BOSCH SOFTWARE INNOVATIONS GMBH

Schöneberger Ufer 89, 10785 Berlin

PFLICHTPRAKTIKUM WS 16/17

# Benutzeroberfläche für eine cloudbasierte Anwendung

*Author:*
Hung Le

Erich - Weinert - Strasse 26
10439 Berlin

*Betreuer:*
Alexander Rothenberg

829729
Fachsemester: 6
Technische Informatik

*Beuth Hochschule für Technik Berlin*

20. März 2017

# Inhaltsverzeichnis

# Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Unterschrift:

Datum:

# Kapitel 1

# Einleitung

Im Rahmen des Bachelorstudiums Technische Informatik – Embedded Systems an der Beuth Hochschule für Technik, ist im 5. Semester ein Pflichtpraktikum für die Studenten vorgesehen. Dieses Praktikum soll dem Studenten dazu dienen, sein bisher erworbenes Wissen bei Lösungen praktischer Aufgaben in Industrie und Dienstleistungsunternehmen zu erproben. Während des Praktikums bekommt der Student erste Einblicke in das spätere Arbeitsleben vermittelt und sammelt erste Erfahrungen.

Im Wintersemester 16/17 habe ich mein Pflichtpraktikum bei Bosch-SI im Testcenter absolviert. Das Testcenter von Bosch-SI erschien mir als interessanter Praktikumsgeber, weil ich davor bereits ein halbes Jahr dort Werkstudent war und schon erste positive Eindrücke vom Arbeitsalltag wahrgenommen hatte.

Der Praktikumsbericht soll einen Überblick über den Ablauf, Inhalt, Aufgaben und Erfahrungen meines Praktikums beim Testcenter geben. Am Anfang des Berichts wird das Unternehmen vorgestellt, anschließend wird meine Praktikumsaufgabe erläutert. Hierbei werde ich über die verwendeten Technologien, auf mein Vorgehen und meine erreichten Ziele eingehen.

Abschließend werden meine persönlichen Erfahrung als auch die Bewertung meiner Leistung meines Praktikumsgebers wiedergegeben.

# Kapitel 2

# Unternehmensvorstellung

## 2.1 Das Unternehmen

Die Bosch Software Innovations GmbH, das Software- und Systemhaus der Bosch-Gruppe, konzipiert, entwickelt und betreibt weltweit innovative Software- und Systemlösungen. Die Aktivitäten gliedern sich in die vier Unternehmensbereiche eMobility Solutions, Industrial Technology, Consumer Goods sowie Energy and Building Technology.
Entstanden ist die Tochtergesellschaft, welche bis 2013 den Namen „Bosch Software Technology“ trug, nach Übernahme der „Innovations Software Technology GmbH“ im Jahr 2008. Diese wurde 1997 von sechs IT-Fachkräften gegründet. Im Jahr 2013 erfolgte die Umbenennung in „Bosch Software Innovations GmbH“ und die Übernahme der inubit AG, einem Softwareunternehmen in Berlin, welches Standardsoftware im Bereich Business Process Management (BPM) anbietet. Ein wichtiger Schwerpunkt bei „Bosch Software Innovations GmbH“ ist das „Internet of Things“. Mehr als 150 „Internet of Things“ Projekte hat das Unternehmen bereits absolviert.
Das Unternehmen ist heute mit mehr als 650 Mitarbeitern weltweit mit Standorten in Deutschland, Singapur, China, Bulgarien, Japan und den USA vertreten.

## 2.2 Unternehmenspolitik

Die Umgangsform innerhalb des Unternehmens fällt unmittelbar als sehr angenehm auf. So wurde beispielsweise vor einigen Jahren entschieden, dass sich alle Angestellten innerhalb des Unternehmens duzen. Dies ermöglicht bereits zu Beginn eine schnelle Integration in das Team und in Unternehmensereignisse. Auch werden Diskussionen auf diese Weise sehr offen gehalten und es fällt leicht, Anschluss zu finden.

Bei Bosch Software Innovations wird die agile Entwicklung in Form von Scrum gelebt. Dies ist ein verbreitetes Vorgehensmodell des Projekt- und Produktmanagements, welches speziell zur agilen Softwareentwicklung verwendet wird. Genutzt wird diese Methode, da Software-Projekte oftmals zu komplex sind, um sie von Beginn an in Gänze korrekt einschätzen zu können. Regelmäßige Termine, die zur Absprache dienen, schaffen innerhalb des

Teams Transparenz und einen guten Überblick über die (noch) zu erledigende Arbeit. Diese Entwicklungszyklen, auch „Sprints“ genannt, dauern bei Bosch SI in der Regel zwei Wochen.

Zu Beginn bzw. Ende eines jeden Sprints findet der so genannte „Sprintwechsel“ statt. Teil dieses Termins, der in der Regel einen ganzen Tag einnimmt, ist das „Sprint Review“, in dem die erzielten Ergebnisse von Teammitgliedern dem restlichen Team sowie allen relevanten Stakeholdern vorgestellt werden. Die „Retroperspektive“ dient dem Team zum Austausch von Gedanken und Meinungen über die aktuelle Stimmung im Team und den Verlauf des vorangegangenen Sprints. Hier können Verbesserungsvorschläge zur Arbeitsweise, zum Verhalten oder generelle Anregungen eingebracht werden. Das „Planning“ (bestehend aus „Planning 1“ und „Planning 2“, siehe spätere Erläuterung) dient dazu, die Aufgaben des jeweils folgenden Sprints festzulegen und zu beschreiben. Die Scrum-Methode beinhaltet verschiedene Rollen: Die Rolle des Product Owners, welcher den Kunden vertritt, des Scrum-Masters, der alle Termine leitet und führt und für die allgemeine Kommunikation untereinander verantwortlich ist, sowie des Entwicklerteams. Die stetige Kommunikation und Koordination zwischen den drei Parteien über die gesamte Entwicklungszeit hinweg führt zu einem flexiblen Produkt und einem Team, das gerne, schnell und zielorientiert zusammenarbeitet.

# Kapitel 3

# Tätigkeiten während des Praktikums

## 3.1 Zeitplan

Meine Arbeitswoche während des Praktikums umfasste 32 Stunden. Im Studiengang Technische Informatik an der Beuth Hochschule für Technik sind neben dem Pflichtpraktikum, zwei Wahlpflichtmodule zu belegen. Deshalb ist der Freitag für die zwei Wahlpflichtmodule an der Universität vorgesehen. Die Arbeitszeit kann sich jeder Mitarbeiter frei einteilen, die Zeiterfassung erfolgt auf Vertrauensbasis. Dafür gibt es ein Zeiterfassungssystem, in welches ich meine Arbeitszeiten eintragen konnte. Mein Arbeitstag begann um 9 Uhr morgens und endete nachmittags um 17:30 Uhr.

Für meine Praktikumsaufgabe konnte ich meinen Zeitplan selber einteilen, Fortschritte wurden über die wöchentlichen Meetings besprochen.

## 3.2 Vorgehensweise

Bevor ich mit dem Programmieren der Praktikumsaufgabe angefangen habe, stellte ich mir selber zwei Fragen:

1. Wie strukturiere ich die Praktikumsaufgabe am besten?
2. Was muss ich mir an Theorie aneignen, um diese Aufgabe lösen zu können?

Ich wusste zu diesem Zeitpunkt nicht, wie komplex die Aufgabe werden würde. Relativ schnell bemerkte ich, dass ich nicht mehr weiter wusste und fragte meinen Betreuer nach Hilfe. Er erarbeitete mit mir eine Struktur, die wie folgt aufgebaut ist:

1. Anforderungsanalyse
   (a) Was wird benötigt ?
   (b) Was muss beachtet werden ?
   (c) Zielgruppe der Anwendung ermitteln
   (d) Einarbeitung in bestehende Cloud-Dienste
2. Erarbeitung und Auswahl der zu verwendenden Technologien
3. Prototypische Umsetzung der Anforderung mittels ausgewählter Basistechnologie

Im nachfolgenden Text werde ich auf die einzelnen Themen der Strukturierung meines Betreuer eingehen.

## 3.3 Anforderungsanalyse

### 3.3.1 Cloud Test Manager

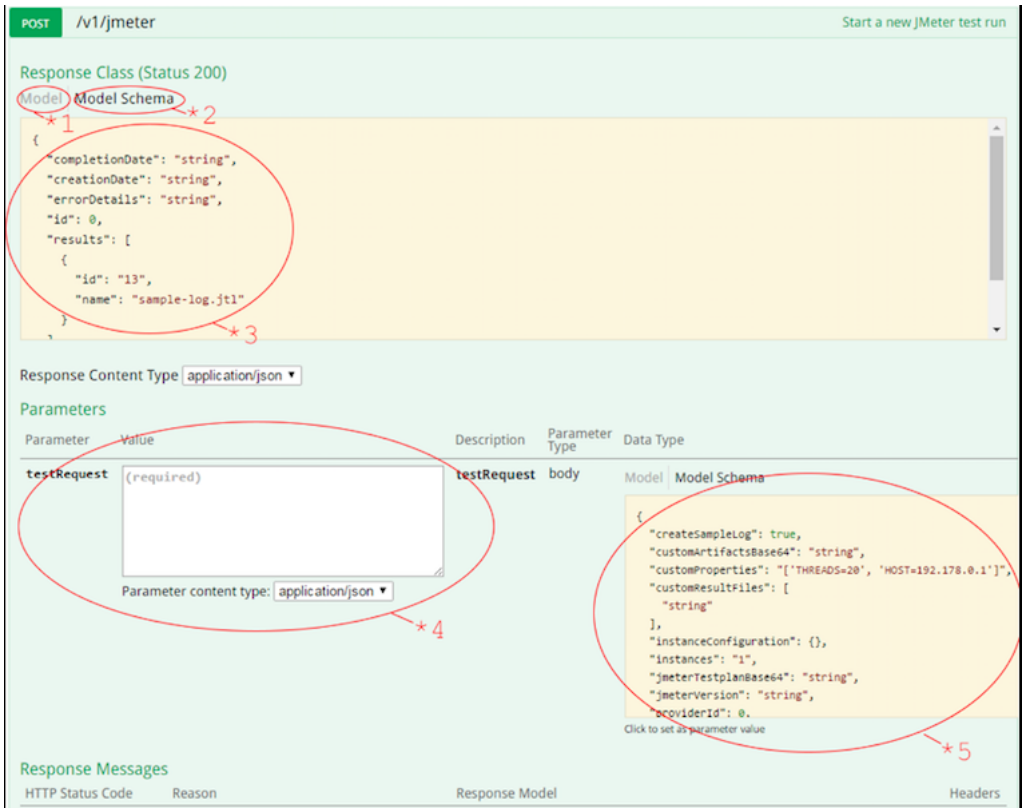
Nachdem die Strukturierung meiner Praktikumsaufgabe feststand, beschäftigte ich mich mit der Anforderungsanalyse. Meine Aufgabe war es eine graphische Oberfläche, auch „graphical user interface“ oder abgekürzt GUI genannt, für eine cloudbasierte Anwendung zu entwickeln. Eine Cloud stellt eine IT – Infrastruktur über das Internet zur Verfügung. Die GUI soll die Komplexität eines Service, namens JMeter abstrahieren. Das Testwerkzeug JMeter stammt von der Firma „Apache“ und ist vorrangig zum Ausführen von Lasttests in einer Client/Server Anwendung. Ein Service beschreibt eine autarke Einheit, die ihre Funktionalität über eine definierte Schnittstelle bereitstellt. Der JMeter Service ist einer von vielen Services, die zusammen den Cloud Test Manager bilden. Der Cloud Test Manager wurde von Mitarbeitern im Testcenter entwickelt und basiert auf einer Microservice Architektur. Eine Microservice Architektur teilt eine große komplexe Anwendung in viele kleine, dabei sind die Anwendungen weitestgehend entkoppelt, d.h. sie sind unabhängig voneinander. Jede Anwendung übernimmt eine Funktionalität und bietet diese Funktionalität über eine Schnittstelle an. Die Services vom Cloud Test Manager kommunizieren via REST. Das Architekturmodell REST besitzt unterschiedliche Komponenten eines verteilten Systems, die über das HTTP-Protokoll miteinander kommunizieren. Der Cloud Test Manager ist in zwei Ebenen unterteilt: Einer oberen Ebene namens „Top Level Services“ und unteren Ebene namens „Base Level Services“. Dabei übernehmen die Base Level Services die interne Funktionalität des Cloud Test Manager und bilden so die Basis. Die Top Level Services, zu welcher auch der JMeter Service gehört, sind über eine URL für den User erreichbar und leiten, abhängig von der Konfiguration, die Daten via REST an die Base Level Services weiter. Eine URL identifiziert eine Ressource und den Ort im Computernetzwerk.

### 3.3.2 API Gateway

Ein Mitarbeiter im Testcenter hat ein API Gateway entwickelt, das via einer REST Schnittstelle mit anderen Services kommuniziert. Dafür sendet der User einen REST – Request an das API Gateway und dieser routet es an den zuständigen Service. Das bedeutet nicht, dass die einzelnen Services nur über das API Gateway miteinander kommunizieren. Das API Gateway ist ein typisches Entwurfsmuster einer Microservice basierten Architektur. Sie fungieren als einziger Kontaktpunkt zwischen Clients und Backendservices. Außerdem werden sie zur einfachen Lastenverteilung genutzt und steuern welche Services für die „Außenwelt“ sichtbar sind.

### 3.3.3 REST – API Dokumentation mittels SwaggerUI

Um herauszufinden welche REST Schnittstellen der JMeter Service besitzt, hat ein Mitarbeiter im Testcenter eine API Dokumentation mit dem Tool SwaggerUI erstellt. Dieses Tool visualisiert und interagiert mit den Ressourcen der API. Mit der Visualisierung der Dokumentation lassen sich Backend Implementierungen leicht realisieren. Als Beispiel reagiert das Tool bei einem Fehlerfall mit der Farbe rot und beim Erfolgsfall mit der Farbe grün. Die Dokumentation der REST – API vom JMeter Service ist technisch orientiert und ohne Vorkenntnisse schwer nutzbar. Um zu demonstrieren, dass die Dokumentation für einen User ohne technische Kenntnisse schwer nutzbar ist, habe ich in der unteren Abbildung den Fall ausgewählt, wo der User einen Test ausführt.



1. Antwortformat des Servers
2. Die Erklärung zu den Parameter des Antwortformats.
3. Hier werden die einzelnen Parameter mit Datentyp der Serverantwort gezeigt.
4. Das Eingabefeld des Requests, der einen Test konfiguriert. Welche Parameter benötigt werden, ist im Punkt *5 zu sehen.
5. Die Parameter mit Datentyp für einen Request.

### 3.3.4 UX - Team

Um das Benutzen des JMeter Service zu vereinfachen, soll die GUI einfach und minimalistisch aufgebaut werden. Ich wusste, dass ich zu der Thematik eine Expertenmeinung brauche. Daraufhin bin ich zu meinem Betreuer gegangen und dieser schickte mich zu einer Mitarbeiterin vom User - Experience – Team oder abgekürzt UX - Team. Sie fand die Idee gut, nachdem ich erklärte, was meine Praktikumsaufgabe war. Wir haben daraufhin ein initiales Meeting vereinbart, in welchem ich meine Praktikumsaufgabe nochmal im Detail erklärte. Im initialen Meeting habe ich jedoch schnell bemerkt, dass hier zwei Sichten die Aufgabe betrachteten. Als Basis für die GUI habe ich die REST – API Dokumentation von SwaggerUI genommen. Als ich das Test Szenario erklärte, kamen bei ihr viele Fragen auf, die für mich selbsterklärend waren.
Aufgrund der Verständnisprobleme haben wir weitere Meetings vereinbart. Um zwischen den Meetings, welche in einem Wochenabstand stattfanden, kommunizieren zu können, haben wir ein Portal namens Realtimeboard benutzt. In diesem Portal ist es möglich grafische Oberflächen zu erstellen und zu beschriften. Dort habe ich einige Entwürfe entwickelt und die Parameter verständlicher erklärt, die im initialen Meeting für Fragen sorgten. Die technischen Details abstrahiert in einer leicht verständlichen graphischen Oberfläche darzustellen, stellte sich als große Herausforderung dar. In den Meetings habe ich meine Entwürfe vorgestellt und erklärt. Allerdings kamen immer neue Fragen, hinsichtlich der Bedeutung und Funktionalität eines Parameters. Ich wusste des öfteren keine Antwort zu ihren Fragen, weil es für mich logisch erschien und ich nichts anderes rein interpretieren konnte. Ihr GUI Aufbau war eine Art Dashboard. Ein Dashboard ist eine Technik mit welcher Informationen/Daten übersichtlich auf einer Fläche dargestellt werden.
Ich wollte den Aufbau meiner GUI nach dem Google Prinzip, minimalistisch halten. Allerdings war der Aufbau meiner GUI für sie nicht nachvollziehbar, weil es zu technisch orientiert war. Basierend auf der SwaggerUI Dokumentation konnten einzelne Tests bzw. mehrere Tests ausgeführt werden. Ich konnte es nur technisch basiert erklären und somit wurde es schwierig auf einen gemeinsamen Nenner zu kommen. Um es verständlicher zu erklären, habe ich meine Kollegen um Hilfe gebeten. Zusammen ist es uns gelungen,

alle Fragen verständlich zu erklären. Auf Basis dieser entstand ein GUI Design, das sowohl einfach benutzbar, minimalistisch und technisch anspruchsvoll ist. Ich bemerkte wie schwer es mir fiel, technisch basierte Dinge einer Person zu erklären, die nicht mit der Thematik vertraut war. Im Laufe des Praktikums kamen immer wieder Verbesserungsvorschläge, die wir im Portal Realtimeboard umsetzten. Somit entstanden sogar in der Implementierungsphase noch Änderungen am Design. Um die GUI „Bosch – konform" zu gestalten, gab es ein Corporate Design, auf welches ich später noch eingehen werde.

## 3.4 Technologieauswahl

Die Technologieauswahl für meine Praktikumsaufgabe war mir überlassen. Im nachfolgenden Text möchte ich meine Technologieoptionen benennen und erklären warum ich mich dafür entschieden habe.

## 3.5 Clientseitige Technologie

Damit der User meine GUI benutzen kann, soll die als Webanwendung fungieren. Eine Webanwendung ist ein Anwendungsprogramm, die in einer Client – Server – Architektur läuft. Eine Client – Server – Architektur beschreibt eine Möglichkeit, Aufgaben und Dienstleistung innerhalb eines Netzwerkes zu verteilen. Ich habe die Entwicklung in drei Bereiche: Aufbau, Aussehen und Funktionalität eingeteilt. Für alle drei Bereiche wurden verschiedene Technologien benutzt. Der Aufbau wurde mit HTML5 und das Aussehen mit CSS3 realisiert. Beide waren auf dem aktuellen Versionsstand. Für die Funktionalität meiner GUI standen mehrere Technologien zur Auswahl, die ich im nachfolgenden Text benennen möchte. Dabei werde ich auf die Vor - und Nachteile jeder Technologie eingehen. Abschließend werde ich im Fazit meine Auswahl begründen.

## 3.6 Ext JS

Ext JS ist ein clientseitiges Javascript Framework für interaktive Webanwendungen. Es bietet eine umfangreiche Sammlung vorprogrammierter Elemente. Ursprünglich war Ext JS eine Sammlung von Funktionen für Yahoo User Interface Library. Wegen steigender Beliebtheit entwickelte sich die unabhängige Bibliothek Ext JS daraus.

### 3.6.1 Vorteile

Ich habe schon in meiner Werkstudentenzeit mit diesem Framework gearbeitet und wusste, auf was zu achten war. Eine große Community und einige Mitarbeiter im Testcenter konnten bei Problemen helfen. Es wurde für

„Rich Internet Applications“ entwickelt, dass bedeutet vielseitige Interaktionsmöglichkeiten mit der Benutzeroberfläche. Es bietet noch eine Vielzahl von vorprogrammierten, modernen Elementen, die sowohl für Desktop- als auch für mobile Anwendungen optimiert sind. Ext JS ist plattformunabhängig und bietet eine moderne Model – View – Controller Architektur. Eine Model – View -Controller Architektur teilt eine Software in drei Komponenten namens Datenmodell(Model), Präsentation(View) und Programmsteuerung(Controller) auf. Mit der Architektur ist ein flexibler Programmentwurf möglich, der eine spätere Änderung oder Erweiterung vereinfacht und die Wiederverwertung von Komponenten ermöglicht.

### 3.6.2 Nachteile

Für die Benutzung von Ext JS ist eine kostenpflichtige Lizenz nötig. Es gab auch keine aktuelle Lizenz, die benutzt werden konnte. Weiterhin ist die Fehlersuche aufwendig, wenn viele vorprogrammierte Elemente benutzt werden.

## 3.7 Spring MVC

Spring MVC ist eine Erweiterung des Spring Frameworks. Das Framework ist ein OpenSource Applikationsframework und basiert auf Java. Als OpenSource wird eine Software bezeichnet, deren Quelltext öffentlich eingesehen werden kann. In Kombination mit Javascript lassen sich dynamische und REST – konforme Webapplikationen umsetzen.

### 3.7.1 Vorteile

Viele Anwendungen im Testcenter wurden bereits mit dem Framework implementiert, somit ist das Know – How bereits vorhanden. Es bietet eine übersichtlichte Model – View -Controller Architektur. Durch das Dependency Injection Entwurfsmuster lassen sich Abhängigkeiten zwischen Objekten einfach realisieren. Dependency Injection bedeutet das Abhängigkeiten eines Objektes oder einer Klasse zur Laufzeit übergeben werden. Das Testen des Codes ist einfach, weil Abhängigkeiten in Unit – und Integrationstests sauber definiert werden können, so dass einzelne Module sowohl in Isolation aber auch im Zusammenspiel getestet werden können. Durch die Benutzung von Spring Komponenten bleibt der Code leserlich und übersichtlich.

### 3.7.2 Nachteile

Die Community von SpringMVC ist im Verhältnis zu anderen Frameworks klein, was die Suche nach bestimmten Fehlern erschwert. Die Konfiguration ist aufwendig, was zur schnellen Unübersichtlichkeit führt. Die Dokumentation ist nicht ausführlich beschrieben, so dass Informationen im Internet gesucht werden müssen.

## 3.8 Angular JS

Angular JS ist ein clientseitiges Javascript Framework für Single Page Application Webanwendungen. Als Single Page Application wird eine Webanwendung bezeichnet, die aus einem HTML Dokument besteht, über das die Inhalte dynamisch nachgeladen werden. Das Framework wurde als Open-Source Projekt von der Firma Google entwickelt.

### 3.8.1 Vorteile

Das Framework besitzt eine große Community und eine ständige Weiterentwicklung durch Google. Das Entwurfsmuster bei diesem Framework ist Dependecy Injection. Das Testen des Programmcodes steht bei diesem Framework im Vordergrund. So sind Abhängigkeiten im Test leicht definierbar und Antworten vom Server werden mit einem Mock – Objekt ersetzt. Ein Mock Objekt ist eine Attrappe eines echten Objekts. Sie implementieren die Schnittstellen über die das zu testende Objekt auf seine Umgebung zugreift. Sie liefern keine Echtdaten zurück, sondern festgelegte Werte. Durch den Architekturstil Model – View – Controller sind viele Teile des Codes wiederverwendbar.

### 3.8.2 Nachteile

Alte Browserversionen unterstützen viele Funktionen von Angular JS nicht. Die Dokumentation von vielen Funktionen ist unvollständig und es gibt kaum Beispiele. Das Framework besitzt eine schlechte Skalierbarkeit. Das bedeutet bei wachsenden Projekten reichen die anfänglichen Anforderungen nicht mehr aus. Es ist notwendig die Programmierung auf Grundlage der neuen Anforderungen zu erstellen.

## 3.9 Fazit

Kein Framework ist perfekt, jedes Framework besitzt seine Vor – und Nachteile. Ich wollte ein Framework nutzen, dass am besten zu der Aufgabe passt und eine Zukunftsperspektive hat. Ext JS habe ich nicht benutzt, obwohl Vorkenntnisse durch die Werkstudentenzeit vorhanden waren. Der Grund war die fehlende Lizenz und die hohen Kosten zur Beschaffung der Lizenz. Somit war die Entscheidung zwischen Angular JS und Spring MVC. Zu Spring MVC habe ich Beispielprojekte und Tutorials angeguckt, die ich nur mit viel Einarbeitung nachvollziehen konnte. Der ausschlaggebende Punkt war die unfertige Dokumentation des Frameworks. Selbst mit dem Know How im Team war es schwierig bei Fehlerfällen den richtigen Lösungsansatz zu finden. Schlussendlich habe ich mich für Angular JS entschieden, da eine große Community vorherrscht und die Weiterentwicklung des Framework, durch Google garantiert wird.

## 3.10 Entwicklungsumgebung

Nachdem ich eine clientseitige Technologie ausgewählt habe, kann ich auf Basis dieser, eine Entwicklungsumgebung auswählen. Im nachfolgenden Text stelle ich drei Entwicklungsumgebungen vor. Anschließend im Fazit, werde ich begründen, welche Entwicklungsumgebung ich für meine Praktikumsaufgabe genutzt habe.

## 3.11 Intelli J

Diese IDE gehört dem Softwareunternehmen JetBrains und wurde für die Programmiersprache Java entwickelt. Es ist durch eine Anzahl von nützlichen Plugins erweiterbar. Bei Javascript Projekten wird diese IDE von den meisten Usern empfohlen. Auch ein Mitarbeiter im Testcenter benutzt die IDE für Javascript Projekte.

## 3.12 Netbeans

Diese IDE ist ein OpenSource Projekt, das als Plattform für eigene Anwendungen fungieren kann. Es besitzt eine große Community, die ständig wächst. Es wurde ursprünglich für die Programmiersprache Java entwickelt, unterstützt jedoch auch C, C++.

## 3.13 Notepad++

Der freie Texteditor Notepad++ ist für Windows Betriebssysteme erhältlich. Durch die Unterstützung von ASCII und verschiedene Unicode – Kodierungen, können fremdsprachige Textdateien verfasst werden. Für viele Programmiersprachen werden Syntax und Struktur durch Code - Faltungen hervorgehoben. Code – Faltung bezeichnet eine Funktion in Editoren um Klassen / Methoden in sogenannte Falten zu gruppieren. Diese Falten werden durch einen Mausklick auf – und zuklappt.

## 3.14 Fazit

Ich wollte ursprünglich IntelliJ für meine Praktikumsaufgabe nutzen, aber die IDE gab es nur in einer 30 tägigen Version kostenlos zum Download, danach waren Gebühren für die Benutzung fällig. Somit habe ich mich vorerst für Netbeans entschieden, weil ein Mitarbeiter im Testcenter mir bei der Installation und Konfiguration geholfen hat. Allerdings bemerkte ich schnell, dass ich die IDE nicht für meine Praktikumsaufgabe nutzen werde. Grund dafür war die aufwendige Konfiguration, um eine eigene Projektstruktur zu erstellen, weil ich mich an die Bosch Struktur für Webapplikationen halten wollte. Somit ist es letztendlich Notepad++ geworden. Mit dem Editor ist

es einfach eine eigene Projektstruktur zu erstellen und übersichtlichen Code, aufgrund der Code-Faltung, zu schreiben.

## 3.15 Prototypische Umsetzung

### 3.15.1 Corporate Design

Um eine Webseite „Bosch-konform“ zu gestalten, existiert ein Git Repository. Git ist ein verteiltes Versionsverwaltungssystem von Dateien. Dieses Repository enthält CSS – Klassen und Angular JS Direktiven für das Corporate Design von Bosch. Damit ich das Corporate Design nutzen konnte, schickte mich ein Mitarbeiter vom UX – Team zu einem Entwickler. Es bedarf großen Konfigurationsaufwands, um an das Design zu kommen. Zusammen mit dem Entwickler, konfigurierten wir die nötigen Schritte für das Repository. Es war ein mühseliger Vorgang, weil für alle Schritte bestimmte Rechte benötigt wurden. Diese Rechte bekamen wir durch unzählige Telefonate.

### 3.15.2 Versionsverwaltungssystem SVN

Das gesamte Projekt wurde im Versionsverwaltungssystem SVN versioniert. Dabei war auf die Struktur zu achten. Um die Strukturierung einzuhalten, richtete ich mich nach anderen Projekten im SVN und habe dementsprechend alles angepasst.

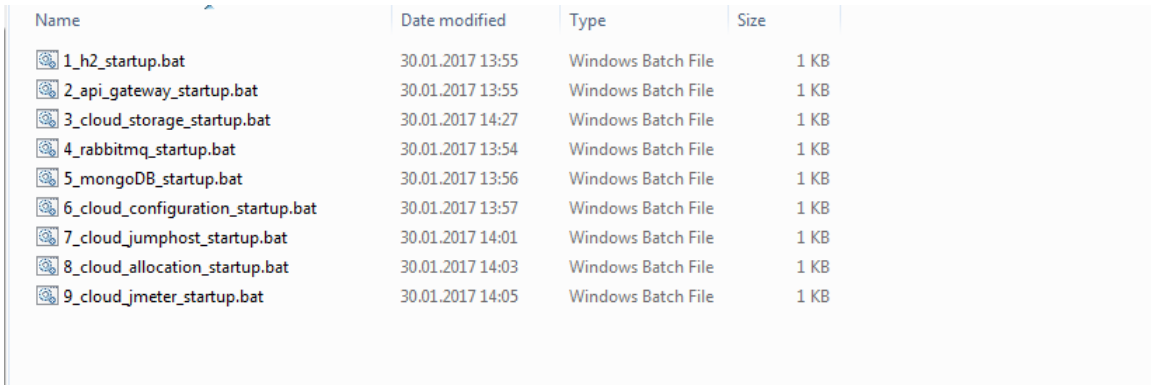
### 3.15.3 Apache Tomcat

Damit meine GUI externe Daten verarbeiten und darstellen kann, benötigte ich einen Webserver. Ich entschied mich für Tomcat, weil im Team bereits damit gearbeitet wurde und die Konfiguration schnell ging. Bei der Konfiguration war auf den Port zu achten, da die Microservices einige bereits benutzten. Damit meine GUI als Webanwendung vom Tomcat ausgeführt wird, komprimierte ich meine Projektdaten in eine ZIP Datei und änderte die Endung der Datei in „.war“. Eine ZIP Datei ist ein Format in der mehrere Dateien zusammengefasst werden und eine „.war“ Datei beschreibt wie eine Webanwendung in eine ZIP bzw. JAR Datei verpackt wird. Nachdem ich die Datei komprimiert und umbenannt habe, verschob ich sie in den „webapps“ Ordner vom Tomcat. Dieser entpackt mir die „.war“ Datei, woraufhin ich den Tomcat startete. Unter der eingestellten URL konnte ich nun meine GUI sehen.

### 3.15.4 Lokale Einrichtung der Microservices vom Cloud Test Manager

Wie bereits beschrieben, basiert der Cloud Test Manager auf einer Microservice Architektur. Um die benötigten Microservices vom JMeter Service lokal zu nutzen, konfigurierte ein Mitarbeiter im Testcenter mit mir die nötigen

Schritte. Dazu haben wir einen Ordner namens „cloud“ vom SVN ausgecheckt. In diesem Ordner befanden sich alle Microservices des Cloud Test Manager. Die Microservices sind mit dem Framework Spring implementiert und die Ausführung erfolgt mit dem Tool Maven. Maven ist ein Build – Management – Tool, d.h. ein fertiges Anwendungsprogramm wird automatisch erzeugt, von Apache und basiert auf Java. Um einen Microservice zu starten, navigierte ich mich zum REST Ordner des jeweiligen Services in der Shell und führte das Kommando: „mvn spring-boot:run“ aus. Hatte ein Mitarbeiter im Testcenter den Code eines Services bearbeitet und im SVN hochgeladen, musste ich vorher den lokalen „cloud“ Ordner updaten. Damit das Tool Maven die Änderung beachtet, führte ich im Ordner des geänderten Services den Befehl „mvn install -DskipTests“ aus. Somit baut er den Service mit den Änderungen nochmal. Manche Services benötigen eine Datenbank, die ein Mitarbeiter und ich ebenfalls lokal konfigurierten. Damit die Services untereinander kommunizieren, konfigurierten wir lokal den Message Broker RabbitMQ. Ein Message Broker ist eine Zwischenanwendung die eine synchrone oder asynchrone Kommunikation ermöglicht. Die Datenbanken sollten immer vor den Services gestartet werden. Ich habe Shell – Skripte geschrieben (siehe Abbildung), um das Starten der Datenbanken und Services zu beschleunigen. Nachdem die Konfigurationen fertig waren und alle Services liefen, fing ich mit dem Programmieren an.



| Name | Date modified | Type | Size |
|---|---|---|---|
| 1_h2_startup.bat | 30.01.2017 13:55 | Windows Batch File | 1 KB |
| 2_api_gateway_startup.bat | 30.01.2017 13:55 | Windows Batch File | 1 KB |
| 3_cloud_storage_startup.bat | 30.01.2017 14:27 | Windows Batch File | 1 KB |
| 4_rabbitmq_startup.bat | 30.01.2017 13:54 | Windows Batch File | 1 KB |
| 5_mongoDB_startup.bat | 30.01.2017 13:56 | Windows Batch File | 1 KB |
| 6_cloud_configuration_startup.bat | 30.01.2017 13:57 | Windows Batch File | 1 KB |
| 7_cloud_jumphost_startup.bat | 30.01.2017 14:01 | Windows Batch File | 1 KB |
| 8_cloud_allocation_startup.bat | 30.01.2017 14:03 | Windows Batch File | 1 KB |
| 9_cloud_jmeter_startup.bat | 30.01.2017 14:05 | Windows Batch File | 1 KB |

## 3.16 GUI Programmierung mit ausgewählten Basistechnologien

Um die Aufgabe mit Angular JS zu implementieren, habe ich mir zuerst die Grundlagen erarbeitet. Durch unzählige Tutorials im Internet, verlief der Einstieg in das Framework ohne große Probleme.

### 3.16.1 App.js

Wie in Angular JS üblich, begann ich mit der sogenannten „app.js“. In der Datei werden alle Module mit ihren Abhängigkeiten deklariert und das Routing konfiguriert, das bedeutet abhängig von der URL werden bestimmte

Dateien aufgerufen. Damit nicht jeder auf wichtige Dateien gelangt, müssen Mechanismen bzw. Funktionalitäten implementiert werden, die das Routen nur unter einer bestimmten Bedingung zulassen. Durch das Dependency Injection Entwurfsmuster und vorgefertigte Module von Angular JS, lassen sich bestimmte Funktionalitäten einfach implementieren. Als Beispiel ist das Routen mit dem Modul „$routeProvider“ sehr einfach zu realisieren. Dabei war lediglich auf die Syntax zu achten, die in der Dokumentation beschrieben wurde.

### 3.16.2 Login GUI

Damit nicht jeder den JMeter Service nutzen kann, ist eine Sicherheitsvorkehrung nötig. Die Sicherheit wird mit einer Login GUI abstrahiert. Somit können nur Benutzer mit einem gültigen Account, den Service nutzen.

Um eine Login GUI zu implementieren, kreierte ich das Modul Login nach dem Architekturstil Model – View – Controller – Service. Die Methoden, um die gewünschte Funktionalität zu implementieren, sind im Service - Teil deklariert und definiert, damit erreichte ich eine klare Trennung von Logik und Funktion. Dadurch wurde der ganze Code übersichtlicher und verständlicher.

Im Controller stellte ich die Abhängigkeit zu den Methoden im Service, mittels Dependency Injection her. Um die Methoden vom Service zu nutzen, verwendete ich „Factory“ Funktionen. Diese Funktionen werden durch Dependency Injection erzeugt und sind im gesamten Modul benutzbar. In Angular JS existiert eine sogenannte „two-way-data-binding“, d.h. wenn die Daten im Model sich ändern, ändern sich auch die Daten in der View und umgekehrt. Somit besitzt die View immer die aktuellen Daten des Model und umgekehrt. Das Objekt „$scope“ beschreibt den gemeinsamen Datenbereich von View und Controller.

Damit der User sich einloggen kann, implementierte ich ein Login - Formular mit HTML. Um Funktionalität in das Formular zu bekommen, fügte ich Angular Direktiven hinzu. Eine Direktive ist „ng-submit=login()“, das bedeutet die Funktion login wird aufgerufen, wenn der User auf den „Send“ Knopf klickt. Im Controller weise ich der Variable „$scope.login“, die Login Methode vom Service zu. Somit wurden die Daten vom Controller an die View weitergeleitet.

Der Server erwartet Benutzername und Passwort vom User, in „Base64“ codierter Form. Die Kodierung „Base64“ wandelt 8 – Bit – Binärdaten in eine Zeichenfolge von ASCII – Zeichen um. Ich nutzte eine vorgefertigte „Base64“ Funktion von einem anderen Projekt. Damit der Server den User authentifiziert, wird der Header Eintrag „Basic“ gefolgt von den „Base64“ codierten Daten überprüft. Hier stieß ich auf meinen ersten Fehler, der mich mehrere

Tage beschäftigte. Der Server antwortete mit dem Fehler: „401 – Unauthorized“. Nach langer Fehlersuche, stellte ich fest, dass ich ein Leerzeichen im Header zwischen „Basic“ und den „Base64“ codierten Daten vergessen habe. Durch den Flüchtigkeitsfehler wurde ich vorsichtiger und überprüfte meinen Code sorgsamer. Nachdem der Header richtig gesetzt wurde und der Server die erwartete Antwort zurückgab, passierte nichts in der View.

Nach dem erfolgreichen einloggen, sollte der User zu der Home GUI gelangen. Durch einen Tipp von einem Mitarbeiter, konnte ich den Fehler finden. Das Problem lag an der Verarbeitung der Daten, die vom Server kamen. Durch eine falsche Bedingung, wurde der Programmcode bei einer erfolgreichen Serverantwort nie ausgeführt. Um auf die Serverantwort richtig zu reagieren, nutzte ich ein „Promise“ Objekt. Dieses Objekt speichert die Antwort des Servers und überprüft mit zwei Funktionen, ob die Antwort erfolgreich oder gescheitert ist.

Schlussendlich klappte der Login und der User wurde zu der Home GUI geroutet. Ist der User auf der Home GUI und lädt die Seite neu, soll er nicht wieder auf die Login GUI geroutet werden. Um dies zu verhindern, packte ich die „Base64“ Daten in Cookies, die bei jedem neu laden überprüft werden. Ein Cookie ist eine Textdatei auf einem Computer / in einem Browser.

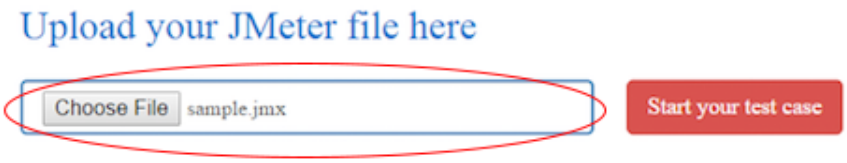
## 3.17 Home GUI

In der GUI führt der User einen JMeter Test aus. Als Basis nutzte ich die Dokumentation von SwaggerUI. Der Architekturstil war ebenfalls Model – View – Controller – Service. Um das Szenario zu abstrahieren, half mir dieses Use – Case:

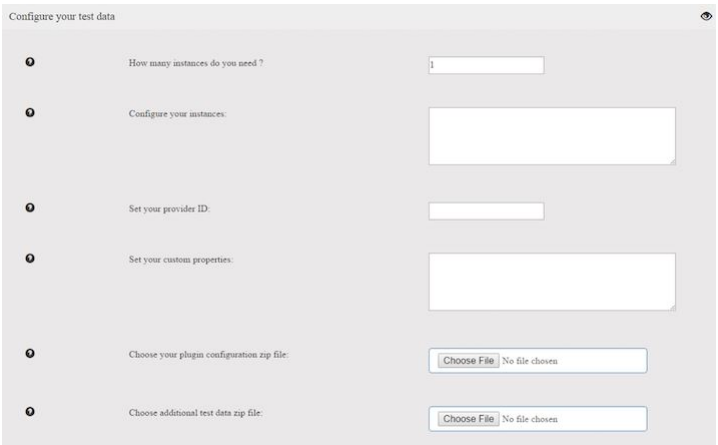
1. User lädt einen JMeter Testplan hoch
2. Daraufhin öffnet sich ein Dropdown Tabelle, in welches der User die Konfiguration der Parameter vornimmt.
3. Der User schickt den Testfall mit der Konfiguration zum Server.
4. Während er auf die Testausführung wartet, soll eine Dropdown Tabelle den User benachrichtigen, im welchen Status sich der Test befindet.
5. Ist der Test ohne Fehler durchgelaufen erscheint eine Dropdown Tabelle mit den Log Dateien, die durch das Anklicken gedownloadet werden. Eine Log Datei ist ein Protokoll aller Aktionen von Prozessen eines Computersystem.
6. Ist der Test fehlgeschlagen, soll der Fehler in einer Dropdown Tabelle angezeigt werden.

Ich werde auf jeden Punkt eingehen und erklären wie ich das mit den Basistechnologien gelöst habe.

Zu 1) Ich erstellte ein Upload Feld mit HTML5 und fügte die Direktive „ng-Model“ mit einem passenden Namen ein. Damit konnte ich die Eingabedaten vom Upload Feld an den Controller übermitteln. Laut der Dokumentation akzeptiert der Server den JMeter Testplan als Datentyp String. Ich benötigte eine Funktion, die Dateien jeglicher Art in einen „Base64“ String umwandelt. Nach einer Recherche im Internet fand ich eine Angular JS Direktive von einem Entwickler, die genau das macht. Er hat seinen Angular Code in seinem Git Repository veröffentlicht. Diese Direktive hat mir viel Zeit erspart und war einfach zu benutzen. Ich habe lediglich in dem Upload Feld die Direktive eingefügt und dieser wandelte mir die Datei, in einen „Base64“ String.

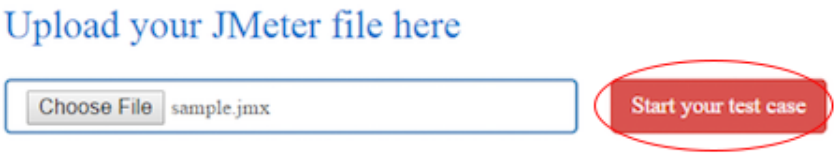



Zu 2) Das Grundgerüst der Dropdown Tabelle mit den entsprechenden Eingabefeldern für die Parameter wurden mit HTML5 implementiert. Was für ein Eingabefeld für die Parameter benutzt wurde, war abhängig vom Datentyp. Diese habe ich ebenfalls auf Basis der Dokumentation gemacht. Die Direktive „ng-Model“ mit entsprechenden Namen wurde ebenfalls bei jedem Eingabefeld hinzugefügt. Damit der User wusste, was der Parameter einstellt, erstellte ich ein Icon links neben dem Eingabefeld. Geht der User mit der Maus über das Icon erscheint eine Beschreibung zum jeweiligen Parameter.



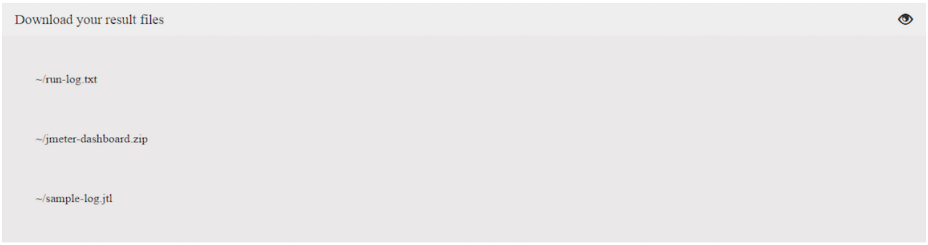

Zu 3) Ein Button mit der Direktive „ng-submit“ habe ich mit HTML5 dafür erstellt. Im Service habe ich eine Factory Funktion für die upload Funktion geschrieben. Da die Prozedur ähnlich wie bei der Login GUI war, wusste ich worauf ich achten musste. Ich erstellte ein Javascript Object Notation Objekt, abgekürzt JSON, nach der Form der Dokumentation von SwaggerUI. Da jede Testausführung Geld kostete, gab mir ein Mitarbeiter im Testcenter ein

Mock - Objekt, der alle Anfragen annimmt und in eine Datei schreibt. Nachdem ich einen Test ausgeführt habe, schaute ich in die Datei, um den Request zu überprüfen. Der Request entsprach der Form der Dokumentation. Damit konnte ich, gegen die Services des Cloud Test Managers testen.




Zu 4) Damit die Dropdown Tabelle in der View immer den aktuellen Zustand des Tests anzeigt, definierte ich eine „$scope.array“ Variable. Diese Variable wurde alle zwanzig Sekunden, mit dem aktuellen Testzustand aktualisiert. Damit ich den Zustand des Test abfragen konnte, benötigte ich die ID. Diese ID bekam ich in der Antwort des Servers, nachdem die Testausführung abgeschickt wurde. Mit der ID war es möglich den aktuellen Status des Test abzufragen. Ich implementierte einen Timer, der alle zwanzig Sekunden den Status abfragt. Dieser Timer stoppte nur, wenn der Zustand „DONE“ oder „ERROR“ auftrat.

Zu 5) Ist der Test erfolgreich abgeschlossen, waren die Log Dateien in der Response Nachricht des Servers. Diese speicherte ich mir in eine „$scope“ Variable, damit ich die View updaten konnte. Um eine Log - Datei zu downloaden, benötigte ich die ID des Tests und die ID der Log Datei. Die ID der Log - Datei bekam ich von der Response Nachricht. Nachdem die zwei ID's vorhanden waren, konnte ich die Log - Dateien downloaden. Bei den Text Log - Dateien ging das downloaden ohne Probleme, aber bei einer ZIP Log - Datei, war der Download korrupt. Um die ZIP Log - Datei runterzuladen, setzte ich beim Request den Datentyp manuell auf „application/octet-stream“. Bis ich das rausgefunden habe, vergingen einige Tage. Im Internet haben viele Leute das gleiche Problem, so dass viele Lösungsvorschläge vorhanden waren.




Zu 6) Damit der User bei einem Fehlerfall benachrichtigt wird, soll der Fehler in der View angezeigt werden. Es gab zwei Fehlerfälle zu beachten, einmal bei der Testausführung und beim Senden des Requests. Beide Fehlerfälle simulierte ich und konnte basierend auf der Response Nachricht, die Daten entsprechend in der View anzeigen. Bei jedem Request habe ich eine Funktion für den Fehlerfall implementiert. Diese Funktion gibt in der Konsole den jeweiligen HTTP – Statuscode und die Stelle, wo der Fehler passiert ist, aus.




Damit ich alle Funktionalitäten des JMeter Service implementierte, bedarf es eine Menge an Geduld und Recherche. Ich habe 60 Tests ausgeführt um den ersten Prototypen zu entwickeln.

## 3.18 Testtool

### 3.18.1 Jasmine

Jasmine ist eine freie Modultest – Bibliothek für Javascript.

Ich habe für meine Module Login und Home, jeweils die Service Methoden mit Jasmine getestet. In Jasmine werden Testszenarien mit „describe“ und eine Testfall mit „it“ beschrieben. Die Syntax ist selbsterklärend und einfach zu benutzen. Um meine Service Methoden zu testen, benötigte ich ein Backend,

das Antworten gibt. Dieses konnte ich mit dem Objekt „$httpBackend“ mocken.

Damit die Tests von Jasmine ausgeführt werden, wird ein Webserver benötigt. Auf der Angular JS Seite wird Karma empfohlen. Durch Tutorials im Internet konfigurierte ich Karma und erweiterte den Webserver mit einem „Code Coverage Tool“. Ein „Code Coverage Tool“ zeigt den Programmcode an, der während der automatisierten Tests ausgeführt werden. In meiner GUI sind ca. 60% des Programmcodes durch die Tests abgedeckt.

# Kapitel 4

# Fazit

Nach 6 Monaten im Testcenter von Bosch Software Innovation kann ich sagen, dass ich mit meinem Praktikum sehr zufrieden bin. Ich hätte nicht gedacht, dass ich in dieser kurzen Zeit so viel lernen werde. Am Anfang des Praktikums war mir nicht klar, wie ich so eine Aufgabe lösen sollte. Doch meine Kollegen halfen mir, wenn ich nicht mehr weiter wusste. Sie nahmen sich immer die Zeit, um meine Fragen verständlich zu beantworten, egal wie trivial sie waren.

Durch die Praktikumsaufgabe durfte ich eine neue Technologie kennenlernen. Ich wollte diese Technologie bereits privat lernen, aber kam aufgrund meines Zeitmanagements, nie dazu. Mit der Praktikumsaufgabe hatte ich die Chance, in einem professionellen Umfeld dies zu tun. Ich probierte meine Praktikumsaufgabe selbstständig zu lösen, weil ich geglaubt habe, dass der Lernfaktor dort am größten war. Doch verzweifelte ich am Anfang, an einfachen Fehlerfällen, so dass ich Hilfe bei meinen Kollegen gesucht habe. Sie gaben mir nie die Antworten, sondern Denkanstöße für den gegebenen Fehlerfall. Durch Geduld und Hartnäckigkeit, konnte ich mich in die neue Technologie einarbeiten und verstand das Konzept dahinter. Nachdem ich das Konzept verstanden habe und die Szenarien abstrahieren konnte, stieg meine Motivation.

Durch die Zusammenarbeit mit anderen Abteilungen lernte ich Dinge aus einem anderen Blickwinkel zu betrachten. So habe ich gelernt, technisch basierte Dinge, auf eine verständliche Art und Weise zu erklären. Mit dem erworbenen Wissen der Universität, konnte ich Relationen zu verwendeten Begriffen und Programmierparadigmen herstellen.

Da ich schon vorher als Werkstudent gearbeitet hatte, verstand ich mich mit allen Mitarbeiter. Die Stimmung im Team ist sehr angenehm und locker. Bosch sorgte regelmäßig für Obst, Gemüse, Süßigkeiten und kostenlosen Getränke, was den Arbeitsaufenthalt sehr angenehm gestaltet. Die Aufgaben decken sich mit meinem späteren Berufsleben. Ich kann jedem ein Praktikum beim Testcenter vom Bosch – SI empfehlen. Durch die Tätigkeit als Werkstudent und Praktikant, lernte ich vieles über mich selbst, technisch neue Dinge und den richtigen Umgang mit Menschen. Ich kann sagen, dass meinen Horizont dadurch erweitert habe. Dafür möchte ich mich bei den Mitarbeitern im Testcenter bedanken.

# Kapitel 5

# Literaturverzeichnis


1. https://docs.angularjs.org/guide/directive
   Stand: 12.02.17
2. https://de.wikipedia.org/wiki/Uniform_Resource_Locator
   Stand: 15.02.17
3. https://docs.angularjs.org/api/ng/service/$q
   Stand: 13.02.17
4. https://de.wikipedia.org/wiki/Microservices
   Stand: 11.02.17
5. https://de.wikipedia.org/wiki/Ext_JS
   Stand: 03.02.17
6. https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
   Stand: 05.02.17
7. http://www.torsten-horn.de/techdocs/maven.htm
   Stand: 06.02.17
8. https://de.wikipedia.org/wiki/Representational_State_Transfer
   Stand: 10.02.17
9. https://de.wikipedia.org/wiki/Apache_JMeter
   Stand: 04.02.17
10. https://de.wikipedia.org/wiki/IntelliJ_IDEA
    Stand: 04.02.17
11. https://de.wikipedia.org/wiki/Notepad%2B%2B
    Stand: 04.02.17
12. https://de.wikipedia.org/wiki/NetBeans_IDE
    Stand: 04.02.17