



# Decisio in the Wild

## Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autor(en): Raphael Horber, Sascha Kaufmann  
Betreuer: Prof. Dr. Eduard Glatz  
Projektpartner: Forventis GmbH Zürich, Panter AG Zürich  
Experte: Roberto Pajetta  
Gegenleser: Prof. Hans Rudin



## Aufgabenstellung zur Bachelorarbeit FS 2015

„Decisio in the Wild“

Gruppe: Horber, Raphael / Kaufmann, Sascha

### Kurzbeschreibung

Entscheidungen sind in der heutigen Multi-Options Gesellschaft schwerer zu treffen als in vergangenen Zeiten. Unter anderem müssen immer mehr Entscheidungen durch Einzelpersonen getroffen werden, da niemand anderes dies übernimmt. Eine computergestützte Lösung soll Entscheidungswilligen, jedoch überforderten Usern helfen, richtige Entscheidungen zu treffen, indem sie Handlungsempfehlungen abgibt. Richtig, in unserem Sinne, ist eine Entscheidung dann, wenn sie mit den eigenen Motiven und Motivationen in Einklang steht. Die Software erleichtert deshalb eine konkrete Entscheidungssituation mit der persönlichen Motivlage des Users.

### Hintergrund

Die Software basiert auf der Vermutung, dass eine Entscheidung umso besser ist, je näher sie am Motivprofil eines Entscheiders liegt. Das Motivprofil basiert auf dem *Zürcher Modell der sozialen Motivation von Norbert Bischof [1]* zu dem ein validierter Fragebogen vorliegt. Dieser wird von der Software zur Erstellung des Motivprofils verwendet. Der Vergleich des Motivprofils mit einer Entscheidungssituation erfolgt über eine Rating-Liste. Die Handlungsempfehlungen basieren auf dem *Forventis durchBlick ©-Prinzip [2]*, das eine allgemeine Entscheidungsregel verkörpert. Diese Regel wird durch Werte aus der Rating-Liste modifiziert und so individuell auf den User angepasst.

### Aufgabe

In einer Vorarbeit [3] wurde eine Android App entwickelt, mit der die gewünschte Funktionalität grossenteils realisiert ist. In der Folgearbeit geht es nun darum folgende weiterführenden Ziele zu erreichen:

- Entwicklung einer serverseitigen Decisio Backend-Applikation
- Ausbau des Decisio-Prototyps zur Produktreife
- Portierung der Decisio-App auf eine weitere Mobilplattform (Android Watch oder Windows Phone).

## Allgemeine Vorgaben

Die Arbeit ist gemäss den allgemeinen Vorgaben [4] durchzuführen. Dies beinhaltet auch Vorgaben zur Berichtsgestaltung.

## Termine

16. Feb. 2015	Arbeitsbeginn
12. Juni 2015	Abgabe des Berichts an den Betreuer (bis 17:00)

Weitere Termine: siehe Terminangaben auf dem HSR-Web (intern).

## Betreuung

Betreuer: Prof. Dr. Eduard Glatz

Während der Durchführung der Arbeit findet nach Möglichkeit regelmässig jede Woche eine Besprechung mit dem Betreuer statt. Dazu werden entsprechende Termine bei Arbeitsbeginn festgelegt. Die Studierenden verfassen vor jeder Besprechung eine Traktandenliste, die spätestens am Vortag dem Betreuer per Email zu senden ist. Jede Besprechung wird von den Studierenden in einem Protokoll dokumentiert.

## Praxispartner

Forventis GmbH

Panter AG

## Referenzen

- [1] Wikipedia; „Zürcher Modell der sozialen Motivation“; Available: [http://de.wikipedia.org/wiki/Z%C3%BCrcher\\_Modell\\_der\\_sozialen\\_Motivation](http://de.wikipedia.org/wiki/Z%C3%BCrcher_Modell_der_sozialen_Motivation)
- [2] Forventis GmbH; „Das Forventis durchBlick®-Prinzip“; Available: <http://www.forventis.ch/durchblick.html>
- [3] Horber, R.; Kaufmann, S.; „Decisio - App für die optimale Entscheidungsfindung“; Studienarbeit HS 2014; HSR - Hochschule für Technik Rapperswil.
- [4] Glatz, E.; „Vorgaben zur Arbeitsdurchführung“; Ausgabe des 9. Februar 2015.

Rapperswil, den 9. Feb. 2015



Eduard Glatz

## Abstract

---

Die persönliche Entscheidungshilfe "Decisio", wurde in unserer Studienarbeit als Android App in Form eines Prototyps entwickelt. Decisio gibt anhand des Motivprofils des Benutzers, welches durch einen Fragebogen ermittelt wird, auf den Benutzer zugeschnittene Handlungsempfehlungen ab. Diesen Prototyp galt es in dieser Arbeit zur Produktreife auszubauen.

Der Hauptfokus unserer Arbeit bestand darin, den in unserer Studienarbeit eingesetzten SFTP-Server durch eine serverseitige Back-End-Applikation abzulösen. Diese soll es den Praxispartnern ermöglichen, die von der Android App zu synchronisierenden Daten einfacher zu erfassen. Neben diesen beiden Hauptzielen, war auch noch eine Portierung der Decisio-App auf eine weitere Mobilplattform (Android Watch oder Windows Phone) gewünscht.

Durch die gute Zusammenarbeit mit den Praxispartnern nach dem agilen Projektvorgehen Scrum entstand eine ansprechende AngularJS Back-End-Applikation, mit welcher man die zu synchronisierenden Daten bearbeiten kann. Die Synchronisierung der Android App wird über einen mit Jersey implementierten RESTful Web Service abgewickelt.

An der Android App wurden neben der Synchronisierung noch einzelne Feinheiten verbessert und zusätzliche kleinere Funktionen eingebaut. Für den weiteren Ausbau des Funktionsumfangs, sowie die Portierung auf eine weitere Mobilplattform blieb am Ende aber leider keine Zeit mehr übrig.

Mit dem entwickelten Produkt lassen sich gute Demonstrationen durchführen. Für die Vermarktung des Produkts ist jedoch die Erfassung weiterer vorgegebener Entscheidungssituationen über die Back-End-Applikation, oder eine Weiterentwicklung der App mit der Funktionalität zum Erstellen von neuen Entscheidungssituationen durch den Benutzer notwendig. Dies ist durch die erstellten Anleitungen und die Dokumentation für die Praxispartner aber gut realisierbar.

## Management Summary

---

### Ausgangslage

In unserer Studienarbeit entwickelten wir die Android App "Decisio". Sie hilft der persönlichen Entscheidungsfindung. Anhand des Motivprofils des Benutzers gibt Decisio auf den Benutzer zugeschnittene Handlungsempfehlungen ab. Das Motivprofil wird anhand eines Fragebogens ermittelt, welcher vorgängig vom Benutzer auszufüllen ist. Diese App erreichte in der Studienarbeit jedoch lediglich die Form eines Prototyps und ist somit noch nicht marktreif.

Die Android App verwendet viele Daten, welche man dynamisch ändern will ohne eine neue App Version zu veröffentlichen. In der Studienarbeit lösten wir dieses Problem mit XML-Dateien, welche von einem Server heruntergeladen wurden. Der Administrator musste diese Dateien von Hand ändern und sich an die XML Spezifikation halten, damit die Aktualisierung auf den Geräten auch korrekt funktioniert. Diesen Missstand wollte man beheben. Dazu soll eine Web-Applikation entwickelt werden, mit welcher man die zu verwendenden Daten ändern kann, ohne sich um die XML Spezifikation kümmern zu müssen.

### Vorgehen

Der Hauptfokus dieser Arbeit bestand darin, die Aktualisierung der XML-Dateien abzulösen. Dadurch muss der Administrator, also die Praxispartner, nicht mehr die XML-Dateien von Hand ändern. Dies wurde mit einer serverseitigen Back-End-Applikation realisiert. Diese besteht aus zwei Teilen. Der erste Teil ist der RESTful Web Service und der zweite Teil die Decisio Administration. Der Service behandelt alle Anfragen der Administration und der Android Clients. Durch Anfragen der Administration an den Service aktualisiert dieser die Daten in der Datenbank. Fragt der Client eine Ressource an, wie z.B. die Entscheidungssituationen, generiert der Service anhand der Daten aus der Datenbank entsprechende XML-Daten. Diese werden an die Android App zurückgesendet. Das Android Gerät kann diese Daten, wie ehemals die XML-Dateien, parsen und seine lokalen Daten entsprechend anpassen.

Zwei weitere Ziele waren geplant. Zum einen wollte man die Android App zur Marktreife ausbauen als auch ihren Funktionsumfang erweitern. Zum anderen sollte eine Portierung auf eine weitere mobile Plattform (Android Watch oder Windows Phone) stattfinden.

Das Projektmanagement basierte auf dem Scrum Framework. Durch diese Variante war eine enge Zusammenarbeit mit den Praxispartnern möglich. Wünsche oder Änderungen von frühen Ideen konnten dadurch teilweise einfließen bevor man mit der eigentlichen Umsetzung begann.

### Ergebnisse

Durch die gute Zusammenarbeit mit den Praxispartnern entstand eine ansprechende AngularJS Back-End-Applikation mit der sich die zu synchronisierenden Daten bequem bearbeiten lassen. Durch die Verwendung von Angular Material konnte sie mit mässigem Aufwand zeitgemäss gestaltet werden.

Die Synchronisierung mit den Android Clients erfolgt über einen RESTful Web Service der mit Jersey implementiert wurde. Dieser stellt die entsprechende Funktionalität zur Verfügung, sodass die XML-Dateien nicht mehr nötig sind.

Bei der Android App wurde die Synchronisierung überarbeitet. Bei der alten Variante mit den XML-Dateien wurden bei Datenänderungen die lokalen Daten auf dem Gerät komplett gelöscht und die neuen eingefügt. Dies hatte zur Folge, dass wenn nur ein Zeichen des Motivprofil-Fragebogens änderte, der komplette Fragebogen neu eingefügt wurde. Durch den Einsatz einer Datenbank im Back-End kann mitverfolgt werden, welche Daten sich wann änderten. Somit kann der Service nur die aktualisierten Daten senden. Diesem Vorgehen wurde die App entsprechend angepasst. Bei geringen Änderungen der Daten benötigt die App Version der Bachelorarbeit bei einer weiteren

Synchronisierung (nicht die Initial-Synchronisierung beim ersten Start) nur noch einen Bruchteil gegenüber der App Version aus der Studienarbeit.

Zusätzlich konnten einzelne Feinheiten verbessert und zusätzlich kleinere Funktionen eingebaut werden. Auch wurde das Design komplett überarbeitet. Die App verwendet nun das Material Theme.

Für die Portierung auf eine weitere Mobilplattform blieb am Ende leider keine Zeit mehr übrig.

## **Ausblick**

Mit den entwickelten Produkten lassen sich gute Demonstrationen durchführen. Die Verwaltung der dynamischen Daten ist nun einiges einfacher zu handhaben. Für die Vermarktung des Produkts reicht dies jedoch noch nicht. Zurzeit sind weniger als zehn vorgegebene Entscheidungssituationen vorhanden, damit lassen sich noch nicht viele Bedürfnisse potentieller Benutzer abdecken. Zum anderen ist noch der Ausbau des Funktionsumfangs der App ausstehend. Das Erstellen eigener Entscheidungssituationen fehlt leider noch immer. Durch diesen Nachteil fällt die Tatsache mit den wenigen vorgegebenen Entscheidungssituationen noch mehr ins Gewicht.

Das Beheben dieser Missstände ist mit den erstellten Anleitungen und Dokumentationen, sowie der Administration für die Praxispartner aber gut möglich.

## Inhaltsverzeichnis

---

Aufgabenstellung .....	1
Abstract.....	3
Management Summary .....	4
Inhaltsverzeichnis .....	6
I. Technischer Bericht .....	7
1. Einleitung.....	7
1.1 Einführung anhand eines konkreten Benutzers.....	7
1.2 Übersicht .....	18
2. Anforderungs-Analyse .....	20
2.1 Administration.....	20
2.2 Android App.....	24
2.3 User Story: Petras Geschichte & Ablaufdiagramme.....	27
2.4 User Story: Demo Anleitung zu Petras Geschichte .....	27
2.5 User Story: Anleitung für die Weiterentwicklung von Decisio Front- und Backend .....	27
2.6 Spike: System mit fehlenden Daten testen und Probleme beheben .....	28
3. Domain-Analyse .....	29
3.1 Domain Modell: Android App.....	29
3.2 Domain Modell: Back-End-Applikation .....	30
4. Design und Umsetzung.....	33
4.1 Back-End .....	33
4.2 Administration.....	47
4.3 Android App.....	55
4.4 Spike: System mit fehlenden Daten testen und Probleme beheben .....	79
4.5 Qualitätssicherung.....	80
4.6 Systemtest .....	83
4.7 Unit Tests.....	117
4.8 Test der Antwortzeiten.....	119
5. Schlussfolgerungen.....	120
5.1 Ergebnisse.....	120
5.2 Schlussfolgerungen.....	120
5.3 Ausblick.....	121
6. Glossar .....	122
7. Literaturverzeichnis.....	124
7.1 Weitere Literatur .....	125

# I. Technischer Bericht

---

## 1. Einleitung

---

In unserer Studienarbeit entwickelten wir die Android App "Decisio". Decisio gibt anhand des Motivprofils des Benutzers auf den Benutzer zugeschnittene Handlungsempfehlungen ab. Das Motivprofil wird mit einem Fragebogen ermittelt, welcher vorgängig vom Benutzer auszufüllen ist. Die Android App verwendet dynamische Daten, die von einem Server geladen werden, um für deren Änderungen keine neue App Version herausgeben zu müssen.

Zur weiteren Einführung in die Thematik unserer Studienarbeit, sowie dieser Bachelorarbeit, soll das Kapitel 1.1 dienen. Dabei werden aus Sicht eines Beispielbenutzers (Petra), sowie im zweiten Teil aus Sicht der Android App, mehrere Abläufe, von der ersten Benutzung der App bis zur Durchführung von Entscheidungssituationen, erklärt.

### 1.1 Einführung anhand eines konkreten Benutzers

#### 1.1.1 Persona

- Petra
- 40 Jahre
- Lebt in Zürich
- Ist verheiratet
- Leiterin einer Versicherungsagentur in der Innenstadt
- Mutter eines Kindes im Schulalter

Petra verbringt ihre Freizeit hauptsächlich mit der Familie. Es ist ihr wichtig, dass die Familie zusammenhält, sich regelmässig beim Essen trifft und auch vertrauensvolle Gespräche führen kann.

Im Beruf mag sie den Kontakt mit den Kunden, aber auch jenen mit ihren MitarbeiterInnen. Sie hat es geschafft, eine sehr gute Stimmung am Arbeitsplatz aufzubauen, sodass sich die Firma für sie auch wie eine kleine Familie anfühlt. Was sie an ihrem Job weniger mag, ist die zunehmende Konkurrenz und Jagd auf kurzfristige Erfolge. Zusätzlich schätzt sie die ewigen Machtkämpfe auf ihrer Führungsstufe nicht besonders. Sie geniesst es im Mittelpunkt zu stehen und wenn man ihr für ihre Erfolge gratuliert. Solches Feedback bedeutet ihr viel.

Montagabends spielt sie jeweils Hockey mit ihren alten Freunden in einem Verein - dies auf Amateurstufe. Dabei geht es ihr vor allem um den Ausgleich, die körperliche Bewegung und den Kontakt zu ihren Teamkameradinnen. Abgesehen von diesem Hobby ist Petra eher eine Stubenhockerin, die es gerne gemütlich hat und kaum Aufregung sucht.

#### 1.1.2 Szenario

Petra steht heute vor der Entscheidung, ob sie das Präsidentenamt im Hockeyverein übernehmen soll.

Ihr gehen verschiedene Fragen durch den Kopf: Werde ich noch genügend Zeit für meine Familie haben? Will ich in meiner Freizeit mehr Verantwortung übernehmen? Was meint wohl mein Mann dazu? Worauf sollte ich bei dieser Entscheidung sonst noch achten?



Sie hat gehört, dass eine App existiert, welche sie in solchen Situationen unterstützen kann und der Ratschlag auf sie und ihre individuellen Bedürfnisse abgestimmt ist. Sie entscheidet sich, bei Decisio Rat zu holen.

### 1.1.3 Ablauf: Verwendung der App aus Benutzersicht

#### 1.1.3.1 Installation

Petra installiert sich die App "Decisio" auf ihrem Smartphone.

#### 1.1.3.2 Erster Start

Beim ersten Start wird Petra aufgefordert den Disclaimer zu akzeptieren. Sie akzeptiert ihn, ansonsten würde die App sich schliessen.

Danach erscheint kurz ein Synchronisierungs-Dialog, aber kurz darauf eine Fehlermeldung. Für die initiale Synchronisierung werde eine Internet-Verbindung benötigt. Petra hat jeweils den Datenverkehr deaktiviert, ausser sie benötigt ihn explizit.

Sie bestätigt die Meldung worauf sich die App schliesst. Nach dem Aktivieren der Datenverbindung öffnet sie die App wieder. Nach etwa einer halben Minute sind alle Daten synchronisiert und Petra wird ein Formular mit vier soziodemographischen Fragen präsentiert (Geschlecht, Geburtsjahr, Einkommensklasse und Abschluss).

Die ersten beiden Fragen beantwortet sie mit "female" als ihr Geschlecht und "1975" als ihr Geburtsjahr. Bei den beiden letzten Fragen will sie keine Angabe tätigen. Sie versucht zu speichern, wird jedoch aufgefordert alle Fragen zu beantworten. Daraufhin öffnet sie die Auswahlliste der Optionen und erkennt die Option "no answer" (dt. keine Angabe), welche sie für die zwei letzten Fragen wählt. Durch ein erneutes Speichern des Formulars wird sie auf den Motivprofil-Fragebogen weitergeleitet.

Bei der Liste dieser Fragen, fragt sie sich, wozu sie diese beantworten soll und erkennt hierbei den Hinweis, wie der Bereich zu interpretieren ist. Ebenfalls zögert sie, da sie von den vielen Fragen nahezu erschlagen ist. Trotzdem beginnt sie die Fragen zu beantworten (Abbildung 1 bis Abbildung 8 folgen). Auch hier speichert sie die Antworten am Ende.

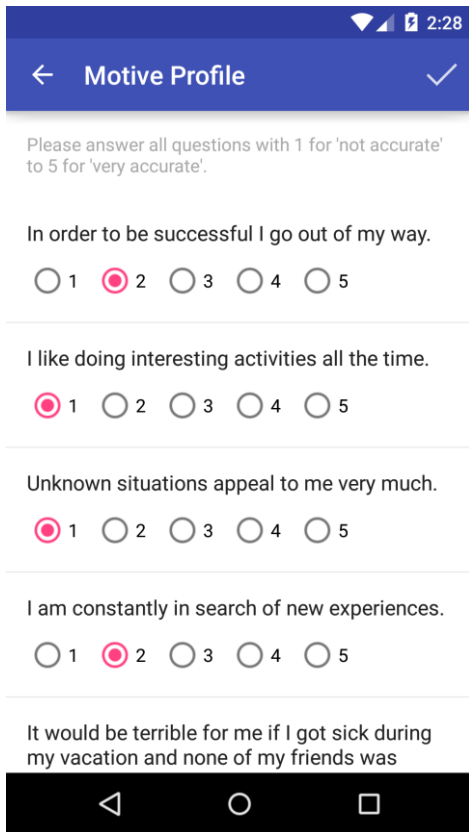


Abbildung 1: Fragebogen Seite 1

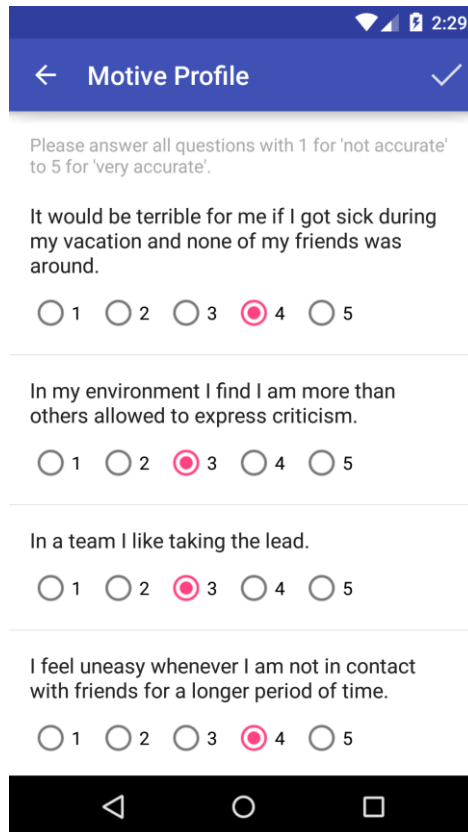


Abbildung 2: Fragebogen Seite 2

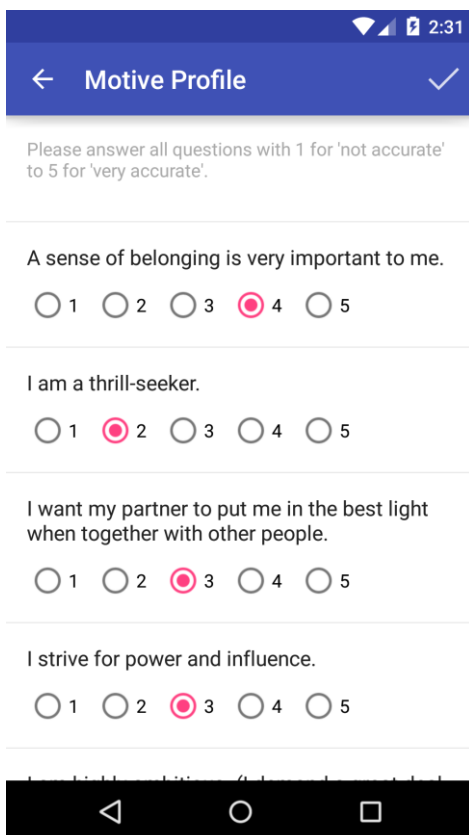


Abbildung 3: Fragebogen Seite 3

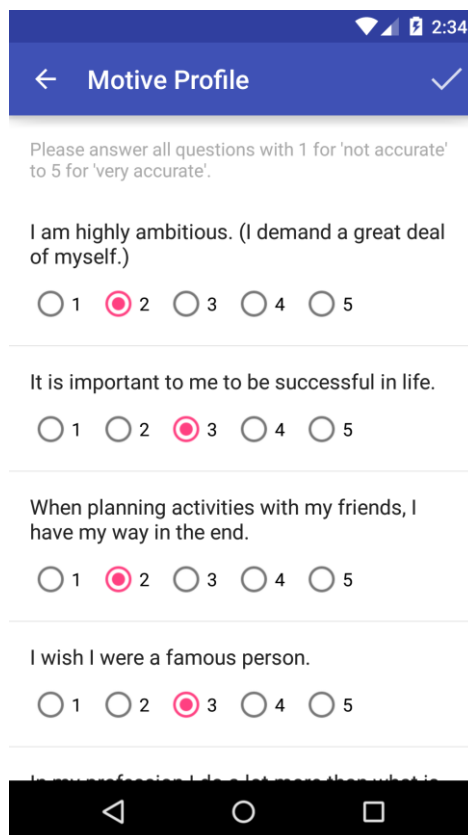


Abbildung 4: Fragebogen Seite 4

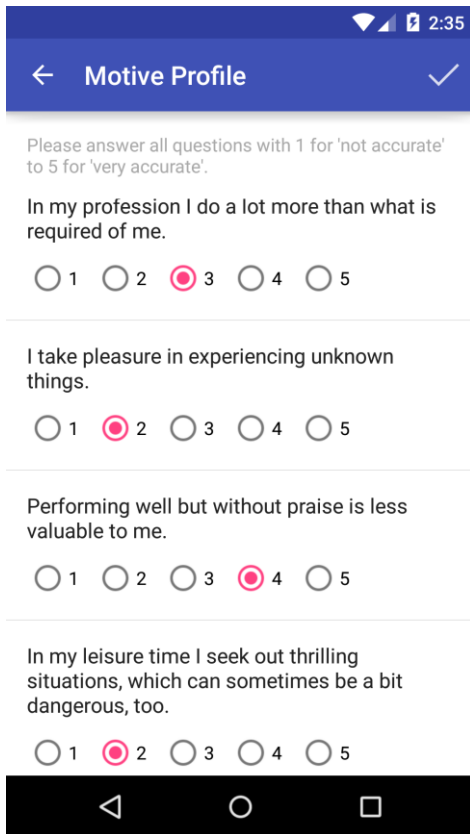


Abbildung 5: Fragebogen Seite 5

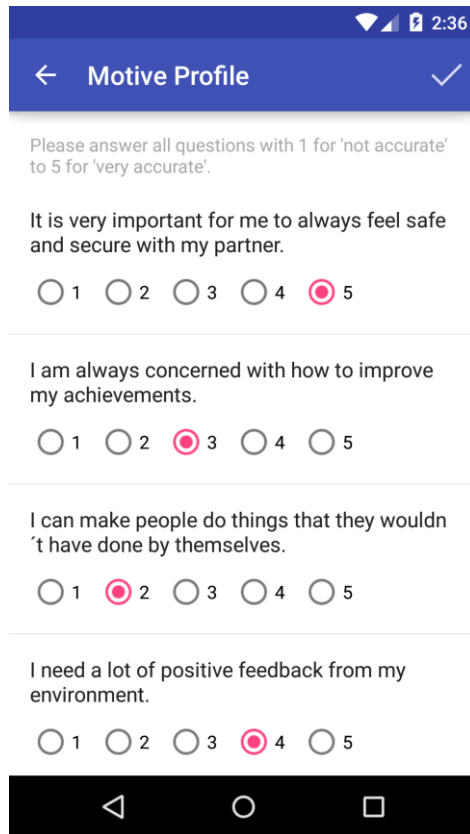


Abbildung 6: Fragebogen Seite 6

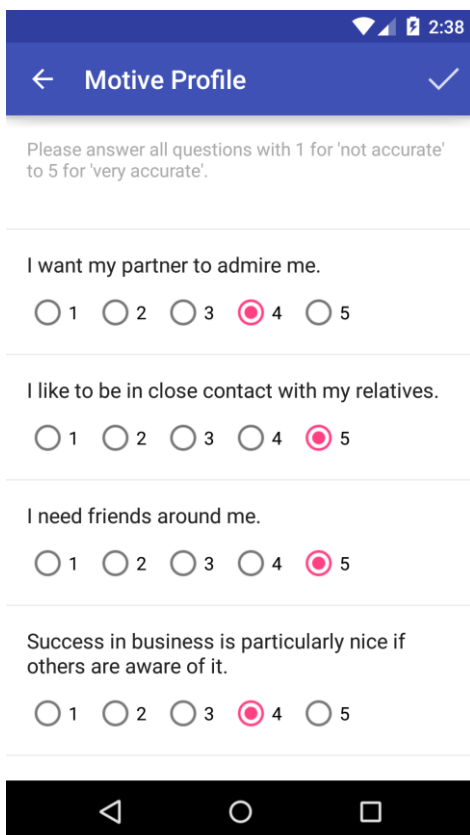


Abbildung 7: Fragebogen Seite 7

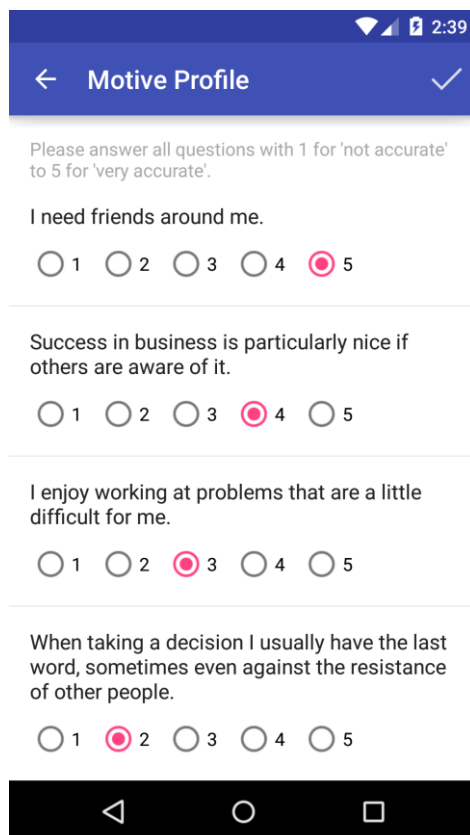
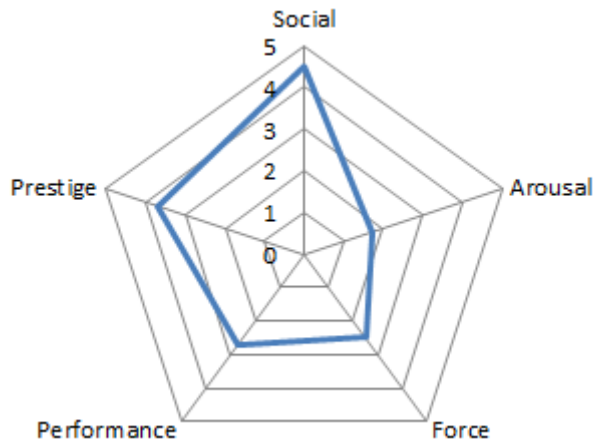


Abbildung 8: Fragebogen Seite 8

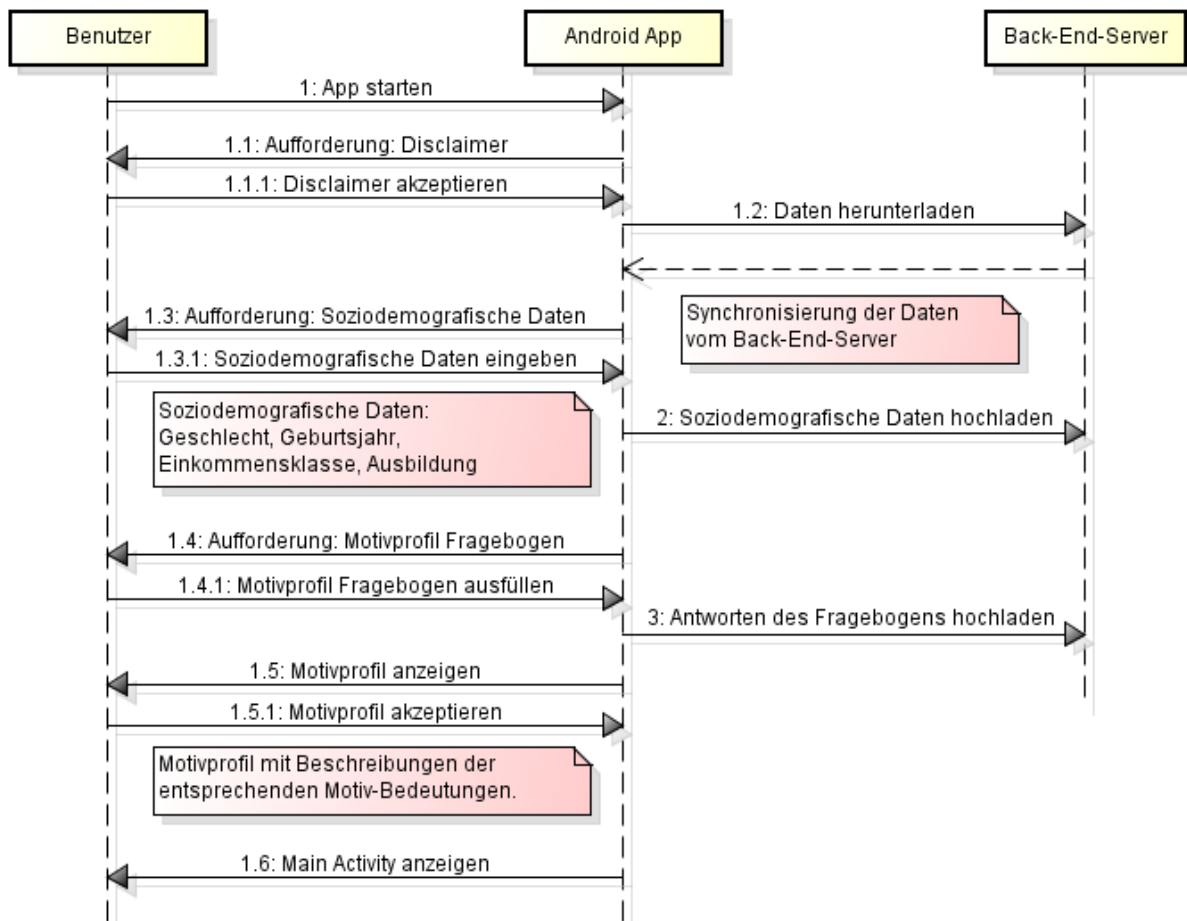
Zum Schluss wird Petra ihr persönliches Motivprofil präsentiert (die Einstufungen in den fünf Motiven). Sie findet heraus, dass ihr soziale Sicherheit und Geborgenheit sehr wichtig sind. In diesem Motiv hat sie einen höheren Wert als der Durchschnitt der Frauen in ihrem Alter. Leistung und Macht würden ihr dagegen nicht so viel bedeuten und die Unternehmungslust sei bei ihr ebenfalls nicht stark ausgeprägt. Hingegen findet die App, dass sie gerne gelobt wird und es genießt, wenn man ihr applaudiert. Zur Verdeutlichung ihrer Ausprägungen ist in Abbildung 9 ein Netzdiagramm ihres Motivprofils dargestellt.



**Abbildung 9: Darstellung von Petras Motivprofil als Netzdiagramm**

Petra ist positiv überrascht, dass die App sie nach 30 Fragen bereits so gut zu kennen scheint. Dadurch schöpft sie Vertrauen und ist gespannt, welchen Rat ihr die App in ihrer Entscheidungssituation wohl geben wird.

Da dies aber mehr Zeit in Anspruch nahm, als sie ursprünglich dachte, schliesst sie die App vorerst.



**Abbildung 10: Ablauf erstes Mal App starten**

### 1.1.3.3 Entscheidungssituation durchführen

Zu einem späteren Zeitpunkt nimmt sich Petra die Zeit, eine Entscheidungssituation durchzuführen. Dazu öffnet sie die App. Wieder hat sie den Datenverkehr deaktiviert. Deshalb erscheint der Synchronisierungs-Dialog kaum wahrnehmbar und es wird gleich die Startseite angezeigt.

Sie drückt auf "MAKE DECISION", wodurch ihr die vorhandenen Entscheidungssituationen angezeigt werden. Nach einem kurzen Überfliegen findet und wählt sie die Entscheidungssituation "Should I assume an executive office in my club?" (dt. Soll ich in meinem Verein ein Vorstandsamt übernehmen?).

Ihr wird der erste Faktor, "Time for partner" (dt. Zeit für Lebenspartner), mit den vier Optionen "NO PROBLEM", "NOT IDEAL", "PROBLEM" und "NOT RELEVANT" angezeigt. Petra bewertet diesen Faktor als "NOT IDEAL" (orange), zumal ihr Mann sich kürzlich beklagt hat, ihre Liebesbeziehung käme manchmal zu kurz.

Petra wird der zweite Faktor, "Interaction with other people" (dt. Umgang mit anderen Menschen), angezeigt. Diesen Faktor bewertet sie als "NO PROBLEM" (grün), da sie keine Probleme hat mit den Vereinsmitgliedern Kontakte zu knüpfen. Ihr werden noch vier weitere Faktoren angezeigt, welche für die Entscheidungssituation relevant sind. Diese bewertet sie alle als "NO PROBLEM" (grün).

Am Ende wird ihr die Handlungsempfehlung "You can proceed with slight risk" mit ihren Faktorbewertungen angezeigt. Sie erkennt den "Erweitert" Tab, wo ihr der als orange bewertete Faktor angezeigt und eine Problemläuterung dazu beschrieben wird.

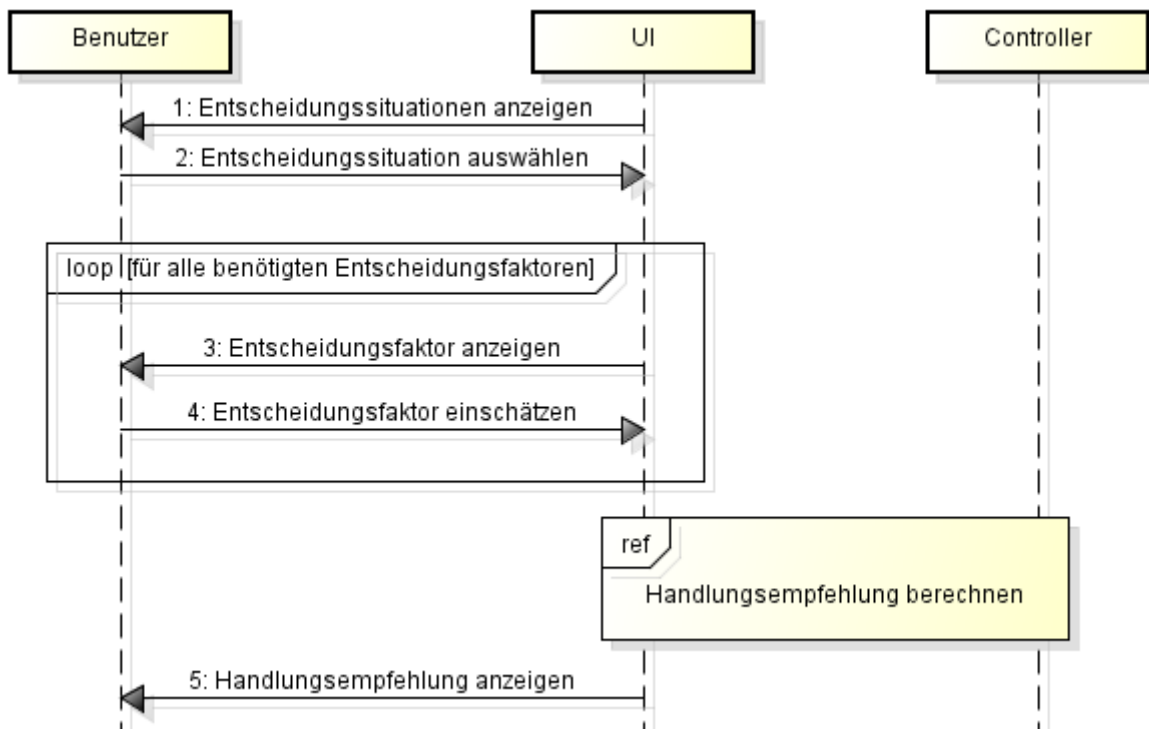


Abbildung 11: Ablauf Entscheidung fällen

## 1.1.4 Ablauf: Hintergrund-Aktivitäten bei Verwendung der App

### 1.1.4.1 Erster Start

Beim Start der App wird im Hintergrund der `StartAppController` gestartet. Dieser stellt eine Zustandsmaschine dar. Da die App das erste Mal gestartet wurde, weist er die Activity an, dem Benutzer den Disclaimer Dialog anzuzeigen.

Hat der Benutzer den Disclaimer akzeptiert, wird der Synchronisierungs-Dialog angezeigt und anschliessend die Synchronisierungsdienste gestartet. Beide Synchronisierungsdienste prüfen zu Beginn, ob eine Internetverbindung verfügbar ist, ansonsten beenden sie sich wieder. Der `UploadSyncInitiator` lädt die Antworten des Benutzers zum Server hoch (soziodemografische Daten und Antworten des Motivprofil-Fragebogens), hat aber in diesem Fall nichts zu tun. Der `DownloadSyncInitiator` lädt alle benötigten XML-Daten herunter und verarbeitet diese lokal. Im Gegensatz zum `UploadSyncInitiator` gibt der `DownloadSyncInitiator` ein Resultat zurück, ob die Synchronisierung funktionierte oder nicht. Da Petra die Datenverbindung deaktiviert hat, ist das Resultat "kein Internet". Der `StartAppController` prüft dieses Resultat und fordert die Activity auf, eine Fehlermeldung anzuzeigen.

Petra öffnet die App wieder, mit aktivierter Datenverbindung. Da sie den Disclaimer bereits akzeptierte, werden wieder die Synchronisierungsdienste gestartet. Da dies der erste Start ist, lädt der `DownloadSyncInitiator` alle XML-Daten vom Server herunter, parst sie und legt die Daten in der lokalen Datenbank ab. Der `StartAppController` erhält das Resultat "Erfolg". Die App ist nun im Zustand "initialisiert". Deshalb weist der Controller die Activity an, dass der Benutzer die soziodemografischen Daten eingeben soll.

Will der Benutzer die Daten speichern, prüft die App, ob alle Fragen beantwortet wurden. Wenn nicht, kann die Activity nicht verlassen werden. Sind alle Fragen beantwortet, wird der `UploadSyncInitiator` gestartet. D.h. sobald Petra speichert, werden ihre Antworten auf den Server hochgeladen. Der `StartAppController` fordert nun die Activity auf, dem Benutzer den Motivprofil-Fragebogen anzuzeigen.

Beim Speichern gilt dasselbe wie bei den soziodemografischen Daten. Auch hier werden Petras antworten vom `UploadSyncInitiator` hochgeladen. Zusätzlich wird Petras persönliches Motivprofil berechnet, lokal gespeichert und ihr präsentiert. Die App ist nun im Zustand "bereit".

### 1.1.4.2 Weiterer Start

Bei jedem weiteren Start der App werden wiederum beide Synchronisierungsdienste gestartet.

Wenn eine Internetverbindung verfügbar ist, lädt der `DownloadSyncInitiator` die XML-Daten vom Back-End-Server herunter. Dazu wird der Zeitpunkt der letzten Synchronisierung benötigt.

Der `UploadSyncInitiator` prüft ebenfalls, ob eine Internetverbindung verfügbar ist. Wenn ja, prüft er zusätzlich, ob noch Daten zum Hochladen ausstehend sind, d.h. ob die Daten beim letzten Mal nicht hochgeladen werden konnten, und lädt diese wenn nötig auf den Back-End-Server hoch. Dies kann jedoch im Hintergrund geschehen, sodass die Main Activity trotzdem bereits angezeigt werden kann.

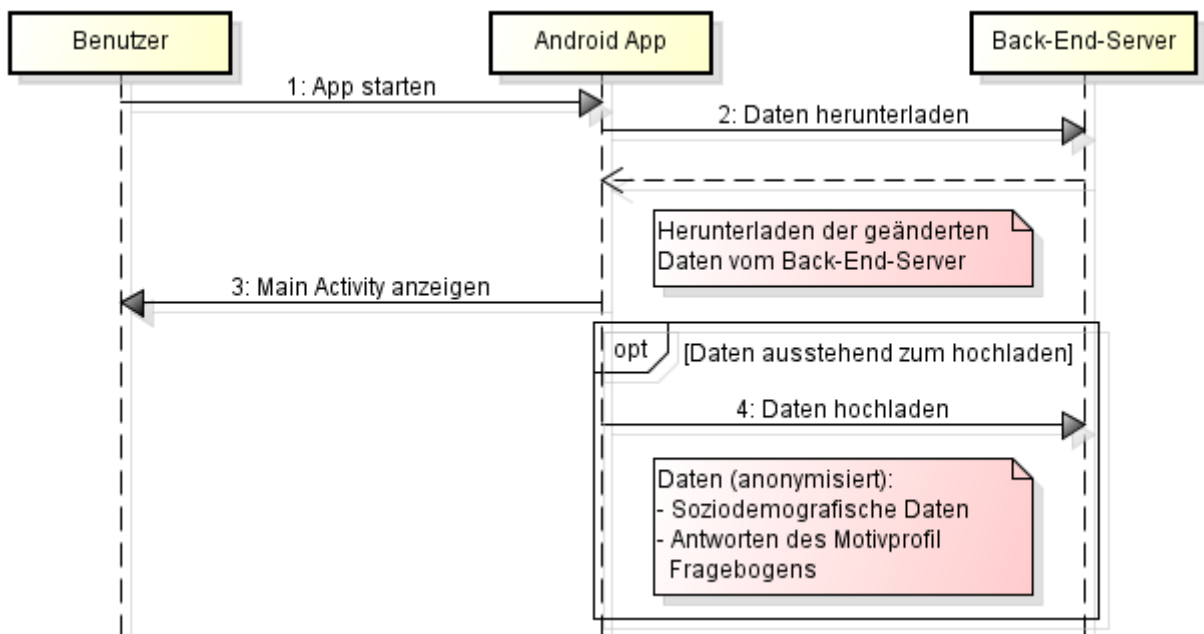


Abbildung 12: Ablauf weiteres App starten

### 1.1.4.3 Entscheidungssituation durchführen

Petra startet die App erneut. Es wird wieder ein `StartAppController` erstellt. Die App ist nun im Zustand "bereit". Er startet die Synchronisierungsdienste. Am Ende, egal welches Resultat der Dienst lieferte, wird der Synchronisierungsdialog ausgeblendet. Da Petra den Datenverkehr deaktiviert hat, ist der Dienst gleich fertig und der Dialog wird sofort wieder ausgeblendet.

Sie will eine Entscheidungssituation durchführen. Die App zeigt ihr die Liste der vorhandenen Entscheidungssituationen an. Beim Auswählen wird die Activity zur Faktor-Bewertung angezeigt.

Die Activity arbeitet mit einem `DecisionController`. Dieser speichert die Faktor-Bewertungen des Benutzers. Petra wird ein Faktor nach dem anderen zur Bewertung vorgelegt.

Nach der Bewertung des letzten Faktors wird zur Activity mit der Anzeige der Handlungsempfehlung gewechselt. Diese erhält über den Aufruf-Intent die Faktor-Bewertungen und berechnet damit die Handlungsempfehlung.

#### 1.1.4.3.1 Berechnung der Handlungsempfehlung

Die Summe `ratingValue` wird auf 0 gesetzt.

Es werden alle Faktoren durchlaufen:

- Ist die Bewertung des Faktors "kein Problem" (grün), "problematisch" (rot) oder "nicht relevant" (grau) wird das entsprechende Gewicht der Einstufung zu `ratingValue` addiert. Bei grün und grau ist dieses 0, bei rot 3.
- Bei orange wird eine Korrektursumme berechnet. Nach der Berechnung werden das Gewicht der Einstufung orange und die Korrektursumme zu `ratingValue` addiert. Die Korrektursumme wird wie folgt berechnet:  
Zu Beginn wird diese auf 0 gesetzt.  
Für jedes Motiv:
  - Wird das Mapping des Faktors zu diesem Motiv ermittelt.
  - Wird der Wert im Motivprofil des Benutzers zu diesem Motiv ausgelesen.
  - Wird der Korrekturbetrag in der Rating-Tabelle mit dem Mapping und dem Profil-Wert nachgeschlagen.
  - Dieser Korrekturbetrag wird zur Korrektursumme addiert.

Dieser Ablauf ist in Abbildung 14 als Sequenzdiagramm dargestellt und in Abschnitt 1.1.4.4 mit einem Beispiel erklärt.

Sind alle Faktoren durchlaufen, wird `ratingValue` mit den definierten Schwellenwerten verglichen. Liegt sie unter dem `warningThreshold`, ist die Handlungsempfehlung positiv. Liegt sie über dem `warningThreshold` und unter dem `doNotProceedThreshold` ist sie "ja, mit Vorsicht". Ansonsten ist die Handlungsempfehlung negativ.

Dieser gesamte Ablauf der Berechnung der Handlungsempfehlung ist in Abbildung 13 als Sequenzdiagramm dargestellt.



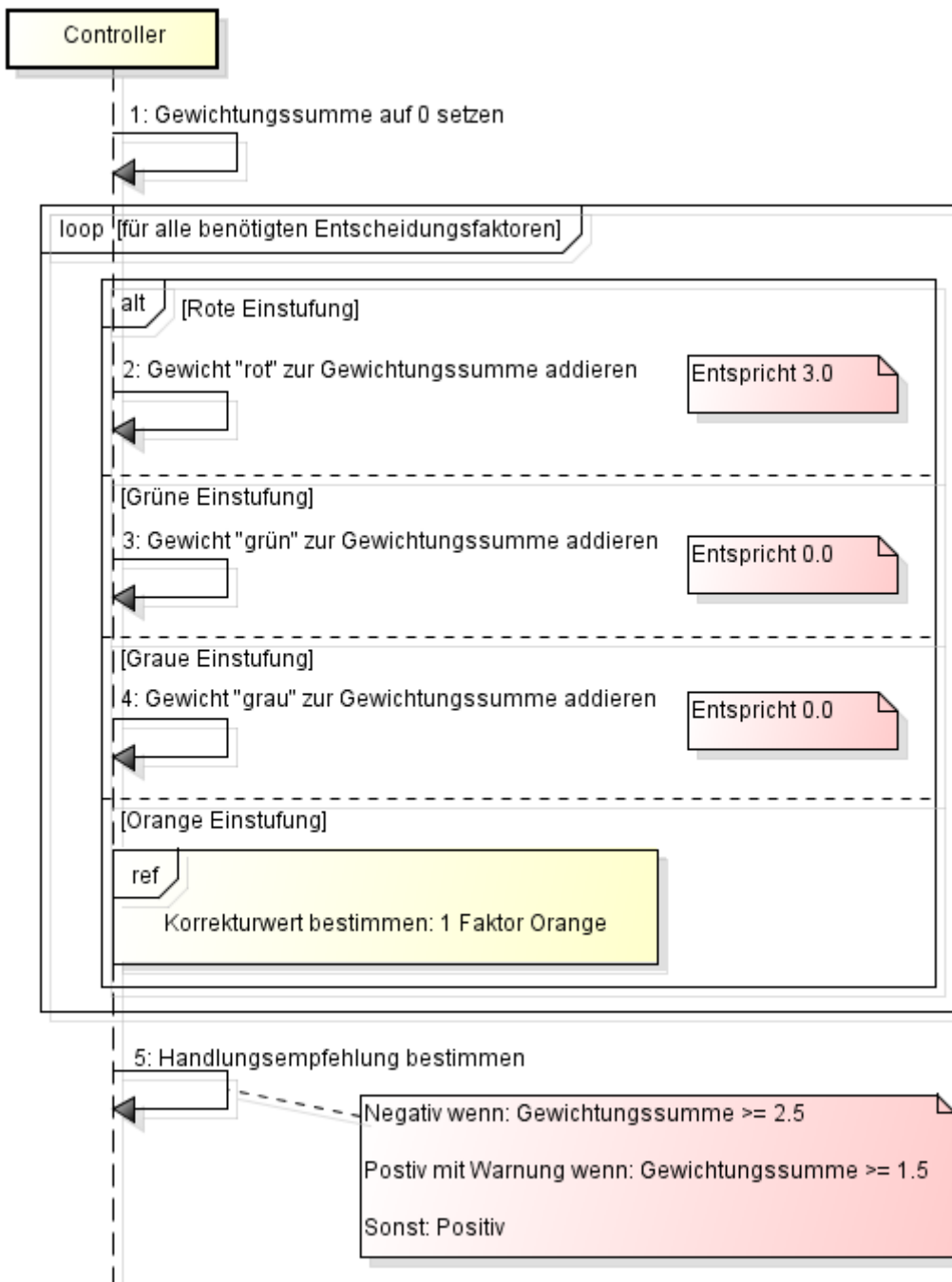
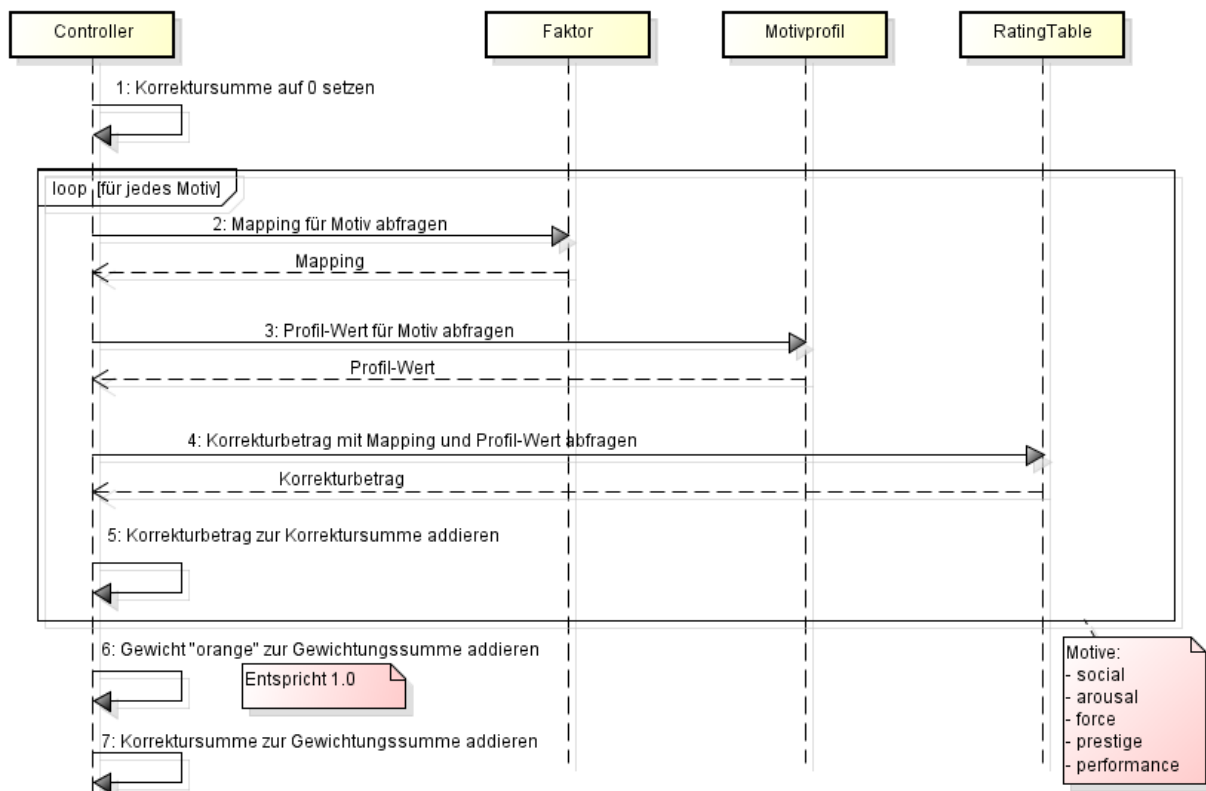


Abbildung 13: Ablauf Handlungsempfehlung berechnen



**Abbildung 14: Ablauf Bestimmung des Korrekturwerts für Orange bewerteten Entscheidungsfaktor**

#### 1.1.4.4 Ein konkretes Beispiel zur Bestimmung des Korrekturwerts

Petras Motivprofil:

Social:	4.50
Arousal:	1.67
Force:	2.50
Prestige:	3.67
Performance:	2.67

Rating-Tabelle:

factorValue	profileValue	correctionAmount
0	1	0
0	2	0
0	3	0
0	4	0
0	5	0
1	1	-0.4
1	2	-0.3
1	3	0
1	4	0.1
1	5	0.3
2	1	-0.3
2	2	-0.2
2	3	0
2	4	0.1
2	5	0.2
...		

5	1	0.2
5	2	0.1
5	3	0
5	4	0.2
5	5	0.4

Entscheidungsfaktor "Time for partner"	
socialMapping:	5
arousalMapping:	0
forceMapipng:	0
prestigeMapping:	2
performanceMapping:	0

Korrektursumme: 0

Korrekturbetrag Motiv social:

Faktor Mapping 5, Profil-Wert 4.50 -> 5, Korrekturbetrag 5 und 5 => 0.4

Korrekturbetrag Motiv arousal:

Faktor Mapping 0, Profil-Wert 1.67 -> 2, Korrekturbetrag 0 und 2 => 0

Korrekturbetrag Motiv force:

Faktor Mapping 0, Profil-Wert 2.50 -> 3, Korrekturbetrag 0 und 3 => 0

Korrekturbetrag Motiv prestige:

Faktor Mapping 2, Profil-Wert 3.67 -> 4, Korrekturbetrag 2 und 4 => 0.1

Korrekturbetrag Motiv performance:

Faktor Mapping 0, Profil-Wert 2.67 -> 3, Korrekturbetrag 0 und 3 => 0

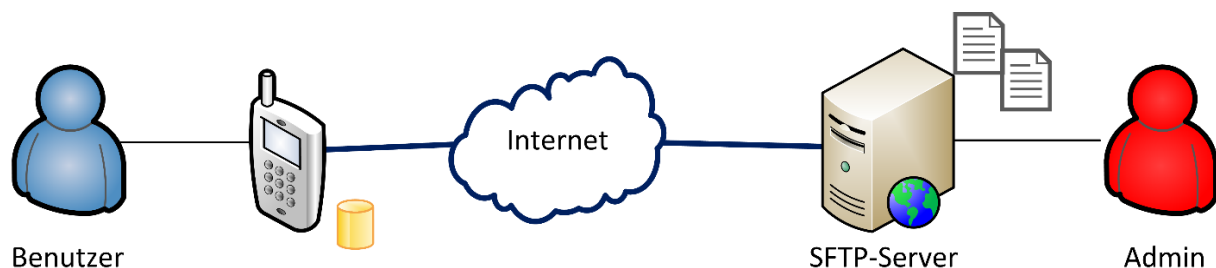
Neue Korrektursumme: 0.5

Diese Korrektursumme (0.5) addiert mit dem Standard Gewicht für Orange-Bewertungen (1.0) stellt das Gewicht des Entscheidungsfaktors "Time for partner" dar und wird nun zur Gewichtungssumme aller relevanten Entscheidungsfaktoren addiert.

## 1.2 Übersicht

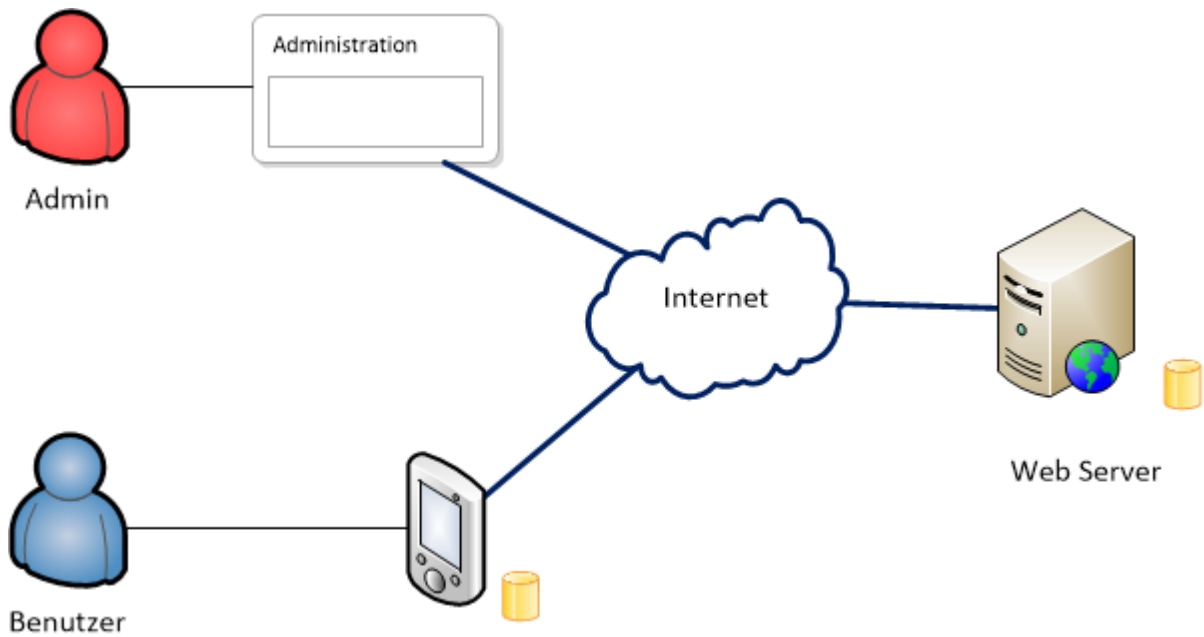
### 1.2.1 Systemübersicht

In unserer Studienarbeit haben wir zum Laden der dynamischen Daten XML-Dateien auf einem SFTP-Server verwendet. Diese Übersicht ist in Abbildung 15 dargestellt.



**Abbildung 15: Systemübersicht der Studienarbeit**

In dieser Bachelorarbeit ist die Entwicklung einer serverseitigen Back-End-Applikation geplant. Die Praxispartner sollen dadurch eine bequemere Art erhalten, die dynamischen Daten zu erfassen oder zu ändern. Dies soll über eine Administrationsseite (die Decisio Administration) im Browser möglich werden. Diese Soll-Übersicht ist in Abbildung 16 dargestellt.



**Abbildung 16: Systemübersicht dieser Bachelorarbeit**

## 1.2.2 Kapitelübersicht

Anforderungs-Analyse	In diesem Kapitel sind die Anforderungen dieser Bachelorarbeit spezifiziert.
Domain-Analyse	In diesem Kapitel wurde die Problemstellung analysiert.
Design und Umsetzung	Dieses Kapitel dokumentiert die Architektur der Back-End-Applikation, getroffene Architekturentscheide, sowie die Änderungen an der Android App.
Schlussfolgerung	In diesem Kapitel sind die Ergebnisse unserer Arbeit, sowie daraus gezogene Schlussfolgerungen beschrieben.

## 2. Anforderungs-Analyse

---

### 2.1 Administration

Zur Bearbeitung der vom Android Client synchronisierten Daten soll eine Administrations-Applikation entwickelt werden. Sie stellt sozusagen das Front-End zum Back-End dar.

Folgende Technologien der Administration wurden von den Praxispartnern gewünscht: Es soll eine Web-Applikation mit dem AngularJS-Framework [1] entwickelt werden. Zusätzlich soll als Style-Grundlage Angular Material [2] zum Einsatz kommen. Die Applikation soll auf Heroku [3] gehostet werden.

In den folgenden User Stories wird der Rahmen der Administration beschrieben.

#### 2.1.1 User Story: Setup RDBMS

Als Entwickler möchte ich eine relationale Datenbank auf Heroku haben.

Akzeptanzkriterien:

- Das Schema der DB bildet die Struktur der XML-Dateien ab.
- Die DB ist mit einem initialen Dataset geseeded.
- Die DB kann z.B. mit einem Skript zurückgesetzt werden.
- Die DB kann z.B. mit einem Skript exportiert werden.

#### 2.1.2 User Story: Zugriff auf Back-End

Als Administrator möchte ich auf das Back-End (die Administration) zugreifen können.

Akzeptanzkriterien:

- Das Back-End erscheint vorerst als Splash Screen im Browser.
- Dieses Back-End läuft auf Heroku.
- Es ist noch keine Security eingebaut.
- Der Splash Screen wird über AngularJS mit Angular Material gerendert.
- Der Splash Screen ist mit Cards [4] aufgebaut.

#### 2.1.3 User Story: Back-End Menu

Als Administrator kann ich über ein Menu im Back-End (in der Administration) navigieren.

Akzeptanzkriterien:


- Als Administrator sehe ich folgende Menüpunkte:
  - Home
  - MotiveProfileSpecs
  - Decisions
  - Factors
  - Questionnaire
  - RatingTable
- Das Menu ist mit "Tabs" [5] umgesetzt.
- Die einzelnen Screens bestehen bereits (mit leerem Inhalt/Titel als Inhalt).

## 2.1.4 User Story: Screen zu motiveProfileSpecifications\_en.xml

Als Administrator möchte ich den Inhalt der Datei `motiveProfileSpecifications_en.xml` über die Administration einsehen und editieren können.

Akzeptanzkriterien:

- Der Inhalt der Datei wird als Liste [6] angezeigt.
- Als Administrator kann ich `aboveExplanation`, `averageExplanation`, `belowExplanation`, `averageThreshold` und `aboveAverageThreshold` editieren.
- Das Feld "motive" wird angezeigt, ist aber nicht editierbar.
- Das Feld "motive" wird zusätzlich auch als Icon dargestellt, siehe Abbildung 17.

 social

 performance

 arousal

 force

 prestige

Abbildung 17: Icons der verschiedenen Motive

## 2.1.5 Änderung: Screen Motive Profiles

Der Screen aus Abschnitt 2.1.4 wird wie folgt angepasst:

- Titel wird umbenannt, von "MotiveProfileSpecs" nach "Motive Profiles".
- Die Bezeichnung des Felds "motive" wird auch als Text angezeigt.
- Das Popup "Motive Icons" mit der Erklärung der Motiv-Icons entfällt.
- Die Anzeige der Schwellwerte des Durchschnittsbereichs "Average range = from 2.33 to 3.67" wird ersetzt durch: "Average range goes from X to Y".
- Die Erläuterungstexte der Einstufungen werden in der Übersicht mit dem entsprechenden Label versehen, sodass nicht nur die Texte allein stehen. Siehe Abbildung 18.
- Die Bezeichnung "MotiveProfileSpecs" im Menü wird zu "Motive Profiles" geändert.

Above Explanation:	Your score is above average.
Average Explanation:	Your score is average.
Below Explanation:	Your score is below average.

Abbildung 18: Motive Profiles: Labels der Erläuterungstexte in der Übersicht

## 2.1.6 Änderung: Sprachfunktionalität in Motive Profiles umsetzen

Als Administrator möchte ich auch für die Motivprofil Spezifikationen mehrere Sprachen verwalten können.

Akzeptanzkriterien:

- Als Administrator kann ich eine neue Sprachdefinition hinzufügen.
- Als Administrator kann ich eine Sprachdefinition löschen.

### 2.1.7 User Story: Screen zu questionnaire\_en.xml

Als Administrator möchte ich den Inhalt der Datei `questionnaire_en.xml` über die Administration editieren können.

Akzeptanzkriterien:

- Der Inhalt der Datei wird als Liste angezeigt.
- Die Tabelle ist sortierbar nach "number" und "motive".
- "motive" wird als Icon dargestellt, siehe 2.1.4.
- Als Administrator kann ich das Motiv einer Question ändern.
- Als Administrator kann ich den Fragetext editieren.
- Als Administrator kann ich neue Fragen hinzufügen.
- Als Administrator kann ich bestehende Fragen löschen.
- Als Administrator kann ich die Reihenfolge der Fragen ändern.

### 2.1.8 User Story: Screen zu factors\_en.xml

Als Administrator kann ich den Inhalt der Datei `factors_en.xml` über die Administration editieren.

Akzeptanzkriterien:

- Die Liste zeigt an: "Name", "Description", "socialMapping", "arousalMapping", "forceMapping", "prestigeMapping", "performanceMapping".
- Die Liste ist alphabetisch sortierbar nach "Name" und absteigend sortierbar nach "socialMapping", "arousalMapping", "forceMapping", "prestigeMapping" und "performanceMapping".
- Die Felder "Name", "Description", "socialMapping", "arousalMapping", "forceMapping", "prestigeMapping", "performanceMapping" sind editierbar. Die Mapping Werte sollen mit der Tastatur, ohne Verwendung der Maus, editiert werden können.
- Als Administrator kann ich mit einem Faktor verknüpfte Decisions einsehen (ohne Link).
- Als Administrator kann ich einen neuen Faktor hinzufügen.
- Als Administrator kann ich einen Faktor löschen.

### 2.1.9 User Story: Screen zu ratingTable.xml

Als Administrator kann ich den Inhalt der Datei `ratingTable.xml` über die Administration editieren.

Akzeptanzkriterien:

- Anzeige der Einträge
- Erstellen eines neuen Mappings (fünf Einträge mit Motivprofil-Wert 1 bis 5)
- Ändern eines Mappings (fünf Einträge mit Motivprofil-Wert 1 bis 5)
- Löschen eines Mappings (fünf Einträge mit Motivprofil-Wert 1 bis 5)
- Es sollen auch negative Werte unterstützt werden.
- Nur der letzte Eintrag soll gelöscht werden können. Die Nummerierung erfolgt automatisch, d.h. beim Erstellen eines neuen Eintrags wird einfach der nächste freie Faktorwert genommen, sodass keine "Löcher" in der Skala entstehen.

### 2.1.10 User Story: Screen zu decisions\_en.xml

Als Administrator kann ich den Inhalt der Datei `decisions_en.xml` über die Administration editieren.

Akzeptanzkriterien:

- Die Entscheidungssituationen und deren Faktoren werden als Liste dargestellt.
- Faktoren können mit "Chips" [7] zu einer "decision" hinzugefügt oder entfernt werden.
- Als Administrator kann ich eine "decision" löschen.
- Als Administrator kann ich eine "decision" hinzufügen.
- Die "question" einer "decision" kann editiert werden.
- Das Upgrade auf Angular Material 0.9 ist vollzogen. Die Layout-Elemente der Screens sind jeweils auf gleicher Höhe angeordnet und die Chips sind verfügbar.

### 2.1.11 Änderung: Alignment des Cancel und Save Buttons in allen Screens

Die Buttons "Cancel" und "Save" werden in jedem Screen nahe beieinander auf der rechten Seite des Screens platziert.

### 2.1.12 RESTful Web Service

Als Entwickler möchte ich den Zugriff der Administration, sowie die Synchronisierung der Android Clients über einen RESTful Web Service anbieten.

Akzeptanzkriterien:

- Sämtliche CRUD Operationen für die User Stories der Administration aus den Abschnitten 2.1.4 bis 2.1.10 werden unterstützt.
- Die Administration kommuniziert mit JSON über die REST Schnittstelle mit der Datenhaltung.
- Der Android Client kommuniziert mit XML über die REST Schnittstelle mit der Datenhaltung.

Diese Anforderung soll in mehreren Tickets (pro Screen in der Administration ein separates Ticket) in den jeweils passenden Sprints umgesetzt werden.

### 2.1.13 Spike: Problem mit Connections anschauen (Connection Pooling)

Als Entwickler möchte ich die Ursache des Problems mit den zu vielen offenen Connections suchen und lösen. Zudem möchte ich den Einsatz eines Connection Pools prüfen und wenn (innerhalb dieses Spikes) zeitlich möglich direkt einsetzen.

### 2.1.14 Spike: Konzept Fehler-Behandlung in der Administration

Als Entwickler möchte ich ein einheitliches Konzept zur Fehler-Behandlung im Back-End Bereich (in der Administration) erstellen. Zudem möchte ich eine Prototyp-Implementation für einen Screen umsetzen.

### 2.1.15 User Story: Fehlerbehandlung in bestehenden Screens und REST Klassen anwenden

Als Administrator möchte ich in allen Screens (Tabs in Decisio Administration) die gleiche, bereits evaluierte Fehlerbehandlung (siehe Abschnitt 2.1.14) haben.

Akzeptanzkriterien:

- Die für den Prototyp Screen (Motive Profiles) umgesetzte Fehlerbehandlung für alle bestehenden Screens anwenden.



### **2.1.16 User Story: Deleted Flag in ausstehenden Klassen umsetzen**

Als Entwickler möchte ich das "Deleted" Flag zur Markierung eines gelöschten Eintrags in der Datenbank (notwendig für die Synchronisierung) in allen Ressourcen anwenden und in den Abfragen korrekt beachten.

### **2.1.17 User Story: Fallback auf Englisch in ausstehenden Klassen umsetzen**

Als Entwickler möchte ich den Fallback auf Englisch in der Synchronisierung der Android Clients (wenn alle Daten für eine bestimmte Sprache als gelöscht markiert wurden) in allen Ressourcen umsetzen.

### **2.1.18 User Story: Update von Angular Material**

Angular Material soll auf die aktuelle Version gewechselt werden, um allenfalls die Probleme mit dem Autocompletion-Feld (zur Spracheingabe und -auswahl) zu beheben.

### **2.1.19 User Story: Zugriffsschutz Backend**

Als Entwickler möchte ich die Administration vor unbefugten Zugriffen schützen.

Akzeptanzkriterien:

- Die Seite ist nur über HTTPS verfügbar.
- Man muss sich mindestens mit einem Passwort in der Administration anmelden müssen (möglichst einfacher Zugriffsschutz).

## **2.2 Android App**

Die in unserer Studienarbeit entwickelte Android App soll erweitert und verbessert werden. In den folgenden User Stories wird beschrieben, in welchem Rahmen dies geschehen soll.

### **2.2.1 User Story: Usability-Fehler beheben**

Als Entwickler will ich den im Systemtest der Studienarbeit gefundenen Usability Fehler beheben: Die Dialoge zum Akzeptieren des Disclaimers und zur Information, dass eine initiale Internetverbindung benötigt wird, können umgangen werden, indem auf ein Bereich ausserhalb des Dialogs geklickt wird, oder der Back-Button gedrückt wird. Dadurch wird die App nicht geschlossen und könnte trotzdem verwendet werden.

### **2.2.2 User Story: APK auf Windows laufen lassen**

Als Kunde will ich die APK-Datei auf meinem Windows-Rechner laufen lassen können. Dazu will ich eine entsprechende Anleitung erhalten.

### **2.2.3 User Story: Deployment-Script erstellen**

Als Entwickler will ich das APK automatisiert auf Google Drive kopieren können. Die Datei soll zur Identifikation mit dem Datum der Kompilierung versehen sein.

### **2.2.4 User Story: Umstellung des Fragebogens auf den REST-Service**

Als Referenz-Implementation soll die Synchronisierung des Motivprofil-Fragebogens vom SFTP-Server auf den REST-Service umgestellt werden.

Dies hat auch zur Folge, dass die Synchronisierung angepasst werden muss (nicht mehr alle Einträge löschen und die erhaltenen Einträge einfügen, sondern partiell synchronisieren).

Akzeptanzkriterien:

- Die Synchronisierung des Motivprofil-Fragebogens erfolgt über den REST-Service.
- Die partielle Synchronisierung ist mit Unit Tests abgedeckt (Android App).

### 2.2.5 User Story: Umstellung der restlichen Ressourcen auf den REST Service

Als Entwickler will ich die Referenz-Implementation der Umstellung auf den REST-Service (siehe Abschnitt 2.2.4) auch für die restlichen Ressourcen übernehmen.

Akzeptanzkriterien:

- Die Synchronisierung aller Ressourcen erfolgt über den REST-Service
- Die partielle Synchronisierung aller Ressourcen ist mit Unit Tests abgedeckt.

### 2.2.6 User Story: Upload der Benutzer-Daten an den REST-Service

Als Administrator will ich die hochgeladenen Daten der Benutzer ebenfalls auf dem neuen Back-End-Server einsehen können.

Akzeptanzkriterien:

- Der REST-Service ist erweitert, sodass er die Daten der Benutzer entgegen nehmen kann (Soziodemografische Daten und Motivprofil-Fragebogen-Antworten).
- Die App ist umgestellt und lädt die Daten auf den REST-Service und nicht mehr auf den SFTP-Server hoch.

### 2.2.7 User Story: Anpassen der Mehrsprachigkeit

Wird eine gelöschte Sprache (wenn alle Einträge einer verwendeten Sprache als gelöscht markiert sind) wieder erstellt, erhält das Gerät nur die Aktualisierungen dieser wieder erstellten Sprache. Hat das Gerät aber vorher bereits einmal synchronisiert und somit durch den Fallback die Englischen Einträge erhalten, so hätte es nun zwei verschiedene Sprachen in der lokalen Datenbank. Das Gerät muss deshalb alle Einträge, die nicht die Sprache der durch die Synchronisierung neu erhaltenen Einträge aufweisen, aus seiner Datenbank löschen. D.h. wenn das Gerät durch die Synchronisierung Einträge in Deutsch erhält, so muss es alle nicht deutschen Einträge aus der lokalen Datenbank löschen. Somit hat das Gerät jeweils nur eine Sprache in seiner lokalen Datenbank.

Dies ist für alle fünf Ressourcen umzusetzen.

### 2.2.8 User Story: Synchronisierungssprache auf Englisch fixieren

Als Benutzer der Android App möchte ich Decisio ausschliesslich in englischer Sprache bedienen. Der Grund hierfür liegt darin, dass die statischen Texte der Android App sonst in Englisch, der Inhalt (die synchronisierten Daten) aber in einer anderen Sprache wären.

Akzeptanzkriterien

- Die Android App hält und synchronisiert ausschliesslich englische Text-Ressourcen.
- Auch auf einem Device mit einer anderen Spracheinstellung erscheinen sämtliche Texte nur auf Englisch.
- Der mit dieser Story betroffene Code ist in unserer Source Code Verwaltung mit einem Kommentar markiert.
- Der Code der User Story aus Abschnitt 2.2.7 bleibt erhalten.

### **2.2.9 User Story: Bezug zu Panter AG löschen**

Den Bezug zur Panter AG in allen Dokumenten und dem Source Code löschen.

Das Copyright gehört der Forventis.

### **2.2.10 User Story: Upgrade auf API Level 21 (Android 5.0)**

Als Praxispartner will ich eine App mit API Level 21, damit es möglich ist, das Design im Material Design zu gestalten.

### **2.2.11 User Story: Upgrade auf Material Theme der Android App**

Als Benutzer der Android App finde ich Decisio im Material Theme vor.

### **2.2.12 Bug: Entscheidungssituation ohne Faktoren**

Wenn eine Entscheidungssituation keine Faktoren hat, stürzt die Android App bei Auswahl dieser Entscheidungssituation ab.

Als Entwickler will ich diesen Fehler beheben.

### **2.2.13 Änderung: Positionierung der Knöpfe bei Faktorbewertung**

Als Benutzer der Android App will ich die Knöpfe "NO PROBLEM", "NOT IDEAL", "PROBLEM" und "NOT RELEVANT" an einer festen Position vorfinden.

### **2.2.14 Änderung: Soziodemografische Daten direkt ersichtlich**

Als Benutzer der Android App will ich die ausgewählte Option der Soziodemografischen Daten direkt in dieser Ansicht sehen. Der Hinweis (z.B. "Please enter your gender") soll bei ausgewählter Option durch die Auswahl ersetzt werden.

### **2.2.15 Änderung: Hervorheben nicht beantworteter Fragen im Fragebogen**

Als Benutzer der Android App will ich beim Versuch, die Antworten des Motivprofil-Fragebogens zu speichern, alle nicht beantworteten Fragen mit einem pinken Rahmen erkennen. Dies soll zusätzlich zur Meldung, dass nicht alle Fragen beantwortet wurden, geschehen.

### **2.2.16 User Story: Splash Screen für Android App**

Als Benutzer der Android App möchte ich auf dem Splash Screen der App eine Einführung lesen können.

Akzeptanzkriterien:

- Der Text kann im Source Code der App geändert werden und kann somit nur mit einer neuen App Version geändert werden.

### **2.2.17 User Story: Entscheidungs-Historie**

Als Benutzer der Android App kann ich meine Durchführungen der Entscheidungssituationen als Historie ansehen.

Akzeptanzkriterien:

- Die App speichert die ausgeführten Entscheidungssituationen mit deren Durchführungs-Zeitpunkt und Handlungsempfehlung.
- Der Benutzer kann sich diese Historie als Liste anzeigen lassen.

## 2.3 User Story: Petras Geschichte & Ablaufdiagramme

Als Externer (Betreuer, Designer, usw.) möchte ich eine einfache Beschreibung zu Decisio erhalten.

Akzeptanzkriterien:

- Ich kann mir innerhalb von fünf Minuten einen Überblick verschaffen.
- Ich sehe die wichtigsten Use Cases in einem Sequenzdiagramm.
- Ich kann basierend auf einem einfachen Beispiel nachvollziehen, wie sich Parameter und Algorithmen im Backend verhalten. Dabei wird ein orange und ein grün bewerteter Entscheidungsfaktor beschrieben.
- In dieser Geschichte wird das Motivprofil auch als Netzdiagramm dargestellt.

## 2.4 User Story: Demo Anleitung zu Petras Geschichte

Als Demonstrator möchte ich Decisio in seinem vollen Funktionsumfang einem Publikum (Hochschule/potentieller Partner oder Kunde) präsentieren können.

Als Demonstrator halte ich mich an das "Skript" (Petras Geschichte). Dieses Skript entspricht einer Schritt-für-Schritt-Anleitung, die mich genau anweist, wie mit der App in einer definierten Situation zu interagieren ist.

Akzeptanzkriterien:

- Die Petra Story enthält exakte Schritt-für-Schritt Anleitungen für die Demo (Screenshots sind erlaubt).
- Ein Laie kann ohne Decisio-Vorwissen die Demo mithilfe der Petra Story durchführen.
- In der Demo wird der Mechanismus (die Algorithmen) anhand der Petra Story praktisch erklärt.
- Es steht ein Data Seed zur Verfügung, welcher alle nötigen Daten für die Durchführung der Demo enthält.

Ausgangslage für Demonstration:

- Die aktuellste App Version ist auf einem Android Gerät installiert
- Schritt-für-Schritt Anleitung für Demonstranten
- Der Data Seed, der die Entscheidungssituation mit den Faktoren beinhaltet (inkl. Mappings), ist eingespielt.

## 2.5 User Story: Anleitung für die Weiterentwicklung von Decisio Front- und Backend

Als Praxispartner möchte ich eine Anleitung für die Weiterentwicklung der Decisio Android App und des Back-Ends zur Verfügung haben.

Akzeptanzkriterien:

- Anhand dieser Anleitung kann ein erfahrener Java-Entwickler (welcher Decisio nicht kennt) Modifikationen an der App rasch umsetzen.
- Anhand dieser Anleitung kann ein erfahrener Java-Entwickler (welcher Decisio nicht kennt) Modifikationen am Back-End rasch umsetzen.
- Der Entwickler ist dabei auf keine proprietäre IDE angewiesen (vi auf Mac/\*nix oder Notepad auf Windows müssen genügen).
- Die Anleitung beschreibt allfällige Open Source Abhängigkeiten (z.B. Maven) und deren Installation auf einem neuen Entwicklungs-PC/Mac.
- Die Anleitung beinhaltet auch Zugriff und Modus Operandi mit der Source Verwaltung.

## 2.6 Spike: System mit fehlenden Daten testen und Probleme beheben

Als Entwickler/Praxispartner möchte ich wissen was passiert, wenn alle Entscheidungssituationen, alle Fragen im Motivprofil-Fragebogen, alle Motivprofil-Spezifikationen, oder alle Einträge der Rating-Tabelle gelöscht sind.

Akzeptanzkriterien:

- Dokumentation dazu, wie die Android App reagiert, ist vorhanden.
- Dokumentation dazu, wie die Administration reagiert, ist vorhanden.
- Gefundene Fehler sind behoben, soweit dies zeitlich in diesem Spike möglich ist.

### 3. Domain-Analyse

#### 3.1 Domain Modell: Android App

Wie in unserer Studienarbeit umgesetzt, hält die Android App nur die momentan wichtigen Informationen für die aktuelle Ansicht in der Domain. Bei Bedarf, d.h. zur Anzeige der entsprechenden Informationen in den verschiedenen Ansichten, lädt sie die benötigten Daten aus der lokalen Datenbank. In Abbildung 19 ist das Domain Modell der Android App dargestellt. Dieses wird in den Abschnitten 3.1.1 bis 3.1.3 noch genauer erklärt.

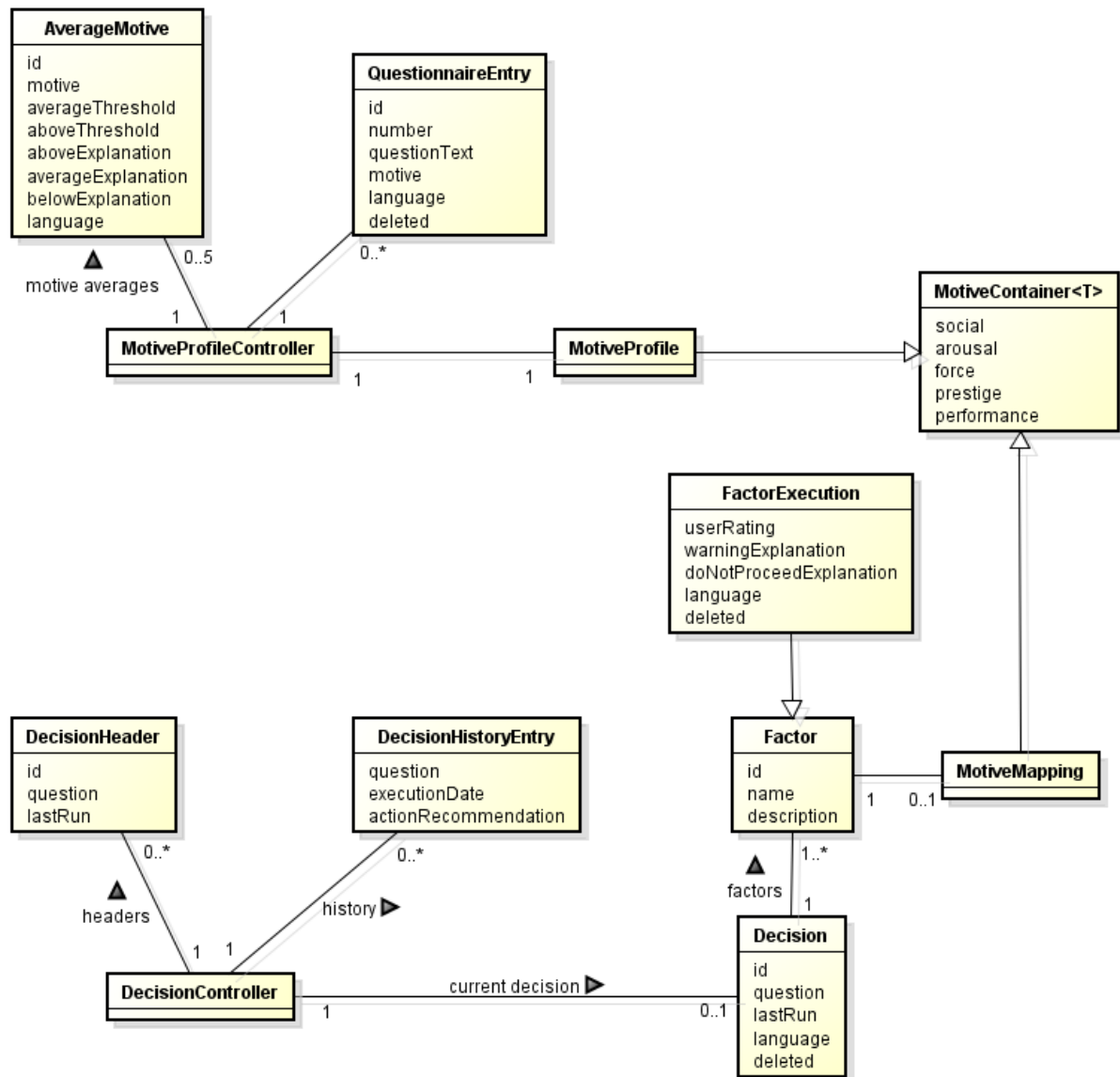


Abbildung 19: Domain Modell der Android App

##### 3.1.1 Erklärungen

Die Domainobjekte haben wir thematisch in zwei Gruppen aufgeteilt:

- **Entscheidungen:** Alle Domainobjekte die zum Ausführen einer Entscheidungssituation oder zur Anzeige der Entscheidungs-Historie nötig sind, werden über die Klasse `DecisionController` angesprochen.

- **Motivprofil:** Alle Domainobjekte, die zum Erstellen des Motivprofils und dessen Visualisierung nötig sind, werden über die Klasse `MotiveProfileController` angesprochen.

Instanzen der Klassen `DecisionController` und `MotiveProfileController` werden nur in den jeweiligen Ansichten erstellt, die Zugriff auf die Entscheidungssituationen bzw. auf das Motivprofil benötigen. In jeder Ansicht (Activity) wird wieder eine neue Instanz erstellt, da es in Android nur mithilfe eines gebundenen Service möglich wäre, dieselben Objekte über mehrere Activities zu verwenden.

### 3.1.2 Wichtige Konzepte

<code>Factor</code>	Stellt einen vorgegebenen Entscheidungsfaktor dar.
<code>FactorExecution</code>	Ein Faktor während der Ausführung einer Entscheidungssituation. Zusätzlich zur Basisklasse <code>Factor</code> werden hier die Einstufung des Benutzers und die anzuzeigenden Detail-Erklärungen gespeichert.
<code>MotiveMapping</code>	Definiert zu jedem Motiv den Wert "Faktor-Mapping" für die Rating-Tabelle als Integer-Werte.
<code>DecisionHeader</code>	Die Headerinformationen der Entscheidungssituationen. Dies wird bei der Auflistung der ausführbaren Entscheidungssituationen verwendet.
<code>MotiveProfile</code>	Enthält die Motivprofil-Werte des Benutzers als Double-Werte, welche aufgrund seiner Antworten zum Motivprofil-Fragebogen berechnet wurden.
<code>AverageMotive</code>	Stellt die Durchschnitts-Motivprofile dar. Pro Motiv gibt es ein Objekt mit den Schwellwerten des Durchschnittsbereichs und den Erklärungen zu den Einstufungen.
<code>MotiveContainer</code>	Diese Klasse stellt die Gemeinsamkeiten der Unterklassen <code>MotiveMapping</code> und <code>MotiveProfile</code> zur Verfügung. Die unterschiedlichen Werte-Typen der Unterklassen (Integer und Double) werden mittels Generics unterstützt.
<code>QuestionnaireEntry</code>	Repräsentiert eine Frage des Motivprofil-Fragebogens.

### 3.1.3 Unterschiede zur Studienarbeit

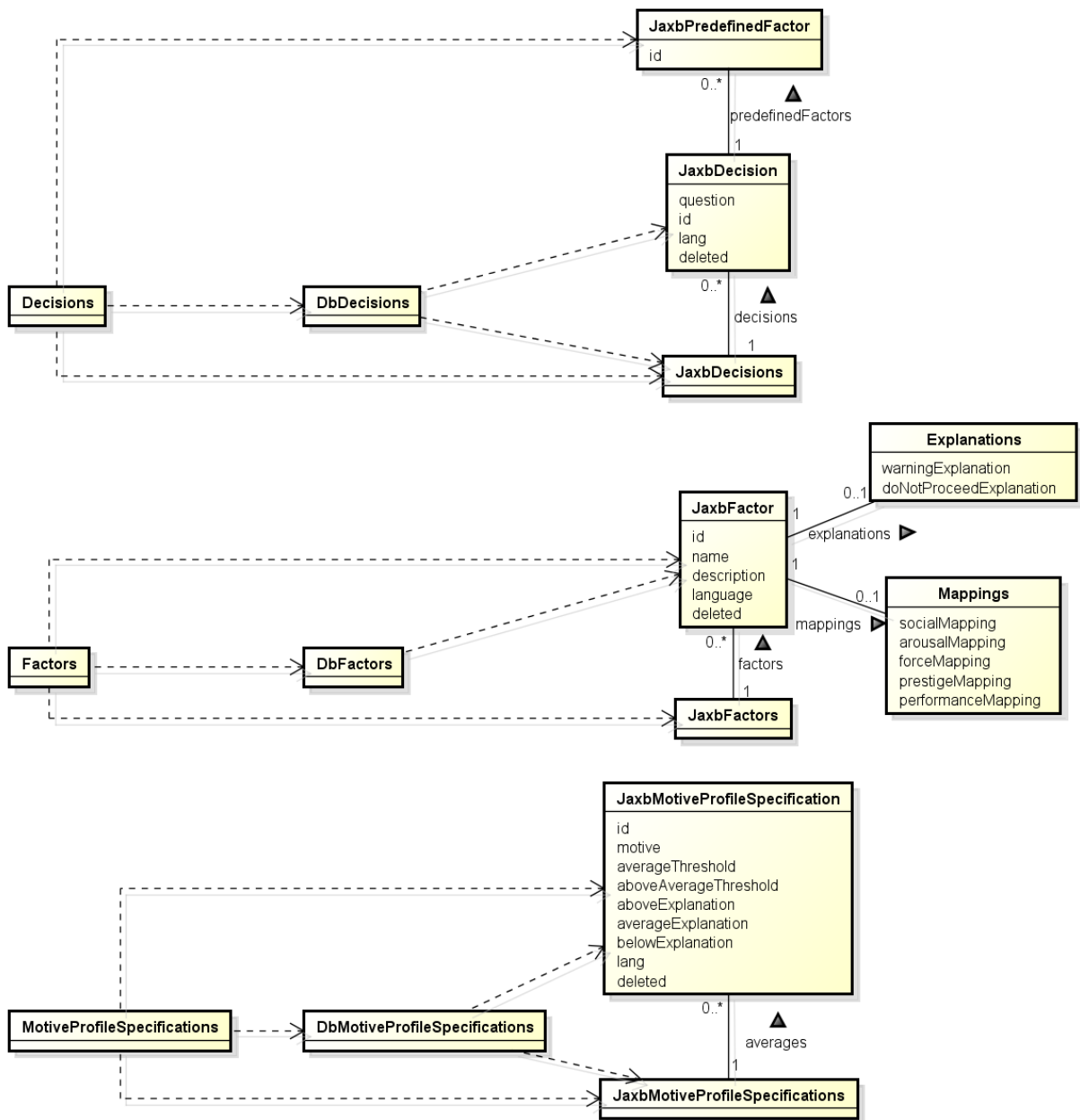
Die Klasse `DecisionController` lädt die Headerinformationen der Entscheidungssituationen erst bei Bedarf und nicht generell im Konstruktor. Daher wurde die Multiplizität von 1..\* auf 0..\* geändert.

Zudem kam die Entscheidungs-Historie als neue Funktionalität hinzu. Dazu muss der `DecisionController` die einzelnen Einträge (Instanzen der Klasse `DecisionHistoryEntry`) bei Bedarf aus der Datenbank laden.

Durch den Einsatz einer Back-End-Applikation sind neu auch die Attribute `id`, `language` und `deleted` in mehreren Domain Objekten dazugekommen.

## 3.2 Domain Modell: Back-End-Applikation

In der Back-End-Applikation setzen wir einen RESTful Web Service ein. Dieser stellt lediglich die Zugriffe auf die Datenbank im Back-End Bereich für die verschiedenen Ressourcen zur Verfügung. Diese Zugriffe können pro Ressource unabhängig behandelt werden, dadurch sieht das Domain Modell sehr abgegrenzt aus.





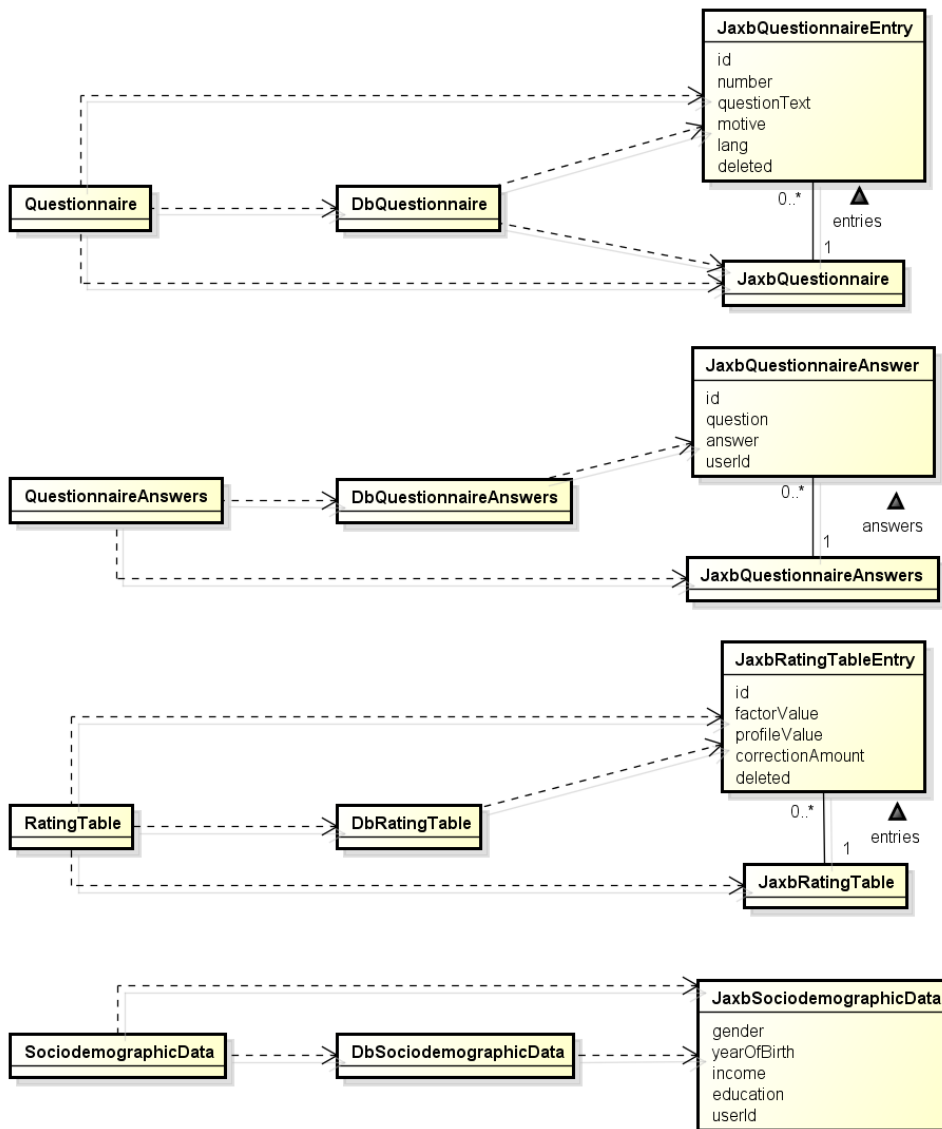


Abbildung 20: Domain Modell der Back-End-Applikation

### 3.2.1 Ergänzungen

Übersichtshalber wurden folgende Abhängigkeiten weggelassen:

- Abhängigkeit von den Klassen [Decisions](#) und [DbDecisions](#) zu [JaxbFactor](#). Diese werden benötigt zur Abfrage der IDs und Namen aller Faktoren für die Bearbeitung einer Entscheidungssituation über die Administration, siehe 4.1.2.4.
- Abhängigkeit von den Klassen [Factors](#) und [DbFactors](#) zu [JaxbDecision](#). Diese werden benötigt zur Abfrage aller Entscheidungssituationen die einen bestimmten Entscheidungsfaktor verwenden, siehe 4.1.2.5.
- Abhängigkeit von den Klassen [Decisions](#), [DbDecisions](#), [Factors](#), [DbFactors](#), [MotiveProfileSpecifications](#), [DbMotiveProfileSpecifications](#), [Questionnaire](#) und [DbQuestionnaire](#) zur Klasse [JaxbLanguage](#). Diese werden benötigt zur Abfrage der vorhandenen Sprachen der jeweiligen Ressourcen.

## 4. Design und Umsetzung

### 4.1 Back-End

Im Back-End Bereich setzen wir eine Java Applikation auf einem Heroku Server [3] ein. Um Änderungen am Code auf dem Heroku Server vornehmen zu können, muss der von Heroku zur Verfügung gestellte Toolbelt [8] lokal installiert werden.

#### 4.1.1 Datenbank

Auf dem Back-End Server setzen wir eine PostgreSQL [9] Datenbank ein. Das Datenbankschema wurde aus den früheren XML-Dateien auf dem SFTP Server abgeleitet und im Kapitel 4.1.1.1 dokumentiert. Um Änderungen an der Datenbank vornehmen zu können, muss PostgreSQL [9] auf dem lokalen Rechner installiert werden. Zudem muss das `bin` Verzeichnis der PostgreSQL Installation zur `PATH` Umgebungsvariable des lokalen Systems, oder des aktuellen Benutzers, hinzugefügt werden.

Danach kann mit den in Tabelle 1 beschriebenen SQL-Skripten das Datenbankschema erstellt, sowie ein initiales Dataset eingefügt werden.

SQL-Skript Datei	Beschreibung
Reset.sql	Löscht bestehende Tabellen in der Datenbank (falls vorhanden) und führt die beiden Skripte "Schema.sql" und "Initialize_DataSet.sql" aus.
Schema.sql	Erstellt das Datenbankschema in der Datenbank.
Initialize_DataSet.sql	Erstellt ein initiales Dataset in der Datenbank. Diese Inhalte wurden von den XML-Dateien der Studienarbeit übernommen.

**Tabelle 1: Beschreibungen der SQL-Skripte**

Um die in Tabelle 1 beschriebenen Skripte auszuführen, muss über den Kommandozeilen-Befehl

```
heroku pg:psql --app decisio-backend HEROKU_POSTGRESQL_COBALT
```

eine PSQL Shell geöffnet werden. Bei erfolgreicher Authentisierung steht die geöffnete PSQL Shell mit einer Verbindung zum Heroku Server einsatzbereit und die Skripte können mit dem Befehl `\i` ausgeführt werden. Beispiel:

```
\i Reset.sql
```

Die PSQL Shell kann mit dem Befehl `\q` wieder geschlossen werden.

Um ein Backup der Datenbank auf dem Heroku Server zu erstellen, kann über das auf dem Heroku Server aktivierte PG Backups Add-On mit dem Befehl

```
heroku pg:backups capture --app decisio-backend
```

ein Backup erstellt werden, welches über den Web-Zugang der Heroku Applikation heruntergeladen werden kann.

#### 4.1.1.1 Datenbankschema

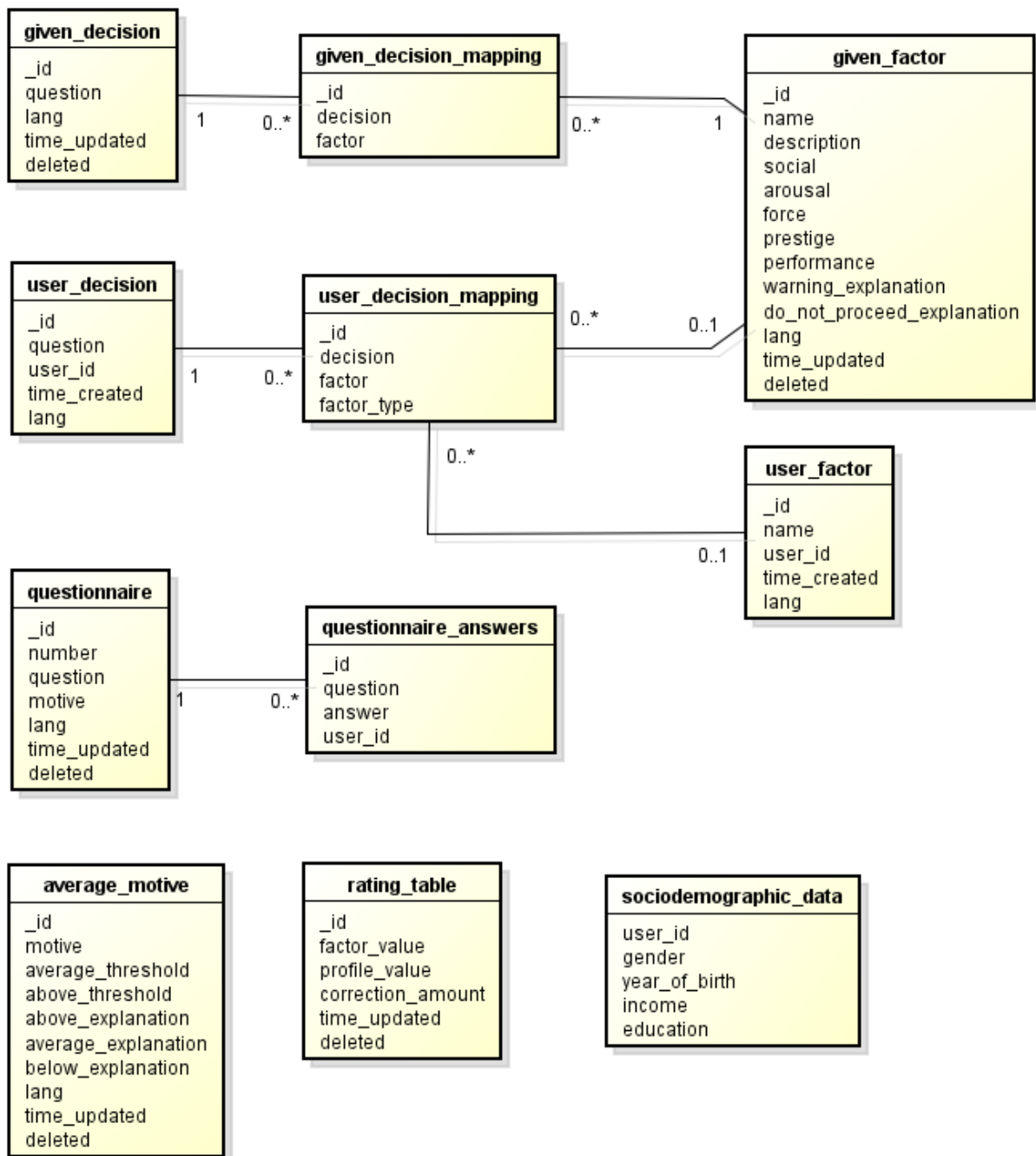


Abbildung 21: Datenbankschema der Back-End-Applikation

#### 4.1.1.2 Connection Pool

Um die Performance möglichst hoch zu halten, verwenden wir einen Connection Pool. Wird beim Pool eine Verbindung angefragt, gibt dieser eine freie zurück oder öffnet eine neue (sofern die maximale Anzahl nicht überschritten ist). Sobald die Applikation die Methode `close` auf der Verbindung aufruft, wird sie als "frei" dem Pool zurückgegeben. [9]

### 4.1.1.3 Löschen von Einträgen

Zu löschende Einträge in der Datenbank werden lediglich mit dem `deleted` Flag als gelöscht markiert. Dies haben wir so umgesetzt, dass die Android Clients beim Entfernen von Einträgen, einfach benachrichtigt werden können. Der entsprechende Eintrag wird im XML-Inhalt ebenfalls mit einem `deleted` Flag markiert. Die Android Clients können dann den entsprechenden Eintrag aus ihrer lokalen Datenbank endgültig löschen.

Zudem könnten die Einträge in der Datenbank dadurch auch später einmal wiederhergestellt werden, falls das Löschen nicht beabsichtigt war.

Für die Tabelle der Motivprofil Spezifikationen (`average_motive`) gibt es eine Unique-Bedingung, sodass eine Spezifikation für ein bestimmtes Motiv nur einmal für dieselbe Sprache definiert werden kann. Diese Bedingung muss jedoch ausser Kraft treten, wenn der Eintrag als gelöscht markiert ist, sonst könnte man die Spezifikation nicht erneut wieder erstellen. Dieses Problem haben wir mit einem "Partial Unique Index" gelöst, also einem eindeutigen Index, der nur für diejenigen Einträge erstellt wird, die das `deleted` Flag nicht gesetzt haben.

Dieselbe Umsetzung haben wir für die Unique-Bedingung in der Rating-Tabelle (nur ein aktiver Eintrag für die gleiche Faktor- und Motivprofil-Wert Kombination) verwendet.

### 4.1.2 RESTful Web Service

Das Back-End stellt einen Web Service zur Verfügung. Dieser arbeitet nach dem REST Paradigma. Gemäss dem Richardson Maturity Model [10] erfüllt der Service *Level 2: HTTP Verbs*.

Wir verwenden `POST` für Update-Operationen und `PUT` zur Erstellung von Ressourcen.

Als Technologie setzen wir Jersey [11], die Referenzimplementation von JAX-RS, ein. Dadurch kann man eine gute Code Trennung für die verschiedenen Ressourcen realisieren, sowie sehr einfach die verschiedenen HTTP Operationen definieren. Die Back-End Administration kommuniziert mit dem REST Service ausschliesslich mit JSON, der bereits implementierte Android Client jedoch weiterhin mit XML. Um dies mit kleinem Aufwand gewährleisten zu können, haben wir uns für den Einsatz von JAXB [12] entschieden, was sich sehr gut mit JAX-RS kombinieren lässt. JAXB erlaubt eine sehr einfache Definition der auszutauschenden Objekte mithilfe von Java Annotationen. Die Objekte, welche mittels JSON oder XML ausgetauscht werden, sind in Kapitel 4.1.2.8 dokumentiert.

Folgend sind die Verfügbaren URLs mit den entsprechenden HTTP-Verben, POST-Bodies und Resultaten aufgelistet. Der Timestamp ist jeweils die Anzahl Millisekunden seit dem 1. Januar 1970, dem Start der Unixzeit.

#### 4.1.2.1 Motivprofil-Fragebogen

URL	HTTP-Verb	POST-Body	Resultat
<code>/questionnaire?lang=:lang</code>	GET	<leer>	Liefert die Fragen in der Sprache <code>:lang</code> . Encoding: JSON.
<code>/questionnaire?lang=:lang&amp;since=:timestamp</code>	GET	<leer>	Liefert die Fragen in der Sprache <code>:lang</code> die sich seit <code>:timestamp</code> geändert haben. Wird <code>:lang</code> nicht angegeben, so wird Englisch als Standard verwendet. Encoding: XML.
<code>/questionnaire/languages/</code>	GET	<leer>	Liefert die vorhandenen Sprachen. Encoding: JSON.

URL	HTTP-Verb	POST-Body	Resultat
/questionnaire/:id	GET	<leer>	Liefert die Frage mit der ID <code>:id</code> . Encoding: JSON.
/questionnaire/	PUT	neues JSON-Objekt	Erstellt die übergebene Frage mit einer generierten ID.
/questionnaire/:id	POST	aktualisiertes JSON-Objekt	Aktualisiert die Frage mit der ID <code>:id</code> gemäss dem übergebenen Objekt.
/questionnaire/:id	DELETE	<leer>	Löscht die Frage mit der ID <code>:id</code> .
/questionnaire/up/:id	POST	<leer>	Verschiebt die Frage mit der ID <code>:id</code> um eine Position nach oben.
/questionnaire/down/:id	POST	<leer>	Verschiebt die Frage mit der ID <code>:id</code> um eine Position nach unten.

**Tabelle 2: RESTful Web Service API: Motivprofil-Fragebogen**

Beim Löschen einer Frage werden alle Fragen der Sprache mit einer höheren Position um eins nach unten verschoben. Somit entstehen keine Löcher/leeren Positionen. Wird eine Frage verschoben, muss nicht geprüft werden, ob die Ziel-Position besetzt ist, da diese immer besetzt ist. Dadurch kann die Frage an der Ziel-Position in der entsprechenden Sprache mit der zu verschiebenden Frage getauscht werden.

#### 4.1.2.2 Motivprofil Spezifikationen

URL	HTTP-Verb	POST-Body	Resultat
/motive-profile-specifications/languages/	GET	<leer>	Liefert eine Liste mit den vorhandenen Sprachen der Motivprofil Spezifikationen. Encoding: JSON.
/motive-profile-specifications/:lang	GET	<leer>	Liefert die Spezifikationen in der Sprache <code>:lang</code> . Es werden fünf Spezifikationen geliefert (für jedes Motiv eine Spezifikation). Encoding: JSON.
/motive-profile-specifications/:lang? since=:timestamp	GET	<leer>	Liefert die Spezifikationen der Sprache <code>:lang</code> , die seit <code>:timestamp</code> geändert wurden. Encoding: XML.
/motive-profile-specifications/:lang/:motive	GET	<leer>	Liefert die Spezifikation des Motivs <code>:motive</code> in der Sprache <code>:lang</code> . Encoding: JSON.
/motive-profile-specifications/:lang	PUT	Array neuer JSON-Objekte	Erstellt die Spezifikationen gemäss des übergebenen Arrays für die Sprache <code>:lang</code> . Es wird ein Array mit den fünf Spezifikationen benötigt (für jedes Motiv eine Spezifikation).

URL	HTTP-Verb	POST-Body	Resultat
/motive-profile-specifications/:lang/:motive	POST	aktualisiertes JSON-Objekt	Aktualisiert die Spezifikation des Motivs <code>:motive</code> in der Sprache <code>:lang</code> gemäss dem übergebenen Objekt.
/motive-profile-specifications/:lang	DELETE	<leer>	Löscht die Spezifikationen aller fünf Motive in der Sprache <code>:lang</code> .

**Tabelle 3: RESTful Web Service API: Motivprofil Spezifikationen**

Die PUT und DELETE Operationen erstellen bzw. löschen jeweils alle Einträge für eine gesamte Sprachdefinition. Dies wurde so gewählt, da es keinen Sinn macht, Spezifikationen in einer Sprache zu haben, bei der nicht alle fünf Motive definiert sind.

#### 4.1.2.3 Rating-Tabelle

URL	HTTP-Verb	POST-Body	Resultat
/rating-table/	GET	<leer>	Liefert alle Einträge der Rating-Tabelle. Encoding: JSON.
/rating-table?since=:timestamp	GET	<leer>	Liefert alle Einträge der Rating-Tabelle, die seit <code>:timestamp</code> geändert wurden. Encoding: XML.
/rating-table/:factorValue	GET	<leer>	Liefert die fünf Einträge aus der Rating-Tabelle, welche den Faktorwert <code>:factorValue</code> aufweisen. Encoding: JSON.
/rating-table/	PUT	Array neuer JSON-Objekte	Erstellt die Einträge gemäss des übergebenen JSON-Arrays. Es werden fünf Einträge benötigt.
/rating-table/:factorValue	POST	Array aktualisierter JSON-Objekte	Aktualisiert die Einträge der Rating-Tabelle, welche den Faktorwert <code>:factorValue</code> aufweisen, mit den Einträgen aus dem JSON-Array.
/rating-table/:factorValue	DELETE	<leer>	Markiert alle Einträge der Rating-Tabelle, die den Faktorwert <code>:factorValue</code> aufweisen als gelöscht.  Ist nur für den aktuell höchsten Faktorwert möglich.

**Tabelle 4: RESTful Web Service API: Rating-Tabelle**

Bei der PUT Operation wird automatisch der nächste freie Faktorwert ermittelt. Es werden nur die Einträge berücksichtigt, die nicht als gelöscht markiert sind. Somit ist die Nummerierung der Faktorwerte durchgehend. Die Faktorwerte stellen somit eine Skala dar.

Bei der DELETE Operation wird geprüft, ob der gewünschte Eintrag den aktuell höchsten Faktorwert aufweist. Wenn ja, wird er als gelöscht markiert. Ansonsten wird eine Fehlermeldung erzeugt.

#### 4.1.2.4 Vorgegebene Entscheidungssituationen

URL	HTTP-Verb	POST-Body	Resultat
/decisions?lang=:lang	GET	<leer>	Liefert alle Entscheidungssituationen der Sprache <code>:lang</code> . Encoding: JSON.
/decisions?lang=:lang&since=:timestamp	GET	<leer>	Liefert die Entscheidungssituationen der Sprache <code>:lang</code> die sich seit <code>:timestamp</code> geändert haben. Wenn <code>:lang</code> nicht angegeben wurde, so wird Englisch als Standard verwendet. Encoding: XML.
/decisions/languages/	GET	<leer>	Liefert eine Liste der vorhandenen Sprachen. Encoding: JSON.
/decisions/factors/:lang	GET	<leer>	Liefert die ID und Namen aller verfügbaren Faktoren der Sprache <code>:lang</code> . Dies wird für die Auflistung in der Administration benötigt. Encoding: JSON.
/decisions/:id	GET	<leer>	Liefert die Entscheidungssituation mit der ID <code>:id</code> . Encoding: JSON.
/decisions/	PUT	neues JSON-Objekt	Erstellt die übergebene Entscheidungssituation mit einer neu generierten ID.
/decisions/:id	POST	aktualisiertes JSON-Objekt	Aktualisiert die Entscheidungssituation mit der ID <code>:id</code> gemäss dem übergebenen Objekt.
/decisions/:id	DELETE	<leer>	Markiert die Entscheidungssituation mit der ID <code>:id</code> als gelöscht.

**Tabelle 5: RESTful Web Service API: Vorgegebene Entscheidungssituationen**

Bei der DELETE Operation werden zusätzlich die Faktor-Mappings der betroffenen Entscheidungssituation gelöscht.

#### 4.1.2.5 Vorgegebene Entscheidungsfaktoren

URL	HTTP-Verb	POST-Body	Resultat
/factors?lang=:lang	GET	<leer>	Liefert die Entscheidungsfaktoren der Sprache <code>:lang</code> . Encoding: JSON.

URL	HTTP-Verb	POST-Body	Resultat
/factors?lang=:lang&since=:timestamp	GET	<leer>	Liefert die Entscheidungsfaktoren der Sprache <code>:lang</code> die sich seit <code>:timestamp</code> geändert haben. Wird <code>:lang</code> nicht angegeben, so wird Englisch als Standard verwendet. Encoding: XML.
/factors/languages/	GET	<leer>	Liefert eine Liste der vorhandenen Sprachen. Encoding: JSON.
/factors/:id/decisions	GET	<leer>	Liefert den Fragetext aller Entscheidungssituationen, die den Faktor mit der ID <code>:id</code> verwenden. Encoding: JSON.
/factors/:id	GET	<leer>	Liefert den Entscheidungsfaktor mit der ID <code>:id</code> . Encoding: JSON.
/factors/	PUT	neues JSON-Objekt	Erstellt den übergebenen Entscheidungsfaktor mit einer neu generierten ID.
/factors/:id	POST	aktualisiertes JSON-Objekt	Aktualisiert den Entscheidungsfaktor mit der ID <code>:id</code> gemäss dem übergebenen Objekt.
/factors/:id	DELETE	<leer>	Markiert den Entscheidungsfaktor mit der ID <code>:id</code> als gelöscht.

**Tabelle 6: RESTful Web Service API: Vorgegebene Entscheidungsfaktoren**

Bei der DELETE Operation werden zusätzlich alle vorhandenen Mappings zu Entscheidungssituationen mit dem betroffenen Entscheidungsfaktor gelöscht, d.h. der Faktor wird von den Entscheidungssituationen entfernt und die entsprechenden Aktualisiert-Zeitstempel der Entscheidungssituationen geändert.

#### 4.1.2.6 Antworten des Motivprofil-Fragebogens

URL	HTTP-Verb	POST-Body	Resultat
/questionnaire-answers/:uuid	PUT	Neue oder aktualisierte Antworten in XML.	Speichert die Antworten des Benutzers mit der Identifikation <code>:uuid</code> zum Motivprofil-Fragebogen. Dabei werden die allenfalls vorhandenen alten Daten des Benutzers vorgängig gelöscht.

**Tabelle 7: RESTful Web Service API: Antworten des Motivprofil-Fragebogens**



### 4.1.2.7 Soziodemografische Daten des Benutzers

URL	HTTP-Verb	POST-Body	Resultat
/sociodemographic-data/:uuid	PUT	Neue oder aktualisierte XML Daten.	Speichert die soziodemografischen Daten des Benutzers mit der Identifikation <code>:uuid</code> . Dabei werden die allenfalls vorhandenen alten Daten des Benutzers vorgängig gelöscht.

**Tabelle 8: RESTful Web Service API: Soziodemografische Daten des Benutzers**

### 4.1.2.8 Fehlerbehandlung im RESTful Web Service

Damit in der Administration eine Fehlermeldung angezeigt werden kann, haben wir uns dafür entschieden, im Fehlerfall eine `RestException` zu werfen. Dies ist eine eigene Klasse von uns, welche die Klasse `WebApplicationException` erweitert. Wenn eine `WebApplicationException` geworfen wird, wandelt Jersey diese automatisch in eine 500 Server Error HTTP Antwort mit dem entsprechenden Fehlermeldungstext um und liefert sie als Antwort. Den Fehlermeldungstext senden wir direkt als Plaintext, also nicht in ein JSON Objekt eingepackt.

Somit müssen wir in jeder definierten HTTP Operation der entsprechenden Ressourcen die allfällig geworfenen `SQLExceptions` oder `RuntimeExceptions` abfangen und die entsprechende Fehlermeldung daraus in eine `RestException` packen.

### 4.1.3 Android Client Synchronisierung

Um den Netzwerk Verkehr gering zu halten, sowie den Android Client in der Synchronisierung zu beschleunigen, haben wir uns dafür entschieden, dass lediglich die geänderten Einträge heruntergeladen werden. Dazu sendet der Android Client den Zeitpunkt seiner letzten Synchronisierung zum Back-End-Server (`since` Angabe) und erhält nur die seit diesem Zeitpunkt geänderten Daten vom RESTful Web Service. Dieses Vorgehen nennen wir **partielle Synchronisierung**.

Im Zusammenspiel mit der Mehrsprachigkeit der Daten auf dem Back-End-Server bringt die partielle Synchronisierung ein Problem mit sich: Wenn eine erstellte Sprache wieder gelöscht wird, dann müssen die Android Clients wieder auf die Standardsprache Englisch zurück weichen. Dies nennen wir einen **Fallback**. Der Fallback wird in mehreren Szenarios vollzogen:

- Wenn der Android Client bereits einmal die neuerdings als gelöscht markierte Sprache (z.B. Italienisch) synchronisiert hat, so sollen ihm alle nicht gelöschten Einträge in Englisch gesendet werden (initialer Fallback).
- Wenn der Android Client bereits einmal synchronisiert hat, seitdem die angeforderte Sprache als gelöscht markiert wurde, so wurden ihm bereits die Englischen Einträge gesendet. Daher muss er lediglich die allfällig geänderten Englischen Einträge aktualisieren.
- Wenn der Android Client eine Sprache anfordert, die noch nie existiert hat, so kann die normale partielle Synchronisierung für Englisch durchgeführt werden.

Das für mehrere Ressourcen benötigte Verhalten wurde in den `DbHelper` genommen, um duplizierten Code zu vermeiden. Die Fallback-Szenarios sind in Abbildung 22 noch einmal visuell dargestellt.

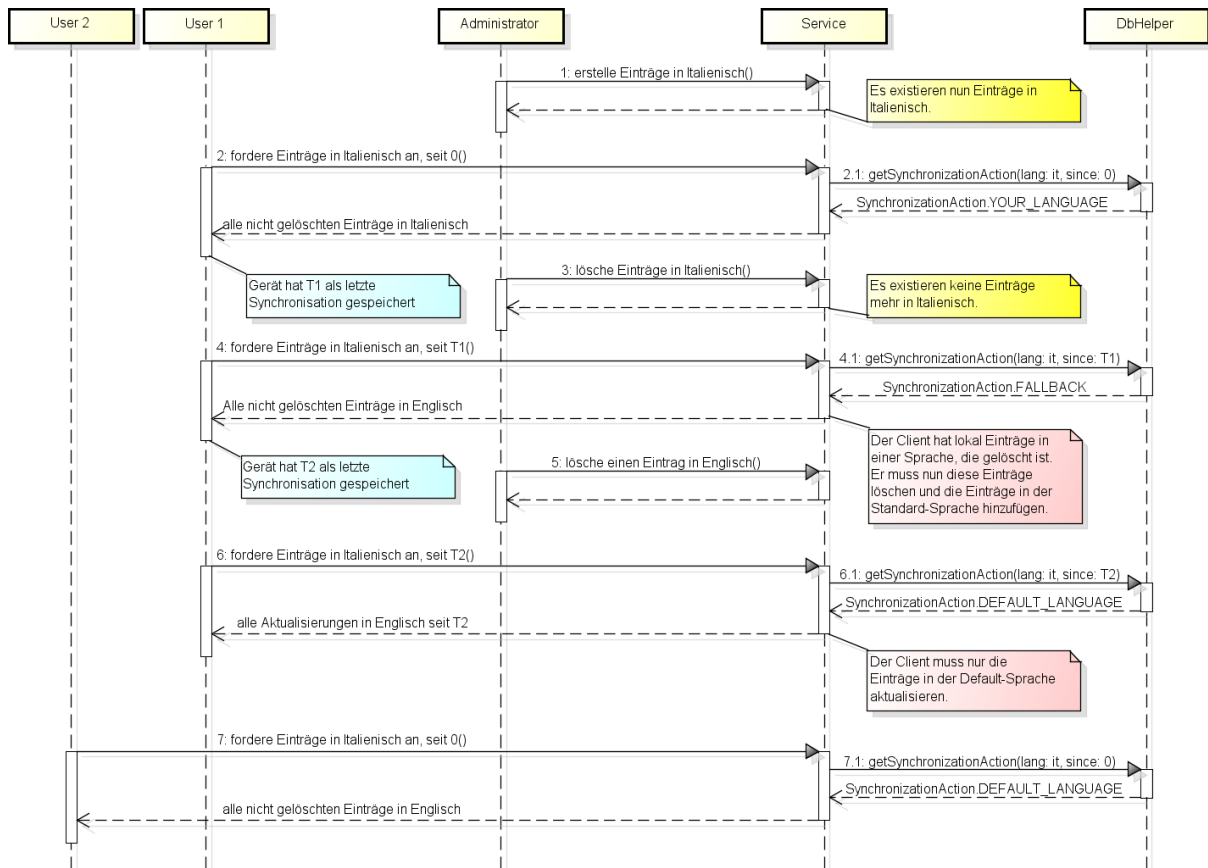


Abbildung 22: Visualisierung der Fallback Szenarien

#### 4.1.3.1 Wiedererstellen einer gelöschten Sprache

Wird eine gelöschte Sprache trotzdem wieder erstellt, so kann es zu Problemen führen, wenn der Android Client bereits einmal die Englischen Einträge durch den Fallback synchronisiert hat. In diesem Szenario werden dem Android Client lediglich die seit der letzten Synchronisierung wieder erstellten Einträge in der angefragten Sprache gesendet. Dadurch hat der Client nun zwei Sprachversionen vorhanden.

Deshalb ist folgende Änderung im Android Client nötig: Wenn der Android Client bei der Synchronisierung Einträge in einer Sprache erhält, z.B. Italienisch, so muss er lokal alle nicht italienischen Einträge löschen. Diese Änderungen wurden im entsprechenden Sub-Abschnitt von 4.3.5 beschrieben.

#### 4.1.4 Datenaustausch Formate

Bei den XML Spezifikationen ist bei den Einträgen ein Attribut `deleted` vorhanden. Dies ist ein Boolean Wert, der anzeigt, ob der Eintrag als gelöscht markiert ist. Ist dieses Flag gesetzt, so werden nur die Attribute `id` und `deleted` zum Android Client übermittelt, sodass dieser den Eintrag lokal löschen kann.

#### 4.1.4.1 Sprachangaben

JSON Objekt	Feld	Beschreibung	Vorkommen
language	---	Eine Sprachangabe für die angeforderte Ressource.	0..* (als Array)
language	language	Eine Sprachidentifikationen. Wird im Stil 'en' oder 'en-us' geliefert.	1

**Tabelle 9: JSON Spezifikation für Sprachangaben einer angeforderten Ressource**

#### 4.1.4.2 Motivprofil-Fragebogen

XML Element	Attribut	Beschreibung	Vorkommen
questionnaire	---	Liste der Fragen des Motivprofil-Fragebogens für eine bestimmte Sprache.	1
questionnaire/question	---	Eine bestimmte Frage des Motivprofil-Fragebogens.	0..* (als Array)
questionnaire/question	id	Die ID dieser Frage.	1
questionnaire/question	number	Die Nummer (Position) dieser Frage.	1
questionnaire/question	motive	Das Motiv dieser Frage.	1
questionnaire/question	lang	Die Sprache dieser Frage.	1
questionnaire/question	<value>	Der Fragetext dieser Frage.	1

**Tabelle 10: XML Spezifikation des Motivprofil-Fragebogens**

JSON Objekt	Feld	Beschreibung	Vorkommen
question	---	Liste der Fragen des Motivprofil-Fragebogens für eine bestimmte Sprache.	0..* (als Array)
question	id	Die ID dieser Frage.	1
question	number	Die Nummer (Position) dieser Frage.	1
question	motive	Das Motiv dieser Frage.	1
question	lang	Die Sprache dieser Frage.	1
question	<value>	Der Fragetext dieser Frage.	1

**Tabelle 11: JSON Spezifikation des Motivprofil-Fragebogens**

#### 4.1.4.3 Antworten zum Motivprofil-Fragebogen

XML Element	Attribut	Beschreibung	Vorkommen
questionnaireAnswers	---	Liste der Antworten zu den Fragen des Motivprofil-Fragebogens.	1
questionnaireAnswers/questionnaireAnswer	---	Eine Antwort auf eine bestimmte Frage des Motivprofil-Fragebogens.	0..* (als Array)

XML Element	Attribut	Beschreibung	Vorkommen
questionnaireAnswers /questionnaireAnswer /question	---	Der Fragetext dieser Frage.	1
questionnaireAnswers /questionnaireAnswer /answer	---	Die Antwort des Benutzers auf diese Frage (Wert zwischen 1 und 5).	1

**Tabelle 12: XML Spezifikation der Antworten zum Motivprofil-Fragebogen**

Bei der Übermittlung der Antworten zum Motivprofil-Fragebogen haben wir uns für das Mitschicken des Fragetexts, und somit gegen das Mitschicken der ID der Frage, entschieden. Das hat den Vorteil, dass immer klar ist, auf welche Frage der Benutzer geantwortet hat. Würde man nur die ID der Frage mitschicken, so wäre bei einer Überarbeitung der Frage nicht klar, auf welche Version der Frage der Benutzer geantwortet hat.

#### 4.1.4.4 Motivprofil Spezifikationen

XML Element	Attribut	Beschreibung	Vorkommen
motiveProfileSpecifications	---	Die Liste von Motivprofil Spezifikationen für eine bestimmte Sprache.	1
motiveProfileSpecifications /average	---	Eine Motivprofil Spezifikation, bezogen auf ein bestimmtes Motiv für eine bestimmte Sprache.	5
motiveProfileSpecifications /average	motive	Das Motiv dieser Motivprofil Spezifikation.	1
motiveProfileSpecifications /average	average Threshold	Der Schwellwert für dieses Motiv, ab dem man als durchschnittlich gilt.	1
motiveProfileSpecifications /average	above Average Threshold	Der Schwellwert für dieses Motiv, ab dem man als überdurchschnittlich gilt.	1
motiveProfileSpecifications /average	lang	Die Sprache dieser Motivprofil Spezifikation.	1
motiveProfileSpecifications /average /aboveExplanation	---	Der Beschreibungstext für überdurchschnittliche Wertungen für dieses Motiv.	1
motiveProfileSpecifications /average /averageExplanation	---	Der Beschreibungstext für durchschnittliche Wertungen für dieses Motiv.	1
motiveProfileSpecifications /average /belowExplanation	---	Der Beschreibungstext für unterdurchschnittliche Wertungen für dieses Motiv.	1

**Tabelle 13: XML Spezifikation für Motivprofil Spezifikationen**

JSON Objekt	Feld	Beschreibung	Vorkommen
average	---	Eine Motivprofil Spezifikation, bezogen auf ein bestimmtes Motiv für eine bestimmte Sprache.	1 oder 5 (als Array)
average	motive	Das Motiv dieser Motivprofil Spezifikation.	1
average	averageThreshold	Der Schwellwert für dieses Motiv, ab dem man als durchschnittlich gilt.	1
average	aboveAverage Threshold	Der Schwellwert für dieses Motiv, ab dem man als überdurchschnittlich gilt.	1
average	lang	Die Sprache dieser Motivprofil Spezifikation.	1
average	aboveExplanation	Der Beschreibungstext für überdurchschnittliche Wertungen für dieses Motiv.	1
average	averageExplanation	Der Beschreibungstext für durchschnittliche Wertungen für dieses Motiv.	1
average	belowExplanation	Der Beschreibungstext für unterdurchschnittliche Wertungen für dieses Motiv.	1

**Tabelle 14: JSON Spezifikation für Motivprofil Spezifikationen**

#### 4.1.4.5 Rating-Tabelle

XML Element	Attribut	Beschreibung	Vorkommen
ratingTable	---	Alle Einträge der Rating-Tabelle.	1
ratingTable/entry	---	Ein einzelner Eintrag in der Rating-Tabelle.	0..*
ratingTable/entry	factor Value	Der Faktorwert des Eintrags. Dieser Wert wird von den Entscheidungsfaktoren verwendet.	1
ratingTable/entry	profile Value	Der Motivprofil Wert des Eintrags.	1
ratingTable/entry	correction Amount	Der Korrekturbetrag des Eintrags. Dieser wird über factorValue und profileValue in der Rating-Tabelle gesucht.	1
ratingTable/entry	id	Die ID dieses Eintrags. Wird zur eindeutigen Identifikation bei der Synchronisierung benötigt.	1
ratingTable/entry	deleted	Gelöscht-Markierung, siehe Abschnitt 4.1.2.8	1

**Tabelle 15: XML Spezifikation der Rating-Tabellen Einträge**

JSON Objekt	Feld	Beschreibung	Vorkommen
entry	---	Ein einzelner Eintrag in der Rating-Tabelle.	1 oder * (als Array)
entry	factorValue	Der Faktorwert des Eintrags. Dieser Wert wird von den Entscheidungsfaktoren verwendet.	1
entry	profileValue	Der Motivprofil Wert des Eintrags.	1
entry	correctionAmount	Der Korrekturbetrag des Eintrags. Dieser wird über factorValue und profileValue in der Rating-Tabelle gesucht.	1
entry	id	Die ID dieses Eintrags. Wird zur eindeutigen Identifikation bei der Synchronisierung benötigt.	1

**Tabelle 16: JSON Spezifikation der Rating-Tabellen Einträge**

#### 4.1.4.6 Vorgegebene Entscheidungssituationen

XML Element	Attribut	Beschreibung	Vorkommen
decisions	---	Liste der vorgegebenen Entscheidungssituationen für eine bestimmte Sprache.	1
decisions/decision	---	Eine bestimmte Entscheidungssituation.	0..* (als Array)
decisions/decision	id	Die ID dieser Entscheidungssituation.	1
decisions/decision	question	Die Frage dieser Entscheidungssituation.	0..1
decisions/decision	lang	Die Sprache dieser Entscheidungssituation.	0..1
decisions/decision	deleted	Gelöscht-Markierung, siehe Abschnitt 4.1.2.8	1
decisions/decision /predefinedFactor	--	Vorgegebene Entscheidungsfaktoren, die sich auf diese Entscheidungssituation auswirken.	0..* (als Array)
decisions/decision /predefinedFactor	id	Die ID des vorgegebenen Entscheidungsfaktors.	1

**Tabelle 17: XML Spezifikation der vorgegebenen Entscheidungssituationen**

JSON Objekt	Feld	Beschreibung	Vorkommen
decision	---	Liste der vorgegebenen Entscheidungssituationen für eine bestimmte Sprache.	0..* (als Array)
decision	id	Die ID dieser Entscheidungssituation.	1

JSON Objekt	Feld	Beschreibung	Vorkommen
decision	question	Die Frage dieser Entscheidungssituation.	1
decision	lang	Die Sprache dieser Entscheidungssituation.	1
decision	<value>	Der Fragetext dieser Frage.	1
decision /predefinedFactor	--	Vorgegebene Entscheidungsfaktoren, die sich auf diese Entscheidungssituation auswirken.	0..* (als Array)
decision /predefinedFactor	id	Die ID des vorgegebenen Entscheidungsfaktors.	1

**Tabelle 18: JSON Spezifikation der vorgegebenen Entscheidungssituationen**

#### 4.1.4.7 Soziodemografische Daten

XML Element	Attribut	Beschreibung	Vorkommen
sociodemographicData	---	Soziodemografische Daten eines bestimmten Benutzers.	1
sociodemographicData /gender	---	Die Angabe des Geschlechts des Benutzers.	1
sociodemographicData /yearOfBirth	---	Die Angabe des Geburtsjahres des Benutzers.	1
sociodemographicData /income	---	Die Angabe der Einkommensklasse des Benutzers.	1
sociodemographicData /education	---	Die Angabe der Ausbildung des Benutzers.	1

**Tabelle 19: XML Spezifikation der soziodemografischen Daten**

#### 4.1.5 Zugriffsschutz

Ein Aspekt des Zugriffsschutzes ist das Verhindern von HTTP Aufrufen. Dazu haben wir den Filter `ch.hsr.decisio.HttpFilter` erstellt. Dieser leitet alle HTTP Requests nach HTTPS weiter. In der `web.xml` Datei wird er Jetty [13] bekannt gemacht. Als URL-Pattern ist `/*` gesetzt, womit er auf alle Seiten angewendet wird. Versucht man nun auf die Administration oder eine Ressource mit HTTP zuzugreifen, wird man automatisch auf dieselbe Seite/Ressource mit Verwendung von HTTPS weitergeleitet.

Zudem ist eine einfache Benutzer Authentisierung erforderlich. Diese haben wir in der Konfigurationsdatei `web.xml` und zusätzlichem Code in der `Main` Klasse gelöst. Dabei weisen wir den Servlet Container (bei Heroku [3] ist dies Jetty [13]) an, eine HTTP Basic Authentication durchzuführen:

- In der `web.xml` Datei haben wir einen `security-constraint` für das URL-Pattern `/admin/*` und somit die gesamte Administration gesetzt, für welchen die admin-Rolle gesetzt wurde.
- Um nicht-authentisierte manuelle Requests über die REST Schnittstelle, d.h. Anfragen die nicht über die Administration gemacht werden, auszuschliessen, haben wir einen zweiten `security-constraint` für das URL-Pattern `/*` für die HTTP Methoden PUT, POST und

DELETE eingefügt. Somit ist auch für die manuelle Veränderung von Daten auf dem Back-End über die REST Schnittstelle eine Authentisierung nötig.

- Damit das Hochladen der Benutzerdaten trotzdem funktioniert, wurde ein weiterer `security-constraint` hinzugefügt. Dieser hat die URL-Pattern `/sociodemographic-data/*` und `/questionnaire-answers/*` für die HTTP Methode PUT. Der Tag `auth-constraint` ist nicht vorhanden, somit ist keine Authentisierung nötig.
- In der `Main` Klasse haben wir die Verwendung eines `HashLoginService` hinzugefügt, welcher den Benutzer zur definierten admin-Rolle aus der angegebenen Datei `realm.properties` zuweist. In dieser Datei können somit die gültigen Benutzer-Namen und deren Passwörter editiert werden.

## 4.2 Administration

Die Administration wurde mit AngularJS [1] erstellt. Die Aufteilung der Module wurde in Abschnitt 4.2.1 beschrieben. Oft gebrauchte Funktionen wurden im Root-Scope [14] definiert und sind in Abschnitt 0 dokumentiert.

Für die Kommunikation mit dem RESTful Web Service verwenden wir sogenannte Services. Dabei wird für jede Ressource des RESTful Web Service ein eigener Angular Service verwendet. Diese sind in Abschnitt 4.2.3 dokumentiert.

Für die Handhabung der Daten für die entsprechenden Ansichten (Views) setzen wir separate Controller ein. Diese sind in Abschnitt 4.2.4 beschrieben.

### 4.2.1 Module

#### 4.2.1.1 decisioAdmin

Dies ist das Haupt-Modul der Applikation. Es enthält den Route-Provider [15] welcher den Inhalt aufgrund des gewählten Tabs definiert. Anhand des aktuellen URLs werden die zu verwendende View und der zu verwendende Controller gesetzt.

Da der Umfang der Applikation noch gering und übersichtlich ist, haben wir keine weiteren Module erstellt.

### 4.2.2 Root-Scope

Der Root-Scope [14] entspricht einem globalen Scope, aber wird von AngularJS trotzdem gekapselt. Er enthält die Funktionen und Eigenschaften, die von mehreren Controllern verwendet werden.

#### 4.2.2.1 errorFunction

Diese Funktion dient zur einheitlichen Fehlerbehandlung in der Administration. Im Fehlerfall packt diese den HTTP Status Code, sowie dessen Status Beschreibung in den Titel, und den Inhalt der Antwort (als Plaintext) als Inhalt in ein Fehlermeldungsdialog und zeigt diesen an.

### 4.2.3 Services

Die Services verwenden `$resource` [16] von AngularJS [1]. Damit lässt sich mit einem RESTful Web Service interagieren.

Ein Service stellt mehrere Aktionen (actions) zur Verfügung. Jede Aktion hat eine Methode (method) und einen URL. Die Methode definiert, welche HTTP Methode für die Aktion verwendet werden soll. Die URL spezifiziert den Ziel-Pfad des Requests. Pro Service können neben den standardmässigen



Aktionen (get, save, query, remove und delete) auch noch zusätzliche Aktionen definiert werden, welche die Controller dann ausführen können.

Wie in Abschnitt 4.1.2 beschrieben, wird zur Erstellung PUT und zur Bearbeitung POST verwendet. Deshalb definiert jeder Service eine neue Aktion namens `create` mit der Methode PUT. Die Aktion `save` verwendet bereits die Methode POST.

#### 4.2.3.1 Questionnaire

Der Basis-Pfad dieser Ressource ist `/questionnaire/:id`, zur Erklärung der entsprechenden Pfade siehe Abschnitt 4.1.2.1.

Der Service definiert zu den standardmässigen Aktionen noch zusätzlich folgende Aktionen mit den entsprechenden Ziel-URLs:

Aktion	URL
<code>getLanguages</code>	<code>/questionnaire/languages</code>
<code>up</code>	<code>/questionnaire/up/:id</code>
<code>down</code>	<code>/questionnaire/down/:id</code>

**Tabelle 20: Zusätzliche Aktionen des Angular-Service Questionnaire**

#### 4.2.3.2 MotiveProfileSpecs

Der Basis-Pfad dieser Ressource ist `/motive-profile-specifications/:lang/:motive`, wobei das Motiv nicht immer gesetzt ist. Zur Abfrage der vorhandenen Sprachen wird der Pfad `/motive-profile-specifications/languages/` verwendet und als zusätzliche Aktion `getLanguages` auf der Ressource zur Verfügung gestellt.

Zur Erklärung der entsprechenden Pfad-Abschnitte siehe Abschnitt 4.1.2.2.

#### 4.2.3.3 RatingTable

Der Basis-Pfad dieser Ressource ist `/rating-table/:id`, zur Erklärung der entsprechenden Pfade siehe Abschnitt 4.1.2.2.

Ein Eintrag besteht aus den fünf Korrektur-Beträgen, die für einen Faktor-Wert definiert sind. Dabei ist ein Korrektur-Betrag jeweils für einen möglichen Wert (1 bis 5) der Motivprofile der Benutzer definiert. Diese Einträge werden in einem Array gespeichert. `$resource` [16] von Angular kann zwar bei den Aktionen mit Arrays umgehen, aber fügt dem Array selbst die Aktion nicht an, sondern nur den Elementen im Array. Damit beim Erstellen und Ändern jeweils nicht alle fünf Beträge separat gesendet werden müssen, senden die entsprechenden Controller von Hand mit Hilfe von `$http` [17] den entsprechenden Request (PUT oder POST) zum RESTful Web Service.

#### 4.2.3.4 Decisions

Der Basis-Pfad dieser Ressource ist `/decisions/:id`, zur Erklärung der entsprechenden Pfade siehe Abschnitt 4.1.2.4.

Der Service definiert zu den standardmässigen Aktionen noch zusätzlich folgende Aktionen mit den entsprechenden Ziel-URLs:

Aktion	URL
getLanguages	/decisions/languages
getFactors	/decisions/factors/:lang

**Tabelle 21: Zusätzliche Aktionen des Angular-Service Decisions**

#### 4.2.3.5 Factors

Der Basis-Pfad dieser Ressource ist `/factors/:id`, zur Erklärung der entsprechenden Pfade siehe Abschnitt 4.1.2.5.

Der Service definiert zusätzlich zu den standardmässigen Aktionen noch folgende Aktionen mit den entsprechenden Ziel-URLs:

Aktion	URL	HTTP Methode
getLanguages	/factors/languages	GET
getUsingDecisions	/factors/:id/decisions	GET

**Tabelle 22: Zusätzliche Aktionen des Angular-Service Factors**

Die Aktion `getUsingDecisions` lädt die Fragetexte aller Entscheidungssituationen herunter, die den Entscheidungsfaktor mit der Id `:id` verwenden. Dies wird jeweils im Bearbeitungsscreen des entsprechenden Entscheidungsfaktors dargestellt.

#### 4.2.4 Controller

Gemäss der bei AngularJS-Applikationen üblichen Namenskonvention enden alle unsere Controller mit `Ctrl`.

##### 4.2.4.1 MenuTabsCtrl

Dieser Controller deklariert die vorhandenen Tabs des Menüs/der Seitenauswahl. Die Tabs sind in einem Array mit `link` und `label` definiert. Dadurch kann die Indexseite zur Anzeige über diesen Array iterieren.

Der selektierte Tab wird in `selectedIndex` gespeichert. Zum Wechsel der Seite wird mit Angulars `$watch` [18] auf dieser Variable "gehört". Bei einer Änderung des Wertes wird die Funktion `updateLocationPath` aufgerufen. Diese speichert den Pfad des neu gesetzten Tabs in `currentLink` und setzt den URL des Browsers dementsprechend. Damit die Tab-Gruppe den aktiven Tab entsprechend anzeigen kann, enthält der Controller die Methode `getTabClass`. Anhand dem aktuellen Link gibt sie `'active'` oder `'` zurück.

Bei der Initialisierung des Controllers wird anhand des aktuellen Browser-Pfads der `selectedIndex` gesetzt und der `$watch` [18] darauf gesetzt.

##### 4.2.4.2 QuestionnaireCtrl

Dieser Controller enthält die Funktionalität zur Anzeige des Motivprofil-Fragebogens. Die Standard-Sprache ist Englisch und die Standard-Sortierung aufsteigend nach `number`. Bei der Erstellung des Controllers werden die Variablen initialisiert, sowie `loadQuestionnaire` und `loadLanguages` ausgeführt.

Funktionen:

- `loadQuestionnaire`: Fragt die Fragen in der gewünschten Sprache vom Service ab (aktualisiert die Anzeige automatisch).

- `loadLanguages`: Fragt die vorhandenen Sprachen ab (aktualisiert das Drop-Down automatisch).
- `moveQuestionUp`: Setzt den Request ab, um die gewünschte Frage in der Nummerierung eine Position nach oben zu verschieben. Bei Erfolg wird `loadQuestionnaire` ausgeführt.
- `moveQuestionDown`: Wie `moveQuestionUp`, aber verschiebt eine Position nach unten.
- `showDeleteDialogue`: Zeigt den Dialog an, der den Benutzer auffordert, das Löschen zu bestätigen. Bei Bestätigung wird die zu löschende Frage intern gespeichert und `deleteQuestion` ausgeführt.
- `deleteQuestion`: Setzt den Request ab, die von `showDeleteDialogue` gespeicherte Frage, zu löschen. Bei Erfolg werden `loadQuestionnaire` und `loadLanguages` ausgeführt.
- `showMotiveDialogue`: Zeigt einen Info-Dialog an, der eine Legende der Motiv-Icons enthält.

#### 4.2.4.3 QuestionAddCtrl

Enthält die Funktionen zur Erstellung einer neuen Frage im Motivprofil-Fragebogen. Bei der Erstellung dieses Controllers werden die Variablen initialisiert und die bestehenden Sprachen vom Service abgefragt. Die Variable `disableLanguageSelect` wird auf `false` gesetzt, somit lässt sich die Sprache in der View ändern.

Lediglich die Funktion `addQuestion` existiert in diesem Controller. Hat die neue Frage ein gültiges Motiv und eine Sprache gesetzt, wird ein PUT-Request zur Erstellung der Frage abgesetzt. Nach Erhalt einer Antwort, wird wieder der gesamte Fragebogen angezeigt.

Jersey zeigt ein spezielles Verhalten, wenn die annotierte Klasse ein Feld mit `@XmlValue` enthält: Damit Jersey den Wert des Feldes `XmlValue` setzt, muss der Request-Body (das serialisierte JSON-Objekt) die Property `value` am Ende haben. D.h. Bevor der Request abgesetzt werden kann, muss die Eigenschaft `value` gelöscht und neu gesetzt werden, damit `value` am Ende des Strings steht.

Im Fall der Klasse `QuestionnaireEntry` ist das Feld `questionText` der `XmlValue` Wert. Würde `addQuestion` den beschriebenen Workaround nicht durchführen, wäre bei einer neuen Frage der Fragetext leer.

#### 4.2.4.4 QuestionEditCtrl

Enthält die Funktionen zur Bearbeitung einer bestehenden Frage des Motivprofil-Fragebogens. Bei der Erstellung werden die Variablen initialisiert und die aktuelle Frage anhand der ID vom Service abgefragt. Die Variable `disableLanguageSelect` wird auf `true` gesetzt, somit lässt sich die Sprache in der View nicht ändern und der Array mit den Sprachen enthält nur die Sprache dieser Frage.

Lediglich die Funktion `saveQuestion` existiert in diesem Controller. Hat die Frage ein gültiges Motiv und eine Sprache gesetzt, wird ein POST-Request zur Aktualisierung der Frage abgesetzt. Nach Erhalt einer Antwort, wird wieder der gesamte Fragebogen angezeigt.

#### 4.2.4.5 MotiveProfileSpecsCtrl

Dieser Controller enthält die Funktionalität zur Anzeige der Motivprofil Spezifikationen. Die Standard-Sprache ist Englisch. Bei der Erstellung des Controllers wird `loadLanguages` und `loadSpecifications` ausgeführt.

Funktionen:

- `loadLanguages`: Fragt die vorhandenen Sprachen ab, zu welcher Motivprofil Spezifikationen definiert wurden (aktualisiert das Drop-Down automatisch).

- `loadSpecifications`: Lädt die Motivprofil Spezifikationen für die gewählte Sprache vom Backend-Server via REST Schnittstelle herunter (aktualisiert die Anzeige automatisch).
- `showDeleteDialogue`: Zeigt den Dialog zur Bestätigung der Löschung der Motivprofil Spezifikationen für die ausgewählte Sprache an. Bei Bestätigung ruft es die Methode `deleteLanguage` auf.
- `deleteLanguage`: Löscht die Motivprofil Spezifikationen in der ausgewählten Sprache. Wenn der Request erfolgreich war, lädt die Methode die aktuell vorhandenen Sprachen erneut vom Server ab und aktualisiert die Anzeige.

#### 4.2.4.6 MotiveProfileSpecEditCtrl

Dieser Controller enthält die Funktionalität zur Bearbeitung einer Motivprofil Spezifikation für ein Motiv in einer bestimmten Sprache. Bei der Erstellung des Controllers werden die Variablen initialisiert und die Motivprofil Spezifikation des gewählten Motivs vom Service heruntergeladen und automatisch angezeigt.

Die Funktion `saveSpecification` setzt einen POST Request zum REST Service ab und schickt dabei die geänderten Werte als JSON Inhalt mit. Die Validation der eingegebenen Daten wird von der entsprechenden View gemacht. Dabei wird geprüft, dass alle Felder definiert wurden, sowie die Gültigkeit der Schwellwerte mittels Regex Pattern sichergestellt.

#### 4.2.4.7 MotiveProfileSpecsAddCtrl

Der Controller enthält die Funktionalität zur Erstellung aller 5 Motivprofil Spezifikationen in einer neuen Sprache. Bei der Erstellung des Controllers werden die 5 Spezifikationen initialisiert, d.h. jeweils ein neues Objekt angelegt und das Motiv darin gesetzt.

Die Funktion `addSpecifications` setzt einen PUT Request zum REST Service ab. Dabei setzt sie die gesetzte Sprache in die URL und die 5 Motivprofil Spezifikationen als JSON Array in den Request Body. Die Validation der eingegebenen Daten wird von der entsprechenden View gemacht.

#### 4.2.4.8 RatingTableCtrl

Dieser Controller enthält die Funktionalität zur Anzeige der Rating-Tabelle. Bei der Erstellung werden die Variablen initialisiert, sowie `loadEntries` ausgeführt.

Beim Löschen ist zu beachten, dass mit "Eintrag" die fünf Korrektur-Beträge eines bestimmten Faktor-Wertes gemeint sind.

Funktionen:

- `loadEntries`: Fragt die Einträge der Rating-Tabelle vom Service ab (aktualisiert die Anzeige automatisch).
- `showDeleteDialogue`: Zeigt den Dialog an, der den Benutzer auffordert, das Löschen eines Eintrages zu bestätigen. Bei Bestätigung wird der zu löschende Eintrag intern gespeichert und `deleteEntry` ausgeführt.
- `deleteEntry`: Setzt den Request ab, den von `showDeleteDialogue` gespeicherten Eintrag, zu löschen. Bei Erfolg wird `loadEntries` ausgeführt.

#### 4.2.4.9 RatingTableAddCtrl

Enthält die Funktionen zur Erstellung eines neuen Eintrags in der Rating-Tabelle. Bei der Erstellung des Controllers wird ein Eintrag mit den Korrektur-Beträgen 0 erstellt.

Der Controller enthält nur die Funktion `addEntry`. Wie in Abschnitt 4.2.3.3 beschrieben, wird mit Hilfe von `$http` [17] der Request von Hand gesendet. Nach Erhalt einer Antwort, wird wieder die gesamte Rating-Tabelle angezeigt.

#### 4.2.4.10 RatingTableEditCtrl

Enthält die Funktionen zur Bearbeitung eines bestehenden Eintrags der Rating-Tabelle. Bei der Erstellung wird der aktuelle Eintrag anhand des Faktor-Werts vom Service abgefragt.

Der Controller enthält nur die Funktion `saveEntry`. Wie in Abschnitt 4.2.3.3 beschrieben, wird mit Hilfe von `$http` [17] der Request von Hand gesendet. Nach Erhalt einer Antwort, wird wieder die gesamte Rating-Tabelle angezeigt.

#### 4.2.4.11 DecisionsCtrl

Dieser Controller enthält die Funktionalität zur Anzeige der vorgegebenen Entscheidungssituationen. Bei der Erstellung werden die Variablen initialisiert, sowie `loadDecisionsAndFactors` und `loadLanguages` ausgeführt.

Funktionen:

- `loadDecisions`: Fragt die vorgegebenen Entscheidungssituationen vom Service ab. Dies ist separat, damit beim Löschen einer Entscheidungssituation die vorgegebenen Faktoren nicht erneut abgefragt werden. Die Anzeige wird automatisch aktualisiert.
- `loadDecisionsAndFactors`: Führt `loadDecisions` aus und fragt die vorgegebenen Entscheidungssituationen vom Service ab. Die Anzeige wird automatisch aktualisiert.
- `loadLanguages`: Fragt die vorhandenen Sprachen ab (aktualisiert das Drop-Down automatisch).
- `showDeleteDialogue`: Zeigt den Dialog an, der den Benutzer auffordert, das Löschen einer Entscheidungssituation zu bestätigen. Bei Bestätigung wird die zu löschende Entscheidungssituation intern gespeichert und `deleteDecision` ausgeführt.
- `deleteDecision`: Setzt den Request ab, die von `showDeleteDialogue` gespeicherte Entscheidungssituation, zu löschen. Bei Erfolg werden `loadDecisions` und `loadLanguages` ausgeführt.

#### 4.2.4.12 DecisionCtrl

Dieser Controller enthält die Funktionen zur Erstellung einer neuen und zur Bearbeitung einer vorhandenen vorgegebenen Entscheidungssituation. Diese wurden zusammengeführt um duplizierten Code zu vermeiden.

Der Route-Provider (siehe Abschnitt 4.2.1.1) setzt zusätzlich eine Variable `type`, welche diesem Controller als `actionType` zur Verfügung gestellt wird. Anhand dieser initialisiert er die Variablen bei der Erstellung entsprechend.

Entspricht `actionType` dem Wert `'edit'` wird `disableLanguageSelect` auf `true` gesetzt und die vorgegebene Entscheidungssituation anhand der ID vom Service abgefragt. War dies erfolgreich, werden zusätzlich die vorgegebenen Faktoren in der Sprache der Entscheidungssituation abgefragt.

Beim `actionType` `'add'` (und jedem anderen Wert ausser `'edit'`) wird eine neue leere Entscheidungssituation erstellt. `disableLanguageSelect` wird auf `false` gesetzt und die vorhandenen Sprachen werden vom Service abgefragt.

Die Variable `disableLanguageSelect` definiert, ob sich die Sprache ändern lässt. Bei `false` lässt sie sich ändern und bei `true` nicht.

Funktionen:

- `loadFactors`: Fragt die vorgegebenen Faktoren mit der gesetzten Sprache der Entscheidungssituation vom Service ab. Bei Erfolg wird `setNamesFromIds` ausgeführt. Die Auflistung am Ende des Formulars wird automatisch aktualisiert.

- `setNamesFromIds`: Durchläuft die Faktoren der Entscheidungssituation. Zu jeder Faktor-ID wird der Name gesucht. Ist dieser gefunden wird die ID mit einem Objekt aus ID und Name ersetzt. Dies ist nötig, damit die Chips [7] im Formular bei den bereits hinzugefügten Faktoren den Namen anzeigen können.
- `addDecision`: Hat die Frage eine Sprache gesetzt, wird ein PUT-Request zur Erstellung der Entscheidungssituation abgesetzt. Bei Erfolg wird wieder die Liste der Entscheidungssituationen angezeigt.
- `saveDecision`: Hat die Frage eine Sprache gesetzt, wird der POST-Request zur Aktualisierung der Entscheidungssituation abgesetzt. Bei Erfolg wird wieder die Liste der Entscheidungssituationen angezeigt.

#### 4.2.4.13 FactorsCtrl

Dieser Controller enthält die Funktionalität zur Anzeige der vorgegebenen Entscheidungsfaktoren. Die Standard-Sprache dabei ist Englisch. Bei der Erstellung des Controllers wird `loadContent` ausgeführt.

Funktionen:

- `loadContent`: Fragt die vorhandenen Sprachen ab, zu welchen Entscheidungsfaktoren definiert wurden (aktualisiert das Drop-Down automatisch). Bei Erfolg wird `loadFactors` ausgeführt, ansonsten wird `error` zur Fehlerbehandlung ausgeführt.
- `loadFactors`: Lädt die Entscheidungsfaktoren für die gewählte Sprache vom Back-End-Server via REST Schnittstelle herunter (aktualisiert die Anzeige automatisch). Im Fehlerfall wird `error` ausgeführt.
- `showDeleteDialogue`: Zeigt den Dialog an, der den Benutzer auffordert, das Löschen eines Entscheidungsfaktors zu bestätigen. Wenn ja, wird der zu löschende Entscheidungsfaktor an die Methode `deleteFactor` übergeben und ausgeführt.
- `deleteFactor`: Setzt den Request ab, den von `showDeleteDialogue` übergebenen Entscheidungsfaktor zu löschen. Bei Erfolg wird `loadContent` ausgeführt. Wenn ein Fehler aufgetreten ist, wird `error` ausgeführt.
- `error`: Lokale Fehlerbehandlungsmethode. Dabei wird lediglich ein Flag gesetzt, welches in der View abgefragt werden kann, danach wird für eine einheitliche Fehlerbehandlung die `errorFunction` im Root-Scope aufgerufen.

#### 4.2.4.14 FactorAddCtrl

Dieser Controller enthält die Funktionen zur Erstellung eines neuen vorgegebenen Entscheidungsfaktors. Bei der Erstellung des Controllers wird ein neues `Factors` Objekt angelegt (siehe 4.2.3.5), bei dem die Sprache anhand des gesetzten Werts vom Route-Provider direkt initialisiert wird. Zudem werden die vorhandenen Sprachen über die Methode `getLanguages` abgefragt (siehe 4.2.3.5), sodass diese in der View entsprechend zur Auswahl gestellt werden können. Die Variablen `factorLoaded` und `disableLanguageSelect` werden entsprechend gesetzt, sodass die Edit- und Add-View dasselbe Formular wiederverwenden können.

Der Controller enthält nur die Funktion `addFactor`. Dabei wird geprüft, dass die Sprache eingegeben oder ausgewählt wurde. Wenn der Entscheidungsfaktor erfolgreich erstellt werden konnte, wird wieder die Übersicht der Entscheidungsfaktoren für die gewählte Sprache angezeigt. Andernfalls wird die `errorFunction` im Root-Scope für eine einheitliche Fehlerbehandlung ausgeführt.

#### 4.2.4.15 FactorEditCtrl

Der Controller enthält die Funktionalität zur Bearbeitung eines vorgegebenen Entscheidungsfaktors in einer bestimmten Sprache. Bei der Erstellung des Controllers werden die benötigten Variablen initialisiert und der Entscheidungsfaktor via Service (siehe 4.2.3.5) heruntergeladen und automatisch angezeigt. Konnte dieser erfolgreich heruntergeladen werden, so wird die Sprache daraus extrahiert und das Feld zur Spracheingabe mit der Variable `disableLanguageSelect` deaktiviert. Dies verhindert das nachträgliche Ändern der Sprache, was Probleme mit Referenzen der Entscheidungssituationen bewirken könnte.

Zudem werden die Fragetexte der Entscheidungssituationen heruntergeladen, die diesen Entscheidungsfaktor verwenden (siehe 4.2.3.5). Diese werden in der View entsprechend als Übersicht dargestellt.

Die Funktion `saveFactor` setzt einen POST Request zum REST Service ab und schickt dabei die geänderten Werte als JSON Inhalt mit. Die Validation der eingegebenen Daten wird von der entsprechenden View gemacht. Dabei wird geprüft, dass alle Felder (ausser die optionalen Explanation-Texte) definiert wurden. Bei Erfolg, wird wieder die Übersicht der Entscheidungsfaktoren in der aktuellen Sprache angezeigt. Andernfalls wird die `errorFunction` im Root-Scope für eine einheitliche Fehlerbehandlung ausgeführt.

#### 4.2.5 Fehlerbehandlung

Die Fehlerbehandlung wird in allen Controllern und Views gleich gemacht: Die View zeigt einen Ladebalken an, bis die benötigten Daten geladen wurden (`dataLoaded` Flag), oder ein Fehler beim Laden eingetreten ist (`errorOccurred` Flag). Der Inhalt der Seite wird erst angezeigt, wenn die Daten geladen wurden (`dataLoaded` Flag). Somit bleibt die Seite leer, wenn ein Fehler aufgetreten ist. Die Controller setzen die entsprechenden Flags und führen im Fehlerfall die in Abschnitt 4.2.2.1 beschriebene Fehlerbehandlungsfunktion aus. Zur Verdeutlichung wurde dies in Abschnitt 4.2.4.13 für die Entscheidungsfaktoren dokumentiert.

Die Controller zur Visualisierung der vorhandenen Inhalte einer Seite müssen typischerweise mehrere Anfragen zum RESTful Web Service absetzen, um die Daten korrekt darstellen zu können (z.B. die verfügbaren Sprachen, sowie die Daten in einer bestimmten Sprache abfragen). Damit in einem schwerwiegenderen Fehlerfall, der alle Datenabfragen fehlschlagen lässt (z.B. wenn der Web Service nicht verfügbar wäre) nicht gleich mehrere Fehlermeldungsdialoge angezeigt werden, haben wir uns dafür entschieden, die Anfragen nacheinander und nur bei Erfolg der vorangehenden Anfrage abzusetzen. Schlägt nun die Abfrage der verfügbaren Sprachen bereits fehl, so werden die Daten für eine bestimmte Sprache gar nicht erst abgefragt, sondern direkt die Fehlermeldung angezeigt.

Die Controller zum Hinzufügen eines Neuen Eintrags (`*AddCtrl`) müssen typischerweise keine Daten vorab laden, deshalb haben sie das Flag `dataLoaded` Standardmässig auf `True` und das Flag `errorOccurred` Standardmässig auf `False` gesetzt. Eine Ausnahme hierzu stellt der `DecisionCtrl` im Add-Fall dar, welcher alle verfügbaren Entscheidungsfaktoren für die gesetzte Sprache lädt, um diese für das entsprechende Feld vorschlagen zu können.

Wurden die benötigten Daten für die Seite einmal geladen, so müssen die weiteren Service Funktionen, z.B. Löschen oder Speichern eines Eintrages, sich nicht um das `errorOccurred` und `dataLoaded` Flag kümmern. Sie müssen lediglich noch im Fehlerfall die Fehlerbehandlungsfunktion aus Abschnitt 4.2.2.1 aufrufen, oder bei Erfolg der Operation auf eine gewünschte Seite umleiten (z.B. Aktualisieren der Übersichtsseite).



## 4.3 Android App

In diesem Abschnitt sind jeweils die Änderungen an den einzelnen Klassen zur Umsetzung beschrieben. Dieser Ansatz wurde gewählt, um besser und detaillierter aufzuzeigen, welche Teile in dieser Arbeit an der Vorarbeit geändert wurden.

### 4.3.1 Usability Fehler beheben

In der Vorarbeit [19] wurde ein Usability Fehler gefunden. Die Dialoge konnten mit Cancel-Events (ausserhalb des Dialogs klicken oder Back-Button) abgebrochen werden. Damit dies nicht mehr möglich ist, können der `DisclaimerDialogue`, der `NoInternetDialogue` und der `SyncFailedDialogue` nicht mehr abgebrochen werden (`cancelable = false`).

### 4.3.2 Splash Screen

Auf der Startseite der Android App haben wir einen Beschreibungstext von Decisio hinzugefügt. Dazu wurde in der `MainActivity` eine neue `TextView` hinzugefügt, welche den Beschreibungstext aus der neu definierten String-Ressource anzeigt. Diese `TextView` hat als `layout_weight` den Wert 1 gesetzt. Dies führt dazu, dass sie den noch verfügbaren restlichen Teil des Bildschirms einnimmt und somit den "Make decision" Knopf ans untere Ende des Bildschirms verschiebt.

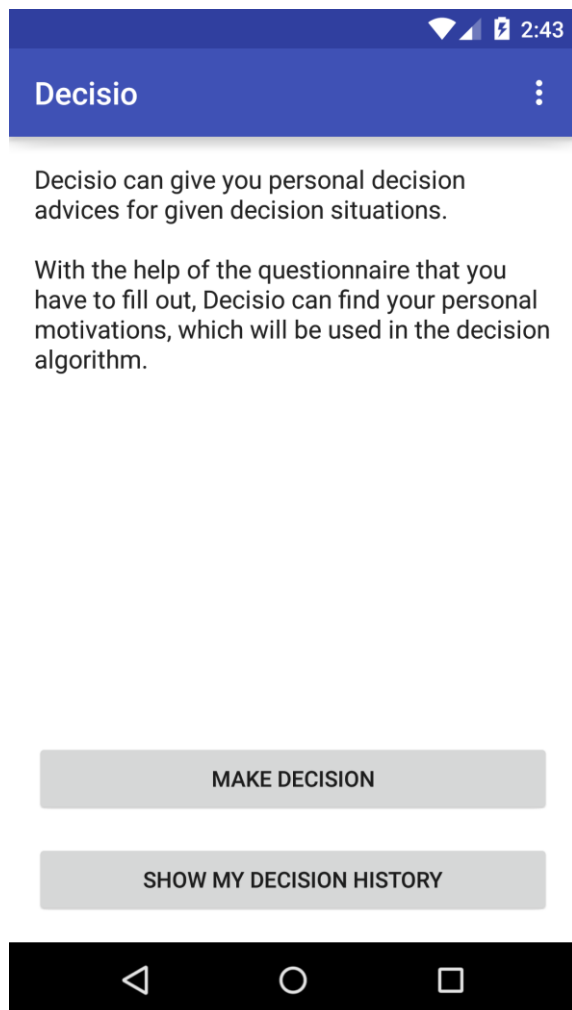


Abbildung 23: Screenshot der angepassten `MainActivity`



### 4.3.3 Bug: Entscheidungssituation ohne Entscheidungsfaktoren

Bei Tests mit den Praxispartnern wurde ein zusätzlicher Fehler in der Android App gefunden: Wenn man versuchte eine Entscheidungssituation durchzuführen, die keine Entscheidungsfaktoren definiert hatte, stürzte die App ab.

Dieses Problem haben wir in der Methode `getDecisionHeaders` in der Klasse `DBControllerDecisions` gelöst, indem wir in der Datenbankabfrage eine Bedingung eingefügt haben. Diese prüft, ob die Entscheidungssituation mindestens einen Mapping Eintrag in der Tabelle `GivenDecisionMapping` aufweist, welcher auf einen vorhandenen Entscheidungsfaktor in der Tabelle `GivenFactor` verweist.

Somit wird eine Entscheidungssituation gar nicht erst in der Android App angezeigt, wenn sie nicht mindestens einen vorhandenen Entscheidungsfaktor aufweist.

### 4.3.4 Umstellung der Ressourcen auf den REST-Service

Die Umstellung auf den RESTful Web-Service und die partielle Synchronisierung fand in zwei Schritten statt. Zuerst wurde als Referenz-Implementation die Aktualisierung des Motivprofil-Fragebogens migriert. Im nächsten Sprint wurden die restlichen Ressourcen in Angriff genommen. Details zur partiellen Synchronisierung entnehmen Sie bitte dem Abschnitt 4.1.3.

Während dem Parse-Vorgang werden die entsprechenden Domain-Objekte instanziiert und mit den geparsten Werten gefüllt. Damit die Domain-Objekte die neuen Informationen speichern können, wurden sie ausgebaut. Bei den Konstruktoren stellte sich die Frage, ob ein zweiter Konstruktor erstellt, oder der bestehende erweitert werden soll. Wir entschieden uns für die Variante Erweitern. Sonst würden die anderen Properties mit Standardwerten gefüllt werden, welche nicht korrekt wären. Besonders bei der Sprache wäre dies relevant.

Aus dieser Entscheidung resultierten die Anpassungen an den Abfrage-Methoden in den Datenbank-Controllern. Da jeweils lediglich ein bis zwei Spalten mehr abgefragt werden, empfinden wir dies nicht als grossen Mehraufwand für die Applikation.

#### 4.3.4.1 data.Preferences

Der Klasse `Preferences` wurden vier neue Konstanten für die Synchronisierung via RESTful Web-Service hinzugefügt:

Konstante	Beschreibung
<code>SYNCHRONIZATION_BASE_URL</code>	Basis-URL (Host) des Service
<code>SYNCHRONIZATION_LANGUAGE</code>	Zu verwendende Sprache für die Synchronisierung (wurde auf Englisch fixiert).
<code>SYNCHRONIZATION_CONNECT_TIMEOUT</code>	Connect-Timeout der Verbindung zum Service
<code>SYNCHRONIZATION_READ_TIMEOUT</code>	Read-Timeout der Verbindung zum Service

**Tabelle 23: Neue Konstanten in der Klasse `Preferences`**

#### 4.3.4.2 data.domain

Bei manchen Domain-Klassen wurden die Typen der Properties mit primitiven Typen durch die entsprechende Wrapper-Klasse ersetzt. Somit unterstützen die Properties auch `null` als Wert. Während des Parsens können Sie auf `null` gesetzt bleiben und während des Speicherns kann gegen `null` geprüft werden, anstelle einer eigens definierten Konstante. Dies betrifft die Klassen `AverageMotive`, `Decision`, `Factor`, `FactorExecution`, `QuestionnaireEntry` und `RatingTable.Entry`.

Aufgrund der partiellen Synchronisierung haben die erhaltenen XML-Daten mehr Inhalte. Um dies während des Parsens in den Domain-Objekten abzulegen, wurden diesen Properties hinzugefügt. Für diese neuen Properties wurden jeweils Getter-Methoden erstellt. In Tabelle 24 ist aufgelistet, welche Domain-Klasse welche neuen Properties erhielt:

Domain-Klasse	neue Properties
<code>AverageMotive</code>	<code>id</code> und <code>language</code>
<code>Decision</code>	<code>language</code> und <code>deleted</code>
<code>FactorExecution</code>	<code>language</code> und <code>deleted</code>
<code>QuestionnaireEntry</code>	<code>id</code> , <code>language</code> und <code>deleted</code>
<code>RatingTable.Entry</code>	<code>id</code> und <code>deleted</code>

**Tabelle 24: Den Domain-Klassen hinzugefügte Properties für die partielle Synchronisierung**

Diese Änderungen sind in den folgenden Abschnitten nicht mehr erläutert.

#### 4.3.4.3 `data.domain.AverageMotive`

Der Konstruktor wurde um zwei Parameter `id` und `language` erweitert. Er initialisiert nun ebenfalls die beiden neuen Properties. Da der `MotiveProfileController` als Fallback Standard-Motive ohne Spezial-Eigenschaften setzt, wurde ein zweiter Konstruktor erstellt. Dieser verlangt nur das Motiv und initialisiert alle Properties mit Standardwerten. Für Details siehe Abschnitt 4.3.4.26.

#### 4.3.4.4 `data.domain.Decision`

Der Konstruktor wurde überarbeitet. Er erwartet nun die Parameter `id`, `question`, `lastRun`, `language` und `deleted` und setzt diese Properties entsprechend. Der Parameter `never` fällt weg, da er durch `lastRun` abgelöst wurde.

Aufgrund des neuen Konstruktors wurden die Setter-Methoden `setId`, `setQuestion` und `setLastRun` obsolet und deshalb entfernt. Da `id` nun ein Objekt ist (Long), vergleicht `compareTo` sie mit `equals` anstelle von `==`. Diese Methode ist nötig für das Parsen. Die Entscheidungssituationen werden als Key in eine TreeMap gefüllt.

Da neu die ID im XML-Inhalt mitgesendet wird, hat jede geparste Entscheidungssituation bereits eine ID. Dadurch wurde die Konstante `NEW_DECISION` überflüssig und entfernt.

#### 4.3.4.5 `data.domain.FactorExecution`

Der Konstruktor, welcher auch das `mapping` erwartet, wurde erweitert. Er erwartet nun zusätzlich den Parameter `deleted` und setzt diese Property ebenfalls. Der Aufruf von `this` im anderen Konstruktor wurde entsprechend um das Argument `false` erweitert.

#### 4.3.4.6 `data.domain.QuestionnaireEntry`

Der Konstruktor wurde um die Parameter `id`, `language` und `deleted` erweitert, damit er alle Properties initialisiert.

#### 4.3.4.7 `data.domain.RatingTable.Entry`

Der Konstruktor wurde um die Parameter `id` und `deleted` erweitert, damit er alle Properties initialisiert.

#### 4.3.4.8 data.database.DBSchema

Den Klassen (Tabellen) `GivenDecision`, `GivenFactor`, `Questionnaire` und `AverageMotive` wurde je die Konstante für die zusätzliche Spalte `language` hinzugefügt. Die Konstante `SQL_CREATE_STRING` wurde in diesen Klassen jeweils ergänzt, dass auch die Spalte `language` in der Tabelle erstellt wird. Bei `AverageMotive.SQL_CREATE_STRING` wurde auch das Erstellen der Spalte `_ID` ergänzt, da auch hier vom Server nun eine ID zur Verfügung steht.

Durch die neue partielle Synchronisierung fällt das Löschen der Tabelle weg. Deshalb ist die Konstante `SQL_DELETE_TABLE`, die den Befehl zum Löschen der Datenbanktabelle enthält, nicht mehr nötig. Sie wurde bei den Klassen (Tabellen) `GivenDecision`, `GivenFactor`, `GivenDecisionMapping`, `Questionnaire` und `RatingTable` entfernt.

Durch die Verwendung einer Datenbank im Back-End Bereich werden bei der Synchronisierung der Daten jeweils die betroffenen IDs der Einträge zum Android Client mitgeschickt. Diese auf dem Server produzierte ID wird in allen Tabellen im Android Client direkt in der Spalte `_ID` als Identifikation verwendet, und nun nicht mehr einfach automatisch inkrementiert.

#### 4.3.4.9 data.database.DBController

Die private Property `never` wurde entfernt. Der XML-Parser für die vorgegebenen Entscheidungssituationen setzt den Wert für die letzte Durchführung der Entscheidungssituation (lokalisierter String "never"). Dadurch ist dieses Datenbankfeld immer gesetzt und es wird kein Standardwert mehr benötigt.

Die Methoden zur Synchronisierung wurden abgelöst durch Methoden mit der partiellen Synchronisierung. Dadurch ruft der `DBController` auch die entsprechende neue Methode im Untercontroller auf. Folgend ist eine Tabelle mit dem alten- und neuen Methodennamen und dem Abschnitt in dem die neue Methode erläutert ist.

alter Methodenname	neuer Methodenname	Abschnitt
<code>insertGivenDecisions</code>	<code>syncGivenDecisions</code>	4.3.4.11
<code>insertGivenFactors</code>	<code>syncGivenFactors</code>	4.3.4.12
<code>updateAverageMotives</code>	<code>syncAverageMotives</code>	4.3.4.13
<code>insertQuestionnaire</code>	<code>syncQuestionnaire</code>	4.3.4.14
<code>updateRatingTable</code>	<code>syncRatingTable</code>	4.3.4.15

**Tabelle 25:** Liste der neuen Synchronisierungsmethoden in `DBController`

#### 4.3.4.10 data.database.DBControllerDecisionsAndFactors

Diese Klasse erhielt mit den neuen Methoden eine zu hohe Komplexität. Deshalb wurde sie aufgeteilt in `DBControllerDecisions` und `DBControllerFactors`. Entsprechend wurden die Aufrufe in `DBController` angepasst.

Folgend ist eine Auflistung der neuen Klassen mit den verschobenen Methoden aus der Klasse `DBControllerDecisionsAndFactors`.

- `DBControllerDecisions` (Abschnitt 4.3.4.11)
  - `insertGivenDecisions`
  - `insertGivenDecision`
  - `insertGivenDecisionMapping`
  - `getGivenDecision`
  - `getFactorsToDecision`
  - `getDecisionHeaders`

- `updateLastRunFor`
- `truncateGivenDecisions`
- `DBControllerFactors` (Abschnitt 4.3.4.12)
  - `insertGivenFactors`
  - `insertGivenFactor`
  - `factorExists`
  - `truncateGivenFactors`

Für Details zu den Änderungen an den entsprechenden Methoden siehe entsprechende Abschnitte.

#### 4.3.4.11 `data.database.DBControllerDecisions`

Die Methoden `insertGivenDecisions`, `insertGivenDecision`, `insertGivenDecisionMapping` und `truncateGivenDecisions` wurden entfernt. Diese waren für die vorherige Synchronisierungs-Weise (ohne Partielle-Synchronisierung) konzipiert.

Um die neue partielle Synchronisierungs-Weise zu unterstützen, wurden folgende Methoden erstellt:

- `syncGivenDecisions`: Sie ist die Einstiegs-Methode, die von `DBController` aufgerufen wird. Sie durchläuft die erhaltenen vorgegebenen Entscheidungssituationen (`Decision`-Objekte) und ruft mit ihr entweder `deleteGivenDecision` oder `insertOrUpdateGivenDecision` und `updateGivenDecisionMapping` auf.
- `insertOrUpdateGivenDecision`: Trägt eine neue vorgegebene Entscheidungssituation ein oder aktualisiert eine bestehende. Zu Beginn ruft sie `checkAndGetColumnValues` auf. Gab der Aufruf `null` zurück, wird mit diesem Eintrag nicht fortgefahren. Sonst ruft sie `givenDecisionExists` mit der übergebenen `Decision` auf, um zu prüfen, ob bereits eine Situation mit dieser ID eingetragen ist. Wenn ja, wird sie aktualisiert, ansonsten wird eine neue eingetragen.
- `checkAndGetContentValues`: Sie überprüft, ob alle Properties gesetzt sind. D.h. ob die Entscheidungssituation korrekt geparkt wurde. Wenn nicht, gibt sie `null` zurück, um zu signalisieren, dass es keine gültige Situation ist. Ansonsten gibt sie die gefüllten `ContentValues` zurück. Diese können so direkt an die `insert` oder `update` Methode der `SQLiteDatabase`-Klasse übergeben werden.
- `givenDecisionExists`: Prüft, ob eine vorgegebene Entscheidungssituation mit der entsprechenden ID in der Datenbank vorhanden ist und gibt das Ergebnis als `boolean` zurück.
- `updateGivenDecisionMapping`: Sie löscht zu Beginn die Mappings mit der übergebenen Entscheidungssituations-ID. Danach fügt sie die IDs der Faktoren ein, die sich auf diese Entscheidungssituation auswirken (die zu ihr "gehören").
- `deleteGivenDecision`: Löscht die vorgegebene Entscheidungssituation und ihre Faktor Mappings mit der entsprechenden ID.

Die Methode `getGivenDecision` wurde aktualisiert. Sie verlangt den Parameter `never` nicht mehr, da dieser mit dem Wert aus dem Abfrage-Resultat überschrieben würde. Neu wird von der Datenbank auch `language` abgefragt. Beim Domain-Objekt `Decision` werden neu auch `language` und `deleted` gesetzt. Die Methode `getFactorsToDecision` fragt neu auch `language` ab. Sie setzt `language` und `deleted` beim Domain-Objekt `FactorExecution`. Für Erklärungen siehe Abschnitt 4.3.4.

#### 4.3.4.12 `data.database.DBControllerFactors`

Die Methoden `insertGivenFactors`, `insertGivenFactor`, `factorExists` und `truncateGivenFactors` wurden entfernt. Diese waren für die vorherige Synchronisierungs-Weise (ohne Partielle-Synchronisierung) konzipiert.

Um die neue partielle Synchronisierungs-Weise zu unterstützen, wurden folgende Methoden erstellt:

- `syncGivenFactors`: Sie ist die Einstiegs-Methode, die von `DBController` aufgerufen wird. Sie durchläuft die erhaltenen vorgegebenen Entscheidungsfaktoren (`FactorExecution`-Objekte) und ruft mit dem Faktor entweder `deleteGivenFactor` oder `insertOrUpdateGivenFactor` auf.
- `insertOrUpdateGivenFactor`: Trägt einen neuen vorgegebenen Entscheidungsfaktor ein oder aktualisiert einen bestehenden. Zu Beginn ruft sie `checkAndGetColumnValues` auf. Gab der Aufruf `null` zurück, wird mit diesem Faktor nicht fortgefahren. Sonst ruft sie `givenFactorExists` mit dem übergebenen `FactorExecution` auf, um zu prüfen, ob bereits ein Faktor mit dieser ID eingetragen ist. Wenn ja, wird er aktualisiert, ansonsten wird ein neuer eingetragen.
- `checkAndGetContentValues`: Sie überprüft, ob alle Properties gesetzt sind. D.h. ob der vorgegebene Entscheidungsfaktor korrekt geparkt wurde. Wenn nicht, gibt sie `null` zurück, um zu signalisieren, dass es kein gültiger Faktor ist. Ansonsten gibt sie die gefüllten `ContentValues` zurück. Diese können so direkt an die `insert` oder `update` Methode der `SQLiteDatabase`-Klasse übergeben werden.
- `givenFactorExists`: Prüft, ob ein vorgegebener Entscheidungsfaktor mit der entsprechenden ID in der Datenbank vorhanden ist und gibt das Ergebnis als `boolean` zurück.
- `deleteGivenFactor`: Löscht den vorgegebenen Entscheidungsfaktor mit der entsprechenden ID.

#### 4.3.4.13 data.database.DBControllerProfiles

Die Methoden `updateAverageMotives` und `updateAverageMotive` wurden entfernt. Diese waren für die vorherige Synchronisierungs-Weise (ohne Partielle-Synchronisierung) konzipiert.

Um die neue partielle Synchronisierungs-Weise zu unterstützen, wurden folgende Methoden erstellt:

- `syncAverageMotive`: Sie ist die Einstiegs-Methode, die von `DBController` aufgerufen wird. Sie durchläuft die erhaltenen Motivprofil-Spezifikationen (`AverageMotive`-Objekte) und ruft mit der Spezifikation `insertOrUpdateAverageMotive` auf.
- `insertOrUpdateAverageMotive`: Trägt eine neue Motivprofil-Spezifikation ein oder aktualisiert eine bestehende. Zu Beginn ruft sie `checkAndGetColumnValues` auf. Gab der Aufruf `null` zurück, wird mit dieser Spezifikation nicht fortgefahren. Sonst ruft sie `entryExists` mit dem übergebenen `AverageMotive` auf, um zu prüfen, ob bereits eine Spezifikation mit dieser ID eingetragen ist. Wenn ja, wird sie aktualisiert, ansonsten wird eine neue eingetragen.
- `checkAndGetContentValues`: Sie überprüft, ob alle Properties gesetzt sind. D.h. ob die Motivprofil-Spezifikation korrekt geparkt wurde. Wenn nicht, gibt sie `null` zurück, um zu signalisieren, dass es keine gültige Spezifikation ist. Ansonsten gibt sie die gefüllten `ContentValues` zurück. Diese können so direkt an die `insert` oder `update` Methode der `SQLiteDatabase`-Klasse übergeben werden.
- `entryExists`: Prüft, ob eine Motivprofil-Spezifikation mit der entsprechenden ID in der Datenbank vorhanden ist und gibt das Ergebnis als `boolean` zurück.

Ein Löschen ist nicht nötig. Ein Eintrag kann nie als gelöscht markiert sein, da das Back-End nur das Löschen einer ganzen Sprache erlaubt (siehe Abschnitt 4.1.2.2). Bei der Synchronisierung werden Einträge einer anderen Sprache, die gelöscht werden müssen, nicht gesendet. Diese werden durch `deleteEntriesWithLanguageOtherThan` gelöscht. Siehe auch Abschnitt 4.3.5.1.

Die Methode `getAverageMotives` wurde aktualisiert. Von der Datenbank werden nun auch `id` und `language` abgefragt und beim Domain-Objekt `AverageMotive` gesetzt. Für Erklärungen siehe Abschnitt 4.3.4.

#### 4.3.4.14 `data.database.DBControllerQuestionnaire`

Die Methoden `insertQuestionnaire`, `insertQuestion`, `questionExists(SQLiteDatabase, QuestionnaireEntry)` und `truncateQuestionnaire` wurden entfernt. Diese waren für die vorherige Synchronisierungsweise (ohne Partielle-Synchronisierung) konzipiert.

Um die neue partielle Synchronisierungsweise zu unterstützen, wurden folgende Methoden erstellt:

- `syncQuestionnaire`: Sie ist die Einstiegs-Methode, die von `DBController` aufgerufen wird. Sie durchläuft die erhaltenen Fragen (`QuestionnaireEntry`-Objekte) und ruft mit der Frage entweder `deleteQuestion` oder `insertOrUpdateQuestion` auf.
- `insertOrUpdateQuestion`: Trägt eine neue Frage ein oder aktualisiert eine bestehende. Zu Beginn ruft sie `checkAndGetColumnValues` auf. Gab der Aufruf `null` zurück, wird mit dieser Frage nicht fortgefahren. Sonst ruft sie `questionExists` mit dem übergebenen `QuestionnaireEntry` auf, um zu prüfen, ob bereits eine Frage mit dieser ID eingetragen ist. Wenn ja, wird sie aktualisiert, ansonsten wird eine neue eingetragen.
- `checkAndGetContentValues`: Sie überprüft, ob alle Properties gesetzt sind. D.h. ob die Frage korrekt geparkt wurde. Wenn nicht, gibt sie `null` zurück, um zu signalisieren, dass es keine gültige Frage ist. Ansonsten gibt sie die gefüllten `ContentValues` zurück. Diese können so direkt an die `insert` oder `update` Methode der `SQLiteDatabase`-Klasse übergeben werden.
- `questionExists(SQLiteDatabase, Long)`: Prüft, ob eine Frage mit der entsprechenden ID in der Datenbank vorhanden ist und gibt das Ergebnis als `boolean` zurück.
- `deleteQuestion`: Löscht die Frage mit der entsprechenden ID.

Die Methode `getQuestionnaire` wurde aktualisiert. Von der Datenbank wird nun auch `id` abgefragt. Beim Domain-Objekt `QuestionnaireEntry` werden auch `id` und `deleted` gesetzt. Für Erklärungen siehe Abschnitt 4.3.4.

#### 4.3.4.15 `data.database.DBControllerRatingTable`

Die Methoden `updateRatingTable`, `insertRatingTableEntry` und `truncateRatingTable` wurden entfernt. Diese waren für die vorherige Synchronisierungsweise (ohne Partielle-Synchronisierung) konzipiert.

Um die neue partielle Synchronisierungsweise zu unterstützen, wurden folgende Methoden erstellt:

- `syncRatingTable`: Sie ist die Einstiegs-Methode, die von `DBController` aufgerufen wird. Sie durchläuft die erhaltenen Rating-Tabellen-Einträge (`RatingTable.Entry`-Objekte) und ruft mit dem Eintrag entweder `deleteEntry` oder `insertOrUpdateEntry` auf.
- `insertOrUpdateEntry`: Trägt einen neuen Rating-Tabellen-Eintrag ein oder aktualisiert einen bestehenden. Zu Beginn ruft sie `checkAndGetColumnValues` auf. Gab der Aufruf `null` zurück, wird mit diesem Eintrag nicht fortgefahren. Sonst ruft sie `entryExists` mit dem übergebenen `RatingTable.Entry` auf, um zu prüfen, ob bereits ein Eintrag mit dieser ID eingetragen ist. Wenn ja, wird er aktualisiert, ansonsten wird ein neuer eingetragen.



- `checkAndGetContentValues`: Sie überprüft, ob alle Properties gesetzt sind. D.h. ob der Rating-Tabellen-Eintrag korrekt geparkt wurde. Wenn nicht, gibt sie `null` zurück, um zu signalisieren, dass es kein gültiger Eintrag ist. Ansonsten gibt sie die gefüllten `ContentValues` zurück. Diese können so direkt an die `insert` oder `update` Methode der `SQLiteDatabase`-Klasse übergeben werden.
- `entryExists`: Prüft, ob ein Rating-Tabellen-Eintrag mit der entsprechenden ID in der Datenbank vorhanden ist und gibt das Ergebnis als `boolean` zurück.
- `deleteEntry`: Löscht den Rating-Tabellen-Eintrag mit der entsprechenden ID.

Die Methode `getRatingTable` wurde aktualisiert. Von der Datenbank wird nun auch `id` abgefragt. Beim Domain-Objekt `RatingTable.Entry` werden auch `id` und `deleted` gesetzt. Für Erklärungen siehe Abschnitt 4.3.4.

#### 4.3.4.16 application.network

Durch den RESTful Web-Service sind keine SFTP-Verbindungen mehr nötig. Deshalb wurde die Methode `checkInternetConnectivity` in die eigene Klasse `ConnectionHelper` verlagert und die Klasse `Sftp` gelöscht. Dadurch ist die ehemals verwendete Bibliothek JSch (Java Secure Channel) ebenfalls nicht mehr nötig. Das gesamte `libs` Verzeichnis wurde gelöscht. Es enthielt die JAR-Datei der Bibliothek und eine Textdatei mit dem Lizenztext.

Dadurch fällt die Abhängigkeit von JSch weg. Aus der Datei `app/build.gradle` wurde unter `dependencies` die Zeile `compile files('libs/jsch-0.1.51.jar')` entfernt.

#### 4.3.4.17 application.network.DownloadSyncInitiator

Es wurden Überarbeitungen vorgenommen, um die Ressourcen via RESTful Web-Service anzufordern.

Die Konstanten mit den Dateinamen wurden durch Konstanten mit Pfadangaben ersetzt. Tabelle 26 zeigt auf, welche Konstante durch welche abgelöst wurden.

alte Konstante (Dateiname)	neue Konstante (Pfad)
<code>DECISIONS_FILE</code>	<code>DECISIONS_PATH</code>
<code>FACTORS_FILE</code>	<code>FACTORS_PATH</code>
<code>RATING_TABLE_FILE</code>	<code>RATING_TABLE_PATH</code>
<code>QUESTIONNAIRE_FILE</code>	<code>QUESTIONNAIRE_PATH</code>
<code>MOTIVE_PROFILE_SPECIFICATIONS_FILE</code>	<code>MOTIVE_PROFILE_SPECS_PATH</code>

**Tabelle 26: DownloadSyncInitiator: Ablösung der Konstanten**

Folgend ist eine Auflistung an den Änderungen der Sub-Klasse `SyncTask` beschrieben:

- Die Property `parsers` speichert nun die XML-Parser, die einen `RestCallTask` zur Folge haben und ihm übergeben werden.
- Die Methode `initiateParsers` initiiert und speichert nun die XML-Parser für die REST-Calls. Als Key wird neu der Pfad und nicht mehr der Dateiname in der Map gespeichert.
- Die Methode `downloadAllFiles` wurde nach `prepareAndInvokeAllTasks` umbenannt. Sie durchläuft die `parsers`. Für jeden Parser in `parsers` wird ein `RestCallTask` gespeichert. Am Ende werden alle gestartet.
- In `run` wird anstelle von `downloadAllFiles` neu `prepareAndInvokeAllTasks` aufgerufen.

- In der Methode `SyncTask.initiateParsers` wurde die Initialisierung der Parser angepasst. Folgend ist aufgelistet, welcher Parser welche Konstante vorher als Key hatte und welche Konstante der neue Key ist:

Parser	alter Key	neuer Key
<code>DecisionsXmlParser</code>	<code>DECISIONS_FILE</code>	<code>DECISIONS_PATH</code>
<code>FactorsXmlParser</code>	<code>FACOTRS_FILE</code>	<code>FACOTRS_PATH</code>
<code>RatingTableXmlParser</code>	<code>RATING_TABLE_FILE</code>	<code>RATING_TABLE_PATH</code>
<code>QuestionnaireXmlParser</code>	<code>QUESTIONNAIRE_FILE</code>	<code>QUESTIONNAIRE_PATH</code>
<code>MotiveProfileSpecificationsXmlParser</code>	<code>MOTIVE_PROFILE_SPECIFICATIONS_FILE</code>	<code>MOTIVE_PROFILE_SPECS_PATH</code>

**Tabelle 27: DownloadSyncInitiator: Neue Initialisierung der XML-Parser**

Die Sub-Klasse `RestCallTask` wurde erstellt. Es wird eine Verbindung zur URL der entsprechenden Ressource des Service erstellt. Nachdem die Header und Timeouts gesetzt sind, wird verbunden. Danach erhält der Parser den `InputStream` der Verbindung, wie ehemals beim `FileDownloadTask`. Trat ein Fehler auf, gibt der Task `DownloadTaskResult.ERROR` zurück, sonst `DownloadTask.SUCCESS`. Dieser Task verwendet `DownloadTask.NOT_UPDATED` nicht mehr. Bei diesem Task wird partiell synchronisiert, womit das Löschen und Wieder-Eintragen (Truncate) sowieso gespart wird.

Die Sub-Klasse `FileDownloadTask` wurde entfernt, weil sie nicht mehr verwendet wird. Beim enum `DownloadTaskResult` wurde der Wert `NOT_UPDATED` entfernt, ebenfalls weil er durch die partielle Synchronisierung obsolet wurde.

#### 4.3.4.18 application.xml.XmlSpecification

Neu haben die erhaltenen XML-Daten mehr Inhalte. Dies hatte zur Folge, dass die Spezifikationen erweitert werden mussten. In Tabelle 28 ist aufgelistet, welche Unterklasse von `XmlSpecification` welche neuen Konstanten (Attribute) erhielt:

innere Klasse	neue Konstante (Attribute)
<code>Decisions</code>	<code>ATTRIBUTE_ID, ATTRIBUTE_LANGUAGE, ATTRIBUTE_DELETED</code>
<code>Factors</code>	<code>ATTRIBUTE_LANGUAGE, ATTRIBUTE_DELETED</code>
<code>MotiveForm</code>	<code>ATTRIBUTE_ID, ATTRIBUTE_LANGUAGE, ATTRIBUTE_DELETED</code>
<code>MotiveProfileSpecification</code>	<code>ATTRIBUTE_ID, ATTRIBUTE_LANGUAGE</code>
<code>RatingTable</code>	<code>ATTRIBUTE_ID, ATTRIBUTE_DELETED</code>

**Tabelle 28: Den Subklassen von XmlSpecification hinzugefügte Konstanten**

#### 4.3.4.19 application.xml.XmlParser

Die Methode `skipMyEndTag` wurde entfernt. Sie wurde ursprünglich von den Parsern verwendet, um unnötige Aufrufe von `...EndTag` zu verhindern. Da das Backend jeweils den End-Tag weglässt, falls der Tag keinen Inhalt hat, kann diese Methode nicht mehr verwendet werden. Mit dem Löschen dieser Methode wurde auch die Property `currentTag` überflüssig und deshalb gelöscht.

Die Annotation `@UsedViaReflection` wurde hinzugefügt. Sie kann bei den Parser-Methoden gesetzt werden. Sie soll verhindern, dass Parser-Methoden gelöscht werden, da diese als nicht verwendet angezeigt werden, aber trotzdem nötig sind, weil sie via Reflection aufgerufen werden.



#### 4.3.4.20 application.xml.\*XmlParser

Gewisse Änderungen wurden an allen fünf spezifischen XML-Parsern durchgeführt.

Dies sind:

- Die Parse-Methode `skipMyEndTag` wird nicht mehr aufgerufen. Dies würde zu Fehlern führen, wenn es ein leeres Element ist (`<myTag />`).
- Ein Eintrag in der Liste der geparsten Elemente wird nur erstellt, wenn eine ID gesetzt ist. Ob er gültig ist, wird im Datenbank-Controller geprüft.
- Die Methode `forcePersistence` gibt nicht mehr `false` zurück, falls die Liste mit den geparsten Elementen leer ist. Neu bedeutet eine leere Liste nicht mehr, dass die Synchronisierung der Datei fehlerhaft war. Z.B. wenn es keine Aktualisierungen seit der letzten Synchronisierung gibt.
- Allen Parse-Methoden, die via Reflection aufgerufen werden, wurde die Annotation `@UsedViaReflection` angefügt, siehe Beschreibung in Abschnitt 4.3.4.19.

Dies betrifft konkret die Klassen `DecisionsXmlParser`, `FactorsXmlParser`, `MotiveProfileSpecificationsXmlParser`, `QuestionnaireXmlParser` und `RatingTableXmlParser`.

#### 4.3.4.21 application.xml.DecisionsXmlParser

Die Parse-Methode `decisionTagStart` wurde überarbeitet. Neu werden auch die Attribute `id`, `language` und `deleted` gelesen. Für jeden gelesenen Wert gibt es einen Standard-Wert, falls er nicht gelesen wurde/werden konnte. Es ist möglich, dass der Tag ein leeres Element ist. Deshalb wird das Element neu in `decisionTagStart` zur Liste der geparsten Element hinzugefügt und nicht mehr in `decisionTagEnd`. Deshalb wurde `decisionTagEnd` überflüssig und entfernt. Die Methode `predefinedFactorTagStart` springt im Dokument nicht mehr weiter (ruft kein `parser.next` mehr auf). Neu wird in `forcePersistence` die Methode `syncGivenDecisions` für die partielle Synchronisierung aufgerufen.

Damit der Parser mit weniger Aufwand (ohne DBController) getestet werden kann, wurde die Sichtbarkeit der Property `decisions` auf package-local gesetzt.

#### 4.3.4.22 application.xml.FactorsXmlParser

Die Parse-Methode `factorTagStart` wurde überarbeitet. Neu werden auch die Attribute `id`, `language` und `deleted` gelesen. Für jeden gelesenen Wert gibt es einen Standard-Wert, falls er nicht gelesen wurde/werden konnte. Es ist möglich, dass der Tag ein leeres Element ist. Deshalb wird das Element neu in `factorTagStart` zur Liste der geparsten Element hinzugefügt und nicht mehr in `factorTagEnd`. Neu wird in `forcePersistence` die neue Methode `syncGivenFactors` für die partielle Synchronisierung aufgerufen.

#### 4.3.4.23 application.xml.MotiveProfileSpecificationsXmlParser

Die Parse-Methode `averageTagStart` wurde überarbeitet. Neu werden auch die Attribute `id` und `language` gelesen. Für jeden gelesenen Wert gibt es einen Standard-Wert, falls er nicht gelesen wurde/werden konnte. Aufgrund der neuen Attribute wurden zwei neue Properties `id` und `language` nötig. Diese werden in `resetCurrentMotiveValues` ebenfalls auf `null` gesetzt. Neu wird in `forcePersistence` die neue Methode `syncAverageMotives` für die partielle Synchronisierung aufgerufen.

#### 4.3.4.24 application.xml.QuestionnaireXmlParser

Die Parse-Methode `questionTagStart` wurde überarbeitet. Neu werden auch die Attribute `id`, `language` und `deleted` gelesen. Für jeden gelesenen Wert gibt es einen Standard-Wert, falls er nicht gelesen wurde/werden konnte. Neu wird in `forcePersistence` die neue Methode `syncQuestionnaire` für die partielle Synchronisierung aufgerufen.

#### 4.3.4.25 application.xml.RatingTableXmlParser

Die Parse-Methode `entryTagStart` wurde überarbeitet. Neu werden auch die Attribute `id` und `deleted` gelesen. Für jeden gelesenen Wert gibt es einen Standard-Wert, falls er nicht gelesen wurde/werden konnte. Neu wird in `forcePersistence` die neue Methode `syncRatingTable` für die partielle Synchronisierung aufgerufen.

#### 4.3.4.26 application.MotiveProfileController

Der `MotiveProfileController` enthält die Methode `getDefaultAverageMotives`. Sie setzt Standard Motivprofil Spezifikationen, falls die Abfrage keine gültigen Daten liefert. Für diese Profile sind keine Erläuterungen nötig, sondern lediglich die Schwellwerte und das Motiv, damit die Anzeige korrekt funktioniert. Da `data.domain.AverageMotive` neue Properties erhielt und somit der Konstruktor noch mehr Argumenten erhielt, wurde, wie in Abschnitt 4.3.4.3 beschrieben, ein zweiter Konstruktor erstellt. Dieser verlangt nur das Motiv und setzt Standard-Schwellwerte. Die Methode `getDefaultAverageMotives` verwendet nun diesen neuen Konstruktor.

#### 4.3.4.27 AboutActivity

Da die Bibliothek JSch entfernt wurde, wurde das Layout der `AboutActivity` angepasst. In `activity_about.xml` wurde die TextView mit dem Lizenztext-Titel und die ScrollView mit dem Lizenztext selbst (inkl. Unterelementen) gelöscht.

Die nicht mehr verwendeten Texte `license_title` und `license_text` wurden aus `res/values/string.xml` gelöscht.

### 4.3.5 Anpassen der Synchronisierungssprache und der Mehrsprachigkeit

Während Tests mit den Praxispartnern haben wir das Thema "Synchronisierung" noch einmal vertieft angeschaut und besprochen. Dabei kam zum Vorschein, dass auch wenn die dynamischen Inhalte der App übersetzt werden können, die statischen Texte in der App momentan aber immer in Englisch bleiben. Darum einigten wir uns darauf, die Android App vorerst nur in Englisch anzubieten. Für weitere Sprachen werden dann separate App Versionen erstellt.

Die Synchronisierungssprache wurde daher in der Android App auf Englisch fixiert.

#### 4.3.5.1 data.database.DBController

Dieser Klasse wurde eine statische Methode `deleteEntriesWithOtherLanguageThan` hinzugefügt. Diese Methode löscht alle Einträge aus der angegebenen Datenbank Tabelle (als Argument übergeben), welche nicht die durch die Synchronisierung erhaltene Sprache enthält. Die erhaltene Sprache wird ebenfalls als Argument übergeben.

Diese Methode wurde im `DBController` erfasst, um duplizierten Code in den Klassen `DBControllerDecisions`, `DBControllerFactors`, `DBControllerProfiles` und `DBControllerQuestionnaire` zu vermeiden. Die jeweiligen synchronisierungs-Methoden dieser Controller rufen nun die Methode `deleteEntriesWithOtherLanguageThan` mit den für sie passenden Argumenten auf.

Der Grund für diese Methode liegt darin, dass nicht mehrere Sprachen in der lokalen Datenbank existieren, wenn die Android App bei der Synchronisierung der Daten eine andere Sprache erhält, als die Einträge die sie bereits lokal in der Datenbank hat. Obwohl die Android App momentan nur in Englisch verfügbar ist, haben wir diese Funktion beibehalten, um den Fallback Mechanismus des RESTful Web Service für die Zukunft (wenn weitere Sprachversionen der App erstellt werden) korrekt zu unterstützen. Dieser Mechanismus wurde in Abschnitt 4.1.3 beschrieben.

Für die Klasse `DBControllerDecisions` und `DBControllerFactors` werden jeweils nur die Faktoren bzw. die Entscheidungssituationen, welche eine andere Sprache aufweisen, gelöscht. Die Mappings der Faktoren zu den Entscheidungssituationen bleiben aber noch in der Datenbank bestehen. Da wir in der lokalen Datenbank der Android Geräte nicht mit Fremdschlüsseln arbeiten und dieser Sprachfallback sowieso nie eintritt, wenn man die Sprachdaten im Back-End im Griff hat, haben wir uns dafür entschieden, diese ungültigen Mapping Einträge trotzdem in der Datenbank zu belassen. Wir sehen hier keinen Handlungsbedarf, weil sich diese Mapping Einträge sowieso auf keine Entscheidungssituation auswirken können, da die Identifikation der Entscheidungssituation bzw. des Faktors nicht mehr existiert, siehe Tabelle 29 als Erklärung.

Entscheidungssituation	Mapping Eintrag	Entscheidungsfaktor	Resultat
Eintrag mit der ID <b>23</b> wurde gelöscht.	< <b>23</b> , <b>11</b> >	Eintrag mit der ID <b>11</b> ist noch vorhanden.	Da die Entscheidungssituation gelöscht wurde, kann diese gar nicht erst zur Auswahl angezeigt werden.  Das Mapping hat keine Auswirkung mehr.
Eintrag mit der ID <b>25</b> ist noch vorhanden.	< <b>25</b> , <b>16</b> >	Eintrag mit der ID <b>16</b> wurde gelöscht.	Der Entscheidungsfaktor ist nicht mehr vorhanden und wird daher in der SELECT Abfrage mit der INNER JOIN Bedingung nicht ermittelt werden.  Das Mapping hat keine Auswirkung mehr.
Eintrag mit der ID <b>42</b> wurde gelöscht.	< <b>42</b> , <b>34</b> >	Eintrag mit der ID <b>34</b> wurde gelöscht.	Da die Entscheidungssituation gelöscht wurde, kann diese gar nicht erst zur Auswahl angezeigt werden. Der Faktor muss und kann daher auch nicht in der SELECT Abfrage mit der INNER JOIN Bedingung ermittelt werden.  Das Mapping hat keine Auswirkung mehr.

**Tabelle 29: Mapping zu gelöschter Entscheidungssituation oder Entscheidungsfaktor**

### 4.3.6 Upload der Benutzer-Daten an den REST-Service

Die vom Benutzer im Android Client erfassten Daten wurden in der Vorarbeit auf den verwendeten SFTP-Server als XML-Dateien hochgeladen. Dies sind die soziodemografischen Daten, sowie die Antworten des Benutzers im Motivprofil-Fragebogen. Diese Daten werden jetzt neu auf den erstellten RESTful Web Service hochgeladen. Dazu wurden die beiden Klassen `UploadQuestionnaireAnswersTask` und `UploadSociodemographicDataTask` überarbeitet: neu wird die von Java angebotene Klasse `URLConnection` [20] verwendet, wobei die verwendete URL beim Aufruf der Methode `openConnection()` eine Instanz der Klasse `HttpsURLConnection` erstellt. Auf dieser Instanz wird noch der Content-Type auf XML, sowie die HTTP Request-Methode auf PUT gesetzt, um den RESTful Web Service korrekt anzusprechen.

Das gemeinsame Verhalten (Abfrage und aktualisieren der Shared-Preference, sowie die `upload`-Methode und die dafür benötigten Sub-Methoden) der beiden Klassen wurde in Form eines Refactorings in eine gemeinsame Elternklasse `UploadTask` extrahiert. Diese definiert eine

abstrakte Methode `getXml`, welche in der `upload`-Methode verwendet wird und von den Sub-Klassen implementiert werden muss.

Somit ist der einzig nötige Code in den Klassen `UploadQuestionnaireAnswersTask` und `UploadSociodemographicDataTask` die Implementation der `getXml`-Methode und der Aufruf des Konstruktors der Elternklasse.

### 4.3.7 Gewählte Option in "Personal Information" anzeigen

In der `SociodemographicDataActivity` (Ansicht Personal Information) soll die gewählte Option angezeigt werden. D.h. sobald eine Option gesetzt wird, soll nicht mehr die Information "Please enter your..." sondern direkt die gewählte Option erscheinen.

#### 4.3.7.1 `data.Preferences`

Damit die Namen der Optionen, welche in "Personal Information" bearbeitet werden, zentral gesammelt sind, wurde die Property `sociodemographicDataPreferences` erstellt. Damit sie ausserhalb verwendet werden kann, wurde für sie eine Getter-Methode hinzugefügt.

Die Properties `numberValues` (sie speichert die Namen der Optionen, die einen Zahlenwert speichern: letzte Synchronisierung und Geburtsjahr des Benutzers) und `sociodemographicDataPreferences` erfüllen beide ähnliche Zwecke. Beide stellen einen Container zum Speichern bestimmter Property-Namen zur Verfügung. Der Datentyp von `sociodemographicDataPreferences` wurde auf `List<String>` gesetzt. Es wird jeweils die Methode `contains` verwendet. Daher kam Array als Datentyp nicht in Frage. Jener von `numberValues` wurde auf `List<String>` gewechselt, damit sie einheitlich sind (war vorher ein Array). Auch spricht nichts dafür sie weiterhin als Array zu speichern (ausser eventuell die minimale Speicherverbrauchsersparnis).

Beide Properties werden in der Methode `initialiseVariables` mit den entsprechenden Werten bestückt. Sie wurde entsprechend angepasst und erweitert.

#### 4.3.7.2 `view.SociodemographicDataFragment`

Um die gewünschte Funktionalität zu erbringen, musste die Klasse `SociodemographicDataFragment` angepasst werden. Sie implementiert nun das Interface `OnSharedPreferenceChangeListener` von Android. Es enthält eine Methode, die als Callback dient, welche aufgerufen wird, wenn eine `SharedPreferences` ändert.

Aufgrund dessen wurde dem Fragment die Methode `onSharedPreferenceChanged` hinzugefügt. Sie prüft, ob die geänderte Einstellung eine des Screens "Personal Information" ist, wenn ja, wird die Anzeige aktualisiert. Diese Überprüfung findet mit Hilfe der neuen Getter-Methode `getSociodemographicDataPreferences` von Preference statt.

Dieser neue `OnSharedPreferenceChangeListener` muss nun beim System registriert werden, damit der Callback auch stattfindet. Deshalb implementiert das Fragment neu auch `onResume` und `onPause`. In `onResume` wird das Fragment als Listener registriert und in `onPause` wieder abgemeldet.

### 4.3.8 Fixe Positionierung der Knöpfe bei der Faktor-Bewertung

Führt man eine Entscheidungssituation durch, befinden sich die Knöpfe für die Bewertung der verschiedenen Faktoren auf unterschiedlicher Höhe. Grund dafür sind die unterschiedlich langen Beschreibungen der Faktoren. Je nach Länge benötigt sie eine oder zwei Zeilen.

Nach mehreren Versuchen mit unterschiedlichen Layout-Optionen gelang die Umsetzung nicht. Problematisch sind Geräte mit geringer Auflösung. Bei ihnen passen die Knöpfe nicht mehr auf einmal auf die Anzeige, deshalb befinden sich die Inhalte in einer `ScrollView`. Es gibt die Layout-

Eigenschaft `match_parent`, welche das entsprechende Element vergrößert, bis die Grösse des Überelements erreicht ist. Bei einem Unterelement einer `ScrollView` hat diese bei der Höhe jedoch keine Auswirkung, da die Höhe einer `ScrollView` nicht erreicht werden kann. Setzt man die Layout-Eigenschaft `fillViewport` der `ScrollView`, wird nur noch die Beschreibung ohne die Knöpfe angezeigt.

Um trotzdem eine fixe Positionierung zu erreichen, wurde eine minimale Höhe der Faktor-Beschreibung gesetzt. Dazu wurde in `res/values/dimens.xml` ein Wert `factor_rating_description_min_height` hinzugefügt. Im Layout `res/layout/activity_make_decision.xml` wurde `android:minHeight` der Faktor-Beschreibung auf die erstellte Dimension gesetzt. Falls eine Beschreibung mehr Platz als zwei Zeilen benötigt, wird trotzdem der gesamte Text angezeigt. Dann sind die Knöpfe jedoch wieder verschoben.

### 4.3.9 Im Motivprofil-Fragebogen unbeantwortete Fragen markieren

Füllt man den Motivprofil-Fragebogen aus und will die Antworten speichern, dann wird geprüft, ob alle Fragen beantwortet wurden. Hat man nicht alle Fragen beantwortet erscheint eine Meldung mit dem entsprechenden Hinweis. Neu sollen zusätzlich die Fragen, die nicht beantwortet sind, mit einem roten Rahmen markiert werden.

Dazu wurde die Datei `res/drawable/red_border.xml` erstellt. Sie definiert ein rotes Rechteck mit weisser Füllung. Somit ist dies der eigentliche Rahmen.

Der `QuestionListViewAdapter` wurde erweitert. Um zu speichern, ob die Rahmen gezeichnet werden sollen, wurde die Property `drawBorders` (Boolean) hinzugefügt. Zum Zeichnen des Rahmens wurde die Methode `drawBorder` erstellt. Sie verlangt die neu erstellte View und die Nummer der Frage. Ist das Zeichnen aktiviert und die Frage nicht beantwortet, fügt sie der View das erstellte Drawable als Hintergrund hinzu. Diese Methode wird neu zusätzlich in `getView` aufgerufen. Somit wird beim Erstellen einer View falls nötig ein Rahmen gezeichnet. Damit die `QuestionnaireActivity` das Zeichnen der Rahmen aktivieren kann, wurde die Methode `enableDrawingBorders` hinzugefügt. Sie setzt die Property `drawBorders` auf `true` und zeichnet die Views neu.

In der `QuestionnaireActivity` wurde die Methode `verifyAnswers` aktualisiert. Sind nicht alle Fragen beantwortet, ruft sie neu die neue Methode `enableDrawingBorders` auf.

### 4.3.10 Bezug zu Panter AG löschen

In der Datei `res/values/strings.xml` wurde `copyright` von "Copyright © 2014 Forventis GmbH & Panter AG" nach "Copyright © 2015 Forventis GmbH" geändert.

### 4.3.11 Upgrade auf API Level 21

In der Vorarbeit wurde die Mindest-Version auf Android 4.0 (Level 15) festgelegt. Zu dieser Zeit war Android 5.0 und das Material Design noch in der Entwicklungsphase. Die Praxispartner wünschten sich eine App in Material Design. Da Android 5.0 (Level 21) nun veröffentlicht ist, wurde die Mindest-Version nochmals diskutiert. Obwohl Android 5.0 erst mässig verbreitet ist [21], einigten wir uns auf diese Version.

Um den API-Level zu erhöhen, wurde die Datei `app/build.gradle` angepasst. Bei den Direktiven `compileSdkVersion`, `minSdkVersion` und `targetSdkVersion` wurde der Wert von 15 auf 21 gewechselt.

## 4.3.12 Upgrade auf Material Theme

Damit das Anheben der Mindest-Version einen Nutzen hat, wurde das generelle Theme zum Material Theme gewechselt. Dies hatte auch zur Folge, dass alle Layouts überarbeitet werden mussten. In den folgenden Unterkapiteln sind die Änderungen an den einzelnen Dateien beschrieben.

Sofern keine Begründung vermerkt ist, wurden die Änderungen vorgenommen, um die Richtlinien des Material Designs [22] einzuhalten. Für die Style-Definition der Texte sind neu die Styles des Material Theme des Android Systems gesetzt. Damit die Texte besser lesbar sind, ist jeweils nicht der komplette Wert `@android:style/TextAppearance.Material.<Name>` vermerkt, sondern nur `<Name>`.

### 4.3.12.1 view.helper.DialogueHelper

Beim Erstellen eines Dialogs wurde in der Klasse `DialogueHelper` die Farbe des Titels und der Linie zwischen Titel und Inhalt jeweils gesetzt. Wird als Basis-Theme das Material Theme verwendet, führen die entsprechenden Methoden-Aufrufe zu Exceptions. Deshalb wurde die Methode `setDialogueTitleColor` gelöscht. Entsprechend wurde der Aufruf aus `showSyncDialogue`, `showDisclaimerDialogue` und `showErrorDialogue` entfernt.

### 4.3.12.2 res/layout/activity\_about.xml

Bei allen drei TextViews wurde der Style `Subhead` gesetzt.

### 4.3.12.3 res/layout/activity\_choose\_decision.xml

Als Root-Layout wird neu das `LinearLayout` verwendet. Dies wird in jeder Activity verwendet. Die Paddings wurden entfernt, somit werden die Trennlinien bis an den Rand gezeichnet.

### 4.3.12.4 res/layout/activity\_main.xml

Die TextView mit dem Einleitungstext hat den Style `Subhead`.

### 4.3.12.5 res/layout/activity\_make\_decision.xml

Android zeigt einen "Overscroll" Effekt an, wenn man in der ScrollView das Ende der Liste erreicht. Damit dieser bündig mit der blauen Titelleiste ist, wurde das Attribut `layout_marginTop` der ScrollView entfernt. Dies befindet sich nun in der TextView des Faktor-Namens (Abbildung 25, Nr. 1), damit dieser trotzdem genügend Abstand hat.

Das Attribut `style` der TextView mit dem Faktor-Namen wurde auf `Headline` gesetzt (Abbildung 25, Nr. 1). Da nicht mehr eine eigene Style-Definition verwendet wird, wurde das Attribut `textColor` ergänzt. Der `layout_marginBottom` wurde den Richtlinien angepasst.

Die TextView mit der Faktor-Beschreibung hat nun den Style `Subhead` (Abbildung 25, Nr. 2).

Bei allen vier Knöpfen wurde die Farbe in `background` durch Drawables ersetzt. Somit zeigen auch die farbigen Knöpfe beim Klicken den Welleneffekt an. Folgende Drawables wurden für die Knöpfe erstellt und entsprechend als `background` gesetzt: `res/drawable/button_gray.xml`, `res/drawable/button_green.xml`, `res/drawable/button_orange.xml` und `res/drawable/button_red.xml`.

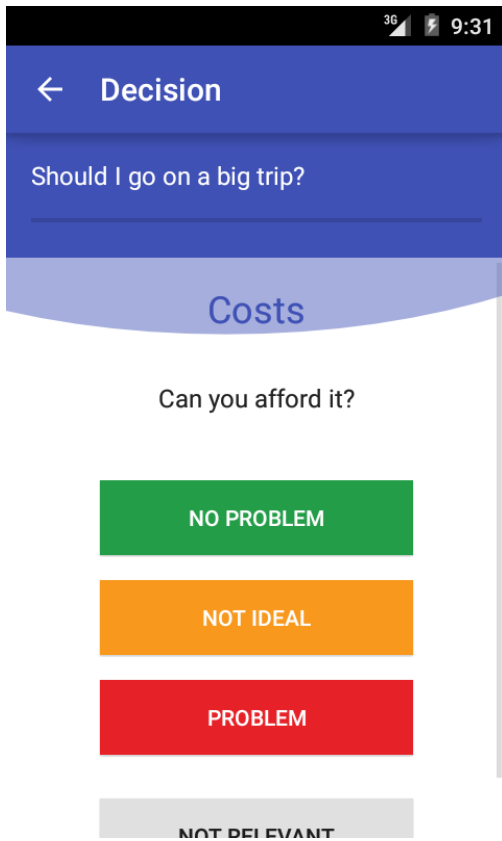


Abbildung 24: `MakeDecisionActivity`: "Overscroll"-Effekt

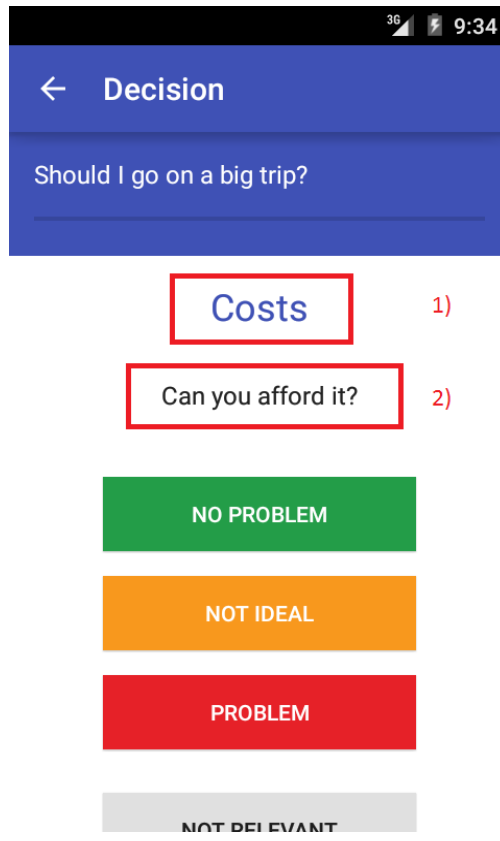


Abbildung 25: `MakeDecisionActivity`: Layout-Elemente

#### 4.3.12.6 `res/layout/activity_motive_profile.xml`

Als Root-Layout wird neu das `LinearLayout` verwendet. Dies wird in jeder Activity verwendet.

#### 4.3.12.7 `res/layout/activity_questionnaire.xml`

Der `TextView` mit dem Infotext wurde der Style `Body1` hinzugefügt.



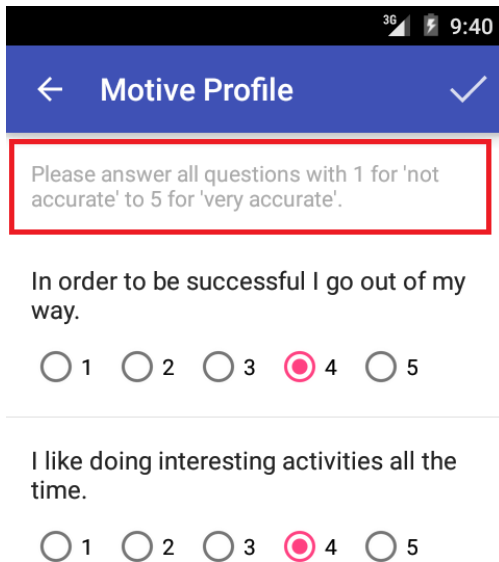


Abbildung 26: `QuestionnaireActivity`: Infotext (rot umrahmt)

#### 4.3.12.8 `res/layout/decision_execution_title.xml`

Das Attribut `style` der `TextView` für den Fragetext der Entscheidungssituation wurde auf `Subhead` gesetzt. Da nicht mehr eine eigene Style-Definition verwendet wird, wurde das Attribut `textColor` ergänzt. Der `layout_marginBottom` wurde den Material Design Richtlinien angepasst.

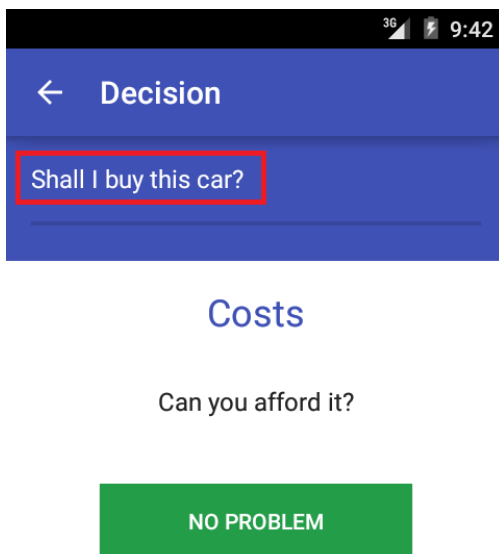


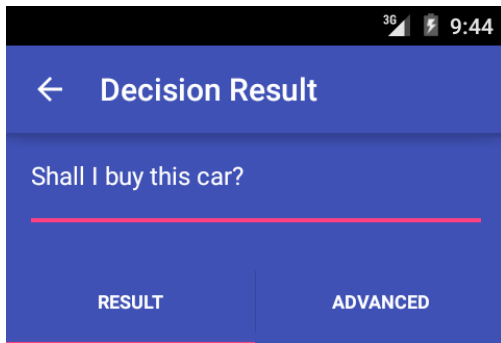
Abbildung 27: `MakeDecisionActivity`: Fragetext (rot umrahmt)

#### 4.3.12.9 `res/layout/decision_result_tab_result.xml`

Das Attribut `style` der `TextView` mit der Handlungsempfehlung wurde auf `Headline` gesetzt (Abbildung 28, Nr. 1). Der `layout_marginBottom` wurde den Material Design Richtlinien angepasst.

Der `TextView` mit der Handlungsempfehlung der App wurde der Style `Subhead` gesetzt (Abbildung 28, Nr. 2).





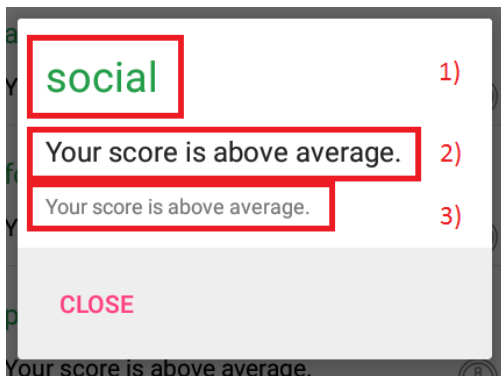
**Abbildung 28: DecisionResultActivity: Layout-Elemente**

#### 4.3.12.10 res/layout/dialogue\_motive\_profile\_detail.xml

Die TextView mit dem Motiv-Namen erhielt den Style `Headline` (Abbildung 29, Nr. 1). Der `layout_marginBottom` wurde den Material Design Richtlinien angepasst.

Der TextView mit der durchschnittsmässigen Einstufung wurde der Style `Subhead` hinzugefügt (Abbildung 29, Nr. 2). Ebenfalls wurde `layout_marginBottom` aktualisiert.

Der Style `Caption` wurde der TextView mit der Beschreibung der Einstufung hinzugefügt (Abbildung 29, Nr. 3).



**Abbildung 29: Dialog Motivprofil-Details: Layout-Elemente**

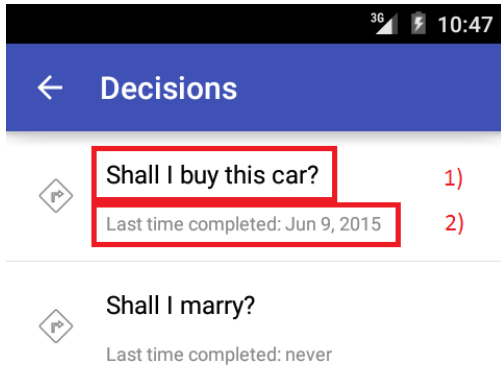
#### 4.3.12.11 res/layout/listview\_choose\_decision.xml

Damit der Welleneffekt auch bei der Liste der vorhandenen Entscheidungssituationen angezeigt wird, wurde dem LinearLayout als `background` das Drawable `list_selector` gesetzt. Siehe dazu auch Abschnitt 4.3.12.19. Zusätzlich wurden Paddings hinzugefügt, weil diese nicht mehr im Activity-Layout enthalten sind.

Der ImageButton wurde durch eine ImageView ersetzt um den Welleneffekt zu ermöglichen. Zusätzlich wurde `layout_marginRight` durch `layout_marginEnd` ersetzt. Dies wird von Android Lint empfohlen, weil somit auch RTL-Sprachen (rechts nach links) unterstützt würden.

Der Style der TextView mit dem Namen der Entscheidungssituation wurde auf `Subhead` gewechselt (Abbildung 30, Nr. 1). Da nicht mehr eine eigene Style-Definition verwendet wird, wurde das Attribut `textColor` ergänzt. Ebenfalls wurde `layout_marginBottom` angepasst.

Die TextView mit der letzten Durchführung der Entscheidungssituation hat neu den Style `Caption` (Abbildung 30, Nr. 2). Da nicht mehr eine eigene Style-Definition verwendet wird, wurde das Attribut `textColor` ergänzt.



**Abbildung 30: ChooseDecisionActivity: Layout-Elemente**

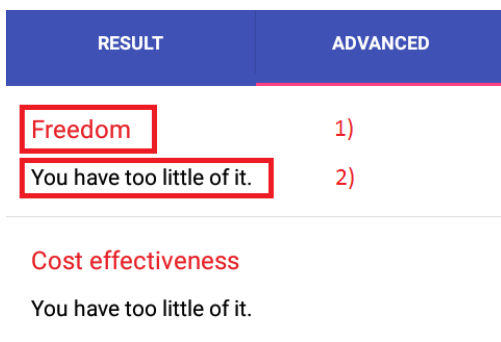
#### 4.3.12.12 res/layout/listview\_decision\_result.xml

Die TextView mit dem Faktor-Namen erhielt den Style `Body1`.

#### 4.3.12.13 res/layout/listview\_decision\_result\_advanced.xml

Der Style der TextView mit dem Namen des Faktors wurde auf `Subhead` gewechselt (Abbildung 31, Nr. 1). Der `layout_marginBottom` wurde angepasst.

Die TextView mit der Erklärung hat neu den Style `Body1` (Abbildung 31, Nr. 2). Da nicht mehr eine eigene Style-Definition verwendet wird, wurde das Attribut `textColor` ergänzt.



**Prestige/status**  
You might be short with it.

**Abbildung 31: DecisionResultActivity Tab Advanced: Layout-Elemente**

#### 4.3.12.14 res/layout/listview\_motive\_profile.xml

Damit der Welleneffekt auch bei der Liste des Motivprofils angezeigt wird, wurde dem LinearLayout als `background` das Drawable `list_selector` gesetzt. Siehe dazu auch Abschnitt 4.3.12.19.

Der Style der TextView mit dem Namen des Motivs wurde auf `Subhead` gewechselt (Abbildung 32, Nr. 1). Der `layout_marginBottom` wurde angepasst.

Die TextView mit der Erklärung erhielt den Style `Body1` (Abbildung 32, Nr. 2).

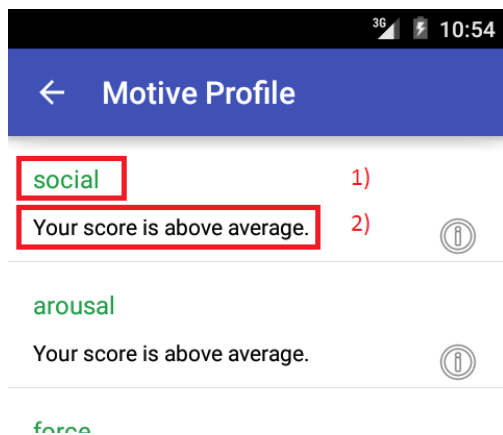


Abbildung 32: `MotiveProfileActivity`: Layout-Elemente

#### 4.3.12.15 `res/layout/listview_question.xml`

Die TextView mit dem Faktor-Namen erhielt den Style `Subhead` und `layout_marginBottom` wurde angepasst.

Bei den RadioButtons wurde `layout_marginRight` durch `layout_marginEnd` ersetzt. Somit würden auch RTL-Sprachen (rechts nach links) unterstützt werden.

#### 4.3.12.16 `values/colours.xml`

Anhand der Material Design Spezifikation [22] wurden die Farben `blue` und `pink` angepasst, sowie die Farben `blue_dark` und `gray_secondary` erstellt. Für den Welleneffekt der farbigen Buttons wurde `default_ripple` hinzugefügt.

#### 4.3.12.17 `values/dimens.xml`

Um die Abstände zwischen Texten gemäss der Material Design Spezifikation [22] zu realisieren, wurden folgende Dimensionen ergänzt: `margin_between_headline_and_subhead`, `margin_between_subhead_and_body`, `margin_between_subhead_and_caption` und `margin_between_subhead_and_caption`. Die Dimension `activity_vertical_margin_small` wird aufgrund der neu erstellten nicht mehr verwendet und wurde deshalb gelöscht.

#### 4.3.12.18 `values/styles.xml`

Die wichtigste Änderung ist der Wechsel zum Material Theme. Dazu wurde der `parent` von `AppTheme` auf `android:Theme.Material.Light.DarkActionBar` gesetzt. Dieser Wechsel hatte zur Folge, dass alle Definitionen ausser `windowBackground` des Styles nicht mehr nötig sind und gelöscht wurden. Somit konnten auch die restlichen Style-Definitionen gelöscht werden.

Damit das Theme den Designwünschen der Praxispartner entspricht, wurden die Style-Farben und das Theme für die Dialoge erstellt. Dadurch ist die Akzent-Farbe pink. Damit die Dialoge diese auch verwenden, wurde der Style `AppTheme.AlertDialogueTheme` erstellt.

Das Material Theme hat standardmässig Punkte zur Kennzeichnung des Menüs. Somit wurden die selbst-erstellten Drawables `res/drawable/menu_dots.xml` und `res/drawable/menu_dots.xml` überflüssig und gelöscht. Diese wurden in der Vorarbeit erstellt, um das Layout an das Material Theme anzulehnen.

#### 4.3.12.19 drawable/list\_selector.xml

Verwendet eine App das Material Theme, zeigt Android standardmässig einen Welleneffekt an, wenn ein Listenelement ausgewählt wird. Bei gewissen Listen funktionierte dies nicht von Anfang an. Deshalb wurde das Drawable `list_selector` erstellt. Es kann als Hintergrund eines Listenelements gesetzt werden, damit dieses den Effekt anzeigt.

Die Fragen im Motivprofil-Fragebogen sind auch Listenelemente. Da diese aber nicht ausgewählt werden (sondern die RadioButtons), zeigen sie den Effekt bewusst nicht an. Zusätzlich würde die Ansicht unnötig unübersichtlich.

#### 4.3.12.20 drawable/red\_border.xml

Mit dem Material Theme als Grundlage für die Anzeigen, sind die ausgewählten RadioButtons jeweils pink. Wird beim Ausfüllen des Motivprofil-Fragebogens eine Frage nicht beantwortet, wird diese umrahmt. Damit nicht zu viele Farben verwendet werden, wurde die Farbe dieses Rahmens von rot auf pink gewechselt.

### 4.3.13 Entscheidungs-Historie

Eine neue Funktionalität in der Android App ist die Entscheidungs-Historie des Benutzers. Dabei wird gespeichert welche Entscheidungssituation wann ausgeführt wurde, und wie die Handlungsempfehlung der App lautete.

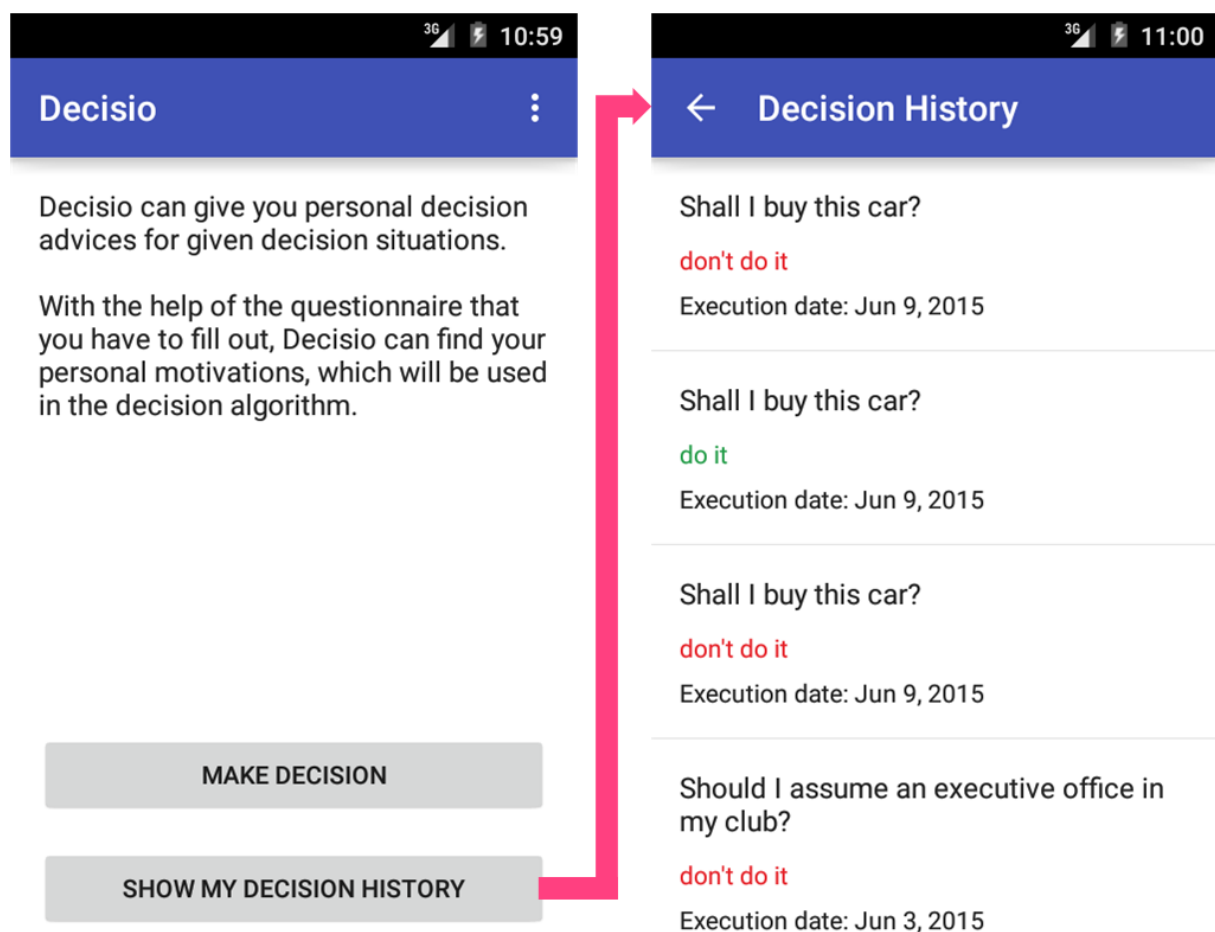


Abbildung 33: `MainActivity` und `DecisionHistoryActivity`

In den folgenden Unterabschnitten sind die dafür notwendigen Code-Erweiterungen beschrieben.

#### 4.3.13.1 view.MainActivity

In der Startseite der Applikation wurde ein neuer Knopf eingefügt, welcher beim Klick darauf, die neu erstellte Ansicht `DecisionHistoryActivity` öffnet, siehe Abschnitt 4.3.13.2.

#### 4.3.13.2 view.DecisionHistoryActivity

Diese Klasse ist neu hinzugekommen und stellt die Anzeige der Entscheidungs-Historie dar. Wird diese Ansicht angezeigt, lädt die `DecisionHistoryActivity` die einzelnen Einträge der Entscheidungs-Historie mit der Funktion `getDecisionHistory` der Klasse `DecisionController`. Anschliessend erstellt sie eine neue Instanz der Klasse `DecisionHistoryListViewAdapter` und übergibt dabei die geladenen Einträge im Konstruktor. Dieser Adapter wird anschliessend auf der entsprechenden ListView gesetzt, was bewirkt, dass die geladenen Einträge mit dem entsprechenden Layout angezeigt werden.

#### 4.3.13.3 view.DecisionResultActivity

Diese Ansicht wird verwendet, um die berechnete Handlungsempfehlung nach der Durchführung einer Entscheidungssituation anzuzeigen. Sie stammt aus unserer Studienarbeit und wurde nur geringfügig geändert:

- Neu wird zur Anzeige der Handlungsempfehlung die Methode `setResultViewsFor(ActionRecommendation, TextView, TextView)` der Klasse `DecisionResultHelper` aufgerufen, siehe Abschnitt 4.3.13.5.
- Neu wird nach der Berechnung der Handlungsempfehlung die Methode `addDecisionToHistory` der Klasse `DecisionController` aufgerufen, siehe Abschnitt 4.3.13.6.

#### 4.3.13.4 view.helper.DecisionHistoryListViewAdapter

Diese Klasse wird, wie in Abschnitt 4.3.13.2 beschrieben, dazu verwendet die geladenen Einträge der Entscheidungs-Historie in einer ListView zu visualisieren. Sie erweitert die Klasse `ArrayAdapter`, die von Android für solche Aufgaben angeboten wird.

Der Konstruktor dieser Klasse übergibt die geladenen Einträge an den Konstruktor der Klasse `ArrayAdapter`. Dadurch wird die weitere Programmierung vereinfacht, indem z.B. die Methode `getCount` nicht überschrieben werden muss. Die Anzahl der zu visualisierenden Elemente in der ListView wird direkt aus der Anzahl der übergebenen Einträge der Entscheidungs-Historie ermittelt. Die Klasse `DecisionHistoryListViewAdapter` muss daher nur noch die Methode `getView` überschreiben und dabei die entsprechenden Text-Werte des jeweils anzuzeigenden Eintrags der Entscheidungs-Historie auf den dafür vorgesehenen TextViews setzen. Für die Anzeige der berechneten Handlungsempfehlung wird die Methode `setResultViewFor(ActionRecommendation, TextView)` der Klasse `DecisionResultHelper` verwendet (siehe Abschnitt 4.3.13.5).

#### 4.3.13.5 view.helper.DecisionResultHelper

In der Ansicht `DecisionResultActivity` und neu auch in der Ansicht `DecisionHistoryActivity` wird Code zur Visualisierung der berechneten Handlungsempfehlung benötigt. Um diesen Code gemeinsam nutzen zu können, wurde er in die Klasse `DecisionResultHelper` verschoben und ist über die folgenden Methoden verfügbar:

- `setResultViewFor(ActionRecommendation, TextView)`: Setzt den Text für die übergebene Handlungsempfehlung (Argument `ActionRecommendation`) mit der entsprechenden Farbe (Grün, Orange oder Rot) in der übergebenen TextView und liefert als Rückgabewert den Beschreibungstext zur Handlungsempfehlung zurück. Die Rückgabe wurde

gemacht, um die switch-case Abfrage der Handlungsempfehlung für die Methode `setResultViewsFor(ActionRecommendation, TextView, TextView)` wiederverwenden zu können.

- `setResultViewsFor(ActionRecommendation, TextView, TextView)`: Ruft die Funktion `setResultViewFor(ActionRecommendation, TextView)` auf und setzt zusätzlich den Beschreibungstext zur Handlungsempfehlung (Rückgabewert der aufgerufenen Methode) in der zweiten übergebenen TextView. Diese Methode wird von der `DecisionResultActivity` aufgerufen.

#### 4.3.13.6 application.DecisionController

Diesem Controller wurde die Methode `addDecisionToHistory` hinzugefügt, um eine durchgeführte Entscheidungssituation zur Entscheidungs-Historie hinzuzufügen. Zudem wurde die Methode `getDecisionHistory` hinzugefügt, um alle Einträge der Entscheidungs-Historie abzufragen. Beide Methoden delegieren ihre Aufgabe an den `database` Layer, siehe Abschnitt 4.3.13.7.

Zudem wurde die Enumeration `ActionRecommendation` aus dieser Klasse entfernt, siehe Abschnitt 4.3.13.10.

#### 4.3.13.7 data.database.DBControllerDecisions

Diesem Datenbank-Controller wurde die Methode `addToDecisionHistory` hinzugefügt, um eine durchgeführte Entscheidungssituation für die Entscheidungs-Historie in der Datenbank zu speichern. Ebenfalls wurde die Methode `getDecisionHistory` hinzugefügt, um alle Einträge der Entscheidungs-Historie aus der Datenbank zu lesen.

Die Methode `addToDecisionHistory` speichert den Durchführungszeitpunkt der Entscheidungssituation in Anzahl Millisekunden seit dem 1. Januar 1970, dem Start der Unixzeit. Dies erlaubt eine einfache Sortierung der Einträge in der Datenbank nach deren Durchführungszeitpunkt. Die Handlungsempfehlung wird über die Methode `name()` der Enumeration `ActionRecommendation` in der Datenbank gespeichert, was eine einfache Rückwandlung in den Enumerationswert mit der Methode `valueOf(String)` ermöglicht.

Die Methode `getDecisionHistory` liefert eine Instanz der Klasse `DecisionHistoryEntry` pro Eintrag in der Entscheidungs-Historie. Die Einträge werden absteigend nach ihrem Durchführungszeitpunkt sortiert, sodass es eine wirkliche Historie darstellt.

Beide Methoden wurden über die Klasse `DBController` zugänglich gemacht, welche ein Facade Objekt für den Datenbank Layer darstellt und die Operationen an den entsprechenden Sub-Controller delegiert.

#### 4.3.13.8 data.database.DBSchema

Dieser Klasse wurde eine weitere Sub-Klasse (Tabelle) hinzugefügt: `DecisionHistory`. Dabei werden die benötigten Spalten definiert und in der `onCreate` Methode die Tabelle mit dem zusammengestellten `SQL_CREATE_TABLE`-Statement erstellt.

#### 4.3.13.9 data.domain.DecisionHistoryEntry

Diese Klasse stellt ein neues Domain-Objekt dar. Sie wird benötigt, um einen Eintrag in der Entscheidungs-Historie abzubilden. Sie enthält die folgenden Eigenschaften, welche durch den Konstruktor initialisiert werden:

- `question`: Der Fragetext der durchgeführten Entscheidungssituation (z.B. "Shall I do more sport?").

- `executionDate`: Das Durchführungsdatum der Entscheidungssituation, direkt als Text zur Visualisierung formatiert.
- `actionRecommendation`: Die berechnete Handlungsempfehlung der App, dargestellt als ein Wert der Enumeration `ActionRecommendation`.

#### 4.3.13.10 data.domain.ActionRecommendation

Diese Enumeration stellt die drei möglichen Handlungsempfehlungen der App dar: `PROCEED`, `WARNING` und `DO_NOT_PROCEED`. Sie wurde aus der Klasse `DecisionController` entfernt und in den `domain` Layer verschoben. Damit kann auch die Klasse `DBControllerDecisions` darauf zugreifen (für die Entscheidungs-Historie), ohne zirkuläre Abhängigkeiten zwischen den Schichten zu erzeugen.

### 4.3.14 Refactoring

In dieser Arbeit haben wir einige zusätzliche Code Verbesserungen gegenüber der Studienarbeit vorgenommen, welche nicht in die Umsetzung einzelner User Stories fallen. Diese Änderungen sind in den folgenden Unterkapiteln beschrieben.

#### 4.3.14.1 application.DecisionController

An dieser Klasse haben wir eine Verbesserung bezüglich dem Laden der Headerinformationen der Entscheidungssituationen vorgenommen. Bisher wurden diese im Konstruktor der Klasse `DecisionController` geladen. Das hatte zur Folge, dass diese Informationen unnötigerweise auch für die `MakeDecisionActivity` (Ansicht der Faktorbewertungen), die `DecisionResultActivity` (Ansicht der Handlungsempfehlung) und neu auch für die `DecisionHistoryActivity` (Ansicht der Entscheidungs-Historie) geladen werden. Da die Headerinformationen der Entscheidungssituationen nur in der `ChooseDecisionActivity` (Ansicht der verfügbaren Entscheidungssituationen) benötigt werden, wurde diese Funktionalität in eine separate Methode `getDecisionHeaders` extrahiert.

In der vorherigen Implementation griff die Klasse `ChooseDecisionListViewAdapter`, welche die Auflistung der vorhandenen Entscheidungssituationen macht, auf jede Headerinformation einzeln zu (mit der Methode `getHeaderAt(int index)`). Neu wird dieser Klasse direkt die Liste der Headerinformationen der vorhandenen Entscheidungssituationen im Konstruktor übergeben und an den Konstruktor der Basisklasse `ArrayAdapter` weitergegeben. Dadurch muss die `getCount` Methode der Klasse `ArrayAdapter` nicht mehr überschrieben werden. Daher kann auch die Methode `getDecisionCount` aus der Klasse `DecisionController` entfernt werden, die lediglich an dieser einen Code-Stelle benötigt wurde.

#### 4.3.14.2 view.helper.MotiveProfileHelper

In dieser Klasse befindet sich die Methode zur Ermittlung des anzuzeigenden Texts der Motiv-Einstufung im Motivprofil bezüglich des Durchschnitts, siehe die Rot umrandeten Texte in Abbildung 34. Diese drei möglichen Texte waren fälschlicherweise als Strings direkt in der Klasse `MotiveProfileHelper` enthalten und wurden während dieses Refactorings neu als String Ressourcen definiert.

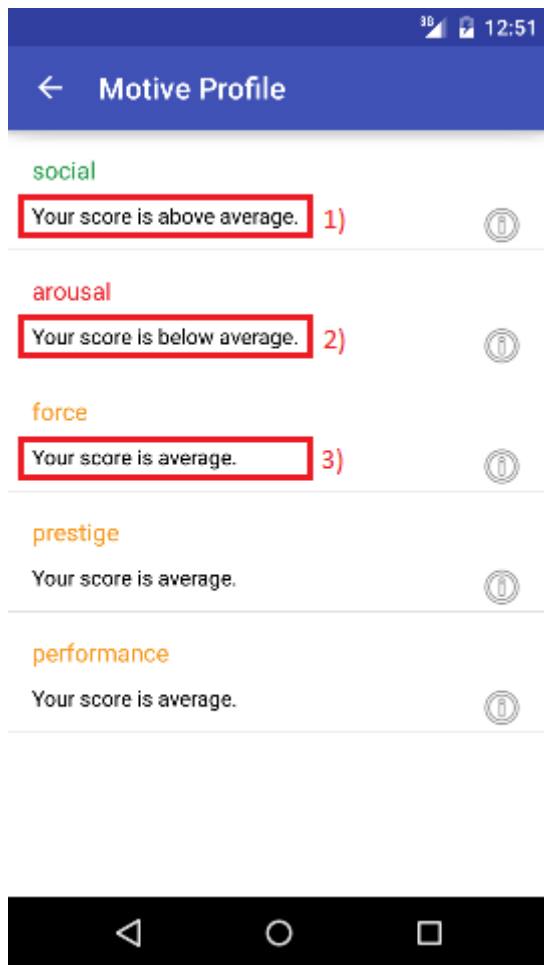


Abbildung 34: Darstellung der betroffenen Texte mit roter Umrandung

## 4.4 Spike: System mit fehlenden Daten testen und Probleme beheben

### 4.4.1 Android App

Bis anhin führte das Speichern eines leeren Motivprofil-Fragebogens zu einem Absturz der App. Grund dafür war eine Division durch 0. Die Methode `generateAndSaveProfile` in `application.MotiveProfileController` prüft zwar, ob der Fragebogen nicht `null` ist, jedoch nicht, ob er leer ist. Beim Berechnen der Durchschnittswerte wurde dann durch 0 geteilt. Deshalb wurde diese Bedingung ergänzt. Ist der Fragebogen leer, wird kein Benutzer-Profil erstellt.

Bei der Berechnung der Handlungsempfehlung war ebenfalls eine Anpassung nötig. Fehlt das Benutzer-Profil oder die Rating-Table stürzte die App ab, wenn ein Faktor als orange eingestuft wurde. Um dies zu beheben wurde die Methode `calculateActionRecommendation` in `application.DecisionController` verbessert. Falls ein Faktor als orange eingestuft wurde, ermittelt sie die Korrektursumme (siehe Abschnitt 1.1.4.3.1) nur, wenn das Benutzer-Profil und die Rating-Table vorhanden sind.

Sofern der Motivprofil-Fragebogen leer ist, also keine Fragen enthält, wird das Motivprofil des Benutzers nicht erstellt. Wollte sich der Benutzer sein Motivprofil anzeigen lassen, war nichts passiert. Deshalb wurde nun ein Toast erstellt, der den Benutzer informiert, dass das Profil nicht geladen werden konnte. Dazu wurde in der Klasse `view.MotiveProfileActivity` in der



Methode `showMotiveProfile` das Erstellen des Toast hinzugefügt. Die Meldung wurde als Ressource in `res/values/strings.xml` ergänzt.

Wenn die App keine Entscheidungssituationen oder keine Faktoren gespeichert hat, wird eine leere Auswahl der Entscheidungssituation angezeigt, aber die App stürzt nicht ab.

Konnte die App keine Motivprofil-Spezifikationen abrufen, kann der Benutzer sein Profil trotzdem einsehen. Die App erstellt Spezifikationen mit Standardwerten für die Anzeige.

## 4.4.2 Administration

Die Administration wurde ebenfalls getestet. Bei allen fünf Seiten (Motive Profiles, Decisions, Factors, Questionnaire, Ratingtable) wurden alle Einträge gelöscht. Diese Löschung fand einmal pro Seite separat (gelöscht, Datenbank zurücksetzen, nächste Seite) und einmal "gleichzeitig" (bei allen fünf Seiten alles gelöscht und dann die Datenbank zurückgesetzt) statt. Nachdem die Seite keine Einträge mehr zum Anzeigen hatte, wurden wieder Einträge erstellt.

Es traten jeweils keine Probleme auf. Die Seite wurde jeweils ohne Einträge und ohne Fehlermeldung dargestellt. Auch die Ladebalken werden jeweils ausgeblendet. Das Erstellen neuer Einträge verlief ebenfalls problemlos.

## 4.5 Qualitätssicherung

### 4.5.1 Code Reviews

#### 4.5.1.1 Löschen aller Daten einer Sprache

Titel	Inhalt
<b>Verfasser</b>	Raphael und Sascha
<b>Nachprüfer</b>	Gemeinsame Überlegungen
<b>Datum</b>	24.04.2015
<b>Beschreibung</b>	Analyse: Was geschieht in der Synchronisierung, wenn alle Daten (z.B. alle Entscheidungssituationen) einer Sprache gelöscht werden?
<b>Betroffene Elemente</b>	Alle bisher implementierten Datenbank Klassen, dabei jeweils die Methode zur Ermittlung der Daten für die Android Client Synchronisierung.
<b>Bemerkungen</b>	Voraussetzung: Englische Sprache ist definiert (Ist-Zustand).  Im folgenden Beispiel wird die Ist-Situation beschrieben und von Sprache Deutsch ausgegangen, wobei der Android Client bereits einmal die deutschen Daten heruntergeladen hat, diese dann aber im Back-End vollständig (alle Einträge) als gelöscht markiert wurden: <ul style="list-style-type: none"><li>• Android Client macht Anfrage für Sprache Deutsch. REST Service sieht, dass es deutsche Daten in der Datenbank gibt (keine Prüfung des <code>deleted</code> Flag).<ul style="list-style-type: none"><li>– Beschluss: Dieses Vorgehen so beibehalten, denn die zu synchronisierenden Daten werden erst später ermittelt.</li></ul></li><li>• REST Service ermittelt die zu synchronisierenden Daten für die Sprache Deutsch. Dabei werden alle nicht bereits vom Android Client gelöschten Daten ermittelt. Diese werden in XML gepackt und dem Android Client mitgeteilt, dass diese ebenfalls gelöscht werden müssen.</li></ul>

Titel	Inhalt
	<ul style="list-style-type: none"> <li>– Beschluss: Dieses Vorgehen erweitern, mit einer zusätzlichen Abfrage, ob es nur noch als gelöscht markierte Daten in der Datenbank für die Sprache Deutsch gibt. Wenn ja, so soll ein "Fallback" auf Englisch geschehen. Dabei sollen alle vorhandenen Daten in Englisch abgefragt und ebenfalls zum Android Client geschickt werden. Dieser löscht somit den Rest der deutschen Daten und trägt die englischen Daten lokal ein.</li> </ul>
Status	Abgeschlossen

#### 4.5.1.2 Ticket 19434: REST API: Decisions

Titel	Inhalt
Verfasser	Raphael
Nachprüfer	Sascha
Datum	24.04.2015
Beschreibung	Abfrage aller geänderten Entscheidungssituationen für Android Clients im Vergleich zu Abfrage aller gültigen Entscheidungssituationen für Decisio Administration.
Betroffene Elemente	<p>REST Klasse <code>Decisions</code>, dabei die Methoden:</p> <ul style="list-style-type: none"> <li>• <code>getDecisionsAsXml(lang, since)</code></li> <li>• <code>getDecisionsAsJson(lang)</code></li> </ul>
Bemerkungen	<ul style="list-style-type: none"> <li>• Der Unterschied in diesen 2 Methoden ist, dass für die Decisio Administration alle als gelöscht markierten Entscheidungssituationen nicht relevant sind und daher nicht zurückgegeben werden müssen. Für die Android Clients müssen diese jedoch beachtet werden und mit dem <code>deleted</code> Flag versehen werden, sodass diese auf den Android Geräten lokal gelöscht werden können. <ul style="list-style-type: none"> <li>– Beschluss: Diese beiden Funktionalitäten sollten daher in separaten Methoden implementiert werden. Um duplizierten Code zu vermeiden, sollen diese beiden Methoden gleichen Code mit einer gemeinsam genutzten Methode verwenden.</li> </ul> </li> </ul>
Status	Nachbearbeitung

#### 4.5.1.3 Ticket 19434: REST API: Decisions (Nachprüfung)

Titel	Inhalt
Verfasser	Raphael
Nachprüfer	Sascha
Datum	27.04.2015
Beschreibung	Die Umsetzung der Synchronisierung der Android Clients für die vorgegebenen Entscheidungssituation, nach der Analyse aus 4.5.1.1.
Betroffene Elemente	<ul style="list-style-type: none"> <li>• <code>DbHelper.getLanguageOrDefault(connection, language, since, tableName, languageColumn, deletedColumn, timestampColumn)</code></li> <li>• <code>DbDecisions.getDecisions(lang, since)</code></li> <li>• <code>DbDecisions.getDecisions(lang)</code></li> </ul>

Titel	Inhalt
<b>Bemerkungen</b>	<ul style="list-style-type: none"> <li>• In der <code>getLanguageOrDefault</code> Methode des <code>DbHelpers</code> werden nicht alle beschriebenen Fälle korrekt abgedeckt. <ul style="list-style-type: none"> <li>– Wenn <code>since == 0L</code>, dann wird die Default Sprache PARTIALLY synchronisiert, sollte aber lediglich alle vorhandenen Elemente der Default Sprache herunterladen (FULL) → Weiteres <code>LanguageResult</code>. Ist zurzeit bei allen Synchronisierungen mit Timestamp der Fall (wurde deshalb nicht angepasst). Dies ist meiner Meinung nach Sache des DB-Controllers, ob er nun eine Initial-Synchronisierung durchführt oder nicht. (27.04.2015/rhorber) Initial-Synchronisierung im <code>DbController</code> umgesetzt, jedoch werden zwei Connections geöffnet.</li> <li>– Die <code>since == 0L</code> Abfrage müsste vor der Abfrage, ob die Frage jemals existiert hat erfolgen. Auch da könnte noch nie synchronisiert worden sein und daher alle vorhandenen Elemente der Default Sprache benötigt werden. Diese Abfrage ist nur nötig, um zu prüfen, ob der Client die gelöschte Sprache bereits gelöscht hat oder nicht. Die Abfrage <code>since == 0L</code> ist für ein "early-return", um die Datenbank-Abfrage zu sparen. (27.04.2015/rhorber)</li> </ul> </li> <li>• In der <code>getLanguageOrDefault</code> Methode des <code>DbHelpers</code> sollte für <code>COUNT(*)</code> jeweils ein <code>long</code> verwendet werden (und kein <code>int</code>). Zudem sollte auch kein Vergleich des <code>since</code> Arguments (<code>long</code>) mit einem <code>int</code> Wert gemacht werden. umgesetzt (27.04.2015/rhorber)</li> <li>• Die einzelnen Methodenabschnitte in der <code>getLanguageOrDefault</code> Methode würde ich lesbarkeitshalber in separate Methoden auslagern. aufgeteilt (27.04.2015/rhorber)</li> <li>• Die letzte Query in der <code>getLanguageOrDefault</code> Methode könnte man auch in der entsprechenden Db Klasse machen (auf <code>ArrayList</code>). Dadurch würde man sich eine Datenbank-Abfrage sparen. Missverständnis mit dem Sinn der dritten SQL-Abfrage (27.04.2015/rhorber)</li> <li>• Die Methode <code>getLanguageOrDefault</code> hat sich jetzt mehr zu einer Art "Abfrage der Synchronisierungs-Aktion" verändert, daher ist mein Vorschlag, sie in <code>getSynchronizationAction</code> umzubenennen. umbenannt (27.04.2015/rhorber)</li> <li>• Die Methode <code>getDecisions(lang)</code> für die Decisio Administration soll kein Aufruf der Methode <code>getLanguageOrDefault</code> machen. Die Anfrage kann ja nur für Sprachen kommen, die wirklich in der Datenbank verfügbar sind. Wenn man dem GET-Argument vertraut, ja. Da nur eine leere Liste zurückgegeben würde, umgesetzt und Unit Tests angepasst. (27.04.2015/rhorber)</li> <li>• Verbesserungsvorschläge von rhorber zu <code>LanguageResult</code>: <ul style="list-style-type: none"> <li>– In eine eigene Klasse/Datei verschieben und allenfalls umbenennen. Umbenennen besonders dann, wenn die Methode im <code>DbHelper</code> nicht mehr <code>getLanguageOrDefault</code> heisst. In eigene Klasse <code>SynchronizationAction</code> verschoben und nach <code>SynchronizationAction</code> umbenannt. (27.04.2015/rhorber)</li> <li>– Die Sprache gleich bei der Instanziierung mitgeben oder setzen, damit sie nicht der Methode <code>getLanguage</code> nochmals mitgegeben werden muss. umgesetzt (27.04.2015/rhorber)</li> </ul> </li> </ul>

Titel	Inhalt
Status	Abgeschlossen

## 4.6 Systemtest

Als Ergänzung zu den automatisierten Unit Tests haben wir einen zusätzlichen manuellen Systemtest geplant und durchgeführt. In den folgenden Abschnitten sind die Testfälle mit den erwarteten und den tatsächlichen Resultaten beschrieben.

### 4.6.1 Vorbereitung und allgemeine Vorbedingungen

Die Android App muss vor den Tests auf einem echten Gerät (kein Emulator) installiert worden sein. Zudem muss das Gerät Zugang zum Internet haben.

Für die Tests der Administration muss ein PC mit Zugang zum Internet zur Verfügung stehen.

Führen sie das Reset-Skript aus Abschnitt 4.1.1 auf dem Back-End-Server aus.

## 4.6.2 Systemtestspezifikation und Durchführung

Testdatum: 09.06.2015

Getestete Version: Revision 218

Verwendete Geräte: Samsung Galaxy S4 active (GT-I9295) mit Android 5.0.1

### 4.6.2.1 Testfälle der Studienarbeit

Die folgenden Unterabschnitte sind Testfälle, die wir aus unserer Studienarbeit übernommen haben, da mögliche eingebaute Fehler in der Android App ebenfalls ausgeschlossen werden sollen. Dabei haben wir die Abschnitte 4.6.2.1.2, 4.6.2.1.3, 4.6.2.1.5 und 4.6.2.1.6 nur geringfügig geändert, um die Daten in der Back-End-Datenbank anstatt auf dem SFTP-Server anzuschauen.

#### 4.6.2.1.1 App starten

**Vorbedingungen:** App wurde neu auf dem Gerät installiert und besitzt noch keine Daten im Speicher.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App.	System öffnet die App und zeigt den Disclaimer Text in einem Dialog an.	Bestanden
2	Lehnen Sie den Disclaimer Text ab.	App wird geschlossen.	Bestanden
3	Starten Sie die App erneut.	System öffnet die App und zeigt den Disclaimer Text erneut in einem Dialog an.	Bestanden
4	Schliessen Sie die App und schalten Sie jegliche Internetverbindungsmöglichkeiten (WLAN, mobile Daten, ...) ab. Starten Sie die App ein weiteres Mal.	System schliesst und öffnet die App. Disclaimer Text wird in einem Dialog angezeigt.	Bestanden
5	Akzeptieren Sie den Disclaimer Text.	System zeigt eine Informationsmeldung an, dass einmal eine Internetverbindung benötigt wird und schliesst bei Bestätigung die App.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
6	Schalten Sie eine Internetverbindungsmöglichkeit ein und starten Sie die App.	System startet die App lädt die neuesten Einstellungen vom Server herunter. Der Disclaimer Text wird nicht angezeigt.  System fordert den Benutzer zur Eingabe der Soziodemografischen Daten auf.	Bestanden
7	Führen Sie den Testfall "Soziodemografische Daten erfassen" aus.		Bestanden
8	Schliessen Sie die App und schalten Sie jegliche Internetverbindungsmöglichkeiten (WLAN, mobile Daten, ...) ab. Starten Sie die App ein weiteres Mal.	System schliesst und öffnet die App.  System fordert den Benutzer direkt zum Ausfüllen des Motivprofil-Fragebogens auf.	Bestanden
9	Führen Sie den Testfall "Motivprofil erstellen" aus.	System zeigt nun die Startseite der App an.	Bestanden
10	Schliessen Sie die App und starten Sie die App ein weiteres Mal.	System zeigt direkt die Startseite der App an.	Bestanden

#### 4.6.2.1.2 Soziodemografische Daten erfassen

##### Vorbedingungen:

- Testfall "App starten" wurde bis Schritt 7, jedoch noch nicht weiter ausgeführt.
- Für diesen Testfall ist der Zugriff auf den Back-End-Server erforderlich.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App, falls noch nicht erledigt.	System fordert den Benutzer zum Eingeben der Soziodemografischen Daten auf.	Bestanden
2	Geben Sie als Geschlecht 'Männlich' und als Jahrgang '1980' ein.  Versuchen Sie die Eingaben zu speichern.	System zeigt Meldung an, dass noch nicht alle Daten eingegeben wurden.  Back-Navigation nicht möglich.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
3	Wählen Sie als Abschluss die erste Option. Versuchen Sie die Eingaben zu speichern.	System zeigt Meldung an, dass noch nicht alle Daten eingegeben wurden. Back-Navigation nicht möglich.	Bestanden
4	Wählen Sie als Einkommensklasse die zweite Option. Versuchen Sie die Eingaben zu speichern.	System akzeptiert die Eingaben und verlässt die Soziodemografischen Daten.	Bestanden
5	Greifen sie auf die Datenbank auf dem Back-End-Server zu und lassen sie sich die soziodemografischen Daten des Benutzers anzeigen.	Soziodemografische Daten des Benutzers enthalten die soeben eingegebenen Daten.	Bestanden

#### 4.6.2.1.3 Motivprofil erstellen

##### Vorbedingungen:

- Testfall "App starten" wurde bis Schritt 9, jedoch noch nicht weiter ausgeführt.
- Für Schritt 4 sind Informationen über den Fragebogen auf dem Server notwendig.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App, falls noch nicht erledigt.	System fordert den Benutzer zum Ausfüllen des Motivprofil-Fragebogens auf.	Bestanden
2	Beantworten Sie einige, jedoch noch nicht alle Fragen des Fragebogens und versuchen Sie die Eingaben zu speichern.	System zeigt Meldung an, dass noch nicht alle Fragen beantwortet wurden. Back-Navigation nicht möglich.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
3	<p>Beantworten Sie alle Fragen des Fragebogens wie folgt:</p> <ul style="list-style-type: none"> <li>- Alle Fragen zu Motiven 'performance' und 'arousal' mit Wert: 1</li> <li>- Alle Fragen zu Motiven 'social' und 'force' mit Wert: 5</li> <li>- Alle Fragen zu Motiv 'prestige' mit einem Wert im Durchschnittsbereich für 'prestige'.</li> </ul> <p>Versuchen Sie die Eingaben zu speichern.</p>	System akzeptiert die Eingaben und zeigt das daraus erstellte Motivprofil an.	Bestanden
4	Überprüfen Sie die Werte des Motivprofils.	<ul style="list-style-type: none"> <li>- Die Motive 'performance' und 'arousal' sind als Rot und unter dem Durchschnitt angezeigt.</li> <li>- Die Motive 'social' und 'force' sind als Grün und über dem Durchschnitt angezeigt.</li> <li>- Das Motiv 'prestige' ist als Orange und durchschnittlich angezeigt.</li> </ul>	Bestanden
5	Akzeptieren Sie das Motivprofil.	System zeigt Startseite der App an.	Bestanden
6	Greifen sie auf die Datenbank auf dem Back-End-Server zu und lassen sie sich die Antworten zum Fragebogen des Benutzers anzeigen.	Antworten zum Motivprofil-Fragebogen des Benutzers enthalten die in Schritt 3 eingegebenen Daten.	Bestanden

#### 4.6.2.1.4 Entscheidung fällen

##### Vorbedingungen:

- Disclaimer akzeptiert, Soziodemografische Daten eingegeben und Motivprofil erstellt.
- Für Schritt 5 sind Informationen über die Entscheidungssituation auf dem Server notwendig.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App, falls noch nicht erledigt.	System öffnet die Startseite der App.	Bestanden



Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Wählen Sie die Option, um eine Entscheidung zu fällen.	System zeigt den Katalog der vorhandenen Entscheidungssituationen an.	Bestanden
3	Wählen Sie eine vorgegebene Entscheidungssituation aus.	System zeigt Seite zur Einstufung des ersten Entscheidungsfaktors an.	Bestanden
4	Stufen Sie den Faktor als 'Orange' ein.	System zeigt Seite zur Einstufung des nächsten Entscheidungsfaktors an.	Bestanden
5	Stufen Sie die nächsten Faktoren wie folgt ein: - Einen Faktor als 'Grün'. - 2 Faktoren als 'Rot'. - Einen Faktor als 'Nicht relevant'. - Alle nachfolgenden Faktoren als 'Orange'.	System führt den Benutzer durch die zur Entscheidungssituation gehörenden Entscheidungsfaktoren. Die Reihenfolge entspricht der Reihenfolge in der Administration.	Bestanden
6	Überprüfen Sie die Handlungsempfehlung und die Zusammenfassung der Faktor-Einstufungen.	System zeigt negative Handlungsempfehlung an. Die entsprechenden Faktoren werden korrekt angezeigt (Grün/Orange/Rot). Der als 'Nicht relevant' eingestufte Faktor erscheint nicht in der Zusammenfassung.	Bestanden
7	Führen Sie eine Back-Navigation durch.	System zeigt Katalog der vorhandenen Entscheidungssituationen an. Die Einstufungen der Entscheidungsfaktoren können nicht verändert werden.	Bestanden
8	Überprüfen Sie das Datum, wann die soeben ausgeführte Entscheidungssituation das letzte Mal ausgeführt wurde.	Das Datum der letzten Ausführung wurde auf das heutige Datum gesetzt.	Bestanden

#### 4.6.2.1.5 Soziodemografische Daten bearbeiten

##### Vorbedingungen:

- Disclaimer akzeptiert, Soziodemografische Daten eingegeben und Motivprofil erstellt.

- Für diesen Testfall ist der Zugriff auf den Back-End-Server erforderlich.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App.	System öffnet die Startseite der App.	Bestanden
2	Wählen Sie die Option, um ihre soziodemografischen Daten zu bearbeiten.	System öffnet die Seite der soziodemografischen Daten.	Bestanden
3	Ändern Sie alle Daten und notieren Sie sich die geänderten Werte.	Soziodemografische Daten geändert.	Bestanden
4	Speichern Sie die Daten.	Soziodemografische Daten gespeichert. System öffnet die Startseite der App.	Bestanden
5	Wählen Sie die Option, um ihre soziodemografischen Daten zu bearbeiten.	System öffnet die Seite der soziodemografischen Daten.	Bestanden
6	Überprüfen Sie, ob Ihre geänderten Werte erfolgreich gespeichert und geladen wurden.	Soziodemografische Daten entsprechen den geänderten Werten.	Bestanden
7	Greifen sie auf die Datenbank auf dem Back-End-Server zu und lassen sie sich die soziodemografischen Daten des Benutzers anzeigen.	Soziodemografische Daten des Benutzers enthalten die soeben geänderten Daten.	Bestanden

#### 4.6.2.1.6 Motivprofil bearbeiten

##### Vorbedingungen:

- Disclaimer akzeptiert, Soziodemografische Daten eingegeben und Motivprofil erstellt.
- Für Schritt 3 sind Informationen über den Fragebogen auf dem Server notwendig.
- Für diesen Test Case ist der Zugriff auf den Back-End-Server erforderlich.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten Sie die App.	System öffnet die Startseite der App.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Wählen Sie die Option, um Ihr Motivprofil zu bearbeiten.	System öffnet den Motivprofil-Fragebogen. Der Fragebogen enthält die letzten Einstufungen des Benutzers.	Bestanden
3	Ändern Sie die Einstufungen aller Fragen zum Motiv "performance" auf 5 und speichern Sie das Motivprofil.	Einstufungen geändert und gespeichert.	Bestanden
4	Wählen Sie erneut die Option, um ihr Motivprofil zu bearbeiten.	System öffnet den Motivprofil-Fragebogen.	Bestanden
5	Überprüfen Sie, ob Ihre geänderten Werte erfolgreich gespeichert und geladen wurden.	Die Einstufungen der Fragen entsprechen den geänderten Werten.	Bestanden
6	Wählen Sie die Option, um Ihr Motivprofil anzuzeigen.	System öffnet Motivprofil Ansicht.	Bestanden
7	Überprüfen Sie, dass Ihr Wert für "performance" auf Grün und über dem Durchschnitt angezeigt wird.	Wert für "performance" wird als Grün und über dem Durchschnitt angezeigt.	Bestanden
8	Greifen sie auf die Datenbank auf dem Back-End-Server zu und lassen sie sich die Antworten zum Fragebogen des Benutzers anzeigen.	Antworten zum Motivprofil-Fragebogen des Benutzers enthalten die in Schritt 3 eingegebenen Daten.	Bestanden

## 4.6.2.2 Administration

### 4.6.2.2.1 Testfall zu User Story: Setup RDBMS

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Datenbank auf dem Back-End-Server zu.	Relationale Datenbank ist auf dem Back-End-Server verfügbar.	Bestanden
2	Fragen sie einige Daten der Datenbank ab.	Die DB ist mit einem initialen Dataset geseeded	Bestanden
3	Löschen sie bestehende Daten aus der Datenbank und fügen sie einige neue Daten ein.	Mindestens 1 Eintrag gelöscht und 1 neuer Eintrag erstellt.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
4	Führen sie einen Reset der Datenbank mit dem in Abschnitt 4.1.1 beschriebenen Skript.	Datenbank kann mit einem Skript resettet werden. Die Datenbank enthält den in Schritt 3 erstellten Eintrag nicht mehr, dafür aber den gelöschten Eintrag wieder.	Bestanden
5	Exportieren sie die Daten der Datenbank anhand der Beschreibung in Abschnitt 4.1.1.	Die Werte in der DB können exportiert werden.	Bestanden

#### 4.6.2.2.2 Testfall zu User Story: Zugriff auf Back-End

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Das Back-End erscheint mit einem Splash Scen im Browser.	Bestanden
2	Überprüfen sie den Link der Administration.	Das Back-End läuft auf Heroku.	Bestanden
3	Schauen sie den Quellcode der Seite an.	Die Seite ist mit AngularJS aufgebaut und enthält auch Angular Material Elemente.	Bestanden
4	Prüfen sie, ob der Splash Screen mit "Cards" umgesetzt wurde.	Der Splash Screen wurde mit Cards umgesetzt.	Bestanden

#### 4.6.2.2.3 Testfall zu User Story: Back-End Menu

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser und enthält eine Menu Navigation.	Bestanden
2	Überprüfen sie, dass folgende Menüpunkte vorhanden sind: HOME, MOTIVE PROFILES, DECISIONS, FACTORS, QUESTIONNAIRE, RATINGTABLE.	Die beschriebenen Menüpunkte sind vorhanden.	Bestanden
3	Überprüfen sie, dass die Menüpunkte als "Tabs" umgesetzt wurden.	Menu wurde mit Tabs umgesetzt.	Bestanden

#### 4.6.2.2.4 Testfall zu User Story: Screen zu motiveProfileSpecifications\_en.xml

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Öffnen sie den Tab "MOTIVE PROFILES".	Der Inhalt der Seite wird als Liste dargestellt.	Bestanden
3	Wählen sie die "Editieren" Funktion für einen Eintrag und prüfen sie die Funktionalität.	Die Felder aboveExplanation, averageExplanation, belowExplanation, averageThreshold und aboveAverageThreshold sind editierbar.  Das Feld "motive" wird angezeigt, ist aber nicht editierbar.  Das Feld "motive" wird zusätzlich auch als Icon dargestellt.	Bestanden
4	Ändern sie alle Werte dieses Eintrags, notieren sie sich die geänderten Werte und speichern sie die Änderungen.	Die Änderungen können gespeichert werden.	Bestanden
5	Stellen sie sicher, dass die geänderten Werte aus Schritt 4 erfolgreich gespeichert wurden und in der Übersichtsliste der Administration erscheinen.	Die geänderten Werte erscheinen im Tab "MOTIVE PROFILES" in der Administration.	Bestanden
6	Stellen sie sicher, dass das Feld "motive" zusätzlich auch als Icon in der Übersicht dargestellt wird.	Das Feld "motive" wird im Tab "MOTIVE PROFILES" in der Administration als Icon dargestellt.	Bestanden
7	Stellen sie sicher, dass sie die Inhalte aller 5 Motive in der Übersicht erscheinen und sich editieren lassen.	Die Spezifikationen aller 5 Motive lassen sich in der Administration bearbeiten.	Bestanden

#### 4.6.2.2.5 Testfall zu Änderung: Screen Motive Profiles

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Öffnen sie den Tab "MOTIVE PROFILES".	Die Überschrift der Seite und der Name des Tabs lautet "MOTIVE PROFILES".	Bestanden
3	Prüfen sie, dass beim Feld "motive" nicht nur das Icon, sondern auch der Name des Motivs als Text angezeigt wird.	Name des Motivs wird in der Übersichtsliste und in der Editierung auch als Text angezeigt.	Bestanden
4	Prüfen sie, dass der Button "Motive Icons" nicht erscheint.	Der Button "Motive Icons" erscheint nicht in der Übersichtsliste und auch nicht in der Editierung.	Bestanden
5	Prüfen sie die Angabe des Durchschnittsbereichs in der Übersichtsliste.	Der Durchschnittsbereich wird in der Form "Average range goes from X to Y" in der Übersichtsliste.	Bestanden
6	Prüfen sie, dass in der Übersichtsliste die Erläuterungstexte mit dem entsprechenden Label angezeigt werden, siehe 2.1.5.	Die Erläuterungstexte erscheinen mit dem entsprechenden Label in der Übersichtsliste.	Bestanden

#### 4.6.2.2.6 Testfall zu Änderung: Sprachfunktionalität in Motive Profiles umsetzen

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu und öffnen sie den Tab "MOTIVE PROFILES".	Die Administration erscheint im Browser.	Bestanden
2	Fügen sie eine neue Sprachdefinition für "fr" hinzu. Die Daten müssen keinen speziellen Inhalt haben, es muss dabei aber jedes Feld ausgefüllt werden.	Die neue Sprachdefinition erscheint in der Übersichtsliste und enthält die eingegebenen Daten.	Bestanden
3	Löschen sie die in Schritt 2 erstellte Sprachdefinition.	Die gesamte Sprachdefinition wurde gelöscht und lässt sich nicht mehr in der Übersichtsseite anzeigen.	Bestanden

#### 4.6.2.2.7 Testfall zu User Story: Screen zu questionnaire\_en.xml

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Öffnen sie den Tab "QUESTIONNAIRE".	Der Inhalt der Seite wird als Liste dargestellt.	Bestanden
3	Sortieren sie die Übersichtsliste nach "number".	Die Liste ist sortierbar nach "number". Zudem sind die Buttons zur Verschiebung der Einträge sichtbar.	Bestanden
4	Sortieren sie die Übersichtsliste nach "motive".	Die Liste ist sortierbar nach "motive". Die Buttons zur Verschiebung der Einträge sind nicht sichtbar.	Bestanden
5	Überprüfen sie, dass das Feld "motive" als Icon dargestellt wird.	Das Feld "motive" wird als Icon dargestellt.	Bestanden
6	Wählen sie die "Editieren" Funktion für einen Eintrag und prüfen sie die Funktionalitäten.	Das Motiv einer Frage kann geändert werden. Der Fragetext kann editiert werden.	Bestanden
7	Ändern sie alle Werte dieses Eintrags, notieren sie sich die geänderten Werte und speichern sie die Änderungen.	Die Änderungen können gespeichert werden.	Bestanden
8	Stellen sie sicher, dass die geänderten Werte aus Schritt 7 erfolgreich gespeichert wurden und in der Übersichtsliste der Administration erscheinen.	Die geänderten Werte erscheinen im Tab "QUESTIONNAIRE" in der Administration.	Bestanden
9	Fügen sie eine neue Frage hinzu.	Neue Fragen können definiert werden.	Bestanden
10	Ändern sie die Reihenfolge der Fragen, sodass die in Schritt 9 erstellte Frage nicht mehr am Ende erscheint.	Die Reihenfolge der Fragen kann geändert werden.	Bestanden
11	Löschen sie die erstellte Frage aus Schritt 9.	Bestehende Fragen können gelöscht werden.	Bestanden

#### 4.6.2.2.8 Testfall zu User Story: Screen zu factors\_en.xml

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Öffnen sie den Tab "FACTORS" und prüfen sie den Inhalt der Seite.	Der Inhalt der Seite wird als Liste dargestellt. Die Liste zeigt an: "Name", "Description", "socialMapping", "arousalMapping", "forceMapping", "prestigeMapping", "performanceMapping".	Bestanden
3	Sortieren sie die Übersichtsliste nach "name".	Die Liste ist alphabetisch sortierbar nach "name".	Bestanden
4	Sortieren sie die Übersichtsliste nach "socialMapping".	Die Liste ist absteigend sortierbar nach "socialMapping".	Bestanden
5	Sortieren sie die Übersichtsliste nach "arousalMapping".	Die Liste ist absteigend sortierbar nach "arousalMapping".	Bestanden
6	Sortieren sie die Übersichtsliste nach "forceMapping".	Die Liste ist absteigend sortierbar nach "forceMapping".	Bestanden
7	Sortieren sie die Übersichtsliste nach "prestigeMapping".	Die Liste ist absteigend sortierbar nach "prestigeMapping".	Bestanden
8	Sortieren sie die Übersichtsliste nach "performanceMapping".	Die Liste ist absteigend sortierbar nach "performanceMapping".	Bestanden
9	Wählen sie die "Editieren" Funktion für einen Eintrag und prüfen sie die Funktionalitäten.	Die Felder "Name", "Description", "socialMapping", "arousalMapping", "forceMapping", "prestigeMapping", "performanceMapping", "warningExplanation", "doNotProceedExplanation" sind editierbar. Die Mapping Werte können mit der Tastatur, ohne Verwendung der Maus, editiert werden.	Bestanden
10	Überprüfen sie, dass sie die mit dem Faktor verknüpften Entscheidungssituationen einsehen können.	Mit einem Faktor verknüpfte Decisions können eingesehen werden (ohne Link).	Bestanden



Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
11	Ändern sie alle Werte dieses Eintrags, notieren sie sich die geänderten Werte und speichern sie die Änderungen.	Die Änderungen können gespeichert werden und erscheinen erfolgreich geändert in der Übersichtsliste der Administration.	Bestanden
12	Fügen sie einen neuen Faktor hinzu.	Neue Faktoren können definiert werden.	Bestanden
13	Löschen sie den erstellten Faktor aus Schritt 12.	Bestehende Faktoren können gelöscht werden.	Bestanden

#### 4.6.2.2.9 Testfall zu User Story: Screen zu ratingTable.xml

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Öffnen sie den Tab "RATINGTABLE" und prüfen sie den Inhalt der Seite.	Der Inhalt der Seite wird als Liste dargestellt.	Bestanden
3	Erstellen sie einen neuen Eintrag mit Werten für die 5 verschiedenen Motivprofil Werte.	Ein neuer Eintrag kann erstellt werden.	Bestanden
4	Ändern sie den in Schritt 3 erstellten Eintrag: Ändern sie alle Werte, notieren sie sich die geänderten Werte und speichern sie die Änderungen. Wählen sie dabei mindestens einen positiven und einen negativen Wert.	Die Änderungen können gespeichert werden und erscheinen erfolgreich geändert in der Übersichtsliste der Administration. Negative Werte werden unterstützt.	Bestanden
5	Löschen sie den in Schritt 3 erstellten Eintrag.	Einträge können gelöscht werden.	Bestanden
6	Prüfen sie, dass nur der letzte Eintrag gelöscht werden kann.	Nur der letzte Eintrag kann gelöscht werden.	Bestanden
7	Fügen sie einen neuen Eintrag hinzu und prüfen sie die Nummerierung des Faktorwerts.	Die Nummerierung der Faktorwerte erfolgt automatisch. In der Skala ist kein "Loch" entstanden, d.h. es besteht immer noch eine fortlaufende Nummerierung der Faktorwerte.	Bestanden

#### 4.6.2.2.10 Testfall zu User Story: Screen zu decisions\_en.xml

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Öffnen sie den Tab "DECISIONS" und prüfen sie den Inhalt der Seite.	Die Entscheidungssituationen und deren Faktoren werden als Liste dargestellt.	Bestanden
3	Fügen sie eine neue Entscheidungssituation hinzu. Definieren sie dafür mindestens 2 Faktoren.	Eine Entscheidungssituation kann hinzugefügt werden.	Bestanden
4	Ändern sie den in Schritt 3 erstellten Eintrag: Ändern sie den Fragetext der Entscheidungssituation. Fügen sie der Entscheidungssituation einen neuen Faktor hinzu und entfernen sie einen in Schritt 3 erfassten Faktor von der Entscheidungssituation.	Die Änderungen können gespeichert werden und erscheinen erfolgreich geändert in der Übersichtsliste der Administration.  Die "question" einer "decision" kann editiert werden.  Faktoren können zu einer "decision" hinzugefügt oder entfernt werden.	Bestanden
5	Löschen sie den in Schritt 3 erstellten Eintrag.	Einträge können gelöscht werden.	Bestanden
6	Schauen sie den Quellcode der Seite an.	Die Version von Angular Material ist mind. 0.9 und die Chips zum Bearbeiten der zur Entscheidungssituation ausschlaggebenden Faktoren funktionieren.	Bestanden

#### 4.6.2.2.11 Testfall zu Änderung: Alignment des Cancel und Save Buttons in allen Screens

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Öffnen sie den alle Tabs und öffnen sie darin jeweils die "Editieren" Funktion. Prüfen sie in den Editier-Screens, dass die Buttons "Save" und "Cancel" nahe beieinander angezeigt werden.	Die Buttons "Cancel" und "Save" werden in jedem Screen nahe beieinander auf der rechten Seite des Screens platziert.	Bestanden

#### 4.6.2.2.12 Testfall zu RESTful Web Service

Die Umsetzung der REST Schnittstelle wurde mit den Tests der Administration schon grösstenteils geprüft. Folgend sind nur einige zusätzliche Schritte beschrieben.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Öffnen sie den Tab "DECISIONS", verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie das Format der erhaltenen Daten vom REST Service.	Die Administration kommuniziert mit JSON über die REST Schnittstelle mit der Datenhaltung	Bestanden
3	Bearbeiten sie eine Entscheidungssituation und verfolgen sie dabei den Netzwerkverkehr ihres Browsers.	Auch Requests zur Bearbeitung der Daten in der Administration erfolgen über JSON.	Bestanden
4	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, siehe Abschnitt 4.1.2.4. Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie das Format der erhaltenen Daten vom REST Service.	Der Android Client kommuniziert mit XML über die REST Schnittstelle mit der Datenhaltung.	Bestanden

#### 4.6.2.2.13 Testfall zu Spike: Problem mit Connections anschauen (Connection Pooling)

Diese User Story kann nicht in Form eines Testfalls getestet werden, da eine nicht geschlossene Datenbank Verbindung an sehr vielen Stellen auftreten könnte. Dafür haben wir aber für alle Ressourcen entsprechende Unit Tests erstellt.

Zum Test des Connection Pools dienen die Tests der Antwortzeiten für eine grosse Anzahl an Benutzern. Dies ist in Abschnitt 4.8 dokumentiert.

#### 4.6.2.2.14 Testfall zu Spike: Konzept Fehler-Behandlung in der Administration

Zur Prüfung dieser User Story müssen Fehler in der Verbindung zur Datenbank simuliert werden. Dies ist mit geringem Aufwand nur auf dem lokalen Rechner möglich, indem der Datenbank Service ausgeschaltet wird.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Decisio Administration auf ihrem lokalen Rechner. Greifen sie auf die lokal laufende Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Schalten sie ihren lokal laufenden Datenbank Service aus. Öffnen sie anschliessend den Tab "MOTIVE PROFILES" in der Administration.	Es erscheint eine Fehlermeldung, dass die Verbindung zur Datenbank scheiterte. Die Übersichtsliste der Motivprofil Spezifikationen ist leer.	Bestanden
3	Starten sie ihren lokalen Datenbank Service wieder. Öffnen sie nun eine Motivprofil Spezifikation in der Administration.	Die Motivprofil Spezifikation wird zur Bearbeitung in der Administration angezeigt.	Bestanden
4	Schalten sie ihren lokal laufenden Datenbank Service aus und versuchen sie die Motivprofil Spezifikation in der Administration zu speichern.	Es erscheint eine Fehlermeldung, dass die Verbindung zur Datenbank scheiterte. Es bleibt die Anzeige zur Bearbeitung der Motivprofil Spezifikation angezeigt.	Bestanden
5	Öffnen sie die Übersichtsliste der Motivprofil Spezifikationen und wählen sie die Option zum Hinzufügen eines Eintrages.	Die Seite zum Hinzufügen einer neuen Sprachdefinition für Motivprofil Spezifikationen erscheint.	Bestanden
6	Versuchen sie die Motivprofil Spezifikationen in der Administration hinzuzufügen.	Es erscheint eine Fehlermeldung, dass die Verbindung zur Datenbank scheiterte. Es bleibt die Anzeige zum Hinzufügen der Motivprofil Spezifikationen angezeigt	Bestanden

#### 4.6.2.2.15 Testfall zu User Story: Fehlerbehandlung in bestehenden Screens und REST Klassen anwenden

Zur Prüfung dieser User Story müssen Fehler in der Verbindung zur Datenbank simuliert werden. Dies ist mit geringem Aufwand nur auf dem lokalen Rechner möglich, indem der Datenbank Service ausgeschaltet wird.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Decisio Administration auf ihrem lokalen Rechner. Greifen sie auf die lokal laufende Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Führen sie die Schritte 2 bis 6 des Testfalls in Abschnitt 4.6.2.2.14 für den Tab "FACTORS" aus.	Die Fehlermeldung wird immer in einem gleich aussehenden Dialog angezeigt. Im Formular zum Erstellen eines neuen Faktors erscheint eine Fehlermeldung.	Bestanden
3	Führen sie die Schritte 2 bis 5 des Testfalls in Abschnitt 4.6.2.2.14 für den Tab "DECISIONS" aus.	Die Fehlermeldung wird immer in einem gleich aussehenden Dialog angezeigt. Im Formular zum Erstellen einer neuen Entscheidungssituation erscheint eine Fehlermeldung und das Formular erscheint nicht.	Bestanden
4	Führen sie die Schritte 2 bis 6 des Testfalls in Abschnitt 4.6.2.2.14 für den Tab "QUESTIONNAIRE" aus.	Die Fehlermeldung wird immer in einem gleich aussehenden Dialog angezeigt. Im Formular zum Erstellen einer neuen Frage erscheint eine Fehlermeldung.	Bestanden
5	Führen sie die Schritte 2 bis 6 des Testfalls in Abschnitt 4.6.2.2.14 für den Tab "RATINGTABLE" aus.	Die Fehlermeldung wird immer in einem gleich aussehenden Dialog angezeigt.	Bestanden

#### 4.6.2.2.16 Testfall zu User Story: Deleted Flag in ausstehenden Klassen umsetzen

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Löschen sie eine vorgegebene Entscheidungssituation, einen Entscheidungsfaktor, einen Eintrag der Rating-Table und eine Frage aus dem Motivprofil-Fragebogen über die Administration. Notieren sie sich jeweils, welche Einträge sie gelöscht haben.	Einträge erfolgreich gelöscht. Die Einträge erscheinen nicht mehr in der Übersichtsliste in der Administration.	Bestanden
3	Fügen sie eine neue Sprachdefinition "de" für die Motivprofil Spezifikationen hinzu. Löschen sie die soeben hinzugefügte Sprachdefinition wieder.	Sprachdefinition der Motivprofil Spezifikationen erfolgreich gelöscht.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
4	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, mit dem letzten Synchronisierungszeitpunkt '0', für alle Ressourcen (siehe Abschnitt 4.1.2 zur Hilfe). Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie die erhaltenen Daten vom REST Service.	Die gelöschten Einträge aus den Schritten 2 und 3 erscheinen nicht in den Daten vom REST Service.	<b>Fehlgeschlagen:</b> die gelöschten Elemente der Rating Tabelle werden als gelöscht angezeigt
5	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, mit dem letzten Synchronisierungszeitpunkt im Januar 2015, für alle Ressourcen (siehe Abschnitt 4.1.2 zur Hilfe). Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie die erhaltenen Daten vom REST Service.	Die gelöschten Einträge aus den Schritten 2 und 3 erscheinen mit gesetztem "Deleted-Flag" in den Daten vom REST Service. Eine Ausnahme bilden die Motivprofil-Spezifikationen: Es werden die Spezifikationen in Englisch zurückgegeben.	<b>Bestanden</b>

#### 4.6.2.2.17 Testfall zu User Story: Fallback auf Englisch in ausstehenden Klassen umsetzen

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	<b>Bestanden</b>
2	Fügen sie eine Entscheidungssituation, einen Entscheidungsfaktor und eine Frage im Motivprofil-Fragebogen für die neue Sprache "it" in der Administration hinzu.  Fügen sie zudem eine neue Sprachdefinition "it" für die Motivprofil Spezifikationen hinzu.	Einträge für die neue Sprache "it" hinzugefügt.	<b>Bestanden</b>

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
3	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, mit der Sprache "it" für alle Ressourcen (siehe Abschnitt 4.1.2 zur Hilfe). Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie die erhaltenen Daten vom REST Service.	Die in Schritt 2 erfassten Einträge der Sprache "it" wurden als Antwort des REST Service geschickt.	Bestanden
4	Löschen sie alle in Schritt 2 erstellten Einträge der Sprache "it" wieder.	Alle Einträge der Sprache "it" wurden gelöscht.	Bestanden
5	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, mit der Sprache "it" für alle Ressourcen (siehe Abschnitt 4.1.2 zur Hilfe). Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie die erhaltenen Daten vom REST Service.	Als Antwort des REST Service wurden die englischen Einträge in allen Ressourcen gesendet (Fallback).	Bestanden
6	Stellen sie mit ihrem Browser eine Anfrage zur Synchronisierung der Android App, mit einer noch nie definierten Sprache, z.B. "es", für alle Ressourcen (siehe Abschnitt 4.1.2 zur Hilfe). Verfolgen sie dabei den Netzwerkverkehr ihres Browsers und überprüfen sie die erhaltenen Daten vom REST Service.	Als Antwort des REST Service wurden die englischen Einträge in allen Ressourcen gesendet.	Bestanden

#### 4.6.2.2.18 Testfall zu User Story: Update von Angular Material

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Schauen sie den Quellcode der Seite an.	Die Angular Material Version ist mindestens 0.9	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
3	Öffnen sie die Bearbeitungsseiten der folgenden Tabs: DECISIONS, FACTORS, QUESTIONNAIRE. Prüfen sie dabei das Feld zur Spracheingabe.	Der Wert im Feld zur Spracheingabe ist nicht veränderbar. Es werden keine Sprachvorschläge in einem Dropdown angezeigt.	Bestanden
4	Öffnen sie die Seiten zum Hinzufügen neuer Einträge in den folgenden Tabs: DECISIONS, FACTORS, QUESTIONNAIRE. Prüfen sie dabei das Feld zur Spracheingabe.	Der Wert im Feld zur Spracheingabe ist veränderbar. Es werden Sprachvorschläge angezeigt und mit dem eingegebenen Text gefiltert.	Bestanden

#### 4.6.2.2.19 Testfall zu User Story: Zugriffsschutz Backend

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie mittels explizitem setzen von HTTP auf die Administration über ihren Browser zu.	Das Anmeldefenster erscheint. Die Seite wurde auf HTTPS umgeleitet.	Fehlgeschlagen: Das Passwort wird von der Seite mit HTTP angefragt.
2	Prüfen sie, ob sie ohne Eingabe eines Passworts auf die Administration zugreifen können.	Man muss sich mindestens mit einem Passwort bei der Administration anmelden.	Bestanden
3	Greifen sie nun mittels explizitem setzen von HTTPS auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser. Die Seite ist über HTTPS verfügbar.	Bestanden
4	Prüfen sie, ob sie ohne Eingabe eines Passworts auf die Administration zugreifen können.	Man muss sich mindestens mit einem Passwort bei der Administration anmelden.	Bestanden
5	Geben sie nun den korrekten Benutzer-Namen und ein NICHT korrektes Passwort ein.	Authentisierung fehlgeschlagen.	Bestanden
6	Geben sie nun einen NICHT korrekten Benutzer-Namen und das korrekte Passwort ein.	Authentisierung fehlgeschlagen.	Bestanden
7	Geben sie nun den korrekten Benutzer-Namen und das korrekte Passwort ein.	Authentisierung erfolgreich. Zugriff auf die Administration ist erst jetzt möglich.	Bestanden



### 4.6.2.3 Android App

Als Ergänzung zu den Testfällen der Studienarbeit sind folgend die Testfälle zum Überprüfen der User Stories dieser Arbeit beschrieben.

#### 4.6.2.3.1 Testfall zu User Story: Usability-Fehler beheben

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Schalten Sie jegliche Internetverbindungs-möglichkeiten (WLAN, mobile Daten, ...) ab.  Löschen sie alle Daten der Android App und starten sie die App.	Der Disclaimer Dialog wird angezeigt.	Bestanden
2	Probieren sie den Disclaimer Dialog zu umgehen, indem sie auf den Bereich ausserhalb des Dialogs auf dem Display drücken.	Disclaimer Dialog kann nicht umgangen werden.	Bestanden
3	Probieren sie den Disclaimer Dialog zu umgehen, indem sie den Back-Button auf ihrem Gerät drücken.	Disclaimer Dialog kann nicht umgangen werden.	Bestanden
4	Akzeptieren sie den Disclaimer Dialog.	Es erscheint eine Fehlermeldung, dass eine initiale Internetverbindung benötigt wird.	Bestanden
5	Probieren sie den Internetverbindungs-Dialog zu umgehen, indem sie auf den Bereich ausserhalb des Dialogs auf dem Display drücken.	Internetverbindungs-Dialog kann nicht umgangen werden.	Bestanden
6	Probieren sie den Internetverbindungs-Dialog zu umgehen, indem sie den Back-Button auf ihrem Gerät drücken.	Internetverbindungs-Dialog kann nicht umgangen werden.	Bestanden

#### 4.6.2.3.2 Testfall zu User Story: APK auf Windows laufen lassen

Diese User Story kann nicht in Form eines Testfalls getestet werden. Die Anleitung wurde von den Praxispartnern abgenommen, daher ist sie für uns abgeschlossen.

#### 4.6.2.3.3 Testfall zu User Story: Deployment-Script erstellen

Diese User Story kann nicht in Form eines Testfalls getestet werden. Dieses Skript haben wir jeweils dazu verwendet, um den Stand der App nach Umsetzung einer User Story automatisiert auf Google Drive zu laden.

#### 4.6.2.3.4 Testfall zu User Story: Umstellung des Fragebogens auf den REST-Service

##### Vorbedingungen:

- Alle Daten der Android App wurden gelöscht.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Fügen sie eine neue Frage im Motivprofil-Fragebogen hinzu und verschieben sie diese, sodass sie nicht mehr an letzter Stelle erscheint. Ändern sie zudem den Text einer vorgegebenen Frage geringfügig.	Eine neue Frage im Motivprofil-Fragebogen wurde über die Administration hinzugefügt und eine bestehende Frage wurde geändert.	Bestanden
3	Starten sie die App, akzeptieren sie den Disclaimer und geben sie die soziodemografischen Daten ein.	System zeigt den Motivprofil-Fragebogen an.	Bestanden
4	Stellen sie in der App sicher, dass die in Schritt 2 erstellte Frage an korrekter Position im Fragebogen angezeigt wird. Stellen sie zudem sicher, dass die in Schritt 2 geänderte Frage korrekt in der App erscheint.	Die Synchronisierung des Motivprofil-Fragebogens erfolgt über den REST-Service der Administration.	Bestanden
5	Füllen sie den gesamten Motivprofil-Fragebogen aus, schauen sie sich ihr Motivprofil an und schliessen sie die Android App.	Motivprofil wurde erstellt.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
6	Löschen sie die in Schritt 2 erstellte Frage aus dem Motivprofil-Fragebogen über die Administration. Bearbeiten sie die in Schritt 2 bearbeitete Frage, sodass sie wieder ihrer Ursprünglichen Form entspricht.	Frage aus dem Motivprofil-Fragebogen gelöscht und die bestehende Frage wieder zurückgesetzt.	Bestanden
7	Starten sie die Android App erneut, öffnen sie den Motivprofil-Fragebogen und stellen sie sicher, dass die in Schritt 2 erstellte Frage nicht mehr erscheint und die bearbeitete Frage korrekt erscheint.	Die Synchronisierung des Motivprofil-Fragebogens erfolgt über den REST-Service der Administration.	Bestanden

#### 4.6.2.3.5 Testfall zu User Story: Umstellung der restlichen Ressourcen auf den REST Service

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Fügen sie einen neuen Eintrag in der Rating-Tabelle über die Administration ein, welche für alle Motivprofil Werte den Korrekturwert 1.0 aufweist und notieren sie sich den Faktorwert dieses Eintrags.	Neuen Eintrag in der Rating-Tabelle über die Administration hinzugefügt.	Bestanden
3	Fügen sie einen neuen Entscheidungsfaktor über die Administration hinzu und wählen sie als Mapping für alle Motive den in Schritt 2 erstellten Faktorwert.	Neuen Entscheidungsfaktor über die Administration erstellt.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
4	Fügen sie eine neue Entscheidungssituation mit mindestens 2 vorgegebenen Entscheidungsfaktoren und dem in Schritt 3 erstellten Entscheidungsfaktor hinzu.	Neue Entscheidungssituation über die Administration erstellt.	Bestanden
5	Starten sie die Android App neu.	Startseite der Android App wird angezeigt.	Bestanden
6	Wählen sie die Option, um eine Entscheidung zu fällen und stellen sie sicher, dass die in Schritt 4 erstellte Entscheidungssituation zur Auswahl erscheint.	Die Synchronisierung der Entscheidungssituationen erfolgt über den REST-Service der Administration.	Bestanden
7	Wählen sie die in Schritt 4 erstellte Entscheidungssituation aus und stellen sie sicher, dass alle Faktoren, die in Schritt 4 definiert wurden, angezeigt werden (insbesondere den in Schritt 3 erfassten Entscheidungsfaktor). Stufen sie dabei den in Schritt 3 erstellten Faktor als orange und alle weiteren Faktoren als grün ein.	Die Synchronisierung der Entscheidungsfaktoren erfolgt über den REST-Service der Administration.	Bestanden
8	Führen sie die Entscheidungssituation zu Ende und überprüfen sie, dass die Handlungsempfehlung negativ ist.	Die Synchronisierung der Rating-Tabelle erfolgt über den REST-Service der Administration.	Bestanden
9	Öffnen sie ihr Motivprofil und notieren sie sich ihren mit dem Durchschnittswert verglichenen Wert für das Motiv "force".	Durchschnittsmässige Einstufung für Motiv "force" eingesehen.	Bestanden
10	Bearbeiten sie über die Administration die Beschreibung des in Schritt 9 ermittelten Werts für das Motiv "force" geringfügig.	Beschreibung für Motiv "force" über die Administration geändert.	Bestanden
11	Schliessen sie die Android App und starten sie sie erneut.	Startseite der App wird angezeigt.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
12	Öffnen sie ihr Motivprofil und lassen sie sich das Motiv "force" anzeigen. Überprüfen sie, dass die Beschreibung ihrer Änderung aus Schritt 10 entspricht.	Die Synchronisierung der Motivprofil-Spezifikationen erfolgt über den REST-Service der Administration.	Bestanden
13	Setzen sie die Administration wieder auf den ursprünglichen Zustand zurück: Löschen sie die Entscheidungssituation aus Schritt 4, den Faktor aus Schritt 3, den Rating-Tabellen-Eintrag aus Schritt 2 und setzen sie die Motivprofil-Spezifikation aus 10 zurück.	Administration wieder zurückgesetzt.	Bestanden

#### 4.6.2.3.6 Testfall zu User Story: Upload der Benutzer-Daten an den REST-Service

Diese User Story wurde bereits über die in den Testfällen "Soziodemografische Daten erfassen", "Soziodemografische Daten bearbeiten", "Motivprofil erstellen" und "Motivprofil bearbeiten" beschriebenen Schritte getestet.

#### 4.6.2.3.7 Testfall zu User Story: Anpassen der Mehrsprachigkeit

Diese User Story kann mit der aktuellen App Version nicht getestet werden, da die Synchronisierungssprache auf Englisch fixiert wurde. Der betroffene Code wurde aber mit Unit Tests abgedeckt.

#### 4.6.2.3.8 Testfall zu User Story: Synchronisierungssprache auf Englisch fixieren

##### Vorbedingungen:

- Alle Daten der Android App wurden gelöscht.
- Die Systemsprache des Android Geräts ist auf Deutsch eingestellt.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
2	Fügen sie einen neuen Entscheidungsfaktor, eine neue Entscheidungssituation und eine Frage im Motivprofil-Fragebogen jeweils für die neue Sprache "de" hinzu.	Die neuen Einträge für die Sprache "de" wurden erfolgreich über die Administration erstellt.	Bestanden
3	Fügen sie eine neue Sprachdefinition "de" für die Motivprofil Spezifikationen hinzu.	Neue Sprachdefinition "de" für die Motivprofil Spezifikationen über die Administration erstellt.	Bestanden
4	Starten sie die App, akzeptieren sie den Disclaimer und geben sie die soziodemografischen Daten ein.	Motivprofil-Fragebogen wird angezeigt. Die Einträge sind ausschliesslich in Englisch. Die in Schritt 2 erstellte Frage erscheint nicht im Fragebogen.	Bestanden
5	Füllen sie den Fragebogen aus und schauen sie sich die Beschreibungstexte ihres Motivprofils an.	Die Texte der Motivprofil Beschreibungen sind ausschliesslich in Englisch. Die in Schritt 3 erstellte Definition wird nicht verwendet.	Bestanden
6	Wählen sie die Option zum Fällen einer Entscheidung. Schauen sie sich die Entscheidungssituationen an.	Die Entscheidungssituationen sind ausschliesslich in Englisch. Die in Schritt 2 erstellte Entscheidungssituation wird nicht angezeigt.	Bestanden
7	Wählen sie eine Entscheidungssituation aus und führen sie diese aus. Achten sie dabei auf die Sprache der Entscheidungsfaktoren.	Die Entscheidungsfaktoren sind ausschliesslich in Englisch vorhanden.	Bestanden

#### 4.6.2.3.9 Testfall zu User Story: Bezug zu Panter AG löschen

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Die Startseite der App wird angezeigt.	Bestanden
2	Öffnen sie die "About" Ansicht in der Android App und überprüfen sie das Copyright.	Das Copyright gehört der Forventis. Die Panter AG wird nicht angezeigt.	Bestanden

#### 4.6.2.3.10 Testfall zu User Story: Upgrade auf API Level 21 (Android 5.0)

##### Vorbedingungen:

- Ein zusätzliches Android Gerät mit einer Android Version < 5.0 (kleiner als API Level 21) ist vorhanden.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Versuchen sie die Android App auf dem Gerät mit der Android Version < 5.0 zu installieren.	Installation nicht möglich.	Bestanden
2	Stellen sie sicher, dass die App auf einem Gerät mit mindestens Android Version 5.0 lauffähig ist.	Android App läuft auf Geräten mit Android Version 5.0 oder höher.	Bestanden

#### 4.6.2.3.11 Testfall zu User Story: Upgrade auf Material Theme der Android App

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Startseite der App wird angezeigt.	Bestanden
2	Stellen sie sicher, dass die Action Bar der App, sowie die Buttons auf der Startseite im Material Design erscheinen (mit Welleneffekt auf Buttons).	Das Design der App liegt im Material Design vor.	Bestanden
3	Öffnen sie ihr Motivprofil und prüfen sie dabei den Dialog und die ListView (mit Welleneffekt).	Das Design der App liegt im Material Design vor.	Bestanden
4	Öffnen sie die Ansicht ihrer persönlichen Informationen und prüfen sie dabei die ListView (mit Welleneffekt).	Das Design der App liegt im Material Design vor.	Bestanden
5	Öffnen sie den Motivprofil-Fragebogen und prüfen sie dabei das Aussehen dieser Ansicht.	Das Design der App liegt im Material Design vor.	Bestanden
6	Wählen sie die Option zum Füllen einer Entscheidung und prüfen sie dabei die ListView der Entscheidungssituationen (mit Welleneffekt).	Das Design der App liegt im Material Design vor.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
7	Wählen sie eine Entscheidungssituation aus und prüfen sie die Ansicht der Ansicht zur Faktoreinstufung (Buttons mit Welleneffekt).	Das Design der App liegt im Material Design vor.	Bestanden
8	Führen sie die Entscheidungssituation durch und überprüfen sie die Resultat Ansicht (ListViews mit Welleneffekt).	Das Design der App liegt im Material Design vor.	Bestanden

#### 4.6.2.3.12 Testfall zu Bug: Entscheidungssituation ohne Faktoren

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Greifen sie auf die Administration über ihren Browser zu.	Die Administration erscheint im Browser.	Bestanden
2	Fügen sie einen neuen Entscheidungsfaktor hinzu.	Neuen Entscheidungsfaktor über die Administration erstellt.	Bestanden
3	Fügen sie eine neue Entscheidungssituation hinzu, welche den Faktor aus Schritt 2 benutzt.	Neue Entscheidungssituation über die Administration erstellt.	Bestanden
4	Starten sie die Android App neu und wählen sie die Option zum Fällen einer Entscheidung.	Die vorgegebenen Entscheidungssituationen werden aufgelistet. Die in Schritt 3 erstellte Entscheidungssituation wird angezeigt.	Bestanden
5	Führen sie die in Schritt 3 erstellte Entscheidungssituation aus und stellen sie dabei sicher, dass der in Schritt 2 erstellte Entscheidungsfaktor zur Einstufung angezeigt wird.	Die Entscheidungssituation wird mit dem definierten Entscheidungsfaktor korrekt angezeigt und kann ausgeführt werden.	Bestanden
6	Löschen sie nun den in Schritt 2 erstellten Entscheidungsfaktor in der Administration.	Entscheidungsfaktor über die Administration gelöscht.	Bestanden



Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
7	Schliessen sie die Android App und starten sie die App erneut. Wählen sie die Option zum Fällen einer Entscheidung.	Die vorgegebenen Entscheidungssituationen werden aufgelistet. Die in Schritt 3 erstellte Entscheidungssituation wird NICHT angezeigt.	Bestanden
8	Löschen sie die in Schritt 3 erstellte Entscheidungssituation über die Administration.	Entscheidungssituation über die Administration gelöscht.	Bestanden
9	Schliessen sie die Android App und starten sie die App erneut. Wählen sie die Option zum Fällen einer Entscheidung.	Die vorgegebenen Entscheidungssituationen werden aufgelistet. Die in Schritt 3 erstellte Entscheidungssituation wird NICHT angezeigt.	Bestanden

#### 4.6.2.3.13 Testfall zu Änderung: Positionierung der Knöpfe bei Faktorbewertung

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Die Startseite der App wird angezeigt.	Bestanden
2	Wählen sie die Option zum Fällen einer Entscheidung und wählen sie eine Entscheidungssituation aus.	Ansicht zum Einstufen des ersten Entscheidungsfaktors wird angezeigt.	Bestanden
3	Achten sie auf die Positionen der Knöpfe "NO PROBLEM", "NOT IDEAL", "PROBLEM" und "NOT RELEVANT", während dem sie mehrere Faktoren einstufen.	Die Positionen der Knöpfe bleiben an festen Positionen, für unterschiedliche Faktoren.	Bestanden
4	Wählen sie die Entscheidungssituation "Should I assume an executive office in my club?". Achten sie dabei auf die Positionierung der Knöpfe bei Faktoren, dessen Beschreibungen über mehrere Zeilen dargestellt werden.	Die Positionen der Knöpfe bleiben an festen Positionen, für unterschiedliche Faktoren.	Bestanden

#### 4.6.2.3.14 Testfall zu Änderung: Soziodemografische Daten direkt ersichtlich

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Startseite der App wird angezeigt.	Bestanden
2	Öffnen sie die Ansicht ihrer persönlichen Informationen.	In der Übersichtsansicht der persönlichen Informationen werden direkt die eingegebenen Daten angezeigt.	Bestanden
3	Ändern sie einige Angaben ihrer persönlichen Informationen.	Die geänderten Angaben werden direkt in der Übersichtsansicht angezeigt.	Bestanden

#### 4.6.2.3.15 Testfall zu Änderung: Hervorheben nicht beantworteter Fragen im Fragebogen

##### Vorbedingungen:

- Alle Daten der Android App wurden gelöscht.
- Das Gerät hat eine Internetverbindungsmöglichkeit eingeschaltet.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App, akzeptieren sie den Disclaimer und geben sie die soziodemografischen Daten ein.	Der Motivprofil-Fragebogen wird angezeigt.	Bestanden
2	Beantworten sie einige Fragen des Fragebogens, lassen sie aber mindestens 2 Fragen unbeantwortet. Versuchen sie die Eingaben zu speichern.	Es wird eine Fehlermeldung angezeigt, dass nicht alle Fragen beantwortet wurden.	Bestanden
3	Suchen sie die nicht beantworteten Fragen im Fragebogen und prüfen sie, dass diese mit einem pinken Rahmen dargestellt werden.	Die nicht beantworteten Fragen werden mit einem pinken Rahmen versehen.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
4	Beantworten sie mindestens eine weitere Frage, lassen sie aber mindestens eine Frage unbeantwortet. Prüfen sie, dass die beantwortete Frage KEINEN pinken Rahmen, die unbeantwortete Frage aber weiterhin einen pinken Rahmen besitzt.	Die nicht beantworteten Fragen werden mit einem pinken Rahmen versehen.	Bestanden

#### 4.6.2.3.16 Testfall zu User Story: Splash Screen für Android App

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Die Startseite der App wird angezeigt.	Bestanden
2	Prüfen sie, dass auf der Startseite der App ein Einführungstext der App vorhanden ist.	Einführungstext der App ist auf der Startseite verfügbar.	Bestanden

#### 4.6.2.3.17 Testfall zu User Story: Entscheidungs-Historie

##### Vorbedingungen:

- Disclaimer akzeptiert, soziodemografische Daten eingegeben und Motivprofil erstellt.

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
1	Starten sie die Android App.	Die Startseite der App wird angezeigt.	Bestanden
2	Führen sie die Entscheidungssituation "Should I assume an executive office in my club?" mit lauter grün-Bewertungen durch.	Es wird eine positive Handlungsempfehlung angezeigt.	Bestanden
3	Führen sie die Entscheidungssituation "Shall I marry?" mit lauter grün-Bewertungen durch.	Es wird eine positive Handlungsempfehlung angezeigt.	Bestanden

Schritt	Beschreibung	Soll-Resultat	Ist-Resultat
4	Führen sie die Entscheidungssituation "Should I assume an executive office in my club?" mit lauter rot-Bewertungen durch.	Es wird eine negative Handlungsempfehlung angezeigt.	Bestanden
5	Kehren sie zur Startseite der App zurück und öffnen sie ihre Entscheidungs-Historie.	<p>Es wird die Entscheidungs-Historie wie folgt angezeigt:</p> <ul style="list-style-type: none"> <li>• Oberster Eintrag: "Should I assume an executive office in my club?" mit negativer Handlungsempfehlung.</li> <li>• Zweiter Eintrag: "Shall I marry?" mit positiver Handlungsempfehlung.</li> <li>• Dritter Eintrag: "Should I assume an executive office in my club?" mit positiver Handlungsempfehlung.</li> </ul> <p>Als Ausführungsdatum ist bei den 3 Einträgen das heutige Datum gesetzt.</p>	Bestanden

#### 4.6.2.4 Testfall zu User Story: Petras Geschichte & Ablaufdiagramme

Diese User Story kann nicht in Form eines Testfalls getestet werden. Die Geschichte mit Erklärungen und Ablaufdiagrammen wurde von den Praxispartnern abgenommen, daher ist sie für uns abgeschlossen.

#### 4.6.2.5 Testfall zu User Story: Demo Anleitung zu Petras Geschichte

Diese User Story kann nicht in Form eines Testfalls getestet werden. Die Anleitung wurde von den Praxispartnern abgenommen, daher ist sie für uns abgeschlossen.

#### 4.6.2.6 Testfall zu User Story: Anleitung für die Weiterentwicklung von Decisio Front- und Backend

Diese User Story kann nicht in Form eines Testfalls getestet werden. Die Anleitung wurde von den Praxispartnern abgenommen, daher ist sie für uns abgeschlossen.

#### 4.6.2.7 Testfall zu Spike: System mit fehlenden Daten testen und Probleme beheben

Zu dieser User Story wurde bereits eine entsprechende Dokumentation erstellt (siehe Abschnitt 4.4). Deshalb ist für diese User Story kein Testfall nötig.

### 4.6.3 Verbesserungsmöglichkeiten

- Bei der Rating-Table werden bei Timestamp 0 auch gelöschte Einträge gesendet. Dies ist aber nicht kritisch, da das Android Gerät einfach unnütze Daten erhält, diese jedoch nicht weiter beachtet.
- Greift man explizit mit HTTP auf die Administration zu, erscheint die Passwort-Abfrage vom Host `http://...`. D.h. das Passwort wird im Klartext übertragen und erst nach dieser Eingabe wird man auf HTTPS umgeleitet.

## 4.7 Unit Tests

### 4.7.1 Service

Zum automatisierten Testen des RESTful Web Service wird JUnit [23] eingesetzt. Um die Datenbank-Operationen zu testen, erstellten wir Tests mit Fakes und solche die auf einer Test-Datenbank arbeiten. Diesen Ansatz wählten wir, damit das effektive Resultat von den Operationen getestet werden kann. Z.B. das Verschieben nach dem Löschen einer Frage des Motivprofil-Fragebogens. Auch sagt uns das Vorgehen des Android Frameworks zu, weshalb wir diese Funktionalität nachbauten. Wenn eine Testklasse die Test-Datenbank verwendet, kann die Testklasse in der `tearDown`-Methode das leeren der Tabelle(n) anordnen. Siehe auch Abschnitt 4.7.3.

Da die Tests mit einer realen Datenbank zeitintensiver sind, kam das Bedürfnis auf, die Laufzeit der Tests zu reduzieren. Um dies zu realisieren wurden die Tests in Kategorien aufgeteilt. So kann jede Kategorie separat getestet werden, d.h. die Testklassen jeder Kategorie können separat ausgeführt werden. In folgender Tabelle sind die erstellten Kategorien aufgelistet:

Kategorie	Beschreibung	Anzahl Tests in der Kategorie
DatabaseTests	Tests auf der realen Test-Datenbank, hohe Laufzeit	146
FastTests	Tests mit Fakes, geringe Laufzeit	183
MarshalTests	Marshal-/Unmarshal-Tests, mittlere Laufzeit	51
ThroughputTests	Durchsatz-/Antwortzeiten-Tests, hohe Laufzeit	1

**Tabelle 30: Kategorien und deren Anzahl JUnit-Tests**

Die Testabdeckung beträgt 94% aller Klassen und 92% aller Zeilen. Die Klasse `Main` erstellt die Server-Instanz von Jetty. Die Klasse `HttpFilter` leitet HTTP Anfragen nach HTTPS weiter. Diese beiden Klassen werden nicht getestet.

Coverage Summary for Package 'decisio': 94% classes, 92% lines covered			
Element	Class, %	Method, %	Line, %
database	100% (10/10)	98% (85/86)	99% (873/878)
domain	100% (18/18)	100% (133/133)	100% (197/197)
Decisions	100% (1/1)	90% (9/10)	45% (21/46)
Factors	100% (1/1)	90% (9/10)	97% (46/47)
HttpFilter	0% (0/1)	0% (0/4)	0% (0/14)
Main	0% (0/1)	0% (0/2)	0% (0/24)
MotiveProfileSpe...	100% (1/1)	88% (8/9)	97% (39/40)
Questionnaire	100% (1/1)	90% (10/11)	47% (25/53)
QuestionnaireAn...	100% (1/1)	66% (2/3)	87% (7/8)
RatingTable	100% (1/1)	87% (7/8)	97% (37/38)
RestException	100% (1/1)	100% (1/1)	100% (2/2)
Sociodemograph...	100% (1/1)	66% (2/3)	87% (7/8)

Abbildung 35: Testabdeckung der JUnit-Tests

#### 4.7.2 Administration

Für die Unit-Tests der Administration wird Karma [24] verwendet. Mit Karma lassen sich die Services und Controller einer AngularJS-Applikation testen. Anstelle von realen Zugriffen auf den REST-Service, wird der Service `$httpBackend` [25] von AngularJS [1] verwendet. Bei diesem Fake lassen sich die erwarteten HTTP Requests und deren Antworten definieren. Falls ein Request abgesetzt wird, der nicht erwartet wurde oder nicht der Erwartung entspricht, schlägt der entsprechende Test fehl.

Total haben wir 182 Karma-Tests erstellt. Bei den Controllern erreichen wir dadurch eine Testabdeckung der Zeilen von 94%. Bei den Services beträgt die Abdeckung 98%.

Coverage Summary for 'admin': 100% files, 93% lines covered	
Element	Statistics, %
controller	100% files, 94% lines covered
img	
node_modules	
services	100% files, 98% lines covered
test	
views	
app.js	80% lines covered
index.html	
karma.conf.js	
styles.css	

Abbildung 36: Testabdeckung der Karma-Tests

#### 4.7.3 Android App

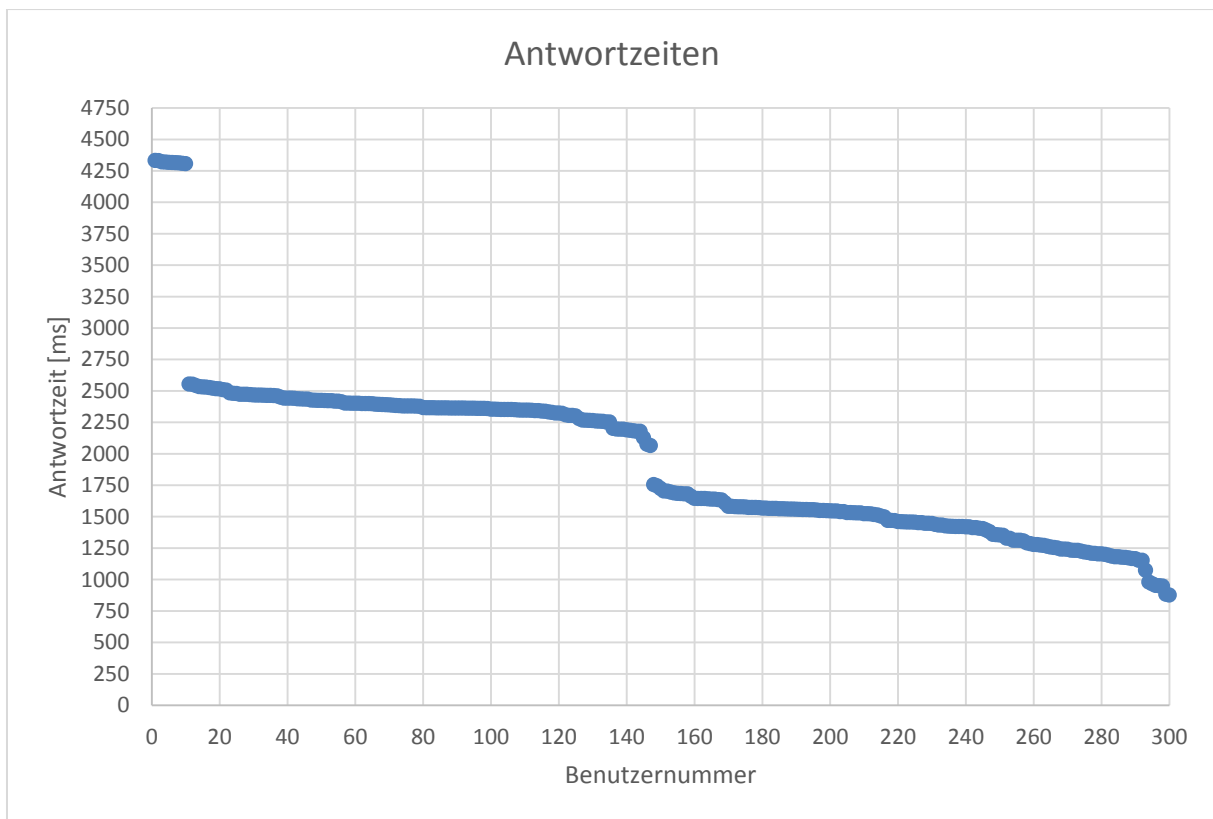
Bei der Android App bestehen Tests aus unserer Vorarbeit [19]. Diese mussten teilweise angepasst oder ausgebaut werden, z.B. aufgrund der neuen Synchronisierung. Neu sind es 261 Unit Tests.

## 4.8 Test der Antwortzeiten

Zum Testen der Antwortzeiten erstellten wir einen Unit Test, der mit insgesamt 500 Threads die verschiedenen Ressourcen zur Synchronisierung abfragt. Dies soll den gleichzeitigen Zugriff von 100 Benutzern simulieren. Diesen Test führten wir zur Messung dreimal aus.

Die Durchschnitts-Antwortzeit liegt bei 1.953 Sekunden. Die schnellste Antwort erfolgte in 876 Millisekunden, die langsamste in 4.332 Sekunden.

Folgendes Diagramm zeigt die Verteilung der Antwortzeiten für die verschiedenen Benutzer:



**Abbildung 37: Verteilung der Antwortzeiten**

Wie zu erkennen ist, betrifft die maximale Antwortzeit lediglich 4%. Des Weiteren können über 50% der Benutzer innerhalb von 1750 Millisekunden bedient werden.



## 5. Schlussfolgerungen

---

### 5.1 Ergebnisse

Durch die gute und enge Zusammenarbeit mit den Praxispartnern entstand eine ansprechende AngularJS Administration mit der sich die zu synchronisierenden Daten bearbeiten lassen. Durch die Verwendung von Angular Material konnte sie zeitgemäss gestaltet werden. Sie konnte aufgrund der Struktur von AngularJS modular entwickelt werden. Die Services und Controller liessen sich jeweils gut und mit wenig Aufwand in die Applikation eingliedern.

Als Ersatz für den SFTP-Server wurde der RESTful Web Service erstellt, der in Jersey implementiert ist. Die App wurde aktualisiert, sodass sie den RESTful Web Service zur Synchronisierung benutzt. Somit konnte der SFTP-Server abgelöst werden. Ein weiterer Vorteil ist die partielle Synchronisierung. Somit benötigt eine Synchronisierung der Änderungen signifikant weniger Zeit.

Die Umstellung der App auf das Material Theme führte zu einer noch ansprechenderen App. Sie ist somit für die Zukunft gut gerüstet und besitzt kein veraltetes Design. Durch die kleinen Verbesserungen konnten die Fehler der Vorarbeit behoben werden. Zusätzlich konnte die Funktionalität geringfügig erweitert werden.

### 5.2 Schlussfolgerungen

Als Prozessmodell Scrum einzusetzen war sicherlich die richtige Entscheidung. Auch sinnvoll war es, dass wir uns seriös in AngularJS einarbeiteten. Somit war die Administration von Beginn an professionell entwickelt. Es kam nie die Situation auf, dass man ein "Gebastel" reparieren musste.

Auf der Seite des RESTful Web Service waren wir am Anfang viel zu optimistisch. Die eigentliche Implementierung ging nach einer Einarbeitungszeit zügig vorwärts. Jedoch blieb der Zeitbedarf für das Erstellen der Fakes für die Tests und der Unit Tests selbst ziemlich konstant. Somit waren wir bis spät in der Projektzeit noch mit dem Service beschäftigt und konnten uns noch nicht vollständig der App widmen. Deshalb blieb der Ausbau des Funktionsumfangs auch so klein.

In der Vorarbeit wurde gewünscht, die Möglichkeit mehrerer Sprachen einzubauen. Dies hatte zur Folge, dass alles für mehrere Sprachen vorbereitet war. In dieser Arbeit wollte man diese Möglichkeiten nutzen. Dies führte dazu, dass wir in der jetzigen Situation mehrere mögliche Fallback-Szenarien abdecken. Jedoch gelang dies relativ autonom. Bei den Sprint Review Sitzungen mit den Praxispartnern haben wir diverse Mock Screens mit der Sprachfunktionalität vorgeführt, trotzdem wurde nie konkret darüber diskutiert. Die jetzige Lösung für die Mehrsprachigkeit ist für die Praxispartner aber generell nicht zufriedenstellend. Es fehlt eine zentrale Verwaltung sowie Kopiermöglichkeiten von erfassten Daten, die nicht sprach-spezifisch sind. Daher floss zu viel Zeit in die Mehrsprachigkeit und die Problematik ist dennoch nicht gelöst.

Im Systemtest ist ein Fehler im Zugriffsschutz der Administration aufgetaucht: Das Zugangspasswort wird dabei bei einem explizitem Zugriff mit HTTP im Klartext übertragen und die Umleitung auf HTTPS erfolgt erst nach der Authentisierung. Dieser Fehler kann aber mit einem Workaround umgangen werden, indem direkt die Administration mit HTTPS aufgerufen wird.

## 5.3 Ausblick

Für eine weitere Fortsetzung des Projekts ist eine solide, professionelle Grundlage vorhanden. Die Administration verwendet aktuelle Komponenten und ist zeitgemäss gestaltet. Ihr muss lediglich noch die Funktionalität zur Anzeige der Benutzerdaten ergänzt werden. Der Service erfüllt die Anforderungen der jetzigen Bedürfnisse und wird in naher Zukunft keine Aufmerksamkeit benötigen.

Als nächstes sollten drei Dinge in Angriff genommen werden: Zum einen fehlt bei der Android App noch immer die Möglichkeit als Benutzer Entscheidungssituationen zu erfassen. Des Weiteren sollten mehr vorgegebene Entscheidungssituationen erstellt werden. Als Drittes muss die Situation mit der Mehrsprachigkeit von Grund auf neu in Angriff genommen werden.

Diese Massnahmen sollten aber mit den erstellten Anleitungen und Dokumentationen für die Praxispartner gut realisierbar sein.

## 6. Glossar

Begriff	Beschreibung
<b>Administration</b>	AngularJS-Applikation, welche das Front-End zum Back-End bildet.
<b>Android App</b>	Die clientseitige Android App, die den Benutzer unterstützt Entscheidungen zu fällen. (Das eigentliche Produkt.)
<b>Back-End-Applikation</b>	Die Applikation auf dem Back-End-Server, zur Bearbeitung der vorhandenen Entscheidungssituationen, des Motivprofil-Fragebogens, usw.
<b>Client-Applikation</b>	Die clientseitige Android App.
<b>CRUD</b>	Bearbeiten: Create, Read, Update und Delete von einzelnen Einträgen
<b>Entscheidungsfaktor</b>	Eine Entscheidung wird anhand mehrerer Faktoren gefällt. Ein Faktor ist eine messbare Eigenschaft des Benutzers (z.B. besitzt er viel oder wenig Geld)
<b>Entscheidungsregel</b>	Anhand dieser Regel wird die Entscheidung gefällt. Sie behandelt die als orange eingestufteten Entscheidungsfaktoren. Liegt die Summe der Gewichtungen dieser Faktoren über den definierten Schwellwerten, wird die Entscheidung "positiv mit Warnung" oder "negativ".
<b>Faktor</b>	Siehe Entscheidungsfaktor.
<b>Gewichtung</b>	Ein Entscheidungsfaktor besitzt pro Motivationssystem einen Korrekturbetrag. Diese werden summiert und zu 1 addiert und fließen so als Gewichtung für den Entscheidungsfaktor in die definitive Entscheidungsregel ein.
<b>Korrekturbetrag</b>	Der Korrekturbetrag wird aus der Rating-Tabelle bestimmt. Dieser Betrag bezieht sich auf ein Motivationssystem.
<b>Rating-Tabelle</b>	Die Rating-Tabelle bestimmt die einzelnen Korrekturbeträge.
<b>Mapping</b>	Als Mapping wird die Zugehörigkeit der Entscheidungsfaktoren zu Entscheidungssituationen bezeichnet. Ein Entscheidungsfaktor wirkt sich auf eine oder mehrere Entscheidungssituationen aus.
<b>Motivprofil</b>	Das Motivprofil definiert, wie sich der Benutzer in den unterschiedlichen Motivationssystemen einschätzt. Es wird aus dem ausgefüllten Fragebogen berechnet.
<b>Motivationssystem</b>	Die fünf Motivationssysteme gemäss dem Zürcher Modell der sozialen Motivation [26]: Soziale Sicherheit, Erregung, Macht, Geltung, Leistung. (social, arousal, force, prestige, performance)
<b>Motiv</b>	Siehe Motivationssystem.
<b>Rating</b>	Als Rating werden die vom Benutzer angegebenen Werte im Motivprofil-Fragebogen bezeichnet.
<b>Schwellwerte der Entscheidungsregel</b>	Diese zwei Werte definieren, ab wann eine Entscheidung "positiv mit Warnung" oder "negativ" wird. Siehe auch Entscheidungsregel.
<b>Spike</b>	Ein Begriff aus dem Scrum Umfeld. Damit gemeint ist eine Arbeit, für die nur maximal die jeweils für den Spike definierte Arbeitszeit aufgewendet wird.

Begriff	Beschreibung
<b>vorgegebener Entscheidungs-faktor</b>	Diese Entscheidungsfaktoren wurden vordefiniert auf dem Back-End-Server abgelegt und werden von den Client-Applikationen heruntergeladen.
<b>vorgegebene Entscheidungs-situation</b>	Diese Entscheidungssituationen wurden vordefiniert auf dem Back-End-Server abgelegt und werden von den Client-Applikationen heruntergeladen.

## 7. Literaturverzeichnis

---

- [1] «AngularJS Framework,» Google Inc., [Online]. Available: <https://angularjs.org/#>. [Zugriff am 12. Juni 2015].
- [2] «Angular Material Design,» Google Inc., [Online]. Available: <https://material.angularjs.org/#/>. [Zugriff am 12. Juni 2015].
- [3] «Heroku,» Heroku, Inc., [Online]. Available: <https://www.heroku.com/>. [Zugriff am 12. Juni 2015].
- [4] «Angular Material: Cards,» Google Inc., [Online]. Available: <https://material.angularjs.org/#/demo/material.components.card>. [Zugriff am 12. Juni 2015].
- [5] «Angular Material: Tabs,» Google Inc., [Online]. Available: <https://material.angularjs.org/#/demo/material.components.tabs>. [Zugriff am 12. Juni 2015].
- [6] «Angular Material: List,» Google Inc., [Online]. Available: <https://material.angularjs.org/#/demo/material.components.list>. [Zugriff am 12. Juni 2015].
- [7] «Angular Material: Demo Chips,» Google Inc., [Online]. Available: <https://material.angularjs.org/#/demo/material.components.chips>. [Zugriff am 12. Juni 2015].
- [8] «Heroku Toolbelt,» Heroku, Inc., [Online]. Available: <https://toolbelt.heroku.com>. [Zugriff am 12. Juni 2015].
- [9] «PostgreSQL: Documentation: 7.4: Connection Pools and Data Sources,» PostgreSQL, [Online]. Available: <http://www.postgresql.org/docs/7.4/static/jdbc-datasource.html>. [Zugriff am 12. Juni 2015].
- [10] M. Fowler, «Richardson Maturity Level,» 18 03 2010. [Online]. Available: <http://martinfowler.com/articles/richardsonMaturityModel.html>. [Zugriff am 12. Juni 2015].
- [11] «Jersey,» Oracle Corporation, [Online]. Available: <https://jersey.java.net/>. [Zugriff am 12. Juni 2015].
- [12] «Java Architecture for XML Binding (JAXB),» Oracle Corporation, [Online]. Available: <http://www.oracle.com/technetwork/articles/javase/index-140168.html>. [Zugriff am 12. Juni 2015].
- [13] «Jetty,» The Eclipse Foundation, [Online]. Available: <http://www.eclipse.org/jetty/>. [Zugriff am 12. Juni 2015].
- [14] «Angular Service \$rootScope,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ng/service/\\$rootScope](https://code.angularjs.org/1.3.15/docs/api/ng/service/$rootScope). [Zugriff am 12 Juni 2015].
- [15] «Angular Provider \$routeProvider,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ngRoute/provider/\\$routeProvider](https://code.angularjs.org/1.3.15/docs/api/ngRoute/provider/$routeProvider). [Zugriff am 12. Juni 2015].
- [16] «Angular Service \$resource,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ngResource/service/\\$resource](https://code.angularjs.org/1.3.15/docs/api/ngResource/service/$resource). [Zugriff am 12. Juni 2015].
- [17] «Angular Service \$http,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ng/service/\\$http](https://code.angularjs.org/1.3.15/docs/api/ng/service/$http). [Zugriff am 12. Juni 2015].

- [18] «Angular Funktion \$watch,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ng/type/\\$rootScope.Scope#\\$watch](https://code.angularjs.org/1.3.15/docs/api/ng/type/$rootScope.Scope#$watch). [Zugriff am 12. Juni 2015].
- [19] R. Horber und S. Kaufmann, «Decisio - App für die optimale Entscheidungsfindung,» *Studienarbeit HS 2014, HSR - Hochschule für Technik Rapperswil*, 19. Dezember 2014.
- [20] «URLConnection (Java Platform),» Oracle Corporation, [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/net/URLConnection.html>. [Zugriff am 12. Juni 2015].
- [21] «Dashboards | Android,» Google Inc., [Online]. Available: <https://developer.android.com/about/dashboards/index.html#Platform>. [Zugriff am 12. Juni 2015].
- [22] «Material design - Google design guidelines,» Google Inc., [Online]. Available: <http://www.google.com/design/spec/>. [Zugriff am 12. Juni 2015].
- [23] «JUnit - About,» JUnit, [Online]. Available: <http://www.junit.org/>. [Zugriff am 12. Juni 2015].
- [24] P. Tarasiewicz und R. Böhm, «Tests mit Karma automatisiert ausführen,» in *AngularJS - Eine praktische Einführung in das JavaScript-Framework*, Heidelberg, dpunkt.verlag GmbH, 2014, pp. 259-267.
- [25] «Angular Service \$httpBackend,» Google Inc., [Online]. Available: [https://code.angularjs.org/1.3.15/docs/api/ngMock/service/\\$httpBackend](https://code.angularjs.org/1.3.15/docs/api/ngMock/service/$httpBackend). [Zugriff am 12. Juni 2015].
- [26] Wikipedia, «Zürcher Modell der sozialen Motivation,» [Online]. Available: [http://de.wikipedia.org/wiki/Z%C3%BCrcher\\_Modell\\_der\\_sozialen\\_Motivation](http://de.wikipedia.org/wiki/Z%C3%BCrcher_Modell_der_sozialen_Motivation). [Zugriff am 12. Juni 2015].

## 7.1 Weitere Literatur

Für "best practice" Umsetzungen unter Android: <http://developer.android.com/guide/index.html>

Für allgemeine Lösungen von Android-Problemen, die nicht unter developer.android.com behandelt wurden: <http://www.stackoverflow.com>

Für die Realisierung der Rating-Tabelle, des Motivprofil-Fragebogens und die vorgegebenen Entscheidungssituationen mit deren Faktoren erhielten wir Arbeitsdokumente der Praxispartner, welche nicht öffentlich zugänglich sind.