Projektbericht

Fragebogen zu Schadensdaten für das Institut für Bauforschung e. V.

Alba Cruz Torres, Frank Köhn und Michael Nickel

Inhaltsverzeichnis

[Abbildungsverzeichnis 4](#_Toc1060355)

[Tabellenverzeichnis 5](#_Toc1060356)

[Einleitung 6](#_Toc1060357)

[Themenbeschreibung 6](#_Toc1060358)

[Motivation 6](#_Toc1060359)

[Gruppenmitglieder und Rollen-/Aufgabenverteilung 7](#_Toc1060360)

[Technologien 7](#_Toc1060361)

[UIkit 7](#_Toc1060362)

[Vue.js 8](#_Toc1060363)

[10](#_Toc1060364)

[Spring/MVC 10](#_Toc1060365)

[JPA 12](#_Toc1060366)

[REST 12](#_Toc1060367)

[PostgreSQL 13](#_Toc1060368)

[Softwareentwurf 13](#_Toc1060369)

[Funktionale Anforderungen 13](#_Toc1060370)

[Nicht funktionale Anforderungen 14](#_Toc1060371)

[Use Case 14](#_Toc1060372)

[16](#_Toc1060373)

[16](#_Toc1060374)

[Aktivitätsdiagramm 17](#_Toc1060375)

[Mockups 18](#_Toc1060376)

[Komponentendiagramm 20](#_Toc1060377)

[Entity-Relationship-Modell 21](#_Toc1060378)

[Klassendiagramm 22](#_Toc1060379)

[Implementierungsdetails 23](#_Toc1060380)

[HTML/CSS/JavaScript 23](#_Toc1060381)

[Java 28](#_Toc1060382)

[Fazit 30](#_Toc1060383)

[Ausblick 30](#_Toc1060384)

[Installationshinweise 31](#_Toc1060385)

[Benutzung 31](#_Toc1060386)

[Literaturverzeichnis 32](#_Toc1060387)

# Abbildungsverzeichnis

[1: UIkit Einbindung 7](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060388)

[2: Vue.js Beispiel 8](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060389)

[3: v-bind Beispiel 9](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060390)

[4: v-if Beispiel 9](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060391)

[5: v-model Beispiel 10](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060392)

[6: MVC 11](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060393)

[7: Use Case Fragebogen bearbeiten 15](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060394)

[8: Use Case System 15](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060395)

[9: Use Case textuell 16](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060396)

[10: Use Case related information 16](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060397)

[11: Aktivitätsdiagramm 17](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060398)

[12: Startseite Mockup 18](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060399)

[13: Frage Mockup 19](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060400)

[14: Komponentendiagramm 20](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060401)

[15: Entity-Relationship-Modell 21](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060402)

[16: Klassendiagramm 22](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060403)

[17: HTML: Kategorie und Fortschrittsanzeige 23](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060404)

[18: Kategorie und Fortschrittsanzeige in der Browseransicht 23](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060405)

[19: HTML: Fehlermeldung 24](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060406)

[20: Validierung von Fragen die Radio-Buttons beinhalten 24](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060407)

[21: Fehlermeldung Browseransicht 24](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060408)

[22: Beispielfrage 25](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060409)

[23: Beispielfrage in der Browseransicht 25](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060410)

[24: Speicherung der Antworten 26](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060411)

[25: POST Request 27](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060412)

[26: Verarbeitung der Antworten 28](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060413)

[27: Speicherung einer Antwort in der Datenbank 29](file:///F:\Info\GitHub\WebEngineering\Dokumentation\Projektbericht.docx#_Toc1060414)

# Tabellenverzeichnis

[1: JPA Annotationen 12](#_Toc1060415)

[2: REST Methoden 12](#_Toc1060416)

# Einleitung

## Themenbeschreibung

Das Ziel dieses Projektes ist die Erstellung eines Fragebogens zu Schadensdaten für das Institut für Bauforschung e. V. in Hannover. Dazu gehört auch die anschließende Übertragung der Daten in eine Excel Tabelle. Das Institut für Bauforschung e. V. hat als eine von vielen Kernaufgaben die Forschung im Gebiet der Bauschäden und deren Ursachen. Aufgrund der Tatsache, dass die durchschnittlichen Bauschäden in Deutschland über die Jahre von 2009 bis 2013 von 55.077 Euro auf 66.947 Euro gestiegen sind[[1]](#footnote-1), ist es wichtig, hier die Ursachen zu untersuchen, um die Bauqualität in Zukunft zu verbessern. Der Fragebogen ist über eine Webseite erreichbar und besteht aus 20 Fragen, die verschiedene Themen zum Planungsprozess, der Ausschreibung und Vergabe, der Ausführung, der Abnahme, den Schadenskosten und der Entwicklung der Bauqualität abdeckt. Jeder Benutzer kann den Fragebogen nur einmal ausfüllen, damit verhindert wird, dass die Daten verfälscht werden. Außerdem ist es nur möglich den Fragebogen bis Ende Februar zu beantworten.

Zur Realisierung des Fragebogens werden konventionelle Web Technologien verwendet. Dazu gehören HTML und CSS, um das Layout der Webseite zu erstellen. Außerdem wird JavaScript verwendet, um die dynamische Veränderung der Seite zu ermöglichen. Dazu gehört auch die Kommunikation mit dem Server über HTTP-Requests. Natürlich wird hier eine Datenbank benötigt, um die Übertragung der Antworten in eine Excel Tabelle zu gewährleisten. Das Übertragen der Daten gehört mit zum Projektumfang.

## Motivation

Die Auswertung von Schadensdaten ist in der Bauforschung ein sehr relevantes Thema, da Fehler beim Bau von Wohnraum verheerende Folgen haben können, die unter allen Umständen vermieden werden müssen. Um die Forschung in diesem Bereich voranzutreiben bietet ein Fragebogen eine sehr gute Möglichkeit, um die verschiedenen Ursachen ausfindig zu machen. Mit den heutigen Technologien ist es kein Problem einen solchen Fragebogen zu realisieren, der für alle Baufirmen zugänglich ist. Außerdem dient das Projekt dazu, seine Kenntnisse im Bereich Webentwicklung zu verbessern. Dazu zählt sowohl die Front-End Entwicklung, mit den dazugehörigen Technologien, als auch die Back-End Entwicklung. Gerade die Front-End Entwicklung bietet dank der vielen Frameworks für JavaScript und CSS viele Möglichkeiten zur Vertiefung seiner Kenntnisse.

## Gruppenmitglieder und Rollen-/Aufgabenverteilung

Die Gruppenmitglieder bestanden aus Alba Cruz Torres, Frank Köhn und Michael Nickel. Allgemein wurde das Projekt in die zwei Hauptaufgaben Front-End und Back-End unterteilt, wobei das Front-End weiter in die Aufgaben HTML/CSS und JavaScript Entwicklung unterteilt wurde. Das Back-End umfasst die Datenbankanbindung und die Java Persistence API, die als Schnittstelle zur Datenbank fungiert. Alba übernahm die Entwicklung des Seitenlayouts mit HTML/CSS, Frank hat sich mit der JavaScript Programmierung beschäftigt und Michael hat die Back-End Entwicklung übernommen. Da jedoch alle Aufgaben miteinander verknüpft sind und eine enge Zusammenarbeit fordern, gab es häufig Überschneidungen bei der Aufgabenverteilung. So war es häufig notwendig, dass jedes Gruppenmitglied auch andere Aufgaben übernommen hat. Durch die enge Zusammenarbeit der gesamten Gruppe, wurden viele Probleme zusammen gelöst.

# Technologien

In den folgenden Punkten werden auf die verschiedenen Technologien, die für das Front-End sowie für das Back-End benutzt werden, eingegangen. Es werden nur allgemeine Beispiele gezeigt, da die Implementation später noch Thema des Berichts ist. Einige Technologien, wie die Java Persistence API die als Schnittstelle zur Datenbank dient, oder das Front-End Framework UIkit, welches zur Gestaltung des Web Interfaces dient, waren bereits vorgegeben. Allerdings war es auch möglich andere Technologien einzusetzen, die die Programmierung besser gestaltet und mehr Übersichtlichkeit in den Programmcode bringt. Zunächst werden die Technologien, die im Front-End eingesetzt wurden, besprochen und anschließend befasst sich das Kapitel mit den Back-End Technologien.

## UIkit[[2]](#footnote-2)

UIkit ist ein unkompliziertes modulares CSS-Framework zur einfachen Gestaltung von Webseiten. Die Einbindung in ein bestehendes HTML Dokument verläuft ganz einfach wie bei jeder anderen CSS Datei auch über das <head> Element.

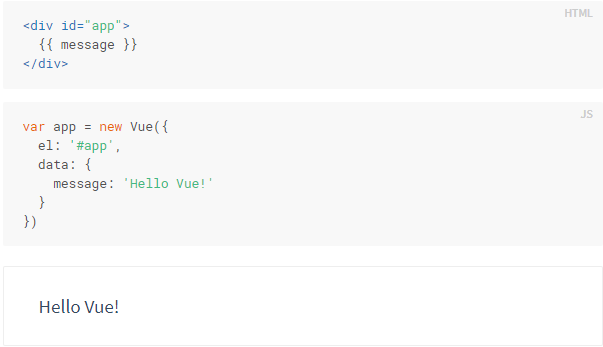
https://i.gyazo.com/563d18563df7ec36ef52567556ada52a.png

1: UIkit Einbindung

Nach der Einbindung kann man die verschiedenen Komponenten von UIkit innerhalb des <body> Elements des HTML Dokuments nutzen. Die Komponenten werden dann über das class Attribut eines HTML Elements eingefügt. So nimmt UIkit dem Programmierer viele Gestaltungsaufgaben ab, da man selbst keine, oder nur sehr wenige, CSS Anweisungen für die verschiedenen Elemente verfassen muss.

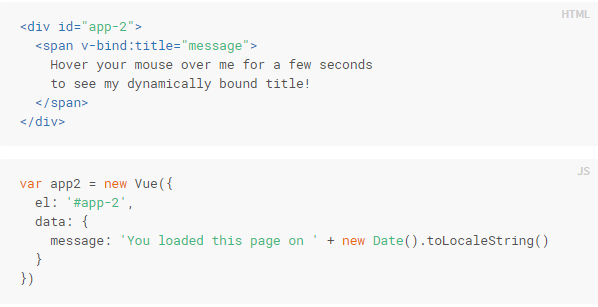
## Vue.js[[3]](#footnote-3)

Vue.js ist ein progressive JavaScript Framework, um User Interfaces zu bauen. Das Framework ist sehr beliebt und wird bei sehr erfolgreichen Internetseiten, wie zum Beispiel Gitlab, Nintendo oder Alibaba, benutzt. Progressive heißt, dass die Hauptbibliothek, die sich auf das Front-End konzentriert, sehr klein ist. Dazu können dann Plugins hinzugefügt werden, wie ein Router oder Server seitiges Rendern. Die Einbindung erfolgt wie bei jeder anderen JavaScript Datei auch über das script-Tag im HTML Dokument. Es gibt aber auch eine Vue CLI, womit man über eine GUI oder über die Konsole, mithilfe von vordefinierten Plugins und Einstellungen, Vue Projekte erstellen kann. Der Vorteil von Vue.js im Vergleich zu jQuery ist, dass Vue.js ein virtuelles DOM aufbaut, welches performanter ist als das übliche DOM, wie es bei jQuery der Fall ist.

Damit die Implementierungsdetails später besser verständlich sind, werden hier die Hauptmerkmale von Vue.js erläutert die bei unserer Anwendung verwendet werden. Die sogenannte Template Syntax sorgt dafür, dass das deklarative Erstellen von Daten zum DOM möglich ist. Was das genau heißt sieht man in diesem Beispiel:

2: Vue.js Beispiel

Durch die geschweiften Klammern im div-Element sorgt jede Änderung an der Variablen „message“ im Datenbestand für eine automatische Aktualisierung des HTML Elements.

Zusätzlich gibt es noch bestimmte Attribute die „directive“ genannt werden. Ein solches Attribut ist „v-bind“.

3: v-bind Beispiel

Durch diese Bindung wird das „title“ Attribut immer auf dem gleichen Stand von der „message“ Variable in der JavaScript Datei gehalten. Das gleiche kann man mit jedem beliebigen Attribut eines HTML-Elements machen.

Ein weiteres „directive“ Attribut ist „v-if“.



4: v-if Beispiel

Das span Element ist durch die „v-if“ Bindung zu der Variablen „seen“ nur sichtbar, wenn die Variable true ist.

Das letzte „directive“ Attribut, welches Anwendung in unserer Software findet, ist das „v-model“. Das „v-model“ Attribut ermöglicht die wechselseitige Bindung zwischen einem Input Feld und dem aktuellen Status der Software (siehe Abbildung 5). Dadurch wird jede Änderung eines Input Feldes direkt in der Ansicht übernommen.



## 

5: v-model Beispiel

## Spring/MVC[[4]](#footnote-4)[[5]](#footnote-5)

Das Spring Framework ist ein Open-Source Framework für die Java-Plattform. Es vereinfacht die Java Programmierung und fördert gute Programmierpraktiken. In unserem Anwendungsfall hat sich Spring angeboten, da das Framework eine Implementierung mit dem Model View Controller (MVC) Architekturmuster ermöglicht.

Das MVC-Konzept ist für die Erstellung von Webanwendungen sehr beliebt, da es eine klare Struktur und eine gute Erweiterbarkeit bietet. Dies ist dadurch möglich, dass bei einer neuen Anwendung das bestehende Modell benutzt werden kann und nur die Steuerung und die Präsentation neu implementiert werden müssen. Wie der Name schon vermuten lässt, kann man das MVC Architekturmuster in drei Komponenten aufteilen: Das Modell (Model), die Präsentation (View) und die Steuerung (Controller). Ein Modell repräsentiert die logische Struktur von Daten, enthält aber keinerlei Informationen zur Benutzeroberfläche. Die Präsentation sorgt für die Darstellung der Oberfläche, in unserem Fall also die Darstellung des Fragebogens über die Webseite. Die Benutzerinteraktionen werden an die Steuerung weitergegeben. Zusätzlich wird die Präsentation über Änderungen der Daten im Modell unterrichtet.

Die Steuerung ist das Bindeglied zwischen dem Modell und der Präsentation. Es sorgt für die Kommunikation zwischen den beiden Komponenten und verwaltet die Präsentation, indem es, wie weiter oben schon beschrieben, die Benutzerinteraktionen entgegennimmt, diese auswertet und entsprechend agiert.

6: MVC

Außerdem enthält Spring das Prinzip der Dependency Injection, wodurch den Objekten die benötigten Ressourcen zugewiesen werden und der Programmierer diese nicht selbst suchen muss.

## JPA[[6]](#footnote-6)

Die Java Persistence API, oder kurz JPA, ist eine Schnittstelle für Java-Anwendungen, die für die Übertragung von Objekten zu Datenbankeinträgen verantwortlich ist. Sie vereinfacht das objektrationale Mapping, indem Java Objekte zur Laufzeit über eine Sitzung in relationale Datenbanken gespeichert werden können.

Hierzu werden in Java Entities erstellt, die im Grunde nichts anderes sind als Plain Old Java Objects (POJO). Diese bilden einzelne Tabellen in der Datenbank ab. Die Attribute einer Klasse entsprechen dabei den Spalten und eine Instanz der Klasse entspricht einer Zeile in der Tabelle. Die Klassen werden als Entities über Annotationen erkannt. Die folgende Tabelle gibt Auskunft über die benutzten Annotationen.

|  |  |
| --- | --- |
| Annotation | Beschreibung |
| @Entity | Die Annotation spezifiziert eine Klasse als Entität. |
| @Table | Diese Annotation deklariert den Namen der Tabelle in der Datenbank. |
| @Id | Spezifiziert den Primärschlüssel der Klasse. |
| @GeneratedValue | Spezifiziert wie die Attribute initialisiert werden können. |
| @Column | Die Annotation spezifiziert ein Attribut beziehungsweise eine Spalte für eine Tabelle. |
| @SequenceGenerator | Dient zur Definition einer Sequence für die @GeneratedValue Annotation. |
| @OneToOne | Definiert eine Eins-zu-Eins Beziehung zwischen zwei Tabellen. |
| @JoinColumn | Spezifiziert eine Assoziation zwischen Entitäten. |

: JPA Annotationen

## REST[[7]](#footnote-7)

Representational State Transfer oder auch REST genannt, ist eine API für die Kommunikation zwischen Client und Server in Netzwerken. Verteilte Systeme können sich so über das Programmierparadigma austauschen. Da das Konzept parallel zu HTTP 1.1 entwickelt wurde, liefert das World Wide Web bereits einen Großteil der nötigen Infrastruktur für REST. Der Datenaustausch erfolgt meist über JSON oder XML. In der Praxis wird REST über HTTP/HTTPS realisiert, indem die Services über die URL angesprochen werden und die entsprechende HTTP-Methode angibt, welche Operation ausgeführt wird. Die häufigsten Methoden werden in der folgenden Tabelle aufgelistet:

|  |  |
| --- | --- |
| GET | Fordert Daten von einem Server an. |
| POST | Übermittelt Daten an einen Server. |
| PUT | Ändert bestehende Daten auf einem Server. |
| DELETE | Löscht bestehende Daten von einem Server. |

2: REST Methoden

## PostgreSQL[[8]](#footnote-8)

PostgreSQL ist ein relationales Open-Source Datenbankmanagementsystem. Es ist frei verfügbar und kann ohne Lizensierung benutzt werden. Ursprünglich wurde es als universitäres Projekt entwickelt und seit 1996 wird es von vielen Programmierern weiterentwickelt. Verwendet wird die SQL Datensprache, jedoch muss man dank JPA keine eigenen SQL Queries verfassen. Hinzu kommt, dass PostgreSQL auf dem Client-Server-Modell basiert, wodurch Client und Server nicht auf demselben Rechner laufen müssen. Da das Smart Monitoring System bereits mit PostgreSQL läuft, wurde auch für dieses Projekt das System verwendet.

# Softwareentwurf

Dieses Kapitel beschäftigt sich mit den Anforderungen der Software und den dazugehörigen Systemmodellierungen und Entwürfen. Der Software Entwurf ist ein wichtiger Bestandteil eines Projektes, um eine hohe Qualität der Software zu garantieren. Außerdem verringert es das Risiko von Fehlentwicklungen und bietet allen Beteiligten zu jedem Zeitpunkt einen Überblick des Software Konzeptes. In den folgenden Punkten werden zunächst die funktionalen sowie nicht funktionalen Anforderungen besprochen, bevor auf die weiteren Spezifikationen der Software, in Form von UML Diagrammen, Use Cases und Mockups, eingegangen wird.

## Funktionale Anforderungen

Grundsätzlich dient der Fragebogen für das Institut für Bauforschung e. V. zum Ausfüllen für Sachverständiger, Baugewerbe und Bauindustrien im Bezug auf Schadensdaten. Die Daten werden in einer Datenbank gespeichert, um hinterher in eine Excel Tabelle, zur weiteren Auswertung, überführt zu werden. Bevor der eigentliche Fragebogen beginnt, erhält der User einen kurzen Informationstext. Wie schon erwähnt wurde, enthält der Fragebogen 20 Fragen, wobei jede Frage eine eigene Seite bekommt.

Das Ausfüllen erfolgt über verschiedene Eingabe- und Bedienelemente. Bei Fragen wo der User mehrere Antworten auswählen darf, werden Checkboxen benutzt. Falls nur eine Antwort ausgewählt werden darf, kommen Radio-Buttons zum Einsatz. Wenn Zahleneingaben nötig sind, wird eine Kombination aus Schiebereglern und Textfeldern benutzt. Textfelder kommen außerdem zum Einsatz, wenn der User einen eigenen Text verfassen soll.

Die Navigation erfolgt über zwei Buttons zum Wechseln der Seiten beziehungsweise zum Starten und Absenden des Fragebogens. Außerdem enthält jede Seite Information zur Fragestellung, damit der User immer weiß wie er die jeweilige Frage zu beantworten hat. Falls Eingaben fehlerhaft sind, wird der User daran gehindert zur nächsten Frage zu springen und erhält Informationen zum Fehler.

Zusätzlich enthält die Software eine Authentifizierung, damit ein User den Fragebogen nicht mehrmals ausfüllen kann.

## Nicht funktionale Anforderungen

Gerade bei einem Fragebogen ist die Usability von besonderer Bedeutung, da sich der User nicht lange mit dem Ausfüllen eines Fragebogens beschäftigen will. Also muss die Seite simpel gestaltet und intuitiv nutzbar sein. Außerdem sollte die Erledigung nicht mehr als 10 bis 15 Minuten beanspruchen.

Zuverlässigkeit muss auch gewährleistet sein, da die korrekte Datenübertragung in die Datenbank essenziell für einen Fragebogen ist, sonst würde dieser seinen gesamten Zweck verfehlen.

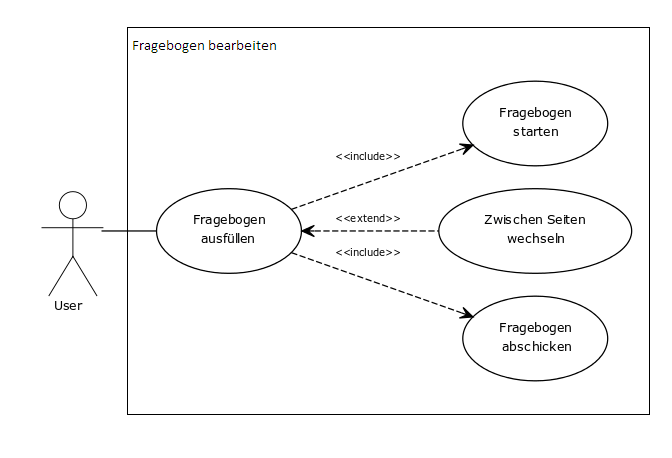
Die Webseite muss von jedem Rechner mit Internetzugang erreichbar sein.

Der Wechsel zwischen den Seiten und die Eingabe über die Bedienelemente muss zudem schnell vonstattengehen. Dies hängt auch mit der Usability der Seite zusammen.

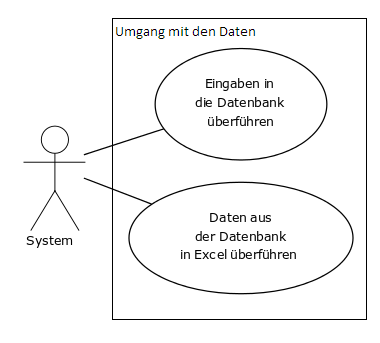
## Use Case

Für den Fragebogen wurden zwei Use Case Diagramme entwickelt. Das erste Diagramm bezieht sich auf den User und seinen Anwendungsfällen. Das zweite Diagramm ist aus Sicht des Systems. Der User füllt den Fragebogen aus, indem er die Fragen beantwortet. Dies beinhaltet auch die Anwendungsfälle „Fragebogen starten“ und „Fragebogen abschicken“. Zudem wird das Ausfüllen durch den Anwendungsfall „Zwischen Seiten wechseln“ erweitert.

Das System dagegen kümmert sich darum die Eingaben eines komplett ausgefüllten Fragebogens in die Datenbank zu überführen und die Daten aus der Datenbank in eine Excel Tabelle einzutragen.

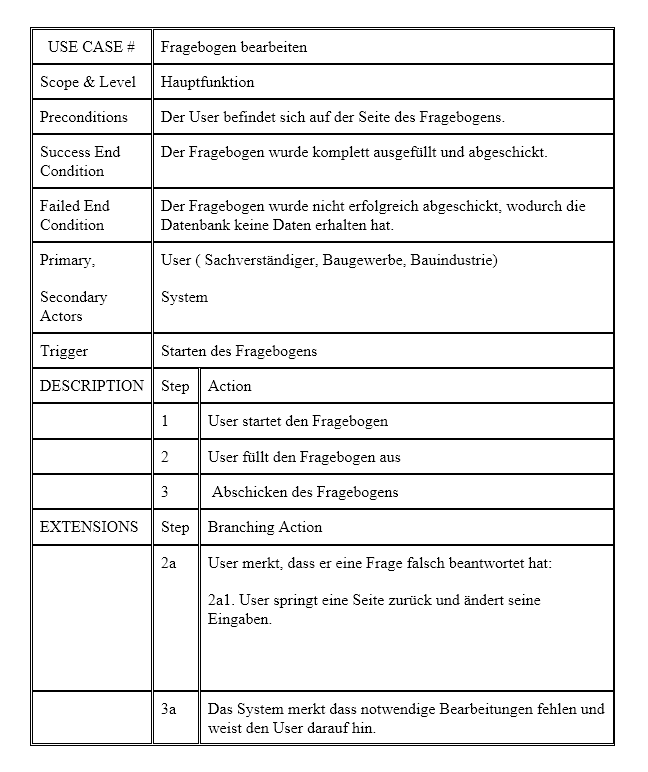


7: Use Case Fragebogen bearbeiten



8: Use Case System

Die folgende Tabelle zeigt die textuelle Beschreibung des Use Cases.



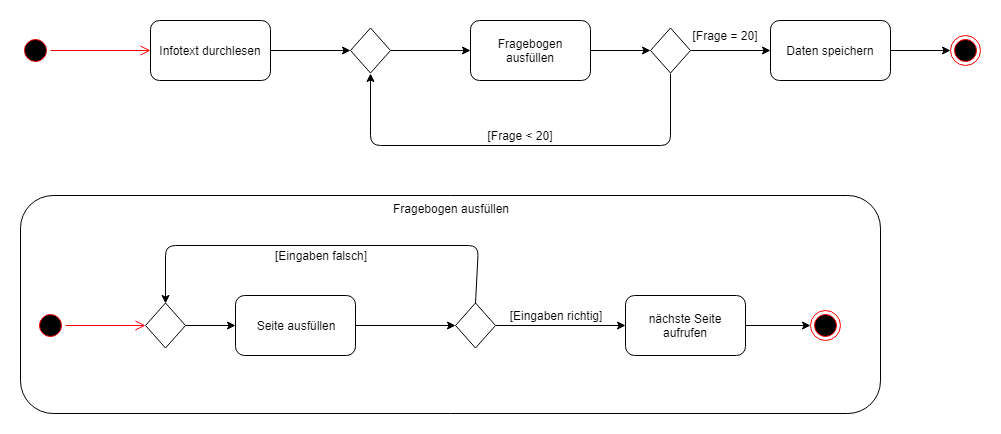
9: Use Case textuell

## UseCaseRelatedInformation.png

## 

10: Use Case related information

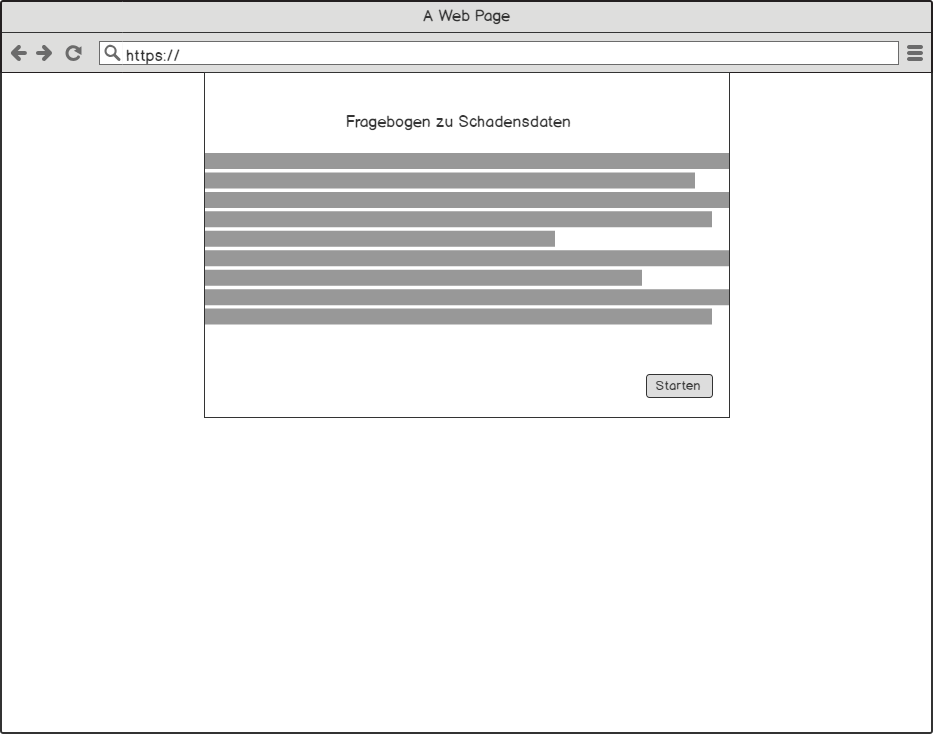
## Aktivitätsdiagramm

Unter einem Aktivitätsdiagramm versteht man ein Verhaltensdiagramm, welches Aktionen und Kontroll- beziehungsweise Datenflüsse eines Systems grafisch darstellt.

11: Aktivitätsdiagramm

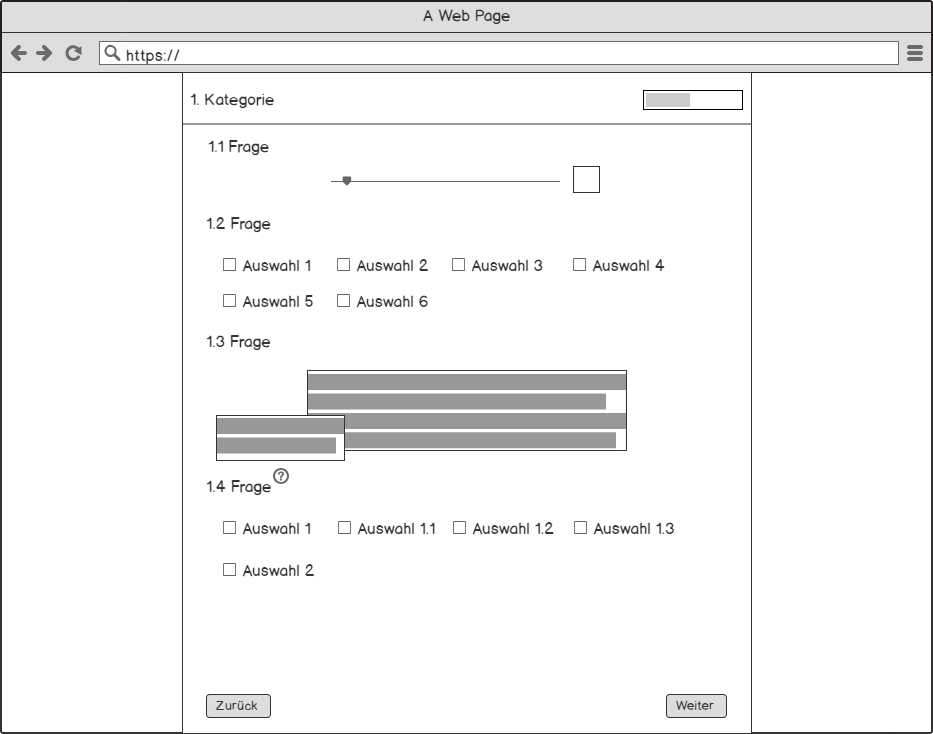
Man sieht im oberen Ablauf die allgemeine Abfolge bei der Bearbeitung des Fragebogens. Der User liest sich zuerst den Informationstext durch, bevor er den eigentlichen Fragebogen ausfüllt. Die Aktion „Fragebogen ausfüllen“ wurde im zweiten Diagramm noch genauer beschrieben. Die Seite wird ausgefüllt und wenn die Eingaben richtig sind, kann der User die nächste Seite beziehungsweise die nächste Frage aufrufen. Ansonsten bekommt der User die Meldung, dass die Eingaben falsch sind und bleibt weiterhin auf der Seite, um die Eingaben zu korrigieren. Die Aktivität „Fragebogen ausfüllen“ wird solange ausgeführt bis die letzte Frage beantwortet wurde, worauf die Speicherung der Daten folgt.

## Mockups

Mockups sind bei der Planung von Webseiten sehr wichtig, da man so sehr schnell eine Visualisierung und einen ersten Eindruck von den Funktionalitäten und der Gestaltung einer Webseite erhält. Die Mockups für den Fragebogen liefern einen Eindruck für die Startseite und die eigentlichen Fragen des Fragebogens.

12: Startseite Mockup

Die Startseite ist sehr einfach gestaltet. Man erkennt den Titel des Fragebogens ganz oben und direkt darunter befindet sich der angesprochene Informationstext, damit der User über alles was er wissen muss Bescheid weiß. Unterhalb des Textes in der rechten Ecke befindet sich dann der Start Button, um die eigentliche Umfrage zu starten.

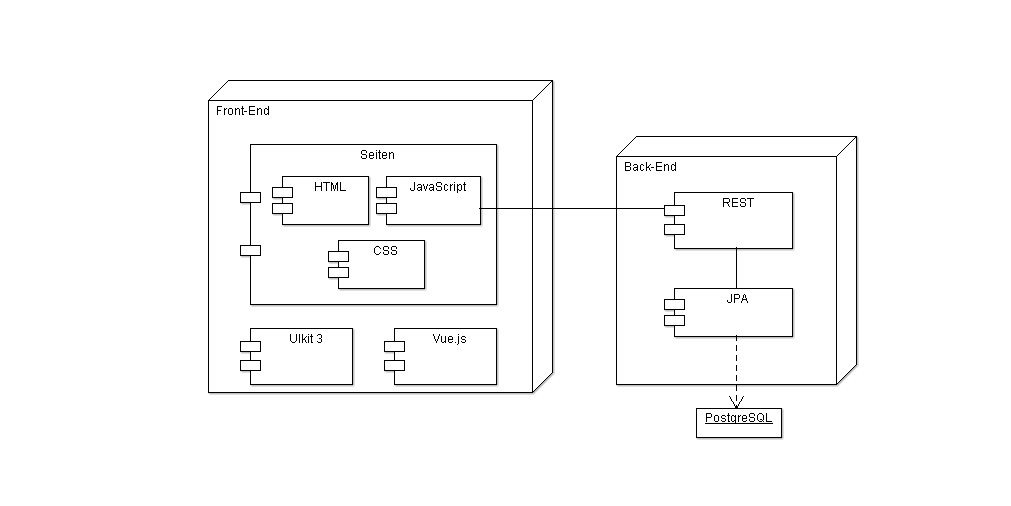


13: Frage Mockup

Das Mockup für die eigentliche Frage enthält den ersten Entwurf des Fragebogens. Ganz oben links befindet sich die Kategorie des Fragebogens und oben rechts befindet sich eine Fortschrittsanzeige, damit der User immer weiß wie viele Fragen er beantwortet hat und wie viele noch übrig sind. Direkt unter den beiden Objekten befinden sich die Fragen dieser Kategorie. Wie schon erwähnt wurde enthält die fertige Umfrage nur eine Frage pro Seite, dennoch bietet das Mockup einen guten Überblick über die erwähnten Eingabe- und Bedienelemente. Die erste Frage zeigt die Kombination aus Schieberegler und Textfeld zur Eingabe einer Zahl. Die zweite sowie vierte Frage zeigen Checkboxen, während die dritte Frage ein Textfeld enthält. Neben der vierten Frage erkennt man auch das kleine Fragezeichen Icon, welches als Hilfsmittel dient und einen Informationstext zur Frage liefert, wenn man mit der Maus herüber gleitet. Unten links und rechts befinden sich die Buttons zum Wechseln der Seite.

## Komponentendiagramm

Komponentendiagramme bieten eine gute Möglichkeit die Struktur eines Systems darzustellen. Es enthält die verschiedenen Komponenten mit deren Schnittstellen und zudem auch die Abhängigkeitsbeziehungen und Konnektoren zwischen den Komponenten. Die Anfertigung eines Komponentendiagramms vereinfacht außerdem den Implementierungsprozess, da das Diagramm als Richtlinie fungiert. Die folgende Abbildung umfasst das Komponentendiagramm unseres Systems.

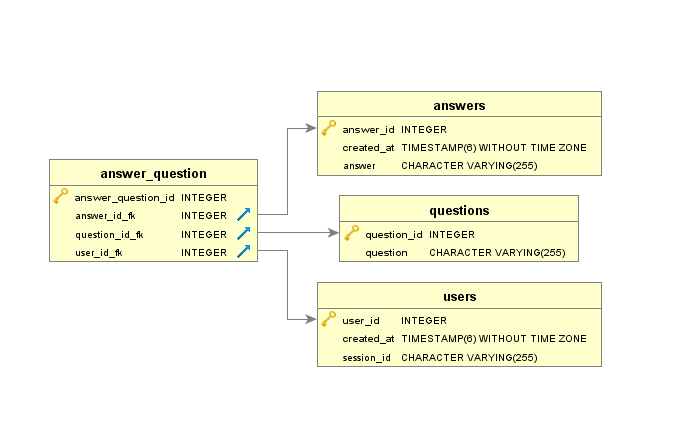
Man erkennt zunächst einmal die Einteilung in die Hauptblöcke Front-End und Back-End. Das Front-End enthält die Komponente „Seiten“, womit die gesamten Dokumente gemeint sind, die vom User wahrgenommen werden. Die Seiten werden noch einmal in die Komponenten HTML, CSS und JavaScript unterteilt, die alle dazu dienen die Webseiten zu gestalten. Des Weiteren umschließt das Front-End auch noch die UIkit 3 und Vue.js Komponenten, welche als Frameworks zur besseren Entwicklung der Seite dienten.

14: Komponentendiagramm

Das Back-End umfasst die REST und JPA Komponente. Die REST Komponente beinhaltet alle Methoden für die Client-Server Kommunikation. JPA hingegen umfasst alle Entitäten zur Zuordnung und Übertragung in die PostgreSQL Datenbank, weshalb man hier eine Verbindung dieser zwei Komponenten erkennen kann.

## Entity-Relationship-Modell

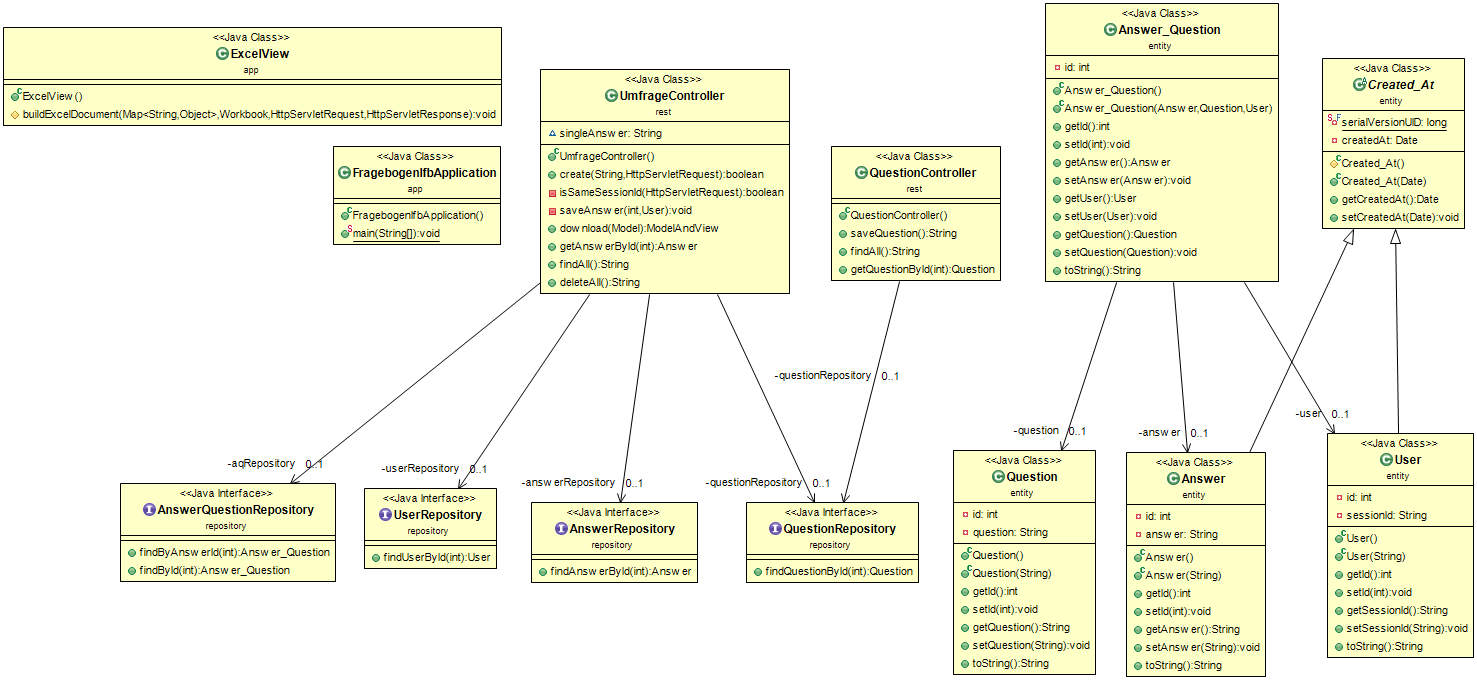
Das Entity-Relationship-Modell zeigt die verschiedenen Entitäten und deren Beziehungen zueinander, welche in der PostgreSQL Datenbank gespeichert sind.



15: Entity-Relationship-Modell

Das Modell enthält 4 Entitäten. Jede Entität besitzt eine ID, welche gleichzeitig der Primärschlüssel jeder Entität ist. Die Fragen der Umfrage werden in der „questions“ Entität über einen String mit maximal 255 Zeichen abgespeichert. Die Antworten werden in der „answers“ Entität gespeichert. Dies geschieht auch über einen String mit maximal 255 Zeichen. Außerdem besitzt die Entität das Attribut „created\_at“, zur Speicherung des Datums wann die Antwort in die Datenbank eingetragen wurde. Die „users“ Entität besitzt auch das Attribut „created\_at“ und eine Session ID, welche verhindert, dass ein User den Fragebogen mehrmals ausfüllt. Genaueres dazu wird später erläutert. Schließlich gibt es noch die „answer\_question“ Entität, die über Fremdschlüssel auf alle anderen Entitäten verweist. So enthält ein Eintrag in der Tabelle den eigenen Primärschlüssel und jeweils die Fremdschlüssel der Entitäten „answers“, „questions“ und „users“.

## Klassendiagramm

Ein Klassendiagramm ist ein Strukturdiagramm zur grafischen Darstellung von Klassen und deren Beziehungen. Das Klassendiagramm für unsere Software bezieht sich nur auf das Back-End, genauer gesagt auf die Java Klassen.

16: Klassendiagramm

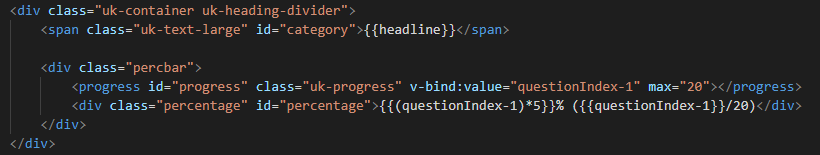
Auf der rechten Seite des Klassendiagramms erkennt man zunächst die JPA Entitäten „Question“, „Answer“, „User“ und „Answer\_Question“. Diese Klassen erstellen die im Entity-Relationship-Modell angesprochenen Entitäten in der PostgreSQL Datenbank. Die Klasse „Created\_At“ dient dabei als abstrakte Klasse die von „Answer“ und „User“ erweitert wird, damit hier keine Redundanzen entstehen. Die Klassen „UmfrageController“ und „QuestionController“ dienen als REST Schnittstellen zur Datenbank. Sie enthalten Methoden zur Verarbeitung und Speicherung der Daten. Man erkennt auch, dass beide Klassen die Repositories für die jeweiligen Entitäten enthalten. Diese Repositories werden zur Laufzeit implementiert und liefern viele Methoden zum Speichern, Löschen und Finden von Entitäten der Datenbank. Schließlich gibt es noch die Klassen „FragebogenIfbApplication“ und „ExcelView“. Die erste Klasse dient zum Starten der Anwendung, während die zweite für die Erstellung des Excel Dokuments zuständig ist.

# Implementierungsdetails

Hier werden aussagekräftige Code-Ausschnitte gezeigt, um das Verständnis für die Software zu verbessern.

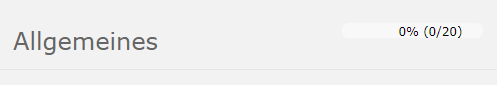
## HTML/CSS/JavaScript

Insgesamt gibt es 4 HTML Dokumente für die Software. Es gibt die Startseite, die den Informationstext enthält, das Impressum, die Datenschutzerklärung und natürlich ein Dokument für alle Fragen. Da die Startseite, das Impressum und die Datenschutzerklärung nicht sehr aufwendig sind und nicht viel Raum für Code-Ausschnitte bieten, fokussiert sich dieser Abschnitt mit dem HTML Dokument für die Fragen und den passenden JavaScript und CSS Ausschnitten.

Zunächst wird die Implementierung der Kategorie und der Fortschrittsanzeige erklärt.

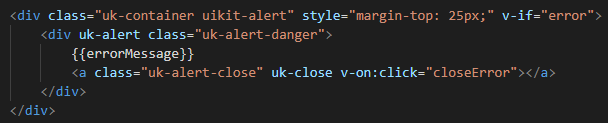
17: HTML: Kategorie und Fortschrittsanzeige

Wie man sehen kann ist sowohl die Kategorie als auch die Fortschrittsanzeige in einem div-Element eingebettet. Dieser Block besitzt durch die UIkit Klasse „uk-heading-divider“ eine Trennlinie, damit die Kategorie und die Fortschrittsanzeige vom Rest des Fragebogens separiert ist. Die Kategorie wird über ein span-Element unter dem Namen „headline“ dargestellt. Die doppelten geschweiften Klammern gehören zur Template Syntax von Vue.js, die bereits vorher im Bericht erklärt wurde. Die Fortschrittsanzeige befindet sich in einem neuen div-Element mit dem class Attribut „percbar“, womit in der CSS-Datei die Weite auf 30% gesetzt wird. Außerdem wird mithilfe von „position: relative“ und „float: right“ dafür gesorgt, dass die Fortschrittsanzeige rechts platziert wird. Die eigentliche Fortschrittsanzeige wird über das progress Element und der UIkit Komponente „uk-progress“ realisiert. Über das „v-bind“ Attribut wird dafür gesorgt, dass der Wert (value) immer auf demselben Stand ist wie der Index für die Frage („questionIndex“). Da wir 20 Fragen in der Umfrage haben, wird das „max“ Attribut auf 20 gesetzt. Der User sieht dann zunächst den Prozentsatz und zusätzlich noch wie viele Fragen er bereits beantwortet hat. Dies wird auch wieder über „questionIndex“ geregelt. Das Ganze sieht im Browser dann so aus:



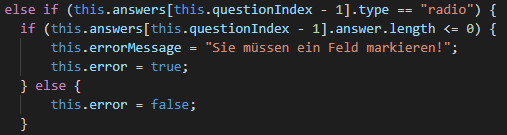
18: Kategorie und Fortschrittsanzeige in der Browseransicht

Ein weiteres wichtiges Merkmal der Seite sind die Fehlermeldung, falls Eingaben fehlen oder falsche getätigt wurden.

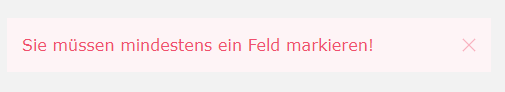


19: HTML: Fehlermeldung

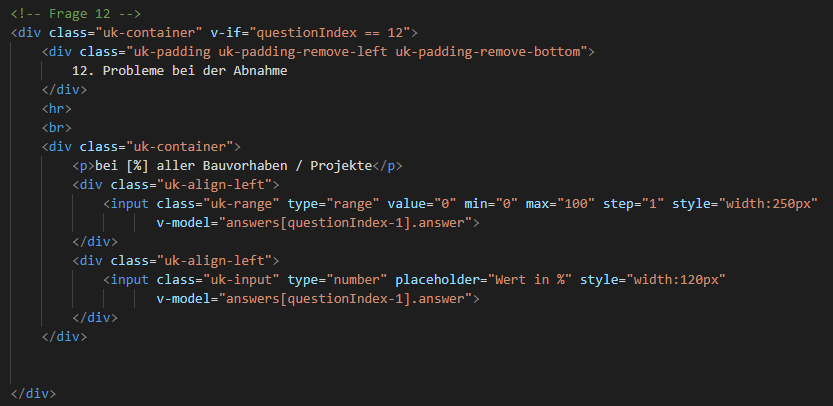
Die Fehlermeldung wird über die UIkit Komponente „UIkit-alert“ dargestellt, die bereits die passende Darstellung für unseren Anwendungsfall liefert. Das div-Element wird über das „v-if“ Attribut an die Variable „error“ gebunden, die je nachdem ob ein Fehler gefunden wurde oder nicht, true oder false ausgibt. Falls die Variable nun auf true steht, wird die Fehlermeldung ausgegeben. Die Validierung der Eingaben erfolgt über eine JavaScript-Methode, indem je nach Typ der Frage (Checkbox, Radio-Button etc.) bestimmte Bedingungen erfüllt sein müssen, damit kein Fehler erscheint. Die folgende Abbildung zeigt die Überprüfung für Fragen mit Radio-Buttons.



20: Validierung von Fragen die Radio-Buttons beinhalten

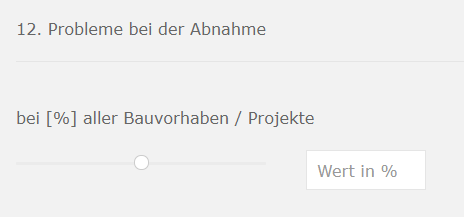
Bei den Radio-Buttons wird überprüft ob eine Antwort vorhanden ist. Falls Sie nicht in den Daten enthalten ist wird „error“ auf true gesetzt und eine Fehlermeldung (errorMessage) erzeugt. Dies sieht dann so aus:

21: Fehlermeldung Browseransicht

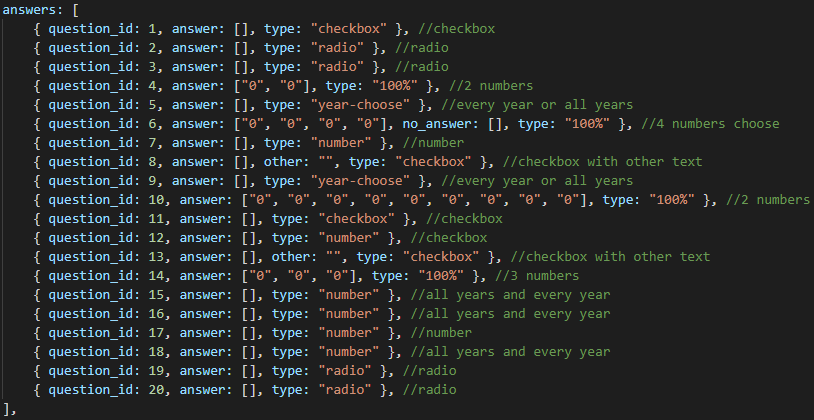
Die eigentliche Frage sieht folgendermaßen aus:

22: Beispielfrage

Da alle Fragen in einem HTML Dokument gespeichert werden, wird wieder die Variable „questionIndex“ benutzt, um je nach Index die richtige Frage für den User darzustellen. Dazu wird jede Frage in ein div Element gepackt und mithilfe des „v-if“ Attributes der Index der Frage überprüft. Hier sehen wir, dass die Frage nur angezeigt wird, falls der Index bei 12 ist. Die grobe Unterteilung innerhalb des Elements sieht bei jeder Frage gleich ist. An erster Stelle steht die Frage, die in einem eigenen div-Element verpackt ist. Da drunter befinden sich dann die Eingabeelemente. In diesem Fall handelt es sich um einen Schieberegler mit einem Inputfeld, wobei beide voneinander abhängig sind, das heißt wenn eine Änderung bei dem Schieberegler vorgenommen wird ändert sich auch der Input und anders herum. Für die Elemente werden wieder die passenden UIkit Komponenten, in diesem Fall „uk-range“ und „uk-input“, benutzt. Die Browseransicht sieht dann so aus:



23: Beispielfrage in der Browseransicht

Wie man im Beispiel erkennen kann erfolgt die Speicherung über das „v-model“ Attribut, indem die Eingaben mit dem jeweiligen Index der Frage im „answers“ Array gespeichert werden. Dieses JSONArray sieht folgendermaßen aus:

24: Speicherung der Antworten

Das JSONArray enthält 20 verschiedene JSONObjects die jeweils eine Frage des Fragebogens repräsentieren. Jedes Objekt enthält den Index der Frage, die Antwort und den Typen. Zweimal tritt auch noch der Schlüssel „other“ auf. Dieser wird gebraucht, um die Antworten der Fragen zu speichern die ein Textfeld für größere Eingaben besitzen.

Wenn man sich nun nochmal das v-model-Attribut aus Abbildung 18 anguckt, sieht man, wie die Speicherung erfolgt. Und zwar werden die Werte aller Antworten je nach Frage im „answer“ Schlüssel des JSONArrays gespeichert.

Wenn die letzte Frage erreicht wurde und der User auf den Button „Beenden“ gedrückt hat, wird die Methode mit dem POST Request aufgerufen.

25: POST Request

Da der Server die Daten als String braucht, wird das JSONArray mit den Antworten über die „stringify“ Methode in einen String konvertiert. Dann werden noch die richtigen Header für den Request gesetzt. In diesem Fall ist es nur nötig den Content-Type auf application/json und die Zeichenkodierung auf UTF-8 zu setzen. Schließlich erfolgt der Request mit der richtigen Adresse, den Antworten und den Headern. Die Antwort vom Server ist nun entweder true oder false, je nachdem ob der User den Fragebogen zuvor schon mal ausgefüllt hat. Wenn dies der Fall ist, bekommt der User die Fehlermeldung „Sie haben die Umfrage bereits durchgeführt!“, sonst wird der User zur nächsten Seite weitergeleitet.

## Java

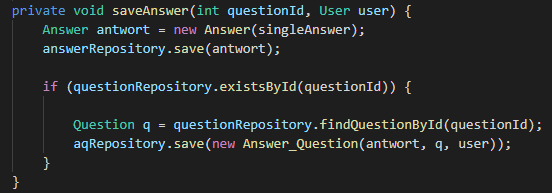
Die wichtigste Methode des Back-Ends ist die Verarbeitung der Antworten und die anschließende Speicherung in der Datenbank. Die folgende Methode ist dafür zuständig:

26: Verarbeitung der Antworten

Zunächst fallen die Spring Annotation „@CrossOrigin“ und „@PostMapping“ auf. Die erste Annotation ist dafür zuständig, um Zugriffe von einer Seite auf Ressourcen eines anderen Servers zu ermöglichen. Normalerweise braucht man die Annotation in unserem Anwendungsfall nicht, da wir Ressourcen immer vom gleichen Server brauchen, jedoch bekommt man gelegentlich Cross-Origin Fehlermeldung, wenn man Chrome benutzt, weshalb die Annotation hier verwendet wird. „@PostMapping“ ist die Kurzform für „@RequestMapping(method = RequestMethod.POST)“ und dient zum Verarbeiten von ankommenden POST Requests. Das „value“ Attribut in der Annotation bindet die Methode an einen bestimmten Pfad. In unserem Fall also „/saveAnswers“.

Dann werden noch die Attribute „produces“ und „consumes“ verwendet, die der Methode einmal sagen welcher Medientyp produziert wird und welcher Typ konsumiert wird. Man erkennt, dass der Content-Type unseres Requests äquivalent zum konsumierten Medientypen ist. Die Parameter der Methode sind beinhalten einmal „@RequestBody final String answer“ für den Zugriff auf den Body des HTTP-Requests, was in unserem Fall das JSONArray mit den Antworten wäre. Der zweite Parameter „HttpServletRequest request“ wird benutzt, um auf die Session ID des Requests zuzugreifen.

In der Methode wird zunächst überprüft ob der User den Fragebogen bereits ausgefüllt hat, indem die gesamte User Tabelle der Datenbank durchgegangen wird, um zu schauen ob die Session ID bereits enthalten ist oder nicht. Wenn die Session ID bereits vorkam, hat der User den Fragebogen bereits ausgefüllt und die Antworten werden nicht gespeichert. Wenn der User den Fragebogen zum ersten Mal ausgefüllt hat, wird die Session ID zunächst in der Tabelle gespeichert. Danach wird das JSONArray durch iteriert. Dabei wird für jedes Objekt der Index der Frage gespeichert („questionId“), bevor die Antwort des Objektes im String „singleAnswer“ gespeichert wird. Hier wird zuvor noch unterschieden, ob die Antwort in einem JSONArray gespeichert wurde, falls eine Frage mehrere Eingabemöglichkeiten bietet, oder nur eine Antwort beinhaltet. Am Ende einer Iteration wird dann die Methode „saveAnswer“ aufgerufen, welcher die „questionId“ und den „user“ als Parameter bekommt und für die Speicherung einer Antwort für eine einzelne Frage zuständig ist (siehe Abbildung 27).



27: Speicherung einer Antwort in der Datenbank

Hier wird ein neues „Answer“ Objekt mit der Antwort („singleAnswer“) erzeugt. Das Objekt wird dann mithilfe des Repositories für die Tabelle der Antworten („answers“) in der Datenbank gespeichert. Außerdem erfolgt die Speicherung in der „answer\_question“ Tabelle, indem ein neues „Answer\_Question“ Objekt mit der Antwort, der dazugehörigen Frage und dem User erzeugt wird, welches dann auch mithilfe eines Repositories in der Datenbank eingetragen wird.

# Fazit

Das Projekt ermöglichte es, viel Erfahrung in der Umsetzung von Webanwendungen zu sammeln. Besonders die Nutzung einiger Frameworks für das Front-End und Back-End haben neue Erkenntnisse gebracht. Außerdem war die Zusammenarbeit mit dem Institut für Bauforschung e. V. sehr erfreulich, da die Anwendung auch einen Nutzen gefunden hat.

## Ausblick

Durch die gute Erweiterbarkeit ist es möglich den Fragebogen dynamisch zu gestalten, um weitere Fragen hinzuzufügen. Dies ermöglicht weitere Anwendungsfälle außerhalb der Bauschadensforschung.

# Installationshinweise

Die Schritte 1 bis 5 kann man vom Wiki Eintrag des Smart Monitoring Systems im Gitlab übernehmen.[[9]](#footnote-9) Danach müssen die Fragen über den Aufruf einer bestimmten URL gespeichert werden. Dies geschieht über den relativen Pfad „/saveAllQuestions“.

Wenn man die Applikation zum Beispiel mit dem Kontextpfad „/umfrageIFB“ gestartet hat, muss man die URL „scl2-ifm-min.ad.fh-bielefeld.de:8080/umfrageIFB/saveAllQuestions“ aufrufen, damit die Fragen in der Datenbank gespeichert werden.

# Benutzung

Wenn man den Kontextpfad vom Installationshinweis benutzt hat, ruft man die URL

„scl2-ifm-min.ad.fh-bielefeld.de:8080/umfrageIFB/index.html“ auf, um zur Startseite zu gelangen. Der restliche Verlauf des Fragebogens ist selbsterklärend, da er sich nicht von anderen Fragebögen unterscheidet.

Zum herunterladen der Excel Tabelle muss die URL

„scl2-ifm-min.ad.fh-bielefeld.de:8080/umfrageIFB/download“ aufgerufen werden. Nach dem Aufruf öffnet sich der Datei Explorer und man hat die Möglichkeit, die Excel Datei in einem beliebigen Speicherort zu speichern.

# Literaturverzeichnis

*Bauherren-Schutzbund e. V.* (kein Datum). Von https://www.bsb-ev.de/fileadmin/user\_upload/Bauherren-Schutzbund/Aktuell/Studien/15\_Forschungsbericht\_IFB\_Bauschaeden\_und\_Bauschadenkosten\_2015.pdf abgerufen

*Installationsanleitung - Produktiveinsatz*. (kein Datum). Von http://git01-ifm-min.ad.fh-bielefeld.de/Forschung/scl/2015\_03\_SCL\_SmartMonitoring\_Backend/wikis/Installationsanleitung-Produktiveinsatz abgerufen

*JPA (Java Persistence API)*. (kein Datum). Von https://www.torsten-horn.de/techdocs/java-jpa.htm abgerufen

*Model View Controller*. (kein Datum). Von https://de.wikipedia.org/wiki/Model\_View\_Controller abgerufen

*PostgreSQL*. (kein Datum). Von https://www.postgresql.org/ abgerufen

*Representational State Transfer*. (kein Datum). Von https://de.wikipedia.org/wiki/Representational\_State\_Transfer abgerufen

*Spring*. (kein Datum). Von https://spring.io/ abgerufen

*UIkit*. (kein Datum). Von https://getuikit.com/docs/introduction abgerufen

*Vue.js*. (kein Datum). Von https://vuejs.org/v2/guide/ abgerufen

1. (Bauherren-Schutzbund e. V.) [↑](#footnote-ref-1)
2. (UIkit) [↑](#footnote-ref-2)
3. (Vue.js) [↑](#footnote-ref-3)
4. (Spring) [↑](#footnote-ref-4)
5. (Model View Controller) [↑](#footnote-ref-5)
6. (JPA (Java Persistence API)) [↑](#footnote-ref-6)
7. (Representational State Transfer) [↑](#footnote-ref-7)
8. (PostgreSQL) [↑](#footnote-ref-8)
9. (Installationsanleitung - Produktiveinsatz) [↑](#footnote-ref-9)