ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Konzeption und Realisierung eines generischen und hierarchischen Datenmodells für die interaktive Selektion von ortsabhängigen Merkmalen

Masterarbeitan der Universität Ulm

Vorgelegt von:
Benedikt Löffler
benedikt.loeffler@uni-ulm.de

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

Betreuer:
Marc Schickler

2016

Fassung 11. Mai 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Wir treffen täglich viele Entscheidungen. Viele sind von einander abhängig und beeinflussen sich gegenseitig. Schwierig wird es, wenn wir uns in der Thematik nicht auskennen oder sie einfach nicht verstehen. So kann die Sprache eine große Hürde darstellen. Um diese Entscheidungsfindung zu vereinfachen, ist eine grafische Aufbereitung und eine interaktive Bearbeitung hilfreich.

Diese Arbeit beschäftigt sich mit der Umsetzung eines Datenmodells, mit dem es möglich ist, die Entscheidungsfindung generisch in einer hierarchischen Struktur abzubilden. Sowie mit der Realisierung einer Anwendung, mit der diese Struktur interaktiv, durch Selektion von Merkmalen durchlaufen werden kann. Dabei werden zwei HyperMedia Elemente vorgestellt, die zur Steigerung der Interaktivität beitragen.

Danksagung

Ein besonderer Dank geht an Marc Schickler, für die Themenfindung und der Betreuung dieser Masterarbeit. Ebenso an Prof. Dr. Manfred Reichert für die Möglichkeit in seienem Institut diese Arbeit schreiben zu dürfen.

Sowie bei allen Weiteren die zum Gelingen dieser Arbeit beigetragen haben.

Inhaltsverzeichnis

1

Einführung

Täglich treffen wir etwa 20.000 Entscheidungen, so sagt es der Psychologe Prof. Dr. i.R. Ernst Pöppel aus München [Ret16]. Viele fallen unbewusst, sind alltäglich und unbedeutend, andere vielleicht von langfristiger Tragweite und wollen gut überlegt sein. Manchmal haben diese Entscheidungen einen komplexen Hintergrund, es bestehen Zusammenhänge, die miteinander verwoben oder sogar voneinander abhängig sind. Hinzu kommt, dass -wenn Ursache und Wirkung nicht klar ersichtlich sind- diese schwer zu verstehen sind. Insbesondere, wenn sie in einer uns neuen, unbekannten Thematik auftreten.

Gerade in der Informatik gilt es permanent Entscheidungen zu treffen. Die if-then-else-Programmierung steht dabei sicher als das geläufigste, wenn auch einfachstes Beispiel. Es ist also umso hilfreicher, wenn ein System Entscheidungen für uns automatisch erledigt, je schwieriger die Anforderung. Bei komplizierten Abläufen muss dieses System anhand bestimmter Kriterien ein Objekt zuerst automatisch erkennen und selbstständig korrekt klassifizieren.

Es gilt für jedes Kriterium eine oder gar mehrere Entscheidungen zu erstellen, um diese zu kategorisieren. Wenn diese Entscheidungen -wie in den meisten Fällen- voneinander abhängig sind und aufeinander aufbauen, ist es sinnvoll diese in einer Baumstruktur anzuordnen. So durchläuft das Objekt den Baum vom Stamm weg immer detaillierter bis am Ende des Astes die Klasse des Objektes exakt bestimmt ist.

Aus der Problematik heraus, komplexe und zusammenhängende Entscheidungen leichter und verständlicher darzustellen, ist diese Arbeit entstanden. Dafür wird die Struktur der Entscheidungsbäume verwendet. Zum besseren Verständnis werden die komplexen Abläufe, die auf Entscheidungen basieren, interaktiv visualisiert.

Es stellt sich die Frage, wie es denn möglich wäre, dass Migranten, die nach Deutschland einreisen ohne die Sprache zu verstehen, im Krankheitsfall selbstständig mittels eines Diagnosesystems die für sie notwendige fachärztliche Behandlung herausfinden können.

So entstand die Idee, diese Entscheidungsfindung grafisch zu visualisieren. Eine Applikation bildet den menschlichen Körper ab. Nun kann der Patient Schritt für Schritt auswählen, wo und welche Art von Beschwerden ihn plagen.

In einem weiteren Beispiel liesse sich der Entscheidungsbaum auch auf die Fehlersuche bei einem Kraftfahrzeug anwenden. Die moderne Bordelektronik verfügt über eine Vielzahl an vernetzten Sensoren, Aktoren und mikrocontrollergestützten Steuereinheiten. Ein Fehler muss deshalb nicht zwangsläufig an der Stelle aufgetreten sein, an der er sich für den Fahrer bemerkbar macht.

1.1 Zielsetzung

In dieser Arbeit wird die Konzeption und die Realisierung einer Anwendung beschrieben, in der es auf der einen Seite möglich ist, verschiedene Entscheidungsbäume zu erstellen. Diese sind generisch gehalten, sodass die unterschiedlichsten Entscheidungen abgebildet werden können. Zum anderen ist es möglich, dass die erstellen Entscheidungsbäume interaktiv visualisiert werden, sodass diese Entscheidungsbäume durchlaufen werden können. Dabei wird, der Ort des Merkmales, der für eine Entscheidung steht, berücksichtigt.

1.2 Aufbau der Arbeit

Diese Masterarbeit ist in sechs Kapitel untergliedert. In Kapitel 2 werden die verwendeten Technologien und Frameworks beschrieben. Im 3. Kapitel wird neben der Anforderungsanalyse, die Architektur und das Datenmodell erklärt, sowie das Konzept der späteren Implementierung. Das Kapitel 4 beschäftigt sich mit der Implementierung des Systems, dazu wird das zuvor vorgestellte Konzept und Datenmodell umgesetzt. Im 5. Kapitel

werden zwei Beispiele beschreiben, die mit dieser Anwendung umgesetzt wurden. Zum Abschluss wird die Masterarbeit zusammengefasst und einen Ausblick auf mögliche Erweiterungen der Anwendung gegeben.

2

Grundlagen

In diesem Kapitel werden die verwendeten Technologien beschrieben. Es werden die eingesetzten Frameworks kurz vorgestellt und allgemeine Grundlagen kurz erläutert.

2.1 Spring Framework

Die Serverimplementierung basiert auf dem Spring Framework. Spring ist ein quelloffenes Applikationsframework für die Java-Plattform. Das Framework besteht aus verschiedenen Applikationskomponenten zur Entwicklung von Webanwendungen basierend auf der Java EE Plattform [Inc16]. Dabei wird kein spezielles Programmiermodell verwendet, sondern es steht die Förderung von einfachen und guten Programmierpraktiken im Vordergrund. Vor allem, da es sich um eine gute Alternative zum Enterprise JavaBeans (EJB) Modells handelt, ist es in der Java Community sehr beliebt.

2.2 REST

Bei REST handelt es sich um ein Programmierparadigma für verteilte Systeme. Es wird vor allem bei Webservices oft eingesetzt [VB15]. REST stellt eine Abstraktion der Struktur und des Verhaltens des World Wide Web dar. Der größte Unterschied, im Vergleich zu anderen Maschine-zu-Maschine Kommunikationsmodellen, ist die einheitliche Schnittstelle für alle Ressourcen.

Der Architekturstil REST basiert auf sechs Prinzipien, deren Implementierung nicht weiter vorgegeben ist.

2.2.1 Client-Server

Wie bei den meisten Webservices, wird ein Client-Server-Modell verlangt, wobei der Server den Dienst bereitstellt, welchen der Client anfragt.

2.2.2 Zustandslosigkeit

Die Kommunikation bei REST ist zustandslos. Das heißt, der Server merkt sich keinen Zustand zwischen den Nachrichten. Jede Anfrage muss alle dazugehörigen Informationen beinhalten, um diese beantworten zu können.

Dies ist wichtig für die Skalierbarkeit der Anwendung. Der Server muss sich keine Informationen zu einem Zustand einer laufenden Kommunikation speichern. So könnte für Load Balancing mehre Server verwendet werden. Dabei kann jeder Server je Anfrage bearbeiten, da die Nachricht alle relevanten Informationen, die nötig sind für die Bearbeitung, beinhaltet.

Viele Webanwendungen umgehen diese Zustandslosigkeit über Cookies. Einige Webanwendungen werden wesentlich komfortabler mit den Einsatz von Cookies. Wenn die Anwendung einen geschützten Bereich besitzt, müsste sich der Benutzer bei jeder Anfrage erneut authentifizieren, also jedes Mal seine Anmeldedaten mitsenden. Mithilfe von Cookies kann eine Session aufgebaut werden. Dafür wird nach der einmaligen Authentifizierung ein Session Cookie gesetzt, welches aus einem möglichst langen zufälligen Wert, einer Session ID, besteht. Diese Session ID wird auf dem Server mit dem authentifizierten Benutzer verbunden. Der Client schickt jetzt bei jeder neuen Anfrage diese Session ID in der Anfrage mit. Der Server weiß anhand der Session ID, um welchen Benutzer es sich handelt. Dafür muss er sich aber die verbundenen Sessions merken und ist nicht mehr zustandslos.

In der Abbildung 2.1 wird die Kommunikation veranschaulicht. In der ersten Anfrage schickt der Client die Daten, die für die Authentifizierung nötigt sind. Das sind meisten ein Benutzernamen und ein Passwort. Der Server validiert diese Daten und generiert eine Session ID, diese sendet er als Cookie an den Client zurück. Das Cookie hat eine bestimmte Lebensdauer bis es vom Client gelöscht wird. Der grüne Balken symbolisiert

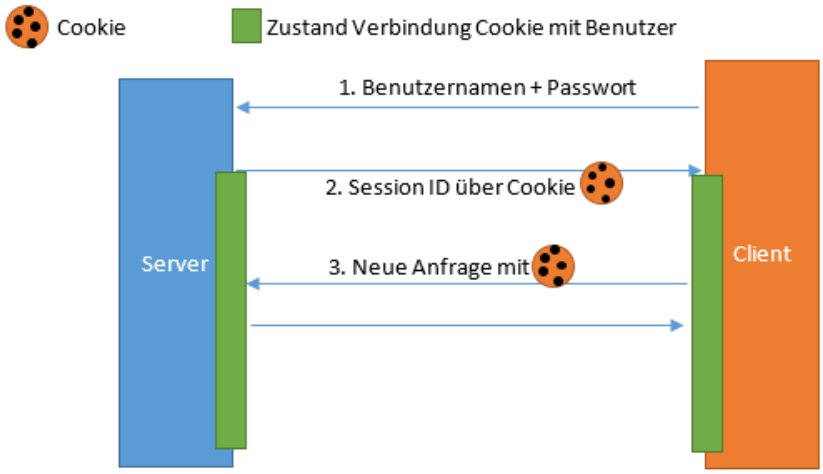

Abbildung 2.1: Ablauf der Kommunikation mit cookies

diese. Wenn der Client nun erneut eine Ressource anfragt, schickt er zur Authentifizierung nur sein Cookie mit. Der Server löst die Session ID wieder auf und weiß, welcher Benutzer diese Anfrage gestellt hat.

2.2.3 Caching

Beim Caching wird vermieden redundante Informationen zu übertragen. Beim Stellen einer Anfrage wird mitgesendet, welcher Version die Daten sind, die bereits schon mal nachgefragt wurden und noch zwischengespeichert sind. Der Server sendet die Daten nur dann, wenn sie sich verändert haben.

2.2.4 Einheitliche Schnittstelle

Das Hauptunterscheidungsmerkmal wird in vier Untereigenschaften unterteilt.

Adressierbarkeit von Ressourcen

Jede Ressource wird über eine eindeutige URI adressiert.

Repräsentationen zur Veränderung von Ressourcen

Die Ressourcen werden anderes gespeichert als sie dem Client präsentiert werden. Meistens in JSON oder XML. Jede Nachricht enthält genügend Informationen um die Ressource zu verändern oder zu löschen.

Selbstbeschreibende Nachrichten

Jede Nachricht sollte sich selber beschreiben, also wie sie verarbeitet werden muss. Dafür werden häufig die HTTP-Methoden verwendet.

"Hypermedia as the Engine of Application State"(HATEOAS)

Bei HATEOAS werden, bis auf den Einstiegspunkt, alle URLs zu den Ressourcen vom Server bereitgestellt.

So kann die Schnittstelle zu Ressourcen vom Server verändert werden, während bei SOAP ein fixes Interface bereitgestellt wird.

In jeder Antwort gibt es ein Feld *links* mit den URLs für die verknüpfe Ressourcen. Diese können bei nachfolgenden Anfragen direkt weiter verwendet werden.

2.2.5 Mehrschichtige Systeme

Für den Client versteckt, sollte hinter der Schnittstelle ein mehrschichtiges System stecken. So können die Skalierbarkeit und die Caching Möglichkeiten erhöht werden.

2.2.6 Code on Demand (optional)

Hierbei handelt es sich um ein optionales Prinzip.
Der Client sollte Code zur Ausführung je nach Bedarf dynamisch nachladen. Bei einer Webanwendung, wäre das zum Beispiel ein JavaScript Code der nachgeladen wird, wenn eine bestimmte Funktionalität benötigt wird.

2.2.7 CRUD

CRUD ist ein Akronym und steht für die elementaren Funktionen, die nötig sind um mit Ressourcen interagieren zu können.

Create, erstellen
Dient zum Erstellen einer neuen Ressource.

Read, lesen
Dient um eine bestehende Ressource abzufragen.

Update, aktualisieren
Dient um eine bestehende Ressource abzuändern und zu aktualisieren.

Delete, löschen
Dient um eine bestehende Ressource zu löschen.

Wie diese Funktionen implementiert werden, ist nicht spezifiziert. Die Implementierung ist von den Ressourcen und deren Technologie zum Speichern dieser Ressourcen abhängig. Oft wird bei REST hierfür HTTP verwendet, da dies die Funktionen implementiert und verwendet [Res16].

2.3 HTTP Methoden

Um mit einen REST Server zu kommunizieren wird oft das Hypertext Transfer Protocol (HTTP) verwendet. Es ist ein Protokoll für die zustandslose Übertragung von Daten. Es wird vor allem für das World Wide Web für die Übertragung von Internetseiten verwendet.

Das Protokoll arbeitet nach dem Request - Response Verfahren. Es wird eine Anfrage gestellt auf die dann eine Antwort folgt. Die Nachrichten besten aus einem Header und einem Body [Tut16]. Im Header werden Informationen über die Anfrage und Informationen über den Body bereitgestellt. Im Body befinden sich die Daten der Anfrage. Der Header und Body werden durch eine Leerzeile getrennt.

2.3.1 POST

POST sendet Daten an den Server. Sie dient als Create zum Erstellen neuer Ressourcen.

2.3.2 GET

Die GET Methode dient zum Abfragen einer bestehenden Ressource. Sie wird daher als READ verwendet.

2.3.3 PUT

Mit PUT können Daten für eine bereits bestehende Ressource gesendet werden. Sie wird daher als Update verwendet.

2.3.4 DELETE

Über die DELETE Methode kann eine bestehende Ressource gelöscht werden. Sie ist dieselbe wie bei den CRUD Funktionen.

```
1 POST /Resource HTTP/1.1
2 Host: domain.de
3 Content-Type: application/json
4 Content-Length: 40
5
6 {"Name":"Benedikt","Matrikelnummer":123}
```

Listing 2.1: Beispiel für eine HTTP POST Anfrage.

Bei einer HTTP Anfrage wird als erstes die verwendete Methode genannt, danach für welche Ressource die Anfrage gilt und als letztes, in der ersten Zeile, wird die verwendete Version angegeben. In der zweiten Zeile wird der Host angeben, da auf einen Server mehrere Domains betrieben werden können. Es muss hier angegeben

werden, für welchen Host die Anfrage gilt. Danach wird angegeben, in welchem Format die angebenden Daten im Body kodiert sind und in der letzten Zeile des Headers wird noch die Länge des Bodys angeben. Im Body steht der Inhalt, welcher an die Ressource gesendet wird. Im Listing 2.1 ist eine Beispiel Nachricht für eine HTTP POST Request abgebildet.

2.4 JSON

JSON (JavaScript Object Notation) ist ein schlankes und einfaches Datenformat für den Austausch von Daten. Es können Daten wie Variablen, Objekte und Arrays in lesbarer Form gespeichert und übertragen werden. Es ist unabhängig von der Programmiersprache, wird aber meistens im Webbereich verwendet, vor allem in Verbindung mit JavaScript und Ajax.

In JavaScript gibt es Funktionen um aus Objekten JSON zu erstellen und auch zurück zu konvertieren.

Mit JSON.stringify() wird ein JavaScript-Objekt in einen JSON String konvertiert. Mit der Methode JSON.parse() ein JSON String zurück in ein JavaScript Objekt.

Die Vorteile von JSON gegenüber von XML ist, dass es einfacher gehalten ist, was die Lesbarkeit erleichtert. Zudem benötigt es weniger Übertragungsvolumen und es kann direkt in JavaScirpt Objekt geparst werden. Dafür können aber keine komplizierten Auszeichnungen abgebildet werden und auch nicht gegen ein spezifisches Schema validiert werden.

```
1 {"Name":"Benedikt","Matrikelnummer":123}
```

Listing 2.2: Beispiel für einen JSON String.

Wird der hier angegebene String 2.2 über die Funktion JSON.parse() in ein Objekt konvertiert, hat dieses ein Attribut ”Name”, welcher den Wert ”*Benedikt*” hat und ein Attribut ”Matrikelnummer” mit dem Wert ”*123*”.

2.5 JavaScript

JavaScript ist eine Skriptsprache. Sie dient unter anderen zum dynamischen Manipulieren des Document Object Model (DOM). So kann eine höhere Interaktivität in normalerweise statische Webseiten gebracht werden. So können zum Beispiel Formulareingaben überprüft und validiert werden, ohne dass dieses Formular abgesendet werden muss. Oder es kann dynamisch Inhalt nachgeladen werden, sodass ein ”unendliches” Scrollen auf einer Seite ermöglicht wird.

Durch ein JavaScript kann überprüft werden, ob der Benutzer zum Seitenende gescrollt hat. Wenn das der Fall ist, wird eine Ajax-Anfrage an den Server gesendet, ohne die Seite neu zu laden. Die Antwort enthält den neuen Inhalt, welcher dynamisch in die Seite eingebunden wird. So kann der Benutzer einfach weiter scrollen und muss nicht eine extra Seite für den Inhalt laden.

Heute wird JavaScript nicht mehr nur eingesetzt um Webseiten dynamischer und interaktiver zu machen, sondern in verschiedenen Bereichen, wie zum Beispiel zur Programmierung von Webservern über NodeJS oder sogar zum Programmieren von Mikrocontrollern.

Durch JavaScript wurde es möglich Logik auch auf der Clientseite auszuführen. Über die Zeit verbreitete sich JavaScript immer weiter und wurde auch immer leistungsstärker, sodass es heute in modernen Webanwendungen ein wichtiger und zentraler Bestandteil ist.

2.6 AngularJS

Bei AngularJS handelt es sich um JavaScript MVC Framework, welches von der Firma Google entwickelt wurde [Goo16]. Es läuft Clientseitig und dient zur Erstellung von sogenannten Single Page Applications. Zu den Haupt-Features zählen unter anderem [TB16] :

2.6.1 Two-Way Data Binding

Über dieses Feature kann zum Beispiel eine Eingabe aus einen Formularfeld direkt mit einer Variablen im Controller gekoppelt werden. Wird der Wert im Formular verändert, wird diese sofort im Controller aktualisiert. Wird die Variable im Controller verändert, wird der neue Wert wiederum sofort auf dem Frontend dargestellt.

2.6.2 Ability to extend HTML

Es gibt die Möglichkeit durch sogenannte Direktiven neue HTML TAGs zu erstellen. Daraus ergibt sich eine schönerer HTML Code in dem nicht nur Dutzende DIV Elemente vorkommen.

2.6.3 Abstraction of Lower-Level DOM operations

Durch Angular ist es nicht mehr nötig auf das Lower-Level DOM direkt zuzugreifen, um dieses zu manipulieren.

2.6.4 Good testability

Durch die schwache Kopplung können die einzelnen Komponenten gut getestet werden.

2.6.5 Scope

Der Scope wird gerne als Kleber zwischen Controller und View bezeichnet. Es stellt die Verbindung für die Kommunikation zwischen Controller und View bereit, dies erfolgt über das Two-Way Data Binding. Wird eine Variable in einem Controller in das Scope Objekt gespeichert, wird diese in der View direkt aktualisiert. Es kann auch ein HTML Eingabefeld an eine Variable im Scope gebunden werden. Wird der Wert in diesem Eingabefeld geändert, kann dieser im Controller über Scope abgefragt werden.

```
1 $scope.foobar = "Hallo World";
```

Listing 2.3: In AngularJS eine Variable in den Scope speichern.

```
1 <htmlTag> {{foobar}} </htmlTag>
```

Listing 2.4: In AngularJS eine Variable im Template anzeigen.

2.7 Clientseitige Webanwendungen

Die meisten klassischen Webanwendungen sind Serverbasiert, das heißt die Hauptlast wird auf dem Server verarbeitet. Der Client stellt eine Anfrage, welche der Server verarbeitet. Er macht die entsprechenden Berechnungen und Datenbankabfragen und generiert daraus eine dynamische HTML-Seite, die er an den Client zurückschickt. Der Client braucht diese nur noch darstellen.

Durch den Einsatz von zum Beispiel JavaScript kann die Last auf dem Server reduziert werden und auf den Client verlagert. Eine Form von Clientseitigen Webanwendungen sind Single Page Applications.

In der Abbildung 2.2 wird veranschaulicht, welche Möglichkeiten es gibt, um die Last zu verteilen. Die *dezentrale Präsentation* wird bei klassischen Webseiten angewendet, der Server kümmert sich um die Datenhaltung und die Verarbeitung beziehungsweise um die Aufbereitung der Daten und sendet diese zum Client, der nur für die Präsentation zuständig ist. Die *verteilte Verarbeitung* wird bei Webanwendungen eingesetzt. Es geschieht eine Vorverarbeitung auf dem Server, der Client trägt aber beispielsweise über JavaScript ebenfalls zur Verarbeitung bei. Die *dezentrale Verarbeitung* entspricht dem Modell der Single Page Applikationen. Der Server ist nur noch für die Datenhaltung zuständig, die Verarbeitung und die Präsentation übernimmt der Client. Das letzte Modell der *verteilten Datenhaltung* wird selten bei Webseiten eingesetzt. Da die Datenhaltung ebenfalls auf der Clientseite stattfindet. Diese würde einer webbasierten Smartphone App entsprechen, bei dem nur spezielle Daten vom Server nachgeladen, beziehungsweise synchronisiert werden.

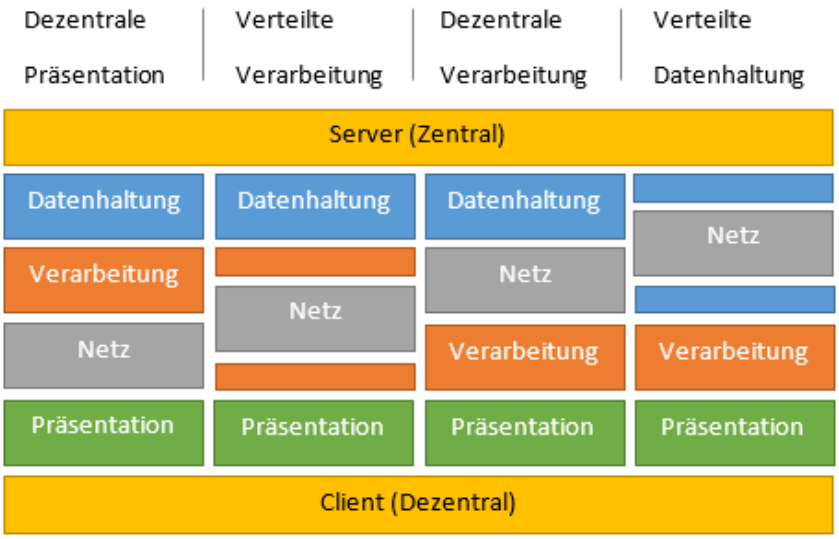

Abbildung 2.2: Möglichkeiten der Aufgabenverteilung zwischen Client und Server

2.8 Single Page Application

Bei klassischen Webanwendungen wird beim Klick auf einen Link ein HTTP Request an den Server gesendet, der mit der entsprechenden Response antwortet. Die Response enthält im Body meistens den kompletten HTML Code, welcher erst gerendert werden kann, wenn der komplette Code samt Style Sheets übertragen wurde.

Bei Single Page Applications wird beim ersten Aufruf die komplette Seite geladen. Bei einem Klick auf einen Link wird der Content dynamisch geändert, ohne die Seite neu zu laden. Gegebenenfalls werden über einen asynchronen Ajax Request Daten von zum Beispiel einem REST Server geladen.

2.8.1 Unterschiede zwischen Single Page Application und klassischen Webseiten

Der größte Unterschied ist, dass der Präsentationsfluss bei Single Page Application nicht unterbrochen wird, da ein komplettes Neuladen der Seite nie stattfindet. Dadurch kann eine höhere Interaktivität geboten werden. So können Desktop Anwendungen in

Browsern nachgebildet werden, die sich auch wie native Programme verhalten und auch optisch sich an diesen orientieren. Dabei kann der Vorteil der Crossplatform Webbrowser ausgenutzt werden um Anwendungen auf unterschiedlichen Plattformen anbieten zu können [Sco15].

Zudem wird der Server entlastet. Zum einen entfällt das Zusammensetzen der Webseite und zum anderen müssen nicht mehr bei jedem Request abhängige Ressourcen wie zum Beispiel Style Sheets, JavaScript Dateien oder Bilder übertragen werden. Zudem wird der Server enorm dadurch entlastet, das er zustandslos ist, im Sinne des Ursprungsgedankens von HTTP, da der Zustand im Client gehalten wird.

Durch die entstandene Trennung von Präsentation- und Datenschicht, findet eine Dezentralisierung statt. Der Webserver stellt nur noch Daten bereit und könnte relativ einfach ausgetauscht werden.

Natürlich haben Single Page Applications auch Nachteile. Dadurch, dass keine klassischen Links aufgerufen werden, ändert sich die URL nicht mehr. Bei klassischen Links wird eine neue URL aufgerufen und so ein neuer Request an den Server gesendet. Bei Single Page Applications wird meistens bei den Klick auf einem Link ein JavaScript ausgeführt, das den Request an den Server sendet und die Antwort dynamisch in den DOM einpflegt. Durch die gleichbleibende URL ergeben sich folgende Effekte:

- Es gibt keine History mehr. Ein Klick auf den Browser ”Zurück”-Button lädt die zuvor besuchte Seite.
- Es können keine Lesezeichen (Bookmarks) für einen bestimmten Zustand gesetzt werden. Da immer die Startseite der Single Page Application geladen werden würde.
- Das gleiche gilt für ein Neuladen der Seite (reload). Hierbei würde ebenfalls der Startzustant der Single Page Application geladen , da die URL auf diesen zeigt.
- Dadurch, dass der Zustand Clientseitig gehalten wird, geht dieser bei einem Neuladen der Seite verloren.

AngularJS löst dieses Problem mit dem Modul ngRoute. Über dieses Modul können Routen zu bestimmten Controller definiert werden. Dies geschieht indem das Hash-

Fragment, wie im Listing 2.5, der URL dafür verwendet wird. Dieses Modul erzeugt zudem Historyeinträge im Browser, sodass der Zurückbutton wieder verwendet werden kann.

```
1 http://domain/#/tree/1
```

Listing 2.5: Beispiel URL mit Hash-Fragment.

So können auch Lesezeichen zu den einzelnen Seiten gesetzt werden. Dabei muss beachtet werden, dass hierdurch der Benutzer eventuell in einen nicht definierten Zustand landet, da rt inmitten der Applikation startet und nicht den definierten Zustand über den Startcontroller geht.

2.9 Bootstrap

Bootstrap ist ein HTML Frontend Framework, welches von Twitter entwickelt und als OpenSource Projekt veröffentlicht wurde. Das Framework beinhaltet HTML, CSS und JavaScript Komponenten zum schnellen, einheitlichen und flexiblen Gestalten von Webanwendungen [Spu13].

Der Inhalt wird bei Bootstrap Anwendungen auf eine Grid System aufgebaut. Dieses besteht aus zwölf Spalten, die individuell genutzt werden können [Twi16].

Es kann angegeben werden, wie viele Spalten ein Div nutzen soll. Der Inhalt wird von Bootstrap je nach verfügbarem Platz organisiert.

2.9.1 Responsive Webdesign

Seit der Version 2.0 des Bootstrap Frameworks, wird Responsive Webdesgin unterstützt. Bei Responsive Webdesign handelt sich um die Gestaltung der Webseite und Organisierung des Inhaltes unter der Berücksichtigung von Eigenschaften des Endgerätes [Fra12]. Eigenschaften, wie zum Beispiel die Bildschirmauflösung, Gerätetype wie Desktop Computer, Smartphone oder Tablet oder auch Eingabemöglichkeiten wie per Touchscreen oder Maus.

Abbildung 2.3: Darstellung der Anwendung auf verschieden Displaygrößen.

2.10 Entscheidungsbaum

Entscheidungsbäume sind in der Informatik weitverbreitet. Sie werden unter anderem zur automatischen Klassifikation eingesetzt oder zum Abbilden formaler Regeln [YS95] [JR08]. Für die automatischen Klassifikation wird oft der ID3 Algorithmus eingesetzt [Qui86].

Entscheidungsbäume bestehen aus geordneten und gerichteten Bäumen. Ein Knoten beinhaltet einen Test zu einem Attribut oder mehreren Attribute des zum Beispiel zu klassifizierenden Objekts. Ein Blatt in einem Entscheidungsbaum enthält eine Entscheidung beziehungsweise die Klasse des Objekts.

Beim durchlaufen des Entscheidungsbaums wird in jeden Knoten ein Test vollzogen. Anhand diesen Ergebnis wird entschieden in welchen Kindsknoten weitergegangen wird. In diesen Kindsknoten wird wieder ein Test durchgeführt, woraufhin wieder tiefer durch den Baum gegangen wird [SLP11].

Wird ein Blatt erreicht, dann wurde das Objekt klassifiziert beziehungsweise wurde eine Entscheidung getroffen.

2.10.1 Related Work

Im Internet sind diverse ”*Symptom Checker*” Webanwendungen auffindbar, die eine ähnliche Funktion bieten. Doch diese Anwendungen beziehen sich alle auf medizinische Diagnosen. Der Aufbau dieser Anwendung ist bei allen recht ähnlich. Der interaktive Part der Anwendung bezieht sich oft nur auf die erste Selektion. Meistens kann nach der Wahl, ob Mann oder Frau auf einer Grafik die Körperregion selektiert werden zu dem weitere Symptome näher beschrieben werden. Diese Symptom-Auswahl erfolgt aber fast immer durch Multiple-Choice bzw. Dropdown-Elemente.

Die Anwendungen sind alle Webanwendungen die klassisch mit HTML aufgebaut sind und teilweise von JavaScript für die Interaktivität unterstützt werden. Es werden alle Daten unverschlüsselt über HTTP übertragen. Bei sehr persönlichen und intimen Fragen sollte der Datenschutz bedacht werden.

Gerade im medizinischen Bereich Diagnos zu erstellen ist sehr riskant, da Fehldiagnosen fatale Folgen haben können. Wie ein Bericht im British Medical Journal zeigt, gab es nur bei 34 % der eingegebenden Symptome die richtige Diagnose [Sü15].

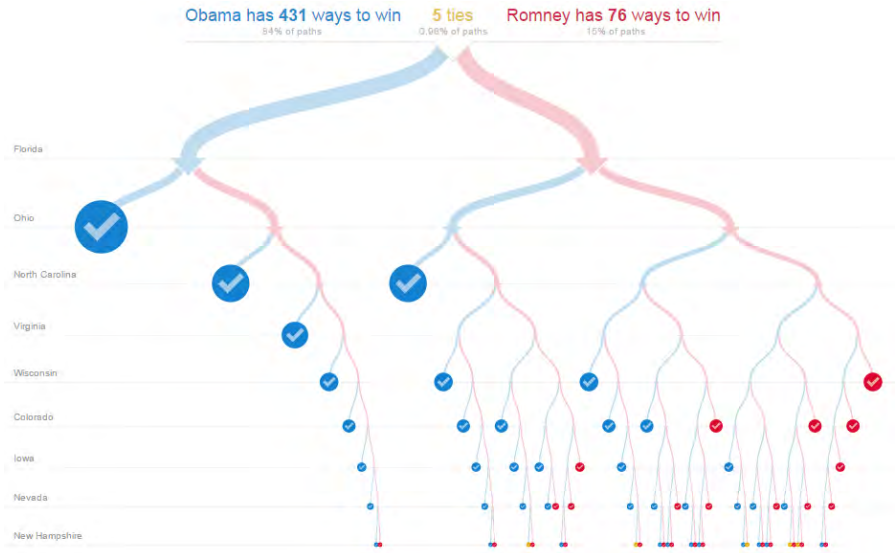

Abbildung 2.4: US Präsidentschaftswahlen In Jahre 2012 [The16]

Es gibt ebenso einige ”*interacitve decision trees*”. Ein sehr schön gestalteter und interaktiver Entscheidungsbaum ist auf der Internetseite der *New York Times* [The16] zu finden. Dieser veranschaulicht die Präsidentschaftswahlen 2012 in den USA. Über diesen Entscheidungsbaum wird dargestellt, wer Präsident wird, Barack Obama oder Mitt Romney, je nachdem, wer in welchen Bundesstaat die Vorwahlen gewinnt. Dieser Entscheidungsbaum ist aber nur für diesen Anwendungszweck, also nicht generisch. Der Entscheidungsbaum ist in Abbildung 2.4 zu sehen.

Ein Knoten steht für eine Vorwahl. Zu jeden dieser Knoten gibt es immer zwei Entscheidungen, Obama oder Romney. Die Blätter in diesem Entscheidungsbaum sind Wahlentscheidungen und repräsentieren, wer, in diesem Fall die Präsidentschaftswahl, gewinnt.

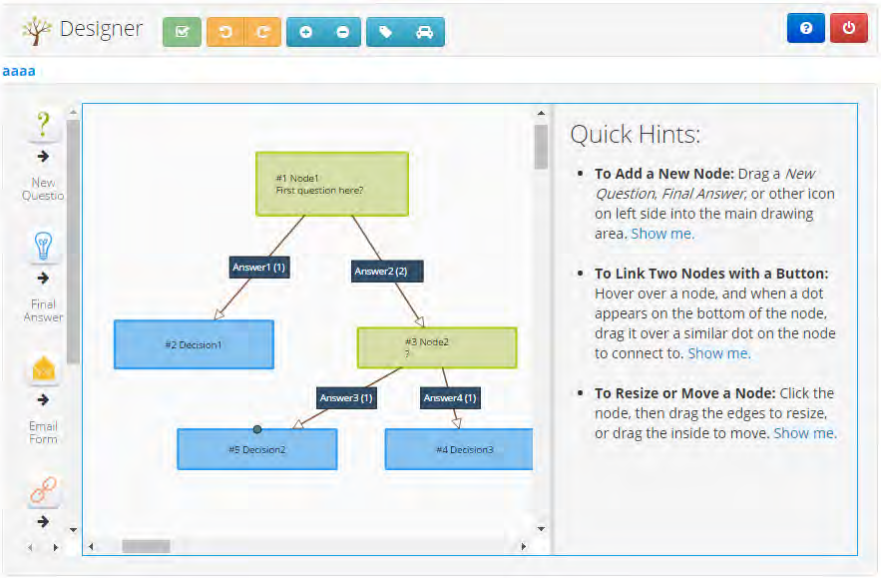

Abbildung 2.5: Grafischer Editor für Entscheidungsbäume [Zin16]

Eine weitere Anwendung, zum Erstellen von Entscheidungsbäumen, ist Zingtree [Zin16]. Hier können über einen interaktiven Designer Entscheidungsbäume grafisch erstellt werden, wie in der Abbildung 2.5 zu sehen ist. In dieser Anwendung können generischer Entscheidungsbäume erstellt werden. Beim erstellen einer neuen Node muss eine textuelle Frage gestellt werden. Zu dieser Frage kann, über einen einfachen WYSIWYG-Editor

[1], detaillierter beschrieben und gestaltet werden. Hier können auch Bilder hinzugefügt werden. Die Antworten können nur aus einfachen Textzeilen bestehen. Zudem gibt es noch ein weiteres Element ”Solution”, also Lösung. Eine Antwort kann mit einer Lösung verbunden werden.

Die Datenstruktur verwendet die Fragen als Nodes. Diese Nodes werden über die jeweiligen Antworten miteinander verbunden. Eine Antwort kann auch mit einer Lösung verbunden werden. Diese dient als Blatt in dem Entscheidungsbaum.

In Zingtree können zwar interaktive und generische Entscheidungsbäume erstellt, aber die Frage, Antwort und Lösung kann nur aus Text bestehen. Dies schränkt die Interaktivität stark ein. Das Problem für die Migranten, woraus diese Arbeit entstanden ist, kann mit dieser Anwendung nicht gelöst werden.

[1] WYSIWYG, ist ein Akronym und steht für ”What You See Is What You Get”. Das bedeutet der Text wird im Editor genau so dargestellt, wie er später Ausgeben wird.

3

Entwurf

In diesem Kapitel wird der Entwurf der Anwendung beschrieben. Zuerst werden die Anforderungen definiert. Über die Softwareanalyse wird untersucht mit welchen Techniken die Anwendung umgesetzt werden kann. In Kapitel 3.5 wird das Datenmodell beschrieben, auf das der Server aufbaut.

In Kapitel 3.4.2 wird auf die Umsetzung des Servers eingegangen und in 3.4.1 auf die des Clients. Das Konzept, welches später in Kapitel 4 implementiert wird, wird in Kapitel 3.8 erklärt.

3.1 Vision

Das entwickelte System sollte zum einen möglichst generisch sein, also möglichst wenig feste Datentypen haben und auch keine unveränderlichen Abläufe. Es sollte möglich sein, dass einfach andere Medien Typen verwendet werden können.

Zum anderen soll das System einen hierarchischen Ablauf haben, also soll es möglich sein, eine Baumstruktur abbilden zu können.

Des Weiteren soll es möglich sein, auf der Frontend Seite eine möglichst interaktive Bedienung bieten zu können, Möglichst ohne Unterbrechung des Präsentationsflusses, so dass der Eindruck eines nativen Programmes entsteht.

3.2 Anforderungsanalyse

Aus der Vision 3.1 können folgende Anforderungen formuliert werden.

3.2.1 Funktionale Anforderungen

In der Tabelle 3.1 werden die funktionalen Anforderungen aufgelistet.

Anforderung	Beschreibung
Baumstruktur	Die Entscheidungen sollen in einer Baumstruktur gespeichert werden
Generisch	Die Entscheidungen sollen generischen sein und unterscheide Typen von Nodes speichern
Hierarchisch	Die Entscheidungen sollen hierarchisch aufeinander aufbauen
Ortsabhängig	Die Auswahl der Entscheidungen soll ortsabhängig vom Merkmal sein

Tabelle 3.1: Funktionale Anforderungen

3.2.2 Nicht-Funktionale Anforderungen

In der Tabelle 3.2 werden die nicht funktionalen Anforderungen aufgelistet.

Anforderung	Beschreibung
Plattform unabhängig	Die Anwendung soll auf unterschiedlichen Plattformen verwendbar sein
Präsentationsfluss	Die Anwendung soll einen einheitlichen Präsentationsfluss haben, der nicht unterbrochen wird

Tabelle 3.2: Nicht-Funktionale Anforderungen

3.3 Architektur

Die Architektur, die sich aus den Anforderungen ergibt, entspricht einer klassischen Architektur 3.1, so wie sie bei den meisten Webanwendungen verwendet wird. Wobei hier der Fokus auf einer Single Page Application liegt, welche die Last mehr auf den Client verlagert und den Server entlastet. Dabei stellt der Server nur die Daten über eine REST API bereit, der Client kümmert sich um die Präsentation und die Zustandshaltung.

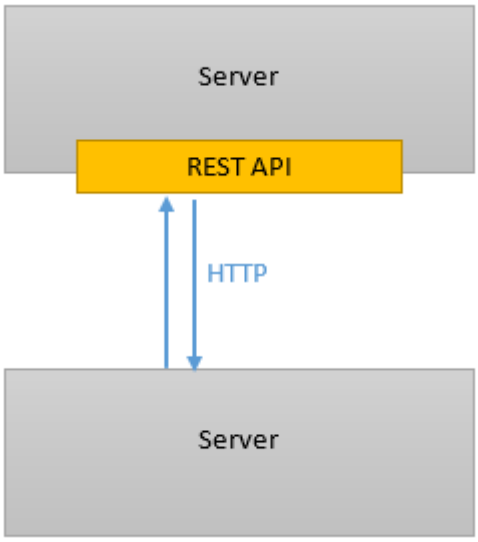

Abbildung 3.1: Einfache Architektur der Anwendung

Der Server sollte recht einfach gehalten werden, er soll nur für die Datenhaltung zuständig sein. Wie und wo die Daten gespeichert werden, dafür ist der Server selbst zuständig. Die Kommunikation für den Austausch von Daten soll über eine API erfolgen.

3.4 Softwareanalyse

In dem nachfolgenden Kapitel wird analysiert, welche Software für die Umsetzung in Frage kommt.

3.4.1 Client

Um sich bei der Erstellung der Client Anwendung unnötige Schreibarbeit zu sparen und einen möglichst komfortablen Umgang mit der REST API zu haben, bietet sich es an, auf ein Framework zurückzugreifen.

Bei der Erstellung wurde das AngularJS Framework von Google verwendet, da es sehr weit verbreitet ist und es eine große Community gibt, woraus sich ein recht großen Dokumentations- und Tutorial-Umfang ergibt.

3.4.2 Server

Bei der Erstellung des Servers gibt es ebenfalls sehr viele Möglichkeiten zur Realisierung. Es bietet sich an einen REST Server zu verwenden, da dieser die Anforderungen erfüllt und oft im Web Bereich verwendet wird.

Dreamfactory.com

Dreamfactory ist ein Service Provider über den es möglich ist, eine REST API zu bereits bestehenden Datenbanken zu erstellen. Dazu muss die Datenbank bereits existieren. Bei Änderungen an der Datenbank muss die API jedes Mal entsprechend angepasst werden. Dreamfactory kann als SaaS (Software as a Service) direkt online genutzt werden oder auch auf einem eigenen Server betrieben werden.

Die Onlineversion kam für dieses Projekt nicht infrage, da so eine Abhängigkeit zu Dreamfactory entstanden wäre und die Account Verwaltung ebenfalls problematisch ist. Welche E-Mail Adresse verwenden? Wer verwaltet das Passwort? Für die eigene Installation würde ein extra Server benötigt werden. Zudem muss die Datenbank nach wie vor manuell erstellt und verwaltet werden.

MEAN

Eine weitere Möglichkeit den Server zu erstellen, wäre den MEAN Stack zu verwenden [Hol15] [Hav14]. MEAN steht für die Anfangsbuchstaben von:

MongoDB
Eine weit verbreitete NoSQL Datenbank.

Express.js
Ein Framework das auf der Serverseite bei der Entwicklung von Webanwendungen unterstützt. Des Weiteren kann mit Hilfe des Frameworks eine REST API aufgebaut werden.

AngularJS
Ein Framework um Clientseitig Single Page Applications zu erstellen

Node.js
Ein schneller und einfacher Webserver.

Der Vorteil des MEAN Stacks ist, dass auf allen Ebenen, von der Clientseite bis hin zum Server die Kommunikation mit der Datenbank, mit JavaScript programmiert werden kann.

Da die MongoDB eine NoSQL Datenbank ist, müssen auch keine Tabellen mit festen Strukturen von Hand erstellt werden. Aber es ließen sich im Datenmodell keine Rekursionen abbilden, wie es in einer Baumstruktur vorkommt. (Ein Knoten hat wiederum Knoten als Kinder) Also kam der MEAN Stack ebenfalls nicht infrage.

Java Persistence API

In Java gibt es die Möglichkeit, über die Java Persistence API, sogenannte Entitys bestehend aus Plan old Java Objects (POJO) auf einer Datenbank abbilden zu lassen. Hierbei übernimmt die Persistence API die komplette Verwaltung und Kommunikation mit der Datenbank.

Der Server hätte auch in JavaEE geschrieben werden können, doch aus Erfahrungswerten aus früheren Projekten und Vorlesungen ergab sich, dass zu diesen ein relative hoher Konfigurationsaufwand mit diversen XML-Dateien gehört.

Aus diesen zuvor aufgeführten Gründen ergibt sich folgendes Ergebnis und schlussendlich auch die verfolgte Lösung. Der Server ist mit dem Spring Framework entstanden, bietet nach außen eine REST API an, für die Kommunikation mit den Client nach innen, werden die Daten über die Java Persistence API auf die in Spring integrierte H2 Datenbank abgebildet. Um die Erstellung und Verwaltung kümmert sich das Spring Framework automatisch. Es müssen lediglich die entsprechenden Entitys angelegt werden und über ein Repository Interfaces gekennzeichnet werden, dass zu diesen eine REST API generiert werden soll.

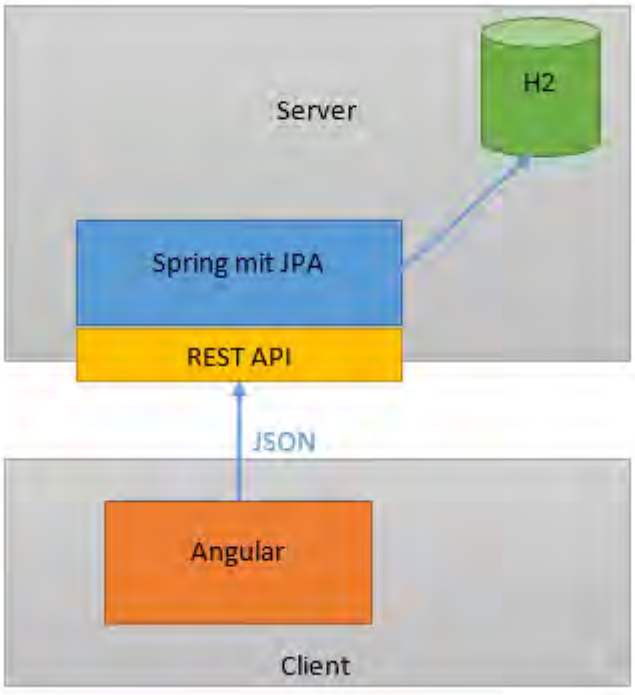

Abbildung 3.2: Erweiterte Architektur der Anwendung

In der Abbildung 3.2 wurde die Architektur vervollständigt. Auf der Cleintseite über nimmt das AngularJS Framework die meiste Arbeit. Auf der Serverseite das Spring Frameworks.

Es stellt die REST API bereit und übernimmt die Datenhaltung über die integrierte H2 Datenbank.

3.5 Datenmodell

Die zugrunde liegende Datenstruktur entspricht einem Entscheidungsbaum. Um die Anwendung möglichst allgemein zu halten, ohne sie für einen bestimmten Anwendungszweck zu schreiben, muss das entsprechende Grundgerüst zum Speichern der Entscheidungsbäume generisch sein.

Im verglich zu den bereist existierenden Anwendungen 2.10.1, soll es in dieser möglich sein, nicht nur Textelement zu verwenden. Der Datentype soll generisch sein, so wird dieser in einem abstrakten HyperMedia Element ausgelagert.

Die meisten Entscheidungsbäumen bestehen aus Nodes, welche die Frage beinhalten. Diese Nodes sind wiederum mit weiteren Nodes verbunden. Diese Verbindung erfolgt über die Antworten zu der Frage in der Node. Dieses Modell kann nur für einfache Entscheidungsbäume genutzt werden, die nur aus textuellen Fragen und Antworten bestehen. Da hier aber HyperMedia Elemente, für die Repräsentation von Fragen und Antworten verwendet werden, ist die Datenstruktur etwas komplexer.

Die Wurzel des Entscheidungsbaums besteht aus einem Node. Jede Node beinhaltet ein Hypermedia Element, das kann zum Beispiel ein Bild sein oder eine textuelle Frage. Zu dieser Node gibt es mindestens eine, aber auch mehrerer Decisions, also Entscheidungen beziehungsweise Antworten. Jede Decision beinhaltet, ähnlich wie eine Node, ein Hypermedia Element, welches die Entscheidung repräsentiert. Das können zum Beispiel textuelle Antworten sein, aber auch Bilder oder Antworten aus Audio-Samples. Eine Decision kann eine Node als Nachfolger haben, wenn der Baum noch tiefer ist. Wenn die Decision keinen Nachfolger hat, also nicht mit einer weiteren Node verknüpft ist, ist das eine abschließende Endscheidung. Das kann eine Antwort beziehungsweise Entscheidung auf die allgemeine zugrundeliegende Frage zu diesem Entscheidungsbaums sein.

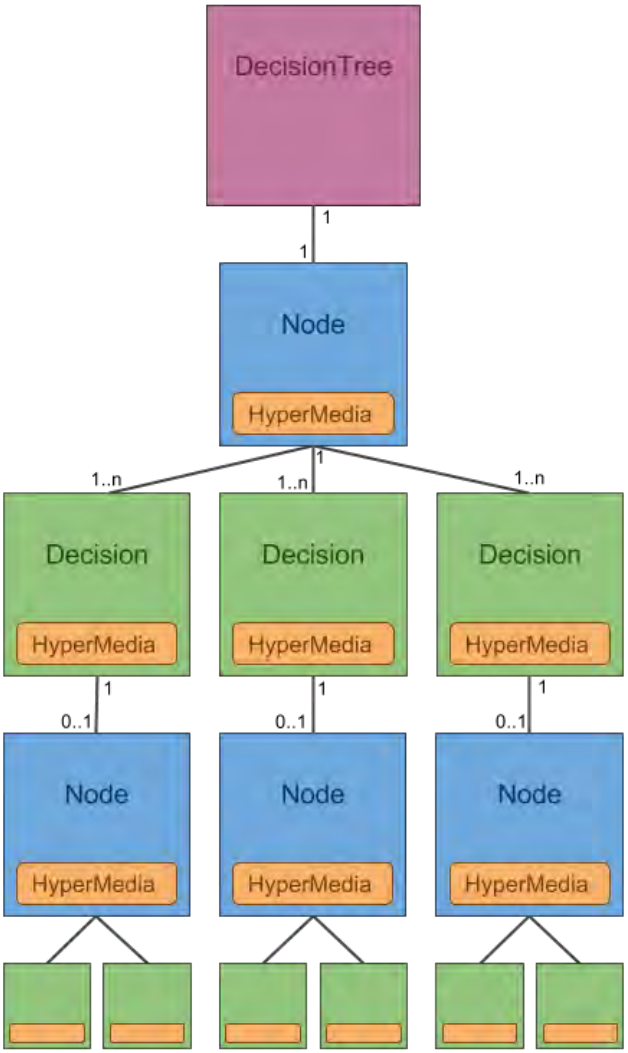

Abbildung 3.3: Darstellung des Datenmodells

In der Abbildung 3.3 wird die Struktur des Datenmodells veranschaulich dargestellt. Die Baumstruktur ist deutlich zu erkennen. Das Element DecisionTree bildet den Einstiegspunkt für die Applikation. Hier kann dem Baum ein Namen und eine Beschreibung gegeben werden. Hier ist auch die erste Node verknüpft, mit der das Durchlaufen des Baumes beginnt. Auf die erste Node folgen drei Decisions, die wiederum mit weiteren Nodes verbunden sind. Jeder Knoten beinhaltet ein Hypermedia, welches den Inhalt repräsentiert.

3.6 REST Server nach KISS

Wie bereits in Kapitel 3.4.2 beschrieben, wird der Server mit Hilfe des Spring Frameworks umgesetzt. Ein wichtiger Aspekt war den Server nach dem KISS Prinzip [1] möglichst einfach aufzubauen. Da sich gerade bei der Entwicklung die Datenstruktur oft ändert, sollte der Server nicht jedes Mal wieder geändert und angepasst werden. So müssten, wenn ein neues Feld in einer Datenbank hinzugefügt wird, die bereits existierenden Datensätze entsprechend migriert und angepasst werden. Zum anderen wird er in der Single Page Applikation nur für die Datenhaltung benötigt, da die Datenverarbeitung auf der Clientseite ausgeführt wird.

Durch die Java Persistence API, die im Spring Framework implementiert ist, entfällt für den Entwickler die komplette Arbeit mit der Datenbank. Dieser muss nur Entitys anlegen. Wobei eine Entity eine Tabelle in der Datenbank repräsentiert und ein Objekt dieser Klasse einen Eintrag in dieser. Zudem muss die Art der Relationen zwischen den einzelnen Entitys angeben werden und für jedes Attribut eine Setter- und Getter-Methode.

[1] Das KISS Prinzip besagt eine möglichst einfache aber bewerte Lösung zu nehmen. KISS ist ein Apronym und wird meisten mit der Bedeutung ”Keep It Small and Simple” ausgeschrieben.

```
1 @Entity
2 public class SimpleClass {
3         @ID
4         private long id;
5         private String someData;
6 ...
7 }
```

Listing 3.1: Eine Entity Klasse ohne die Getter- und Settermethoden.

3.7 Interaktive Client

Um eine möglichst moderne Anwendung zu erhalten, die auf unterschiedlichen Plattformen läuft, empfiehlt es sich eine Webanwendung zu schreiben. Wie bereits erwähnt, lohnt es sich um sich unnötige Schreibarbeit zu sparen und zum anderen auch zur Vermeidung von Fehlern, ein Framework zu verwenden.

Durch die Anforderungen an das Framework, Single Page Applikationen zu unterstützen, entsteht eine moderne Anwendung ohne Unterbrechung des Präsentationsflusses. Zum anderen muss die Kommunikation mit einer REST API möglich sein, so dass mit dem Server kommuniziert werden kann, der für die Datenhaltung zu ständig ist.

3.8 Konzept

Hier wird das Konzept der Anwendung beschrieben. Zuerst wird auf die Klassen des Servers eingegangen, danach die Idee hinter den HyperMedia Elementen erklärt. Zum Abschluss dieses Kapitels wird der Gedankengang hinter den Controller des Clients erläutert.

3.8.1 Server

Das im Kapitel 3.5 vorgestellte Datenmodell wird auf der Serverseite implementiert. Dazu gibt es eine Klasse HyperMedia, die für die Datenhaltung dieser Elemente zuständig

ist. Diese ist möglichst abstrakt gehalten, sodass die unterschiedlichsten Elemente in diesem gespeichert werden können.

Die Klassen Node und Decision repräsentieren Knoten in dem Entscheidungsbaum. Diese Klassen haben jeweils ein Objekt der Klasse HyperMedia, welches für die Datenhaltung des Inhaltes zuständig ist.

Die Wurzel des Entscheidungsbaums bildet ein Objekt der Klasse DecisionTree. Das Node Objekt in dieser Klasse bildet den Einstiegspunkt in den Entscheidungsbaum. Die Node Klasse repräsentiert eine Frage zu einem Attribut. Diese Frage wird über das HyperMedia Objekt gespeichert. Des Weiteren werden in der Node alle möglichen Entscheidungen gespeichert, es muss aber mindestens eine sein.

Die einzelnen Entscheidungen werden durch Objekte der Klasse Decision repräsentiert. Die eigentliche Entscheidung, die später visualisiert dargestellt wird, ist wieder in einem HyperMedia Objekt ausgelagert. In einer Decision wird gegebenenfalls auch die Nachfolgernode gespeichert.

3.8.2 HyperMedia

Im Vergleich zu den bereits existierenden Anwendungen für interaktive Entscheidungsbäume soll es bei dieser Anwendung möglich sein, nicht nur textbasierte Nodes und Decisions zu erstellen.

Als Basisformat für ein HyperMedia Element, so wie es in einer Node beziehungsweise in einer Decision vorkommt, soll aus einem Text, einem Bild oder einer Audiodatei bestehen. Diese können auch direkt in HTML als natives Element dargestellt werden.

Um mehr Interaktivität in die Anwendung zu bekommen, gibt es für die Decisions ein weiteres Format und zwar die imgarea. Bei diesem Format handelt es sich um einen Bildausschnitt. Diesen Ausschnitt gibt es nur, wenn das HyperMedia Element der Node ein Bild ist. Auf diesem Bild kann ein neuer Bereich ausgewählt werden, welcher als eine Entscheidung verwendet wird.

Um diesen Bereich darzustellen, war ursprünglich angedacht, dies über das HTML Element map zu realisieren [SEL16]. In diesem map Element können über das Element

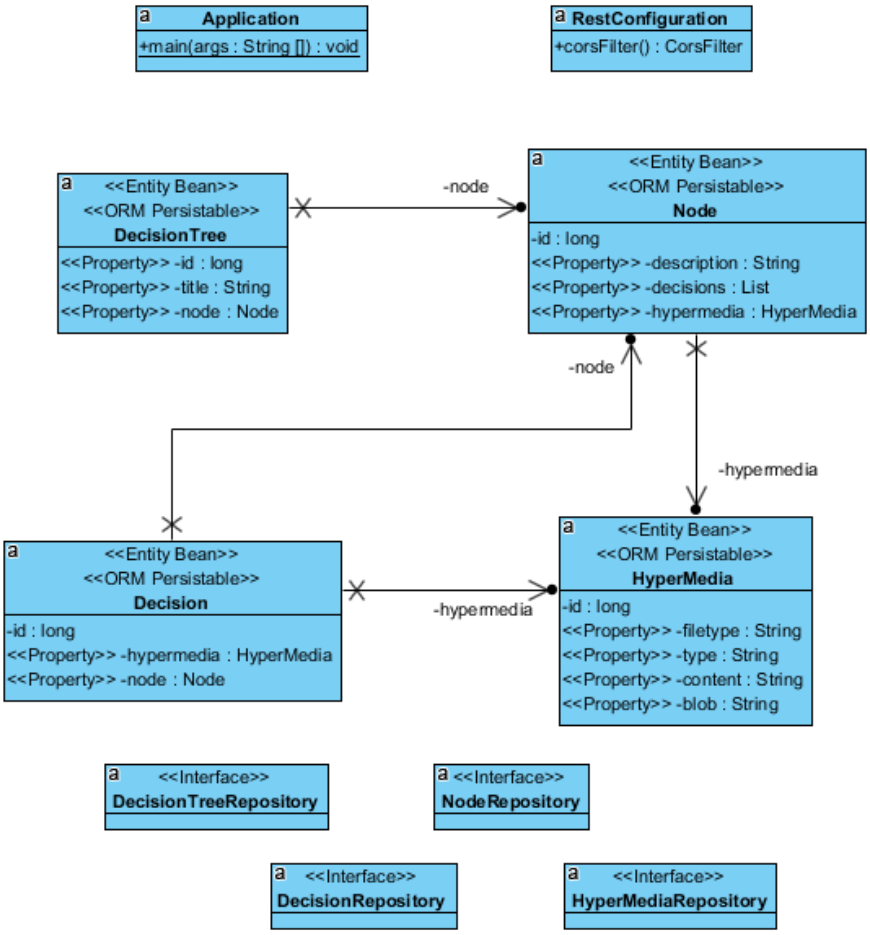

Abbildung 3.4: Klassendiagramm aller verwendeten Klassen auf dem Server

area Bereiche definiert werden, die HyperLinks enthalten. Doch dieses Element kann nur sehr umständlich mit CSS formatiert werden. Stattdessen werden diese Bereiche durch einfache Rechtecke visualisiert.

In der Anwendung werden auf dem Bild der Node, die verschiedenen Bildausschnitte dargestellt. Diese können dann angeklickt werden, um zu nächsten Node zu gelangen.

Ein zusätzliches Element für mehr Interaktivität sind inline Antworten. Dazu kann als HyperMedia Element der Node inline gewählt werden. Die Decisions, zu dieser Node, können dann nur noch vom Typ Text sein. Der Text der Decision, wird als Button direkt in der Anwendung dargestellt.

Die zwei Elemente lassen sich sehr gut miteinander kombinieren. So können auf einem größeren Bild verschiedene Bereiche als imgarea festgelegt werden. Als Nachfolgernode, der jeweiligen Bereiche, wird eine Node vom Typ inline erstellt, welche die Entscheidungen zu diesem Bereich beinhaltet.

In der Anwendungen kann ein Bildbereich ausgewählt werden und in diesem werden dann die Buttons mit den verscheiden Entscheidungen dargestellt. In Abbildung 3.5 wird dargestellt, wie inline Antworten visualisiert werden.

3.8.3 Client

Die Clientapplikation wird mit dem Webframework AngularJS umgesetzt. Diese basiert auf dem MVC Konzept. Das heißt für die Datenverarbeitung gibt es verschiedene Controller und für die Darstellung verschiedene Views. Das Modell für die Datenhaltung übernimmt der Server.

Es gibt drei Arten von Controllern. Das erste sind Controller, die etwas in einer Liste anzeigen. Diese werden zum Auflisten von bestehenden Entscheidungsbaum benötigt, sowie zum Auflisten aller Nodes. Dafür ist der *TreeListController* beziehungsweise der *NodeListController* zuständig.

Die zweite Art von Controllern ist der *TreeCreateContoller* und der *NodeCreateController*. Diese Controller erstellen einen neuen Entscheidungsbaum beziehungsweise eine neue Node.

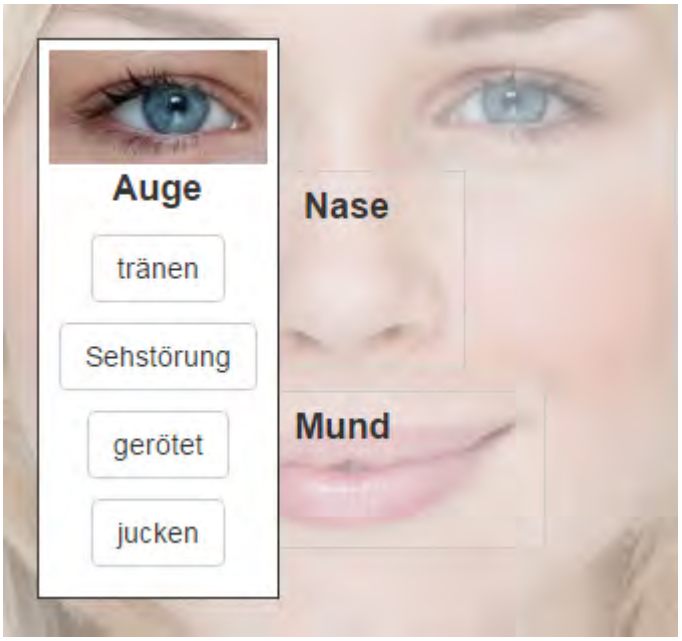

Abbildung 3.5: Inline Decisions für die Node Auge

Die dritte Art sind die View Controller, diese dienen zum Anzeigen von Entscheidungsbäumen, Nodes und HyperMedia Elementen.

Jeder Controller ist mit einer View in der app.js verbunden. Der *NodeCreateController* dient zum Erstellen einer neuen Node. Da eine Node immer mit mindestens einer Decision verbunden ist wird, werden die dazugehörigen Decisions in diesem Schritt gleich mit erstellt.

Ein Controller, der eine Sonderrolle besitzt ist der *NodeDialogSelectController*. Dieser dient um eine Decision mit einer Node zu verbinden.

Zudem gibt es keinen gesonderten Controller zum Erstellen von HyperMedia Elementen. Diese werden bei Erstellen einer Node benötigt und werden im *NodeCreateController* ebenfalls erstellt.

Beim Aufrufen einer URL wird ein Controller mit einer View geladen, daher ist ein Controller für eine Aufgabe zuständig und ist von den anderen Controllern somit unabhängig.

Hier ist es wie bei einer klassischen Webseite, bei dieser wird bei jedem Aufruf genau eine Datei aufgerufen.

Der Ablauf beim Erstellen eines neuen Baums ist, dass zuerst die einzelnen Nodes mit ihren jeweiligen Decisions Erstellt werden. Beim erstellen einer neuen Node, müssen dafür zuerst die HyperMedia Element erzeugt werden, sodass diese bei der eigentlichen Erstellung der Node beziehungsweise Decsison bereits zur Verfügung stehen.

Wenn die Nodes erstellt wurden, können diese mit anderen Verbunden werden, um die Baumstruktur abzubilden. Zum Schluss muss ein Entscheidungsbaum noch mit einer Node verbunden werden. Dieses Element wird zum Wurzelelement des Baumes.

4

Implementierung

In diesem Kapitel wird die Implementierung des Servers und des Clients erläutert. Dabei wird der in Kapitel 3 vorgestellte Entwurf umgesetzt.

4.1 Server

Um in Spring eine von der Java Persistence API verwaltete Instanz zu erstellen, muss die Klasse mit der Annotation *@Entity* gekennzeichnet werden. Zudem benötigt jede Klasse eine ID vom Typ Long, die als ID über die Annotation *@Id* gekennzeichnet wird. Über die Annotation *@GeneratedValue(strategy = GenerationType.AUTO)* wird der Java Persistence API mitgeteilt, dass automatisch generiert wird. Zudem benötigt jedes Attribut der Klasse eine Setter- und eine Getter-Methode, über die die API kommuniziert.

4.1.1 DecisionTree

Die Klasse DecisionTree.java hat zu der ID noch ein Attribut title über das dem Entscheidungsbaum ein Titel für die bessere Unterscheidung gegeben werden kann. Über das Attribute node wird das erste Blatt im Baum verknüpft, das mit der Annotation *@OneToOne* gekennzeichnet wird, da es eine 1-zu-1 Beziehung zwischen den DecisionTree und der Node gibt.

4.1.2 Node

Die Klasse Node.java, die ein Blatt mit einer Fragestellung im Baum repräsentiert, hat ein Attribut description, ein Attribut hypermedia und ein Attribut decisions. Über das Attribut description kann der Node eine Beschreibung gegeben werden. Über das Attribut hypermedia wird die Node mit dem entsprechenden HyperMedia Element verbunden, es ist hierbei wieder eine 1-zu-1 Beziehung, die mit der Annotation *@OneToOne* gekennzeichnet wird. Bei dem Attribut decisions handelt es sich um eine Liste. Diese Liste sind die entsprechenden Antworten auf die Frage. Da es mehr als eine Antwort geben kann, handelt es sich hierbei um eine Liste und wird mit der Annotation *@OneToMany* gekennzeichnet, da eine Node mit mehreren Descision verbunden sein kann. Die Annotation *@OneToMany* benötigt noch den Typ der Zielklasse mit der das Objekt verbunden wird. Das wird über den Parameter targetEntity bei der Annotation gesetzt, der den Wert der Zielklasse bekommt, in diesem Fall also targetEntity=Decision.class.

4.1.3 Decision

Die Klasse Decision.java ist ähnlich zur Node aufgebaut. Sie hat zum einen ebenfalls ein Attribut hypermedia um eine Verknüpfung zu dem entsprechenden HyperMedia Objekt herzustellen, welches auch mit der Annotation *@OneToOne* gekennzeichnet ist. Des Weiteren hat diese Klasse ein Attribut node, das mit der Annotation *@OneToOne* gekennzeichnet ist, um eine Verbindung zur nächsten Node im Baum zu erstellen. Da auf jeder Decision nur eine oder keine Node folgen kann reicht hier eine 1-zu-1 Beziehung.

4.1.4 HyperMedia

Die letzte Klasse, die noch fehlt, ist die HyperMedia.java Klasse. Sie repräsentiert ein HyperMedia Objekt, wie es in den Nodes und Decisions vorkommt. Die Klasse besteht aus verschiedenen Attributen um Mediaelemente zu speichern. In dem Attribute filetype wird der Basistype gespeichert, dieser besteht aus Werten, die bei der Erstellung einer neuen Node im Frontend ausgewählt werden kann. Das kann zum Beispiel der Wert

TXT für ein Textelement sein, IMG wenn ein Bild gespeichert werden soll oder AUDIO wenn eine Audiodatei gespeichert werden soll. Das Attribute type speichert den Typ des gespeicherten Mediaelements in Mimetype [1] Format. Da es nicht genügt zu wissen, ob ein Bild gespeichert wurde, muss zudem noch gespeichert werden, wie das Bild codiert wurde. Zum Beispiel als JPG, Gif oder PNG. Im nächsten Attribute content wird der Inhalt des Elements gespeichert, zum Beispiel bei textuellen Fragen wird die eigentliche Frage in diesem Feld gespeichert. Das letzte Attribut blob wird bei der Speicherung von Bildern hinzugefügt. Da der Server nur aus einer Datenbank besteht, können die Dateien nicht einfach gespeichert werden. Um sie dennoch verwalten zu können, werden diese auf der Clientseite in BASE64 [2] codiert und dann über die REST API an den Server gesendet, der diese wiederum einfach als einen langen String ansieht und so in der Datenbank speichern kann.

4.1.5 CrudRepository

Um aus den Entitys, die über die Java Persistenz API verwaltet werden, eine REST API zu erstellen, muss im Spring Framework ein Repository Interface erstellt werden. Dazu wird ein neues Interface erstellt, welches von CrudRepository erbt. Dieses Interface wird dann noch mit der *@RepositoryRestResource* Annotation gekennzeichnet. Über die Parameter collectionResourceRel und path kann bei dieser Annotation angeben werden, unter welchem Pfad dieses Element über die REST API erreichbar ist. Das Interface CrudRepository stellt die generischen CRUD Operationen zur Verfügung.

4.1.6 SpringBootApplication

Um dem Spring Framework mitzuteilen, wo sich der Einstiegspunkt der Applikation findet, muss die Klasse, in der sich die main befindet, mit der Annotation *@SpringBootApplication* gekennzeichnet werden. In der main selber muss die Klasse dann an die SpringApplication übergeben werden.

[1] Der Mimetype dient zur Klassifizierung der Daten. Dieser besteht aus einen allgemeinen Typ und einen Subtype, wie zum Beispiel image/jpeg.

[2] BASE64 dient zum Kodieren von 8-Bit-Binärdaten. Dabei werden die Binärdaten in ”lesbare” Zeichen (A-Z, a-z, 0-9, +, /) kodiert.

```
1 @SpringBootApplication
2 public class Application {
3
4         public static void main(String[] args) {
5                 SpringApplication.run(Application.class, args);
6         }
7 }
```

Listing 4.1: Die Application Klasse, der Startpunkt der Anwendung .

4.1.7 CorsFilter

Eine Besonderheit, welche die Entwicklung der Anwendung einfacher macht, ist die Klasse RestConfiguration. Sie dient zur Konfiguration des CorsFilters für die Spring Anwendung. Bei Cors handelt es sich um die Cross-Origin Resource Sharing, dies dient um Sonderregeln für die Same-Origin-Policy zu erstellen. Die Same-Origin-Policy ist ein Sicherheitsmechanismus, der sicherstellt, das Ressourcen nur von selben Ursprung URL stammen. Bei Webanwendungen kann zum Beispiel JavaScript und CSS Dateien nur von derselben URL, also demselben Webservice geladen werden. Über den CorsFilter können zusätzliche URLs erlaubt werden. Für die Entwicklung wurden alle erlaubt, da zum Testen die Anwendung teilweise lokal ausgeführt, auf einem Testserver, der unter localhost läuft oder auf einen Webserver mit einer Domain. Sobald die Anwendung in der produktiven Umgebung eingesetzt wird, muss dieser Filter entsprechend angepasst, beziehungsweise könnte er ganz deaktiviert werden.

4.2 Client

In diesem Kapitel wird die Implementierung des Client erklärt. Hier wird die Umsetzung der Webanwendung über AngularJS beschreiben.

4.2.1 index.html

Wie bei jeder Webanwendung bildet die index.html den Einstiegspunkt der Webanwendung. Diese beinhaltet das Basisdesign, welches hier mit Bootstrap aufgebaut ist. Hier werden alle benötigten CSS-Dateien eingebunden, sowie die benötigten JavaScript Dateien.

Durch das Attribut *ng-app* in body Tag wird die Anwendung zur einer Angular Applikation. Dabei wird Angular zugleich mitgeteilt, welches Angular Modul geladen werden soll.

```
1 <body ng-app="decisiontree">
```

Listing 4.2: Definition des Angular Moduls

Über das Attribut *ng-view* wird der Container gekennzeichnet, in den die Templates später gerendert werden.

```
1 <div class="container" ng-view></div>
```

Listing 4.3: Definition des Angular Containers für die Views

Im unteren Teil der index.hmtl werden die JavaScript Dateien eingebunden, sodass die Webseite geladen und gerendert werden kann und dann erst die JavaScript Dateien geladen werden.

Als erstes wird das JQuery Framework eingebunden, das eine einfachere DOM Manipulation ermöglicht. Dann werden die JavaScript Dateien des Bootstrap Frameworks eingebunden, als nächstes das Angular Framework mit dessen Addons. Danach wird die app.js, die das Angular Modul beinhaltet, und die entsprechenden Angular Controller. Und zuletzt noch ein Plugin für das JQuery Framework, das später bei der Erstellung von HyperMedia Element verwendet wird.

4.2.2 app.js

In der app.js wird das Angular Modul definiert. Diesem Modul wird ein Namen gegeben und die Abhängigkeit zu weiteren Modulen. Des Weiteren können hier Konstanten

definiert werden. Hier wird eine Konstante für den Mimetype für plain text definiert. Durch das Modul ngRoute können hier Routen definiert werden, unter denen die Templates mit ihren entsprechenden Controller verbunden werden.

In einem Auszug aus der app.js 4.4 wird die Route für die URL auf die Endung http://domain/tree definiert. Wenn diese URL aufgerufen wird, wird das Template list.html geladen und der Controller TreeListController verwendet. Hier wird auch eine default Route definiert, also wenn keine spezielle URL genannt wird oder eine, die nicht definiert ist. In diesen Fall wird die /tree Route geöffnet.

```
1  app.config(function($routeProvider){
2    $routeProvider
3        .when("/tree", {
4          templateUrl: "app/templates/tree/list.html",
5          controller: "TreeListController"
6        })
7        ...
8
9      .otherwise({redirectTo: '/tree'});
10 });
```

Listing 4.4: Definition der Routen in Angular

4.2.3 Controller

In den nachfolgenden Kapiteln werden die jeweils einzelnen Controller und ihre Funktionen kurz erläutert.

TreeCreateController

Über diesen Controller kann ein neuer Entscheidungsbaum erstellt werden. Dieser ist unter der Route /tree/new zu erreichen. Das verwendete Template besteht aus einem Formular, wobei die einzelnen Formularfelder in der form.html ausgelagert sind, so dass

diese zum Beispiel für ein Controller zum Editieren vorhandener Entscheidungsbäumen wieder verwendet werden könnten. Im Controller selber wird nur die Funktion createTree definiert, welche beim Absenden des Formulars aufgerufen wird. Das Eingabefeld für den Titel ist mit den ng-model tree.title verbunden. Beim Aufruf von createTree wird das tree Objekt, welches sich im Scope befindet, per HTTP POST an dir URL der REST API /tree gesendet. Ist das erfolgreich, wird auf die URL /tree weitergeleitet.

TreeListController

Der TreeListController ist dafür zuständig, alle vorhandenen Entscheidungsbäume in einer Liste anzuzeigen. Dazu wird eine HTTP GET Anfrage an die REST API auf die URL /tree gesendet. Die Antwort wird in dem Scope in das Objekt trees gespeichert. Des Weiteren werden zwei Funktionen definiert. Die Funktion id extrahiert aus einem tree Objekt die ID des Baums. Die openNode Funktion wird dafür benötigt, um den Baum mit der ersten Node zu verknüpfen. Dazu wird ein ngDialog geöffnet. Dies wird im Kapitel 4.2.3 genauer erläutert.

Das zum TreeListController gehörende Template beinhaltet eine Tabelle in der die Entscheidungsbäume aufgelistet werden. Die Bäume befinden sich im Scope im trees Objekt, welches über ng-repeat die Tabelle mit den entsprechenden Daten füllt.

```
1  <tr ng-repeat="tree in trees">
2    <td>{{tree.title}}</td>
3      <a href="#/tree/{{id(tree)}}" class="btn btn-default">
4        view
5      </a>
6      <a href="#" class="btn btn-default"
7       ng-click="openNodes(tree._links.self.href)">
8        link Node
9      </a>
10   </td>
11 </tr>
```

Listing 4.5: ng.repeat Direktive inAngularJS.

Hier wird die Funktion id verwendet um die Links auf den jeweiligen Entscheidungsbaum aufzubereiten und die Funktion openNode, welches den Dialog öffnet.

Die ng-repeat Direktive ist eine Art ”foreach”-Schleife für HTML, nur dass hier der HTML-Code, der zwischen dem Element, das mit der Direktive ausgezeichnet ist, wiederholt wird. In diesen Fall wird für jedes Element in trees eine neue Spalte (<tr>) erzeugt.

NodeCreateController

Der NodeCreateController ist ein recht komplexer Controller im Vergleich zu den anderen. Dieser Controller dient dem Erstellen von neuen Nodes. Da auf eine Node mindestens eine Decision folgt, werden diese gleich mit erstellt.

Zum NodeCreateController gehört das create.html Template, welches das Grundgerüst für das Formular beinhaltet. Die einzelnen Formularfelder sind wieder ausgelagert und werden durch ein ng-include eingebunden.

Da der Inhalt einer Node und einer Decision im Prinzip aus jeweils einem HyperMedia Element bestehten, sind die einzelnen Formularfelder ausgelagert. Zum Erstellen einer Node wird das Formular create.html eingebunden, welches das HTML Formular beinhaltet. Die Formularfelder befinden sich in einer separaten Dateien. Über ein Dropdown kann der Filetype des HyperMedia Elements ausgewählt werden. Dieses ist über das ng-model mit dem media.filetype verbunden. Als Filetype kann der Typ des HyperMedia Elements 3.8.2 gewählt werden.

Je nachdem, welcher Filetype in dem Dropdown Menü ausgewählt wird, wird über eine ng-if Abfrage das passende Eingabefeld angezeigt. So wird bei txt ein *<input type=”text”* bei img beziehungsweise audio wird eine *<input type=”file”* angezeigt, also ein Eingabefeld zum Hochladen von Dateien.

Da es eine dynamische Anzahl an Decisions gibt, mindestens eine, aber beliebig viele, werden diese in ein Array decisions gespeichert und über einen ng-repeat, je nach Anzahl der Einträge in diesen Array, die Datei createDecision.html für jeden Eintrag eingebunden. Dieses Template beinhaltet, ähnlich wie das Template für die Node, das HTML Formular für eine neue Decision. Die Formularfelder sind dabei wieder in eine Datei formDecision.html ausgelagert, die in das Formular eingebunden wird. Wie bereits erwähnt, ist eine Decision ähnlich zu einer Node aufgebaut, es gibt ein Dropdown

Feld, über das der Filetype ausgewählt werden kann. Hier kann ähnlich wie bei der Node zwischen txt, img und audio ausgewählt werden. Wenn bei der Node der Filetype inline ausgewählt ist, wird das Dropdown ausgeblendet, da es bei inline nur textbasierte Decisions gibt.

Eine Besonderheit bildet der Filetype imgarea. Bei diesem Typ von HyperMedia, kann ein Bereich eines Bildes gewählt werden. Dabei wird das Bild aus den HyperMedia Element der Node geladen, auf diesen wird dann ein Bereich ausgewählt. Dieser Bereich ist dann das HyperMedia Element der Decision. Über die Methode loadHyperMedia wird das Bild, welches in der Node im FileUpload Feld ausgewählt ist geladen und in dem Objekt decision.mediaimg gespeichert. Dieses Objekt wird über die Direktive ng-src geladen und dargestellt. Hier kommt das Plugin für JQuery zum Einsatz, mit diesen kann ein Bereich auf einen Bild ausgewählt werden. Abbildung 4.1, stellt dar wie ein Ausschnitt in JCrop ausgewählt werden kann.

Abbildung 4.1: Ein Bildausschnitt über das JCrop Plugin für JQuery

Die Decisions werden in einem Array gespeichert, da die Anzahl nicht von vornherein fest steht. Wenn eine zusätzliche Entscheidung hinzugefügt werden soll, muss also ein Element in das Array hinzugefügt werden. Das macht die Funktion addDecision. Diese Funktion fügt ein neues Objekt mit der dem Inhalt id: decisionX hinzu, wobei X

die Zahl des Objekts ist. Diese Funktion ist an einen Button über die Direktive ng-click gebunden. Bei einem Klick auf diesen Button wird die Funktion aufgerufen, welche ein neues Element in das Array hinzufügt. Dieses wird durch das 2-Way-Databinding von Angular und der ng-repeat Direktive direkt angezeigt.

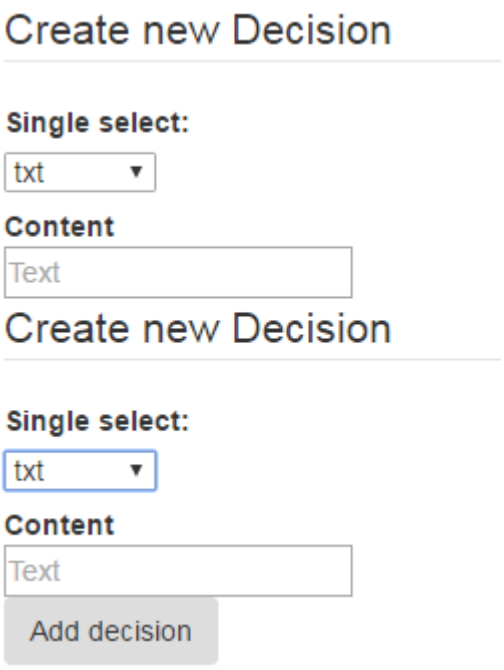

Abbildung 4.2: Hinzufügen einer neuen Decision

```
<div ng-repeat="decision in decisions">
  <div
   ng-include="'app/templates/hypermedia/createDecision.html'">
  </div>
</div>
<button class="btn" ng-click="addDecision()">
  Add decision
</button>
```

Listing 4.6: HTML-Code zum hinzufügen neuer Decisions

Die Hauptaufgabe des NodeCreateController ist es die Node zu erstellen und zu speichern. Dafür muss als erstes das HyperMedia Element der Node erstellt werden, danach die HyperMedia Elemente der Decisions und dann die Decisions selber. Für diese wer-

den aber die jeweiligen HyperMedia Element benötigt. Sind alle Decisions gespeichert sind, kann die Node erstellt werden und zusammen mit dem Node-HyperMedia Element und den Decisions gespeichert werden.

Ein großes Problem ist die asynchrone Ausführung und Laufzeit bei JavaScript. So wird nach dem Absenden einer HTTP Anfrage eine Funktion aufgerufen, ein sogenannter Callback, in dem die Antwort verarbeitet werden kann. Wann und ob überhaupt jemals diese Funktion aufgerufen wird, kann dabei nicht definiert werden. So wird zum Erstellen einer Decision das entsprechende HyperMedia Element benötigt, welches erst in dem Callback der entsprechenden HTTP POST Anfrage bekannt ist. Also kann die Decision erst im Callback des HyperMedia Elements erstellt werden. So entsteht eine Anzahl an offenen Verbindungen und erst wenn alle wieder geschlossen sind kann, die eigentliche Node erstellt werden, da für diese alle anderen Elemente benötigt werden.

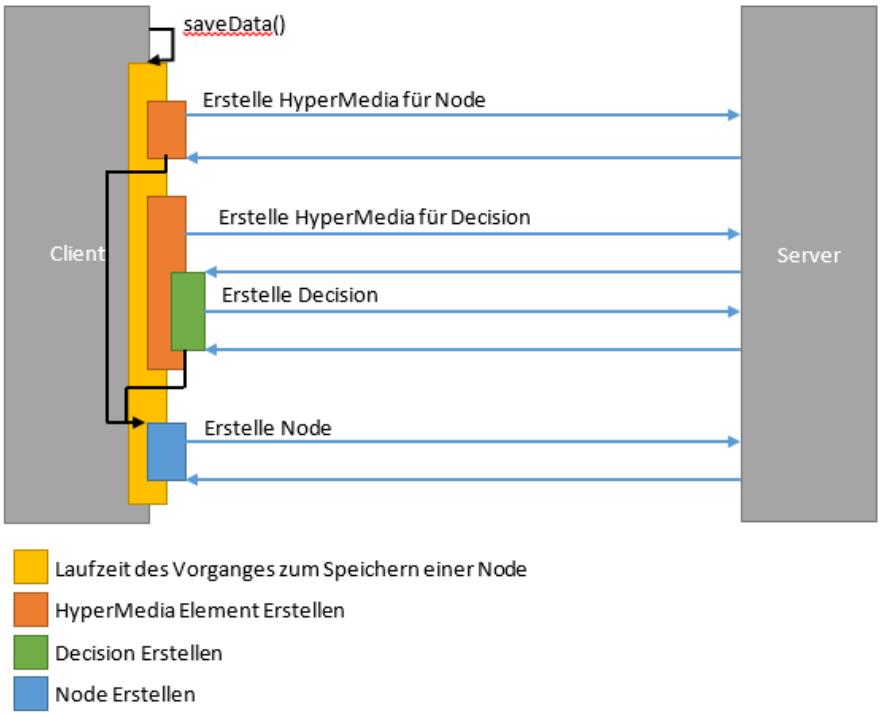

Abbildung 4.3: Ablauf der Kommunikation bei Erstellen einer neuen Node

Das Speichern wird über die Funktion saveData angestoßen, welche über einen Button aufgerufen wird. Der Vorgang ist visualisiert in der Abbildung 4.3 dargestellt.

Als erster Schritt wird der Filetype des HyperMedia Elements der Node ausgewertet und entweder die Datei geladen, wenn es sich um ein Bild oder eine Audiodatei handelt, der type kann aus dem Upload Dialog ausgelesen werden und ist direkt im Mimetype Format. Die Datei wird Binär eingelesen und dann BASE64 codiert und in das blob Attribut gespeichert. Handelt es sich um Text als HyperMedia Element, muss der Mimetype manuell gesetzt werden. Das media Objekt wird dann an die REST API an die URL für HyperMedia Element gesendet. In dem success Callback wird die URL des Ergebnisobjekts gespeichert, da diese für die Node später benötigt wird. Dies ist in der Abbildung durch den ersten Block in orange dargestellt.

Als zweiter Schritt wird die Funktion createDecisions aufgerufen, in dieser Funktion wird per forEach Schleife über die Elemente des Array decisions iteriert. Das ist der zweite Block in dunklere orange auf der Abbildung 4.3. Für jede Decision in diesem Array wird eine Funktion als Callback aufgerufen. In dieser Funktion wird der Filetype ausgewertet. Ein Unterschied zur Node ist der Filetype imgare, der einen Bildausschnitt darstellt. In diesem Fall wird der Wert des JCrop Plugins ausgewertet und im JSON Format gespeichert. Diese bestehen aus einem Objekt mit x-Position, y-Position, Höhe und Breite des gewählten Ausschnitts. Danach wird die Funktion createDecision aufgerufen und die Decision als Parameter übergeben.

In der Funktion createDecision wird das decision Objekt an die REST API gesendet, an die URL um ein neues HyperMedia Element zu erstellen. Dessen Antwort wird im Callback der success Funktion verarbeitet. Da das HyperMedia Element für das eigentliche Decisionelement benötigt wird, wird in der Callback Funktion das Decisionelement erstellt. Eine Decision besteht aus einem HyperMedia Element und ist gegebenenfalls mit einer Node verknüpft. Da es aber noch keine weitere Node gibt, besteht die Decision nur aus einem HyperMedia Element, welches in der Callback zur Verfügung steht. Es wird also ein neues Objekt erstellt, welches nur aus einem HyperMedia Element besteht. Dieses wird an die REST API decision gesendet. In dem Callback der success Funktion zum Erstellen einer neuen Decision wird die Antwort, also die neue Decision, in einem Array gespeichert. In der Abbildung 4.3 ist das durch den grünen Block dargestellt.

Sind jetzt alle benötigten Objekte erstellt wurden, könnte jetzt die Node erstellt werden. In der Abbildung 4.3 ist hierfür der blaue Block abgebildet.

Hier kommt aber wieder das Problem der asynchronen Ausführung von JavaScript zu tragen. Es kann nicht gesagt werden, ob alle Callbacks der gesendeten POST Nachrichten ausgeführt wurden.

Dafür musste ein kleiner Workaround eingebaut werden, um feststellen, wann alle Abfragen erfolgreich abgearbeitet sind und somit alle Decision erstellt wurden. Dafür wird vor jeder Anfrage ein neuer Wert in ein Array hinzugefügt. Der Wert an sich ist uninteressant, wichtig ist die Anzahl der Werte. In dem Callback einer POST Nachricht wird ein Wert wieder aus diesem Array entfernt. Somit ergibt die Länge des Arrays die Anzahl noch offener Verbindungen. Über die Funktion addNewConn(), welche vor der HTTP Nachricht aufgerufen wird, wird ein Element in das Array openConnections hinzugefügt und über die Funktion removeConn() wird wieder ein Element aus dem Array entfernt. Angular bietet eine watch Listener. Dieser Listener überwacht eine Variable, sobald diese verändert wird. Dann wird ein Callback aufgerufen. Hier wird die Länge des Arrays openConnections überwacht, wenn eine neue Verbindung geöffnet oder geschlossen wird, wird jetzt der Callback aufgerufen. In diesem Callback wird als erstes die Länge des Arrays openConnections überprüft, da nur eine Länge von 0 von Interesse ist. Dieser Fall tritt zwei Mal auf. Das erste Mal zu Beginn, wenn das Array initialisiert wird und noch keine Werte hat. Das zweite Mal wenn alle Verbindungen abgearbeitet wurden. Der zweite Fall ist hierbei wieder der Interessante, in diesem ist das nodeMedia Element nicht null. Diese Element ist das HyperMedia Element der Node und wurde bereits angelegt. Zudem hat in diesem Fall das Array decisionsNode eine Länge von ungleich Null. In diesen Array werden alle erstellten Decisions gespeichert und da es mindestens eine gibt, hat dieses Array nur nach der Initialisierung die Länge Null. Wenn also das nodeMedia Element existiert und das decisionsNode Array eine Länge von ungleich Null besitzt, wurden alle Verbindungen abgearbeitet und die eigentliche Node kann erstellt werden. Das Node Objekt besteht aus einer Beschreibung, einem HyperMedia Element und aus den Decisions. Dieses Objekt wird an REST API zum Erstellen neuer Nodes gesendet. Ist dies erfolgreich und die Node wurde erstellt, wird auf die URL /node weitergeleitet.

Ein anderer Lösungsweg für die asynchrone Ausführung von JavaScript wären eventuell die von Angular zur Verfügung stehenden Promises. Ein Promise, also ein Versprechen, wird als Rückgabewert beim Senden einer HTTP Nachricht zurückgegeben. Ein Versprechen, dass diese Anfrage verarbeitet wird. Der Angular Service q bietet die Möglichkeit durch die Methode all ein Callback zu definieren, der aufgerufen wird, wenn alle Promises erfüllt, also abgearbeitet wurden. Das Problem bei der Implementierung war, dass die Promises verschaltet waren. Es waren die Promises nach dem Erstellen der HyperMedia Elemente für die Decisions erfüllt, bevor die eigentliche Decisions erstellt waren. Und der Callback wurde nicht mehr aufgerufen, weshalb die zuvor beschriebene Implementierung erhalten blieb.

NodeListController

Der NodeListController ist ähnlich zum TreeListController ausgebaut. Die Aufgabe des Controllers ist alle Nodes aufzulisten. Dafür wird eine HTTP GET Anfrage an die REST API gesendet, um alle Nodes zu erfassen. Diese werden in dem Objekt nodes gespeichert und in einer Tabelle wieder über ng-repeat Direktive dargestellt. Wie im TreeListController gibt es zu dem eine Funktion um an die ID einer Node zu kommen und die Funktion viewMedia um das HyperMedia Element der Node zu öffnen und darzustellen.

Hierfür wird ein ngDialog geöffnet und als Template die hypermedia/view.html verwendet und als Controller der MediaViewController als data Parameter wird die URL auf das HyperMedia Element in der REST API übergeben. Die Darstellung des HyperMedia Elements wird in Kapitel 4.2.3 beschrieben.

NodeViewController

Im NodeViewController wird eine Node mit ihren Decisions dargestellt, zum Einen um die einzelnen Decision sich nochmal anschauen zu können und zum anderen, um die Decisions gegebenenfalls mit einer Node zu verbinden.

Im Controller wird eine HTTP GET Anfrage an die REST API auf die URL /node/ID gesendet, die ID wird als routeParams in Angular übergeben. Im Callback dieser Anfrage

wird das entsprechende Node Objekt in den Scope gespeichert und eine neue Anfrage gesendet, um alle Decisions dieser Node zu erhalten. Diese werden wiederum dem Callback in den Scope gespeichert.

Zum dem gibt es noch zwei Funktionen in diesem Controller. Die viewMedia ist ähnlich zu der im NodeListController 4.2.3, es wird nur ein entsprechend andere data Parameter übergeben.

Die openNodes Funktion ist dieselbe, wie im TreeListController und dient dazu eine Decision mit einer Node zu verbinden. Die Funktionsweise wird im Kapitel 4.2.3 erklärt.

Im Template wird zum einen die Beschreibung der Node ausgeben und in einer Tabelle die Decisions aufgelistet. Dies erfolgt wieder mit der ng-repeat Direktive, in dem über das Array decisions iteriert wird.

MediaViewController

Der MediaViewController dient dazu, ein HyperMedia Element dazustellen. Dafür wird eine HTTP GET Anfrage an die entsprechende URL an die REST API gesendet. Die URL wird über den data Parameter des ngDialog Services übergeben. Diese Daten können über das ngDialogData Objekt aus dem Scope geladen werden. Im Callback müssen die Daten, wenn es sich um ein Bild oder eine Audiodatei handelt, noch aufbereitet werden. In HTML können Media Objekte direkt dargestellt werden, wenn diese BASE64 codiert sind. Dazu muss in das Source-Attribute des entsprechenden Tags mit data: Mimitype; base64, blob gefüllt werden.

```
1 <img src="..."/>
```

Listing 4.7: In HTML ein BASE64 codiertes Bild ein binden.

Das Template gibt im Prinzip einfach nur alle Daten aus. Es wird nur unterschieden, ob der filetype img beziehungsweise audio ist. In diesem Fall wird das entsprechende HTML Tag verwendet.

In Abbildung 4.4 ist beispielhaft ein HyperMedia Element geöffnet. Es werden die Attribute des Elements angezeigt und das eigentliche HyperMedia Element.

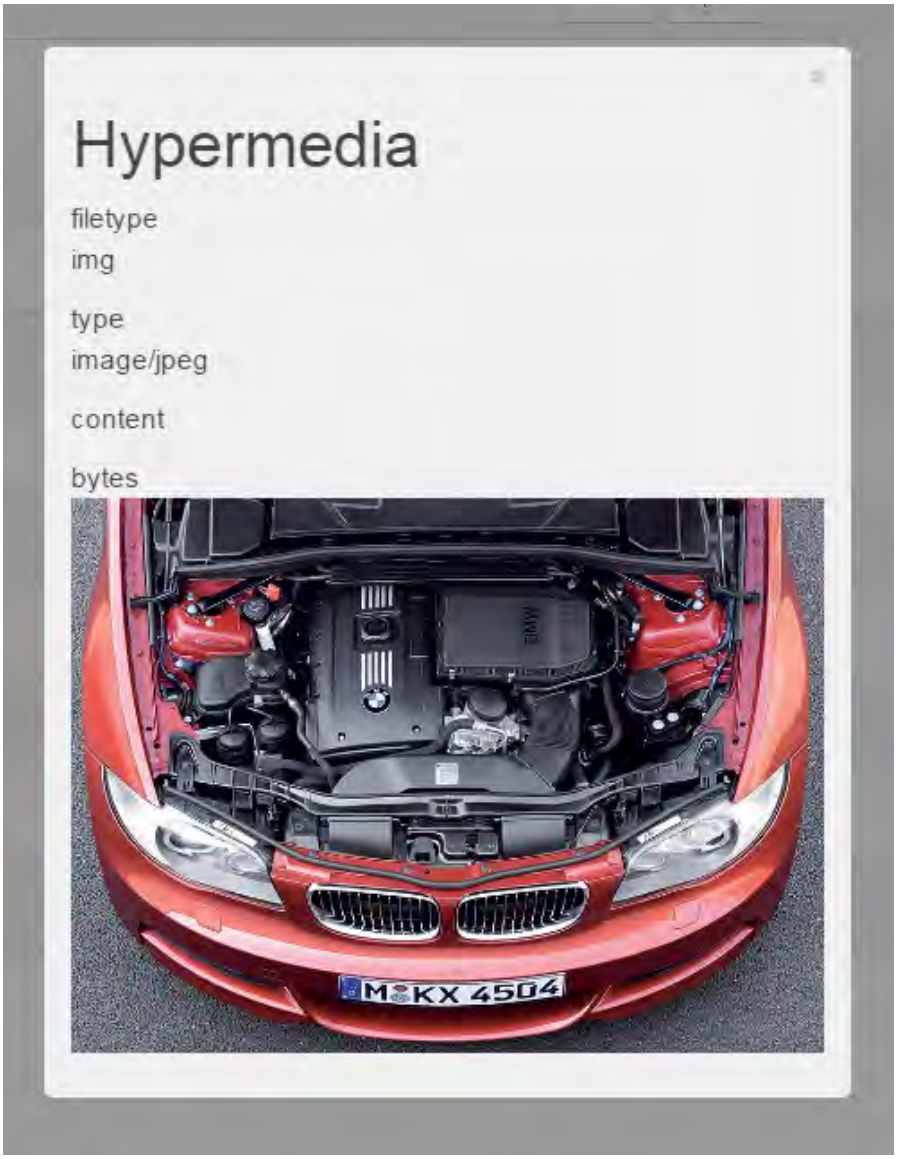

Abbildung 4.4: Ein HyperMedia Element in einem ngDialog geöffnet.

NodeDialogSelectController

Der NodeDialogSelectController dient dazu eine Decision mit einer Nachfolger Node zu verbinden, beziehungsweise den Tree mit einer Startnode zu verbinden. Dazu wird die Node, welche die zu verknüpfende Decision enthält, über den NodeViewController geöffnet und der Button *link Node* geklickt. Das führt die Funktion openNodes aus, welche wiederum einen ngDialog öffnet. Als data Parameter wird die aktuelle Decision, als neuer Parent mitgegeben, als Controller der NodeDialogSelectController und als Template das dialog/select.html.

Im NodeDialogSelectController wird eine HTTP GET Anfrage an die REST API gesendet um alle Nodes zu bekommen. Diese werden über den Scope in das Array nodes gespeichert. Im Template werden alle Nodes in einer Tabelle wieder über ng-repeat Direktive dargestellt. Soll nun die Decision mit einer dieser Nodes verbunden werden, wird über den Button *select* die Funktion selectNode im Controller aufgerufen. Diese Funktion enthält als Parameter die REST API URL auf die Node. Dann wird eine HTTP PATCH Nachricht an die Decision gesendet in dem das Node Attribut auf das im Parameter Übergebene upgedatet wird. Danach schließt der Dialog.

Diese Funktion wird auch im TreeListController verwendet, um einen Entscheidungsbaum mit der ersten Node zu verbinden. Dazu wird als Parameter der Funktion openNodes die URL des entsprechenden Entscheidungsbaums mitgegeben.

TreeViewController

Der TreeViewController ist dafür zuständig den Entscheidungsbaum zu visualisieren, sodass dieser durchlaufen werden kann. Die ID des Baumes wird per routeParams an den Controller übergeben. Als erstes wird der tree über die REST API über eine HTTP GET Anfrage geladen. Ist noch keine Decision gewählt, wird die Startnode aus dem tree geladen. Andernfalls, wenn eine Decision ausgewählt wurde, wird ein weiterer routeParams gesetzt mit der ID der Decision. So kann die Nachfolgernode der Decision geladen werden.

Ist die Node geladen, wird das entsprechende HyperMedia Element der Node geladen. Dieses HyperMedia Element wird in den Scope gespeichert und gegebenenfalls wird, wenn es sich um ein Bild oder Audiodatei handelt, das src Attribut zusammengesetzt.

```
1 $scope.nodeMedia.src =
2  "data:" + $scope.nodeMedia.type +
3  ";base64," + $scope.nodeMedia.blob;
```

Listing 4.8: Zusammensetzten des src Attributs.

Sind alle benötigten Daten für das Darstellen der Node geladen, können die Decisions geladen werden. Dafür wird wieder eine HTTP GET Anfrage an die REST API gesendet und zwar an die URL für die Decisions der Node. Danach wird für jede Decision das entsprechende HyperMedia Element über die REST API angefragt. In dem entsprechendem Callback wird der filetype ausgewertet und die Daten entsprechend aufbereitet. Für img und audio wird das src-Attribute zusammengesetzt und für imgare die Daten, die in im JSON Format gespeichert wurden, geparst. Die Decision wird in einem Array im Scope gespeichert.

Im TreeViewController sind zudem noch weitere Funktionen definiert. Die Funktion viewDecsion öffnet einen ngDialog zum Anzeigen einer endgültigen Decision. Dazu wird dem ngDialog der MediaViewController und das entsprechende Template übergeben. Es gibt wieder die id Funktion um eine ID aus einem Objekt zu extrahieren. Über die Highlight Funktion kann eine Decision gehighlightet werden. Dies dient dazu den imgarea Bereiche in der Node einfacher zu finden. Dazu wird, wenn auf eine Decision geklickt wird, zu der eigentlichen Node hoch gescrollt und das entsprechende Element mit der CSS-Klasse imgarea-decision-highlight für drei Sekunden hervorgehoben.

Die Funktion openDecision wird beim Auswählen einer Decision geöffnet. Zu Beginn der Funktion wird überprüft, ob eine Decision als Parameter übergeben wird. Ist das nicht der Fall, wird die aktuell gewählte Decision zurückgesetzt. Das ist nötig, falls inline Decisions gewählt wurden und es soll eine andere inline Decision selektiert werden. Dann muss die Auswahl zurückgesetzt werden. Dass erfolgt dadurch, dass keine Decision gewählt wird. Wenn eine Decision als Parameter übergeben wird, wird über die REST API die passende Nachfolgernode dieser Decision geladen, um den Typ

dieser Node zu überprüfen. Ist dieser vom Typ inline, werden die Decisions der Node geladen und für jede Decision das entsprechende HyperMedia Element um diese in der inlineDecisionsList der Ursprungs Decision, die als Parameter übergeben wurde, zu speichern. Ist die Nachfolgernode der Decision nicht vom Typ inline, wird auf eine andere Route umgeleitet.

```
1 $location.url("/tree/" + $scope.treeid +
2  "/decision/" + $scope.id(decision));
```

Listing 4.9: Umleiten auf eine andere URL.

Wird versucht die Nachfolgernode zuladen und ein Fehler tritt auf, gibt es keine Nachfolger und somit handelt es sich um eine Endentscheidung. Dafür wird die Node über die viewDecision Funktion geladen und in einem ngDialog dargestellt. Leider gibt es keine schönere und einfachere Möglichkeit festzustellen, ob es eine Nachfolgernode gibt.

Das entsprechende Template ist die view.html. Im oberen Teil wird die Node dargestellt, im unteren die Decisions. Für die Darstellung der Node wird zuerst über ein ng-switch anhand des nodeMedia.filetype ausgewählt, was genau dargestellt wird. Aufgrund der Einfachheit wird zuerst auf die im Code weiter unter stehenden Entscheidungen eingegangen. Wenn der filetype audio ist, wird ein HTML audio-Tag eingeblendet.

```
1 <div ng-switch-when="audio">
2   <audio id="bytes" controls ng-src="{{nodeMedia.src}}"></audio>
3 </div>
```

Listing 4.10: ng-switch für den Typ Audio.

Der Default Fall der ng-switch Anweisung stellt den content des HyperMedia Elements der Node einfach in einen HTML Absatz da.

```
1 <div ng-switch-default>
2   <p>{{nodeMedia.content}}</p>
3  </div>
```

Listing 4.11: ng-switch für den Defaultwert.

Wenn es sich bei dem HyperMedia Element der Node um ein Bild handelt, ergeben sich gegebenenfalls einige Besonderheiten. Da, falls die Decisions zu dieser Node vom Typ imgarea sind, diese auf dem Bild angezeigt, wie in Abbildung 4.5 die Bereiche Auge, Nase, Mund und Ohr zu sehen sind.

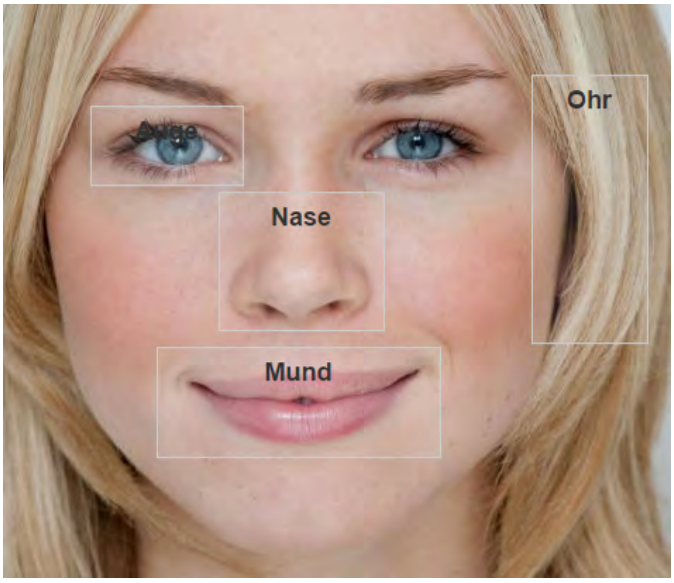

Abbildung 4.5: Darstellung der imgarea Bereiche

Wenn die Nachfolger Node zu einer Decision vom Typ imgarea wiederum vom Typ inline ist, werden die Antworten in der imgarea dargestellt. Die Position und die Größe werden aus dem src Attribute genommen. Dies sind die Werte, welche über das JCrop Plugin ausgelesen werden. In Abbildung 4.6 wurde die Decision Nase ausgewählt, dazu wird in diesen Bereich die inline Decisions angezeigt.

Im Template wird hierfür in der ng-switch für img, bevor das Bild dargestellt wird, über eine ng-repeat für jede decision in der decisionsList zuerst überprüft, ob die Decision vom Typ imgarea ist. Ist das der Fall wird -wenn die inlineDecisionsList leer ist- die imgarea über einen Div mit der CSS-Klasse imgarea-decision dargestellt. Der Inhalt dieses Divs ist die Beschreibung, die bei der Erstellung der imgarea mit angegeben werden muss. Zudem ist dieser div mit einer ng-click Direktive mit der Funktion openDecision

verbunden, sodass bei einem Click auf diesen Bereich die entsprechende Decision geöffnet wird.

Falls die inlineDecisionsList Elemente enthält, wird nicht nur der Rahmen und die Beschreibung der imgarea dargestellt, sondern der Bildausschnitt und unter diesem die einzelnen inline Decisions. Zudem wird das eigentliche Bild der Node mit einem grauen Div abgedunkelt. Die einzelnen Decisions werden durch Bootstrap Buttons dargestellt. Diese Buttons sind über eine ng-click Direktive mit der Funktion openDecision verbunden, sodass die entsprechende Decision bei einem Klick auf diese geöffnet wird.

Der Div der Klasse grey, der für das Abdunkeln des Node Bilds zuständig ist, hat eine ng-click Direktive über welche die Funktion openDecision aufgerufen wird. Dabei wird die Funktion ohne Parameter aufgerufen, sodass die gewählte Decisionauswahl wieder zurückgesetzt wird. Nach den ganzen Direktiven und Sonderfällen für die Darstellung der Decisions wird das eigentliche Bild der Node über einen HTML img Tag eingeblendet.

Im unteren Teil des Templates werden die Decisions dargestellt. Dafür wird für jede decision in der decisionsList ein Bootstrap Panel erstellt. In den Titel des Panels wird - wenn es sich um den Typ txt oder imgarea handelt- das content Attribute des HyperMedia Elements des Decisions genommen. Im anderen Fall wird auf ”Decision X” gesetzt, wobei X durch eine fortlaufende Nummer ersetzt wird. In den Panel Body wird dann das eigentliche HyperMedia Element der Decision dargestellt. Ein Bild wird in einem img Tag dargestellt und für audio ein Audio Tag verwendet.

```
1 <img
2  class="imgarea"
3  style="position: absolute; top:-399px; left:-251px;
4  clip:rect(399px,369px,498px,251px);"
5  ng-src="...
```

Listing 4.12: HTML-Code für das Darstellen eines Bildausschnitts

Eine Besonderheit stellt hierbei wieder der Typ imgarea dar. Bei dieser wird ein Div um den eigentlichen Bildausschnitts gelegt, dieser Div wird über ng-click Direktive mit der Highlight Funktion verbunden, so dass bei einem Klick der Ausschnitt auf dem Node Bild hervorgehoben wird. In diesen Div befindet sich ein HTML img Tag mit dem

gesamten Bild, aus dem HyperMedia Element der Node. Dieses Bild wird aber durch den CSS-Befehl clip und den passenden Werten aus den Daten des JCrop Plugins zugeschnitten, dass nur der eigentliche Bildausschnitt dargestellt wird.

Abbildung 4.6: Darstellung bei Auswahl eines imgarea Bereiches mit inline Decisions.

5

Beispielszenarien

In diesem Kapitel wird zuerst erklärt, wie vorgegangen werden muss, wenn ein neuer Entscheidungsbaum erstellt wird. Danach wird das Szenario 5.2 erklärt, warum diese Arbeit überhaupt entstanden ist. Die Situation, in der ein Migrant selber die richtige fachärztliche Behandlung suchen kann. Als zweites Szenario 5.3, wird ein Selbstdiagnosesystem für Fehler an einem Fahrzeug erklärt.

5.1 Allgemein

Beim Aufrufen der URL wird die Startseite der Applikation geladen, wie in Abbildung 5.1 dargestellt. Auf dieser befindet sich die Übersicht über alle Entscheidungsbäume. Diese werden in einer Liste dargestellt. Über den Button *new Tree* kann ein neuer Entscheidungsbaum erstellt werden.

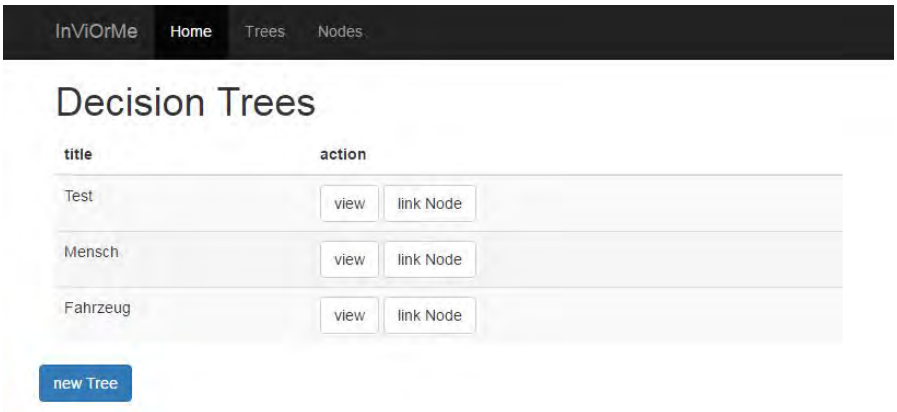

Abbildung 5.1: Startseite der Applikation

5.1.1 Neuen Baum erstellen

Um einen neuen Entscheidungsbaum zu erstellen, wird nach einem Klick auf den Botton *new Tree*, der entsprechende Controller mit dem passenden Template geladen.

Hier muss dem neuen Entscheidungsbaum ein Titel gegeben werden. Dieser wird auf der Startseite in der Tabelle angezeigt. Die Verknüpfung mit der Startnode erfolgt später auf der Übersichtsseite.

5.1.2 Neuen Node erstellen

Um eine neue Node zu erstellen, muss über die Navigation *Nodes* ausgewählt werden. Auf der Übersichtsseite werden alle existierende Nodes in einer Liste angezeigt.

Um eine Neue zu erstellen, muss auf den Button *new Node* geklickt werden. Danach wird das passende Formular geladen. In der Abbildung 5.2 ist dieses Formular dargestellt. Der Node muss eine Beschreibung gegeben werden, diese wird für die Liste auf der Übersichtsliste benötigt. Im Abschnitt *Create new HyperMedia* wird das HyperMedia Element der Node erstellt. Hier kann über das Dropdown Menü der Typ des HyperMedia Elements gewählt werden.

Im Abschnitt *Create new Decision* muss mindestens eine Decision erstellt werden. Über den Button *Add Decision* kann eine weite Decision hinzugefügt werden. Für jede Decision kann ebenfalls über ein Dropdown Menü der Typ des HyperMedia Elements der Decision ausgewählt werden. In dem Beispiel auf Abbildung 5.2 wird als Typ für das HyperMedia Element der Node ein Bild ausgewählt und als Typ für die Decision imgarea, die auf dem Bild der Node direkt gezeichnet werden kann.

5.1.3 Nodes verbinden

Um eine Decision mit einer Nachfolgernode zu verbinden, um die Baumstruktur abzubilden, muss die Nodeübersicht, über die Navigationsleiste geöffnet werden.

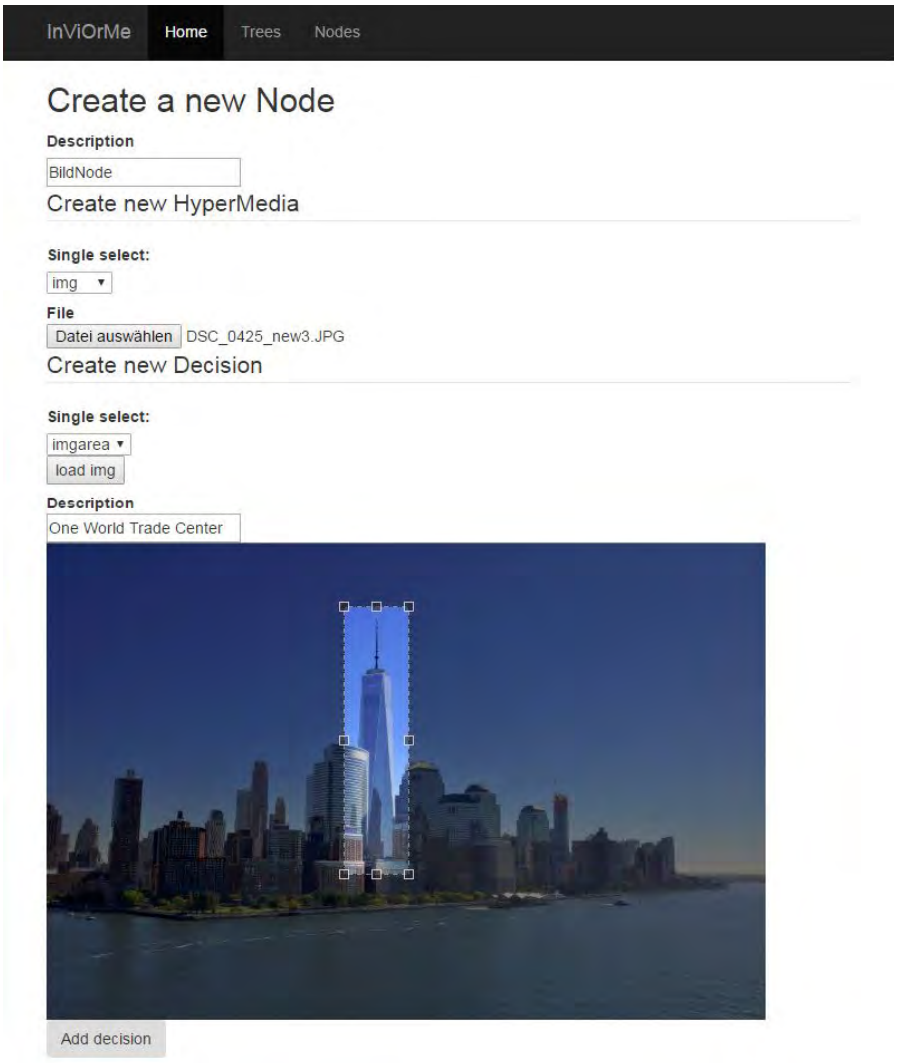

Abbildung 5.2: Erstellen einer neuen Node

Über den Button *view* kann das HyperMedia Element der Node angezeigt werden, dies geschieht über den NodeViewController aus Kapitel 4.2.3. Nach den Klick auf dem Button wird ein *ngDialog* geöffnet und die Informationen zu dieser Node angezeigt.

Über den Button *open* wird die Node geöffnet und es werden alle Decisions zu dieser Node aufgelistet, Abbildung 5.3.

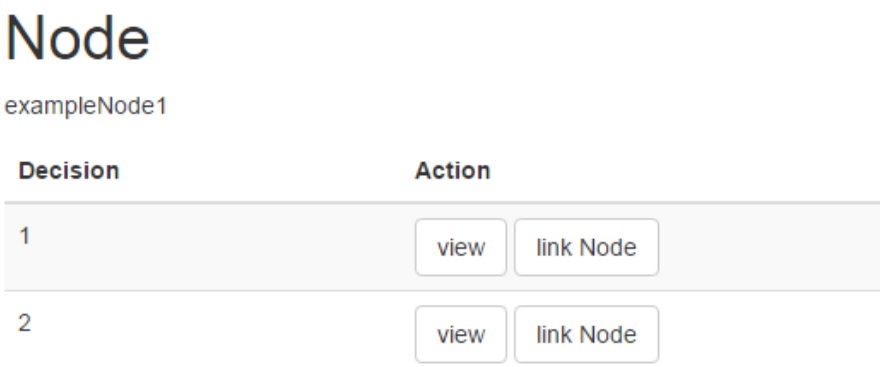

Abbildung 5.3: Übersicht über alle Decisions einer Node

Hier kann wiederum das HyperMedia Element der jeweiligen Decision über einen *ngDialog* geöffnet werden. Dazu muss auf der Übersicht auf den Button *view* geklickt werden.

Abbildung 5.4: Eine Decision mit eine Node verbinden.

Über den Button *link Node* kann eine Decision mit einer Node verbunden werden. Nach einem Klick auf diesen Button, wird ein *ngDialog* geöffnet, Abbildung 5.4. Dieser Dialog beinhaltet den NodeDialogSelectController 4.2.3, dieser listet alle vorhandene Nodes auf, über den Button *select* kann diese Node ausgewählt werden. Diese Node ist dann mit der ausgewählt Decision verbunden.

Um einen Entscheidungsbaum mit einer Startnode zu verbinden, muss zuerst über die die Navigationsleiste *Trees* ausgewählt werden. In der Liste werden alle vorhandenen Entscheidungsbäume aufgelistet. Über den Button *link Node* kann dieser mit einer Node verbunden werden. Dazu wird wieder ein ngDialog geöffnet, über den eine Node über *select* ausgewählt werden kann.

5.2 Szenario: Menschlicher Körper

In diesem Beispiel Szenario geht es um die Problemstellung woraufhin diese Arbeit entstanden ist. Wenn Flüchtige nach Deutschland einreisen, haben diese oft einen langen und schwierigen Weg hinter sich. Um zum Beispiel Verletzungen oder andere gesundheitliche Probleme vom richtigen Facharzt behandeln lassen zu können, müssen diese zuerst erfasst werden. Danach soll anhand der Verletzung beziehungsweise der Beschwerde der richtige Facharzt gefunden werden.

Dazu muss die Verletzung beziehungsweise die Beschwerde von jemanden nonverbal erfasst werden. Hier kommt das Problem der Sprachbarriere zu gelten. Zum einen muss der Migrant seine Beschwerde beschreiben können und zum anderen muss derjenige, der die Zuordnung von Beschwerde zu Facharzt übernimmt, dies auch verstehen.

Aus dieser Problematik entstand die Idee, dies in einem Entscheidungsbaum zu vereinfachen. Der Migrant kann über einen Touchscreen auf einem Körper auswählen, wo er welche Beschwerden hat, diese werden erfasst und es kann direkt mitgeteilt werden, welchem Facharzt er aufsuchen sollte.

Dieses Szenario ist nur Beispielhaft. Hier soll nur die Grundgedanke symbolisiert werden. Um es in der Praxis einzusetzen, sollte es mit entsprechender Fachkompetenz vervollständigt werden.

Für das gedachte Szenario sind die deutschen Begriffe natürlich nicht sehr sinnvoll. Hier könnten diese durch passende Piktogramme oder Bilder ersetze werden. Oder es kann der Baum komplexer aufgebaut werden, sodass zu beginn eine Sprache gewählt werden kann.

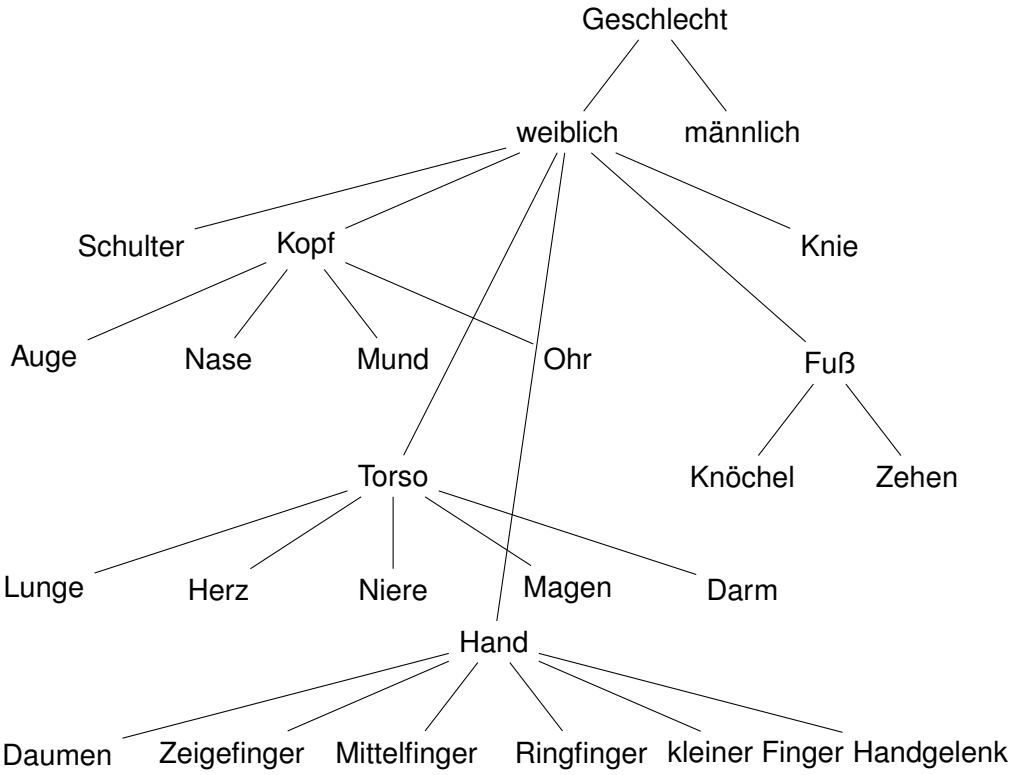

Abbildung 5.5: Darstellung der Nodes für Szenario 1

In Abbildung 5.5 ist die Baumstruktur der Beispielanwendung abgebildet. Es sind nur die Nodes aufgelistet. Die finalen Decisions wurden aufgrund der Übersichtlichkeit weggelassen. Zudem werden nur die Knoten für das Geschlecht weiblich angezeigt, bei männlich sind weitgehend dieselben Nodes vorhanden.

In der ersten Entscheidung muss zwischen männlich und weiblich entschieden werden. Je nach dieser Entscheidung wird ein männlicher oder ein weiblicher Körper dargestellt.

Auf diesen kann zwischen verschiedenen imgarea Decisions gewählt werden. Hier kann zwischen Kopf, Schulter, Torso, Hand, Knie und Fuß gewählt werden. In Abbildung 5.6

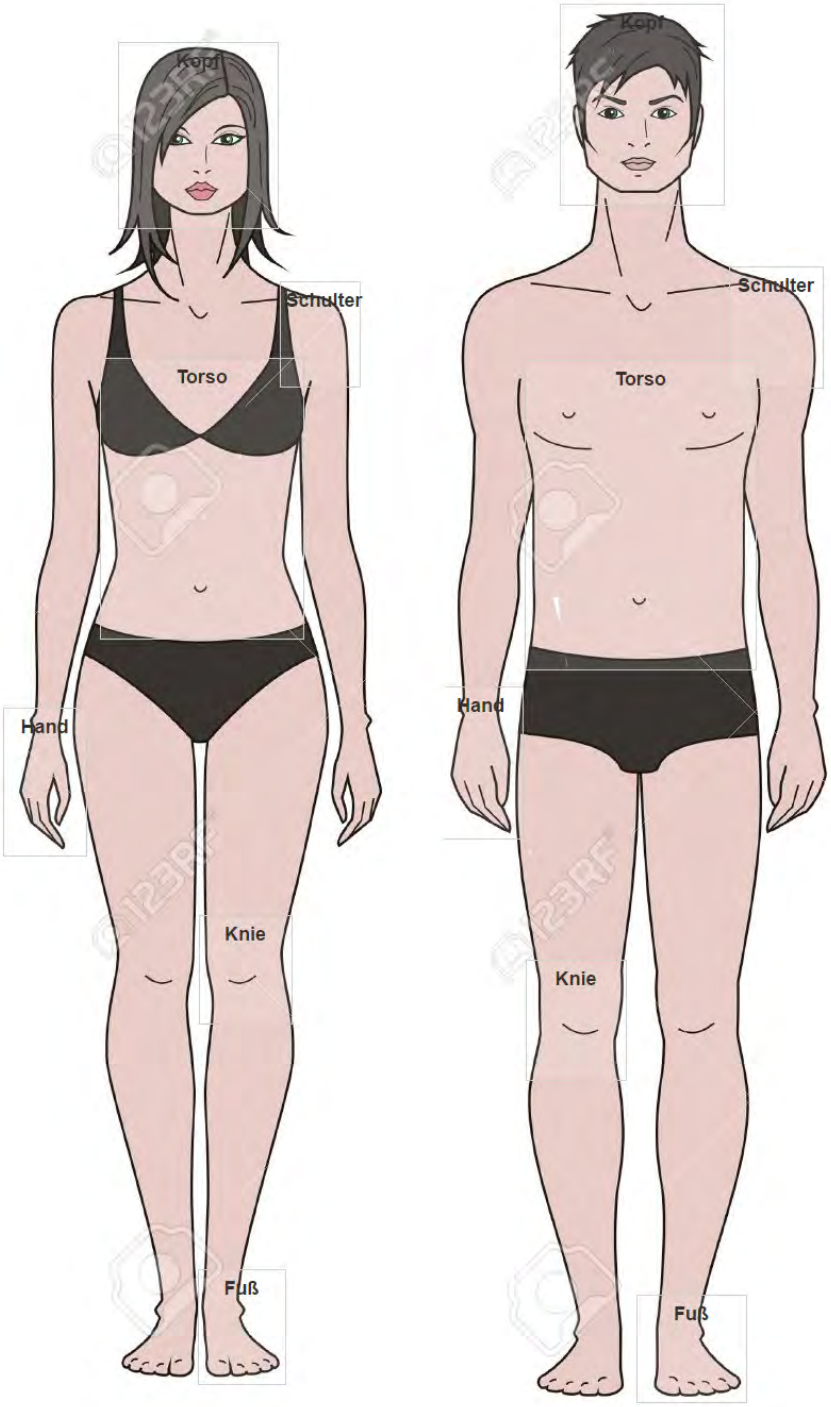

Abbildung 5.6: Körper mit imgarea Decisions [Bat16]

sind beide Nodes neben einander Dargestellt, wobei natürlich nur die Node angezeigt wird, welche ausgewählt wurde.

Bei den Decisions Schulter und Knie handelt es sich um inline Decisions. In Abbildung 5.7 sind die inline Decisions für Schulter dargestellt, dies sind Schmerzen und Bewegungseinschränkung.

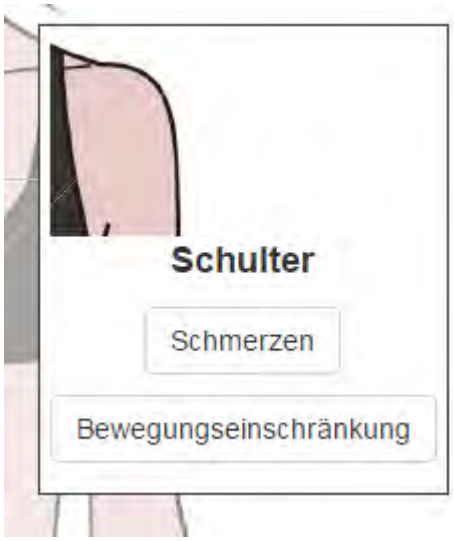

Abbildung 5.7: Inline Decisions für die Node Schulter

Für das Knie sind die inline Decisions in Abbildung 5.8 dargestellt. Hier kann geschwollen, gebrochen und gerötet ausgewählt werden.

Wird die Decision Kopf ausgewählt wird eine neue Node geladen. In Abbildung 5.9, ist dies dargestellt. Auf dieser Node werden wieder imgarea als Decisions gewählt werden. Diese sind Auge, Nase, Mund und Ohr. Die Decision der einzelnen imgarea sind wieder inline Decisions. Die Decisions für Auge sind gerötet, Sehstörung, tränen und jucken. Für die Node Nase sind diese Gebrochen, Schmerzen und Schnupfen. Für Mund Zahnschmerzen und Schluckbeschwerden. Und für die Node Ohr sind die Decisions Tinnitus, Schwerhörigkeit, Taubheit und Schmerzen

Wird die imgarea Torso ausgewählt, wird die Node des Torsos geladen. Hier kann wieder zwischen verschiedenen imgareas gewählt werden. Wie in Abbildung 5.10 zu sehen ist. Es gibt Lunge, Herz, Niere, Magen und Darm. Die Entscheidungen zu diesen Bereichen

Abbildung 5.8: Inline Decisions für die Node Knie

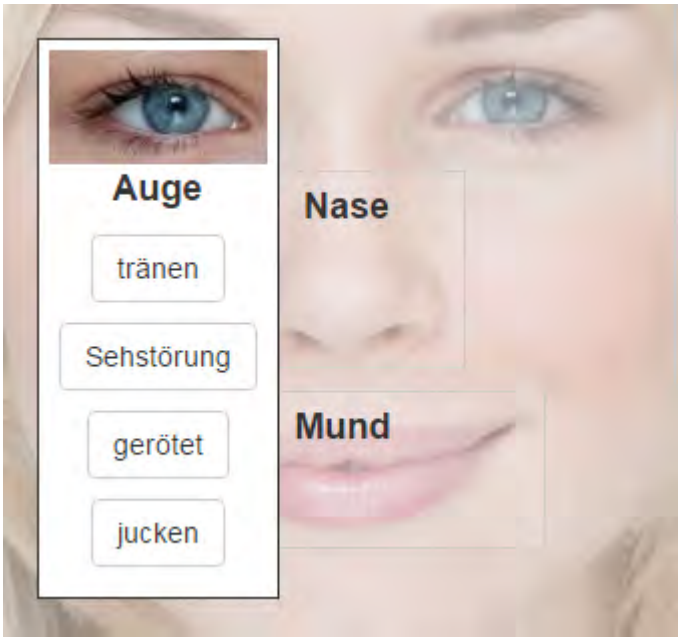

Abbildung 5.9: Inline Decisions für die Node Kopf [Har16]

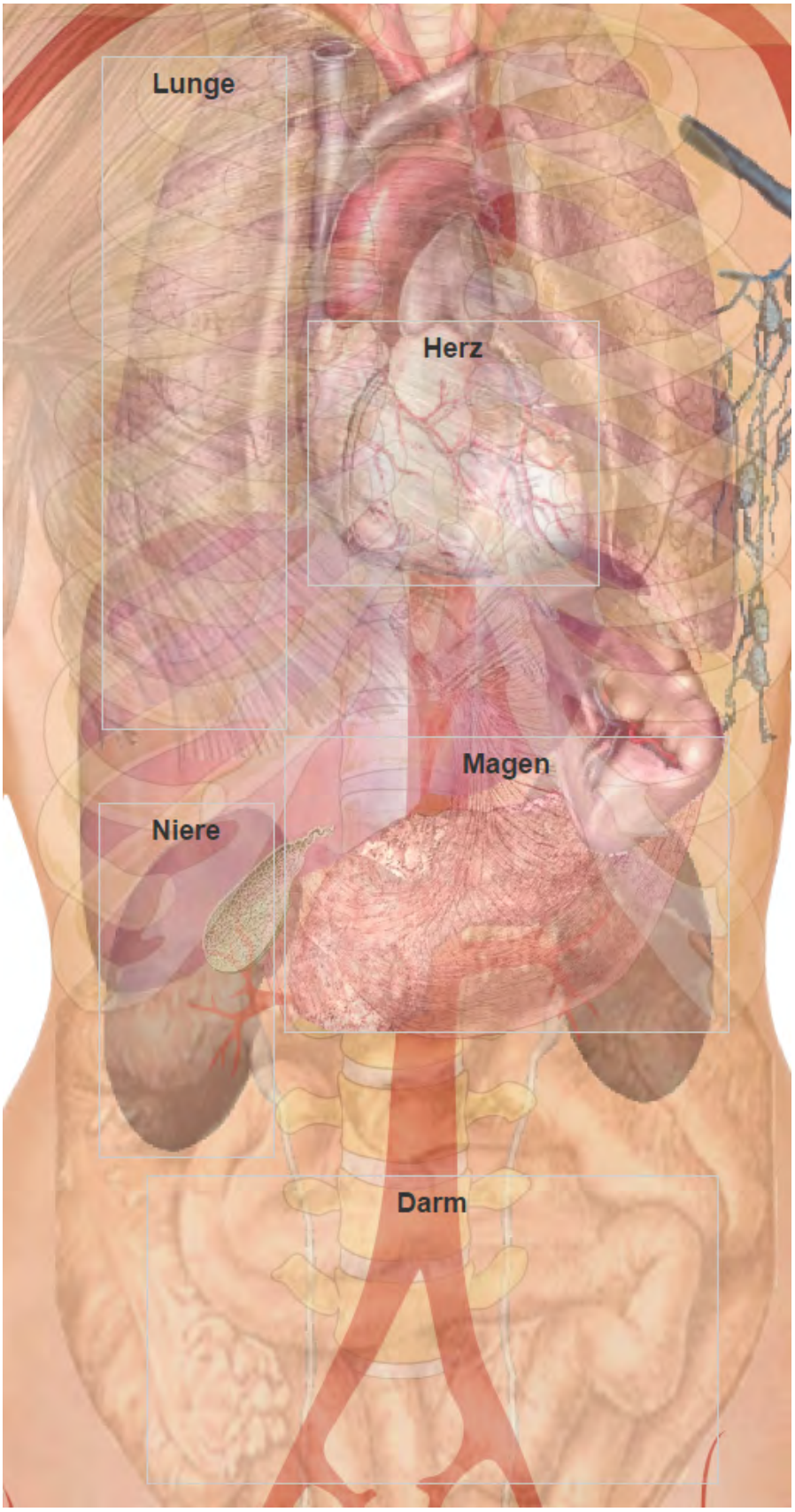

Abbildung 5.10: Inline Decisions für die Node Torso [Hä16]

sind ebenfalls inline Decisions. Für die Node Lunge, Niere, Darm und Magen sind die Decisions drücken, Schmerzen, Krämpfe und stechen. Die Decisions für die Node Herz sind Bluthochdruck, drücken, stechen, Schmerzen, rasen und Krämpfe.

Beim Fuß gibt es die Node Knöchel und die Node Zehen. Die inline Decisions dazu sind jeweils gebrochen, gerötet und geschwollen.

Das Szenario könnte auch erweitert werden, mit der entsprechenden Anpassungen der Applikation, sodass Vitale-Sensoren, von einem Smartphone, mit in den Entscheidungsprozess einfließen [SSP+13]. So könnte die Herzfrequenz überprüft werden und je nach Wert entsprechende Fragen bezüglich des Herzkreislaufs gefragt werden.

5.3 Szenario: Fahrzeugdiagnosesystem

Als weiteres Szenario dient beispielhaft ein Diagnosesystem für ein modernes Kraftfahrzeug. Hier könnte es zwei Wege geben, wie dieses eingesetzt wird.

Zum einen, könnte es Online auf der Internetseite einer Werkstatt sein. So kann der Kunde, von zu Hause aus, seine Probleme alle aufführen und einen Auftrag an die Werkstatt schicken. Diese erhält diesen Fehlerbericht und kann gegebenenfalls die nötigen Ersatzteile bestellen und sobald der Kunde das Auto bringt, die aufgeführten Fehler reparieren.

Die Anwendung kann auch in der Werkstatt laufen. Kommt der Kunde in die Werkstatt, kann ein Mitarbeiter die entsprechende Fehler im Servicedialog erfassen und einen Fehlerbericht direkt anfertigen.

In der Abbildung 5.11 sind die Nodes des Baumes abgebildet. Auch hier sind nur ein paar Nodes aufgelistet. Für diesen Baum gibt es natürlich noch beliebig viele Nodes hinzugefügt werden, um ein reales Beispiel zu bekommen.

Hier ist es ähnlich zum Szenario vom menschlichen Körper 5.2. Dieselben Nodes die unter dem Element *Hersteller A* abgebildet sind, gibt es für den Knoten *Hersteller B* ebenfalls. Die endgültigen Decisions wurden auch hier, aufgrund der Übersichtlichkeit, weggelassen.

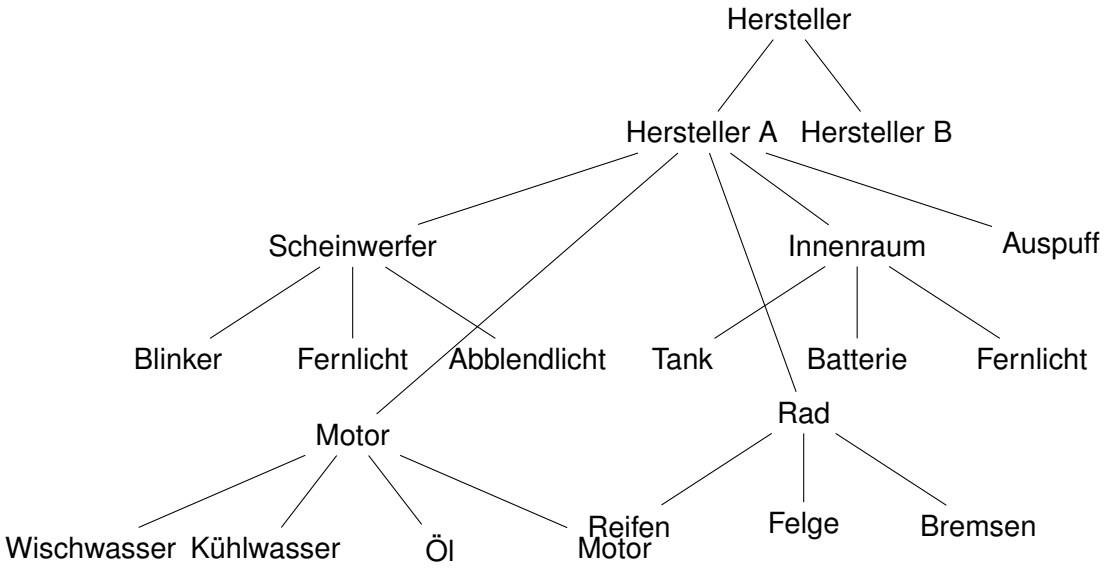

Abbildung 5.11: Darstellung der Nodes für Szenario 2

Nach der Herstellerwahl, wird eine Node mit dem entsprechenden Fahrzeug dargestellt. In der Abbilung 5.12 wird exemplarisch die Node für *Hersteller A* dargestellt.

Abbildung 5.12: Darstellung der Node BMW mit inline Decisions [tes16]

Auf der Node können, wieder über imgareas, verschiedene Decisions ausgewählt werden. Hier kann zwischen Scheinwerfer, Motor, Innenraum, Auspuff und Rad gewählt werden.

Beim Scheinwerfer gibt es die Decisions Blinker, Fernlicht und Abblendlicht. Für die es wiederum als Decision Birne defekt und Höheneinstellung gibt.

Bei der Node Motor 5.13 gibt es die imgarea Wischwasser, Kühlwasser, Öl und den Motor. Für die Flüssigkeiten gibt es die inline Decisions Nachfüllen und Auswechseln.

Für die Node Rad kann zwischen Reifen und Felge gewählt werden, wobei bei Reifen die Decisions Platt und Abgefahren gibt.

Wenn die inline Decision Auspuff auf dem Fahrzeugbild gewählt wird, werden drei Bilder mit verschieden farbigen Abgasqualm dargestellt. Hier kann ein Bild ausgewählt was am ehesten zutrifft. Der Qualm ist auf einem Bild weiß, auf einem weiteren bläulich und auf dem letzten Bild ist er gräulich.

Als weitere Node gibt es den Innenraum. Hier wird eine Instrumententafel dargestellt, auf der die verschiedenen Kontroll- und Warnleuchten abgebildet sind. Um die Komplexität in Grenzen zu halten, wurde in diesem Beispiel nicht alle Leuchten mit einer imgarea versehen.

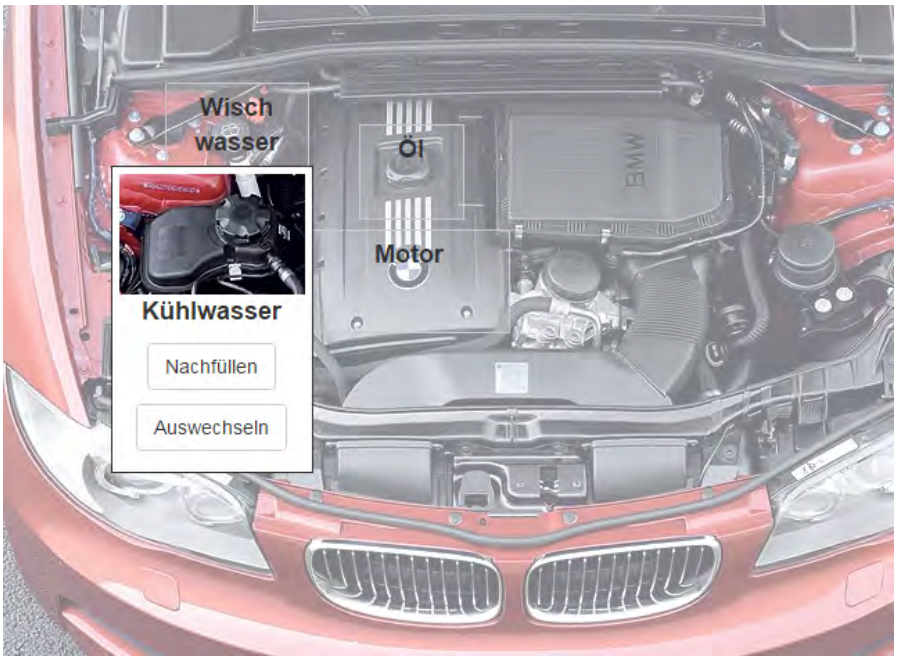

Abbildung 5.13: Darstellung der Nodes für Motor

6

Zusammenfassung und Ausblick

In diesem Kapitel wird die Masterarbeit noch einmal zusammengefasst und ein Ausblick geben, wie die entstandene Anwendung noch weiter entwickelt werden kann.

6.1 Zusammenfassung

In dieser Arbeit sollte zuerst ein Konzept entwickelt werden welches zum einen ein generische und hierarchische Datenmodell für die Speicherung beinhaltet und auf der anderen Seite eine interaktive Selektion von ortsabhängigen Merkmalen ermöglicht. Anschließen sollte diese Konzept realisiert werden.

Die in dieser Arbeit entstandene Anwendung deckt beide Bereiche ab. Der entstandene Server ist für die Speicherung zu ständig. Da die Datenstruktur für dieses Datenmodell einem Entscheidungsbaums entspricht, ist es möglich einzelne Nodes zu erstellen und diese zu einer Baumstruktur über die Decisions zu verbinden. Da der Inhalt der Node beziehungsweise der Decision in einem HyperMedia Element gekapselt ist, ist das Datenmodell unabhängig des Inhaltes. So kann das generische Modell jederzeit durch neue HyperMedia Element erweitert werden. Die Baumstruktur bringt eine Hierarchie in das Datenmodell.

Der Client ist in einer Webapplikation entstanden. Dies hat den großen Vorteil der Plattformunabhängigkeit. Durch die Verwendung von Frameworks ist eine moderne Single Page Application entstanden, die ohne Unterbrechung des Präsentationflusses ein einheitliches Bild ergibt.

Durch die neu entwickelten HyperMedia Element imgarea und inline Decision, kann eine neue Interaktivität in die Auswahl von Entscheidungen gebracht werden.

6.2 Ausblick

Dadurch, dass der Server nur für die Datenhaltung zuständig und vollkommen unabhängig der Präsentation ist, könnte auch ein anderer Client programmiert werden. Zum Beispiel eine Applikation für Smartphones [SRP+15] [GSP+14] [SPSR15].

Es könnte auch eine neue Backend Software zum bequemeren Erstellen eines Entscheidungsbaum geschrieben werden. Bei der über einen grafischen Editor, ähnlich wie bei der Zingtree Anwendung, zum Beispiel die einzelnen Nodes per drag-and-drop verbunden werden können.

Die Anwendung kann auch jederzeit um neue HyperMedia Elemente erweitert werden. So kann sich neue Element überlegt werden, um die Anwendung noch Interaktiver gestalten. Oder neue Element, um das Einsatzgebiet zu erweitern. So könnte er um Fragebogen Element erweitert werden, um einen Fragebogen abzubilden, bei dem die Fragen von eine abhängig sind und je nach Auswahl andere Fragen folgen [SSP+14] [SSPR15].

Ein weiterer Punkt, welcher in dieser Anwendung fehlt, ist ein Bearbeiten und eine Löschenfunktion. Im Moment können nur neue Nodes und neue Decisions erstellt werden, aber nicht mehr im Nachhinein geändert werden.

Ein weiterer Aspekt, welcher sehr wichtig ist, bevor die Anwendung produktiv eingesetzt werden kann, ist der Punkt Sicherheit. Zum einen, sind die CORS Filter deaktiviert und zum anderen wäre eine Authentifizierung und Autorisierung wichtig. Sodass nicht jeder neue Nodes anlegen oder gar löschen kann.

Zum Schluss sollte noch die Möglichkeit geschaffen werden, die finalen Entscheidungen weiter zu verarbeiten. In der Anwendung werden diesr zwar erkannt, aber nicht weiter behandelt. So müsste, wie im Szenario des menschlichen Körpers 5.2 die finalen Entscheidungen mit dem entsprechenden Arzt verbunden werden. Oder im zweiten Sze-

nario, des Fahrzeugdiagnosesystem 5.3, die erfassten Mängel in einem Fehlerbereich zusammengefasst werden.

Literaturverzeichnis

[Bat16] BATALOVA, Olga: *Darstellung der menschlichen Figur, Mann Frau, Kind*. http://de.123rf.com/photo_19087532_darstellung-der-menschlichen-figur-s-mann-frau-kind.html?fromid=eHdHVE0zak5wTTZKZXg3TmdEUVowdz09. Version: 22.02.2016

[Fra12] FRAIN, Ben: *Responsive Web Design with HTML5 and CSS3*. Packt Publishing, 2012. – ISBN 1849693188

[Goo16] GOOGLE, Inc.: *AngularJS API Docs*. https://docs.angularjs.org/. Version: 30.04.2016

[GSP+14] GEIGER, Philip ; SCHICKLER, Marc ; PRYSS, Rüdiger ; SCHOBEL, Johannes ; REICHERT, Manfred: Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, 2014, 383–394

[Har16] HARRIS, Paul: *The girl who works in a chip shop who has 'Britain's most beautiful face'*. http://www.dailymail.co.uk/femail/article-2132896/Florence-Colgate-Girl-Britains-beautiful-face.html. Version: 22.02.2016

[Hav14] HAVIV, Amos Q.: *MEAN Web Development*. Packt Publishing - ebooks Account, 2014. – ISBN 1783983280

[Hol15] HOLMES, Simon: *Getting MEAN with Mongo, Express, Angular, and Node*. Manning Publications, 2015. – ISBN 1617292036

[Hä16] HÄGGSTRÖM, Mikael: *Female with organs*. https://commons.wikimedia.org/wiki/File:Female_with_organs.png. Version: 18.03.2016

[Inc16] INC. PIVOTAL SOFTWARE: *spring.io*. https://spring.io/. Version: 11.01.2016

[JR08] JIANG, Prof. X. ; ROTHAUS, Kai: *Mustererkennung*. http://cvpr.uni-muenster.de/teaching/ws08/mustererkennungWS08/script/ME08.pdf. Version: 2008

[Qui86] QUINLAN, J. R.: Induction of Decision Trees. In: *Mach. Learn.* 1 (1986), März, Nr. 1, 81–106. http://dx.doi.org/10.1023/A:1022643204877. – DOI 10.1023/A:1022643204877. – ISSN 0885–6125

[Res16] RESTAPITUTORIAL.COM: *Using HTTP Methods for RESTful Services*. http://www.restapitutorial.com/lessons/httpmethods.html. Version: 21.04.2016

[Ret16] RETTING, Daniel: *Im Zwiespalt – Warum fallen leichte Entscheidungen schwer?* http://www.alltagsforschung.de/im-zwiespalt-warum-fallen-leichte-entscheidungen-schwer/. Version: 03.05.2016

[Sco15] SCOTT, Emmit: *SPA Design and Architecture: Understanding Single Page Web Applications*. Manning Publications, 2015. – ISBN 1617292435

[SEL16] SELFHTML: *HTML/Multimedia und Grafiken/verweissensitive Grafiken*. https://wiki.selfhtml.org/wiki/HTML/Multimedia_und_Grafiken/verweissensitive_Grafiken. Version: 06.05.2016

[SLP11] SAWADE, Christoph ; LANDWEHR, Niels ; PRASSE, Paul: *Entscheidungsbäume*. http://www.cs.uni-potsdam.de/ml/teaching/ws11/ml/Entscheidungsbaeume.pdf. Version: 2011

[SPSR15] SCHICKLER, Marc ; PRYSS, Rüdiger ; SCHOBEL, Johannes ; REICHERT, Manfred: An Engine Enabling Location-based Mobile Augmented Reality Applications. Version: 2015. http://dbis.eprints.uni-ulm.de/1137/. In: *10th International Conference on Web Information Systems and Technologies (Revised Selected Papers)*. Springer, 2015 (LNBIP 226), 363–378

[Spu13] SPURLOCK, Jake: *Bootstrap*. O'Reilly Media, 2013. – ISBN 1449343910

[SRP+15] SCHICKLER, Marc ; REICHERT, Manfred ; PRYSS, Rüdiger ; SCHOBEL, Johannes ; SCHLEE, Winfried ; LANGGUTH, Berthold: *Entwicklung mobiler Apps: Konzepte, Anwendungsbausteine und Werkzeuge im Business und E-Health*. Springer-Verlag, 2015

[SSP+13] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; NIENHAUS, Hans ; REICHERT, Manfred: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, 2013, 509–518

[SSP+14] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; MAIER, Fabian ; REICHERT, Manfred: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, 2014, 371–382

[SSPR15] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; REICHERT, Manfred: Process-Driven Data Collection with Smart Mobile Devices. Version: 2015. http://dbis.eprints.uni-ulm.de/1136/. In: *10th International Conference on Web Information Systems and Technologies (Revised Selected Papers)*. Springer, 2015 (LNBIP 226), 347–362

[Sü15] SÜDDEUTSCHE ZEITUNG DIGITALE MEDIEN GMBH: *Mehr Zufall als Diagnose*. http://www.sueddeutsche.de/gesundheit/symptom-checker-mehr-zufall-als-diagnose-1.2556320. Version: 01.12.2015

[TB16] TARASIEWICZ ; BÖHM: *Moderne Web-Anwendungen mit AngularJS*. http://nodecode.de/vortrag-moderne-web-anwendungen-mit-angularjs-deutsch. Version: 26.01.2016

[tes16] TEST-DRIVEZ.COM: *BMW M4*. http://test-drivez.com/data_images/makers/bmw/bmw-07.jpg. Version: 05.03.2016

[The16] THE NEW YORK TIMES: *512 Paths to the White House*. http://www.nytimes.com/interactive/2012/11/02/us/politics/paths-to-the-white-house.html. Version: 06.05.2016

[Tut16] TUTORIALS POINT: *HTTP - Methoden*. http://www.tutorialspoint.com/de/http/http_methods.htm. Version: 21.04.2016

[Twi16] TWITTER, Inc.: *Grid system*. http://getbootstrap.com/css/#grid. Version: 27.03.2016

[VB15] VARANASI, Balaji ; BELIDA, Sudha: *Spring REST*. Apress, 2015. – ISBN 1484208242

[YS95] YUAN, Yufei ; SHAW, Michael J.: Induction of fuzzy decision trees. In: *Fuzzy Sets and Systems* 69 (1995), Nr. 2, 125 - 139. http://dx.doi.org/http://dx.doi.org/10.1016/0165-0114(94)00229-Z. – DOI http://dx.doi.org/10.1016/0165–0114(94)00229–Z. – ISSN 0165–0114

[Zin16] ZINGTREE: *Interactive Decision Trees*. http://zingtree.com/. Version: 05.05.2016

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Name: Benedikt Löffler Matrikelnummer: 855603

Erklärung

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

Ulm, den .....

Benedikt Löffler