Für weitere Informationen über den Nutzer könnten Fragebögen eingesetzt werden [32]. Mithilfe dieser Daten wäre es denkbar, dass sich die Anwendung automatisch an den Nutzer anpasst. Um Fragebögen auf mobilen Geräten zu realisieren wurde bereits eine Vielzahl an Frameworks und Konzepten entwickelt [33] [34] [35] [36] [37].

Fernen könnte eine Serververbindung eingerichtet werden, von der komplette Trainingseinheiten oder bestimmte Töne bezogen werden können. Auch der Upload selbst erstellter Trainingseinheiten könnte darüber stattfinden. Optimal wäre es dabei, wenn die Töne, die bereits auf dem Server vorhanden sind, mit den Tönen der Anwendung synchronisert werden, da der Up- und Download für Tondateien unter Umständen viele Ressourcen benötigt.

Abgesehen von technischen Erweiterungen wäre es interessant eine Studie mithilfe der Anwendung durchzuführen. Dabei könnte einerseits herausgefunden werden, ob und in welchem Maß Tinnitus-Patienten im räumlichen Hören eingeschränkt sind und inwieweit sich die Einschränkungen durch Training mit der Anwendung reduzieren lassen. Andererseits wäre ein ausführlicher Vergleich der angebotenen Modi denkbar. Durch einen Vergleich von Modus eins und zwei könnte bestimmt werden, wie wichtig es für die Lokalisationsfähigkeit ist, den Kopf zu drehen. Des Weiteren könnte evaluiert werden, welche Frequenzbereiche für welche Probanden bestmögliche Ergebnisse liefern.

All diese Möglichkeiten zeigen auf, wie viele neue Chancen sich durch die technische Entwicklung der letzten Jahre ergeben. Gerade im Bereich der Medizin wäre es wünschenswert, dass Patienten durch die neu entstehenden Mittel schneller und erfolgreicher geholfen werden kann.

Literaturverzeichnis

[1] Streppel, M., Walger, M., von Wedel, H., Gaber, E.: Heft 29 - Hörstörungen und Tinnitus. Robert Koch Institut, aus der Reihe Gesundheitsberichterstattung des Bundes (2006)

[2] Schickler, M., Pryss, R., Schobel, J., Schlee, W., Probst, T., Reichert, M.: Towards Flexible Remote Therapeutic Interventions. In: 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017), IEEE Computer Society Press (2017)

[3] Schickler, M., Schobel, J., Pryss, R., Reichert, M.: Mobile Crowd Sensing ? A New way of collecting data from trauma samples? In: XIV Conference of European Society for Traumatic Stress Studies (ESTSS) Conference. (2015) 244

[4] Schickler, M., Pryss, R., Stach, M., Schobel, J., Schlee, W., Probst, T., Langguth, B., Reichert, M.: An IT Platform Enabling Remote Therapeutic Interventions. In: 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017), IEEE Computer Society Press (2017)

[5] Schickler, M., Pryss, R., Reichert, M., Heinzelmann, M., Schobel, J., Langguth, B., Probst, T., Schlee, W.: Using Wearables in the Context of Chronic Disorders - Results of a Pre-Study. In: 29th IEEE Int'l Symposium on Computer-Based Medical Systems. (2016) 68–69

[6] Pschyrembel, W., Witzel, S.: Pschyrembel Klinisches Wörterbuch 2012. 263., neu bearb. und erw. aufl. edn. de Gruyter, Berlin [u.a.] (2011)

[7] Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Supporting Remote Therapeutic Interventions with Mobile Processes. In: 6th IEEE International Conference on AI & Mobile Services (IEEE AIMS 2017), IEEE Computer Society Press (2017)

[8] Spieß, M.: Ambulante Tinnitus Bewältigungstherapie - eine Langzeitbeobachtung des Therapieerfolgs. dissertation, Universität Ulm (2015)

[9] Kreuzer, P.M., Vielsmeier, V., Langguth, B.: Chronischer Tinnitus – eine interdisziplinäre Herausforderung. Deutsches Ärzteblatt | Jg. 110 | Heft 16 (2013)

[10] Langguth, B., Biesinger, E.: Textbook of Tinnitus. A. Moller, B. Langguth, D. DeRidder, T. Kleinjung (2011)

[11] Tassa, P.A., Adamchica, I., Freunda, H.J., von Stackelbergc, T., Hauptmanna, C.: Counteracting tinnitus by acoustic coordinated reset neuromodulation. Restorative Neurology and Neuroscience (2012)

[12] Rebentisch, B.: Behandlung Wie behandelt man objektiven Tinnitus? Tinnitus Magazin, Partnerportal der deutschen Tinnitus Stiftung (2017)

[13] Seidl, A.H.: Entwicklung und erfahrungsabhängige Plastizität neuronaler Mechanismen für Schalllokalisation bei Säugern . dissertation, Ludwig-Maximilian-Universität München (2003)

[14] Zimmermann, A.: Eigenschaften des Richtungshörens beim Menschen. Hauptseminar Sehen und Hören, SS 2004 , Universität Ulm (2004)

[15] Seeber, B.U.: Binaurales Hören mit Cochlea Implantaten. MRC Institute of Hearing Research, University Park, Nottingham, NG7 2RD, UK, DAGA Berlin (2010)

[16] Yost, W., Dye, R.: Properties of Sound Localization by Humans. Neurobiology of Hearing: The Central Auditory System, R.A. Altshuler et al. (ed.), Raven Press, Ltd., New York (1991)

[17] Baumgartner, J.: Richtungshören - Ein Richtungshörsystem für mobile Roboter in echoarmer Umgebung. Fakultät Informatik, Technische Universität Darmstadt (2015)

[18] Stumpe, J.: Realität und Wahrnehmung in der Akustik. AirWeb Seminar, Informatik, Universität Oldenburg (2000)

[19] li Zhong, X., sun Xie, B.: Head-Related Transfer Functions and Virtual Auditory Display. Soundscape Semiotics - Localization and Categorization (2014)

[20] Allan, A.: Basic sensors in iOS: "Programming the accelerometer, gyroscope, camera, and magnetometer-Cover. 1st ed edn. O'Reilly, Sebastopol, CA (2011)

[21] Platt, C.: Encyclopedia of electronic components, Volume 3, Sensors. First edition (online-ausg.) edn. Maker Media, San Francisco, CA (2016)

[22] Nyce, D.S.: Position sensors. John Wiley & Sons, Inc, Hoboken, New Jersey (2016)

[23] Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 383–394

[24] Watzka, B., Scheler, S., Wilhelm, T.: Beschleunigungssensoren. Praxis der Naturwissenschaften – Physik in der Schule 61, Nr. 7, 2012, S. 25 - 33 (2012)

[25] Geiger, P., Pryss, R., Schickler, M., Reichert, M.: Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices. Technical Report UIB-2013-09, Ulm University, Ulm (2013)

[26] Wilken, J.: Ionic in action: hybrid mobile apps with Ionic and AngularJS. Online-ausg. edn. Manning Publications, Shelter Island, New York (2016)

[27] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: A Configurator Component for End-User Defined Mobile Data Collection Processes. In: Demo Track of the 14th International Conference on Service Oriented Computing (ICSOC 2016). (2016)

[28] Khanna, R., Harlington, M., eds.: Getting started with Ionic: get up and running with developing effective hybrid mobile apps with Ionic. Online-ausg. edn. Packt Publishing, Birmingham, UK (2016)

[29] Camden, R.: Apache Cordova in action. Online-ausg. edn. Manning Publications, Shelter Island, NY (2016)

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

[31] Schobel, J., Pryss, R., Schickler, M., Ruf-Leuschner, M., Elbert, T., Reichert, M.: End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications. In: 5th IEEE International Conference on Mobile Services (MS 2016), IEEE Computer Society Press (2016) 1–8

[32] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: Towards Flexible Mobile Data Collection in Healthcare. In: 29th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2016). (2016) 181–182

[33] Schobel, J., Ruf-Leuschner, M., Pryss, R., Reichert, M., Schickler, M., Schauer, M., Weierstall, R., Isele, D., Nandi, C., Elbert, T.: A generic questionnaire framework supporting psychological studies with smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 69–69

[34] Schobel, J., Schickler, M., Pryss, R., Reichert, M., Elbert, T.: A Domain-Specific Framework for Collecting Data in Trials with Smart Mobile Devices. In: XIV Conference of European Society for Traumatic Stress Studies (ESTSS) Conference. (2015)

[35] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-Driven Data Collection with Smart Mobile Devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[36] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[37] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: A Lightweight Process Engine for Enabling Advanced Mobile Applications. In: 24th International Conference on Cooperative Information Systems (CoopIS 2016). Number 10033 in LNCS, Springer (2016) 552–569

A

Quelltexte

```
1  private makeSoundID(_callback){
2    let aSK = []; //assignedSoundKeys / schon belegte Sound IDs
3    this.storage.keys().then(_keys => {
4      for(let k of _keys){
5        if(k.startsWith(keys.prefixSound)){
6          //keys.prefixSound (globale Variable): "Sound_"
7          aSK.push(Number(k.substring(keys.prefixSound.length)));
8        }
9      }
10     if(aSK.length == 0){
11       //Wenn noch keine SoundSource dann id:0
12       _callback(0);
13       return;
14     }
15     aSK.sort((k1,k2) => k1 - k2);
16     if(aSK[0] != 0){
17       //Wenn keine SoundSource mit id 0 dann id:0
18       _callback(0);
19       return;
20     }
21     for(let i=0;i<aSK.length-1;i++){
22       if(aSK[i]+1 != aSK[i+1]){
23         _callback(aSK[i]+1);
```

```
24          //Wenn VorgaengerID+1 nicht Nachfolger ID ist,
25          //dann id:VorgaengerID+1, da sortiert
26          return;
27        }
28      }
29      //Keine Luecke der Nummerierung, id: letzter Eintrag +1
30      _callback(aSK[aSK.length-1]+1);
31      })
32    }
```

Listing A.1: Generieren einer ID am Beispiel der soundID

```
1  for(let i=0;i<this.unit.sources.length;i++) {
2    this.unit.sources[i].audioSource.stop();
3    let angleRadians = Math.atan2(this.unit.sources[i].z,
4                                  this.unit.sources[i].x);
5    let angleDegrees = (360+(angleRadians * 180 / Math.PI))%360;
6    let accuracy:any;
7    if(this.angles[i] == angleDegrees){
8      //Wenn Winkel genau uebereinstimmen
9      accuracy = 0;
10   }else if(this.angles[i] > angleDegrees){
11     //Wenn angegebener Winkel groeser als richtiger Winkel
12     if(this.angles[i]-angleDegrees <= 180){
13       //Wenn Unterschied der Winkel weniger als 180 Grad ist
14       accuracy = this.angles[i]-angleDegrees;
15     }else{
16       //Wenn Unterschied der Winkel mehr als 180 Grad ist
17       //dann ist der Weg "ueber die 0" kuerzer
18       accuracy = 360-this.angles[i]+angleDegrees;
19     }
20   }else{
```

```
21      //analog zu oben
22      if(angleDegrees-this.angles[i] <=180){
23        accuracy = angleDegrees-this.angles[i];
24      }else{
25        accuracy = 360-angleDegrees+this.angles[i];
26      }
27    }
28    this.results.push({"name":this.names[i],"accuracy":accuracy});
29  }
```

Listing A.2: Berechnung der Genauigkeit für Trainingseinheiten mit Modus 1

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Martin Randelzofer Matrikelnummer: 833308

Erklärung

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

Ulm, den .....

Martin Randelzofer