Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Eine Webanwendung zur Verwaltung einer Fehler-Maßnahmen-Datenbank

Anh Nghia Tran

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dr. Ivan Bogicevic, Thomas Schäfter, Maximilian Ast
Beginn am:	01. Dezember 2017
Beendet am:	01. Juni 2018
CR-Nummer:	D.2.2, H.3.4

Kurzfassung

Die Bestandsteile dieser Bachelorarbeit bestehen aus der Konzeption und Entwicklung einer Webanwendung unter der Verwendung des neuen Webtechnologie-Frameworks „Angular“. Diese Webanwendung ermöglicht dem Nutzer benutzerfreundlich die sogenannte Fehler-Maßnahmen-Datenbank (FMD) zu verwalten. Außerdem passt diese Webanwendung sich automatisch an die Auflösung und Darstellung des jeweiligen Endgeräts an. Um die Webanwendung zu realisieren, werden ausführlich die Softwareentwicklungsschritte beschrieben und ausgeführt sowie das verwendete Framework „Angular“ detailliert untersucht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API Programmierschnittstelle. 8, 16

BCI Bosch Connected Industry. 8, 11

CDK Component Development Kit. 8, 42

CLI Command Line Interface. 8, 36

CSS Cascading Style Sheet. 8, 36

DB Datenbank. 8, 20

DOM Document Object Model. 8, 67

FMD Fehler-Maßnahmen-Datenbank. 3, 8

FR Fehlerrate. 8

GUI Graphische Benutzeroberfläche. 7, 8

HTML Hypertext Markup Language. 8, 31

HTTP Hypertext Transfer Protocol. 8, 43

ID Identifier. 8, 61

IDE Integrierte Entwicklungsumgebung. 8, 47

IT Informationstechnik. 8, 11

JSON JavaScript Object Notation. 8, 24

KBS Knowledge-Base-Service. 8, 16

LAN Local Area Network. 8, 11

LCID Language Code Identifier. 8, 61

NPM Node Package Manager. 8, 36

PC Computer. 8, 11

REST Representational-State-Transfer. 8, 13

RxJS Reactive Extensions für JavaScript. 8, 40

SCSS Simple Cascading Style Sheet. 8, 32

SQL Structured Query Language. 8, 24

TSC Technical Steering Committee. 8, 36

URL Uniform Resource Locator. 8, 35

UX User Experience. 8, 45

WebUI Benutzeroberfläche der Webanwendung. 8, 13

XML Extensible Markup Language. 8, 42

1 Einleitung und Zielsetzung

1.1 Einleitung

In der Epoche der ersten industriellen Revolution wurden Maschinen mit Wasser- und Dampfkraft angetrieben. Wenn heute über diese Epoche gesprochen wird, so wird sie als Industrie 1.0 bezeichnet. Durch elektrische Energie und die Produktion an Fließbändern entstand die zweite industrielle Revolution (Industrie 2.0). Diese Revolution ebnete den Weg für die Massenfertigung und Automatisierung. In der dritten industriellen Revolution (Industrie 3.0) wurde die Produktion mit dem Einsatz von Elektronik und Informationstechnik (IT) weiter automatisiert. Heute befinden wir uns in der vierten industriellen Revolution (Industrie 4.0), welche den Fokus auf die Digitalisierung analoger Techniken und die Vernetzung dieser legt.[Fri14]

In der heutigen Zeit, der Industrie 4.0, verwirklichen viele Unternehmen die Produktionsprozesse überwiegend über ein Netzwerk (Local Area Network (LAN) / Internet / usw.). Aus diesem Grund wird es immer mehr Bedarf für webbasierte Anwendungen zur Kommunikation mit Anlagen geben. Mit der heutigen Technologie unterscheidet sich die Funktionalität einer Webanwendung kaum von einer Desktopanwendung, welche auch als lokal installierbare Anwendung bezeichnet werden kann.

Einer der wichtigsten Vorteile einer Webanwendung ist, dass die Algorithmen und logischen Funktionen auf einem Server liegen und dort ausgeführt werden. Im Gegensatz dazu liegen bei einer Desktopanwendung die Algorithmen und logischen Funktionen auf dem Computer (PC) oder auf der Maschine des Besitzers und werden dort ausgeführt. Außerdem spielt das Betriebssystem des Anwenders beim Verwenden einer Webanwendung keine wichtige Rolle mehr. Der Nutzer braucht nur noch einen beliebigen Webbrowser zu starten und die Webadresse einer Webanwendung einzugeben. Zudem werden bei einer Webanwendung weniger Systemressourcen vom Endgerät beansprucht, da die Berechnungen vom Server vorgenommen werden.

Bei einer existierenden, neuen Version der Anwendung muss der Nutzer eine Aktualisierung auf seinem Gerät nicht durchführen, da er direkt auf die neue Version der Webanwendung zugreifen kann. Ein weiterer wichtiger Vorteil beim Verwenden einer Webanwendung ist die Mobilität. So kann beispielsweise ein Operator bei Stillstand einer Produktionsanlage mit Hilfe einer Webanwendung direkt auf die FMDen zugreifen, um die Fehler zu analysieren bzw. mögliche Lösungsmöglichkeiten zu diesen Fehlern suchen. Dabei muss er nicht an der Maschine sitzen, um die Fehleranalyse vorzunehmen, sondern kann mit einem Mobilgerät, das eine bestehende Netzwerkverbindung hat, an einem beliebigen Ort diese Störung analysieren und gegebenenfalls beheben. Somit werden die Stillstandszeiten verkürzt und die Produktionszeit erhöht.

In vielen Unternehmen wird Industrie-4.0-Technologie in der Produktion eingesetzt. Das Unternehmen **BOSCH** wendet auch diese Technologie als neue Geschäftseinheit an, welche auch als Bosch Connected Industry (BCI) bezeichnet wird. BCI setzt sich als Ziel fest, dass durch den

Einsatz dieser Technologie die Kunden bei der kompletten Wertstromvernetzung unterstützt werden können. Die einzelnen Linienverbände sowie deren Intra- und Extralogistik lassen sich miteinander vernetzen. Die Mitarbeiter werden regelmäßig über zahlreiche Anwendungen und Softwaredienste unterstützt, um die aktuellen Informationen über den Zustand der Produktion, den Standort oder den Liefertermin der Waren zu erhalten. Außerdem lassen sich interne Transportprozesse sowie außerbetrieblichen Warentransport rund um die Uhr überwachen und zurückverfolgen.[Wei18]

1.2 Motivation

Wie in der Einleitung bereits erwähnt, wird es in der Produktion immer wichtiger, die Stillstandszeiten zu verringern. Durch den Ausfall einer Produktionsanlage entstehen schwerwiegende Konsequenzen in Hinsicht auf die Wirtschaftlichkeit eines Unternehmens.

Mit Hilfe mordener Industrie-4.0-Lösungen von Bosch BCI können Maschinenfehler in einer kürzeren Zeit behoben werden. Außerdem ist es mit einer existierenden FMD möglich die auftretenden Fehler zu analysieren, die möglichen Ursachen zu identifizieren und automatisch Vorschläge für entsprechenden Gegenmaßnahmen zu erhalten.

Dabei wäre es gut, wenn zu jedem Maschinenfehler eine erfolgreiche Maßnahme in der FMD existieren würde. Das Problem besteht darin, wie der Operator die Fehler und Maßnahmen verwalten kann. Die Verwaltung der FMD ist zurzeit noch sehr umständlich. Die Unterstützung durch Software zum Erstellen, Bearbeiten und Erweitern der Datensätze ist noch immer nicht zufriedenstellend.

Hauptziel meiner Bachelorarbeit ist die Entwicklung einer Webanwendung zur Verwaltung einer FMD in der Robert Bosch GmbH. Es ermöglicht dem Nutzer diese Datenbank auf einer benutzerfreundlichen Weboberfläche zu verwalten und in dieser zu navigieren. Außerdem hat der Operator die Möglichkeit direkt bei einer fehlerhaften Station mit seinem Mobilgerät die angezeigten Fehler-Codes nachzuschlagen und die vorgeschlagenen Maßnahmen durchzuführen. Bei einer neuen, erfolgreichen Maßnahme oder bei einem neuen Fehler kann er auch direkt die Datenbank mit Hilfe der Webanwendung erweitern. Eine Bewertung der vorgeschlagen Maßnahmen, welche er durchgeführt hat, kann er dabei ebenfalls abgeben.

1.3 Aufbau der Arbeit

Meine Bachelorarbeit werde ich wie eine typische Erstellung einer Webanwendung strukturieren. Die Struktur des Bachelorprojekts besteht aus:

- Analyse und Durchführung der Konzeption
- Implementierung des Frontends der Webanwendung
- Realisierung der Verbindung zum Service, welche mit der Datenbank in Verbindung steht
- Evaluierung der Anwendung mit Oberflächen-Tests, Usability-Test und Abgabe des fertigen Produkts.

Die Einrichtung des Backends, Services und der Implementierung der Datenbank steht unter der Verantwortung anderer angestellten Entwickler des Unternehmens. Deswegen wird dieser Teil nicht weiter in meiner Bachelorarbeit vertieft.

Weil es sich hier um eine neue Webanwendung handelt, wird zuerst im **Kapitel 2** die Anforderung zur Anwendung analysiert und konzipiert. Nach der abgeschlossenen Anforderungsanalyse werden anhand dieser im **Kapitel 3** Mockups kreiert. Diese Mockups stehen für einen groben Entwurf der Benutzeroberfläche der Webanwendung (WebUI) und geben eine Übersicht über die Webanwendung. Im **Kapitel 4** wird die angewandte Programmiersprache, Austauschtechnik zwischen WebUI und Service sowie die Representational-State-Transfer (REST) Schnittstelle angesprochen. Nachdem die Web-Entwicklungswerkzeuge besprochen wurden, geht es mit **Kapitel 5** über die Implementierung der Webanwendung und die Logik der Verbindung zwischen WebUI und Web Service weiter. Nach der erfolgreichen Implementierung der Webanwendung wird die Evaluierung in **Kapitel 6** durchgeführt. Hier wird die WebUI und die Logik der Webanwendung getestet. Außerdem wird Feedback in Form von Ausfüllen eines Fragebogens von Testnutzern, die die Anwendung testen und nutzen, protokolliert. Zum Schluss wird es im **Kapitel 7** eine Zusammenfassung der Bachelorarbeit geben und ein Ausblick formuliert.

1.4 Verwandte Themen

Nach langer Literaturrecherche stoße ich auf einige Fehlermanagement-Unternehmen, die in einer ähnlichen Weise wie die Lösung in meiner Bachelorarbeit funktionieren.

Das Fehlermanagement sammelt größtenteils webbasiert die Daten. Die internen und externen Fehler werden detailliert erfasst und analysiert. Durch die erhaltenen Fehlerdaten werden die Korrekturmaßnahmen definiert und vorschlagen.

Die erhaltenen Daten an Fehlern und Maßnahmen werden nicht mehr komplett auf verteilten, Excel-basierten Speichern gelagert, sondern werden in einer gepflegten Datenbank integriert. Bei Bedarf kann der Nutzer problemlos nach Maßnahmen für einen bestimmten Fehler suchen und diese anschließend durchführen, damit in der Produktion keine Blockaden entstehen.

2 Anforderungsanalyse und Konzeption

In diesem Kapitel werden die Anforderungen analysiert und schließlich daraus ein Konzept der Webanwendung dargestellt. Erst wird eine allgemeine Struktur, wie das Projekt systematisch aufgebaut sein soll, festgelegt.

Anschließend wird eine Kurzfassung des aktuellen Zustands (Ist-Analyse) von dem Projekt vorgelegt. Anhand dieser Ist-Analyse erfolgt eine Analyse der Anforderungen, welche in funktionale und nicht-funktionale Anforderungen aufgeteilt werden.

Nach der Anforderungsanalyse werden die Anwendungsfälle erforscht und dargestellt, damit die Stakeholder der Anwendung festgelegt werden können. Nach der Festlegung der Stakeholder der Anwendung wird am Ende die Architektur der Webanwendung erklärt.

2.1 Projektstruktur

Aus den Projektphasen (Abbildung 2.1) ergibt sich die Projektorganisation. Es entspricht dem typischen sequetiellen Vorgehensmodell zur Softwareentwicklung einschließlich der Entwicklung der Webanwendungen.

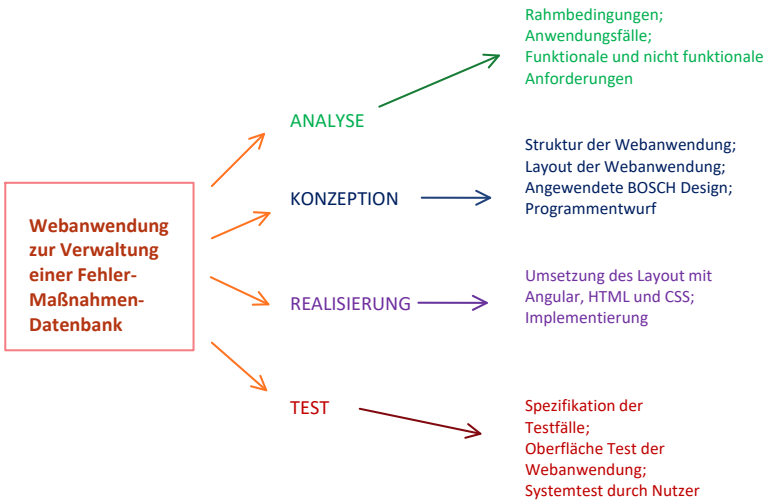

Abbildung 2.1: Softwareprojektstrukturplan

Quelle: eigene Abbildung

Eine Analyse von funktionalen und nicht-funktionalen Anforderungen, wie auch eine Analyse der Anwendungsfälle wird in der ersten Phase unternommen. Anhand dieser Analyse wird die Strukturierung des Inhalts und ein passendes Design der Webanwendung erstellt. Das daraus entstehende Konzept wird technisch in eine Anwendung und am Ende getestet. Die entstehende Webanwendung wird anschließend im Betrieb angewendet und bei Bedarf weiterentwickelt.

2.2 Ist-Analyse

Am Anfang meiner Bachelorarbeit wurde in der Firma über den Zustand der aktuellen Lösung für den Zugriff auf die FMD besprochen.

Bis jetzt existiert bereits eine grundlegende Komponente, welche „Knowledge-Base-Service (KBS)“ genannt wird. In der Abbildung 2.2 wird dieses System grob dargestellt. Dieser KBS bietet eine sogenannte Programmierschnittstelle (API) an, damit externe Systeme über REST mit dieser kommunizieren können. Innerhalb dieses Systems existiert eine FMD und eine Service-Komponente, welche die Funktionalität der API anbietet und gleichzeitig mit der Datenbank kommunizieren kann. Die Bestandsteile vom Service sowie der Datenbank sind unter Verantwortung anderer Mitarbeiter und werden hier nicht weiter vertieft.

Um einen Überblick über die Struktur der REST API des Systems zu erhalten, wird ein Model-Ausschnitt entnommen (siehe Listing 2.1). Dieser Code-Ausschnitt zeigt, dass das zurückgelieferte REST-Ergebnis (RestResult) aus einer Fehlerobjekte-Liste besteht.

Diese Liste beinhaltet mehrere Fehler-Objekte (KsError). Ein Fehler-Objekt besitzt wiederum mehrere Variablen, wie Kennzeichnen (id), Mandatsreferenz (mandateId), Erstellungsdatum (creationDate), Änderungsdatum (changeDate), eine Liste von Fehlercode-Objekte (KsErrorCode), welche den Fehler spezifiziert, eine Liste der Ursachen-Objekte (KsCause), welche auch aus diesem Fehler stammt und eine Liste von sprachenabhängigen Beschreibungs-Objekte (KsTranslation), welche diese Fehler beschreiben.

Genau sowie KsError besitzt KsErrorCode eine Variable id und es existieren dazu weitere Variablen wie Codesnummer (code), Kontextswert (contextValue) und Kontextstyp (contextType). Das KsCause-Objekt besitzt eine Variablen id, creationDate, changeDate, eine Liste aus KsTranslation-Objekte. Ergänzt wird dieses Objekt durch eine Liste von Lösungs-Objekten (KsHint).

Das KsTranslation-Objekt besitzt ebenfalls eine Variable id. Zusätzlich enthält es weitere Variablen wie den Title (title), eine Beschreibung (description) und eine Microsoft Locale ID (lcid), welche die Sprache spezifiziert. Wie KsCause beinhaltet KsHint die Variable id, creationDate, changeDate und eine Liste aus KsTranslation-Objekten. Eine Ergänzung zum KsHint ist die Varialbe Bewertung, welche aus einer Liste von Bewertungs-Objekten (KsRating) stammt. Diese KsRating-Objekt besitzt auch, wie andere Objekte, die Variable id, creationDate und wird mit den Variablen Bewertungswert (value), lcid, eine Kommentar (comment) erweitert.

Nach der groben Betrachtung lässt sich diese aktuelle Lösung nutzen. Das Fazit, nach einem Gespräch mit den Mitarbeitern des Unternehmens, ist folgendes: Dieses System wird in Zukunft als eine Nutzerschnittstelle in mehreren Anwendungen des Unternehmes intergriert. Leider ist das erhaltende Ergebnis durch die Nutzung von REST auf der vorhandenden API sehr unübersichtlich.

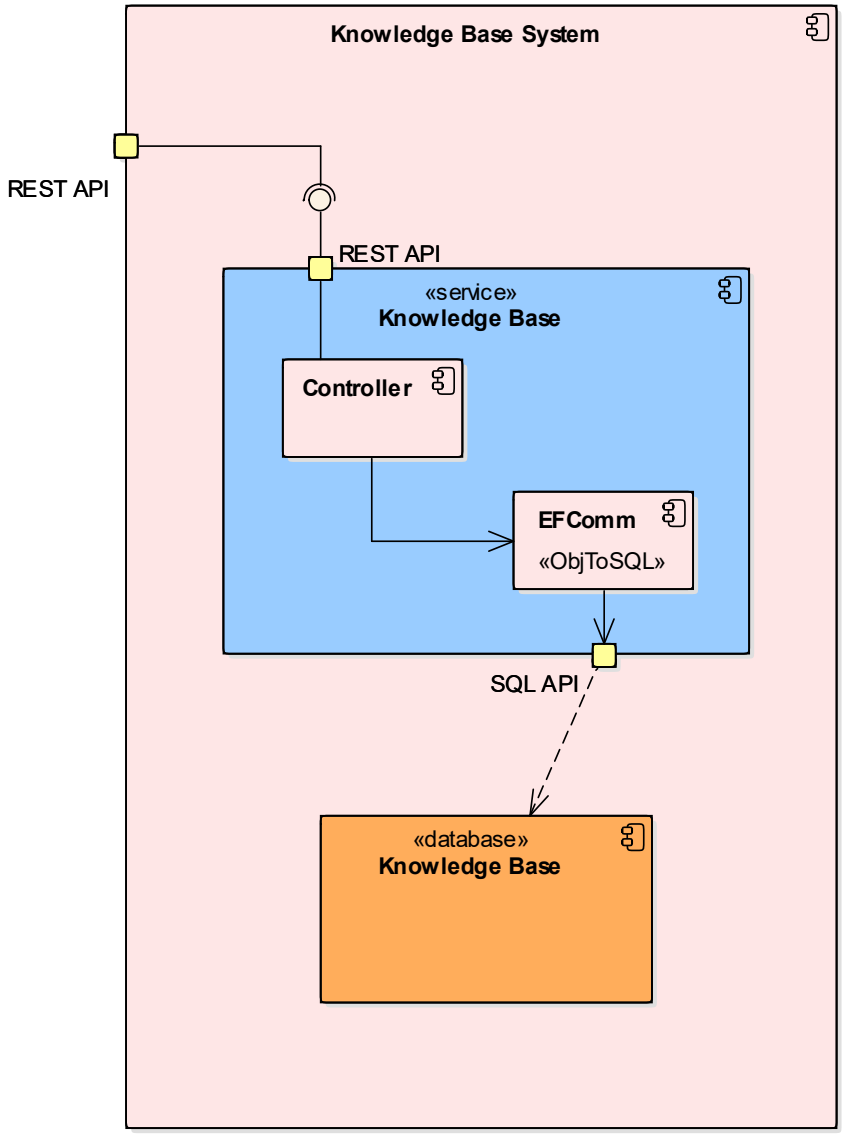

Abbildung 2.2: Ist-Zustand des Systems

Quelle: eigene Abbildung

Es existiert keine GUI, welche das Ergebnis besser darstellen kann. Außerdem besitzt es auch keine Möglichkeit, um die FMD zu verwalten.

Es lässt sich nicht vermeiden, dass eine alternative Lösung benötigt wird, um dieses Problem benutzerfreundlicher zu gestalten und um die FMD zu administrieren. Eine dieser alternativen Lösung wäre eine benutzerfreundlichere Webanwendung zu entwickeln, welche eine Verwaltungsfunktion für die FMD besitzt.

Listing 2.1 Ausschnitt einer REST Model Struktur

```
RestResult[IList[KsError]] {
   result (Array[KsError], optional)
}
KsError {
   id (string, optional),
   mandateId (integer, optional),
   creationDate (string, optional),
   changeDate (string, optional),
   errorCodes (Array[KsErrorCode], optional),
   causes (Array[KsCause], optional),
   translations (Array[KsTranslation], optional)
}
KsErrorCode {
   id (string, optional),
   code (string),
   contextValue (string),
   contextType (string)
}
KsCause {
   id (string, optional),
   creationDate (string, optional),
   changeDate (string, optional),
   translations (Array[KsTranslation], optional),
   hints (Array[KsHint], optional)
}
KsTranslation {
   id (string, optional),
   title (string, optional),
   description (string, optional),
   lcid (integer, optional)
}
KsHint {
   id (string, optional),
   creationDate (string, optional),
   changeDate (string, optional),
   translations (Array[KsTranslation], optional),
   ratings (Array[KsRating], optional)
}
KsRating {
   id (string, optional),
   value (integer, optional),
   creationDate (string, optional),
   lcid (integer, optional),
   comment (string, optional)
}
```

Quelle: Aus einem API-Entwicklung-Tool von BOSCH Server

2.3 Anforderungsanalyse

In diesem Kapitel werden die grundlegende Anforderungen dieses Bachelorprojekts dargestellt. Die Anforderung einer Software besteht aus zwei Anforderungen, welchen als die funktionalen und nicht-funktionalen Anfoderungen bezeichnet werden.

Eine funktionale Anforderung wird nach der Definition aus dem Buch[Bal10] als Funktionalität des Systems festgelegt. Diese Anforderung sollte die folgende Frage beantworten:
Was muss der Entwickler implementieren, damit der Nutzer seine Aufgabe durchführen kann?

Anders als eine funktionale Anforderung steht eine nicht-funktionale Anforderung für die Anwendungseigenschaft und Softwarequalitätsmerkmale, welche für die Nutzung des Systems wichtig sind.

2.3.1 Funktionale Anforderungen

Aus den Unternehmens-Anforderungen lassen sich die funktionalen Anforderungen ableiten.

Nr.	Anforderung-Titel	Priorität
FR_010	Auflistung aller Fehler	Hoch

Tabelle 2.1: 01.Funktionale Anforderung aus der Anforderungsliste der Firma

Die erste funktionale Anforderung lässt sich aus der Tabelle 2.1 festlegen, dass die Anwendung dem Nutzer die Möglichkeit anbietet, um alle existierenden Fehler in der FMD darzustellen.

Nr.	Anforderung-Titel	Priorität
FR_020	Hinzufügen, Entfernen und Bearbeitung von Fehlern	Hoch

Tabelle 2.2: 02.Funktionale Anforderung aus der Anforderungsliste der Firma

Wenn die Fehler aufgelistet werden, soll es nach der zweiten funktionale Anforderung in der Tabelle 2.2 die Möglichkeit für den Nutzer geben, dass er diesen Fehler bearbeiten und entfernen kann. Außerdem bietet die Anwendung dem Nutzer auch die Möglichkeit neue Fehler einzutragen.

Nr.	Anforderung-Titel	Priorität
FR_030	Anzeigen aller Übersetzungen zu einem Fehler	Hoch

Tabelle 2.3: 03.Funktionale Anforderung aus der Anforderungsliste der Firma

In der dritten, funktionalen Anforderung aus der Tabelle 2.3 soll es ermöglicht werden, dass der Nutzer alle verfügbaren Übersetzungen zu einem Fehler einsehen kann. Der Nutzer kann in seiner gewünschten Sprache die Fehler-Details einsehen.

Nr.	Anforderung-Titel	Priorität
FR_040	Hinzufügen, Entfernen und Bearbeitung von Übersetzungen eines Fehlers	Hoch

Tabelle 2.4: 04.Funktionale Anforderung aus der Anforderungsliste der Firma

Laut der funktionalen Anforderung aus der Tabelle 2.4 muss die Anwendung die Möglichkeit anbieten, dass die Übersetzung eines Fehlers bearbeitet bzw. gelöscht werden kann. Ein neuer Übersetzungs-Eintrag für einen Fehler soll für der Nutzer auch möglich sein.

Nr.	Anforderung-Titel	Priorität
FR_050	Anzeigen aller Fehlercodes zu einem Fehler	Hoch

Tabelle 2.5: 05.Funktionale Anforderung aus der Anforderungsliste der Firma

Zu jedem Fehler existieren mehrere Fehlercodes, welche für jeden Fehler spezifiziert sind. Diesen sollen laut der Anforderung in der Tabelle 2.5 dem Nutzer aufgelistet werden, wenn der Nutzer einen Fehler auswählt.

Nr.	Anforderung-Titel	Priorität
FR_060	Hinzufügen, Entfernen und Bearbeitung von Fehlercodes eines Fehlers	Hoch

Tabelle 2.6: 06.Funktionale Anforderung aus der Anforderungsliste der Firma

Die Anwendung soll für der Nutzer nach der Anforderung in der Tabelle 2.6 eine Möglichkeit anbieten, um diese Fehlercodes zu editieren bzw. zu löschen. Außerdem soll es dem Nutzer möglich sein neue Fehlercode-Einträge anzulegen.

Nr.	Anforderung-Titel	Priorität
FR_070	Anzeigen aller Ursachen zu einem Fehler	Hoch

Tabelle 2.7: 07.Funktionale Anforderung aus der Anforderungsliste der Firma

Zu jedem Fehler gibt es mehrere Ursachen, welche zu diesem Fehler führen. Deshalb soll es nach der Anforderung in der Tabelle 2.7 möglich sein, dass der Nutzer diese Ursache betrachten kann, wenn er einen Fehler aussucht.

Nr.	Anforderung-Titel	Priorität
FR_080	Hinzufügen, Entfernen und Bearbeitung von Ursachen eines Fehlers	Hoch

Tabelle 2.8: 08.Funktionale Anforderung aus der Anforderungsliste der Firma

Wie bei Fehler und Fehlercodes soll die Ursache nach der Anforderung der Tabelle 2.8 verarbeitet bzw. gelöscht werden können. Falls eine neue Ursache zum Fehler existiert, soll dem Nutzer die Möglichkeit angeboten werden, diese in die Datenbank (DB) einzutragen.

Nr.	Anforderung-Titel	Priorität
FR_090	Anzeigen aller Übersetzungen zu einer Ursache	Hoch

Tabelle 2.9: 09.Funktionale Anforderung aus der Anforderungsliste der Firma

Es existieren Beschreibungen in mehreren Sprachen zu jeder Ursache. Deshalb soll es nach der Anforderung in der Tabelle 2.9 der Nutzer die Option haben, diese Übersetzungen zu einer Ursache zu betrachten.

Nr.	Anforderung-Titel	Priorität
FR_100	Hinzufügen, Entfernen und Bearbeitung von Übersetzungen einer Ursache	Hoch

Tabelle 2.10: 10.Funktionale Anforderung aus der Anforderungsliste der Firma

Der Nutzer soll diese Übersetzung der Ursache nicht nur betrachten, sondern soll auch nach der Anforderung in der Tabelle 2.10 eine Option haben, diese Übersetzung zu bearbeiten bzw. zu löschen, nachdem er eine Ursache ausgewählt hat.

Nr.	Anforderung-Titel	Priorität
FR_110	Anzeigen aller Lösungen zu einer Ursache	Hoch

Tabelle 2.11: 11.Funktionale Anforderung aus der Anforderungsliste der Firma

Zu jeder Ursache gibt es auch viele Lösungen, welchen diese Ursache beheben können. Diese Lösungen müssen nach der Anforderung in Tabelle 2.11 dem Nutzer aufgelistet werden, wenn er nach einer Ursache sucht.

Nr.	Anforderung-Titel	Priorität
FR_120	Hinzufügen, Entfernen und Bearbeitung von Lösungen einer Ursache	Hoch

Tabelle 2.12: 12.Funktionale Anforderung aus der Anforderungsliste der Firma

Wenn eine Aktualisierung einer Lösung erfoderlich ist, soll nach der Anforderung in Tabelle 2.12 dem Nutzer die Option bereitgestellt werden, diese Lösungen einer ausgewählten Ursache zu ändern bzw. zu löschen. Falls eine neue Lösung existiert, soll dem Nutzer auch die Möglichkeit zum Eintragen dieser Lösung ermöglt werden.

Nr.	Anforderung-Titel	Priorität
FR_130	Anzeigen aller Übersetzungen zu einer Lösung	Hoch

Tabelle 2.13: 13.Funktionale Anforderung aus der Anforderungsliste der Firma

Wie bei der Ursache existiert auch die Beschreibung für Lösung in mehreren Sprachen. Diese Übersetzungen sollen nach der Anforderung in Tabelle 2.13 dergestellt werden, wenn der Nutzer eine Lösung ausgewählt hat.

Nr.	Anforderung-Titel	Priorität
FR_140	Hinzufügen, Entfernen und Bearbeitung von Übersetzungen einer Lösung	Hoch

Tabelle 2.14: 14.Funktionale Anforderung aus der Anforderungsliste der Firma

Diese Übersetzungen, welche nach dem Auswählen einer Lösung dem Nutzers dargestellt werden, soll nach der Anforderung in Tabelle 2.14 veränderbar sein bzw. die Möglichkeit zum Entfernen

bestehen. Falls es eine neue Übersetzung zu dieser Lösung gibt, muss dem Nutzer auch ermöglicht werden diese Übersetzung einzutragen.

Nr.	Anforderung-Titel	Priorität
FR_150	Anzeigen aller Bewertungen zu einer Lösung	Hoch

Tabelle 2.15: 15.Funktionale Anforderung aus der Anforderungsliste der Firma

Jede Lösung besitzt mehrere Bewertungen, welche die Schwierigkeit der Lösung messen. Diesen Bewertungen soll nach der Anforderung in Tabelle 2.15 dem Nutzer vorgelegt werden, wenn er eine Lösung auswählt.

Nr.	Anforderung-Titel	Priorität
FR_160	Hinzufügen, Entfernen und Bearbeitung von Bewertungen einer Lösung	Mittel

Tabelle 2.16: 16.Funktionale Anforderung aus der Anforderungsliste der Firma

Wenn eine Änderung der Bewertung einer Lösung benötigt wird, muss es laut der Anforderung in Tabelle 2.16 für den Nutzer eine Möglichkeit geben, um diese Bewertung zu bearbeiten und zu löschen. Wenn eine neue Bewertung zu einer Lösung vorhanden ist, soll der Nutzer auch die Option haben, diese Bewertung einzutragen.

Nr.	Anforderung-Titel	Priorität
FR_170	Unterstützung Mehrsprachigkeit der Oberfläche	Mittel

Tabelle 2.17: 17.Funktionale Anforderung aus der Anforderungsliste der Firma

Laut der Anforderung in Tabelle 2.17 soll die Webanwendung eine mehrsprachige GUI besitzen, damit der Nutzer die Webanwendung auf eine gewünschte Sprache einstellen kann.

2.3.2 Nicht-funktionale Anforderungen

Im oberen Unterkapitel wurde die funktionale Anforderung aufgelistet. In diesem Kapitel werden die nicht-funktionalen Anforderungen dargestellt, welchen zu diesem Projekt gehören sollen.

Nach der Einigung mit den Mitarbeitern von zwei BOSCH-Entwicklungsabteilungen (PA-ATMO1/SGS43 und INST-INL/ESW31) werden folgende nicht-funktionale Anforderungen nach der internationalen Norm **IOS/IEC 25010**, welche als Leitfaden für Qualitätkrieterien und die Bewertung von Softwareprodukten [Wik16] gilt, festgelegt.

Nr.	Anforderung-Titel	Priorität
NFR_010	Benutzerfreundlichkeit	Hoch

Tabelle 2.18: 01.Nicht-Funktionale Anforderung

Die Verwendung der Webanwendung soll für Nutzer inuitiv sein. Der Nutzer soll nach den Anforderung in Tabelle 2.18 mit wenigem Aufwand und in kurzer Zeit durch die Anwendung

navigieren sowie sie verwenden bzw. die wichtigen Funktionen der Webanwendung ausführen können.

Nr.	Anforderung-Titel	Priorität
NFR_020	Plattformübergreifend	Hoch

Tabelle 2.19: 02.Nicht-Funktionale Anforderung

Nach der Anforderung der Tabelle 2.19 soll die Anwendung unabhängig von der Plattform funktionieren. Deswegen wird hier eine Weblösung vorgelegt, mit der diese Anwendung auf jedem Endgeräte betrieben werden kann. Das bedeutet, dass der Nutzer diese Anwendung auf dem PC bzw. einem mobilen Endgerät, wie Smartphone oder Tablet, betreiben kann.

Nr.	Anforderung-Titel	Priorität
NFR_030	Browser und Betriebsysteme Unabhägigkeit	Hoch

Tabelle 2.20: 03.Nicht-Funktionale Anforderung

Außer der Plattformunabhägigkeit soll diese Anwendung nach der Anforderung in der Tabelle 2.20 in unterschiedlichen Browsern und Betriebsystemen genutzt werden können.

Nr.	Anforderung-Titel	Priorität
NFR_040	Gestaltungsrichtlinie	Hoch

Tabelle 2.21: 04.Nicht-Funktionale Anforderung

Die Gestaltung der Anwendung soll nach der Anforderung in Tabelle 2.21 einheitlich nach vorgegeben Designvorlagen vom Unternehmen geschehen. Außerdem soll die Anwendung für jede Anzeige-Auflösung angepassen sein, damit der Nutzer mit jedem verwendeten Endgerät leicht navigieren kann.

Nr.	Anforderung-Titel	Priorität
NFR_050	Zuverlässigkeit	Hoch

Tabelle 2.22: 05.Nicht-Funktionale Anforderung

Laut der Anforderung in der Tabelle Tabelle 2.22 soll die Anwendung zuverlässig sein. Das bedeutet, dass bei der Nutzung dieser Anwendung keine unwartenden Fehler auftauchen dürfen. Falls ein Fehler aufgetaucht, muss die Anwendung den Fehler sicher abfangen und behandeln können.

Nr.	Anforderung-Titel	Priorität
NFR_060	Perfomance	Hoch

Tabelle 2.23: 06.Nicht-Funktionale Anforderung

Die Leistung, die in der Anforderung in Tabelle Tabelle 2.23 beschrieben ist, bedeutet, dass die Anwendung nach maximal einer Sekunde auf eine Anfrage reagieren und Ergebnisse zurückliefern soll.

Nr.	Anforderung-Titel	Priorität
NFR_070	Sicherheit	Mittel

Tabelle 2.24: 07.Nicht-Funktionale Anforderung

Die Sicherheit als Anforderung in Tabelle 2.24 wird hier aus den folgenden Gründen als letzter Aspekt aufgeführt:

- Es ist **nicht** erfoderlich ein Authentifizierungs-Algorithmus zu erstellen.
- Der Verbindungsaufbau zwischen WebUI und Server erfolgt durch eine öffentliche API. Deswegen wird der Service- und Datenbankinhalt nach außen unsicherbar sein.
- Außerdem wird diese Webanwendung innerhalb des Unternehmensnetzwerks betrieben. Aus diesem Grund ist ein Teil der Sicherheit dieser Anwendung abgedeckt.

2.4 Anwendungsfallanalyse

In der Abbildung 2.3 werden die Anwendungsfälle vom Nutzer als Adminstrator oder als Operator dargestellt. Es gibt nur zwei verfügbare Rollen, die dieses System bedient. Die Berechtigungen zwischen den beiden Rollen sind gleichwertig, weil keine spezifische Anwendungsfälle für bestimmte Nutzer zugewiesen werden und es keine Authorisierung im System notwendig ist.

2.5 Programmarchitektur

Die Architektur der Webanwendung basiert auf einer „Client-Server“-Architektur. Das bedeutet, dass die Webanwendung auf dem Server läuft und der Client nur die Ergebnisse erhält.

Die Austausch der Anfrage und das zurückgelieferte Ergebnis geschieht über ein Netzwerk. Die Webanwendung wird in drei Schichten unterteilt, welche als Datenmodell, Web-Server und WebUI von der Webanwendung bezeichnet sind. Bestimmte Aufgaben werden von bestimmen Schichten übernommen und eine Veränderung einer der Komponenten wirkt sich nur minimal auf andere aus.

In der Abbildung 2.4 werden die Schichten der Anwendung dargestellt. Hier ist der Webbrowser die Präsentationschicht. Der Webbrowser wird die Eingabe der Nutzer übernehmen und als Anfrage über eine REST-API an den Server weiterleiten. Umgekehrt wird das Ergebnis vom Server auch über diese API geschehen.

Die Web-Server-Schicht beinhaltet die Webanwendung und die KBS. Die Client-Anfragen werden in der Webanwendung bearbeitet und diese Anfrage wird in Form von einer JavaScript Object Notation (JSON)-Datei an die REST API vom KBS weitergeleitet. Diese REST API wandelt die JSON-Datei in ein C#-Objekt um bzw. bei der Rückmeldung vom Service in eine JSON-Datei. Dieses C#-Objekt wird an die Komponente „Controller“ weitergeleitet. Diese erstellt aus dem Objekt die Structured Query Language (SQL) Anfrage, welche an die Komponente „EFComm“

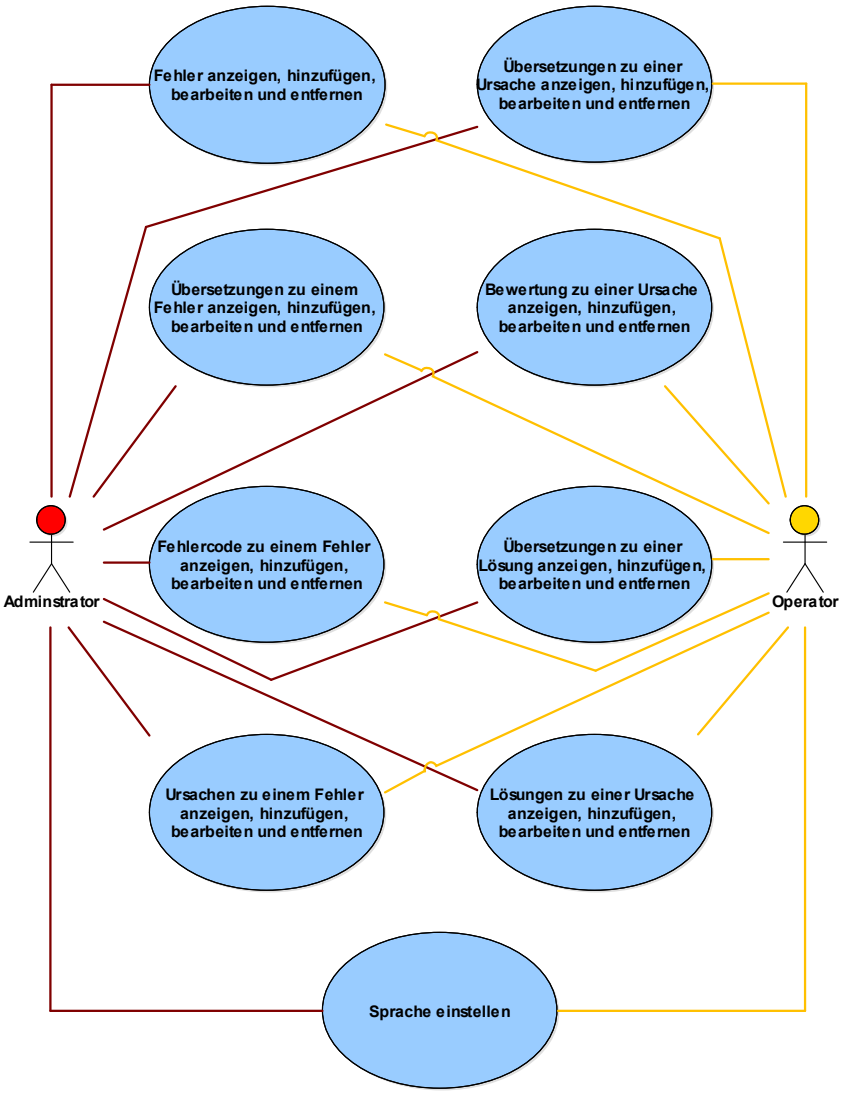

Abbildung 2.3: Anwendungsfälle für Adminstrator und Operator
Quelle: eigene Abbildung

übergeben wird. Die „EFComm“-Komponente kommuniziert über eine SQL API mit der FMD, welche sich auf dem MSSQL DB-Server befindet.

Die FMD reagiert auf Anfragen wie Daten abrufen, existierende Daten bearbeiten bzw. entfernen oder neue Daten einfügen, unterschiedlich und liefert die Rückmeldung an den KBS zurück. Dieser spricht die Webanwendung an und das erhaltende Ergebnis wird schließlich dem Nutzer im Webbrowser präsentiert.

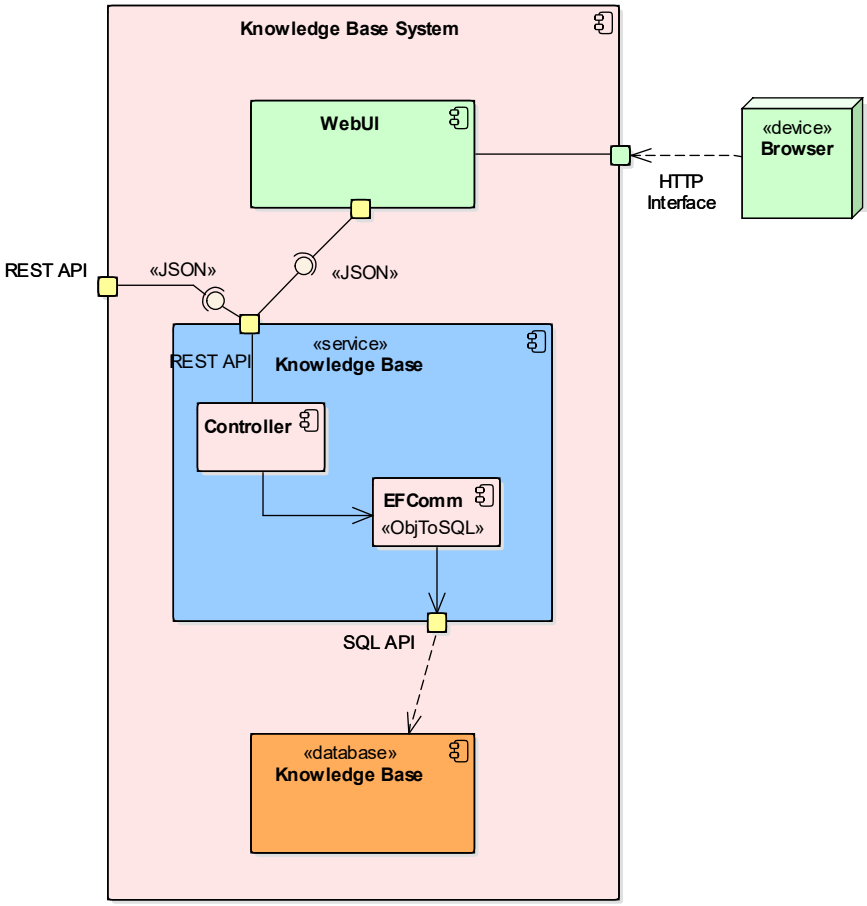

Abbildung 2.4: Architektur des Programmes

Quelle: eigene Abbildung

3 Entwurf und Umsetzung des Weboberflächen-Konzepts

Aus den gesammelten Anfoderungen vom Kapitel 02 wird es hier in diesem Kapitel um den Entwurf und der Umsetzung der WebUI gehen. Zu Beginn werden die Aufgaben der Webanwendung analysiert, welche die nötigen Funktionen vorgeben. Im weiteren Verlauf wird über die Gestaltung der WebUI gesprochen und wie man die Funktionen intuitiv darstellt. Zum Schluss wird erklärt, wie die WebUI technisch umgesetzt wird.

3.1 Analyse der Aufgabe der Webanwendung

In Anbetracht der vorgegebenden, funktionalen Anforderung soll die Webanwendung eine Administrationsfunktion auf der FMD ermöglichen. Das bedeutet, dass die WebUI dieser Webanwendung die Verwaltungsmöglichkeit für den Nutzer anbietet. Der Nutzer soll alle verfügbaren Fehler betrachten können bzw. diese im Detail bearbeiten und entfernen können. Außerdem soll die Webanwendung dem Nutzer die Option bereitstellen, dass er neues Fehler-Wissen in der FMD einfügen kann.

Um diesen Funktionsumfang für den Nutzer darzustellen, haben der zuständige Betreuer und ich uns auf eine Lösung geeinigt, welche in dieser Abbildung 3.1 wiedergeben wird.

Auf der Startseite der Webanwendung werden alle verfügbaren Fehler in einer Liste Abbildung 3.1 **(Nr. 6)** angezeigt. Eine Filterfunktion Abbildung 3.1 **(Nr. 7)** hilft dem Nutzer diese Darstellung der Fehlern mit speziellen Eigenschaften eines Fehlercode einzuschränken.

Auf der linken Seite der Abbildung existieren die zwei Hauptfunktionen der Webanwendung, welche mit der Nummer **1** in Abbildung 3.1 und Nummer **2** in Abbildung 3.1 markiert sind. Die Funktion Abbildung 3.1 **(Nr. 1)** ist die Darstellung der Startseite der Webanwendung und hat die Aufgabe alle Fehlern inkl. optionale Nutzerfilter darzustellen. Die Funktion Abbildung 3.1 **(Nr. 2)** ermöglicht dem Nutzer die Erstellung von neuen Fehler inkl. aller verfügbaren Eigenschaften. Mit Klick auf die Settings-Funktion Abbildung 3.1 **(Nr. 3)** kann der Nutzer allgemeine Einstellungen, wie die Webanwendung-Sprache, vornehmen. Hinter der Imprint-Funktion versteckt sich das Impressum der Webanwendung.

In der Startseite der Webanwendung werden nicht nur die Fehlern als Liste dargestellt, sondern sie bietet dem Nutzer ebenso an, durch Bestätigung auf den hervorgehobenden Text Abbildung 3.1 **(Nr. 8)** die Details vom jeweiligen Fehler einzusehen bzw. zu bearbeiten. Außerdem kann der Nutzer diesen Fehler durch die Bestätigung auf das Lösch-Symbol Abbildung 3.1 **(Nr. 9)** entfernen. Des Weiteren existert eine Server-Status-Anzeige Abbildung 3.1 **(Nr. 5)**, welche immer den aktuellen Server-Status anzeigt.

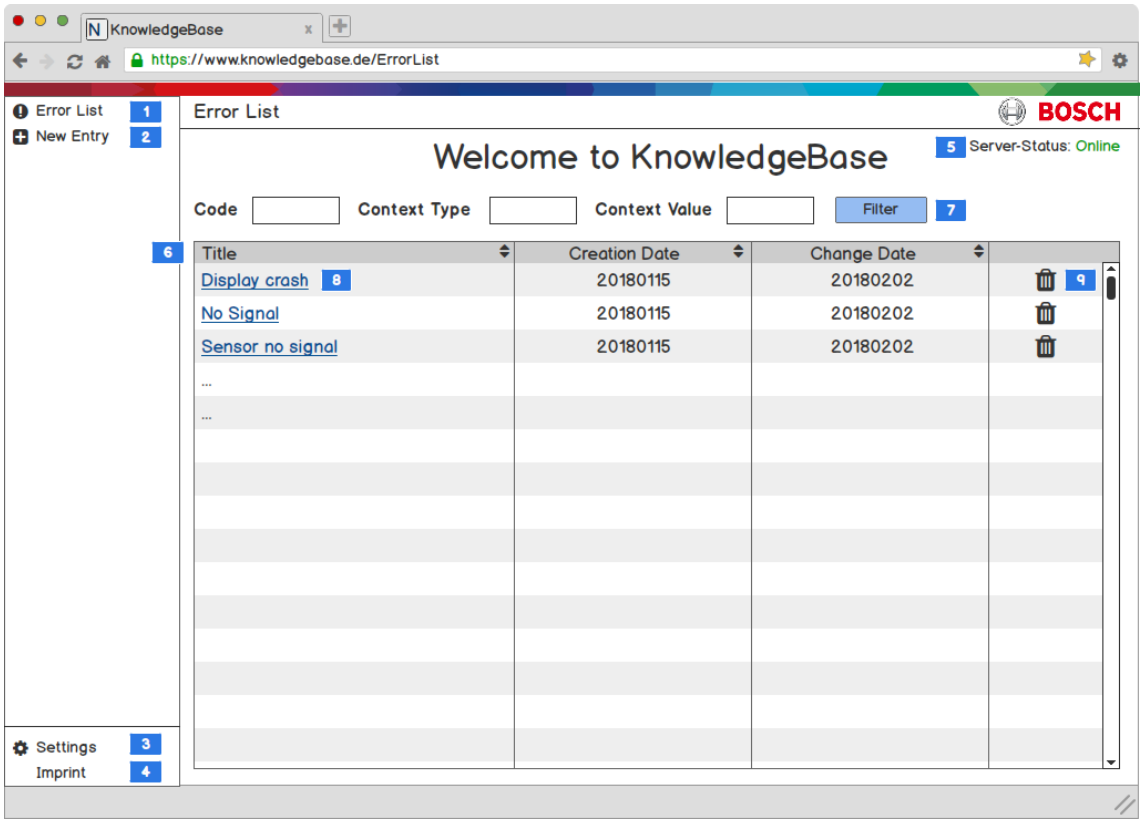

Abbildung 3.1: MockUp von der Startseite der Webanwendung
Quelle: eigene Abbildung

Wenn die zweite Hauptfunktion Abbildung 3.1 **(Nr. 2)** bestätigt wird, navigiert die Webanwendung den Nutzer weiter in die Erstellungs-GUI, welche in folgenden Abbildung 3.2 dargestellt wird.

Der Nutzer kann hier einen neuen Fehler anlegen. Die Eigenschaften dieses Fehlers, welcher aus Fehlercodes, Ursachen, Übersetzungen besteht, kann durch Bestätigung der jeweiligen Add-Buttons wie beispielsweise Abbildung 3.2 **(Nr. 1)** eingefügten werden. Die erstellten Eigenschaften können wieder durch die Bestätigung auf die hervorgehobenen Texte betrachtet und durch Bestätigung des Stift-Symbols Abbildung 3.2 **(Nr. 2)** bearbeitet bzw. durch das Lösch-Symbol entfernt werden.

Ein weiterer, wichtiger Aspekt dieser Webanwendung ist, dass sie in der ersten Version keine Protokollierung und Login-Mechanismen besitzen soll. Eine Implementierung von Login-Mechanismen ist für eine zukünftige Version geplant. Die Sicherheit wird dennoch gewahrt, da die Seite nur im Produktionsnetz erreichbar ist.

3.2 Gestaltung der Benutzeroberfläche

Im vorherigen Kapitel wurde ein Überblick über die Funktionalität der Webanwendung anhand der WebUI gegeben. In diesem Kapitel werden die Designvorgaben, die von der ersten Konzeption bis zur fertigen Webanwendung benötigt werden, genannt.

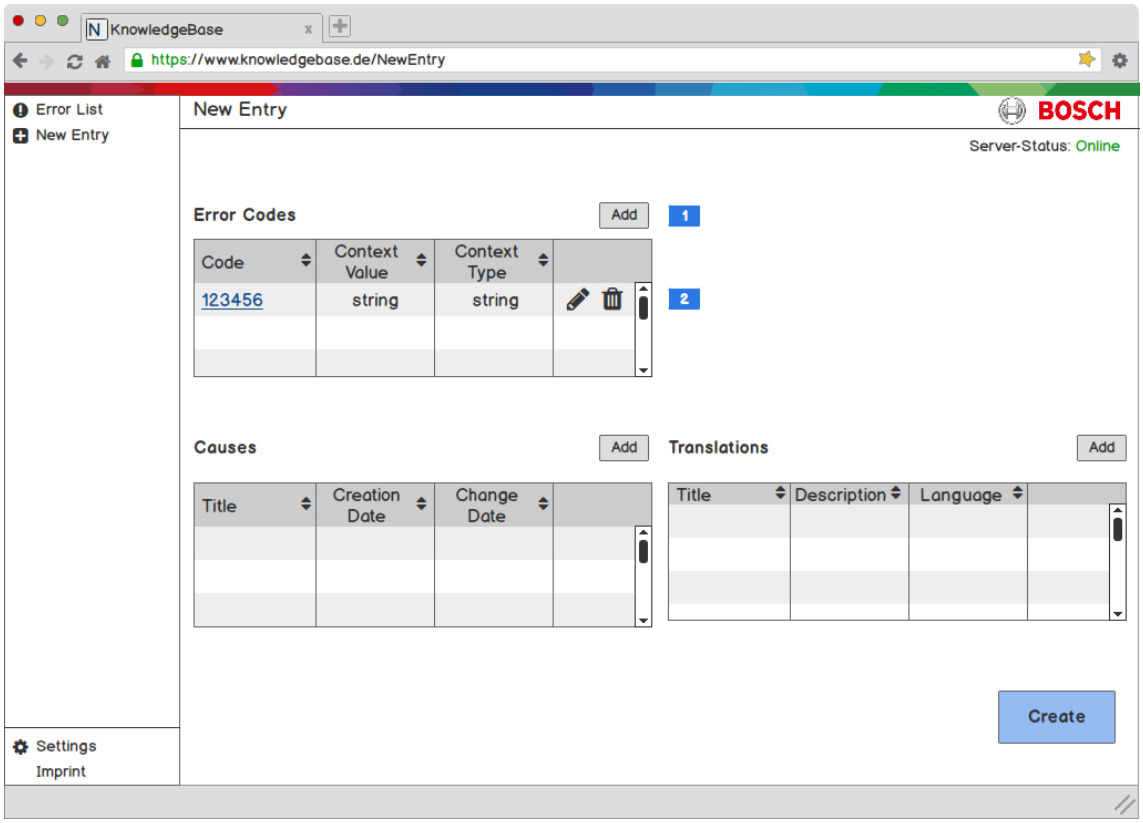

Abbildung 3.2: MockUp von der Fehler-Erstellung GUI der Webanwendung
Quelle: eigene Abbildung

Im Gespräch mit der dafür zuständigen Person wurde sich darauf geeinigt, dass bereits eine einheitliche und festgelegte Designstruktur der Firma existiert, welche angewendet werden muss. Der Aufbau dieser Struktur ermöglicht das einzigartige Image des Unternehmens für alle seiner entwickelten Webanwendung nach außen zu tragen. Diese werden nachfolgend Abbildung 3.3 dargestellt.

Diese Struktur kann in vier grundlegenden Teilstrukturen zerlegt werden. Der erste Teil dieser Struktur besteht aus einer spezifischen Grafik, welche auch als **Supergraphic** (Abbildung 3.4) bezeichnet wird. Diese Supergrafik ist eine wichtige, visuelle Identität des Unternehmens und stammt vom Markenversprechen, was hilft sich von anderen Unternehmen zu unterscheiden. Sie wird als Gestaltungselement als Symbol/Logo oder Hintergrundgrafik verwendet.

Der zweite Teil vom Design besteht aus einer Sidebar, welche wiederum aus zwei Unterteile zusammengesetzt ist. Das erste Unterteil der Sidebar beinhaltet alle verfügbaren Hauptoptionen der Webanwendung. Im zweiten Unterteil der Sidebar sind die grundlegenden Optionen definiert, beispielsweise **Einstellungen** und **Impressum**.

Der dritte Teil dieses Designs beinhaltet ein **Seitentitel** und das Unternehmenslogo. Wie bei der Supergrafik ist dieses Logo (Abbildung 3.5), welches auch als **Brandmark** bezeichnet wird, ein spezifisches Design des Unternehmens. Dieses Brandmark setzt sich aus dem Ankersymbol und dem Logotype zusammen. Außerdem ist es eine Pflicht diese Brandmark auf allen digitalen Medien, die auf bestimmten Veranstaltungen präsentiert werden, zu verwenden.

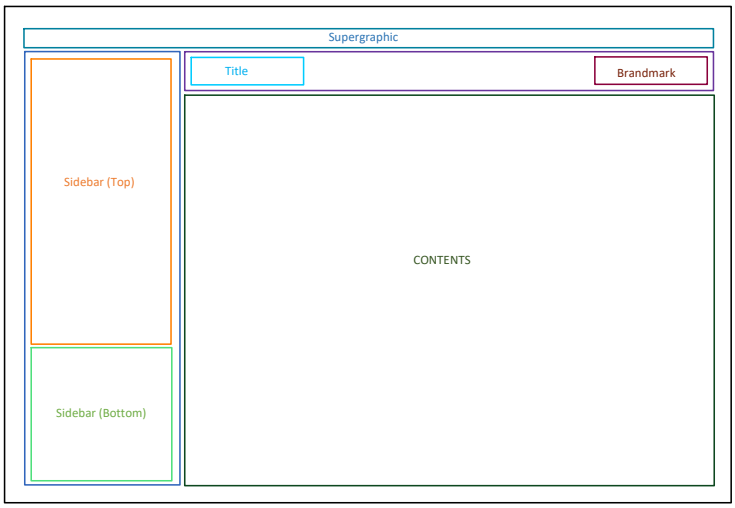

Abbildung 3.3: Einheitliche, festgelegte Designstruktur der Webanwendung
Quelle: eigene Abbildung

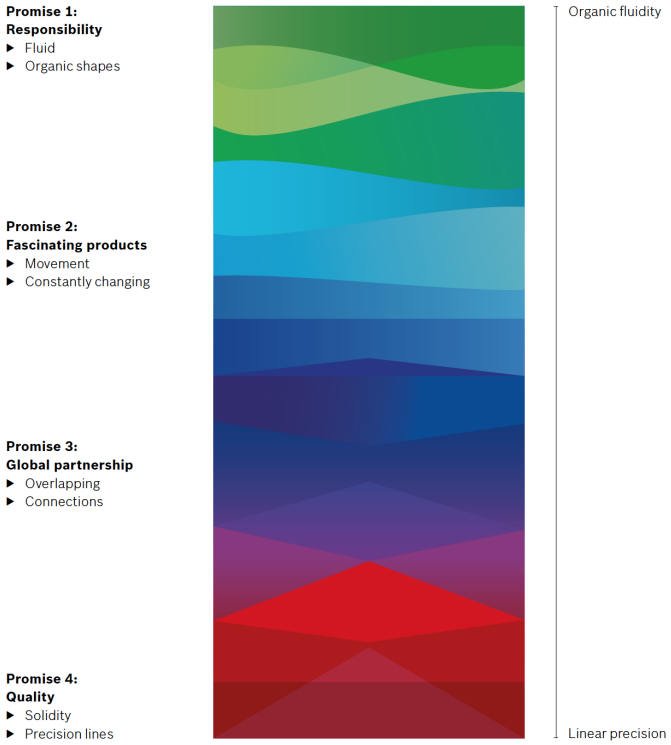

Abbildung 3.4: Supergrafik des Unternehmens
Quelle: Abbildung vom Dokument der zuständigen Designabteilung

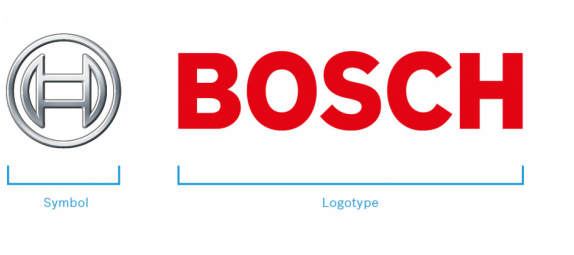

Abbildung 3.5: Logo des Unternehmens
Quelle: Abbildung von Dokument der zuständigen Designabteilung

Der letzte Teil dieses Designs ist der **Seiteninhalt**. Innerhalb dieses Teils befindet sich die Darstellung aller verfügbaren Hauptoptionen der Webanwendung. Der Inhalt wird für jede Hautoption unterschiedlich sein und an diese angepasst.

Ein weiterer, wichtiger Aspekt bei der Gestaltung der WebUI ist das oben genannte Kriterium der Benutzerfreundlichkeit. Die WebUI soll möglichst intuitiv und einfach gestaltet sein. Dieses Kriterium kann durch verständliche Button-Beschriftungen, eine klare Popup-Fenster-Abfrage, Hinweise für Eingabefelder und selbsterklärende Symbole erreicht werden.

Außerdem wird durch die angewendete Designvorlage des Unternehmens, welche hier verwendet wird und „Look and Feel“ genannt wird, der einzigartige Eindruck der Webanwendung erzeugt. Die Designvorlage wird in **Abschnitt 4.4** kurz erwähnt.

Die Verwendung der Webanwendung soll nicht nur auf dem Rechner, sondern auch auf mobilen Geräten ermöglicht werden. Deshalb wird die WebUI den unterschiedlichen Auflösung angepasst, damit eine optimale Darstellung der Inhalte erzielt wird.

Ein beispielhafte Darstellung der Webanwendung in der normalen Ansicht, welche mit dem zuständigen Mitarbeiter abgesprochen ist, wird in dieser Abbildung 3.6 dargestellt.

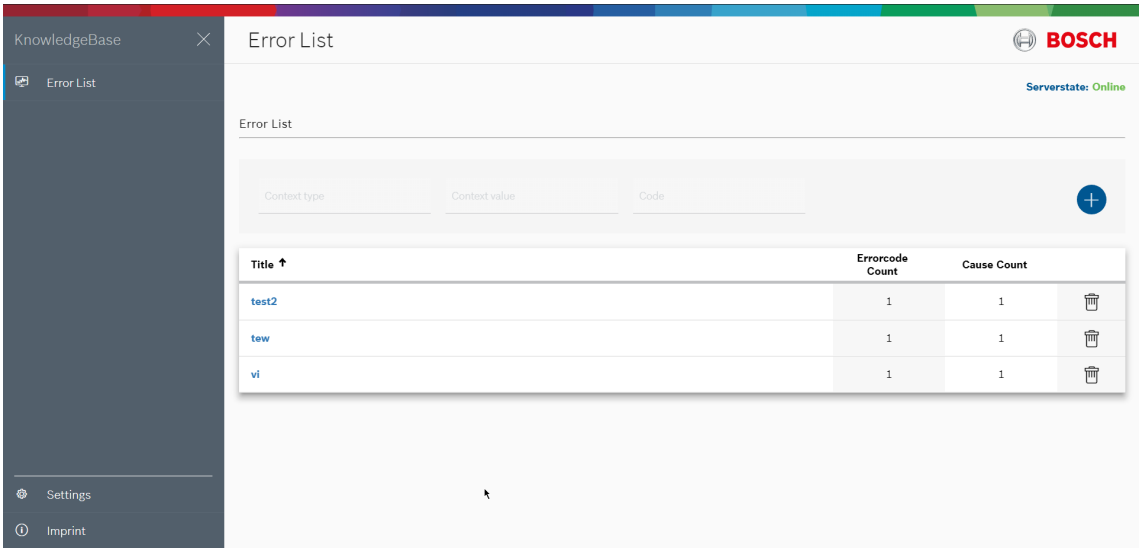

Abbildung 3.6: Darstellung der Hauptseite der Webanwendung (Fehler Liste)
Quelle: Abbildung aus der eigenen Webanwendung

3.3 Umsetzung des Weblayouts

Um die obengenannte Gestaltung umzusetzen, bietet das Unternehmen bereits eine grundlegende Vorlage, welche die Basis-Design-Komponenten beinhaltet. Diese Basis-Komponenten werden in einem Hypertext Markup Language (HTML) Code-Ausschnitt dargestellt (siehe Listing 3.1).

Listing 3.1 Grundsätzliche Aufbau-Vorlage der Webanwendung

```
<bci-content>
   <bci-sidebar [sidebarLinks]="sidebarLinks" [applicationTitle]="title">
      <bci-sidebar-footer [sidebarFooterLinks]="sidebarFooterLinks"></bci-sidebar-footer>

      <bci-header [header]="topLineTitle" logoLink="http://www.bosch.com">
      </bci-header>

      <div fxFlexFill fxFlex="1 1 auto" style="overflow:auto;">
         <router-outlet>
         </router-outlet>
      </div>
   </bci-sidebar>
</bci-content>
```

Quelle: Code-Ausschnitt aus der vorgegebenen Designvorlage von BOSCH

Im HTML-Code-Ausschnitt wird der grundlegende Aufbau der WebUI gegliedert. Innerhalb der HTML-tags *<bci-content> ... </bci-content>* liegen die Basis-Komponenten der WebUI.

Diese Komponente besteht aus einer Sidebar, welche mit dem HTML-tag *<bci-sidebar> ... </bci-sidebar>* gekenntzeichnet ist und mit den Attributen **„sidebarLinks“** inkl. **„applicationTitle“** erweitert wird. Das Attribut **„sidebarLinks“** steht für die Verlinkungstitel von einzelnen verfügbaren Optionen der Webanwendung im oberen Bereich der Sidebar. Das **„applicationTitle“**-Attribut steht für die Bezeichnung der Webanwendung.

Außerdem wird diese Sidebar-Komponente aus den SidebarFooter-tags *<bci-sidebar-footer> ... </bci-sidebar-footer>*, Header-tags *<bci-header> ... </bci-header>* und den div-tags * ... * zusammengesetzt.

Der **SidebarFooter-tag** definiert den Verlinkungstitel der verfügbaren Optionen vom unteren Bereich der Sidebar. Der Header-tag übernimmt die Anzeige des Titels in der oberen Linie inkl. der Erscheinung vom Unternehmenslogo.

Anders als andere tags besitzt der **div-tag** bei Angular ein besonderes Attribut, welches bei Änderung der Fenster-Auflösung reagiert und den Inhalt dieses tags anpasst. Solch ein Verhalten vom Style wird entweder von der Angular-Material-Bibliothek oder der selbstentwickelnden Style-Bibliothek des Unternehmens importiert, welches im folgenden Code-Ausschnitt der Simple Cascading Style Sheet (SCSS) (siehe Listing 3.2) dargestellt ist.

Listing 3.2 Style-Importierung der Webanwendung

```
@import "~@bci/ng-material-theme/sass/theme";
@import "~@bci/generic-web-styles/sass/bci-styles";
```

Quelle: Code-Ausschnitt aus der globalen Style-Definition von BOSCH

Zuletzt befindet sich innerhalb dieses **div-tags** noch ein anderer tag *<router-outlet> ... </router-outlet>*. Dieser **router-outlet-tag** übernimmt die Navigationaufgabe vom Vernetzungsrouter der

einzelnen Darstellungskomponenten, welcher in einem Routersmodule fest definiert ist. Das heißt, dass innerhalb diesen tags unteschiedliche Darstellungen gezeigt werden bzw. zwischen diesen Darstellungen navigiert werden kann.

Um diese vorgebende Designvorlage verwenden zu können, muss man durch viele Konfigurationschritte gehen. Außerdem muss die Design-Bibliothek korrekt eingebunden werden.

Nachdem die Konfigurationschritte der Designvorlage erfolgsreich abgeschlossen sind, wird die Darstellung innerhalb der oben genannten **router-outlet-tags** eingepflegt.

Das Einpflegen der Darstellung wird im folgenden Code-Ausschnitt (siehe Listing 3.3) aus dem Bachelorprojekt dargestellt.

Listing 3.3 Darstellung der Fehlerliste der Webanwendung

```
<div class="contentView">
  <!-- show command bar with filter and add -->
  <div class="components">
    <bci-error-command-bar #errorCommandBar
        (_submitAddedDataEvent)="errorList.addedError($event)"
        (_filterButtonEvent)="errorList.showErrorList($event)"></bci-error-command-bar>
  </div>

  <!-- show error list -->
  <div class="components">
    <bci-error-list #errorList></bci-error-list>
  </div>
</div>
```

Quelle: Code-Ausschnitt der Fehlerliste HTML aus dem eigenen Projekt

Der erste **div-tag** wird mit einer definierten Style-Klasse verbunden, welcher in Listing 3.4 dargestellt ist. Durch diese Definition **class="contentView"** werden die Elemente innerhalb des *<div class="contentView» ... * einen Abstand von **20 Pixel** links und rechts zum Rand haben. Die Elemente, die sich innerhalb dieses **div-tags** befinden, sind wiederum in zwei weitere **div-tags** unterteilt, welche jeweils eine Kind-View-Komponente beinhalten und mit der Style-Klasse **class="components"** gebunden sind. Dieses Style definiert den Abstand von **10 Pixel** zwischen jeder Kind-Komponente, um eine klare Darstellung zu erzielen.

Listing 3.4 Definierte Style-Klasse für die Fehlerliste

```
.contentView {
  padding: 0 20px;
}

.components {
  padding: 10px 0px;
}
```

Quelle: Code-Ausschnitt der Fehlerliste SCSS aus dem eigenen Projekt

Die erste Kind-View-Komponente ist eine Befehlsleiste, welche die sogenannte **„command bar“** ist und mit dem Tag *<bci-error-command-bar ...> ... </bci-error-command-bar>* beginnt und endet. Diese besitzt eine Fehler-Filterfunktion, um die Fehlerliste mit bestimmten Aspekten zu beschränken. Außerdem steht eine Hinzufügefunktion zur Verfügung, um die neuen Fehler anzulegen.

Die zweite Kind-View-Komponente ist die Fehlerliste selbst und mit dem Tag *<bci-error-list ...> ... </bci-error-list>* beginnt und endet sie. Diese stellt die Fehler in einer Daten-Tabelle dar. Die genaue Erklärungen der einzelnen View-Komponenten werden im **Kapitel 5** gegeben.

4 Übersicht über die Webentwicklungswerkzeuge

Im oberen Kapitel ging es kurz über die angewendete Webtechnologie für mein Bachelorprojekt. In diesem Kapitel wird genauer auf die angewendete Webtechnologie **Angular (Abschnitt 4.1)**, der Transportdatei **JSON (Abschnitt 4.2)**, der vorgegebenen **REST-Schnittstelle (Abschnitt 4.3)** vom Service, der vorgebenen **Designvorgabe (Abschnitt 4.4)** vom Unternehmen und zuletzt kurz auf die verwendete **Entwicklungsumgebung (Abschnitt 4.5)** eingegangen.

4.1 Angular

Angular ist ein JavaScript-Framework, welches darauf spezialisiert ist, die Webseite mit einer einzelnen Seite zu erstellen. Die Inhalte der Unterseiten werden nebenbei dynamisch in der Seite ausgeführt. Um diese zu ermöglichen, wird von Angular ein Router bereitgestellt, welcher eine bestimmte Seite in Abhängigkeit von Uniform Resource Locator (URL) lädt und auf der Oberfläche ausgibt.

Weiterhin bietet Angular für asynchrone Server Abfragen die sogenannte „Observables“ an, welche asynchrone Ergebnisse liefern. Somit muss die Seite nicht auf die gesamte Antwort des Servers warten, sondern die vollständige Darstellung der Seite kann bereits laden kann und der Inhalt wird nach und nach dynamisch eingebunden.

Die Daten werden bei Angular mit der Oberfläche verbunden, was bedeutet, dass der Entwickler sich entscheiden kann, in welche Richtung die Daten fließen sollen. Einerseits können die Daten vom Modell zur Oberfläche gesendet werden, damit die Oberfläche sich aktualisiert, wenn die gebundenen Daten sich ändern, anderseits kann auch eine Änderung auf der Oberfläche eine Auswirkung auf dem Datenmodell haben. Der Datenfluss kann natürlich auch nach Wunsch in beide Richtungen erfolgen. Die Daten werden aus dem Modell geladen und bei den Änderungen vom Nutzer wird das Datenmodell aktualisiert.

Anstatt der Verwendung von **Twitter-Bootstrap** wird in dieses Projekt **„Angular Material“** verwendet. Solche Materialien können durch diese Referenz [Goo10d] eingesehen werden.

4.1.1 Übersicht

Das Angular Framework wird von Google seit 2010 und freien Mitarbeitern entwickelt bzw. unterstützt. Die Lizenz, welche der Code von Angular unterliegt, ist die **MIT-Lizenz**. Dieser Code ist zur freien Verwendung verfügbar.

Außerdem bietet Angular auch wie andere Frameworks viele Charakterisierungen und spezielle Funktionen. Die wesentlichen, verwendeten Funktionen und Module, welche in diesem Projekt Anwendung finden, werden in folgenden Unterkapiteln schrittweise erklärt.

Um Angular Projekt zu entwickeln bzw. auszuführen können, wird eine JavaScript-Laufzeitumgebung benötigt. Die Empfehlung der Entwickler ist die Verwendung von **Node.js** [Goo10f].

4.1.2 Node.js

Eine JavaScript-Laufzeitumgebung ist **Node.js**, welche auf der Erstellung von Netzwerkanwendungen spezialisiert ist [Joy18]. Die Node.js Entwicklung läuft über das GitHub Repository von Node und wird von **Technical Steering Committee (TSC)** überwacht und verwaltet.

Um die Nutzung von Node und den verfügbaren Modulen zu ermöglichen, wird zu Node der **Node Package Manager (NPM)** mitgeliefert und installiert. Durch die Kommandozeile von NPM können neue Pakete hinzugefügt werden, welche in einem Projekt benötigt werden.

NPM enthält mehr als 600.000 JavaScript Pakete, die zum Herunterladen zur Verfügung stehen [npm14]. Die Entwickler können über die Webseite von NPM nach benötigten Paketen suchen und in ihren Projekten integrieren. Auch viele Angular-Komponenten können über NPM, wie **Angular Command Line Interface (CLI)** oder **Angular Materialen**, installiert werden.

4.1.3 Angular CLI

Für viele Aspekte eines Projekts kann die Angular Kommandozeile genutzt werden. Die Kommandozeile ist in erster Linie für die Kompilierung der Anwendung und das Hosten von Webservices zuständig. Um das Hosten durchzuführen, muss der Befehl **„ng serve“** in der Kommandozeile eingegeben werden. Dadurch wird der Webserver auf einem lokalen Port, welcher Standardport 4200 ist, ausgeführt. Nach dem Abschließen der Kompilierung kann die Webanwendung über einem Webbrowser auf URL **„http://localhost:4200/“** aufgerufen werden.

Außerdem besitzt die Angular CLI weitere Funktionen, welchen die Entwicklung mit Angular vereinfacht. Beispielsweise kann mit dem Kommando **„ng new“** ein neues Projekt, welcher bereits alle wichtigsten Komponenten beinhalten, erstellt werden und mit dem Kommando **„ng generate“** können neue Klassen, Komponenten, Services usw. generiert werden [Goo10a].

4.1.4 TypeScript

Die verwendete Programmiersprache für die Entwicklung von Angular ist **TypeScript**, welche eine von Microsoft entwickelte Programmiersprache ist. Diese basiert auf JavaScript und ergänzt diese um einige Komponenten. Bei der Kompilierung wird TypeScript Code wieder in JavaScript Code umgewandelt. Das bedeutet dann auch, dass TypeScript mit alten JavaScript Versionen rückwärtskompatibel ist.[Tut18]

4.1.5 Komponenten

Die erstandene Seite von Angular setzt sich aus verschiedenen Komponenten zusammen. Eine Komponente besteht aus einem HTML-Teil, einem TypeScript-Teil und einem oder mehreren Cascading Style Sheet (CSS)-Teilen.

Ein neuer HTML-Container wird bei der Erstellung einer neuen Komponente definiert. Die Komponente kann in eine HTML-Datei über diesen Container eingebunden werden. Mithilfe von Datenbindungen (siehe Abschnitt 4.1.6) können der Komponente Daten übergeben werden. Außerdem werden die Funktionen, welche von der Komponente gesendeten Events sind, bereitgestellt.

Der verwendete Backend-Code in dieser Komponente befindet sich im TypeScript-Teil der Komponente. Innerhalb vom Konstruktor der Komponente können notwendige Services geladen werden, welchen vom Root-Modul in den jeweiligen Komponenten eingebunden werden.

Das zentrale Root-Modul hat die Aufgabe alle Komponenten einer Webanwendung zu registrieren und verwalten. Diese Komponenten werden durch das Router-Modul in die Seiten geladen.

4.1.6 Datenbindungen

Die Kommunikation von Daten zwischen den Komponenten wird bei der Entwicklung in Angular über die Datenbindungen gestaltet. Es gibt unterschiedliche Arten von Bindungen, welche im folgenden Listing 4.1 dargestellt sind.

Listing 4.1 Datenbindung Codebeispiel

```
// Einweg-Datenbindung mit festen Wert
   1. <input [placeholder]="'value'" />

// Einweg-Datenbindung mit string variable 'name'
   2. <input [placeholder]="name" />

// Zweiweg-Datenbindung mit string variable 'value'
   3. <input [(ngModul)]="value" />

/* Bei dem Klick-Event wird die Funktion 'buttonClicked' aufgelöst,
 * welche Event-Objekt als Übergabeparameter mitgeliefert ist
 */
   4. <button (click)="buttonClicked($event)"></button>
```

Quelle: eigenes Codebeispiel

In diesem Codebeispiel sind zwei Einweg-Datenbindungen und eine Zweiweg-Datenbindung abgebildet. Die Funktionalität von den Datenbindungen haben gewissermaßen Ähnlichkeiten. Bei einer Komponente können die Datenbindungen in zwei Richtungen erfolgen. Sie werden entweder als Input oder Output implementieren.

Um die gebundenen Daten als Input zu übergeben und in der Komponente verwenden zu können, werden die Variablen mit **„[...]“** deklariert, wie im **Codebeispiel 1 und 2** abbildet. Sie können entweder als Variable **(wie im Codebeispiel 2)** oder als ein fester Wert **(wie im Codebeispiel 1)** übergeben werden. Dies wären dann die Einweg-Datenbindungen.

Um eine Zweiweg-Datenbindung zu ermöglichen, werden zu den eckigen Klammern zusätzlich die runden Klammern **(wie im Codebeispiel 3)** ergänzt. Dabei wird die Änderung in der Komponente auch zu einer Änderung im Datenmodel an der gebundenen Variable. Das bedeutet, dass beispielsweise die Variable **"value"** einen String **"MusterString"** als Eintrag bekommt. So wird zu Beginn auch in dem entsprechenden Eingabefeld dieser Eintrag **"MusterString"** angezeigt. Falls eine Änderung von **„value“** in diesem Eingabefeld von Nutzer erfolgt, wird auch direkt die Variable **„value“** geändert und die Eingabe vom Nutzer kann später weiterverwendet werden.

Anders als beim Input handelt es sich beim Output oft um Events, welche vom Nutzer erzeugt werden können. Es gibt viele Event-Arten, welche in der Regel in fast jeder Komponente bereits erhalten oder von den Entwicklern speziell auf die Komponente abgestimmte Events sind. Wie im **Codebeispiel 3** können Events mit **„(...)“** an bestimmte Funktionen gebunden werden. Wenn die Komponente das Event sendet, wird die zugeordnete Funktion ausgeführt [Goo10h].

4.1.7 Module

Die Module werden als Verwaltung für alle Komponenten genutzt. Es können verschiedene Module für bestimmte Bereiche erstellt werden, welchen für diesen Bereich wesentliche Komponenten administrieren. Um die komplette Webanwendung zu verwalten, existiert ein zentrales Modul, welches auch als Root-Modul bezeichnet wird.

Alle definierten, verwendeten Komponenten und Module in der Webanwendung müssen in diesem Root-Modul deklariert werden. Es gibt keine Ausnahme, ob die Komponenten direkt, oder über eingebundene Module importiert werden.

Dieses Root-Modul legt für alle Deklarationen, Importe, Provider ein Array vor, welches bei der Moduldefinition angegeben wird. Ein Beispiel zu diesem Root-Modul wird im folgenden Listing 4.2 der Webanwendung dargestellt.

Zu Beginn werden in diesem Modul die Komponenten der Webanwendung deklariert, anschließend die Importe. Die Services werden in Providers integriert und die Basiskomponente wird zuletzt festgelegt.

Falls ein externes Modul in der Webanwendung verwendet wird, muss es zuerst mithilfe von NPM installiert und anschließend im Root-Modul importiert bzw. im Import-Array vom Root-Modul eingefügt werden. Dadurch kann dieses Modul, welches durch das Root-Modul importiert wurde, für alle Komponenten der Webanwendung verwendet werden, ohne in jedes Komponent einzeln zu importieren [Goo10e].

4.1.8 Services

Um bestimmte Funktionen in verschiedenen Komponenten zur Verfügung zu stellen, kann ein Service in Angular nützlich sein. Der Service wird in einer eigenen Klasse definiert und anschließend im Provider-Array vom Root-Modul der Webanwendung integriert. Dieser Service kann durch die sogenannte **„Dependency Injection“** für alle Komponenten erreichbar sein.

Listing 4.2 Code-Ausschnitt aus dem Webanwendung Root-Modul

```
@NgModule({
  declarations: [
    AppComponent,
    ErrorListViewComponent,
    ServerStatusComponent,
    ErrorListFilterComponent,
    ErrorListComponent,
    ...

  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    ...
  ],
  providers: [
    TitleService,
    KbsDataService,
    FilterService,
    ...
  ],
  bootstrap: [AppComponent]
})
```

Quelle: eigener Code-Ausschnitt aus dem Root-Modul

In Komponenten von Angular existieren einige Services, welche bereits implementiert sind und kann jederzeit über den Konstruktor einer Komponente angefordert werden. Ein Beispiel für die Implementierung von Services im Konstruktor wird im folgenden Listing 4.3 dargestellt.

Listing 4.3 Code-Ausschnitt aus einer Komponente der Webanwendung

```
constructor(
  private _route: ActivatedRoute,
  private _router: Router,
  private _kbsDataService: KbsDataService,
  private _titleService: TitleService
) {}
```

Quelle: eigener Code-Ausschnitt vom Konstruktor einer Komponente

Die Services im oberen Code-Ausschnitt stammen aus unterschiedlichen Quellen. Die zwei ersten Services, welche **„ActivatedRoute“** Service und **„Router“** Service sind, sind bereits in Angular implementiert und können im Prinzip jederzeit aufgerufen werden. Die nächsten zwei Services **„KbsDataService“** bzw. **„TitleService“** sind Services, welche im Provider-Array vom Root-Modul integriert und von der „Dependency Injection“ zur Verfügung gestellt werden.

Die Services, welche in diesem Beispiel sind, können, ohne diese direkt zu importieren, in der Komponente verwendet werden.

Anstatt viele existierende Instanzen der Services, wenn jede Komponente den Service selbst initiiert, existiert zur Laufzeit durch die „Dependency Injection“ nur eine Instanz der Services. Außerdem wird durch die Verwendung eines Services ermöglicht, dass auf ein Element von verschiedenen Komponenten aus, ohne dass dieses Element mehrfach existieren muss, zugegriffen werden kann [Goo10b].

4.1.9 Router

Der Router wird für das dynamische Laden der Seiteninhalte verwendet. Es existiert ein sogenanntes **„Router-Modul“**, welches vom Entwickler implementiert werden muss. Die Pfade inkl. den zu ladenden Komponenten werden in diesem Modul in einem Array definiert. Anhand des definierten URL-Pfads kennt der Router, welche Seite gefordert ist und ruft die entsprechende Komponente als Seiteninhalt auf.

Es besteht auch die Möglichkeit, dass eine Weiterleitung mit einem bestimmten Pfad zu definieren oder eine Wildcard-Zuweisung mit der Syntax **„**“** für unbekannte Routes festzulegen. Ein Beispiel vom „Router-Modul“ wird im folgenden Listing 4.4 abgebildet.

Listing 4.4 Code-Ausschnitt aus Router-Modul der Webanwendung

```
const routes: Routes = [
  { path: "", pathMatch: "full", component: ErrorListViewComponent },
  { path: 'errors/:errorId', component: ErrorDetailsViewComponent },
  { path: '**', component: ErrorListViewComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [
    RouterModule
  ]
})
```

Quelle: eigener Code-Ausschnitt der Route-Definition

Eine Besonderheit vom Router ist, dass dieser ebenfalls Parameter in der URL durch einen trennenden Schrägstrich und einen Doppelpunkt entgegen nehmen kann. Das heißt, dass in dem Beispiel mit der URL **„errors/1234567“** sich der Parameter **„errorId=1234567“** ergibt. Mithilfe vom Service **„ActivatedRoute“** kann dieser Parameter in einer Komponente, welche diesen Service integriert, unter den Alias **„errorId“** entnommen werden.

Um die Registrierung von Routen im Programm nutzen zu können, muss im Import-Array dieses Modules der Eintrag **„RouterModule.forRoot(routes)“** existieren.

Der Startpunkt der Router muss zudem im Root-Template mit dem Tag **„<base href='/'>“** festgelegt werden. Von dort aus beginnt das Routing.

Anschließend, um die Ausgabe der einzelnen Komponenten zu ermöglichen, muss noch der Tag **„<router-outlet> </router-outlet>“** gesetzt werden. Die Ausgabe vom Router-Modul wird aus diesem Tag veröffentlicht [Goo10g].

4.1.10 Observables

Die Reactive Extensions für JavaScript (RxJS) bietet die asynchronen Arbeitsabläufen in einem Projekt an. Der asynchronen Abläufen bzw. callback-basiertem Code wird mithilfe von RxJS einfacher abhandeln [SS17].

Bei Server-Abfragen bzw. beim nötigen Anfragen nach mehreren Werte tauchen die asynchronen Abläufe am meisten auf. Um dieses Problem zu lösen, bietet RxJS die sogenannte **„Observables“** an, welche die Anfragen asynchron abhandeln und gleichzeitig mehrere Werte zurückliefern.

Auf die Fertigstellung eines Prozesses, wie bei synchronen Funktionen, muss nicht gewartet werden und der Code kann hierbei normal weiterlaufen, während der Server durch die Anfrage beschäftigt ist. Dieses Verhalten wird in dieser Abbildung 4.1 grob dargestellt.

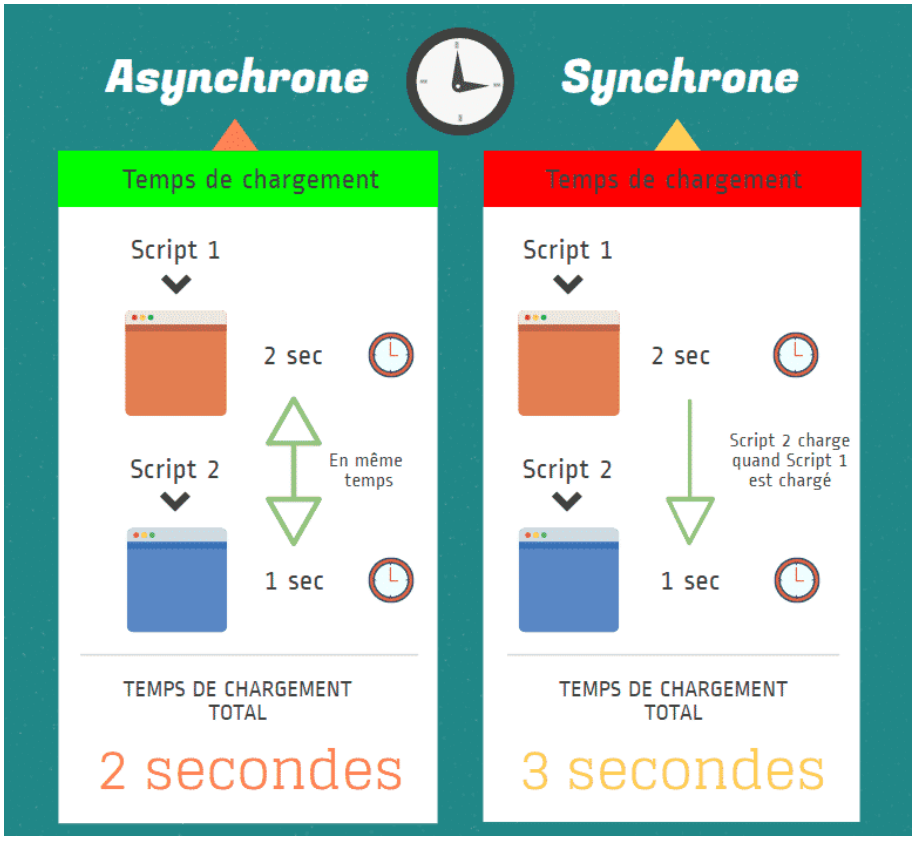

Abbildung 4.1: Beispiel Abbildung über Vorteil zwischen asynchrone gegenüber synchrone Anfragen

Quelle: Abbildung aus dem Artikel [Mar16]

Um die zurückgelieferten Werte der Observable zu bearbeiten, muss diese Observable abonniert werden, welche auch als **„Observable Subcribed“** bezeichnet ist. Eine Callback-Funktion wird beim Abonnieren übergeben, welche mit jeder erhaltenden Rückgabe ausgeführt wird.

Um die Funktionalität von Observables bisschen klar darzustellen, wird es im folgendem Listing 4.5 abgebildet.

Die Werte **„firstElement“** und **„secondElement“** sollen direkt vom Subcriber ausgeben werden und nach einem Zeitraum von 5 Sekunden den Wert **„thirdElement“** in der Konsole anzeigen. Um die Asynchronität darzustellen, wird jeweils bevor der Subcriber beginnt und nachdem der Subcriber endet jeweils eine Marke gesetzt.

Die zwei Vorteile von Observables sind einerseits die Möglichkeit, dass die Observables mehrfach Werte zurückliefern können und anderseits der Code weiter ausgeführt wird.

Listing 4.5 Code Beispiel für Observable Verhalten

```
1  var observer = Rx.Observable.create(subject => {
2    subject.next("firstElement");
3    subject.next("secondElement");
4    setTimeout(() => subject.next("thirdElement"), 5000);
5  });
6
7  console.log("start");
8  observer.subscribe(result => console.log(result));
9  console.log("next");
10
11 /* Output: start firstElement secondElement next thirdElement */
```

Quelle: eigenes Codebeispiel

4.1.11 Angular Material

Eine Angular Framework-Erweiterung, welche die Layout-Entwicklung mit einheitlichen Styles erweitert, ist Angular Material. Angular bietet viele vordefinierte Komponenten mit verschiedenen Eingabeparametern und Events für Styles an. Der Style von solchen Komponenten ist schon fest implementiert und kann nur geringfügig angepasst werden. Über die Schnittstelle von einzelnen Komponenten können die Eigenschaften solcher Komponenten bestimmet werden. Beispielsweise können die Datenbindungen angewendet werden. Eine ausführliche Auflistung bzw. ein Code-Beispiel zu jeder Angular Material-Komponente kann auf der Webseite von Angular Material entnommen werden.

Um Angular Material verwenden zu können, muss Angular Material bzw. das Angular Component Development Kit (CDK) installiert werden. Die verwendeten Komponenten aus den installierten Bibliotheken können anschließend im Root-Modul importiert und im Import Array eingetragen werden. Die Verwendung steht für alle entsprechenden Komponenten frei, wenn ein Modul importiert wurde. Ein entsprechender Tag der Komponenten muss im HTML-Code eingetragen und die nötigen Parameter gesetzt werden [Goo10c].

Ein Beispiel von der Verwendung von Angular Material wird im folgenden Listing 4.6 dargestellt. Der **„mat-icon-button“** wird aus der „Angular Material“-Bibliothek kreiert, welches aus dem Icon Material des Unternehmes stammt. Außerdem wird dieser Button mit einem Klick-Event implementiert. Falls dieser Button vom Nutzer angeklickt wird, führt die zugehörige Methode **„delError(element.id, element.title)“** aus, welche in TypeScript definiert ist und die Parameter gleichzeitig der Methode mitliefert.

Listing 4.6 Code Beispiel für Angular Material Button

```
1 <button mat-icon-button (click)="delError(element.id, element.title)">
2   <mat-icon class="Bosch-Ic Bosch-Ic-delete"></mat-icon>
3 </button>
```

Quelle: eigenes Codebeispiel

4.2 JSON

Ein Austauschformat, welches auf Text basiert und unabhängig von Programmsprachen ist, ist **JSON**. Wie Extensible Markup Language (XML) übernimmt JSON die Datenübertragung zwischen den Anwendungen und den Datenaustausch innerhalb der Anwendung. Der Vorteil von JSON beruht nicht nur auf der breiten Unterstützung von allen populären Programmiersprachen, sondern auch die Darstellung von Objekten als Text bzw. die leichte Übertragung solcher Texte und die Umkehrung, also von Text zu Objekt.

Ein JavaScript-Objekt kann wie im folgenden Listing 4.7 in einen String für die Datenübertragung verwandelt bzw. zurückverwandelt werden.

Listing 4.7 Code Beispiel für handle mit JSON

```
// Beispielobjekt
  errorOrgObj = {
  id: "123456",
  mandateId: "12",
  creationDate: "20.02.2018",
  changeDate: "20.03.2018",
  translation: {
    id: "56",
    lcid: 0
  }}

// Konvertiert Beispielobjekt in String
  errorString = JSON.stringify(errorOrgObj);
  console.log (errorString);

  /* Output:
  *{
  * "id":"123456",
  * "mandateId":"12",
  * "creationDate":"20.02.2018",
  * "changeDate":"20.03.2018",
  * "translation":{
  *  "id":"56",
  *  "lcid":0
  * }
  }*/

/* Rückverwandeln diese String und vergleicht diesen mit der originale Objekt*/
  errorObj = JSON.parse(errorString);
  console.log(JSON.stringify(errorOrgObj) === JSON.stringify(errorObj));

  // Output: true
```

Quelle: eigenes Codebeispiel

4.3 REST-Schnittstellen

REST-Schnittstelle ist eine von mehreren Schnittstellen, welchen Daten verarbeiten bzw. bereitstellen.

Die Daten können durch REST-Schnittstellen gelesen, geschrieben und gelöscht werden. Solche Aktionen werden durch die Arten von **Hypertext Transfer Protocol (HTTP)-Anfragen (beispielsweise GET, POST, DELETE, PUT)** ausgeführt. Die Anfragen werden als URL mit der entsprechenden Komponente an der REST eingereicht und bearbeitet. Wenn eine Anfrage Rückgabewerte erwartet, wird eine Repräsentation der Ressource im XML Format oder im JSON Listing 4.8 Format kreiert und zurückgeben.

Listing 4.8 JSON als Rückgabewert

```
1  "error": {
2    "id":"123456",
3    "mandateId":"12",
4    "creationDate":"20.02.2018",
5    "changeDate":"20.03.2018",
6    "translation":{
7      "id":"56",
8      "lcid":0
9    }
10 }
```

Quelle: JSON Beispiel aus dem Unternehmen

Die erhaltenen Rückgaben können jeweils vom Empfänger entschlüsselt und als Objekte dargestellt werden [GF10] [New17].

Für die Daten- bzw. Informationsanfragen, welche für das Bachelor Projekt nötig sind, ist eine REST-Schnittstelle zur Verfügung gestellt. Diese REST-Schnittstelle wird von einem zuständigen Entwickler gepflegt. Es existiert eine Unterteilung von Fehlern mit Alias **„Errors“**, welche wiederum in Fehlercodes mit Alias **„ErrorCodes“** usw. aufgeteilt sind. Durch solche Unterteilungen können beispielsweise zu einem Fehler eine konkrete Fehlercode und eine entsprechende Lösung, die auch mit bestimmten Sprachen definiert sind, ausgesucht werden.

Diese Schnittstelle wird jetzt zurzeit über eine GUI, welche als **„Swagger-GUI“** bezeichnet wird, dokumentiert. Die Interaktion mit der Schnittstelle ist dadurch benutzerfreundlicher und stellt einen einfacheren Einstieg für die Entwickler dar. Nachfolgend ist eine ausgeschnittene Abbildung 4.2, welche zurzeit als alternative Visualisierung der REST-Schnittstelle dient.

„Swagger-GUI“ ermöglicht dem Entwicklungsteam bzw. dem Nutzer dieser GUI die Schnittstelle zu visualisieren und darin zu interagieren, ohne das Wissen über eine Implementierungslogik zu besitzen. Sie wird automatisch aus der definierten Swagger-Spezifikation generiert. Die visuelle Dokumentation vereinfacht die Backend-Implementierung [Sma18].

Die nötigen Daten können durch verschiedene URL-Endungen abgefragt werden. Die **GET**-Parameter können auch die Details der nötigen Daten wiedergeben, wie beispielsweise die Details eines Fehlers. Die REST-Schnittstelle wertet die erhaltene URL aus und gibt bei Bedarf eine Repräsentation der dazugehörigen Daten als JSON-Objekt zurück.

4.4 Designvorgabe: BOSCH-„Look and Feel“

Das Designsystem vom Unternehmen wird auch als **„Atomic Designsystem“** bezeichnet. Atomares Design ist eine Methodik zur Erstellung von Designsystemen. Ähnlich wie bei der Chemie

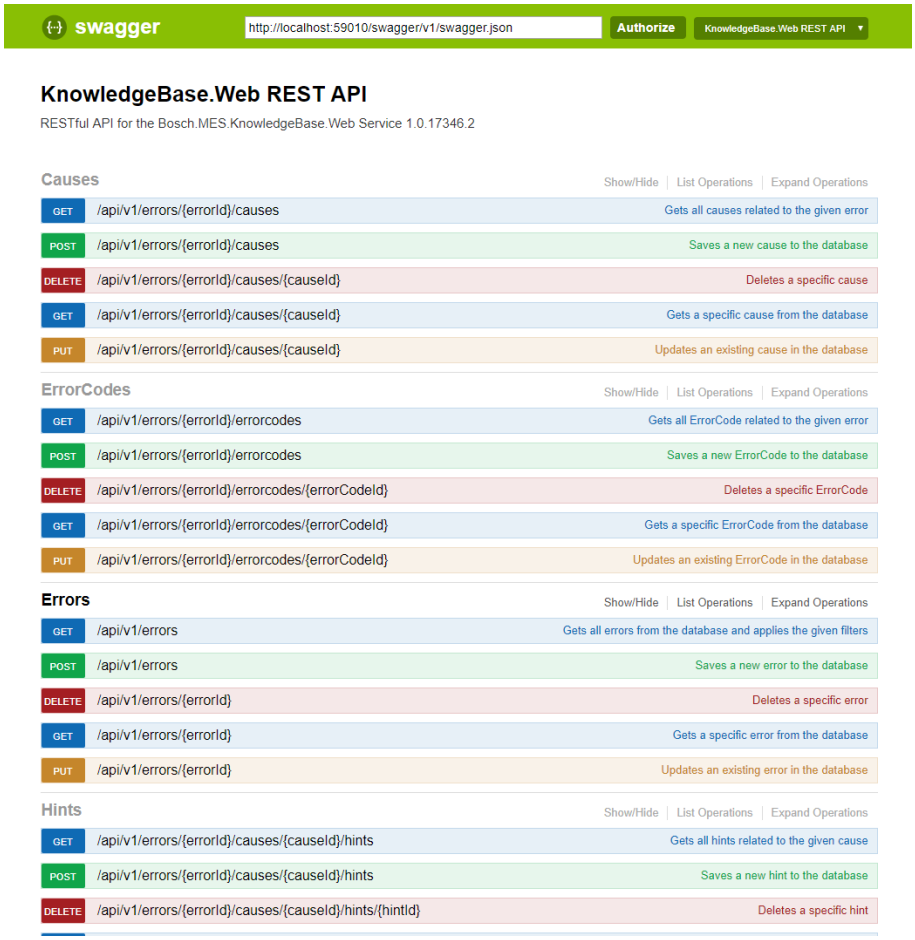

Abbildung 4.2: Ausschnitt Abbildung über aktuelle, verwendete Swagger-GUI
Quelle: Abbildung aus der zurzeit verwendeten Swagger-GUI des Unternehmens

werden Systemkomponenten in verschiedene Ebenen, welche als Atome, Moleküle, Organismen, Vorlagen und Seiten bezeichnet werden, eingeteilt. Die Atome verbinden sich zu Molekülen, welche wiederum zusammen komplexere Organismen bilden. Sie erstellen schließlich die Vorlagen und Seiten, welche eine Benutzeroberfläche darstellen. Diese Vorgänge lassen sich in dieser Abbildung 4.3 wiedergeben.

Die Vorteile solcher atomaren Design-Methodiken sind eine klare Methodik für Designer bzw. Kunden, ein klares Bild, wie ein Designsystem funktioniert und bildet eine Brücke zwischen Design und Entwicklung. Weiter fördert es die Konsistenz sowie Skalierbarkeit, welche sich positiv auf die Nutzererfahrung und die Markenwahrnehmung auswirken.

Das Design vom Unternehmen wurde jahrelang entwickelt und wird noch immer ständig von der User Experience (UX)-Abteilung erweitert. Jeden dieser einzelnen Elemente des atomaren Designs werden für die einheitliche Darstellung der Anwendungen des Unternehmens festgelegt und vorgegeben.

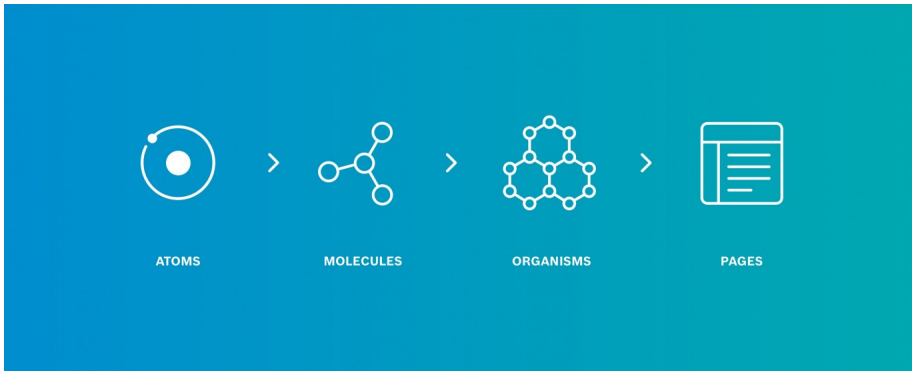

Abbildung 4.3: Atomic Designsystem
Quelle: Abbildung aus dem Design Dokument des Unternehmens

4.4.1 Atome

Die Grundbausteine des Designsystems vom Unternehmen sind sogenannte **„Atome“** (siehe Abbildung 4.4). Solche Designelemente können nicht weiter zerlegt werden und lassen sich zu komplexeren Elementen zusammenfügen. Gegenüberstellend zur Schnittstelle sind Atome HTML-Tags, welche beispielsweise ein Label, Eingabefeld oder Button sein können. Außer diesen können Atome auch aus abstrakten Elementen wie Farbpaletten, Schriftarten oder Animationen bestehen.

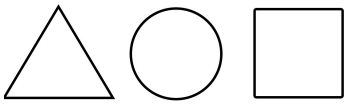

Abbildung 4.4: Designelemente: Atomare
Quelle: Abbildung aus dem Designdokument des Unternehmens

4.4.2 Moleküle

Die Moleküle entstehen aus Gruppen von Atomen, welche miteinander verbunden sind und das Rückgrat vom Entwurfssystem vom Unternehmen bilden (siehe Abbildung 4.5). Sie besitzen eigene Eigenschaften und fungieren als Einheit. Beispielsweise kann eine Suchfeldkomponente aus einer Zusammensetzung von einem Label, einem Eingabefeld und einem Button, welcher beim Betätigen eine bestimmte Suchfunktion ausführt. Die Moleküle sind relativ einfach gebaut, aber sind funktional und können bei Bedarf wiederverwendet werden.

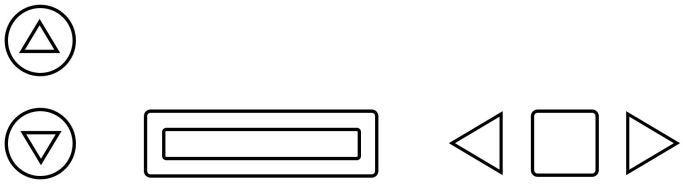

Abbildung 4.5: Designelemente: Moleküle
Quelle: Abbildung aus dem Designdokument des Unternehmens

4.4.3 Organismen

Die Zusammensetzungen aus Molekülen, welche einen relativ komplexen, eindeutigen Abschnitt einer Schnittstelle bilden, sind sogenannte Organismen (siehe Abbildung 4.6). Ein Organismus ist eine geschlossene Funktionseinheit, in dieser keine weitere Verschachtlung möglich ist. Ein Beispiel wäre die Kopfzeile oder Fußzeile einer Benutzerschnittstelle. Außerdem arbeiten die Organismen eigenständig und sind tragbare, wiederverwendbare Komponenten. Sie können aus ähnlichen oder verschiedenen Molekülarten bestehen.

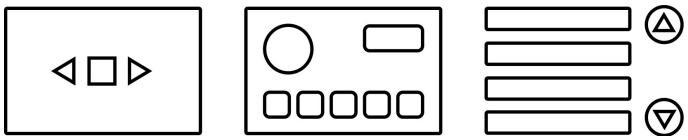

Abbildung 4.6: Designelemente: Organismen
Quelle: Abbildung aus dem Designdokument des Unternehmens

4.4.4 Seite

Das letzte, entstandene Ergebnis, was einem Benutzer angezeigt wird, ist die Seite (siehe Abbildung 4.7). Sie bietet einen greifbaren Kontext für Atome, Moleküle und Organismen. Außerdem stellt sie die Effektivität des Designsystems des Unternehmens dar.

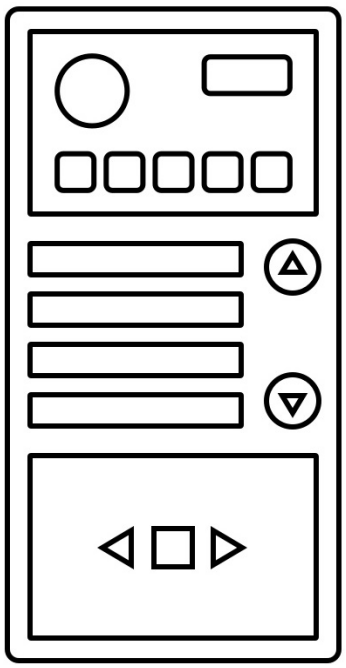

Abbildung 4.7: Designelemente: Seite
Quelle: Abbildung aus dem Designdokument des Unternehmens

4.5 Entwicklungsumgebung

Um die Entwicklung mit der Webtechnologie **„Angular“** zu ermöglichen, gibt es viele mögliche Integrierte Entwicklungsumgebungen (IDEs). Die zwei am meisten verwendeten IDEs sind Visual Studio Code und WebStorm. Andere IDEs werden hier nicht in Betracht gezogen, da es sonst den Rahmen der Bachelorarbeit sprengen würde.

4.5.1 WebStorm

Eine bekannte IDE, welche von der Firma JetBrains entwickelt und für die Programmiersprache JavaScript spezialisiert ist, ist WebStorm [Wik17c]. Außer der Programmiersprache JavaScript unterstützt WebStorm auch andere Programmiersprachen, wie HTML5, Node.js, Bootstrap, Angular, TypeScript usw.. und wird oft in der Entwicklung von webbasierten Mobile-Anwendungen eingesetzt. Die Funktionalität dieser IDE kann mit der Integration von verfügbaren Plugins, welche zum Teil von JetBrains selbst und zum Teil von der Community entwickelt wurden, erweitert werden.

Wie andere IDEs der Firma JetBrains bietet dieser auch eine minimale Benutzeroberfläche, einen einfachen Einstieg, klare Refactoring-Funktionalität, integrierte Unit-Tests und intelligente Code-Vervollständigung bzw. Formatierung an.

4.5.2 Visual Studio Code

Eine weitere häufig genutzte IDE, welche von Microsoft entwickelt wird, heißt Visual Studio Code [Wik17b]. Es ist ein kostenloser Texteditor, welcher in verschiedenen Betriebssystemen betrieben werden kann und für die Entwicklung von Webanwendung sehr beliebt eingesetzt wird. Außer der Unterstützung von webbasierten Programmiersprachen kann Visual Studio Code auch wie WebStorm mithilfe von Plugins erweitert und damit auch zur Entwicklung von vielen anderen Programmiersprachen verdet werden.

Visual Studio Code basiert auf dem Electron Framework und bietet wie WebStorm auch Debugging sowie Code Autovervollständigung an.

4.5.3 Vergleich zwischen WebStorm und Visual Studio Code

Die Entwicklung der Webtechnologie **„Angular“** kann mit diesen beiden IDEs ermöglicht werden. Es gibt viele Artikel, welche diese beiden IDEs gegenüber stellt. Diese IDEs werden hier unter folgenden Aspekten, welche in Projekt Einstieg, TypeScript-Unterstützung und Performanz unterteilt sind, untersucht[Ró16].

Projekt Einstieg

Die Projekt-Einrichtung von Visual Studio Code ist simpel und hat eine Ähnlichkeit gegenüber WebStorm. Mit Visual Studio Code kann man mehrere Aufgaben gleichzeitig ausführen, hingegen bei WebStorm nur jeweils eine Aufgabe.

WebStorm verfügt über einen integrierten Task-Runner, welcher an vordefinierten Einstellungen für alle gängigen Tools angepasst werden kann. Außerdem hat JetBrains IDE viele nützliche Funktionen, wie Code Linters (JSlint, TSlint), welche bei Visual Studio Code als Plugin selbst integriert werden müssen. Bei WebStorm sind sie bereits vorinstalliert. Die Installation ist hier nicht der wichtige Vergleichspunkt, sondern der Aufwand, um nach der Installation diese Plugins mit dem Projekt einwandfrei lauffähig zu bekommen.

So erleichtert WebStorm bei der Erstellung eines neuen Projektes den Entwicklern die Generierungschritte, was bei Visual Studio Code mit vielen Aufwänden verbunden ist, diese Schritte abzuschließen.

TypeScript

TypeScript wird optimal in Visual Studio Code unterstützt, weil es auch eine Technologie von Microsoft ist. Visual Studio Code benötigt keine große Einrichtung im Editor für TypeScript. Es funktioniert tadellos. Falls eine Konfiguration für TypeScript als **„tsconfig.json“**-Datei im Projekt integriert wird, erkennt das Visual Studio Code automatisch und implementiert die Datei. Alle Fehler bei der Codeeingabe werden direkt vom TS-Linter hervorgehoben.

Eine großer Vorteil von Visual Studio Code ist die Geschwindigkeit. Bei Visual Studio Code erfolgt die Kompilierung sofort und nicht wie bei WebStorm, wo es länger dauert. Andererseits fehlt Visual Studio Code gegenüber WebStorm ein integrierter TypeScript-Compiler, welcher eine Liste aller Fehler nach Daten geordnet anzeigt. Es spart viel Zeit, um die Fehler in großen Projekten mit vielen Daten zu finden.

Performanz

Für diesen Aspekt kann Visual Studio Code den größten Vorteilen bieten. WebStorm ist nicht die schnellste IDE auf dem Markt. In kleinen Projekten kann die IDE eingesetzt werden, jedoch ist sie dann immer noch deutlich langsamer als Visual Studio Code. Damit kann es passieren, dass die Aktualisierung von Informationen nach dem Korrigieren eines Codefehlers bei komplexen, großen Projekten deutlich nachlässt. Beispielsweise kann der integrierte TypeScript-Compiler von WebStorm einige Sekunden für die Fertigstellung seines Jobs benötigen.

Dieses Problem betrifft Visual Studio Code nicht. Es existiert keine Verzögerung und die Aktualisierung zeigt sofort Wirkung.[Ró16]

5 Implementierung der Webanwendung

Im **Kapitel 3** wurden der Entwurf und die Umsetzung vom Weboberflächenkonzept angesprochen und im **Kapitel 4** wurden die Grundlagen der Verwendung vom Angular-Framework erklärt.

In diesem Kapitel werden diese Kenntnisse und Konzepte angewendet, um die Webanwendung zu verwirklichen. Als erstes wird eine kurze Darstellung im **Unterkapitel 5.1** gezeigt und anhand dieses Aufbaus die Webanwendung wiedergeben. Des Weiteren werden kurz die verwendeten Module im **Unterkapitel 5.2** bzw. im **Unterkapitel 5.3** über den aufgeteilten Container erläutert. Weil der Rahmen der Bachelorarbeit beschränkt ist, wird nur ein Teil der Module sowie der Container herausgefiltert und konkretisiert.

5.1 Aufbau

Im Folgenden ist die Implementierung Abbildung 5.1 der Webanwendung zu dargestellt, um einen schnellen Überblick zu erhalten dargestellt.

In der obersten Hierarchie der Implementierung befinden sich einmal das **„App-Module“** und die **„App-Component“**. Die App-Component bildet dabei die grundlegende Darstellung der Webanwendung, welches auch als **Hauptkomponente** bezeichnet wird. Diese Komponente nutzt das App-Module, um die einzelnen Komponenten, Views, Module und Services verwenden zu können. Das App-Module, welches ähnlich App-Components als **„Hauptmodul“** bezeichnet wird, importiert **andere Module** und deklariert die **Komponenten** bzw. **Services**.

5.2 Module

Die Module sind Bausteine der Webanwendung, mit diesen die Webanwendung zusammengesetzt werden kann. Die Module können unabhängig voneinander implementiert werden und schließlich im Root Module einpflegt und verwendt werden. Ein Modul kann aus einer Komponente sowie einem Service oder aus mehreren Komponenten bzw. mehreren Services bestehen, welche für die eingepflegten Module verwendet werden.

5.2.1 App-Module

Das App-Module ist das übergeordnete Modul der Webanwendung, welches über alle anderen Module steht und einen sogenannten Kommunikationspfad für die einzelnen Module und Komponenten innerhalb der Webanwendung definiert. Dieses Modul besteht aus mehreren Arrays, welche sich hauptsächlich aus einem Array für die **Deklaration**, den **Import** und dem **Bootstrap** zusammensetzen (siehe Listing 5.1).

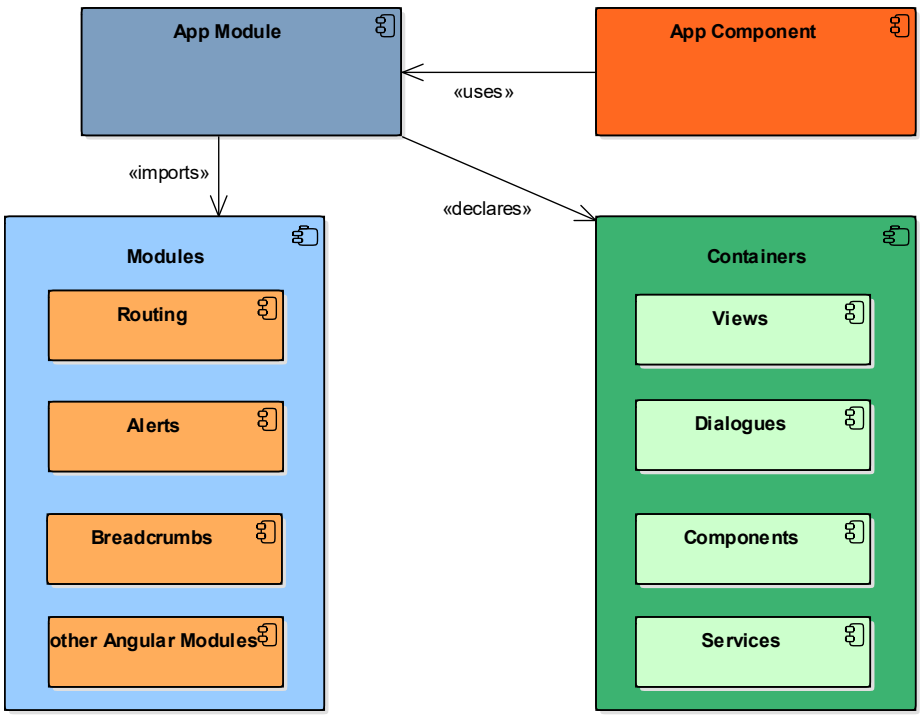

Abbildung 5.1: Implementierungs-Struktur der Webanwendung
Quelle: eigene Abbildung

Innerhalb des Arrays für die **Deklaration** befinden sich alle verfügbaren, definierten Komponenten der Webanwendung. Bei der Neuerstellung einer Komponente muss die Komponente in diesem Array eingetragen werden, damit diese in der Webanwendung genutzt werden kann. Beispielsweise befindet sich im Code-Ausschnitt die App-Component in diesem Array, das heißt, dass diese Komponente in der Webanwendung verwendet werden kann.

Ein weiteres wichtiges Array ist das **Import-Array**. Dieses importiert alle benötigten Module, welche für die Webanwendung in Verwendung sind. Um „Angular-Material“ benutzen zu können, muss das Modul Angular-Material genauso in diesem Array integrieren. Im Code-Auschnitt sind beispielsweise *BrowserModule* und *BrowserAnimationsModule* importiert. Das ermöglicht die Nutzung des Internet-Browsers und die Animation innerhalb diesem.

Das **EntryComponents-Array** ist kein standardisiertes Array des App-Modules. Dieses Array ist ein wichtiger Bestandsteil des App-Modules, wenn in der Webanwendung Dialoge verwendet werden. Alle definierten Dialoge müssen zusätzlich in diesen Array eingetragen werden, um in der Webanwendung referenziert werden zu können. Die *Settings-Component* im Code-Ausschnitt ist eines von den Dialogen, welche für den modalen Settings-Dialogue benutzt werden.

Ein Weiteres Array, welches nicht im App-Module standardisiert ist, ist das **Provider-Array**. Dieses Array übernimmt nach seiner Benennung die Nutzungsfreigabe der Services für alle Komponenten. Das bedeutet beispielsweise, dass der Service *TitleService*, welcher für die Änderung vom Titel zuständig ist, für alle Komponenten der Webanwendung zur Verfügung steht.

Listing 5.1 Code-Ausschnitt aus App-Module der Webanwendung

```
import {...} from ...
...
@NgModule({
  declarations: [
    // App Component
    AppComponent,
    ...
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    ...
  ],
  entryComponents: [
    // Setting dialog
    SettingComponent,
    ...
  ],
  providers: [
    TitleService,
    ...
  ],
  bootstrap: [AppComponent]
  })
export class AppModule {
}
```

Quelle: eigener Code-Ausschnitt aus App-Module

Das letzte Array, das **Bootstrap-Array**, wird mit Komponenten integriert, welche beim Start der Webanwendung ausgewählt bzw. ausgeführt werden. Hier wird die *App-Component* als Hauptkomponente ausgewählt und beim Start der Webanwendung als erstes ausgeführt.

Das App-Module stellt die ganze Webanwendung als ein Paket dar und kann auch in eine andere Webanwendung importiert und verwendet werden. Somit kann jeder Entwickler seine eigenen Module implementieren und bei Bedarf in andere integrieren.

5.2.2 Andere Module

Außer dem Hauptmodul **App-Module** existieren auch weitere Module, welche in das App-Modul importiert werden können, damit alle Komponenten der Webanwendung die Möglichkeit haben, diese zu verwenden.

Falls die Webanwendung aus mehreren View-Components besteht und diese unter einer bestimmten URL angezeigt werden soll, muss ein eigenes **Routing-Module** (siehe Listing 5.2) implementiert und schließlich in das App-Module importiert werden.

In diesem Routing-Module wird eine Routing-Konstante definiert, welche die URL-Pfade und die dazugehörige View-Component beinhaltet. Ein Beispiel wäre die Definition des URL-Pfads der Error-List-View-Component mit der Syntax **path: "errors"**. Zu diesem URL-Pfad existieren Child-View-Components, welche in der Child-Array-Syntax **„children: [...]“** geschrieben werden.

Listing 5.2 Code-Ausschnitt aus Routing-Module der Webanwendung

```
import {...} from ...
...
const routes: Routes = [
  ...
  // url path for error list
  {
    path: "errors", children: [
      { path: "", component: ErrorListViewComponent },
      // url path for error details
      {
        path: ":errorId", children: [
          { path: "", component: ErrorDetailsViewComponent },
          ...
          ]
  },
  ...
];
...
```

Quelle: eigener Code-Ausschnitt vom Routing-Module

Die erste Child-View-Component ist die Error-List-View-Component selbst, welche mit der Syntax **„component: ErrorListViewComponent“** festgelegt ist. Mit dieser Definition kann die Darstellung der Fehler unter dem URL-Pfad **„.../errors“** angezeigt werden.

Des Weiteren besitzt die Child-View-Component mit dem URL-Pfad-Syntax **„path: ":errorId"“** wiederum eine weitere Child-View-Component. Diese Child-View-Component ist die Fehlerdetail-darstellung für bestimmte Fehler-Ids, welche mit der Syntax **„component: ErrorDetailsView-Component“** gebunden wird. Unter dieser URL-Pfad-Definition **„.../errors/:errorId“** können die Fehlerdetails für bestimmte Fehler angezeigt werden. Um die Details an einen bestimmten Fehler zu fixieren, wird die Fehler-Id im Parameter „:errorId“ eingebunden.

Außer diesem **Routing-Module** wird zusätzlich für die Bachelorarbeit ein Breadcrumbs-Modul entwickelt. Dieses Modul stellt einen Pfad für die Navigation dar und erleichtert dadurch das Wechseln zwischen den View-Components.

Es befindet eine große Anzahl an Modulen in dieser Webanwendung, welche verwendet bzw. entwickelt wurden. Die vertiefende Erklärungen einzelner Module werden den Rahmen der Bachelorarbeit überschreiten, deswegen wird hier nicht weiter darauf eingegangen.

5.3 Container

Die Module werden innerhalb des Bachelorprojekts in mehrere Container aufgeteilt, damit eine übersichtliche Darstellung gegeben ist.

Als erstes werden die View-Components erläutert, also welche Darstellung die Webanwendung besitzt. Des Weiteren werden die Komponenten, welche für die Abbildung der View-Components verwendet werden, erklärt. Anschließend werden die Dialoge in den Komponenten kurz erläutert, die für diese Webanwendung erstellt wurden.

Weiter werden die Services, welche unterschiedliche Funktionen besitzen, kurz erklärt. Abschließend werden die angewendeten Konverter und verwendeten Intefaces bzw. definierten Klassen beschreiben.

5.3.1 Views

Die Funktionen der Webanwendung wurden schon in **Kapitel 2** festgelegt. Um diese Funktionalität zu gestalten, wurde eine Besprechung mit dem zuständigen Mitarbeiter des Unternehmens durchgeführt. Die Besprechung ergab, dass die Webanwendung in zwei Darstellungen gespalten wird, welche sich im Views-Container befinden.

Die erste Darstellung besteht aus einer **List-View**. Diese Listen sind dafür da, um die Fehlern, Fehlercodes, Ursachen, Lösungen, Bewertungen und Übersetzungen darzustellen bzw. zu verwalten. Die Daten, die in der jeweiligen Liste angezeigt werden sollen, werden vom zuständigen Mitarbeiter festgelegt.

Ein Beispiel einer List-View von Fehlercodes werden in der Abbildung 5.2 dargestellt. Diese List-View besteht grundsätzlich aus einer **Befehlsleistenkomponente**, welche als „Command-Bar“ (Nummer 1 in der Abbildung) bezeichnet wird, und aus einem **List-Component** (Nummer 2 in der Abbildung), die wie eine Tabelle ist, um die benötigen Daten anzeigen zu lassen.

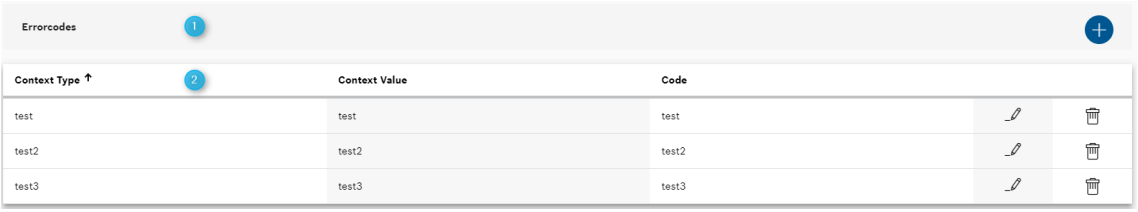

Abbildung 5.2: Darstellung der Fehlercodes in der List-View
Quelle: Abbildung aus der eigenen Webanwendung

Die zweite Darstellung innerhalb dieses Containers ist die **Details-View**. Diese Ansicht ist nur für Fehler, Ursache und Lösung implementiert. Die Gestaltung der anzuzeigenden Daten in der jeweiligen Details-View wird ebenfalls hier festgelegt.

Die Abbildung 5.3 stellt die Details-View eines bestimmten Fehler dar. Die Ansicht besteht auch aus mehreren Komponenten, welche in diesem Fall aus einer **Fehler Details-Component** (Nummer 1 in der Abbildung), **Fehlercode List-View-Component** (Nummer 2 in der Abbildung) und **Ursache List-View-Component** (Nummer 3 in der Abbildung) zusammengesetzt ist.

5.3.2 Dialoge

Um die Fehler, Fehlercode, Ursache, Lösung, Bewertung und Übersetzung zu erstellen bzw. zu verwalten, werden diese Daten durch den sogenannten **Modal-Dialog** gesammelt und durch die REST-API an den Server weitergeleitet. Die Dialoge werden in diesem Dialogues-Container integriert.

Es wurde festgelegt, dass für die Fehler zwei Modal-Dialoge entwickelt werden, welche zum einen die Fehler hinzufügen und zum anderen die Fehler löschen. Für Fehlercode, Ursache, Lösung,

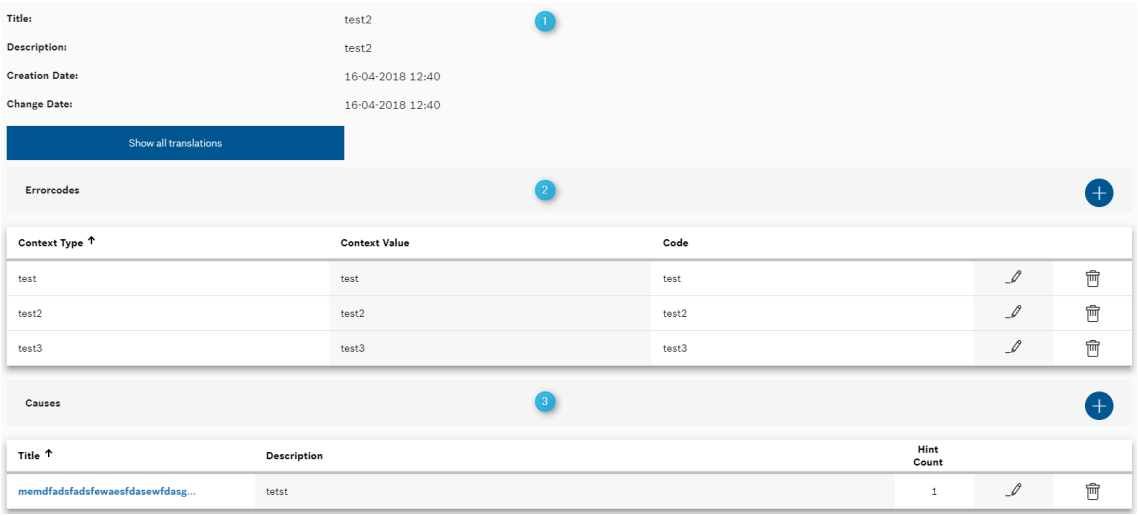

Abbildung 5.3: Darstellung des Fehlers in der Details-View
Quelle: Abbildung aus der eigenen Webanwendung

Bewertung und Übersetzung wird zusätzlich ein weiterer Modal-Dialog definiert, welcher die existierenden Daten Fehlercode, Ursache, Lösung, Bewertung bzw. Übersetzung bearbeitet.

Der erste Modal-Dialog ist der **„Add-Dialogue“**, welcher in dieser Abbildung 5.4 abgebildet ist. Um in diesen Modal-Dialog zu gelangen, soll der Nutzer auf den **Plus-Button** der **Befehlsleistenkomponente** klicken.

Ein Beispiel eines Modal-Dialogs in der Abbildung ist der „Add-Dialogue“ der Fehlerübersetzung. Dieser ermöglicht dem Nutzer die Fehler-Daten hinzuzufügen, welche aus einem Titel, einer Beschreibung und einer dazugehörigen Sprache bestehen. In den Eingabefeldern, welche für den Titel (Nummer 1 in der Abbildung) und Beschreibung (Nummer 2 in der Abbildung) gedacht sind, kann der Nutzer einen Titel und eine Beschreibung zu dem Fehler eingetragen. Diese Eingabefelder sind jeweils mit einer Beschränkung von Eingabezeichen und eine Validierung auf leere Eingabe implementiert. Außerdem kann der Nutzer zu seinen einzufügenden Daten beim Fehler eine Sprache durch das sogenannten **Dropdown-Menü** (Nummer 3 in der Abbildung) auswählen. Wenn alle Validierungen der eingegebenen Daten dürchgeführt sind, wird der **„Add-Button“** (Nummer 4 in der Abbildung) aktiviert und kann vom Nutzer bestätigt werden. Außerdem kann der Nutzer diesen „Add-Dialogue“ verlassen und den Hinzufügen-Prozess unterbrechen, indem er auf den **„Cancel-Button“** (Nummer 5 in der Abbildung) klickt.

Ein weiterer Modal-Dialog, welcher eine Ähnlichkeit zum „Add-Dialogue“ hat, ist der **„Edit-Dialogue“**. Um diesen „Edit-Dialogue“ zu öffnen, muss der Nutzer auf den **symbolisierten Stift-Button** der jeweiligen Zeile des **List-Components** klicken.

Ein Unterschied zum „Add-Dialogue“ ist, dass beim Öffnen dieses „Edit-Dialogues“ bereits Daten von der Zeile, in welche der Nutzer den „symbolisierten Stift-Button“ geklickt hat, geladen und angezeigt werden. Diese Daten werden durch die sogenannte „Datenbindung“ in diesem Modal-Dialog eingebunden. Dadurch kann der Nutzer diese Daten ändern bzw. aktualisieren.

Die Abbildung 5.5 stellt den „Edit-Dialogue“ der Fehlerübersetzung dar. Der Nutzer kann den existierenden Titel, die Beschreibung und Sprache zu dieser Fehlerübersetzung aktualisieren. Wie

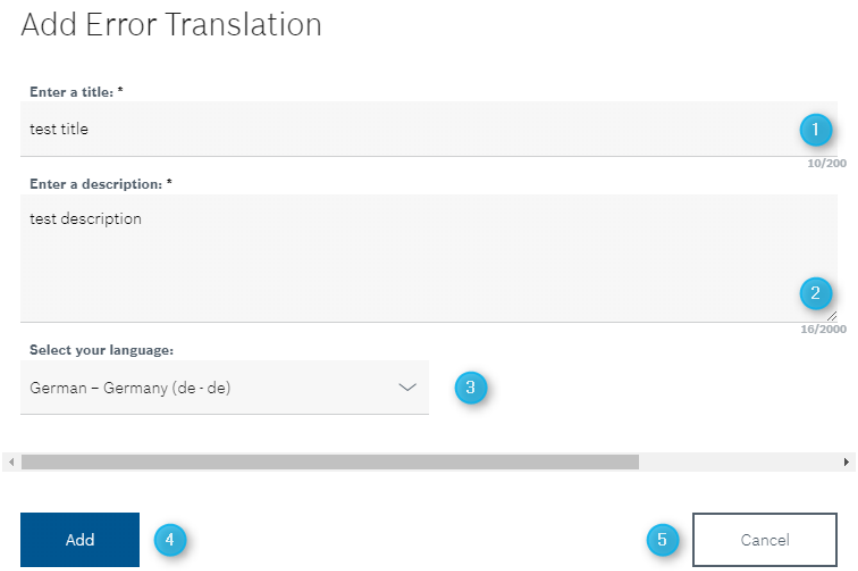

Abbildung 5.4: Fehlerübersetzung des modalen Add-Dialogues
Quelle: Abbildung aus der eigenen Webanwendung

beim „Add-Dialogue“ existiert eine Beschränkung der Eingabezeichen und eine Validierung von leeren Zeichen im Eingabefeld des Titels (Nummer 1 in der Abbildung) und der Beschreibung (Nummer 2 in der Abbildung). Der Nutzer kann hier auch die Sprache zu der Übersetzung mithilfe des **Dropdown-Menüs** (Nummer 3 in der Abbildung) ändern. Genauso wie beim „Add-Dialogue“ wird der **„Save-Button“** (Nummer 4 in der Abbildung) erst aktiviert, wenn alle Änderungen in den Eingabefeldern validiert sind. Der Nutzer kann auch den „Edit-Dialogue“ verlassen bzw. den Änderungsprozess abbrechen, wenn er auf den **„Cancel-Button“** (Nummer 5 in der Abbildung) klickt.

Der letzte Modal-Dialog ist der **„Delete-Dialogue“**. Dieser Modal-Dialog ist dafür da, um die Löschung eines bestimmten Eintrags in der Tabelle, welcher beim Klicken auf dem **symbolisierten Mülleimer-Button** der zugehörigen Zeile des **List-Components** existiert, vorzunehmen. Der Nutzer kann dadurch bestimmte Einträge der Tabelle löschen.

Die Abbildung 5.6 stellt beispielsweise den „Delete-Dialogue“ der Fehlerübersetzung dar. In diesem Modal-Dialog wird der Nutzer zu einer Bestätigung zum Löschen einer bestimmten Fehlerübersetzung aufgefordert. Beim Klicken auf den **„Yes-Button“** (Nummer 1 in der Abbildung) stimmt der Nutzer der Löschung dieser Übersetzung zu und die Löschung wird durchgeführt. Der Nutzer kann auch diesen Lösch-Prozess durch Bestätigung auf den **„No-Button“** (Nummer 2 in der Abbildung) verneinen.

5.3.3 Components

Die Components, welche die List-View, Details-View bzw. Dialogues verwenden, werden in diesem Container entwickelt.

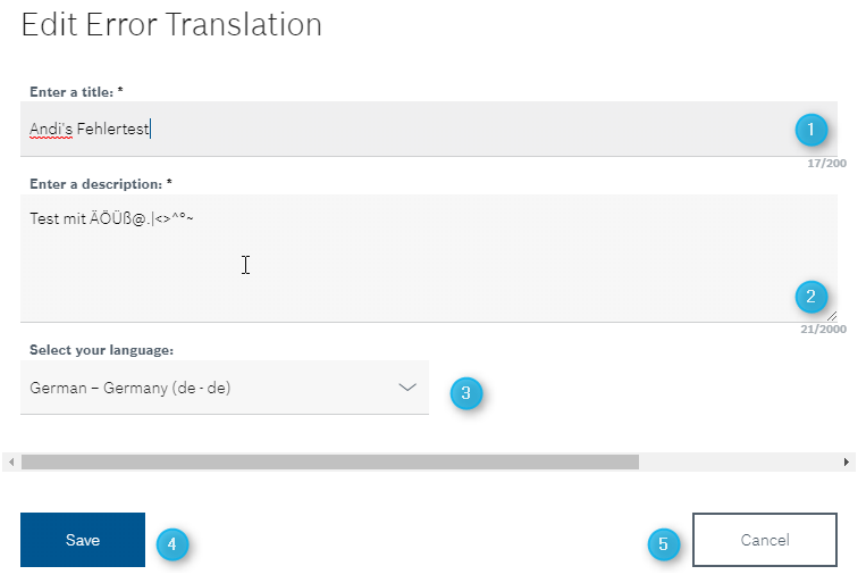

Abbildung 5.5: Fehlerübersetzung des modalen Edit-Dialogues
Quelle: Abbildung aus der eigenen Webanwendung

Abbildung 5.6: Fehlerübersetzung des modalen Delete-Dialogues
Quelle: Abbildung aus der eigenen Webanwendung

Um die List-View darzustellen, werden zwei Komponenten benötigt, welchen aus einer **Befehlsleistenkomponente** und einer **List-Component** zusammengesetzt sind.

In dieser Abbildung 5.7 wird eine **„Befehlsleistenkomponente“** aus der Fehlerübersetzung abgebildet. Grundsätzlich setzt sich diese „Befehlsleistenkomponente“ aus einem **Titel** (Nummer 1 in der Abbildung) und einem **Add-Button** (Nummer 2 in der Abbildung) zusammen. Durch die Bestätigung dieses Buttons gelangt der Nutzer in den „Add-Dialogue“ und kann in diesem Dialog eine Fehlerübersetzung hinzufügen.

Außerdem besitzt diese „Befehlsleistenkomponente“ ein Interaktion-Event mit der **„List-Component“**, welche in dieser Abbildung 5.8 der Fehlerübersetzung dargestellt ist. Das bedeutet, dass beim Hinzufügen einer neuen Fehlerübersetzung vom Nutzer und dessen Zustimmung im

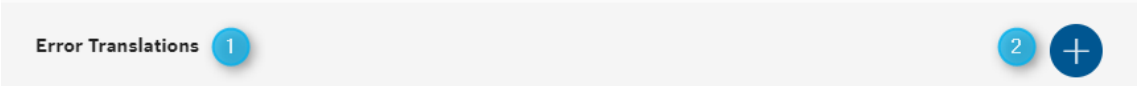

Abbildung 5.7: Befehlsleistenkomponente der Fehlerübersetzung
Quelle: Abbildung aus der eigenen Webanwendung

„Add-Dialogue“, die Fehlerübersetzung die „List-Component“ aktualisiert und neue Daten in dieser Liste anzeigt.

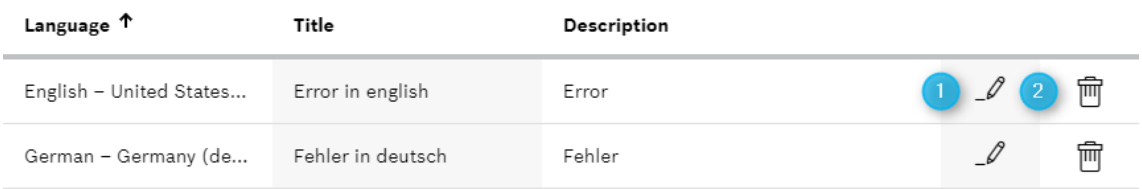

Abbildung 5.8: List-Component der Fehlerübersetzungen
Quelle: Abbildung aus der eigenen Webanwendung

Die „List-Components“ sind jeweils an die unterschiedlichen Nutzungen, also für Fehler, Fehlercode, Ursache, Lösung, Bewertung oder Übersetzung, angepasst. Die anzuzeigenden Daten der Spalten werden für jeden diesen Fälle unterschiedlich sein. In diesem Beispiel ist es eine „List-Component“ der Fehlerübersetzung, welche in fünf Spalten aufgeteilt ist. Die ersten drei Spalten zeigen jeweils die Daten der Fehlerübersetzung, welche in „Sprache“, „Titel“ und „Beschreibung“ aufgeteilt sind. Der Nutzer kann von dieser Darstellung erfahren, welche Übersetzungen zu diesem Fehler existieren. Diese Daten werden im Hintergrund von einem Service, welcher mit der REST-API vom Server kommuniziert und die Daten von diesem sammelt, eingebunden.

Außerdem besitzt diese „List-Component“ noch für jede Datenzeile zwei Funktionen. Eine Funktion ist die **Bearbeiten-Funktion**, welche der Nutzer mit Klick auf dem **symbolisierten Stift-Button** (Nummer 1 in der Abbildung) ausführen kann. Nach der Bestätigung dieses „Stift-Buttons“ erscheint der „Edit-Dialogue“. Die Datensätze in der Zeile vom angeklickten „Stift-Button“, welcher hier aus Sprache, Titel und Beschreibung besteht, wird in den „Edit-Dialogue“ geladen und der Nutzer kann diese Datensätze ändern und aktualisiert. Nach der Aktualisierung wird ein Event ausgelöst, welches die „List-Component“ zum Neuladen auffordert. Die geänderten Datensätze werden in der Liste angezeigt.

Eine weitere Funktion dieser „List-Component“ ist die **Löschen-Funktion**. Diese Funktion kann der Nutzer durch Klicken auf den **symbolisierten Mülleimer-Button** (Nummer 2 in der Abbildung) ausführen. Nach dem Klicken zeigt es dem Nutzer den „Delete-Dialogue“ an. Dieser fragt den Nutzer nach einer Bestätigung zum Löschen der Datensätze in der Zeile vom ausgewählten „Mülleimer-Button“. Genauso wie bei der „Bearbeiten-Funktion“ wird ein Event ausgelöst, welches die „List-Component“ zum Neuladen auffordert. Die ausgewählten Datensätze werden aus der Liste entfernen.

Eine weitere Komponente, welche oft in Dialogen zur Verwendung kommt, ist die **„Sprachmenükomponente“**, welche in dieser Abbildung 5.9 dargestellt ist. Durch die Bestätigung dieser „Sprachmenükomponente“ werden alle verfügbaren Sprachen dem Nutzer angezeigt. Der Nutzer kann anschließend eine Sprache für seine Verwendung auswählen. Nach dem Auswählen der

Sprache wird diese Sprache als ausgewählte Sprache im Sprachmenü vermerkt und angezeigt. Diese ausgewählten Sprachdaten werden im Anschluss an den Service für die Anfrage durch die REST-API vom Server benutzt.

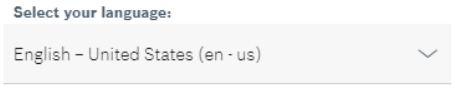

Abbildung 5.9: Sprachmenükomponente
Quelle: Abbildung aus der eigenen Webanwendung

Die „Details-View“ ist eine weitere Hauptansicht der Webanwendung. Diese Ansicht wird für die Darstellung von Fehler, Ursache oder Lösung verwendet. Sie besteht aus einer **„Details-Component“**, welche in der Abbildung 5.10 abgebildet ist, und einer „List-View“ bzw. mehreren „List-Views“. Die Ansicht stellt die Details zu Fehler, Ursache oder Lösung dar, welche aus den **Detaildatensätzen** (Nummer 1 in der Abbildung) und einem **Button** (Nummer 2 in der Abbildung), um die Übersetzungen von Fehler, Ursache oder Lösung anzuzeigen, zusammengesetzt sind. Der Nutzer kann in dieser „Details-Component“ der Abbildung die Fehler Details, welche einen „Titel“, eine „Beschreibung“, ein „Erstellungsdatum“ und ein „Änderungsdatum“ enthalten, betrachten. Diese Datensätze werden genauso wie bei der „List-Component“ von einem Service, welcher mit der REST-API vom Server kommuniziert und die Daten von diesem sammelt, eingebunden. Außerdem kann der Nutzer durch die Bestätigung von **„Show all translations“-Button** in die Übersetzungen „List-View“ zu dem Fehler gelangen. Dort werden alle Übersetzungen zu diesem Fehler aufgelistet.

Abbildung 5.10: Details-Component
Quelle: Abbildung aus der eigenen Webanwendung

5.3.4 Services

Die Datensätze, welche für die „List-View“ und die „Details-View“ bereitgestellt sind, werden von einem Service durch die Kommunikation mit der REST-API vom Server gesammelt und in den jeweiligen Komponenten eingebunden. Dieser Service ist einer der Services, die als **„KBS“** bezeichnet und in diesem Services-Container entwickelt werden.

Um diese Service-Funktionalität besser zu verstehen, wird ein Code-Ausschnitt 5.3 helfen. Dieser Service stellt für jede Komponente, welche diesen Service implementiert, diese Funktionalität bereit und sind für Fehler, Fehlercode, Ursache, Lösung und Bewertung unterschiedlich strukturiert.

In diesem Code-Beispiel werden die Funktionalitäten von diesem Service für Ursache kurz erklärt. Es existieren in diesem Beispiel für Ursache insgesamt sechs Methoden, welchen für unterschiedliche Funktionen eingesetzt werden.

Die erste Methode **„getCauses(...)...“** ist dafür zuständig, die Ursachen vom Server durch eine **„GET-Anfrage“**, welche an die REST-API vom Server gerichtet ist, zu erhalten. Um diese „GET-Anfrage“ durchzuführen, benötigt diese Methode zwei Parameter, welche aus einem Wert Language Code Identifier (LCID) und einer Identifier (ID) eines Fehler bestehen. Bei erfolgreichen Anfragen wird diese Methode einen Rückgabewert als eine Observable mit dem Typ Array die Ursachen zurückliefern. Dieses Ergebnis wird der abonnierenden Komponente zugewiesen. Diese Methode wird von der „Ursache List-View“ verwendet, um alle Ursachen zu einem Fehler anzuzeigen.

Die zweite Methode aus diesem Beispiel ist die Methode **„getCauseDetails(...)...“**, welche in der „Ursache Details-View“ benutzt wird. Die Funktionalität dieser Methode ist ähnlich der ersten Methode. Diese Methode sammelt durch eine „GET-Anfrage“ Daten vom Server und liefert diese an die abonnierenden Komponenten. Anders als die erste Methode ist, dass bei dieser Methode ein zusätzlicher ID-Wert der Ursache gebraucht wird, um die Daten von einer bestimmten Ursache als Rückgabewert zu erhalten.

Die dritte Methode ist **„createCause(...)...“**. Diese Methode ermöglicht die **„POST-Anfrage“** an die REST-API vom Server. Das bedeutet, dass diese Methode eine Hinzufügen-Funktion für die Ursache ermöglicht. Diese Methode benötigt, um diese Anfrage durchführen zu können, außer dem Übergabeparameter ID-Wert von einem bestimmten Fehler noch die hinzuzufügenden Daten der Ursache, welche vom „Ursache Add-Dialogue“ als Typ Map zurückgeliefert werden. Die erhaltenen Daten (als Type Map) werden in einem JSON Objekt umgewandelt und bei der „POST-Anfrage“ an die REST-API mitgeliefert. Der Rückgabewert dieser Methode ist eine HTTP-Antwort, welche aus einem Objekt mit mehreren Eigenschaften in HTTP besteht. Eine HTTP-Eigenschaft, welche hier, damit der Hinzufügen-Prozess dem Nutzer Feedback geben kann, genutzt wird, ist der HTTP-Status-Code. Anhand diesem HTTP-Status-Codes wird der Prozess validiert und bei allen verschiedenen Codes unterschiedliche Rückmeldungen anzeigen.

Die vierte Methode, welche eine Ähnlichkeit zu der dritten Methode besitzt, ist **„updateCause(...)...“**. Diese Methode realisiert die **„PUT-Anfrage“** an die REST-API vom Server. Die Verwendung dieser Methode ist mit dem „Ursache Edit-Dialogue“ verbunden. Das bedeutet, dass diese Anfrage, außer die übergebende Parameter aus dem ID-Wert des speziellen Fehler bzw. der speziellen Ursache, noch die enthaltenen Daten als Type Map benötigt werden. Auch wie bei der Methode „createCause(...)...“ werden die entgegengenommenen Daten vom „Ursache Edit-Dialogue“ in ein JSON Objekt konvertiert und an den Server gesendet. Die zurückerhaltene HTTP-Antwort wird genau wie bei der Methode „createCause(...)...“ abgehandelt, um den Nutzer eine Rückmeldung zu liefern.

Anstatt, dass wie bei der Methode „updateCause(...)...“ ein Aktualisierungsprozess ausgeführt wird, ist die fünfte Methode **„deleteCause(...)...“** für ein Entfernen einer Ursache zuständig. Das heißt, dass diese Methode die **„DELETE-Anfrage“** an die REST-API vom Server verwirklicht. Um diese Anfrage durchführen zu können, benötigt diese Methode als Übergabeparameter einmal den ID-Wert vom bestimmten Fehler und der Ursache. Die zurückgelieferte HTTP-Antwort wird wie die Methoden „createCause(...)...“ und „updateCause(...)...“ bearbeitet, um das Feedback für den Nutzer anzeigen zu können.

Die letzte Methode in diesem Code-Ausschnitt ist die Methode **„getCauseTranslations(...)...“**. Diese Methode wird von der „Ursache Übersetzungen List-View“ abonniert, um alle Ursache Übersetzungen für den Nutzer darzustellen. Um diese **„GET-Anfrage“** an die REST-API vom Server zu ermöglichen, wird der übergebende Parameter als ID-Wert vom speziellen Fehler und der speziellen Ursache benötigt. Diese Methode liefert nach der Durchführung ein Observable vom Typ Array mit den Übersetzungen zurück. Diese Daten werden schließlich in die „Ursache Übersetzungen List-View“ eingebunden und für den Nutzer angezeigt.

Listing 5.3 Code-Ausschnitt aus KBS

```
...
  /**
  * this method gets causes result on special error id
  * and parser it in an array
  * @param lcidValue special lcid set on setting
  * @param errorIdValue uses for get all causes
  */
  public getCauses(lcidValue: number, errorIdValue: string): Observable<ICause[]> {...}

  /**
  * this method gets the cause details on special error id
  * @param lcidValue special lcid set on setting
  * @param errorIdValue special error id to get cause details
  * @param causeIdValue special cause id to get cause details
  */
  public getCauseDetails(lcidValue: number, errorIdValue: string, causeIdValue: string):
      Observable<ICause> {...}

  /**
  * this method creates a cause
  * and post this cause with response to rest api
  * @param errorIdValue uses for post cause of special error id
  * @param causeDataMap is post data of cause
  */
  public createCause(errorIdValue: string, causeDataMap: Map<any, any>):
      Observable<HttpResponse<any>> {...};

  /**
  * this method updates cause with given error id and cause id
  * and put this cause with response to rest api
  * @param errorIdValue uses for put cause of special error id
  * @param causeIdValue uses for put cause of special cause id
  * @param causeDataMap is put data of cause
  */
  public updateCause(errorIdValue: string, causeIdValue: string, causeDataMap: Map<any, any>):
      Observable<HttpResponse<any>> {...}

  /**
  * this method gets error id and cause id
  * and delete this cause with response to rest api
  * @param errorIdValue uses for delete cause of special error id
  * @param causeIdValue uses for delete cause of special cause id
  */
  public deleteCause(errorIdValue: string, causeIdValue: string):
      Observable<HttpResponse<any>> {...}

  /**
  * this method gets cause translations result on special error id \& cause id
  * and parser it in an array
  * @param errorIdValue uses for get all cause translations
  * @param causeIdValue uses for get all cause translations
  */
  public getCauseTranslations(errorIdValue: string, causeIdValue: string):
      Observable<ITranslation[]> {...}
...
```

Quelle: eigener Code-Ausschnitt vom KBS

6 Evaluation der Webanwendung

Im Kapitel 5 wurde die Implementierung der Webanwendung vertieft, wodurch ein Überblick über die Implementierungsstruktur gegeben wurde. Ein wichtiger Aspekt in der Softwareentwicklung, welcher oft wegen mangelnder Zeit, höherem Aufwand und Kosten vernachlässigt wird, ist die Evaluation der Anwendung.

Eine Evaluation einer Webanwendung besteht oft aus einem **UITest**, **UnitTest** und **Bewertungsformular**, welcher von mehreren Nutzern der Webanwendung ausgefüllt wird und aus der statischen Bewertungssammlung zusammengesetzt ist.

Ein UItest konzentriert sich auf die Tests, welche auf der sogenannten GUI basieren. Eine UI-Testautomatisierung ist ein wichtiger Bestandteil eines Projektes. Der Tester kann anhand der angewendeten UI-Testautomatisierungs-Tools die Testfälle selbst definieren, indem diesen durch Aufnahme und Wiedergabe zusammengesetzt sind. UITest hört sich simpel an, leider bieten die UI-Testautomatisierungs-Tools nicht immer ein strukturiertes Aufbauen der Testfälle an. Deswegen werden vernünftige, wartbare und analysierbare Testfälle in diesem Fall gefordert.[Sch17]

Da ein UItest viel Aufwand kostet und das Unternehmen kein vernünftiges UI-Testautomatisierungs-Tool besitzt, welche für die Webanwendung verwendet werden kann, wird in der ersten Überlegung auf eine manuelle UItest per Excel-Tabelle gesetzt. Innerhalb dieser Tabelle werden die Testfälle, welche aus den Punkten wie „Durchführungsbeschreibung“, „erwartetes Ergebnis“ und „Fehlerbeschreibung“ zusammengesetzt sind, definiert. Der Aufwand, um eine komplette GUI-Abdeckung manuell zu testen, ist sehr hoch, deswegen wurde entschieden, dass UItests weggelassen werden und konzentriert sich stattdessen mehr auf Unittests, welche im Abschnitt 6.1 erklärt werden. Anschließend wird anhand des Bewerbungsbogens, auf welches im Abschnitt 6.2 genauer eingegangen wird, das Webanwendung-Feedback ausgewertet.

6.1 Unittest

Ein Unittest, welcher auch als ein Modultest oder ein Komponententest bezeichnet wird, wird oft angewendet, um die Funktionalität der Anwendung zu testen. Der Unittest gehört zu den sogenannten White-Box-Tests. Das bedeutet, dass der zu testende Quellcode bekannt ist und auf diesem die Testfälle basieren werden. Der ganze Quellcode muss beim Testen mindesten einmal ausgeführt werden. Eine der Eigenschaften solcher Tests ist die Isolierung von Testobjekten. Das heißt, dass diese Unittests ohne Interaktion innerhalb der Module in der Anwendung ausgeführt werden können. Dies führt auch dazu, dass alle erforderten Services und das Backend durch Hilfsobjekte, welche auch als Mock bezeichnet werden, simuliert werden müssen.[Wik18]

Um eine Unittest in Angular durchführen zu können, wird das Tool **Karma** (siehe Abschnitt 6.1.1) und das Framework **Jasmine** (siehe Abschnitt 6.1.2) zum Einsatz kommen. Die Konfiguration vom

Karma-Tool wird bereits bei der Neuerstellung eines Angular-Projekts angelegt bzw. integriert und kann nach Bedarf angepasst werden. Ein Unittest in der Webanwendung wird anschließend in der Abschnitt 6.1.3 vorgestellt und erklärt.

6.1.1 Karma

Ein Produkt vom AngularJS-Team ist Karma, welches entwickelt wurde, da das Team kein Testtool auf dem Markt gefunden hat, dass die AngularJS-Funktionen vernünftig testet. Karma wird beim Testen mit Angular flüssig wiedergegeben und bietet die Flexibilität Abstimmungen auf den eigenen Workflow vorzunehmen. Dazu gehört die Möglichkeit den Quellcode auf verschiedenen Browsern und Geräten einschließlich Smartphones oder Tabletts zu testen.

Karma bietet auch die Möglichkeit, außer der Verwendung vom Test-Framework Jasmine (siehe Abschnitt 6.1.2), andere Test-Frameworks zu verwenden oder in verschiedenen, kontinuierlichen Integrationsdienste zu integriert zu werden.[Mor16]

Bei der Neuerstellung eines Angular-projekts wird immer eine Karma-Konfigurations-Datei mit angelegt, welche in folgenden Listing 6.1 grob dargestellt ist.

Listing 6.1 Code-Ausschnitt aus der Karma-Konfiguration der Webanwendung

```
1  ...
2  module.exports = function (config) {
3    config.set({
4    ...
5    frameworks: ['jasmine', '@angular/cli'],
6    plugins: [
7      require('karma-jasmine'),
8      require('karma-chrome-launcher'),
9      ...
10   ],
11   ...
12   autoWatch: true,
13   browsers: ['Chrome'],
14   ...
15   });
16 };
```

Quelle: eigener Code-Ausschnitt aus der Karma-Konfigurationsdatei

Die Karma-Konfiguration besteht aus einer Javascript-Funktion, welche die Testausführungseinstellung festlegt. Innerhalb des **„Frameworks“-Arrays** wird das verwendete Test-Framework festgesetzt, welches als Testrahmen eingesetzt wird. Im Normalfall wird in einem Angular-Projekt das Test-Framework „Jasmine“ angewendet. Bei der Verwendung anderer Test-Frameworks wird eine Registrierung des verwendeten Test-Frameworks innerhalb dieses „Frameworks“-Arrays benötigt.

Gefolgt von diesem „Frameworks“-Array ist das **Plugins-Array** aufgeführt. Innerhalb dieses Arrays wird das nötige Plugin definiert. Die zwei dargestellten Plugins im Codeausschnitt werden benötigt, um Karma mit dem Test-Framework „Jasmine“ und unter der Verwendung vom Internetbrowser „Google Chrome“ ausführen zu können.

Die weiteren Eigenschaften dieser Konfiguration sind einmal das **autoWatch** und einmal das **Browsers-Array**. Das „autoWatch“, welches auf **true** gesetzt ist, führt die Tests im Überwachungsmodus aus. Das heißt, wenn innerhalb eines Tests Änderungen existieren und gespeichert werden, dann wird der Test neu erstellt und erneuert ausgeführt. Das „Browsers-Array“ legt den Internetbrowser fest, in dem der Test ausgeführt werden soll. Standardmäßig wird der Internetbrowser **„Google Chrome“** innerhalb dieses Array festgelegt. Bei der Verwendung anderer Internetbrowser muss der jeweilige Internetbrowser installiert und in diesem Array registriert werden.

6.1.2 Jasmine

Zum Testen von JavaScript-Code wird „Jasmine“ als ein verhaltensorientiertes Entwicklung-Framework verwendet, welches mit Karma kompatibel ist. Wie bei Karma ist „Jasmine“ das empfohlene Test-Framework innerhalb der AngularJS-Dokumentation.[Mor16]

Außerdem ist „Jasmine“ frei von Abhängigkeiten und benötigt kein sogenanntes Document Object Model (DOM), welches eine Spezifikation einer Programmierschnittstelle ist, um HTML oder XML als eine Baumstruktur darzustellen.[Wik17a]

„Jasmine“ bietet viele Features an, um das Testen zu erleichtert. Ein bemerkenswertestes Feature ist das sogennante **„spy“**. Ein „spy“ lässt die Funktion ausspionieren und Attribute verfolgen. Damit kann man überprüfen, ob, wie oft und mit welchen Argumenten eine auspionierte Funktion aufgerufen wurde.

Um eine einfache Darstellung eines „Jasmine Beispiels“ zu zeigen, wird im folgenden Listing 6.2 ein eigenes Testcode-Beispiel vorgestellt.

Listing 6.2 Beispiel Test-Code unter der Verwendung vom Test-Framework „Jasmine“

```
1  ...
2  // the describe title of test
3  describe('Add function test',() => {
4    // one special testcase
5    it('spec to test 1 + 1', () => {
6      // expect that 1 + 1 = 2
7      expect(1+1).toEqual(2);
8    });
9  });
10
11 // Output in browser: test success
```

Quelle: eigenes Code-Beispiel

In diesem Beispiel wird ein einfacher, erfolgreicher Testfall vorgestellt, welche die Summe aus zwei Nummern **„1“** eine Nummer **„2“** ergibt. Grundsätzlich besteht ein „Jasmine“-Test aus einer globalen Jasmine-Funktion **„describe“**, welche mit zwei Parametern, einem **string**-Wert, und einer Funktion. In diesem „string“-Wert wird eine globale Beschreibung für das Testobjekt festgelegt. Innerhalb dieser globalen „describe“-Funktion werden spezielle Testfälle definiert, welche wiederum aus einem „string“-Wert für die Beschreibung und einer Funktion bestehen. Diese spezielle Testfallfunktion wird als **„it“** bezeichnet. In dieser speziellen Testfallfunktion können mehrere, erwartete Verhalten, welche als **„expect“** gekennzeichnet sind, definiert werden. Nach jedem „expect“ folgt eine sogenannte **„Matcher“**-Funktion. Die verwendete „Matcher“-Funktion

ist hier **„toEqual()“**, welche die Gleichheit zwischen dem erwarteten Wert mit dem erhaltenen Wert vergleicht und eine Rückmeldung zurückgibt. Weitere „Matcher“-Funktionen und eine ausführliche Einleitung werden auf der Jasmine-Homepage gegeben.

6.1.3 Aufbau

Wie oben schon erwähnt wird bei der Erstellung eines neuen Angular-Projekts immer eine Karma-Konfiguration mit integriert. Genauso wird bei der Erstellung neuer Komponenten und Services mithilfe von Angular CLI immer eine Testdatei mit angelegt. Diese kann durch den Suffix **„spec.ts“** erkannt werden.

Um die eigene Webanwendung zu testen, wird das folgende Vorgehen festgelegt. Zuerst werden die Objekte, die einen Rückgabewert liefern, wie Services, getestet und anschließend die Komponenten-Objekte nach ihren HTML-Elementen sowie ihren Funktionen untersucht. Eine komplette Auflistung einzelner Testobjekte würde der Umfang dieser Bachelor sprengen. Deswegen wird beispielhaft ein Service-Testobjekt in diesem Listing 6.3 vorgestellt.

Dieser zu testende Service hat die Funktion, dass der LCID-Wert im lokalen Speicher des verwendeten Internetbrowsers hinterlegt wird und ermöglicht dadurch diesen Wert wiederzuverwenden. In diesem Test wird getestet, ob erstens dieser Service erstellt wird, wenn die Webanwendung startet, und zweitens dieser Service den standardisierten, definierten LCID-Wert zurückliefert, falls kein LCID Wert im lokalen Speicher existiert.

Mit der globalen Funktion „describe“, welche aus einer Bezeichnung **„LocalStorageService“** und einer Funktion zusammengesetzt ist, fängt der Test an. Anschließend wird ein Parameter für den Standard-LCID-Wert **„defaultLcid“** und ein Mock-Service **„localStorageService“**, welcher das gleiche Verhalten wie der tatsächliche Service besitzt, kreiert.

Nach dem erzeugten Parameter vom Mock-Service „localStorageService“ wird ein Okjekt dieses Mock-Services diesem Parameter innerhalb der **„beforeEach“**-Funktion zugewiesen. Die „beforeEach“-Funktion wird in diesem Test vor jedem Testfall und die **„afterEach“-Funktion** nach jedem Testfall ausgeführt.

Anschließend folgt die nächste „beforeEach“-Funktion. Anders als andere „beforeEach“-Funktionen wird diese Funktion asynchron ausgeführt. Diese Funktion beinhaltet eine spezielle „Jasmine“-Definition. Diese konfiguriert ein Testmodul, welches dem App-Modul aus dem vorherigem Abschnitt 5.2.1 gleichgesetzt werden kann. Das bedeutet, dass innerhalb dieser Konfiguration alle nötigen Module, Komponenten und Services für den Test implementiert sein müssen, um den Test durchführen zu können. Wie im Code-Ausschnitt dargestellt ist, wird in dieser Testkonfiguration ein sogenanntes **„Provider-Array“** implementiert. Innerhalb dieses Arrays wird der Service, welcher für der Test notwendig ist, definiert. Der verwendete Service für der Test ist ein Mock-Service vom Service „LocalStorageService“ anstatt der Service selbst.

Die vordefinierte Konfiguration wird danach in der nächsten „beforeEach“-Funktion als Parameter „localStorageService“ geladen. Dadurch werden alle Funktionalitäten dieses Mock-Services für den Test verfügbar sein.

Listing 6.3 Beispiel Test-Code eines Services

```
...
describe('LocalStorageService', () => {

  let defaultLcid : number = 1033;
  // param of mock service
  let localStorageService : MockLocalStorageService ;

  // inject local storage mock
  beforeEach(() => {
    localStorageService = new MockLocalStorageService();
  });

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      providers: [
        // use value instance LocalStorageService
        { provide: LocalStorageService, useValue: localStorageService }
      ]
    });
  }));

  beforeEach(() => {
    localStorageService = TestBed.get(LocalStorageService);
  });

  afterEach(() => {
    localStorageService.clearDataFromLocalStorage();
  });

  it('should be create the service', () => {
    expect(localStorageService).toBeTruthy();
  });

  it('should set default lcid (1033), if no lcid data save in local storage', () => {
    // check of local storage empty
    expect(localStorageService.getDataOfLocalStorage("lcid")).toBeNull("local storage is not
        empty");
    // check default lcid
    localStorageService.castLcid.subscribe(lcidData => {
      expect(lcidData).toEqual(defaultLcid, "default lcid is not 1033");
    })
  });
  ...
});
```

Quelle: eigener Code-Ausschnitt aus der „LocalStorageService“-Test-Datei

Damit ein sauberes Testergebnis erzielt werden kann, wird nach jedem Testfall in der „afterEach“-Funktion der lokale Speicher vom verwendeten Mock-Service durch die Funktion **„clearDataFromLocalStorage()“**, welche im Mock-Service deklariert ist, entleert.

Nachdem alle nötige Testkonfigurationsschritte festgelegt sind, kann die Definition einzelner Testfälle beginnen. Im ersten Testfall, welcher in der „it“-Funktion definiert ist und mit der Bezeichnung **„should be create the service“** spezifiziert ist, wird die Existenz dieses Services beim

Starten der Webanwendung erwartet. Dieses erwartete Verhalten wird mit dem Kennzeichnen „expect“ gefolgt von einer „Match“-Funktion **„toBeTruthy()“** ausgedrückt.

Der zweite Testfall wird für die Situation spezialisiert, dass bei keiner Existenz vom LCID-Wert im lokalen Speicher des Internetbrowsers der Mock-Service den standardisierten LCID-Wert (hier **1033**) zurückliefert. Um diese Situation zu testen, wird innerhalb der „it“-Funktion, welche mit **„should set default lcid (1033), if no lcid data save in local storage“** bezeichnet ist, erst auf den existierenden LCID-Wert im Internetbrowser gewartet. Dieses Verhalte besteht wiederum aus einem Kennzeichnen „expect“ gefolgt von einer „Match“-Funktion **„toBeNull()“**. In „expect“ wird der LCID-Wert der Funktion **„localStorageService.getDataOfLocalStorage("lcid")“** des Mock-Services, welcher diesen Wert aus dem lokalen Mock-Speicher holt, erwartet, dass dieser Wert nicht existieren darf. Bei der fehlerhaften Rückmeldung der „toBeNull()“ „Match“-Funktion wird die definierte Fehlermeldung **„local storage is not empty“** innerhalb dieser Funktion als zusätzliche Information vorgelegt.

Anschließend, wenn das vorherige, erwartete Ergebnis positiv ist, wird mit dem nächsten Test fortgefahren. In diesem Test wird auf Gleichheit zwischen dem erhaltenen LCID-Wert vom Mock-Service und dem vordefinierten, erwarteten Standard-LCID-Parameter geprüft.

Das erfolgreiche Testergebnis wird schließlich in dieser Abbildung 6.1 vom Internetbrowser „Google Chrome“ dargestellt.

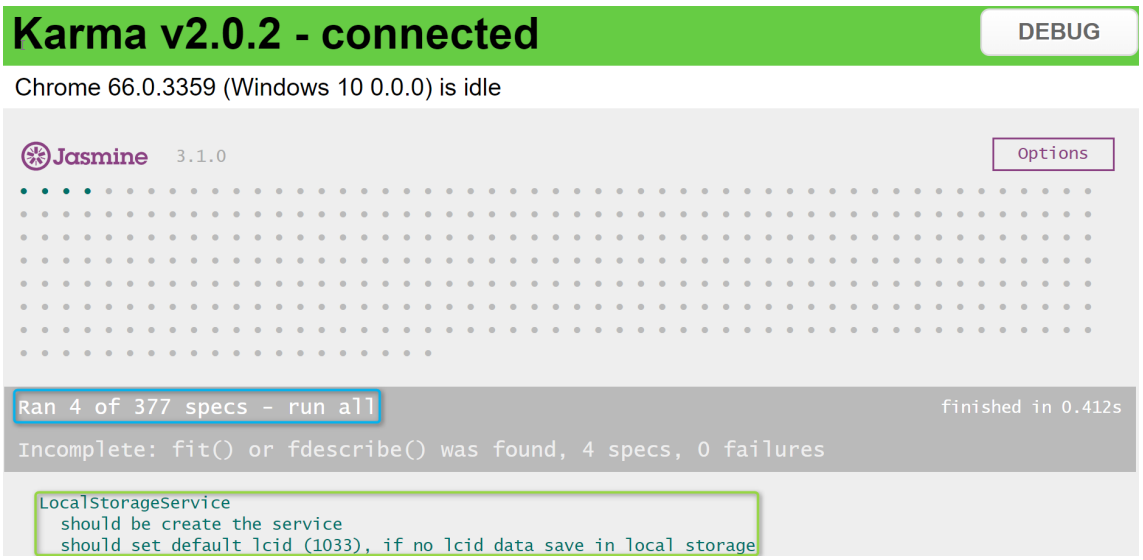

Abbildung 6.1: 2 von 377 erfolgreiche Testmeldung in Internetbrowser „Google Chrome“
Quelle: Abbildung aus eigener Testmeldung

6.2 Protokoll

Im vorherigen Abschnitt 6.1 wurden die Unittest vorgestellt. In diesem Abschnitt 6.2 wird das Protokoll in Form eines Bewertungsformulars dargestellt. Ein grober Aufbau dieses Bewertungsformulars wird im Abschnitt 6.1 erklärt. Anschließend wird eine Auswertung des Bewertungsformulars im folgenden Abschnitt 6.2.2 aufgelistet.

6.2.1 Aufbau

Dieses Bewertungsformular ist in drei wichtige Teile unterteilt, welche im Folgenden in Abbildung 6.2 und Abbildung 6.3 abgebildet sind.

Der erste Teil dieses Bewertungsformulars wird für das Sammeln von Information über den Bewerter, welcher nach dem **„Namen“** und seiner **„Position im Unternehmen“** fragt, verwendet. Der Bewerter kann anschließend in jeweiligen Textfeld die Information eintragen.

der zweite Teil dieser Bewertung besteht aus 10 Fragen mit auswählbaren Bewertungstufen, welche mit **„Trifft zu“**, **„Trifft eher zu“**, **„teils, teils“** und **„Trifft eher nicht zu“** bezeichnet sind. Die Fragen werden die Performance und die Gestaltung dieser Webanwendung gezielt bewerten.

Um weitere Ergänzungen und Bemerkungen des Bewerters zu erfassen, wurde im dritten Teil des Bewertungsformulars zusätzlich ein Textfeld integriert, damit Bewerter weitere Informationen für den Entwickler festhalten können.

6.2.2 Auswertung

Das Bewertungsformular wird insgesamt von drei Bewertern ausgefüllt und es wird jeweils eine Rückmeldung gegeben. Alle drei Bewerter besitzen die Position als Softwareentwickler im Unternehmen. Ein Beispiel eines Bewerters wird in folgender Abbildung 6.4 und Abbildung 6.5 dargestellt. Die Information über den „Namen“ vom jeweiligen Bewerter wird aus Datenschutzgründen nicht angegeben.

Allgemein betrachtet sind die Ergebnisse aus den Bewertungsformularen sehr positiv ausgefallen. Die Darstellung der Webanwendung wird als sehr zufrieden eingestuft. Das Ausführen von einzelnen Operationen ist sehr schnell. Außerdem existieren für die Webanwendung sehr ausführliche Testfälle.

Aus dem Feedback gingen auch konstruktive Verbesserungsvorschläge hervor. Zum Beispiel soll die Darstellung bei langen Listen oder die Tab-Darstellung bei langen Ansichten aufgeteilt werden und eine Darstellungskompabilität für den Microsoft Internet Explorer geschaffen werden.

Webanwendung „NEXEED KnowledgeBase“ Bewertung

Name:

Position im Unternehmen:

Frage	Trifft zu	Trifft eher zu	teils, teils	Trifft eher nicht zu
Ist das Anwendungsinterface klar strukturiert?	☐	☐	☐	☐
Können die gewünschten Operationen in kurzer Zeit durchgeführt werden?	☐	☐	☐	☐
Existiert ein unterstütztes Navigieren in der Anwendung?	☐	☐	☐	☐
Wird Feedback bei der Durchführung von Operationen gegeben?	☐	☐	☐	☐
Ist das Layout gut strukturiert?	☐	☐	☐	☐
Existiert eine saubere Darstellung des Inhalts?	☐	☐	☐	☐
Sind Farbe und Animation gut gestalten?	☐	☐	☐	☐
Sind Schriften gut lesbar?	☐	☐	☐	☐
Existiert Verbesserungsbedarf in der Anwendung? Wenn ja: Was muss noch überarbeitet/ergänzt werden? Geben Sie bitte eine kurze Beschreibung an?	☐	☐	☐	☐
Wird das erwartete Ergebnis nach der Durchführung von Operationen geliefert?	☐	☐	☐	☐
Insgesamt bin ich zufrieden mit dieser Webanwendung.	☐	☐	☐	☐

Abbildung 6.2: Erste Seite aus dem eigenen, erstellten Bewertungsformular für die Webanwendung

Quelle: Abbildung aus dem eigenen Bewertungsformular

Weitere Feedback und Kommentare:

Abbildung 6.3: Zweite Seite aus dem eigenen, erstellten Bewertungsformular für die Webanwendung

Quelle: Abbildung aus dem eigenen Bewertungsformular

Webanwendung „NEXEED KnowledgeBase“ Bewertung

Name:	
Position im Unternehmen:	Softwareentwickler

Frage	Trifft zu	Trifft eher zu	teils, teils	Trifft eher nicht zu
Ist das Anwendungsinterface klar strukturiert?	☒	☐	☐	☐
Können die gewünschten Operationen in kurzer Zeit durchgeführt werden?	☒	☐	☐	☐
Existiert ein unterstütztes Navigieren in der Anwendung?	☒	☐	☐	☐
Wird Feedback bei der Durchführung von Operationen gegeben?	☒	☐	☐	☐
Ist das Layout gut strukturiert?	☒	☐	☐	☐
Existiert eine saubere Darstellung des Inhalts?	☒	☐	☐	☐
Sind Farbe und Animation gut gestalten?	☒	☐	☐	☐
Sind Schriften gut lesbar?	☒	☐	☐	☐
Existiert Verbesserungsbedarf in der Anwendung? Wenn ja: Was muss noch überarbeitet/ergänzt werden? Geben Sie bitte eine kurze Beschreibung an?	☐	☐	☒	☐
* Lange Listen mit Pages aufteilen				
Wird das erwartete Ergebnis nach der Durchführung von Operationen geliefert?	☒	☐	☐	☐
Insgesamt bin ich zufrieden mit dieser Webanwendung.	☒	☐	☐	☐

Abbildung 6.4: Erste Seite aus dem eigenen, erstellten Bewertungsformular für die Webanwendung

Quelle: Abbildung aus einem zurückerhaltenen Bewertungsformular

Weitere Feedback und Kommentare:

+ Sehr umfangreiche Tests
+ Gutes Feedback in UI (Toast)
+ Gute Perfomance
+ Insgesamt sehr gute Umsetzung

Abbildung 6.5: Zweite Seite aus dem eigenen, erstellten Bewertungsformular für die Webanwendung

Quelle: Abbildung aus einem zurückerhaltenen Bewertungsformular

7 Zusammenfassung und Ausblick

Im Kapitel 1 wurde die Technologie der Industrie 4.0 grob vorgestellt und wie diese Technologie als Vorteil im Unternehmen eingesetzt wird. Anschließend wird das Problem bekannt gemacht, wie es zu diesem Bachelorthema gekommen ist. Als nächstes wird im Kapitel 2 über den aktuellen Stand der Lösung geschrieben, um das Problem zu umgehen und wie das Problem durch die Lösung dieser Bachelorarbeit entgegnet werden kann. Außerdem werden innerhalb dieses Kapitels die Anforderungen genau analysiert, welche aus funktionalen und nicht-funktionalen Anforderungen bestehen. Nachdem die Anforderungen festgelegt sind, wird mit dem Entwurf und der Umsetzung der GUI der Webanwendung im Kapitel 3 mithilfe von MockUps begonnen. Das Kapitel 4 befasst sich ausführlich mit den verwendeten Entwicklungs-Frameworks, der Programmsprache, IDE und Designvorgabe vom Unternehmen. Nach den geklärten Festsetzungen wird mit der Implementierung, welche genau in Kapitel 5 erläutert werden, fortgefahren. Die Entwicklung wird anschließend in Kapitel 6 mit der Evaluation mit Unittest und einem Bewertungsformular abgeschlossen.

Zusammenfassung

Es wird in der heutigen Produktion immer wichtiger die Stillstandszeiten der Produktionsphasen im Unternehmen zu verringern. Falls beispielsweise ein Ausfall einer Produktionsanlage am Fließband auftaucht, führt es zu großen, wirtschaftlichen Schäden, welche gern vermieden werden.

Um solche Probleme rechtzeitig zu beheben, existiert bereits innerhalb der Industrie 4.0 Technologie eine sogenannte FMD, welche aus einer Sammlung aller Fehler und dazugehörigen Maßnahmen besteht. Somit hilft es dem Operator diesen existierenden Fehlern zu beheben.

Leider kann der Operator solche Fehler nicht direkt am Standort beheben bzw. die existierende FMD verwalten, weil keine GUI existiert, welche diese Operation zu ermöglicht.

Um dieses Problem zu lösen, wird im Rahmen dieser Bachelorarbeit gefordert, eine Webanwendung zu entwickeln, welche diese Möglichkeit für den Operator bietet. Die Webanwendung soll in erster Linie alle Fehlern, Fehlercodes, Ursachen, Lösungen, Bewertungen und jeweiligen Übersetzungen darstellen. Des Weiteren soll es möglich sein, um neue Fehlern, Fehlercodes, Ursachen, Lösungen, Bewertungen oder Übersetzungen anzulegen oder existierende Fehlern, Fehlercodes, Ursachen, Lösungen, Bewertungen oder Übersetzungen zu verwalten.

Um eine solche Darstellung möglich zu machen, wurde mit dem zuständigen Mitarbeiter im Unternehmen, der für diese Bachelorarbeit zuständig ist, besprochen, dass die Darstellung für die Fehlern, Fehlercodes, Ursachen, Lösungen, Bewertungen und jeweiligen Übersetzungen in Listen untergliedert werden. Die Daten für den einzelnen Fall wird direkt durch den Service der Webanwendung, welche mit der REST API in Verbindung steht, geladen.

Für jede dieser Listen existieren die Funktionen, um neue Datensätze in der jeweiligen Liste anzulegen, alte Datensätze aus bestimmten Listen zu bearbeiten oder zu löschen. Für jede Funktion wird ein zuständiger, modaler Dialog entwickelt, welcher für jeden Falle spezialisiert ist. Außerdem soll es für Fehlern, Ursachen und Lösungen zusätzlich eine Detailansicht geben, in dieser alle nötigen Details aufgelistet sind.

Reflektion

Es gibt viele Schwierigkeiten, die im Laufe der Bachelorarbeit aufgetaucht sind, welche mit Mühe überwunden werden konnten. Als erste Schwierigkeit wäre die Abhängigkeit zwischen den zuständigen Abteilungen. Da jede Abteilung für bestimmte Aspekte der Webanwendung zuständig ist, muss immer wieder ein Besprechungstermin gefunden werden, um einer Lösung für ein Problem zuzustimmen. Dies hängt wiederum von der Verfügbarkeit der Besprechungsräumen bzw. den zuständigen Mitarbeitern ab.

Eine weitere Schwierigkeit war die neue, verwendete Webtechnologie „Angular“. Die Entwicklung mit dieser Webtechnologie musste selbst beigebracht werden, was nicht ganz einfach war. Das verwendete „Angular“-Framework ist nicht das einzige Framework, welches erlernt werden musste. Da für den Test das Test-Framework „Jasmine“ und das Tool „Karma“ verwendet wurde, musste sich genauso mit diesem Framework und dem Tool auseinandergesetzt werden.

Die letzte Schwierigkeit war die Anpassung der vorgegebenen Designvorlage, weil jede Webanwendungen vom Unternehmen unter diesem Design dargestellt werden muss. Die Darstellung muss mit der jeweiligen Abteilung geklärt werden, bevor die Entwicklung der Webanwendung begonnen werden konnte.

Trotz der ganzen Probleme wurde eine Menge Wissen mit der Webtechnologie „Angular“, des Test-Frameworks „Jasmine“ und dem Tool „Karma“ gewonnen, was für die Zukunft sehr vorteilhaft sein kann. Außerdem wurde das Blicken in die Alltagstätigkeiten und dem Teamwork innerhalb der Abteilung erweitert. Darüber hinaus wurde die Kenntnis über die Strukturierung in der Entwicklung solcher Software gewonnen.

Ausblick

Diese Webanwendung wird demnächst je nach Verwendungsbereich eingepflegt und für jeden Mitarbeiter, welcher die alte Lösung noch verwendet, zur Verfügung gestellt.

Außerdem wird diese Webanwendung als GUI in jeden Bereichen innerhalb des „BOSCH“ Unternehmens integriert, welche das FMD verwenden. Dadurch kann das FMD benutzerfreundlich verwaltet werden. Hauptsächlich wird diese Webanwendung in der Fertigung eingesetzt.

Da in dieser Webanwendung im aktuellen Stand noch keine Login-Funktion implementiert hat, kann für die Zukunft eine Login-Funktion eingebaut werden. Somit können bestimmte Operationen für einen bestimmten Personalkreis freigegeben werden und andere Operationen nicht.

Die einzig verfügbare und dargestellte Sprache dieser Webanwendung ist im aktuellen Stand nur Englisch. Das wird sich in Zukunft noch ändern, sodass diese Webanwendung auch in der deutschen Sprache zur Verfügung steht.

Literaturverzeichnis

[Bal10] H. Balzert. „Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering“. In: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum Akademischer Verlag, 9. Jan. 2010, S. 455–474. ISBN: 978-3-8274-2247-7. DOI: https://doi.org/10.1007/978-3-8274-2247-7_16. URL: http://www.ebook.de/de/product/23085500/helmut_balzert_lehrbuch_der_softwaretechnik_basiskonzepte_und_requirements_engineering.html (zitiert auf S. 19).

[Fri14] T. W. Frick. *Von Industrie 1.0 bis 4.0 – Industrie im Wandel der Zeit*. Hrsg. von T. W. Frick. 2014. URL: http://industrie-wegweiser.de/von-industrie-1-0-bis-4-0-industrie-im-wandel-der-zeit/ (zitiert auf S. 11).

[GF10] K. Griffin, C. Flanagan. „Browser based Communications Integration using Representational State Transfer“. In: *Novel Algorithms and Techniques in Telecommunications and Networking*. Hrsg. von T. Sobh, K. Elleithy, A. Mahmood. Dordrecht: Springer Netherlands, 2010, S. 323–328. ISBN: 978-90-481-3662-9 (zitiert auf S. 44).

[Goo10a] Google. *Angular CLI*. 2010. URL: https://cli.angular.io/ (zitiert auf S. 36).

[Goo10b] Google. *Angular Dependency Injection*. 2010. URL: https://angular.io/guide/dependency-injection (zitiert auf S. 39).

[Goo10c] Google. *Angular Material Start Einleitung*. 2010. URL: https://material.angular.io/guide/getting-started (zitiert auf S. 42).

[Goo10d] Google. *Angular Material*. 2010. URL: https://material.angular.io/ (zitiert auf S. 35).

[Goo10e] Google. *Angular Modules*. 2010. URL: https://angular.io/guide/ngmodules (zitiert auf S. 38).

[Goo10f] Google. *Angular Quickstart*. 2010. URL: https://angular.io/guide/quickstart (zitiert auf S. 36).

[Goo10g] Google. *Angular Routing & Navigation*. 2010. URL: https://angular.io/guide/router (zitiert auf S. 40).

[Goo10h] Google. *Angular Template Syntax*. 2010. URL: https://angular.io/guide/template-syntax (zitiert auf S. 38).

[Joy18] Joyent. *Über Node.js*. 2018. URL: https://nodejs.org/en/about/ (zitiert auf S. 36).

[Mar16] M. Martinot. „Installer Google Tag Manager : Présentation et guide d’installation (+ Google Analytics)“. In: (7. Dez. 2016). URL: https://www.mathieu-martinot.com/outils/google-tag-manager/ (zitiert auf S. 41).

[Mor16] A. Morgan. „Testing AngularJS with Jasmine and Karma (Part 1)“. In: (13. Juni 2016). URL: https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1 (zitiert auf S. 66, 67).

[New17] J. Newmarch. „REST“. In: *Network Programming with Go: Essential Skills for Using and Securing Networks*. Berkeley, CA: Apress, 2017, S. 221–245. ISBN: 978-1-4842-2692-6. DOI: 10.1007/978-1-4842-2692-6_14. URL: https://doi.org/10.1007/978-1-4842-2692-6_14 (zitiert auf S. 44).

[Ró16] R. Różański. „Visual Studio Code vs WebStorm“. In: *Visual Studio Code vs WebStorm* (30. Juni 2016). URL: https://blog.goyello.com/2016/06/30/visual-studio-code-vs-webstorm/ (zitiert auf S. 48, 49).

[SS17] D. Schwab, M. Steyer. „Angular: Das Praxisbuch zu Grundlagen und Best Practices, ab Version 4“. In: Animals. O'Reilly, 2017. ISBN: 9783960101451. URL: https://books.google.de/books?id=Yzk2DwAAQBAJ (zitiert auf S. 40).

[Sch17] A. Schladebeck. „Warum alle UI Testing hassen… und warum der schlechte Ruf (teilweise) nicht verdient ist!“ In: (13. Apr. 2017). URL: https://jaxenter.de/ui-testing-kolumne-56149 (zitiert auf S. 65).

[Sma18] SmartBear. *Swagger UI*. 2018. URL: https://swagger.io/swagger-ui/ (zitiert auf S. 44).

[Tut18] Tutorialspoint. *TypeScript - Overview*. 2018. URL: https://www.tutorialspoint.com/typescript/typescript_overview.htm (zitiert auf S. 36).

[Wei18] S. Weinzierl. „Connected Industry: Bosch gründet neue Geschäftseinheit“. In: (19. Feb. 2018). URL: https://www.produktion.de/nachrichten/unternehmen-maerkte/connected-industry-bosch-gruendet-neue-geschaeftseinheit-127.html (zitiert auf S. 12).

[Wik16] Wikipedia. *ISO/IEC 25000*. 14. März 2016. URL: https://de.wikipedia.org/wiki/ISO/IEC_25000 (zitiert auf S. 22).

[Wik17a] Wikipedia. *Document Object Model*. 29. Dez. 2017. URL: https://de.wikipedia.org/wiki/Document_Object_Model (zitiert auf S. 67).

[Wik17b] Wikipedia. *Visual Studio Code*. 2017. URL: https://de.wikipedia.org/wiki/Visual_Studio_Code (zitiert auf S. 48).

[Wik17c] Wikipedia. *WebStorm*. 2017. URL: https://de.wikipedia.org/wiki/WebStorm (zitiert auf S. 48).

[Wik18] Wikipedia. *Modultest*. 27. Apr. 2018. URL: https://de.wikipedia.org/wiki/Modultest (zitiert auf S. 65).

[npm14] I. npm. *npm*. 2014. URL: https://docs.npmjs.com/getting-started/what-is-npm (zitiert auf S. 36).

Alle URLs wurden zuletzt am 29. Mai 2018 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift