ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für Datenbanken und Informationssysteme

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:
Marc Schickler

2016

Fassung 23. November 2016

Satz: PDF-LaTeX 2ε

Inhaltsverzeichnis

1 Einleitung

Bei der Betreuung von Patienten durch einen Therapeuten, ob nun in der Psycho-, Physiotherapie oder sogar in der Diätassistenz, sind neben der eigentlich **Therapie** / **Behandlung** durch den Therapeuten - Übungen und Fragebögen für Zuhause fast immer Bestandteil der Behandlung. Wie in Abbildung 1.1 aufgezeigt werden die Aufgaben in der Sitzung besprochen und sollen dann vom Patienten selbstständig und ohne Betreuung bearbeitet werden.

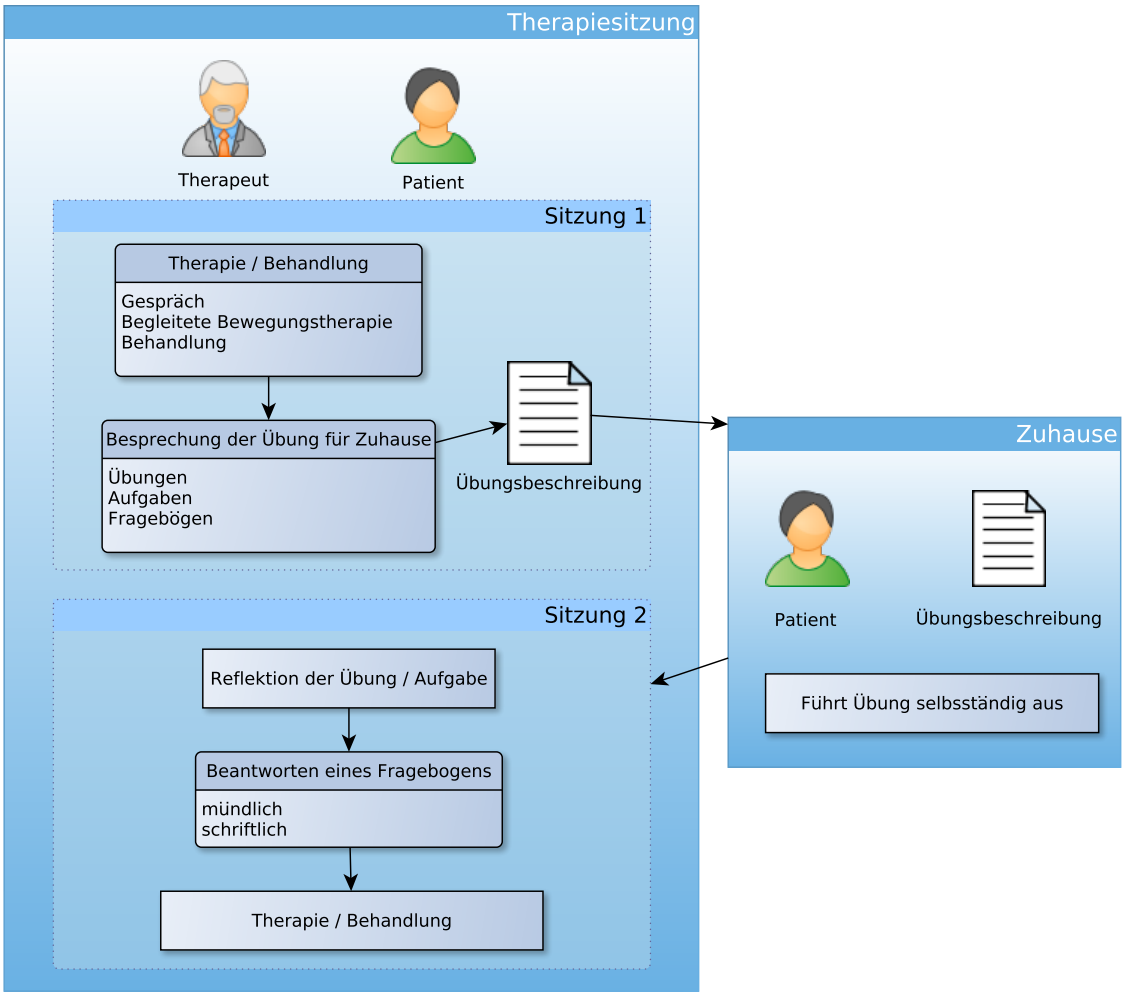

Abbildung 1.1: Grundlegender Ablauf einer Therapie

Diese Aufgaben sind wichtiger Bestandteil für den Behandlungserfolg [10]. 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 [9].

Bisher wurden dieses Aufgaben auf einen Blatt Papier festgehalten und der Patient war selbst dafür Verantwortlich diese in seinen Tagesablauf einzuplanen. Die einzige Vorgabe war es die Aufgabe(n) bis zum nächsten Termin x mal zu machen. Der Therapeut gab im besten Fall lediglich eine Empfehlung wann die Aufgabe zu erledigen ist, nicht jedoch einen genauen Plan an den sich der Patient halten konnte.

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 wodurch dieser die Möglichkeit hat den Behandlungserfolg zu steigern.

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 vor allem 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 zwar nur 11% der befragten an, die Aufgabe nicht erledigt zu haben. Nicht einmal die Hälfte jedoch gaben dagegen 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

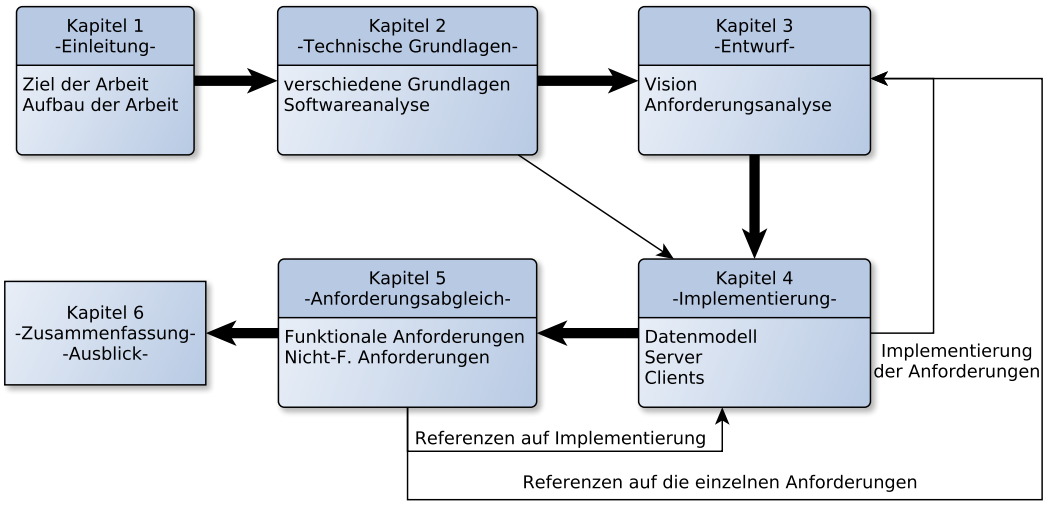

Abbildung 1.2: Aufbau der Arbeit

Nach dieser Einleitung in das Thema wird im nächsten Kapitel auf die **technischen Grundlagen** der Arbeit eingegangen. Dieses Kapitel gibt zum einen Grundlagenwissen welches zum besseren Verständnis der Arbeit, insbesondere Kapitel Implementierung, beitragen soll und zum anderen wird eine **Software/Technologie Analyse** durchgeführt welche sich vorwiegend mit dem Thema Cross-Plattform-Entwicklung für mobile Endgeräte befasst.

Im darauf folgenden Kapitel « **Entwurf** » wird zuerst die **Vision** der Arbeit vorgestellt, welche darüber Aufschluss gibt was genau von der entwickelten Plattform erwartet wurde und wie die Mechanismen von statten gehen sollten. Anschließend werden diese Erwartungen im Unterkapitel [3.2] **Anforderungsanalyse** in **Anforderungen** umgesetzt.

In Kapitel 4 wird im Detail erklärt auf welche Weise diese **Anforderungen** umgesetzt sind. Zuerst wird das zugrundeliegende **Datenmodell** beschrieben. Anschließend werden die **Implementierungen** von **Server** und den beiden **Clients** näher beschreiben.

Im Verlauf von Kapitel 5 werden die **Anforderungen** aus Kapitel 3.2 im einzelnen betrachtet und gezeigt ob diese im laufe der Arbeit erfüllt wurden. Hierbei werden Querverweise zu Kapitel 4 hergestellt um es dem Leser einfacher zu machen die zugrundeliegende **Implementierung** der **Anforderungen** zu finden.

Im letzten Kapitel wird eine Zusammenfassung sowie ein Ausblick über eine Mögliche Weiterführung der Arbeit gegeben.

2 Technischen Grundlagen

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

2.1 Daten beim Aufrufen oder wechseln einer Seite übergeben

Bei Webanwendungen wie die im Rahmen dieser Arbeit entstandenen, ist es immer wieder nötig, zum Beispiel beim Wechsel zu einer anderen Seite Daten an diese zu übergeben. Dies kann auf verschiedenen Wegen geschehen.

Bei kleiner Datenmengen wie Beispielsweise einer Id wird diese meinst in der URL übergeben. Diese wird dann an die eigentlich aufzurufende URL (z.B. http://eine.ip/index.html) nach einem **« ? »** angehängt. Die aufzurufende URL hat dann Beispielsweise die Form *http://eine.ip/index.html?patientId=eine1Id*. Wird die Seite auf diese Art aufgerufen kann die Variable patientId dann von dieser ausgelesen werden. Soll mehr wie eine Variable übergeben werden,können diese mit dem **« & »** Zeichen an einander angehängt werden.

Wenn größere Datenmengen zu übertragen sind, werden diese innerhalb des HTTP Objektes in dessen Rumpf übergeben. Dies wird in dieser Anwendung Beispielsweise verwendet, wenn nach der Anfrage an den Datenbank Server dessen Antwort mit dem Objekt im JSON Format Beispielsweise eines Patienten zurück gesendet wird.

2.2 REST-Service

Um eine gemeinsame Datenbasis zwischen den beiden Clients für Therapeut und Patient zu schaffen bietet der im Rahmen dieser Arbeit entwickelte Server einen REST-Service zum abrufen der in der Datenbank gespeicherten Daten an. Mit dessen Hilfe können die Daten zu den Patienten, Aufgaben und Fragebögen über das Internet mittels HTTP Anfragen auf jedem Gerät abgerufen und verändert werden. Das Programmierparadigma eines REST-Services soll dabei verschiedenen Prinzipien folgen.

Zum einen soll dieser Service **Zustandslos** sein. Bei jedem Aufruf müssen somit alle zum verstehen der Nachricht nötigen Daten übermittelt werden. Ein weiteres Prinzip ist die **Adressier-**

barkeit der einzelnen Ressourcen, welches in dieser Anwendung durch die einzigartige Id jedes Objekts realisiert wurde.

2.2.1 Die HTTP-Methoden

Um mit dem Service zu interagieren stehen verschiedene HTTP-Methoden zu Verfügung. Welche hier kurz erklärt werden.

HTTP-GET Mittels HTTP-GET wird dem Server signalisiert, dass Daten abgerufen werden wollen. Hierbei kann eine allgemeine Anfrage gestellt werden um alle vorhandenen Daten zu erhalten oder mittels der unter Adressierbarkeit angesprochen Id ein bestimmtes Objekt angefordert werden.

HTTP-POST Diese Methode wird verwendet um ein neues Objekt in der Datenbank anzulegen. Der Server antwortet hierbei mit dem Angelegten Objekt welches die vom Server erzeugte, einzigartige Id enthält.

HTTP-PUT Mit Hilfe der PUT-Methode kann ein auf dem Server vorhandenes Objekt verändert werden, hierbei muss die Id bekannt sein und übermittelt werden.

HTTP-DELETE Durch diese Methode kann ein Objekt anhand seiner Id auf dem Server gelöscht werden.

2.3 Sicherheit

Da es sich bei den ausgetauschten Daten um brisante Patientendaten handelt sollte das Thema Sicherheit sehr groß geschrieben werden. Da es jedoch den Rahmen dieser Arbeit überzogen hätte, wird es nur theoretisch betrachtet und wurde während der Implementierung außer Acht gelassen. Bevor die Plattform mit realen Daten verwendet wird muss in jedem Fall zuerst verhindert werden, dass unbefugte auf die Daten der Patienten zugreifen können. In diesem Abschnitt sollen Ideen und Anregungen gegeben werden wie dies bewerkstelligt werden kann. Da in einem REST-Service nicht von Haus aus Schutzmechanismen eingebaut sind liegt es in der Hand des Entwicklers die oben beschriebenen, unsicheren HTTP Methoden vor dem Zugriff durch unberechtigte zu schützen.

2.4 HTML 5

In dieser Sektion wird nicht auf HTML5 allgemein eingegangen, es werden ausschließlich die Elemente erläutert welche verwendet wurden und durch die momentan neuste Spezifikation der Hypertext Markup Language spezifiziert sind. Ganz allgemein ist zu sagen, dass es HTML5 erlaubt interaktivere Anwendungen zu gestalten. HTML ist dabei die Sprache die verwendet wird, um das Aussehen der Seite zu spezifizieren. In der entwickelten Anwendung wurde sie in der Webanwendung des Therapeuten und auch in der Mobilen App für den Patienten verwendet.

Das **nav** Element wurde in der Therapeuten Webanwendung verwendet um die Navigationsleiste zu spezifizieren. Hier sind nur die Links zu den einzelnen Hauptseiten definiert. Durch die in den meisten Browsern automatisch verwendete CSS Eingenschaft *display : block* wird diese als Box über die gesamte breite der Seite angezeigt [5]. Durch Twitter Bootstrap bekommt die Navigationsleiste ihr aussehen und weitere Funktionalität.

Bei den verschiedenen **input** Elementen in beiden Clients konnte mittels HTML5 spezifiziert werden welche Art von Eingabe in dem jeweiligen Textfeld erwartet wird. Dies wurde verwendet um die Eingabe des Geburtsdatums des Patienten mit Hilfe des Input-Typs **date** zu erleichtern. Mittels des Input-Typs **email** wird automatisch kontrolliert ob die eingegebene E-Mail Adresse dem Standard für diese entspricht.

2.4.1 Verschlüsselung der Daten

Um die Daten der Patienten zu sichern ist es unabdingbar, die Verbindung zwischen Server und Client zu verschlüsseln. Hierdurch wird verhindert, dass sich jemand in die Verbindung einklinkt und die Nachrichten mit liest. Hierzu benötigt sowohl der Server als auch der Client jeweils einen sogenannten **public** und einen **privat key**. Der Server benötigt zudem ein Zertifikat, welches der Client mit Hilfe seiner lokal gespeicherten Zertifikate kontrollieren kann. Hierdurch kann der Client nun sicher sein das er wirklich mit dem gewünschten Server kommuniziert.

Anschließend wird eine Zufallszahl ausgetauscht, auf deren Grundlage die weitere Kommunikation symmetrisch Verschlüsselt wird. Für diesen Austausch stehen verschiedene Vorgehen zur Auswahl. Bei einem Vorgehen wird der Schlüssel vom Client erzeugt und dann mit Hilfe des **public Keys** des Servers verschlüsselt und diesem gesendet. Der Server wiederum kann diese verschlüsselte Nachricht mit seinem **privat key** entschlüsseln und erhält so die Zufallszahl für die symmetrisch verschlüsselte Verbindung. Ein weiteres Vorgehen für den Schlüsselaustausch wäre das Diffie-Hellman-Schlüsselaustauschverfahren. Hierbei erarbeiten sich die beiden Parteien gemeinsam den Schlüssel für die spätere sichere Verbindung.

2.4.2 Authentifizierung bei Server

Ein sicherer Austausch der Daten bringt nun aber gar nichts, wenn jedermann eine sichere Verbindung zum Server aufbauen kann. Ein weiterer Sicherheitsmechanismus sollte daher sein, dass jeder der Daten vom Server holt vorher verifiziert wird, ob er befugt ist die gewünschten Daten abzurufen. Hierzu ist es unabdingbar, eine Authentifizierung des Nutzers beim Server in die Plattform einzubauen. Hierzu wird typischerweise Benutzername und Passwort verwendet. Man könnte aber auch die Gegebenheit ausnutzen, dass der Patient mindestens ein mal beim Therapeuten vor Ort ist und Beispielsweise über das ab scannen eines QR-Codes, welcher im Therapeuten Client angezeigt wird, den Patienten Client autorisieren. Hierdurch würde das lästige Passwort wegfallen, wenn der Patienten Client aber aus irgend einem Grund seine Zugangsdaten vergisst, kann der Patient sich erst wieder bei seiner nächsten Sitzung bei seinem Therapeuten einloggen.

2.5 Erstellung von Antwort abhängigen Fragebögen

Um dem Therapeuten innerhalb des für ihn konzipierten Clients die Möglichkeit zu geben eigene Fragebögen zu erstellen wurde versucht ein Modul zu finden das sich einfach in die Anwendung integrieren lässt. Es gibt einige Module für das Angular Framework, welche eine Oberfläche bieten um Fragebögen zu erstellen. Jedoch war eine der Voraussetzungen, dass die Fragen von der Antwort der vorigen Frage abhängig sein sollte.

Keines der gefundenen Framework bot jedoch dieses Feature. Somit wurde untersucht welche anderen Anwendungen es gibt, welche für diesen Zweck angepasst werden könnten. Hierbei fiel ein Tool zur Modellierung von Business Prozessen auf. Dieses, BPMN-IO [1] genannte Tool, kann sehr einfach in Angular eingebunden werden und bietet eine graphische Oberfläche. Im so genannten **BPMN-IO Editor** können Unterschiedliche Elemente auf eine Leinwand gezogen werden und beliebig mit einander verbunden werden. Jedem Element und jeder Verbindung kann dabei eine Beschriftung hinzugefügt werden.

Durch das hinzufügen eines Eigenschaftspanels wird es Möglich gemacht den einzelnen Elementen Metainformationen anzufügen. In Abbildung 2.1 ist ein Screenshot des Editors zu sehen.

Da dieses Modul sehr Modular aufgebaut ist, ist es möglich es nach den eingenen Vorstellungen zu verändern. So ist die Möglichkeit gegeben das Aussehen so wie das Verhalten ganz und gar so anzupassen das aus diesem Modellierungstool für Business Prozesse ein Tool zur Modellierung von Fragebögen werden kann. Hierdurch kann die Oberfläche so weiter vereinfacht werden, dass das daraus entstehende Tool ohne große Einarbeitung verwendet werden kann.

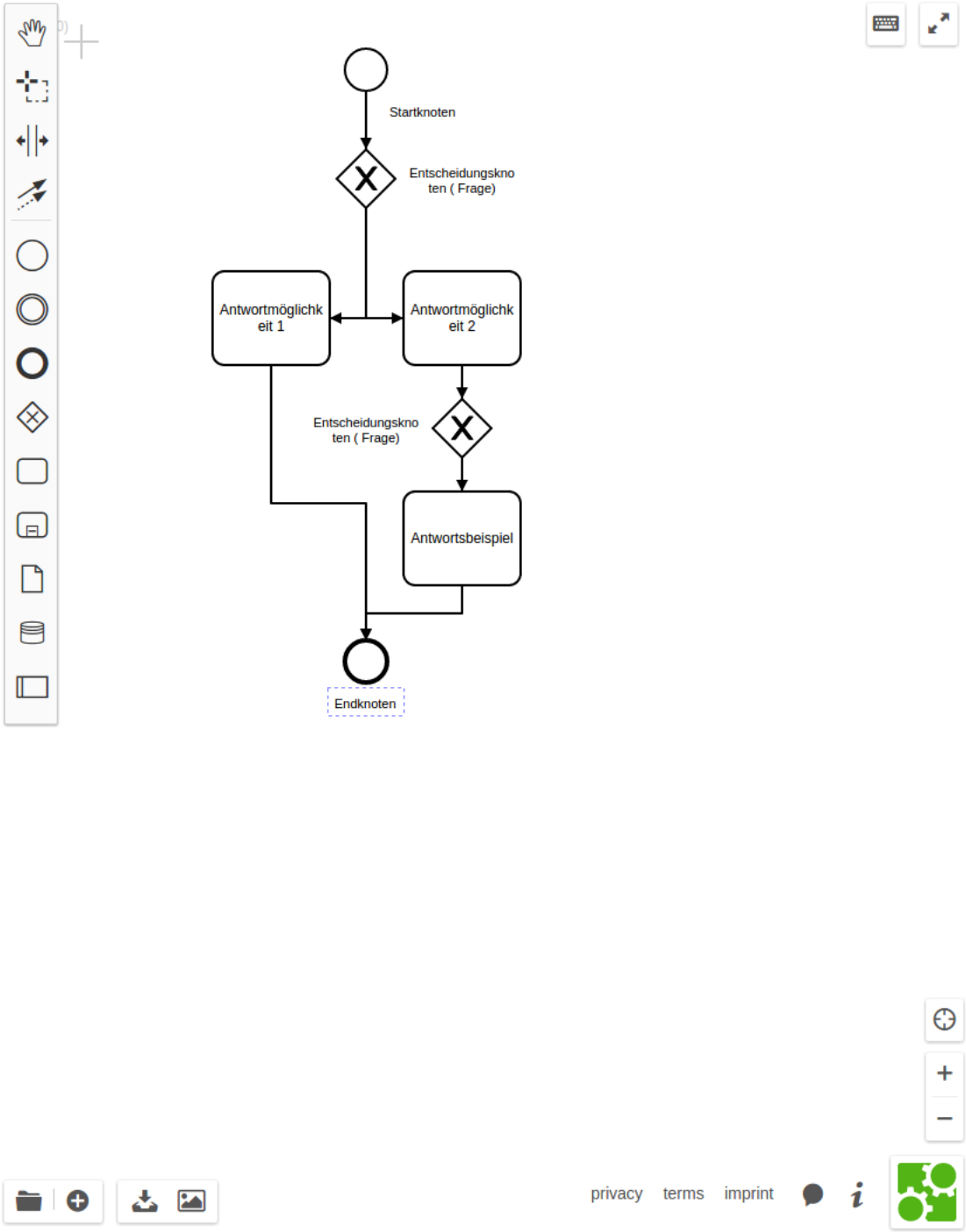

Abbildung 2.1: Screenshot des BPMN-IO Editors

2.5.1 Aufbau der exportierten XML Datei

Der auf diese weiße modellierte Fragebogen kann dann als XML-Datei exportiert werden. Im folgenden Abschnitt wird der Aufbau dieser Datei aufgezeigt, da im Kapitel **??** näher auf diesen Aufbau eingegangen wird. Um den Fragebogen innerhalb des Patienten Clients anzeigen zu können wird diese XML-Datei in JSON geparst

Jedes BPMN-Element im Editor, wie auch die Verbindungspfeile werden in einem XML-Element Abgebildet. so wird Beispielsweise das in Abbildung 2.1 zu sehende **StartEvent** als **bpmn:startEvent** abgebildet. Jedes einzelne XML-Element hat eine Attribut **id** in welchem eine einzigartige Id gespeichert ist, welche aus dem Typ des Elements + einer Zufallszahl besteht. Wodurch jedes Element eindeutig identifiziert werden kann. Wurde, wie im Beispiel oben zu sehen, ein Text an das Element angehängt wird dieser in dem Attribut **name** gespeichert.

Im Fall, dass ein Verbindungspfeil(**SequenceFlow**) zwischen zwei Elementen besteht, beinhalten die XML-Elemente die Elemente **bpmn:incoming** und/oder **bpmn:outcoming**. Dessen Text spezifiziert die Id eines SequenceFlows. Dieser wird durch ein eigenes XML-Element spezifiziert. Das Element eines SequenceFlows beinhaltet immer die Attribute **sourceRef** und **targetRef**. Diese spezifizieren Anfang und Ende des Pfeils. Um nun also das Folgeelement zu einem anderen zu finden muss der ausgehende SequenceFlows gefunden werden. Das unter **targetRef** spezifizierte Element ist das Folgeelement.

Das Endelement wird durch **bpmn:endEvent** spezifiziert. Dies wurde verwendet um zu erkennen wenn der Fragebogen beendet ist.

2.6 Single-Page-Application (SPA)

Die im laufe dieser Arbeit entstandenen Clients wurden beide als **Webanwendung (*s*ingle-*p*age-*a*pplication)** implementiert. Sie werden zwar beide in Unterschiedliche Kontexte eingebettet basieren im Kern aber beide auf AngularJS. Eine SPA unterscheidet sich zu einer regulären Internetseite darin, dass sie nur aus einer einzelnen HTML Seite besteht. Wenn innerhalb dieser Webanwendung zu einer anderen Seite gewechselt wird initiert dies keinen Verbindungsaufbau zum Webserver. Die Informationen über die neue Seite wurden bereits beim Seitenaufruf geladen.

Hierdurch wird der Server auf zwei Arten entlastet. Zum einen da seltener eine neue Verbindung initiiert werden muss und zum anderen weil ein Großteil der Logik auf den Client ausgelagert wird. Nach dem Laden der Hauptseite wird der Server nur noch kontaktiert um Nutzdaten zu liefern, welche jedoch auch auf einen anderen Server ausgelagert werden können.

Mit Hilfe des **Web Storage** (auch **DOM Storage** oder Supercookies genannt) ist es Möglich die Daten auch auf dem Client persistent zu speichern, wodurch eine gewissen offline Funktionalität implementiert werden kann. Dies funktioniert mit SPA's besonders gut da hier Logik gespeichert

wird und nicht wie im Fall einer regulären Webseite vor verarbeitete HTML Seiten. Für den Nutzer auf dem Endgerät hat diese den Vorteil das die Seitenübergänge fließender von statten gehen, da die benötigten Daten meistens schon vorhanden sind.

2.6.1 Aufbau einer SPA - Am Beispiel von AngularJS

Hier wird kurz auf den Aufbau einer **single-page-application** eingegangen um ein Grundlegendes Verständnis für die Implementierung in Kapitel 4 zu bieten.

Wie oben erwähnt besteht eine SPA nur aus einer HTML Seite, dies ist jedoch nur die Halbe Wahrheit, Zur besseren Übersicht werden die einzelnen Seiten der Anwendung dennoch in eigene HTML Dateien Ausgelagert. Diese beinhalten jedoch ausschließlich die anzuzeigenden HTML Elemente der Seite und wird mittels AngularJS in die gewohnte **head-body** Umgebung der **index.html** eingebunden. Hierdurch wurde die in einer **Model-View-Controller Architektur** gewöhnliche **View** vom Rest der Anwendung abgekapselt. Die **Controller** werden bei **AngularJS** in eigene JavaScript Dateien ausgelagert. Diese sind für die Logik der Seite zuständig. Das **Model** beinhaltet hierbei das oben erwähnte **Web Stroage** und die Anbindung an einen Datenbank Server.

Die **index.html** ist nun also diese eine Seite die sich der Nutzer vom Server herunterlädt. Diese jedoch verlinkt die JavaScript Dateien der verwendeten **npm module** und Controllern. Ebenso werden innerhalb der **index.html** die **CSS** Dateien im Header der HTML Datei eingebunden. Anschließend wird im **Body** die Navigationsleiste spezifiziert.

Die dort verwendeten **Routen** werden in der app.js spezifiziert. Zuerst wird in dieser Datei jedoch das **angular.module** definiert, welches sozusagen als **Container** für alle anderen Teile der Anwendungen fungiert. Eine neuer **Controller** Beispielsweise, wird mittels **angular.module(«moduleName»).*controller*(«Name»),function(«Abhängigkeiten»){ «Code» });** spezifiziert und ist somit Teil des **modules**.

Die Startseite bzw. die aufgerufenen Seiten werden werden dann in die ebenfalls in der **index.html** definierten **ng-view** angezeigt. Jede der Seiten bzw. Routen besteht aus einer HTML Datei welche die **View** spezifiziert und einem Controller welche der Übersichtlichkeit in jeweils eigene Dateien ausgelagert sind. Im Startprojekt von Ionic werden diese in einer einzigen Datei spezifiziert.

2.7 Anwendungsentwicklung

Am Anfang jeder 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 für jede einzelne zu implementieren. Aus diesem Dilemma heraus entstanden verschie-

denen 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 Therapeuten/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.

2.7.1 Technologie Analyse für mobile App Entwicklung

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% [6] der Smartphone Benutzer ab.

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.

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, welche jedoch von dem meisten neuer Browsern erfüllt werden.

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. Dadurch, dass die App auf eine Webserver gehostet ist, kann diese schnell veröffentlicht und aktualisiert werden. Wenn der Nutzer die Anwendung mit einer aktiven Internet Verbindung öffnet, aktualisiert sich die App automatisch.

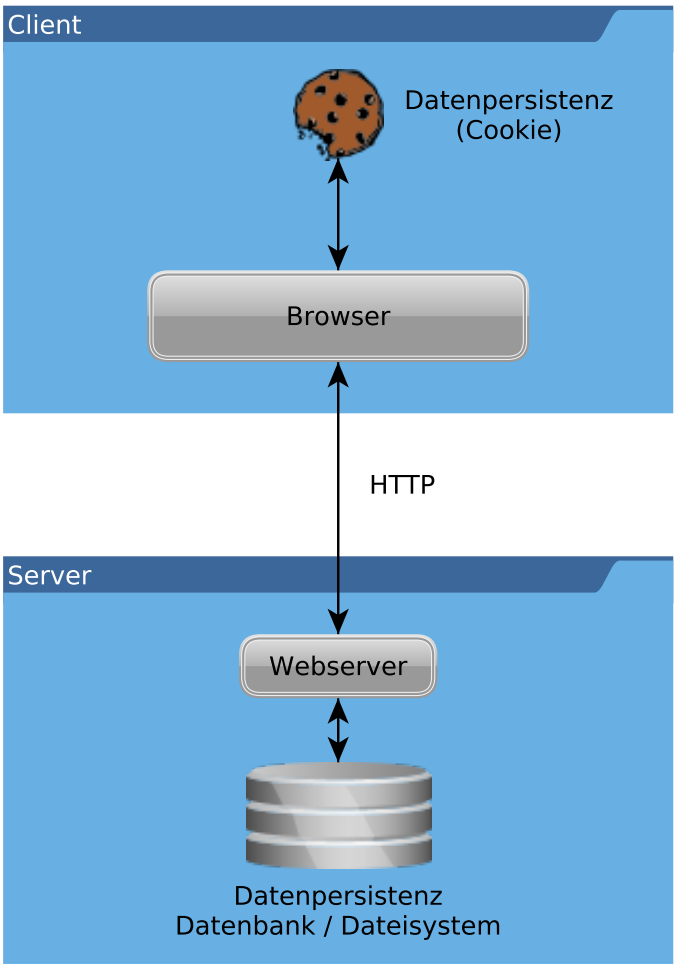

Abbildung 2.2: Schematische Darstellung einer WebApp

Sofern es sich bei der Anwendung nicht um eine sogenannte **Single Page Application** handelt, wodurch die Berechnungen auf den Client ausgelagert werden würden, übernimmt der Server, welcher die Anwendung hostet einen Großteil der Datenverarbeitung. Dann dient der Client lediglich der Anzeige der Anwendung und Interaktion mit dem Nutzer.

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, wodurch Apps welche nicht auf das Betriebssystem zugeschnitten sind meist keinen sehr großen Anklang bei den Nutzern findet.

Hybride Apps basieren wie auch die Web-Apps auf den Webtechnologien HTML5, CSS und JavaScript. Laufen aber im Gegensatz zu dem Web-Apps in einem mitgelieferten Minibrowser(Webview Container) oder, für den Nutzer nicht sichtbar, im vom Betriebssystem bereitgestellten Webbrowser. Dann werden jedoch jegliche Bedienelemente, wie Beispielsweise die URL-Leiste oder sonstige Buttons durch das Framework ausgeblendet.

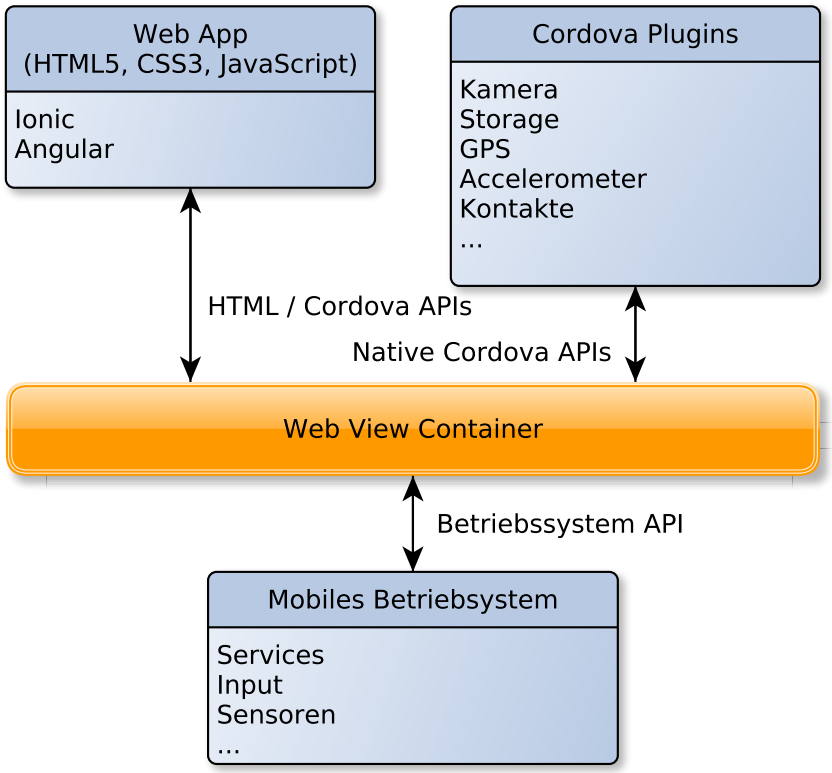

Abbildung 2.3: Aufbau des Ionic-Cordova Frameworks

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.

Native Cross Plattform Apps mit gemeinsamer Codebasis verspricht **Xamarin**, ein Projekt das auf der quellen-offenen Implementierung von Microsofts .NET Framework, Mono basiert [3]. 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 vor allem für rechenintensivere Anwendungen interessant macht.

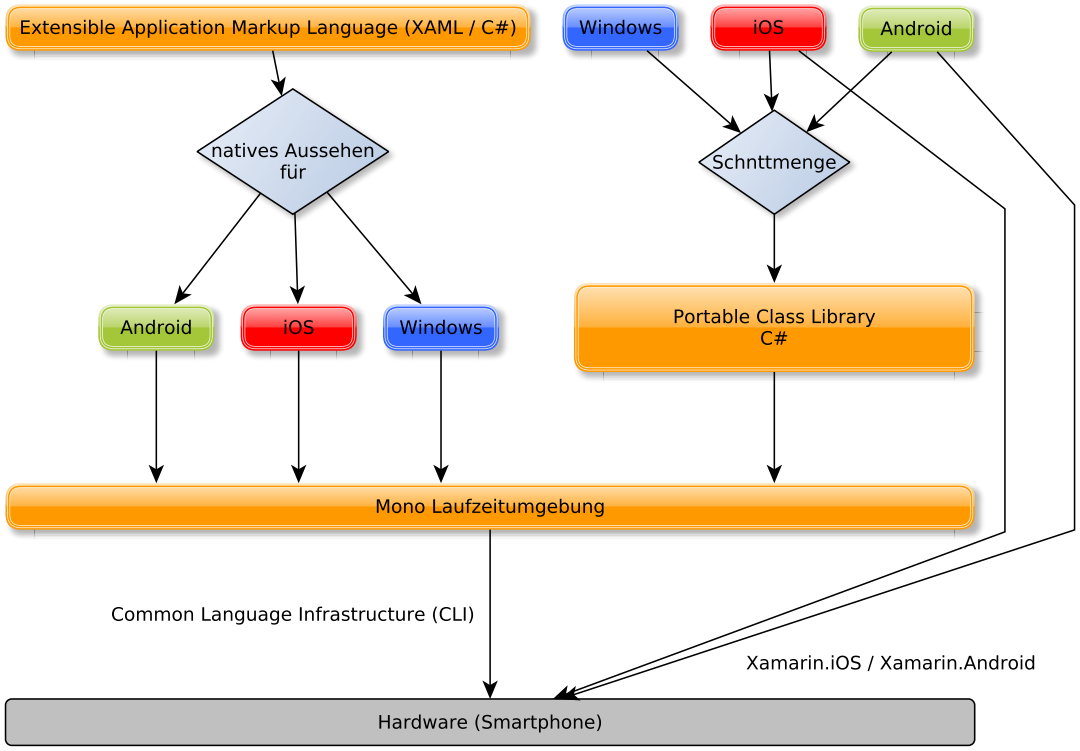

Abbildung 2.4: Aufbau des Xamarin Frameworks

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

Im folgenden Kapitel soll untersucht werden mit Hilfe welcher Technologien die Plattform realisiert werden kann. Hierbei wird eine Abwägung getroffen, welche Vor- und Nachteile die einzelnen Technologien mit sich bringen. Zuerst werden Technologien für den Desktop Client des Therapeuten untersucht. Anschließend wird die Entwicklung einer mobilen Anwendung für den Patienten Client untersucht.

vielleicht noch Tabellen der Vorteile aller

2.8.1 Therapeuten Client

Der Therapeuten Client sollte eine Desktop Anwendung sein. Diese benötigt keine intensive Rechenkapazität, Grafikleistung oder dergleichen. Genau dies wären jedoch die Vorteile der nativen Entwicklung, neben der näheren Anbindung von Peripheriegeräten. Da aber auch diese für die Anwendung nicht von Nöten sind, wird untersucht, mit Hilfe welcher Technologien eine möglichst breites Feld an Endgeräten abgedeckt werden kann.

Hierfür gibt es sehr viele unterschiedliche Plattformen und Frameworks welche auf völlig unterschiedlichen Technologien aufbauen. Was alle gemeinsam haben, ist eine Laufzeitumgeben in welcher das entwickelte Programm später läuft. Eine der bekanntesten ist hier wohl Java aber auch Mono, die quellen offene Implementierung von Microsofts .NET Framework sollte hier nicht außer acht gelassen werden.

Des weiteren gibt es wie auch für die mobile Cross-Plattform-Entwicklung Frameworks, welche einen Mini Browser bereit stellen welcher die Anzeige und Interaktion verwaltet. Die eigentliche Anwendung ist eine **single-page-application**, welche zusammen mit dem Mini Browser, sozusagen in diesen integriert, ausgeliefert wird. Ein Beispiel für diese Technologie ist das TideSDK mit dessen Hilfe Webentwickler Anwendungen für Windows, Mac OS X und Linux entwickeln können.

Eine weitere Anforderung an die zu entwickelnden Plattform ist die globale Persistenz der Daten über beide Clients. Diese Anforderung wurde mittels eines Webservers gelöst. Aus diesem Umstand heraus wurde beschlossen, den Client für den Therapeuten in Form einer **single-page-application** [2.6] zu implementieren, da diese auf jedem Geräten mit aktuellem Browser angezeigt werden kann und auf dem selben Server wie die Datenpersistenz gehostet werden kann. Hierdurch ist es dem Therapeuten auch möglich die Anwendung auf einem Tablet oder Handy zu öffnen, ohne diese Installieren zu müssen.

2.8.2 Patienten Client

Der Patienten Client sollte eine App für mobile Endgeräte sein, da der Patient dann einfach an die Erledigung seiner Übung erinnert werden kann. Abgesehen davon bietet ein Smartphone viele Sensoren, welche zur Unterstützung der Therapie hilfreich sein können. Um möglichst viele Patienten mit dieser Anwendung erreichen zu können, wurde beschlossen diese nicht nativ nur für eine Plattform zu entwickeln. In Tabelle 2.8.2 sind die Vorteile der nativen App Entwicklung aufgezeigt.

Vorteil	Beschreibung
maximale Performance	nativer Code, dadurch keine Laufzeitumgebung nötig die Multi-Plattform-Code interpretiert
geringere Reaktionszeiten	direktes ansprechen von Geräte API, Prozessor und Speicher
Keine Beschränkung durch Laufzeitumgebung	Keine Laufzeitumgebung die den Entwickler auf die gebotene API einschränkt

Tabelle 2.1: Vorteile der nativen App Entwicklung

Es gibt viele Plattformen welche es erlauben, gleichzeitig für mehrere mobile Betriebssysteme zu entwickeln. Hier sollen einige dieser Plattformen und Frameworks genannt sowie deren Vor- und Nachteile aufgezeigt werden.

Ionic

Ionic ist ein Open-Source-Framework welches es Erlaubt den geschriebenen Quellcode plattformübergreifend zu verwenden. Das Framework basiert auf Apache Cordova und AngularJS und wird mittels der Webtechnologien HTML, CSS und JavaScript geschrieben. AngularJS liefert die Grundstruktur der App, auf die Später in diesem Kapitel noch genauer eingegangen wird.

Cordova liefert hierbei die Laufzeitumgebung der App, wodurch diese in einer mitgelieferten, nativen **WebView** auf dem Gerät angezeigt wird. Dieser WebView-Container bietet im Gegensatz zum nativen Browser den Vorteil, dass durch ihn die nativen APIs des Gerätes angesprochen werden können. Hierdurch ist es möglich auf alle Fähigkeiten wie Benachrichtigungen, Kamera, Speicher, Kontakte,... des Gerätes zuzugreifen.

Ionic selbst kümmert sich vor allem darum, dass sich die App auf jedem Endgerät nativ anfühlt, es werden alle Elemente an die Betriebssystem Richtlinien angepasst. Sowohl das Aussehen als auch die Platzierung, Beispielsweise des Zurück-Buttons oder des Kontext Menüs, wird angepasst, je nachdem für welche Plattform die App gebaut wird.

Des weiter bietet Ionic verschiedene Mechanismen, welche die Entwicklung angenehmer machen. Mittels des **Ionic Creator**s kann die App im Browser oder in einer mitgelieferten GUI direkt auf *quasi emulierten* Geräten getestet werden. Durch das sogenannte **live reload** wird die App ob nun im Browser oder auf dem Gerät laufend neu geladen, wenn eine Codeänderung vorgenommen wurde.

Xamarin

Der größte Vorteil von Xamarin bei der Cross-Plattform-Entwicklung ist die Performance. Während hybride App eine schlechtere Performance gegenüber native Entwickelten Apps haben schafft es Xamarin, dadurch dass es nativen Code generiert, teilweise sogar eine bessere Performance an den Tag zu legen wie mit **Objective-C** programmierte iOS Apps [7]. Auch im Vergleich mit nativ entwickelten Anwendungen für Android schneidet Xamarin sehr gut bis hin zu besser ab [2]. Nur Apps welche in C++ entwickelt wurden sind sowohl auf iOS als auch auf Android absoluter Spitzenreiter wenn es um die Performance geht.

Dadurch, dass Xamarin nativen Code erzeugt es nach einer Änderung im Quellcode jedes mal bevor man diese Sichtbar machen kann nötig die App zu Bauen und auf ein Gerät oder in den Emulator auszuliefern, was jedes mal relativ viel Zeit in Anspruch nimmt. Eine **live-view** Funktion wie es Ionic anbietet gibt es auf dieser Plattform nicht.

Einer der größten Nachteile von Xamarin ist der Preis. Wobei es seit einiger Zeit eine kostenlose Version gibt, welche für Forschung, Open Source Projekte sowie Bildung und kleine Teams verwendet werden kann. Wenn man mit einer solchen App Geld verdienen möchte, will Xamarin auch etwas davon haben.

3 Entwurf

Im folgenden Kapitel wird der Entwurf der Plattform vorgestellt. Dies beinhaltet die Vision, welche aufzeigt was von der zu entwickelten Anwendung erwartet wird und welche Arbeitsabläufe von ihr abgedeckt werden sollen. In der darauf folgenden Anforderungsanalyse [3.2] wird sich damit auseinander gesetzt, welche Einzelteile zur Realisierung der Plattform benötigt werden und welche Anforderungen an diese gestellt werden.

3.1 Vision

Mit Hilfe der zu entwickelnden Plattform soll es möglich sein Therapieaufgaben / Übungen und Fragebögen im Zuge einer Therapeutischen Betreuung zwischen Therapeut und Patient auf digitalem Weg auszutauschen. Hierdurch soll das in Kapitel 1 erwähnte Blatt Papier auf welchem die Aufgabe oder Übung beschreiben ist wegfallen und durch eine zeitgemäße App für das mobile Endgeräte des Patienten ersetzt werden.

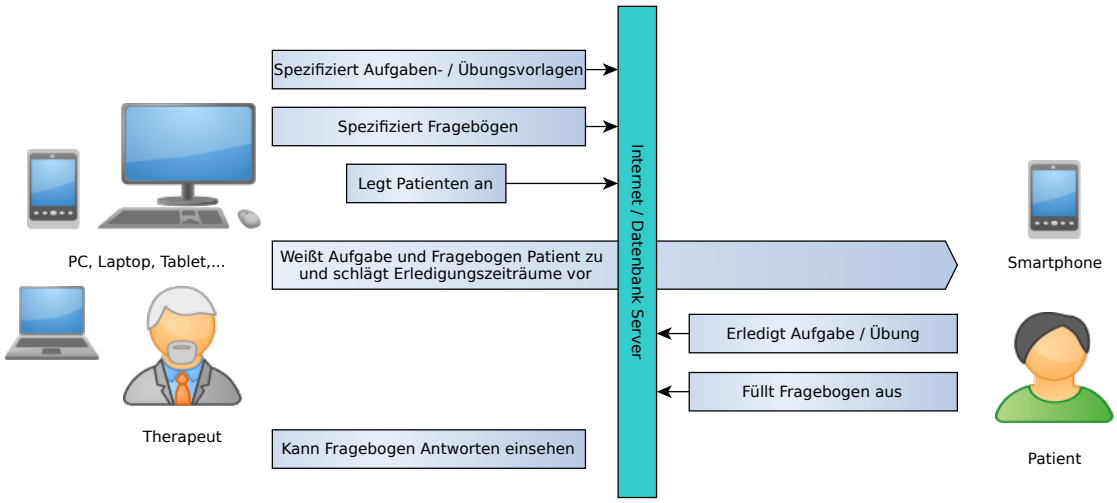

Abbildung 3.1: Workflow Diagramm vom erstellen der Aufgabe/Übung inc. Fragebogen bis zur Erledigung/Beantwortung

Dies gibt dem Therapeuten auch die Möglichkeit, digitale Medien zur Veranschaulichung der Aufgabe / Übung zu verwenden, welche der Patient während er die Aufgabe oder Übung macht vorliegen hat. Wodurch diesem die korrekte Durchführung der Aufgabe erleichtert wird.

Hierdurch wird es auch möglich völlig neue Aufgaben und Übungen zu konzipieren. Dem Patienten könnten auf diesem Weg auch Medien wie etwa Bilder oder Videos mit nach Hause gegeben werden. Denkbar wäre auch interaktive Medien wie Spiele oder ähnliches.

Der Therapeut möchte in einer Desktop Anwendung seine Patienten verwalten können, sowie Vorlagen für Therapeutische Aufgaben, Übungen und Fragebögen erstellen. Um diese dann den einzelnen Patienten zuweisen zu können. Ebenso soll es für den Therapeuten möglich sein Fragebögen mit von den Antworten abhängigen Fragen zu erstellen um diese nach erfolgreicher Erledigung der Therapeutischen Aufgabe oder Übung 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 des Patienten soll dann auf das Profil des Patienten, welches 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. Hierdurch soll der Anteil der Patienten welche ihre Aufgabe gar nicht erledigen verringert werden.

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 sowohl der Therapeuten Client als auch die mobile Anwendung Zugriff auf die Daten hat.

Der Therapeuten Client soll dem Therapeuten die Oberfläche bieten mithilfe welcher es ihm möglich sein muss die Patienten, Therapieaufgaben / Übungen und Fragebögen verwalten zu können. Die Patienten müssen angelegt, angezeigt, bearbeitet und gelöscht werden können.

Des weiter muss es dem Therapeuten möglich sein, Aufgaben zu spezifizieren, diese sollen neben Name und Beschreibung auch verschieden Medien beinhalten können. Der Therapeut

soll außerdem die Möglichkeit haben Kontrollmechanismen an die Aufgabe anzuhängen, welche ihm Feedback darüber geben ob und wie der Patient die Aufgabe erledigt hat, damit dieser sich nicht nur auf das mündliche Feedback des Patienten verlassen muss.

In der Anzeige eines Patienten soll der Therapeut die zuvor erstellten Therapieaufgaben / Übungsvorlagen sowie Fragebögen diesem zuweisen können. Bei der Zuweisung gibt der Therapeut das/die Ereignisse(Zeit, Ort, etc.) an zu denen der Patient an die Aufgabe erinnert werden soll.

Ebenso muss der Therapeuten Client einen Mechanismus bieten in welchem der Therapeut Fragebögen erstellen kann, in welchen die nächste Frage von der Antwort der zuvor gestellten Frage abhängig ist.

Nachdem ein Fragebogen von einem Patienten beantwortet wurde muss es dem Therapeuten möglich sein, diesen einzusehen.

Der Patienten Client soll auf möglichst vielen mobilen Plattformen lauffähig sein, damit so wenig Patienten wie möglich von dieser neuen Methode der Therapeut - Patient Interaktion ausgeschlossen werden.

Der Patient muss nach dem ersten Start der mobilen Anwendung Verbindung zu seinem Profil herstellen können, etwa durch Eingabe eines Benutzernamens oder einer ID. Dieser Vorgang sollte durch einen Sicherheitsmechanismus geschützt werden, um die Daten des Patienten vor dem Zugriff unbefugter zu schützen.

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 das Startereignis der Aufgabe eingetreten ist. Die mobile Anwendung soll die Möglichkeit bieten, die vom Therapeuten vorgegebenen Ereignisse anzupassen.

Ist das Startereignis eingetroffen, soll der Patienten Client eine Ansicht bieten, in welcher dieser die Details zu der anstehenden Aufgabe / Übung einsehen kann. Des weiteren muss er die Möglichkeit haben, die angehängten Materialien auf möglichst einfache Weiße einzusehen um das erledigen der Aufgabe / Übung so einfach wie es nur geht zu gestalten. Je weniger Hindernisse der Patient hierbei hat, desto höher ist die Chance, dass dieser die Aufgabe auch wirklich macht. Was wiederum den Therapieerfolg erhöht. Die Detailansicht zu einer bestimmten Aufgaben sollte jederzeit aufrufbar sein, nicht nur wenn das Startereignis eingetroffen ist.

Der Client für den Patienten sollte ebenso die Option bieten, eine Übersicht über alle zugewiesenen Aufgaben anzuzeigen.

3.2.1 Funktionale Anforderungen

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

Datenbank Server

Für den Datenbank Server können die in Tabelle 3.2.1 aufgezeigten Funktionalen Anforderungen spezifiziert werden.

Nr.	Anforderung	Beschreibung
1.1	Datenpersistenz	Die empfangenen Daten müssen persistent gespeichert werden
1.2	Daten Sicherung	Automatische sicherung der Daten auf Zweitsystem
1.3	Datenschnittstelle	Schnittstelle für den Datenaustausch mit unterschiedlichen Geräten

Tabelle 3.1: Funktionale Anforderungen an den Datenbank Server

Therapeuten Client

Für den Client des Therapeuten sind dies die in Tabelle 3.2 dargestellten Anforderungen.

Nr.	Bezeichnung	Beschreibung
2.1	**Patientenverwaltung**	Die Patienten sollen mit ihren persönlichen Daten angezeigt und verwaltet werden können
2.1.1	anlegen	Einen neuen Patienten hinzufügen
2.1.2	editieren	Einen bestehenden editieren
2.1.3	löschen	Einen bestehenden löschen
2.2	**Aufgabenverwaltung**	Verwaltung und Anzeige von Aufgaben / Übungsvorlagen
2.2.1	anlegen	Eine neue Vorlage hinzufügen
2.2.2	editieren	Einen bestehenden editieren
2.2.3	löschen	Eine bestehenden löschen
2.3	**Fragebogenverwaltung**	Verwaltung und Anzeige von Fragebögen
2.3.1	anlegen	Eine neue Vorlage hinzufügen
2.3.2	editieren	Einen bestehenden editieren
2.3.3	löschen	Eine bestehenden löschen

Tabelle 3.2: Anforderungen an den Therapeuten Client

Aufgaben-/Übungsvorlagen

Dabei sollen die Aufgaben/Übungen mittels der in Tabelle 3.3 aufgezeigten Daten spezifiziert werden können.

Nr.	Bezeichnung	Beschreibung
A.1	**Name**	Ein kurzer, aussagekräftiger Name der Aufgaben-/Übungsvorlage
A.2	**Beschreibung**	Die textuelle Beschreibung der Aufgabe/Übung in beliebiger Länge
A.3	**Materialien**	Die Möglichkeit beliebig viele, unterschiedliche Materialien anzuhängen
A.4	**Kontrollmechanismen**	Spezifikation von Kontrollmechanismen welche Rückmeldung darüber geben ob die Aufgabe erledigt worden ist
A.5	**Kategorisierung**	Mechanismus um die Aufgaben/Übungen in Kategorien einordnen zu können um die Übersichtlichkeit zu erhöhen
A.6	**Verschiedene Erledigungskontexte**	Verschiedene Wege um zu spezifizieren wann der Patient an die Aufgabe erinnert werden soll

Tabelle 3.3: Anforderungen an die Vorlage einer Aufgabe/Übung

Fragebögen

Die Anforderungen an die Fragebogenerstellung werden in der Tabelle 3.4 aufgezeigt.

Nr.	Bezeichnung	Beschreibung
B.1	**Name**	Ein kurzer, aussagekräftiger Name um den Fragebogen identifizieren zu können
B.2	**Beschreibung**	Für eine textuelle Beschreibung des Fragebogens
B.3	**Antwortabhängige Fragen**	Möglichkeit um nächste Frage von der gegebenen Antwort des Patienten abhängig zu machen
B.4	**Verschiedene Fragetypen**	Option bieten um unterschiedliche Fragetypen stellen und beantworten zu können

Tabelle 3.4: Anforderungen an die Fragebogen Erstellung

Durch vier unterschiedliche Fragetypen können in der Regel alle Arten von Fragen abgebildet werden. Diese Typen werden in der Tabelle 3.5 aufgezeigt und näher spezifiziert.

Nr.	Fragetyp	Beschreibung
C.1	**Einzelantwort**	Aus mehreren Antwortmöglichkeiten kann **genau eine** ausgewählt werden
C.2	**Mehrfachantwort**	Aus mehreren Antwortmöglichkeiten können **beliebig viele** aus gewählt werden
C.3	**Bewertung**	Zu den einzelnen Fragen können **numerische Bewertungen** in einen definierten Bereich abgegeben werden
C.4	**Freitext**	Die Frage wird durch einen **freien Text** beantwortet

Tabelle 3.5: Übersicht über die benötigten Fragetypen

Patienten Client

Für den Client des Patienten sind dies die in Tabelle 3.6 dargestellten Anforderungen.

Nr.	Anforderung	Beschreibung
3.1	User Identifikation	Der Patient stellt Verbindung zu dem von Therapeut erstellten Profil her
3.2	Aufgabenanzeige	Übersicht der zugewiesenen Therapeutischen Aufgaben
3.3	Detailanzeige der Aufgabe	Eine detaillierte Anzeige der Aufgabe mit Name, Erklärung und angefügten Materialien
3.4	Anzeige der Materialien	Die angehängten Materialien müssen vom Patienten innerhalb des Clients eingesehen werden können
3.5	Verändern des Erledigungskontext	Der von Therapeuten vorgegebene Erledigungskontext soll vom Patienten veränderbar sein
3.6	Erinnerung	Der Patient soll an die Aufgabe erinnert werden
3.7	Fragebogen beantworten	Es muss möglich sein, einen Fragebogen zu beantworten

Tabelle 3.6: Funktionale Anforderungen an den Patienten Client

3.2.2 Nicht-Funktionale Anforderungen

Aus der Vision [3.1] und der Anforderungsanalyse [3.2] können folgende Nicht-Funktionale Anforderungen sowohl für den Datenbank Server als auch für die beiden Clients abgeleitet werden.

Datenbank Server

Für den Datenbank Server können die in Tabelle 3.2.2 aufgezeigten AnforderungenaAbgeleitet werden.

Nr.	Anforderung	Beschreibung
4.1	kurze Reaktionszeiten	Der Server sollte möglichst geringe Reaktionszeiten haben
4.2	Sicherheit	Die Daten sind vor dem Zugriff unbefugter zu schützen

Tabelle 3.7: Nicht-Funktionale Anforderungen an den Server

Therapeuten / Patienten Client

Für die beiden Clients gelten die in Tabelle 3.2.2 beschriebenen Anforderungen.

Nr.	Anforderung	Beschreibung
5.1	Natives Design	Die Anwendung soll auf jedem Gerät das gewohnte Design haben um die Usability zu erhöhen
5.2	vertretbare Reaktionszeiten	Keine unzumutbaren Wartezeiten
5.3	einfache Bedienung	Die Anwendung ziel auf eine möglichst einfach Bedienung ab welche keine Fachkenntnisse benötigt

Tabelle 3.8: Nicht-Funktionale Anforderungen an die Clients

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.

Mehr auf Inhalt der Bilder eingehen, zusammenhang klar machen -> FETT und in Tabbellen erläutern

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.

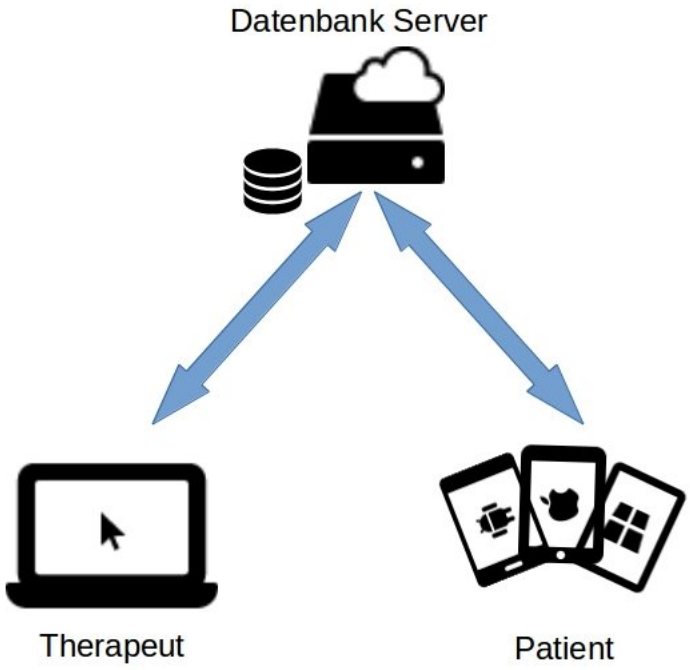

Abbildung 4.1: Client-Server Architektur der Plattform

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.

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.

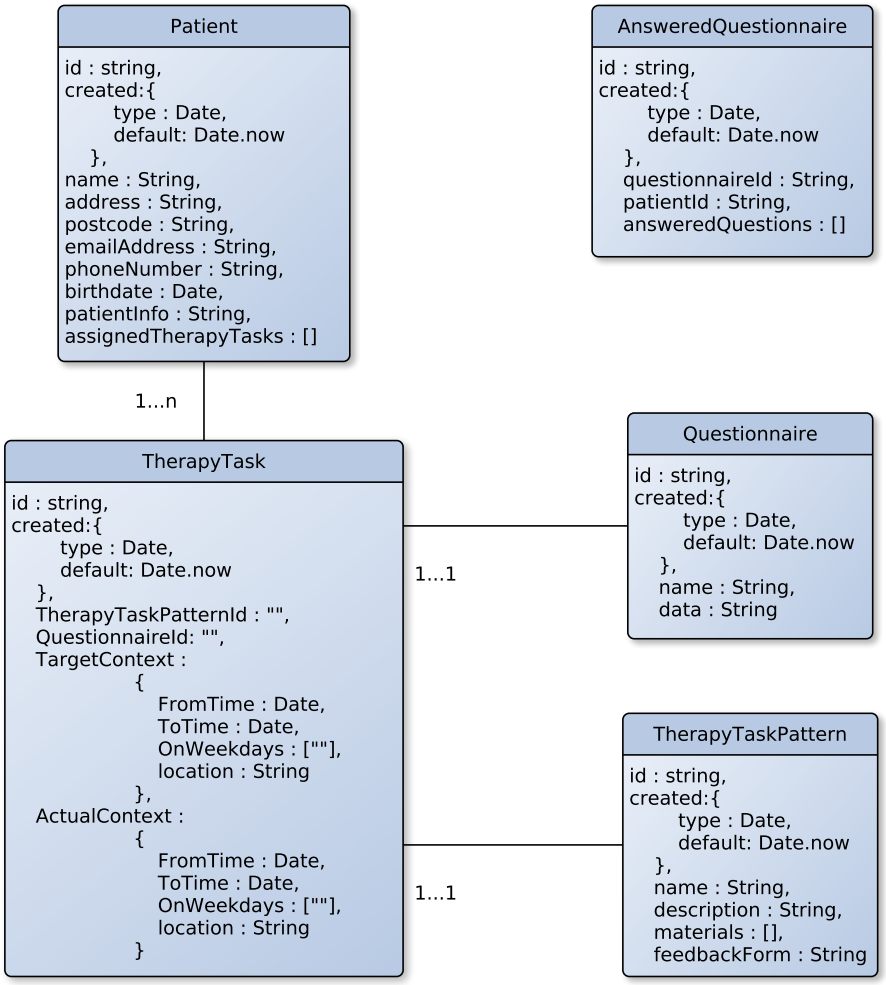

Abbildung 4.2: Visualisierung des Datenmodells

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 Datenbank Server

Zur Implementierung des in Kapitel 3.2 angesprochenen Webservers wurde **NodeJS** [4] 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.

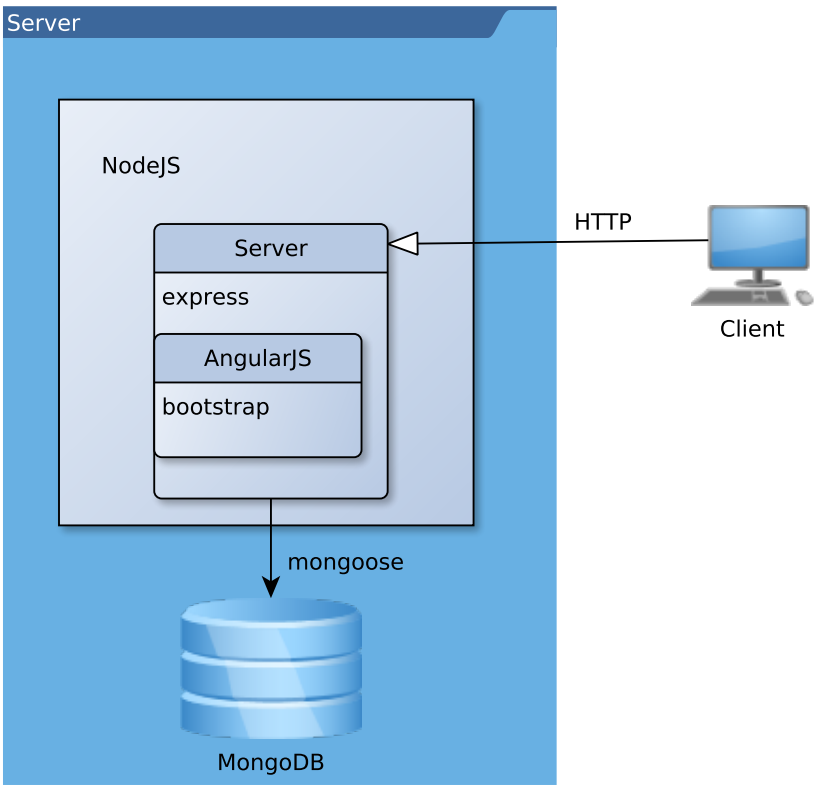

Abbildung 4.3: Übersicht über die Architektur des Servers

Wie in der **package.json** unter **scripts** -> **start** angegeben 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.

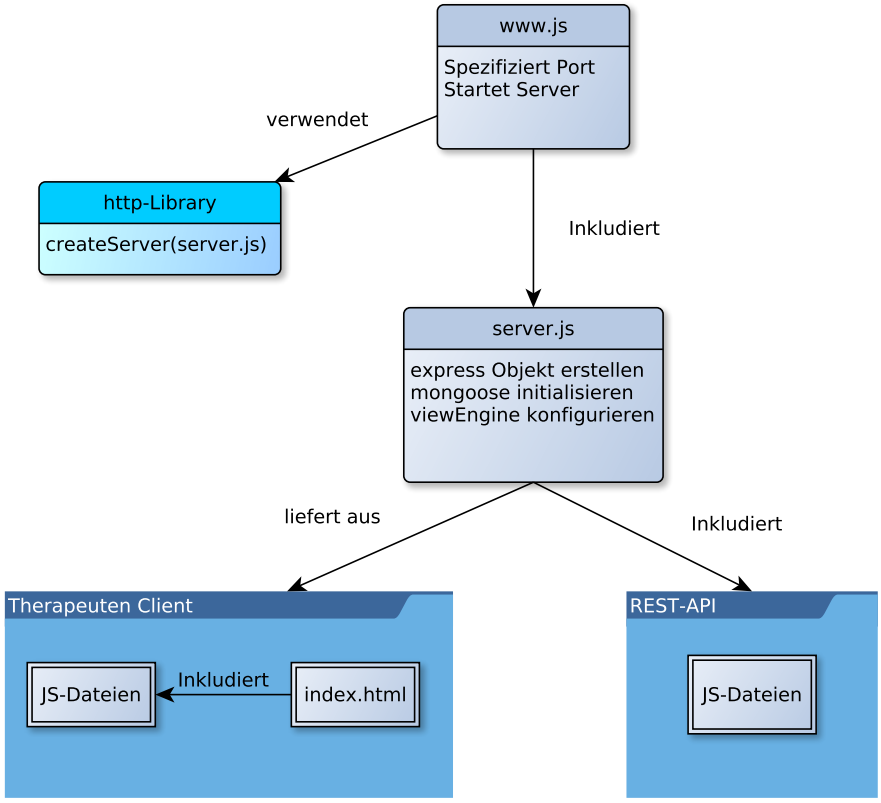

Abbildung 4.4: Script Architektur des Servers

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 wie in Abbildung 4.5 veranschaulicht.

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.

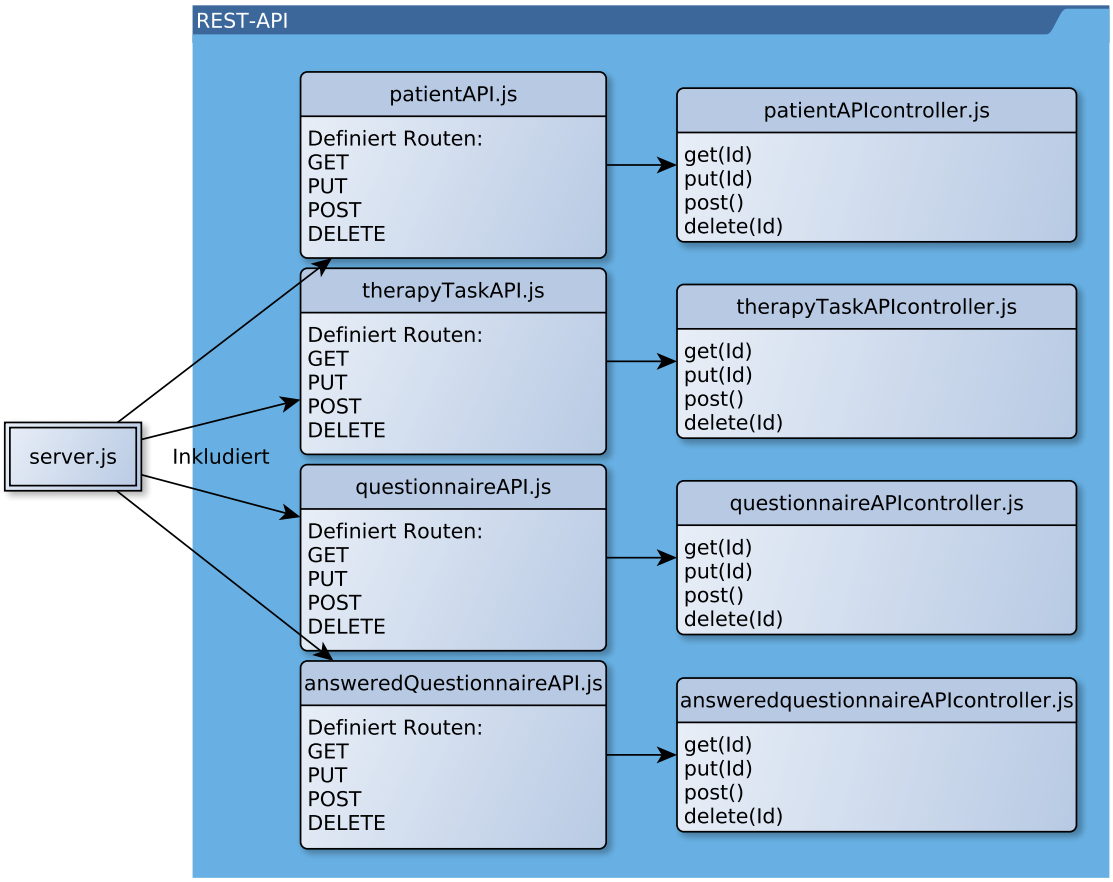

Abbildung 4.5: Architektur der REST-API

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 Client

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 wie in Abbildung 4.6 veranschaulicht.

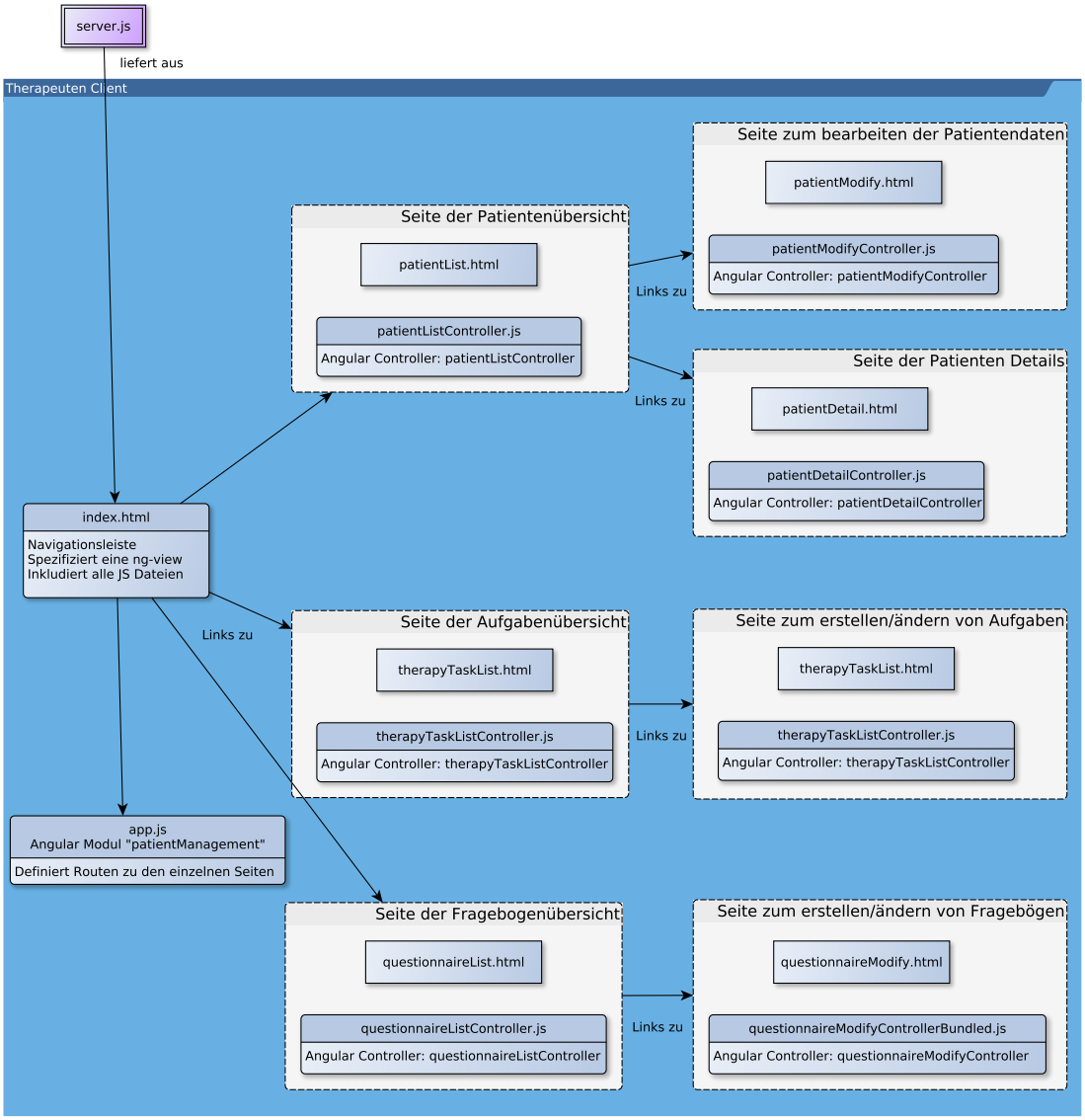

Abbildung 4.6: Übersicht über die Seiten des Therapeuten Clients

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 Funktion 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.

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 [4.8, 4.7, 4.9] gezeigt, darzustellen.

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

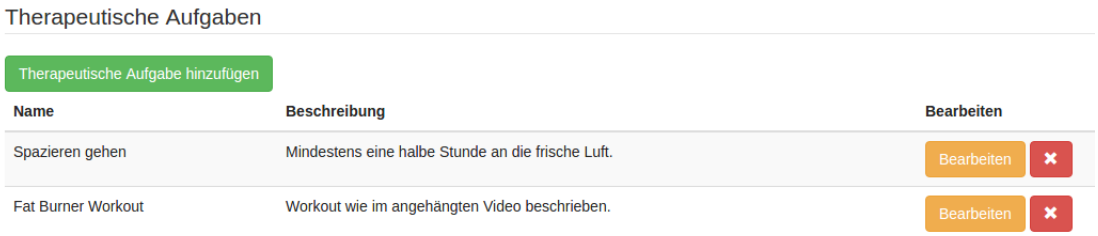

Abbildung 4.7: Übersicht über die Therapeutischen Aufgaben

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.

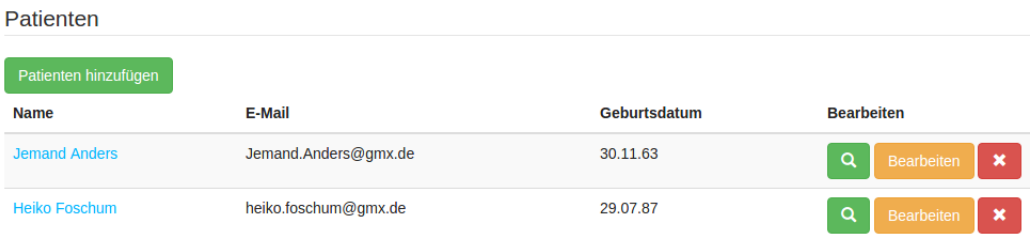

Abbildung 4.8: Seite der Patientenübersicht

Übersicht der Patienten

Die in Abbildung 4.8 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 in der URL übergibt. 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 4.9 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.

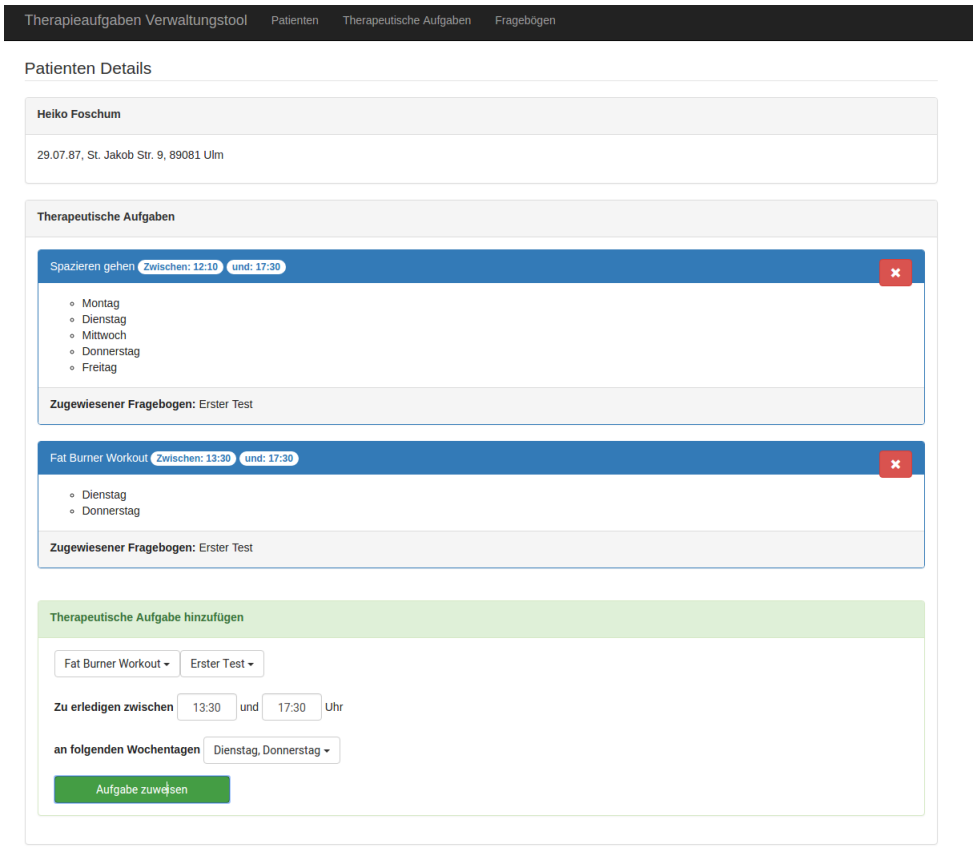

Abbildung 4.9: Ansicht der Patientenübersicht im Web Frontend

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 Fra-

gebögen 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 4.10). Auf deren Grundlage der Therapeut die Therapie verwalten und verbessern kann. Um die Übersichtlichkeit 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.

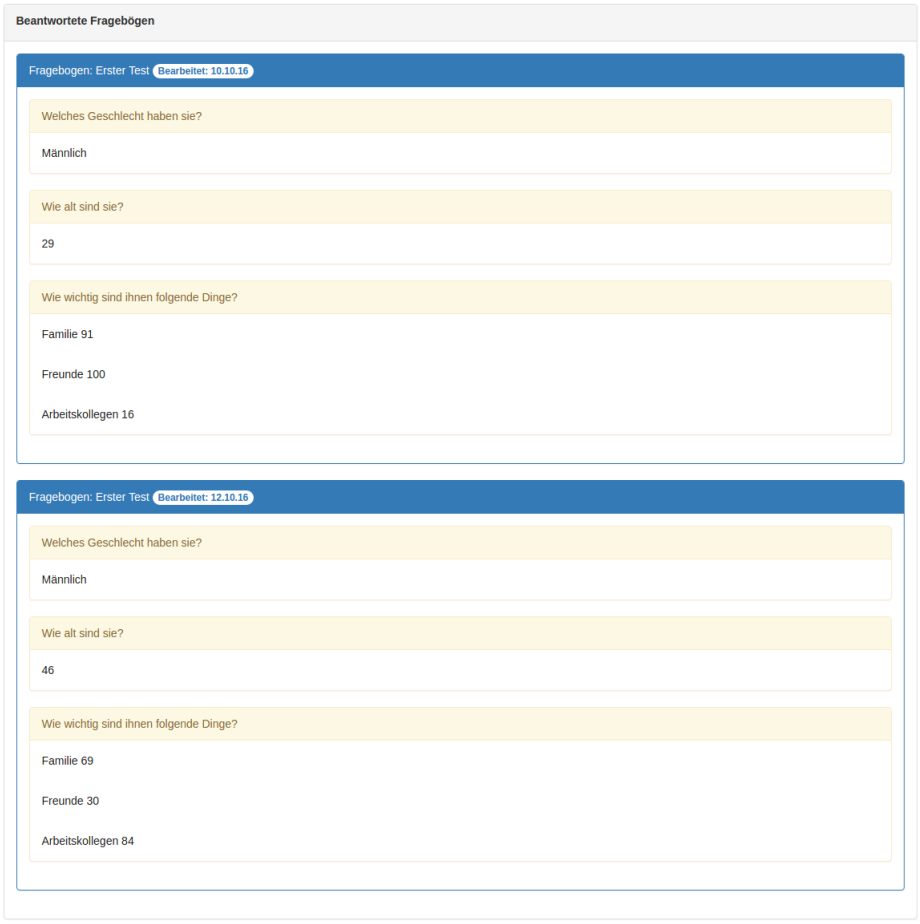

Abbildung 4.10: Seite der Patientenübersicht

Verwaltung der Therapeutischen Aufgaben / Übungen

Anhand der Ansicht in Abbildung 4.7 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 4.11 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="saveTherapyTask()"** 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.

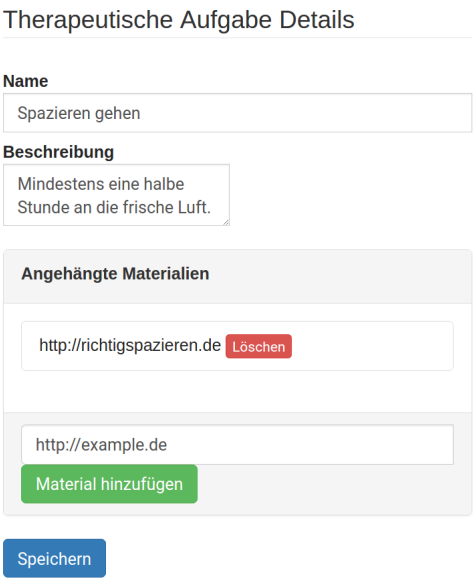

Abbildung 4.11: Formular zum Editieren oder Erstellen von Therapeutischen Aufgaben

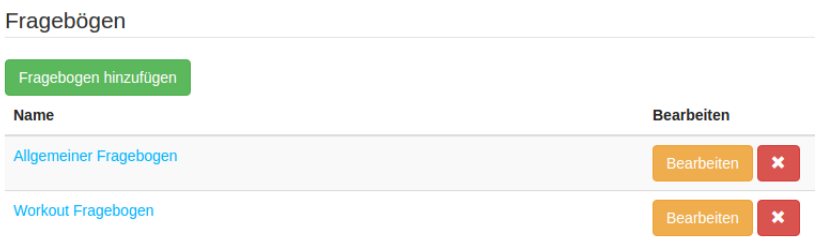

Abbildung 4.12: Übersicht über die Fragebögen

Verwaltung der Fragebögen

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.

Die Seite zum erstellen und editieren der Fragebögen 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.

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

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.

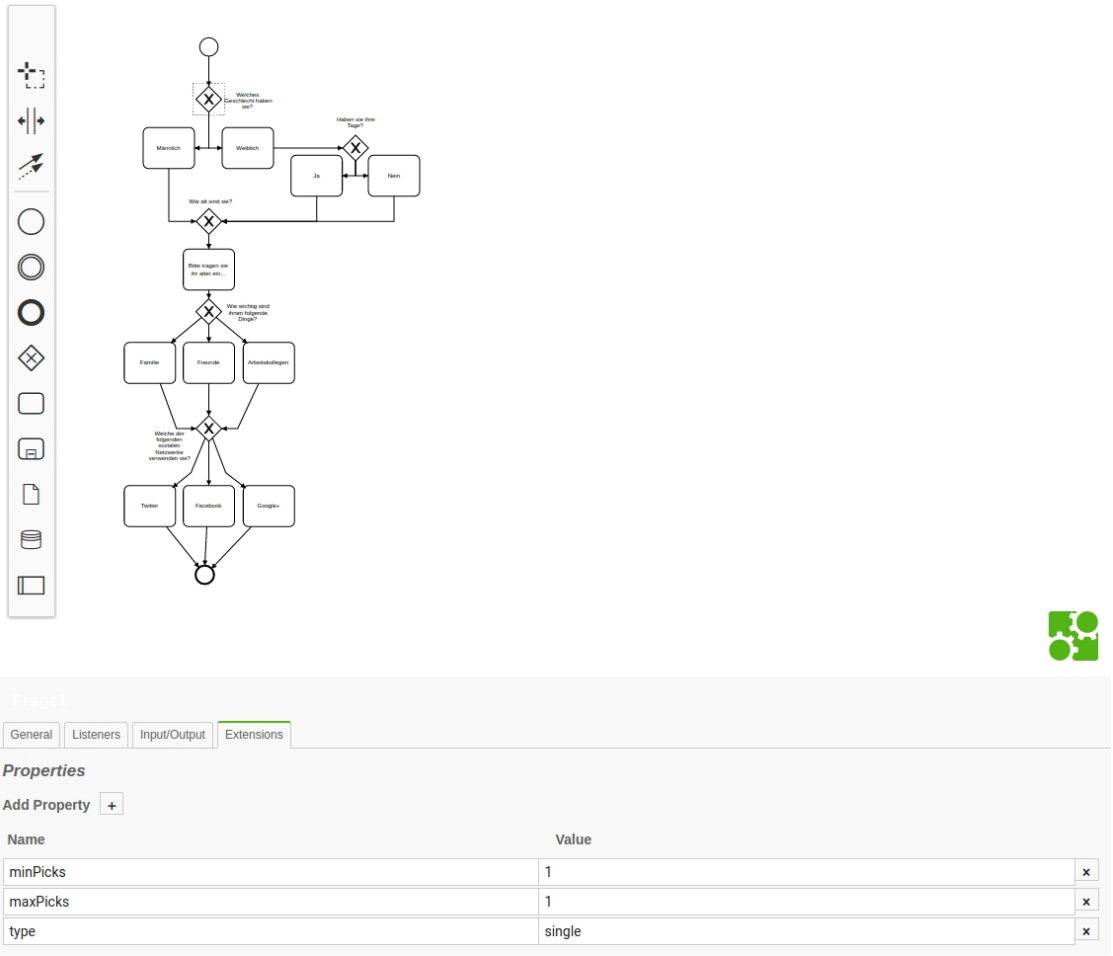

Abbildung 4.13: Ansicht des BPMN Bearbeitungstool

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.

Da der BPMN Editor ein Programmteil des Therapeuten Clients ist, welcher auf dem Client des Anwenders läuft aber dennoch Abhängigkeiten zu verschiedenen **npm** Modulen hat, war es hier nötig das Programm **browserify** zu verwenden, welches in Abbildung 4.14 veranschaulicht wird. **browserify** ist ein Kommandozeilen Tool und nimmt JavaScript Datei entgegen. Es schaut dann welche Abhängigkeiten in dieser Datei mittels **require** eingebunden sind und erzeugt daraus eine JavaScript Datei welche den gesamten benötigten Code beinhaltet. Hierdurch können auf dem Server installierte Abhängigkeiten auf den Client des Anwenders gebracht und dort ausgeführt werden.

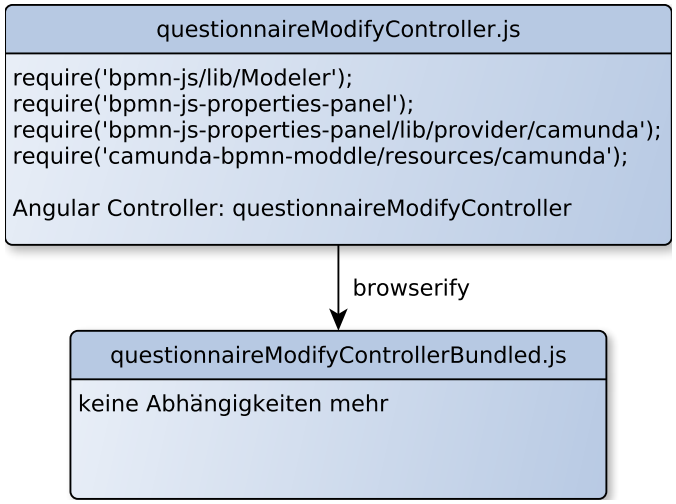

Abbildung 4.14: Veranschaulichung von **browserify**

4.3 Patienten Client

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 4.16 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.

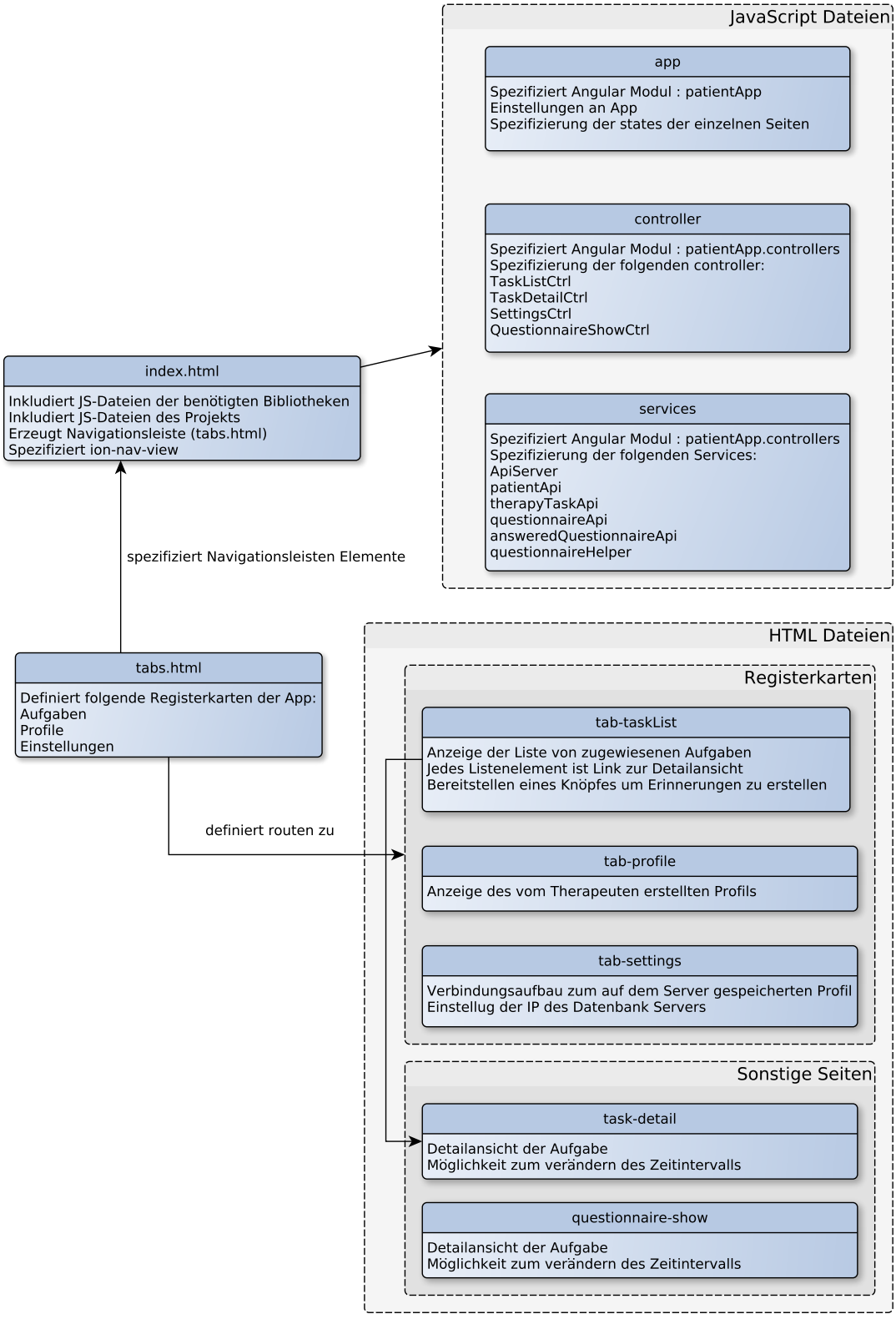

Abbildung 4.15: Übersicht der Architektur des Patienten Clients

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.

Die erste factory (APIServer) wurde erstellt um die Adresse des Datenbankservers System weit an einem zentralen Punkt speichern zu können. Alle Methoden welche mittels HTTP-Anfragen auf den Server zugreifen inkludieren diese factory und lesen aus dieser jene IP Adresse aus.

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 innerhalb der aufgerufenen Seite auf die übergebenen 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.

hier kann noch mehr zum Einstellungs Tab hin

4.3.1 Anzeige der Aufgaben

Um dem Patienten ein Übersicht über seine zugewiesenen Aufgaben geben zu können wurde die in Abbildung 4.16 gezeigte Seite eingerichtet. Diese beinhaltet neben der Übersicht über die Aufgaben Buttons zum erstellen einer Betriebssystem Erinnerungen.

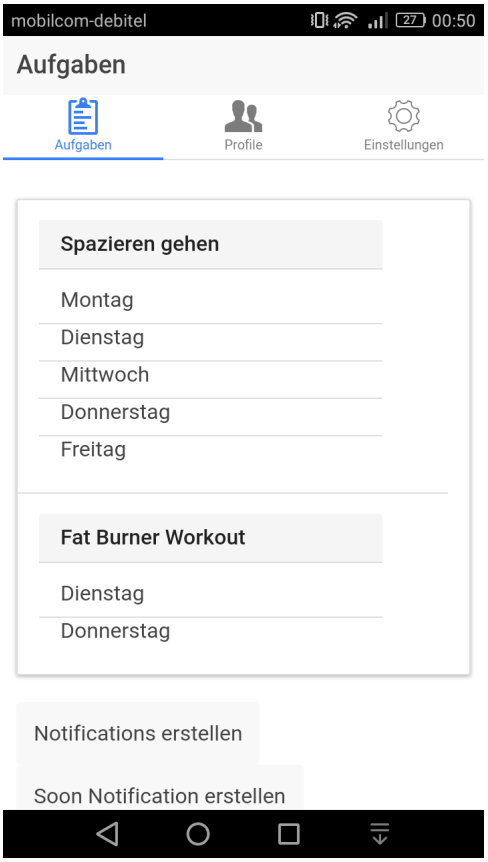

Abbildung 4.16: Ansicht der Aufgaben auf einem Android Smartphone

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 Services vom Server geladen, sofern diese nicht schon lokal vorhanden sind.

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

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

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, verschiedene 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.

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.

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"
        }
        });
};
```

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.

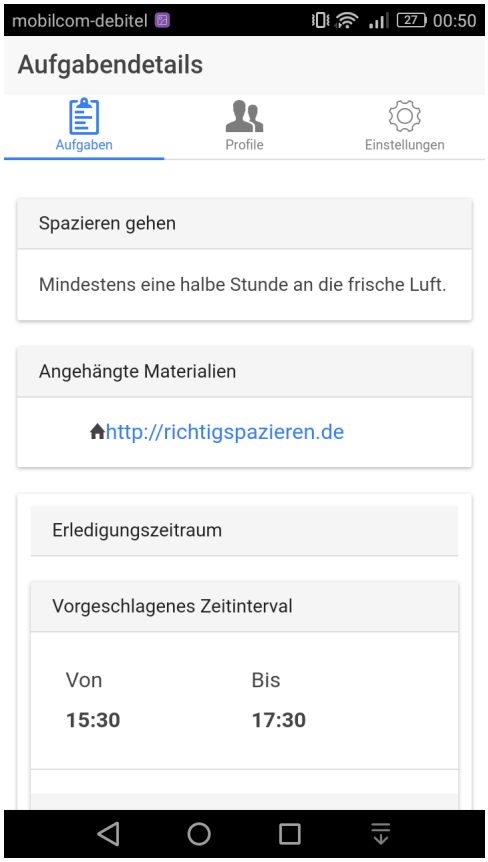

Abbildung 4.17: Name, Erläuterung und angehängte Materialien einer Therapeutischen Aufgabe

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 4.17 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.

Um dem Patienten die Möglichkeit zu bieten, das vom Therapeuten vorgeschlagene Zeitintervall zu verändern, wurde die in Abbildung 4.18 gezeigten Schaltflächen in diese Seite eingepflegt.

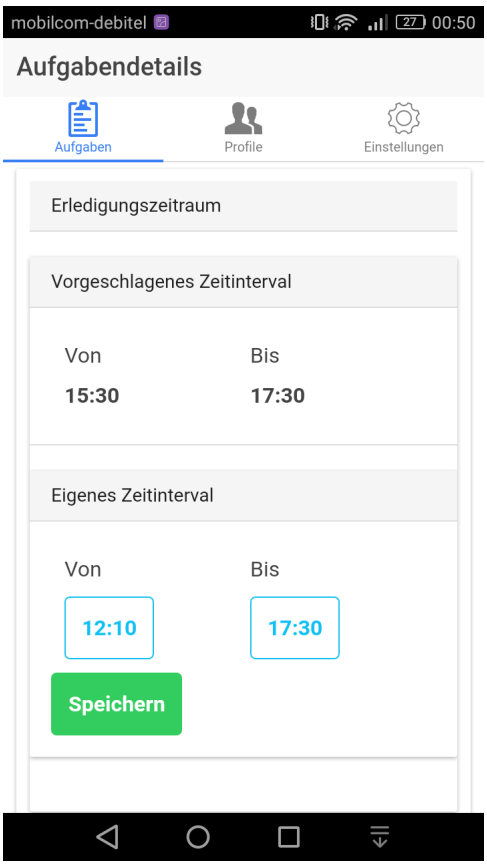

Abbildung 4.18: Übersicht über die Fragebögen

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 4.18 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 4.19 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.

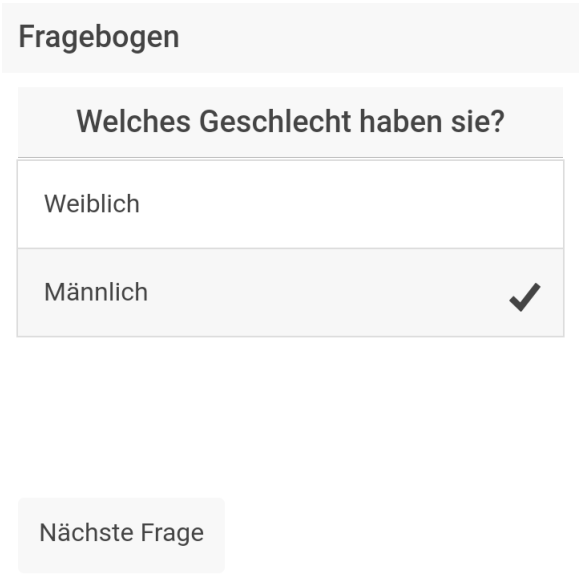

Abbildung 4.19: Fragebogenseite beim Fragetyp single

Frage vom Typ multi In der Abbildung 4.20 ist als Beispiel die Seite gezeigt, wenn der Aufgabentyp **multi** ist. Dieser Typ erlaubt es dem Endanwender mehrere der zur Verfügung stehenden Antworten auszuwählen.

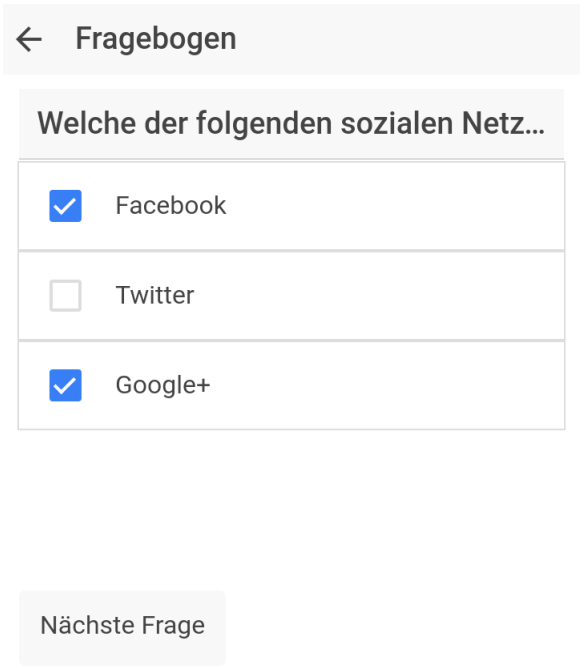

Abbildung 4.20: Fragebogenseite beim Fragetyp multi

Bilder größer und newpage ???

Frage vom Typ text Der in Abbildung 4.21 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 4.21: Fragebogenseite beim Fragetyp text

Frage vom Typ rating In Abbildung 4.22 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 4.22: 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** [8] 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 Anforderungsabgleich

Im folgenden Kapitel wird ein Anforderungsabgleich vorgenommen, dieser gibt Aufschluss darüber welcher der in Kapitel Anforderungsanalyse [3.2] spezifizierten Anforderungen schlussendlich implementiert wurden. Hierbei wird im einzelnen auf die jeweiligen Anforderungen eingegangen und Querverweise in das Kapitel Implementierung [4] hergestellt. Um die Reihenfolge von Kapitel 3.2 beizubehalten wird mit den Funktionalen Anforderungen begonnen, anschließend werden die Nicht-Funktionalen Anforderungen untersucht.

5.1 Funktionale Anforderungen

In diesem Kapitel werden die funktionalen Anforderungen untersucht. Sowohl an den Datenbank Server sowie für die beiden Clients.

5.1.1 Datenbank Server

Im folgenden Abschnitt werden die Anforderungen an den Datenbank Server aus Tabelle 3.2.2 untersucht.

Anforderung 1.1 - Datenpersistenz ✓ wurde wie in Kapitel 4.1.1 mittels einer MongoDB Datenbank auf einem Webserver umgesetzt, diese legt ihr Daten für den Nutzer nicht sichtbar auf dem Dateisystem ab, wo diese persistent gespeichert werden.

Anforderung 1.2 - Daten Sicherung X Dieses Feature wurde nicht Implementiert. Da bisher nicht mit realen Daten bearbeitet wurde war auch eine Sicherung dieser nicht nötig.

Anforderung 1.3 - Datenschnittstelle ✓ Wie in Kapitel 4.1.1 beschreiben, wurde mit Hilfe des **express** Frameworks eine REST-API implementiert welche HTTP anfragen entgegen nimmt. Hierdurch ist es Möglich, Daten aus der Datenbank anzufragen, zu verändern bzw. löschen so wie neue Datensätze anzulegen.

5.1.2 Therapeuten Client

Die Anforderungen an den Therapeuten Client **wurden alle erfüllt.**

Anforderung 2.1 - Patientenverwaltung ✓ Dieses Feature **wurde Implementiert** indem dem Therapeuten Client wie in Kapitel 4.2 beschreiben, eine Seite hinzugefügt wurde auf derer er eine Übersicht über seine Patienten hat. Hier können Patienten auch gelöscht werden - Anforderung 2.1.3.

Zusätzlich wurde wie in Kapitel 4.2 beschreiben eine Seite erzeugt mit welche der Therapeut Patienten anlegen bzw. editieren kann - Anforderungen 2.1.1 und 2.1.2.

Anforderung 2.2 - Aufgabenverwaltung ✓ Mittels der in Kapitel 4.2 beschriebenen Seiten ist es dem Therapeuten möglich die Vorlagen für Aufgaben / Übungen zu verwalten. Wie bei der Patientenverwaltung gibt es eine Übersicht in welcher einzelne Aufgaben ausgewählt oder gelöscht (Anforderung 2.2.3) werden können. Mittels der ebenso in Kapitel 4.2 beschriebenen Seite zum anlegen und editieren von Aufgaben werden den Anforderungen 2.2.1 und 2.2.2 abgedeckt.

Anforderung 2.3 - Fragebogenverwaltung ✓ Wie für die Verwaltung von Patienten und Aufgaben/Übungen wurde auch für die Fragebogenverwaltung eine eigene Seite Implementiert, welche in Kapitel 4.2 beschreiben wird. Diese bietet die Möglichkeit Fragebögen zu löschen(Anforderung 2.3.3). Die ebenfalls in Kapitel 4.2 beschriebene Seite zum erstellen und editieren von Fragebögen erfüllt die Anforderungen 2.3.1 und 2.3.2.

5.1.3 Aufgaben-/Übungsvorlagen

Die Anforderungen an die Aufgaben-/Übungsvorlagen wurden, wie in Tabelle 5.1 beschrieben, Teilweise erfüllt. A.1 bis A.3 sowie A.6 wurden wie in Kapitel 4.2 beschreiben erfüllt.

Nr.	Bezeichnung	Erfüllt?
A.1	**Name**	✓
A.2	**Beschreibung**	✓
A.3	**Materialien**	✓
A.4	**Kontrollmechanismen**	X
A.5	**Kategorisierung**	X
A.6	**Verschiedene Erledigungskontexte**	Es wurde nur der Zeitkontext implementiert

Tabelle 5.1: Anforderungsabgleich der Aufgaben-/Übungsvorlagen

5.1.4 Fragebögen

Die in Tabelle 3.4 gezeigten Anforderungen wurden abgesehen von Anforderung B.2 **erfüllt**. Anforderung B.2 wurde als nicht mehr wichtig erachtet, da der Name ausreicht, um einen Fragebogen zu identifizieren. Durch die grafische Darstellen mittels des in Kapitel 4.2 beschrieben Tools zur Erstellung/Bearbeitung von Fragebögen wird das wiederfinden eines bestimmten Fragebogens weiter vereinfacht.

Anforderung B.3 wurde erfüllt, indem es mittels des grafischen Tools möglich ist, die Fragen und Antworten nach belieben mit einander zu verbinden. Hierdurch kann der Therapeut frei wählen, welche Frage auf eine gegebene Antwort folgt.

Verschiedene Antwortmöglichkeiten wie in **Anforderung B.4** gefordert wurden wie in Kapitel 4.2 beschreiben mit Hilfe eines Zusatz Panels realisiert. Da dies aber nicht sehr intuitiv ist, sollte hier die Möglichkeit wahrgenommen werden, dass man in dem verwendeten Tool neue Elemente einbauen kann. Hierdurch wäre es Beispielsweise möglich zusätzliche Elemente für die einzelnen Fragetypen zu Verfügung zu stellen. Da in dem Tool auch auf Klicks reagiert werden kann, wäre zur Vereinfachung dieses Vorgangs auch ein Kontextmenü denkbar, in welchem man einem allgemeinen Frageelement einen Typ zuweisen kann.

Fragetypen Wie in der Tabelle 4.1 im Kapitel 4.2 zu sehen wurden die Anforderungen C.1 bis C.4 in vier verschiedenen Typen von Fragen realisiert.

Anforderung C.1 (Einzelantwort)
wurde als Typ **single** spezifiziert.

Anforderung C.2 (Mehrfachantwort)
wurde als Typ **multi** spezifiziert.

Anforderung C.3 (Bewertung)
wurde als Typ **text** bewertet.

Anforderung C.4 (Freitext)
wurde als Typ **rating** spezifiziert.

5.1.5 Patienten Client

Die Anforderungen 3.1 bis 3.7 aus Tabelle 3.2 wurden wie in Kapitel 4.3 beschrieben implementiert. Ein Überblick über die einzelnen Anforderungen mit Querverweisen in das Implementierungskapitel sind in Tabelle 5.2 zu finden.

Nr.	Anforderung	Erfüllt?	Beschreibung
3.1	User Identifikation	✓	4.3
3.2	Aufgabenanzeige	✓	4.3.1
3.3	Detailanzeige der Aufgabe	✓	4.3.1
3.4	Anzeige der Materialien	✓	4.3.1
3.5	Verändern des Erledigungskontext	✓	4.3.1
3.6	Erinnerung	✓	4.3.1
3.7	Fragebogen beantworten	✓	4.3.2

Tabelle 5.2: Anforderungsabgleich des Patienten Clients

5.2 Nicht-funktionale Anforderungen

Im folgen Kapitel werden die nicht-funktionalen Anforderungen beleuchtet.

5.2.1 Datenbank Server

Die nicht-funktionalen-Anforderungen des Datenbank Servers welche in Tabelle 3.2.2 aufgezeigt sind, werden im folgenden Abschnitt untersucht.

Anforderung 4.1 - kurze Reaktionszeiten ✓ Da der Server auf erprobten Technologien wie NodeJS und dem Express Framework beruht. Kann davon aus gegangen werden das diese auch für größere Nutzerzahlen geeignet sind. In der erstellten Testumgebung ist die Reaktionszeit der Anwendung nie negativ aufgefallen.

Anforderung 4.2 - Sicherheit X Da die Plattform bisher nicht mit realen Patientendaten verwendet wurde, war keine Notwendigkeit diese zu verschlüsseln. Bevor die Plattform mit realen Daten verwendet wird, müssen diese vorher vor dem Zugriff unbefugter geschützt werden. Hierzu sind im Kapitel 2.3 Anregungen gegeben.

5.2.2 Therapeuten / Patienten Client

Im folgen Absatz werden die in Tabelle 3.2.2 spezifizierten, nicht-funktionalen Anforderungen an die beiden Clients untersucht.

Anforderung 5.1 - Natives Design ✓ Da der **Therapeuten Client** in Form einer Website implementiert wurde, kann es für diesen kein Natives Design geben. Dieser hat dann aber den Vorteil, dass er auf jedem Gerät gleich aussieht. Einzig bei mobile Endgeräte, gemessen an der Displaygröße, wird das User-Interface der Anwendung angeglichen um kleinen Displays gerecht zu werden. Darauf, welches Betriebssystem zugrunde liegt wird nicht reagiert.

Das native Design des **Patienten Clients** wurde durch die Verwendung von Ionic weitgehendst umgesetzt. Die wichtigsten Bedienelemente werden durch Ionic an das jeweilige Betriebssystem angeglichen. Man sieht jedoch schon noch, dass die App im Herzen eine Website ist. Durch eine Überarbeitung des User-Interfaces kann das native Feeling der App extrem erhöht werden.

Anforderung 5.2 - vertretbare Reaktionszeiten ✓ Da für beide Clients sehr erprobte Technologien verwendet wurden und die Anwendungen auch keine großen Rechenleistungen benötigen wurde keine Untersuchungen zu den Reaktionszeiten vorgenommen. Während der Verwendung der Testumgebungen sind auch hier keine unzumutbaren Verzögerungen aufgefallen.

6 Zusammenfassung und Ausblick

Im Anschluss an die Theorie wird im folgenden Kapitel eine Zusammenfassung der Arbeit gegeben. An Anschluss daran wird ein Ausblick auf eine mögliche Weiterentwicklung der im Rahmen dieser Arbeit konzipierten und realisierten Plattform gegeben.

6.1 Zusammenfassung

Im Verlauf dieser Arbeit wurde eine Plattform zur Unterstützung von Therapeutischen Betreuungen konzipiert und realisiert. Diese kann in jedweder Therapeuten - Patienten Beziehung eingesetzt werden und soll den, für den Therapieerfolg, so wichtigen Transfer der Therapie in den Alltag erleichtern.

Der Fokus der Plattform liegt momentan zum einen auf den Aufgaben und Übungen welche der Patient Zuhause selbstständig erledigen soll und zum anderen auf Fragebögen welche vom Therapeut erstellt und vom Patient ausgefüllt werden können.

Zum einen soll mit dieser Plattform der Patient unterstützt werden indem ihm dabei geholfen wird seine Aufgabe/Übung wie vorgegeben zu erledigen. Da ein großes Problem von derartigen Therapien ist, dass die Aufgaben/Übungen nicht, oder nicht korrekt ausgeführt werden.

Zum anderen soll aber auch der Therapeut Informationen und Daten geliefert bekommen. Dieses Feedback kann wiederum dazu eingesetzt werden die Therapie sowie die gestellten Aufgaben/Übungen besser an den Patienten anzupassen. Des weiteren können die erhobenen Daten in der Wissenschaft verwendet werden um die Therapie Grundlegend zu verbessern.

Aus der Software Analyse ging hervor, dass ein geeigneter Weg zur Realisierung einer solchen Plattform ein Client für den Therapeuten in Form einer Webanwendung so wie einer Cross-Plattform-App für Android und iOS als Patienten Client ist. Der Daten Austausch dieser beiden Programme erfolgt über einen im Internet erreichbaren Server, welcher die Daten beider Seiten persistent in einer Datenbank speichert.

Die Webanwendung dient dem Therapeuten dazu die Patienten, Aufgaben-/Übungsvorlagen sowie Fragebögen zu verwalten. Diese können anschließend den jeweiligen Patienten zu bestimmten Zeiten und Wochentagen zugewiesen werden. An den so spezifizierten Zeitintervallen wird der Patient am Anfang des Zeitintervalls daran erinnert, dass er seine Aufgabe zu erledigen hat. Am Ende des Zeitintervalls hat er die Möglichkeit den von Therapeut angehängten Fragebogen auszufüllen.

Innerhalb der App muss der Patient zuerst Verbindung zu dem vom Therapeuten angelegten Profil ausnehmen. Anschließend hat er eine Übersicht über die ihm zugeteilten Aufgaben/Übungen. Über diese Ansicht gelangt der Nutzer auch zur Detailansicht einer Aufgabe in welcher deren Beschreibung und angehängte Materialien gezeigt werden.

Die meisten der Anforderungen welche in der Anforderungsanalyse 3.2 spezifiziert wurde, konnten im Zuge dieser Arbeit in das Konzept eingearbeitet werden. Aus dem Anforderungsabgleich 5 geht hervor, dass abgesehen vom Sicherheitsaspekt eine lauffähige Software entstanden ist. Was wiederum zeigt, dass es Möglich ist, die Therapeut - Patient Beziehung auf einem Zeitgemäßen Weg zu unterstützen. Auf eine Weise in welcher beide Seiten gleichermaßen dazu beitragen den Therapieerfolg zu erhöhen.

6.2 Ausblick

Der nächste Schritt bei der Weiterentwicklung dieser Plattform sollte in erster Linie einmal der Sicherheitsaspekt der kommunizierten Daten sein, noch bevor diese mit realen Patientendaten verwendet wird. Anschließend sollten die sonstigen Anforderungen welche im Anforderungsabgleich 5 als *nicht implementiert* markiert sind in die Plattform eingepflegt werden.

Hierzu gehören vor allem die Kontrollmechanismen da es für den Therapeuten oft schwierig ist, zu beurteilen ob eine Aufgabe/Übung wirklich gemacht wurde. Aus dem Grund, dass er nur das Feedback durch den Patienten hat, welche oft zum Lügen neigt. Dieser Umstand macht es dem Therapeuten schwer, zu beurteilen wie effektiv die gestellten Aufgaben wirklich sind.

Um die Aufgaben- und Übungsvorlagen global zu verbessern, sollte der Plattform eine Feature hinzugefügt werden, welches es erlaubt die gesammelten Daten im größeren Stil weiter zu verarbeiten. Momentan können die von den Patienten erzeugen Daten über die vom Datenbank Server bereitgestellte API abgerufen werden und liegen dann für die Weiterverarbeitung im JSON-Format zu Verfügung.

Hierzu wäre es Denkbar die Plattform auch zur Interaktion zwischen Therapeuten untereinander oder mit der Forschung zu verwenden. Es könnten sowohl die Aufgaben/Übungsvorlagen als auch die Fragebögen in einer Art Marktplatz angeboten werden. Dies setzt jedoch voraus, dass es dem Therapeuten möglich gemacht wird, Materialien auf die Plattform hochzuladen und somit individuellere Aufgaben-/Übungsvorlagen zu erstellen. Hierbei sollte nicht außer Acht gelassen werden, dass der Client des Patienten allerhand Peripherie und Sensoren bietet welche in die Gestaltung der Aufgaben und Übungen mit einfließen können sollten.

Des weiteren wäre es Denkbar, den Client des Therapeuten so zu erweitern, dass er in diesem seine komplette Verwaltung der Patienten erledigen kann. Andernfalls wäre es in den meisten Praxen wohl nötig die Patienten innerhalb dieser Plattform zusätzlich zu verwalten.

Im Fall, dass sich diese Plattform wirklich über Praxis und Forschung ausdehnt, wäre es auch durchaus Denkbar, dass sich ein *sich krankfühlender Mensch* den Client für den Patienten her-

unter lädt, seine Leiden schildert und noch bevor er den ersten Termin bei seinem Therapeuten in der Nähe hat schon Hilfe bekommt. In erster Linie voll Automatisch bis hin zur Verbindung mit dem Therapeuten seiner Wahl.

Neben der Weiterentwicklung der Plattform, ist es dringend Notwendig den Mehrwert dieser zu untersuchen. Im Zuge einer Studie sollte untersucht werden, ob dem Patienten durch seinen Client dabei geholfen wird,seine Aufgabe/Übung zu erledigen und ob die engere Bindung zwischen Therapeut und Patient zu einem besseren Therapieergebnis führt. Eine weitere Frage ist, in wie weit ein Patient bereit wäre seine erzeugten Daten mit der Wissenschaft zu Teilen. Hierbei fällt auch die von den Krankenkassen oft geforderte, bessere Einsicht in die Patientendaten ein. Durch Vergünstigungen beim Krankenkassenbeitrag könnte der Patient weiter Motiviert werden seine Aufgaben zu erledigen. Hierbei wäre es auch Denkbar, den Patienten spielerisch dazu zu bringen seine Aufgaben und Übungen zu machen oder dem Therapeuten das gewünschte Feedback zu bringen.

Literaturverzeichnis

[1] *BPMN.io*. https://bpmn.io/, . – zuletzt gesehen: 8.11.2016

[2] *Medium*. https://medium.com/@harrycheung/mobile-app-performance-redux-e512be94f9.dat95bdzb, . – zuletzt gesehen: 8.11.2016

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

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

[5] *SelfHTML, display*. https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Anzeige/display, . – zuletzt gesehen: 8.11.2016

[6] *Statista*. https://de.statista.com/statistik/daten/studie/198435/umfrage/marktanteile-der-smartphone-betriebssysteme-am-absatz-in-deutschland/, . – zuletzt gesehen: 8.11.2016

[7] *Windowsunited*. https://windowsunited.de/2016/02/27/erlaeuterung-was-genau-ist-xamarin-und-wieso-hat-microsoft-es-gekauft/, . – zuletzt gesehen: 8.11.2016

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

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

[10] 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