ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

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

Konzeption und Realisierung einer konfigurierbaren Plattform zur flexiblen Datenerhebung mittels moderner Web-technologien

Masterarbeit an der Universität Ulm

Vorgelegt von:
David Damaschk
david.damaschk@uni-ulm.de

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

Betreuer:
Johannes Schobel

2016

Fassung 25. April 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Der Einsatz papierbasierter Fragebögen ist heutzutage ein weit verbreitetes Mittel zur Erhebung von Daten in Studien und Umfragen. Jeder Teilnehmer erhält von einem Mitarbeiter einen Fragebogen, der anschließend bearbeitet wird. Nach der Bearbeitung müssen diese Daten für die Analyse digitalisiert werden. Diese Vorgehensweise stellt einen hohen Ressourcenverbrauch dar, insbesondere bei einer großen Anzahl an Studienteilnehmern. Die anschließende Digitalisierung der Antworten ist ebenfalls mit einem hohen Zeitaufwand verbunden. Dabei können während der Übertragung der Antworten diese falsch übertragen und somit Ergebnisse verfälscht werden. Eine Möglichkeit, diesen Problemen entgegenzutreten, wäre der Einsatz eines digitalen Fragebogensystems.

Im Rahmen dieser Arbeit wird eine Plattform zur digitalen Datenerhebung entwickelt. Auf einem Server sollen Fragebögen hochgeladen werden können. Mit einem modernem Web-Client soll es möglich sein, diese Fragebögen an Nutzer zu verteilen und zu bearbeiten. Nach der Bearbeitung dieser Fragebögen werden die Antworten an den Server gesendet und dort persistiert. Komplexe Fragebögen mit einer verzweigten Ausführungslogik sollen dabei unterstützt sowie Benutzer während der Bearbeitung auf unvollständige Abgaben aufmerksam gemacht werden.

Inhaltsverzeichnis

1

Einleitung

Heutzutage werden Umfragen und Studien immer noch mithilfe papierbasierter Fragebögen durchgeführt. Jeder Teilnehmer erhält dazu einen Fragebogen, den er zu bearbeiten hat. Die ausgefüllten Fragebögen werden danach von einem Mitarbeiter eingesammelt und digitalisiert. Anschließend kann mit der Auswertung der Antworten begonnen werden.

Die Verwendung papierbasierter Fragebögen stellt besonders im digitalen Zeitalter einen unnötigen Ressourcenverbrauch dar. Der Zeitaufwand für die Durchführung von Studien und Umfragen gilt ebenfalls nicht zu vernachlässigen. Für die Teilnehmer kostet der Anfahrtsweg und der Rückweg wertvolle Zeit, die sinnvoller genutzt werden könnte. Zudem muss sich immer ein Mitarbeiter Zeit nehmen und vor Ort sein für die Verteilung und Entgegennahme der Fragebögen. Die Digitalisierung der Ergebnisse ist ein weiterer zeitaufwändiger Prozess, bei der Übertragungsfehler auftreten können und somit die

Ergebnisse verfälscht werden. Fragebögen, in denen basierend auf den bisherigen Antworten des Benutzers zusätzliche Fragen beantwortet werden müssen beziehungsweise entfallen, können sehr komplex sein. Dies kann dazu führen, dass Fragebögen unvollständig bearbeitet werden und somit Ergebnisse nutzlos sind und Umfragen oder Studien im schlimmsten Fall wiederholt werden müssen. Die Ergebnisse müssen abschließend archiviert werden, wodurch auf lange Sicht hohe Lagerkosten entstehen.

Eine Möglichkeit, mit der die oben genannten Probleme gelöst werden können, ist die vollständige Bearbeitung von Fragebögen auf digitalen Endgeräten wie zum Beispiel Desktop-PCs. Dazu existieren bereits verschiedene Fragebogensysteme [1]. Diese unterstützen die Bearbeitung von Fragebögen über das Internet. Hier treten jedoch datenschutzrechtliche Bedenken auf, da die Ergebnisse extern gelagert werden und eine sichere Übertragung der Daten im Internet vom Anbieter gegebenenfalls nicht gewährleistet ist. Zusätzlich kann die Erstellung der Fragebögen ein erhöhtes Know-How erfordern. Komplexe Fragebögen, die Entscheidungsverzweigungen besitzen, setzen Programmierkenntnisse voraus [2]. Mediziner, die Fragebögen für ihre Studien erstellen, fehlen oft solche Kenntnisse oft und benötigen daher einen Experten für die Erstellung der Fragebögen. Dies verursacht weitere Kosten und einen hohen Koordinationsaufwand.

Das Projekt QuestionSys [3] des Instituts für Datenbanken und Informationssysteme an der Universität Ulm entwickelt ebenfalls ein digitales Framework zur flexiblen Datenerhebung. Damit soll es möglich sein, Fragebögen einfach zu erstellen, verteilen, bearbeiten, auszuwerten und anschließend zu archivieren. Im Rahmen dieser Arbeit soll dazu ein System entwickelt werden, mit der die Fragebögen über das Internet verteilt und bearbeitet werden können. Während der Bearbeitung der Fragebögen soll auf unvollständige Eingaben hingewiesen werden, um unvollständige Abgaben zu vermeiden. Zudem können Fragebögen über mehrere Seiten verteilt werden und dem Benutzer nur die Fragen dargestellt werden, die für ihn von Relevanz sind. Die Ergebnisse sollen abschließend auf dem System archiviert werden. Die abgegebenen Ergebnisse sollen vertraulich behandelt werden.

1.1 Zielsetzung

Das Ziel dieser Masterarbeit soll es sein, basierend auf den in Kapitel 1 beschriebenen Problemen eine Web-Plattform zu entwickeln. Mit dieser Plattform sollen Fragebögen über das Internet verteilt und bearbeitet werden können. Ein zentraler Server ist für die Speicherung der Ergebnisse und für die Verteilung der Fragebögen zuständig. Die Bearbeitung soll durch einen modernen Web-Client durchgeführt werden. Es sollen Fragebögen mit komplexen Ausführungsregeln unterstützt werden, mit der basierend auf den bisherigen Antworten des Benutzers der weitere Verlauf individuell behandelt werden kann. Die Ergebnisse sollen ausnahmslos verschlüsselt gespeichert werden und der Zugriff so geregelt sein, dass nur berechtigte Personengruppen die Daten einsehen können. Bei Bedarf sollen Fragebögen zusätzlich anonym bearbeitet werden können.

1.2 Aufbau der Arbeit

In Kapitel 1 wird in das Thema der digitalen Fragebogenerhebung und dessen Vorteile eingeführt. In Kapitel 2 wird die Idee der digitalen Datenerhebung diskutiert und die grundsätzlichen Technologien im Web-Bereich erläutert. In Kapitel 3 werden aktuelle Fragebogensysteme analysiert und anschließend die Nutzerrollen des Systems erstellt. Aufbauend auf der Analyse der Systeme werden die funktionalen und nicht-funktionalen Anforderungen und der verwendete Styleguide für das System erläutert. Die Struktur des Fragebogenmodells wird in Kapitel 4 vorgestellt und darauf aufbauend die Struktur des Antwortmodells erstellt. Anschließend wird die Architektur des Systems beschrieben und ein Datenmodell vorgestellt. Zusätzlich wird die Benutzeroberfläche anhand ausgewählter Seiten unter Berücksichtigung der Anforderungen und des Styleguides aus Kapitel 2 diskutiert. In Kapitel 5 werden die eingesetzten Technologien für die Umsetzung des Systems beschrieben und die Umsetzung anhand von Programmierbeispielen vorgestellt. Kapitel 6 fasst die Masterarbeit zusammen und stellt einen Ausblick über mögliche Erweiterungen vor.

2

Grundlagen

In Kapitel 2.1 wird zunächst die Idee der digitalen Datenerhebung durch das Projekt *QuestionSys* [3] vorgestellt. Zusätzlich wird erläutert, welche Aufgabenbereiche des Projekts von den Komponenten des zu entwickelnden Systems übernommen werden. Kapitel 2.2 erläutert wichtige Grundlagen zur Web-Entwicklung.

2.1 QuestionSys

Die Erfassung von Daten mithilfe papierbasierter Fragebögen ist im Kontext wissenschaftlicher Studien selbstverständlich. Die erfassten Daten werden im nächsten Schritt digital übertragen und ausgewertet. Diese Vorgehensweise besitzt jedoch einige Nachteile. Zum einen können Fragebögen unvollständig bearbeitet werden. Zusätzlich können durch den zeitaufwändigen Prozess der Digitalisierung der Daten, Fehler entstehen und

somit Ergebnisse verfälscht werden. Die anschließende Archivierung der Daten kann auf lange Sicht zu Platzproblemen führen [4].

Mit dem Projekt QuestionSys sollen diesen Problemen entgegengewirkt werden. Die Idee ist, dass der Lebenslauf eines Fragebogens vollständig digital abläuft. Dieser Lebenslauf kann in die Bereiche Erstellung, Verteilung, flexible Datenerhebung, Evaluierung und Analyse sowie in die Archivierung und Versionierung der Fragebögen gegliedert werden. Diese Phasen werden in QuestionSys von den darin enthaltenen Komponenten Konfigurator, Server und Client übernommen. Das Zusammenspiel dieser Komponenten ist in Abbildung 2.1 dargestellt. In den nachfolgenden Kapiteln 2.1.1, 2.1.2 und 2.1.3 werden die Aufgaben dieser Komponenten basierend auf dem Lebenszyklus eines Fragebogens vorgestellt.

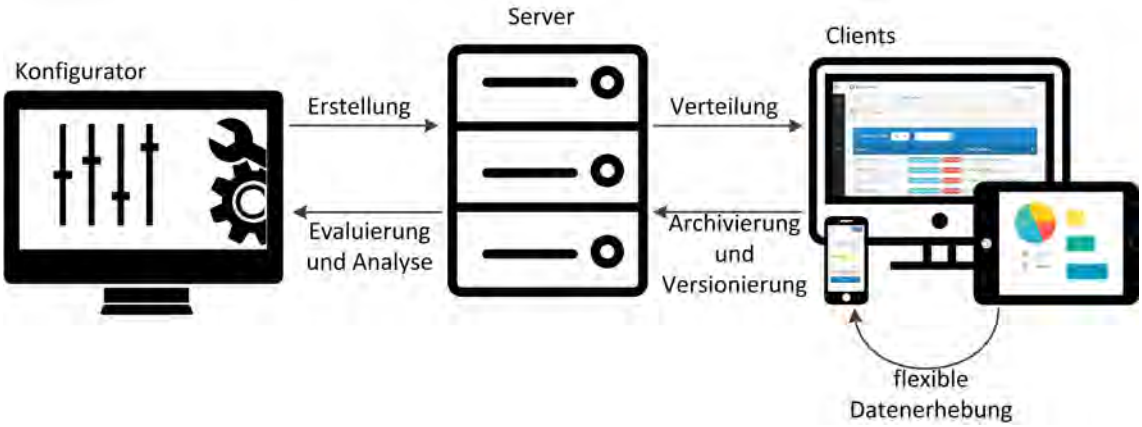

Abbildung 2.1: Übersicht des Fragebogensystems QuestionSys.

2.1.1 Konfigurator

Der Konfigurator ist für die Erstellung des Fragebogenmodells und zukünftig auch für die Evaluierung der Antworten zuständig. Während der Erstellung des Fragebogens kann die Anzahl der Seiten, Fragen und die Ausführungslogik festgelegt werden. Zusätzlich können Abhängigkeiten zwischen den Seiten definiert und mehrere Sprachen unterstützt werden [5]. Eine Exportfunktion ermöglicht es, die erstellten Fragebogenmodell auf den

zu Server zu exportieren. In Zukunft sollen vollständig bearbeitete Fragebögen vom Konfigurator ausgewertet und die Ergebnisse zurück an den Server gesendet werden.

2.1.2 Server

Die Serverseite speichert die exportierten Fragebogenmodelle des Konfigurators sowie die erhobenen Antworten der Nutzer und ist zuständig für die Verteilung der Fragebögen an die Clients. Die Ergebnisse der Fragebögen sollen zukünftig anhand festgelegter Regeln vom Konfigurator evaluiert werden. Ein Teil dieser Abschlussarbeit ist dabei die Entwicklung der Serverseite, mit der Fragebögen hochgeladen, verteilt und archiviert werden könnnen.

2.1.3 Client

Für die Erhebung der Daten können mehrere Clients zum Einsatz kommen. In dieser Arbeit wird dazu neben der Serverkomponente ein moderner Web-Client entwickelt. Mithilfe einer Web-Schnittstelle zum Server werden die Fragebögen an die Nutzer verteilt sowie die Antworten der Nutzer auf der Serverseite persistiert. Der Aufbau der generischen Fragebögen und die Interpretation der Ausführungslogik wird dabei vollständig vom Client durchgeführt.

2.2 Web-Entwicklung

Für die Entwicklung dieser Abschlussarbeit stellt der Browser den Client dar. Dazu werden für die Umsetzung die Technologien JavaScript, HTML5 und CSS3 verwendet. Für die Serverseite kommt die Programmiersprache PHP zum Einsatz.

Hypertext Markup Language (HTML) wird für den inhaltlichen Aufbau einer Webseite im Browser eingesetzt. In der neuesten Version (HTML5) werden verschiedene Multimediainhalte wie zum Beispiel Audio, 2D- und 3D-Grafiken und Video unterstützt. Zusätzlich

bietet es einen lokalen Speicher an, mit der Schlüssel-Werte Paare gespeichert werden können.

JavaScript ist eine Skriptsprache und zuständig für die Anwendungslogik auf der Clientseite. In der aktuellen Version ist JavaScript auch als ECMAScript 6 bekannt und unterstützt unter anderem die Verwendung von Klassen und Modulen. Es unterstützt zudem eine objektorientierte, prozedurale sowie funktionale Programmierung. Die Ausführung der Skripte wird dabei von einem im Webbrowser enthaltenem Interpreter übernommen.

Cascading Style Sheets (CSS) ist eine Gestaltungssprache, die das Aussehen einer Seite mit einem Stylesheet definiert. Durch die Trennung von Inhalten und Gestaltungsvorgaben können Stylesheets wiederverwendet werden. Die neueste Version CSS3 unterstützt dabei unter anderem die Unterscheidung der Bildschirmgrößen durch Media Queries, wodurch die Darstellung von Seiten auf mobilen Endgeräten mit kleineren Bildschirmgrößen angepasst werden kann.

PHP: Hypertext Preprocessor (PHP) ist eine Skriptsprache und wird auf der Serverseite eingesetzt. Es bietet einen integrierten Webserver sowie eine Unterstützung verschiedener Datenbanken wie zum Beispiel MySQL oder SQLite an. Weiterhin unterstützt es die Konzepte der objektorientierten und funktionalen Programmierung.

Im Vergleich zu klassischen Webseiten soll sich die gesamte Anwendungslogik auf der Clientseite befinden. Dafür wird der Client als *Single-page-Application*[1] *(SPA)* [6] implementiert. Das Ziel ist es, mit dem Einsatz einer SPA und den darin verwendeten Technologien JavaScript, HTML und CSS, dem Nutzer das Gefühl zu geben, er verwende eine native Desktop-Anwendung. Bisherige Ansätze wie zum Beispiel Java Applets, Adobe Flash und Microsoft Silverlight setzen die Installation von Plugins voraus. Im folgenden Kapitel 2.2.1 wird die Funktionsweise einer SPA erklärt und mit dem Einsatz traditioneller Webanwendungen verglichen.

[1] zu dt. „Einzelseiten-Webanwendung“

2.2.1 Vergleich klassischer Webanwendungen mit SPA's

In diesem Kapitel wird die Funktionsweise einer klassischen Webanwendung mit der Funktionsweise einer SPA verglichen. Die Umsetzung einer klassischen Webanwendung ist in Abbildung 2.2 dargestellt. Für jeden Aufruf einer HTML-Seite wird eine neue Anfrage an den Server gesendet. Diese Anfragen werden im Server in der Präsentationsschicht durch Controller bearbeitet.

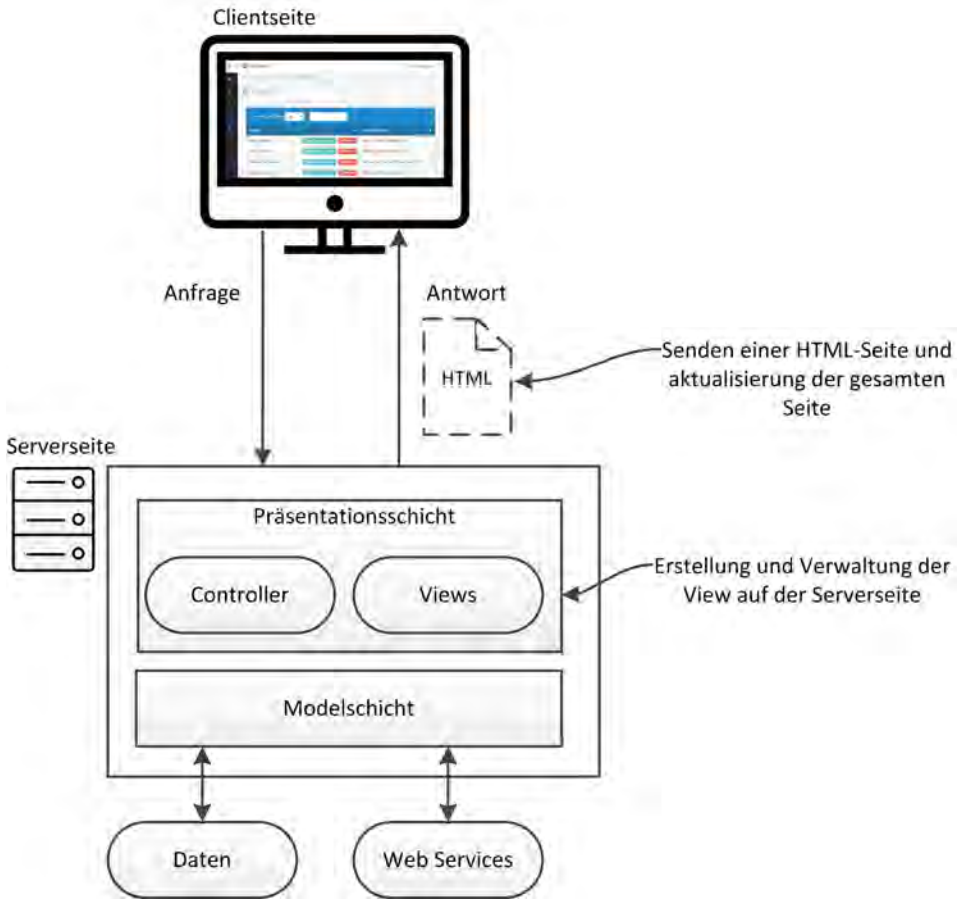

Abbildung 2.2: In einer klassichen Webanwendung wird jede View auf der Serverseite generiert (nach [6]).

Controller interagieren dabei mit der Modellschicht, um auf Daten zuzugreifen. Anschließend wird mit den Daten mithilfe von Vorlagen eine View erstellt und an den Client zurückgesendet. In einer klassischen Webanwendung repräsentiert eine View für

gewöhnlich die gesamte HTML-Seite, wohingegen in einer SPA eine HTML-Seite in mehrere Views unterteilt werden kann. Die Clientseite ist anschließend für die Visualisierung der Webanwendung zuständig und zeigt dem Nutzer die neuen Inhalte an.

Im Gegensatz dazu befindet sich in einer SPA die gesamte Anwendungslogik auf der Clientseite. Mit diesem Ansatz wird die Präsentationssicht vom Server ausgelagert und der Client ist zuständig für die Darstellung neuer Inhalte. Der Server ist nur noch für die Authentifizierung, Autorisierung und für den Zugriff auf Daten zuständig. In Abbildung 2.3 wird die Umsetzung der Architektur mit einer SPA dargestellt.

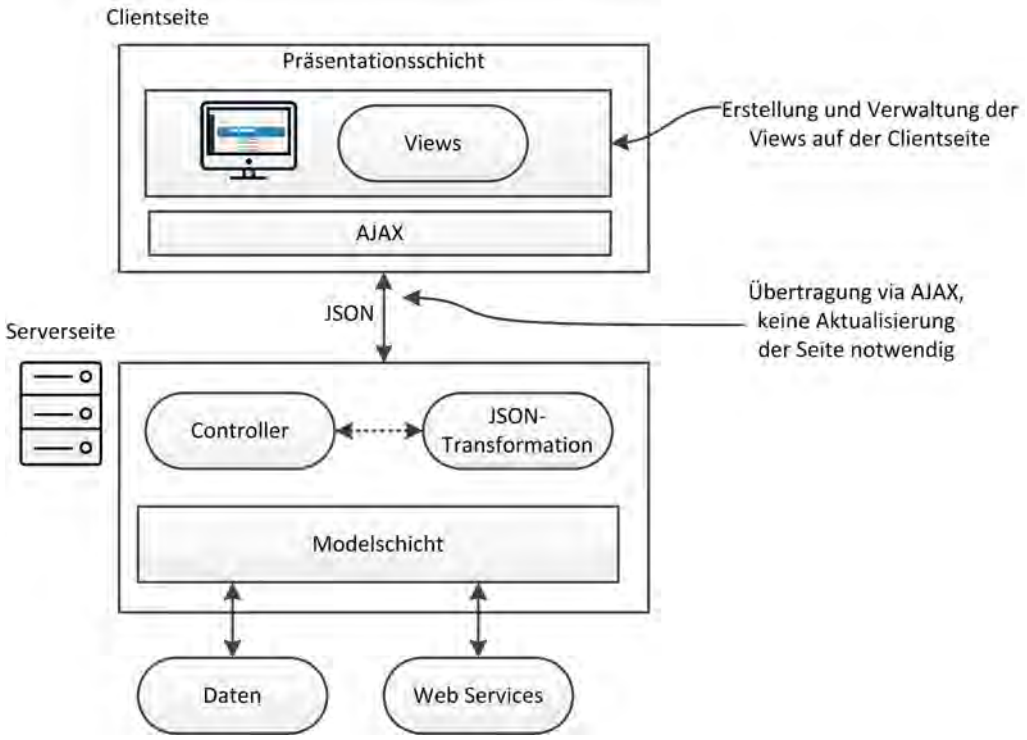

Abbildung 2.3: In einer SPA wird jede View auf der Clientseite generiert und mit den Daten vom Server befüllt (nach [6]).

Die Präsentationsschicht ist dabei komplett losgelöst vom Server und die Kommunikation zwischen Server und Client findet mit *Asynchronous JavaScript and XML (AJAX)* [7] statt. Dadurch ergeben sich gewisse Vorteile gegenüber der klassischen Webanwendung, die im folgenden vorgestellt werden.

In einer SPA existiert nur eine HTML-Seite, die für die Darstellung des Inhalts zuständig ist. Je nach Bedarf werden dazu einzelne Inhalte, im folgenden auch Views genannt, in den DOM-Baum geladen beziehungsweise miteinander ausgetauscht. Abbildung 2.4 verdeutlicht den Austausch von Views in einer SPA.

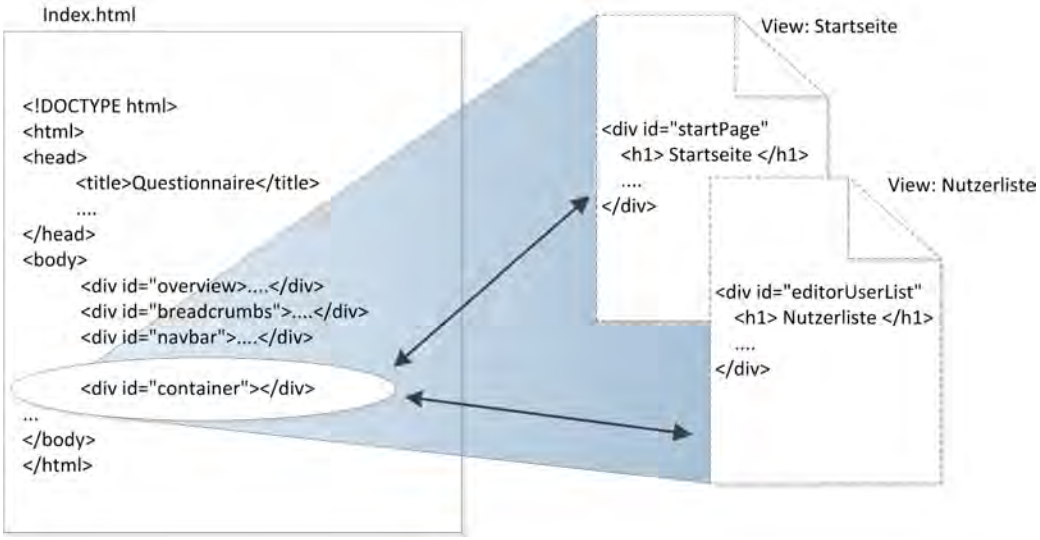

Abbildung 2.4: Sofortiger Austausch von Views in einer SPA, wodurch sich diese wie eine native Anwendung anfühlt (nach [6]).

In Abbildung 2.4 wird der Inhalt der Anwendung in der `Index.html` dargestellt. Das Element mit der id `container` ist für den eigentlichen Hauptinhalt zuständig. Soll nun beispielweise die View der Startseite dargestellt werden, findet eine DOM-Manipulation statt, in der nur die View dieses Elements mit der View der Startseite ausgetauscht wird. Da das gesamte HTML-Markup auf der Clientseite vorhanden ist, geschieht der Austausch der Views deutlich schneller wie es bei einer klassischen Webseite der Fall wäre. Eine SPA verhält sich somit eher wie eine native Anwendung.
Durch die losgelöste Präsentationsschicht ergibt sich zudem der Vorteil, dass die Clientseite unabhängig von der Serverseite gewartet und aktualisiert werden kann. Anfragen vom Client an den Server können schneller bearbeitet werden, da die Serverseite nur noch die Daten liefern muss, die der Client entsprechend verarbeitet und darstellt. Es werden keine vollständigen HTML-Seiten oder HTML-Markup an den Client geliefert,

wodurch sich die benötigte Datenbandbreite verringert. Nachteilig ist jedoch eine einmalige längere Ladezeit beim Starten der SPA, da die gesamte Logik vom Server geladen werden muss.

3

Anforderungsanalyse

In diesem Kapitel werden die Anforderungen für das zu entwickelnde System definiert und erläutert. Die Erkenntnisse aus diesem Kapitel sind essentiell für die spätere Entwurfs- und Entwicklungsphase. In Kapitel 3.2 werden zunächst die verschiedenen Nutzerrollen des Systems beschrieben. Aufbauend darauf werden in Kapitel 3.3 die funktionalen Anforderungen nach den definierten Nutzerrollen gruppiert. Abschließend werden in Kapitel 3.4 die nicht-funktionalen Anforderungen diskutiert, mit der die Qualitätseigenschaften der Plattform festgelegt werden. Zuletzt wird ein Styleguide präsentiert, mit der das Design der Benutzeroberfläche definiert wird.

3.1 Vorhandene Fragebogensysteme

In diesem Kapitel werden vorhandene Fragebogensysteme analysiert. Die Analyse bestehender Fragebogensysteme hat den Vorteil, dass ein Einblick in realistische Funktionen gewonnen wird und diese verbessert in das neue System übernommen werden können [8]. Nach der Analyse wird im Fazit die Funktionen der Systeme zusammengefasst vorgestellt.

3.1.1 SurveyMonkey

SurveyMonkey ist eine Plattform zur Erstellung und Durchführung von (anonymen) Online-Umfragen [1]. Für die Erstellung von Umfragen stehen 15 verschiedene Fragetypen bereit, die zusätzlich an Bedingungen aus früheren Antworten geknüpft werden können. Zudem können Umfragen in verschiedenen Sprachen erstellt werden. Das Hinzufügen von Bildern und Videos ist ebenfalls möglich. Umfragen können an Kontakte versendet werden sowie weitere Kontakte mit Angabe der E-Mail-Adresse eingeladen werden. Für die Darstellung auf mobilen Endgeräten existiert zudem eine mobile App für Android und iOS. SurveyMonkey bietet zudem eine Auswertung der Gesamtergebnisse sowie der Einzelergebnisse an. Die Ergebnisse können in verschiedene Formate exportiert werden wie zum Beispiel in das JSON-Format. Eine Statistik stellt Informationen über bereits abgegebene Antworten dar.

3.1.2 LimeSurvey

LimeSurvey ist ein Open-Source Umfragetool für die Erstellung von Umfragen und Studien [2]. Es können 28 verschiedene Fragetypen in einem Fragebogen verwendet werden. Mit Programmierkenntnissen können Bedingungen an frühere Antworten verknüpft werden. LimeSurvey unterstützt anonyme und mehrsprachige Fragebögen sowie die Einbindung von Bildern und Videos. Einladungen zu Umfragen können per E-Mail an Benutzer versendet werden. Zusätzlich besteht die Möglichkeit, Erinnerungen per E-Mail zu senden. Umfragen können automatisiert werden, in dem das Startdatum und

das Ablaufdatum angegeben wird. Es existiert zudem die Möglichkeit, Fragebögen zu pausieren und zu einem späteren Zeitpunkt fortzusetzen. Individuelle Bewertungen sowie eine Gesamtübersicht der Ergebnisse stehen zudem bereit. Ergebnisse können in verschiedene Formate wie zum Beispiel das CSV-Format exportiert werden. Mit einer Benutzerverwaltung können Nutzer verschiedene Rechte im System erhalten, um zum Beispiel Fragebögen zu erstellen. Mithilfe vorgefertigter Vorlagen wird die Darstellung der Fragebögen auf mobilen Endgeräten optimiert.

3.1.3 SoSci Survey

Mit dem Softwarepaket Sosci Survey lassen sich Onlinefragebögen mit mehr als 30 Fragetypen erstellen [9]. Fragebögen lassen sich anonym bearbeiten. Komplexe Funktionen wie zum Beispiel die Möglichkeit, anhand der bisherigen Antworten den Fragebogenverlauf zu steuern, erfordern Programmierkenntnisse in PHP. Zusätzlich können Bilder, Audio und Videos eingebunden werden sowie mehrsprachige Fragebögen erstellt werden. Es werden mobile Geräte unterstützt, in dem Fragen mit verschiedenen Vorlagen erstellt werden können.

3.1.4 Fazit

In diesem Kapitel werden die Möglichkeiten der Fragebogensysteme zusammengefasst. Die betrachteten Fragebogensysteme bieten insgesamt sehr ähnliche Funktionalitäten an und unterscheiden sich nur in wenigen Punkten voneinander.

Mit einer Benutzerverwaltung können den Nutzern verschiedene Rechte am System zugeteilt werden. Beispielsweise können bestimmte Nutzer das System verwalten und andere Nutzer Fragebögen versenden. Für das Versenden von Fragebögen kann das Startdatum und die Laufzeit des Fragebogens angegeben werden. Fragebögen und Erinnerungen können per E-Mail verschickt werden. Das Versenden von anonymen Fragebögen ist ebenfalls möglich. Es können individuelle Ergebnisse sowie Gesamtergebnisse dargestellt werden und es existiert eine Exportmöglichkeit der Ergebnisse in verschiedenen Formaten. Zudem können Fragebögen in verschiedenen Sprachen

bearbeitet werden sowie zu bearbeitende Fragebögen zu einem späteren Zeitpunkt fortgesetzt werden. Neben Desktop-Endgeräten werden mobile Endgeräte durch den Einsatz mobiler Apps oder geeigneter Vorlagen unterstützt.

3.2 Nutzerrollen

Basierend auf Kapitel 3.1 werden in diesem Kapitel verschiedene Nutzerrollen im System vorgestellt. Über diese Rollen wird geregelt, welcher Funktionsumfang einem Nutzer zur Verfügung steht. Einem Nutzer können dabei mehrere Rollen zugewiesen werden. Jede Rolle setzt andere Systemkenntnisse voraus, mit denen der Nutzer vertraut sein muss, um alle Funktionalitäten erfolgreich nutzen zu können. Im Folgenden werden die verschiedenen Rollen näher beschrieben.

3.2.1 Gast

Ein Benutzer hat die Rolle Gast, sobald er mit dem System interagiert und dabei nicht in das System eingeloggt ist. Seine Funktionsmöglichkeiten sind stark eingeschränkt und beschränken sich auf die Funktionen *Registrierung*, *Login* und *Passwort vergessen* (siehe Kapitel 3.3).

3.2.2 Standardnutzer

Sobald sich ein Benutzer im System erfolgreich eingeloggt hat, weist ihm das System die Rolle Standardnutzer zu. Dadurch erhält er unter anderem die Möglichkeit, *Fragebögen zu bearbeiten*. Standardnutzer können bereits nach einer kurzen Einarbeitungszeit alle Funktionalitäten des Systems verwenden.

3.2.3 Editor

Editoren sind in der Lage, *Fragebögen an verschiedene Nutzer zu senden*, *Ergebnisse anzuzeigen* und *Nutzer sowie Gruppen zu verwalten*. Ein Benutzer mit der Rolle Editor

besitzt gleichzeitig auch die Rolle Standardnutzer. Die Funktionsmöglichkeiten sind im Vergleich zum Standardnutzer erheblich erweitert und daher benötigt ein Editor eine höhere Einarbeitungszeit, um alle Funktionen zu verwenden.

3.2.4 Administrator

Administratoren sind für die Verwaltung des Systems zuständig. Sie weisen anderen Nutzern Rollen zu, erstellen Fragebögen und sind für die Nutzerverwaltung zuständig. Administratoren können kritische Funktionen ausführen wie beispielweise das Löschen eines Benutzers und benötigen eine intensive Einarbeitung in das System.

3.3 Funktionale Anforderungen

In diesem Kapitel werden zunächst die funktionalen Anforderungen grob beschrieben, die aus den Erkenntnissen des Kapitels 3.1 gewonnen wurden. Diese Anforderungen sind in Abbildung 3.1 als Anwendungsfalldiagramm sortiert nach den beschriebenen Nutzerrollen aus Kapitel 3.2 dargestellt. Im weiteren Verlauf werden auf diese Anforderungen textuell detailliert eingegangen. Dabei werden mögliche Ausnahmen und Besonderheiten bei der Verwendung der Funktionen erläutert.

Ein Nutzer soll in der Rolle als Gast die Möglichkeit haben, sich im System zu registrieren und bei Verlust seines Passworts ein neues zu setzen. Standardnutzer sollen sich in das System einloggen, ihre Daten ändern können und individuelle Einstellungen zur Sprache und Benachrichtigungen per E-Mail einstellen können. Weiterhin können Standardnutzer Fragebögen ausfüllen. Editoren können ihre eigenen Nutzer, Gruppen und Fragebögen verwalten, Fragebögen versenden, Ergebnisse zu ausgefüllten Fragebögen einsehen und diese Ergebnisse exportieren. Administratoren sollen die Möglichkeit besitzen, die Nutzer des Systems sowie erstellte Fragebögen zu verwalten.

Abbildung 3.1: Anwendungsfalldiagramm gruppiert nach Nutzerrollen

FA1 Login (Gast): Der Benutzer soll in der Rolle Gast die Möglichkeit haben, sich in das System mit seiner E-Mail-Adresse und seinem Passwort einzuloggen. Durch den Login weist ihm das System mindestens die Rolle Standardnutzer zu, mit dem weitere Funktionen freigeschaltet werden. Falls der Benutzer falsche Anmeldedaten eingibt, soll ihm eine Fehlermeldung angezeigt werden.

FA2 Registrierung (Gast): Der Benutzer kann in der Rolle Gast einen Account registrieren. Dazu muss der Nutzer seinen Namen, seine E-Mail-Adresse, ein Passwort und die Sprache auswählen. Bei einer erfolgreichen Registrierung soll der Benutzer eine Erfolgsmeldung erhalten und der nächste Schritt zur vollständigen Registrierung angezeigt werden. Für den Nutzer besteht der nächste Schritt daraus, dass er eine Bestätigungsemail mit einem Aktivierungslink erhält. Sobald der Nutzer

den Aktivierungslink aufruft, ist die Registrierung abgeschlossen.
Fehlt bei der Registrierung eine der Angaben beziehungsweise ist die E-Mail-Adresse bereits vergeben oder das Passwort zu kurz, soll dem Nutzer eine Fehlermeldung angezeigt werden.

FA3 Optionen (Standardnutzer): In den Optionen soll der Benutzer in den Bereichen *Nutzerdaten*, *Passwort ändern* und *Sonstiges* seine Daten ändern können. Im Bereich *Nutzerdaten* kann er seinen Namen und seine E-Mail-Adresse ändern. Ist das Namensfeld leer oder die E-Mail-Adresse nicht gültig beziehungsweise bereits vergeben, erhält der Nutzer entsprechende Fehlermeldungen.
Im Bereich *Passwort ändern* soll der Benutzer sein Passwort ändern können. Tritt bei der Änderung des Passworts ein Fehler auf (beispielsweise ist bei der wiederholten Eingabe das neue Passwort nicht identisch), so erhält der Benutzer eine entsprechende Fehlermeldung.
Im Bereich *Sonstiges* soll es möglich sein, die Sprache des Systems zu ändern und anzugeben, ob E-Mail-Benachrichtigungen zu neuen Fragebögen versendet werden sollen.

FA4 Nutzer- und Rollenmanagement (Administrator): Im Nutzer- und Rollenmanagement sollen Administratoren Benutzern verschiedene Rollen zuweisen und entfernen. Zudem sollen Daten des Benutzers geändert und Nutzer aus dem System gelöscht werden können. Bei einer Änderung eines Nutzers erhält der Administrator eine entsprechende Nachricht. Wird ein Nutzer gelöscht, so soll der Vorgang bestätigt werden.

FA5 Fragebogen hochladen (Administrator): Administratoren sollen Fragebögen hochladen können. Dabei soll ein Fragebogen einen Titel sowie eine Beschreibung besitzen. Zu jedem Fragebogen gilt, dass dieser individuelle Einstellungsmöglichkeiten besitzt. Es soll es in den Einstellungen möglich sein, einen Fragebogen in einem bestimmten Abstand wiederholt an Nutzer zu senden. Es muss die Anzahl der Wiederholungen des Fragebogens, der Abstand in Tagen zwischen jedem Fragebogen und die maximale erlaubte Durchführungszeit des Fragebogens in Tagen angegeben werden. Nachdem ein Fragebogen hochgeladen wurde, soll

der Administrator eine Meldung erhalten und zur vorherigen Seite zurück geleitet werden.

FA6 Nutzer- und Gruppenübersicht (Editor): Editoren sollen eine Nutzer- sowie eine Gruppenübersicht besitzen. In der Nutzerübersicht sollen Nutzer hinzugefügt und entfernt werden. Editoren sollen zudem die Möglichkeit besitzen, Gruppen zu erstellen sowie zu löschen und Nutzer in diese Gruppen hinzuzufügen und auch entfernen zu können. Vor dem Löschen eines Nutzers oder einer Gruppe soll der Löschvorgang vom Editor bestätigt werden. Jegliche Änderungen sollen dem Editor mit einer Benachrichtigung angezeigt werden.

FA7 Fragebogenübersicht (Editor): Eine Fragebogenübersicht soll alle Fragebögen des zuständigen Editors auflisten. Über diese Übersicht können Fragebögen in einer seperaten Ansicht geändert werden können. Zusätzlich soll es möglich sein, Fragebögen direkt zu löschen. Vor dem Löschen eines Fragebogens soll der Vorgang bestätigt werden und bei einer erfolgreichen Durchführung des Löschvorgangs eine Meldung dem Editor eingeblendet werden.

FA8 Fragebogeneinstellungen bearbeiten (Editor): Die Daten zu einem Fragebogen sollen angepasst werden können. Dazu zählt die Änderung des Titels, der Beschreibung und der Optionen (siehe FA5). Fragebögen, die der Editor vor der Änderung an die Benutzer versendet hat, sollen nicht von den Änderungen in den Optionen betroffen sein. Bei einer Änderung des Fragebogens soll der Editor eine Meldung erhalten.

FA9 (Anonyme) Fragebögen versenden (Editor): Editoren sollen Fragebögen an Nutzer versenden können. Der Versand soll dabei in zwei Schritten erfolgen. Im ersten Schritt wird entweder ein Fragebogen aus der Fragebogenübersicht zum Versenden ausgewählt oder eine Gruppe beziehungsweise ein Nutzer aus der Nutzer- beziehungsweise Gruppenübersicht ausgewählt. Im nächsten Schritt sollen dem Editor auf einer neuen Seite verschiedene Konfigurationsmöglichkeiten angeboten werden. Dabei hat der Editor die Möglichkeit den zu versendenden Fragebogen auszuwählen. Zudem kann er die Nutzer- und Gruppen auswählen, an die der Fragebogen versendet werden soll. Es soll die Möglichkeit bestehen,

dass Nutzer zum Fragebogen eingeladen werden können, die noch nicht am System registriert sind. Dazu muss der Editor den Namen des Nutzers und die E-Mail-Adresse angeben. Optional kann der neue Nutzer in die Gruppen des Editors zugewiesen werden. Zusätzlich sollen anonyme Fragebögen an bisherige Nutzer und neue Nutzer versendet werden können. Nicht-registrierte Nutzer sollen über einen Link zugreifen können, der dem Editor nach dem Versenden eines anonymen Fragebogens erstellt wurde. Außerdem soll ein Startdatum festgelegt werden können, ab dem der Fragebogen bearbeitet werden kann. Nach Absenden des Fragebogens soll der Editor eine Meldung erhalten und auf die vorherige Seite zurück geleitet werden. Nutzer erhalten am Tag des Startdatums eine Einladung via E-Mail, mit der Sie über einen Link auf den Fragebogen zugreifen können.

FA10 Fragebögen ausfüllen (Standardnutzer): Ein Nutzer soll einen erhaltenen Fragebogen ausfüllen können. Dabei soll der Benutzer die Möglichkeit besitzen, diesen jederzeit zu pausieren und zu einem anderem Zeitpunkt fortzusetzen. Außerdem soll der Nutzer die Möglichkeit besitzen, einen Fragebogen abzulehnen. Bevor ein Fragebogen abgelehnt werden kann, muss der Ablehnungsvorgang bestätigt werden. Bei einem erfolgreicher Ablehnung wird diese dem Nutzer bestätigt.

FA11 Status der Fragebögen einsehen (Editor): Editoren sollen den Status der Fragebögen zu jedem Nutzer einsehen können. `Abgeschlossene Fragebögen` wurden erfolgreich vom Benutzer bearbeitet. Fragebögen mit dem Status `In Bearbeitung` können von Nutzern bearbeitet werden. Für zu bearbeitende Fragebögen soll der Editor den Nutzern eine Erinnerung per E-Mail versenden können. Für `offene Fragebögen` gilt, dass diese noch nicht vom Nutzer bearbeitet werden können, da das Startdatum noch in der Zukunft liegt. Ein Fragebogen hat den Status `Abgelehnt`, sobald der Nutzer den Fragebogen abgelehnt hat oder diesen nicht in der vorgegebenen Zeit bearbeitet hat. Für abgelehnte Fragebögen soll der Editor die Möglichkeit besitzen, diese für die Nutzer wieder freizuschalten.

FA12 Ergebnisübersicht (Editor): Zu jedem Fragebogen soll eine Auswertungsübersicht mit den wichtigsten Ergebnissen angezeigt werden. Auf dieser Seite können

die Ergebnisse der jeweiligen Nutzer aufgerufen und allgemeine Informationen zum Status der versendeten Fragebögen eingesehen werden.

FA13 Export der Ergebnisse (Editor): Es soll zu jedem Fragebogen möglich sein, die gesamte Auswertung sowie die Auswertung der einzelnen Nutzer in verschiedenen Formaten herunter laden zu können.

3.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Qualitätseigenschaften, die das entwickelnde System besitzen muss. Die Qualitätseigenschaften dienen als Konstrukt für die Umsetzung des Systems, da Sie die Entwicklungsmöglichkeiten der funktionalen Anforderungen einschränken. Für die Klassifizierung der nicht-funktionalen Anforderungen wird die Norm ISO-25000 [10] verwendet. Dabei werden in diesem Kapitel auf die Qualitätsmerkmale dieser Norm eingegangen, die in Abbildung 3.2 dargestellt sind.

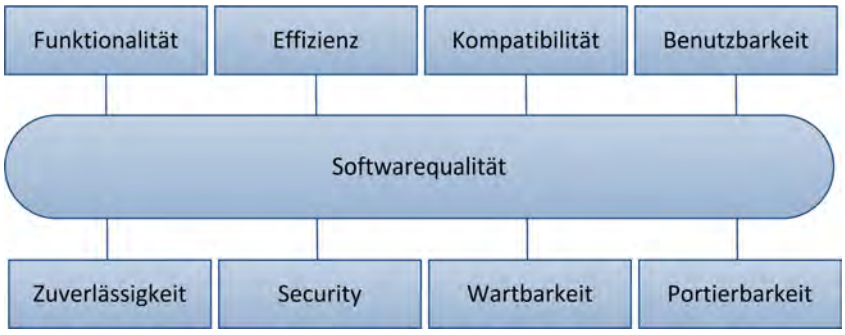

Abbildung 3.2: Nicht-funktionale Anforderungen nach ISO 25010 [10].

NFA1 Funktionalität: Für die Funktionalität gilt, dass der Benutzer alle geforderten funktionalen Anforderungen verwenden kann. Diese Anforderungen müssen funktional richtig sein, das heißt die gewünschten Ergebnisse liefern. Darüber hinaus sollen die Funktionen in einem angemessenen Ausmaß die Durchführung der Aufgaben unterstützen.

NFA2 Effizienz: Für die Effizienz gilt, dass die funktionalen Anforderungen in einer für den Benutzer angemessenen Zeit bearbeitet werden sollen. Das bedeutet,

dass nur die Funktionen auf der Serverseite umgesetzt werden sollen, die auch zwingend eine Internetverbindung benötigen.

NFA3 Kompatibilität: Das System soll in einer gemeinsamen Umgebung mit anderen Systemen problemlos funktionieren und sich nicht gegenseitig negativ beeinflussen. Komponenten, die miteinander kommunizieren, sollen bewährte Datenformate verwenden, um die Zusammenarbeit zu vereinfachen.

NFA4 Benutzbarkeit: Die Bedienbarkeit soll für den Benutzer angenehm sein und der Aufwand für die Verwendung der Funktionen so niedrig wie möglich gehalten werden. Unnötige Arbeitsschritte sollen zudem vermieden werden. Die Funktionen sollen zudem für den Benutzer verständlich und leicht erlernbar sein.

NFA5 Zuverlässigkeit: Das System soll zuverlässig funktionieren. Falls während der Benutzung Fehler auftreten, soll das System mit diesen sinnvoll umgehen können und den Zustand des Systems in einen konsistenten Zustand wiederherstellen. Rückmeldungen über Fehler sollen dem Benutzer die Möglichkeit bieten, eingegebene Daten zu korrigieren und Informationen über den Grund des Fehlers zu erhalten.

NFA6 Sicherheit: Für die Sicherheit gilt, dass Daten und Informationen vor unerlaubten Zugriffen geschützt sind. Es muss sichergestellt sein, dass Benutzer nur auf die Daten zugreifen können, für die Sie eine Berechtigung besitzen. Das System selbst muss sicherstellen, dass eine Manipulation und ein unerlaubter Zugriff auf Daten nicht möglich sind. Bei einem Zugriff auf die Daten muss das System nachweisen können, dass der Benutzer auch wirklich die Person ist, die Sie vorgibt. Sensible Daten auf dem System sollen verschlüsselt gespeichert werden.

NFA7 Wartbarkeit: Die Architektur des Systems soll so umgesetzt werden, dass eine spätere Änderung, Erweiterbarkeit und Wartbarkeit des Systems ermöglicht wird. Dabei sollen neue oder zu ändernde Inhalte ohne Änderungen des Systems erfolgen können.

NFA8 Portierbarkeit: Für die Portierbarkeit gilt, dass das System ohne Änderungen auf verschiedene Betriebssysteme funktioniert. Für die Serverseite gilt daher, dass bewährte Technologien verwendet werden soll. Clientseitig muss im Zuge der

immer größeren Zunahme an mobilen Endgeräten die Nutzung jener berücksichtigt werden. Desktop-Endgeräte und mobile Endgeräte wie Smartphones und Tablets besitzen unterschiedliche Bildschirmauflösungen, die für eine hohe Benutzerfreundlichkeit jeweils eine angepasste Benutzeroberfläche benötigen. In Kapitel 3.5 wird dazu näher auf die Faktoren der Portierung eingegangen.

3.5 Styleguide

In diesem Kapitel werden Styleguides beschrieben, die das Design der Benutzeroberflächen definieren. Durch eine konsequente Umsetzung dieser Styleguides wird die Benutzerfreundlichkeit durch eine einheitliche und konsistente Benutzeroberfläche verbessert. Im Folgenden werden die *acht goldenen Regeln des Interface Designs* von Ben Shneiderman [11] erklärt, welche bei der Umsetzung der Benutzeroberfläche befolgt werden sollten.

SG1 Streben nach Konsistenz: Wieder auftauchende Inhalte, Meldungen und Funktionen wie z.B. hinzufügen, weiter blättern oder in sich ähnelnde Anforderungen sollten nach dem gleichem Muster aufgebaut und bedienbar sein.

SG2 Shortcuts für erfahrene Benutzer anbieten: Für oft verwendete Funktionen sollten Interaktionsschritte minimal gehalten werden und Abkürzungen zu diesen Funktionen zur Verfügung stehen.

SG3 Informatives Feedback anbieten: Für jede Systeminteraktion sollte dem Nutzer eine Systemrückmeldung angezeigt werden. Oft vorkommende Systeminteraktionen benötigen nur eine dezente Rückmeldung, wohingegen Interaktionen, die selten vorkommen oder große Auswirkungen besitzen, eine deutliche und ausführliche Rückmeldung anzeigen sollten.

SG4 Dialoge abgeschlossen gestalten: Eine Sequenz von Interaktionen sollte in einen Anfangsteil, Mittelteil und in einen Endteil gruppiert werden können. Eine informative Rückmeldung nach Beendigung einer solchen Gruppeninteraktion stellt

die meisten Nutzer zufrieden. Somit kann sich ein Nutzer voll und ganz auf die nächste Aufgabe konzentrieren.

SG5 Einfache Fehlerbehandlung anbieten: Das Interface sollte so gestaltet sein, dass fehlerhafte Benutzereingaben nicht möglich sind. Da dies in der Realität allerdings selten umsetzbar ist, sollte das System dem Benutzer einfache Möglichkeiten anbieten, fehlerhafte Eingaben ohne große Mehrarbeit seitens des Benutzers zu korrigieren.

SG6 Aktionen rückgängig machen: Durch dieses Feature wird dem Benutzer die Angst genommen, dass unwiderrufliche Fehler auftreten. Zudem bestärkt es den Nutzer, neue Funktionen auszuprobieren und sich besser mit dem System vertraut zu machen. Eine Aktion, die sich rückgängig machen lässt, kann dabei eine einzelne Aktion, ein Dateneintrag oder eine Gruppe von Aktionen sein.

SG7 Benutzerkontrolle: Besonders erfahrene Benutzer sollten das Gefühl haben, zu jedem Zeitpunkt die vollständige Kontrolle über das System zu besitzen und dass das System auf ihre Interaktionen reagiert und nicht umgekehrt. Das System sollte daher so konzipiert sein, dass Interaktionen vom Benutzer anstatt vom System initiiert werden.

SG8 Kurzzeitgedächtnis entlasten: Das menschliche Kurzzeitgedächtnis kann etwa 7 Informationen gleichzeitig speichern. Daher sollte die Benutzeroberfläche einfach und strukturiert aufgebaut sein. Die Darstellung von Informationen auf mehreren Seiten gleichzeitig sollte vermieden und die Fensterbewegungsfrequenz reduziert werden. Eine ausreichende Zeit für das Erlernen von komplexen Interaktionen sollte gegeben sein.

Neben den oben genannten Styleguides gibt es weitere Regeln, die insbesondere die Nutzung des Systems mit mobilen Endgeräten berücksichtigen. Wie aus den Anforderungen aus Kapitel 3.4 hervorgegangen ist, muss die Benutzeroberfläche für viele verschiedene Endgeräte angepasst sein. Dies ist auch unter dem Begriff *Responsive Design* bekannt. Im Folgenden werden kurz die Herausforderungen für die Unterstützung mobiler Endgeräte vorgestellt und anschließend passende Lösungen präsentiert [12].

RD 1 Kleine Bildschirmgröße: Mobile Endgeräte bieten nur einen sehr beschränkten Bereich für die Darstellung von Informationen an. Daher ist es wichtig zu entscheiden, welche Elemente für den Nutzer bei der Bedienung von essentieller Bedeutung sind.

RD 2 Unterschiedliche Bildschirmbreiten: Die Anzahl von mobilen Endgeräten mit ihren unterschiedlichen Bildschirmbreiten kann einen großen Mehraufwand bei der Erstellung der verschiedenen Designs verursachen.

RD 3 Touch Screens: Kleine Elemente können nur schwer präzise mit den Fingern berührt werden. Daher müssen Elemente eine bestimmte Mindestgröße von mindestens 1 cm besitzen und zwischen ihnen genügend Platz vorhanden sein. Dadurch verringert sich jedoch der vorhandene Platz für die Darstellung weiterer Elemente.

RD 4 Schwierigkeit des Schreibens: Es ist sehr mühsam, Texte auf einem mobilen Endgerät mithilfe einer Softwaretastatur zu schreiben. Daher sollten Interaktionen so konzipiert sein, dass möglichst wenig geschrieben werden muss.

RD 5 Herausforderungen des Umfelds: Mobile Endgeräte können in verschiedenen Umgebungsszenarien zum Einsatz kommen, wie zum Beispiel im Zug oder draußen unter starker Sonneneinstrahlung. Die verschiedenen Umgebungsbedingungen können die Verwendung des Systems erschweren.

Um die Herausforderungen für mobile Enderäte zu meistern, beschreibt Jenifer Tidwell mehrere Lösungen.

RD 6 Vermeide unnötige Elemente im System: Unnötige Elemente, die den Benutzer von der Arbeit ablenken wie zum Beispiel Werbung oder unnötige Bilder sollen vermieden werden. Es sollen nur die für den Benutzer relevanten Elemente zur Verwendung des Systems vorhanden sein.

RD 7 Optimierung von oft verwendeten Interaktionen: Nachdem alle Interaktionssequenzen des Systems definiert sind, sollen diese nach folgenden Heuristiken optimiert werden:

- Es soll das Schreiben auf dem mobilen Endgerät auf so wenige Zeichen wie möglich beschränkt werden.
- Das Nachladen von neuen Seiten soll minimiert werden.
- Das Scrollen in horizontaler Richtung soll soweit es geht vermieden und Inhalte optimiert in vertikaler Richtung dargestellt werden.
- Die Anzahl der Interaktionen für die Verwendung einer Funktionalität soll auf ein Minimum reduziert werden.

In diesem Kapitel wurden die Anforderungen an das System erstellt. Dazu wurden bereits existierende Fragebogensysteme auf ihren Funktionsumfang untersucht. Basierend auf dem Vergleich der Systeme wurden die Nutzerrollen des Systems definiert. Zusätzlich wurden die funktionalen sowie nicht-funktionalen Anforderungen und ein Styleguide erstellt. Auf den Anforderungen aufbauend wird in Kapitel 4 das Konzept und der Entwurf des Systems vorgestellt.

4

Konzept und Entwurf

In diesem werden Kapitel Konzepte und Entwürfe zum entwickelnden System vorgestellt. Zunächst wird in Kapitel 4.1 das verwendete Fragebogenmodell erläutert. Darauf aufbauend wird in Kapitel 4.2 ein Antwortenmodell erstellt. Kapitel 4.3 beschreibt die Abbildung eines Fragebogens auf ein Prozessmodell. Die Struktur des Clients und des Servers sowie das Datenmodell wird in Kapitel 4.4 diskutiert. Abschließend wird in Kapitel 4.5 die Benutzeroberfläche des Systems beschrieben.

4.1 Struktur des Fragebogenmodells

Für die prozessorientierte Verarbeitung der Fragebögen ist es wichtig, sich detailliert mit dem Aufbau der Fragebogenmodells zu beschäftigen. Daher werden in diesem Kapitel die wichtigsten Elemente und die Struktur kurz beschrieben. Eine detaillierte

Beschreibung der Elemente findet sich [13] und [14].
Der Konfigurator (siehe Kapitel 2) liefert den gesamten Inhalt eines Fragebogens in Form einer ZIP-Datei. Darin enthalten ist das Fragebogenmodell als XML-Dokument und mögliche Bilder und Videos. Abbildung 4.1 beschreibt das Zusammenspiel zwischen den verschiedenen Elementen des Fragebogenmodells und stellt den Fragebogenablauf als direkten Graphen dar.

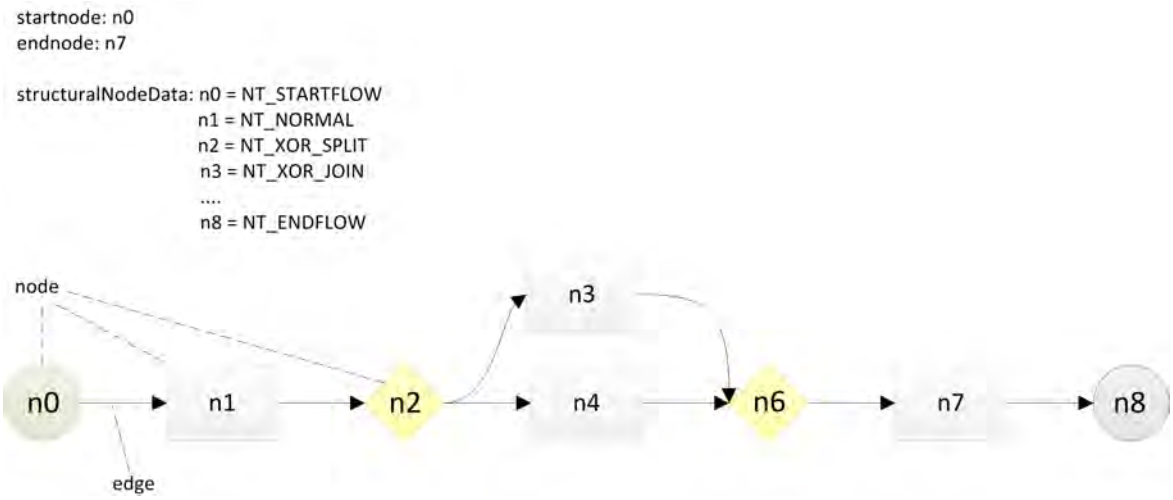

Abbildung 4.1: Übersicht der Fragebogenelemente.

Ein Element `node` (Knoten) beinhaltet Informationen über eine Fragebogenseite, stellt einen Start- beziehungsweise Endknoten oder ein Gateway (Kontrollknoten) dar. Zu jedem Knoten existiert ein `structuralNodeData` Element, die den Typen des jeweiligen Knotens angeben. In Abbildung 4.1 ist beispielsweise der Knoten `n0` der Startknoten, `n2` ein `XOR-Split` Gateway und `n1` eine Fragebogenseite.

Fragebogenseiten können aus einer Vielzahl von Elementen `Headline`, `Media`, `Text` und `Question` aufgebaut sein (siehe Abbildung 4.2). `Headline`, `Media` und `Text` stellen nur passive Inhalte dar, wohingegen `Question` Nutzereingaben für den weiteren Verlauf des Fragebogens erfordern. Jedes `Question` Element enthält eine Anzahl an Fragen. Eine Frage kann beispielsweise eine Single-Choice Frage sein, mit der genau eine Antwort aus mehreren Antwortmöglichkeiten ausgewählt werden muss. Zu jeder Frage wird ein eindeutiges Datenelement spezifiziert, in der die Antwort der jeweiligen Frage gespeichert werden soll. Diese Zuordnung ist sowohl für die spätere

Auswertung des Fragebogens als auch für die Funktionsweise der Gateways (z.B. `XOR-Join` Gateway) wichtig.

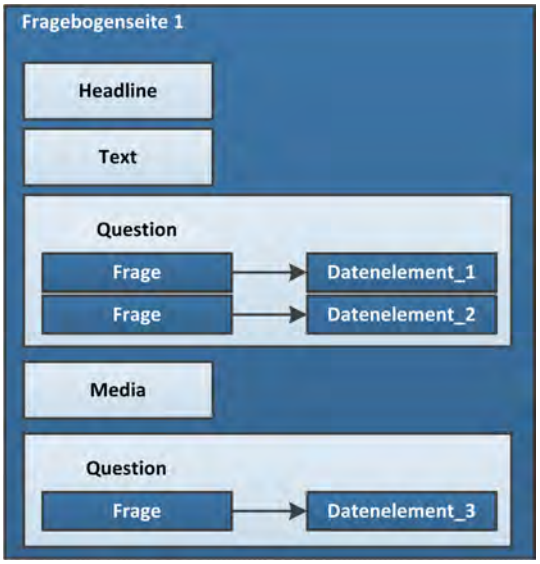

Abbildung 4.2: Struktur einer Fragebogenseite an einem Beispiel dargestellt.

Neben den bisher betrachteten Fragebogenseiten existieren zusätzliche Gateways (siehe Abbildung 4.1). Gateways sind entweder vom Typ `XOR-Split` bzw. `XOR-Join`. In Zukunft sollen zudem `AND-Split` und `AND-Join` Gateways unterstützt werden. Da Fragebogenmodelle mit parallelen Verzweigungen noch nicht vom Konfigurator erstellt werden können, wird in dieser Arbeit auf die Ausführung dieser Knoten verzichtet. Ein `XOR-Split` Gateway setzt zwingend einen `XOR-Join` Gateway im weiteren Verlauf des Graphen voraus, welcher alle verzweigten Pfade vom `XOR-Split` Gateway wieder zusammenführt. Um den weiteren Pfad zu bestimmen, besitzt ein `XOR-Split` Gateway eine Anzahl von sogenannten `decisionStatement` Elementen (siehe Abbildung 4.3). Jede dieser Bedingungen besitzt eine boolesche Entscheidungsfunktion, die auf die bisherigen Antworten des Benutzers verweisen, sowie eine `decisionID`, welche den nächsten Knoten bestimmt. Das erste `decisionStatement` ist dabei immer der `ELSE`-Fall, der automatisch ausgewählt wird, falls alle anderen Entscheidungsfunktionen nicht zutreffen. Die restlichen `decisionStatements` werden nacheinander interpretiert. Sobald eines dabei den booleschen Wert `wahr` zurückliefert, wird die dazugehörige `decisionID` für den Kontrollknoten ausgewählt.

Im Folgenden wird näher auf die Entscheidungsfunktionen eingegangen. Wie in Abbildung 4.3 in der Entscheidungsfunktion mit der `decisionID 1` zu sehen ist, kann eine Entscheidungsfunktion aus einer oder mehreren Funktionen bestehen. Jede Entscheidungsfunktion enthält als ersten Parameter das Datenelement, in dem die Antworten vom Benutzer gespeichert sind. Des Weiteren ist die Anzahl der Parameter abhängig von der zu verwendenden Funktion.

Abbildung 4.3: Aufbau und Beispiel von Entscheidungsfunktionen.

In Abbildung 4.3 wird die Auswertung einer Entscheidungsfunktion an einem Beispiel verdeutlicht. Diese besteht aus den Funktionen `xor.operations.selected()` und `xor.operations.contains()`. Wurde die Antwort *Januar* vom Benutzer ausgewählt und enthält ein vom Benutzer eingegebener Text das Wort *Rom*, wird die Entscheidungsfunktion mit `wahr` ausgewertet und für die Auswahl des nächsten Knotens die `decisionID 1` verwendet.

Das Element `edge` (Kante) stellt eine Verbindung zwischen zwei Knoten her. Jede Kante enthält dabei Referenzen auf den Ursprungs- und den Zielknoten, sowie eine Beschreibung des Kantentyps. Ein Beispiel zu den Elementen im XML-Format findet sich im Quelltext 4.1. Wie im Beispiel zu sehen ist, ist der nächste Knoten in der Bearbeitung nach `n14` der Knoten `n9`. Besitzen mehrere Kanten den gleichen Ursprungsknoten, so kann es sich dabei um einen `XOR-Split` oder `AND-Split` Gateway handeln. Der Knoten `n8` ist beispielsweise ein `XOR-Split` Gateway. Die möglichen Werte in den `edgeCodes`

entsprechen den möglichen `deciscionIDs`, die in den Entscheidungsfunktionen der `XOR-Split` Knoten bestimmt werden. Wurde zum Beispiel im Kontrollknoten `n8` die decisionID `1` durch die Entscheidungsfunktionen bestimmt, so wird im nächsten Schritt der Knoten `n9` bearbeitet.

```
1  <edge destinationNodeID="n9" edgeType="ET_CONTROL" sourceNodeID="n14">
2    <edgeType>ET_CONTROL</edgeType>
3  </edge>
4  <edge destinationNodeID="n10" edgeType="ET_CONTROL" sourceNodeID="n8">
5    <edgeType>ET_CONTROL</edgeType>
6    <edgeCode>0</edgeCode>
7  </edge>
8  <edge destinationNodeID="n11" edgeType="ET_CONTROL" sourceNodeID="n8">
9    <edgeType>ET_CONTROL</edgeType>
10   <edgeCode>1</edgeCode>
11 </edge>
```

Quelltext 4.1: Beispiel von Kanteninformationen im XML-Format.

Außerdem existieren die Elemente `startNode` und `endNode`, mit denen der Start- beziehungsweise der Endknoten des Prozessmodells spezifiziert wird. Nach Ausführung des Endknotens ist der Fragebogen erfolgreich abgeschlossen.
Basierend auf diesem Kapitel wird im nächsten Kapitel die Struktur der Antworten beschrieben.

4.2 Struktur des Antwortenmodells

In diesem Kapitel wird die Struktur des Antwortenmodells beschrieben. Der Aufbau und Inhalt dieses Modells ist von essentieller Bedeutung für die spätere Auswertung. Um die Kriterien für eine geeignete Struktur des Antwortenmodells zu definieren, wird zunächst überprüft, in welchen Prozessschritten auf die Antworten zugegriffen werden.
Aus den Anforderungen in Kapitel 3.3 gehen bereits Prozessschritte hervor, die auf die Antworten zugreifen. Um bisherige Antworten zu korrigieren, soll der Benutzer bei der Bearbeitung eines Fragebogens Seiten `vorwärts` und `zurück` blättern können.

Das bedeutet, dass das System in der Lage sein muss, die bisherigen Antworten zu rekonstruieren und dem Benutzer anzuzeigen. Der Nutzer soll auch die Möglichkeit besitzen, bereits gestartete Fragebögen zu einem `späteren Zeitpunkt fortsetzen zu können` und seine bisherigen Antworten einzusehen.
Wie in Kapitel 2 erläutert wurde, sollen in Zukunft alternative Systeme für die Bearbeitung von Fragebögen existieren. Für mögliche Einschränkungen oder Besonderheiten dieser Systeme ist es daher sinnvoll, Informationen zum verwendeten Client zu erfassen.
Dabei wird die Auswertung deutlich vereinfacht, wenn alle Systeme ihre Antworten in einem einheitlichem Format abliefern. Eine einfache Erweiterbarkeit soll ebenfalls gegeben sein, um in Zukunft neue Fragetypen oder weitere Informationen hinzuzufügen.
Aus diesen Überlegungen lassen sich folgende Kriterien für die Struktur des Modells ableiten:

- Antworten sollen eindeutig rekonstruierbar sein.
- Informationen bezüglich des Clients sollen vorhanden sein.
- Eine einheitliche Struktur soll gegeben sein.
- Das Modell soll leicht erweitert werden können.

Im Folgenden wird aufbauend auf den oben genannten Kriterien das entworfene Antwortenmodell beschrieben. Die Datenelemente aus der Fragebogenstruktur (siehe Kapitel 4.1) sind eindeutig und bilden daher die obersten Elemente im Modell. Zu jeder Frage existieren weitere Zusatzinformationen, die in den Datenelementen gespeichert werden. Diese sind:

- **answers:** Beinhaltet Informationen zur den möglichen Antworten.
- **dataElementName:** Name des Datenelements.
- **dataprotected:** Gibt an, ob das Element sensible Daten enthält, die nach der Auswertung gelöscht werden müssen.
- **exportName:** Exportname des Datenelements, mit der die Antworten nach Kategorien sortiert werden können.

- **meta:** Enthält die Startzeit und Endzeit der Fragebogenseite und Informationen über den Client.
- **name:** Interner Name des Datenelements.
- **questionType:** Der Fragetyp zum Datenelement.

Das Element `answers` ist ein Array, welche alle Antwortmöglichkeiten zu einer Frage enthält. Beispielsweise kann eine Single-Choice Frage mehrere Antwortmöglichkeiten anbieten, wohingegen eine `Freetext` Frage nur eine Antwortmöglichkeit zulässt. In den erstellten Fragebögen ist es möglich, dass einzelne Antwortmöglichkeiten auf eine Frage den gleichen Namen besitzen. Um eine Namenskollision zu vermeiden, wird in den Entscheidungsfunktionen der `XOR-Split` Gateways auf die richtige Antwort verwiesen, in dem die Position der Antwort im Array zusammen mit einem Unterstrich vor dem Namen der Antwort angehängt ist. Diese Vorgehensweise wird auch im Antwortenmodell übernommen. Eine Antwort zu einer Frage ist wie folgt aufgebaut:

- **exportName:** Exportname der Antwort, welches für die spätere Auswertung im System wichtig ist.
- **inputFormat:** Spezifiziert das Format der Eingabe. Diese kann beispielsweise die Eingabe auf Ganzzahlen oder Daten beschränken.
- **name:** Interner Name der Antwort.
- **text:** Enthält die Antwort des Benutzers, falls das Element userInput aktiv ist.
- **userInput:** Gibt an, ob der Benutzer weitere Informationen zur Antwort eingeben kann. Dies kann zum Beispiel bei einer Single-Choice Frage oder bei einer Freetext Frage der Fall sein.
- **value:** Gibt an, ob diese Antwortmöglichkeit ausgewählt wurde oder nicht.

Insgesamt wurde ein einheitliches Antwortenmodell vorgestellt, welches unabhängig vom zu verwendeten System ist. Antworten können durch die eindeutigen Datenelemente rekonstruiert werden. Namenskollisionen werden vermieden, sodass die richtigen Antworten in den Entscheidungsfunktionen ausgewertet werden. Durch die Angabe des Fragetyps können neue Fragetypen einfach hinzugefügt werden und bei Bedarf um neue

Elemente erweitert werden. Die Metainformationen können wichtige Informationen zu Clients liefern. In Abbildung 4.4 ist eine mögliche Single-Choice Frage basierend auf der vorgestellten Struktur abgebildet.

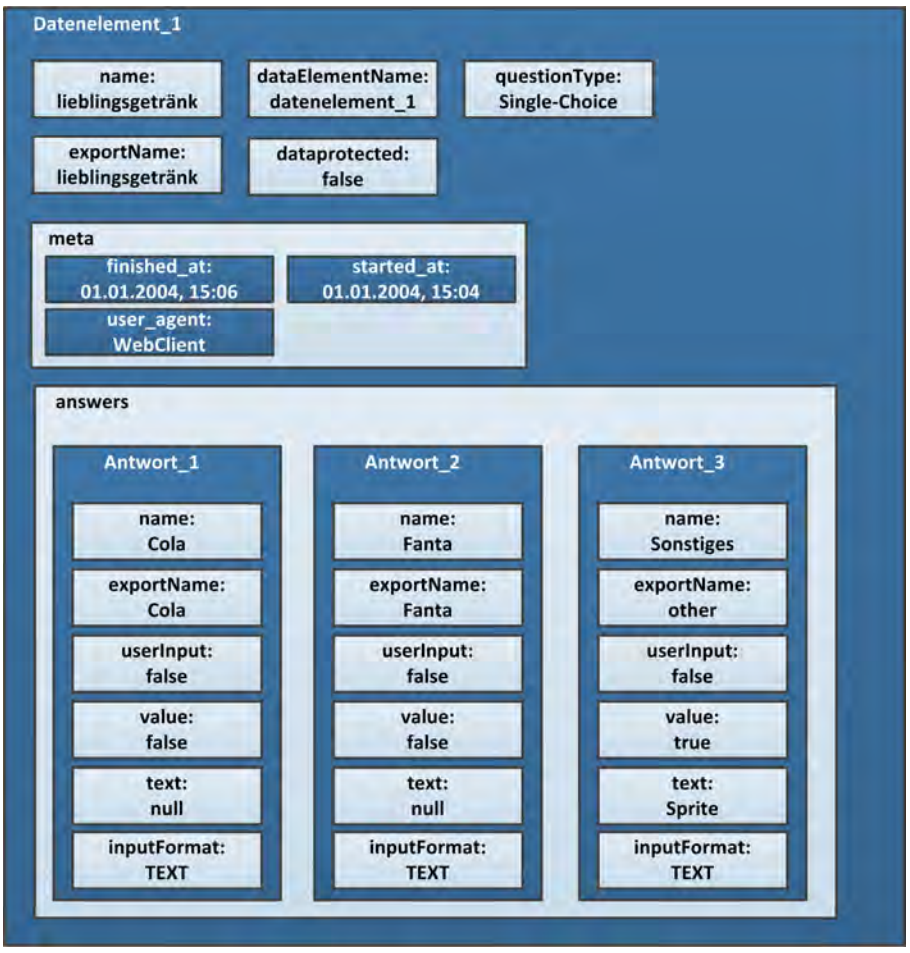

Abbildung 4.4: Struktur eines Datenelements zu einer Single-Choice Frage: *"Welches ist ihr Lieblingsgetränk?"* Der Benutzer hat die Antwort *Sonstiges* ausgewählt und als userInput *Sprite* eingegeben.

Für die Single-Choice Frage *"Welches ist ihr Lieblingsgetränk?"* stehen die Antwortmöglichkeiten *Cola*, *Fanta* und *Sonstiges* zur Auswahl. Dabei wurde *Sonstiges* ausgewählt und als individueller Text *Sprite* eingetragen. Aus den Metainformationen geht hervor,

wann die Frage gestartet und beendet wurde. Zudem wird angegeben, dass die Frage mit einem Web-Client beantwortet wurde.

4.3 Prozessgesteuerte Fragebogenbearbeitung

In diesem Kapitel wird beschrieben, wie ein Fragebogenmodell auf ein Prozessmodell abgebildet werden kann [15]. Dies stellt die Grundlage für die spätere Umsetzung der Fragebögen im System dar. Darauf aufbauend wird der Prozessablauf anhand eines Beispiels erläutert.

Jedes Fragebogenmodell kann im System mehrfach in Form einer Fragebogeninstanz ausgeführt werden. Die einzelnen Fragebogenseiten werden als Aktivitäten dargestellt und die Überführung in andere Aktivitäten werden durch die Entscheidungsfunktionen und Kanten im Fragebogenmodell definiert. Jede Aktivität kann auf Datenelemente zugreifen, welche die Antworten der Fragen auf einer Seite speichern. In Abbildung 4.5 ist die Umsetzung des Fragebogenmodells auf das Prozessmodell dargestellt.

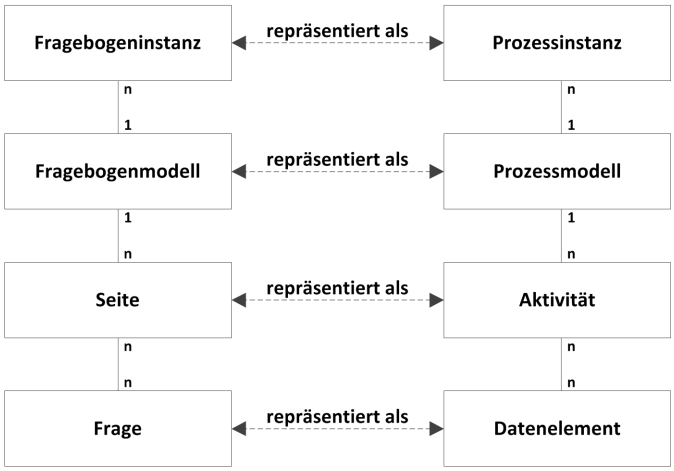

Abbildung 4.5: Abbildung des Fragebogenmodells auf ein Prozessmodell nach [15].

Die Umsetzung einer prozessgesteuerten Fragebogenbearbeitung wird anhand eines *Fragebogens zur Wohnungssuche* verdeutlicht. Das Prozessmodell des Fragebogens

ist in der BPMN 2.0 (Business Process Model and Notation) in Abbildung 4.6 dargestellt [16]. Zusätzlich sind Anmerkungen angebracht, die eine Verbindung zwischen dem Prozess- und dem Fragebogenmodell herstellen.

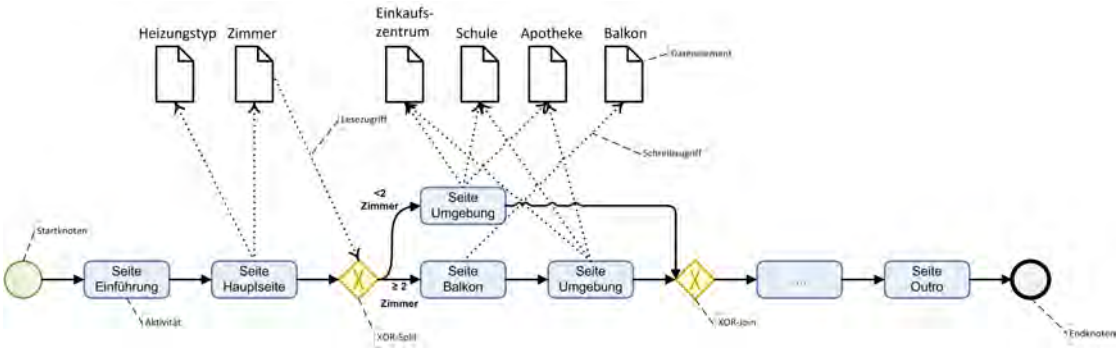

Abbildung 4.6: Anwendungsfall: Ein gekürzter Fragebogen (inklusive Anmerkungen).

Der Fragebogen beginnt mit einem Einführungstext, in dem der Benutzer Informationen zum vorliegenden Fragebogen und gegebenenfalls Anweisungen für die Bearbeitung enthält. Nachdem der Benutzer die nächste Seite aufruft, wird die Aktivität *Seite Hauptseite* angezeigt. In dieser Aktivität werden die Datenelemente `Heizungstyp` und `Zimmer` geschrieben. Dabei speichern die Datenelemente die eingegebenen Antworten des Benutzers für die angezeigten Fragen. Auf die Frage *"Wieviele Zimmer soll die Wohnung besitzen?"* kann der Benutzer seine bevorzugte Zimmeranzahl auswählen.
Nach Bearbeitung dieser Seite wird im nächsten Schritt der `XOR-Split` Gateway aktiviert. Daraufhin wird nach der Auswertung der Entscheidungsfunktionen basierend auf den Antworten des Anwenders ein ausgehender Pfad des `XOR-Split` Knotens aktiviert.
Dazu wird während der Auswertung der Entscheidungsfunktionen das Datenelement `Zimmer` gelesen. Bevorzugt der Benutzer eine Wohnung mit mindestens zwei Zimmern (≥ 2), so wird im nächsten Schritt die Aktivität *Seite Balkon* aktiviert. Hat der Benutzer jedoch weniger als zwei Zimmer (< 2) ausgewählt, so wird die Aktivität *Seite Umgebung* aktiviert. Nach Bearbeitung der angezeigten Fragebogenseiten können dem Benutzer weitere Seiten angezeigt werden, die aus Platzgründen hier nicht weiter aufgeführt wer-

den. Der Fragebogen wird mit der Aktivität *Seite Outro* beendet, in dem ein möglicher Abschlusstext dem Benutzer angezeigt wird.

Abbildung 4.7 zeigt eine beispielhafte Umsetzung einer laufenden Prozessinstanz, in der die Aktivität *Seite Hauptseite* ausgeführt sind. Es können zur gleichen Zeit mehrere Instanzen eines Fragebogens aktiv sein. Das System stellt basierend auf dem Prozessmodell die aktuell benötigte Fragebogenseite als Aktivität dar. Nach dem Beenden dieser Aktivität wird diese durch die nächste Aktivität ausgetauscht. Jede Aktivität kann dabei aus mehreren Elementen bestehen (siehe Kapitel 4.1). Die Aktivität *Seite Hauptseite* besteht beispielsweise aus einer Drop-Down Frage (Zimmer) und einer Single-Choice Frage (Heizungsart). Die Antworten dieser Fragen werden in den entsprechenden Datenelementen gespeichert (siehe Kapitel 4.2).

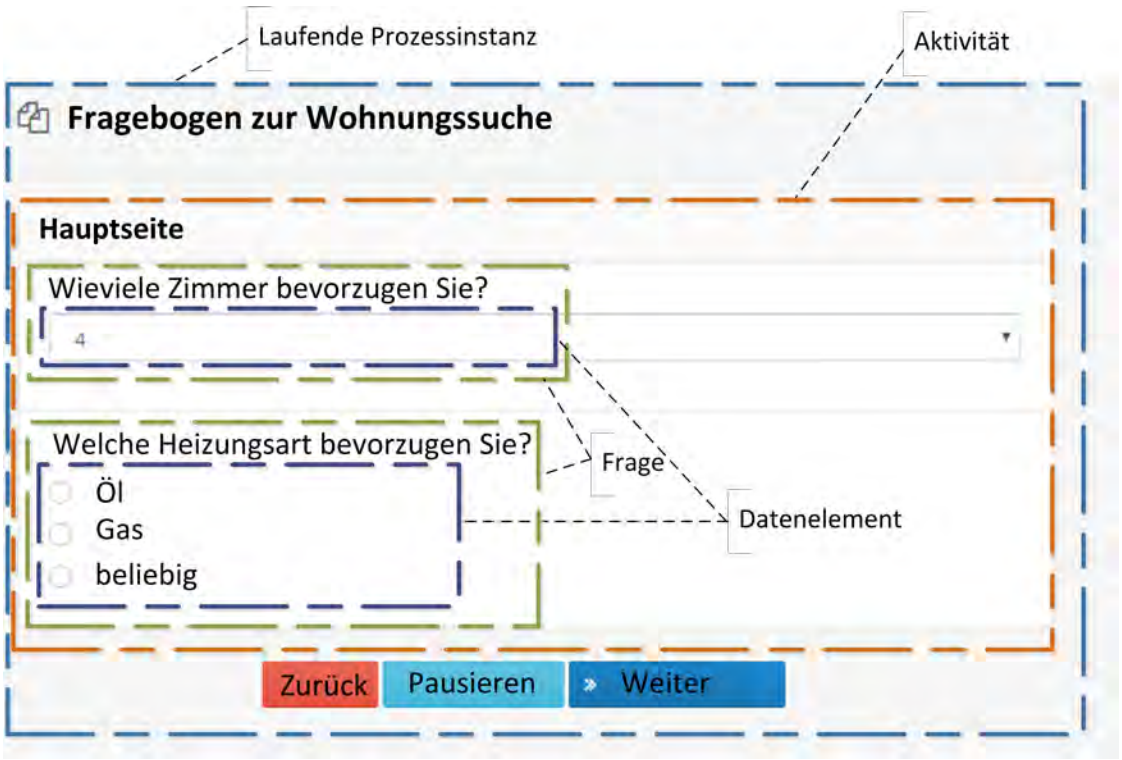

Abbildung 4.7: Darstellung einer laufenden Instanz der Aktivität *Seite Hauptseite* (inklusive Anmerkungen).

4.4 Systemarchitektur

In diesem Kapitel wird die Interaktion zwischen den Komponenten der entwickelten Client-Server-Architektur beschrieben, die für einige Anforderungen aus Kapitel 3.3 benötigt wird. Zwischen dem Client und dem Server findet dabei eine Datenübertragung statt, die in der Regel vom Client initiiert wird. Die Funktionen des Servers umfassen die Datenhaltung, Sicherheitskontrollen und das Bereitstellen von Third-Party-Services wie beispielsweise einen E-Mail-Service. Zunächst werden verschiedene Server-Architekturen und die dazugehörigen Clients diskutiert [17, 18].

Thin Server: Bei einem *Thin Server* liegt die Anwendungslogik und das HTML-Markup komplett auf der Clientseite. Der Server stellt lediglich Daten über eine API bereit. Der Vorteil in diesem Modell ist die hohe Skalierbarkeit durch die geringeren Hardwareanforderungen, da komplexe Funktionen auf der Clientseite ausgeführt werden. Ein weiterer Vorteil ist die geringere Datenauslastung, da die Webseiten auf der Clientseite bereits das gesamte HTML-Markup enthalten und nur die Dateninformationen in Form von JSON/XML vom Server geladen werden müssen. Eine ständige Verbindung zwischen Client und Server ist hierbei nicht zwingend notwendig. Durch die clientseitige Anwendungslogik erhält der Benutzer ein sofortiges Feedback bei durchgeführten Aktionen. Ein Nachteil ist jedoch, dass es bei dem ersten Aufruf der Webseite zu einer höheren Ladezeit kommt, da die gesamte Anwendungslogik mit heruntergeladen werden muss.

Thick Stateful Server: Ein *Thick Stateful Server* beinhaltet die gesamte Anwendungslogik und speichert den aktuellen Zustand des Clients. Der Client dient dabei nur als Benutzerschnittstelle. Bei jeder Anfrage vom Client sendet der Server die gesamte HTML-Seite mit den dazugehörigen JavaScript-Dateien und aktualisiert den eigenen Zustand. Der Nachteil dieser Architektur sind die höheren Hardwareressourcen auf der Serverseite sowie eine höhere Datenkommunikation, welche die Auslastung des Servers erhöht. Ist keine Internetverbindung vorhanden, kann der Client keine weiteren Funktionen ausführen. Demgegenüber steht der Vorteil, dass für den Client geringere

Hardwareressourcen benötigt werden.

Thick Stateless Server: Diese Architektur ist ähnlich zum Thick Stateful Server. Der Unterschied besteht jedoch darin, dass der aktuelle Zustand auf dem Client gespeichert wird und bei jeder Anfrage durch AJAX-Requests mitgesendet wird. Dadurch ist es möglich, nur Teile der Seite zu aktualisieren, wodurch sich die Netzwerkauslastung mitunter drastisch verringert. Ein weiterer Vorteil ist, dass der aktuelle Zustand auf dem Client nicht mehr auf der Serverseite gehalten wird und somit eine Anfrage auf verschiedene Server ausgelagert werden kann.

In dieser Arbeit wird eine Thin Server Architektur umgesetzt. Die gesamte Anwendungslogik liegt dabei auf der Clientseite, die Datenhaltung auf der Serverseite. Für die Anbindung von Web-Services wie zum Beispiel der Versand von E-Mails, die auf der Clientseite nicht implementiert werden können, ist der Server zuständig. Eine Übersicht über die Gesamtarchitektur ist dabei in Abbildung 4.8 dargestellt. Die Clientseite dient dem Benutzer als Benutzerschnittstelle und rendert die Benutzeroberfläche. Datenanfragen vom Client an den Server finden über HTTPS statt, die benötigten Daten werden im JSON-Format zurückgeliefert. Der Server kommuniziert dabei mit der Datenbank und gegebenenfalls mit Web-Services.

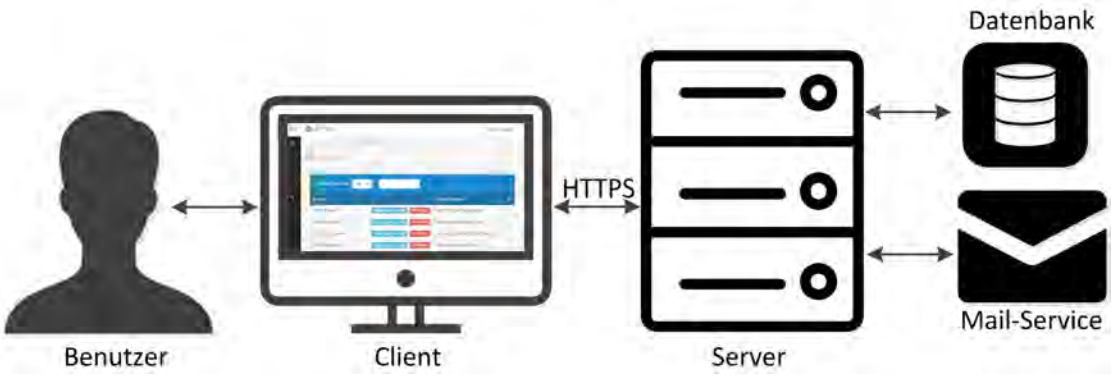

Abbildung 4.8: Übersicht der Gesamtarchitektur.

Im Folgenden wird näher auf die Architektur des Clients und des Servers eingegangen.

4.4.1 Architektur des Clients

Die Architektur des Client beinhaltet die drei Komponenten *View*, *ViewModel* und *Model*. Der Aufbau entspricht dabei der Struktur des MVVM-Patterns, welches eine Weiterentwicklung des MVC-Patterns darstellt [19]. Jede Komponente dieses Patterns hat ihren eigenen Aufgabenbereich, die im Folgenden beschrieben werden. Die Trennung der Aufgaben in verschiedene Bereiche hat den Vorteil, dass Komponenten mehrfach wieder verwendet werden können und sich dadurch die Komplexität verringert. Zudem wird das Hinzufügen neuer Komponenten vereinfacht. In Abbildung 4.9 ist die Architektur dargestellt.

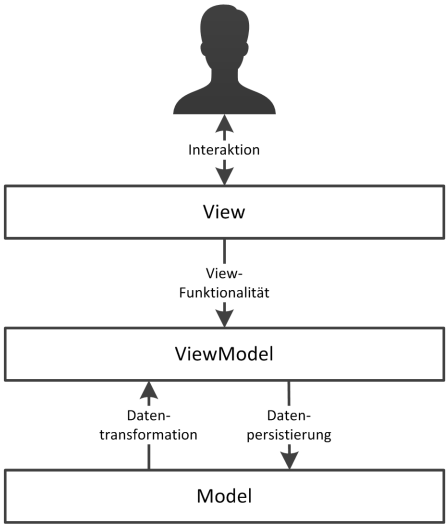

Abbildung 4.9: Das Model-View-ViewModel Pattern (MVVM).

View: Die View hat die Aufgabe, Daten des Models auf der Benutzeroberfläche darzustellen. Ein Beispiel ist die Darstellung einer Fragebogenseite.

ViewModel: Das ViewModel ist für die Funktionalität zuständig und transformiert die Daten, um sie vernünftig verwenden zu können. Außerdem werden hier Funktionen zur Verfügung gestellt, die von der View verwendet werden, wie zum Beispiel die Anwendungslogik für die Behandlung von Interaktionen. Die Kommunikation zwischen dem Server und dem Client wird ebenfalls im ViewModel realisiert.

Model: Das Model repräsentiert die Daten, welche vom Server beziehungsweise vom Client verwendet werden. Über die Kommunikation mit dem Server werden die Daten aktualisiert.

4.4.2 Architektur des Servers

Der Server ist in die Komponenten *Controller*, *Middleware*, *Model* und *Transformer* unterteilt. Die Aufteilung der Aufgabenbereiche in verschiedene Komponenten basiert dabei auf dem Prinzip *Seperation of Concerns*[1] und vereinfacht die Programmierung. Dadurch wird eine übersichtlichere Architektur ermöglicht, wodurch diese weniger fehleranfällig und einfacher zu warten ist. In Abbildung 4.10 ist die Serverstruktur graphisch dargestellt.

Anfragen an den Server werden zunächst an die Middleware-Komponente weitergeleitet. Ihre Hauptaufgabe liegt in der Sicherheitsüberprüfung der einzelnen Anfragen und ist zuständig, nicht autorisierte Anfragen gegebenenfalls abzuweisen. Dadurch werden die Sicherheitsanforderungen aus dem Kapitel 3.4 berücksichtigt. Die Middleware beinhaltet die Komponenten *ApiVersionMiddleware*, *AuthenticateMiddleware* und *RoleMiddleware*. Falls eine Anfrage nur im angemeldeten Zustand durchgeführt werden darf, weist die Komponente AuthenticateMiddleware Anfragen von nicht angemeldeten Benutzern ab. Mithilfe der RoleMiddleware wird überprüft, ob ein Benutzer die benötigte Rolle für eine entsprechende Anfrage besitzt.
Damit das System in Zukunft wartbar und portierbar bleibt, wird die Komponente ApiVersionMiddleware eingesetzt. Damit können in Zukunft Anfragen an neue Versionen des Systems gesendet werden. Neue und stark veränderte Funktionen können somit einfacher implementiert werden, ohne dass Clients mit älteren Versionen nicht mehr auf das System zugreifen können.
Erlaubte Anfragen werden vom Dispatcher an die jeweiligen Controller weitergeleitet. Im Controller wird auf die Datenbank zugegriffen, welche die Informationen als Model zurückliefert. Controller können zudem auf sogenannte *Transformers* zugreifen, welche die Daten des Models auf eine bestimmte Art und Weise transformieren. Dadurch kön-

[1] zu dt. „Trennung der Belange“

nen dem Client nur die Daten gesendet werden, die er benötigt. Die Steuerung der Third-Party-Services geschieht ebenfalls über die Controller. Ein Beispiel für eine Third-Party-Service Funktionalität ist etwa der Versand von Erinnerungsbenachrichtigungen per E-Mail an den Benutzer (siehe Kapitel 3.3).

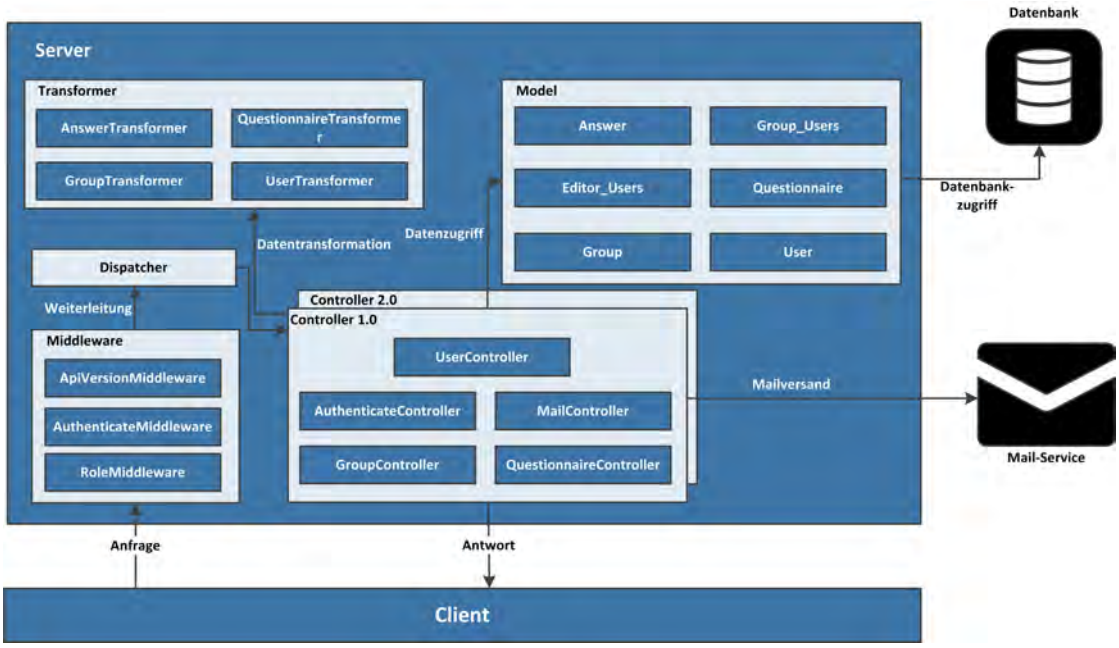

Abbildung 4.10: Struktur der Serverseite

4.4.3 Datenmodell

Das Datenmodell basiert auf den Anforderungen aus Kapitel 3 und stellt einen wichtigen Grundbaustein für die spätere Systemumsetzung dar. Das verwendete Datenmodell wird in Abbildung 4.11 beschrieben. Dabei wird ein relationales Datenmodell verwendet, welches als ER-Diagramm in der Krähenfußnotation dargestellt ist [20]. Im Folgenden werden die wichtigsten Entitäten des Datenmodells erklärt.

user: Das Modell `user` beinhaltet alle Informationen über einen Nutzer. E-Mail, Passwort und Name und Sprache werden bereits bei der Registrierung gespeichert und können vom Nutzer verändert werden. Die Spalte `options` speichert dabei die Informationen zu Sprache und enthält weitere individuelle Nutzereinstellungen wie zum Beispiel die

E-Mail-Abonnierung. Für die Aktivierung des Benutzers während der Registrierung ist zudem das Attribut `is_temp` relevant, die Informationen zum Aktivierungslink enthält. Die Spalte `login_hash` wird für die Passwort-Vergessen Funktion verwendet. Falls ein Editor einen neuen Nutzer während dem Versand eines Fragebogens zum Fragebogensystem einlädt, wird mit dem Attribut `requires_password` dem Nutzer angezeigt, dass er noch ein Passwort setzen muss.
Die Attribute `created_at` und `updated_at` werden zu jedem neu erstellten Datensatz im gesamten Datenmodell erstellt und enthalten Informationen zur Erstellung und Veränderung der Datensätze. Nutzer, die vom Administrator gelöscht wurden, werden mit den Attributen `is_hidden` und `hidden_at` markiert.

roles und **role_users:** Die Modelle `roles` und `role_users` enthalten Informationen zu den Rollen im System. Rollen können einen Namen, eine Beschreibung sowie eine Abkürzung enthalten. Über `role_users` erfolgt die Zuteilung der Rollen an die Nutzer.

groups und **group_users:** In diesen beiden Modellen werden die Informationen der Gruppen gespeichert. Eine Gruppe ist einem Editor zugewiesen und enthält einen Titel und eine Beschreibung. Über `group_users` werden die Nutzer in die Gruppen zugewiesen.

questionnaire: Das Modell `questionnaire` speichert die Informationen zu den hochgeladenen Fragebögen. Ein Fragebogen wird dabei einem Editor zugewiesen. Zudem besitzt ein Fragebogen einen Titel und eine Beschreibung. Der gesamte Inhalt eines Fragebogens wird in dem Attribut `body` gespeichert. Wird ein Fragebogen gelöscht, so wird dieser entsprechend mit den Attributen `is_deleted` und `deleted_at` markiert. Die Einstellungen des Fragebogens werden unter `options` gespeichert.

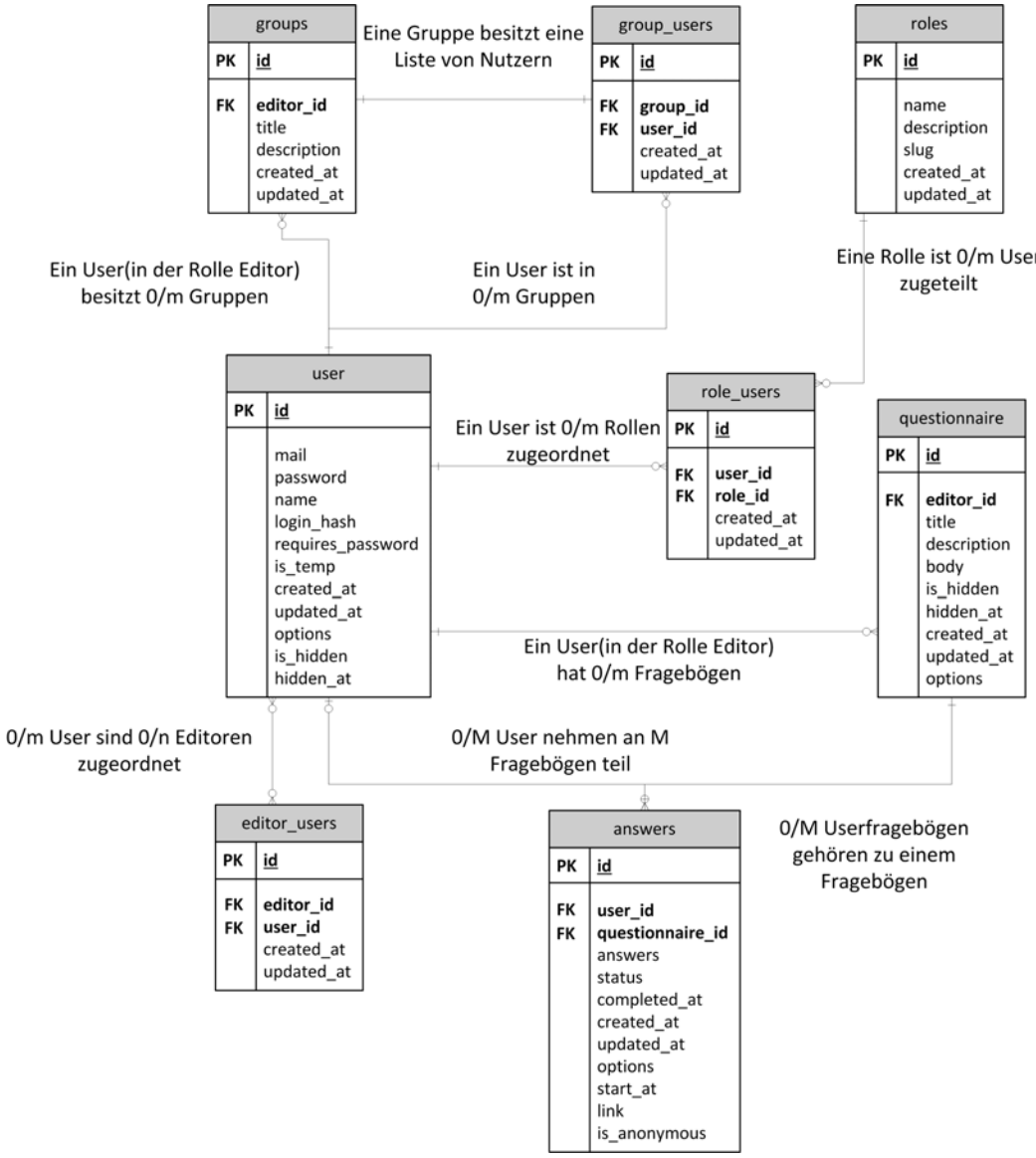

Abbildung 4.11: Datenmodell des Servers basierend auf den Anforderungen aus Kapitel 3.

answers: Die Antworten der bearbeiteten Fragebögen werden im Modell `answers` gespeichert. Jedes Datenelement wird einem Nutzer und einem Fragebogen zugeordnet. In der Entität `options` werden die Einstellungen des Fragebogens gespeichert. Damit betreffen Änderungen in den Einstellungen des Fragebogens keine früheren

Antwort-Datensätze. Die Antworten zu den Fragen sind im Attribut `answers` gespeichert. Der Status der Bearbeitung und das Abschlussdatum werden in `status` und `completed_at` gespeichert. Mit der Spalte `started_at` kann das System regelmäßig überprüfen, ob die Zeit für die Bearbeitung eines Fragebogens bereits abgelaufen ist. Mit *link* können die Fragebögen per vom jeweiligen Benutzer erreicht werden und *is_anonymous* gibt an, ob dieser Fragebogen anonym bearbeitet werden soll.

4.5 Graphische Benutzeroberfläche

In diesem Kapitel wird die graphische Benutzeroberfläche des Systems dargestellt. Aufbauend auf Kapitel 3.5 werden die darin beschriebenen Styleguides in der Benutzeroberfläche berücksichtigt und umgesetzt. In Abbildung 4.12 sind die verschiedenen Oberflächen basierend auf den Anforderungen aus Kapitel 3.3 dargestellt und nach den definierten Rollen aus Kapitel 3.2 unterteilt. Die graphische Oberfläche aller Seiten sind in den Anlagen enthalten.

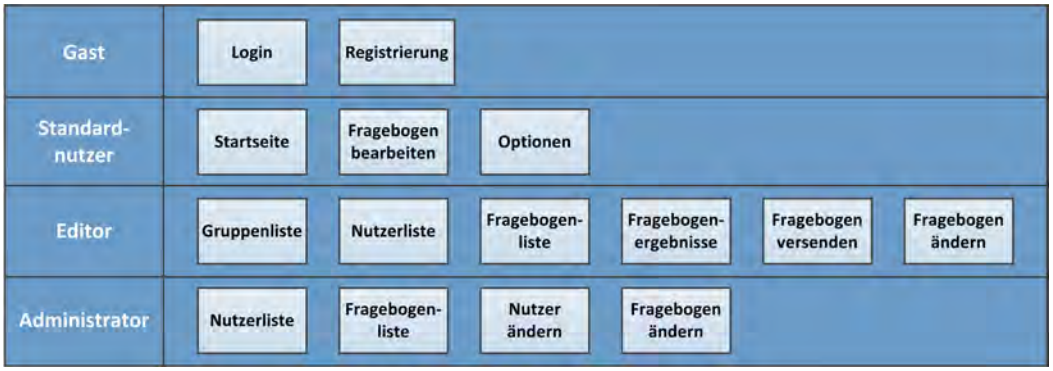

Abbildung 4.12: Übersicht aller Benutzeroberflächen im System.

Im Folgenden werden anhand einiger ausgewählten Benutzeroberflächen wichtige Designaspekte erläutert. Weiterhin beruhen viele designtechnische Aspekte auf [21] und [22].

Der Grad der Benutzerfreundlichkeit des Systems ist unter anderem abhängig von der Umsetzung des Layouts. Das Layout soll über die verschiedenen Seiten möglichst

konsistent aufgebaut sein, damit sich ein Nutzer schnell zurecht finden kann. In Abbildung 4.13 ist die *Startseite* der Anwendung dargestellt. Die Startseite kann grob in die Bereiche *Übersichtsleiste*, *Navigationsleiste*, *Breadcrumbs* und *Inhalt* unterteilt werden. Abgesehen von der Registrierungs- und der Loginseite, welche ein anderes zu sich konsistentes Layout verwenden, ist jede Seite in die oben genannten Bereiche unterteilt.

Die Übersichtsleiste zeigt auf der rechten Seite das Logo des Systems an, wodurch das System eindeutig für den Nutzer erkennbar wird [22]. In der Navigationsleiste kann der Benutzer auf die Funktionen aus Kapitel 3.3 zugreifen. Zum Beispiel kann er über die Kategorie *Administrator* alle Fragebögen anzeigen, einen Fragebogen bearbeiten oder die Nutzerliste einsehen. Die Menüleiste kann zudem zugeklappt werden, um den Fokus auf den eigentlichen Inhalt zu richten. In der Navigationsleiste ist zudem eine Kurzbeschreibung des Nutzers dargestellt, mit der er über die zugewiesenen Rollen Auskunft erhält und auf die Optionen zugreifen kann. Die Aufteilung der Funktionen in die Bereiche *Administrator*, *Editor*, *Fragebögen* und *Ergebnisse* helfen dem Benutzer, Funktionen schneller zu finden. Für oft verwendete Funktionen wie zum Beispiel die *Einsicht der Ergebnisse* sind die notwendigen Interaktionsschritte minimal gehalten. Weiterhin wird die aktuell verwendete Funktion in der Navigationsleiste hervorgehoben, damit der Benutzer sich besser im System zurechtfindet. Zusätzlich werden dem Benutzer in der Breadcrumbs-Navigation die aktuelle Verzweigung angezeigt, in der er sich gerade im System befindet.

Der eigentliche Inhalt der Seite beansprucht den Großteil der Fläche. Der Inhalt kann optional einen Titel besitzen, mit dem die Aufgabe des Hauptinhalts klarer definiert sind. Auf der Startseite werden beispielsweise für den Editor alle Fragebögen aufgelistet. Diese Fragebögen sind in den Kategorien *Abgeschlossen*, *In Bearbeitung*, *Offen* und *Abgelehnt* unterteilt. Der Editor kann hier auf die Ergebnisse der Fragebögen zugreifen sowie die Nutzerliste nach dem Status der Fragebögen einsehen.

In der Nutzerliste hat der Editor die Möglichkeit, Erinnerungsmails und abgelehnte Fragebögen wiederholt an Nutzer zu senden. Ebenfalls wird der aktuelle Tab hervorgehoben, den der Editor zurzeit offen hat. Damit der Editor bestimmte Fragebögen schneller finden kann, existiert zudem eine Suchfunktion.

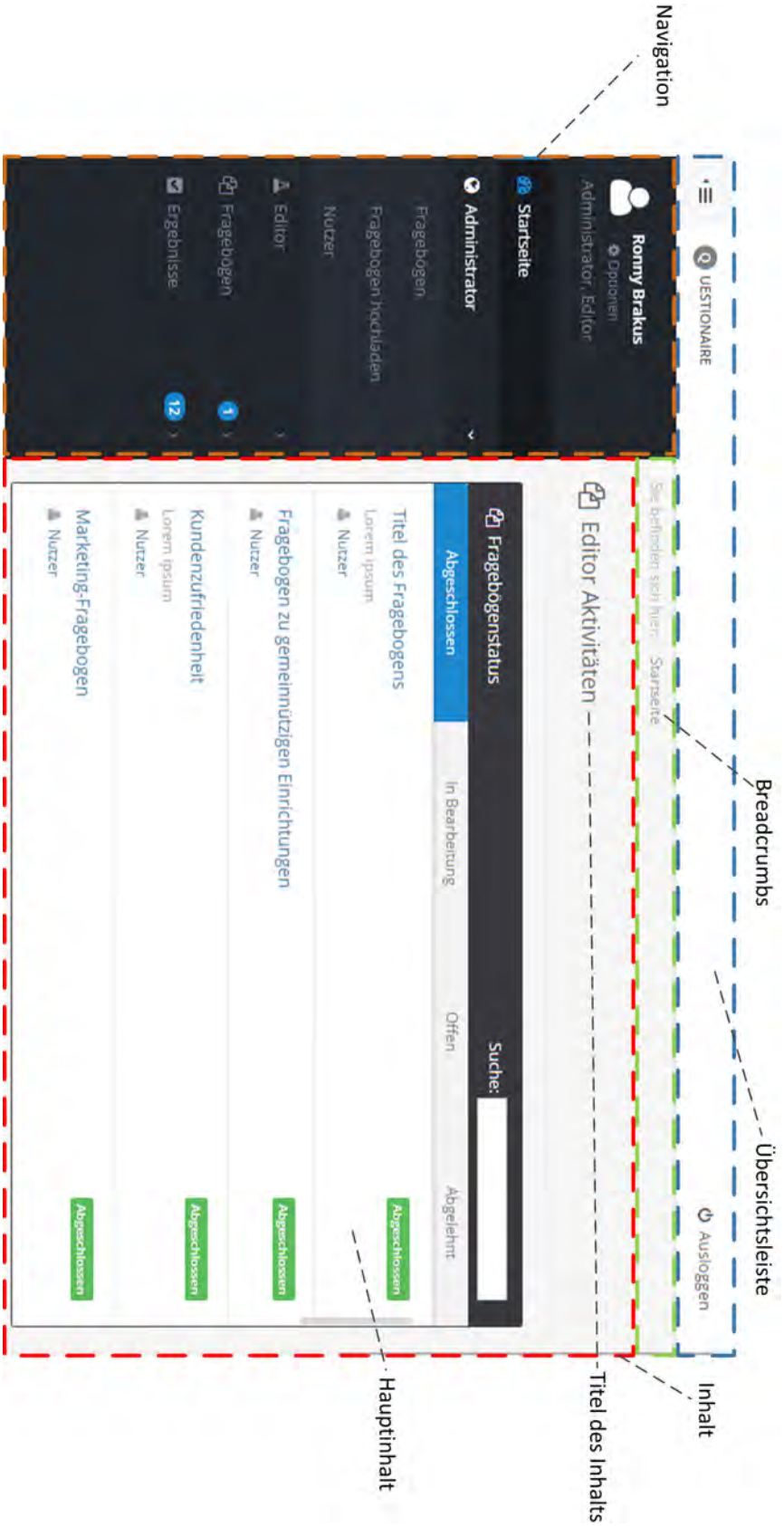

Abbildung 4.13: Graphische Oberfläche der Seite *Startseite* auf einem Desktop-Endgerät mit Anmerkungen.

In Abbildung 4.14 ist die Seite *Nutzerliste des Editors* dargestellt. Der Editor kann hier seine eigenen Nutzer sehen, diese Löschen oder Fragebögen versenden. Zudem kann er neue Nutzer hinzufügen.

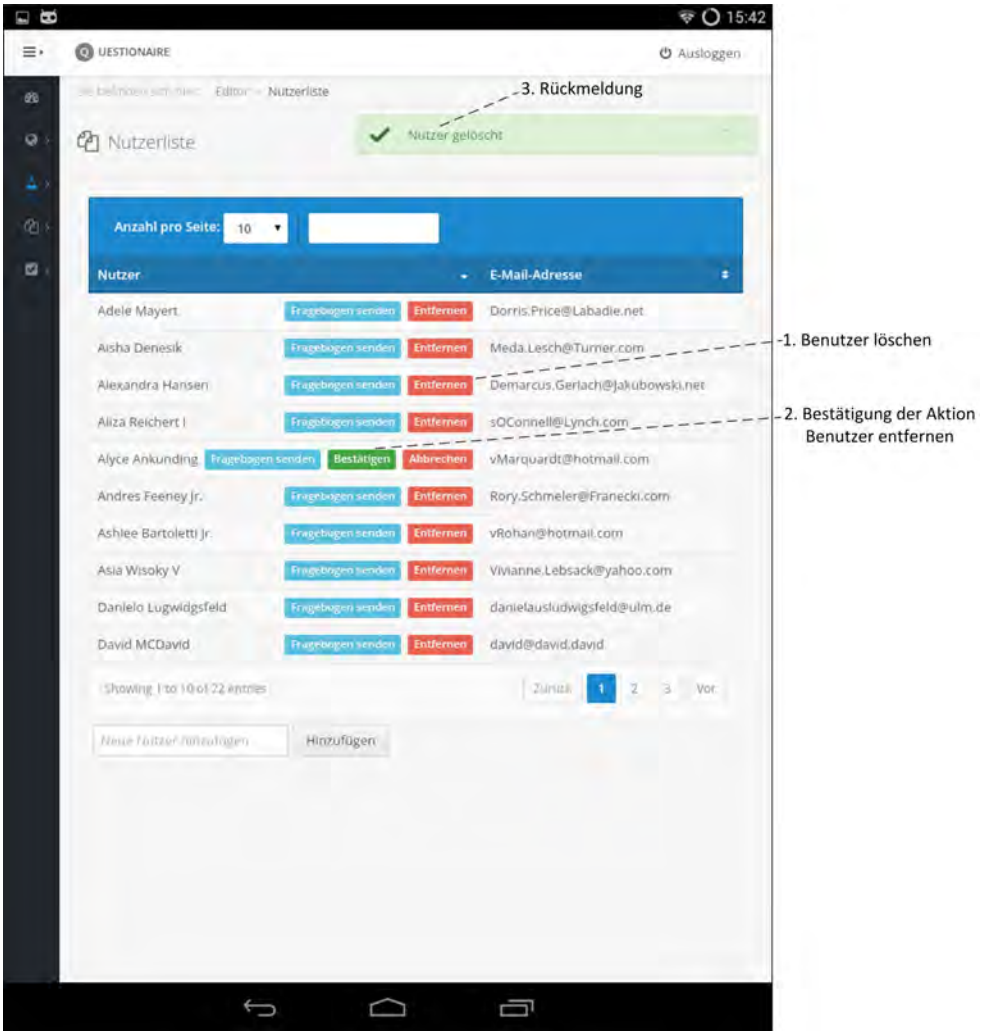

Abbildung 4.14: Graphische Oberfläche der Seite *Nutzerliste* auf einem mobilen Endgerät mit Anmerkungen.

Um das Layout konsistent zu halten, wird auf allen Seiten, die eine Liste von Nutzern, Gruppen oder Fragebögen darstellen, das gleiche Tabellenlayout verwendet. Weiterhin wird darauf geachtet, dass zwischen anliegenden Aktionsbuttons ein größerer Abstand vorhanden ist, damit der Benutzer auf mobilen Endgerät die richtige Aktion durchführen

kann und nicht versehentlich eine falsche oder ungewollte Aktion ausführt.
Bei kritischen Aktionen, wie zum Beispiel dem Löschen von Objekten (siehe 1. Schritt in Abbildung 4.14), muss der Benutzer diese Aktion bestätigen (siehe 2. Schritt in Abbildung 4.14). Bestätigt der Nutzer die Aktion, so erhält er nachfolgend eine Rückmeldung (siehe 3. Schritt in Abbildung 4.14). Im System werden erfolgreiche Aktionen immer grün dargestellt und fehlgeschlagene Aktionen rot.
Abbildung 4.15 zeigt die graphische Oberfläche der Seite *Fragebogen senden* auf einem Tablet.

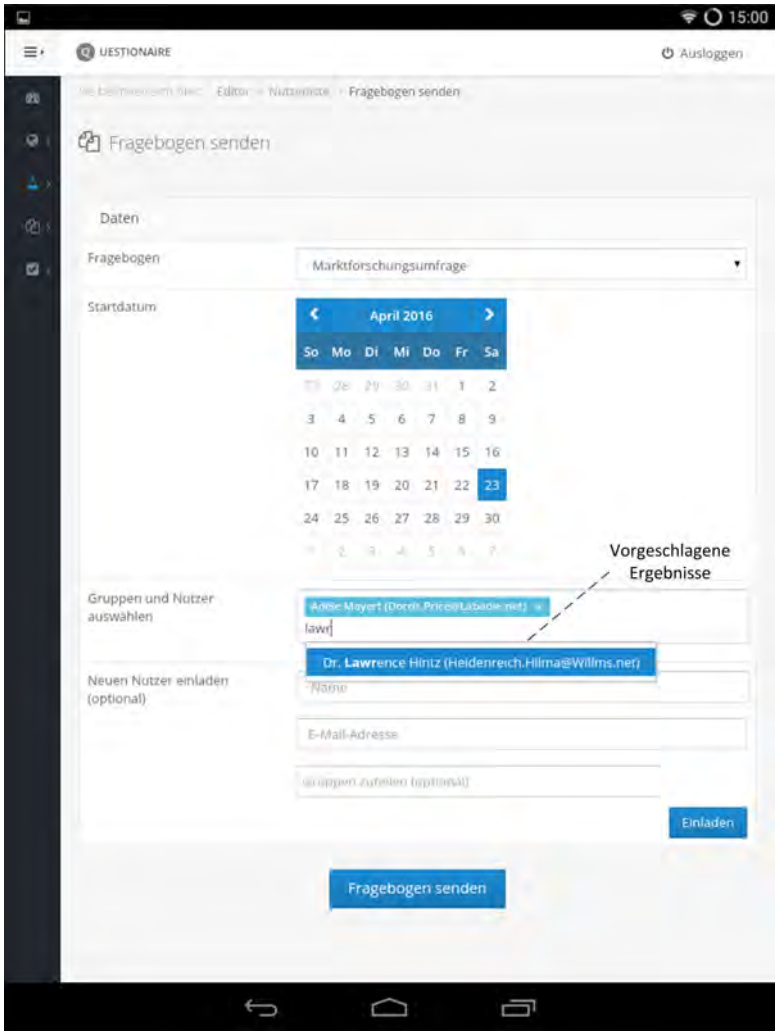

Abbildung 4.15: Graphische Oberfläche der Seite *Frabogen senden* auf einem mobilen Endgerät mit Anmerkungen.

Der Editor kann auf dieser Seite über die Seiten *Nutzerliste* aus Abbildung 4.14, *Gruppenliste* sowie *Fragebogenliste* gelangen. Im dargestellten Formular kann der Fragebogen, die Nutzer und die Gruppen ausgewählt werden. Zusätzlich kann das Startdatum des Fragebogens eingestellt werden. Neue Nutzer können ebenfalls eingeladen und optional in Gruppen eingeteilt werden. Im Hinblick auf die Benutzerfreundlichkeit wurden folgende Punkte zusätzlich beachtet: Gelangt der Editor über die Seite *Nutzerliste* auf die Seite, so wird der ausgewählte Nutzer automatisch zu den Fragebogenempfängern hinzugefügt. Gleiches gilt für den Aufruf über die Seiten *Gruppenliste* sowie *Fragebogenliste*. Außerdem werden beim Eintragen von Nutzern und Gruppen Vorschläge basierend auf der Eingabe angezeigt, die der Editor mit einem Klick übernehmen kann. Für die Benutzerfreundlichkeit wird zudem das Startdatum als Kalender dargestellt.

Abbildung 4.16 stellt die *Loginseite* dar. Der Nutzer kann sich hier in das System anmelden, zur Seite *Registrierung* wechseln und die Spräche ändern. Falls der Benutzer für eine Aktion ein Formular unvollständig ausfüllt, werden die fehlenden Felder rot hervorgehoben. In Abbildung 4.16 wird der Benutzer vom System auf den fehlenden Eintrag im Passwort-Feld aufmerksam gemacht.

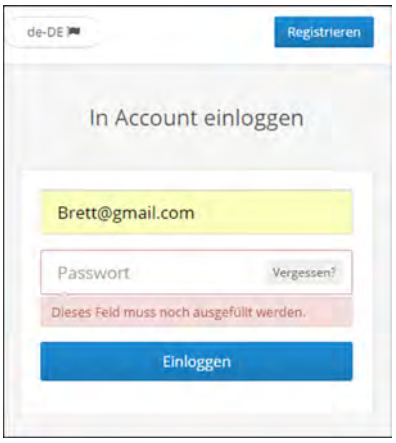

Abbildung 4.16: : Graphische Oberfläche der Seite *Login*.

Nachdem in diesem Kapitel das Konzept und der Entwurf des Systems vorgestellt wurde, wird in Kapitel 5 die Implementierung des Systems erläutert.

5

Implementierung

In diesem Kapitel wird die Umsetzung des Systems anhand der erhobenen Anforderungen aus Kapitel 3 und unter Berücksichtigung des Entwurfs aus Kapitel 4 vorgestellt. Zunächst wird in Kapitel 5.1 verschiedene Frameworks für die Umsetzung des Clients beschrieben und näher auf das Framework AngularJS eingegangen. Anschließend wird das eingesetzte Framework auf der Serverseite in Kapitel 5.2 vorgestellt. Weitere Aspekte der Implementierung wie zum Beispiel die Authentifizierung des Clients am Server oder der eingesetzte E-Mail-Service werden in den Kapiteln 5.3-5.6 vorgestellt. Abschließend wird in Kapitel 5.7 die konkrete Umsetzung des Systems an ausgewählten Beispielen erklärt.

5.1 SPA-Frameworks und AngularJS

Für die Implementierung der Clientseite wird basierend auf Kapitel 4.4.1 ein MV*-Framework verwendet. Dazu existieren diverse Frameworks wie zum Beispiel *AngularJS* [23], *Backbone.js* [24] oder *Ember.js* [25]. Die wichtigsten Besonderheiten dieser Frameworks wie das UI-Binding oder die Wiederverwendbarkeit von Komponenten, die im späteren Verlauf dieses Kapitels diskutiert werden, werden von allen Frameworks unterstützt und sind ähnlich umgesetzt. Im Vergleich zu Backbone.js und Ember.js gibt es in AngularJS eine größere Unterstützung seitens der Community sowie eine große Auswahl an Third-Party-Modulen. Daher wird in dieser Arbeit AngularJS verwendet. Im folgenden wird das Konzept von AngularJS erklärt.

AngularJS ist ein JavaScript-Framework, das für die Entwicklung von dynamischen HTML-Anwendungen eingesetzt wird und für die Entwicklung von SPA's nach dem MVVM-Pattern geeignet ist. Eine HTML-Seite repräsentiert die View im MVVM-Pattern. Dabei kann eine HTML-Seite in einer SPA in verschiedene Bereiche eingeteilt werden, die jeweils eine eigene View darstellen (siehe Kapitel 2.2.1). Für jede dieser Views existieren *Controller*, welche die Funktionalität implementieren und das ViewModel im MVVM-Pattern darstellen. Ein Controller kann dabei für die Funktionalität von mehreren Views zuständig sein. Dies kann zum Einsatz kommen, sobald mehrere Views ähnlich aufgebaut sind, beispielsweise für die Anzeige der Gruppen- und der Nutzerliste. Durch die Wiederverwendbarkeit der Controller reduziert sich der Programmieraufwand und die Wartung des Clients wird vereinfacht. Weiterhin existieren *Services*, die gemeinsame Funktionalitäten implementieren und von mehreren Controllern verwendet werden können. Ein Beispiel dafür ist ein URI-Service, mit der ein Controller eine Anfrage an den Server senden kann. In Abbildung 5.1 ist das Zusammenspiel dieser Komponenten graphisch dargestellt. Der `ListController` ist dabei für die Funktionalitäten der `GroupView` und `EditorUserlistView` zuständig. Der `MainController` implementiert die Logik in der `MainView`. Für Anfragen an den Server verwenden beide Controller den Service `URI-Service`.

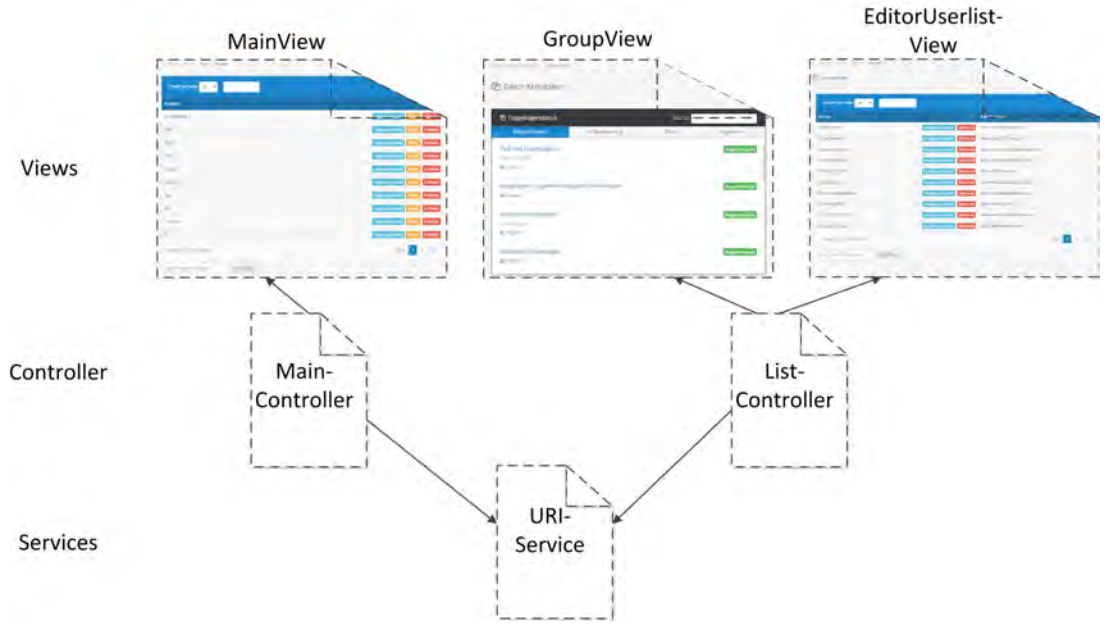

Abbildung 5.1: Controller implementieren die Funktionalität der Views und greifen auf Services für gemeinsame Funktionen zu.

AngularJS ermöglicht die automatische Synchronisierung von Daten zwischen dem Model und der View, die auch als *Zwei-Wege-Datenbindung* [26] bekannt ist. Dazu wird in AngularJS basierend auf einer HTML-Vorlage eine Live-View erstellt. Änderungen an dieser Live-View, beispielsweise durch eine Eingabe des Benutzers, werden sofort vom Model übernommen. Ändert sich das Model, beispielsweise durch eine neue Anfrage an den Server, wird die View ebenfalls aktualisiert. Durch den geringeren Programmieraufwand verringert sich die Komplexität des Clients. Im Gegensatz dazu wird in einer klassischen Webanwendung die View basierend auf einer Vorlage einmalig mit den Daten des Models erstellt. Dieser Vorgang wird als *Ein-Weg-Datenbindung* bezeichnet. Füllt ein Benutzer in dieser View ein Formular aus, so werden die Daten des Models nicht automatisch aktualisiert. Dies erzeugt seitens des Entwicklers einen Mehraufwand, der die Daten der View und des Models synchron halten muss. In Abbildung 5.2 ist der Unterschied zwischen einer Ein-Weg-Datenbindung und Zwei-Wege-Datenbindung graphisch dargestellt.

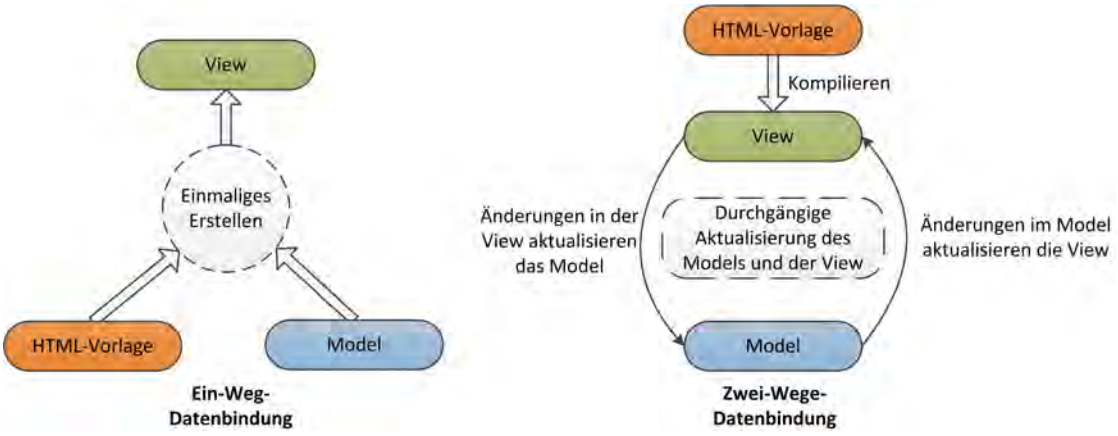

Abbildung 5.2: Unterschied zwischen einer Ein-Weg-Datenbindung und Zwei-Wege-Datenbindung.

Für die Verbindung zwischen der View und dem Model werden sogenannte *Scopes* verwendet [27]. Scopes sind Objekte in der Anwendung, die neben der Verbindung zwischen Model und View unter anderem auch eine Verbindung zwischen Funktionen im Controller und der View herstellen. Wie in Abbildung 5.3 zu sehen ist, wird mit dem Scope-Objekt der Controller `MainController` (1) mit der View (2) verknüpft. Funktionsaufrufe und Model-Daten stehen für gewöhnlich in geschweiften Klammern. Ausnahmen bilden *Direktiven*, mit der eigenes HTML-Markup erstellt werden kann und beispielsweise Schleifen und Controllerzuweisungen implementiert werden können. In unserem Beispiel verweist die Direktive `ng-controller` auf den MainController (1), die im Hintergrund über das Scope-Objekt mit dem Controller verknüpft wird (2). In der Direktive `ng-repeat` (3) werden alle Objekte aus dem Objekt `questionnaire` durchiteriert und das HTML-Markup dazu dem Benutzer dargestellt. Im Controller werden die Objekte und Funktionen mit der Variable `$scope` an die View gebunden. Daraufhin kann die gebundene Funktion `sumQuestionnaires()` (4) mit geschweiften Klammern in der View aufgerufen werden. Diese berechnet beispielsweise die Anzahl der Fragebögen und stellt das Ergebnis in der View dar. In Kapitel `sec:beispiele` wird die Umsetzung der Zweig-Wege-Datenbindung an Beispielen vorgestellt. Im Folgenden wird das eingesetzte Framework auf der Serverseite beschrieben.

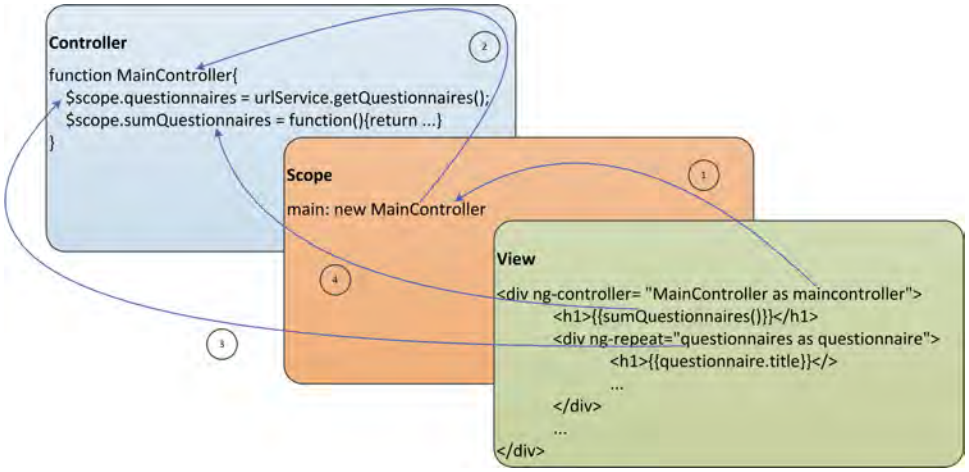

Abbildung 5.3: Einsatz von Scopes für die Verbindung zwischen Model, Funktionen und View (nach [27]).

5.2 Laravel Framework

Für die Implementierung des Servers wird das PHP-Framework *Laravel* [28] verwendet. Der Einsatz von Frameworks ermöglicht eine effizientere Entwicklung, da auf bereits implementierte Funktionen und Bibliotheken zurückgegriffen werden kann [29]. Zusätzlich bieten diese Sicherheitsmechanismen wie zum Beispiel den Schutz vor SQL-Injection oder Verschlüsselung von sensiblen Daten an, die den Zugriff auf nicht autorisierte Daten erschweren.

Laravel ist sowohl für die Entwicklung klassischer Webanwendungen nach dem MVC-Pattern als auch für die Entwicklung von SPA's geeignet [30]. Es ist modular aufgebaut, wodurch die einzelnen Komponenten von Laravel unabhängig aktualisiert, sowie neue Komponenten integriert werden können. Beispielsweise basiert die im System verwendete Token-Authentifizierung (siehe Kapitel 5.5) auf dem `jwt-auth` Modul. Zusätzlich bietet Laravel ein flexibles Routing an, mit der unter anderem Anfragen über Middlewares geleitet werden und dabei gegebenenfalls abgewiesen werden können. Für die Umsetzung der Transformer (siehe Kapitel 4.4.2) wird das Konzept Fractal verwendet. Fractal wird für die Repräsentation und Transformation von Daten eingesetzt und dient

als Vermittlungsschicht zwischen den Daten in der Datenbank und dem Versand der Daten. Der Vorteil von Fractal ist, dass jegliche Änderungen der Datenbankschemas keine Auswirkungen auf die Struktur der versendeten Daten besitzen. Daten werden grundsätzlich im JSON-Format ausgeliefert und können zudem weitere Zusatzinformationen enthalten, wie zum Beispiel die Anzahl der übermittelten Ressourcen. Der Einsatz von Transformers mithilfe von Fractal, das Routing zusammen mit den Middlewares werden in Kapitel 5.7 vorgestellt.

Ein weiterer Vorteil von Laravel ist das Konfigurationsmanagement, in der beispielsweise die geheimen Schlüssel und die Datenbankeinstellungen gespeichert werden. Dies erlaubt eine schnelle Portierung von einem Server auf einen anderen Server. Weiterhin bietet Laravel einen eigenen *ORM (Object-relational mapping)* sowie einen *Query Builder* namens *Eloquent* an, mit der ohne SQL-Anweisungen auf die Datenbank zugegriffen werden kann. Dabei werden bekannte Datenbanksysteme wie MySQL, SQLite oder PostgreSQL unterstützt, die ohne Änderungen des Codes ausgetauscht werden können. Für die Verschlüsselung von sensiblen Daten auf dem Server wie zum Beispiel die Antworten zu Fragebögen, wird AES ([31] eingesetzt.

5.3 Bootstrap

Bootstrap [32] ist ein HTML-, CSS- und JavaScript-Framework, mit der sich responsive Webanwendungen entwickeln lassen. Es bietet eine breite Unterstützung durch verschiedene Gestaltungsvorlagen für Schriftarten, Formulare, Tabellen, Navigationselemente und vieles mehr an. Durch den Einsatz von Bootstrap lassen sich Seiten auch optimal für verschiedene Endgeräte darstellen, wodurch die Anforderungen aus Kapitel 3 berücksichtigt werden.

5.4 JavaScript Object Notation (JSON)

Für die Übertragung von Daten wird das für Menschen und Maschinen lesbare, leichtgewichtige Datenformat *JavaScript Object Notation (JSON)* [33] verwendet, welches für

die Übermittlung von Daten zwischen dem Client und dem Server zum Einsatz kommt. Dabei besteht eine JSON-Datei aus mehreren Schlüssel-Werte Paaren. Der Schlüssel wird als String gespeichert, mit dem anschließend auf den dazugehörigen Wert zugegriffen werden kann. Die Werte können dabei Strings, Zahlen, Boolesche Werte, Arrays oder aus JSON-Dateien bestehen. Ein Beispiel für eine JSON-Datei ist in Quelltext 5.1 dargestellt. Diese beinhaltet Informationen über den angemeldeten Nutzer.

```
1 {  "id":8,
2    "name":"Ronny Brakus",
3    "email":"Brett@gmail.com",
4    "settings":"{\"language\":\"de-DE\",\"mailAbo\":\"true\"}",
5    "roles":[
6       {
7          "id":1,
8          "name":"Administrator",
9       }]}
```

Quelltext 5.1: JSON-Datei mit Nutzerinformationen.

5.5 Token Based Authentication

Dieses Kapitel behandelt *JSON Web Token (JWT)* [34], das für die Authentifizierung der Nutzer verwendet wird. Zunächst wird der Aufbau eines JWT-Tokens erklärt. Anschließend wird JWT mit der bisher verbreiteten Authentifizierungsmethode basierend auf Cookies verglichen.

Ein JWT-Token ist ein Base64 kodierter String, welches ein JSON-Objekt beinhaltet. Es ist in die drei Abschnitte `Header`, `Payload` und `Signature` aufgeteilt. Abbildung 5.4 stellt die drei Abschnitte graphisch dar. Der `Header` enthält grundlegende Informationen über das JWT-Token. Es beinhaltet die Information, dass es sich bei dem kodierten String um ein JWT-Token handelt und gibt den verwendeten Algorithmus für die Signierung der `Signature` an. Die `Payload` selber enthält weitere Metainformationen, wie zum Beispiel eine eindeutige Identifizierung des Nutzers. Die Identifizierung des Nutzers

kann dabei beispielsweise über die im System gespeicherte ID des Nutzers erfolgen. Folgende Metainformationen sind zwingend anzugeben:

- **sub** (Subject): Eindeutige Identifikation des Nutzers,
- **iat** (Issued at): Angabe in Form eines Zeitstempels über das Erstellungsdatum,
- **exp** (Expiry): Angabe in Form eines Zeitstempels über das Ablaufdatum,
- **nbf** (not before): Angabe in Form eines Zeitstempels über die frühest mögliche Nutzung,
- **iss** (Issuer): Angaben über den Herausgeber,
- **jti** (JWT id): Eine eindeutige ID des JWT-Tokens basierend auf sub und iat.

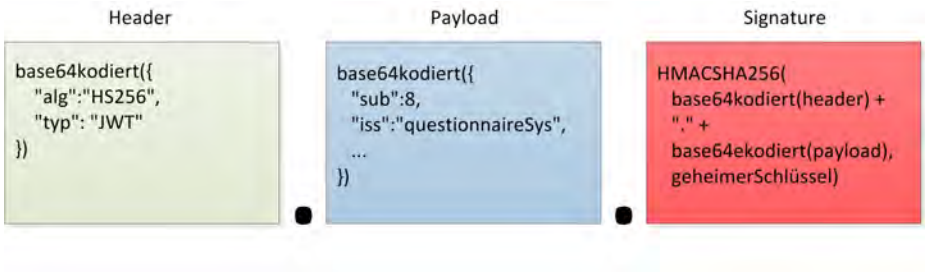

Abbildung 5.4: Aufbau eines JWT-Tokens.

JWT erlaubt zusätzlich die Verwendung weiterer Metainformationen in der `Payload`, etwa zusätzliche Informationen über den Nutzer oder dessen Rollen. Werden sensitive Daten in der `Payload` übertragen, können diese mit *JSON Web Encryption (JWE)* [35] verschlüsselt werden. Die `Signature` dient zur Authentifizierung des Nutzers. Diese wird mit dem im `Header` angegebenen Algorithmus verschlüsselt. Die verschlüsselte Signatur setzt sich aus dem kodierten String des `Headers` und der `Payload` zusammen, die jeweils durch einen Punkt voneinander getrennt sind.

Die Verwendung von JWT zur Authentifizierung ist als Sequenzdiagramm in Abbildung 5.5 dargestellt. Die Kommunikation zwischen Client und Server findet über HTTPS und verhindert Angriffsvektoren wie zum Beispiel Man-in-the-Middle-Angriffe. Für die Adressierbarkeit von Ressourcen wird das Programmierparadigma *Representational*

State Transfer (REST) [36] verwendet. Im ersten Schritt sendet der Nutzer dabei seine Logindaten an den Server. Dieser verifiziert die Daten und sendet bei einem Erfolg dem Client einen JWT-Token. Für jede weitere Kommunikation mit dem Server sendet der Client im Authorization-Header den JWT-Token mit. Beispielsweise wird in Abbildung 5.5nach der Anmeldung am System vom Client ein Fragebogen angefordert. Die Serverseite überprüft die Signatur und falls diese korrekt ist, sendet er dem Nutzer den angeforderten Fragebogen im JSON-Format zurück.

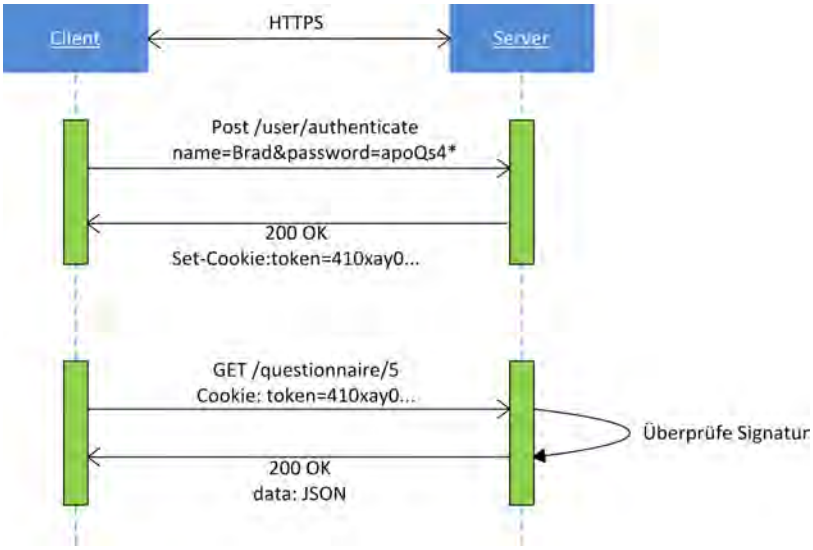

Abbildung 5.5: Authentifizierung des Nutzers mit JWT.

Durch den Einsatz von JWT-Tokens gegenüber regulären Cookies ergeben sich mehrere Vorteile. JWT-Tokens erlauben eine zustandslose Kommunikation, da sie bereits alle Informationen über den Nutzer in sich tragen und keine weiteren Informationen auf dem Server gespeichert werden müssen. Dadurch ergibt sich zudem der Vorteil, dass bei einer hohen Auslastung des Servers Anfragen von Clients auf verschiedene Server ausgelagert werden können und das Gesamtsystem so leichter skalierbar ist. Es wird lediglich der private Schlüssel für die Signatur auf allen Servern benötigt. Typische Angriffsvektoren bei der Verwendung von Cookies wie zum Beispiels *Cross-Site Request Forgery (CSRF)* [37] kommen in JWT nicht vor. Durch die zustandslose Kommunikation

müssen zudem keine Datenbankabfragen zur Überprüfung der Sitzung durchgeführt werden, wodurch der Einsatz von JWT effizienter ist.

5.6 Mail-Service

Für den Versand von Einladungen zu Fragebögen (FA9) oder für die Aktivierung des Nutzers während der Registrierung (FA2) wird ein Mail-Service benötigt. Zum Einsatz kann die bereits in Laravel integrierte Mail-Funktion mit SMTP verwendet werden oder Drittanbieter, wie zum Beispiel Mandrill [38], SendinBlue [39] oder SendGrid [40]. Die verschiedenen Vor- und Nachteile dieser Services sind in Tabelle 5.1 gegenübergestellt.

	Vorlagen	WYSIWYG	Personalisierte Mails	Kostenlos	Serverseitiger Versand
Laravel	•		•	•	•
Mandrill	•	•	•		
SendinBlue	•	•	•	•	
SendGrid	•	•	•	•	

Tabelle 5.1: Vergleich verschiedener Mail-Services.

Die Erstellung von E-Mail-Vorlagen mit HTML-Markup ist in allen untersuchten Diensten möglich. Ein Vorteil von Mandrill, SendinBlue und SendGrid ist die Erstellung der Vorlagen mit einem *What You See Is What You Get (WYSISWYG)* Editor, wodurch Vorlagen schneller erstellt werden können. Zusätzlich bieten Mandrill, SendinBlue und SendGrid eine E-Mail Vorschau für verschiedene Endgeräte an. In allen Services können Vorlagen mit Platzhaltern versehen werden, wodurch sich personalisierte Mails versenden lassen. Der Versand von E-Mails wird in Laravel komplett selbst übernommen, wohingegen in den anderen Mail-Services eine Schnittstelle für den Versand angeboten wird. Im Vergleich zu Laravel, SendinBlue (bis 9.000 Mails pro Monat) und SendGrid (bis 12.000 Mails pro Monat) ist die Verwendung von Mandrill seit 16.03.2016 kostenpflichtig.

Für den Versand von E-Mails wird basierend auf Tabelle 5.1 der externe Dienst SendGrid verwendet. Die Datenübertragung ist niedriger, da unter anderem Bilder bereits in den Vorlagen von SendGrid gespeichert sind und nicht für jeden Nutzer mitgesendet werden müssen. Ein Massenversand ist ebenfalls möglich, sodass nur eine Anfrage an den Service gesendet werden muss und dabei mehrere Personen eine personalisierte E-Mail erhalten.

Die Funktionsweise für die Versendung von E-Mails durch SendGrid ist in Abbildung 5.6 dargestellt. Mit dem Mail-Controller wird ein API-Rest-Aufruf an SendGrid gesendet. Dieser enthält alle Informationen über die Empfänger, Platzhalter sowie die ID der ausgewählten Vorlage. In SendGrid wird die passende Vorlage ausgewählt und die Platzhalter mit den übertragenen Daten ausgetauscht. Anschließend werden die E-Mails an die Empfänger versendet. Die Umsetzung im System wird in Kapitel 5.7 vorgestellt.

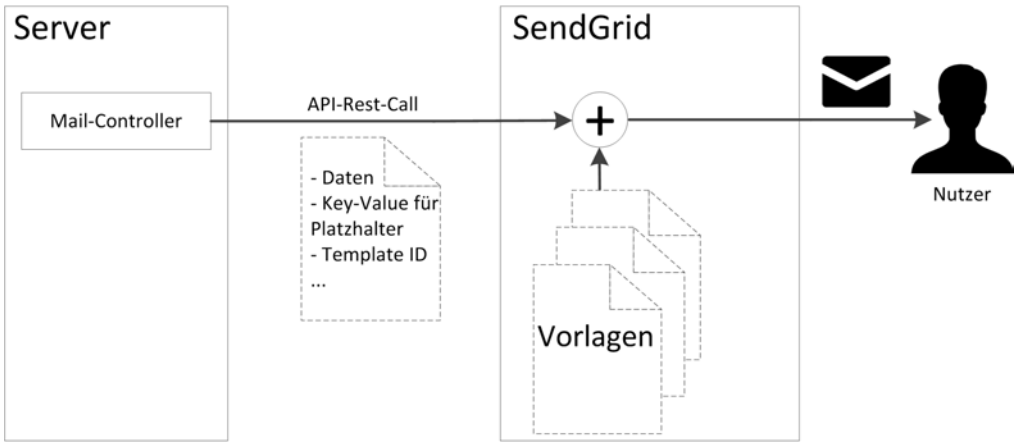

Abbildung 5.6: Funktionsweise von SendGrid.

5.7 Ausgewählte Aspekte der Implementierung

In diesem Kapitel werden ausgewählte Aspekte der Implementierung an einem Anwendungsszenario vorgestellt. Das Anwendungsszenario beruht auf den Anforderungen aus Kapitel 3.3 und beschreibt eine typische Interaktion mit dem System aus Sicht verschiedener Nutzerrollen. Die einzelnen Schritte des Szenarios sind in Abbildung

5.7 dargestellt. Die Implementierungsaspekte werden anhand der Schritte *Login*, *Fragebogen versenden* und *Fragebogen bearbeiten* vorgestellt. Die dabei vorgestellten Funktionen analog können auf die weiteren Bereiche des Systems übertragen werden, sodass ersichtlich wird, wie das Gesamtsystem funktioniert.

Im ersten Schritt (1) meldet sich der Benutzer am System an. Der Benutzer kann hier zusätzlich die von ihm präferierte Sprache wählen oder sich neu registrieren. Nach einer erfolgreichen Anmeldung gelangt der Benutzer auf die Hauptseite (2). Hier kann der Benutzer in der Rolle Editor eine Übersicht zu den Fragebögen einsehen. Mit Klick auf einen Fragebogen gelangt er zu den Fragebogenergebnissen. Zusätzlich kann zu jedem Status der Fragebögen eine Nutzerliste angezeigt werden. In dieser Nutzerliste werden die Nutzer nach dem Status des Fragebogens gruppiert. Im Status *Abgelehnt* und *Abgelaufen* kann der Editor einem Nutzer den Fragebogen erneut senden. Für *offene* Fragebögen kann eine Erinnerungsbenachrichtigung an den Benutzer gesendet werden. Als Standardnutzer können auf der Hauptseite die offenen Fragebögen eingesehen, bearbeitet oder abgelehnt werden.

Im nächsten Schritt (3) versendet der Benutzer in der Rolle Editor einen Fragebogen an mehrere Nutzer. Dazu gelangt er über die Shortcut-Funktion *Fragebogen senden* in der Navigationsleiste auf die zuständige Seite. Hier können Nutzer zum Fragebogen eingeladen und das Startdatum ausgewählt werden. Nachdem der Benutzer einen Fragebogen versendet hat, erhalten alle Benutzer, die eine E-Mail-Benachrichtigung wünschen, eine E-Mail mit einem Link zum Bearbeiten des Fragebogens zum ausgewählten Startdatum. Anschließend können Benutzer die Fragebögen bearbeiten (4). Im Folgenden werden einzelne Aspekte der Implementierung aus Abbildung 5.7 vorgestellt.

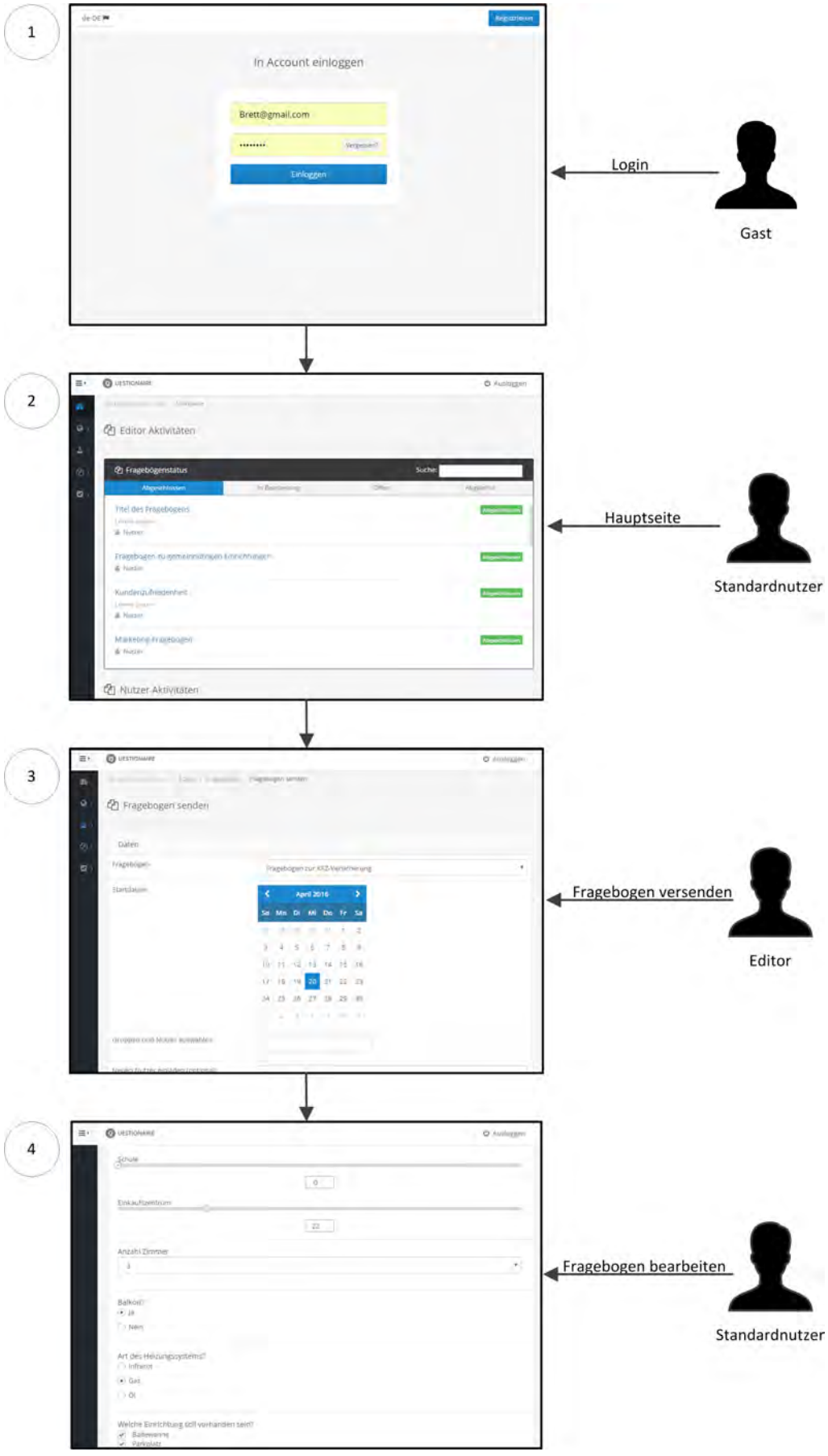

Abbildung 5.7: Die einzelnen Schritte im Anwendungsszenario.

Login

In diesem Kapitel wird die clientseitige Umsetzung der 2-Wege-Datenbindung, Services und das Routing im Client vorgestellt. Weiterhin wird die serverseitige Umsetzung der Middlewares und das Routing erklärt. Zur Übersicht ist die generelle Bearbeitung einer Anfrage auf dem Server in Abbildung 5.8 dargestellt.

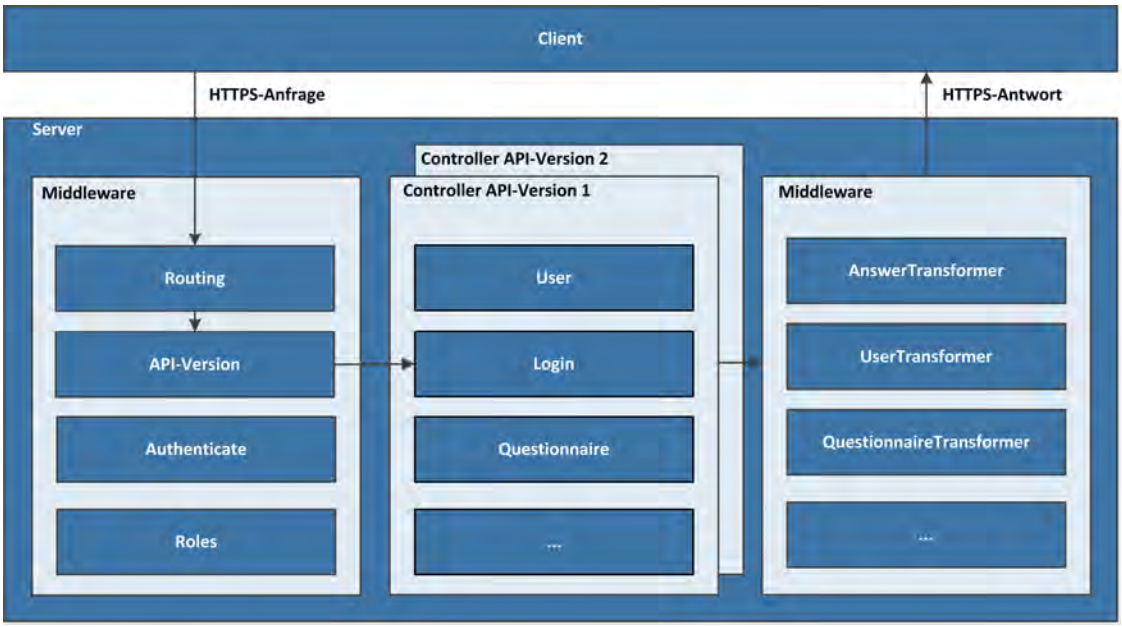

Abbildung 5.8: Lebenszyklus einer Anfrage auf der Serverseite am Beispiel des Logins.

Für die Anmeldung am System sendet der Client zunächst eine Anfrage mit der HTTP-Post Methode an den Server. Sie enthält die Logindaten des Benutzers sowie die ausgewählte API-Version im HTTP-Header. Die Anfrage wird im nächsten Schritt an die Routing-Middleware und anschließend an die API-Version-Middleware geleitet. Die Routing-Middleware verknüpft die Anfrage an die zuständige Methode im Controller. Diese legt zusätzlich fest, ob eine Anfrage weitere Middlewares benötigt, bevor die Anfrage endgültig an die zuständige Methode im Controller weitergeleitet und dort bearbeitet werden kann. Für die Anmeldung am System bearbeitet die API-Version-Middleware die Anfrage und sendet diese an die richtige Version des Controllers. Der Controller überprüft die Daten des Benutzers und falls diese mit den Daten der Datenbank über-

einstimmen, wird ein JWT-Token (siehe Kapitel 5.5) an den Benutzer zurückgeliefert. Zusätzlich besteht die Möglichkeit, dass nach Bearbeitung einer Anfrage durch den Controller weitere Middlewares zugeschaltet werden. Diese werden vom Controller festgelegt. Ein Beispiel dazu wird in Kapitel 5.7 vorgestellt. Nach erfolgreicher Anmeldung wird von der Loginseite zur Hauptseite gewechselt. Im weiteren Verlauf werden zunächst die Funktionalitäten auf dem Client vorgestellt und anschließend die einzelnen Schritte auf dem Server diskutiert.

Nachdem der Benutzer seine Anmeldedaten eingetragen hat und sich anmeldet, wird die Methode `signIn()` im zuständigen Controller aufgerufen. Diese Methode wird mithilfe der 2-Wege-Datenbindung mit der dargestellten Seite verknüpft. Ein Auszug des Controllers ist in Quelltext 5.2 beschrieben. Zeile 1 beinhaltet eine Anzahl von Services, die vom Controller verwendet werden. Dazu zählen beispielsweise die Services zur Authentifizierung (`$auth`), zum Verbindungsaufbau (`$http`) sowie der URI-Service (`uriService`) und der Login-Service (`loginService`).

```
1  function LoginController(growl, urlService, loginService, $auth, $state,
       $http, $rootScope, $scope) {
2    $scope.signIn = function signIn() {
3      if(checkLogin() == false){ return};
4      var credentials = {
5        email: $scope.login_email,
6        password: $scope.login_password
7      }
8      $auth.login(credentials).then(function(response) {
9        return $http.get(uriService.getUserInformation());
10     }, function(error) {
11       // Fehlerbehandlung, Darstellen von Informationen auf dem Client
12     }).then(function(response) {
13         loginService.handleLogin(response);
14     });
15   };
16   ...}
```

Quelltext 5.2: Auszug des Login-Controllers.

Der Authentifizierungsservice beinhaltet Funktionen für die Anmeldung am System und ist zuständig, dass nach einer erfolgreichen Authentifizierung das ausgehandelte JWT-Token sowie die verwendete Version der API bei jeder Anfrage an den Server automatisch gesetzt und mitgesendet werden. In Zeile 3 werden die Angaben des Benutzers auf korrekte Eingaben überprüft. Bei fehlenden oder falschen Eingaben wird mit der Methode `checkLogin()` dem Benutzer eine Fehlermeldung angezeigt. In Zeile 8 wird eine asynchrone Anfrage an den Server mit den Benutzerdaten gesendet. Weitere Nutzerinformationen, wie zum Beispiel die Nutzerrollen und Spracheinstellungen, werden in Zeile 9 vom Server angefragt. Dazu wird der URI-Service verwendet, welcher auszugsweise im Quelltext 5.3 vorgestellt ist. Über diesen Service werden zentral alle URIs verwaltet, um sie bei Bedarf schnell ändern zu können.

In Quelltext 5.2 initialisiert der Login-Service in Zeile 13 die Anwendung basierend auf den erhaltenen Nutzerdaten aus Zeile 9. Abhängig von den Rollen hat der Benutzer Zugriff auf die Administrator- und Editorfunktionalitäten. Zusätzlich wird basierend auf den Spracheinstellungen des Nutzers die Sprache des Clients eingestellt. Nach erfolgreicher Initialisierung im Login-Service wird automatisch zur Hauptseite gewechselt. Da bereits alle Inhalte auf der Clientseite vorhanden sind, findet der Wechsel auf die nächste Seite ohne Ladezeiten statt.

```
1 .service('uriService',[function() {
2   return {
3     getUserInformation : function() {
4       return 'api/user';
5     },
6     ...
7    }}])
```

Quelltext 5.3: Auszug des URI-Services.

Für den Austausch von Seiten wird ein clientseitiger Routing-Service verwendet. Ein Auszug des Services ist in Quelltext 5.4 dargestellt. Ein Wechsel auf die Hauptseite geschieht mit dem Befehl `$state.go('main')`. Die definierte Seite in Zeile 7 wird dabei mit der aktuellen Seite ausgetauscht und für die Funktionalität der neuen Seite wird der definierte Controller aus Zeile 8 verwendet.

```
1 $stateProvider.state('users', {
2     url: '/users',
3     templateUrl: '../views/userView.html',
4     controller: 'UserController as user'
5   }).state('main', {
6     url: '/home',
7     templateUrl: '../views/mainView.html',
8     controller: 'MainController as main'});
```

Quelltext 5.4: Auszug des Routing-Services.

Auf der Serverseite wird die Anfrage zunächst von der Routing-Middleware bearbeitet. Quelltext 5.5 stellt einen Auszug dieser Middleware vor. In der Routing-Middleware können Anfragen nach verschiedenen Kriterien wie zum Beispiel Präfixe oder Nutzerrollen gruppiert und zusätzliche Middlewares eingebunden werden. Beispielsweise durchlaufen alle Anfragen mit dem Präfix `api` (Zeile 1) die API-Version-Middleware (Zeile 2). Anfragen, die nur von authentifizierten Nutzern versendet werden dürfen, müssen zudem die Authentifizierungs-Middleware durchlaufen (Zeile 4). Des Weiteren können Anfragen zusätzlich nach den Nutzerrollen definiert werden, sodass bestimmte Anfragen nur von Administratoren oder Editoren versendet werden können (Zeile 6 und 8).

```
1 Route::group(['prefix' => 'api'], function () {
2     Route::group(['middleware' => 'apiversion'], function () {
3         Route::post('authenticate', '{api-namespace}\
              AuthenticateController@authenticate');
4     Route::group(["middleware" => "jwt.auth"], function () {
5             Route::get('user', '{api-namespace}\
                  AuthenticateController@getAuthenticatedUserInformation');
6       Route::group(["middleware" => "role", "role" => "Administrator"],
            function () {
7         Route::get('admin/editors', '{api-namespace}\
              UserController@editors');});});
8       Route::group(["middleware" => "role", "role" => "Editor"], function
            () {
9         Route::get('editor/groups', '{api-namespace}\
              GroupController@groups');});});});});});
```

Quelltext 5.5: Auszug der Routing-Middleware.

Der Einsatz der Middlewares in Bezug auf die Anmeldung am System wird anhand der API-Version-Middleware erklärt. Quelltext 5.6 stellt die wichtigsten Inhalte dieser Middleware vor. Die angefragte API-Version wird in Zeile 6 gelesen. Die Zeilen 7 bis 11 sind für die Fehlerbehandlung im Fall einer fehlenden (Zeile 7-10) oder nicht gültigen Version (Zeile 11) zuständig. Bei einer gültigen API-Version wird in Zeile 13 der Präfix `api-namespace` durch die übermittelte API-Version ersetzt. Der Präfix entspricht dabei dem in der Router-Middleware zugeordneten Methode für die Anfrage. Abschließend wird in Zeile 15 die Anfrage an den Controller beziehungsweise an die nächste Middleware weitergeleitet. Sendet der Client beispielsweise eine Anfrage zur Authentifizierung mit der Version `v1` im Header, wird der zuständige Methodenaufruf durch `v1\AuthenticateController@authenticate` ersetzt. Die Anfrage wird an die Methode `authenticate()` im `AuthenticateController` der Version 1 weitergeleitet (siehe Quelltext 5.5, Zeile 3).

```
1  class ApiversionMiddleware {
2      public function handle(Request $request, Closure $next)
3      {
4          $route = $request->route();
5          $actions = $route->getAction();
6          $requestedApiVersion = ApiVersion::get($request);
7          if(!$requestedApiVersion) {
8              $errormessage = [...]; // Fehlernachricht mit HTTP-Statuscode
                   400
9              return response(json_encode($errormessage), 400)->header('
                   Content-Type', "application/json; charset=UTF-8");
10         }
11         if (!ApiVersion::isValid($requestedApiVersion)) {//
               Fehlernachricht mit HTTP-Statuscode 400}
12         $apiNamespace = ApiVersion::getNamespace($requestedApiVersion);
13         $actions['uses'] = str_replace('{api-namespace}', $apiNamespace,
               $actions['uses']);
14         $route->setAction($actions);
15         return $next($request);
16     }}
```

Quelltext 5.6: Auszug der Routing-Middleware.

Nachdem die Anfrage durch alle Middlewares bearbeitet wurde, wird die zuständige Methode im Controller aufgerufen. In Quelltext 5.7 ist die zuständige Methode für die Authentifizierung dargestellt. Zunächst werden in Zeile 3 die übergebenen Benutzerdaten aus der Anfrage extrahiert. Sind die Benutzerdaten korrekt, wird in Zeile 6 ein JWT-Token generiert und in Zeile 15 an den Client zurückgesendet.

```
1      public function authenticate(Request $request){
2          // grab credentials from the request
3          $credentials = $request->only('email', 'password');
4          try {
5              // attempt to verify the credentials and create a token for
                   the user
6              if (! $token = JWTAuth::attempt($credentials)) {
7  //                 return response($credentials['password']);
8                  return response()->json(['error' => 'invalid_credentials'
                       ], 401);
9              }
10         } catch (JWTException $e) {
11             // something went wrong whilst attempting to encode the token
12             return response()->json(['error' => 'could_not_create_token'
                   ], 500);
13         }
14         // all good so return the token
15         return response()->json(compact('token'));
16 }}
```

Quelltext 5.7: Auszug der Methode `authenticate()` im Controller `AuthenticateController`.

Fragebogen versenden

In diesem Kapitel wird die Funktionsweise von *Eloquent*, *Transformers* (siehe Kapitel 5.2) und der Versand von E-Mails vorgestellt. Der Editor kann in der Navigationsleiste über die Shortcut-Funktion *Fragebogen senden* einen Fragebogen an Nutzer senden. Für die Darstellung der Seite *Fragebogen senden* werden zunächst drei Anfragen an den Server gesendet, mit der die Ressourcen `Fragebögen`, `Nutzer` und `Gruppen`

des Editors geladen werden. Für die Anfrage der Fragebögen wird in Quelltext 5.8 stellvertretend die Bearbeitung auf dem Server vorgestellt.

```
1 public function editorQuestionnaires(Manager $fractal,
      QuestionnaireTransformer $questionnaireTransformer){
2   $questionnaires =  Questionnaire::where('editor_id',  Auth::user()->id)
      ->where('is_hidden', false)->get();
3   $meta =  $this->buildMetaBlock( $questionnaires,
      $questionnaireTransformer);
4   return $this->response->withCollection($questionnaires, new
      QuestionnaireTransformer, null, null, $meta);
5 }
```

Quelltext 5.8: Methode `editorQuestionnaires()` aus dem Controller `QuestionnaireController`.

Die Abbildung von Daten aus der Datenbank auf Modelle wird mit Eloquent realisiert. In Zeile 2 repräsentiert das Modell `Questionnaire` einen Dateneintrag eines Fragebogens. Anfragen an die Datenbank werden mithilfe des Query Builders bearbeitet, die der SQL-Syntax ähnelt. So werden mit den Methoden `where()` und `get()` alle Einträge aus der Datenbank geladen, die den Bedingungen in den WHERE-Klauseln entsprechen. In Zeile 2 werden alle Fragebögen mit der ID des Editors geladen, die nicht vom Editor gelöscht wurden. In Zeile 3 wird basierend auf den erhaltenen Fragebögen Meta-Informationen erstellt, die beispielsweise die Anzahl der zurückgegebenen Ressourcen beinhalten. In Zeile 4 werden die Daten an die Transformer-Middleware `QuestionnaireTransformer` gesendet, in der die Daten transformiert werden. Ein Auszug der Transformerklasse findet sich in Quelltext 5.9.

```
1 public function transform(Questionnaire $questionnaire)
2 {   return [
3     'id'           => $questionnaire->id,
4     'title'        => $questionnaire->title,
5     'options'      => $questionnaire->options,
6     'description'  => $questionnaire->description
7   ];}
```

Quelltext 5.9: Transformierung von Fragebögen aus der Datenbank.

Jede einzelne Ressource wird mit der Methode `transform()` bearbeitet und enthält die in der Methode angegebenen Informationen (Zeile 4-6). Damit wird außerdem festgelegt, welche Informationen einer Ressource an den Nutzer gesendet werden und in welcher Reihenfolge die enthaltenen Daten aufgebaut sind. Die gesamten Informationen werden zusammen mit den Meta-Informationen anschließend im JSON-Format an den Client übertragen. Quelltext 5.10 stellt den Inhalt einer solchen JSON-Datei vor.

```
1  {
2    "data": [
3      {
4        "id": 1,
5        "title": "Fragebogen zu gemeinnützigen Einrichtungen",
6        "options": "{\"count\":1,\"interval\":0,\"duration\":0}",
7        "description": ""
8      },
9      {
10       "id": 2,
11       "title": "Kundenzufriedenheit",
12       "options": "{\"count\":1,\"interval\":0,\"duration\":0}",
13       "description": "Lorem ipsum"
14     }
15   ],
16   "meta": {
17     "count": 2
18   }
19 }
```

Quelltext 5.10: Erstellte JSON-Datei mithilfe des Transformers `QuestionnaireTransformer`.

Nachdem der Editor die Seite Fragebogen versenden ausgefüllt hat, wird auf der Serverseite die Anfrage bearbeitet. Zu jedem Nutzer wird eine eigene Antwortdatei erstellt und der Status des Fragebogens basierend auf dem Startdatum erstellt. Einladungen werden zum jeweiligen Startdatum an die Nutzer gesendet. Ein Auszug des E-Mail Versands findet sich in Quelltext 5.11.

```
1  public static function email($answers){
2    $sendgrid = new \SendGrid(env("SENDGRID_API_KEY"));
3    $email = new \SendGrid\Email();
4    $recipients = array();
5    $names = array();
6    $subject = array();
7    $body = array();
8    $links = array();
9    foreach($answers as $answer){
10     $user = User::where('id', $answer->user_id)->first();
11     $recipients[] = $user->email;
12     $names[] = $user->name;
13     $subject[] = MailController::getPersonalizedSubject($user);
14     $body[] = MailController::getPersonalizedBody($user);
15     $links[] = MailController::generatePersonalizedQuestionnaireLink(
           $answer->link);
16   }
17   $email
18     ->setSmtpapiTos(array($recipients))
19     ->setFrom(env("SENDGRID_EMAIL_ADDRESS"))
20     ->setTemplateId(env("SENDGRID_INVITATION_TEMPLATE_ID"))
21     ->setHtml(" ") //Activate HTML Template
22     ->setSubject('%subject%')
23     ->addSubstitution("%body%", $body)
24     ->addSubstitution("%name%", $names)
25     ->addSubstitution("%link%", $links)
26     ->addSubstitution("%subject%", $subject);
27   $res = $sendgrid->send($email);
28   return $res->getCode();
29 }
```

Quelltext 5.11: Über Transformer erstellte JSON Antwort.

In der dargestellten Methode werden alle Nutzer übergeben, die eine Einladung zu einem Fragebogen erhalten sollen. Für den Versand wird in Zeile 2 eine Instanz von SendGrid erstellt. In den Zeilen 9 bis 16 wird in einer Schleife für jeden Nutzer eine personalisierte E-Mail erstellt. Dabei werden Titel und Inhalt basierend auf der eingestellten Sprache des Nutzers und ein eindeutiger HTML-Link zum Fragebogen erstellt. Anschließend wird

in den Zeilen 17 bis 26 der Versand vorbereitet. Es wird die zugehörige E-Mail-Vorlage ausgewählt und die Empfänger angegeben. Zusätzlich werden die personalisierten Daten angegeben, welche die Attribute in der E-Mail-Vorlage ersetzen. Ein Attribut ist beispielsweise `%subject%`, welches durch den übergebenen Titel basierend auf den Spracheinstellungen des Nutzers ersetzt wird. Anschließend werden die Daten an die SendGrid-API gesendet. Der erfolgreiche Versand der E-Mails mit SendGrid wird mit dem Statuscode aus Zeile 27 überprüft und anschließend an den Client übermittelt. Abbildung 5.9 stellt eine Einladung zu einem Fragebogen dar, die mit SendGrid erstellt wurde.

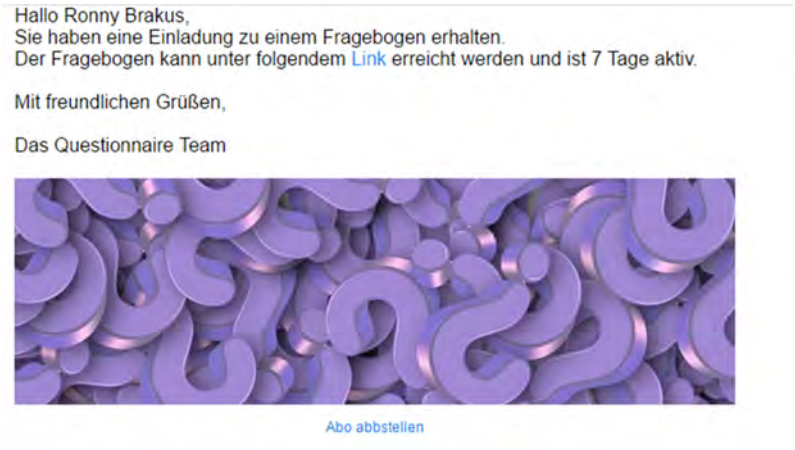

Abbildung 5.9: E-Mail für eine Einladung zur Bearbeitung eines Fragebogens.

Fragebogen bearbeiten

In diesem Kapitel wird die Funktionsweise der Zwei-Wege-Datenbindung und der Einsatz von Direktiven an dem Bearbeiten eines Fragebogens vorgestellt. Ein Nutzer kann über das Item `Fragebögen` in der Navigationsleiste seine vorhandenen Fragebögen einsehen und diese über einen Klick bearbeiten. Zusätzlich kann er, falls der Benutzer die Benachrichtigungen per E-Mail zu neuen Fragebögen aktiviert hat, über einen erhaltenen Link auf den Fragebogen direkt zugreifen. Im Hintergrund fragt der Client das zugehörige Fragebogenmodell und die bisherigen Antworten des Benutzers im System ab. Wurde der Fragebogen bereits zu einem früheren Zeitpunkt bearbeitet, kann der Benutzer

diesen an der entsprechenden Stelle wieder aufnehmen und fortsetzen. Die Bearbeitung eines Fragebogens ist dabei prozessgesteuert umgesetzt (siehe Kapitel 4.3).

Der Aufruf eines Fragebogens startet eine neue Prozessinstanz. Eine Fragebogenseite wird in der Prozessinstanz als Aktivität ausgeführt und in der View dem Benutzer dargestellt. Zunächst wird überprüft, ob der Fragebogen bereits bearbeitet wurde. Falls ja, wird dem Benutzer die zuletzt bearbeitete Fragebogenseite dargestellt. Ansonsten wird mit dem Startknoten begonnen. Blättert der Benutzer zur nächsten Seite, so wird der nächste Knoten im Fragebogenmodell gesucht und je nach Typ des Knotens unterschiedliche Aktionen durchgeführt. Quelltext 5.12 stellt einen Auszug des Programmcodes für die Behandlung des aktuellen Knotens dar.

Handelt es sich um eine Fragebogenseite (Zeile 9-11), so wird diese dem Benutzer dargestellt. Bei einem `XOR-Split` Gateway (Zeile 13-15) wird über Entscheidungsregeln basierend auf den bisherigen Antworten des Benutzers der nächste Knoten bestimmt und mit diesem weiter gearbeitet. Bei einem `XOR-Join` Gateway (Zeile 17-18) wird der ausgewählte Pfad wieder zusammengeführt und es wird mit dem nächsten Knoten weiter gearbeitet. Wurde der Endknoten (Zeile 5-7) erreicht, so werden noch mögliche Abschlussinformationen angezeigt und der Fragebogen beendet.

```
1  switch(currentNodeType){
2    case "NT_STARTFLOW":
3      showDescription(node);
4      break;
5    case "NT_ENDFLOW":
6      showDescription(node);
7      end();
8      break;
9    case "NT_NORMAL":
10     showDescription(node);
11     showActivity(node);
12     break;
13   case "NT_XOR_SPLIT":
14     edgeCode = evaluateXorSplit(node);
15     gotoNextNode(node);
16     break;
17   case "NT_XOR_JOIN":
```

```
18      gotoNextNode(node);
19      break;
20  }
```

Quelltext 5.12: Behandlung des aktuellen Knotens im Fragebogenmodell je nach Knotentyp.

Handelt es sich bei dem aktuellem Knoten um eine Fragebogenseite, so werden die einzelnen Elemente dieses Knotens aus der XML-Datei gelesen und dem Benutzer dargestellt. In Quelltext 5.13 ist ein Auszug der Funktion für die Bearbeitung der einzelnen Fragebogenelemente dargestellt.

Zunächst wird in Zeile 2 und 3 die Elemente der Fragebogenseite ausgelesen und anschließend ab Zeile 6 für die Darstellung vorbereitet. Je nach Typ des Elements werden die Informationen aufbereitet und anschließend im Array `questionnairesData` gespeichert. Als Beispiel dazu wird die Aufbereitung des Elements `question` ab Zeile 16 vorgestellt. In Zeile 18 wird der Fragetyp und in Zeile 19 die eigentliche Frage in das Objekt `questionnaire` gespeichert. Weitere Informationen, wie die einzelnen Unterelemente (beispielsweise eine Auswahl von mehreren Single-Choice Antworten) und Informationen zum Datenelement, sowie die Angabe, ob die Frage bearbeitet werden muss, werden in den Zeilen 20-22 gelesen. Basierend auf diesen Informationen wird ein Antwortobjekt in Zeile 23 erstellt (siehe Kapitel 4.2). Die gespeicherten Elemente werden in Zeile 31 mit der View durch das Objekt `$scope` verknüpft und können dem Benutzer dargestellt werden.

```
1  function showQuestionnaireData(node){
2    var questionnaire = JSON.parse(xml.find("node[id='" + node + "']").find
         ("configurationEntry").text());
3    var elements = questionnaire.elements;
4    var questionnairesData = [];
5    current_questions = [];
6    angular.forEach(elements, function(element, key) {
7      var elementType = element.elementType.id;
8      var questionnaire = {};
9      questionnaire.elementType = elementType;
10     questionnaire.name = element.name;
11     switch (elementType){
```

```
12          case "headline":
13            ... //read element and push it to questionnairesData
14          case "text":
15            ... //read element and push it to questionnairesData
16          case "question":
17            var items = JSON.parse(element.versions[version].
                  typeQuestion.items);
18            questionnaire.questionType = items.questionType;
19            questionnaire.text = getLangTextValue(items.languageTexts);
20            questionnaire.items = items;
21            questionnaire.optional = (element.versions[version].
                  typeQuestion.optional);
22            questionnaire.dataElementName = (element.versions[version].
                  typeQuestion.dataElementName);
23            createAnswerStub(questionnaire);
24            questionnairesData.push(questionnaire);
25            current_questions.push(questionnaire);
26            break;
27          case "media":
28            ... //get Media from Server and push it to questionnairesData
29        }});
30      $scope.questionnairesData = questionnairesData;
31    }
```

Quelltext 5.13: Vorbereitung der Fragebogenelemente für die Darstellung der Fragebogenseite.

Die Verknüpfung der Daten wird durch das `$scope` Objekt realisiert (siehe Kapitel 5.1). Zur Veranschaulichung ist in Quelltext 5.14 ein Auszug der View dargestellt. In Zeile 1 wird die Direktive `ng-repeat` verwendet. `ng-repeat` ermöglicht das Durchlaufen eines Array-Objektes, welches mit der View verknüpft wurde. Dabei wird für jedes Element im Array das enthaltene HTML-Markup dieser Direktive erstellt und dem Benutzer angezeigt.

In Quelltext 5.14 wird jedes Fragebogenelement der Fragebogenseite durchlaufen und basierend auf dem Typen des jeweiligen Elements das richtige HTML-Markup dargestellt. Fragebogenelemente können beispielsweise Fragen, Texte oder Bilder sein. Für Elemente vom Typ `question` wird das HTML-Markup des jeweiligen Fragetyps dargestellt.

Dies geschieht mit den Direktiven `ng-switch` (Zeile 2) und `ng-switch-when` (Zeile 3). In Kombination verhalten sich diese Direktiven wie ein Switch Statement aus bereits bekannten Programmiersprachen. Zeile 4-11 stellt das HTML-Markup für eine Dropdown Frage vor, mit der eine Antwort aus einer Liste ausgewählt werden kann. Mit der Direktive `ng-model` wird die ausgewählte Antwort im Model gespeichert. Die Antwortmöglichkeiten der Frage werden mit der Direktive `ng-repeat` in das HTML-Markup eingebunden (Zeile 7).

In Zeile 9 wird die Methode `getLangTextValue()` aufgerufen, die zuvor mit dem `$scope` Objekt mit der View verknüpft wurde. Mithilfe dieser Methode wird der Fragetext in der richtigen Sprache dargestellt. Bearbeitet der Benutzer den Fragebogen und ändert die Antwort der Dropdown Frage, wird die Methode `updateSelectionByIndex` aufgerufen. Dies wird mit der Direktive `ng-selected` in Zeile 8 ermöglicht, die bei jeder Änderung der Antwort aufgerufen wird.

```
1  <div class="widget-section" ng-repeat="questionnaire in
       questionnairesData">
2    <div ng-switch on="questionnaire.elementType">
3      <div class="" ng-switch-when="question">
4        <div ng-switch on="questionnaire.questionType">
5          <div ng-switch-when="DROPDOWN_CHOICE">
6            <select class="form-control" ng-model="answers[
                 questionnaire.dataElementName].selectedItem">
7              <option ng-repeat="item in questionnaire.items.items"
8              ng-selected="updateSelection(questionnaire.dataElementName)">
9              {{getLangTextValue(item.identifierContent.languageTexts)}}
10             </option>
11           </select>
12       </div>...</div>...</div>...</div>...</div>
```

Quelltext 5.14: Vorbereitung der Fragebogenelemente für die Darstellung der Fragebogenseite.

In diesem Kapitel wurde die Implementierung des Systems anhand ausgewählter Beispiele vorgestellt. Ein abschließendes Fazit und mögliche Erweiterungen des Systems werden in Kapitel 6 diskutiert.

6

Fazit

Papierbasierte Fragebögen sind in der digitalen Zeit kein sinvolles Mittel mehr für die Durchführung von Studien oder Umfragen und der damit verbundenen Erhebung von Daten. Sie stellen einen hohen Ressourcenverbrauch dar, benötigen viel Zeit für die Verteilung und können auch unvollständig bearbeitet werden. Die Auswertung kann zudem durch fehlerhafte Digitalisierung der erhobenen Daten verfälscht werden.
Bisherige existente Fragebogensysteme benötigen oft ein großes Vorwissen für die Erstellung von Fragebögen. Der Einsatz von Experten erfordert einen hohen Koordinationsaufwand und stellt unnötige Mehrkosten dar. Zusätzlich sind die Daten auf Plattformen Dritter gespeichert, was zu datenschutzrechtlichen Problemen führen kann.

Um die oben genannten Probleme zu lösen, wurde im Rahmen dieser Arbeit eine Plattform zur digitalen Datenerhebung entwickelt. Ein Server soll dabei die Arbeit übernehmen, digitale Fragebögen an Clients zu verteilen und die Antworten persistieren. Mit einem modernen Web-Client sollen anschließend Fragebögen bearbeitet werden.

Vor der Umsetzung des Systems wurden zunächst ähnliche bereits verwendete Fragebogensysteme wie zum Beispiel *SurveyMonkey* analysiert. Daraus konnten die Anforderungen für das System abgeleitet werden. Das System besitzt unter anderem ein Benutzermanagement und ermöglicht die Versendung von (anonymen) Fragebögen. Diese Fragebögen können zusätzlich per E-Mail versendet werden, die der Benutzer auch unter seinen Eintellungen deaktivieren kann. Das Ausführen komplexer Logik von Fragebögen sowie verschiedene Bildschirmgrößen werden unterstützt. Zusätzlich dazu wurden qualitative Eigenschaften des Systems spezifiziert sowie ein Styleguide für die Benutzeroberfläche festgelegt.

Nach der Anforderungserhebung wurde mit dem Konzept und Entwurf des Systems begonnen. Dabei wurde das verwendete Fragebogenmodell vorgestellt und darauf aufbauend das verwendete Antwortmodell konzipiert. Anschließend wurde die Architektur des Clients und des Servers entworfen. Eine Thin-Server Architektur sorgt dafür, dass die Anwendungslogik auf der Clientseite implementiert wurde und somit die Serverkomponente entlastet wird. Das Datenmodell wurde basierend auf den erhobenen Anforderungen erstellt. Abschließend wurde die graphische Benutzeroberfläche anhand ausgewählter Beispiele vorgestellt, die den definierten Styleguide berücksichtigen.

Aufbauend auf der Konzept- und Entwurfsphase wurde mit der Implementierung der Plattform begonnen. Durch den Einsatz von AngularJS wurde die gesamte Anwendungslogik auf der Clientseite implementiert. Für die Darstellung auf verschiedenen Endgeräten wurde Bootstrap verwendet. Der Versand von E-Mails an Benutzer, der nicht auf dem Client implementiert werden konnte, wurde nach einer Analyse verschiedener externer Dienste von SendGrid übernommen. Auf der Serverseite wurde Laravel verwendet. Dieser ist als API-Server konzipiert, an dem Anfragen vom Client gesendet werden und dort bearbeitet werden. Abhängig von der API-Version und der Nutzerrolle des Clients, werden Anfragen unterschiedlich behandelt oder abgelehnt. Die Serverkomponente dient zudem als Bindeglied zwischen dem E-Mail-Dienst SendGrid und dem Client.

Diese Abschlussarbeit gliedert sich in das Projekt QuestionSys ein. Mit einem Konfigurator können komplexe Fragebögen von Experten ohne Programmierkenntnisse erstellt werden. Eine Serverkomponente ist für die Verteilung der Fragebögen sowie

Persistierung zuständig. Die Fragebögen selber werden durch verschiedene Endgeräte wie zum Beispiel Desktop-PCs, Tablets oder Smartphones bearbeitet. Die Daten werden anschließend an den Server gesendet, der diese wiederum für die Auswertung an den Konfigurator sendet. Nach Auswertung der Daten erhält der Server die dazugehörigen Ergebnisse, die anschließend auf dem Client dargestellt werden.

In dieser Arbeit wurde ein Fragebogensystem zur flexiblen Datenerhebung entwickelt. Eine Serverkomponente ist zuständig für die Verteilung und Persistierung von Fragebögen. Für die Bearbeitung der Fragebögen ist ein moderner Web-Client entwickelt worden. Es werden komplexe Fragebögen mit Verzweigungslogik unterstützt. Eine unvollständige Bearbeitung von Fragebögen wird durch eine Überprüfung verhindert und dem Benutzer aufmerksam gemacht. Zusätzlich können anonyme Fragebögen versendet werden. Durch eine Nutzerverwaltung können Benutzer am System registriert werden und von einem Editor Fragebögen zur Bearbeitung per E-Mail erhalten.

6.1 Ausblick

Bisher ist es während der Erstellung eines Fragebogens möglich, die zulässige Dauer für die Bearbeitung und die Anzahl der Wiederholungen eines Fragebogens festzulegen. In Zukunft wäre es zudem denkbar, diese Einstellungen zu erweitern. Beispielsweise könnte ein Fragebogen erst bearbeitet werden, nachdem einer oder mehrere Fragebögen bearbeitet wurden und somit Abhängigkeiten zwischen Fragebögen erstellt werden.

Anonyme Fragebögen unterstützen zurzeit keine Einstellmöglichkeiten wie beispielsweise die Wiederholung des Fragebogens in einem bestimmten Intervall, da hier bereits mehr Informationen über den Nutzer gespeichert werden müssen. Dadurch können Fragebögen in Zusammenhang mit einem Nutzern gebracht werden, was dem Sinn anonymer Fragebögen entgegenläuft. Zukünftige Erweiterungen, wie die erlaubte Bearbeitung eines Fragebogens nach der Bearbeitung eines zuvor versendeten Fragebogens sollten daher so implementiert werden, dass keine Rückschlüsse auf den Benutzer geschlossen werden können. Ein Ansatz für die Unterstützung der Einstellmöglichkeiten wäre die Erstellung eines HTML-Links nach erfolgreicher Bearbeitung eines anonymen

Fragebogens. Hier müsste das Ausführungsdatum und Wiederholungsmöglichkeiten sowie weitere Abhängigkeiten zwischen bereits bearbeiteten Fragebögen anonym gespeichert werden.

Zum jetzigen Zeitpunkt können mit dem Konfigurator keine `AND-Split` und `AND-Join` Gateways, sowie Synchronisationskanten erstellt werden und werden daher noch nicht vom Client unterstützt. Da Fragebögen nicht in Echtzeit parallel abgearbeitet werden können, wäre eine Möglichkeit, die verschiedenen Pfade des `AND-Split` Gateways nacheinander zu bearbeiten. Dabei müssen jedoch mögliche Synchronisationskanten berücksichtigt werden und gegebenenfalls der aktive Pfad pausiert und mit den anderen Pfaden weitergearbeitet werden.

In der Ergebnisübersicht kann bisher der Fragebogenstatus der teilzunehmenden Benutzer als Statistik dargestellt werden. Für die Darstellung der Ergebnisse müssen diese jedoch zunächst ausgewertet werden. Solch ein System soll im Rahmen des Projektes QuestionSys in Zukunft implementiert werden. Die Antworten der Fragebögen sollen an dieses System gesendet werden, dort analysiert und die Ergebnisse anschließend zurückgesendet werden. Eine Exportfunktion für die zukünftigen Ergebnisse ist bereits implementiert. Diese können im Format JSON und CSV exportiert werden.

In Zukunft wäre es auch möglich, dass weitere Clientsysteme Fragebögen vom Server erhalten und bearbeiten können. Dazu können neben Smartphones und Tablets auch Wearables wie Smartwatches in Betracht gezogen werden. Ergebnisse zur Bearbeitung von Fragebögen mit Smartwatches finden sich in [41]. Der Versand der Fragebögen sowie das Antwortenmodell ist dabei unabhängig vom Endgerät konzipiert worden. Ein Benutzer muss sich lediglich am System einloggen können, um Fragebögen zu bearbeiten.

Literaturverzeichnis

[1] SurveyMonkey: Kostenloses Softwaretool für Fragebögen & Umfragen. (https://de.surveymonkey.com/) Zugriff: 2016-03-31.

[2] LimeSurvey: The most popular FOSS survey tool on the web. (https://www.limesurvey.org/de/) Zugriff: 2016-03-31.

[3] Universität Ulm: QuestionSys. (https://www.uni-ulm.de/in/iui-dbis/forschung/projekte/questionsys.html) Zugriff: 2016-03-25.

[4] Schobel, J., Pryss, R., Reichert, M.: Using Smart Mobile Devices for Collecting Structured Data in Clinical Trials: Results From a Large-Scale Case Study. In: 28th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2015), Sao Carlos, Brazil. (S. 13 - 18, 22-25 Juni, 2015)

[5] Schobel, J., Pryss, R., Schickler, M., Manfred, R.: Towards Flexible Mobile Data Collection in Healthcare. (In: IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS 2016), Dublin and Belfast, Ireland) Accepted for publication.

[6] Scott, E.J.: SPA Design and Architecture: Understanding Single Page Web Applications. Manning Publications (2015)

[7] Eichorn, J.: Understanding AJAX: Using JavaScript to Create Rich Internet Applications. Prentice Hall (2006)

[8] Lauesen, S.: Software Requirements: Styles & Techniques. Addison-Wesley Professional (2002)

[9] SoSci Survey: Fragebogen online erstellen, Befragung im Internet durchführen. (`https://www.soscisurvey.de/`) Zugriff: 2016-03-31.

[10] Bergsmann, J.: Requirements Engineering für die agile Softwareentwicklung. dpunkt (2014)

[11] Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. Pearson (2009)

[12] Tidwell, J.: Designing Interfaces. O'Reilly Media (2011)

[13] Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. PhD thesis, Ulm University (2000)

[14] Schulte, J.: Technical Conception and Implementation of a Configurator Environment for Process-aware Questionnaires Based on the Eclipse Rich Client Platform. Master's thesis, Ulm University (2014)

[15] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014)

[16] OMG: Business Process Model (2011). Business Proecess Model and Notation (BPMN) Version 2.0. OMG Specification, Object Management Group. `http://www.omg.org/spec/BPMN/2.0/PDF/` (2011) Zugriff: 2016-02-05.

[17] Subhash, C.Y.: An Introduction to Client/Server Computing. New Age International Pvt Ltd Publishers (2009)

[18] Maly, F., Kriz, P., Slaby, A.: Mobile oriented software architecture M client M client server. In: Proceedings of the ITI 2011 33rd International Conference on Information Technology Interfaces (ITI). Cavtat, Croatia. (2011)

[19] Tarasiewicz, P., Böhm, R.: AngularJS. dpunkt (2014)

[20] Brackett, M.: Data Resource Design. Technics Publications (2012)

[21] Krug, S.: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders (2013)

[22] Schlatter, T., Levinson, D.: Visual Usability. Morgan Kaufmann (2013)

[23] AngularJS: Superheroic JavaScript MVW Framework. https://angularjs.org/ (2009) Zugriff: 2016-02-20.

[24] Backbone.js. http://backbonejs.org/ (2010) Zugriff: 2016-02-20.

[25] : Ember.js - A framework for creating ambitious web applications. http://emberjs.com/ (2011) Zugriff: 2016-02-20.

[26] AngularJS: Developer guide: Data binding. https://docs.angularjs.org/guide/databinding (2016) Zugriff: 2016-02-21.

[27] AngularJS: Developer guide: Conceptual overview. https://docs.angularjs.org/guide/concepts#scope (2016) Zugriff: 2016-02-21.

[28] Laravel: The PHP Framework For Web Artisans. https://laravel.com/ (2016) Zugriff: 2016-02-26.

[29] Maurice, F.: PHP 5.6 und MySQL 5.7, 4th Edition. dpunkt (2015)

[30] Bean, M.: Laravel 5 Essentials. Packt Publishing (2015)

[31] National Institute of Standards and Technology: Announcing the ADVANCED ENCRYPTION STANDARD (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (2001) Zugriff: 2016-02-26.

[32] Bootstrap: The world's most popular mobile-first and responsive front-end framework. (http://getbootstrap.com) Zugriff: 2016-03-23.

[33] Credle, R., Armstrong, A., Atkinson, C., Bonner, R., Pirie, G., Singh, I., Williams, Nigel an Wilson, M., Wooley, M.: Implementing IBM CICS JSON Web Services for Mobile Applications. IBM Hursley Park, Hursley UK: IBM Rebooks (2013)

[34] Jones, M., Bradley, J., Sakimura, N.: RFC 7519 - JSON Web Token (JWT). https://tools.ietf.org/html/rfc7519 (2015) Zugriff: 2016-03-02.

[35] Jones, M., Hildebrand, J.: RFC 7516 - JSON Web Encryption (JWE). https://tools.ietf.org/html/rfc7516 (2015) Zugriff: 2016-03-02.

[36] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California (2000)

[37] Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM Conference on Computer and Communications Security. CCS '08, New York, NY, USA, ACM (2008) 75–88

[38] Mandrill: Transaction Email from MailChimp - Mandrill. `https://www.mandrill.com/` (2016) Zugriff: 2016-03-13.

[39] SendinBlue: E-Mailing- und SMS-Marketing Software. `https://de.sendinblue.com/` (2016) Zugriff: 2016-03-13.

[40] Sendgrid: Marketing and Transactional Email Service. `https://sendgrid.com/` (2016) Zugriff: 2016-03-13.

[41] Schickler, M., Pryss, R., Reichert, M., Heinzelmann, M., Schobel, J., Langguth, B., Probst, T., Schlee, W.: Using Wearables in the Context of Chronic Disorders. (In: IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS 2016), Dublin and Belfast, Ireland) Accepted for publication.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Name: David Damaschk Matrikelnummer: 722867

Erklärung

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

Ulm, den .....

David Damaschk