

WACHTER PARTNER

Abschlussprüfung Sommer 2016

FACHINFORMATIKER (ANWENDUNGSENTWICKLUNG)

DOKUMENTATION ZUR BETRIEBLICHEN PROJEKTARBEIT

REALISIERUNG EINER VISUELLEN INTERESSENABFRAGE ALS ANGULAR.JS- EINZELSEITEN-WEBAPPLIKATION

Prüfungsbewerber:

Johannes Kreiner, Identnummer 155 1665788

Ausbildungsbetrieb:

Meteosat Software-Institut

Betriebliches Praktikum:

Wachter Partner WPWA Team GmbH

INHALTSVERZEICHNIS

1 PROJEKTBESCHREIBUNG	1
1.1 PROJEKTUMFELD	1
1.2 PROJEKTZIEL	1
1.3 PROJEKTBEGRÜNDUNG	1
1.4 PROJEKTSCHNITTSTELLEN	2
1.5 PROJEKTABGRENZUNG	2
2 PROJEKTPLANUNG	2
2.1 PROJEKTPHASEN	2
2.2 ABWEICHUNGEN VOM PROJEKTANTRAG	2
2.3 RESSOURCENPLANUNG	3
2.4 ENTWICKLUNGSPROZESS	3
3 ANALYSE	3
3.1 IST-ANALYSE	3
4 ENTWURF	4
4.1 ABLAUFPLAN DER APPLIKATION	4
4.2 AUSWAHL DES FRONT-END-FRAMEWORKS	4
4.3 RELATIONALES DATENBANKMODELL	4
4.4 UML-KLASSENDIAGRAMM PHP	4
5 IMPLEMENTIERUNG	5
5.1 IMPLEMENTIERUNG DES FRONT-ENDS	5
5.1.1 IMPLEMENTIERUNG DER HTML-SEITEN	5
5.1.2 IMPLEMENTIERUNG DES ANGULAR.JS-MODULS	5
5.1.3 IMPLEMENTIERUNG DES LAYOUTS	10
5.2 IMPLEMENTIERUNG DES BACK-ENDS	12
5.2.1 IMPLEMENTIERUNG DES PHP-PROGRAMMTEILS	12
5.2.2 IMPLEMENTIERUNG DER MYSQL-DATENBANK	15
6 QUALITÄTSKONTROLLE	15
6.1 BESCHREIBUNG DER TESTS	15
6.1.1 AUTOMATISIERTE TESTS	15
6.1.2 MANUELLE TESTS	15
6.2 VERSIONIERUNG	16
7 WIRTSCHAFTLICHKEITSBETRACHTUNG	16
7.1 PROJEKTKOSTEN	16
8 FAZIT	16
8.1 SOLL-/IST-VERGLEICH	16

GLOSSAR	I
QUELLENVERZEICHNIS	II
ANHANG	III
A.1 GROBE ZEITPLANUNG	III
A.2 DETAILLIERTE ZEITPLANUNG	III
A.3 RESSOURCENPLAN UND VERWENDETE HARD- UND SOFTWARE	IV
A.4 ABLAUFPLAN	VI
A.5 MVVM-AUFBAU	VII
A.6 NUTZWERTANALYSE	VII
A.7 RELATIONALES DATENBANKMODELL (AUSZUG)	VII
A.8 LISTE DER FORMULARELEMENTE (AUSZUG)	VIII
A.9 UML-KLASSENDIAGRAMM PHP	VIII
A.10 APP.JS UND ROUTES.JS	IX
A.11 GETINDEX() UND NEXTSTEP()	IX
A.12 EXKURS: MODULTESTS	IX
A.13 AUSGABE "TEST.HTML"	X
A.14 DIRECTIVES	X
A.15 ISUNCHECKED()	X
A.16 BACKEND.JS	X
A.17 INSERT-KLASSE	XI
A.18 RESPONSIVE DESIGN TESTS	XII
A.19 SOLL-/IST-VERGLEICH ZEITPLAN	XII

HINWEIS:

Kursiv geschriebene Wörter können im Glossar nachgeschlagen werden.

1 PROJEKT BESCHREIBUNG

1.1 PROJEKTUMFELD

Das Projekt wurde von mir in den Räumen der Agentur WACHTER PARTNER WPWA TEAM GMBH (im Folgenden durch WPWA abgekürzt) in 80335 München durchgeführt. WPWA wurde 2008 gegründet und ist eine kleine Werbeagentur mit 11 Mitarbeitern, von denen etwa die Hälfte Freelancer sind. Zu den Aufgaben von WPWA gehören u.a. Beratung, Gestaltung, Foto und Film sowie Web-Entwicklung. Im Bereich Web-Entwicklung wurden schon eine Vielzahl an Onlineauftritten für Firmen wie SIXT, LORINGHOVEN oder auch EVENTLICHT erstellt. Auch Online-Shops wie z.B. für SOULFOOD LOWCARBERIA hat WPWA schon erfolgreich umgesetzt.

Der Kunde ROOMME hat sich für seinen Online-Shop eine Erweiterung in Form einer visuellen Interessenabfrage mit Kontaktdaten-Erfassung gewünscht (im Folgenden Applikation bzw. Webapplikation genannt) und WPWA hat diesen Auftrag angenommen.

1.2 PROJEKTZIEL

Im Webbrowser seiner Wahl kann der Endbenutzer von der Webseite des Online-Shops von ROOMME durch einen Klick auf einen Link zur Applikation gelangen. Die Applikation besteht aus mehreren, in der Reihenfolge festgelegten Ansichten, u.a. mit aus *HTML*-Formularelementen bestehenden Fragen und Antwortmöglichkeiten, zum Beispiel zu den eigenen Vorlieben bezüglich Baustil, Einrichtungsstil, gewünschten und unerwünschten Farben und der Wohnungsgröße.

Es findet eine Validierung der Antworten statt, sodass ein Weiterkommen nur bei korrekter Benutzung möglich ist. Der Fortschritt wird über einen sogenannten Fortschrittsbalken angezeigt (in Prozent und grafisch). Im letzten Schritt kann der Endbenutzer auf einen Submit-Button ("Absenden") klicken und die Daten an einen definierten Kundenberater von ROOMME senden. Dies geschieht, indem die Formulardaten aus dem Front-End (*HTML*, *CSS*, *JavaScript* bzw. *Angular.js*) an das Back-End (*PHP*) weitergeleitet und in einer *MySQL*-Datenbank hinterlegt werden. Aus den Daten innerhalb der Datenbank wird dann eine E-Mail generiert und an eine hinterlegte E-Mail-Adresse versendet.

1.3 PROJEKTBEGRÜNDUNG

Das Projekt wurde von WPWA angenommen, da mit der Verwendung von *Angular.js* nicht nur eine kosteneffiziente Umsetzung des Projektes möglich war, sondern dessen Technologie möglicherweise auch für eine Vielzahl zukünftiger Projekte zur Anwendung kommen könnte. Deshalb dienen sowohl das Projekt, als auch diese Dokumentation als Vorlage bzw. Hilfe für die Erstellung weiterer Webapplikationen.

1.4 PROJEKTSCHNITTSTELLEN

Der Endbenutzer kann die Webapplikation in jedem aktuellen Browser und auf einer Vielzahl von Geräten wie PCs, Notebooks über Tablets bis hin zu Smartphones aufrufen.

Die Webapplikation läuft auf einem *Apache*-Server mit *MySQL* und *PHP*.

Genehmigt wurde das Projekt von der Geschäftsleitung der WPWA sowie dem Leiter der Entwicklungsabteilung. Diesen Instanzen wurde auch der aktuelle Stand immer wieder und nach Fertigstellung präsentiert. Dabei wurde die gesamte Durchführung als Agenturleistung von WPWA der Firma ROOMME in Rechnung gestellt.

1.5 PROJEKTABGRENZUNG

Dieses Projekt behandelt nicht die Erstellung eines separaten Online-Shops, sondern lediglich die davon unabhängige Interessenabfrage als Webapplikation. Die Möglichkeit, sich als Endbenutzer und als Mitarbeiter von ROOMME einloggen zu können, wird erst nach diesem Projekt implementiert.

Basiskenntnisse in der Erstellung von *HTML*-Seiten und der Implementierung von *CSS*-Stilen werden vorausgesetzt, da eine genauere Beschreibung dieser Vorgänge den Rahmen dieser Dokumentation sprengen würde. Deshalb wird z.B. das Einbinden von Skript-Dateien nicht ausdrücklich erwähnt, obwohl es trotzdem stattgefunden hat.

2 PROJEKTPLANUNG

2.1 PROJEKTPHASEN

Das Projekt wurde innerhalb von 70 Stunden durchgeführt. Eine grobe sowie eine detaillierte Zeitplanung finden sich im Anhang unter A.1 Grobe Zeitplanung bzw. A.2 Detaillierte Zeitplanung.

2.2 ABWEICHUNGEN VOM PROJEKTANTRAG

Auf eine ausführliche Amortisationsrechnung wurde verzichtet, da die Kosten für das Projekt direkt nach Abschluss durch Bezahlung des Kunden gedeckt wurden. Die Zeit, die somit bei der Wirtschaftlichkeitsbetrachtung eingespart werden konnte, wurde für umfangreichere Entwürfe verwendet.

Die im Antrag angegebene Planung der Umsetzung des *MVVM*-Entwurfsmusters wurde nicht in einem eigenen Punkt, sondern bei der Auswahl des *JavaScript*-Frameworks behandelt, da die durch *Angular.js* relativ einfache Verwendung dieses Entwurfsmusters und damit die Planung der Umsetzung stark mit der Entscheidung, das *Angular.js*-Framework zu verwenden, zusammenhing.

Der Punkt "Implementierung der *CSS*-Stile" wurde zu "Implementierung des Layouts" umbenannt, da das Layout auch mithilfe von *Angular.js* und *HTML* umgesetzt werden musste.

Außerdem wurden bei der Implementierung des Layouts auch einige Einsparungen vorgenommen (weniger Animationen), um mehr Zeit für die Erstellung eines Ablaufplans, eines *UML*-Klassendiagramms und für ein sichereres Back-End zu bekommen.

Diese Änderungen wurden bereits in der Planungsphase berücksichtigt.

2.3 RESSOURCENPLANUNG

Die benötigten Ressourcen sowie die verwendete Hard- und Software finden Sie im Anhang unter A.3 Ressourcenplan und verwendete Hard- und Software.

2.4 ENTWICKLUNGSPROZESS

Als Entwicklungsprozess wurde die agile Softwareentwicklung verwendet, sodass während der Implementierung ständige Rücksprache mit der Geschäftsführung, dem Kunden und dem Leiter der Entwicklungsabteilung bestand. Auch wurde *BDD* mit dem Test-Framework *Jasmine.js* eingesetzt.

3 ANALYSE

3.1 IST-ANALYSE

Das sogenannte "Personal Shopping" (auf die eigene Persönlichkeit zugeschnittenes Einkaufen) ist ein fester Bestandteil von modernem E-Commerce (dt. Online-Handel). Hierbei werden die Interessen und Wünsche des Kunden in einer initialen Abfrage erfasst, die es später dem Händler ermöglicht, genauer auf die Anforderungen des Kunden einzugehen und ihm die Entscheidungsfindung zu erleichtern. Online-Shops wie *UNIQUE* (www.uniquefragrance.de) und *OUTFITTERY* (www.outfittery.de) machen es vor.

Dem Kunden werden beim ersten Besuch mehrere Fragen unterschiedlicher Kategorien gestellt, die er im Ausschlussverfahren beantworten kann. Anschließend gibt der Kunde seine Kontaktdaten an, um später von einem Mitarbeiter per E-Mail oder Telefon kontaktiert zu werden.

Dadurch haben die Händler von Anfang an personalisierte Daten, und ersetzen gleichzeitig eine mitunter zeit- und kostenaufwendigere Initialberatung.

Weil diese relativ neue Art des Einkaufens im Bereich Inneneinrichtung bzw. Innenarchitektur bislang nicht vertreten ist und hierdurch mehr Privatkunden generiert werden können, hat sich die Firma *ROOMME* (Kunde von *WPWA*) eine visuelle Interessenabfrage mit Kontaktdaten-Erfassung des Benutzers als zeitgemäße *Angular.js*-Einzelseiten-Webapplikation gewünscht.

4 ENTWURF

4.1 ABLAUFPLAN DER APPLIKATION

Damit die Reihenfolge der Schritte sowie des weiteren Entwurfs (z.B. Datenbankmodell) und die für die Implementierung benötigten *HTML-Templates*, Datenbank-Tabellen und -Attribute sowie *Angular.js*- und *PHP*-Attribute bestimmt werden konnten, wurde ein Ablaufplan erstellt, den Sie im Anhang unter A.4 Ablaufplan finden können.

4.2 AUSWAHL DES FRONT-END-FRAMEWORKS

Als Front-End-Framework ist *Angular.js* (basierend auf *JavaScript*) zum Einsatz gekommen. Damit wurde für das Front-End der Applikation ein *MVVM*-Entwurfsmuster (Model View ViewModel) mit dem Aufbau umgesetzt, wie im Anhang unter A.5 MVVM-Aufbau¹ zu sehen ist. Die durch *Angular.js* relativ einfache Umsetzung dieses Entwurfsmusters und die dadurch bedingte verständliche Strukturierung des Programmcodes waren ein Grund für diese Entscheidung. Zudem ist die spürbare Geschwindigkeit der Applikation höher, als durch die weitaus Performance-beeinträchtigendere *DOM*-Manipulation des in der Branche noch häufiger eingesetzten *jQuery*.

Die Dokumentation von *Angular.js* befindet sich dagegen nicht ganz auf Augenhöhe im Vergleich zu *jQuery*, weil es noch nicht so lange zum Einsatz kommt. Trotzdem mangelt es nur an Beispielen und nicht an umfangreichen Erklärungen der Bestandteile, weshalb dieser Nachteil zu vernachlässigen ist. Das Vorhaben, eine Eigenentwicklung zu erstellen, wurde aus Zeit- und Kostengründen ausgeschlossen. Sowohl *jQuery* als auch *Angular.js* können, da sie unter der *MIT-Lizenz* stehen, kostenlos verwendet werden. Eine Nutzwertanalyse für diesen Vergleich finden Sie im Anhang unter A.6 Nutzwertanalyse.

4.3 RELATIONALES DATENBANKMODELL

Für die *MySQL*-Datenbank wurde von mir mit dem Programm *MYSQL-WORKBENCH* ein Modell entworfen (siehe Anhang unter A.7 Relationales Datenbankmodell). Dieses Modell basiert auch auf den Erkenntnissen, die durch den Ablaufplan gewonnen werden konnten (z.B. über die benötigten Tabellen). Für die Attribute, deren Datentypen sowie evtl. vorhandenen Werte (z.B. bei Dropdown-Listen) wurde abgewägt, welche Formularelemente benötigt werden (siehe Anhang unter A.8 Liste der Formularelemente). Mit *MYSQL-WORKBENCH* wurde der benötigte *MySQL*-Code (Erstellung der Tabellen und Einfügen der Werte) durch die sogenannte "Forward Engineering"-Funktion von *MYSQL-WORKBENCH* generiert.

4.4 UML-KLASSENDIAGRAMM PHP

Die benötigten Klassen sowie Attribute und Methode habe ich mit der Webapplikation *DRAW.IO* in einem *UML*-Klassendiagramm dargestellt, wie im Anhang unter A.9 UML-Klassendiagramm PHP zu sehen ist.

¹ Vgl. <https://msdn.microsoft.com/en-us/library/ff798384.aspx>

5 IMPLEMENTIERUNG

5.1 IMPLEMENTIERUNG DES FRONT-ENDS

5.1.1 IMPLEMENTIERUNG DER HTML-SEITEN

Zum Anfang wurde gleich eine `robots.txt`-Datei erstellt, welche den GOOGLE-Bot (und andere Bots) am Durchsuchen der Seite hindert, damit die Seite nicht im unfertigen Zustand bei der GOOGLE-Suche erscheinen kann.

Daraufhin begann die Erstellung der *HTML*-Seite. Innerhalb der `questionnaire.html`-Datei befindet sich ein `<div>`-Container mit `ui-view` als Attribut, der den Inhalt der `index.html` aus dem Ordner `/questionnaire/` darstellt. Diese `index.html` enthält wiederum ein `<form>`-Tag in welchem sich ebenfalls ein `<div>`-Container mit dem Attribut `ui-view` befindet. Dieser Container bekommt die Inhalte der verschiedenen *Templates* aus den Unterordnern der jeweiligen Schritte der Abfrage. Durch diesen verschachtelten Aufbau bleibt die eigentliche Seite immer gleich und die jeweiligen Inhalte werden dynamisch ausgetauscht. Dies ermöglicht das `ui-view` Attribut, welches durch das *Angular.js*-Modul *UI-Router* verwendet werden kann.

Im Anschluss wurden in den jeweiligen *HTML*-Templates der Formular-Schritte die benötigten Formular-Elemente eingefügt (wie im Entwurf festgelegt).

Nach Erledigung dieses Schrittes konnte ich mit der Implementierung des *Angular.js*-Moduls beginnen. Beachten Sie bitte, dass die *HTML*-Templates währenddessen teilweise geändert werden mussten, da *Angular.js* auch durch die Einbindung von Attributen innerhalb der *HTML*-Tags implementiert wird. Des Weiteren wurden später einige Änderungen vorgenommen, die Sie im Abschnitt 5.1.3 Implementierung des Layouts finden können.

5.1.2 IMPLEMENTIERUNG DES ANGULAR.JS-MODULS

Für die Implementierung des *Angular.js*-Moduls wurde von der Entwicklungsleitung entschieden, dass die Version 1.5.0 von *Angular.js* und 0.2.10 von *UI-Router* verwendet wird.

Dafür wurde das *Angular.js*-Attribut `ng-app` dem `<body>`-Tag der `questionnaire.html`-Datei mit dem Wert `questionnaireApp` hinzugefügt, um festzulegen, dass die Applikation in diesem Bereich existiert. Anschließend wurden die ersten *JavaScript*-Dateien erstellt:

```
/
/scripts/
  app.js Die Initialisierung der Angular.js-Applikation.
  routes.js Die möglichen Routen bzw. Views der Applikation.
```

Den Quellcode dieser zwei Dateien finden Sie im Anhang unter A.10 `app.js` und `routes.js`.

Als erstes wurde in der Datei `app.js` das eigentliche *Angular.js*-Modul `angular.module()` erstellt, sowie dessen Abhängigkeiten per *DI* festgelegt. Bis hierhin war die Applikation nur von `ui.router` abhängig, später wurden jedoch noch weitere Abhängigkeiten eingefügt. Die

an `angular.module()` angehängte `run()`-Methode wird bei der Implementierung des Layouts erläutert.

In der Datei `routes.js` wurden dem `angular.module()` die spezifischen Konfigurationen von *UI-Router* innerhalb der Methode `config()` übergeben. Zu den Abhängigkeiten der Konfiguration gehören in diesem Fall `$stateProvider` (legt die einzelnen Views fest) sowie `$urlRouterProvider` (behandelt in diesem Fall, auf welche Seite umgeleitet wird, wenn keine View festgelegt wurde). Mit `$stateProvider.state()` wurde dann jede Ansicht separat mit folgenden Einstellungen konfiguriert:

`url`: Der Pfad, der in der Adresszeile angegeben werden soll.

`templateUrl`: Der Pfad zur tatsächlichen *HTML*-Datei.

`controller`: Die Funktionalität, welche die Seite später innehaben sollte.

Der Controller musste dabei nur für die Hauptseite der Abfrage eingerichtet werden, da sich die Funktionalität auf die komplette Abfrage bezieht. Zu beachten sei, dass mit der Reihenfolge der einzelnen `state()`-Methoden auch die Reihenfolge der einzelnen Schritte der Abfrage, wie im Ablaufplan, festgelegt wurde. Für den Controller wurde im Ordner `/scripts/` ein Unterordner `/controllers/` erstellt, um später evtl. weitere Controller implementieren zu können. Als erste Methoden des Controllers wurden folgende erstellt:

`getIndex()`

- Um den aktuellen Schritt (wird innerhalb von *UI-Router* auch "state", also "Zustand" genannt) in der Abfrage zu erfahren.
- Dabei wird mit einer Schleife jeder Zustand mit dem aktuellen verglichen. Wenn eine Übereinstimmung stattfindet, wird der momentane Index des Zustandes in der Variable `currentIndex` gespeichert und von der Methode zurückgegeben.

`nextStep()`

- Um zum nächsten Schritt zu gelangen.
- Diese Methode verlinkt auf den Zustand mit einem Index nach dem aktuellen Index.

Den Quellcode dieser Methoden finden Sie im Anhang unter A.11 `getIndex()` und `nextStep()`. Wie Sie sehen, bekommt die Methode `nextStep()` die Variable `currentIndex` übergeben. Dies erfolgt, indem man im *HTML*-Template des jeweiligen Zustandes die Methode `nextStep(getIndex())` aufruft. Konkret erhält das Attribut `ng-click` des `<input type="button"/>`-Tags des Zustandes den Wert `nextStep(getIndex())`, wodurch bei jedem Klick auf den `<input type="button"/>`-Tag diese Methode ausgeführt wird. Dieser Vorgang wurde für jeden Zustand bis auf den letzten wiederholt. Nun konnte man jeden Schritt per Klick auf "Weiter" ansehen.

Exkurs: Modultests

Um prüfen zu können, dass von der Methode `getIndex()` wirklich ein Index zurückgegeben wird, wurde an dieser Stelle der erste von später mehreren Unit-Tests mit dem Test-Framework *Jasmine.js* implementiert. Dafür wurde eine *JavaScript*-Datei mit dem Namen `questionnaire-controller.spec.js` im selben Ordner wie der eigentliche Controller angelegt (`/scripts/controllers/`).

Im Prinzip funktionieren die Tests mit *Jasmine.js* mit diesen drei Funktionen:

- `describe()` Diese Funktion sollte den Namen der zu testenden Einheit enthalten.
- `it()` Diese Funktion sollte beschreiben, welche Bedingung die zu testende Einheit erfüllen sollte.
- `expect()` Diese Funktion prüft, ob die Bedingung erfüllt ist. Für die Methode `getIndex()` gilt dann folgende Aussage: "controllers/questionnaire-controller.js sollte einen Index erhalten, also prüfe, ob ein Index vorhanden ist".

Die tatsächliche Implementierung ist nur wenig komplizierter, wie Sie im Anhang unter A.12 Exkurs: Modultests sehen können. Erwähnenswert sei noch, dass durch die *Jasmine.js*-Funktion `beforeEach()` bestimmte Anweisungen vor jedem Test ausgeführt werden können, und in diesem Fall vor jedem Test das `questionnaireApp`-Hauptmodul sowie der `questionnaireController` selbst für den Verlauf des Tests durch das *Angular.js*-Modul *ngMock* imitiert (engl. mocked) werden. Um das Testergebnis später betrachten zu können, wurde zunächst eine neue *HTML*-Datei `test.html` im Hauptverzeichnis angelegt.

Beachten Sie, dass alle Skriptdateien eingebunden werden mussten, welche auch die eigentliche Applikation beinhaltet. Zusätzlich wurden die verschiedenen *Jasmine.js*-Programmteile sowie das *ngMock*-Modul eingebunden. Nun konnte die `test.html` aufgerufen werden, um die Testergebnisse zu sehen. Die Ausgabe dieser Datei finden Sie im Anhang unter A.13 Ausgabe "test.html". Dieser Schritt wurde im Laufe der Implementierung und später in der Qualitätskontrolle immer wieder wiederholt.

Weitere Implementierung:

An diesem Punkt habe ich mit der Validierung der einzelnen Formularelemente begonnen. Ursprünglich sollte die Validierung durch eigens angefertigte Methoden erfolgen, z.B. wäre dann die Eingabe von Buchstaben in ein Feld, das für Zahlen vorgesehen wäre, deaktiviert worden. Aus Gründen der Benutzerfreundlichkeit und einer möglichen Zeitersparnis heraus, wurde an dieser Stelle gegen solch eine Validierungsform entschieden und ein gleichwertiger, modernerer Ersatz eingebunden, wie Sie weiter unten feststellen werden. Doch zunächst zurück zum Beginn: Um die Kontrolle über die genauen Validierungsmaßnahmen zu erhalten und die Standard-Validierung durch *HTML* zu deaktivieren, wurde zuerst dem übergeordneten *HTML-Template* des Formulars (`/questionnaire/index.html`) im `<form>`-Tag das Attribut `novalidate` vergeben.

Als nächstes kam dann auch das *Angular.js*-Attribut `ng-disabled` ins Spiel:

Dieses Attribut wurde den `<input type="button" />`-Tags der jeweiligen Schritte vergeben und hat folgende Funktionalität: wenn die Bedingung des `ng-disabled`-Attributs ein `true` zurückgibt, wird das `<input type="button" />`-Element deaktiviert. Für die Bedingung wurde für die meisten Schritte das *Angular.js*-eigene Objekt-Attribut `$invalid` benutzt. Dieses Attribut prüft, ob das Formular im aktuellen Schritt validiert ist und gibt dementsprechend einen Booleschen Wert `true` oder `false` zurück, der dann innerhalb des Controllers abgefragt werden kann.

Um zu bestimmen, ob das Formular validiert ist, musste den jeweiligen Formularelementen das Attribut `ng-model` (mit Werten in diesem Format: `Formulardaten-Objekt.Formulardaten-Attribut`) hinzugefügt werden, da *Angular.js* nur so weiß, dass dieses Element geprüft werden muss. In diesem Beispiel wurde das Formulardaten-Objekt `formData` genannt und das dazugehörige Attribut genauso wie der Wert des `name`-Attributes. Eine Ausnahme bilden hierbei Checkbox-Formularelemente, da diese unterschiedliche `ng-model`-Werte benötigen, um eine Mehrfachauswahl zu ermöglichen.

Für Debug-Zwecke mit der GOOGLE CHROME-Erweiterung *ng-inspector* und für die spätere Speicherung der Werte wurde an dieser Stelle das `formData`-Objekt innerhalb des `questionnaireControllers` mit einem leeren Objekt initialisiert.

Zusätzlich zu den `ng-model`-Werten mussten den jeweiligen Formularelementen die zugehörigen Attribute, die geprüft werden sollten, hinzugefügt werden:
Allen Elementen - außer den Checkboxes und einer Seite mit drei Dropdown-Listen (`/reuse/index.html`) - gemeinsam wurde das Attribut `required` gegeben, damit keine leeren Eingaben möglich sind. Dann waren die für die jeweiligen Formularelemente spezifischen Attribute an der Reihe, wobei der Datenbankentwurf dabei als Vorlage galt. Auf der Seite `/reuse/index.html` wurde jedem `<select>`-Tag noch das Attribut `ng-init` mit dem Wert `formData.reuse1 = 'none'` bis `formData.reuse3 = 'none'` gegeben, damit die drei Dropdown-Listen standardmäßig die Auswahlmöglichkeit "-- Keine Möbel ausgewählt --" vorselektiert haben. Als weiteres Beispiel für `/home-size/index.html`: Hier wurde das Attribut `ng-maxlength='6'` vergeben (da bei maximal zwei Nachkommastellen eine vierstellige Quadratmeter-Zahl das Maximum sein sollte).

Da *Angular.js* jedoch ohne Modifikationen keine falschen Eingaben verhindert (aber sehr wohl das Absenden dieser), sondern nur Fehlermeldungen anzeigt (durch das `$invalid`-Attribut bestimmt), wurde an diesem Punkt das *Angular.js*-Modul *ngMessages* eingebunden. Mit *ngMessages* ist es mit wenigen Codezeilen möglich, kontextbezogene Fehlermeldungen als herkömmliche *HTML*-Elemente anzeigen zu lassen. Wie schon bei *UI Router* wurde das *ngMessages*-Modul innerhalb von `/scripts/app.js` per *DI* eingebunden.

Um die *Template*-Dateien schlank zu halten, wurden von mir für die verschiedenen Fehlermeldungen mehrere separate *HTML*-Dateien erstellt. Dabei erhielten die Eingabefeld-Typen `email`, `number` und `text` jeweils eine eigene Version. Um später evtl. unterschiedliche Funktionen hinzufügen zu können, wurden diese zusätzlichen Templates nicht einfach durch das *Angular.js*-Attribut `ng-include` eingefügt, sondern als speziell angefertigte *HTML*-Elemente mittels den *Angular.js*-eigenen sogenannten *directives*.

Mit *Directives* ist es möglich, eigens erstellte *HTML*-Elemente oder -Attribute zu definieren, sowohl mit beliebigem Namen und wenn nötig auch mit speziellen Funktionen. Für diese *Directives* wurden im Ordner `/scripts/` noch ein Unterordner `directives` sowie drei *JavaScript*-Dateien erstellt:

`email-messages.js`, `number-messages.js` und `text-messages.js`
In diesen *Directives* wird mit dem Attribut `templateUrl` auf die jeweiligen *HTML*-Dateien verwiesen. Ein Beispiel können Sie im Anhang unter A.14 *Directives* sehen.

Nun ließen sich die eigenen *HTML*-Elemente in die jeweiligen *Templates* einsetzen, z.B. mit `<number-message></number-message>`. Dieser Schritt wurde für die Seiten `/customer/index.html`, `/home-size/index.html` sowie `/room-dimensions/index.html` wiederholt, mit den jeweils passenden *HTML*-Elementen.

Nachdem die Validierung der meisten Formularelemente implementiert war, konnte ich mich den Formular-Abschnitten mit Checkbox-Gruppen widmen. Da ein `required`-Attribut keine leeren Checkboxes zulassen würde, dies aber gegen den Zweck von Checkboxes spricht, musste ein anderer Weg gefunden werden. Auch war an diesem Punkt klar, dass das *Angular.js*-Objekt-Attribut `$invalid` für diese Formularelemente nicht funktionieren würde. Um dies zu umgehen, musste dem `questionnaireController` zunächst eine weitere Methode hinzugefügt werden: `isUnchecked()`

Diese Methode durchläuft mittels einer `for`-Schleife jeden potenziellen Wert der Checkbox-Gruppe und prüft, ob diese ein `false` zurückgeben, also nicht angekreuzt sind. Nur wenn mindestens ein Wert auch tatsächlich vorhanden ist (und damit definiert bzw. auf `true` gesetzt ist), gibt die Methode ein `false` zurück, also ein "Nein, mindestens eine Box ist angekreuzt" (da das `ng-disabled`-Attribut bei einem `true` den Button deaktivieren würde). Da alle `formData`-Attribute vor dem Ausfüllen des Formulars `undefined` sind, wurden an dieser Stelle die Attribute der `formData`-Objekte, die Mehrfachauswahlen repräsentieren, mit `false`-Werten initialisiert. Den Quellcode dieser Methode finden Sie im Anhang unter A.15 `isUnchecked()`.

Als Wert für das `ng-disabled`-Attribut wurde für die Formularelemente mit Checkboxes dann die `isUnchecked()`-Methode mit den zu prüfenden Formulardaten eingetragen, also z.B. `isUnchecked(formData.interiorStyle)` für das *Template* `/questionnaire/interior-style/index.html`.

An diesem Punkt war die Implementierung der Validierung abgeschlossen und die Verbindung zum Back-End konnte eingerichtet werden. Um die einzelnen Programmbestandteile möglichst modular gestalten zu können, wurde für die Verbindung zum Back-End ein *Angular.js*-Service umgesetzt. Dafür musste zunächst ein Ordner `/scripts/services/` und darin eine Datei `backend.js` erstellt werden.

In dieser *JavaScript*-Datei (zu sehen im Anhang unter A.16 `backend.js`) befinden sich zwei Methoden:

`insert()` Bekommt die Formulardaten `formData` übergeben und sendet diese an den zu diesem Zeitpunkt noch zu erstellenden *PHP*-Controller `insertAction.php`, um die Formulardaten in eine *MySQL*-Datenbank zu speichern.

`sendMail()` Diese Methode führt die `selectAction.php`-Datei aus, um eine Datenbankabfrage auszuführen sowie den erhaltenen Datensatz formatiert an eine in der `selectAction.php`-Datei festgelegte E-Mail-Adresse zu versenden.

Der Aufruf der `insert()`- und `sendMail()`-Methoden wurde dabei wie folgt implementiert: Um die Formulardaten mit dem Betätigen des Submit-Buttons an das Back-End senden zu

können, musste innerhalb des `questionnaireControllers` eine neue Methode `submitForm()` erstellt werden. Anschließend musste das Back-End-Service per *DI* eingefügt werden. Das *Angular.js*-Attribut `ng-submit` des *HTML*-Tags des Formulars (`<form>`-Tag innerhalb von `/questionnaire/index.html`) bekam dann `submitForm()` als Wert, um diese Methode beim Klick auf das Element ausführen zu können.

Die `submitForm()`-Methode wiederum beinhaltet einen Aufruf der `insert()`- sowie `sendMail()`-Methoden und eine Zuweisung des `currentDatetime`-Attributs, welches das Datum und die Uhrzeit des Absendens speichert, da dieses automatisch vergeben werden sollte. Der Wert des `currentDatetime`-Attributs wird wie folgt definiert:

```
$scope.formData.currentDatetime =  
new Date().toISOString().slice(0, 19).replace('T', ' ');2
```

Durch die angehängten Methoden wird das *JavaScript*-Datums-Format in ein *MySQL*-Format konvertiert.

Somit wurden beim Absenden die Formulardaten an den *PHP*-Controller `insertAction.php` sowie gleichzeitig eine E-Mail mit diesen Daten durch den `selectAction.php`-Controller versendet (siehe 5.2 Implementierung des Back-Ends).

Um ein mehrfaches Absenden zu verhindern und dem Endbenutzer eine Meldung über das erfolgreiche Absenden zu geben, wurde im `questionnaireController` zuerst ein Attribut `formSuccess` erstellt, welches den Status des Absendens als Booleschen Wert speichert. Daraufhin wurde der `insert()`-Methode noch die *Angular.js*-eigene `success()`-Methode angehängt. Diese erhielt als Parameter eine Funktion, die wiederum den Parameter `data` enthält. Die Anweisung in dieser Funktion, die `formSuccess` auf `true` setzt, wird ausgeführt, wenn die Formulardaten erfolgreich an `insertAction.php` übergeben wurden.

Nun konnte man mehrere *HTML*-Elemente einfügen bzw. abändern, die bei einem `true` von `formSuccess` ein- und ausgeblendet werden. Dafür wurden diesen Elementen die *Angular.js*-Attribute `ng-show` und `ng-hide` vergeben. Der Absenden-Button hat also `ng-hide="formSuccess"` bekommen, er wird also ausgeblendet, wenn die Daten abgeschickt wurden. Damit war die Implementierung des *Angular.js*-Moduls an dieser Stelle abgeschlossen.

5.1.3 IMPLEMENTIERUNG DES LAYOUTS

Die *CSS*-Stile wurden in *Less* geschrieben und mit dem Programm *KOALA* zu *CSS* kompiliert.

Als erstes grafisches Element wurde der sogenannte Fortschritts-Balken implementiert. Dieser zeigt dem Benutzer an, wie weit er in der Umfrage ist. Dabei stellt jeder Schritt 10% dar, da es sich auch um zehn Fragen handelt.

Zuerst musste das *HTML*-Markup dafür implementiert werden. Dabei ist zu beachten, dass durch `ng-class` bei jedem nächsten Schritt eine zusätzliche Klasse vergeben wird, um jeweils die Breite erhöhen zu können, sodass der Balken den Fortschritt anzeigen kann. Die zusätzlichen Klassen werden durch den Wert von `ng-class`

² Vgl. <http://stackoverflow.com/questions/5129624/convert-js-date-time-to-mysql-datetime>

vergeben, welcher u.a. aus folgenden Bedingungen besteht:

'step0' : getIndex() > 1, 'step10' : getIndex() > 2, 'step20' :
getIndex() > 3. So hat der Fortschrittsbalken bei Schritt eins die Klasse step0 und bei
Schritt zwei zusätzlich step10 bekommen, und so weiter. Außerdem sollte der Fortschritt in
Prozent angegeben werden, wofür unter /scripts/filters/ ein neuer Filter percentage
erstellt wurde.

Dieser Filter wandelt einen Dezimalwert in eine Zahl ohne Nachkommastellen um und hängt
ein Prozentzeichen an. Im *HTML*-Markup des Fortschrittsbalkens wird durch die
getIndex()-Methode berechnet, bei wie viel Prozent der Fortschritt ist. Durch das
anhängen von | (Pipe-Symbol) und dem Namen des Filters, also in diesem Fall
percentage, wird die davorstehende Anweisung umformatiert, in diesem Fall zu einer
ganzzahligen Prozentzahl.

Während der Implementierung der Stile ist mir aufgefallen, dass man besonders auf
Mobilgeräten mit kleinen Bildschirmen teilweise nach unten scrollen musste, um auf den
Weiter-Button klicken zu können. Die nächste Formularseite wurde dann aber auf der selben
Scrollhöhe angezeigt. Um dies zu beheben, wurden folgende Schritte umgesetzt: In der
app.js wurde an angular.module() noch eine run()-Methode angehängt, damit eine
bestimmte Funktion bei jedem Aufruf des Moduls ausgeführt wird. Diese Methode enthält als
Parameter wiederum eine Funktion mit dem Parameter \$rootScope, welches den
Geltungsbereich der kompletten Applikation definiert (also den Inhalt des <body>-Tags mit
dem ng-app-Attribut). Danach wurde die \$on()-Methode des \$rootScope-Objektes, mit
den Parametern \$stateChangeSuccess und einer weiteren Funktion eingefügt. Die
Aussage dieses Methodenaufrufes heißt nun: "Führe bei jedem Aufruf der Applikation die
Anweisungen der \$on()-Methode aus, wenn sich der Zustand ändert (also von einem Schritt
zum nächsten oder vorherigen gewechselt wird)". Die Anweisung lautet:

```
document.body.scrollTop = document.documentElement.scrollTop = 03;
```

Hierdurch wird gewährleistet, dass bei jedem Schritt an den Anfang der Seite gescrollt wird.

Da eine genaue Beschreibung der Implementierung der CSS-Stile den Rahmen dieser
Dokumentation sprengen würde, werden hier nur die wichtigsten Vorgänge erwähnt.

- Die Formularelemente der Checkboxes und Radiobuttons wurden durch eigene
Elemente ersetzt (durch den Hintergrund vom <label>-Tag des jeweiligen
Formularelementes dargestellt).
- Für wiederkehrende Werte, wie bestimmte Farben, wurden *Less*-Variablen
verwendet.
- Breakpoints für verschiedene Darstellungsgrößen wurden eingefügt. Durch die
Breakpoints und die Verwendung von größtenteils relativen Einheiten im Gegensatz
zu festen Pixel-Größen (z.B. bei Schriftgrößen) wurden die Stile nach den
Paradigmen des "Responsive Webdesigns" umgesetzt.

³ Vgl. <http://stackoverflow.com/questions/26444418/autoscroll-to-top-with-ui-router-and-angularjs>

5.2 IMPLEMENTIERUNG DES BACK-ENDS

5.2.1 IMPLEMENTIERUNG DES PHP-PROGRAMMTEILS

Obwohl der *PHP*-Teil der Applikation zuerst nur "Controller" heißen sollte, wurde ein *MVC*-Entwurfsmuster umgesetzt, dessen Aufbau sich auch in der Ordnerstruktur widerspiegelt:

```
/
/backend/
    /controllers/
    /models/
```

Dabei sollte dieses Back-End-Modul - wie im vorigen Abschnitt erwähnt - einen Controller mit Dateinamen `/controllers/insertAction.php` besitzen, der das Einfügen der aus dem *Angular.js*-Modul gesendeten Daten in eine *MySQL*-Datenbank (und somit deren Speicherung) und einen Controller `/controllers/selectAction.php`, der das gleichzeitige Versenden einer E-Mail mit den Daten aus der Datenbank an den Mitarbeiter von Roomme auslöst. Die E-Mail ist in diesem Fall dann die View. Im Ordner `/models/` hingegen befinden sich die benötigten Klassen und deren Attribute und Methoden.

Die `insertAction.php`-Datei ruft mehrere Funktionen und Methoden auf, um u.a. die Formulardaten aus dem *Angular.js*-Modul in einer *PHP*-Variable zu speichern. Dafür wurde in dieser Datei die Funktion `file_get_contents()` mit Parameter `'php://input'` der Variablen `$postData` zugewiesen, um die Formulardaten im ursprünglichen (durch *Angular.js* bestimmten) *JSON*-Format in `$postData` abzuspeichern. Für die weitere Verwendung mussten die Formulardaten dann noch in ein Format für *PHP* dekodiert werden, was mit der Funktion `json_decode()` möglich ist. Diese Funktion wurde einer neuen Variablen `$formData` zugewiesen und hat zwei Parameter übergeben bekommen, `$postData` und `true`. Das `true` besagt, dass die Daten in ein assoziatives Array umgewandelt werden, wodurch sie mit dem Attributs-Namen (auch "key" bzw. Schlüssel genannt) aufgerufen werden können.

Da für die gewünschte weitere Funktionalität diverse *PHP*-Dateien erstellt werden sollten, wurde in der `insertAction.php` auch das Hauptverzeichnis des Back-Ends festgelegt (dies musste für alle zusammengehörigen *PHP*-Dateien wiederholt werden). Wenn man kein Hauptverzeichnis bestimmt, ist das Hauptverzeichnis für *PHP* immer das der aufgerufenen Datei, was die Gefahr von schwer kontrollierbarem Verhalten erhöhen kann⁴.

Die Zuweisung wurde wie folgt definiert:

```
$DIR = $_SERVER['DOCUMENT_ROOT'] . "/roomme/backend";
```

Nachdem das Hauptverzeichnis festgelegt wurde, konnte man nun die später benötigte und im Anschluss erstellte Datei `/models/Dbconfig.php` mit der `require_once()`-Funktion einfügen. Als nächstes habe ich dabei die Konfiguration der Datenbank-Verbindung implementiert, wofür die Datei `Dbconfig.php` im Ordner `/backend/models/` erstellt wurde. Diese Datei sollte auch als Verbindungsglied zwischen allen anderen *PHP*-Dateien gelten. Die `Dbconfig.php` enthält die Zuweisung der Datenbank-Zugangsdaten sowie die Erstellung einer Objektinstanz der *PHP*-eigenen Klasse *PDO* mit Namen `$con` (Abkürzung

⁴ Vgl. <http://yagudaev.com/posts/resolving-php-relative-path-problem/>

für "connection"), in dem die aktuelle Verbindung abgespeichert wird. Ein *PDO* ermöglicht die Benutzung von besonderen Methoden.

Nachdem in dieser Datei wieder das Hauptverzeichnis festgelegt wurde, konnte man die restlichen Dateien mit der `require_once()`-Funktion einfügen, wobei hier die *PHP*-Datei der Klasse `Database` mit Pfad bzw. Namen `/backend/models/Database.php` den Anfang machte. Diese sollte als Eltern-Klasse für die verschiedenen Datenbank-Einsätze gelten (wie dem Einfügen oder Auswählen von Datensätzen).

In der `Database`-Klasse wurden alle Datenbank-Attribute mit der Sichtbarkeit `protected` (damit nur Kindklassen auf die Attribute zugreifen können) deklariert sowie das *PDO* `$con` dem `Database`-Attribut `$db` über den Konstruktor zugewiesen. Damit bekommt eine Kindklasse von `Database` automatisch Zugriff auf die Datenbank, sofern das `$db`-Attribut verwendet wird (allerdings nicht, wenn eine Kindklasse einen abgeänderten Konstruktor vorweist; hierbei muss das `$con`-*PDO* der Objektinstanz einer Kindklasse explizit übergeben werden). Außerdem wurde eine `public` Methode `closeConnection()` implementiert, welche nach Ausführung der *MySQL*-Anweisungen die Verbindung zur Datenbank schließen sollte.

Nun konnte ich in der `Dbconfig.php` eine Objektinstanz `$database` der Klasse `Database` mit dem Parameter der Verbindung `$con` erstellen. Als ersten Datenbank-Einsatz für die Applikation wurde das Einfügen eines neuen Datensatzes implementiert. Hierfür musste eine Klasse `Insert` mit dem Dateinamen `Insert.php` im Ordner `/backend/models/` erstellt und in der `Dbconfig.php` per `require_once()`-Funktion eingefügt werden. Diese Klasse erbt von der `Database`-Klasse und verfügt über ein `private` (damit ein Zugriff nur von dieser Klasse aus möglich ist) Attribut `$insert`, welches in der Methode `insertData()` einen *MySQL*-Befehl ausführt sowie ein `private` Attribut `$sql`, welches diesen *MySQL*-Befehl definiert.

Zunächst musste der Konstruktor für diese Klasse von dem der Elternklasse abgewandelt werden. Weil die Formulardaten aus dem *Angular.js*-Modul nur in der `Insert`-Klasse, aber nicht in der `Database`-Klasse benötigt werden, wurde der Konstruktor mit einem zusätzlichen Parameter versehen, den Formulardaten selbst. Dieser wurde `$formData` genannt. Damit der Konstruktor der `Insert`-Klasse auch den Inhalt des Konstruktors der Elternklasse übernimmt, musste dieser im Konstruktor der `Insert`-Klasse mit `parent::__construct($con)` aufgerufen werden.

Als nächstes wurden innerhalb des `Insert`-Konstruktors die Schlüssel-Wert-Gruppierungen des assoziativen Arrays `$formData` den dazugehörigen Attributen, vererbt von `Database`, zugewiesen.

An diesem Punkt wurde die `insertData()`-Methode implementiert. Dem Aufruf der `prepare()`-Methode des `$con`-*PDOs* `$db` wurde durch das `$sql`-Attribut ein `INSERT INTO`-*MySQL*-Befehl mit Platzhaltern als Attribut-Namen als Parameter übergeben. Dieser Methodenaufruf wiederum wurde im Attribut `$insert` gespeichert. Nun konnte man mit der *PDO*-Methode des Objektes `$insert`, genannt `bindParam()`, den Platzhaltern die jeweiligen Werte zuweisen und per *PDO*-Methode des Objektes `$insert`, genannt `execute()`, den `INSERT INTO`-*MySQL*-Befehl ausführen lassen.

Der Grund für diese zunächst als umständlich erscheinende Lösung ist, dass so eine *MySQL*-Injection in den allermeisten Fällen verhindert wird⁵. Würde man direkt die Attributnamen statt Platzhalter einfügen, könnte schadhafter Code wie z.B. `" 'Name' ; DELETE FROM `customer` ;"` direkt mit dem Einfügen ausgeführt werden und in diesem Beispiel zu Datenverlust führen. Da die `prepare()`-Methode jedoch ein sogenanntes "prepared statement" ermöglicht, wird nicht ein kompletter String als *MySQL*-Befehl, sondern die tatsächlichen Werte der Platzhalter mit dem Befehl zusammen ausgeführt. Den genauen Befehl finden Sie ausschnittsweise im Anhang unter A.17 Insert-Klasse.

Um der *Insert*-Klasse die Formulardaten zu übergeben, wurde per `require_once()`-Funktion die *Insert.php*-Datei in der *Dbconfig.php*-Datei eingefügt und dann in der *insertAction.php*-Datei eine Objektinstanz der *Insert*-Klasse `$insert` erstellt. Als Parameter mussten `$con` (die Datenbank-Verbindung) und `$formData` (die eigentlichen Formulardaten) übergeben werden.

Nun konnte man die Methode zum Einfügen der Daten in die Datenbank in der *insertAction.php* per `$insert->insertData()` aufrufen (und diese Datei dann wiederum aus dem *Angular.js*-Modul über die Methode `Backend.insert(formData)` ausführen).

Zum Auswählen der Daten für die E-Mail-Ausgabe wurde als nächstes die *Select*-Klasse implementiert. Diese fand ihren Platz im Ordner `/backend/models/` in der Datei *Select.php* und erweitert wie die *Insert*-Klasse die *Database*-Klasse, um deren Konstruktor (und damit die Datenbankverbindung über das `$db`-Attribut) und die `closeConnection()`-Methode vererbt zu bekommen. Die *Select*-Klasse hat drei Methoden:

<code>selectData()</code>	Hier wird der <i>MySQL</i> - <i>SELECT</i> -Befehl aus dem <code>\$sql</code> -Attribut ausgeführt, um die Daten aus der Datenbank abzufragen.
<code>displayData()</code>	Hier speichert eine <i>while</i> -Schleife den abgefragten Datensatz in die für die Ausgabe benötigten Variablen. Zurückgegeben wird von dieser Methode dann der Text für die E-Mail, mitsamt allen geforderten Informationen.
<code>sendMail()</code>	Mit den Parametern <code>\$address</code> und <code>\$title</code> .

In der `sendMail()`-Methode befindet sich der Aufruf der `selectData()`-Methode (um den *MySQL*-*SELECT*-Befehl auszuführen) sowie die Funktion `mail()`, die drei Parameter aufnimmt: Eine E-Mail-Adresse, den E-Mail-Betreff und dann die Nachricht, zurückgegeben von der Methode `displayData()`. Nachdem die *Select*-Klasse erstellt wurde, konnte die *Select.php*-Datei per `require_once()`-Funktion in der *Dbconfig.php*-Datei eingefügt werden.

Zum Ausführen des *SELECT*-Vorgangs wurde dann noch ein Controller mit Dateinamen *selectAction.php* innerhalb von `/backend/controllers/` erstellt, mit ähnlichem Aufbau

⁵ Vgl. <http://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php/60496#60496>

wie in der `insertAction.php`-Datei. Ein wichtiger Unterschied ist hier, dass die `sendMail()`-Methode via `$select->sendMail("email-des@kundenberaters.de", "Betreff der E-Mail")` aufgerufen wurde. Damit der `SELECT`-Befehl nicht ausgeführt wird, bevor der `INSERT`-Befehl ausgeführt wurde, wurde die *PHP*-eigene `sleep()`-Funktion mit 10 als Parameter vor der `sendMail()`-Methode ausgeführt. Damit wird erst 10 Sekunden gewartet, bevor die `sendMail()`-Methode ausgeführt wird. Mit der `sendMail()`-Methode war es nun möglich, E-Mail-Nachrichten mit den Daten der Kunden automatisch an den Kundenbetreuer zu senden, nachdem die Umfrage durch den Kunden abgeschlossen wurde.

Zum Abschluss der Implementierung des *PHP*-Moduls wurde - ebenfalls in den `insertAction.php`- und `selectAction.php`-Dateien - noch die `closeConnection()`-Methode der `Database`-Objektinstanz `$database` aufgerufen, um die Verbindung zur Datenbank nach Beendigung der Vorgänge zu schließen.

5.2.2 IMPLEMENTIERUNG DER MYSQL-DATENBANK

Da bereits in der Entwurfsphase bei der Erstellung des Datenbankmodells mittels `MYSQL-WORKBENCH` der benötigte *MySQL*-Code generiert wurde, musste der Code nur noch auf dem Datenbankserver, innerhalb von `PHPMYADMIN`, ausgeführt werden.

6 QUALITÄTSKONTROLLE

6.1 BESCHREIBUNG DER TESTS

6.1.1 AUTOMATISIERTE TESTS

Die unter 5.1.2 Implementierung des *Angular.js*-Moduls beschriebenen Modultests mit *Jasmine.js* wurden für verschiedene Module implementiert und im Laufe der Entwicklung immer wieder durch Neu-Laden der `test.html`-Ausgabe durchgeführt.

6.1.2 MANUELLE TESTS

Die manuellen Tests im Bereich *Angular.js* wurden unter anderem mit der `CHROME`-Erweiterung *ng-inspector* durchgeführt, welche den Wert aller in der aktuellen Ansicht vorhandenen Attribute und Variablen sowie alle vorhandenen Methoden und Funktionen darstellt.

Der *PHP*-Teil wurde durch Vergleichen der Werte in der *MySQL*-Datenbank über `PHPMYADMIN` getestet. Mit der E-Mail, die beim Absenden des Formulars automatisch gesendet wird, gab es eine weitere Möglichkeit, um die Übertragung der korrekten Daten festzustellen. Außerdem wurde während der Entwicklung eine *MySQL*-Injection von mir selbst versucht, um diese Sicherheitslücke ausschließen zu können.

Die Webapplikation wurde auch in verschiedenen Browsern und auf verschiedenen Geräten getestet. Beispiele dazu finden Sie im Anhang unter A.18 Responsive Design Tests. Diese Tests wurden von mir und von verschiedenen Mitarbeitern von WPWA durchgeführt.

6.2 VERSIONIERUNG

Von Beginn des Projektes an wurde immer wieder der zum jeweiligen Zeitpunkt aktuelle Stand in der Versionsverwaltung über das Programm SOURCETREE festgehalten. Der Übersichtlichkeit wegen wurde (fast) jedem Commit auch eine Markierung zum dazugehörigen Bereich oder Thema vergeben, wie z.B. [JS] für *JavaScript* oder [Template] für die *HTML*-Templates.

7 WIRTSCHAFTLICHKEITSBETRACHTUNG

7.1 PROJEKTKOSTEN

Jegliche verwendete Software ist entweder zur freien Verwendung oder Open Source verfügbar (z.B. *Angular.js* und *MySQL*) oder schon bezahlt (z.B. Server-Software und *SUBLIME TEXT*). Ebenso wurde die verwendete Hardware schon vor dem Projekt bezahlt. Auf dem Webserver werden auch andere Webseiten gehostet, sodass diesbezüglich auch keine weiteren Kosten angefallen sind. Da ich, bedingt durch die Teilnahme an einer externen Vollzeitausbildung einen Stundensatz von 0 € hatte, blieben nur die Kosten für den Büroarbeitsplatz sowie für Internet und Strom. Für diese Punkte wurde von einem pauschalen Stundensatz von 10 € ausgegangen. Somit ergeben sich folgende Projektkosten: Durchführungszeit von 70 Stunden x 10 € Kosten pro Stunde, also Projektkosten von gesamt 700,00 €. Dieser Betrag wurde vollständig in die Kundenrechnung integriert.

8 FAZIT

8.1 SOLL-/IST-VERGLEICH

Die Zeit von 70 Stunden wurde insgesamt gehalten, auch wenn es teilweise Verschiebungen in der Zusammensetzung dieser Stunden gab. Die Implementierung des *PHP*-Teils hat zwei Stunden länger gedauert, als geplant. Da aber die Validierung der Eingaben entgegen der ursprünglichen Planung nur mit *ngMessages* und nicht mit manueller Validierung (z.B. u.a. durch Zählen der Zeichenlänge mit einer Schleife) erledigt wird, hat dieser Vorgang zwei Stunden weniger gedauert. Einen Vergleich zwischen den Ist- und Soll-Stunden finden Sie im Anhang unter A.19 Soll-/Ist-Vergleich Zeitplan. Das Projekt wurde von WPWA für die weitere Entwicklung und Integration in den Online-Shop von ROOMME abgenommen, wodurch die Zukunftsperspektive geklärt ist.

ANHANG

GLOSSAR

Angular.js	Clientseitiges JavaScript-Framework
Apache	Freier HTTP Webserver
BDD	Behaviour-Driven-Development
CSS	Cascading Style Sheets
DI	Dependency Injection
DOM	Document Object Model
HTML	Hypertext Markup Language
Jasmine.js	BDD-Framework für JavaScript-Modultests
JavaScript	Clientseitige Skriptsprache
jQuery	JavaScript-Framework für u.a. einfache DOM-Manipulation
JSON	JavaScript Object Notation
Less	Erweiterte Stylesheet-Sprache
MIT-Lizenz	Lizenz für Benutzung von u.a. quelloffener Software, vom Massachusetts Institute of Technology stammend
MVC	Model View Controller
MVVM	Model View ViewModel
MySQL	Relationales Datenbankmanagementsystem
ng-inspector	Angular.js-Debugging-Erweiterung für Google Chrome
ngMessages	Angular.js-Erweiterung für kontextbezogene Fehlermeldungen
ngMock	Angular.js-Erweiterung für imitierte Module in Tests
PDO	PHP Data Object
PHP	Backronym für PHP: Hypertext Processor
Template	HTML-Seite für dynamische Webseiten
UI-Router	Angular.js-Erweiterung für verschachtelte Ansichten in Einzelseiten-Applikationen
UML	Unified Modeling Language
WPWA	Wachter Partner Werbeagentur

QUELLENVERZEICHNIS

- [1] Microsoft
<https://msdn.microsoft.com/en-us/library/ff798384.aspx>

- [2] Kuizinas, Gajus
<http://stackoverflow.com/questions/5129624/convert-js-date-time-to-mysql-datetime>

- [3] TaylorMac
<http://stackoverflow.com/questions/26444418/autoscroll-to-top-with-ui-router-and-angularjs>

- [4] Yagudaev, Michael
<http://yagudaev.com/posts/resolving-php-relative-path-problem/>

- [5] Theo
<http://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php/60496#60496>

ANHANG

A.1 GROBE ZEITPLANUNG

Projektphase	Ist-Stunden
1 Analyse	3
2 Entwurf	10
3 Implementierung	35
4 Qualitätskontrolle	9
5 Wirtschaftlichkeitsbetrachtung	1
6 Erstellen der Dokumentation	12
Gesamt	70

A.2 DETAILLIERTE ZEITPLANUNG

Projektphase	Gesamt-stunden	Zwischen-summe Stunden	Teil-summe Stunden
1 Analyse	3		
1.1 Durchführung der Ist-Analyse		3	
2 Entwurf	10		
2.1 Ablaufplan der Applikation		1	
2.2 Auswahl des JavaScript-Frameworks		4	
2.3 Relationales Datenbankmodell		3	
2.4 UML-Klassendiagramm PHP		2	
3 Implementierung	35		
3.1 Implementierung der HTML-Seiten		4	
3.1.1 Implementierung der statischen Abschnitte			2
3.1.2 Implementierung der Templates			2
3.2 Implementierung des Angular.js-Moduls		23	
3.2.1 Implementierung der Controller			5
3.2.2 Implementierung der Services			5
3.2.3 Implementierung von UI-Router-Konfigurationen			4
3.2.4 Implementierung der Validierung			4
3.2.5 Implementierung der Back-End-Anbindung			1
3.2.6 Implementierung der Modultests			4
3.3 Implementierung des Back-Ends		5	

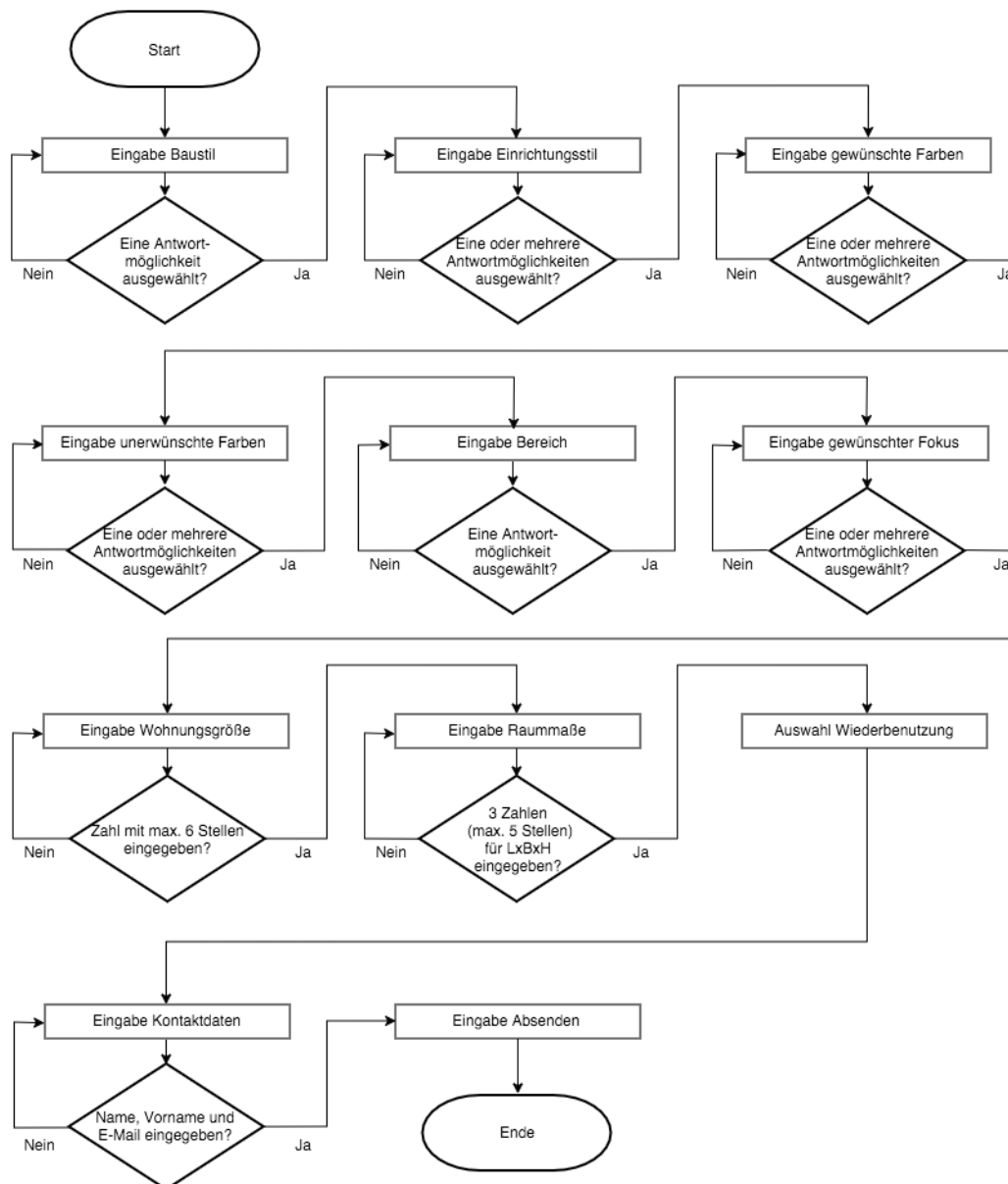
3.3.1 Implementierung des PHP-Programnteils			4
3.3.2 Implementierung der MySQL-Datenbank			1
3.4 Implementierung des Layouts		3	
3.4.1 Implementierung der CSS-Stile			2
3.4.2 Implementierung der Animationen			1
4 Qualitätskontrolle	9		
4.1 Durchführung der automatisierten Tests		4	
4.2 Durchführung der manuellen Tests		4	
4.3 Versionierung		1	
5 Wirtschaftlichkeitsbetrachtung	1		
5.1 Projektkosten		1	
6 Erstellen der Dokumentation	12		
Gesamt	70		

A.3 RESSOURCENPLAN UND VERWENDETE HARD- UND SOFTWARE

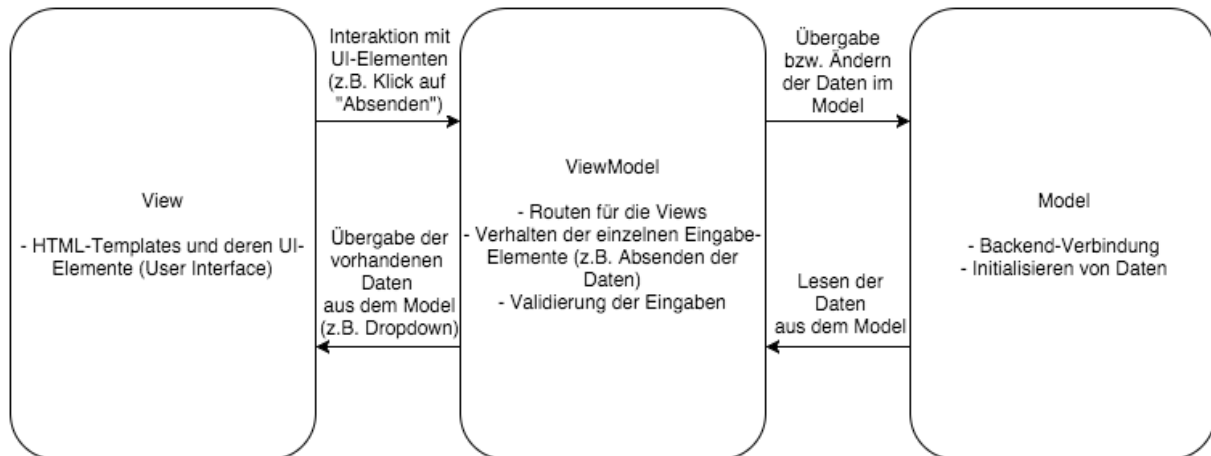
Typ	Bezeichnung	Rolle
Personal		
	Entwickler und Autor	Projektausführung und Dokumentation
Hardware		
	27-Zoll Apple iMac (Late 2009)	Arbeits- und Testgerät
	Apple iPhone 6	Testgerät
	Apple iPad 2	Testgerät
	Asus Notebook	Testgerät
	HTC One M8	Testgerät
	Samsung Galaxy Tab 8.9	Testgerät
	Virtueller Linux-Server	Hosting der Webapplikation
Software		
Betriebssysteme	Apple OS X 10.11	Arbeits-/Test-Betriebssystem
	Apple iOS 9.3	Test-Betriebssystem
	Microsoft Windows 7	Test-Betriebssystem
	Microsoft Windows 10	Test-Betriebssystem
	Google Android (verschiedene Versionen)	Test-Betriebssystem
Browser	Google Chrome (verschiedene Versionen)	Arbeits-/Test-Browser

	Apple Safari 9	Test-Browser
	Microsoft Internet Explorer 11	Test-Browser
	Microsoft Edge 25	Test-Browser
	Mozilla Firefox 45	Test-Browser
Entwicklungs-Tools	Sublime Text 2	Code-Editor
	Koala	Less-Compiler
	ng-inspector	Chrome-Erweiterung für Angular.js-Debugging
	Oracle MySQL Workbench	Programm zum Erstellen von Datenbank-Modellen sowie daraus generiertem SQL-Code
	Atlassian SourceTree	Versionsverwaltung
Erweiterungen für Programmiersprachen	Google Angular.js	JavaScript-Framework
	Less	CSS-Präprozessor
	UI-Router	Angular.js-Erweiterung für verschachtelte Ansichten
	Google ngMessages	Angular.js-Erweiterung für kontextbezogene Fehlermeldungen
	Google ngMock	Angular.js-Erweiterung für imitierte Module bei Tests
	Jasmine.js	Modultest-Framework für JavaScript
Dokumentations- Tools	Apple TextEdit	Notizzettel
	draw.io	Zeichenprogramm für Diagramme
	Microsoft Word	Textverarbeitung bzw. Programm zur Erstellung der Dokumentation
Server-Software	Parallels Plesk 12.5	Server-Verwaltung
	CentOS Linux 7.2	Server-Betriebssystem
	Oracle MySQL 5.5.44	Datenbank-Server
	Apache	Web-Server
	PHP 5.6.19	Serverseitige Programmiersprache

A.4 ABLAUFPLAN



A.5 MVVM-AUFBAU

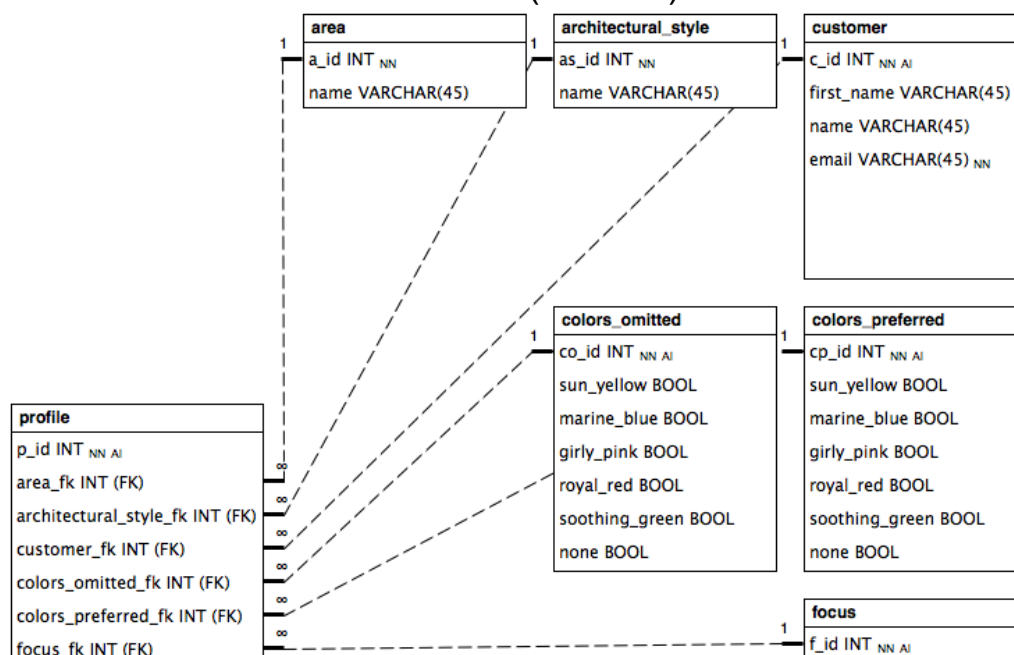


A.6 NUTZWERTANALYSE

Eigenschaft	Eigenentwicklung	jQuery	Angular.js	Gewichtung
Dokumentation	0	5	4	5
Performance	3	3	5	4
Wartbarkeit	3	3	5	4
Gesamt	24	49	60	13
Nutzwert	1,85	3,77	4,62	

(0 = gering, 5 = optimal)

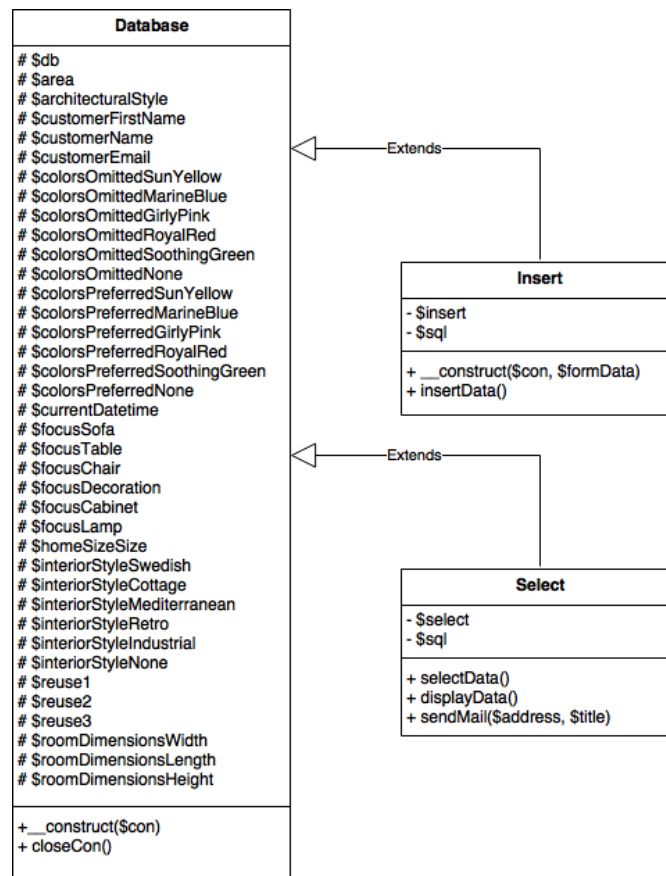
A.7 RELATIONALES DATENBANKMODELL (Auszug)



A.8 LISTE DER FORMULARELEMENTE (AUSZUG)

Schritt-Nr.	Name	Typ	Für Validierung Benötigt
1	Baustil	Radiobuttons	Ja
2	Einrichtungsstil	Checkboxes	Ja
...
7	Wohnungsgröße	Eingabefeld "number"	Ja
8	Raummaße	Drei Eingabefelder "number"	Ja
9	Wiederverwendung	Drei Dropdown-Listen	Nein
10	Kontaktdaten	Zwei Eingabefelder "text" und ein Eingabefeld "email"	Nur Vorname nicht
11	Absenden	Absendebutton	-

A.9 UML-KLASSENDIAGRAMM PHP



A.10 APP.JS UND ROUTES.JS

/scripts/app.js (Auszug)

```
1 ▼ (function() {  
2   angular.module('questionnaireApp', ['ui.router', 'ngMessages'])  
3   .run(function($rootScope) {  
4     $rootScope.$on('$stateChangeSuccess', function() {  
5       document.body.scrollTop = document.documentElement.scrollTop = 0;  
8     }  
9   });  
10 }  
11 )
```

/scripts/routes.js (Auszug)

```
1 ▼ (function() {  
2   angular.module('questionnaireApp')  
3   .config(['$stateProvider', '$urlRouterProvider', function($stateProvider, $urlRouterProvider) {  
4     $stateProvider  
5     .state('questionnaire', {  
6       url: '/questionnaire',  
7       templateUrl: 'templates/pages/questionnaire/index.html',  
8       controller: 'questionnaireController'  
9     })  
10    .state('questionnaire.architectural-style', {  
11      url: '/architectural-style',  
12      templateUrl: 'templates/pages/questionnaire/architectural-style/index.html'  
13    })  
14  })  
15 }  
16 )
```

A.11 GETINDEX() UND NEXTSTEP()


```
68 ▼ | $scope.getIndex = function() {  
69   for (var i = 0; i < $state.get().length; i++) {  
70     if ($state.get()[i] === $state.current) {  
71       currentIndex = i;  
72     }  
73   }  
74   return currentIndex;  
75 };  
76  
77 $scope.nextStep = function(currentIndex) {  
78   $state.go($state.get()[currentIndex + 1]);  
79 };
```

A.12 EXKURS: MODULTESTS

/scripts/controllers/questionnaire-controller.spec.js (Auszug)

```
( 1 ▼ (function() {  
2   describe('controllers/questionnaire-controller.js', function() {  
3     var $controller;  
4  
5     beforeEach(angular.mock.module('questionnaireApp'));  
6  
7     beforeEach(angular.mock.inject(function(_$controller_) {  
8       $controller = _$controller_;  
9     }));  
10  
11    it('should get an index', function() {  
12      var $scope = {};  
13      var controller = $controller('questionnaireController', {$scope: $scope});  
14  
15      expect($scope.getIndex()).not.toBe(undefined);  
16    });  
17  });  
18 }  
19 )
```

A.13 AUSGABE "TEST.HTML"

 **Jasmine** 2.3.3 Options

3 specs, 0 failures finished in 0.02s

controllers/questionnaire-controller.js
 should get an index
 should validate checkboxes
services/backend.js
 should send data to the backend

A.14 DIRECTIVES

/scripts/directives/number-messages.js (Auszug)

```
1 ▼ (function() {  
2   angular.module('questionnaireApp')  
3   .directive('numberMessages', [function() {  
4     return {  
5       restrict: 'E',  
6       templateUrl: 'templates/partials/messages/number/index.html'
```

A.15 ISUNCHECKED()

```
56 ▼ $scope.isUnchecked = function(formData) {  
57   var unchecked = true;  
58  
59   for (key in formData) {  
60     if (formData[key] !== false) {  
61       unchecked = false;  
62     }  
63   }  
64  
65   return unchecked;  
66 };
```

A.16 BACKEND.JS

/scripts/services/backend.js

```
1 ▼ [(function() {  
2   angular.module('questionnaireApp')  
3  
4   .factory('Backend', ['$http', function($http) {  
5     return {  
6       insert: function(formData) {  
7         return $http({  
8           method: 'POST',  
9           url: 'backend/controllers/insertAction.php',  
10          data: formData  
11        });  
12      },
```

A.17 INSERT-KLASSE

/backend/models/Insert.php

```
1 <?php
2 class Insert extends Database {
3     private $insert;
4
5     private $sql =
6         "INSERT INTO `roomme`.`customer` (
7             `first_name`,
8             `name`,
9             `email`
10        ) VALUES (
11            :customerFirstName,
12            :customerName,
13            :customerEmail
14        );
15
16 ...
17
127 public function __construct($con, $formData) {
128     parent::__construct($con);
129     $this->area = $formData['area'];
130     $this->architecturalStyle = $formData['architecturalStyle'];
131
132 ...
133
168 public function insertData() {
169     $this->insert = $this->db->prepare($this->sql);
170     $this->insert->bindParam(':area', $this->area);
171     $this->insert->bindParam(':architecturalStyle', $this->architecturalStyle);
172
173 ...
174
207     $this->insert->execute();
208 }
209 }
```

A.18 RESPONSIVE DESIGN TESTS



iPhone 6



iPad 2

A.19 SOLL-/IST-VERGLEICH ZEITPLAN

Projektphase	Soll	Ist	Differenz
Analyse	3 Stunden	3 Stunden	-
Entwurf	10 Stunden	10 Stunden	-
Implementierung	35 Stunden	37 Stunden	+ 2 Stunden
Qualitätskontrolle	9 Stunden	7 Stunden	- 2 Stunden
Wirtschaftlichkeitsbetrachtung	1 Stunde	1 Stunde	-
Erstellen der Dokumentation	12 Stunden	12 Stunden	-
Gesamt	70 Stunden	70 Stunden	-