ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fa
In
un
Ins
Inf

Konzeption und Realisierung einer Plattform zur Unterstützung von therapeutischen Übungen und Aufgaben

Masterarbeit

Vorgelegt von:
Heiko Foschum
heiko.foschum@gmx.de

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

Betreuer:
Dipl.-Ing. Marc Schickler

2016

Fassung 8. November 2016

Satz: PDF-LaTeX 2ε

Inhaltsverzeichnis

1 Einleitung

Bei der Betreuung von Patienten durch einen Psychotherapeuten sind Übungen und Fragebögen für Zuhause oft Bestandteil der Behandlung. Die Aufgaben werden in der Sitzung besprochen und sollen dann vom Patienten selbstständig und ohne Betreuung bearbeitet werden. Diese Aufgaben sind wichtiger Bestandteil für den Behandlungserfolg[5]. Sie helfen beim Transfer der Therapie in den Alltag. Jedoch ist eins der größten Probleme bei dieser Vorgehensweise, dass die Aufgaben oft gar nicht oder nur unvollständig erledigt werden [4].

1.1 Ziel der Arbeit

Im Zuge dieser Arbeit wurde das Konzept einer Plattform entwickelt und umgesetzt, welches auf der einen Seite dem Patienten hilft sein Aufgabe zu erledigen. Auf der anderen Seite aber auch dem Therapeuten Daten und Feedback zu den Übungen liefert. Auf Grundlage aktueller Software wurde so eine Plattform implementiert, die es dem Therapeuten erlaubt seine Patienten innerhalb einer Webanwendung zu verwalten. Er hat die Möglichkeit, Therapieaufgaben und Fragebögen zu gestalten und diese dem jeweiligen Patienten zuzuordnen. Anhand einer App wird der Patient sobald das Zeitintervall angefangen hat, daran erinnert seine Aufgabe zu erledigen und zeigt die Aufgabenbeschreibung an. Am Ende des definierten Zeitintervalls hat der Patient die Möglichkeit den zugeordneten Fragebogen auszufüllen. Welcher nach Beendigung wiederum direkt vom Therapeuten eingesehen werden kann. Ziel ist es vor allem, das die Aufgaben Vollständig und immer erledigt werden. Aber auch die erzeugte Resonanz und Daten soll zum Therapieerfolg beitragen. Hierdurch können die Therapeutischen Aufgaben verbessert werden und auf den einzelnen Patienten abgestimmt werden. In einer Studie von Helbig und Fehm [**?**] gaben nur 11% der befragten an die Aufgabe nicht erledigt zu haben. Nicht einmal die Hälfte jedoch gaben an, die Aufgabe genau so wie beschreiben ausgeführt zu haben. Durch das Anzeigen der Aufgabenbeschreibung soll dem Patienten geholfen werden die Aufgabe genau so durchzuführen wie vorgegeben. Durch die angehängten Materialien in jeglicher digitalen Form kann der Patient vielseitig bei seinem persönlichen Erfolg unterstützt werden.

1.2 Aufbau der Arbeit

2 Grundlagen

Im diesem Kapitel werden die Theoretischen Grundlagen der verwendeten Technologien im einzelnen betrachtet.

REST, HTML(5),HTTP(GET,SET,...),Client-Server Architektur, JavaScript, Angular, Mongoose MongoDB Client Webanwendungen, single page application Bootstrap Zentralisierung der Daten (Sicherheit)

Android debug bridge ADB

2.1 Daten in einer URL übergeben / auch mehrere

2.2 REST-Service

2.3 HTML 5

2.4 Zentralisierung der Daten

2.5 Sicherheit

2.6 BPMN-IO

2.6.1 Aufbau der exportierten XML Datei

2.7 Anwendungsentwicklung

Am Anfang jeder mobilen Anwendung steht die Entscheidung auf welchen Geräten diese laufen soll. Natürlich besteht die Möglichkeit eine Anwendung nativ für eine bestimmte Plattform zu entwickeln. Um die Anwendung auf anderen Plattformen zu bringen ist es dann jedoch nötig diese praktisch neu zu implementieren. Aus diesem Dilemma heraus entstanden verschiedenen Frameworks die es dem Entwickler möglich machen direkt für mehrere Plattformen zu entwickeln.

Hierdurch ist es dann möglich seinen Code nur einmal zu schreiben und diesen dann direkt, mithilfe eines Frameworks, wie zum Beispiel eines Browsers auf den verschiedene Plattformen laufen zu lassen. Es müssen dann nur einige wenige Abschnitte des Codes nativ für die jeweilige Plattform geschrieben werden, wenn diese nicht verallgemeinerbar sind bzw. nicht schon durch eine Bibliothek vereinheitlicht wurden. Ein Beispiel wäre hier der Zugriff auf Dateien, hier ist es nicht möglich den Zugriff direkt zu verallgemeinern, da die verschiedenen Betriebssysteme unterschiedliche Dateisysteme Verwenden.

Eine der Grundvoraussetzungen dieser Arbeit sollte die Verfügbarkeit für möglichst viele Patienten sein. Aus diesem Grund wurde vorab untersucht durch welche Techniken es möglich ist, die Hürde der vielen verschiedenen Endgeräte und damit verbundenen Betriebssysteme umgehen zu können. Durch die enge Bindung der Therapeutensoftware an den Server viel die Wahl von vorn herein auf eine sogenannte **Single-Page-Application** in Form einer Website.

2.7.1 Anwendungsentwicklung für mobile Geräte

Das größte Problem der Anwendungsentwicklung für Smartphones ist, dass es viele verschiedene mobile Betriebssystemen gibt. Ein kleiner Trost für jeden Entwickler ist dabei, dass sich der Hauptteil der Nutzer auf einige wenige Plattformen beschränkt. Mit einer Anwendung für Android, IOS und Windows kann man somit den größten Teil der Nutzer erreichen. Diese decken über 99% [**?**] der der Smartphone Benutzer ab.

PhoneGap + Ionic[1 und 2] "What the ionic framework provides is the native look and feel and the user interface interactions"

Ionic - Cordova

Proxys zum Debuggen im Browser - Das CORS-Problem

Xamarin

(http://thinkapps.com/blog/development/develop-for-ios-v-android-cross-platform-tools/)

Plattform spezifische Entwicklung

Durch die nativen Anwendungsentwicklung wird eine mobile Anwendung nur jeweils für eine Plattform entwickelt. Hierdurch hat man aber den Vorteil, dass man durch wegfallen jegliches Frameworks die maximale Geschwindigkeit und minimale Reaktionszeiten erzielt. Dies ist vor allem wichtig bei Rechenintensiven Anwendungen wie Spielen oder ähnlichem. Des weiteren hat man durch die native Entwicklung den vollen Zugriff auf jegliche im Smartphone verbaute Hardware und Betriebssystem Features. Bei Verwendung eines Frameworks müssen hier oft Abstriche gemacht werden.

die entwicklung für die einzelnen Plattformen aufzeigen

Cross Plattform Entwicklung

Von Cross Plattform Entwicklung spricht man, wenn der Code der entwickelten Anwendung nicht nur für eine spezifische Plattform verwendbar ist. Durch verschiedene Frameworks ist es möglich, dass der Entwickler die Anwendung nur ein mal schreib und diese dann auf den meist verwendeten Betriebssystemen läuft oder für diese Übersetzt wird. Dies wird Grundlegend auf **3 verschiedenen** Wegen erreicht:

Web-Apps / Webanwendungen laufen im, vom Betriebssystem zur Verfügung gestellten Webbrowser. Hierbei ist das Betriebssystem völlig egal, es werden nur einige Voraussetzungen an den Webbrowser gestellt. Ebenso wird eine aktive Internetverbindung benötigt wenn die App ausschließlich online zur Verfügung steht. Um dies zu umgehen wurde in HTML5 verschiedenen Möglichkeiten eingebaut um Code und Daten lokal zwischenzuspeichern zu können. Dies bietet den Vorteil, dass die App schnell veröffentlicht und aktualisiert werden kann. Wenn der Nutzer die Anwendung mit einer aktiven Internet Verbindung öffnet, aktualisiert sich die App automatisch. Der Nachteil dadurch das die App im Webbrowser läuft ist, dass man nur Zugriff auf die Funktionen hat, die der verwendete Browser bietet. Da ein Webbrowser nicht zwangsläufig auf einem mobilen Gerät mit diversen Sensoren laufen muss, ist der zugriff auf die gängigen Sensoren in einem mobilen Endgerät meinst sehr eingeschränkt. Auf Grund dessen muss man sich bei der Web-App Entwicklung meist auf Kamera, Datenpersistenz und GPS beschränken. Durch die Zentralisierung der App sieht diese auf jedem Gerät gleich aus. Dies erscheint im ersten Augenblick positiv, der Benutzer jedoch ist an das Bedienkonzept seines Betriebssystems gewohnt.

Hybride Apps basieren wie auch die Web-Apps auf den Webtechnologien HTML5, CSS und JavaScript und laufen aber im Gegensatz zu dem Web-Apps in einem mitgelieferten Minibrowser(Webview Container).Dieser Container ist in einer nativen App eingebettet was den Zugriff auf die System APIs des Geräts erlaubt. Durch das einbetten von nativen Code ist der Zugriff auf alle vom Betriebssystem zur Verfügung gestellten Funktionen möglich. wie z.b. GPS, Kamera, Betriebssystem Benachrichtigungen und die verschiedenen Sensoren(Beschleunigung, Umgebungslicht, Hall, Gyroskop,...). Da die Anwendung im Herzen eine Internetseite ist, ist es implizit möglich diese unter einer URL zu veröffentlichen und wie eine Web-App zu behandeln. Dann jedoch auch mit den Einschränkungen die diese bietet. Für den Entwickler sehr angenehm sind oft angebotene "Live-View"Technologien. Diese erlauben durch speichern einer HTML oder JavaScript Projektdatei ein neu laden der App im Webbrowser zu initiieren. Dies bietet ein schnelles Designen der Oberfläche und implementieren grundlegender Funktionen. System eigene Funktionen müssen jedoch auf einem Gerät oder im Emulator getestet werden. Hierzu ist dann compilieren, packen und ausliefern notwendig, was etwas Zeit benötigt.

inhalt des links mit reinbringen. link im kommentar

Ionic machts für jedes Gerät individuell.....

Wikipediaartikel is nochmal anders beschrieben

Native Cross Plattform Apps mit gemeinsamer Codebasis verspricht Xamarin, ein Projekt das auf der quellen-offenen Implementierung von Microsofts .NET Framework, Mono basiert [1]. Mittels des Mono Frameworks ist es möglich C# Code auf verschiedenen Betriebssystemen laufen zu lassen. Der geschriebene Code wird dann beim Bauvorgang der App in Nativen Code umgesetzt. Es fällt somit im Gegensatz zu den Hybriden Apps, das Framework auf dem Endgerät weg, wodurch Rechenkapazität gespart wird. Was diese Plattform vorallem für rechenintensive Anwendungen interessant macht. Bei der Erstellung einer Cross Plattform App mittels Xamarin muss man zuerst festlegen, auf welchen Betriebssystemen diese später laufen soll. Aus dieser Auswahl wird eine Schnittmenge der Funktionen gebildet die zur Verfügung stehen und eine sogenannte **Portable Class Library** erzeugt. Diese fungiert dann als einheitliche Schnittstelle bei Api aufrufen an das Betriebssystem. Es sind jedoch nur die Funktionen Plattform-übergreifend nutzbar, welche von allen ausgewählten Betriebssystemen angeboten werden und von Mono implementiert sind. Sollte der Entwickler Zugriff auf System-eigene Funktionen benötigen, welche nicht von Mono abgedeckt werden, hat er mit Xamarin.iOS und Xamarin.Android die Möglichkeit alle System-eigenen Funktionen verwenden zu können. Zur Vereinheitlichung unter einer gemeinsamen GUI ist es dann aber nötig die Funktion für jedes Betriebssystem separat zu entwickeln und zu pflegen. Mittels Xamarin.Forms ist es möglich die GUI der App Plattform-übergreifend zu gestalten. Hierzu wird die Oberfläche mittels einer eigenen, XML ähnlichen, Sprache namens **Extensible Application Markup Language (kurz XAML)** beschrieben. Die so spezifizierten Views werden dann auf die einzelnen Betriebssysteme zugeschnitten, um dem Anwender eine gewohnte Umgebung zu bieten.

2.8 Software Analyse

und die Implementierung der Anwendung vorgestellt. Im Entwurf wird die Vision, die Anforderungsanalyse und der daraus resultierende Architekturentwurf aufgezeigt. Darauf folgt eine Software Analyse, welche sich damit auseinander setzt welche Frameworks verwendet werden könnten um die gestellte Aufgabe zu lösen. Anschließend wird im Unterkapitel Implementierung [4] auf selbige eingegangen. Dies beinhaltet das zugrundeliegende Datenmodell so wie die Erläuterung der Implementierungen des Servers sowie der beiden Clients für Therapeut und Patient.

3 Entwurf

Im folgenden Kapitel wird der Entwurf der Anwendung vorgestellt. Dies beinhaltet die Vision, welche aufzeigt was von der zu entwickelten Anwendung erwartet wird. In der darauf folgenden Anforderungsanalyse [3.2] wird sich damit auseinander gesetzt, was man zum erreichen dieser Ziele benötigt.

nochmal überarbeiten wenn Kapitel Final

3.1 Vision

Mit Hilfe der zu entwickelnden Plattform soll es möglich sein Therapieaufgaben im Zuge einer Therapeutischen Betreuung zwischen Therapeut und Patient auszutauschen.

Der Therapeut möchte in einer Desktop Anwendung seine Patienten verwalten können, sowie Vorlagen für Therapeutische Aufgaben. Um die Aufgaben dann den einzelnen Patienten zuweisen zu können. Ebenso soll es für den Therapeuten möglich sein Fragebögen mit Antwort abhängigen Fragen zu erstellen um diese nach Therapeutischen Aufgaben von den Patienten beantworten zu lassen.

Um möglichst viele Patienten mit dieser Plattform unterstützen zu können soll diese für möglichst viele Smartphone Betriebssysteme implementiert werden. Die wichtigsten sind hierbei Android, IOS und WindowsPhone. Die Mobile App soll dann auf das Profil des Patienten, dass durch den Therapeuten angelegt wurde, zugreifen. Hierbei werden die Daten über die zugewiesenen Aufgaben und Fragebögen abgerufen. Zu den vom Therapeut vorgegebenen Zeiten werden dann Betriebssystem Erinnerungen erstellt. Welche den Patienten an das erledigen der Aufgabe erinnern. Die vorgegebene Zeitspanne soll vom Patienten verändert werden können. Nach vollenden der Aufgabe oder am Ende der gegebenen Zeitspanne soll der Patient die Möglichkeit haben den zugehörigen Fragebogen auszufüllen.

3.2 Anforderungsanalyse

Um eine gemeinsame Datenbasis zu schaffen muss es einen im Internet erreichbaren Server geben. Dieser hält die vom Therapeuten eingegebene Daten. Dies erreicht man durch einen beliebigen persistenten Speicher, wie Beispielsweise eine Datenbank oder ein Dateisystem. Der Server bietet hierbei eine Webschnittstelle zur Datenbank an, über welche die mobile Anwendung Zugriff auf die Daten hat.

Die Desktop Anwendung soll dem Therapeuten die Oberfläche bieten mithilfe welcher es ihm möglich sein muss die Patienten, Therapieaufgaben und Fragebögen verwalten zu können. Die Patienten müssen angezeigt, angelegt, bearbeitet und gelöscht werden können. In der Anzeige eines Patienten soll der Therapeut die zuvor erstellten Therapieaufgaben und Fragebögen zuweisen können. Die Aufgaben sollen neben Name und Beschreibung auch verschieden Medien beinhalten können. Bei der Zuweisung gibt der Therapeut das/die Ereignisse(Zeit, Ort, etc.) an zu denen der Patient an die Aufgabe erinnert werden soll.

Die mobile Anwendung muss auf möglichst vielen mobilen Plattformen lauffähig sein. Der Patient muss nach dem ersten Start der mobilen Anwendung Verbindung zu seinem Profil herstellen, etwa durch Eingabe eines Benutzernamens oder einer ID. Anschließend müssen die Daten zu den hinterlegten Therapieaufgaben über die Webschnittstelle des Servers geladen und weiterverarbeitet werden. Die Ereignisse zu den Aufgaben werden in so weit verarbeitet, dass der Patient an die Therapieaufgabe erinnert wird, wenn der Erledigungszeitraum beginnt. Die mobile Anwendung soll die Möglichkeit bieten, die vom Therapeuten vorgegebenen Ereignisse anzupassen.

3.2.1 Funktionale Anforderungen

Aus der Vision [Kapitel 3.1] und der Anforderungsanalyse [Kapitel 3.2] können folgende Funktionale Anforderungen abgeleitet werden.

Therapeuten Client

Für den Client des Therapeuten sind dies die in Tabelle 3.2.1 dargestellten Anforderungen.

Anforderung	Beschreibung
Patientenverwaltung	Die Patienten sollen mit ihren Persönlichen Daten verwaltet werden können
Aufgabenverwaltung	Es muss möglich sein, Aufgaben zu erstellen und die Patienten zuzuweisen
Materialien	Den Aufgaben sollen verschiedenste Materialien angehängt werden können
Kontrollmechanismen	Der Therapeut soll den Aufgaben Kontrollmechanismen hinzufügen können
Fragebogenverwaltung	Es sollen Fragebögen mit antwortabhängigen Fragen erstellt und den Patienten zugewiesen

Patienten Client

Für den Client des Patienten sind dies die in Tabelle 3.2.1 dargestellten Anforderungen.

Anforderung	Beschreibung
Datenpersistenz	Die vom Therapeuten eingegebene Daten müssen persistent gespeichert werden
globale Datenpersistenz	Die gespeicherten Daten sollen auf allen Geräten synchron sein
Plattform unabhängig	Die Anwendung soll von möglichst vielen Geräten aus verwendet werden können

Tabelle 3.1: Nicht-Funktionale Anforderungen an die Clients

Anforderung	Beschreibung
User Identifikation	Der Patient stellt Verbindung zu dem von Therapeut erste
Aufgabenanzeige	Übersicht der zugewiesenen Therapeutischen Au
Detailanzeige der Aufgabe	Eine detaillierte Anzeige der Aufgabe mit Name, Erklärung und a
Anzeige der Materialien	Die angehängten Materialien müssen vom Patienten innerhalb des Client
Verändern des Erledigungskontext	Der von Therapeuten vorgegebene Erledigungskontext soll vom Pa
Erinnerung	Der Patient soll an die Aufgabe erinnert werd
Fragebogen beantworten	Es muss möglich sein, einen Fragebogen zu bean

3.2.2 Nicht-Funktionale Anforderungen

Aus der Vision[3.1] und der Anforderungsanalyse[3.2] können folgende Nicht-Funktionale Anforderungen abgeleitet werden. Diese sind für beide Clients die selben, weswegen hier im Gegensatz zu den Funktionalen Anforderungen keine Unterscheidung getroffen wird.

4 Implementierung

In diesem Kapitel wird auf die Implementierung der Anwendung eingegangen. Dies beinhaltet den Architekturentwurf, das zugrundeliegende Datenmodell, sowie die Implementierungen des Servers und der beiden Clients für Therapeut und Patient.

Um eine gemeinsame Datenbasis zwischen Therapeuten Client und Patienten App zu schaffen wurde eine Datenbank Server aufgesetzt. Dieser beinhaltet eine Verbindung zu einer MongoDB Datenbank. Des weiteren wurde eine Webserver Implementiert welcher eine Web API anbietet. Dieser fungiert als Bindeglied zwischen Internet und der Datenbank. Er implementiert die HTTP Methoden GET, PUT, POST und DELETE. Hierdurch können von überall auf der Welt die auf dem Server gespeicherten Daten zu Patienten,Therapieaufgaben und Fragebögen abgerufen, verändert oder gelöscht werden.

Als weiterer Baustein für die Plattform wurde eine Client für den Therapeuten auf Basis einer Webseite implementiert. Der Vorteil einer Website besteht darin, dass es völlig egal is welches Betriebssystem der Therapeut verwendet. Es wird lediglich ein Webbrowser und eine aktive Internetverbindung benötigt. Änderungen die an den Patientendaten vorgenommen werden, werden direkt auf dem Datenbank Server gespeichert. Der Client selbst hält dabei keine Daten, außer die temporären, welche zur Anzeige benötigt werden.

Für den Patienten wurde eine Cross Plattform App entwickelt, welche sich die benötigten Daten ebenso von dem oben erwähnten Datenbank Server lädt. Die App basiert auf den Technologien AngularJS, Cordova und Ionic. Durch sie wird der Patient mittels einer Betriebssystem Erinnerung an seine Therapeutische Aufgabe erinnert wenn das Start Event eingetroffen ist. Durch Klick auf die Erinnerung wird dem Patient die Aufgabenbeschreibung angezeigt. Wodurch ihm die korrekte Ausführung der Aufgabe erleichtert werden soll.

Durch die Verwendung von AngularJS für den Therapeuten Client wurde erreicht, dass beide mittels JavaScript und HTML5 implementiert werden können.

4.1 Datenmodell

Um eine einheitliche Datenstruktur auf den beiden Clients und dem Server zu behalten, wurde das folgende Datenmodell entworfen.

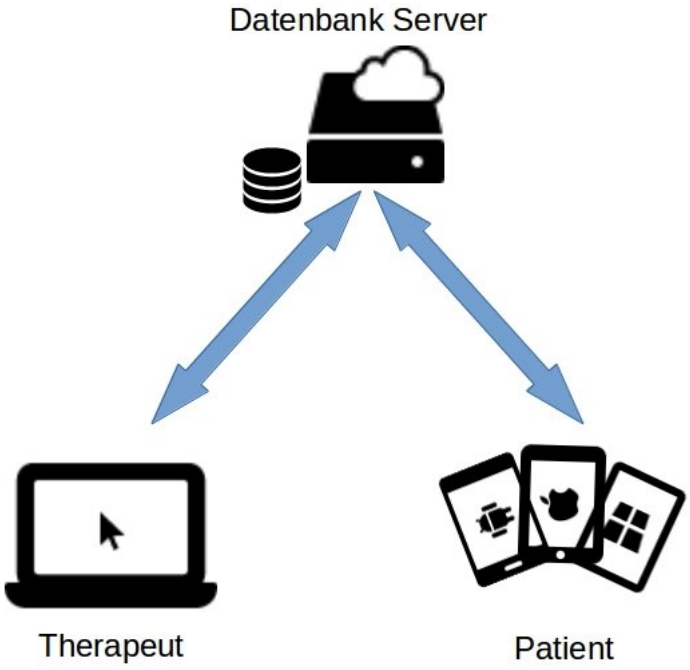

Abbildung 4.1: Client-Server Architektur der Plattform

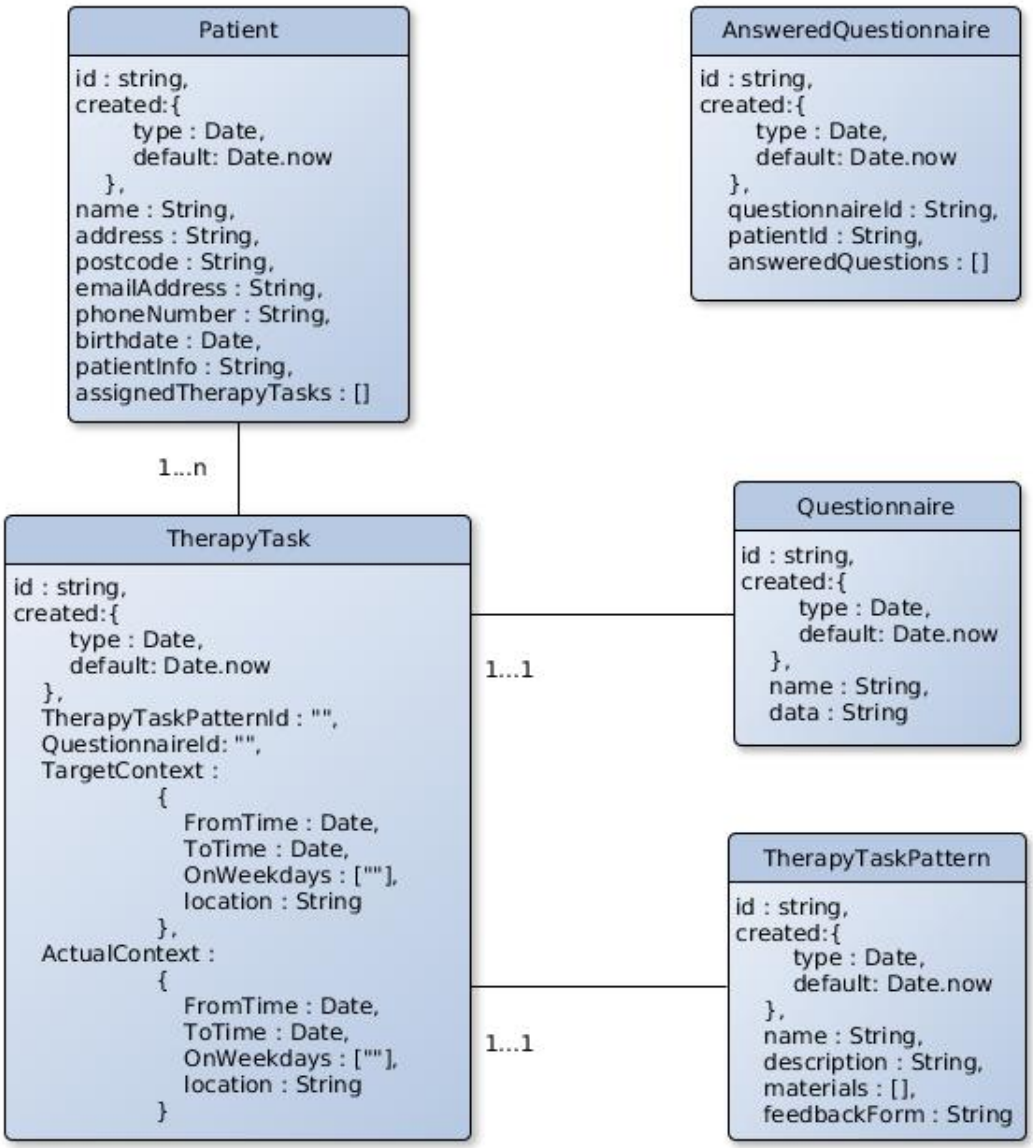

Abbildung 4.2: Visualisierung des Datenmodells

Auf Grundlage des Datenmodells [Abbildung 4.2] können alle innerhalb der Anwendung benötigten Daten in der Datenbank gespeichert werden. Jede Datenstruktur beinhaltet eine global einzigartige **ID**. Diese stellt sicher das die Objekte in der Datenbank wieder gefunden werden kann. Allen voran möchte der Therapeut seine Patienten verwalten können. Hierzu wurde eine Klasse "Patientängelegt. Diese beinhaltet neben der ID, Felder für den Namen, die Adresse und andere Personenbezogene Daten, die in erster Linie dazu verwendet werden, um den Patienten eindeutig zu Identifizieren. Des weiteren beinhaltet diese Klasse ein Array in dem die zugeordneten Therapieaufgaben gespeichert werden können. Somit is es dem Therapeuten möglich den angelegten Patienten beliebig viele Therapieaufgaben zuzuordnen.

Diese Informationen werden in der Klasse **TherapyTask** gespeichert. Diese ist zwar immer Bestandteil der Klasse Patient, wurde aber zur besseren Abgrenzung von dieser separiert. TherapyTask beinhaltet jeweils ein Feld für die Id eines Fragebogens sowie die Id einer Aufgabenvorlage. Hierdurch ist es möglich die Daten des zugehörigen, vom Therapeut ausgewählten, Fragebogen und die der Aufgabe vom Server zu laden und im Client anzuzeigen.

Die vom Therapeuten vorgegebenen Zeitintervalle oder Orte zu/an denen die Aufgabe erledigt werden sollte können unter dem Eintrag **TargetContext** gespeichert werden. Zum Abschluss der Arbeit besteht dieser Kontext aus der **FromTime**, welche den Start der zeitlichen Erledigungszeitintervalls beschreibt und der **ToTime** welche das Ende des Intervalls angibt. In dem Array **OnWeekdays** werden die Wochentage gespeichert an denen die Übung zu erledigen ist. Der Variable **location** soll in Zukunft dazu verwendet werden können um das Startevent nicht nur Zeit sonder auch Ortsabhängig definieren zu können. Unter dem Eintrag **ActualContext** können die vom Patienten veränderten Erledigungsdaten gespeichert werden.

Die erstellten Fragebögen werden in der Klasse **Questionnaire** gespeichert. Diese beinhaltet neben einem Feld für den Namen des Fragebogens ein weiteres um die vom Fragebogendesigner erstellten Daten im XML Format als String zu speichern. Die Klasse **TherapyTaskPattern** wird verwendet um die Vorlagen für Therapieaufgaben zu speichern. Diese beinhaltet neben dem Namen und der Beschreibung der Aufgabe ein Array, in welchem angehängte Materialien gespeichert werden können. Dies können jegliche Art von Zeichenfolgen sein, werden aber bisher nur verwendet um Internetadressen zu speichern, da sich hinter diesen theoretisch jede Information verbergen kann.

4.1.1 REST-API

Zur Implementierung des in Kapitel 4 angesprochenen Webservers wurde *NodeJS* [2] als Basis verwendet. Dies bietet eine JavaScript Laufzeitumgebung. Mittels des Package-Managers *npm* wurde dann das Web-Framework *express* installiert. Durch dieses Framework bietet eine grundlegende Implementierung eines Webservers. Dieser bietet einen lauffähigen HTTP Server sowie eine Grundlegende Implementierung eines sogenannten Routers und einem Webfrontend. Um eine spätere Trennung von REST-API und Therapeuten Client einfach zu machen sind diese von

einander getrennt. Hierzu wurden aus dem Express-Starter-Projekt alle Abhängigkeiten entfernt die zur Darstellung einer Website benötigt werden.

Wie in der *package.json* unter ßcripts> ßtartängegeben wird beim Start des Servers durch *npm start* die Datei www.js ausgeführt. Diese Startet mittels des *http* Moduls einen Webserver welcher auf den angegebene Port horcht. Zusätzlich wird durch einen *require* die server.js eingebunden in welcher der Server weiter spezifiziert ist. In dieser Datei wird express eingebunden und gestartet. Unter Verwendung des *mongoose* Moduls wird die Verbindung zur Datenbank durch *mongoose.connect('mongodb:///localhost/patient')* hergestellt. Diese muss vorher auf dem System installiert ud gestartet werden.

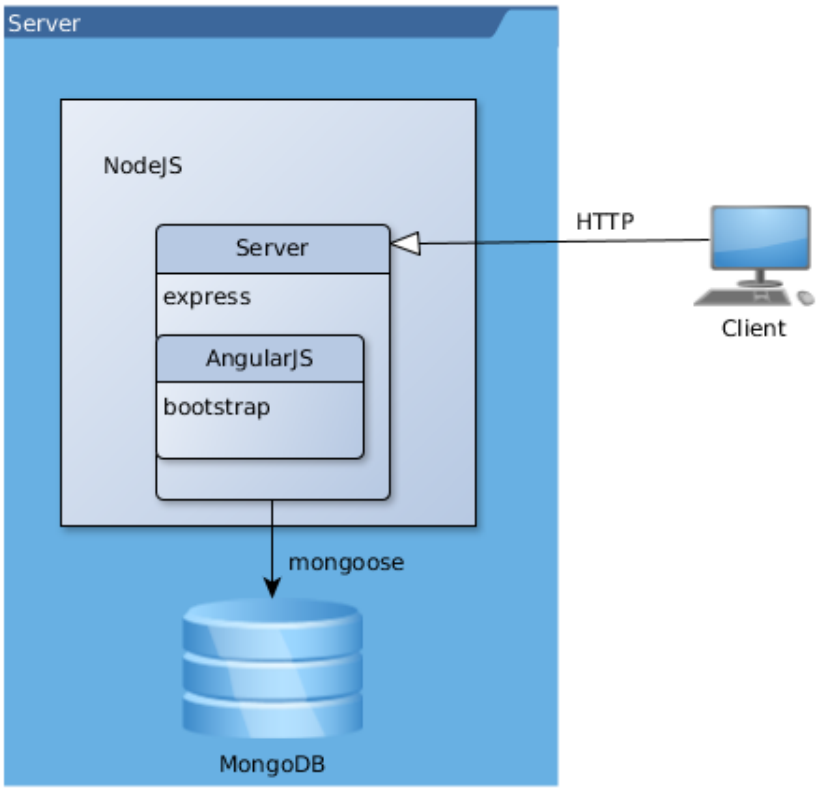

Abbildung 4.3: Übersicht über die Architektur des Servers

Anhand des eingebauten Routers können durch unterschiedlicher URLs bestimmte Programmteile auszuführen. Mittels *express.use('/patientAPI',patient)* wurde eine Universelle Route für die Patienten-API erstellt. Diese leitet die HTTP Anfrage an die hinter *patient* spezifizierte Datei (patientsAPI.js) weiter. Hier werden die Routen der Patienten API näher spezifiziert.

Zur Umsetzung der API wurden diesem Router unterschiedliche URL-Pfade mittels *router.get('/:patientId', patients.show);* bekannt gemacht. Durch hinzufügen dieser Route wird der Server unter seiner gewohnten Adresse + /**:patientId** die Antwort der Funktion **patients.show()** zurückgeben. Diese Route wird fortan für alle GET Anfragen in Form von /patientAPI/«beliebigerString» verwendet. Der Zusatz :patientId gibt an, dass die angehängte Daten innerhalb von patients.show() unter

der Variable PatientId abgerufen werden kann. Innerhalb von patients.show() wird dann mittels *req.params.patientId* die ID des angefragten Patientenobjekts ausgelesen.

Durch das *npm* Paket *mongoose* wird eine Verbindung zu einer auf dem Server laufenden MongoDB Datenbank hergestellt. In dieser werden die Daten zu den Patienten, Aufgaben und Fragebögen gespeichert. *mongoose* wurden dann die in Abbildung 4.2 gezeigten Modelle der Klassen bekannt gemacht. Hierzu wurden Schemas zu dem Klassen definiert und unter eines einzigartigen Namen den mongoose Modellen mittels *mongoose.model(«Modell Name«,«Schema Objekt»)* hinzugefügt. Nun kann mittels *Patient.load(«ID»,function())* ein Patient mittels seiner ID aus der Datenbank geladen werden. Anschließen wurden Routen zum aufrufen aller Patienten und zum speichern, ändern und löschen einzelner. Für die Web-APIs der Therapeutischen Aufgaben, Fragebögen (Muster und Beantwortete) wurde die Routen und Schemas analog erstellt.

4.2 Therapeuten Anwendung

Eine weitere Route wurde eingerichtet, unter welcher sich der Nutzer, in diesem Fall der Therapeut, eine sogenannte **Single Page Application** in Form einer HTML und damit verbundene JS und CSS Dateien herunter lädt. Diese beinhalten die komplette Logik und Formatierung der Anwendung. Sie basiert auf dem AngularJS Framework und wird mittels HTML5 CSS(Cascading Style Sheets) und JavaScript implementiert. Durch AngularJS bekommt man eine komplett lauffähiges Grundgerüst einer Single Page Application.

nicht nur Anwendung erklären, mehr auf Implemetierung eingehen

Durch die Verwendung von *Twitter Bootstrap* wird der Browserapplikation eine Endgerät abhängige Darstellung hinzugefügt. Das bedeutet, dass die Applikation der Größe des Displays, welches sie anzeigt, angepasst wird. Bei kleineren Geräten wird unter anderem die Navigationsleiste in ein SideMenue verschoben, welches auf/-zugeklappt werden kann. Um die horizontal liegende Navigationsleiste auf kleinen Geräten darstellen zu können. Hierdurch wird auch das Gefühl einer klassischen App auf Handy und Tablet gegeben. *Bootstrap* bietet auf seiner Internetseite einfach Beispiele an. Diese wurden verwendet, um eine erste Grundstruktur des Designs der Applikation zu bekommen.

In der *index.html* werden die benötigten CSS Dateien von AngularJS und Bootstrap geladen. Diese Spezifizieren das Aussehen der geläufigsten HTML Elemente. Zusätzlich werden die benötigten JavaScript Dateien der Angular App und der verwendeten Module eingebunden. Anschließend wird der Grundaufbau der Applikation spezifiziert. Diese besteht aus einer Navigationsleiste mit Links zu den verschiedenen Seiten der Anwendung und einem Container in welchem die sogenannte *ng-view* verankert ist. In dieser View wird Angular alle aufgerufenen Seiten anzeigen.

Im *body* Element wird das Wurzelelement der Angular App *ng-app="patientManagement"* angegeben. Dieses Angular Modul wird in der *app.js* mittels *var app = angular.module("patientManagement", [<Abhängigkeiten>])* erzeugt. Anschließend wird der *config* Routine dieses Moduls eine Funkti-

on hinzugefügt, in welcher dem *routerProvider* die einzelnen Route der Seite mittels *routeProvider.when("/<URL>", <Einstellungen>)* bekannt gemacht werden. Mit Hilfe des Einstellungsobjekts wird die zur Seite gehörende HTML Datei und die JavaScript Datei welche den Controller beinhaltet spezifiziert. Hierdurch ist es möglich der **View** einen **Controller** anzuhängen welcher in der angegebenen Datei mittels *app.controller(«Controller Name», function («Abhängigkeiten»)«Code»)* angelegt wird. Dieser Code wird vor dem Anzeigen der Seite aufgerufen.

Bildchen dazu malen,Node < express+Mongo < Angular < Bootstrap

Durch diese Trennung zwischen Model und View in Verbindung mit der Anbindung an die API des Servers entsteht eine Klassische Model-View-Controller Architektur. Diese trennt das Aussehen, die View, von deren Logik, dem Controller. Hierdurch ist es möglich Seiten wie in Abbildung [4.5, 4.4, 4.6] gezeigt, darzustellen.

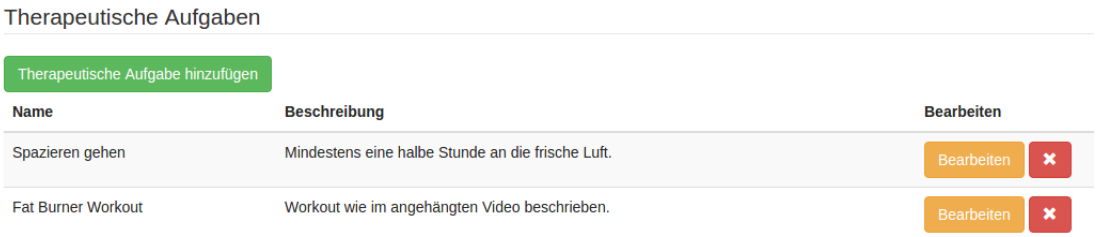

Abbildung 4.4: Übersicht über die Therapeutischen Aufgaben

Durch den Controller werden im Hintergrund mittels einer HTTP Anfrage die benötigten Daten zu den Patienten etc. über die Web-API vom Server abgerufen. Diese werden dann gegebenenfalls noch verarbeitet und im sogenannten Scope abgelegt, welcher in den oben genannten Abhängigkeiten in den Controller eingebunden sein muss. Der Scope ist die Verbindung zur View und dient dieser als Datengrundlage.

Innerhalb der View kann dann iterativ auf die Daten zugegriffen werden um diese anzuzeigen oder Formulare und andere Eingabemethoden zum verändern der Daten anbieten. Mittels Angular können die Daten anhand einer Art Vorschleife in der gewünschten Form angezeigt werden. Wodurch die Daten nicht vorformatiert werden müssen. Buttons können mit im Scope abgelegten Funktionen verknüpft werden, womit auf Nutzerinteraktionen reagiert wird.

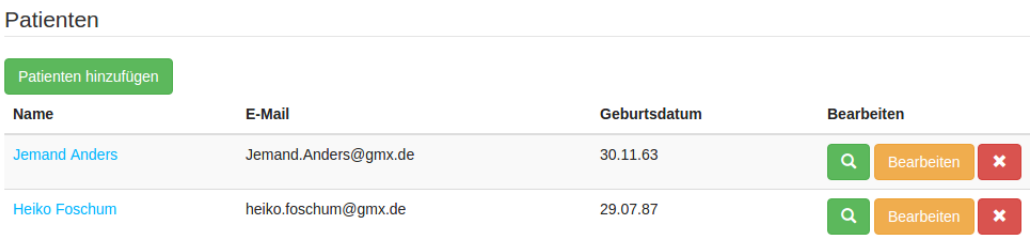

Abbildung 4.5: Seite der Patientenübersicht

Übersicht der Patienten

Die in Abbildung 4.5 gezeigte Ansicht bietet eine Übersicht der auf dem Server gespeicherten Patienten. Hierzu wird mittels der *get*-Methode des *http* Moduls eine *asynchroner HTTP request* an der Server gesendet. Im falle einer positiven Antwort werden die so erhaltenen Daten im Scope abgelegt. Von nun an können diese in der View durch den *{{«Variable»}}* Operator verwendet werden.

Mit Hilfe von *ng-repeat="patient in patients"* werden diese Daten in einer Tabellenstruktur angezeigt. Durch *ng-repeat* kann für jedes Patientenobjekt im *patients* Array die nachstehenden HTTP Elemente zur Anzeige gebracht werden. Mit Hilfe von *<a href=¨/patientDetails/{{patient._id}} >¨* kann dann innerhalb des *ng-repeat* beinhaltenden Elements Beispielsweise ein Link erzeugt werden, welcher die ID des jeweiligen Patienten beinhaltet. Hierdurch wird der Nutzer zur Detailansicht des Patienten weitergeleitet. Die angehängte ID kann dort dann anhand des *routeParams* Moduls ausgelesen werden.

Um das löschen eines Patienten zu implementieren wurde dem Löschbutton eine Funktion mittels *scope.deletePatient = function(patient)* im Scope hinterlegt. Diese verwendet die *delete* Methode des *http* Moduls um eine HTTP *delete* Anfrage an den Server zu senden. Bei einer erfolgreichen Anfrage wird das zu löschende Patienten Objekt noch lokal aus den bestehenden Objekten entfernt.

Detailansicht zu einem Patienten

In dieser Ansicht können dem Patienten die Aufgaben und Fragebögen zugewiesen werden. Wie in Abbildung 4.6 zu sehen, wird hierzu neben der Aufgabe ein Fragebogen und eine Zeitspanne sowie die Wochentage zu denen die Aufgabe zu erledigen ist ausgewählt. Hierzu werden im Controller alle existierenden Therapeutischen Aufgaben und Fragebögen vom vom Server geladen. Sowie das Objekt des durch die ID spezifizierten Patienten.

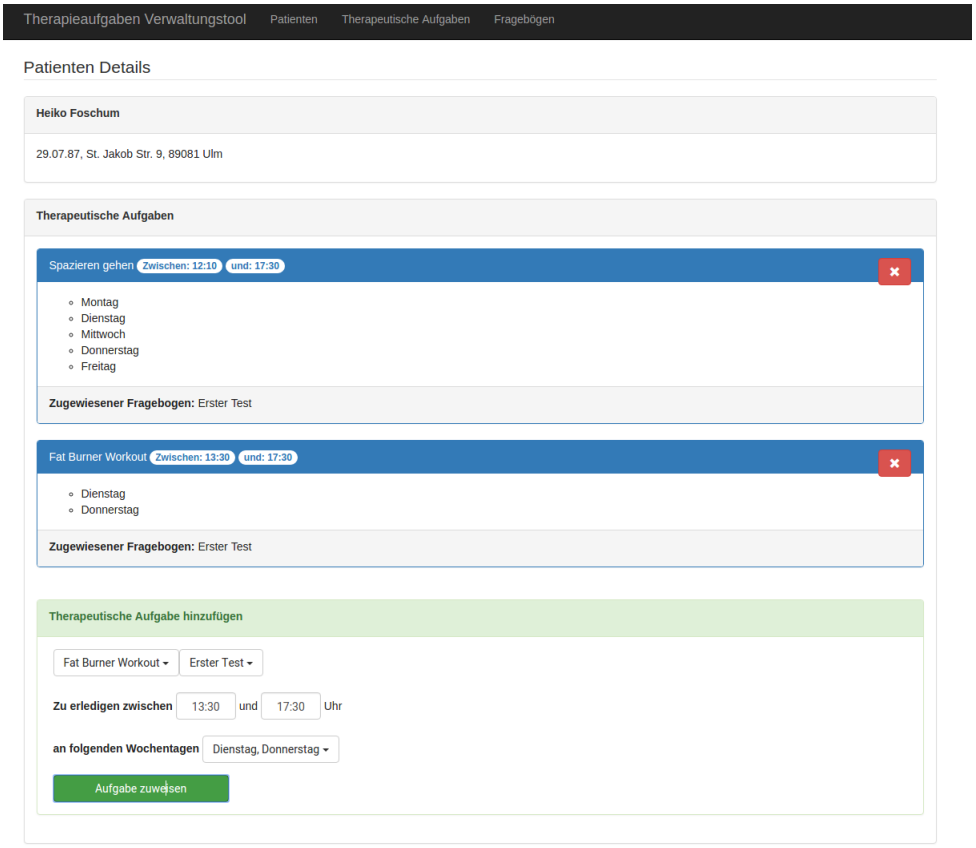

Abbildung 4.6: Ansicht der Patientenübersicht im Web Frontend

Da dieses Objekt nur die IDs der zugewiesenen Therapeutischen Aufgaben beinhaltet wurde dem Scope eine Funktion *getTaskById(«id»)* hinzugefügt welche den Name einer Aufgabe zur gegebenen ID zurück liefert. Diese wird in der View mittels *getTaskById(Task.PatternId).name* verwendet, um in der Liste der zugewiesenen Aufgabe den Name und nicht die ID einer Aufgabe anzuzeigen.

Um neue Aufgaben hinzufügen zu können, wird im Scope eine Aufgaben Objekt erzeugt, in welches die Eingaben des Nutzers gespeichert werden. Durch *ng-model* kann ein Eingabefeld mit einer Variable des Scopes verbunden werden. Durch das *2-way-binding* von Angular haben Veränderungen im Eingabefeld direkten Einfluss auf die Variable und umgekehrt. Somit kann mittels eines Buttons der vom Typ*dropdown-toggle* ist die Auswahlfelder für die Aufgaben Fragebögen und Wochentage implementiert werden. Um eine angenehme Eingabemethode für die Zeitpunkte zu bieten wurde das *bs-timepicker* Modul hinzugefügt und eingebunden.

Der **Speichern**-Button ruft die im Scope hinterlegte Funktion *addTherapyTask()* auf. Diese hängt im Falle das alle benötigten Felder gefüllt sind, das oben erwähnte Aufgabenobjekt im Patientenobjekt an die bestehenden Therapeutischen Aufgaben an. Anschließend wird das veränderte lokale Patientenobjekt zur Speicherung an den Server geschickt. Zusätzlich werden auf dieser Seite die vom Patienten bereits beantworteten Fragebögen angezeigt(Abbildung 4.7). Auf deren Grundlage der Therapeut die Therapie verwalten und verbessern kann. Um die Übersichtlich-

keit zu verbessern wurde der View ein weiteres *panel* hinzugefügt in welchem diese Angezeigt werden. Diesen *panels* kann durch hinzufügen unterschiedlicher Klassen ein unterschiedliches aussehen gegeben werden. Sie bestehen aus einem *panel-heading* und einem *panel-body* und werden mittels *ng-repeat* iterativ angezeigt.

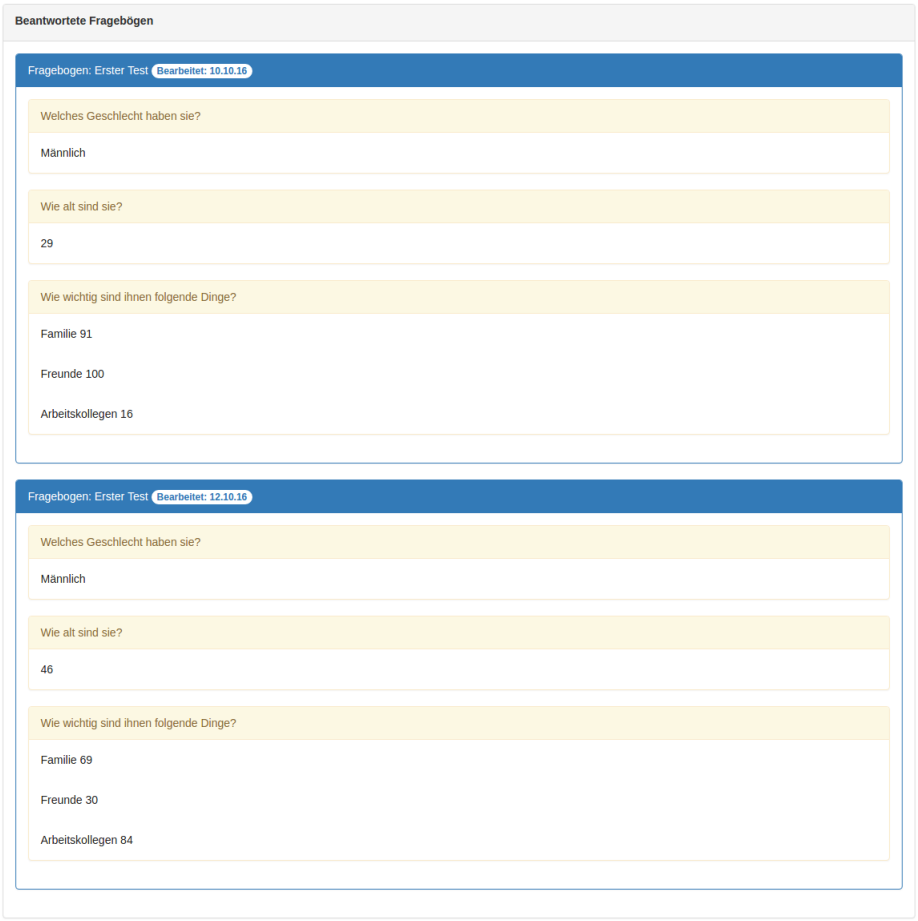

Abbildung 4.7: Seite der Patientenübersicht

Verwaltung der Therapeutischen Aufgaben

Anhand der Ansicht in Abbildung 4.4 hat der Therapeut eine Übersicht über alle bereits angelegten Therapeutischen Aufgaben. Da die Implementierung der Übersichtsseite im der für die Patienten wird auf diese nicht weiter eingegangen. Durch das Hinzufügen einer neuen Aufgabe kann diese, wie in 4.8 gezeigt, durch Name, Beschreibung und beliebigen Materialien spezifiziert werden. Angehängte Materialien werden mittels URLs angegeben. Anhand derer können eine Vielzahl von Aufgaben gestellt oder unterstützt werden. Beispielsweise durch Videos, GPS Koordinaten, anderen Apps, Bilderserie, oder Texten.

Um diese Daten entgegen nehmen zu können wurde eine Seite erzeugt, welche sowohl zum anlegen so wie zum editieren einer Aufgabe dient. Einziger unterschied der beiden ist, dass beim editieren einer Aufgabe deren ID beim Seitenaufruf übergeben wird. Ist dies der Fall wird

das Objekt der Aufgabe vom Server geladen und mit dessen Daten das Formular gefüllt, das zum ändern der Daten implementiert wurde. Wird keine ID beim Aufruf der Seite mit gegeben, wird ein neues Aufgabenobjekt erzeugt.

Dieses Formular zur Veränderung der Daten wird wie in HTML5 gewohnt mittels eines *form* Elements eingeleitet. Durch Angular ist es mit Hilfe von *ng-submit=ßaveTherapyTask()"* möglich die angegebene Funktion beim Absenden des Formulars aufzurufen. Diese Funktion entscheidet dann anhand daran, ob eine ID beim Aufruf der Seite mit gegeben wurde ob eine neue Aufgabe mittels eines HTTP *posts* angelegt oder eine existierende durch einen HTTP *put* verändert werden muss.

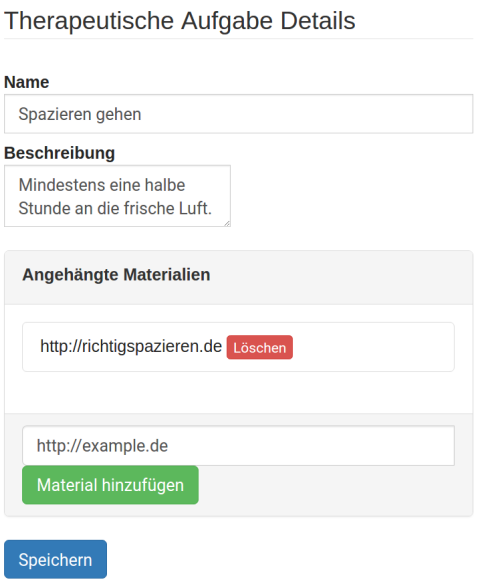

Abbildung 4.8: Formular zum Editieren oder Erstellen von Therapeutischen Aufgaben

Fragebogenverwaltung

Auch für die Verwaltung der Fragebögen wurde eine eigenen Seite eingerichtet. Diese bietet das löschen und editieren der Fragebögen. Es gibt einige gute Node Module, welche eine fertige Oberfläche zum editieren von Fragebögen zu Verfügung stellen.

Jedoch haben alle das Problem , dass es mit diesen nicht möglich ist, Fragen abhängig von der Antwort des Patienten zu machen. Um diese Problematik zu umgehen, wurde ein Tool in die Anwendung eingebaut, dass eigentlich zum modellieren von Business Prozessen verwendet wird und auf der **B**usiness **P**rocess **M**odel and **N**otation beruht. Dieses Tool von BPMN.io (Camunda BPM) kann völlig auf die eigenen Bedürfnisse zugeschnitten werden. Es is möglich alle Elemente zu verändern. Somit wäre die Möglichkeit gegeben, im Zuge einer finalen Implementierung der Anwendung dem Fragebogeneditor das aussehen des BPMN Editors zu nehmen. Im Umfang der konzeptionellen Implementierung wurde hierauf jedoch verzichtet.

Type	Beschreibung
single	Der Patient kann **genau eine** Antwort auswählen
multi	Der Patient kann **mehrere** Antworten auswählen
text	Zu jeder Frage kann ein **freier Text** geschrieben werden
rating	Mittels Slidern können **Bewertungen** abgegeben werden

Tabelle 4.1: Übersicht über die Typen der Fragen

Die Seite zum erstellen und editieren besteht somit aus dem BPMN Editor und einer Funktion zum Speichern des modellierten Fragebogens. Um den Editor anzeigen zu können wurde der View ein * * Element mit der ID canvas hinzugefügt. Im mit der View verbundenen Controller wird wiederum geschaut, ob die ID eines Fragebogens beim Aufruf der Seite mit gegeben wurde. Ist dies der Fall wird diese Vom Server geladen.

Anschließend wird eine Instanz des eingebundenen Moduls *BPMNModeler* erzeugt. Diesem wird der bestehende Fragebogen in XML-Form mitgegeben sowie die ID des * * Elements als *container* spezifiziert. Innerhalb dieses Elements wird fortan der Editor angezeigt.

Durch das hinzufügen eines Eigenschaftsfensters zum Editor, können den einzelnen Frage vier verschiedene Typen zugewiesen werden. Diese Typen sind **single, multi, text und rating** und werden in Tabelle 4.1 näher beschrieben. Dieser wurde beim erzeugen des Editor Objekts als zusätzliche Option definiert und mit einem weiteren * * Elements mit der ID *properties* verknüpft.

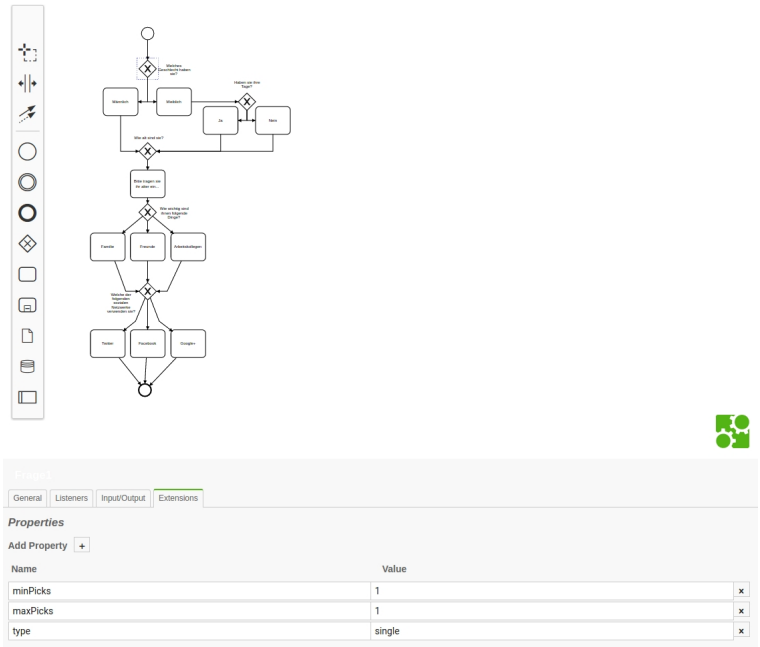

Abbildung 4.9: Ansicht des BPMN Bearbeitungstool

Mittels des BPMN Editors ist es dann möglich Fragebögen zu gestalten. Fragen werden mittels einem sogenannten Gateway spezifiziert. In dessen Beschreibung kann der Text jener Frage eingetragen werden. Im Eigenschaftsfenster des Gateways kann dann unter Extensions eine neue

property mit Namen **type** hinzugefügt werden. Diese kann die oben erwähnten Typen von Fragen annehmen, welche dem Patienten unterschiedliche Arten der Antwortmöglichkeiten geben. Die einzelnen Antworttexte werden durch sogenannte Tasks spezifiziert und mittels Pfeilen mit dieser verbunden. Die Antworten können dann wiederum mit einer weiteren Frage verbunden werden. Somit kann modelliert werden bei welcher Antwort welche Frage folgt.

4.3 Patienten App

Die mobile Cross-Plattform-App wurde auf Grundlage des Ionic Startprojekts **tabs** implementiert. Dieses Projekt bietet das Grundgerüst einer App mit 3 Tabs welche auf jedem Gerät Betriebssystem abhängig angezeigt wird. Diese Web-App [siehe. Kapitel 2.7.1] basiert wie auch der Therapeuten Client auf AngularJS. Im Gegensatz zu diesem wird aber nicht Bootstrap sonder die Ionic eigene Entwicklung für die Oberfläche verwendet. Auf dieser Grundlage entstand die in Abbildung 4.10 gezeigte Anwendung. Diese bietet einen Tab in welchem der Patient eine Übersicht über die ihm zugeteilten Aufgaben hat mit der Möglichkeit diese im Detail zu betrachten. Sowie einen Tab in dem er sein Profil einsehen kann und einen für die Einstellungen. Jeder dieser Tabs wird durch eine HTML Seite und dem zugehörigen Controller spezifiziert.

Die HTML Seiten werden im Ordner *templates* abgelegt. Die JavaScript Dateien sind im Ordner *js* zu finden. Die App besteht aus 3 JavaScript Dateien. Die services.js in welcher wiederverwendbarer Code in Form von sogenannten *factorys* untergebracht werden soll. Diese haben einen Anwendungsweiten eindeutigen Namen und können mit dessen Hilfe in jeden Controller als Abhängigkeit eingebunden und verwendet werden.

Um die Server-API von der Anwendung zu abstrahieren, wurde für jede der vier APIs eine eigenen *factory* angelegt. Diese bieten jeweils eine Funktion zum Abrufen aller vorhanden Einträge sowie das Abrufen eines bestimmten Objekts Anhand seiner ID. Mit Hilfe der Funktionen *remove, create* und *update* können Einträge auf dem Server angelegt oder unter Verwendung der ID verändert und gelöscht werden. Hierbei wird mit den selben Methoden welche auch schon beim Therapeuten Client verwendet wurden auf den API-Server zugegriffen. Es wurde nur die externe IP Adresse des Servers anstelle der *localhost* Variable verwendet. Um diese IP einstellbar zu machen wurde für diese ein Eingabefeld im Einstellungstab implementiert.

Um Daten auf dem Endgerät dauerhaft speichern zu können, wurde der Anwendung das Modul *ngStorage* hinzugefügt. Diese bietet zum einen den *sessionStorage* welcher die Daten bis zum schießen der Anwendung behält und den *localStorage* welcher als persistenter Datenspeicher verwendet wird.

In der app.js wird das Angular Modul **patientApp** mit seinen Abhängigkeiten spezifiziert. In einer Konfigurationsroutine von diesem Modul wird zu jeder Seite der Applikation ein *state* im *stateProvider* angelegt. In jedem *state* wird spezifiziert unter welcher URL dieser aufgerufen wird, sowie die zugehörige HTML Datei und der Name des Controllers. Hierbei kann ein *state* einem anderen untergeordnet werden. Dies ist nötig, um Folgeseiten von z.B. Tabs zu definieren.

Um Daten beim Wechsel zu einer anderen Seite übergeben zu können, kann mittels */URL/:«Variablenname»* in einem *state* definiert werden, dass dieser *state* für URLs vom Typ */URL/«irgend ein String»* verwendet wird. Mit Hilfe des *routeParams* Moduls kann dann auf der Aufgerufenen Seite auf die mitgegebenen Daten zugegriffen werden.

Beim ersten Start der Applikation wird der Patient auf den Einstellung Tab geleitet. Hier kann er mittels einer Schaltfläche ein Popup öffnen in welchem er seinen Namen angibt. Wird ein Profil mit diesem Namen auf dem Server gefunden wird dieses Lokal gespeichert. Und die Schaltfläche zum auswählen eines Profils deaktiviert. Vor dem produktiven verwenden der App sollte hier ein weiterer Sicherheitsmechanismus eingebaut werden um zu verhindern, dass unbefugte auf das Profil eines Patienten zu greifen können.

4.3.1 Anzeige der Aufgaben

Um dem Patienten ein Übersicht über seine zugewiesenen Aufgaben geben zu können wurde die in Abbildung 4.10 gezeigte Seite eingerichtet. Diese beinhaltet neben der Übersicht über die Aufgaben Buttons zum erstellen einer Betriebssystem Erinnerungen.

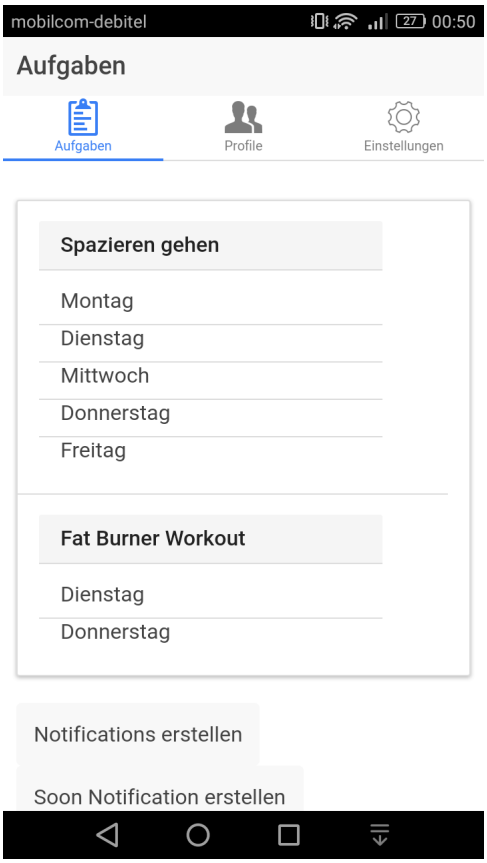

Abbildung 4.10: Ansicht der Aufgaben auf einem Android Smartphone

Hierzu wurde eine Controller Namens *TaskListCtrl* erstellt und zusammen mit der *tab-taskList.html* in den Router eingetragen. Im Controller werden dann die Daten zum Patienten und den existierenden Aufgaben Vorlagen über die oben erwähnten Servicesvom Server geladen, sofern diese nicht schon lokal vorhanden sind.

Factory!??!

In der View wird dann wieder mittels *ng-repeat* über die im Patienten Objekt vorhandenen Aufgaben iteriert. Jeder so erzeugt *list-card-devider* Item beinhaltet das HTML-Tag *ui-sref="tab.task-detail({taskId : $index})"*. Hierdurch werden die einzelnen Listenelemente zu Link, welche zur Detailansicht der jeweiligen Aufgabe führen. *tab.task-detail* ist hierbei der im Router spezifizierte Name der Detailseite. Mittels *({taskId : $index})* wird dem Controller der Index der ausgewählten Aufgabe übergeben. *$index* ist eine von *ng-repeat* bereitgestellte Variable.

Da im Patienten Objekt nur die Id der zugewiesenen Aufgaben gespeichert sind wurde die in Algorithmus 4.1 gezeigte Funktion implementiert. Diese iteriert über die vorhandenen Aufgaben und gibt das passenden Objekt zur Id zurück. Diese Funktion wird wiederum in der View mittels *{{getTaskById(task.PatternId)[0].name}}* verwendet.

Algorithm 4.1: Funktion welche den Namen einer Aufgabe abhängig von der Id zurückliefert

```
1 $scope.getTaskById = function (TaskId) {
2         return $.grep(therapyTaskApi.all(false), function (e, x) {
3         return e._id == TaskId;
4 }); };
```

Um die Betriebssystemerinnerungen zu den Aufgaben zu erstellen wurden Funktionen erstellt, welche mit den Buttons in der View verbunden sind.

Erstellen von Betriebssystem Erinnerungen

Da im Patienten Objekt nur die Wochentage gespeichert sind zu denen die Aufgabe erledigt werden soll, ist noch nicht spezifiziert welches Datum diese Tage haben. Dies wird jedoch benötigt um die Betriebssystem Erinnerungen zu erstellen.

Innerhalb der Oben erwähnten Funktion zur Erstellung der Erinnerungen wird zuerst eine *Date*-Objekt durch den Standard Constructor erstellt. Das so erzeugt Objekt beinhaltet die Informationen zur momentanen Zeit.

Im *Date-Objekt* werden die Wochentage als Zahlen zwischen 0 und 6 gespeichert. Wobei die 0 den Sonntag repräsentiert. Um die Daten der Aufgabenzeitpunkte zu berechnen, wird die rechnerische Distanz zwischen dem Wochentag des Heute-Objekts und den TODO-Wochentagen berechnet. Diese Distanz wird dann auf das Datum des Heute-Objekts addiert. Durch die Verwendung der *setDate* Methode kümmert sich das Objekt selbst darum, wenn gegebenenfalls ein Monatswechsel oder ähnliches besteht.

Anschließend werden mittels des *Cordova Local-Notification Plugins* die einzelnen Erinnerungen erstellt. Das Erinnerungsobjekt muss, wie in Algorithmus 4.2 Beispielhaft dargestellt, verschie-

dene Eigenschaften beinhalten. Eine *Id*, hier wird eine laufende Nummer verwendet. Unter *at* wird das Datum der Erinnerung angegeben. Als *title* wird der Name der Aufgabe verwendet.

Algorithm 4.2: Beispiel einer Funktion zum hinzufügen einer Betriebssystemerinnerung

```
$scope.addSoonNotification = function () {
        var soonDate = new Date();
        soonDate.setSeconds(soonDate.getSeconds() + 15);
        cordova.plugins.notification.local.schedule({
                id: 1987,
                title: "Titel der Erinnerung",
                text: "Klick mich!",
                at: soonDate,
                data: {
                QuestionnaireId: "57e795d2db97f7bd0ee6564a"
        }
        });
};
```

Zusätzlich kann der Erinnerung ein *data* Objekt angehängt werden. In diesem können beliebige Daten gespeichert werden. Dies wird verwendet um die Ids der Aufgabenvorlage und des Fragebogens zu speichern.

Mittels *$rootScope.$on('$cordovaLocalNotification:click',function (event, notification, state) {}* kann auf den Klick des Nutzers auf die Erinnerung reagiert werden. Es wird dann je nachdem ob es der Beginn oder das Ende des gegebenen Zeitintervalls ist, entweder die Aufgabenbeschreibung oder der zugehörige Fragebogen angezeigt.

Hierzu wird mittels des *StateProviders* zu eine der Seiten gesprungen und die Id des aufzurufenden Objekts übergeben.

Aufgabendetails einsehen und bearbeiten

Um die Details zu einer Aufgabe einsehen zu können, wurde eine View (*task-detail*) mit dem zugehörigen Controller *TaskDetailCtrl* erstellt. Mittels der Route *tab.task-detail* kann auf diese zugegriffen werden. Diese Route übergibt die in der URL angefügten Daten unter der Variable *taskId*. Diese Id gibt den Index der Aufgabe im Array des Patienten Objekts an. Hierdurch kann beim Aufruf der Seite spezifiziert werden, welche Aufgabe zur Anzeige gebracht wird.

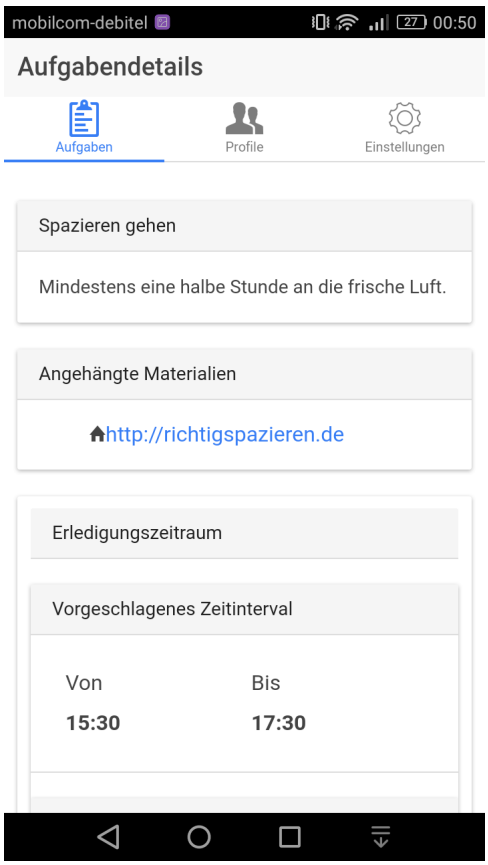

Abbildung 4.11: Name, Erläuterung und angehängte Materialien einer Therapeutischen Aufgabe

Der Controller läd das Patientenobjekt und die angegebene Aufgabe. Über die gespeicherte Id der Aufgabevorlage wird diese anschließend vom Server abgerufen. Hierdurch ist sichergestellt, dass der Nutzer die Aktuelle Version der Aufgabe angezeigt bekommt. Somit ist gewährleistet, dass der Therapeut jederzeit Änderungen an den Aufgaben vornehmen kann und diese auf dem Endgerät des Patienten immer aktuell sind.

Diese Informationen werden dann wir in Abbildung 4.11 gezeigt, angezeigt. Neben dem Titel der Aufgabe wird die Beschreibung und die angehängten Materialien angezeigt. Zum Anzeigen der hinter der URL spezifizierten Informationen wurde eine In-App-Browser in die Anwendung eingebunden.

näher auf In-App-Browser eingehen

Um dem Patienten die Möglichkeit zu bieten, das vom Therapeuten vorgeschlagene Zeitintervall zu verändern, wurde die in Abbildung 4.12 gezeigten Schaltflächen in diese Seite eingepflegt.

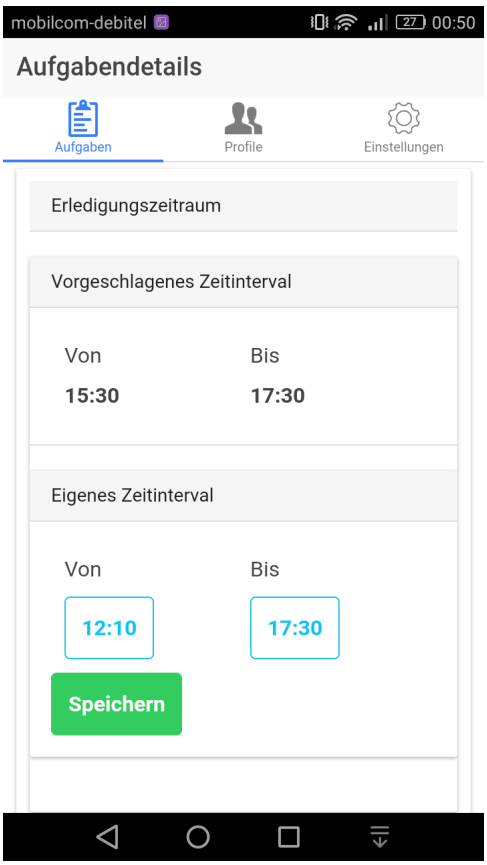

Abbildung 4.12: Übersicht über die Fragebögen

Da Ionic von Haus aus keinen Funktion zum wählen einer Zeit mitliefert wurde hier das *ionicTimePicker Plugin* zur Anwendung hinzugefügt. Dieses wird in der *config*-Phase der App initialisiert. Hier wird der Zeitformat auf 24 Stunden gestellt, sowie die Schrittweite und die Beschriftung der Knöpfe spezifiziert. Durch einen Klick auf die in Abbildung 4.12 zu sehende Schaltfläche wird mittels *ionicTimePicker.openTimePicker()* ein Popup zur Anzeige gebracht in dem der Patient die Möglichkeit hat eine Uhrzeit ohne Datum etc. auszuwählen.

Nach dem Bestätigen der Eingabe werden die neuen Zeitinformationen erst Lokal gespeichert. Durch einen Klick auf die Speichern Schaltfläche kann das veränderte Patientenobjekt dann zum Dauerhaften Speichern an den Server gesendet werden.

4.3.2 Anzeige der Fragebögen

Um dem Patienten die Möglichkeit zu geben, die gestellten Fragebögen zu beantworten wurde eine Seite entwickelt, die ihr Aussehen nach dem Typ der Aufgabe angleicht. In der View *questionnaire-show* sind die HTML-Elemente für alle vier (in Tabelle 4.1 beschriebenen) Fragetypen vorhanden. Diese sind jedoch durch die *ng-hide* Direktive, die durch AngularJS angeboten wird, ausgeblendet. Diese Direktiven sind mit im Scope abgelegten Bool Variablen verbunden. Hierdurch können die einzelnen Elemente durch im Controller abgelegten Code angezeigt oder

ausgeblendet werden. Ist die zugehörige Variable **wahr** so ist das HTML-Element ausgeblendet. Somit variiert das Aussehen der Seite je nach Fragetyp.

Frage vom Typ single In Abbildung 4.13 ist ein Beispiel zu einer Frage des Typs **single** auf einem Android Smartphone zu sehen. Hier kann der Nutzer nur eine der gegebenen Antwortmöglichkeiten auswählen. Die nächste Frage ist, je nachdem was der Therapeut spezifiziert hat, abhängig von der gegebenen Antwort.

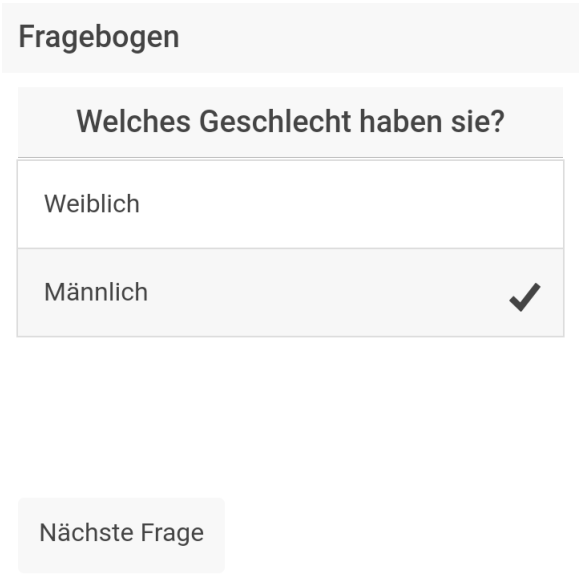

Abbildung 4.13: Fragebogenseite beim Fragetyp single

Bilder größer und newpage ???

Frage vom Typ multi In der Abbildung 4.14 ist als Beispiel die Seite gezeigt, wenn der Aufgabentyp **multi** ist. Dieser Typ erlaubt es dem Endanwender mehrere der zur Verfügung stehenden Antworten auszuwählen.

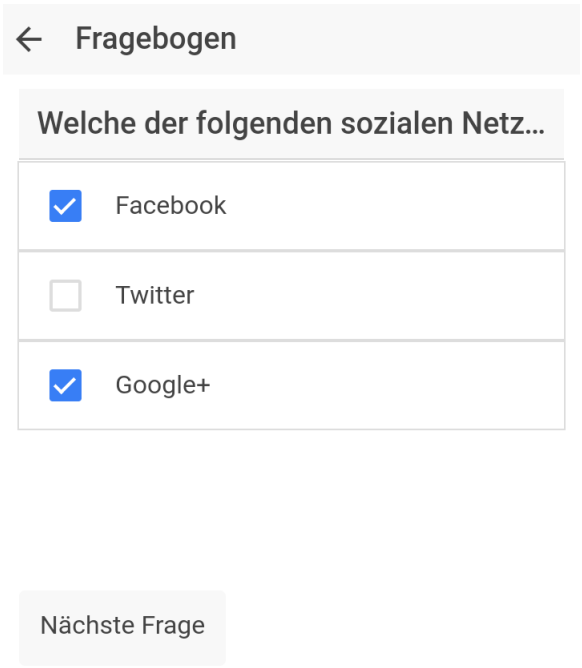

Abbildung 4.14: Fragebogenseite beim Fragetyp multi

Frage vom Typ text Der in Abbildung 4.15 gezeigt Fragentyp gibt dem Anwender due Möglichkeit die Antworten als Freitext zu beantworten. Da das beurteilen von solch einer Antwort den Rahmen dieser Arbeit gesprengt hätte, kann die nächste Frage nicht von der Antwort des Nutzers abhängig gemacht werden.

Abbildung 4.15: Fragebogenseite beim Fragetyp text

Frage vom Typ rating In Abbildung 4.16 ist das Beispiel einer Frage des Typs **rating** abgebildet. Diese können verwendet werden, wenn der Patient etwas bewerten soll. Die Auswahl resultiert in Werten zwischen 0 und 100.

Abbildung 4.16: Fragebogenseite beim Fragetyp rating

Die Route zu dieser Seite stellt in dieser Anwendung eine Besonderheit dar, da sie es zulässt mehrere Variablen in der URL zu übergeben. Neben der Id des Fragebogens kann zusätzlich die Id der anzuzeigenden Frage in der Variable *questionId* übergeben werden.

In dem der View zugehörigen Controller *QuestionnaireShowCtrl* wird das Fragebogen Objekt anhand der übergebenen Id über den oben erwähnten Service vom Server geladen. Anschließend wird der im XML-Format gespeicherte Fragebogen mit Hilfe des *xml2json-Parsers* [3] in ein JavaScript Objekt geparst. Dies erleichtert den Zugriff auf die einzelnen Objekte erheblich.

Um den Umgang mit einem Fragebogen weiter zu vereinfachen wurde eine weiterer Service Namens *questionnaireHelper* implementiert. Dieser bietet verschiedenen Methoden:

getStart
: Welche das Startobjekt zurückliefert

getNextQuestionObj
: Diese Methode erwartet ein Antwort Objekt und liefert die auf diese folgende Frage zurück

getQuestionType
: Liefert den Typ einer Frage zurück

getPossibleAnswers
: Gibt die möglichen Antworten zu einer Frage als Array zurück

getQuestionObjById

Gibt das Objekt einer Frage anhand derer Id zurück

getAnswerObjById

Gibt das Objekt einer Antwort anhand derer Id zurück

Wird nun diese Seite aufgerufen, wird zuerst mittels *if (typeof $stateParams.questionId == "undefined")* geschaut, ob unter der Variable *questionId* eine bestimmte Frage spezifiziert wurde. Ist dies der Fall wird dieses mit Hilfe der Methode *getQuestionObjById* aus dem Fragebogen Objekt extrahiert und als aktuelle Frage verwendet. Im Fall, dass keine Frage spezifiziert ist, wird das Start Objekt gesucht und von diesem aus die nächste Frage als aktuelle Frage gewählt.

Anschließend werden anhand der Scope Variablen zuerst alle Fragen HTML-Elemente ausgeblendet. Mittels der Methode *getQuestionType* wird dann der Typ der Frage ermittelt und das jeweilige HTML-Element wieder aktiviert.

Um die Antwort des Nutzers entgegen nehmen zu können wird ein zum Fragentyp passendens Antwortsobjekt erstellt. Dies beinhaltet in jedem Fall den Typ der Frage, um ein späteres auswerten und anzeigen einfach zu machen.

Ist die Frage vom Typ **single** wird in diesem Objekt die Frage und die gegebene Antwort gespeichert. Hier wurde auf eine Verbindung zum Fragebogen verzichtet um zu verhinder, dass sich dieser verändert und die gegebene Antwort dann nicht mehr zur eigentlich gestellten Frage passt.

Bei einer Frage vom Typ **multi** wird für jede Antwortmöglichkeit ein eigenes Antwortenobjekt erzeugt. Jedes dieser Objekt beinhaltet die Variable **checked**, in welcher festgehalten wird, ob die jeweilige Antwort ausgewählt wurde.

Im Fall das die Frage vom Typ **rating** ist, wird wiederum für jede Frage ein eigenenes Antwortobjekt erzeugt. Dieses beinhaltet neben den schon erwähnten Variablen eine weitere Namens *value*. In welchen der Wert der angezeigten Schieberegler gespeichert wird.

Beim Fragetyp **text** wird auch für jede gestellte Frage ein eigenes Antwortenobjekt erzeugt welches eine Variable *answerText* beinhaltet mit welcher das Texteingabefeld verbunden ist.

Am Ende der Seite ist eine Schaltfläche eingebaut, welche mit der Funktion *goToNextSite* verbunden ist. Diese Funktion schaut nun bei Fragen des Typs **single** und **multi** welche der Antworten ausgewählt wurde und ermittelt mittels der Funktion *getNextQuestionObj* welche Frage als nächstes kommt. Wenn es noch eine weitere Frage gibt wird mittels *$state.go("questionnaire-show",{questionnaireId: qId, questionId: nextQuestion.Id})* die selbe Seite noch einmal aufgerufen, jedoch mit einer anderen questionId wie zuvor. Dies wird wiederholt, bis es keine weitere ausgehende Frage zur Antwort mehr gibt und somit das Ende des Fragebogens erreicht ist.

5 Anwendungsfälle -> wird wohl Anforderungsabgleich

Hier sollen die die verschiedenen Anwendungsfälle der Anwendung gezeigt werden. Welche Arten von Aufgaben möglich sind / wären. Die Kontrollmechanismen

5.1 Kontrollmechanismen

Welche Arten von Kontrollmechanismen schon vorhanden sind

5.1.1 Weitere denkbare Kontrollen

Ausblick auf mögliche andere

5.2 Feedbackbögen und Auswertung

6 Zusammenfassung und Ausblick

6.1 Anforderungsabgleich

6.2 Zusammenfassung

6.3 Ausblick

Gesammelte Daten auswerten. Patientenverwaltung ausdehnen auf Patientenverwaltungstool. Kontrollmechanismen. Hochladen von Materialien. Andere Materialien...

— Hierbei ist es für den Therapeuten jedoch schwierig, zu beurteilen wie effektiv die Übungen wirklich sind. Schon aus dem Grund, dass er nur das Feedback durch den Patienten hat. Oft werden die Übungen auch vergessen oder schlicht nicht gemacht. Was eine Qualitative Auswertung über den nutzen der Hausaufgaben unmöglich macht. Hierbei wäre es auch Denkbar, den Patienten spielerisch dazu zu bringen seine Hausaufgaben zu machen. Vorstellbar ist auch ein Erfolgssystem oder Anbindung an die Krankenkasse. Durch derartige Mechanismen könnte der Patient weiter Motiviert werden und die Behandlung dadurch verbessert. —

Literaturverzeichnis

[1] *Mono (Software)*. https://de.wikipedia.org/wiki/Mono_(Software), . – zuletzt gesehen: 18.10.2016

[2] *Node JS*. https://nodejs.org, . – zuletzt gesehen: 18.10.2016

[3] *xml2json Parser*. http://goessner.net/download/prj/jsonxml/xml2json.js, . – zuletzt gesehen: 18.10.2016

[4] LYDIA FEHM, Gabriele Fehm-Wolfsdorf: Hausaufgaben in der Psychotherapie"/ Technische Universität Dresden; Psychotherapeutische Praxis, Lübeck. 2001. – Forschungsbericht

[5] SYLVIA HELBIG, Lydia F.: Der Einsatz von Hausaufgaben in der Psychotherapie / Institut für Klinische Psychologie und Psychotherapie, TU Dresden. 2005. – Forschungsbericht

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Heiko Foschum Matrikelnummer: 856456

Erklärung

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

Außerdem erkenne ich die Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis an.

Ulm, den .....

Heiko Foschum