ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Masterarbeit
Im Studiengang Informatik

Konzeption und Realisierung eines Patienten-Edukationsmoduls für eine multizentrische und multinationale mHealth-App für eine paneuropäische Tinnitus-Studie

Verfasser: Fabian Haug
Matrikelnummer: 843435
1. Prüfer: Prof. Dr. Manfred Reichert
2. Prüfer: Prof. Dr. Rüdiger Pryss
Betreuer: Carsten Vogel, M.Sc.

2020

Fassung vom 29. Oktober 2020

Kurzfassung

Immer häufiger werden Apps eingesetzt, um Menschen nicht nur Unterhaltung zu bieten, sondern auch, um sie in wichtigen Lebensbereichen wie zum Beispiel der Gesundheit zu unterstützen. Die Bedeutung und Gewichtung solcher mHealth-Apps wird gerade im „Corona-Jahr“ 2020 durch die Corona-Warn-App deutlich.

Diese Arbeit soll zeigen, wie ein Edukationsmodul einer multizentrischen und multinationalen mHealth-App entwickelt werden kann, um Tinnitus-Patienten zu unterstützen. Mithilfe diesem und weiteren Modulen sollen im Rahmen des europäischen UNITI-Projektes Studien durchgeführt werden, um mögliche weitere Behandlungsmethoden zu identifizieren.

Viele Menschen mit Tinnitus sind nicht ausreichend über ihre Erkrankung aufgeklärt oder tun sich schwer, an nützliche Informationen zu gelangen. Dadurch ist es häufig der Fall, dass sie, bezüglich der neuesten Erkenntnisse, über die Krankheit Tinnitus und deren Behandlungsmöglichkeiten, nicht auf dem neuesten Stand sind. Im Volksmund nur als das „Piepsen oder Klingeln im Ohr“ bekannt, wird Tinnitus oft nicht ernst genug genommen und auch seine indirekten Auswirkungen und Ursachen, wie Depressionen und Stress, werden vernachlässigt. Dabei können schon kleine Tipps und Informationen über Tinnitus den Patienten enorm helfen, mit der Krankheit besser umzugehen oder sogar ihre Symptome deutlich lindern.

In dieser Arbeit wird gezeigt, wie es möglich ist, Patienten oder auch nur Interessierten, Informationen über die Krankheit Tinnitus näher zu bringen, in Form eines „herausfordernden“ Ansatzes. Dabei werden dem Benutzer die Informationen schrittweise, in mehreren Kapiteln und Lektionen, vermittelt und am Ende eines jeden Kapitels steht als letzte Lektion ein Quiz über die zuvor gelesenen Inhalte an. Dadurch soll der Benutzer eine Rückmeldung bekommen, wie gut er die Informationen aufgenommen hat. Zusätzlich fungiert das Erreichen eines bestmöglichen Ergebnisses beim Quiz als Motivation, die Informationen so gut wie möglich zu behalten.

Des Weiteren wird ähnlich, wie in vorangegangen Projekten, wie TrackYourTinnitus, eine Fragebogen-Funktion vorhanden sein, um den Zustand von Tinnitus-Patienten für mögliche Studien zu erfassen und dem Patienten Feedback zu seiner momentanen Situation zu geben.

Im weiteren Verlauf dieser Arbeit wird gezeigt, wie solche Module einer App aufgebaut werden und architektonisch zu realisieren sind. Diese werden dann in die Struktur der Android UNITI-App eingebunden, deren Aufbau in einer anderen Masterarbeit der Universität Ulm gezeigt wird. Dabei wird auf verschiedene Bereiche der Implementierung ausführlicher eingegangen, wie zum Beispiel den Aufbau der Fragebögen, bzw. Quiz mit deren Anzeige und Datenverarbeitung, Visualisierung des Feedbacks, sowie den strukturellen Aufbau und die Darstellung der Kapitel und Lektionen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellen

1 Einleitung

Obwohl mit 15% der Weltbevölkerung ein großer Teil der Menschheit von Tinnitus betroffen ist und ihre Symptome dadurch weitläufig bekannt sind, stellt die Erkrankung die Wissenschaft vor Rätsel. In den letzten Jahren wurden zahlreiche Fortschritte in der Erforschung der Krankheit erzielt, jedoch sind durch die enorme Heterogenität der Erkrankung weiterhin viele Fragen bezüglich der Behandlungsmethoden offen. Darum wird in Initiativen, wie der *European School on Interdisciplinary Tinnitus Research (ESIT),* versucht, durch multidisziplinäre Zusammenarbeit neue Erkenntnisse zu gewinnen. [1; 2]

Dies führt dazu, dass immer häufiger Apps in medizinischen Bereichen und für die Forschung eingesetzt werden. Die Bedeutung und Akzeptanz dieser mHealth-Apps ist in den letzten Jahren, unter anderem durch die Corona Warn App [3] auch bei der Bevölkerung deutlich gestiegen.

In dieser Arbeit soll ein Modul für eine mHealth-App auf dem Android-Betriebssystem realisiert werden, mit der Benutzer, hauptsächlich Tinnitus-Patienten, durch bereitgestellte Informationen in Text, Bild, Ton und Video ihre Kenntnisse über Tinnitus auffrischen bzw. erweitern können. Dabei soll eine regelmäßige Überprüfung des Gelernten für effektiveres Lernen sorgen und nebenbei als Motivation dienen.

Des Weiteren besteht für die Benutzer die Möglichkeit ihre Erkrankung, durch das Ausfüllen eines einmaligen demographischen, sowie regelmäßig wiederkehrenden, mehrmals zu beantwortenden kontinuierlichen Fragenbögen zu dokumentieren. Durch die damit verbundene Teilnahme an einer Studie, ist es für den Benutzer auch möglich, Rückmeldungen zu seinen Antworten zu erhalten. Dies ist ein Versuch, den Patienten grundlegende Tipps und Informationen zu geben, um so möglicherweise ihre Symptome zu lindern oder auch nur, um Interessierte für das Thema Tinnitus zu sensibilisieren.

Mithilfe der Erfassung von Meta-Daten (GPS, Läutstarke, Nutzungsdaten des Gerätes, usw.), zusätzlich zu den Antworten und dem Lernfortschritt der Benutzer, bietet sich für die Wissenschaft die Chance, Zusammenhänge zwischen verschiedenen Krankheitsbildern und bestimmten Lebensweisen oder -umständen zu erkennen.

1.1 Motivation

Dieses Softwareprojekt ist Teil meiner Masterarbeit an der Universität Ulm. Durch Erfahrungen mit Tinnitus und anderen Krankheiten bzw. Schädigungen des Gehörs in meiner Familie ist dieses Thema nicht nur aufgrund der softwarespezifischen Thematik für mich interessant. Die derzeitig vorhandene App TrackYourTinnitus ist weder auf dem neuesten Stand noch für alle neuen Betriebssystemversionen kompatibel. Im Rahmen des UNITI-Projektes soll eine Plattform für Tinnitus-Patienten neu aufgebaut werden. Dabei soll es nicht nur um Studien für die Wissenschaft gehen, sondern auch darum, Erkenntnisse der Wissenschaft wieder an die Benutzer zurückzugeben und so einen Lernprozess einzuleiten.

1.2 Vision

Das Ziel ist die Entwicklung einer einfach zu bedienenden, übersichtlichen und benutzerfreundlichen App. Mit dieser soll der Benutzer leicht verständlich und am besten in seiner ausgewählten Sprache an Informationen gelangen, die Fragebögen ausfüllen können und ein Feedback zu seinen Angaben bekommen. Dadurch wird der Benutzer motiviert, die Informationen konzentriert aufzunehmen und zu behalten. Mithilfe einer Erfolgsrückmeldung bei einem Quiz sollte der Ehrgeiz des Benutzers angeregt werden, das optimale Ergebnis zu erreichen und sich noch intensiver mit den Lehrinhalten der Kapitel auseinanderzusetzen. Dabei soll die App mit ihrer Erinnerungsfunktion helfen, dass der Benutzer die Fragebögen regelmäßig ausfüllt, was zu einer aussagekräftigeren Studie führt. Weiter wäre eine Zertifizierung der App nach IEC 62304 [4] bzw. IEC 82304 wünschenswert, da sie so dem Medizinproduktegesetz [5] entsprechen würde. Mit den erfassten Daten soll die Tinnitusforschung mit dem direkten Bezug zum Patienten vorangetrieben werden. Durch die anonyme Erfassung sind ehrlichere Angaben zu erwarten, wodurch die Daten eine höhere Aussagekraft haben sollten als persönliche Befragungen. Zusätzlich ist das Ziel, durch die weltweite Verbreitung der App, die Zielgruppe und somit auch den erfassten Datensatz so groß wie möglich zu gestalten. Je mehr Daten vorhanden sind, desto einfacher wird es Zusammenhänge zu erkennen und mögliche Falschangaben herauszufiltern.

1.3 Projektkontext

Der Europäischen Union ist das Thema Gesundheit der Bürger sehr wichtig, darum wird in mehrere Projekte zur Förderung und Verbesserung der Gesundheit der EU-Bürger investiert. Eines dieser Projekte ist das UNITI-Projekt. Im Rahmen dieses Projektes soll eine App entstehen, die mehrere Funktionalitäten unterschiedlicher Tinnitus-Apps verbindet. Dazu gehören die in dieser Arbeit näher beleuchteten Fragebögen für wissenschaftliche Studien, wie aus TrackYourTinnitus bekannt, und das Edukationsmodul, sowie ein Modul für auditorische Stimulationen, das in einer anderen Arbeit [6] näher erklärt wird. [7]

Dadurch soll sie gewährleisten, dass ein Benutzer mithilfe einer Plattform auf alle seine gewünschten Behandlungs- und Informationsfunktionen zugreifen kann und nicht auf mehrere unterschiedliche Apps angewiesen ist. Dabei gibt es die Möglichkeit, die App vollständig anonym zu nutzen. Nichtsdestotrotz können Querverweise über die Datensätze der unterschiedlichen Funktionalitäten für wissenschaftliche Arbeiten und Forschung generiert werden.

1.4 Aufbau der Arbeit

Diese Arbeit besteht aus acht Kapiteln. Nach der Einleitung folgt in Kapitel 2 ein kurzer Überblick über bereits existierende „mHealth-Projekte“ und Apps, die eine ähnliche Funktionalität wie die App des UNITI-Projektes aufweisen. Anschließend geht es in Kapitel 3 um die Darstellung der Anforderungen an die betroffenen Module. Danach wird in Kapitel 4 die Architektur der Module beschrieben. Dabei wird eine generelle Übersicht der Architektur gegeben und die Datenstruktur, sowie der allgemeine Aufbau der Module erklärt. In Kapitel 5 werden Code-Teile der Implementierung der Module gezeigt, die zur Erfüllung der Anforderungen beitragen. Ein kleiner Einblick in Kapitel 6 beschreibt die

Möglichkeiten und Funktionen, die die App dem Benutzer bietet. Daraufhin werden in Kapitel 7 die Anforderungen mit dem Stand der Entwicklung abgeglichen. Abschließend erfolgt in Kapitel 8 eine Zusammenfassung und ein Ausblick auf die mögliche Weiterentwicklung des Projektes.

2 Existierende Projekte

In diesem Kapitel werden weitere mHealth-Apps und Projekte, die ähnliche Funktionalitäten wie das UNITI-Projekt besitzen oder thematisch vergleichbar sind, kurz erklärt und beschrieben. Dabei wird auf Ähnlichkeiten und Unterschiede eingegangen oder wie diese Projekte die Überlegungen zum UNITI-Projekt beeinflusst haben.

2.1 Track Your Tinnitus

Das Track Your Tinnitus Projekt wurde bereits 2014 gestartet und hilft, Menschen mit Tinnituserkrankung zu unterstützen. Dabei dient es gleichzeitig zur weiteren Erforschung der Krankheit, speziell beim Verlauf der Schwankungen der Lautstärke und deren Ursachen. Menschen, die an Tinnitus erkrankt sind, hören Töne oder Geräusche, denen keine äußeren Schallquellen zugeordnet werden können. Bei 60% der Patienten kommt es zu Schwankungen der Lautstärke dieser Geräusche. Die App ist darauf ausgerichtet, dass der Benutzer immer wieder, zu unterschiedlichen Zeiten, die Lautstärke des Tinnitus mithilfe der App dokumentiert. Dadurch ist es möglich einen detaillierten zeitlichen Ablauf der Lautstärke nachzuvollziehen und dem Benutzer zu visualisieren. Des Weiteren ist es möglich, diese Schwankungen mit verschiedenen Alltagssituationen in Verbindung zu bringen und so zu überprüfen, wie sich zum Beispiel Stress oder die Umgebungslautstärke auf den Tinnitus auswirken. [8–15]

Das Track Your Tinnitus Projekt ist im Rahmen einer Diplomarbeit von Jochen Herrmann an der Universität Ulm 2014 [16] entstanden und wird jetzt im Rahmen des UNITI-Projektes neu aufgebaut und durch das Fragebogenmodul dieser Arbeit, in der App integriert. Die App für das Track Your Tinnitus Projekt ist im GooglePlay Store[1] und im Apple AppStore[2] verfügbar.

2.2 Corona Health

Die Corona Health[3] App ist in wissenschaftlicher Zusammenarbeit der Universität Würzburg[4], dem Universitätsklinikum Würzburg[5], der Universität Ulm[6], der LA2 GmbH[7], der Universität Regensburg[8] und dem Robert-Koch-Institut[9] entwickelt worden. Dabei wird versucht, die psychischen und physischen Auswirkungen der Corona-Krise wissenschaftlich zu erfassen und den Benutzern konstruktives Feedback zu ihrer Situation zu geben. Momentan werden damit 3 verschiedene Studien

[1] Track Your Tinnitus GooglePlay Store: https://play.google.com/store/apps/details?id=com.jochenherrmann.trackyourtinnitus (15.10.2020)
[2] Track Your Tinnitus Apple AppStore: https://apps.apple.com/de/app/track-your-tinnitus/id787178122 (15.10.2020)
[3] Vgl. Universität Ulm: https://www.corona-health.net/ (15.10.2020)
[4] Universität Würzburg: https://www.med.uni-wuerzburg.de/epidemiologie/startseite/ (15.10.2020)
[5] Universitätsklinikum Würzburg: https://www.ukw.de/startseite/ (15.10.2020)
[6] Universität Ulm: https://www.uni-ulm.de/ (15.10.2020)
[7] LA2 GmbH: https://www.la2.de/ (15.10.2020)
[8] Universität Regensburg: https://www.uni-regensburg.de/medizin/psychiatrie-psychotherapie/forschung/e-health/mitarbeitende/index.html (15.10.2020)
[9] Robert-Koch-Institut: https://www.rki.de/DE/Content/Institut/OrgEinheiten/Abt2/FG26/fg26_node.html;jsessionid=9587058B67BC2B7299A3EC14FF648886.internet051 (15.10.2020)

durchgeführt (Physische Gesundheit für Erwachsene, Körperliche Gesundheit für Erwachsene, Physische Gesundheit für Jugendliche von 12 bis 17 Jahren).

Eine Besonderheit der App besteht darin, dass es für alle Benutzer möglich ist, die App anonym zu nutzen, aber trotzdem ein individuelles Feedback und Informationen zu erhalten. Weiter kann der Benutzer jederzeit frei entscheiden, an welchen Studien er teilnehmen möchte und wird benachrichtigt, wenn er einen neuen Fragebogen ausfüllen soll. Die Corona Health App ist im GooglePlay Store[10] und im Apple AppStore[11] verfügbar.

2.3 Track Your Stress

Das Track Your Stress Projekt, früher Assess Your Stress [17], ist ebenfalls ein Projekt an der Universität Ulm[6] und der Universität Regensburg[8]. Dabei wird das Track Your Tinnitus Projekt genutzt und neu aufgebaut. Die Betrachtung soll dabei nicht ausschließlich auf Tinnitus Symptomen verbleiben, sondern sich auf die Messung des Stresslevels des Benutzers und die Ursachen dafür konzentrieren. Dazu werden, durch regelmäßige, gezielte Befragungen, wissenschaftlich erstellte Skalen berechnet, an denen der Benutzer den Verlauf seines Stresslevels erkennen kann. Für die wissenschaftliche Seite werden während der Befragung auch Lautstärkemessungen am Gerät vorgenommen, um mögliche Zusammenhänge des Stresslevels und der Umgebungslautstärke zu erkennen. Dieses Projekt wurde ins Leben gerufen, nachdem der Deutsche Bundestag das Präventionsgesetz [18], das zur Stärkung der Gesundheitsförderung und Prävention führen soll, verabschiedet hat.

Das Projekt wird derzeit in mehreren Abschlussarbeiten an der Universität Ulm und der Universität Regensburg entwickelt und ist im Moment noch nicht in den gängigen AppStores verfügbar. Die Entwicklung befindet sich im Endstadium und die App wird bald für Android und iOS erscheinen.

2.4 Corona Check Screening

Zu Beginn der Corona-Krise gab es eine Überlastung der Gesundheitsämter durch Anrufe von verunsicherten Bürgern, die Fragen zum neuartigen Virus und ihrem Verhalten, bei einem möglichen Kontakt mit Infizierten hatten. Um die Gesundheitsämter zu entlasten hat das Bayerische Landesamt für Gesundheit und Lebensmittelsicherheit[12] in Zusammenarbeit mit der Universität Würzburg[4], dem Universitätsklinikum Würzburg[5], der Universität Ulm[6], der LA2 GmbH[7] und der Universität Regensburg[8] die Corona Check Screening App[13] entwickelt, die als erste Anlaufstelle für Bürger dienen soll.

Die App enthält einen Live-Ticker, der dem Benutzer aktuelle Informationen über die Corona-Krise bereitstellen soll, sowie eine Seite mit Verhaltenstipps wie zum Beispiel das korrekte und regelmäßige Lüften, mit zugehöriger Erklärung, warum der Tipp hilfreich und sinnvoll ist. Der Benutzer hat dabei die Möglichkeit, diesen Tipp zu bewerten und erkennt gleichzeitig, für wie hilfreich die Community den Tipp erachtet.

[10] Corona Health GooglePlay Store: https://play.google.com/store/apps/details?id=com.dbis.haugxhaug.coronahealth (15.10.2020)
[11] Corona Health Apple AppStore: https://apps.apple.com/de/app/corona-health/id1519399353 (15.10.2020)
[12] Bayerisches Landesamt für Gesundheit und Lebensmittelsicherheit: https://www.lgl.bayern.de/ (15.10.2020)
[13] Vgl. Universität Ulm: https://www.coronacheck.science/de/ (15.10.2020)

Die Hauptfunktionalität der App besteht aus einem Schnellcheck für eine Risikoanalyse bezüglich einer möglichen Infektion. Dabei beantwortet der Benutzer die Fragen, die der Mitarbeiter des Gesundheitsamtes am Telefon stellen würde und bekommt nach der Abgabe einen Ergebnistext mit Handlungsempfehlungen. Das komplette Vorgehen samt Fragen, Ergebnistext und Handlungsempfehlungen richtet sich dabei nach den Vorgaben des Robert-Koch-Instituts. Die App ist für Android[14] und iOS[15] verfügbar.

2.5 MobileTx

Im Projekt [19] der Universität Ulm und speziell der Arbeit von Marc Schickler [20] geht es um die Unterstützung von Patienten und Therapeuten während des gesamten Interventionsprozesses.

Dabei wurde ein Rahmenwerk erstellt, das die Zeit zwischen den Therapiesitzungen sinnvoll und mit der bestmöglichen Wirksamkeit nutzen sollte, um die Therapie flexibler und effektiver zu gestalten. Das Vorgehen besteht darin, dass die Therapeuten Hausaufgaben für die Patienten erstellen können, die die Patienten über ihre mobilen Geräte (Smartphone, Tablet, Smartwatch, usw.) abrufen und bearbeiten können. Nach der Bearbeitung bekommen sie eine Auswertung, wodurch die Therapeuten mit den erhobenen Daten die Therapie flexibel anpassen können. Dabei können sie mit verschiedenen e-Health Tools die Daten evaluieren und neue Zusammenhänge oder Therapiemethoden erkennen. Das Projekt hat bereits gezeigt, dass dieses Rahmenwerk eine erhebliche Verbesserung der Behandlung und Forschung erzielt und wurde in mehreren praktischen Projekten getestet und validiert. [20]

2.6 Kalmeda App

Die Kalmeda App ist eine Entwicklung der mynoise GmbH und „bietet eine wissenschaftlich basierte Tinnitus-Therapie auf der Basis einer kognitiven Verhaltenstherapie“ [21].

Laut Internetauftritt wurde die App von HNO-Ärzten und Psychologen entwickelt. Die therapeutischen Maßnahmen seien wissenschaftlich geprüft, die App sei als Medizinprodukt zugelassen und als digitale Gesundheitsanwendung geführt [21].

Die Funktionsweise der Verhaltenstherapie wird dabei wie nachfolgend beschrieben. Als Benutzer hat man die Möglichkeit, in den Übungen mehrere Level mit jeweils mehreren Etappen zu durchlaufen. Der Benutzer soll für sich selbst erlernen, im Alltag zu entspannen und zu mehr innerer Ruhe finden. Dadurch soll er nach und nach in der Lage sein, Situationen und Dinge zum Positiven zu verändern. Die Etappen bauen systematisch aufeinander auf und sollen dem Benutzer schrittweise helfen, schädliche, tinnitusfördernde Einstellungen zu identifizieren und möglichst durch positive Einstellungen zu ersetzen. Der Benutzer kann sich selbst in der Anwendung persönliche Ziele setzen. Um diese Ziele zu erreichen, werden dem Benutzer zusätzlich Hilfen angeboten, unter anderem kann sich der Benutzer ein Helfernetzwerk aufbauen, das ihn beim Erreichen seiner Ziele unterstützt. [21]

[14] Corona Check Screening GooglePlay Store: https://play.google.com/store/apps/details?id=com.coronacheck.haugxhaug.testyourcorona (15.10.2020)
[15] Corona Check Screening Apple AppStore: https://apps.apple.com/de/app/corona-check-screening/id1504712226?l=de&ls=1 (15.10.2020)

Die mehrmonatige Verhaltenstherapie wird durch mehrere Funktionen ergänzt. Hierzu zählt ein Wissensteil und die Funktionalität, angenehme Hintergrundgeräusche abzuspielen. Außerdem sollen Entspannungsübungen und eine geführte Meditation enthalten sein. Generell gibt es zwei verschiedene Optionen die App zu nutzen.

Kalmeda START steht dem Benutzer kostenfrei zur Verfügung und bietet einen ersten Therapieplan und Entspannungsübungen, vor allem soll sie dem Benutzer aber einen Überblick über die umfassenden Leistungen geben.

In Kalmeda GO hingegen stehen alle Funktionen zur Verfügung. Allerdings ist damit ein kostenpflichtiges Abo, per in-App-Kauf, verbunden. Dieses Abo kann jedoch von der Krankenkasse auf Rezept für 3 Monate übernommen werden. Der Benutzer bekommt beim Einreichen des Rezepts bei der Krankenkasse einen Verifikationscode zurückgeschickt. Mit diesem kann sich der Benutzer auf der Website[16] einloggen und seinen Account für die 3-monatige Tinnitustherapie aktivieren.

Im Groben ähnelt die Kalmeda App sehr der UNITI-App. Allem voran dem Edukationsmodul. In diesem wird der Benutzer auch durch mehrere Kapitel und Lektionen geführt, wodurch die eigene Selbstwahrnehmung gesteigert und der Umgang mit der (Tinnitus-) Erkrankung erleichtert werden soll. Dazu kann der Benutzer in der UNITI-App über die Quiz seinen Lernerfolg überprüfen, was dem Erreichen der Ziele der Kalmeda App entspricht. Vorteile und nette Features der Kalmeda App sind die Hilfsangebote und die Entspannungsübungen mit geführter Meditation. Die Funktionalität der Hintergrundmusik der Kalmeda App wird in der UNITI-App gleichwertig durch das Modul der auditorischen Stimulation [6] erfüllt. Vorteil der UNITI-App ist die vollkommen kostenfreie Nutzung, sowie eine anonyme Registierung, d. h. ein Benutzer kann die App verwenden ohne persönliche Daten für einen Account angeben zu müssen. Im Endeffekt lässt sich jedoch sagen, dass sich die Apps nur marginal unterscheiden und es auf die subjektive Empfindung der Benutzer ankommt, welche ihnen eher zusagt.

2.7 Weitere Android Apps

Es gibt im GooglePlay Store[17] weitere Apps, die sich mit dem Thema Tinnitus beschäftigen. So sind als Top Suchergebnisse die Apps „Beltone Tinnitus Calmer“[18], „Tinnitus Relief App. Klangtherapie“[19] und „Tonal Tinnitus Therapy“[20] zu nennen.

Dabei liegt der Hauptaspekt bei diesen Apps nicht in der Bewertung der Lebenssituation des Patienten, der Informationsbereitstellung über die Erkrankung und somit der Generierung und Sammlung von Wissen und Daten. Das Hauptaugenmerk liegt hier in der Symptombehandlung, die meistens über die auditorische Stimulation gewährleistet wird. Im Falle der Beltone Tinnitus Calmer App kommen geführte Meditationen dazu, da Stress eine große Rolle bei Tinnitus Patienten spielt.

Im Gegensatz zum UNITI-Projekt, das eher auf die medizinischen und wissenschaftlichen Aspekte des Tinnitus konzentriert ist, werden bei den meisten im GooglePlay Store angebotenen Apps, eher Wohlfühl- und Behandlungsaspekte in den Vordergrund gerückt. Beim UNITI-Projekt werden zusätzlich zur Behandlung die Grundsteine für zukünftige Behandlungsmethoden gelegt. Dies geschieht durch

[16] Kalmeda Website: https://www.kalmeda.de/ (15.10.2020)
[17] GooglePlay Store: https://play.google.com/store?hl=de (15.10.2020)
[18] Beltone Tinnitus Calmer GooglePlay Store: https://play.google.com/store/apps/details?id=com.beltone.tinnitus&gl=DE (15.10.2020)
[19] Tinnitus Relief App. Klangtherapie GooglePlay Store: https://play.google.com/store/apps/details?id=com.zdn35.music.songs.audio.tinnitussoundtherapy (15.10.2020)
[20] Total Tinnitus Therapie GooglePlay Store: https://play.google.com/store/apps/details?id=nl.appyhapps.tinnitusmassage (15.10.2020)

die Generierung von Wissen und dem Erkennen von Zusammenhängen beim Benutzer, sowie den wissenschaftlichen Mitarbeitern, die die in den Studien erhobenen Daten auswerten.

3 Anforderungen

Dieses Kapitel definiert die Anforderungen an die behandelten Module des UNITI-Projektes. Dabei sind diese Anforderungen in funktionale und nicht-funktionale Anforderungen unterteilt.

3.1 Funktionale Anforderungen

Dieser Abschnitt zeigt die funktionalen Anforderungen an die App. Dabei werden die wichtigsten Funktionen der App gezeigt. Die folgende Tabelle zeigt eine Aufstellung der funktionalen Anforderungen.

Nr.	Anforderung	Beschreibung
Edukationsmodul		
1.	Ordnung der Informationen	Das Modul soll dem Benutzer eine leicht verständliche Ansicht der Informationen ermöglichen, durch eine Aufteilung in Kapitel, Unterkapitel und Quiz. Dafür soll eine geeignete Datenstruktur erstellt werden.
2.	Fortschrittsanzeige für Kapitel und Lektionen	Dem Benutzer soll klar ersichtlich sein, welche Kapitel, Unterkapitel und Quiz er schon abgeschlossen hat und welches als nächstes zu bearbeiten ist. Dabei soll durch ein vereinfachtes Rangsystem und die Anzeige für das Abschneiden beim Quiz die Motivation, um weiterzumachen oder Bekanntes zu wiederholen, erhöht werden.
3.	Anzeige der Informationen	Der Benutzer soll die Informationen in den jeweiligen Unterkapiteln in verschiedenen Formaten (Text, Bild, Video, Ton, usw.) bereitgestellt bekommen. Durch die Einbindung von Bildern oder auch Videos soll das Lernen interessanter gestaltet werden.
4.	Benachrichtigung als Erinnerung	Die App soll dem Benutzer eine Benachrichtigung schicken, wenn er 3 Tage kein Unterkapitel oder Quiz mehr abgeschlossen hat, um ihn wieder auf die App aufmerksam zu machen.
5.	Erfassung des Lernfortschrittes	Das Modul soll erfassen, wann ein Benutzer ein Unterkapitel abgeschlossen hat und diese Information an den Server senden.
Fragebögen und Quiz		
6.	Quiz ausfüllen	Der Benutzer kann pro Kapitel ein Quiz ausfüllen und die Ergebnisse an den Server senden, um eine detaillierte Auswertung zu erhalten. Eine Auswertung auf richtig und falsch soll lokal stattfinden, um auch bei Offline-Betrieb die Motivation des Benutzers aufrechtzuerhalten.

7.	Statistische Fragebögen ausfüllen	Studienteilnehmer der TrackYourTinnitus-Studie sollen einmalige demographische und kontinuierlich wiederkehrende Fragebögen beantworten können und ihre Ergebnisse an den Server senden. Die Fragebögen sollen dem Studienteilnehmer in der App angezeigt werden.
8.	Zustand eines Fragebogens ändern	Fragebögen sollen je nach Konfiguration vom Benutzer einmal oder mehrmals wiederholt ausgefüllt werden können. Wurde ein einmaliger Fragebogen bereits ausgefüllt, darf er dem Benutzer nicht noch einmal präsentiert werden.
9.	Ergebnisse synchronisieren	Zur Visualisierung der Ergebnisse aus den Fragebögen, sowie für Forschungszwecke, sollten die Ergebnisse aus den Apps an den Server übertragen werden.
10.	Lautstärkemessung	Während des Ausfüllens eines Fragebogens oder Quiz soll es, nach Einverständnis des Benutzers, möglich sein die Umgebungslautstärke über die Mikrofone des Gerätes zu bestimmen.
11.	GPS-Daten erfassen	Die App soll, bei entsprechend genehmigter Berechtigung durch den Benutzer, die GPS-Daten eines Gerätes erfassen können. Um den Datenschutzrichtlinien zu entsprechen, sollten die Daten auf ca. 11 km vergröbert an den Server gesendet werden.
12.	Erfolgsangabe in Prozent	Für die Quiz soll dem Benutzer eine Anzeige bereitgestellt werden, die ihm anzeigt, zu wie viel Prozent er die Fragen eines Quiz richtig beantwortet hat.
13.	An Fragebögen erinnern	Ein Benutzer sollte von der App benachrichtigt werden, wenn ein neuer Fragebogen auf Basis eines hinterlegten Terminplans ausgefüllt werden soll.
Feedback		
14.	Abfragen der Ergebnisse	Um den Kapitelfortschritt und Ergebnisse bei einer Neuinstallation oder Geräte-Wechsel zu erhalten, sollten die Ergebnisse vom Server geladen werden.
15.	Ergebnistexte zusammenfassen	Für ein Quiz soll es die detaillierte Erfolgsangabe nur in Prozent geben, wie in Punkt 12 beschrieben. Die Ergebnistexte sollen den Erfolg für alle Fragen eines Unterkapitels zusammenfassen.
16.	Ergebnisse anzeigen	Um die zeitliche Entwicklung in der App direkt anzeigen zu können, sollten die Ergebnisse aus den Fragebögen und Quiz visualisiert werden.
Nutzungsdaten		
17.	Erfassung von Nutzungsdaten	Es soll möglich sein, dass die App die Nutzungsdaten des Gerätes auslesen kann und für Forschungszwecke in einer kompakten Form an den Server übermittelt. Dafür wird die Zustimmung des Benutzers benötigt, die in der App abgefragt werden soll. Dies soll die Erkennung potenzieller Querverweise zwischen der Nutzung anderer Apps und dem Verlauf der Erkrankung ermöglichen.

	Allgemein	
18.	Nutzung ohne Internetverbindung	Der Benutzer soll die App nach dem Login und dem ersten Laden der Daten auch ohne Internetverbindung im größtmöglichen Funktionsumfang verwenden können.

Tabelle 1: Funktionale Anforderungen

3.2 Nicht-funktionale Anforderungen

Dieser Abschnitt zeigt die nicht-funktionalen Anforderungen an die App. Diese ergeben sich durch Besprechungen mit dem Betreuer und den Verantwortlichen, sowie durch die Einhaltung der gängigen Coding-Richtlinien und Anforderungen an die Struktur. Die folgende Tabelle zeigt eine Aufstellung der nicht-funktionalen Anforderungen.

Nr.	**Beschreibung**	**Problembeschreibung**
1.	Design & Bedienung	Jegliche Bedienung und Bedienelemente sollten für den Benutzer intuitiv und einfach erkenn- und nutzbar sein.
2.	Modularität	Die Implementierung sollte so modular wie möglich sein, um eine einfache Einbindung in die UNITI-App des Gesamtprojektes zu ermöglichen. Des Weiteren soll eine Wiederverwendung einzelner Funktionalitäten, in anderen Projekten, möglich sein.
3.	Erweiterbarkeit	Die Funktionalitäten sollen einfach und ohne großen Mehraufwand erweiterbar sein. Zum Beispiel neue Fragen oder Kapitel.
4.	Mehrsprachigkeit	Die App soll für Benutzer in aller Welt zugänglich gemacht werden und muss dafür in mehreren Sprachen verfügbar sein.
5.	Verfügbarkeit	Um eine große Zahl an Benutzern zu erreichen sollte die App für möglichst viele, auch ältere Betriebssystemversionen verfügbar sein.
6.	Store-ready	Die entstehenden Module sollen eingebaut in der Gesamt-App des UNITI-Projektes bezüglich Funktionalität und Aussehen bereit sein, um im GooglePlay Store veröffentlicht werden zu können.
7.	Zertifizierung für das Medizinproduktegesetz [5]	Für die Sicherheit und das Vertrauen des Benutzers wird eine Zertifizierung der App und ihrer Module für das Medizinproduktegesetz angestrebt. Dies erfordert unter anderem mehrere Testphasen, eine Risikoanalyse und einen fehlerfreien Ablauf der App.

Tabelle 2: Nicht-funktionale Anforderungen

4 Architektur

Dieses Kapitel beschreibt die Architektur der Module dieser Arbeit, die in die UNITI-App des Gesamtprojektes integriert werden. Deren Grundaufbau ist in einer anderen Arbeit [6] nachzulesen. Zuerst wird eine kurze Übersicht über die Architektur gegeben (4.1). In Kapitel 4.2 wird ein typischer Ablauf gezeigt, gefolgt von der detaillierten Beschreibung der Datenstruktur in Kapitel 4.3. Zum Abschluss wird in Kapitel 4.4 die Architektur der Module erklärt.

4.1 Architekturübersicht

Die UNITI-App besteht aus zwei Komponenten: Server und App für ein Android Smartphone. Der Server, sowie die API, wurden vom Institut für Datenbanken und Informationssysteme bereitgestellt. Die App wurde nativ mit Java auf Android entwickelt und greift nicht direkt auf Daten des Servers zu. Jegliche Kommunikation zwischen App und Server läuft über eine REST-ähnliche JSON-API [22].

Die Module werden nach dem MVC-Pattern aufgebaut, bei dem Activities die Views repräsentieren, die Manager-Klassen als Controller fungieren und die Model-Klassen, sowie die lokale Datenbank und der Server die Model-Seite bilden.

Das Modul für die Nutzungsdatenerfassung bildet hier im Gegensatz zum Edukations- und Fragebogenmodul eine Ausnahme, da es sich hierbei nur um Datenverarbeitung handelt. Deshalb wird dieses Modul auch als Library ausgegliedert, um damit für weitere Projekte im mHealth-Bereich noch leichter wiederverwendbar zu sein.

4.2 Genereller Ablauf

Die Benutzung der UNITI-App setzt ein Benutzerkonto voraus. Die Abläufe, um dieses zu erstellen, bzw. zu nutzen werden hier als gegeben angesehen und in [6] näher beschrieben. Die Beschreibung der Abläufe beginnt hier beim Klick auf die entsprechenden Tabs im Bottom-Menü, woraufhin das entsprechende Modul gestartet wird.

Edukationsmodul:

Der Ablauf ist detailliert in Abbildung 1 zu sehen. Beim Start versucht das Modul die Quiz vom Server abzurufen und zu speichern oder zu aktualisieren. Falls dies nicht möglich ist, werden die Quiz aus dem lokalen Speicher geladen, sofern sie zu einem früheren Zeitpunkt schon einmal vom Server geladen wurden. Sollten auch im lokalen Speicher keine Quiz vorhanden sein, bekommt der Benutzer eine Fehlermeldung, dass er den Vorgang mit funktionierender Internetverbindung wiederholen sollte und es momentan nicht möglich ist, ein Quiz zu beantworten. Da die HTML-Dateien, sowie die Bilder und Videos lokal auf dem Gerät gespeichert sind und nicht vom Server ausgeliefert werden, wird für den Benutzer die Kapitelstruktur erzeugt und angezeigt. Danach kann der Benutzer seine nächste freie oder eine bereits absolvierte Lektion starten. Dafür wählt er die gewünschte Lektion aus und je nach Typ bekommt er eine neue Ansicht.

Bei der Wahl eines Unterkapitels bekommt er eine Ansicht der dazugehörigen HTML-Datei. Der Benutzer kann entweder wieder zur Kapitelübersicht zurück, oder, wenn er der Meinung ist alles verstanden zu haben, das Unterkapitel, mit Klick auf den entsprechenden Button, abschließen.

Durch die Wahl eines Quiz kann er die Fragen zum dazugehörigen Kapitel beantworten und absenden. Dadurch erhält er bei entsprechender Internetverbindung direkt eine detaillierte Auswertung. In beiden Fällen wird für den Benutzer die nächste Lektion freigeschaltet und eine Abschlussmeldung, bzw. ein Antwortsatz an den Server gesendet.

Fragebogenmodul:

Der Ablauf wird in Abbildung 2, Abbildung 3, Abbildung 4 und Abbildung 5 genauer gezeigt und hier nur grob beschrieben. Der Start kann hierbei über das Menü oder durch den Klick auf eine entsprechende Benachrichtigung erfolgen. Im zweiten Fall erfolgt eine direkte Weiterleitung zur entsprechenden Ansicht der Fragebogenstruktur, wodurch der Benutzer den gewünschten Fragebogen sofort beantworten kann. Beim Start des Moduls werden die entsprechenden Fragebögen vom Server geladen. Dabei werden die geladenen Fragebögen gespeichert und eventuelle Benachrichtigungen werden gesetzt. Sollte dies nicht funktionieren, wird auch hier auf den lokalen Speicher zurückgegriffen und dem Benutzer wird die entsprechende Fehlermeldung gezeigt.

Ansonsten werden alle Fragebögen angezeigt, die momentan vom Benutzer ausgefüllt werden können. Hierbei muss er die einmaligen Fragebögen beantworten, bevor die kontinuierlich Wiederkehrenden freigeschaltet werden, wodurch der Benutzer auch Benachrichtigungen für diese Fragebögen bekommt. Der Benutzer kann nun einen Fragebogen auswählen und ausfüllen. Durch das Absenden werden zusätzlich zu seinen Antworten auch mögliche andere Daten, je nach Zustimmung des Benutzers und Bedarf des Fragebogens, zu einem Antwortsatz zusammengesetzt und an den Server gesendet. Bei vorhandener Internetverbindung bekommt der Benutzer sofort eine Auswertung und Feedback bereitgestellt.

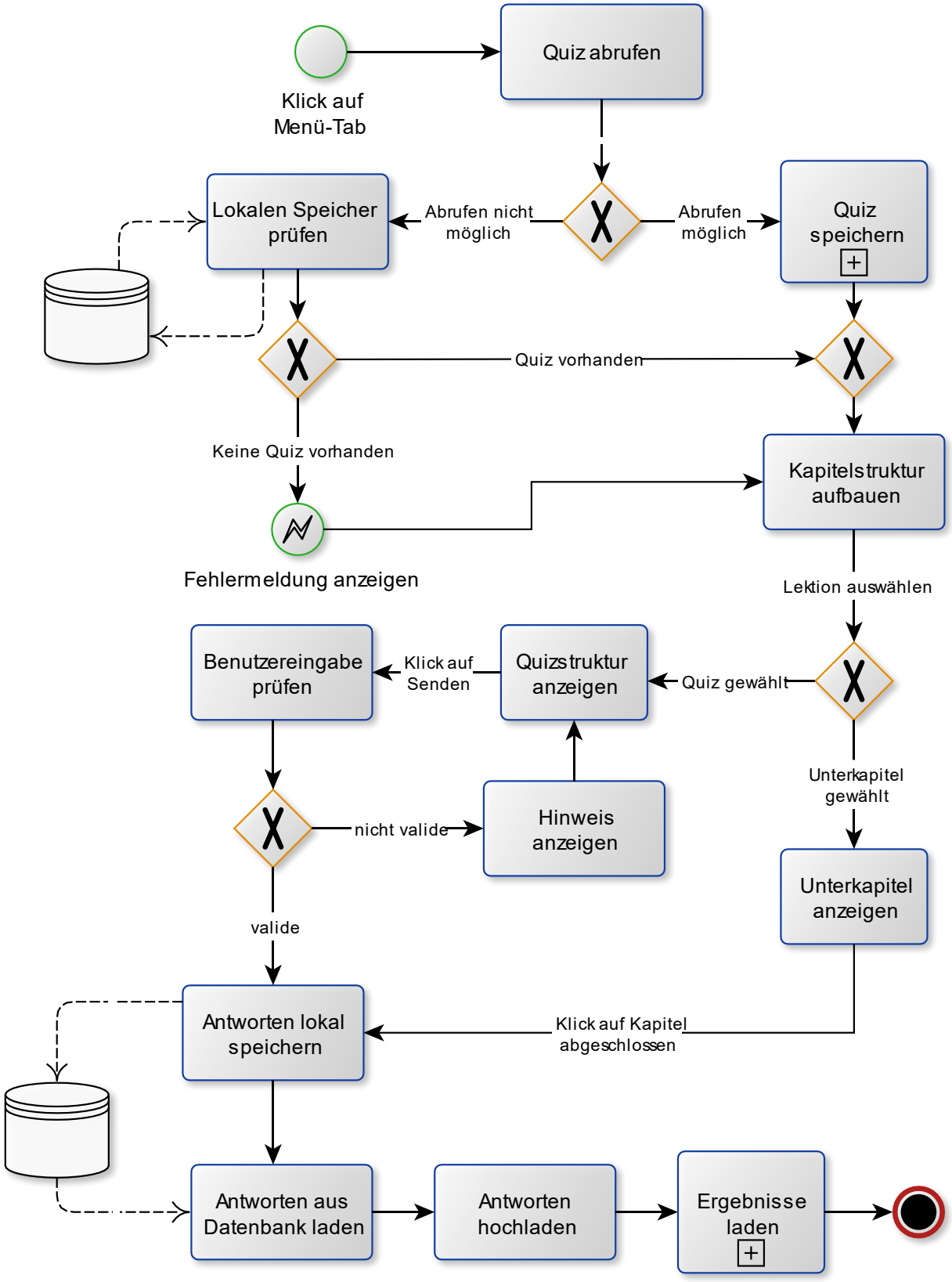

Abbildung 1: Prozessablauf-Diagramm Edukationsmodul

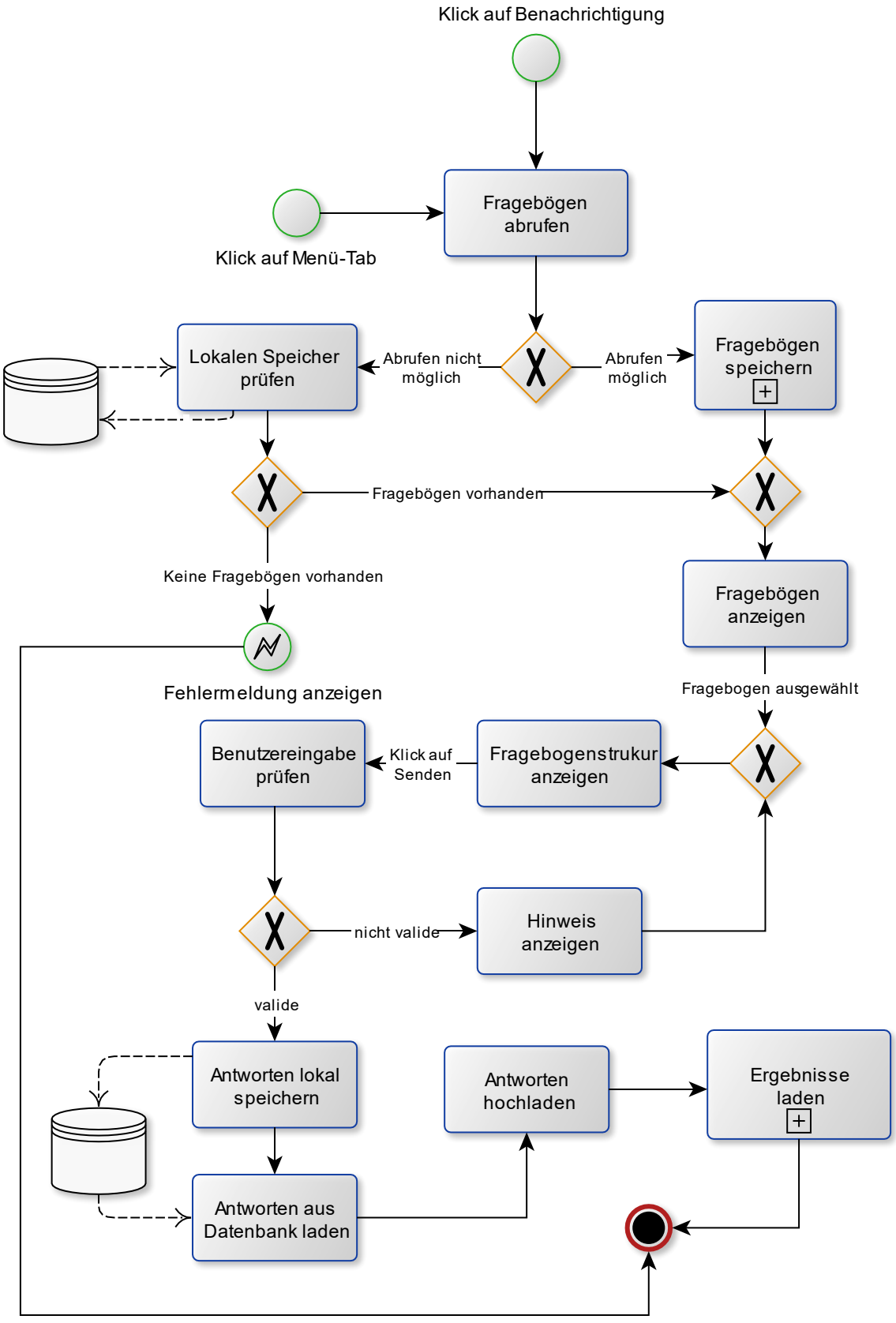

Abbildung 2: Prozessablauf-Diagramm Fragebogenmodul

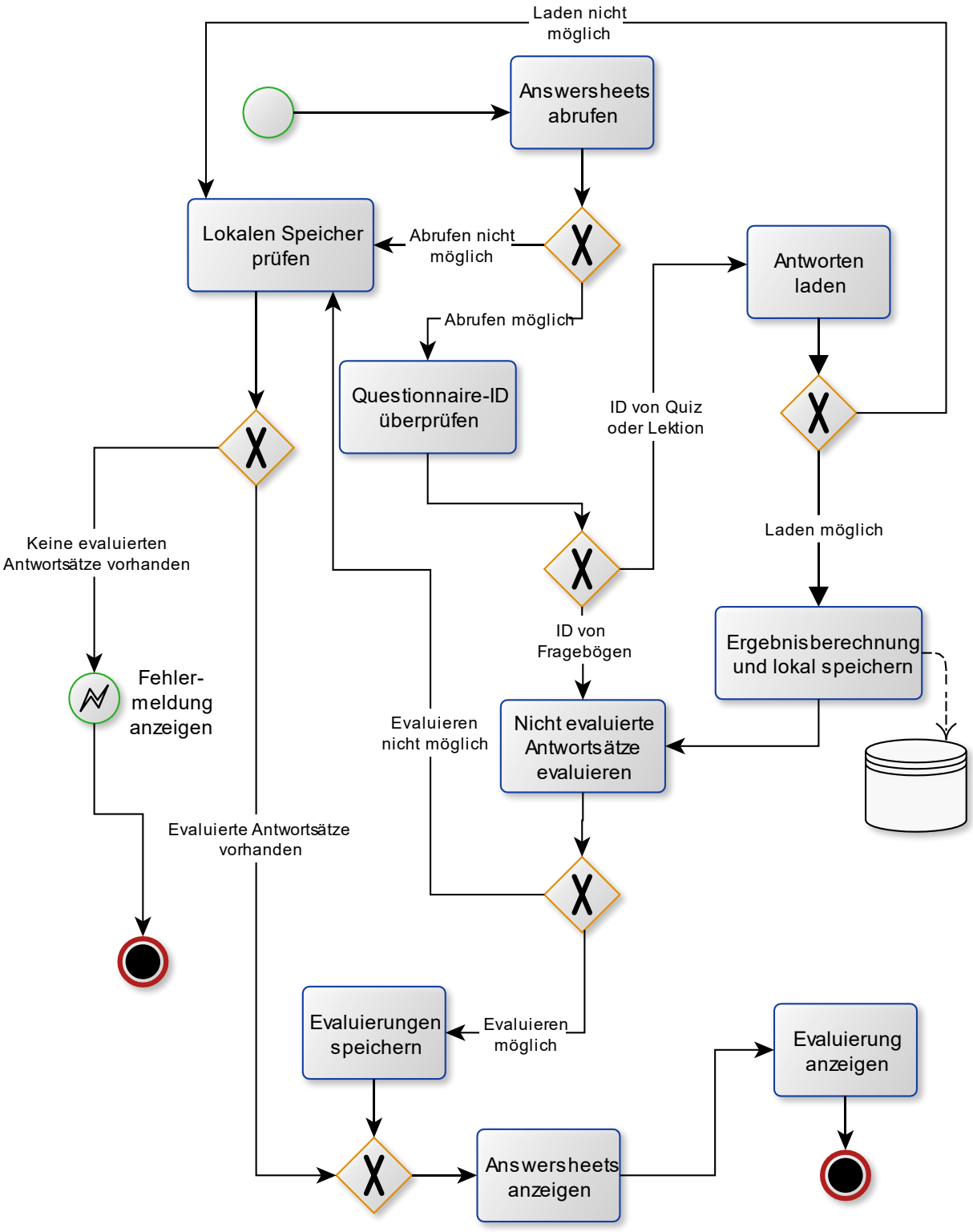

Abbildung 3: Prozessablauf-Diagramm Feedback

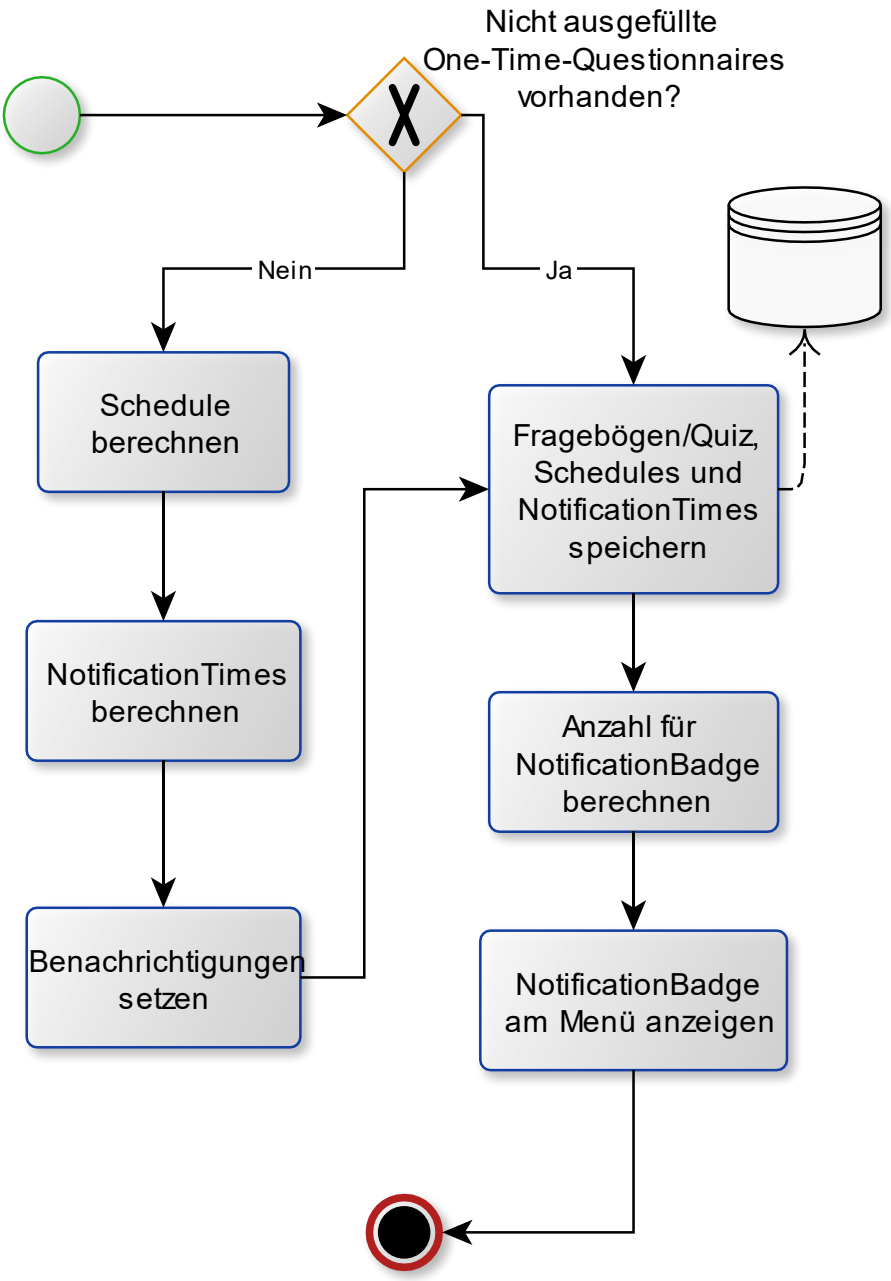

Abbildung 4: Prozessablauf-Diagramm Fragebögen speichern

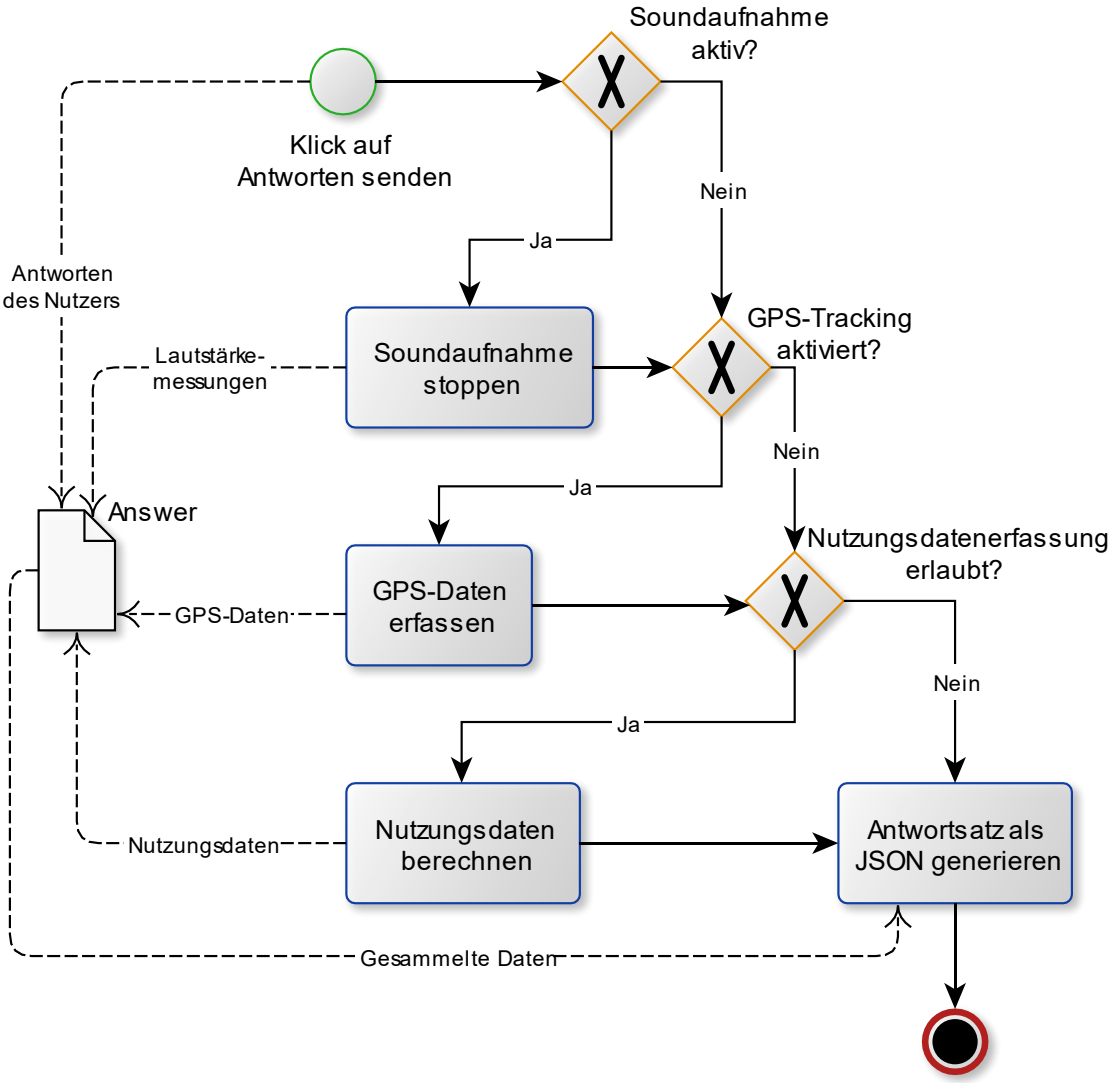

Abbildung 5: Prozessablauf-Diagramm Antwortsatz erstellen

4.3 Datenstruktur

Im Nachfolgenden wird die Datenstruktur der Module erklärt. Dabei wird auf den Aufbau der lokalen Datenbank und der Model-Klassen, die für die Datenhaltung zuständig sind, eingegangen.

4.3.1 Lokale Datenbank

Die Datenstruktur dieser App basiert auf dem, durch vom Server und der API vorgegebene, Datenmodell. Die genauere Nutzung dieses Datenmodells und ihre Umsetzung in der App, folgt in Kapitel 4.4. Hier wird eine Beschreibung der lokalen Datenbank gegeben, die nicht das komplette Datenmodell auf dem Server spiegelt, sondern lediglich den Offline-Betrieb sicherstellen soll. Dadurch wird das vorgegebene Datenmodell vereinfacht und verkleinert. Die folgende Abbildung 6 zeigt den Aufbau, der in der App implementierten SQLite[21] Datenbank.

Die Spalte Email referenziert überall den Benutzer und garantiert damit eine Offline-Nutzung, auch von mehreren Benutzern auf dem gleichen Endgerät.

In der Tabelle *Studies* werden alle vom Server geladenen, vorhandenen Studien gespeichert. Um lokal zu speichern, ob der Benutzer dieser Studie beigetreten ist, um Beitritt angefragt hat oder eingeladen wurde, wird die Tabelle um die Spalte State erweitert, in der diese Information gehalten wird.

Die Tabelle *Questionnaires* enthält alle Informationen zu den Fragebögen, die vom Server, bei der Abfrage der Fragebögen für den jeweiligen Benutzer, geliefert werden. Die Aufspaltung des JSON-Strings in eine klassenähnliche Tabelle ist hier nötig, um beim Austreten aus einer Studie, die Daten offline konsistent halten zu können. Somit werden die Fragebögen, die zur Studie gehören, automatisch gelöscht, ohne eine erneute Anfrage an den Server, um die Fragebögen zu aktualisieren. Weiter wird die Tabelle um ein Locale ergänzt, um zu speichern, in welcher Sprache die Fragebögen für den Benutzer vorliegen.

Die *Schedules* Tabelle lagert dabei die in den Fragebögen enthaltenen Zeitpläne aus. Da diese zum Teil verändert werden können und diese Änderungen nur lokal gespeichert werden sollen.

Die berechneten Benachrichtigungszeiten, die aus den *Schedules* erstellt werden und später dem *AlarmManager*[22] [23] übergeben werden, werden in der Tabelle *Notificationtimes* gespeichert. Dabei wird, für den schnelleren Zugriff, auch in Hinsicht auf die Neusetzung der Benachrichtigungen nach einem Neustart des Gerätes, der Questionnaire-Title mitgespeichert. Dies ist nötig, um dem Benutzer in der Benachrichtigung detailliert anzeigen zu können, von welchem Fragebogen diese kommt.

In der *QuestionnaireStructure* Tabelle wird lediglich der zum Fragebogen gehörende JSON-String, mit den Elementen des Fragebogens, gespeichert. Dabei wird auf eine Aufsplittung verzichtet, da der Fragebogen viele unterschiedliche Elemente vorweist und die entstehende Komplexität der Datenbank keinen Mehrwert generieren, sondern nur der Übersichtlichkeit schaden würde.

Die *Answers* Tabelle enthält den zum Hochladen bereiten JSON-String eines Antwortsets, welches von einem Benutzer ausgefüllt wurde. Die Verweise auf die ID des Fragebogens und auf den Benutzer sind

[21] SQLite: https://www.sqlite.org/index.html (08.08.2020)
[22] AlarmManager: https://developer.android.com/reference/android/app/AlarmManager.html (08.08.2020)

nötig, um alles korrekt hochladen zu können. In der Posted Spalte wird vermerkt, ob das Antwortset bereits erfolgreich an den Server weitergeleitet wurde oder noch nicht.

Die *Answersheets* und *Evaluation* Tabellen sind für die Anzeige der Ergebnisse nötig. Dabei wird die Evaluated Spalte auch benötigt, um zu speichern, ob ein Antwortset bereits evaluiert wurde.

Verschiedene Informationen, die nur aus Key-Value Paaren bestehen (zum Beispiel Login Informationen oder letztes Ergebnis eines Quiz), werden nicht als extra Datenbanktabelle gehalten, sondern in den *SharedPreferences*[23] [23] der App, da dieser Zugriff deutlich schneller und einfacher ist.

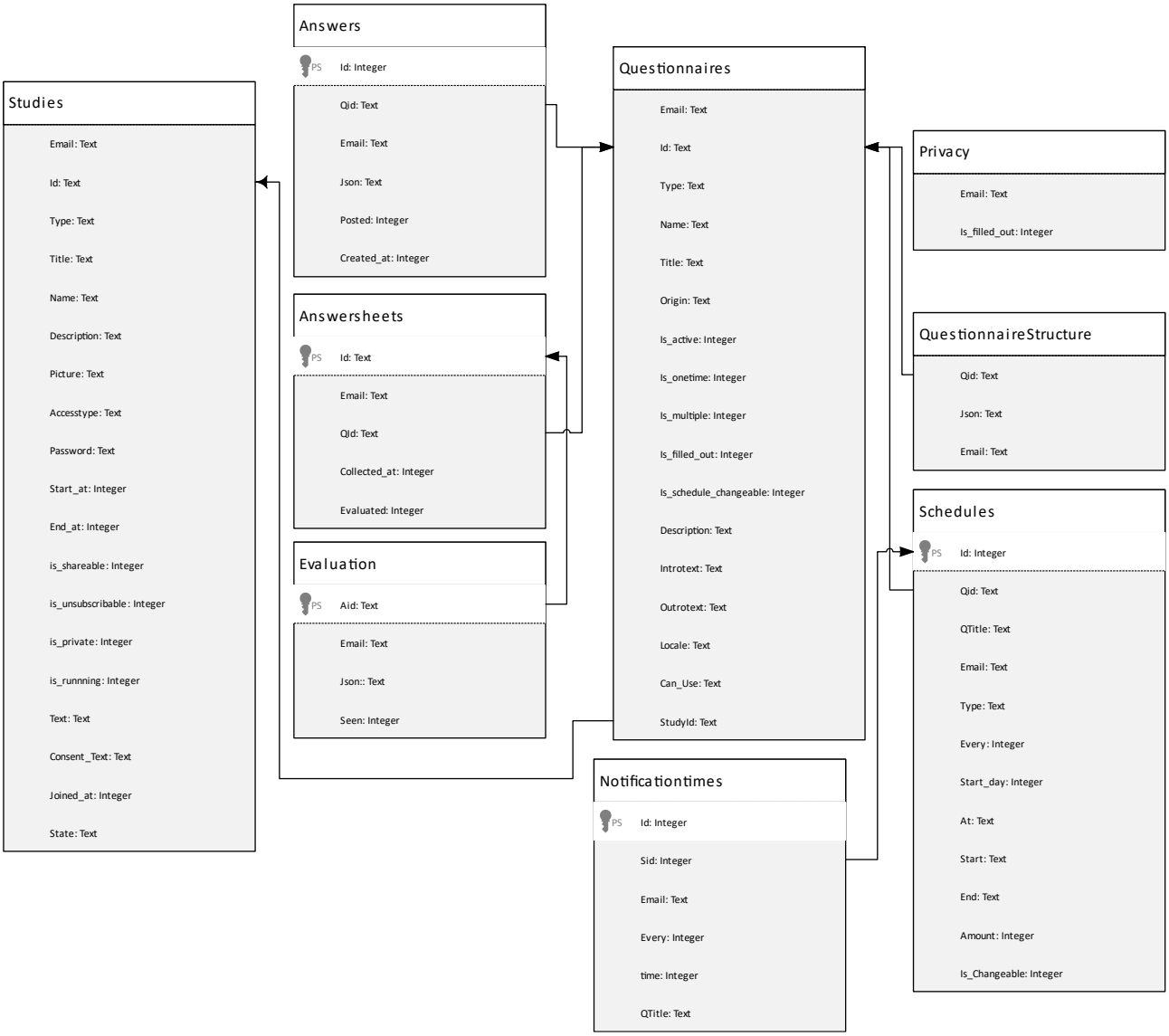

Abbildung 6: UML-Datenbankdiagramm Lokale Datenbank

[23] SharedPreferences: https://developer.android.com/reference/android/content/SharedPreferences.html (08.08.2020)

4.3.2 Model-Klassen

Im Folgenden werden die Java Klassen in generierten UML-Klassendiagrammen gezeigt, die als Datenstruktur für die verschiedenen Module dienen. Dabei gilt zu beachten, dass die gesamte Datenstruktur der UNITI-App größer ist und nicht direkt betroffene, allgemeinere Strukturen in [6] genauer beleuchtet werden.

Die Erklärung beginnt mit den Kapiteln und Lektionen für das Edukationsmodul, dann folgen die Klassen zur Berechnung der Nutzungsdaten. Danach werden alle Klassen für die Fragebögen, sowie Quiz, die in der gleichen Struktur gehalten werden, beleuchtet und abschließend die Klassen für das Feedback an den Benutzer erklärt.

Die Struktur für Kapitel und Lektionen (Abbildung 7) besteht nur lokal auf dem Gerät, da es für diese Struktur momentan keinen Gegenpart auf dem Server gibt. Um trotzdem den Fortschritt des Benutzers auch an den Server weiterleiten zu können, gibt es einen allgemeinen Fragebogen für das Edukationsmodul, der nur ein Element enthält. Darin wird, beim Abschließen eines Unterkapitels, der Name und Zeitpunkt an den Server übermittelt.

Allgemein ist die Struktur so aufgebaut, dass ein Kapitel einen Titel bekommt und eine beliebige Anzahl an Lektionen enthalten kann. Die Lektionen haben alle einen Namen, einen Typ (subChapter oder quiz), einen Titel, einen Status (completed, highlighted und blocked), sowie ein Result. Das Result beinhaltet, im Falle eines Quiz, das Ergebnis.

Der Name nimmt in diesem Fall eine besondere Rolle ein und muss über alle Lektionen einzigartig sein, da über ihn, bei Unterkapiteln, die Zuordnung der HTML-Dateien und bei einem Quiz, die Zuordnung des richtigen Fragebogens, geregelt ist. Für die HTML-Dateien findet diese Zuordnung fest über eine Funktion getHtml(), in der Lection Klasse statt.

Bei einem Quiz wird die Zuordnung dynamisch, über eine Abfrage der Datenbank, welcher Fragebogen den Namen der Lektion hat, realisiert. Das liegt daran, dass sich die ID eines Fragebogens, durch ein Update am Server ändern kann.

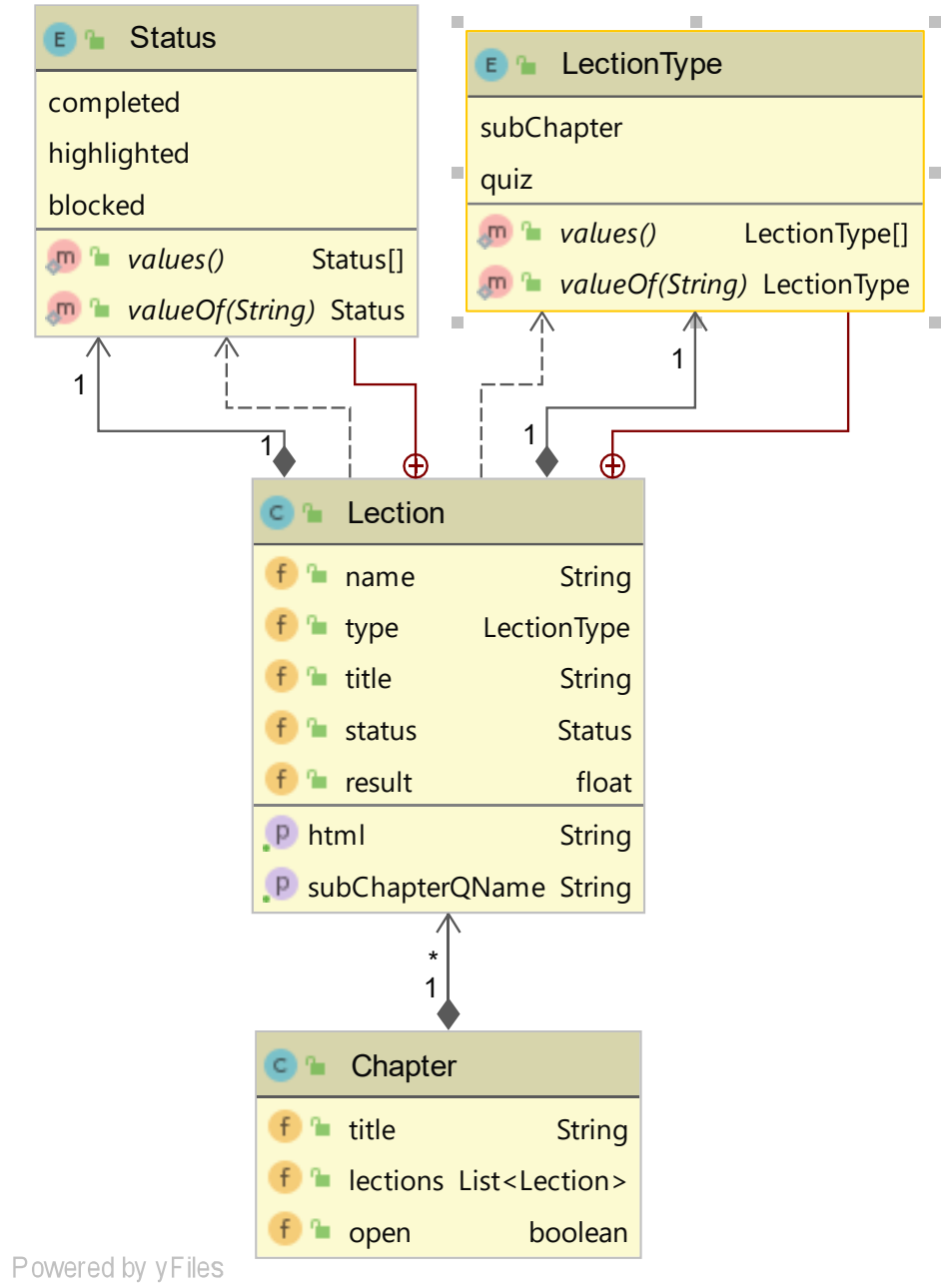

Abbildung 7: UML-Klassendiagramm Kapitel und Lektionen

Für die Nutzungsdatenerfassung gibt es mehrere Klassen (Abbildung 8), um die benötigten Informationen zu berechnen und sinnvoll zu verwalten. In der Klasse UploadUsageStats werden alle berechneten und erfassten Daten anschließend nach den Gewünschten gefiltert, da sonst auf Dauer zu viele Daten an den Server gesendet werden würden. Auf die Implementierung wird in einem späteren Kapitel (Implementierung) genauer eingegangen. In den folgenden Tabellen werden die Daten aus UploadUsageStats und den dazugehörigen Klassen, die an den Server gesendet werden, genauer erläutert. Zeitstempel sind dabei jeweils Millisekunden seit 1970, werden aber für den Upload auf Sekunden seit 1970 „vergröbert“ und Zeitwerte sind immer Gesamtwerte in Millisekunden.

UploadUsageStats

Label	Erklärung
beginTime	Zeitstempel Anfang des Erfassungszeitraums
endTime	Zeitstempel Ende des Erfassungszeitraums
apps	Liste der Apps aus packageNames (wenn sie installiert sind) mit ihren Nutzungsdaten. Genauere Erklärung siehe unten.
top5Apps	Liste der 5 meistgenutzten Apps, die nicht in der Liste apps vorkommen (Aufbau wie apps).
sleepTimes	Liste von Tupeln mit Begin und End Zeitstempel von Zeiträumen, in denen das Gerät mindestens 1 Stunde ohne aktiven Bildschirm war.
screenTime	Enthält 2 Listen mit täglichen Zeit-Werten der Nutzungszeit aller Apps (useTimes) und der Zeit, in der der Bildschirm aktiv war (activeTimes). Sortiert vom ältesten Wert (Index 0) bis zum neuesten Wert.
collected_at	Zeitstempel der Datenberechnung

Tabelle 3: UploadUsageStats Variablen Erklärung Teil 1

Apps und Top5Apps

Label	Erklärung
packageName	Name des Packages der App
completeUseTime	Summierung der täglichen UseTime Werte für diese App
completeFGServiceUseTime	Summierung der täglichen ForegroundService UseTime Werte für diese App
dailyValues	Liste der täglichen Zeitwerte und Zeitstempel der Nutzungszeit
useTime	Nutzungszeit in der die App an diesem Tag im Vordergrund (App auf Bildschirm sichtbar) war
firstUseTime	Zeitstempel, wann die App an diesem Tag das erste Mal im Vordergrund (App auf Bildschirm sichtbar) war
lastUseTime	Zeitstempel, wann die App an diesem Tag das letzte Mal im Vordergrund (App auf Bildschirm sichtbar) war
fgServiceUseTime	Zeit, in der die App an diesem Tag einen ForegroundService genutzt hat. ForegroundServices sind dabei meistens Hintergrundaktivitäten, in denen die App läuft, aber nicht im Vordergrund ist (nicht auf Bildschirm sichtbar, zum Beispiel Spotify)
firstFgServiceUseTime	Zeitstempel, wann die App an diesem Tag das erste Mal einen ForegroundService gestartet hat
lastFgServiceUseTime	Zeitstempel, wann die App an diesem Tag das letzte Mal einen ForegroundService benutzt hat

Tabelle 4: UploadUsageStats Variablen Erklärung Teil 2

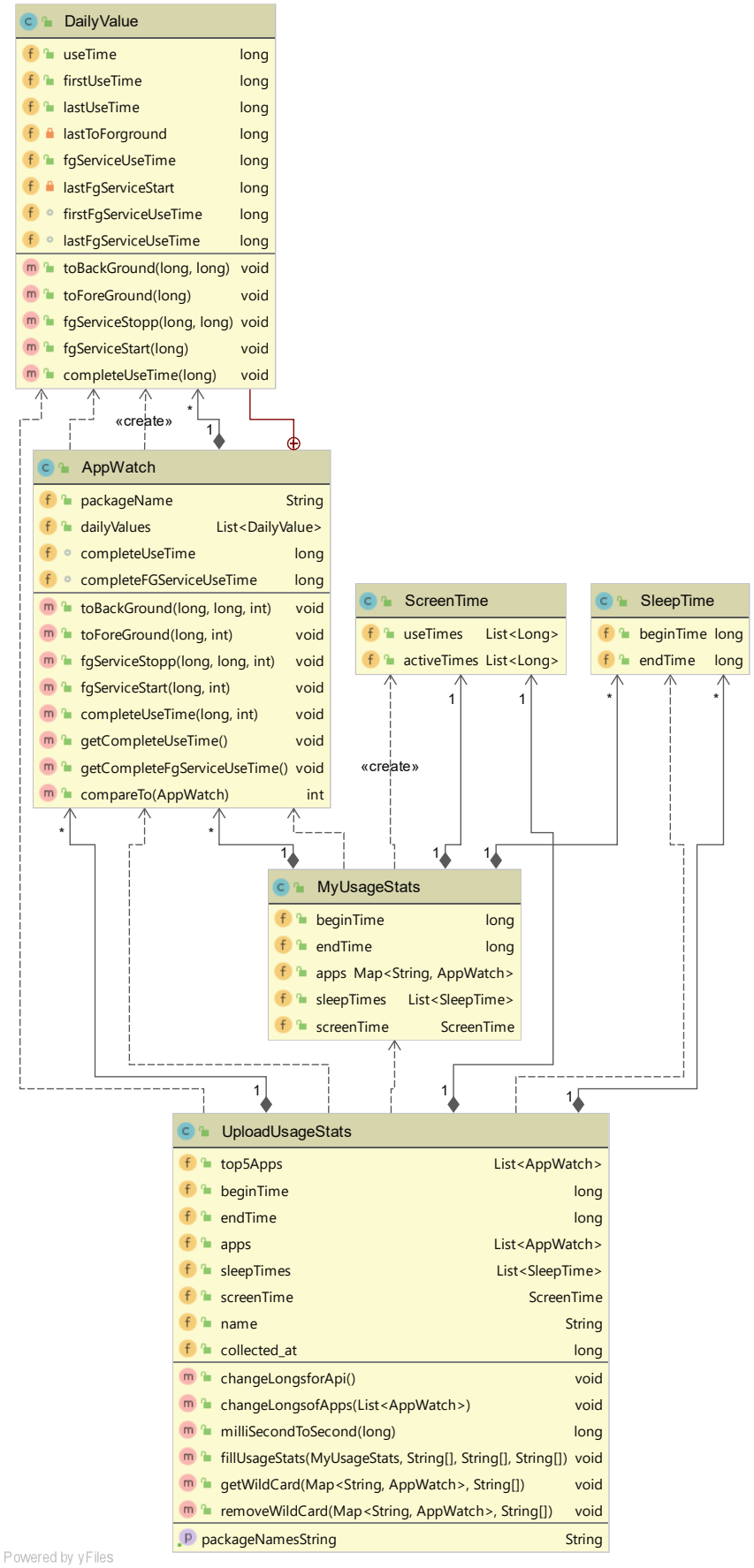

Abbildung 8: UML-Klassendiagramm UsageStats

Die Datenstruktur für Fragebögen und Quiz (Abbildung 9) sind so aufgebaut, dass es eine Klasse mit Informationen über den Fragebogen gibt, wie zum Beispiel Titel, Can_Uses und Schedules. Can_Use Objekte repräsentieren hierbei Informationen, ob bei diesem Fragebogen bestimmte zusätzliche Daten erfasst werden. Das Fragebogenmodul ermöglicht momentan das Erfassen von GPS-Daten, Lautstärkemessungen, Nutzungsdatenerfassung und Bildaufnahmen. Im UNITI-Projekt werden voraussichtlich, bis auf Bildaufnahmen, alle Optionen genutzt, wenn der Benutzer in der Studie, mit einem entsprechenden Fragebogen oder Quiz ist und er der Erfassung zugestimmt hat.

Die Schedules eines Fragebogens beschreiben die Benachrichtigungszeitpunkte. Dabei wird in der Struktur angegeben, in welchem Bereich, wie oft oder zu welchem exakten Zeitpunkt der Benutzer daran erinnert werden soll, den entsprechenden Fragebogen auszufüllen.

Die Klasse Question (Abbildung 9) ist keine direkte Abbildung der Struktur auf dem Server. Sie dient dazu, die verschiedenen Frageelemente, die in unterschiedlichen Klassen (Abbildung 10 und Abbildung 11) gehalten werden, in einer Liste verwalten zu können. Dabei werden wichtige Informationen, auf die häufiger zugegriffen werden muss als auf das eigentliche Element, wie ID, Type und Required (Angabe, ob ein Element im Antwortsatz vorhanden sein muss oder nicht), zusätzlich gehalten. Dadurch soll die Häufigkeit des Parsens der Objekte minimiert werden.

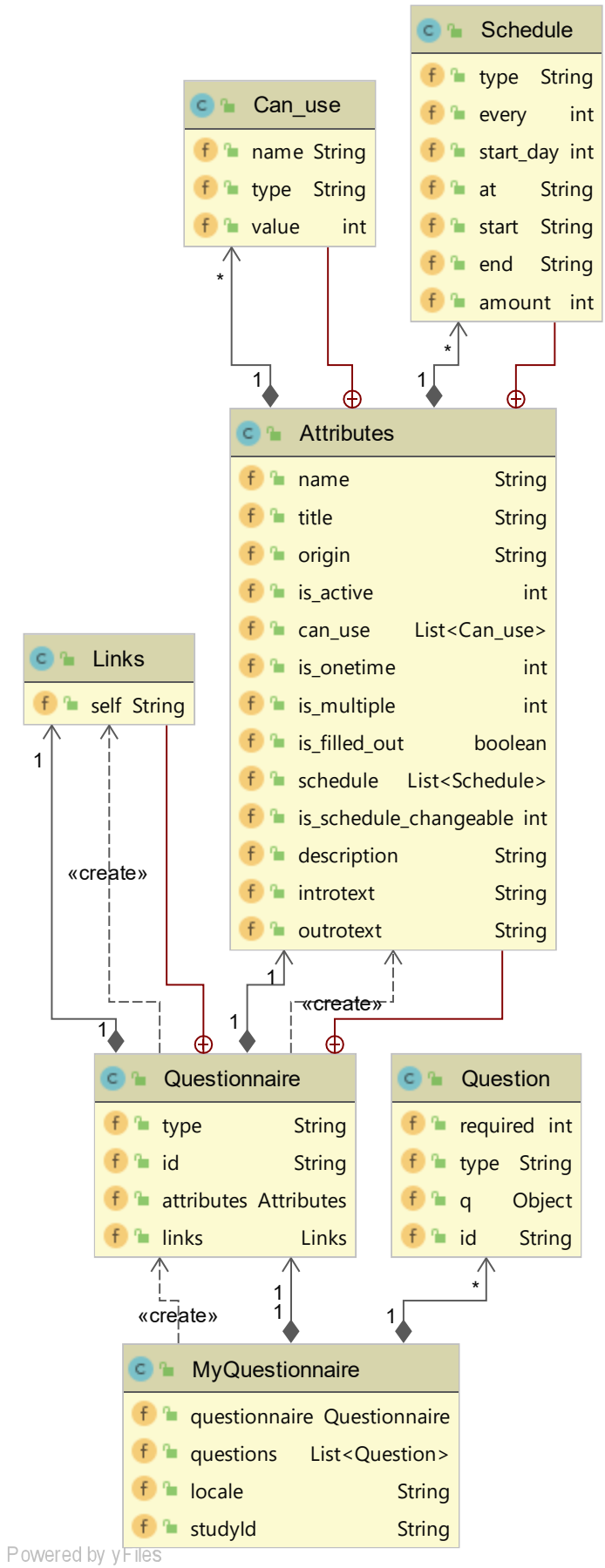

Abbildung 9: UML-Klassendiagramm Questionnaire

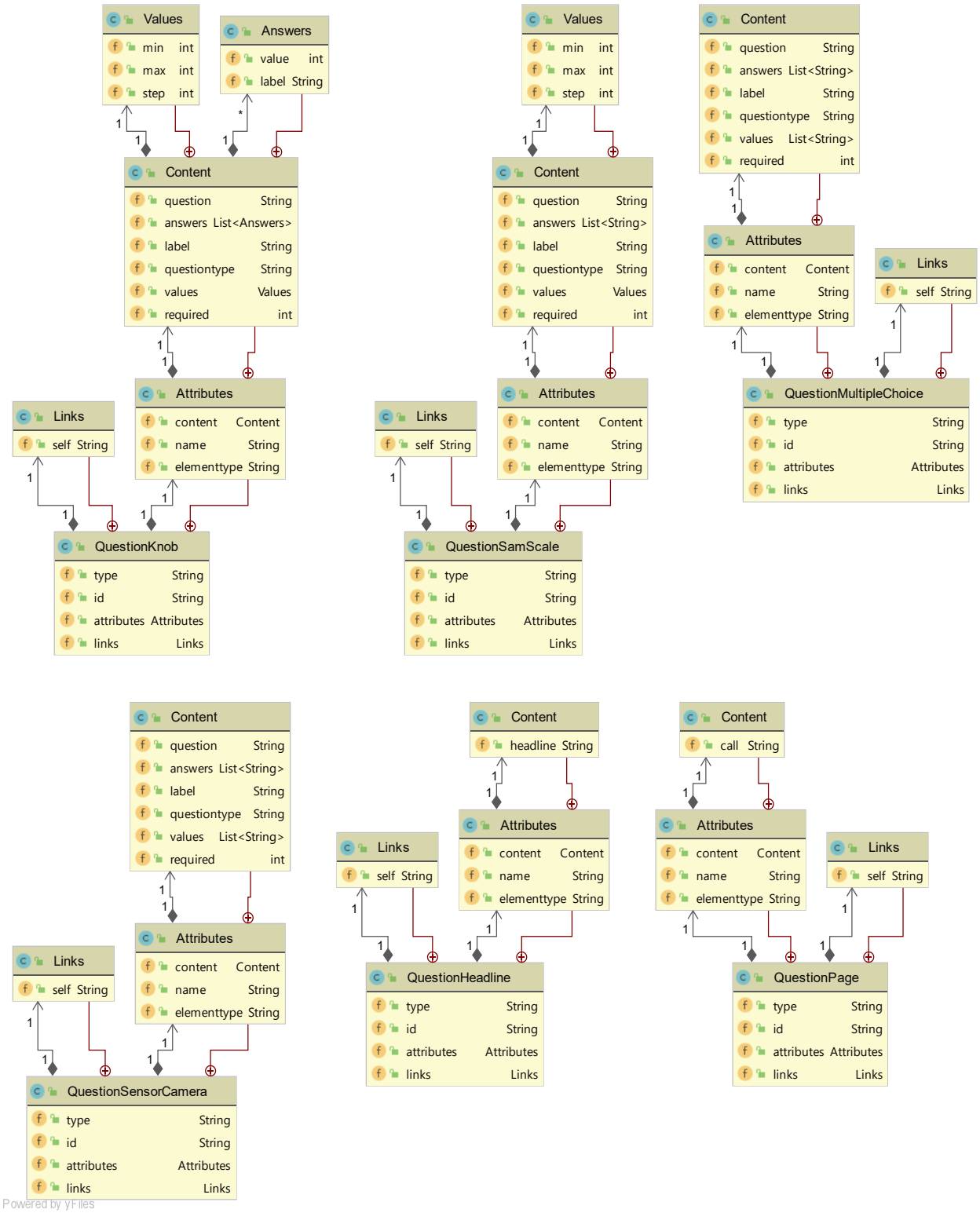

Abbildung 10: UML-Klassendiagramm Questions Teil 1

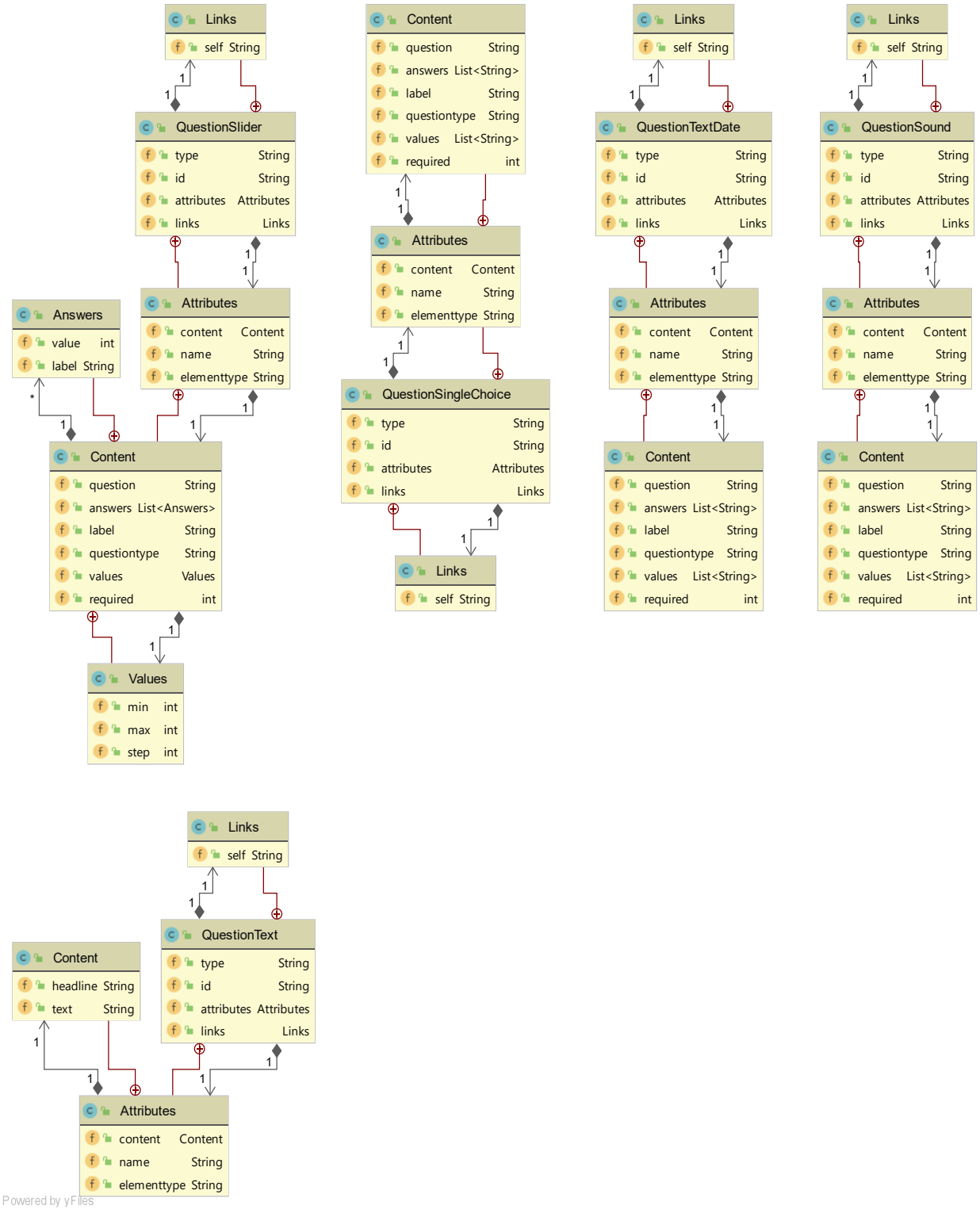

Abbildung 11: UML-Klassendiagramm Questions Teil 2

Für die Rückmeldung an den Benutzer gibt es zwei unterschiedliche Wege. Dabei werden zum einen, bei den meisten Fragebögen und Quiz, automatisch vom Server, aufgrund seiner Antworten, Feedback-Blöcke generiert, die in der App angezeigt werden. Zum anderen werden die Antworten eines Antwortsatzes vom Server zurückgelesen. Dies wird zum Beispiel beim Quiz benötigt, um zu überprüfen, wie viele der Fragen richtig beantwortet wurden, um dem Benutzer seinen Fortschritt für das entsprechende Quiz anzeigen zu können. Diese Überprüfung erfolgt lokal direkt bei der Abgabe und wird nochmal durch die Daten vom Server wiederholt, um den Fortschritt auch bei einem Gerätewechsel zu behalten. Die benutzte Datenstruktur ist in Abbildung 12 zu sehen.

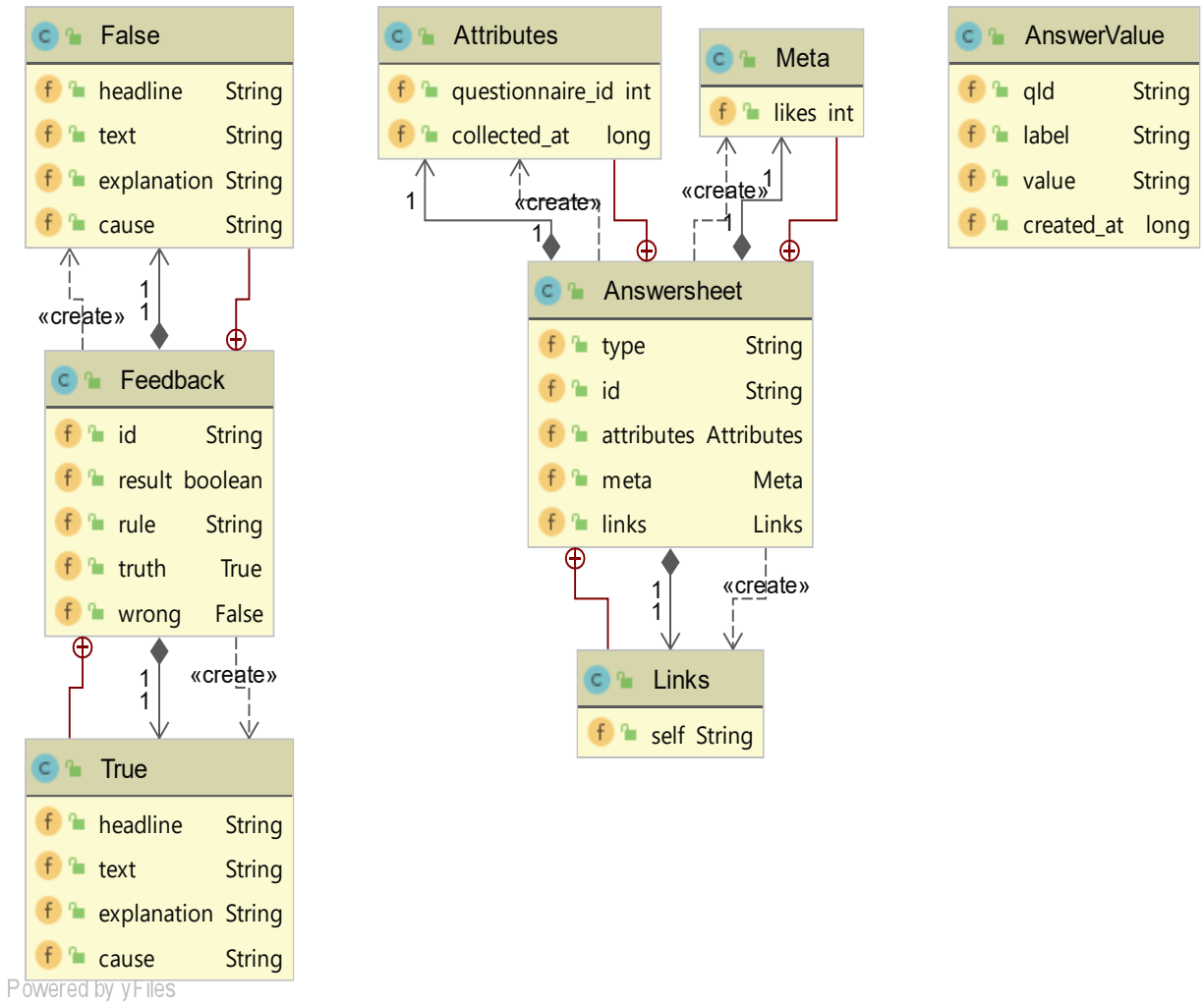

Abbildung 12: UML-Klassendiagramm Feedback

4.4 Architektur der App

Die Module der App sind grundsätzlich nach dem bekannten MVC-Pattern [24] aufgebaut, wobei für die Fragebogenelemente ViewModel-Klassen implementiert wurden. Die genaue Aufteilung und Funktionalitäten der Klassen werden im Nachfolgenden genauer beschrieben.

4.4.1 Model-View-Controller

Die folgende Abbildung 13 zeigt, wie die Module in der App, nach dem Model View Controller Design, aufgebaut sind.

Die Oberklasse für alle Views ist *AppCompatActivity*[24] [23], bzw. die davon erbende, erweiterte *ExtendedAppCompatActivity*. Diese enthält die Funktionen, um die Views nach dem Ausführen von Lade-Operationen zu aktualisieren oder Rückmeldung zu geben. Um dies zu realisieren hat jede ExtendedAppCompatActivity einen *MainManager*, dem sie sich als View übergibt. Alle Views die zusätzlich ein Menü erhalten sollen, um im Hauptmenübereich nach dem Einloggen navigieren zu können, erben von der *MenuActivity*, die dieses Menü zum Navigieren bereitstellt. Die Views bieten folgende Ansichten und Interaktionsmöglichkeiten:

- *ExtendedAppCompatActivity*: Oberklasse, die keine eigene View darstellt, sondern nur Funktionalitäten, die in allen Views benötigt werden, bereitstellt.

- *MenuActivity*: Oberklasse, die keine eigene View darstellt, sondern nur Funktionalitäten, die in allen Views mit Menü-Bedienung benötigt werden, bereitstellt.

- *AnswersheetActivity*: Auflistung der abgegebenen Antwortsätze, die evaluiert wurden.

- *ResultsActivity*: Anzeige der Evaluierung des ausgewählten Antwortsatzes.

- *QuestionnairesActivity*: Auflistung der zu beantwortenden Fragebögen.

- *QuestionnaireStructureActivity*: Anzeige des ausgewählten Fragebogens oder Quiz, mit Ausfüll- und Absendemöglichkeit.

- *ChapterActivity*: Auflistung der Kapitel und Lektionen (Unterkapitel und Quiz) mit Fortschrittsanzeige und Auswahlmöglichkeit der Lektionen.

[24] AppCompatActivity: https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html (28.08.20)

- *LectionActivity*: Anzeige eines Unterkapitels mit WebView und Möglichkeit zum Abschließen des Kapitels.

Der *MainManager* fungiert für alle Views als Controller und verwaltet die Daten zur Laufzeit. So wird zum Beispiel der Anmelde-Token vom Server oder auch die Liste der Fragebögen im *MainManager* als statische Variablen gehalten. In ihm sind Funktionen enthalten, die eine Kommunikation mit dem Server durch einen neuen *ApiConnectionTask* ausführen und die Antworten vom Server verarbeiten. Des Weiteren spricht er den *MyDBHandler* an und sorgt so für das Speichern und Laden in der lokalen Datenbank.

Zusätzlich hat er auch einen Verweis auf den *NotificationManager,* um ihm Meldungen über geänderte Schedules (Benachrichtigungszeiten) mitzuteilen, die über den *ApiConnectionTask* vom Server eingegangen sind. Dieser *NotificationManager* beinhaltet Funktionen, durch die, dem in Android verfügbaren *AlarmManager,* die Benachrichtigungszeitpunkte übergeben werden.

Der UsageStatsHelper bündelt alle Klassen und Funktionen, um die Nutzungsdaten des Gerätes zu erfassen und gefiltert zurückzugeben.

Im *ApiConnectionTask,* erbend von der Klasse *AsyncTask*[25] [23], wird eine variable Verbindung mit dem Server hergestellt. Er kann verschiedene Eingabevariablen (URL-Ende, HTTP-Methode, Header, usw.) erhalten und so unterschiedlichste Verbindungen aufbauen, die zur Kommunikation mit dem Server und der API nötig sind.

Der *MyDBHandler* erbt von *SQLiteOpenHelper*[26] [23]und liefert alle Funktionen, die nötig sind, um vom Server und dem Benutzer erhaltene Daten, zu speichern und wieder zu laden.

[25] AsyncTask: https://developer.android.com/reference/android/os/AsyncTask (01.09.2020)
[26] SQLiteOpenHelper: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html (01.09.2020)

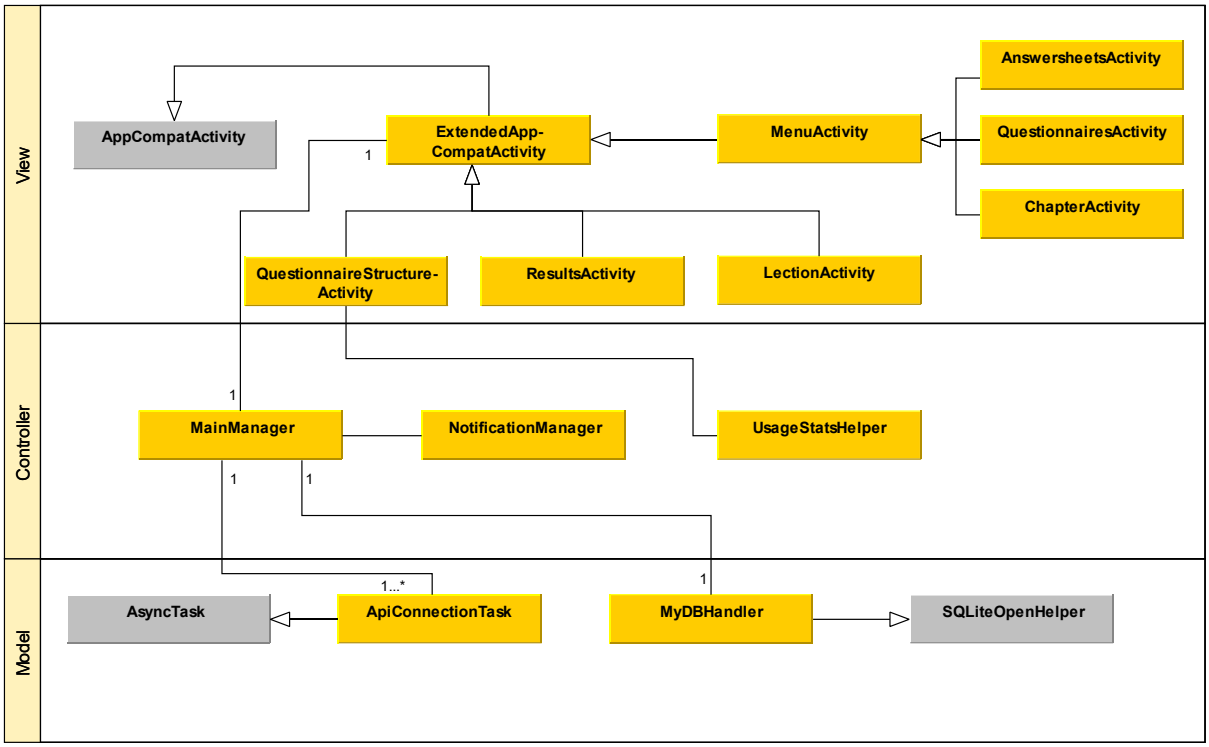

Abbildung 13: UML-Klassendiagramm Model-View-Controller

4.4.2 ViewModel

Das Datenmodell wird in Klassen umgesetzt. Hierbei wird zusätzlich eine Unterteilung in Model und ViewModel gemacht (Abbildung 14), da manche Klassen lediglich für die Anzeige und Eingabe zuständig sind und mit der internen Datenhaltung nicht direkt zu tun haben. Sie bilden nur das Bindeglied zwischen Daten und Benutzerinteraktion.

Die Klasse *QuestionView* wird verwendet, um den Fragebogen anzeigen und ausfüllen zu können. Sie realisiert die Verbindung zwischen der Datenhaltung und den reinen View Elementen. In ihr werden, während der Eingabe, die Antworten gespeichert und sie enthält eine Frage, sowie eine Eingabemöglichkeit. Letztere ist bei Headline, Page und Text null. Diese Eingabemöglichkeit wird durch die Klasse *Input_Base* realisiert, von der die Klassen *Input_MultipleChoice*, *Input_SingleChoice*, *Input_Slider, Input_SamScale, Input_Sound, Input_SensorCamera, Input_Knob* und *Input_TextDate* erben. Diese Klassen stellen View[27] [23] Elemente, zur Eingabe von bestimmten Fragetypen, bereit. *Input_TextDate* kann hier zum Beispiel, je nach Fragetyp, verschiedene Eingabearten für das Eingabefeld haben, zum Beispiel ein Datum, Multiline oder nur einen String.

Die Klasse *Question* enthält, als Object[28] [23] Variable, eine der verschiedenen Fragetypen, die sich in ihrem Aufbau unterscheiden. So wird ermöglicht, dass die unterschiedlichen Fragetypen in einer Liste gehalten werden können. Der Aufbau der verschiedenen Fragetypenklassen (*QuestionHeadline*, *QuestionText*, *QuestionMultipleChoice*, *QuestionSingleChoice*, *QuestionSlider*, *QuestionKnob*, *QuestionSensorCamera, QuestionSamScale, QuestionPage, QuestionSound* und *QuestionTextDate*)

[27] View: https://developer.android.com/reference/android/view/View.html (03.09.2020)
[28] Object: https://developer.android.com/reference/java/lang/Object.html (03.09.2020)

richtet sich nach den JSON Objekten, die vom Server übermittelt werden. So können sie, mithilfe von *Gson*[29], direkt aus dem JSON-String in Java-Objekte gewandelt werden.

So auch bei der Klasse *Questionnaire*. Die Extraklasse *MyQuestionnaire*, die jeweils einen *Questionnaire* enthält, dient dazu, zusätzliche Informationen, die nicht explizit, in den vom Server gelieferten Objekten vorkommen, zu speichern (zum Beispiel zugehörige StudyId und Liste an zugehörigen Fragen).

Die Klasse *MySchedule* dient zur Verarbeitung der Benachrichtigungen, die für einen Fragebogen möglich sind. Die daraus entstehenden Benachrichtigungszeiten werden durch die Klasse *NotificationTime* verwaltet.

Eine Instanz der Klasse *Answer* beinhaltet alle Antworten, von einem Benutzer für einen Fragebogen, als fertigen JSON-String zusammengesetzt und dient zum Hochladen dieser Antworten an den Server.

Die Klasse *Answersheet* wird benötigt, um alle Antwortsätze, die der Benutzer an den Server übermittelt hat, zu organisieren und deren Evaluation anzufragen oder dem Benutzer zu zeigen, dass sie schon evaluiert wurden und er die Ergebnisse ansehen kann.

Diese Ergebnisse werden über die Klasse *Feedback* verwaltet und dem Benutzer bei Bedarf angezeigt.

AnswerValue repräsentiert die Struktur einzelner Antworten eines Antwortsatzes und wird genutzt, um die Fortschritte bei einem Quiz auch über den Server und nicht nur lokal aktuell zu halten. Dies ist hauptsächlich bei einem Gerätewechsel oder einer Neuinstallation von Bedeutung.

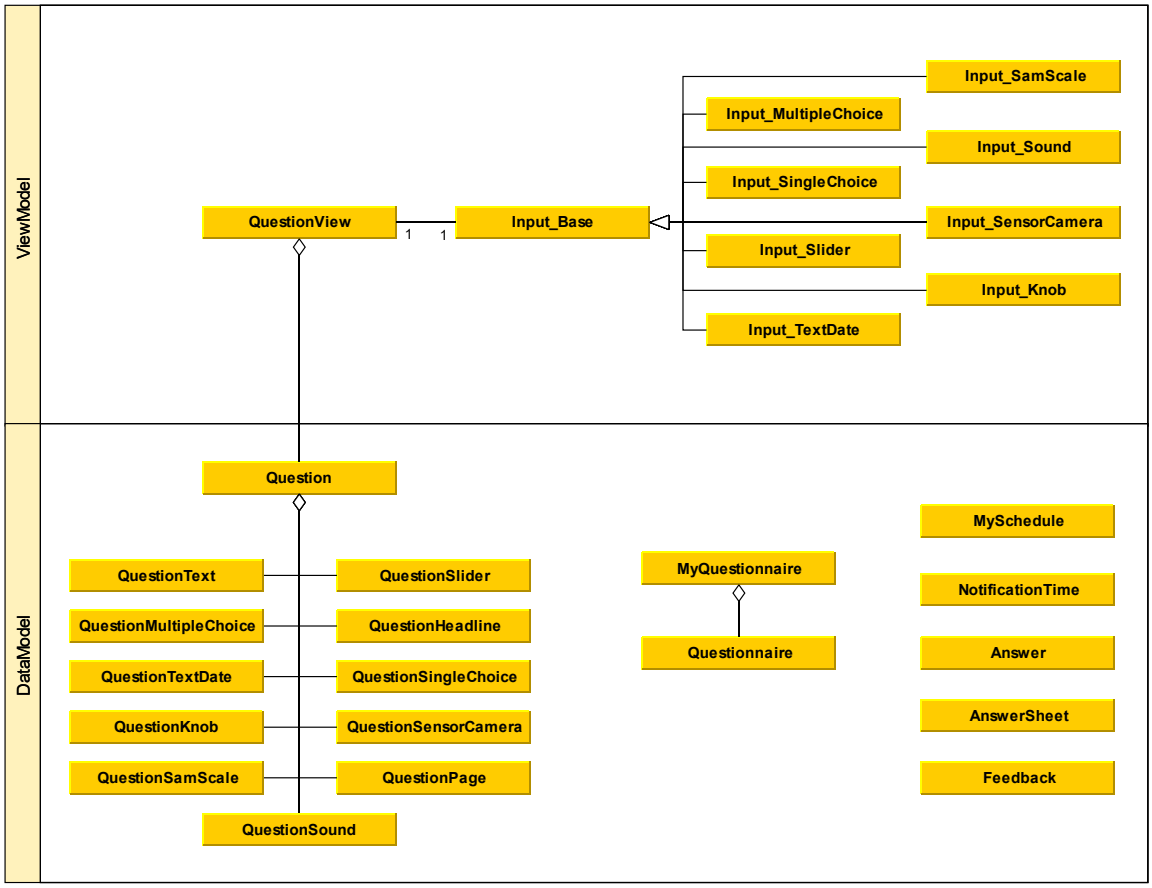

Abbildung 14: UML-Klassendiagramm ViewModel und DataModel

[29] GSON: https://google.github.io/gson/apidocs/com/google/gson/Gson.html (3.09.2020)

4.4.3 ListAdapter, Receiver, Services und Helper

Die ListAdapter in Abbildung 15 sind für *ListViews*[30] [23] in den namentlich dazugehörigen Activitys, die schon in Abbildung 13 zu sehen waren. Die Klasse *JsonParser* wird hier auch genannt, ist aber lediglich eine Hilfsklasse. Darin sind Funktionen enthalten, die entweder Informationen aus einem JSON-String entnehmen und zurückgeben oder Informationen in einen JSON-String verpacken, um diese an den Server senden zu können.

Der *NotificationPublisher* ist eine Klasse, die von *BroadcastReceiver*[31] [23] erbt und auf die dem *AlarmManager* übergebenen Broadcasts reagiert. Er erstellt dabei eine Benachrichtigung mit Text und Titel und meldet es dann dem Benutzer. Beim *NotificationPublisherFB* ist die Funktionalität ähnlich, allerdings wird hier auf das Erhalten von Feedback vom Server reagiert, woraufhin der Benutzer eine entsprechende Benachrichtigung erhält.

Da die Zeiten im *AlarmManager* nach einem Neustart nicht mehr vorhanden sind, wird auch ein *BootCompleteReceiver*, der ebenfalls von *BroadcastReceiver* erbt, benötigt. Dieser wird bei einem Neustart des Gerätes aktiviert und startet einen von *IntentService*[32] [23] erbenden *SetNotificationService*, der die in der Datenbank gespeicherten Benachrichtigungszeitpunkte im *AlarmManager* neu anlegt.

Der *UploadService* wird zu bestimmten Zeitpunkten von der App angestoßen, um auch im Hintergrund, ohne speziellen Anstoß des Benutzers, eine Kommunikation mit dem Server zu ermöglichen. Zum Beispiel um Antwortsätze hochzuladen, bei denen man zum Zeitpunkt der Abgabe möglicherweise keine Internetverbindung hatte.

Die folgenden Adapter erben alle von *ArrayAdapter*[33] [23].

Der *QuestionnaireListAdapter* zeigt alle verfügbaren Fragebögen in eigenen Blöcken an, mit Titel und, wenn vorhanden, speziellem Bild der Studie. Dazu jeweils ein Button mit dem der entsprechende Fragebogen geöffnet werden kann.

Beim *AnswersheetListAdapter* werden die verfügbaren und bereits evaluierten Antwortsätze mit dem Titel des Fragebogens und dem Datum der Erstellung des Antwortsatzes aufgelistet. Mit einem Klick auf einen Block wird zur ResultsActivity gewechselt und das Feedback wird angezeigt.

Durch den *ResultsListAdapter* werden die Feedbacks visualisiert. Für jedes Element wird ein Titel und der dazugehörige Beschreibungstext dargestellt.

Bei Elementen, die mit *Spinnern*[34] [23] realisiert werden, wird der *SpinnerAdapter* genutzt. Dabei dient er hauptsächlich zur Füllung der Spinner mit den entsprechenden Daten.

[30] ListView: https://developer.android.com/guide/topics/ui/layout/listview.html (05.09.2020)
[31] BroadcastReceiver: https://developer.android.com/reference/android/content/BroadcastReceiver.html (05.09.2020)
[32] IntentService: https://developer.android.com/reference/android/app/IntentService (05.09.2020)
[33] ArrayAdapter: https://developer.android.com/reference/android/widget/ArrayAdapter.html (05.09.2020)
[34] Spinner: https://developer.android.com/guide/topics/ui/controls/spinner.html (05.09.2020)

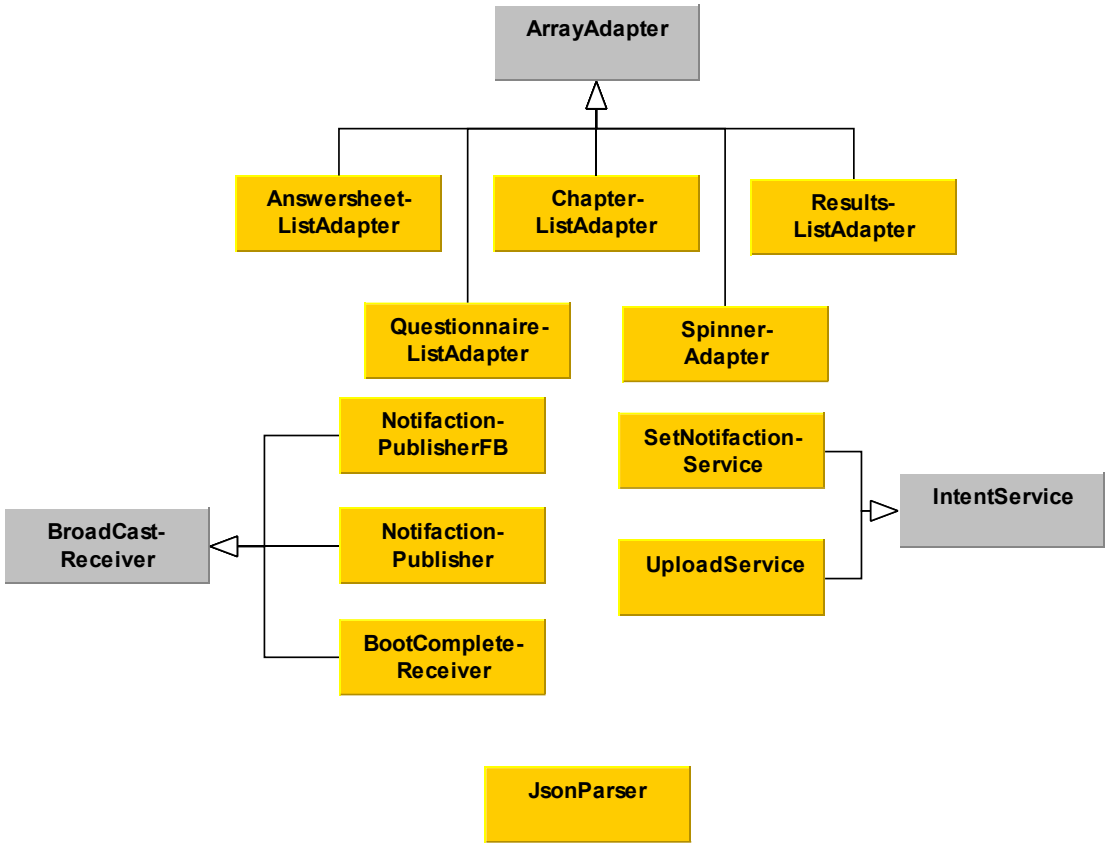

Abbildung 15: UML-Klassendiagramm ListAdapter, Receiver und Services

5 Implementierung

In diesem Kapitel werden bestimmte Bereiche der Implementierung der Module, die für die Erfüllung der Anforderungen nötig sind, genauer beleuchtet. Dafür wird zuerst gezeigt, wie die App die automatischen Benachrichtigungen umsetzt (5.1). Danach wird erklärt, wie die Kapitelstrukturierung und die Berechnung des Status, realisiert wurde (5.2). Darauf folgt in Kapitel 5.3 ein Einblick in die Vererbungsstruktur der Fragebogenelemente für die Ansicht, sowie die Erfassung von Lautstärke und GPS. Zum Abschluss wird die Implementierung der Library für die Nutzungsdatenerfassung genauer beschrieben (5.4).

5.1 Benachrichtigungen

Da die Hauptfunktion des Fragebogenmoduls das regelmäßige Ausfüllen eines Fragebogens ist, stellt die App eine Funktion bereit, die den Benutzer zu genau dieser Aufgabe auffordert und ihn daran erinnert. Dabei gibt es die Möglichkeit, diese Benachrichtigungen von einem Server über eine Internetverbindung an das Gerät zu senden. Dafür gäbe es den *Google Firebase Cloud Messaging* Service[35]. Da jedoch eine der Anforderungen an die App die Bedienbarkeit und Funktionalität, ohne eine dauerhafte Internetverbindung ist, werden die Benachrichtigungen lokal über das jeweilige Gerät erstellt.

In Android werden diese Benachrichtigungen mithilfe eines *AlarmManager* implementiert, der einen Broadcast zu dem gewünschten Zeitpunkt sendet, welcher wiederum von einem Receiver empfangen wird. In diesem Receiver wird die eigentliche Benachrichtigung erstellt und dem Benutzer auf dem Gerät angezeigt. Da die gespeicherten Zeitpunkte im *AlarmManager* einen Neustart des Gerätes nicht überdauern, müssen die Benachrichtigungszeiten nach einem Neustart erneut beim *AlarmManager* registriert werden. Dies wird mittels eines Receivers erreicht, der durch das Neustarten des Gerätes aktiviert wird und in seiner *onReceive* Methode den *SetNotificationService* startet. Dieser wiederum ruft den *MyNotificationManager* auf, um die Benachrichtigungszeitpunkte wieder neu beim *AlarmManager* zu registrieren. Dieser Vorgang wird in der folgenden Abbildung 16 gezeigt.

[35] Google Firebase Cloud Messaging: https://firebase.google.com/docs/cloud-messaging/ (28.09.2020)

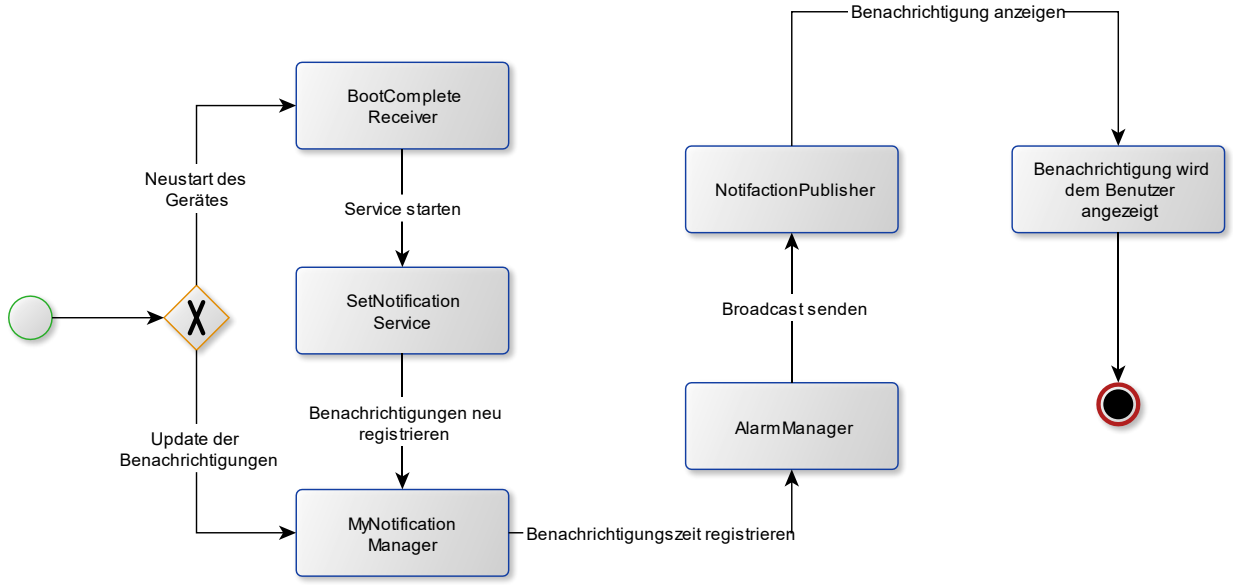

Abbildung 16: Ablaufskizzierung Benachrichtigungen

Die Benachrichtigungszeiträume für den *AlarmManager* werden in der Klasse *MyNotficationManager* erstellt.

Dafür wird, für eine Liste von *MySchedules*, eine Liste von *NotificationTimes* generiert, die die Zeitpunkte der Benachrichtigung repräsentieren. Dies wird mit der Methode in Listing 1 gemacht.

Zuerst wird eine Liste von *NotificationTime* erstellt (Zeile 2). Daraufhin wird geprüft, für welchen Fragebogen bereits Benachrichtigungen erstellt werden sollen. Die in den Shared Preferences gespeicherten Offsets der Studien beziehen sich auf den Zeitpunkt, an dem der einmalige Fragebogen der Studie ausgefüllt wurde. Die Berechnung der Benachrichtigungszeitpunkte bezieht sich immer auf den entsprechenden Offset. Danach wird, aufgrund des Schedule-Typs, unterschieden, wie die *Calendar*[36] [23] Objekte, die benötigt werden, um die Zeit einzustellen, erstellt werden sollen, da es beim Typ „fixed“ nur einen Zeitpunkt gibt und bei „between“ mehrere möglich sind. Bei „fixed“ wird, mit dem angegebenen Wochentag und der Zeit, von Zeile 16 bis 24 ein Calendar erstellt. Aus diesem Calendar wird ein NotificationTime Objekt erstellt und der Liste hinzugefügt (Zeile 25-26). Bei „between“ werden zuerst zwei Calendar erstellt. In Zeile 32-39 mit dem Startzeitpunkt und in Zeile 42-49 mit dem Endzeitpunkt. In Zeile 40 und 50 werden diese Zeitpunkte in Millisekunden gespeichert, um anschließend, falls *amount* größer als 1 ist (Zeile 51), in Zeile 52 die zeitliche Entfernung zwischen zwei Benachrichtigungen zu ermitteln. Damit werden in Zeile 57-63 die restlichen NotificationTime Objekte, mithilfe eines Calendar, erstellt und der Liste hinzugefügt. In Zeile 55 und 56 wurden bereits der Start- und Endzeitpunkt der Liste hinzugefügt.

Falls in Zeile 51 *amount* nicht größer als 1 ist, wird lediglich ein NotificationTime, welcher zwischen Start- und Endzeit liegt der Liste hinzugefügt (Zeile 67). Danach wird unterschieden, ob die Zeitpunkte für den AlarmManager sind (Zeile 76) oder für die Berechnung des NotificationBadge an der BottomNavigationView[37] [23]. Für den AlarmManager müssen die in der Vergangenheit liegenden

[36] Calendar: https://developer.android.com/reference/java/util/Calendar.html (25.09.2020)
[37] BottomNavigationView: https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView (25.09.2020)

Zeitpunkte, gemäß ihres Rhythmus, in die Zukunft verschoben werden. Am Ende wird in Zeile 86 die Liste mit den NotificationTimes zurückgegeben.

Listing 1: MyNotificationManager: GetNotificationTime

```
1  public List<NotificationTime> getNotificationtime(List<MySchedule> mySchedules ,
                       List<MyQuestionnaire> questionnaires, boolean forSetAlarm){
2          List<NotificationTime> returnList = new ArrayList<>();
3          SharedPreferences studysfilledOut =
             context.getSharedPreferences("studysfilledOut", Context.MODE_PRIVATE);
4          Map<Integer,Long> calculateOffsets = new HashMap<>();
5          for (MySchedule mySchedule:mySchedules) {
6              long calculateNtimesOffset = -1;
7              for(MyQuestionnaire myq : questionnaires){
8                  if(myq.questionnaire.id.equals(mySchedule.qID)){
9                      calculateNtimesOffset = studysfilledOut.getLong(myq.studyId,-1);
10                     calculateOffsets.put(mySchedule.id,calculateNtimesOffset);
11                 }
12             }
13             if(calculateNtimesOffset>0){
14                 switch (mySchedule.schedule.type) {
15                     case "fixed":
16                         Calendar c = Calendar.getInstance();
17                         c.setTimeInMillis(calculateNtimesOffset);
18                         if (mySchedule.schedule.start_day != 0) {
19                             c.set(Calendar.DAY_OF_WEEK,
                       getCalendarDayFromDayOfWeek(mySchedule.schedule.start_day));
20                         }
21                         String[] time = mySchedule.schedule.at.split(":");
22                         c.set(Calendar.HOUR_OF_DAY, Integer.parseInt(time[0]));
23                         c.set(Calendar.MINUTE, Integer.parseInt(time[1]));
24                         c.set(Calendar.SECOND, Integer.parseInt(time[2]));
25                         NotificationTime nTimef = new NotificationTime(mySchedule.id,
                       c.getTimeInMillis(), mySchedule.schedule.every, mySchedule.qTitle);
26                         returnList.add(nTimef);
27                         break;
28                     case "between":
29                         String[] starttime = mySchedule.schedule.start.split(":");
30                         String[] endtime = mySchedule.schedule.end.split(":");
31
32                         Calendar cstart = Calendar.getInstance();
33                         cstart.setTimeInMillis(calculateNtimesOffset);
34                         if (mySchedule.schedule.start_day != 0) {
35                             cstart.set(Calendar.DAY_OF_WEEK,
                         getCalendarDayFromDayOfWeek(mySchedule.schedule.start day));
36                         }
37                         cstart.set(Calendar.HOUR_OF_DAY, Integer.parseInt(starttime[0]));
38                         cstart.set(Calendar.MINUTE, Integer.parseInt(starttime[1]));
39                         cstart.set(Calendar.SECOND, Integer.parseInt(starttime[2]));
40                         long startTime = cstart.getTimeInMillis();
41
42                         Calendar cend = Calendar.getInstance();
43                         cend.setTimeInMillis(calculateNtimesOffset);
44                         if (mySchedule.schedule.start_day != 0) {
45                             cend.set(Calendar.DAY_OF_WEEK,
                       getCalendarDayFromDayOfWeek(mySchedule.schedule.start_day));
46                         }
47                         cend.set(Calendar.HOUR_OF_DAY, Integer.parseInt(endtime[0]));
48                         cend.set(Calendar.MINUTE, Integer.parseInt(endtime[1]));
49                         cend.set(Calendar.SECOND, Integer.parseInt(endtime[2]));
50                         long endTime = cend.getTimeInMillis();
51                         if(mySchedule.schedule.amount>1){
52                             long step = (endTime - startTime) /
                                        (mySchedule.schedule.amount - 1);
53                             NotificationTime nTimeS = new NotificationTime(mySchedule.id,
                               cstart.getTimeInMillis(), mySchedule.schedule.every,
                                                        mySchedule.qTitle);
54                             NotificationTime nTimeE = new NotificationTime(mySchedule.id,
                               cend.getTimeInMillis(), mySchedule.schedule.every,
                                                        mySchedule.qTitle);
55                             returnList.add(nTimeS);
56                             returnList.add(nTimeE);
57                             for (int i = 1; i <= (mySchedule.schedule.amount - 2); i++) {
58                                 Calendar cAmount = Calendar.getInstance();
```

```
59                                  cAmount.setTimeInMillis(calculateNtimesOffset);
60                                  cAmount.setTimeInMillis(startTime + (step * i));
61                                  NotificationTime nTimeA =new NotificationTime(mySchedule.id,
                                      cAmount.getTimeInMillis(), mySchedule.schedule.every,
                                                                  mySchedule.qTitle);
62                                  returnList.add(nTimeA);
63                              }
64                          }
65                          else
66                          {
67                              NotificationTime nTimeb =  new NotificationTime(mySchedule.id,
                                  (cstart.getTimeInMillis()+cend.getTimeInMillis()) / 2,
                                  mySchedule.schedule.every, mySchedule.qTitle);
68                              returnList.add(nTimeb);
69                          }
70                          break;
71                      default:
72                          break;
73                  }
74              }
75          }
76          if(forSetAlarm){
77              for(NotificationTime nt : returnList){
78                  updateNTimeWithEvery(nt,calculateOffsets);
79              }
80          }
81          else{
82              for(NotificationTime nt : returnList){
83                  updateNTimeForBadge(nt,calculateOffsets);
84              }
85          }
86          return  returnList;
87      }
```

Danach wird für jede *NotificationTime* ein Benachrichtigungszeitpunkt beim *AlarmManager* registriert. Dies geschieht mit den Funktionen in Listing 2.

Zuerst wird in Zeile 3 vom System eine Instanz von AlarmManager angefragt. Danach wird in Zeile 4 ein sich wiederholender Zeitpunkt erstellt, da die Benachrichtigung sich nach einer gewissen Anzahl von Tagen wiederholen soll. Dafür wird die Funktion getPendingIntent genutzt, die einen *Intent*[38] [23] erstellt. Dieser kennzeichnet die *NotificationPublisher* Klasse als Receiver des Broadcast und bekommt Extrainformationen, wie den Titel des Fragebogens (Zeile 7-13).

Listing 2: MyNotificationManager: scheduleNotification und getPendingIntent

```
1       private void scheduleNotification(NotificationTime nTime)
2       {
3           AlarmManager alarmManager =
              (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
4           alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,nTime.time,
                  AlarmManager.INTERVAL_DAY*nTime.every,
                  getPendingIntent(nTime.id, nTime.qTitle,nTime.sId));
5       }
6
7       private PendingIntent getPendingIntent(int id, String qTitle,int sId){
8           Intent notificationIntent = new Intent(context, NotificationPublisher.class);
9           notificationIntent.putExtra(NotificationPublisher.NOTIFICATION, qTitle );
10          notificationIntent.putExtra(NotificationPublisher.TIME_ID, id);
11          notificationIntent.putExtra(NotificationPublisher.NOTIFICATION_SID,sId);
12          return PendingIntent.getBroadcast(context, id, notificationIntent,
                                        PendingIntent.FLAG_UPDATE_CURRENT);
13      }
```

[38] Intent: https://developer.android.com/reference/android/content/Intent.html (25.09.2020)

Zuletzt wird die *onReceive* Methode des *NotificationPublishers* gebraucht. Diese wird in Listing 3 genauer beschrieben.

Zuerst wird vom System eine Instanz von *NotificationManagerCompat*[39] [23] angefragt (Zeile 3). Dann wird in Zeile 4-21 ein *Intent* für eine Activity erstellt, die beim Klicken auf die Benachrichtigung aufgerufen werden soll (hier QuestionnairesActivity). Dieser wird in Zeile 35, dem in Zeile 24 erstellten *NotificationCompat.Builder*[40] [23] übergeben. In Zeile 22-35 werden weitere Informationen der Benachrichtigung festgelegt, wie Titel, Text, Icon und AutoCancel (die Benachrichtigung verschwindet, nachdem sie geklickt wurde). Dabei wird für alle Android APIs 26+ ein *NotificationChannel*[41] [23] erstellt (Zeile 40-54), da die Benachrichtigung sonst auf neueren Geräten nicht angezeigt werden kann. Zuletzt wird mit dem *NotificationManager* die Benachrichtigung angezeigt (Zeile 37).

Listing 3: NotificationPublisher: onReceive

```
1     public void onReceive(Context context, Intent intent) {
2         c = context;
3         NotificationManagerCompat notificationManager = NotificationManagerCompat.from(c);
4         Intent repeating_intent = new Intent(context,QuestionnairesActivity.class);
5
6         int scheduleId = intent.getIntExtra(NOTIFICATION_SID,0);
7         if(scheduleId != 0)
8         {
9             dbHandler = new MyDBHandler(context,null,null,1);
10            String qId = null ;
11            qId = dbHandler.getQidFromSchedule(scheduleId);
12            if(qId!=null){
13                repeating_intent.putExtra(NOTIFICATION_QID,qId);
14                String studyName = dbHandler.getStudyNameFromQId(qId);
15                QuestionnairesActivity.StudyNames study =
                    QuestionnairesActivity.StudyNames.valueOf(studyName);
16                repeating_intent.putExtra(QuestionnairesActivity.Extra_Study,study);
17            }
18        }
19
20        int ntimeId = intent.getIntExtra(TIME_ID,0);
21        PendingIntent pendingIntent = PendingIntent.getActivity(context,
                      ntimeId,repeating_intent,PendingIntent.FLAG_UPDATE_CURRENT);
22        String qTitle = intent.getStringExtra(NOTIFICATION);
23        CHANNEL_ID = qTitle;
24        NotificationCompat.Builder builder =
                  new NotificationCompat.Builder(context,CHANNEL_ID);
25
26        createNotificationChannel();
27
28        Resources res = c.getResources();
29        String notificationText =res.getString(R.string.notification_text);
30        notificationText = notificationText.replace("{q_name}", qTitle);
31        builder.setContentTitle(res.getString(R.string.app_name));
32        builder.setContentText(notificationText).setStyle(
            new NotificationCompat.BigTextStyle().bigText(notificationText));
33        builder.setSmallIcon(R.mipmap.ic_launcher_shadesofnoise);
34        builder.setAutoCancel(true);
35        builder.setContentIntent(pendingIntent);
36        Calendar cal = Calendar.getInstance();
37        notificationManager.notify((int)cal.getTimeInMillis(),builder.build());
38    }
39
40    private void createNotificationChannel() {
41        // Create the NotificationChannel, but only on API 26+ because
42        // the NotificationChannel class is new and not in the support library
43        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
44            CharSequence name = CHANNEL_ID;
```

[39] NotificationManagerCompat: https://developer.android.com/reference/androidx/core/app/NotificationManagerCompat (25.09.2020)
[40] NotificationCompat.Builder: https://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html (25.09.2020)
[41] NotificationChannel: https://developer.android.com/reference/android/app/NotificationChannel (25.09.2020)

```
45            Resources res = c.getResources();
46            String description = res.getString(R.string.questionnaires);
47            int importance = NotificationManager.IMPORTANCE_DEFAULT;
48            NotificationChannel channel = new NotificationChannel(CHANNEL ID,
                                                                    name, importance);
49            channel.setDescription(description);
50            // Register the channel with the system; you can't change the importance
51            // or other notification behaviors after this
52            NotificationManager notificationManager =
                      c.getSystemService(NotificationManager.class);
53            notificationManager.createNotificationChannel(channel);
54        }
55    }
```

5.2 Edukationsmodul

Das Edukationsmodul hat seine Hauptfunktionalitäten in der ChapterActivity und der LectionActivity. Des Weiteren wird die QuestionnaireStructureActivity für die Quiz verwendet. Die Reihenfolge der Beschreibung der Implementierung orientiert sich am allgemeinen Ablauf für den Benutzer. Beginnend mit der Fortschritterkennungsfunktion der ChapterActivity in Listing 4.

Dabei werden zuerst in Zeile 7 die gespeicherten Informationen über abgeschlossene Lektionen aus den Shared Preferences geladen. Danach wird für jedes Kapitel und seine Lektionen überprüft, ob der Name der Lektion in den bereits abgeschlossenen Lektionen vorkommt (Zeile 14 und 29). Wenn ja, bekommt die Lektion den Status abgeschlossen, ansonsten wird weiter überprüft, ob die letzte Lektion abgeschlossen wurde (Zeile 19 und 35). Ist dies der Fall, gilt für diese Lektion der Status highlighted, da sie als nächstes vom Benutzer zu bearbeiten ist. Im letzten möglichen Fall bekommt die Lektion den Status blocked, da der Benutzer davor andere Lektionen bearbeiten soll. Der Unterschied zwischen den Lektionstypen zeigt sich darin, dass beim Quiz das gespeicherte Ergebnis eingetragen werden muss, um dem Benutzer seinen Erfolgsfortschritt anzeigen zu können.

Im Anschluss wird, mit der Anzahl der gesamten Lektionen und der Anzahl der abgeschlossenen Lektionen, der Gesamtfortschritt des Benutzers berechnet. Aufgrund dessen wird ihm sein Rang und seine noch abzuschließenden Lektionen bis zum nächsten Rang angezeigt, um seine Motivation zu fördern.

Listing 4: ChapterActivity

```
1     public void updateChaptersCompletion()
2     {
3         float lectionsCount = 0;
4         float lectionsCompleted = 0;
5         boolean lastLectionfinished = true;
6
7         SharedPreferences completedLections =
                      myView.getSharedPreferences("completedLections",
                                                  Context.MODE_PRIVATE);
8         for(Chapter chapter : chapters){
9             for (Lection lection : chapter.lections){
10                lectionsCount++;
11                switch (lection.type){
12
13                    case subChapter:
14                        if(completedLections.contains(lection.name)){
15                            lectionsCompleted++;
16                            lection.status = Lection.Status.completed;
17                        }
18                        else{
19                            if(lastLectionfinished){
20                                lection.status = Lection.Status.highlighted;
21                                lastLectionfinished = false;
```

```
22                          }
23                          else{
24                              lection.status = Lection.Status.blocked;
25                          }
26                      }
27                      break;
28                  case quiz:
29                      if(completedLections.contains(lection.name)){
30                          lection.result = completedLections.getFloat(lection.name,0);
31                          lectionsCompleted++;
32                          lection.status = Lection.Status.completed;
33                      }
34                      else{
35                          if(lastLectionfinished){
36                              lection.status = Lection.Status.highlighted;
37                              lastLectionfinished = false;
38                              lection.result = 0;
39                          }
40                          else{
41                              lection.status = Lection.Status.blocked;
42                              lection.result = 0;
43                          }
44                      }
45                      break;
46              }
47          }
48      }
49
50      float completePercentage = lectionsCompleted/lectionsCount;
51
52      if(completePercentage < 0.25){
53          tvChapterRank.setText(R.string.beginner);
54          ivChapterRank.setImageResource(R.drawable.icon_beginner);
55          tvNextRank.setText(((lectionsCount*0.25) - lectionsCompleted)+ " "+
                        res.getString(R.string.lection_till_next_rank) );
56      }
57      else if(completePercentage < 0.5){
58          tvChapterRank.setText(R.string.intermediate);
59          ivChapterRank.setImageResource(R.drawable.icon_intermediate);
60          tvNextRank.setText(((lectionsCount*0.5) - lectionsCompleted)+ " "+
                        res.getString(R.string.lection_till_next_rank) );
61      }
62      else if(completePercentage < 0.75){
63          tvChapterRank.setText(R.string.advanced);
64          ivChapterRank.setImageResource(R.drawable.icon_advanced);
65          tvNextRank.setText(((lectionsCount*0.75) - lectionsCompleted)+ " "+
                        res.getString(R.string.lection_till_next_rank) );
66      }
67      else if(completePercentage < 1){
68          tvChapterRank.setText(R.string.expert);
69          ivChapterRank.setImageResource(R.drawable.icon_expert);
70          tvNextRank.setText((lectionsCount - lectionsCompleted)+ " "+
                        res.getString(R.string.lection_till_next_rank));
71      }
72      else if(completePercentage == 1){
73          tvChapterRank.setText(R.string.master);
74          ivChapterRank.setImageResource(R.drawable.icon_master);
75          tvNextRank.setText(R.string.master_message);
76      }
77  }
```

Für die eigentliche Anzeige der Kapitelstruktur wird ein Listadapter (Listing 5) implementiert. In diesem List-Adapter findet die abschließende Berechnung der Fortschrittsanzeigewerte des Kapitels statt. Da diese sich, durch Aktionen des Benutzers, zur Laufzeit, ändern können und dadurch auch ständig aktualisiert werden müssen, werden sie nicht statisch gehalten, sondern nur zur Laufzeit berechnet und für die Anzeige verwendet. Im Listing 5 ist ein Auszug aus dem ChapterListAdapter, in dem diese Berechnung und Zuweisung für die Ansicht umgesetzt wird.

Dafür wird über die kompletten Lektionen eines Chapters iteriert und je nach Typ der Lektion eine entsprechende View generiert (Zeile 8-36 SubChapter, Zeile 37-67 Quiz). Dabei werden die dazugehörigen View-Elemente mit den enthaltenen Daten gefüllt. Währenddessen werden

Informationen über den Status oder das Ergebnis eines Quiz, für die abschließende Berechnung des kompletten Kapitels, zwischengespeichert. Des Weiteren wird hier bei den verschiedenen Lektionstypen die unterschiedliche Weiterleitung realisiert (Zeile 19-34 und 51-65).

Abschließend wird in Zeile 71 und 72 die Berechnung für das Kapitel in die entsprechenden View-Elemente geschrieben. Durch dieses allgemeine Vorgehen ist es auch im späteren Verlauf noch möglich, vom bisher geplanten Vorgehen (Kapitel mit mehreren Unterkapiteln und ein Quiz am Ende) abzuweichen. Damit wäre es machbar mehrere Quiz, auch zwischen den Lektionen, zu realisieren, ohne den Quellcode erweitern zu müssen.

Listing 5: ChapterListAdapter

```
1  int lectionscompleted = 0;
2  int lectionsCount = 0;
3  float quizresults = 0;
4  float quizCount = 0;
5  for(Lection lection : singleChapter.lections){
6      LayoutInflater lectionInflater = LayoutInflater.from(getContext());
7      View lectionView;
8      switch (lection.type){
9          case subChapter:
10             lectionsCount++;
11             if(lection.status == Lection.Status.completed){
12                 lectionscompleted++;
13             }
14             lectionView = inflater.inflate(R.layout.subchapter_layout,parent,false);
15             TextView tvLectionTitle = (TextView) lectionView.
                                                 findViewById(R.id.tvLectionTitle);
16             tvLectionTitle.setText(lection.title);
17             LinearLayout llLection = (LinearLayout) lectionView.
                                                    findViewById(R.id.llLection);
18             checkStatusofLection(lection.status,llLection);
19             llLection.setOnClickListener(new View.OnClickListener() {
20                 @Override
21                 public void onClick(View v) {
22                     String qId = myDBHandler.getQidFromName(
                                        lection.getSubChapterQName());
23                     if(qId != null){
24                         MainManager.chosenQuestionnaire = qId;
25                         Intent i = new Intent(context,LectionActivity.class);
26                         i.putExtra(LectionActivity.LectionNameExtra,lection.name);
27                         i.putExtra(LectionActivity.LectionHTMLExtra,
                                                        lection.getHtml());
28                         context.startActivity(i);
29                     }
30                     else{
31                         Toast.makeText(context,res.getString(
                                R.string.refresh_failed),Toast.LENGTH_LONG).show();
32                     }
33                 }
34             });
35      llChapterLections.addView(lectionView);
36      break;
37         case quiz:
38             quizCount++;
39             if(lection.status == Lection.Status.completed){
40                 quizresults+=lection.result;
41             }
42             lectionView = inflater.inflate(R.layout.quiz_layout,parent,false);
43             TextView tvQuizTitle = (TextView) lectionView.
                                              findViewById(R.id.tvQuizTitle);
44             tvQuizTitle.setText(lection.title);
45             LinearLayout llQuiz = (LinearLayout) lectionView.
                                                 findViewById(R.id.llQuiz);
46             TextView tvQuizCompletePercentage = (TextView) lectionView.
                                findViewById(R.id.tvQuizCompletePercentage);
47             checkStatusofLection(lection.status,llQuiz);
48
49             tvQuizCompletePercentage.setText(lection.result+"%");
50
```

```
51              llQuiz.setOnClickListener(new View.OnClickListener() {
52                  @Override
53                  public void onClick(View v) {
54                      String qId = myDBHandler.getQidFromName(lection.name);
55                      if(qId != null){
56                      MainManager.chosenQuestionnaire = qId;
57                      Intent intent = new Intent(context,
                                    QuestionnaireStructureActivity.class);
58                      intent.putExtra(QuestionnaireStructureActivity.
                                        Extra LectionName,lection.name);
59                      ((ChapterActivity)context).startActivityForResult(
                                    intent,ChapterActivity.requestCode);
60                      }
61                      else{
62                          Toast.makeText(context,res.getString(
                                R.string.refresh failed),Toast.LENGTH LONG).show();
63                      }
64                  }
65              });
66          llChapterLections.addView(lectionView);
67              break;
68      }
69  }
70
71  tvLectionsCompleted.setText(lectionscompleted+"/"+lectionsCount);
72  tvQsCompletePercentage.setText((quizresults/quizCount)+"%");
```

Durch den Klick auf ein Unterkapitel wechselt der Benutzer zur LectionActivity, wo die Informationen aus der HTML-Datei in einer *WebView*[42] [23] angezeigt werden. In Listing 6 ist zu sehen, wie die HTML-Datei in die Webview geladen (Zeile 19) und ein MyChromeClient (erbend von *WebChromeClient*[43] [23]) hinzugefügt wird (Zeile 16). Dieser sorgt für die Vollbildfunktionalität bei enthaltenen Videos. Am Ende dieser Ansicht kann er das Unterkapitel, durch einen Klick auf einen Button, abschließen und gelangt dadurch wieder zurück zur ChapterActivity. Die weitere Funktionalität dieses Buttons ist in Listing 7 genauer beschrieben.

Dabei wird zu Beginn geprüft, ob der Benutzer das Unterkapitel bereits früher schon abgeschlossen hatte. Falls dies noch nicht geschehen ist, wird eine Answer generiert, die an den Server gesendet werden kann. Dafür ist am Server ein allgemeiner Fragebogen hinterlegt, der nur ein Frageelement enthält und dem Benutzer nie angezeigt wird.

In dieses Element werden in Zeile 15-19 die entsprechenden Inhalte geschrieben, wie der Name der Lektion und der Zeitpunkt des Abschließens. Danach wird ein JSON-String erstellt (Zeile 22), um die Antwort an den Server senden zu können.

Die komplette Antwort wird daraufhin in der lokalen Datenbank gespeichert (Zeile 24) und in den Shared Preferences wird vermerkt, dass dieses Unterkapitel abgeschlossen wurde (Zeile 26-28). Abschließend wird ein UploadService angestoßen (Zeile 30-31), der die gespeicherten und noch nicht an den Server gesendeten Antwortsätze im Hintergrund hochlädt und die Ansicht wird geschlossen (Zeile 33).

Listing 6: LectionActivity OnCreate

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      myView = this;
4      super.onCreate(savedInstanceState);
5      setContentView(R.layout.activity_lection);
6
7      getSupportActionBar().setDisplayHomeAsUpEnabled(true);
8
```

[42] WebView: https://developer.android.com/reference/android/webkit/WebView (10.09.2020)
[43] WebChromeClient: https://developer.android.com/reference/android/webkit/WebChromeClient (10.09.2020)

```
9       lectionName = getIntent().getStringExtra(LectionNameExtra);
10      lectionHtml = getIntent().getStringExtra(LectionHTMLExtra);
11
12      WebView myWebView = (WebView) findViewById(R.id.wvLection);
13      myWebView.getSettings().setJavaScriptEnabled(true);
14      myWebView.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);
15
16      mClient = new MyChromeClient();
17      myWebView.setWebChromeClient(mClient);
18
19      myWebView.loadUrl("file:///android_asset/" + lectionHtml);
20
21      mContentView = (FrameLayout) findViewById(R.id.main_content);
22      mTargetView = (FrameLayout)findViewById(R.id.target_view);
23  }
```

Listing 7: LectionActivity FinishLection

```
1   public void onClickFinishLection(View view) {
2           SharedPreferences completedLections =
                            myView.getSharedPreferences(
                            "completedLections", Context.MODE_PRIVATE);
3
4           if(!completedLections.contains(lectionName)){
5               Answer answer = new Answer();
6               answer.qId = MainManager.chosenQuestionnaire;
7               SharedPreferences appInfo = getSharedPreferences("AppInfo",
                                                Context.MODE_PRIVATE);
8               answer.email = appInfo.getString("email","default");
9               List<Question> questions = manager.getQuestionsToShow();
10              List<QuestionView>allQuestionView = new ArrayList<>();
11
12              for (Question q : questions)
13              {
14                  if(q.required == 1){
15                      QuestionView qv;
16                      qv = new QuestionView(q,this, null,null);
17                      qv.results.add(lectionName);
18                      qv.collected_at = System.currentTimeMillis()/1000;
19                      allQuestionView.add(qv);
20                  }
21              }
22              answer.jsonString = JsonParser.setAnswerJSON(
                            allQuestionView,System.currentTimeMillis()/1000,
                                MainManager.local,null,null, null);
23              answer.created_at = System.currentTimeMillis();
24              manager.saveAnswer(answer);
25
26              SharedPreferences.Editor editor = completedLections.edit();
27              editor.putBoolean(lectionName,true);
28              editor.apply();
29
30              Intent serviceIntent = new Intent(getBaseContext(),UploadService.class);
31              getBaseContext().startService(serviceIntent);
32          }
33          myView.finish();
34      }
```

5.3 Fragebogenmodul

Für die Ansicht der Fragen im Fragebogenmodul gibt es, aufgrund der verschiedenen Typen der Fragen, mehrere verschiedene View-Elemente. Da diese unterschiedlichen Typen aber gemeinsame Grundfunktionalitäten für die Erfassung der Antworten besitzen, wurde eine Vererbungshierarchie eingeführt, wie in Abbildung 14 beschrieben. Die Implementierung der Basis-Klasse wird in Listing 8 genauer gezeigt.

Die Basisklasse enthält einen Verweis auf seine Datenhaltung über ein QuestionView-Objekt. Des Weiteren hat jedes Eingabeelement einen Platzhalter, in dem, in den erbenden Klassen die entsprechenden View-Elemente platziert werden. Zusätzlich gibt es einen Verweis auf ein ProgressWatcher-Objekt, das implementiert wurde, um eine Fortschrittsanzeige für den Benutzer zu generieren, wodurch er weiß wie viele Fragen noch vor ihm liegen.

Die Basisklasse stellt die Funktionen für die Weiterleitung der Eingaben in den View-Elementen zur Datenhaltung bereit. Dafür gibt es verschiedene Vorgänge, wie eine Antwort setzen, eine Antwort löschen, alle Antworten löschen oder mehrere Antworten setzen.

Im Falle einer Frage wird, wenn sie für den Fortschritt des Fragebogens nötig ist, die ID der Frage beim ProgressWatcher als beantwortet oder nicht beantwortet vermerkt und die Fortschrittsanzeige aktualisiert.

Listing 8: Input_Base

```
1 public abstract class Input_Base {
2
3     public QuestionView questionView;
4     public View input_frame;
5     public ProgressWatcher progressWatcher;
6
7     public void setResult(String result,boolean deleteOld){
8         if(deleteOld){
9             questionView.results.clear();
10        }
11        questionView.results.add(result);
12        questionView.collected_at = System.currentTimeMillis()/1000;
13        if(questionView.question.required==1){
14            progressWatcher.questionsAnswered.put(questionView.question.id,true);
15            progressWatcher.updatePB();
16        }
17    }
18
19    public void deleteResult(String resultToDelete){
20
21        questionView.results.remove(resultToDelete);
22        if(questionView.results.isEmpty()){
23                    if(questionView.question.required==1) {
24                        progressWatcher.questionsAnswered.put(
                                    questionView.question.id, false);
25                progressWatcher.updatePB();
26            }
27        }
28    }
29    public void clearResult(){
30        questionView.results.clear();
31        if(questionView.question.required==1) {
32            progressWatcher.questionsAnswered.put(questionView.question.id, false);
33            progressWatcher.updatePB();
34        }
35    }
36
37    public void setResults(List<String> newResults, boolean deleteOld){
38        if(deleteOld){
```

```
            questionView.results.clear();
        }
        questionView.results.addAll(newResults);
    }
}
```

Die Klasse ProgressWatcher dient der Fortschrittanzeige. Dafür wird ihr, bei der Initialisierung, eine Liste an Fragen, die für den Fortschritt des Fragebogens nötig sind, übergeben. Zusätzlich bekommt sie einen Verweis auf eine *ProgressBar*[44] [23] und eine *TextView*[45] [23], um dem Benutzer den berechneten Fortschritt, in Form und Text anzuzeigen.

Mit der Funktion *updatePB* wird dieser Aktualisierungsprozess angestoßen. Dabei wird die gesamte Anzahl der Fragen mit der Anzahl der bereits beantworteten Fragen verglichen.

Die Status der Fragen sind in einer HashMap, als Key-Value Paar aus ID und Boolean-Wert, gehalten.

Dadurch kann, bei einer Änderung des Status, schneller auf das entsprechende Paar zugegriffen werden. Nach der Berechnung wird die ProgressBar und die TextView mit den neuen Werten aktualisiert.

Listing 9: ProgressWatcher

```
public class ProgressWatcher {
    public ProgressBar pbQuestions;
    public Map<String,Boolean> questionsAnswered;
    public TextView tvProgress;
    public Context context;

    public ProgressWatcher (List<Question> requiredQuestions, ProgressBar pbQuestions,
                                    TextView tvProgress, Context context)
    {
          /*...*/

    }


    public void updatePB(){
        float countq = questionsAnswered.size();
        float countqAnsweredTrue = 0;
        Iterator<Map.Entry<String, Boolean>> iterator =
                                        questionsAnswered.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String,Boolean> pair = (Map.Entry<String,Boolean>)iterator.next();
            if(pair.getValue()){
                countqAnsweredTrue++;
            }
        }
        float progress = (countqAnsweredTrue/countq)*100;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            pbQuestions.setProgress((int)progress,true);
        }
        else{
            pbQuestions.setProgress((int)progress);
        }
        DecimalFormat df = new DecimalFormat("#.#");
        Resources res = context.getResources();
        tvProgress.setText(context.getString(R.string.progress)+df.format(progress)+"%");
    }
}
```

[44] ProgressBar: https://developer.android.com/reference/android/widget/ProgressBar (11.09.2020)
[45] TextView: https://developer.android.com/reference/android/widget/TextView (10.09.2020)

Die Vererbung und Implementierung der verschiedenen View-Elemente der Fragen wird beispielhaft an den 3 Klassen Input_TextDate, Input_MultipleChoice und Input_Slider gezeigt. Bei den weiteren erbenden Klassen ist das Vorgehen gleich und unterscheidet sich nur in den View-Elementen für die Eingabe.

Die Klasse Input_TextDate (Listing 10) benötigt einen *EditText*[46] [23] oder einen *DatePicker*[47] [23] für Datumsangaben. Dabei wird aufgrund des QuestionTypes unterschieden, welches View-Element genommen wird und welche Eingabemöglichkeiten der Benutzer beim EditText bekommen soll (Zeile 22-40).

Die vererbten Funktionen aus der Basisklasse werden dann in den Listener-Funktionen der View-Elemente genutzt (Zeile 46 und 56-63).

Listing 10: Input_TextDate

```
1  public class Input_TextDate extends Input_Base {
2
3      EditText et;
4      DatePicker dp;
5
6      public Input_TextDate(QuestionView questionView, Context c,
                             ProgressWatcher progressWatcher){
7          this.questionView = questionView;
8          this.progressWatcher = progressWatcher;
9          et = new EditText(c);
10         LayoutInflater inflater = LayoutInflater.from(c);
11         View v = inflater.inflate(R.layout.datepickerspinner,null);
12         if (android.os.Build.VERSION.SDK_INT >= 21) {
13             dp = (DatePicker) v.findViewById(R.id.datepickerSP);
14             ConstraintLayout cL = (ConstraintLayout) v.findViewById(R.id.clDatePicker);
15             cL.removeView(dp);
16         }
17         else {
18             dp = new DatePicker(c);
19         }
20         et.setTag(this);
21         dp.setTag(this);
22         switch (questionView.question.type){
23             case "TextDate":
24                 input_frame = dp;
25                 break;
26             case "TextArea":
27                 et.setInputType(InputType.TYPE_CLASS_TEXT |
                                    InputType.TYPE_TEXT_FLAG_MULTI_LINE);
28                 input_frame = et;
29                 break;
30             case"TextString":
31                 et.setInputType(InputType.TYPE_CLASS_TEXT |
                                    InputType.TYPE_TEXT_VARIATION_NORMAL);
32                 input_frame = et;
33                 break;
34             case "TextNumber":
35                 et.setInputType(InputType.TYPE_CLASS_NUMBER |
                                    InputType.TYPE_NUMBER_FLAG_DECIMAL);
36                 input_frame = et;
37                 break;
38             default:
39                 break;
40         }
41
42         Calendar calendar = Calendar.getInstance();
43         dp.init(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),
                   calendar.get(Calendar.DAY_OF_MONTH),
                   new DatePicker.OnDateChangedListener() {
44             @Override
45             public void onDateChanged(DatePicker datePicker, int year,
```

[46] EditText: https://developer.android.com/reference/android/widget/EditText (12.09.2020)
[47] DatePicker: https://developer.android.com/reference/android/widget/DatePicker (12.09.2020)

```
                                            int month, int dayOfMonth) {
46                 ((Input_Base)dp.getTag()).setResult(dayOfMonth+"."+month+"."+year, true);
47             }
48         });
49
50         et.addTextChangedListener(new TextWatcher() {
51             @Override
52             public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {
53             }
54
55             @Override
56             public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
57                 if(charSequence.length() == 0){
58                     ((Input_Base)et.getTag()).clearResult();
59                 }
60                 else{
61                     ((Input_Base)et.getTag()).setResult(charSequence.toString(),true);
62                 }
63             }
64             @Override
65             public void afterTextChanged(Editable editable) {
66             }
67         });
68     }
69}
```

Bei der Klasse Input_MultipleChoice (Listing 11) kommen vom Server mehr Informationen für die Antwortmöglichkeiten. Für die Eingabe des Benutzers werden *Checkboxen*[48] [23] verwendet. Dabei wird die Zuordnung der angezeigten Antwort, zum Antwortwert für den Server, mithilfe des Index in den Listen erreicht (Zeile 32 und 33). Dafür müssen die übersetzten Answers und die entsprechenden Values in der gleichen Reihenfolge vorkommen.

Auch hier wurden die Funktionen der Basisklasse wieder in der Listener-Funktion genutzt (Zeile 34-39).

Listing 11: Input_MultipleChoice

```
1 public class Input_MultipleChoice extends Input_Base{
2     List<String> answers;
3     List<String> values;
4     List<CheckBox> checkBoxes;
5     LinearLayout llCheckboxes;
6
7
8     public Input_MultipleChoice(QuestionView questionView, final List<String> answers,
                                                  final List<String>values, Context c,
                                                  ProgressWatcher progressWatcher) {
9         this.answers = answers;
10        this.values = values;
11        this.questionView = questionView;
12        this.progressWatcher = progressWatcher;
13        checkBoxes = new ArrayList<>();
14        llCheckboxes = new LinearLayout(c);
15        Resources res = c.getResources();
16        llCheckboxes.setOrientation(LinearLayout.VERTICAL);
17
18        LinearLayout.LayoutParams questionParams = new LinearLayout.LayoutParams(
              ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.MATCH_PARENT);
19        int marginbetween = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
20                (float) 10, res.getDisplayMetrics());
21        questionParams.topMargin = marginbetween;
22
23        int count = 0;
24        for (String answer : answers) {
25            CheckBox cbx = new CheckBox(c);
26            cbx.setBackgroundColor(res.getColor(R.color.myWhite));
27            cbx.setText(answer);
28            cbx.setTag(this);
29            cbx.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
```

[48] CheckBox: https://developer.android.com/reference/android/widget/CheckBox (12.09.2020)

```
30                @Override
31                public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
32                    int id = answers.indexOf(compoundButton.getText());
33                    String result = values.get(id);
34                    if (compoundButton.isChecked()) {
35                        ((Input_Base) compoundButton.getTag()).setResult(result,false);
36                    }
37                  else {
38                        ((Input_Base) compoundButton.getTag()).deleteResult(result);
39                    }
40                }
41            });
42
43            checkBoxes.add(cbx);
44
45            View divider = new View(c);
46            divider.setBackground(new ColorDrawable(res.getColor(R.color.myBlack)));
47
48            LinearLayout.LayoutParams dividerParams = new LinearLayout.LayoutParams(
                                        ViewGroup.LayoutParams.MATCH_PARENT, 4, 0);
49
50            llCheckboxes.addView(cbx,questionParams);
51            if(count < (answers.size()-1)){
52                llCheckboxes.addView(divider,dividerParams);
53            }
54            count++;
55        }
56        input_frame = llCheckboxes;
57    }
58}
```

Für die Klasse Input_Slider (Listing 12) wird eine *Seekbar*[49] [23] verwendet. Eine besondere Rolle spielt dabei die Tatsache, dass die Eingabewerte der Seekbar immer nullbasiert sind. Allerdings kann es sein, dass vom Server eine Frage kommt, bei der der Benutzer zum Beispiel einen Wert von 4-10 eingeben können soll.

Deshalb werden die Werte immer in ein nullbasiertes Format gebracht, um der Seekbar einen korrekten Wertebereich zu übergeben (Zeile 29). Allerdings muss bei der Anzeige, sowie der Ergebnissetzung, eine Rückwandlung stattfinden (Zeile 39-40 und 49-50).

Eine weitere Besonderheit bei diesem Eingabeelement ist die Anforderung, dass vor einer Eingabe durch den Benutzer, kein Startwert des Sliders zu sehen ist. Dafür wird Progress auf 0 gesetzt und der Thumb der Seekbar transparent gemacht (Zeile 31-33). Sobald der Benutzer die Seekbar einmal anklickt, wird der Thumb wieder sichtbar und der Benutzer kann den gewünschten Wert einstellen.

Dadurch soll verhindert werden, dass der Benutzer, durch eine Voreinstellung, bei seiner Antwort beeinflusst wird. Die Textanzeigen unter der Seekbar, mit minimalem, aktuellem und maximalem Wert sollen dem Benutzer die Eingabe erleichtern.

Listing 12: Input_Slider

```
1 public class Input_Slider extends Input_Base {
2     List<QuestionSlider.Attributes.Content.Answers> answers ;
3     QuestionSlider.Attributes.Content.Values values;
4     SeekBar seekBar;
5     GridLayout gridLayout;
6     CoordinatorLayout coordinatorLayout;
7     TextView minText;
8     TextView maxText;
9     TextView valueText;
10
11    public Input_Slider(QuestionView questionView,
        List<QuestionSlider.Attributes.Content.Answers> answers,
        final QuestionSlider.Attributes.Content.Values values,
                    Context c, ProgressWatcher progressWatcher){
```

[49] SeekBar: https://developer.android.com/reference/android/widget/SeekBar.html (12.09.2020)

```
        this.questionView = questionView;
        this.answers = answers;
        this.values = values;
        this.progressWatcher = progressWatcher;
        seekBar = new SeekBar(c);
        gridLayout = new GridLayout(c);
        coordinatorLayout = new CoordinatorLayout(c);
        minText = new TextView(c);
        maxText = new TextView(c);
        valueText = new TextView(c);

        gridLayout.setColumnCount(1);
        gridLayout.setRowCount(2);
        minText.setText(answers.get(0).label);
        maxText.setText(answers.get(1).label);
        valueText.setText("");

        seekBar.setMax(values.max- values.min);
        seekBar.setTag(this);
        seekBar.setProgress(0);
        Drawable transparentDrawable = new ColorDrawable(Color.TRANSPARENT);
        seekBar.setThumb(transparentDrawable);
        final Resources res = c.getResources();

        seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
                setResult((i +values.min )+"",true);
                valueText.setText(""+(i + values.min));
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
                Drawable thumb = res.getDrawable(R.drawable.seekbar_thumb);
                seekBar.setThumb(thumb);
                int progress = seekBar.getProgress();
                if(progress >=0){
                    setResult((progress +values.min )+"",true);
                    valueText.setText(""+(progress + values.min));
                }
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });

        GridLayout.LayoutParams seekbarParams = new GridLayout.LayoutParams(
                                        GridLayout.spec(0),GridLayout.spec(0));
        seekbarParams.width =GridLayout.LayoutParams.MATCH_PARENT;

        GridLayout.LayoutParams textsParams = new GridLayout.LayoutParams(
                                        GridLayout.spec(1),GridLayout.spec(0));
        textsParams.width =GridLayout.LayoutParams.MATCH_PARENT;

        CoordinatorLayout.LayoutParams minTextParams = new CoordinatorLayout.
                                    LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
                                    ViewGroup.LayoutParams.WRAP_CONTENT);
        minTextParams.gravity = Gravity.LEFT;

        CoordinatorLayout.LayoutParams valueTextParams = new CoordinatorLayout.
                                    LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
                                    ViewGroup.LayoutParams.WRAP_CONTENT);
        valueTextParams.gravity = Gravity.CENTER;

        CoordinatorLayout.LayoutParams maxTextParams = new CoordinatorLayout.
                                    LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
                                    ViewGroup.LayoutParams.WRAP_CONTENT);
        maxTextParams.gravity = Gravity.RIGHT;

        coordinatorLayout.addView(minText,minTextParams);
        coordinatorLayout.addView(valueText,valueTextParams);
        coordinatorLayout.addView(maxText,maxTextParams);

        gridLayout.addView(seekBar,seekbarParams);
        gridLayout.addView(coordinatorLayout,textsParams);

        input_frame = gridLayout;
```

```
82     }
83}
```

Um dem Benutzer auch ohne Internetverbindung direkt anzeigen zu können, wie viel Prozent der Fragen eines Quiz er richtig beantwortet hat, wurde festgelegt, dass die internen Werte (Values) aller richtigen Antworten einer Frage, zusammengerechnet 1000 ergeben müssen. Die dafür verwendete Funktion ist in Listing 13 zu sehen. Dabei wird über alle Fragebogenelemente iteriert und bei allen benötigten Fragen (Required == 1, siehe Zeile 5) überprüft, ob die Summe ihrer Antworten den Wert 1000 ergibt (Zeile 11). Am Ende wird ein Prozentwert aus der Anzahl richtiger Fragen und der Anzahl aller benötigten Fragen errechnet und ein, auf eine Kommastelle gekürzter Wert, zurückgegeben (Zeile 16-18).

Listing 13: CalculateResult

```
1 public static float calculateResult(List<QuestionView> questions){
2   float count = 0;
3   float countCorrect = 0;
4   for(QuestionView qv :questions){
5       if(qv.question.required == 1){
6           count++;
7           int resultOfQuestion = 0;
8           for (String singleResult: qv.results){
9               resultOfQuestion += Integer.valueOf(singleResult);
10          }
11          if(resultOfQuestion == 1000){
12              countCorrect++;
13          }
14      }
15  }
16  float result = (countCorrect/count)*100;
17  BigDecimal bd = new BigDecimal(result).setScale(1, RoundingMode.HALF_EVEN);
18  return bd.floatValue();
19}
```

Für die Erfassung von GPS-Daten wurde eine *GpsListener* Klasse (Listing 14) erstellt. Diese muss wiederum das Interface *LocationListener*[50] [23] implementieren. Dabei war zu beachten, dass Android die Position über mehrere Wege erfassen kann. Diese unterscheiden sich möglicherweise in ihrer Genauigkeit. Zum Beispiel, wenn die normale GPS-Erfassung, beim Aufenthalt in einem Gebäude nicht möglich ist, versucht das Gerät den Standort über eine bestehende Internetverbindung oder über Daten anderer Apps und Geräte zu erfassen.

Dabei ist die Klasse so implementiert, dass die Erfassung des Standortes einmalig stattfindet und das „Tracking“ danach sofort wieder beendet wird, um möglichst wenig Akku zu verbrauchen (Zeile 12-16). Da man zum Startzeitpunkt der Erfassung nicht weiß, welche Option der GPS-Datenerhebung momentan funktioniert, werden die LocationUpdates von allen Providern angefragt, indem 3 Objekte des GpsListener mit den jeweiligen Providern erstellt werden (Zeile 33-76). Die entstehenden Ergebnisse werden dann absteigend ihrer Genauigkeit präferiert (GPS, Network, Passive).

Listing 14: GpsListener

```
1 public class GpsListener implements LocationListener {
2     private LocationManager locationManager;
3     Context mc ;
4     public Location location = null;
5     String provider;
```

[50] LocationListener: https://developer.android.com/reference/android/location/LocationListener (14.09.2020)

```
6     public boolean tracking;
7
8     public GpsListener(Context c, String provider){
9         mc = c;
10        this.provider = provider;
11    }
12    @Override
13    public void onLocationChanged(Location location) {
14        this.location = location;
15        stopListener();
16    }
17
18    @Override
19    public void onStatusChanged(String s, int i, Bundle bundle) {
20
21    }
22
23    @Override
24    public void onProviderEnabled(String s) {
25
26    }
27
28    @Override
29    public void onProviderDisabled(String s) {
30
31    }
32
33    public void getLocation()
34    {
35        try {
36            locationManager = (LocationManager) mc.getSystemService(LOCATION_SERVICE);
37            tracking = true;
38            switch(provider){
39                case LocationManager.GPS_PROVIDER:
40                    if(locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
41                        location=null;
42                        if (location == null) {
43                            if (ActivityCompat.checkSelfPermission(mc,
                                  Manifest.permission.ACCESS_FINE_LOCATION) ==
                                  PackageManager.PERMISSION_GRANTED
                                  && ActivityCompat.checkSelfPermission(mc,
                                  Manifest.permission.ACCESS_COARSE_LOCATION) ==
                                  PackageManager.PERMISSION_GRANTED){
44                                  locationManager.requestLocationUpdates(
                                                LocationManager.GPS_PROVIDER, 0, 0, this);
45                            }
46                        }
47                    }
48
49                    break;
50                case LocationManager.NETWORK_PROVIDER:
51                    if(locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
52                        location = null;
53                        if (ActivityCompat.checkSelfPermission(mc,
                                  Manifest.permission.ACCESS_FINE_LOCATION) ==
                                  PackageManager.PERMISSION_GRANTED
                                  && ActivityCompat.checkSelfPermission(mc,
                                  Manifest.permission.ACCESS_COARSE_LOCATION) ==
                                  PackageManager.PERMISSION_GRANTED)
54                            {
55                                locationManager.requestLocationUpdates(
                                                LocationManager.NETWORK_PROVIDER, 0, 0, this);
56                            }
57                    }
58
59                    break;
60                case LocationManager.PASSIVE_PROVIDER:
61                    if(locationManager.isProviderEnabled(LocationManager.PASSIVE_PROVIDER)){
62                        location=null;
63                        if (ActivityCompat.checkSelfPermission(mc,
                              Manifest.permission.ACCESS_FINE_LOCATION) ==
                              PackageManager.PERMISSION_GRANTED
                              && ActivityCompat.checkSelfPermission(mc,
                              Manifest.permission.ACCESS_COARSE_LOCATION) ==
                              PackageManager.PERMISSION_GRANTED)
64                            {
65                                locationManager.requestLocationUpdates(
                                                LocationManager.PASSIVE_PROVIDER, 0, 0, this);
```

```
                        }
                    }
                    break;
                default:
                    break;
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void stopListener(){
        locationManager.removeUpdates(this);
        tracking = false;
    }
}
```

Nach der Erfassung werden die Standort-Daten, entsprechend ihres Vorhandenseins und Präferierung, in die *MyLocation* Klasse übernommen. In Listing 15 wird die dabei stattfindende Anpassung der Daten für den Server gezeigt. Aus Gründen des Datenschutzes war wichtig, dass die Werte auf eine Nachkommastelle begrenzt werden. Dies führt zu einer Vergröberung der Position auf ca. 11 km. Dadurch soll die Anonymität des Benutzers gewährleistet bleiben aber dennoch GPS-Daten für die wissenschaftliche Auswertung bereitgestellt werden.

Listing 15: MyLocation

```
public class MyLocation {
    String name = "gps";
    long collected_at;
    double latitude;
    double longitude;
    double altitude;

    public MyLocation(double latitude, double longitude, double altidude) {
        this.latitude = cropDecimalPlaces(latitude);
        this.longitude = cropDecimalPlaces(longitude);
        this.altitude = cropDecimalPlaces(altidude);
        collected_at = System.currentTimeMillis()/1000;
    }

    public double cropDecimalPlaces(double value){
        double returnvalue = 0;
        BigDecimal bd = new BigDecimal(value).setScale(1, RoundingMode.HALF_EVEN);
        returnvalue = bd.doubleValue();
        return returnvalue;
    }
}
```

Die Lautstärkenerfassung wurde mithilfe des *MediaRecorder*[51] [23] umgesetzt. In Listing 16 wird ein großer Teil der Implementierung beschrieben. Für das Sammeln und Hochladen der erfassten Daten wurde die Klasse NoiseRecorder implementiert. Darin ist die Lautstärkemessung zu einem Zeitpunkt in Dezibel gespeichert. Die eigentliche Erfassung wird durch einen AsyncTask realisiert, da der Benutzer zu dieser Zeit den Fragebogen ausfüllen und seine Eingabe durch die Messung nicht behindert werden soll. Bei der Erfassung gibt es 2 verschiedene Arten, die durch den Server vorgegeben werden können. Der erste Fall „amount“ bedeutet, dass, wenn möglich, eine vorgegebene Anzahl an Messungen stattfinden soll. Dafür wird, nach dem Öffnen des Fragebogens, alle 10 Sekunden eine Messung durchgeführt, bis die gewünschte Anzahl erreicht wurde oder der Benutzer den Fragebogen absenden möchte.

[51] MediaRecorder: https://developer.android.com/reference/android/media/MediaRecorder (15.09.2020)

Beim zweiten Fall „each“ wird während des Ausfüllens des Fragebogens regelmäßig eine Messung durchgeführt. Die Zeitabstände dieser Messung können vom Server vorgegeben werden.

Die gesammelten Daten werden in beiden Fällen mit den Antworten des Benutzers an den Server gesendet.

Durch diese Programmierung kann man die Erfassung während der Studie verändern, ohne die App updaten zu müssen. Eine Veränderung des Fragebogens am Server reicht aus, da die Fragebögen regelmäßig aktualisiert werden.

Listing 16: Lautstärkemessung

```
1 public class NoiseRecorder {
2
3     String name = "microphone";
4     long collected_at;
5     public double loudness;
6
7     public NoiseRecorder(long collected_at, double loudness) {
8         this.collected_at = collected_at;
9         this.loudness = loudness;
10    }
11 }
12
13 public class RecorderTask extends AsyncTask<Void, Void, Void> {
14    @Override
15    protected Void doInBackground(Void... params) {
16        switch (cuMicrophone.type) {
17            case "amount":
18                for (int i = 0; i < cuMicrophone.value; i++) {
19                    try {
20                        Thread.sleep(10000);
21                    } catch (InterruptedException e) {
22                        e.printStackTrace();
23                    }
24                    if (running) {
25                        record();
26                        count++;
27                    }
28                }
29                break;
30            case "each":
31                while (running) {
32                    try {
33                        Thread.sleep(1000 * cuMicrophone.value);
34                    } catch (InterruptedException e) {
35                        e.printStackTrace();
36                    }
37                    record();
38                }
39                break;
40            default:
41                break;
42        }
43        return null;
44    }
45 }
46
47  private void onRecord(boolean start) {
48    if (start) {
49        startRecording();
50    } else {
51        stopRecording();
52    }
53 }
54
55 private void startRecording() {
56    mRecorder = new MediaRecorder();
57    mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
58    mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
59    mRecorder.setOutputFile(mFileName);
```

```
60      mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
61
62      try {
63          mRecorder.prepare();
64      } catch (IOException e) {
65      }
66
67      mRecorder.start();
68      mRecorder.getMaxAmplitude();
69  }
70
71  private void stopRecording() {
72      if (mRecorder != null) {
73          mRecorder.stop();
74          mRecorder.release();
75          mRecorder = null;
76      }
77  }
78
79  public void record(){
80      NoiseRecorder nm = new NoiseRecorder(System.currentTimeMillis()/1000, getAmplitude());
81      sounds.add(nm);
82  }
83
84  public double soundDb(double ampl)
85  {
86      return  10 * Math.log10( (ampl*ampl));
87  }
88  public double getAmplitude() {
89      if (mRecorder != null) {
90          double d = mRecorder.getMaxAmplitude();
91          return soundDb(d);
92      }
93      else
94      {
95          return 0;
96      }
97  }
```

5.4 Nutzungsdatenerfassung

Für die Nutzungsdatenerfassung wird der *UsageStatsManager*[52] [23] verwendet. Dieser liefert ein Objekt von *UsageEvents*[53] [23] für einen gewünschten Zeitraum. Dieses UsageEvents-Objekt enthält Daten über Aktivitäten des Gerätes. Dabei enthält ein einzelnes *UsageEvents.Event*[54] [23] unter anderem Informationen über den Packagenamen der App, die dieses Event ausgelöst hat, einen Zeitstempel und den Typ des Events. Die mögliche Erfassung verschiedener Daten hängt von der Version des Betriebssystems ab. In der folgenden Tabelle wird aufgelistet, welche und wie detailliert Daten pro Betriebssystemversion erfasst werden können. Neuere Versionen inkludieren alle Erfassungen der älteren Versionen.

Android Version (API Nummer)	Erfasste Daten
Ab 21	- Tägliche aktive Nutzungsdauer der einzelnen Apps (App im Vordergrund sichtbar) - Tägliche Gesamtnutzungsdauer aller Apps (App im Vordergrund sichtbar)

[52] UsageStatsManager: https://developer.android.com/reference/android/app/usage/UsageStatsManager (19.09.2020)
[53] UsageEvents: https://developer.android.com/reference/android/app/usage/UsageEvents (19.09.2020)
[54] UsageEvents.Event: https://developer.android.com/reference/android/app/usage/UsageEvents.Event (19.09.2020)

Ab 28	- Tägliche aktive Screen-Zeiten - Erfassung von „Sleep-Times“ (Zeiträume in denen das Gerät mindestens eine Stunde inaktiv war)
Ab 29	- Gerät-StartUp und ShutDown werden erfasst und zusätzlich, für eine detailliertere Erfassung der Screen-Zeiten und Sleep-Times, genutzt - Tägliche Nutzungsdauer der einzelnen Apps auch im Hintergrund (in Android ForegroundService genannt), um Apps wie Spotify auch bei den meistgenutzen Apps nicht zu übersehen - Tägliche Gesamtnutzungsdauer aller Apps im Hintergrund (completeForegroundServiceUseTime)

Tabelle 5: UsageStats Android-Versionen

Die komplexe Berechnung der entstehenden Werte wird aus Platz- und Übersichtlichkeitsgründen als Zeichnung in Abbildung 17 und Abbildung 18 verdeutlicht. Der genaue Programmcode kann der Library entnommen werden.

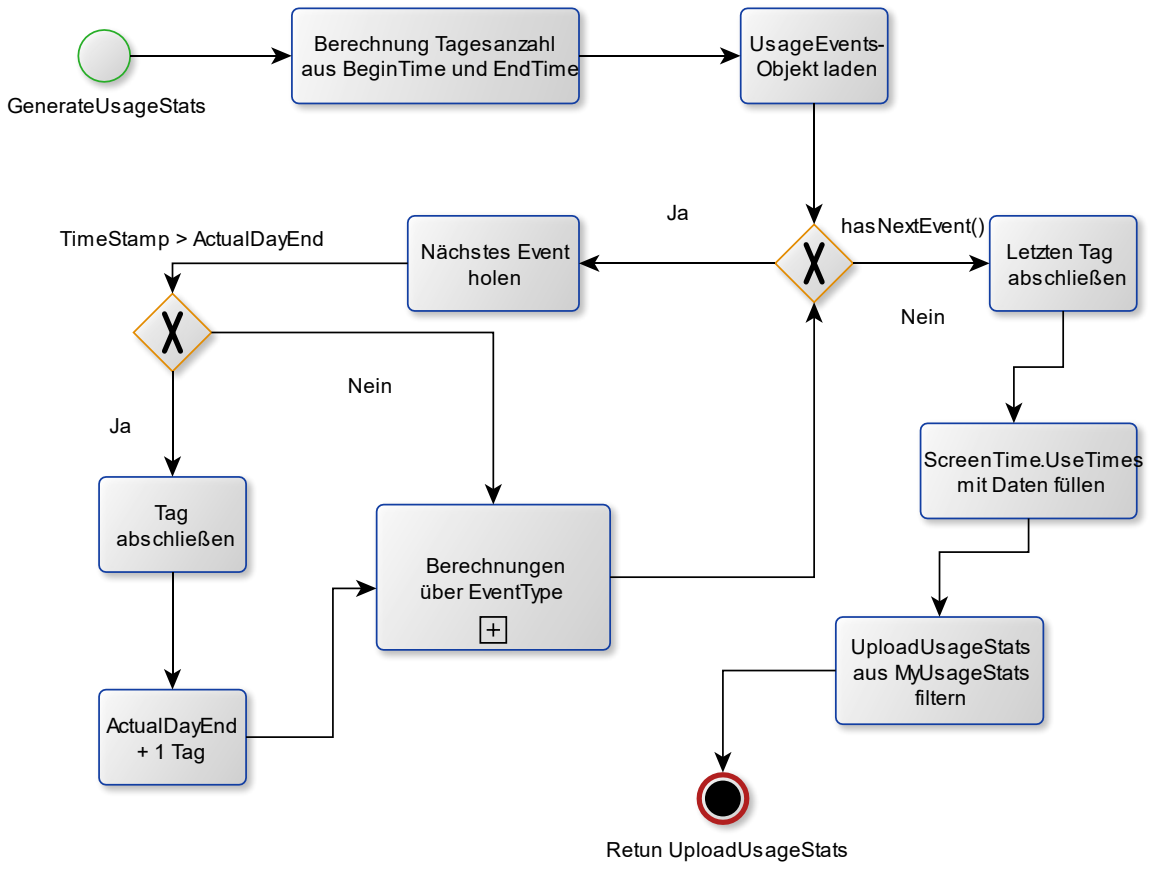

Abbildung 17: Implementierung UsageStats

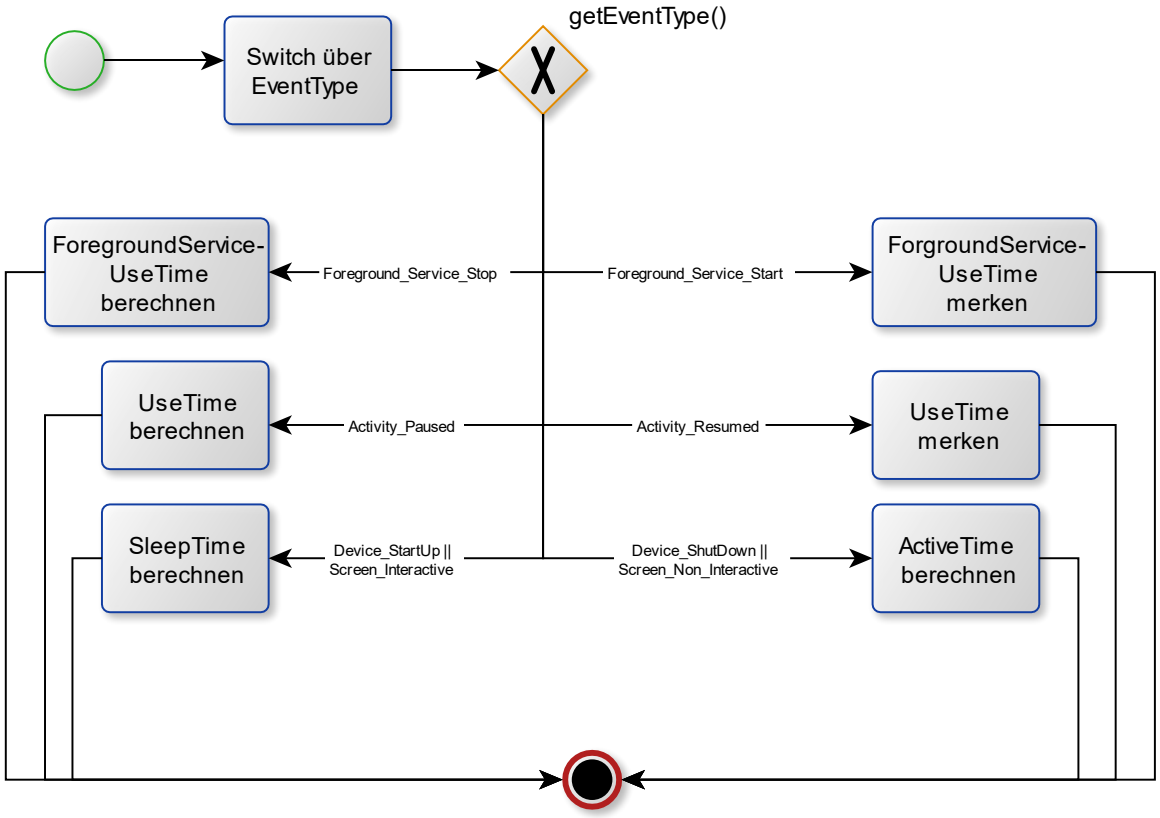

Abbildung 18: Implementierung EventTypes

Da der durchschnittliche Smartphone-User mehr als 80 Apps auf seinem Gerät hat und einen Großteil davon mindestens einmal im Monat nutzt [25], fallen bei dieser Erfassung eine große Menge an Daten an. Durch die Berechnung werden die einzelnen Events schon teilweise zusammengefasst, allerdings bietet die Library zusätzlich die Möglichkeit, gezielt nach bestimmten Apps zu suchen, auf bestimmte Begriffe im Packagename zu matchen oder Apps auszulassen. Dies wird benötigt, um zum Beispiel bei der Vielzahl verschiedener Telefonapps verschiedener Anbieter, keine zu übersehen oder Apps, die nicht bewusst vom Benutzer verwendet werden, wie zum Beispiel Launcher Applikationen, aus den Daten herauszufiltern. Standardmäßig liefert die Library die Gesamtwerte für aktive, sowie nicht aktive Zeiten des Gerätes und die 5 meistgenutzten Apps, die nicht in der gewünschten App-Liste vorkommen. In Listing 17 werden die Funktionen für die Filterung und Anpassung der Daten für den Server gezeigt.

Eine wichtige Funktion zur Anpassung der Daten besteht darin, Zeitstempel, die von Android in Millisekunden seit 1970 geliefert werden, auf Sekunden seit 1970 zu formatieren, um die Form für Zeitstempel auf dem Server einheitlich zu gestalten (Zeile 3-32).

Die nächste Funktion der Klasse beschreibt die Filterung der Daten. Dafür werden zuerst, aus den täglichen Einzeldaten der Apps, die Gesamtnutzungszeiten berechnet, die für den späteren Vergleich benötigt werden (Zeile 35-39). Danach wird, in der Gesamtliste der Apps, nach den Apps aus der gewünschten Liste gesucht. Falls sie in den Daten vorhanden sind, werden sie in die Apps-Liste für den Server übertragen (Zeile 40-47). Es folgen die Suche nach den gewünschten Wildcards (zum Beispiel „phone“) und die Entfernung der unerwünschten Wildcards (zum Beispiel Launcher) in Zeile 48 und 49.

Abschließend werden die übrigen Apps sortiert (näheres in Listing 18), sowie die 5 Meistgenutzten der top5Apps-Liste hinzugefügt (Zeile 51-59).

Eine letzte Funktion der Klasse ist es, einen String mit allen Packagenamen, der erfassten Apps zu erstellen, da es für den Benutzer klar ersichtlich sein soll, welche Aktivitäten und Daten an den Server gesendet wurden.

Listing 17: UploadUsageStats

```
1  public class UploadUsageStats  {
2
3      public void changeLongsforApi(){
4          beginTime = beginTime/1000;
5          endTime = endTime/1000;
6          for(SleepTime sleepTime : sleepTimes){
7              sleepTime.beginTime = milliSecondToSecond(sleepTime.beginTime);
8              sleepTime.endTime = milliSecondToSecond(sleepTime.endTime);
9          }
10         changeLongsofApps(apps);
11         changeLongsofApps(top5Apps);
12     }
13
14     public void changeLongsofApps(List<AppWatch>appsToChange){
15         for(AppWatch app : appsToChange){
16             for(AppWatch.DailyValue dv : app.dailyValues){
17                 dv.firstUseTime = milliSecondToSecond(dv.firstUseTime);
18                 dv.lastUseTime = milliSecondToSecond(dv.lastUseTime);
19                 dv.firstFgServiceUseTime = milliSecondToSecond(dv.firstFgServiceUseTime);
20                 dv.lastFgServiceUseTime = milliSecondToSecond(dv.lastFgServiceUseTime);
21             }
22         }
23     }
24
25     public long milliSecondToSecond(long timeStamp){
26         if(timeStamp <= 0){
27             return timeStamp;
28         }
29         else{
30             return timeStamp/1000;
31         }
32     }
33
34     public void fillUsageStats(MyUsageStats myUsageStats, String[] packageNames,
                                  String[] wildCards, String[] removeWildCards){
35         for (Map.Entry<String, AppWatch> entry : myUsageStats.apps.entrySet()) {
36             AppWatch app = entry.getValue();
37             app.getCompleteFgServiceUseTime();
38             app.getCompleteUseTime();
39         }
40         for(String s : packageNames){
41             AppWatch app = null;
42             app = myUsageStats.apps.get(s);
43             if(app!= null){
44                 apps.add(app);
45                 myUsageStats.apps.remove(s);
46             }
47         }
48         getWildCard(myUsageStats.apps,wildCards);
49         removeWildCard(myUsageStats.apps,removeWildCards);
50
51         List<AppWatch> sortApps = new ArrayList<>(myUsageStats.apps.values());
52         Collections.sort(sortApps);
53         int topAppsCount = 5;
54         if(sortApps.size() < topAppsCount){
55             topAppsCount = sortApps.size();
56         }
57         for(int i = 0 ; i < topAppsCount; i++){
58             top5Apps.add(sortApps.get(i));
59         }
60     }
61
62     public void getWildCard(Map<String,AppWatch> appsList, String[] wildCardPackageNames){
```

```
63          List<AppWatch> wildCards = new ArrayList<>();
64          for(String wildCard : wildCardPackageNames) {
65              for(Map.Entry<String, AppWatch> entry : appsList.entrySet()){
66                  if(entry.getKey().contains(wildCard)){
67                      wildCards.add(entry.getValue());
68                  }
69              }
70          }
71          for(AppWatch app : wildCards){
72              appsList.remove(app.packageName);
73              apps.add(app);
74          }
75      }
76      public void removeWildCard(Map<String,AppWatch> appsList,
                                   String[] wildCardPackageNames){
77          List<AppWatch> wildCards = new ArrayList<>();
78          for(String wildCard : wildCardPackageNames){
79              for(Map.Entry<String, AppWatch> entry : appsList.entrySet()){
80                  if(entry.getKey().contains(wildCard)){
81                      wildCards.add(entry.getValue());
82                  }
83              }
84          }
85          for(AppWatch app : wildCards){
86              appsList.remove(app.packageName);
87          }
88      }
89
90      public String getPackageNamesString(){
91          String returnString = "" ;
92          for (AppWatch app: apps)
93          {
94              returnString += app.packageName +"  \r\n";
95          }
96          for (AppWatch app: top5Apps)
97          {
98              returnString += app.packageName +"  \r\n";
99          }
100
101         if(returnString.isEmpty()){
102             return null;
103         }
104         return  returnString;
105     }
106 }
```

Die wichtigste Datenhaltungsklasse der Library ist die Klasse AppWatch. Sie enthält alle Informationen einer einzelnen App. Die Bedeutung der Variablen kann der Tabelle 4 entnommen werden.

Die Klasse enthält eine Liste an DailyValues, welche die Aufteilung der Gesamtdaten in täglichen Werten repräsentieren. Die Anzahl der DailyValues wird beim Erstellen eines AppWatch-Objektes übergeben und es werden direkt entsprechend viele DailyValues mit ihren Standardwerten generiert (Zeile 7-13). Zeitstempel sind dabei Default -1 und Zeitwerte 0.

Der UsageStatsHelper kann über die Funktionen der Klasse AppWatch und Übergabe des entsprechenden Index des gewünschten Tages, die täglichen Werte verändern, bzw. die Berechnung anstoßen.

Des Weiteren implementiert die Klasse das Interface *Comparable*[55], um eine automatische Sortierung anstoßen zu können. Dabei wird die komplette Nutzungszeit und die komplette Foreground-Service Nutzungszeit verglichen. Bei älteren Android-Versionen geht Letztere mit 0 in die Berechnung und den Vergleich ein, da sie nicht erfasst werden kann.

[55] Comparable: https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html (23.09.2020)

Listing 18: AppWatch und DailyValue

```
1  public class AppWatch implements Comparable<AppWatch>{
2      public String packageName;
3      public List<DailyValue> dailyValues;
4      long completeUseTime;
5      long completeFGServiceUseTime;
6
7      public AppWatch (String packageName, int dayCount){
8          this.packageName = packageName;
9          dailyValues = new ArrayList<>();
10         for(int i = 0; i<dayCount;i++){
11             dailyValues.add(new DailyValue());
12         }
13     }
14
15     public void toBackGround(long timeStamp, long intervalBegin,int dayidx) {
16         dailyValues.get(dayidx).toBackGround(timeStamp,intervalBegin);
17     }
18
19     public void toForeGround(long timeStamp,int dayidx) {
20         dailyValues.get(dayidx).toForeGround(timeStamp);
21     }
22
23     public void fgServiceStopp(long timeStamp, long intervalBegin,int dayidx) {
24         dailyValues.get(dayidx).fgServiceStopp(timeStamp,intervalBegin);
25     }
26
27     public void fgServiceStart(long timeStamp,int dayidx) {
28         dailyValues.get(dayidx).fgServiceStart(timeStamp);
29     }
30
31     public void completeUseTime(long intervalEnd, int dayidx){
32         dailyValues.get(dayidx).completeUseTime(intervalEnd);
33     }
34
35     public void getCompleteUseTime(){
36         long ut=0;
37         for(DailyValue dv : dailyValues){
38             ut += dv.useTime;
39         }
40         completeUseTime = ut;
41     }
42     public void getCompleteFgServiceUseTime(){
43         long ut=0;
44         for(DailyValue dv : dailyValues){
45             ut += dv.fgServiceUseTime;
46         }
47         completeFGServiceUseTime = ut;
48     }
49
50     @Override
51     public int compareTo(AppWatch o) {
52         return (int)( (o.completeUseTime + o.completeFGServiceUseTime) -
                       (completeUseTime + completeFGServiceUseTime) );
53     }
54
55     public class DailyValue {
56         public long useTime;
57         public long firstUseTime;
58         public long lastUseTime;
59         private transient long lastToForground;
60
61         public long fgServiceUseTime;
62         private transient long lastFgServiceStart;
63         long firstFgServiceUseTime;
64         long lastFgServiceUseTime;
65
66         public DailyValue(){
67             firstUseTime = -1;
68             useTime = 0;
69             lastUseTime = -1;
70             lastToForground = -1;
71             firstFgServiceUseTime = -1;
72             fgServiceUseTime = 0;
73             lastFgServiceUseTime = -1;
74             lastFgServiceStart = -1;
```

```
75          }
76          public void toBackGround(long timeStamp, long intervalBegin) {
77              if(lastToForground > 0){
78                  useTime += timeStamp-lastToForground;
79              }
80              else{
81                  firstUseTime = intervalBegin;
82                  if(lastUseTime > intervalBegin){
83                      useTime += timeStamp-lastUseTime;
84                  }
85                  else{
86                      useTime += timeStamp-intervalBegin;
87                  }
88              }
89              if(lastUseTime <0 || timeStamp > lastUseTime){
90                  lastUseTime = timeStamp;
91              }
92          }
93
94          public void toForeGround(long timeStamp) {
95              if(firstUseTime <0 || timeStamp < firstUseTime){
96                  firstUseTime = timeStamp;
97              }
98              lastToForground = timeStamp;
99          }
100
101         public void fgServiceStopp(long timeStamp, long intervalBegin) {
102             if(lastFgServiceStart > 0){
103                 fgServiceUseTime += timeStamp-lastFgServiceStart;
104             }
105             else{
106                 firstFgServiceUseTime = intervalBegin;
107                 if(lastFgServiceUseTime > intervalBegin){
108                     fgServiceUseTime += timeStamp-lastFgServiceUseTime;
109                 }
110                 else{
111                     fgServiceUseTime += timeStamp-intervalBegin;
112                 }
113             }
114             if(lastFgServiceUseTime <0 || timeStamp > lastFgServiceUseTime){
115                 lastFgServiceUseTime = timeStamp;
116             }
117         }
118
119         public void fgServiceStart(long timeStamp) {
120             if(firstFgServiceUseTime <0 || timeStamp < firstFgServiceUseTime){
121                 firstFgServiceUseTime = timeStamp;
122             }
123             lastFgServiceStart = timeStamp;
124         }
125
126         public void completeUseTime(long intervalEnd){
127             if(lastToForground > lastUseTime){
128                 useTime += intervalEnd-lastToForground;
129                 lastUseTime = intervalEnd;
130             }
131             if(lastFgServiceStart > lastFgServiceUseTime){
132                 fgServiceUseTime += intervalEnd-lastFgServiceStart;
133                 lastFgServiceUseTime = intervalEnd;
134             }
135         }
136     }
137 }
```

6 Vorstellung der Module in der App

Dieses Kapitel stellt die Bedienung und Interaktion der Module aus Sicht eines Benutzers vor. Dabei werden die einzelnen Funktionen und Ansichten gezeigt, die dem Benutzer beim Ausführen der App bereitstehen und dargestellt werden. Module, die ausschließlich im Hintergrund arbeiten, wie die Library für die Nutzungsdatenerfassung, haben keine eigene Ansicht und werden hier nur durch ihre Zustimmungsdialoge repräsentiert. Die Informationsgewinnung durch die Kapitel, das Ausfüllen von dazugehörigen Quiz und Fragebögen und die Ansicht der Ergebnisse sind die Hauptfunktionen dieser Module. Die App unterstützt jegliche Android Geräte ab *Android-Version 5.0*, wurde allerdings für die Darstellung auf Smartphones optimiert.

Bei den Inhalten der Screenshots wurde, in manchen Bereichen, auf Daten von anderen Servern zugegriffen, da zum jetzigen Zeitpunkt noch nicht alle aktuellen Daten des UNITI-Projektes auf dem Server verfügbar sind. Dadurch können die Ansichten thematisch abweichen, stellen jedoch das spätere Design dar.

6.1 Vorstellung des Edukationsmoduls

Das Edukationsmodul gibt dem Benutzer die Möglichkeit, eine klare Übersicht über die möglichen Informationen zu erhalten. Weiter werden die Informationen eines einzelnen Unterkapitels so vielfältig wie möglich dargestellt, um Abwechslung für den Benutzer zu schaffen. Dabei steht die Motivation des Benutzers im Vordergrund. Diese soll durch mehrere Maßnahmen gefördert werden. Die genaue Vorgehensweise wird im Folgenden erläutert.

6.1.1 Kapitelstruktur

Für die Ansicht der Kapitelstruktur (Abbildung 19) muss der Benutzer in der Edukation-Studie eingeschrieben sein. Danach kann er über das entsprechende Tab im Bottom-Menü zur Kapitelübersicht gelangen.

Dort werden ihm die verschiedenen Kapitel in einer Liste angezeigt. Die Kapitel sind aufklappbar und enthalten die entsprechenden Lektionen. Für jedes Kapitel werden dem Benutzer die aktuellen Fortschritte und der Status visualisiert. Der Benutzer bekommt Informationen darüber, wie viele Unterkapitel des Kapitels er abgeschlossen hat und eine Prozentangabe, wie gut er in den, im Kapitel enthaltenen Quiz, abgeschnitten hat.

Für den Status der Kapitel, bzw. der Lektionen, gibt es 3 Möglichkeiten:

1. Completed: Diese Kapitel, bzw. Lektionen, wurden vom Benutzer bereits abgeschlossen. Eintrag ist weiß hinterlegt und schwarz unterstrichen.
2. Highlighted: Dieses Kapitel, bzw. Lektion gilt es für den Benutzer als nächstes zu bearbeiten. Eintrag ist weiß hinterlegt und grün unterstrichen.
3. Blocked: Diese Kapitel, bzw. Lektionen, können vom Benutzer noch nicht bearbeitet werden, da es unbearbeitete Kapitel oder Lektionen davor gibt. Eintrag ist grau hinterlegt und schwarz unterstrichen.

Zusätzlich ist oben eine Anzeige für das aktuelle Level des Benutzers. Er kann verschiedene Level erreichen, wie zum Beispiel Beginner oder Fortgeschrittener. Dabei wird ihm auch, zur Förderung der Motivation, dargestellt, wie viele Kapitel er noch bearbeiten muss, um das nächste Level zu erreichen.

Abbildung 19: Kapitelstruktur

6.1.2 Unterkapitel

Nach der Auswahl einer Lektion gelangt der Benutzer, wenn es sich um ein Unterkapitel handelt, zur Unterkapitel-Ansicht (Abbildung 20). Im Falle eines Quiz wird der Benutzer zur Fragebogenstruktur-Ansicht, die in Abbildung 24 gezeigt wird, weitergeleitet.

Das Unterkapitel wird mithilfe einer WebView dargestellt und kann somit unterschiedliche Elemente enthalten. Der Benutzer soll, durch die Abwechslung von Text, Bild oder Video, motiviert bleiben. Im Falle eines Videos besteht für den Benutzer auch die Möglichkeit, sich dieses Video im Vollbildmodus anzusehen. Dies spielt gerade auf Smartphones eine große Rolle, da die Bildschirme nicht sehr groß sind und so möglicherweise Details untergehen könnten.

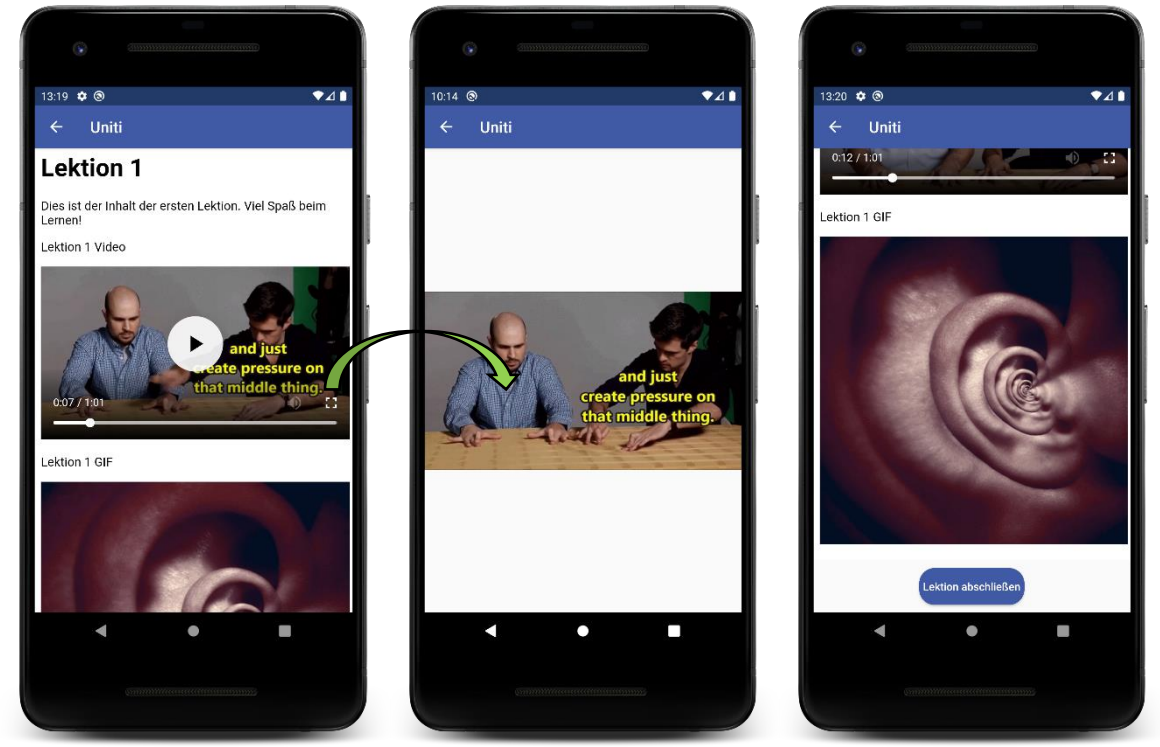

Abbildung 20: Unterkapitel

6.2 Vorstellung des Fragebogenmoduls

Das Fragebogenmodul gibt dem Benutzer die Möglichkeit, einen Fragebogen auszuwählen, zu beantworten und an den Server zu senden. Dabei leistet das Modul dem Benutzer, mit bestimmten Anzeigen und Benachrichtigungen, Hilfestellung, wann welcher Fragebogen auszufüllen ist. Durch die Rückmeldung vom Server wird dem Benutzer, zu seinen Antworten passendes Feedback geliefert und übersichtlich angezeigt.

6.2.1 Fragebögen

Für die Ansicht der Fragebögen muss der Benutzer in der TrackYourTinnitus-Studie eingeschrieben sein. Danach kann er über das entsprechende Tab im Bottom-Menü zur Fragebögen-Ansicht (Abbildung 21) gelangen.

In der Fragebögen-Ansicht werden dem Benutzer seine aktuellen und aktiven Fragebögen zur Auswahl angezeigt. Dabei werden One-time Fragebögen, die noch nicht ausgefüllt wurden, vorrangig behandelt und nur diese dargestellt. Sobald es keine unausgefüllten One-time Fragebögen mehr für den Benutzer gibt, werden die sich wiederholenden Fragebögen zur Auswahl visualisiert. Es wird bei jedem Öffnen dieser Ansicht versucht, die Fragebögen des Benutzers, durch Verbindung zum Server zu aktualisieren. Wenn sich die Fragebögen auf dem Server nicht geändert haben oder die Verbindung fehlschlägt, wird der Stand aus der lokalen Datenbank geladen und dem Benutzer eine entsprechende Fehlermeldung dargestellt. Durch einen Klick, auf einen der angezeigten Fragebögen, gelangt man zu der Fragebogenstruktur-Ansicht (Abbildung 24), in der der ausgewählte Fragebogen angezeigt wird.

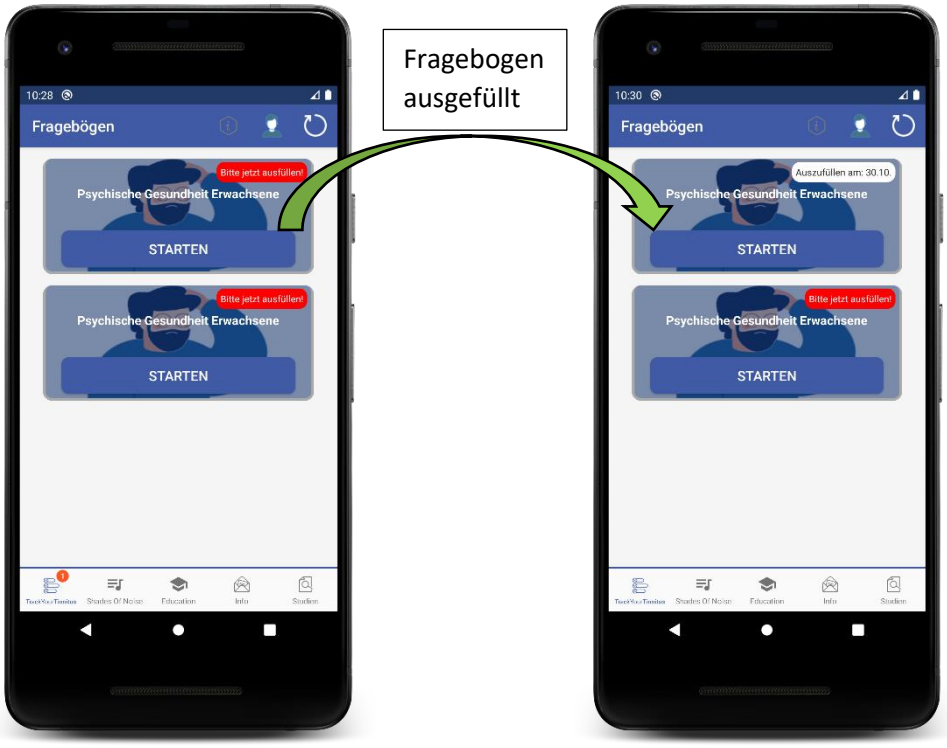

Abbildung 21: Fragebögen-Ansicht

Das Modul stellt die Möglichkeit bereit, verschiedene Daten wie GPS, Umgebungslautstärke oder Bilder zu erfassen. Dafür benötigt man die entsprechende Berechtigung. Um den Benutzer dazu aufzufordern, der App die entsprechende Berechtigung zu erteilen, wird der Standarddialog des Betriebssystems für Berechtigungsabfragen genutzt. Dieser kann sich je nach Betriebssystemversion unterscheiden und wird im Folgenden (Abbildung 22) beispielhaft für Android 10 und Android 7 dargestellt. Der Benutzer hat, je nach Berechtigung, verschiedene Auswahlmöglichkeiten. So wird bei den Berechtigungen für Mikrofon und Kamera gefragt, ob er es zulassen oder verweigern möchte. Beim Standort gibt es bei neueren Android-Versionen die zusätzliche Option, die Berechtigung nur während der App-Nutzung zu gewähren.

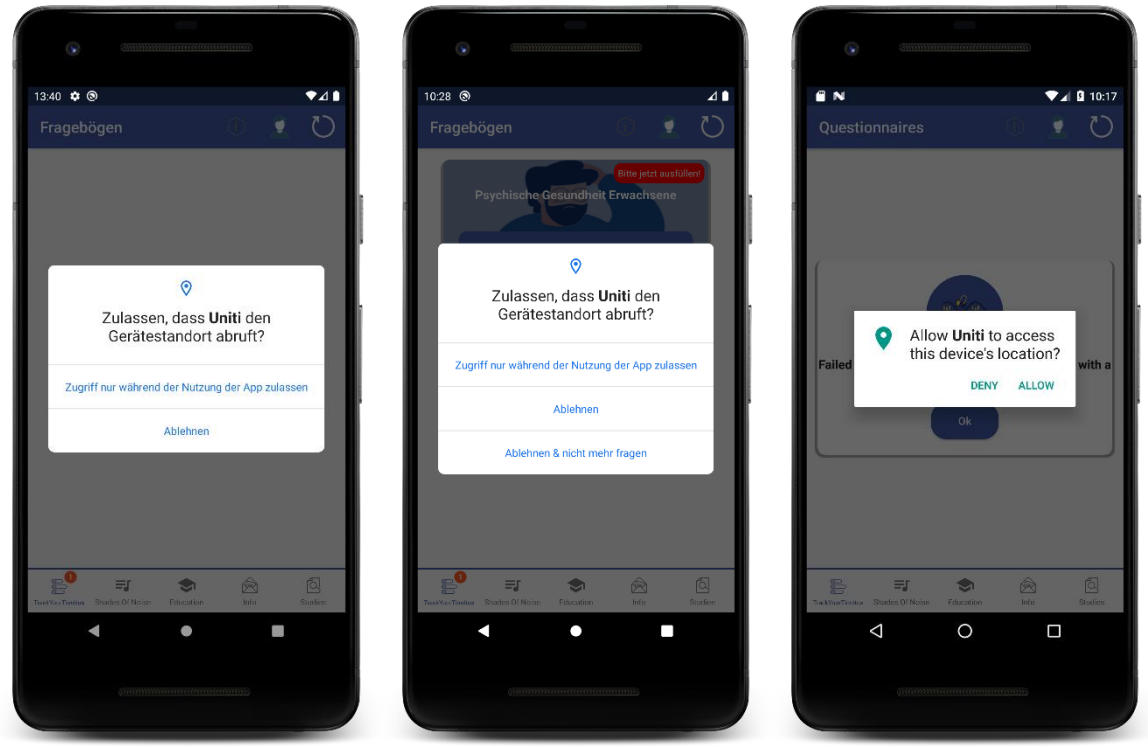

Abbildung 22: Fragebögen-Ansicht Standard-Berechtigungsanfrage (GPS, Mikrofon, usw.)

Die Erfassung der Nutzungsdaten des Gerätes ist eine kritische Berechtigung und kann somit nicht über den Standarddialog des Betriebssystems abgefragt werden. Dafür wird dem Benutzer ein Dialog angezeigt, in dem die gewünschte Berechtigung und ihre Nutzung beschrieben wird. Des Weiteren wird darin erklärt, wie der Benutzer der App diese Berechtigung erteilen kann.

Es werden ihm die folgenden 3 Möglichkeiten als Reaktion auf den Dialog gegeben:

1. Überspringen: Der Dialog wird geschlossen und beim nächsten Aufruf der Fragebögen-Ansicht erneut angezeigt, wenn die Berechtigung noch nicht erteilt wurde.
2. Nein, nicht erneut fragen: Der Dialog wird geschlossen und der Benutzer wird nicht erneut nach der Berechtigung gefragt.
3. Zu den Einstellungen: Der Benutzer wird direkt zu den Einstellungen der Nutzungsdatenberechtigung (Abbildung 23) weitergeleitet. Dort muss der Benutzer, wie im Text des Dialogs beschrieben die UNITI-App auswählen und ihr die Berechtigung erteilen.

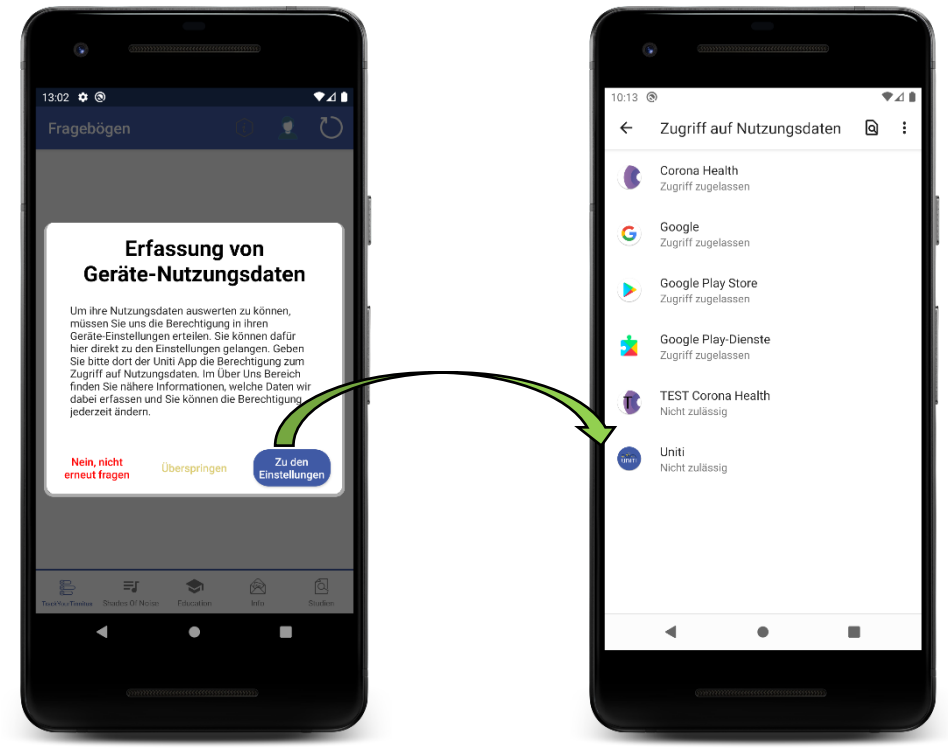

Abbildung 23: Fragebögen-Ansicht Berechtigungsanfrage Gerätenutzungsdaten

6.2.2 Fragebogenstruktur

Die Fragebogenstruktur-Ansicht (Abbildung 24) zeigt dem Benutzer den zuvor ausgewählten Fragebogen mit seinen Frageelementen an. Dies stellt die eigentliche Hauptaufgabe des Moduls dar. Mit dieser Ansicht kann der Benutzer den ausgewählten Fragebogen ausfüllen und an den Server senden, bzw. bei fehlender Internetverbindung zwischenspeichern, damit dieser später hochgeladen werden kann. Danach wird der Benutzer auf die Ansicht der Antwortsätze (Abbildung 26) geleitet, um direkt ein mögliches Feedback sehen zu können.

Für eine bessere Übersicht des Benutzers werden die Fragen auf mehrere, virtuelle Seiten aufgeteilt. Dabei kann der Benutzer, über die Buttons links und rechts unten, die Seiten durchblättern und die Fragen beantworten. Dabei gilt allerdings zu beachten, dass der Benutzer alle obligatorischen Fragen auf einer Seite ausgefüllt haben muss, bevor er die Seite wechseln kann. Eine entsprechende Fehlermeldung und Markierung der nicht beantworteten Fragen sind in Abbildung 25 zu sehen.

Die Navigation findet ausschließlich über die Buttons am unteren Rand statt. Der „Zurück" Button links unten oder der integrierte „Zurück" Button des Betriebssystems blättern zur vorherigen Seite, bzw. beenden das Ausfüllen, wenn sich der Benutzer auf der ersten Seite befindet. Der Button rechts unten blättert die Seiten, mit der Aufschrift „Weiter", nach vorne und sendet den Antwortsatz auf der letzten Seite, mit der Aufschrift „Senden", an den Server.

Um dem Benutzer ein Gefühl zu geben, wie lang er noch für den Fragebogen benötigt, wird ihm oben sein Fortschritt angezeigt.

Die unterschiedlichen Fragetypen, die die App anzeigen kann:

1. Headline: Überschrift des Fragebogens.
2. Text: Text, der zwischen zwei Frageelementen stehen soll.
3. Frage mit Textfeldeingabe: Dabei gibt es verschiedene Typen von Textfeldern, wie zum Beispiel eine Datumseingabe, ein Multiline Textfeld oder ein ganz normales Textfeld.
4. Frage mit SingleChoice: Auswahl einer Antwortmöglichkeit aus mehreren möglichen Antworten, durch Auswählen eines RadioButtons[56] [23]. Auch Ja/Nein Fragen werden durch eine solche SingleChoice Frage angezeigt.
5. Frage mit MultipleChoice: Auswahl mehrerer Antwortmöglichkeiten aus mehreren möglichen Antworten, durch Auswählen der entsprechenden Checkboxen.
6. Frage mit Slider Eingabe: Antworteingabe durch ein beschriftetes Sliderelement.
7. Frage mit *NumberPicker*[57] [23] Eingabe: Antworteingabe durch scrollbares NumberPickerelement.
8. Frage mit SamScaleFaces: Antworteingabe durch Auswählen eines RadioButtons, die unter einer Ansicht von Zeichnungen (SamScaleFaces [26]), die die entsprechende Antwort visualisieren, angeordnet sind.
9. Frage mit Kamera: Antworteingabe durch Bildaufnahme.
10. Frage mit Sound: Antworteingabe durch Bedienung des Players.

[56] RadioButton: https://developer.android.com/guide/topics/ui/controls/radiobutton (23.09.2020)
[57] NumberPicker: https://developer.android.com/reference/android/widget/NumberPicker (23.09.2020)

Abbildung 24: Fragebogenstruktur

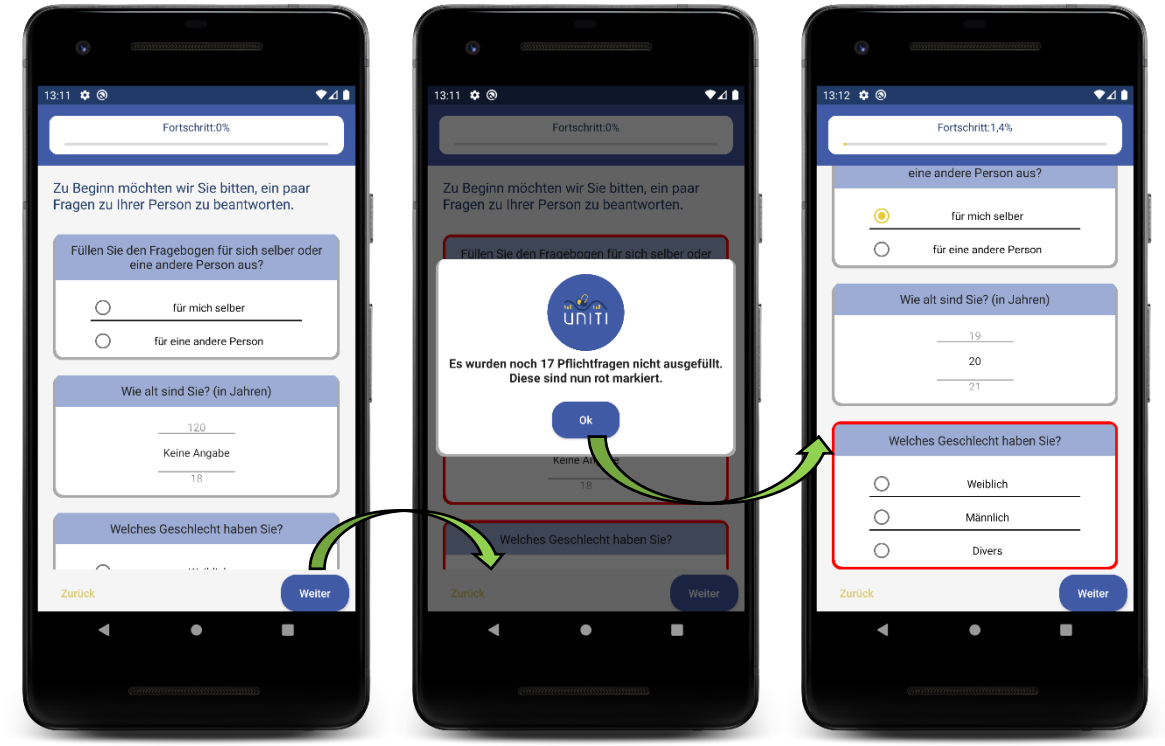

Abbildung 25: Fragebogenstrukur Fehlermeldung

6.2.3 Antwortsätze

Nach dem Beantworten von Fragebögen gibt es für den Benutzer die Möglichkeit, sich in der Antwortsätze-Ansicht (Abbildung 26) anzeigen zu lassen, welche Antwortsätze er bis jetzt an den Server gesendet hat und vom Server evaluiert wurden. Dabei wird bei jedem Öffnen der Ansicht versucht, vom Server neue Antwortsätze und deren Evaluierungen zu laden. Bei Erfolg wird eine Liste der Antwortsätze zur Auswahl dargestellt. Dabei wird dem Benutzer auch der Zeitpunkt, an dem der Antwortsatz hochgeladen wurde, sowie der Name des Fragebogens visualisiert, damit er erkennen kann welcher Antwortsatz der Aktuelle ist. Bei Misserfolg, durch fehlende Internetverbindung, erscheint eine Meldung, dass die Ergebnisse noch nicht mit den lokal gespeicherten Abgaben übereinstimmen. Die bisher in der Datenbank gespeicherten und evaluierten Antwortsätze werden geladen und angezeigt. Durch die Auswahl eines Antwortsatzes gelangt man zur Feedback-Ansicht (Abbildung 27).

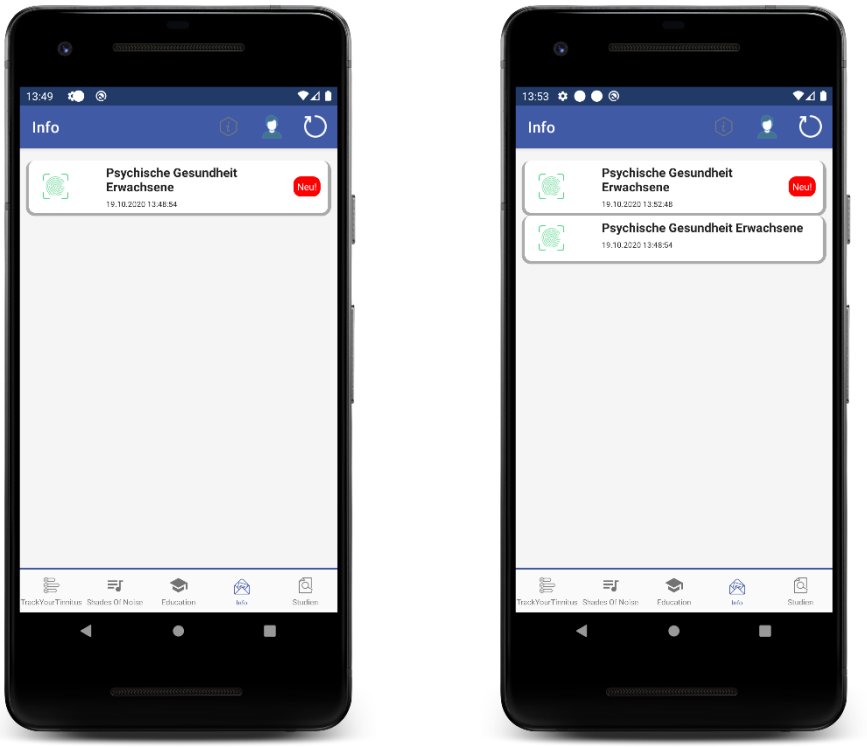

Abbildung 26: Antwortsätze

6.2.4 Feedback

Um dem Benutzer die Ergebnisse seiner Antworten anzuzeigen, gibt es die Feedback-Ansicht (Abbildung 27). Darin wird dem Benutzer das Feedback, zu seinem in der Antwortsätze-Ansicht (Abbildung 26) ausgewählten Antwortsatz, dargestellt. Die verschiedenen Feedbackblöcke werden in einer Liste mit Titel und dazugehörigem Text visualisiert. Die Blöcke sind farblich markiert, um dem Benutzer eine gute Übersicht über positives und negatives Feedback zu geben. Zuerst werden dem Benutzer nur die Titel der Feedbacks angezeigt, um eine bessere Übersichtlichkeit zu gewährleisten. Durch einen Klick auf einen Titel wird der dazugehörige Text sichtbar. Dieser Text verschwindet, durch einen weiteren Klick auf den Titel wieder. Durch die integrierte „Zurück“- Taste oder den Button links oben gelangt der Benutzer wieder zurück zur Antwortsätze-Ansicht.

Abbildung 27: Feedback

7 Anforderungsabgleich

In diesem Kapitel werden die Anforderungen, die in Kapitel 3 definiert wurden, mit der Implementierung und den Funktionen der App abgeglichen.

7.1 Funktionale Anforderungen

Hier werden die funktionalen Anforderungen an die App abgeglichen. Dabei wird überprüft, ob in der aktuellen Implementierung alle funktionalen Anforderungen erfüllt sind.

Nr.	Anforderung	Analyse
Edukationsmodul		
1.	Ordnung der Informationen	Anforderung erfüllt. Siehe Model-Klassen
2.	Fortschrittsanzeige für Kapitel und Lektionen	Anforderung erfüllt. Siehe ListAdapter, Receiver, Services und Edukationsmodul
3.	Anzeige der Informationen	Anforderung erfüllt. Siehe Edukationsmodul
4.	Benachrichtigung als Erinnerung	Anforderung erfüllt. Siehe Benachrichtigungen
5.	Erfassung des Lernfortschrittes	Anforderung erfüllt. Siehe Fragebogenmodul, Edukationsmodul, Kapitelstruktur und Feedback
Fragebögen und Quiz		
6.	Quiz ausfüllen	Anforderung erfüllt. Siehe Fragebogenstruktur und Kapitelstruktur
7.	Statistische Fragebögen ausfüllen	Anforderung erfüllt. Siehe Vorstellung des Fragebogenmoduls
8.	Zustand eines Fragebogens ändern	Anforderung erfüllt. Siehe Vorstellung des Fragebogenmoduls
9.	Ergebnisse synchronisieren	Anforderung erfüllt. Siehe Antwortsätze und Feedback
10.	Lautstärkemessung	Anforderung erfüllt. Siehe Fragebogenmodul
11.	GPS-Daten erfassen	Anforderung erfüllt. Siehe Fragebogenmodul
12.	Erfolgsangabe in Prozent	Anforderung erfüllt. Siehe Kapitelstruktur und Edukationsmodul
13.	An Fragebögen erinnern	Anforderung erfüllt. Siehe Benachrichtigungen
Feedback		
14.	Abfragen der Ergebnisse	Anforderung erfüllt. Siehe Antwortsätze und Feedback
15.	Ergebnistexte zusammenfassen	Anforderung erfüllt. Siehe Antwortsätze und Feedback
16.	Ergebnisse anzeigen	Anforderung erfüllt. Siehe Antwortsätze und Feedback

Nutzungsdaten		
17.	Erfassung von Nutzungsdaten	Anforderung erfüllt. Siehe Model-Klassen und Nutzungsdatenerfassung
Allgemein		
18.	Nutzung ohne Internetverbindung	Anforderung erfüllt. Alle zur Bedienung benötigten Daten werden, wenn sie einmal vom Server geladen wurden, in der lokalen Datenbank (siehe Lokale Datenbank) auf dem Gerät gespeichert. Damit ist, bis auf das vom Server generierte Feedback für einen neu abgegebenen Fragebogen, eine volle Funktionalität auch ohne funktionierende Internetverbindung garantiert.

Tabelle 6: Abgleich funktionale Anforderungen

7.2 Nicht-funktionale Anforderungen

Hier werden die nicht-funktionalen Anforderungen an die App abgeglichen. Dabei wird überprüft, ob in der aktuellen Implementierung alle nicht-funktionalen Anforderungen erfüllt sind.

Nr.	Beschreibung	Problembeschreibung
1.	Design & Bedienung	Anforderung erfüllt. Die Bedienung beschränkt sich auf die bekannten Eingabeelemente, wie Buttons, Checkboxen, und Slider. Des Weiteren sind bei der Kapitelstruktur die Kapitel, Unterkapitel und Quiz farblich klar gekennzeichnet, ob sie verfügbar sind oder nicht.
2.	Modularität	Anforderung erfüllt. Die Module sind nach dem Model-View-Controller Prinzip aufgebaut und wurden in die Gesamtstruktur der UNITI-App integriert. Die Erfassung der Nutzungsdaten wurde als Library getrennt und kann so in jeder beliebigen anderen App wiederverwendet werden.
3.	Erweiterbarkeit	Anforderung erfüllt. Die Module sind mit wenig Aufwand zu erweitern. Durch die Vererbungsstruktur muss für einen neuen Fragetypen nur eine entsprechende Question-Klasse und eine Unterklasse von Input-Base implementiert werden, die die gewünschten Eingabeelemente enthält. Die Verwaltungsstruktur innerhalb des Fragebogens, sowie Laden und Absenden des Fragebogens bleibt davon unberührt. Für das Hinzufügen eines Kapitels, samt Lektionen, muss ein weiterer Eintrag in der Liste erfolgen und die zugehörigen HTML-Dateien zum Projekt hinzugefügt werden. Die Fortschrittserfassung und Anzeige passen sich automatisch an.

4.	Mehrsprachigkeit	Anforderung erfüllt. Durch die ausschließliche Verwendung von lokalen Texten als ID-Verweis auf die String-Ressourcen des Projektes ist es möglich, diese Ressource-Datei für alle gewünschten Sprachen zu übersetzen. Die App zeigt alle lokalen Texte in der auf dem Gerät eingestellten Sprache an, wenn sie vorhanden ist. Die Texte vom Server werden ebenfalls, wenn möglich, in der auf dem Gerät eingestellten Sprache geladen.
5.	Verfügbarkeit	Anforderung erfüllt. Da die Erfassung der Nutzungsdaten des Gerätes erst ab Android 5 möglich ist, wurde die Minimalanforderung auf diese Version gesetzt. Dadurch werden nicht alle Betriebssystemversionen unterstützt, aber immer noch ca. 95% der weltweiten Benutzer.
6.	Store-ready	Anforderung nahezu erfüllt. Die Module und die Gesamt-App sind zu diesem Zeitpunkt bezüglich ihrer Funktionalitäten und ihrem Aussehen fertig für den Store. Allerdings sind noch nicht alle Inhalte des Edukationsmoduls (HTML-Dateien und Quiz) bereitgestellt worden. Dies soll bis Ende des Jahres 2020 erfolgen. Die Store-Freigabe ist Anfang 2021 geplant.
7.	Zertifizierung für das Medizinproduktegesetz [5]	Anforderung noch nicht erfüllt. Die Module sind auf ihre Funktionalität getestet worden, zur abschließenden Zertifizierung müssen allerdings alle Inhalte der App bereitstehen.

Tabelle 7: Abgleich nicht-funktionale Anforderungen

8 Fazit

Die erste Phase der Entwicklung des UNITI-Projektes ist mit dem Abschluss dieser Arbeit abgeschlossen. Die Funktionalität der App ist derart umgesetzt, dass ein Benutzer seine Studien verwalten und eine auditorische Stimulation durchführen kann. Des Weiteren ist es für den Benutzer möglich, Fragebögen bzw. Quiz zu beantworten und auswerten zu lassen, sowie sich durch Informationen in Kapitelstruktur zu arbeiten. Zusätzlich werden gewünschte Meta-Daten, wie GPS, Lautstärke, Bilder und Nutzungsdaten des Gerätes erfasst. Diese letzten drei Punkte waren Teil dieser Arbeit und wurden in den vorherigen Kapiteln genauer beschrieben. Der erste Teil dieses Kapitels fasst die Erkenntnisse der Arbeit zusammen (8.1). Im zweiten Teil werden Ideen behandelt, die möglicherweise in einer Weiterentwicklung der App umsetzbar wären, um die Funktionsweise der App zu erweitern bzw. zu verbessern (8.2).

8.1 Zusammenfassung

Im Rahmen dieser Masterarbeit sind mehrere Module und eine Library für eine mHealth-App für Smartphones und Tablets mit Android Betriebssystem entstanden. Die Anforderungen wurden in mehreren Meetings definiert. Auch während der Arbeit kamen neue Anforderungen und Ideen hinzu, die in dieser Arbeit berücksichtigt worden sind.

Dabei wurde gezeigt, wie eine Datenstruktur erstellt werden kann, die dem Benutzer eine strukturierte und klare Übersicht bietet und dabei den Fortschritt dynamisch visualisiert. Dieser wird lokal, sowie am Server gespeichert, um die Daten des Benutzers auch bei einem Gerätewechsel bereitstellen zu können. Beim Erstellen der Struktur besteht die komplette Freiheit, in welcher Reihenfolge oder Anzahl sich die Quiz und Unterkapitel innerhalb eines Kapitels abwechseln.

Weiter wurde gezeigt, wie beliebig unterschiedliche Fragebögen mit unterschiedlichen Elementen, trotz ihrer Diversität in einer Datenstruktur dargestellt und beantwortet werden können. Beim Erfassen der Antwortsätze werden auch Meta-Daten erhoben, wobei der komplexeste Teil, die Erfassung der Nutzungsdaten, als Library ausgelagert wurde, um eine Wiederverwendung in anderen Projekten zu vereinfachen.

8.2 Ausblick

In diesem Abschnitt werden Ideen behandelt, die während der Entwicklung dieser App aufgekommen sind, um das Projekt zu verbessern.

8.2.1 Verbesserung der Visualisierung der Ergebnisse

Die Visualisierung der Ergebnisse beschränkt sich im Moment noch auf die Anzeige von Ergebnistexten mit deren Hinweistiteln. Für eine bessere Übersicht sind diese bei positiven und negativen Ergebnissen entsprechend farblich gekennzeichnet. Ein Fortschritt zu früheren Implementierungen ist die

Möglichkeit, auf einzelne Fragen genau eingehen zu können und dem Benutzer eine Erfolgsangabe erstellen zu können, ohne ihm dabei die korrekte Antwort jeder einzelnen Frage zu nennen. Für einen weiteren Entwicklungsschritt wäre es möglich, für die Fragebögen wissenschaftliche Skalen aus bestimmten Fragen zu errechnen und dem Benutzer die errechneten Werte in passenden Diagrammen anzuzeigen. Damit hätte der Benutzer die Möglichkeit, seine eigenen Angaben schnell und einfach zu beobachten und zu vergleichen. Ein Beispiel dafür wäre die Angabe zur Lautstärke seines Tinnitus. Daraus könnte der Benutzer möglicherweise ableiten, an welchen Wochentagen der Tinnitus stärker oder schwächer war, wodurch er einen Zusammenhang zu seinen Aktivitäten an den entsprechenden Tagen identifizieren könnte.

8.2.2 Anpassung an Tinnitus-Stärke und Frequenz

Eine weitere Verbesserungsidee für die Datenerfassung wäre eine Anpassung der Fragebögen für den entsprechenden Benutzer, aufgrund seiner Angabe der Tinnitusstärke, möglicherweise auch in Verbindung seiner Tinnitusfrequenz. In einem Projekt der Universität Ulm (Match Your Tinnitus [27]) ist ein Vorgehen zur Erfassung der Tinnitus-Frequenz bereits thematisiert und umgesetzt worden. Dies wäre eine passende Erweiterung des UNITI-Projektes, um noch genauer und individueller auf den Benutzer eingehen zu können und die Datenerfassung nochmals zu verbessern, bzw. zu erweitern.

8.2.3 Live-Informationen und Bewertungen

Die Versorgung der Benutzer mit aktuellen und sich dynamisch verändernden Informationen spielt eine immer größere Rolle. Dafür wäre eine Live-Ticker Funktionalität eine gute Erweiterung der UNITI-App, um den Benutzer ständig über die neuesten Erfolge in der Forschung oder Entwicklungen in der App-Nutzung zu informieren. Das dadurch entstehende Gefühl einer Community, sich zusammen an die Lösung eines alle gemeinsam betreffenden Problems zu machen, erhöht die Motivation eines jeden Einzelnen. Eine Bewertungsfunktion der Informationen in den bereitgestellten Kapiteln würde zusätzlich eine erhöhte Motivation, sowie eine bessere Rückmeldung für die Wissenschaftler, die diese erarbeitet haben, bieten.

8.2.4 Direkte Kommunikation mit medizinischem Personal oder anderen Benutzern

Um die Personalisierung und individuellen Behandlungsmöglichkeiten zu verbessern, wäre es möglich das UNITI-Projekt um eine kleine Chat-Funktion zu erweitern. Darin könnte man eine direkte Kommunikation zwischen einem Benutzer und entsprechend geschultem, medizinischen Personal realisieren. Durch den Zugriff auf die Daten des Benutzers hat der behandelnde Arzt oder Therapeut einen Einblick in den Verlauf der Erkrankung und kann so dem Benutzer direkt und schnell Hinweise bei Auffälligkeiten geben oder mögliche Behandlungsschritte einleiten.

Wenn man diese Chat-Idee weiterdenkt, wäre auch ein Forum für alle Benutzer eine mögliche Verbesserung. Darin könnte, wie durch die Live-Informationen, eine Community entstehen, die ihre Erfahrungen untereinander austauschen und sich gegenseitig Hilfestellung leisten kann.

9 Literaturverzeichnis

[1] Cederroth, CR, Gallus, S, Hall, DA, Kleinjung, T, Langguth, B, Maruotti, A, Meyer, M, Norena, A, Probst, T, Pryss, R, Searchfield, G, Shekhawat, G, Spiliopoulou, M, Vanneste, S, Schlee, W (2019): Editorial: Towards an Understanding of Tinnitus Heterogeneity. Frontiers in Aging Neuroscience, 11:53.

[2] Schlee, W, Hall, DA, Canlon, B, Cima, RFF, Kleine, E de, Hauck, F, Huber, A, Gallus, S, Kleinjung, T, Kypraios, T, Langguth, B, Lopez-Escamez, JA, Lugo, A, Meyer, M, Mielczarek, M, Norena, A, Pfiffner, F, Pryss, RC, Reichert, M, Requena, T, Schecklmann, M, van Dijk, P, van de Heyning, P, Weisz, N, Cederroth, CR (2018): Innovations in Doctoral Training and Research on Tinnitus: The European School on Interdisciplinary Tinnitus Research (ESIT) Perspective. Frontiers in Aging Neuroscience, 9:447.

[3] Robert-Koch-Institut: Corona-Warn-App. https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/WarnApp/Warn_App.html. Abgerufen am 17.09.2020.

[4] Johner Institut: Software & IEC 62304. https://www.johner-institut.de/blog/category/iec-62304-medizinische-software/#:~:text=Die%20IEC%2062304%20ist%20eine,das%20Software%2DRisikomanagement%20(inkl.&text=sowie%20Software%2C%20die%20Teil%20eines%20Medizinprodukts%20ist. Abgerufen am 02.10.2020.

[5] Bundesgesundheitsministerium: Medizinproduktegesetz. https://www.bundesgesundheitsministerium.de/themen/gesundheitswesen/medizinprodukte/definition-und-wirtschaftliche-bedeutung.html. Abgerufen am 01.10.2020.

[6] Haug, J (2020): Konzeption und Realisierung eines Moduls für auditorische Stimulationen für eine multizentrische und multinationale mHealth-App für ein paneuropäische Tinnitus-Studie.

[7] UNITI-Projekt. https://cordis.europa.eu/project/id/848261/de. Abgerufen am 17.09.2020.

[8] Probst, T, Pryss, R, Langguth, B, Schlee, W (2016): Emotional states as mediators between tinnitus loudness and tinnitus distress in daily life: Results from the TrackYourTinnitus application. Scientific Reports, 6.

[9] Probst, T, Pryss, R, Langguth, B, Spiliopoulou, M, Landgrebe, M, Vesala, M, Harrison, S, Schobel, J, Reichert, M, Stach, M, Schlee, W (2017): Outpatient Tinnitus Clinic, Self-Help Web Platform, or Mobile Application to Recruit Tinnitus Study Samples? Frontiers in Aging Neuroscience, 9:113--113.

[10] Probst, T, Pryss, RC, Langguth, B, Rauschecker, JP, Schobel, J, Reichert, M, Spiliopoulou, M, Schlee, W, Zimmermann, J (2017): Does Tinnitus Depend on Time-of-Day? An Ecological Momentary Assessment Study with the “TrackYourTinnitus” Application. Frontiers in Aging Neuroscience, 9:253.

[11] Pryss, R, Probst, T, Schlee, W, Schobel, J, Langguth, B, Neff, P, Spiliopoulou, M, Reichert, M (2017): Mobile Crowdsensing for the Juxtaposition of Realtime Assessments and Retrospective

Reporting for Neuropsychiatric Symptoms. In: Press, ICS (Hrsg), *30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017)*.

[12] Pryss, R, Reichert, M, Herrmann, J, Langguth, B, Schlee, W (2015): Mobile Crowd Sensing in Clinical and Psychological Trials ? A Case Study. In: Press, ICS (Hrsg), *28th IEEE Int'l Symposium on Computer-Based Medical Systems*.

[13] Pryss, R, Reichert, M, Langguth, B, Schlee, W (2015): Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy and Research. In: Press, ICS (Hrsg), *IEEE 4th International Conference on Mobile Services (MS 2015)*.

[14] Pryss, R, Schlee, W, Langguth, B, Reichert, M (2017): Mobile Crowdsensing Services for Tinnitus Assessment and Patient Feedback. In: Press, ICS (Hrsg), *6th IEEE International Conference on AI \& Mobile Services (IEEE AIMS 2017)*.

[15] Schlee, W, Pryss, R, Probst, T, Schobel, J, Bachmeier, A, Reichert, M, Langguth, B (2016): Measuring the Moment-to-Moment Variability of Tinnitus: The TrackYourTinnitus Smart Phone App. Frontiers in Aging Neuroscience, 8:294--294.

[16] Herrmann, J (2014): Konzeption und technische Realisierung eines mobilen Frameworks zur Unterstützung tinnitusgeschädigter Patienten.

[17] Klecina, B (2016): Assess Your Stress: Conceptual re-design of the Track Your Tinnitus system for measuring stress at the workplace. http://dbis.eprints.uni-ulm.de/1395/1/MA_KL_2016.pdf. Abgerufen am 11.09.2020.

[18] Bundesministerium für Gesundheit: Präventionsgesetz. https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/p/praeventionsgesetz.html. Abgerufen am 11.09.2020.

[19] Universität Ulm: MobileTx. https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/mobiletx/. Abgerufen am 29.09.2020.

[20] Schickler, M (2020): Ein Rahmenwerk zur mobilen Unterstützung therapeutischer Interventionen. http://dbis.eprints.uni-ulm.de/1714/. Abgerufen am 29.09.2020.

[21] mynoise GmbH: Kalmeda. https://www.kalmeda.de/. Abgerufen am 02.10.2020.

[22] Masse, M (2011): REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O´Reilly Media, Inc.

[23] Google Inc: Google Developer. https://developer.android.com/. Abgerufen am 20.10.2020.

[24] Bucanek, J (2009): Model-view-controller pattern. Learn Objective-C for Java Developers:353–402.

[25] App Annie: Apps Used 2017. https://www.appannie.com/de/insights/market-data/apps-used-2017/. Abgerufen am 20.09.2020.

[26] Morris, JD (1995): Observations: SAM: the Self-Assessment Manikin; an efficient cross-cultural measurement of emotional response. Journal of advertising research, 35(6):63–68.

[27] Stampf, A: Konzeption und Realisierung einer mobilen Anwendung zur Bestimmung der Tinnitusfrequenz am Beispiel von Android. http://dbis.eprints.uni-ulm.de/1657/1/BA_Stampf_2018.pdf. Abgerufen am 01.10.2020.

Eigenständigkeitserklärung

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

Ulm, den 29. Oktober 2020 …….

Fabian Haug