

# **DIPLOMARBEIT**

## **Optimierung der Fahrzeugrückstellung für ein Car-Sharing-Unternehmen**

**Ausgeführt im Schuljahr 2018/19 von:**

Matej Bradarić, 5AHIF-02  
Lukas Heinzl, 5AHIF-05  
Oliver Hovorka, 5AHIF-06  
Maximilian Suchy, 5AHIF-20

**Betreuer/Betreuerin:**

OSTR Mag. Otto Reichel

St.Pölten, am 1. April 2019

## **Eidesstattliche Erklärung**

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Unterschrift siehe Druckversion

---

Matej Bradarić

Unterschrift siehe Druckversion

---

Lukas Heinzl

Unterschrift siehe Druckversion

---

Oliver Hovorka

Unterschrift siehe Druckversion

---

Maximilian Suchy

St.Pölten, am 05.04.2019

# **Diplomandenvorstellung**



**Matej Bradarić**

**Geburtsdaten:**  
12.02.1999 in Lilienfeld

**Wohnhaft in:**  
Oberdörfel 62  
3172 Ramsau

**Werdegang:**  
2013 - 2019:  
HTBLuVA St.Pölten, Abteilung für Informatik  
2009 - 2013:  
Hauptschule Hainfeld, Informatikzweig

**Kontakt:**  
[bradaricmatej@gmail.com](mailto:bradaricmatej@gmail.com)

# **Diplomandenvorstellung**



Lukas Heinzl

Geburtsdaten:  
06.07.2000 in St. Pölten

Wohnhaft in:  
Russengasse 16  
3100 St. Pölten

Werdegang:  
2014 - 2019:  
HTBLuVA St.Pölten, Abteilung für Informatik  
2010 - 2014:  
BRG / BORG St. Pölten, Informatikzweig

Kontakt:  
[l.heinzl.stp@gmail.com](mailto:l.heinzl.stp@gmail.com)

# **Diplomandenvorstellung**



**Oliver Hovorka**

**Geburtsdaten:**  
**13.03.2000 in St.Pölten**

**Wohnhaft in:**  
**Pernerstorferplatz**  
**3100 St.Pölten**

**Werdegang:**  
**2014 - 2019:**  
**HTBLuVA St.Pölten, Abteilung für Informatik**  
**2010 - 2014:**  
**Privatgymnasium Mary Ward**

**Kontakt:**  
**ohovorka38@gmail.com**

# **Diplomandenvorstellung**



**Maximilian Suchy**

**Geburtsdaten:**  
04.06.2000 in Bruck an der Mur

**Wohnhaft in:**  
Habertheuerstraße 22  
8360 Mariazell

**Werdegang:**  
2014 - 2019:  
HTBLuVA St. Pölten, Abteilung für Informatik  
2010 - 2014:  
Hauptschule Mariazell

**Kontakt:**  
[max@privatevoid.net](mailto:max@privatevoid.net)

## **Danksagung - Lukas Heinzl**

Zu Beginn möchte ich mich bei meiner Familie und speziell bei meinen Eltern bedanken. Durch ihren Beistand sowie ihre finanzielle Unterstützung wurde mir ermöglicht diese Schule zu besuchen und diese Arbeit zu verfassen.

Weiters möchte ich mich bei Herrn OStR. Mag. Otto Reichel bedanken. Durch Ihre fachliche Kompetenz konnten Sie mir stets Tipps geben, wenn ich nicht weiter wusste. Außerdem bedanke ich mich dafür, dass Sie viel Zeit in das Korrekturlesen meiner Arbeit investiert und mir stets konstruktives Feedback gegeben haben.

Ein spezieller Dank geht an Katharina Munk. Ich habe dich im Laufe der 5. Klasse als Beste Freundin gewonnen und du bist mir stets zur Seite gestanden, egal ob in Guten oder Schlechten Zeiten. Ich danke dir dafür.

Ich möchte mich auch bei meinen Gruppenmitgliedern bedanken. Auch wenn es hin und wieder zu kleinen Differenzen kam, so konnten wir stets Fortschritte erzielen und gemeinsam weiterarbeiten. Ich werde diese Zeit positiv in Erinnerung behalten.

Als letztes möchte ich mich bei meinen Freunden und Klassenkollegen bedanken. Ohne euch wäre die Zeit in der HTL nicht so lustig und angenehm gewesen.

## **Danksagung - Oliver Hovorka**

An erster Stelle möchte ich mich ganz klar bei meinen liebevollen Eltern bedanken, die mich über die fünf Jahre der HTL begleitet und immer unterstützt haben. Weiters möchte ich mich bei meinen Klassenkameraden bedanken, denn egal wie schlimm es manchmal war, sie waren immer noch für ein paar Späße zu haben. Insbesondere muss ich hier den Herrn Heinzl erwähnen, der mir viel beigebracht hat und mich immer mit seinem Wissen unterstützt hat.

Ein weiterer Mensch, dem ich viel zu danken habe, ist unsere Klassenvorständin Frau Professor Reichhart. Nicht jeder hätte es geschafft mit meiner pubertären Entwicklung so gut umzugehen, wie sie es tat.

Ebenfalls schulde ich dem Herr Professor Reichel ein großes Dankeschön, da er mich in vielen Dingen sehr gut beraten und intensiv unterstützt hat.

## **Zusammenfassung**

In diesem Projekt wird eine Applikation entwickelt, welche eine Gewinnsteigerung beim Unternehmen Car2Go bewirken und das Arbeitsleben der Mitarbeiter erleichtern soll. Im Wesentlichen soll dabei das Umparken von Autos optimiert werden. Dieses Umparken ist notwendig, wenn Fahrzeuge an wenig frequentierten Orten abgestellt wurden. Solche Fahrzeuge müssen an Orte überstellt werden, an welchen eine deutlich höhere Kundenfrequenz zu erwarten ist (Hotspots).

Die zu entwickelnde Software besteht aus vier Modulen, wobei jedes Projektmitglied für die Entwicklung eines Moduls verantwortlich ist. Die zur Entwicklung benötigten Daten wie zum Beispiel die GPS-Positionen der Fahrzeuge werden dabei von der Firma Car2Go zur Verfügung gestellt. Hauptmodul ist eine mobile Applikation, die auf den Smartphones der Mitarbeiter installiert wird. Diese Applikation wird den Mitarbeitern anzeigen bei welchen Autos Umparken gewinnbringend ist, damit die Mitarbeiter diese Autos an sogenannte Hotspots umparken. Außerdem wird ein Monitoring der Fahrzeugpositionen und der Positionen der einzelnen Smartphones zur Verfügung gestellt.

In einem weiteren Modul wird eine Webseite erstellt, mit deren Hilfe berechtigte Personen auf einer Karte die Positionen aller Fahrzeuge und Mitarbeiter beobachten können. Alle wesentlichen Positionsdaten werden periodisch an den Server gesendet um diese Visualisierung in Echtzeit zu ermöglichen.

Eine weitere wesentliche Teilaufgabe des Projekts ist die Einrichtung eines zentralen Servers, über den die notwendige Datenkommunikation erfolgt.

## **Abstract**

This project's objective is the development of a mobile application, which shall raise the profit of our partner Car2Go and ease the daily work routine of the employees. Simply put, our project will optimize the relocation of cars. This is necessary because cars are sometimes parked at less populated spots and these cars generate less profit, to solve this problem the cars shall then be moved to more highly frequented spots.

The software that is going to be developed will consist of four modules, these four modules will each be developed by one individual student. The company Car2Go will provide all the data that is needed for development.

Primarily there is going to be a mobile application, which shall be installed on each employee's mobile phone. This will show each employee which relocation would be profitable. There will also be a monitoring tool for each smartphone's location.

Secondly, there is going to be a website, which will be able to show authorized persons where all the cars and employees are. All important data will be sent to the server periodically, to make the real time visualization possible.

Furthermore, there is going to be a central server, which will handle all the data communication.

# Inhaltsverzeichnis

<b>Vorwort</b>	<b>i</b>
Diplandenvorstellung . . . . .	ii
Danksagungen . . . . .	vi
Zusammenfassung . . . . .	viii
Abstract . . . . .	ix
<b>Inhaltsverzeichnis</b>	<b>x</b>
<b>1 Individuelle Zielsetzungen</b>	<b>1</b>
1.1 Datenkommunikation im Projekt Car-Sharing . . . . .	1
1.2 Vergleich von REST-APIs für mobile Applikationen . . . . .	2
1.3 Vergleich von geeigneten Webtechnologien . . . . .	2
1.4 Externe Nutzung und Vergleich von APIs und Webservices zur Routenberechnung . . . . .	2
<b>2 Webserver als Kommunikationsschnittstelle für Apps</b>	<b>3</b>
2.1 Einführung . . . . .	3
2.1.1 Kommunikation zwischen Servern und Apps . . . . .	3
2.1.2 Begriffserklärungen . . . . .	3

2.1.3	Gründe für den Einsatz von Web APIs . . . . .	4
2.2	REST . . . . .	5
2.2.1	Geschichte von REST . . . . .	6
2.2.2	Einsatz von REST in Java: JAX-RS . . . . .	9
2.2.3	Java Architecture for XML Binding . . . . .	12
2.2.4	Jackson . . . . .	15
2.3	Reverse Proxying . . . . .	19
2.3.1	Bewertung und Einsatzbeispiele Reverse Proxies . . . . .	20
2.3.2	NGINX . . . . .	20
2.3.3	Cherokee . . . . .	23
2.3.4	Bewertungstabelle . . . . .	24
2.3.5	Begründung . . . . .	25
2.4	Applikationsserver . . . . .	26
2.4.1	Java EE . . . . .	26
2.4.2	GlassFish . . . . .	27
2.4.3	Administration des GlassFish Servers . . . . .	27
2.4.4	Deployment von Webapplikationen mit GlassFish . . . . .	29
2.4.5	WildFly . . . . .	31
2.4.6	Administration des WildFly Servers . . . . .	31
2.4.7	Applikationsdeployment mit WildFly . . . . .	32
2.4.8	Bewertung . . . . .	34
<b>3</b>	<b>Sichere Kommunikation</b>	<b>37</b>
3.1	Allgemeine Konzepte . . . . .	37

3.2	Asymmetrische Verschlüsselung . . . . .	38
3.3	Symmetrische Verschlüsselung . . . . .	40
3.4	Hashing . . . . .	40
3.5	Schlüsseltauschverfahren . . . . .	41
3.6	Praktischer Einsatz von Verschlüsselung . . . . .	41
3.6.1	Transport Layer Security . . . . .	41
3.6.2	HTTPS . . . . .	42
3.6.3	Einsatz von HTTPS und HSTS bei Webservern . . . . .	43
<b>4</b>	<b>Service Management</b>	<b>46</b>
4.1	Allgemeines . . . . .	46
4.1.1	Daemons . . . . .	46
4.1.2	System V Init und RC . . . . .	47
4.1.3	Runlevels . . . . .	47
4.1.4	Beispiel für ein Initskript . . . . .	48
4.2	OpenRC . . . . .	49
4.2.1	Beispiele für Runscripts . . . . .	50
4.2.2	Dependency Management . . . . .	51
4.3	systemd . . . . .	52
4.3.1	Unit Files . . . . .	52
4.3.2	Dependency Management . . . . .	53
4.3.3	Socket Activation . . . . .	53
<b>5</b>	<b>Umsetzung im Projekt</b>	<b>54</b>
5.1	Eingesetzte Hardware . . . . .	54

5.2	Eingesetzte Software . . . . .	54
5.2.1	Entwicklung . . . . .	54
5.2.2	Reverse Proxy . . . . .	55
5.2.3	Applikationsserver . . . . .	55
5.2.4	Datenbank . . . . .	55
5.2.5	Integration in das Service Management . . . . .	56
5.3	Probleme bei der Entwicklung . . . . .	59
5.3.1	Problem: AES-256 nicht unterstützt in Java . . . . .	59
5.3.2	Problem: Umstellung von GlassFish auf WildFly . . . . .	59
<b>6</b>	<b>Grundlagen von REST</b>	<b>61</b>
6.1	Entstehung von REST . . . . .	61
6.1.1	Client-Server-Architektur . . . . .	61
6.1.2	Zustandslosigkeit . . . . .	62
6.1.3	Cache . . . . .	62
6.1.4	Einheitliche Schnittstellen . . . . .	63
6.1.5	Code-On-Demand . . . . .	63
<b>7</b>	<b>Kriterien und deren Gewichtung</b>	<b>64</b>
7.1	Allgemeines . . . . .	64
7.2	Entwicklung . . . . .	65
7.3	Bewertung . . . . .	66
7.3.1	Bewertungsskala . . . . .	67
<b>8</b>	<b>Bewertung der Client APIs</b>	<b>68</b>

8.1	Jersey . . . . .	68
8.1.1	Einleitung . . . . .	68
8.1.2	Bewertung von Jersey . . . . .	69
8.1.3	Entwicklung . . . . .	70
8.1.4	Punkte von Jersey . . . . .	74
8.2	Resteasy . . . . .	75
8.2.1	Einleitung . . . . .	75
8.2.2	Bewertung von Resteasy . . . . .	76
8.2.3	Entwicklung . . . . .	77
8.2.4	Punkte von Resteasy . . . . .	79
8.3	Restlet . . . . .	80
8.3.1	Einleitung . . . . .	80
8.3.2	Bewertung von Restlet . . . . .	81
8.3.3	Entwicklung . . . . .	82
8.3.4	Punkte von Resteasy . . . . .	87
8.4	Retrofit . . . . .	88
8.4.1	Einleitung . . . . .	88
8.4.2	Bewertung von Retrofit . . . . .	89
8.4.3	Entwicklung . . . . .	90
8.4.4	Punkte von Retrofit . . . . .	92
8.5	Zusammenfassung der APIs . . . . .	93
8.5.1	Jersey . . . . .	93
8.5.2	Resteasy . . . . .	93

8.5.3	Restlet . . . . .	94
8.5.4	Retrofit . . . . .	94
<b>9</b>	<b>Verwendung des ausgewählten Client APIs im Projekt</b>	<b>95</b>
9.1	Login . . . . .	97
9.2	Zuweisung eines Autos . . . . .	102
9.3	Fazit der Implementierung von Retrofit . . . . .	106
<b>10</b>	<b>Einführung in die Webentwicklung</b>	<b>107</b>
10.1	Übersicht und Geschichte der Webentwicklung . . . . .	107
10.2	Statische und dynamische Webseiten . . . . .	108
10.3	Dynamische Webseiten . . . . .	109
10.4	Unterschied zwischen serverseitigen und clientseitigen Programmiersprachen . . . . .	110
10.5	Vor- und Nachteile bei client-und serverseitiger Programmierung	111
10.6	PHP . . . . .	113
10.6.1	Vorteile von PHP . . . . .	113
10.6.2	Nachteile von PHP . . . . .	113
10.7	JavaScript . . . . .	114
10.7.1	Vorteile von JavaScript . . . . .	114
10.7.2	Nachteile von JavaScript . . . . .	115
10.7.3	Node.js . . . . .	116
10.8	Java . . . . .	117
10.8.1	Geschichte der Webentwicklung mit Java . . . . .	117
<b>11</b>	<b>Auswahl einer geeigneten Technologie</b>	<b>120</b>

11.1	Anforderungen . . . . .	120
11.2	Implementierung der Karte mit JSF . . . . .	121
11.3	Implementierung der Karte mit JavaScript . . . . .	121
11.3.1	Die Applikation als HTML5 definieren . . . . .	123
11.3.2	Map DOM Elemente . . . . .	124
11.3.3	Das Map Objekt . . . . .	124
11.3.4	Optionen des Map Elements . . . . .	124
11.3.5	Center . . . . .	124
11.3.6	Laden des Maps JavaScript-API . . . . .	125
11.3.7	Erzeugen von Custom Markern . . . . .	125
11.3.8	Bewertung . . . . .	128
11.4	Implementierung einer Karte mit PHP . . . . .	128
11.4.1	Erstellen der Datenbank und der XML Datei . . . . .	129
11.4.2	Erstellen der Karte . . . . .	131
11.4.3	Bewertung . . . . .	133
11.5	Anzeigen von Daten in Form von Tabellen mit JSF . . . . .	133
11.5.1	Bewertung . . . . .	135
11.6	Anzeigen von Daten in Form von Tabellen mit JavaScript . . . . .	135
11.6.1	Bewertung . . . . .	137
11.7	Anzeigen von Daten in Form von Tabellen mit PHP . . . . .	137
11.7.1	Bewertung . . . . .	138
11.8	Finale Auswahl . . . . .	138
<b>12</b>	<b>Verwendung im Projekt</b>	<b>139</b>

12.1	Login . . . . .	139
12.1.1	Die Oberfläche . . . . .	139
12.1.2	Der Controller . . . . .	141
12.2	Die Hauptseite . . . . .	142
12.2.1	Karte . . . . .	143
12.2.2	Tabellen . . . . .	143
12.2.3	Navigationsleiste . . . . .	146
12.3	Benutzerverwaltung . . . . .	147
12.4	Benutzerdatenpflege . . . . .	153
<b>13</b>	<b>Grundlagen von Routenberechnungen</b>	<b>155</b>
13.1	Überblick . . . . .	155
13.2	Funktionsprinzip . . . . .	156
13.2.1	Trilateration . . . . .	156
13.2.2	Multilateration und Distanzberechnung . . . . .	157
13.2.3	Positionen der Satelliten . . . . .	157
13.2.4	Positionsbestimmung . . . . .	157
13.3	Geschichte . . . . .	158
13.3.1	NAVSTAR GPS . . . . .	158
13.3.2	Galileo . . . . .	159
13.4	Zukunftsentwicklung . . . . .	160
13.4.1	GPS-3 . . . . .	160
13.4.2	Fertigstellung von Galileo . . . . .	161
13.5	Einsatzgebiete . . . . .	161

13.5.1	Navigation . . . . .	161
13.5.2	Einsatz im Projekt . . . . .	162
<b>14</b>	<b>Vorstellung diverser Kartenservices</b>	<b>163</b>
14.1	Übersicht . . . . .	163
14.2	OpenStreetMap . . . . .	164
14.2.1	Geschichte . . . . .	164
14.2.2	OpenStreetMap Foundation . . . . .	164
14.2.3	Preismodell . . . . .	165
14.2.4	Funktionsumfang . . . . .	165
14.2.5	Verwendungsbeispiele . . . . .	167
14.3	Google Maps . . . . .	169
14.3.1	Entstehungsgeschichte . . . . .	169
14.3.2	Google LLC . . . . .	169
14.3.3	Preismodell . . . . .	170
14.3.4	Funktionsumfang . . . . .	170
14.3.5	Verwendungsbeispiele . . . . .	172
14.4	Here Maps . . . . .	174
14.4.1	Entstehungsgeschichte . . . . .	174
14.4.2	HERE Global B.V. . . . .	175
14.4.3	Preismodell . . . . .	175
14.4.4	Funktionsumfang . . . . .	176
14.5	Sygic Maps . . . . .	177
14.5.1	Entstehungsgeschichte . . . . .	177

14.5.2	Sygic a.s. . . . .	178
14.5.3	Preismodell . . . . .	178
14.5.4	Funktionsumfang . . . . .	179
<b>15</b>	<b>Bewertung und Auswahl</b>	<b>181</b>
15.1	Bewertungskriterien . . . . .	181
15.2	Anforderungen . . . . .	182
15.2.1	Website . . . . .	182
15.2.2	Mobile Applikation . . . . .	182
15.2.3	Notwendige Funktionen . . . . .	183
15.2.4	Wunschkriterien . . . . .	183
15.2.5	Integration . . . . .	183
15.2.6	Verfügbarkeit . . . . .	183
15.3	Funktionsumfang . . . . .	184
15.4	Preis . . . . .	185
15.4.1	OpenStreetMap . . . . .	185
15.4.2	Google Maps . . . . .	185
15.4.3	Here Maps . . . . .	185
15.4.4	Sygic Maps . . . . .	186
15.4.5	Bewertung . . . . .	186
15.5	Integration . . . . .	187
15.5.1	OpenStreetMap . . . . .	187
15.5.2	Google Maps . . . . .	187
15.5.3	Here Maps . . . . .	188

15.5.4	Sygic Maps . . . . .	188
15.5.5	Bewertung . . . . .	188
15.6	Verfügbarkeit . . . . .	189
15.6.1	OpenStreetMap . . . . .	189
15.6.2	Google Maps . . . . .	189
15.6.3	Here Maps . . . . .	190
15.6.4	Sygic Maps . . . . .	190
15.6.5	Bewertung . . . . .	190
15.7	Auswahl . . . . .	191
15.7.1	Teilergebnisse . . . . .	191
15.7.2	Endbewertung . . . . .	192
15.7.3	Analyse . . . . .	192
<b>16</b>	<b>Verwendung des ausgewählten Dienstes im Projekt</b>	<b>193</b>
16.1	Die gmap-Komponente . . . . .	193
16.1.1	Die wichtigsten Attribute . . . . .	193
16.1.2	Verknüpfung mit dem GM-API . . . . .	195
16.1.3	Verwendungsbeispiel . . . . .	196
16.2	Verwendung der gmap-Komponente im Projekt . . . . .	200
16.2.1	Die Oberfläche . . . . .	200
16.2.2	Der Controller . . . . .	202
16.3	Maps SDK for Android . . . . .	203
16.3.1	Einfaches Beispiel . . . . .	203
16.3.2	Zeichnen von Routen . . . . .	205

16.3.3	Verwendung von Geofences . . . . .	206
16.4	Verwendung des Maps SDKs im Projekt . . . . .	208
16.4.1	Gliederung der App . . . . .	209
16.4.2	Die Main-Activity . . . . .	209
16.4.3	Die AssignCar-Activity . . . . .	212
16.4.4	Die CurTask-Activity . . . . .	214
16.5	Fazit . . . . .	217
<b>17</b>	<b>Kapitelzuordnung</b>	<b>218</b>
<b>Anhang</b>		<b>218</b>
	Tabellenverzeichnis . . . . .	222
	Verzeichnis der Listings . . . . .	223
	Literaturverzeichnis . . . . .	227
	Dokumentation . . . . .	237
	Begleitprotokolle . . . . .	241
	Besprechungsprotokolle . . . . .	245

# Kapitel 1

## Individuelle Zielsetzungen

### 1.1 Datenkommunikation im Projekt Car-Sharing

#### Maximilian Suchy

Die Android-App und die Website mussten über eine einheitliche, sichere Schnittstelle mit dem Server kommunizieren können. Weiters benötigte das Team eine funktionsfähige Entwicklungsinfrastruktur, um die Entwicklung der Clients sowie des Servers reibungslos und effizient abwickeln zu können. Für diese Zwecke stellte der Diplomand Maximilian Suchy die nötigen Serversysteme zur Verfügung und betreute diese während der Entwicklung. Es wurde eine private GitLab-Instanz betrieben, um die Zusammenarbeit zwischen den Entwicklern zu erleichtern. Weiters wurden während der Entwicklung die Applikationsserver und Datenbanken je nach Bedarf installiert und konfiguriert.

Um die Entwicklung parallel mit mehreren Applikationsservern zu ermöglichen, wurde ein Reverse Proxy eingesetzt. Teil der Aufgabe war es, hierfür die passende Software anhand eines Vergleiches zu finden. Eine ähnliche Bewertung fand für die Applikationsserver GlassFish und WildFly statt.

## 1.2 Vergleich von REST-APIs für mobile Applikationen

### Oliver Hovorka

Das Ziel des Diplomanden Oliver Hovorka ist es diverse REST Client APIs zu vergleichen und eines für die Verwendung in dem Projekt Carsharing auszuwählen. Weiters muss die Client-Kommunikation mithilfe des ausgewählten APIs im Rahmen der Projektentwicklung implementiert werden. Diese Implementation muss als mobile Applikation unter Android erfolgen.

## 1.3 Vergleich von geeigneten Webtechnologien

### Matej Bradarić

Es sollen die Technologien JavaScript, PHP und Java im Bereich der Webentwicklung verglichen werden, um ein Fazit zu ziehen, welche für die Entwicklung der Webseite am geeignetsten ist.

Die Hauptbestandteile der Webseite sind eine Karte und Tabellen, welche in der Folge in den verschiedenen Technologien als Beispiele implementiert wurden.  
Zu diesem Zweck beschäftigte sich der Diplomand Matej Bradarić mit dem Vergleich von diesen Technologien.

## 1.4 Externe Nutzung und Vergleich von APIs und Web-services zur Routenberechnung

### Lukas Heinzl

Es sollen verschiedene Services zur Darstellung von Karten sowie zur Routenberechnung verglichen werden. Dazu werden OpenStreetMap, Google Maps, Here Maps und Sygic Maps herangezogen. Der Fokus liegt auf der einfachen Verwendung mit den bereits im Projekt verwendeten Technologien.

Der Diplomand Lukas Heinzl stellt die genannten Services vor und vergleicht sie. Weiters beschreibt er, wie das ausgewählte Service konkret im Projekt zum Einsatz kommt.

# **Kapitel 2**

## **Webserver als Kommunikationsschnittstelle für Apps**

### **2.1 Einführung**

#### **2.1.1 Kommunikation zwischen Servern und Apps**

Moderne Mobilapplikationen sind häufig darauf ausgelegt, große Datenmengen aus dem Internet zu holen oder ins Internet zu stellen, daher benötigen sie eine dauerhafte Internetverbindung und eine Verbindung zu einem Server, der diese Daten zugänglich macht oder empfängt und an andere Teilnehmer des Dienstes verteilt.

#### **2.1.2 Begriffserklärungen**

Im folgenden Abschnitt wird oft von APIs gesprochen. Das Akronym API steht für Application Programming Interface. Im weitesten Sinne ermöglicht ein API den Zugriff auf eine Softwarekomponente und definiert diesen klar, sodass es einem externen Programmierer erleichtert wird, innerhalb seines Programms mit der Komponente umzugehen. Im Zusammenhang des folgenden Abschnitts wird größtenteils von APIs ausgegangen, mit denen Software auf entfernten Rechnern zugegriffen wird. Dieser Zugriff geschieht über das Internet-Protokoll.

Ein Web API bezeichnet ein API für Zugriff auf entfernte Rechner, das auf Webtechnologien wie beispielsweise HTTP und XML aufbaut.

### 2.1.3 Gründe für den Einsatz von Web APIs

Es bietet sich aus diversen Gründen an, die Schnittstelle zur Serverkommunikation mit Webtechnologien zu implementieren. So ist es zum Beispiel leicht möglich, die Applikation zusätzlich zur Native App in Form einer Website anzubieten, da man hier etwa clientseitig über AJAX oder auch über einen REST-Client im serverseitigen Webframework mit der API kommunizieren kann. Des Weiteren sind die Ports 80 und 443, die standardmäßig für Datenverkehr über HTTP verwendet werden, in nahezu jeder Firewall freigeschalten. So sind Benutzer von Mobilapplikationen nicht von deren Verwendung ausgeschlossen, sollten sie sich an Orten mit restriktivem Internetzugriff aufhalten, wie es bei Gratis-WLAN in Hotels oder Restaurants nur allzuoft der Fall ist. Des Weiteren ist mit HTTPS eine einfache, gut etablierte Methode zur Verschlüsselung der gesamten Kommunikation gegeben.

Web-APIs haben sich nicht zuletzt wegen den oben genannten Vorteilen als der De-Facto-Standard für APIs, die über das Internet geleitet werden, etabliert. Dies stellt in sich selbst einen großen Vorteil dar, da durch den starken Einsatz von Web-APIs eine äquivalente große Verfügbarkeit an Tools, mit denen Web-APIs erstellt werden können entstand.

## 2.2 REST

REST ist einer der beliebtesten Standards für RPC und Web-basierte Datenkommunikation und wird am häufigsten über HTTP verwendet, was es gleichzeitig auch zur beliebtesten Methode zur Umsetzung von Web-APIs macht. Eine REST-Anfrage besteht aus einer HTTP-Methode gefolgt von einem URI, der die verlangte REST-Ressource sowie zugehörige Parameter angibt.

```
1 DELETE /api/v1/pictures/293892
```

Listing 2.1: Beispiel für eine REST-Anfrage

Die Antwort auf eine solche Anfrage besteht aus einem oder mehreren Objekten und wird in einem Plaintextformat in Form von XML oder JSON übertragen. In vielen Fällen unterstützt der REST-Server beide Formate und der Client hat die Wahl, welches Format für die Antwort verwendet wird.

```
1 {
2     "action": "delete_picture",
3     "itemId": "293892",
4     "response": {
5         "status": "success",
6         "message": "Picture deleted successfully."
7     }
8 }
```

Listing 2.2: Beispiel für eine REST-Antwort mit JSON

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <request>
3   <action>delete_picture</action>
4   <itemId>293892</itemId>
5   <response>
6     <status>success</status>
7     <message>Picture deleted successfully.</message>
8   </response>
9 </request>
```

Listing 2.3: Beispiel für eine REST-Antwort mit XML

### 2.2.1 Geschichte von REST

#### Vorgänger: XML-RPC

Dave Winer veröffentlichte 1999 die [Su:Web4, XML-RPC Specification]. XML-RPC gilt als eine der ersten Spezifikationen dafür, wie Remote Procedure Calling in verteilten Systemen aussehen könnte. Eine Besonderheit von XML-RPC im Gegensatz zu früheren Implementationen und Spezifikationen von RPC ist der Einsatz von HTTP als darunterliegendes Protokoll. Hierbei wird eine POST-Anfrage an den jeweiligen Server gesendet, in den zusätzlichen Daten der Anfrage wird ein XML-Dokument übertragen, das die aufzurufende Methode und deren Parameter beinhaltet.

```
1 <methodCall>
2   <methodName>test.calculateSum</methodName>
3   <params>
4     <param><value><int>59</int></value></param>
5     <param><value><int>7</int></value></param>
6   </params>
7 </methodCall>
```

Listing 2.4: Beispiel für eine XML-RPC-Anfrage

Aus dem Beispiel ist auszumachen, dass XML-RPC Datentypen in expliziter Form unterstützt. Diese sind in Form eines XML-Elements umgesetzt, welches sich innerhalb des entsprechenden Elements `value`, das den Wert an sich beinhaltet befindet.

Die Spezifikation von XML-RPC ist sehr simpel gehalten, da Einfachheit einer der wichtigsten Punkte bei der Erfindung von XML-RPC war. Dies führte das Problem mit sich, dass die Spezifikation manche Grenzfälle nicht berücksichtigt wurden. In späteren Revisionen musste beispielsweise die Definition des String-Datentyps abgeändert werden, in der ursprünglichen Variante war ein String auf ASCII beschränkt, obwohl XML für wesentlich mehrCharsets konzipiert ist. Der Datentyp für die Angabe eines Zeitstempels besitzt kein Zeitzonennattribut, die Zeitzone soll über einen anderen Weg vom Entwickler angegeben werden.

#### Vorgänger: SOAP

Das Akronym SOAP stand ursprünglich für Simple Object Access Protocol. Bei SOAP handelt es sich um eine Weiterentwicklung von XML-RPC, die weder als „simple“ be-

zeichnet werden kann, noch zwingend „Object Access“, also Zugriff auf Objekte verschafft. Aus diesem Grund wird der Begriff „SOAP“ nun als Eigenname und nicht mehr als Akronym behandelt.

SOAP ist XML-RPC im Allgemeinen sehr ähnlich. Wie beim Vorgänger wird auf HTTP aufgebaut, wobei immer POST-Anfragen verwendet werden. Auch wird die tatsächliche Anfrage in Form eines XML-Dokuments gestellt.

```

1 <?xml version = "1.0" encoding="UTF-8"?>
2 <SOAP-ENV:Envelope
3   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
4   SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
5
6   <SOAP-ENV:Body xmlns:m = "http://www.mycompany.invalid/definition">
7     <m:CallMyMethod>
8       <m:MyParameter>Parameter Content</m:MyParameter>
9     </m:CallMyMethod>
10    </SOAP-ENV:Body>
11 </SOAP-ENV:Envelope>
```

Listing 2.5: Beispiel für eine SOAP-Anfrage

Das Format des Anfragedokuments ist im Vergleich zu XML-RPC wesentlich komplexer und unübersichtlicher. Ein Hintergedanke beim Einsatz von XML bei XML-RPC war die Lesbarkeit der Anfragen. Weiters schenkt man der angeblichen Einfachheit von SOAP durch seine komplexe, seitenlange Spezifikation [Su:Web5, W3C: SOAP 1.2 Specification], die auf mehrere Teile aufgeteilt ist wahrscheinlich wenig Glauben. Die strikten Definitionen lösten aber das Problem der Ambiguitäten von XML-RPC, die bei manchen Grenzfällen zu Problemen führten. Durch dies wurde SOAP dennoch ein weitverbreiteter Standard. Viele große Firmen, darunter auch Microsoft, waren sehr überzeugt vom Potenzial von SOAP.

## Entstehung von REST

Als Lösung des Problems der enormen Komplexität von SOAP wurde Representational State Transfer, abgekürzt REST, geschaffen. REST als solches wurde nicht direkt definiert, sondern ergab sich aus einer Dissertation von Roy Fielding. Effektiv ist REST kein eigenes Protokoll, sondern bedeutet den Einsatz einer Kombination von dem, was bereits gegeben war. Anstatt komplizierter XML-Anfragen setzt REST direkt auf die verschiedenen HTTP-Methoden auf, die es gibt. Um einen Überblick zu verschaffen werden vier wichtige Methoden zusammen mit einer Erklärung ihrer Bedeutung im

Zusammenhang mit REST beschrieben:

- GET: Holt eine Ressource vom Server, durch diese Anfrage wird die Ressource an sich nicht verändert.
- POST: Fügt eine Ressource unter der gegebenen Ressource an.
- PUT: Erstellt oder verändert die angegebene Ressource.
- DELETE: Löscht die Ressource.

REST definiert keine Methoden, wie sie im ursprünglichen Sinne von XML-RPC und SOAP in diesen vorkommen. Stattdessen wird von sogenannten Ressourcen gesprochen, die über den URI angegeben werden. Parametrisierung der aufgerufenen Methoden oder Ressourcen ist ebenfalls innerhalb des URI gelöst, wobei hier sowohl Pfadparameter direkt als Teil der Ressource, als auch dahinter angefügte HTTP-Query-Parameter eingesetzt werden können. Auch die Zusatzdaten, die in einer gewöhnlichen POST-Anfrage vorkommen können im Zusammenhang von REST genutzt werden. Im Teil von Herrn Hovorka werden die Grundprinzipien von REST - darunter die Zustandslosigkeit sowie die strikte Trennung von Client und Server - übersichtsmäßig behandelt.

## 2.2.2 Einsatz von REST in Java: JAX-RS

Die Java API for RESTful Web Services (abgekürzt JAX-RS) beschreibt eine Programmierschnittstelle, mit der REST-basierte Web Services in Java EE implementiert werden können. Die clientseitige Implementation von REST in Java, beispielsweise mithilfe von Retrofit wird im Teil von Herrn Hovorka ausführlich erklärt.

Die Funktionalität des Webservice wird in den Resource Classes abgebildet. Hier wird starker Gebrauch von Annotationen gemacht, womit Informationen wie der aufzurufende Pfad der einzelnen Methoden, sowie etwaige Parameter angegeben werden.

Mehrere Resource Classes werden zu einer Application zusammengeführt. Hierbei wird die Klasse `javax.ws.rs.core.Application` erweitert, die entstandene Klasse wird oft als Konfigurationsklasse bezeichnet. Mithilfe dieser Klasse wird der REST-Application ein eigener Pfad zugewiesen, sowie alle darin befindlichen Resource Classes aufgelistet.

```

1 @javax.ws.rs.ApplicationPath("rest")
2 public class ApplicationConfig extends Application {
3     @Override
4     public Set<Class<?>> getClasses() {
5         Set<Class<?>> resources = new java.util.HashSet<>();
6         addRestResourceClasses(resources);
7         return resources;
8     }
9
10    private void addRestResourceClasses(Set<Class<?>> resources) {
11        resources.add(CarResource.class);
12        resources.add(HotspotResource.class);
13        resources.add(JobResource.class);
14        resources.add(SettingsResource.class);
15        resources.add(UserResource.class);
16    }
17}
18

```

Listing 2.6: Beispiel JAX-RS Application, language=Java

Ein einfaches, häufig anzutreffendes Beispiel ist eine Resource Class welche die Instanzen einer POJO-Klasse über REST den Clients zum Abrufen oder für Veränderungen zur Verfügung stellt.

```

1  @Path("user")
2  @Stateless
3  public class UserResource implements Serializable {
4      /* ... */
5      @GET
6      @Path("{userid}")
7      @Produces(MediaType.APPLICATION_JSON)
8      public Response getUser(@PathParam("userid") int userId,
9          @QueryParam("key") String key) {
10
11         if (!KeyUtil.checkApiKey(key)) {
12             return Response.status(Response.Status.FORBIDDEN).build();
13         }
14         User u = dao.getUser(userId);
15         return Response.ok(u).build();
16     }
17 }
```

Listing 2.7: Beispiel JAX-RS Resource Class

## Parameter

Im obigen Beispiel ist eine sehr einfache Resource Class mit einer einzelnen Methode zu sehen. Die JAX-RS-Annotationen werden hier verwendet, um Pfade, HTTP-Methoden und den Content-Type des Response-Wertes zu definieren. Die geschwungenen Klammern im Pfad `{userid}` beschreiben einen sogenannten Path-Parameter, welcher in weiterer Folge unter der Verwendung der Annotation `@PathParam` als Methodenparameter übergeben werden kann, um so z.B. größere Datenmengen serverseitig zu filtern, einzufügende Werte anzugeben, oder wie hier anhand eines eindeutigen Wertes ein einzelnes Objekt zu identifizieren.

Neben Pfadparametern können auch HTTP-Anfrageparameter auf diese Weise verarbeitet werden. Diese Funktionalität wird durch `@QueryParam` gegeben. Dies ist sinnvoll wenn z.B. die gleiche Information bei mehreren unterschiedlichen Anfragen übertragen werden soll. Dies wäre etwa bei API Keys der Fall, die den Zugriff auf das API regeln.

## Clientseitige Objekte

Es können Objekte von der Clientseite aus im JSON- oder XML-Format zum Server gesendet werden, welcher diese mithilfe von JAXB in ein Java-Objekt umwandelt. Dazu wird die Annotation `@Consumes` eingesetzt, über welche die akzeptierten

MIME-Types angegeben werden können. Dies könnte wie folgt aussehen:

```

1 @POST
2 @Path("/create/")
3 @Produces(MediaType.APPLICATION_JSON)
4 @Consumes(MediaType.APPLICATION_JSON)
5 public Response createUser(User u) {
6     u.setPassword(PasswordUtil.hashPassword(u.getPassword()));
7     return Response.ok(dao.createUser(u)).build();
8 }
```

Listing 2.8: JAX-RS: Clientseitig erzeugte Objekte am Server

## Responses

Der `Response` ist ein Feature von JAX-RS, das es dem Entwickler ermöglicht, den Rückgabewert einer REST-Methode mit HTTP-Statuscodes sowie Headern zu versehen, für REST geeignete Fehlermeldungen zu retournieren und auch Cookies zu setzen. Dabei wird ein `ResponseBuilder` eingesetzt, mit dem über mehrere aufeinanderfolgende Methodenaufrufe alle Informationen über die Antwort gesetzt werden können. Durch den Aufruf der Methode `build()` wird die Antwort zusammengefügt und in der REST-Methode retourniert.

Im obigen Beispiel werden zwei einfache Einsatzgebiete gezeigt:

- Mit `Response.status(Response.Status.FORBIDDEN)` wird als Antwort der HTTP-Statuscode 403 (Forbidden) zum Client übertragen. Unter Verwendung anderer Statuscodes können so schnell und einfach diverse Fehlermeldungen an den Client gesendet werden. Die Enumeration `Response.Status` enthält die wichtigsten HTTP-Statuscodes.
- `Response.ok(object)` deutet auf eine erfolgreiche Anfrage hin. Optional wird ein gegebenes Objekt serialisiert und an den Client gesendet.

Ein Problem, das sich in diesem Zusammenhang ergibt, ist dass die Information über Typsicherheit eines generischen Objekts wie etwa einer Liste verloren geht. Als Abhilfe wird `GenericEntity` verwendet. Generics existieren in Java zur Laufzeit nicht, was es sehr schwer macht, zur Laufzeit Informationen über generische Datentypen zu erhalten. Daher muss dieses Workaround mit einer anonymen Klasse genutzt werden.

```
1 @GET
2 @Produces(MediaType.APPLICATION_JSON)
3 public Response getUsers() {
4     List<User> l = dao.getUsers();
5     GenericEntity<List<User>> ge = new GenericEntity<List<User>>(l) { };
6     return Response.ok(ge).build();
7 }
```

Listing 2.9: JAX-RS: GenericEntity

### 2.2.3 Java Architecture for XML Binding

Die Java Architecture for XML Binding, kurz JAXB, ermöglicht es, im Zusammenhang mit einem JAX-RS Web Service die Java-Objekte in ein XML- oder JSON-Dokument umzuwandeln und vice versa. Über Annotationen kann die Konvertierung manipuliert werden, um so anzugeben, wie etwa ein besonderer Datentyp umgewandelt werden soll, oder dass ein Passwortfeld nicht in der XML/JSON-Darstellung vorhanden sein soll. Folgendes ist ein Beispiel für eine POJO-Klasse, die mit entsprechenden Annotationen versehen ist.

```
1 @Entity
2 @Table(name = "vuser")
3 @XmlRootElement
4 public class User implements Serializable {
5     private Integer id;
6     private String email;
7     private String vorname;
8     private String nachname;
9
10    @XmlTransient
11    private String password;
12
13    @XmlJavaTypeAdapter(DateAdapter.class)
14    private Date deleted;
15
16    private UserType userType;
17    /* ... */
18 }
```

Listing 2.10: JAXB-Annotationen in einer Geschäftsklasse

Es kommen hier drei wichtige Annotationen von JAXB zum Einsatz:

- `@XmlRootElement`: Definiert eine Klasse als konvertierbar für JAXB
- `@XmlTransient`: Diese Annotation gibt an, dass das zugehörige Feld von JAXB ignoriert werden soll. Es scheint daher nicht in der XML- oder JSON-Repräsentation auf.
- `@XmlJavaTypeAdapter`: Gibt an, welcher Adapter für dieses Feld einzusetzen ist. Adapter sind vergleichbar mit Convertern in JSF, sie wandeln ein Feld eines speziellen Typs in einen anderen Typen um, der von JAXB umgewandelt werden kann.

Die zugehörige JSON-Darstellung eines Objekts der obigen Klasse könnte so aussehen:

```
1 {
2     "id": 5,
3     "email": "max.mustermann@mustermail.privatevoid.net",
4     "vorname": "Max",
5     "nachname": "Mustermann",
6     "deleted": "2019-03-03 12:04:53",
7     "userType": {
8         "id": 1,
9         "type": "Relocator"
10    }
11 }
```

Listing 2.11: JAXB JSON-Darstellung

Im Listing wird ersichtlich, dass das Feld `password` entsprechend der Annotation `@XmlTransient` exkludiert wurde. Der Wert des Feldes `type`, ein Objekt der Klasse `UserType`, welche ebenfalls als `@XmlRootElement` annotiert ist, wurde automatisch entsprechend umgewandelt und in das Dokument eingebaut.

Der User in diesem Beispiel wurde bereits gelöscht, demnach enthält das Feld `deleted` den Zeitpunkt des Löschens. Dieser Wert wird wie oben erklärt mithilfe eines Adapters aus einem Date-Objekt erzeugt. Die Klasse `DateAdapter` ist wie folgt aufgebaut:

```

1  public class DateAdapter extends XmlAdapter<String, Date> {
2
3      private static final ThreadLocal<DateFormat> DATEFORMAT
4          = new ThreadLocal<DateFormat>() {
5
6          @Override
7          protected DateFormat initialValue() {
8              return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
9          }
10     };
11
12     @Override
13     public Date unmarshal(String v) throws Exception {
14         return DATEFORMAT.get().parse(v);
15     }
16
17     @Override
18     public String marshal(Date v) throws Exception {
19         return DATEFORMAT.get().format(v);
20     }
21 }
```

Listing 2.12: JAXB XmlAdapter

Die Klasse `XmlAdapter` ist eine abstrakte generische Klasse, deren konkrete Implementierungen zwischen zwei spezifischen Klassen hin und her konvertieren. In den beiden überschriebenen Methoden `marshal` und `unmarshal` wird die Logik zur Konvertierung vom Anwendungsentwickler programmiert. Adapter werden vor allem für Klassen verwendet, die nicht vom Entwickler stammen und daher nicht verändert werden können. Diese Klassen könnten Eigenschaften aufweisen, die eine Serialisierung mit JAXB unmöglich machen. Um dieses Problem zu lösen, wird ein `XmlAdapter` angewandt. Es besteht auch die Möglichkeit, eine eigene Klasse zu entwickeln, die rein dafür existiert, um von JAXB umgewandelt zu werden. Der Adapter passt dann die nicht serialisierbare Klasse an diese Hilfsklasse an.

## 2.2.4 Jackson

Jackson ist ein JSON-Processor für Java und kann bei JAX-RS als JSON-Provider eingesetzt werden, um Java-Objekte ins JSON-Format umzuwandeln und umgekehrt. RESTEasy verwendet standardmäßig Jackson. Jackson selbst wurde allerdings ursprünglich als reiner JSON-Processor konzipiert und ist hierbei der schnellste seiner Art. Zunächst ein Beispiel zur Konvertierung einer einfachen Geschäftsklasse `Pojo` zu JSON und wieder zurück in ein Java-Objekt:

```
1 ObjectMapper mapper = new ObjectMapper();
2 Pojo p = new Pojo(5, "MyPOJO", false);
3 String jsonPojo = mapper.writeValueAsString(p);
4 Pojo parsedPojo = mapper.readValue(jsonPojo, Pojo.class);
```

Listing 2.13: Jackson ObjectMapper

Die Beispielklasse `Pojo` ist wie folgt aufgebaut:

```
1 public class Pojo {
2     private int id;
3     private String name;
4     private boolean something;
5     /* ... */
6 }
```

Listing 2.14: Jackson POJO

Die Klasse `ObjectMapper` ist Teil der Bibliothek `jackson-databind`, die einfache Tools zur JSON-Serialisierung mit Jackson bereitstellt. Im Beispiel wird das Objekt mit der Methode `writeValueAsString` in folgende JSON-Darstellung gebracht:

```
1 {
2     "id" : 5,
3     "name" : "MyPOJO",
4     "something" : false
5 }
```

Listing 2.15: Jackson JSON-Darstellung POJO

Die Zurückkonvertierung wird durch die Methode `readValue` durchgeführt, welche den JSON-String sowie das Klassenobjekt der zu erstellenden Klasse übergeben bekommt.

## Baumstrukturen mit Jackson

Neben dem Parsen von Geschäftsobjekten wäre es auch nützlich, flexibel mit dynamischen Baumstrukturen umgehen zu können. Abhilfe dafür schafft die Klasse `ObjectNode`. Sie ermöglicht es, Baumstrukturen für ein JSON-Dokument in Java zu erstellen, einzelne Werte im Baum auszulesen und diese zu ändern. Es folgt wieder ein einfaches Beispiel:

```
1 ObjectNode node = new ObjectNode(JsonNodeFactory.instance);
2 node.put("number", 1);
3 node.put("bool", true);
4 node.with("sub").put("id", 33);
5 node.with("sub").putPOJO("pojo", p);
6 System.out.println(node.with("sub").get("id"));
7 System.out.println(mapper.writeValueAsString(node));
```

Listing 2.16: Jackson `ObjectNode`

Aufgrund der Methoden `put` und `get` verhält sich eine `ObjectNode` ähnlich wie eine `Map`. Interessant ist jedoch die Methode `with`. Sie returniert eine neue `ObjectNode` die ein Kindelement der darüberliegenden Node ist. So ist es einfach möglich, Baumstrukturen zu erstellen und Werte aus diesen herauszuholen, sofern der Aufbau des JSON-Dokuments bekannt ist. Weiters gibt es eine Hilfsmethode `putPOJO` mit der ein Geschäftsobjekt einfach in die Baumstruktur integriert werden kann.

```

1 {
2   "number" : 1,
3   "bool" : true,
4   "sub" : {
5     "id" : 33,
6     "pojo" : {
7       "id" : 5,
8       "name" : "MyPOJO",
9       "something" : false
10    }
11  }
12 }
```

Listing 2.17: Jackson JSON-Darstellung ObjectNode

## Inline-Konfiguration

Der `ObjectMapper` beinhaltet viele Sonderfeatures, die wahlweise ein- und ausgeschaltet werden können. Die einzelnen Features sind auf dem [Su:Web8, Jackson GitHub Wiki] aufgelistet und beschrieben. Es gibt drei Arten von Features

- Mapper Features: Fundamentale Optionen, die innerhalb eines Mappers nicht On-The-Fly geändert werden können.
- Serialization Features: Optionen der Serialisierung, beeinflussen die Darstellung und Struktur des JSON-Outputs.
- Deserialization Features: Optionen der Deserialisierung, beispielsweise ob JSON-Arrays zu Java-Collections umgewandelt werden sollen.

Features können mit den Methoden `enable` und `disable` des `ObjectMapper`-Objektes für dieses konfiguriert werden.

```

1 mapper.enable(SerializationFeature.INDENT_OUTPUT);
2 mapper.disable(DeserializationFeature.ACCEPT_SINGLE_VALUE_AS_ARRAY);
```

Listing 2.18: Jackson Features

## 2.3 Reverse Proxying

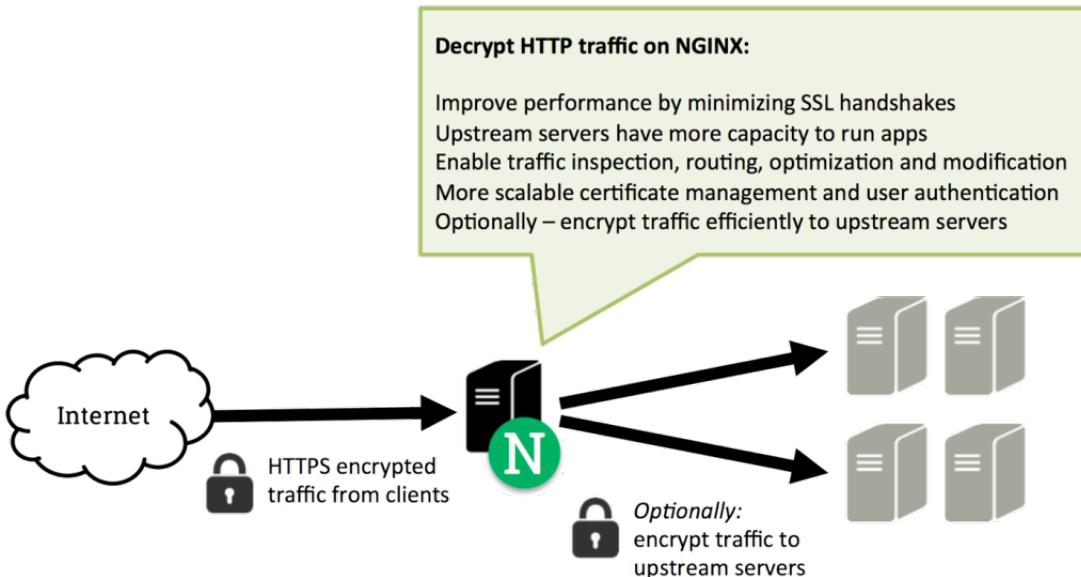


Abbildung 2.1: Reverse Proxying mit SSL-Offloading

1

Der Begriff HTTP-Proxy beschreibt ein Programm, das HTTP-Anfragen im Namen eines Clients tätigt, der darauf eingerichtet ist, diesen Proxy zu verwenden. Ein HTTP-Reverse Proxy ist ein ähnliches Programm, jedoch wird dieser auf der Serverseite eingerichtet was ihn für Clients normalerweise vollkommen transparent macht. Ein Reverse Proxy wird vor allem für folgende Features eingesetzt:

- Transparentes Load Balancing
- SSL-Offloading und damit einheitliches Management der SSL-Zertifikate
- Virtual Hosting mit verschiedenen Webservern im Hintergrund
- Caching und Komprimierung

Reverse Proxies bieten viel Flexibilität, aber auch Sicherheitsvorteile. Viele Serversysteme, die über HTTP mit ihren Clients kommunizieren unterstützen moderne Sicherheitsfeatures nur teilweise bis gar nicht, oder sind in diesem Bereich umständlich zu konfigurieren. Reverse Proxies sind auf Sicherheit ausgelegt und vereinheitlichen die Konfiguration.

<sup>1</sup>Bildquelle: <https://www.nginx.com/blog/nginx-ssl/>

## Performancevorteile

Viele Features, die ein Reverse Proxy bietet können zu einer signifikanten Verbesserung in der Performance der eingesetzten Services führen. Der größte Overhead bei TLS entsteht durch den initialen Handshake, dem erstmaligen Verbindungsauftakt zwischen einem TLS-Client und TLS-Server. Werden im Zusammenhang einer einzigen Applikation mehrere TLS-fähige Backend-Server eingesetzt, mit denen der Client direkt Verbindungen aufbaut, muss mit jedem dieser Server ein eigener Handshake durchgeführt werden. Wird ein Reverse Proxy vor den Backend-Servern eingesetzt, verwaltet dieser alle TLS-Sessions gemeinsam, so müssen Clients nur einen einzigen TLS-Handshake durchführen, um auf alle von den Backend-Servern bereitgestellte Ressourcen zuzugreifen.

### 2.3.1 Bewertung und Einsatzbeispiele Reverse Proxies

In den folgenden Textabschnitten werden der hochperformante Webserver NGINX, der vor allem für seine stark ausgeprägten Features im Bereich Reverse Proxing sehr beliebt ist und der weniger bekannte Webserver Cherokee, der mit einem einzigartigen grafischen Konfigurationstool wirbt, miteinander verglichen und nach verschiedenen Kriterien bewertet, die für den Einsatz im Zusammenhang mit dem Projekt relevant sind. Bewertet wird nach folgenden Kriterien:

- Dokumentation: Wie gut dokumentiert ist die Software? Welche offiziellen Resourcen stehen hierbei zur Verfügung?
- Bekanntheit: Wie vertraut ist die Online-Community im Allgemeinen mit der Software? Wichtig für das Lösen komplexer Probleme, die nicht von offiziellen Dokumenten behandelt werden.
- Konfiguration: Wie leicht ist die Software generell und für die Verwendung als Reverse Proxy zu konfigurieren? Sind nützliche Features einfach anwendbar?
- Flexibilität: Kann der Reverse Proxy in diversen nontrivialen Szenarien eingesetzt werden?

### 2.3.2 NGINX

NGINX ist ein hochperformanter Webserver und Reverse Proxy. Im Bereich der Webserver ist er der größte Konkurrent von Apache, dem meistgenutzten Webser-

ver. Neben HTTP unterstützt NGINX auch TCP/UDP-Streams und kann so als Load Balancer für jede beliebige Serversoftware eingesetzt werden.

## Reverse Proxy mit HTTP

Die Reverse Proxy-Funktionalität von NGINX ist im Modul `http_proxy` enthalten. Um einen HTTP-basierten Applikationsserver als Backend-Server in einer Reverse Proxy-Konfiguration einzusetzen, bedarf es lediglich einer `proxy_pass` Direktive in der Konfiguration von NGINX. Dies sorgt dafür, dass der gesamte eingehende Datenverkehr zum angegebenen Applikationsserver weitergeleitet wird.

```
1 server {
2     server_name glassfish.privatevoid.net;
3     listen 443 ssl http2;
4     ssl_certificate /etc/ssl/certs/wildcard.crt;
5     ssl_certificate_key /etc/ssl/private/wildcard.key;
6     location / {
7         proxy_pass http://127.48.48.1:8080/;
8     }
9 }
```

Listing 2.19: Konfiguration: NGINX HTTP Reverse Proxy

Die obige Konfiguration erzeugt einen eigenen Virtual Host mit HTTPS und Unterstützung für HTTP 2.0, der Zugriff auf einen Applikationsserver im Backend erlaubt. NGINX kommuniziert dabei unverschlüsselt mit dem Backend-Server, welcher daher nicht für HTTPS konfiguriert werden muss.

## Reverse Proxy mit AJP

Zusätzlich zu HTTP kann auch mithilfe eines Third-Party-Moduls das Apache JServ Protocol, das von den meisten Java Application Servern unterstützt wird eingesetzt werden. AJP wurde ursprünglich für den Apache Webserver zusammen mit dem JK-Connector-Modul entwickelt, um ohne großen Overhead über Apache Zugriff auf Servlets von Apache Tomcat, einem einfachen Java-Webserver für das Servlet-API und JavaServer Pages, zu ermöglichen. Es bietet im Einsatz zwischen einem Applikationsserver und einem Reverse Proxy einen signifikanten Performance-Vorteil gegenüber HTTP, da es sich um ein Binärprotokoll handelt und keine HTTP-Header vom Backend behandelt werden müssen. Eine genaue Dokumentation des Apache

JServ Protocol ist in der offiziellen [Su:Web3, AJP Protocol Reference] zu finden.

```
1 server {
2     server_name glassfish.privatevoid.net;
3     listen 443 ssl http2;
4     ssl_certificate /etc/ssl/certs/wildcard.crt;
5     ssl_certificate_key /etc/ssl/private/wildcard.key;
6     location / {
7         ajp_keep_conn on;
8         ajp_pass 127.0.0.1:8282;
9     }
10 }
```

Listing 2.20: Konfiguration: NGINX HTTP Reverse Proxy mit AJP

### 2.3.3 Cherokee

Cherokee ist ein moderner Webserver und Reverse Proxy, der NGINX in vielen Punkten ähnelt. Die Besonderheit von Cherokee ist das grafische Konfigurationstool, welches über eine Weboberfläche zugänglich ist. Die Konfigurationsdatei sollte ausschließlich über dieses Tool bearbeitet werden.

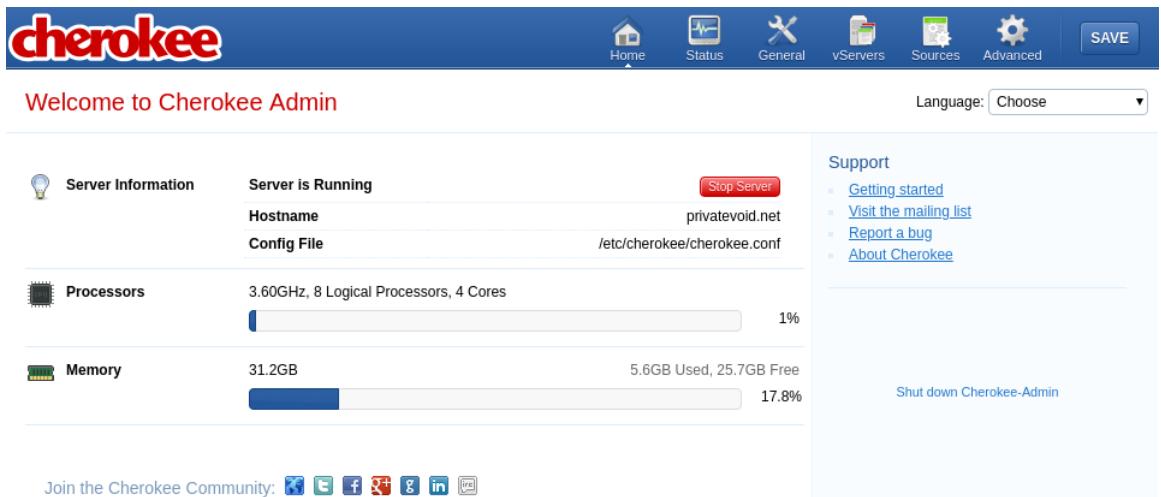


Abbildung 2.2: Cherokee Admin Interface

### Reverse Proxy mit HTTP

Dank der grafischen Konfiguration ist das Erstellen eines Virtual Host mit Reverse Proxy ein Kinderspiel. Cherokee bietet viele Templates, mit denen häufig genutzte Applikationsserver und Webframeworks schnell in ein Reverse Proxy Setup eingebunden werden können. In diesem Beispiel wurde wieder ein GlassFish Server gewählt, im darauffolgenden Konfigurationsdialog musste nur noch der Hostname des Virtual Host sowie IP-Adresse und Port des GlassFish Servers angegeben werden.

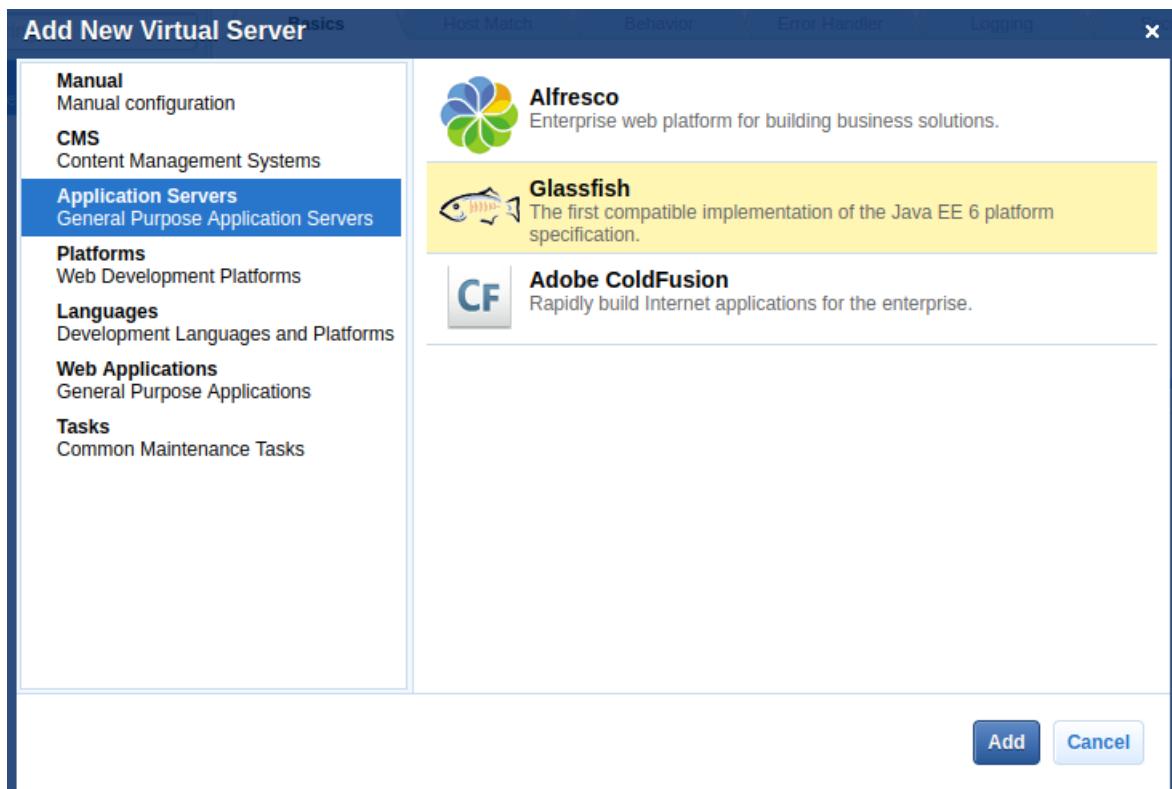


Abbildung 2.3: GlassFish Reverse Proxy mit Cherokee

### 2.3.4 Bewertungstabelle

Server	NGINX	Cherokee
Dokumentation	10	10
Bekanntheit	10	2
Konfiguration	7	9
Flexibilität	10	6

Tabelle 2.1: Bewertungstabelle NGINX und Cherokee

### 2.3.5 Begründung

- Dokumentation: NGINX ist exzellent dokumentiert, von Anleitungen für Anfänger bis zu detaillierten Erklärungen jedes einzelnen Parameters der vielen Module wird in der offiziellen Onlinedokumentation<sup>2</sup> jedes Themas ausgiebig behandelt. Cherokee's Onlinedokumentation<sup>3</sup> ist ebenfalls sehr umfangreich, trotz der Tatsache, dass Cherokee im Allgemeinen durchaus weniger komplex ist als NGINX. Beide Server erhalten volle Punkte.
- Bekantheit: NGINX ist ein direkter Konkurrent zu Apache, dem meistgenutzten Webserver der Welt. In manchen Bereichen ist Apache bereits auf Platz 2 gesunken. Somit ist klar, dass NGINX der Welt bekannt ist. Cherokee hingegen ist ein Nischenprodukt. Hier muss man sich voll und ganz auf die offizielle Dokumentation und Einfachheit der Bedienung verlassen.
- Konfiguration: Die Konfiguration von NGINX ist prinzipiell recht simpel strukturiert. Allerdings sind viele Direktiven nicht selbstverständlich. Vor allem bei komplizierteren Anwendungen kann die Konfiguration schnell unübersichtlich werden. Die grafische Konfiguration von Cherokee präsentiert die vielen Optionen dagegen recht übersichtlich, jedoch sind manche, selten verwendete Optionen nicht dort zu finden, wo man sie vielleicht erwarten würde.
- Flexibilität: Konfiguration in Textform wäre bereits abgeschafft worden, wäre sie nicht weit flexibler als ein grafisches Konfigurationstool. So bietet NGINX in Sachen Konfiguration große Flexibilität. Selbst wenn die gewöhnliche Konfiguration nicht ausreicht, ist es durch NJS möglich, den Server mit einer JavaScript-artigen Sprache auf dynamische Art und Weise zu konfigurieren. Die Konfigurationsoberfläche von Cherokee ist durchaus sehr mächtig, hält aber nicht mit NGINX mit.

---

<sup>2</sup><https://nginx.org/en/docs/>

<sup>3</sup><http://cherokee-project.com/doc/>

## 2.4 Applikationsserver

Die Idee eines Applikationsserver ist es, ein gemeinsames System für die serverseitige Komponente mehrerer Client-Server-Applikationen bereitzustellen, in dem Ressourcen zwischen den Applikationen geteilt werden können. Weiters soll durch diese Vereinheitlichung die Verwaltung mehrerer Applikationen erleichtert werden. Eine der bekanntesten Softwarearchitekturen für Applikationsserver ist die Java Enterprise Edition. Da im Projekt CarSharing ohnehin Java eingesetzt wird und da Java von allen Teammitgliedern beherrscht wird, werden nachfolgend nur Java EE Application Server analysiert. Für das Projekt kommen der GlassFish Server Open Source Edition, welcher die Referenzimplementation von Java EE ist, sowie der ebenfalls quelloffene Java EE Server WildFly von Red Hat infrage.

### 2.4.1 Java EE

Java Platform, Enterprise Edition ist ein Aufsatz auf die Java Platform, Standard Edition, der diese um viele Features erweitert, die für Serverapplikationen sinnvoll sind. Im weiteren Sinne ist Java EE eine Sammlung mehrerer Technologien und APIs für Java, welche die Entwicklung von Enterprise-Applikationen ermöglichen oder einfacher machen. Die einzelnen Technologien sind in den Java Specification Requests definiert. Ein großer Konkurrent von Java EE ist die .NET-Plattform von Microsoft. Folgende Technologien bilden die wichtigsten Komponenten von Java EE:

- JavaServer Faces: Erzeugung dynamischer Webdokumente durch Einsatz einer Expression Language in HTML-Dokumenten, vergleichbar mit PHP oder ASP.NET Razor
- Servlet API: Bietet die Möglichkeit, mittels Java-Code dynamisch Inhalte zu generieren. Viele Webframeworks wie JSF verwenden Servlets, um ihre Funktionalität anzubieten.
- Context and Dependency Injection: Ermöglicht das kontextbasierte „injizieren“ von abhängigen Objekten, verwaltet somit automatisch die lose Kopplung zwischen Modulen.
- Java Persistence API: Zusammen mit JPA und Java Database Connectivity können Geschäftsobjekte in einer Datenbank abgelegt werden.

## 2.4.2 GlassFish

GlassFish Server Open Source Edition ist die von Oracle selbst entwickelte quelloffene Referenzimplementation der Java Enterprise Edition, welche insbesondere als Plattform für Webanwendungen dient. Neben der quelloffenen Variante gab es eine kommerzielle Version, den Oracle GlassFish Server, der Enterprise-Support für GlassFish wurde von Oracle jedoch eingestellt. Als Reaktion darauf verkauft die Payara Services LTD nun Supportverträge für den Payara Server<sup>4</sup>, einem Fork von GlassFish, der bis auf das Aussehen der Administrationskonsole vollkommen identisch ist. GlassFish ist neben IBM WebSphere und WildFly einer der wenigen Applikationsserver, die Java EE Version 8 vollständig implementiert haben.

## 2.4.3 Administration des GlassFish Servers

Der GlassFish Server ist unter anderem aufgrund seiner graphischen, webbasierten Administration Console beliebt, welche die Administration wesentlich erleichtert. Neben dieser Weboberfläche gibt es auch ein Kommandozeilentool, womit die Administration von GlassFish auch automatisiert werden kann. Details zu den folglich beschriebenen Tools findet man in [Su:JavaEE2, GlassFish Server Administration Guide].

### Die GlassFish Administration Console

GlassFish verfügt über eine webbasierte Administrationskonsole, über welche diverse Aspekte einer einzelnen Serverinstanz oder eines Serverclusters konfiguriert werden können, wie beispielsweise HTTP-Listener oder Datenbankverbindungen. Ebenfalls können Webapplikationen direkt über die Weboberfläche hochgeladen und administriert werden. Die Administrationskonsole ist über den Admin-HTTP-Port (standardmäßig 4848) mit einem Browser erreichbar.

### Das Kommandozeilentool `asadmin`

Neben der grafischen Administrationskonsole kann der GlassFish Server auch über ein spezielles Kommandozeilentool `asadmin` administriert werden. Dieses Tool wird auch dazu verwendet, um einen GlassFish Server zu starten. Das Tool verfügt über zahlreiche „Subcommands“, welche in zwei Kategorien eingeteilt sind.

---

<sup>4</sup><https://www.payara.fish/>

- Lokale Subcommands können ohne einen laufenden DAS<sup>5</sup> ausgeführt werden und greifen auf die Dateien der lokalen GlassFish-Instanz zu.
- Remote-Subcommands greifen auf einen DAS zu, über den der Befehl ausgeführt wird. Da hier über das Netzwerk kommuniziert wird, muss der Administrator das Tool nicht vom jeweiligen Server ausführen, sondern kann diese Befehle auch von einer Client-Maschine ohne eine vollständige GlassFish-Installation ausführen.

Das Tool `asadmin` verfügt über zwei verschiedene Modi, um Befehle auszuführen.

- Im Single-Mode wird ein Subcommand inklusive dessen Parameter als Parameter des `asadmin`-Befehls eingegeben. Dieser Modus ist für einfache Befehle sowie die Verwendung in Skripts geeignet.
- Im Multi-Mode wird `asadmin` entweder ohne Parameter oder mit dem Subcommand `multimode` aufgerufen. Dies startet eine Art Shell-Session, in der Subcommands direkt eingegeben werden können, bis die Session mit `exit` oder `quit` beendet wird.

Des weiteren verfügt `asadmin` über eine einfache integrierte Job Control, über die mehrere eventuell langwierige Prozesse im Hintergrund gestartet werden können. Um einen Subcommand im Hintergrund auszuführen muss lediglich der Parameter `--detach` zusätzlich angeführt werden. Im Single Mode retourniert man somit sofort zur System-Shell. Im Multi Mode stehen weitaus mehr Features zur Verfügung. Mit dem Subcommand `list-jobs` können alle bisher gestarteten Hintergrundjobs aufgelistet werden. Zusätzlich werden Statusinformation zu jedem einzelnen Job angezeigt.

asadmin> list-jobs					
	JOB ID	COMMAND	STATE	EXIT CODE	TIME OF COMPLETION
1	1	create-cluster	COMPLETED	SUCCESS	2013-02-15 16:16:16 PST
2	2	deploy	COMPLETED	FAILURE	2013-02-15 18:26:30 PST
Command list-jobs executed successfully					

Listing 2.21: Beispielausgabe für `asadmin list-jobs`

<sup>5</sup>Domain Administration Server

<sup>6</sup>[Su:JavaEE2, GlassFish Server Administration Guide, Example 2?12]

Mit `attach` kann man sich zu einem noch aktiven Hintergrundjob verbinden, um dessen Fortschrittsmeldungen zu erhalten.

Diese und alle weiteren Subcommands von `asadmin` sind in [Su:JavaEE1, GlassFish Server Reference Manual, Sektion 1] ausführlich dokumentiert.

#### **2.4.4 Deployment von Webapplikationen mit GlassFish**

Java EE-Webapplikationen werden üblicherweise in Form einer .war-Datei an den Server übergeben, um dort ausgeführt zu werden. Diese Variante des Applikationsdeployment wird Archive Deployment genannt. GlassFish unterstützt mehrere Deployment-Methoden für diese und andere Varianten.

## Über die Administrationskonsole

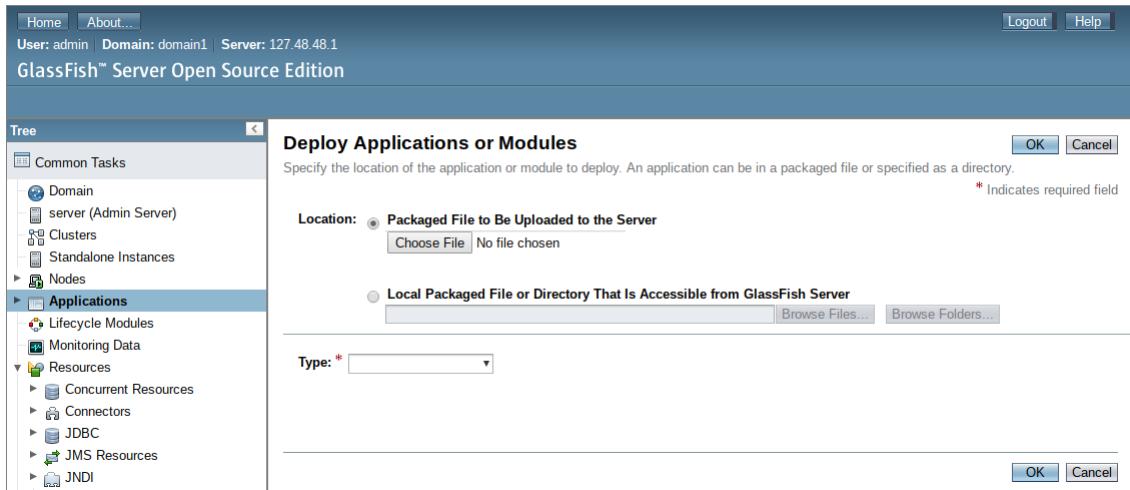


Abbildung 2.4: Deployment über GlassFish Admin Console

Administratoren können von jedem Browser aus über die Admin Console eine .war-Datei zum Server hochladen, um ein Deployment mit dieser einzurichten. Alternativ kann eine bereits am Server befindliche .war-Datei oder ein Verzeichnis mit entsprechendem Inhalt mit einem einfachen Filebrowser ausgewählt werden. Nach der Angabe einer Datei oder eines Verzeichnisses werden einige Optionen zur initialen Konfiguration des Deployments gezeigt. Dies ist bei weitem die einfachste Methode zum Applikationsdeployment, da hier keine speziellen Tools am Client installiert sein müssen und kein spezieller Zugriff auf den Server (wie etwa SSH) benötigt wird.

## Über das Kommandozeilentool `asadmin`

`asadmin` stellt ein Subkommando `deploy` zur Verfügung, das so wie die Admin Console die Möglichkeit, ein Deployment aus einer .war-Datei oder aus einem Verzeichnis zu erstellen bereitstellt. Über Switch-Parameter können diverse Eigenschaften wie Context Root oder ein eigener Deploymentname hinzugefügt werden. Alle Parameter des `deploy`-Befehls sind in [Su:JavaEE1, Seite 1-324] genauer beschrieben. Diese Methode des Deployment Managements fordert entweder eine lokale Installation von `asadmin` oder Remote-Shellzugriff auf den jeweiligen Server. Wird eine lokale Installation verwendet, so wird eine HTTP-Verbindung zum Admin-Port des GlassFish Servers aufgebaut.

### Über IDEs wie NetBeans

Viele IDEs unterstützen die Funktionalität, einen lokalen Applikationsserver auszuführen und auf diesem automatisch ein Deployment mit der Webapplikation einzurichten, die aus einem IDE-Projekt hervorgeht. Dies ermöglicht es Entwicklern, ihre Applikationen schnell zu testen. Manchmal ist es allerdings zwingend erforderlich, die Webapplikation auf einem entfernten Server auszuführen. Für diesen Fall wird von den meisten IDEs auch die Möglichkeit des Remote Deployment angeboten. GlassFish wird hierbei laut offiziellen Angaben von NetBeans und Eclipse unterstützt. Nähere Informationen über das Deployment von Enterprise-Applikationen mit GlassFish im Allgemeinen sind in [Su:JavaEE3, GlassFish Server Application Deployment Guide] auffindbar. Das Deployment spezifisch über IDEs ist auf Seite 1-13 zu finden.

### Gründe für den Einsatz von GlassFish

Mit GlassFish verwendet man die Referenzimplementation von Java EE, was zahlreiche Vorteile mit sich bringt. So bieten viele Development Tools die beste oder sogar exklusive Kompatibilität mit GlassFish. Aus selbigem Grund ist auch die Unterstützung durch die Community sehr gut. Zusätzlich ist GlassFish als die Referenz immer der erste Server, der eine neue Version von Java EE unterstützt.

### 2.4.5 WildFly

WildFly ist ein Java EE-Applikationsserver, welcher ursprünglich vom Unternehmen JBoss entwickelt wurde, welches nun zu Red Hat gehört. WildFly war vor Version 8 bekannt unter dem Namen JBoss Application Server und ist die Open Source-Variante der JBoss Enterprise Application Platform. JBoss AS war bereits vor der Einstellung des Enterprise-Supports für GlassFish der beliebteste Java EE Server. Seitdem der Support von Oracle eingestellt wurde, ist GlassFish nicht mehr für den kommerziellen Einsatz geeignet, was die Popularität von WildFly und JBoss EAP weiter vergrößerte.

### 2.4.6 Administration des WildFly Servers

WildFly stellt zwei Operationsmodi zur Verfügung, zwischen denen sich nur hinsichtlich Konfiguration und Administration ein Unterschied gibt.

- Domain: In diesem Modus können mehrere Server zentral konfiguriert werden. Jeder einzelne Server befindet sich in einer Servergruppe, Deployments werden Gruppen zugeordnet.
- Standalone: In diesem Modus wird jeder Server einzeln konfiguriert und es muss keine Management Domain eingerichtet werden. Dieser Modus ist vor allem für die Fälle geeignet, in denen nur ein Applikationsserver zum Einsatz kommt.

Da im Falle des CarSharing-Projekts nur ein Server eingesetzt wird, wird sich der folgende Textabschnitt auf den Standalone-Modus beschränken. Detaillierte Informationen zu beiden Modi sowie eine Anleitung zum Konfigurieren eines WildFly-Servers können in [Su:WildFly1, WildFly Admin Guide] gefunden

### Die HAL Management Console

Ähnlich wie GlassFish verfügt WildFly über eine webbasierte Administrationskonsole, die HAL Management Console. Diese wird über den HTTP Management Endpoint bereitgestellt, der auch als API-Zugriffspunkt für andere Management-Tools wie JBoss CLI verwendet wird.

### Das JBoss Command Line Interface

Auch wie bei GlassFish gibt es bei WildFly ein Kommandozeilentool zur Administration. Dieses wird JBoss CLI genannt und wird je nach Betriebssystem über `jboss-cli.sh` oder `jboss-cli.bat` gestartet. Im Gegensatz zu `asadmin` von GlassFish liefert das JBoss CLI keine lokalen Befehle mit sich und muss daher immer mit einem laufenden WildFly-Server verbunden sein. Dies bedeutet auch, dass vom CLI kein Standalone-Server gestartet werden kann. Dieser wird stattdessen mit einem eigenen Skript `standalone.sh` direkt gestartet.

Das JBoss CLI verwendet dieselbe API-Schnittstelle wie die HAL Management Console. Nach Aufruf des Befehls `connect` versucht das JBoss CLI eine Verbindung zu einem HTTP Management Endpoint auf `localhost` Port 9990 aufzubauen. Bei Angabe einer IP-Adresse kann man sich ebenfalls zu entfernten WildFly-Servern verbinden.

### 2.4.7 Applikationsdeployment mit WildFly

WildFly unterstützt Applikationsdeployment mithilfe folgender Methoden:

- Über das JBoss CLI mit `deploy`
- Automatisches Deployment mit Deployment Scanner
- Remote Deployment wird von manchen Integrated Development Environments unterstützt

## Über die HAL Management Console

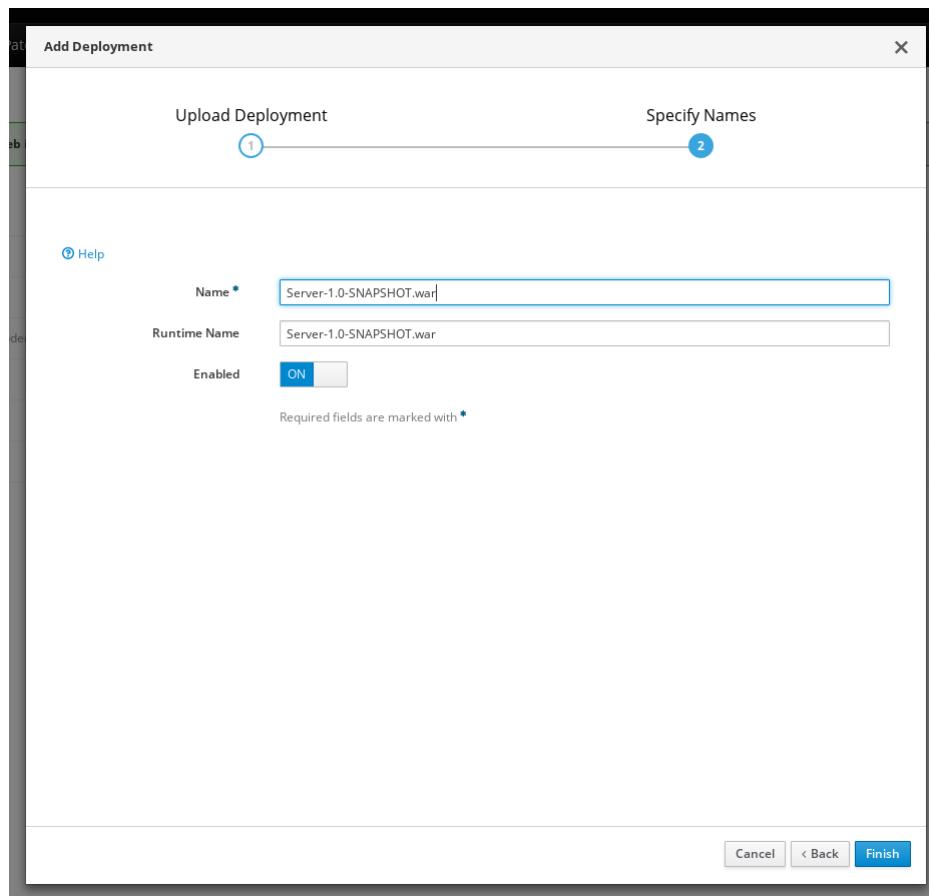


Abbildung 2.5: Deployment über GlassFish Admin Console

Die HAL Management Console bietet viele Features zur Administration von Deployments mit sich. Nach einem intuitiv gestalteten Uploadvorgang können die Inhalte einzelner Deployments direkt auf der Website inspiziert, heruntergeladen und auch modifiziert werden.

### Gründe für den Einsatz von WildFly

Ein wesentlicher Vorteil von WildFly gegenüber von GlassFish ist die stark verkürzte Startzeit, die teilweise sehr nahe an die von minimalistischen Servletcontainern, die keine volle Implementation von Java EE mitliefern herankommt. Zusammen mit der kurzen Startzeit läuft das Deployen von Applikationen auch sehr schnell ab. Auch zeichnet sich WildFly durch geringen Ressourcenverbrauch aus. Durch diese Eigenschaften ist WildFly sehr gut geeignet für die Verwendung während der Entwicklung, da hier oftmals ein lokaler Server unter eventueller Ressourcenknappheit eingesetzt wird. Dieser wird eventuell mehrmals heruntergefahren und wieder gestartet, die enorm verkürzte Startzeit verringert hierbei unnötige Wartezeiten der Entwickler erheblich.

WildFly verfügt über eine relativ große Community, wodurch gute Entwicklerunterstützung geboten wird. Die Weiterentwicklung von WildFly verläuft sehr schnell, dadurch ist er oft einer der wenigen Server, auf dem die neuesten Frameworks bereits verfügbar sind.

#### 2.4.8 Bewertung

In diesem Abschnitt werden die beiden zuvor beschriebenen Applikationsserver in Bezug auf Sinnhaftigkeit des Einsatzes im Projekt bewertet. Dies geschieht in Form einer tabellarischen Darstellung der Bewertung, die einen schnellen Überblick verschaffen soll, sowie einer damit verbundenen Detailbeschreibung und Begründung der Bewertung. Berücksichtigt werden hierbei:

- Entwicklung: Dies beinhaltet IDE-Integration sowie die erstmalige Einrichtung des Servers zur Entwicklung.
- Dokumentation: Beinhaltet sowohl die offiziell verfügbare Dokumentation als auch möglichen Community-Support.
- Administration: Bewertet, wie leicht der Server und die Deployments zu administrieren und in das bestehende System einzubauen sind.
- Integration mit Reverse Proxy: Bewertet, wie gut sich der Server mit Reverse Proxies verhält.
- Integration mit Datenbank: Bewertet, wie gut sich der Server mit dem eingesetzten Datenbanksystem kombinieren lässt.

### Bewertungstabelle

Server	GlassFish Server	WildFly Application Server
Entwicklung	9	10
Administration	7	10
Dokumentation	10	9
Integration Reverse Proxy	8	10
Integration Datenbank	10	8

Tabelle 2.2: Bewertungstabelle GlassFish und WildFly

### Begründung

- Entwicklung: Beide Server verfügen über gute Integration mit NetBeans, wodurch beide für den Einsatz während der Entwicklung sehr gut geeignet sind. GlassFish wird hierbei ein Punkt abgezogen, da dieser sehr lange zum Starten braucht, was bei WildFly nicht in solchem Ausmaß der Fall ist.
- Administration: WildFly erhält hier die volle Punktzahl, da seine Administrations-tools modern und einfach zu verwenden sind. Dank des sehr simpel gehaltenen Skripts `standalone.sh` konnte WildFly sehr schnell in das Service Management System integriert werden und sogar unter Process Supervision gesetzt werden. GlassFish weist hier ein paar Eigenheiten auf, die händisch behoben werden mussten. Mehr dazu im Kapitel Umsetzung.
- Dokumentation: Beide Applikationsserver sind aufgrund der Unterstützung durch große Firmen sehr gut dokumentiert. Beide Server sind sehr beliebt, was für beide guten Community-Support bedeutet, GlassFish ist aufgrund seines Status als Referenzimplementation jedoch etwas besser dran, somit wurden WildFly hier Punkte abgezogen.
- Integration mit Reverse Proxy: NGINX konnte nach geringer Konfiguration mit beiden Applikationsservern arbeiten. WildFly bot jedoch hier und da zusätzliche Features, um die Integration zu erleichtern. Beispielsweise war es möglich, die extern erreichbare URL, die bei der Erzeugung von Links von WildFly verwendet wird, umzustellen. Die Management Console wurde von WildFly nicht über HTTPS geliefert, da WildFly erkannte, dass der Zugriff von localhost getätigt wurde. In GlassFish musste hier der Secure Mode manuell deaktiviert werden, da NGINX dem selbst signierten Zertifikat nicht vertrauen würde.
- Integration mit Datenbank: Dies fiel unter GlassFish sehr leicht, da alle Einstellungen innerhalb der `persistence.xml` stattfinden konnten. Unter WildFly

bereitete der Einsatz einer Datenbank einige Probleme, diese werden mitsamt Lösungen im Umsetzungskapitel erklärt.

# Kapitel 3

## Sichere Kommunikation

### 3.1 Allgemeine Konzepte

Bei sicherer Kommunikation geht es darum, geheime Nachrichten über ein unsicheres Kommunikationsmedium (das Internet) zu übertragen, ohne dass unerwünschte Entitäten diese Nachrichten mitlesen oder gar verfälschen können. Hierfür gibt es drei Punkte, die durch ein Übertragungsverfahren gegeben sein müssen, um solch einen Sachverhalt zu ermöglichen.

- Vertraulichkeit: Der Sender kann bestimmen, wer die Nachrichten lesen darf
- Integrität: Der Sender bestätigt den Inhalt seiner Nachrichten, der Empfänger kann überprüfen, dass diese nicht verändert wurden
- Authentizität: Der Empfänger kann überprüfen, dass eine Nachricht wirklich vom angeblichen Sender stammt

Genauere Begriffserklärungen findet man in  
[Su:Web2, Piete Brooks: Security: Privacy, Authenticity and Integrity].

## 3.2 Asymmetrische Verschlüsselung

Bei asymmetrischer Verschlüsselung spricht man von Schlüsselpaaren, bestehend aus dem Private Key und dem Public Key. Nachrichten, die mit einem von zwei zusammengehörigen Keys verschlüsselt werden, können mit dem jeweils anderen Schlüssel wieder entschlüsselt werden. Die Bezeichnungen Private Key und Public Key sind nicht willkürlich gewählt, der Private Key sollte privat bleiben, da der Public Key vom Private Key aus generiert werden kann. Besitzt man den Private Key, so besitzt man effektive beide Teile des Schlüsselpaares.

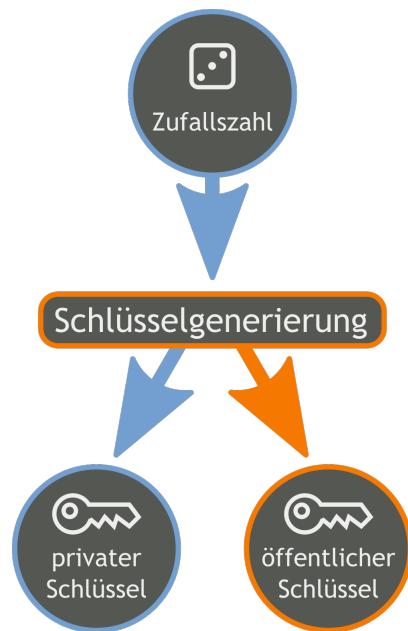


Abbildung 3.1: Erzeugung asymmetrischer Schlüsselpaare

Der Zweck asymmetrischer Verschlüsselung ist der sichere Austausch zweier Nachrichten, sofern beide Enden den Public Key des jeweils anderen kennen. Dies kann beispielsweise zum Austausch eines symmetrischen Schlüssels verwendet werden, da symmetrische Verschlüsselung weitgehend bessere Performance bietet. Mehr dazu im Abschnitt über symmetrische Verschlüsselung.

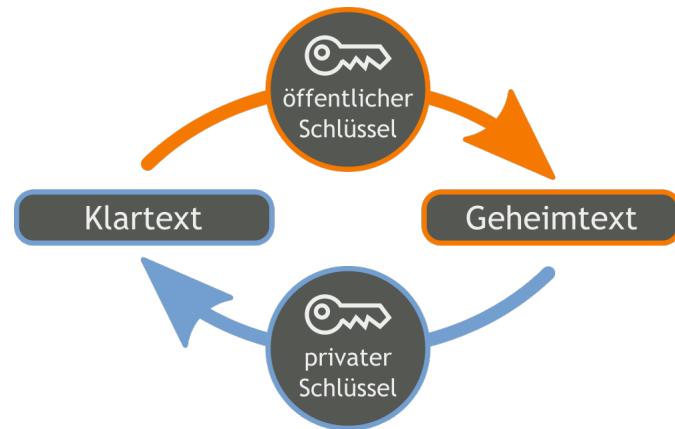


Abbildung 3.2: Asymmetrische Verschlüsselung

Meistens werden generische asymmetrische Verschlüsselungsverfahren wie RSA allerdings nur zur Authentifizierung verwendet. Dies geschieht vor allem bei Webservern mit X.509-Zertifikaten. Diese digitalen Zertifikate werden von einer Certificate Authority ausgestellt. Hierbei handelt es sich meistens um generell als vertrauenswürdig angesehene Organisationen oder CAs, die von solchen Organisationen dazu zertifiziert wurden, weitere Zertifikate auszustellen. Dies nennt sich „Chain of Trust“. Im Zertifikat ist unter anderem der Public Key des Webservers eingebunden. Ein Client kann dessen Authentizität nun bestätigen, indem er eine Nachricht mit dem vom Server vorgezeigten Public Key verschlüsselt und diese an den Server sendet. Kann der Server die Originalnachricht rekonstruieren und dem Client vorzeigen, muss er den Private Key besitzen, durch welchen er die Nachricht entschlüsseln konnte.

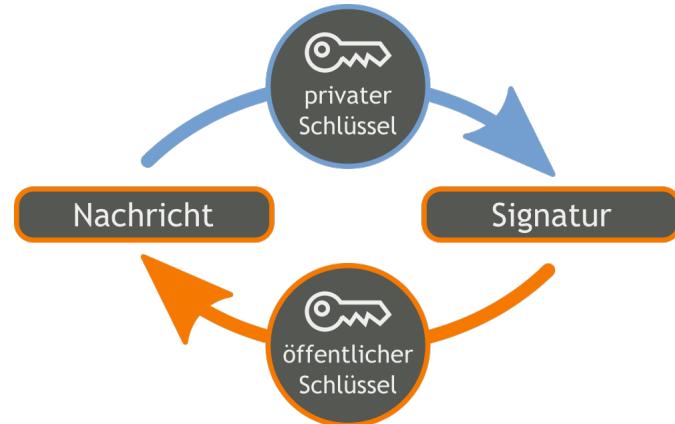


Abbildung 3.3: Authentifizierung mittels Signatur

### 3.3 Symmetrische Verschlüsselung

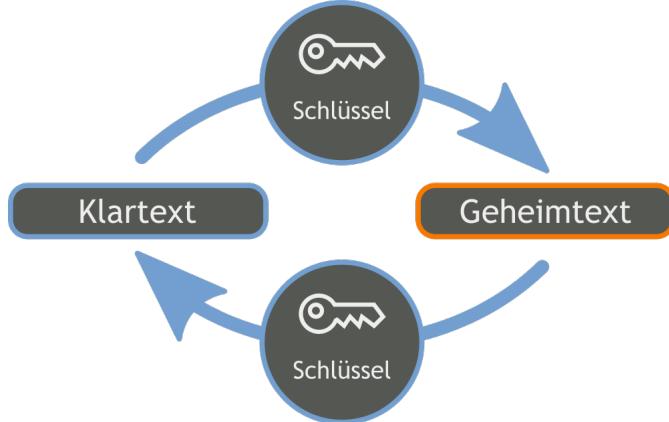


Abbildung 3.4: Symmetrische Verschlüsselung

Der Hauptunterschied von symmetrischer Verschlüsselung zur asymmetrischen Verschlüsselung ist, dass es hier nur einen Schlüssel gibt, der zugleich Verschlüsseln und Entschlüsseln übernimmt. Dies eignet sich also alleine nicht dafür, eine Nachricht vor ungewollten Lesern zu schützen, da ein Teilnehmer der Konversation seinen Schlüssel mit dem anderen Teilnehmer teilen muss. Vorteil der symmetrischen Verschlüsselung ist jedoch, dass sie wesentlich performanter ist, sie eignet sich somit auch für die Übertragung größerer Dateien unter voller Bandbreitenauslastung, ohne dass die Prozessoren der Kommunikationspartner völlig überfordert werden.

### 3.4 Hashing

Eine Hash-Funktion ist eine mathematische Funktion, die nur in eine Richtung gerechnet werden kann. Die ursprüngliche Nachricht kann also nicht aus dem Hash-Wert errechnet werden. Im Gebiet der sicheren kryptologischen Kommunikation werden Hash-Funktionen für digitale Signaturen eingesetzt. Der Ergebniswert einer Hashfunktion hat eine fixe Länge von meist nur ein paar Bytes. Da es ineffizient ist, große Nachrichten mittels asymmetrischer Verfahren zu verschlüsseln wird stattdessen der Hash-Wert der Nachricht gebildet und dieser verschlüsselt.

## 3.5 Schlüsseltauschverfahren

Um symmetrische Verschlüsselung im gewollten Anwendungsgebiet verwenden zu können, benötigt man eine Möglichkeit, den symmetrischen Schlüssel sicher zum anderen Ende zu bringen. Dies kann durch asymmetrische Verschlüsselung durchgeführt werden. Man spricht hierbei von einem Schlüsseltausch. Nach diesem Austausch wird symmetrisch verschlüsselt. Ein Mithörer weiß weiterhin nicht, welcher symmetrische Schlüssel verwendet wird und die beiden Enden können mit voller Performance gesichert kommunizieren.

Nimmt ein Mithörer jedoch die gesamte Kommunikation auf und kommt weit später, etwa durch einen Leak, auf den Private Key des Servers, der zum damals noch gültigen Zertifikat gehört, so ist es ihm möglich, den symmetrischen Schlüssel aus der asymmetrisch verschlüsselten Nachricht zu extrahieren und somit alle weiteren Nachrichten auszulesen.

Eine Methode, die dies unmöglich macht ist die dynamische Erzeugung eines neuen asymmetrischen Schlüsselpaares zur Zeit des Verbindungsaufbaus. Über das dynamische Schlüsselpaar kann ein symmetrischer Schlüssel ausgetauscht werden. Dieses Verfahren ermöglicht die sogenannte Perfect Forward Secrecy. Auch wenn ein Mithörer alle Nachrichten aufzeichnet und später zum Private Key des Servers kommt, kann er nichts damit anfangen, da es immer noch einen Private Key gibt, der nie übertragen wurde und bereits nach dem Ende der Konversation gelöscht wurde.

RSA ist jedoch nicht dazu geeignet, sehr schnell ein neues Schlüsselpaar zu erzeugen. Daher gibt es spezielle asymmetrische Verfahren, die genau für diesen Zweck geschaffen wurden. Das verbreitetste hierbei ist der Diffie-Hellman-Exchange DHE, sowie eine noch sicherere Variante davon, der Elliptic-Curve-Diffie-Hellman-Exchange ECDHE. Zusammen mit einem „statischen“ asymmetrischen Verfahren und einem Hash-Algorithmus zur Authentifizierung sowie einem symmetrischen Verschlüsselungsalgorithmus wird eine sogenannte Cipher Suite gebildet.

## 3.6 Praktischer Einsatz von Verschlüsselung

### 3.6.1 Transport Layer Security

TLS (Transport Layer Security) ist ein häufig genutztes Sicherheitsverfahren und Verschlüsselungsprotokoll für Netzwerkkommunikation, das logisch gesehen ins ISO/OSI-Modell über Layer 4, dem Transport Layer eingebaut wird. Es zeichnet sich da-

durch aus, dass die darüberliegenden Schichten, beispielsweise die Applikationsschicht ohne Veränderung der Applikationsprotokolle weiterhin funktionieren können. Der Vorgänger von TLS ist SSL (Secure Socket Layer). TLS ist prinzipiell nur eine Umbenennung von SSL, daher wird heutzutage noch oft der Begriff SSL verwendet, um TLS zu beschreiben.

TLS gewährt Integrität, Vertraulichkeit und Authentizität der gekapselten Daten. Die Authentizität wird mit X.509-Zertifikaten erreicht, die aufgrund ihres primären Einsatzes mit TLS auch oft SSL-Zertifikate genannt werden. Zur Bereitstellung der Zertifikate wurde ein sogenanntes Web of Trust aufgebaut.

### 3.6.2 HTTPS

Eine der bekanntesten Einsatzformen von TLS ist HTTPS. Das World Wide Web hat sich dahingehend entwickelt, den Großteil aller verfügbaren Ressourcen mit HTTPS verschlüsselt zugänglich zu machen. Entwickler von Webbrowsern und Betreiber von Webservern bewegen die Welt immer mehr in diese Richtung. Vor allem Google will im Browser Chrome in Zukunft Plain-HTTP-Seiten in der Adressleiste als unsicher kennzeichnen und tut dies in manchen Fällen bereits. Betreiber von Webservern setzen oft Konfigurationen ein, in denen jede unverschlüsselte Anfrage automatisch auf die äquivalente HTTPS-Anfrage umgeleitet wird.

Mit HSTS (HTTP Strict Transport Security) wurde ein Sicherheitsmechanismus geschaffen, der Webbrowsern, die zum ersten Mal eine Website aufrufen, sagt, dass diese Website für eine bestimmte Zeitdauer (üblich sind hier sehr lange Zeitspannen wie 2 Jahre) nur verschlüsselt über HTTPS aufgerufen werden soll. Auch soll kein weiterer Datenaustausch erfolgen, sollte die Richtigkeit des Zertifikates des Webservers nicht bestätigt werden können. Das Ziel ist es, dem Benutzer den Zugang im Falle eines sicherheitstechnischen Problems extrem schwer bis unmöglich zu machen, um den ungesicherten Datenaustausch auf alle Fälle zu verhindern. Weiters bietet Google mit dem Projekt HSTS-Prelad eine Liste von Webservern, bei denen ein Browser, der die Liste implementiert, sogar noch vor dem ersten Aufruf die von HSTS gegebene Regelung verwendet. Dies umgeht sogar die letzte kleine potenzielle Sicherheitslücke, in der ein Angreifer die Weiterleitung des Servers beim erstmaligen Zugriff des Benutzers abfängt. Jeder Webseitenbetreiber darf sich kostenlos in diese Liste eintragen.

### 3.6.3 Einsatz von HTTPS und HSTS bei Webservern

Zur Demonstration der Konfiguration von HTTPS bei einem Webserver wird NGINX herangezogen. Bei NGINX wird Support für SSL/TLS durch das Modul `ngx_http_ssl_module` ermöglicht. Ein gewöhnlicher Serverblock ohne HTTPS könnte in etwa so aussehen:

```
1 server {  
2     server_name www.privatevoid.net;  
3     listen 80;  
4     root /var/www/privatevoid/htdocs/www;  
5 }
```

Listing 3.1: Serverblock NGINX ohne HTTPS

Im obigen Beispiel wird ein Domainname für den Virtual Host gesetzt und ein Root-Directory definiert. Danach wird Port 80 für HTTP-Anfragen geöffnet.

Um dieselbe Website nun korrekt über HTTPS freizugeben, werden zwei verschiedene Serverblöcke konfiguriert, einer als Umleitung für unverschlüsselte Anfragen und ein weiterer, der den wirklichen Inhalt liefert.

```

1 server {
2     server_name www.privatevoid.net;
3     listen 80;
4     location / {
5         return 301 https://$server_name$request_uri;
6     }
7 }
8
9 server {
10    server_name www.privatevoid.net;
11    listen 443 ssl;
12    ssl_certificate /etc/letsencrypt/live/privatevoid.net/fullchain.pem;
13    ssl_certificate_key /etc/letsencrypt/live/privatevoid.net/privkey.pem;
14    ssl_protocols TLSv1.2 TLSv1.3;
15    ssl_ciphers "ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:
16        DHE-RSA-AES256-GCM-SHA256";
17    root /var/www/privatevoid/htdocs/www;
18 }
```

Listing 3.2: Serverblock NGINX mit HTTPS

Im ersten Serverblock werden alle Anfragen, die unterhalb der URL / liegen, also alle Anfragen überhaupt, auf das HTTPS-Äquivalent des Servers weitergeleitet werden sollen. Hier kann von den von NGINX gegebenen Variablen `$server_name` und `$request_uri` Gebrauch gemacht werden. Es wird der HTTP-Statuscode 301 (Moved Permanently) verwendet, um dem Browser zu zeigen, dass er immer die HTTPS-Version der Seite aufrufen soll.

Die erste auffällige Änderung im zweiten Block ist nach der anderen Portnummer der Direktive `listen` der zusätzliche Parameter `ssl`. Hiermit wird NGINX informiert, dass auf diesem Port mit SSL oder TLS gekapselter Datenverkehr zu erwarten ist. Die beiden Zeilen darunter geben an, welches Zertifikat NGINX dem Client präsentieren soll und wo der zugehörige Private Key liegt.

Bei den nächsten zwei Zeilen handelt es sich bereits um zusätzliche Sicherheitsvorkehrungen, nämlich werden die verwendbaren TLS-Protokolle auf die zwei aktuellsten Versionen TLSv1.2 und TLSv1.3 eingeschränkt. Danach wird die Liste der einsetzbaren Cipher Suites stark eingeschränkt. Die von der Library OpenSSL bereitgestellten Cipher Suites beinhalten bis heute noch veraltete Verschlüsselungsverfahren wie RC4. In manchen Fällen kann die Verschlüsselung sogar deaktiviert werden. Während des Handshakes könnte ein Hacker beiden Seiten einreden, dass das Gegenüber nur eine solche schwache Verschlüsselungsmethode unterstützt, die der Hacker leicht knacken kann. Um dies zu verhindern wird die Liste der möglichen Cipher Suites serverseitig eingeschränkt.

Schlussendlich wird noch HSTS aktiviert, indem ein entsprechender Header dem zweiten Serverblock hinzugefügt wird. Auch HSTS Preload ist in dieser Konfiguration möglich, da die Weiterleitung von Port 80 sowie alle anderen Anforderungen für die Inklusion in die Preload-Liste erfüllt sind.<sup>1</sup>

```
1 server {
2     server_name www.privatevoid.net;
3     listen 443 ssl;
4     ssl_certificate /etc/letsencrypt/live/privatevoid.net/fullchain.pem;
5     ssl_certificate_key /etc/letsencrypt/live/privatevoid.net/privkey.pem;
6     ssl_protocols TLSv1.2 TLSv1.3;
7     ssl_ciphers "ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:
8                 DHE-RSA-AES256-GCM-SHA256";
9     root /var/www/privatevoid/htdocs/www;
10    add_header Strict-Transport-Security "max-age=63072000;
11          includeSubDomains; preload";
12 }
```

Listing 3.3: Serverblock NGINX mit HTTPS und HSTS

---

<sup>1</sup>Anforderungen zu finden auf: <https://hstspreload.org/>

# Kapitel 4

## Service Management

### 4.1 Allgemeines

#### 4.1.1 Daemons

Ein Daemon ist ein Programm, welches als Hintergrundprozess abläuft und meistens über das Netzwerk mit anderen Programmen kommuniziert. Daemons können verschiedene Rollen haben:

- System Logs: syslog-ng, journald
- Server für den Fernzugriff auf den Rechner: SSH, Telnet
- Server für lokale Interprozesskommunikation: DBus
- Agents: Session Manager für SSH oder GPG
- Sonstige Server: HTTP, FTP, DNS, etc.

Daemons sollten vom System automatisch gestartet werden. Hierzu werden Service Management Systeme in Kombination mit `init` verwendet werden. Eine der ältesten und somit einfachsten Implementationen eines solchen Systems ist der System V Init.

### 4.1.2 System V Init und RC

In UNIX-ähnlichen Systemen wird während des Bootvorganges vom Kernel ein einziger Prozess `init` gestartet, der den Rest des Systems initialisieren soll. Dies ist meistens mit der Verwendung eines Service-Management-Systems verbunden. Weiters ist dieser Prozess aus Sicht des Kernels der einzige, der das System herunterfahren darf.

Der System V Init (sysvinit) ist eine Implementation von `init`, der erstmalig in AT&T's UNIX System III zum Einsatz kam. Der Name kommt davon, dass sich diese Implementation bis zu UNIX System V durchsetzte. `init` selbst ist ein Daemon, der mithilfe eines Servicemanagers wie `rc` je nach konfiguriertem Runlevel X nach der Reihe alle Skripts in `/etc/rcX.d` ausführt. Hierbei handelt es sich üblicherweise um Symlinks (Verknüpfungen) auf Skripts in `/etc/init.d`. Der Sinn dieser Struktur ist es, die in einem Runlevel gestarteten Services einfach ändern zu können, ohne Skripts an einen arbiträren Ort verschieben oder gar löschen zu müssen.

Diese erste und sehr früh entwickelte Methode des Service Management ist sehr einfach und bietet daher kein Dependency Management und keine Parallelisierung. Sollte ein Service s1 kritische Funktionalität für einen Service s2 bereitstellen, so muss darauf geachtet werden, dass s1 vor s2 gestartet wird. Üblicherweise werden hier die Symlinks einfach durch Nummerierung (z.B. `10-s1` und `50-s2`) in die richtige alphanumerische Reihenfolge gebracht, in der sie schließlich von `init` abgearbeitet werden.

Durch die fehlende Parallelisierung kann es zu erheblichen Verzögerungen im Start des Systems kommen. Beispielsweise könnte ein Service noch vor dem Fork, also bevor das `rcX.d`-Skript als abgeschlossen gilt, länger auf eine beliebige Ressource warten. Während dieser Wartezeit könnten problemlos weitere Services, die nicht von ersterem abhängig sind, gestartet werden. Weitere Informationen sind unter [Su:Web1, Unix and Linux startup scripts] zu finden.

### 4.1.3 Runlevels

In `/etc/inittab` können bis zu 7 Runlevels (0-6) definiert werden. Drei davon sind Standard-Runlevels, die wie folgt definiert sind:

- 0 (Halt): Führt einen System Halt (Shutdown) durch.
- 1 (Single User): In diesem Modus ist nur ein Terminal zugänglich, auf dem sich ein Administrator anmelden kann, um kritische Veränderungen am System durchzuführen.

- 6 (Reboot): Führt einen Neustart durch

Die Runlevels 2 bis 5 können beliebig definiert werden. Üblicherweise ist Networking in Runlevel 2 abgeschaltet und Runlevel 3 ist das Standard-Runlevel für den normalen Systembetrieb.

#### 4.1.4 Beispiel für ein Initskript

Dieses Skript wird durch die Shell `bash` interpretiert und ist somit eigentlich ein gewöhnliches Bash-Skript. Um RC-spezifische Funktionen in Initskripts verfügbar zu machen, wird die Datei `/etc/init.d/functions` dazugeladen.

```

1 #!/bin/bash
2 . /etc/init.d/functions
3
4 start() {
5     initlog -c "echo -n Starting FOO server: "
6     /path/to/FOO &
7     touch /var/lock/subsys/FOO
8     success $"FOO server startup"
9     echo
10 }
11
12 stop() {
13     initlog -c "echo -n Stopping FOO server: "
14     killproc FOO
15     rm -f /var/lock/subsys/FOO
16     echo
17 }
18
19 case "$1" in
20     start)
21         start
22         ;;
23     stop)
24         stop
25         ;;
26     status)
27         status FOO
28         ;;
29     restart|reload|condrestart)
30         stop
31         start
32         ;;
33 *)
34     echo "Usage: $0 {start|stop|restart|reload|status}"
35     exit 1
36 esac
37 exit 0

```

Listing 4.1: Initskript /etc/init.d/example

## 4.2 OpenRC

OpenRC ist ein von der Gentoo Foundation entwickeltes modernes Init- und Service Management-System basierend auf `rc` von FreeBSD. Zusätzlich zu den Features von `rc` bietet OpenRC seinen eigenen `init`-Ersatz, `openrc-init`, welcher als Ersatz für

sysvinit verwendet werden kann. Des weiteren bietet OpenRC

- Optionale Startup-Parallelisierung
- Vollautomatische Dependency-Berechnung und dadurch sortiertes Starten
- Prozessabgrenzung mithilfe von `cgroups` unter Linux
- Ressourcenlimitierung der einzelnen Prozesse mittels `ulimit`
- Trennung von Skripts und deren Konfiguration durch `/etc/conf.d`
- Erweiterte Befehle zur manuellen Service-Steuerung
- Stateful init scripts, die überprüfen, ob ein Service bereits läuft
- Integration in Container-Services wie Docker, OpenVZ oder LXC
- Integrierte Netzwerk-Konfiguration durch die Komponente `netifrc`

OpenRC ist wesentlich performanter als die Kombination von sysvinit und RC, nicht nur durch die Parallelisierung, sondern auch durch einen optimierten Skriptinterpreter `openrc-run`, der in den Runscripts anstelle einer normalen, voll funktionsfähigen und weitaus komplexeren Shell wie `bash` verwendet wird.

### 4.2.1 Beispiele für Runscripts

Das folgende Listing zeigt das Runscript für einen sehr einfachen Daemon, der lediglich eine Netzwerkverbindung benötigt.

```
1 #!/sbin/openrc-run
2 # Copyright 1999-2014 Gentoo Foundation
3 # Distributed under the terms of the GNU General Public License v2
4
5 pidfile="/run/stunserver.pid"
6 command="/usr/sbin/stunserver"
7 command_args="${STUNTMAN_PARAMS}"
8 command_background="true"
9
10 depend() {
11     need net
12 }
```

Listing 4.2: Initskript `/etc/init.d/stuntman`

Aus der ersten Zeile ist zu erkennen, dass so wie im obigen Textabschnitt angeben `openrc-run` als Interpreter verwendet wird. Die Befehle der folgenden Zeilen haben folgende Bedeutung:

- `pidfile` beschreibt den Ort, an dem die PID-File dieses Service gespeichert wird.
- `command` beschreibt den von `start-stop-daemon` auszuführenden Befehl
- `command_args` ist eine Liste von Argumenten, die dem Befehl übergeben werden. In diesem Fall sind diese in der Variable `$STUNTMAN_PARAMS` definiert, welche im `conf.d`-Eintrag für STUNTMAN gesetzt wird.
- `command.background` besagt, ob `start-stop-daemon` den auszuführenden Befehl bereits als Hintergrundprozess starten soll. Viele Daemons führen selbst einen Fork aus, sollte dies jedoch nicht der Fall sein, bereitet OpenRC mit dieser Methode Abhilfe.
- `depend()` ist eine Funktion, die von OpenRC verwendet wird, um zu entscheiden, welche Services von diesem (zwingend oder optional) benötigt werden, welche virtuellen Services dieser bereitstellt und nach welchen anderen Services ein Service zusätzlich gestartet werden soll

### 4.2.2 Dependency Management

OpenRC bietet durch die Funktion `depend()` und subsequente Befehle wie `use`, `need`, `want`, `before`, `after` eine sehr flexible Möglichkeit zur Steuerung von Service-Abhängigkeiten. Die einzelnen Befehle werden hier beschrieben, genauere Beschreibungen sowie Beispiele findet man im [Su:Web7, Gentoo Handbook].

- `use` bedeutet, dass dieser Service durch die angeführten Services zusätzliche Funktionalität anbieten kann, diese aber nicht unbedingt benötigt. Die Zusatzfunktionalität muss allerdings vom Benutzer erwünscht werden, der zugehörige Service sich allerdings im selben Runlevel befinden.
- `want` ist vergleichbar mit `use`. Der Unterschied besteht darin, dass die angeführten Services sich nicht im selben Runlevel befinden müssen, um gestartet zu werden. Sind die Services lediglich auf dem System installiert, so werden sie durch `want` gestartet.
- `need` gibt eine Hard Dependency an. Die gegebenen Services müssen auf dem System vorhanden sein und der Zielservice kann nicht ohne diese Abhängigkeiten gestartet werden.

- `before` beschreibt eine Art Reverse Dependency. Dieser Anweisung folgend startet OpenRC den Zielservice vor den angegebenen Services. Ist einer der angegebenen Services nicht im selben Runlevel, so wird dieser ignoriert.
- `after` bildet das Gegenstück zu `before`, besagt also, dass der Zielservice nach den angeführten Services gestartet wird.
- `provide` bietet die Möglichkeit, sogenannte `virtuals` zu definieren. Meistens wird dies verwendet, um die Bereitstellung einer gewissen Funktionalität zu beschreiben. Beispielsweise können Services eine Netzwerkverbindung anfordern, ohne dass sie wissen müssen, welches Netzwerkmanagementtool im Hintergrund verwendet wird.

## 4.3 systemd

systemd ist ein modernes Service Management System von Red Hat, welches ursprünglich als Privatprojekt von Lennart Poettering erstellt wurde, um die Funktionalität von launchd aus macOS in die Linux-Welt zu bringen und so einen modernen Ersatz für sysvinit zu erschaffen. Neben Service Management bietet die systemd-Software-Suite auch einen eigenen Syslog-Daemon, eigenes Netzwerkmanagement, Session Management, einen UEFI-Bootloader, Container Management und vieles mehr. Da hier nur Service Management beschrieben werden soll, wird die Beschreibung von systemd auf dieses Feature beschränkt.

### 4.3.1 Unit Files

Anstelle klassischer Init-Skripts verzichtet systemd vollständig auf die Erweiterbarkeit von Skripts und setzt stattdessen auf die sogenannten Unit Files. Hierbei handelt es sich um relativ einfach gehaltene Konfigurationsdateien im INI-Format.

```
1 [Unit]
2 Description=OpenSSH server daemon
3 Documentation=man:sshd(8) man:sshd_config(5)
4 After=syslog.target network.target auditd.service
5
6 [Service]
7 ExecStartPre=/usr/bin/ssh-keygen -A
8 ExecStart=/usr/sbin/sshd -D -e
```

```
9 ExecReload=/bin/kill -HUP $MAINPID
10
11 [Install]
12 WantedBy=multi-user.target
```

Listing 4.3: Unit /lib/systemd/system/sshd.service

Unit Files sind generell einfacher zu lesen als klassische Initskripts. Zusätzlich bieten sie weit bessere Performance, da zur Ausführung kein Shellprozess gestartet werden muss.

### 4.3.2 Dependency Management

Die zusätzliche Abstraktion von Unit Files gegenüber gewöhnlichen Services bietet auch Vorteile für Dependencies, da nun ein Service nicht nur von anderen Services abhängig sein kann, sondern auch von Mountpoints oder Geräten. Solche Abhängigkeiten wurden zuvor mit eigenen Initskripts wie `localmount` oder mit virtuellen Services wie `net` unter OpenRC als Workaround realisiert.

systemd bietet eine ähnliche Abhängigkeitsstruktur wie OpenRC, mit vielen Zusatzfunktionalitäten. Diese sind unter [Su:Web6, Systemd Unit Configuration] detailliert beschrieben.

### 4.3.3 Socket Activation

Ein sehr bedeutendes Feature von systemd ist Socket Activation, eine Technik, bei der systemd auf diversen Sockets, sowie TCP/UDP als auch UNIX-Sockets auf Verbindungen wartet und bei einer eingehenden Clientverbindung den zugehörigen Service startet und dann die Verbindung durchreicht. Zusätzlich hat dies den Vorteil, dass eine Verbindung nicht unbedingt verloren ist obwohl der dahinterliegende Service abgestürzt ist, da der Socket Activation Daemon die Verbindung aufrecht erhalten kann und den Service einfach erneut starten muss.

Der Hauptgrund für Socket Activation bei systemd ist allerdings die Möglichkeit, Dependency-basiertes Starten zu ermöglichen, ohne Dependencies angeben zu müssen. Dies wird ermöglicht, indem ein abhängiger Service von sich aus eine Verbindung zum Socket Activation Daemon aufbaut und danach einfach der gewünschte Service von systemd gestartet wird. So kann die Startzeit eines Systems erheblich verkürzt werden. Parallelisiertes Starten wird dadurch praktisch automatisiert.

# Kapitel 5

## Umsetzung im Projekt

### 5.1 Eingesetzte Hardware

Alle vier Projektmitglieder verfügen über einen eigenen Laptop, den sie zur Entwicklung einsetzen. Zum Testen der Mobilapplikation wurden auch wahlweise die Privatgeräte der Projektmitglieder eingesetzt. Als Server wurde ein von Hetzner gemieteter Dedicated Server genutzt, der von vier Schülern der Klasse bis dahin privat genutzt wurde.

### 5.2 Eingesetzte Software

#### 5.2.1 Entwicklung

Zur Entwicklung der serverseitigen Teile wurde NetBeans als Entwicklungsumgebung gewählt, da alle Projektmitglieder damit vertraut sind und NetBeans die Erstellung von Enterprise-Applikationen mit vielen eingebauten Tools erleichtert.

Zur Versionsverwaltung wurde auf dem Entwicklungsserver eine Instanz von GitLab installiert, womit ein Continuous Deployment System eingerichtet werden konnte, welches während der Entwicklung der REST-Schnittstelle den Entwicklern dabei half, diese Komponente schneller zu testen. Mit Continuous Deployment konnte das GitLab-System auf aktualisierte Sources im Git-Repository reagieren und diese automatisch am Server mittels Maven zu einer .war-Datei kompilieren und in weiterer Folge diese Datei am Applikationsserver deployen. Dies verringerte die Wartezeiten während dem

Testen erheblich.

### 5.2.2 Reverse Proxy

Als Reverse Proxy wurde NGINX eingesetzt, da dieser bereits vor dem Projekt zum Einsatz kam und die verantwortliche Person des Projektteams sich bereits gut damit zurechtfand. Der Reverse Proxy ermöglichte es dem Projektteam, während der Entwicklung zwei verschiedene Applikationsserver gleichzeitig zu betreiben und die jeweiligen Administrationskonsolen verschlüsselt mit vertrauenswürdigen Zertifikaten öffentlich zugänglich zu machen, sodass Änderungen von jedem beliebigen Browser an jedem beliebigen Rechner getätigt werden konnten.

### 5.2.3 Applikationsserver

Während der Entwicklung wurde mehrmals zwischen den beiden Applikationsservern GlassFish und WildFly gewechselt. Die Entwicklung der Website geschah vorwiegend auf dem lokalen Rechner von Herrn Bradaric unter WildFly. Die REST-Schnittstelle wurde ursprünglich unter GlassFish entwickelt und getestet. Im Laufe des Projekts wurde jedoch auf WildFly umgestellt, was einige Komplikationen mit sich brachte, welche weiter unten in diesem Kapitel beschrieben werden. Das fertige Produkt soll vollständig unter WildFly laufen.

#### JAX-RS-Implementation

Die eingesetzte REST-Library änderte sich zusammen mit dem Applikationsserver, da immer die vom jeweiligen Server mitgelieferte Implementation verwendet wurde. Es ergaben sich nicht genügend Unterschiede zwischen den Implementierungen, dass es sich ausgezahlt hätte, hier einen bevorzugten Kandidaten zu haben.

### 5.2.4 Datenbank

Als Datenbank zur Speicherung der im REST-API zugänglichen Daten wurde PostgreSQL 10 verwendet, da die Mitglieder des Projektteams aufgrund des Einsatzes in der Schule in vergleichbaren Umgebungen bereits damit vertraut waren. Als JDBC-Treiber wurde der offizielle PostgreSQL-Treiber in der Version 42.2 eingesetzt.

## API-Scraping

Der Auftraggeber stellte für das Projekt ein REST-API zur Verfügung, aus dem Beispieldaten zum Testen der Applikation geholt werden konnten. Da der Server lauf Pflichtenheft diese Daten cachen soll und da es während des Projektverlaufs hin und wieder Unsicherheiten bezüglich Verfügbarkeit des APIs gab, wurde ein Skript entwickelt, das periodisch die für das Projekt relevanten Daten aus dem API herunterlädt und in einer Verzeichnisstruktur speichert. Um die gespeicherten Daten in die Datenbank zu laden, wurde ein weiteres Skript entwickelt, das die JSON-Daten mithilfe des Tools `jq`<sup>1</sup> parsen konnte und sie in CSV-Daten umwandelte, die mittels des Tools `psql` von PostgreSQL schnell und effizient in die Datenbank eingesetzt wurden.

### 5.2.5 Integration in das Service Management

Beide Applikationsserver wurden auf dem Server in OpenRC integriert, um das automatische Starten bei Neustart des gesamten Rechners zu gewährleisten. Folgende Initskripts wurden dazu entworfen:

---

<sup>1</sup><https://stedolan.github.io/jq/>

```

1 #!/sbin/openrc-run
2 required_dirs="/opt/glassfish/glassfish5"
3 depend() {
4     use dns
5     need net
6 }
7 pidfile="/opt/glassfish/glassfish5/glassfish/domains/domain1/config/pid"
8 command="/opt/glassfish/glassfish5/bin/asadmin"
9 command_args="start-domain"
10 command_background="false"
11 stop() {
12     $command stop-domain
13 }
```

Listing 5.1: Initskript für GlassFish

Das Initskript für GlassFish ist von der Funktionalität her sehr einfach gehalten. Jedoch weist die Serverkontrollsoftware von GlassFish ein paar Eigenheiten auf, auf die hier Rücksicht genommen werden musste. Der Server wird wie gewöhnlich mit dem Subkommando `start-domain` vom Tool `asadmin` gestartet. Nach dem Start lässt sich GlassFish jedoch nicht wie ein normaler Daemon von einem SIGTERM herunterfahren, dies muss stattdessen mit dem Subkommando `stop-domain` geschehen. Eine überschriebene `stop()` Funktion ermöglichte dies.

```

1 #!/sbin/openrc-run
2 required_dirs="/var/lib/server/wildfly/wildfly-14.0.1.Final"
3 supervisor=supervise-daemon
4 depend() {
5     use dns
6     need net
7 }
8 supervise_daemon_args="-d /var/lib/server/wildfly/wildfly-14.0.1.Final/\
9   -u server --pidfile /run/wildfly.pid"
10 command="/var/lib/server/wildfly/wildfly-14.0.1.Final/bin/standalone.sh"
11 command_args=" >/var/log/wildfly.log 2>&1"
```

Listing 5.2: Initskript für WildFly

WildFly verhält sich viel mehr wie ein gewöhnlicher UNIX-Daemon. Dies macht das Initskript etwas einfacher, zusätzlich kann der Server unter Service Supervision mit `supervise-daemon` gesetzt werden. Somit kann OpenRC die bisherige Ausführungszeit und den letzten Exit Code des Prozesses mitverfolgen und den Service im Falle eines vollständigen Absturzes automatisch neu starten.

## Umstieg auf systemd

Im Zuge der Entwicklung wurde beschlossen, projektspezifische Serverkomponenten in eine virtuelle Maschine zu verschieben. In der virtuellen Maschine wurde ein systemd-basiertes System eingesetzt, somit wurde zur Erleichterung der Administration des Applikationsservers eine Unit File erstellt.

```
1 [Unit]
2 Description=WildFly Application Server
3 Wants=network-online.target
4 After=network-online.target
5
6 [Service]
7 Type=simple
8 User=wfly
9 Group=wfly
10 ExecStart=/opt/wildfly/bin/standalone.sh
11 Restart=always
12 RestartSec=20
13
14 [Install]
15 WantedBy=multi-user.target
```

Listing 5.3: Unit File für WildFly

Dies dient auch als Beispiel, um die Unterschiede zwischen OpenRC-Initskripts und Unit Files aufzuzeigen. Auf den ersten Blick ist zu erkennen, dass die Direktiven der Unit File großteils selbsterklärend und weit übersichtlicher sind. Auch zeigt systemd ein paar seiner Stärken auf, zum Beispiel muss keine PID-File definiert werden, da dies automatisch übernommen wird. Für das Logging muss die Ausgabe des Programms auch nicht in eine eigens angegebene Datei umgeleitet werden, da standardmäßig eine Umleitung in das Journal erfolgt.

## 5.3 Probleme bei der Entwicklung

Durch zahlreiche Umstellungen während der Entwicklung und aufgrund des Einsatzes von für das Projektteam relativ unbekannten Technologien ergaben sich einige Probleme während des Projektverlaufes.

### 5.3.1 Problem: AES-256 nicht unterstützt in Java

Die Idee des Umstiegs auf WildFly entstand neben anderen Gründen aus der Problematik, dass GlassFish als HTTP-Client beim Zugriff auf das REST-API keine Unterstützung für TLS hat und somit die Website nicht auf das REST-API zugreifen konnte. Beim Umstieg auf WildFly wurde zusätzlich ausfindig gemacht, dass Java ohne Erweiterungen die bis dahin von NGINX erzwungene AES-256-Verschlüsselung nicht unterstützt. Es hätte die Java Cryptography Extension (JCE) installiert werden müssen, um dies zu bewerkstelligen. Stattdessen wurden bei NGINX weitere Cipher Suites eingetragen, die AES mit einer Schlüssellänge von 128 bit verwenden, da dies wesentlich weniger Komplikationen bereitet und sicherheitstechnisch immer noch völlig ausreicht.

### 5.3.2 Problem: Umstellung von GlassFish auf WildFly

Im Laufe des Projektes wurde entschieden, dass WildFly als der finale Applikationsserver zum Einsatz kommt, auf dem letztendlich alle Serverkomponenten zusammengeführt werden sollten. Mitten unter der Entwicklung der REST-Schnittstelle wurde versucht, diese ohne Änderungen auf WildFly zu deployen. Kaum überraschend ist, dass dieses Vorhaben nicht geeglückt ist. Da kurzfristig keine adäquate Lösung des Problems gefunden werden konnte, wurde entschieden, vorerst weiterhin auf GlassFish weiterzuarbeiten.

### JDBC und JPA unter WildFly

Später stellte sich heraus, dass das Problem eine falsch konfigurierte JDBC-Verbindung sowie eine falsch konfigurierte Persistence Unit waren. Es wurde auf WildFly über HAL eine neue JDBC-Verbindung eingerichtet, diese wurde mit dem neuen JNDI-Namen in die `persistence.xml` eingetragen. Als Teil der Konfiguration der Persistence Unit gibt es die Option `exclude-unlisted-classes`, welche, wenn

abgeschalten, dafür sorgt, dass die mit `@Entity` annotierten Java-Klassen automatisch in den Persistenzkontext aufgenommen werden. Dieses automatische Erkennen führt bei WildFly aus unbekannten Gründen zu einem Stack Overflow. Um den Fehler zu umgehen wurde diese Option bei der Umstellung aktiviert und die Modelklassen manuell in die Konfigurationsdatei eingetragen.

### **Umstellung von EclipseLink auf Hibernate**

Im weiteren Verlauf der Entwicklung wurde entdeckt, dass nun der Schreibzugriff auf die Datenbank nicht mehr möglich war. Dieses Problem war auf eine andere Implementation von JPA zurückzuführen. GlassFish verwendet standardmäßig EclipseLink um die Entities des Persistenzsystems zu verwalten. Dies wurde auch während der ursprünglichen Entwicklung so eingesetzt. WildFly verwendet jedoch einen anderen Persistence Provider, Hibernate, der mit der internen Verwaltung der Sequences von EclipseLink nicht kompatibel ist. Sequences werden datenbankseitig gespeichert und dienen dazu, fortlaufende Nummerierungen für automatisch erzeugte Primärschlüssel zu haben. Das Problem wurde gelöst, indem die Datenbankstruktur mittels Schema-export von der Java-Seite aus neu erstellt wurde und ein Backup des Datenbestandes eingespielt wurde.

# Kapitel 6

## Grundlagen von REST

### 6.1 Entstehung von REST

Das Ziel dieser Diplomarbeit ist es diverse REST-APIs auszuwerten und zu entscheiden welches für das Projekt CarSharing am besten geeignet ist. Der Architektur Stil *Representational State Transfer* (REST) wurde im Jahre 2000 von Roy Fielding in seiner Dissertation *Architectural Styles and the Design of Network-based Software Architectures* hergeleitet. Er bezieht sich in seiner Arbeit auf diverse Netzwerk-basierte Architektur Stile, aus welchen er dann seinen eigenen Stil ableitet. Dr. Fielding legt in seiner Arbeit diverse Prinzipien fest, denen man in der REST Architektur folgen muss.

#### 6.1.1 Client-Server-Architektur

Das erste Prinzip das Roy Fielding festlegt, ist das sogenannte *Client-Server* Prinzip. Dies besagt, dass man die Server Implementation und die des Clients zu 100 Prozent trennt. Die Trennung dieser zwei zusammenarbeitenden Systeme bringt mehrere Vorteile. Der erste Vorteil, den man erlangt, ist die höhere Portabilität des Clients. Der zweite Vorteil ist die Möglichkeit die beiden Komponenten absolut voneinander getrennt weiterentwickeln zu können.

### 6.1.2 Zustandslosigkeit

Das zweite Prinzip welches Fielding definiert ist das Zustandslosigkeitsprinzip, welches besagt, dass jede Anfrage die an den Server gesendet wird alle Informationen zum Verarbeiten dieser Anfrage beinhalten muss. Der Server darf nicht auf gespeicherte Daten zugreifen um eine Antwort zu senden. Hiermit werden folgende Eigenschaften verbessert: Sichtbarkeit, Zuverlässigkeit und Skalierbarkeit.

#### 1. Sichtbarkeit:

Die Sichtbarkeit wird durch die Informationskapselung gesteigert. Dies gelingt dadurch, dass jede Anfrage nun separat betrachtet werden kann. Durch diese Separation der Anfragen ist es möglich zu garantieren, dass sich die Anfragen nicht gegenseitig beeinflussen.

#### 2. Zuverlässigkeit:

Die Zuverlässigkeit wird durch die Sicherheit vor Teil-Ausfällen gesteigert. Dies wird gewährleistet, da andere Teile des Systems die Aufgaben des ausgefallenen Teils übernehmen können.

#### 3. Skalierbarkeit:

Die Skalierbarkeit wird verbessert da der Server die Zustandsdaten der einzelnen Anfragen nicht speichern oder verwalten muss. Dadurch kann er schneller wieder Ressourcen freigeben und die Implementation wird wesentlich erleichtert, da man kein Ressourcenmanagement zwischen den Anfragen bewältigen muss.

Jedoch hat die Zustandslosigkeit nicht nur Vorteile, sondern auch einen großen Nachteil nämlich die Erhöhung des Netzwerkverkehrs. Dieser erhöht sich dadurch dass nichts gespeichert wird und viele Daten bei jeder Anfrage erneut gesendet werden müssen.

### 6.1.3 Cache

Um dem eben zuvor genannten Problem der Datenvermehrung entgegen zu wirken, können Daten als *Cacheable* oder *Non-cacheable* definiert werden. Bei *Cacheable* Daten ist dem Client oder dem Server erlaubt die Daten zu speichern. Dies ermöglicht es dem Benutzer eine kürzere Latenzzeit zu gewährleisten, währenddessen auch der Netzwerkverkehr verringert wird. Das Speichern der Daten bringt jedoch ein Problem mit sich, es kann nämlich passieren dass die Daten bereits veraltet sind. Daher kann die Konsistenz des Systems nicht völlig gewährleistet werden. Es liegt daher am Entwickler die richtigen Daten als *Cacheable* zu klassifizieren.

### 6.1.4 Einheitliche Schnittstellen

Das Ziel des REST-Architektur Stiles ist es die einzelnen Komponenten voneinander zu trennen und die Datenlieferung so einheitlich wie möglich zu gestalten. Dies vereinfacht die Implementierung wesentlich und weiters können Teile des Systems separat für andere Systeme verwendet werden. Hier ist wiederum auch ein Problem versteckt, nämlich wird die Effizienz des Systems verringert. Da eben alle Daten immer einheitlich gesendet werden, kommt es zum Mitsenden von ungenutzten Daten.

### 6.1.5 Code-On-Demand

Diese Funktionalität von REST ist für die Vereinfachung der Entwicklung von Clients zuständig. Es wird den Entwicklern ermöglicht bereits fertige Codesegmente zu verwenden. Dadurch werden Fehlerquellen deutlich reduziert und die Implementierung erfolgt wesentlich schneller. Hierbei handelt es sich um die einzige optionale Funktionalität von REST. Fielding begründet dies so:

The notion of an optional constraint may seem like an oxymoron. However, it does have a purpose in the architectural design of a system that encompasses multiple organizational boundaries. It means that the architecture only gains the benefit (and suffers the disadvantages) of the optional constraints when they are known to be in effect for some realm of the overall system.

[Ho:Web02, Architectural Styles and the Design of Network-based Software Architectures]

# Kapitel 7

## Kriterien und deren Gewichtung

In diesem Kapitel wird festgelegt welche Kriterien bei jedem Client API untersucht und bewertet werden. Es wird zu jedem API zuerst eine Auswertung der Kriterien und danach eine Bewertung durchgeführt. Weiters wird bestimmt wie die einzelnen Bewertungen (1-5) strukturiert sind. Die Kriterien werden in 2 Kategorien aufgeteilt: Allgemeines und Entwicklung.

### 7.1 Allgemeines

In dieser Kategorie wird behandelt ob es Hilfestellungen (Tutorials, Codeexemplare, usw.) gibt und es wird der momentane Zustand des APIs beschrieben.

1. Gibt es Hilfen zum Erlernen der APIs, wenn ja, welche sind vorhanden?  
**Gewichtung:** 2  
Bei dieser Kategorie ist die Gewichtung eher niedrig angesetzt da bereits Vorwissen zu gewissen APIs aus dem Unterricht besteht.
2. Wie gut ist die API dokumentiert?  
**Gewichtung:** 5  
Diese Kategorie wurde sehr hoch gewichtet, da die Dokumentation eines APIs extrem wichtig ist um die API zu verstehen.
3. Braucht man eine Lizenz für die API, wenn ja, unter welchen Kosten ist die API verfügbar?  
**Gewichtung:** 4

Hier wurde die Gewichtung auch relativ hoch gewählt, da das Budget für das Projekt CarSharing sehr minimal bis nicht existent ist.

4. Gibt es eine Community für dieses API, wenn ja, ist diese Community aktiv und/oder hilfsbereit?

**Gewichtung:** 3

Bei dieser Kategorie ist die Gewichtung mittig angesetzt, da eine Community bei Problemen jeglicher Art stark weiterhelfen kann.

## 7.2 Entwicklung

Die Kategorie Entwicklung beschäftigt sich damit, wie die Entwicklungsumgebungen der diversen APIs aufgebaut sind. Ebenfalls werden hier die Voraussetzungen für die Software und Hardware der verschiedenen APIs behandelt.

1. Kann die API im Produktivbetrieb verwendet werden?

**Gewichtung:** 4

Bei dieser Kategorie ist die Gewichtung hoch angesetzt, da es wichtig ist dass dieses Projekt auch später im Produktivbetrieb gut funktioniert.

2. Welche Java Version ist die minimale die von dem API unterstützt wird?

**Gewichtung:** 1

Hier ist die Gewichtung sehr niedrig festgelegt, da das Projekt in Java 1.8, einer sehr neuen Version programmiert wird und der Client im Produktivbetrieb nur auf Android Geräten installiert wird.

3. Gibt es als Hilfe bei der Modellierung grafische Hilfsmittel?

**Gewichtung:** 3

Diese Kategorie ist mittig gewichtet, da es einerseits nicht viel Modellierung bedarf in dem Projekt CarSharing und andererseits ist es immer gut sich Software zuerst grafisch zu Visualisieren.

4. Wie komplex ist die Entwicklung des Clients?

**Gewichtung:** 5

Hier ist die Gewichtung sehr hoch ausgefallen, da die Komplexität eine große Rolle in der Implementierung darstellt. Es wird bei diesem Punkt besonders auf die Kompatibilität des APIs mit Android geachtet.

5. Welche Android SDK ist die minimale die von dem API unterstützt wird?

**Gewichtung:** 3

Hier ist die Gewichtung sehr niedrig festgelegt, da das Projekt in der SDK 26,

einer sehr neuen Android Version programmiert wird. Es wäre daher nur wichtig für die Rückwertskompatibilität und das ist auch weniger relevant, da die Software nur auf neueren Geräten laufen wird.

6. Wie erprobt ist die Verwendung der APIs in Verbindung mit Android?

**Gewichtung:** 5

Diese Kategorie wurde sehr hoch gewichtet, da die Client Software nur auf Android laufen wird.

## 7.3 Bewertung

Das Ziel dieser Diplomarbeit ist es das beste Client API für das Projekt CarSharing aus zu werten. Daher wird jede Bewertungskategorie mit einer Gewichtung unter besonderer Beachtung des Projekts versehen. Abhängige Kindkategorien werden jedoch nicht gewichtet und bekommen auch keine Note, da sie beim Elternpunkt mit in die Bewertungen einbezogen werden. Es gibt auch zu jeder Gewichtung eine Beschreibung warum diese so gewählt wurde. Diese Gewichtung wird sich im Rahmen von 1-5, wobei 5 das beste ist, bewegen. Bei der Bewertung der einzelnen APIs wird jede Kategorie eine Note bekommen. Diese Bewertungen werden sich ebenfalls von 1-5, wobei 5 hier auch das beste ist, bewegen. Weiters wird es zu jeder Kategorie eine Analyse geben aus welcher die Bewertungen dann hervorgeht. Nach der Bewertung der einzelnen Kategorien werden alle Bewertungen eines APIs mit der Gewichtung der jeweiligen Kategorie multipliziert und daraus ergibt sich dann eine Gesamtnote für die API.

### 7.3.1 Bewertungsskala

Alle einzeln vergebenen Bewertungen fügen sich nach der Bewertung in eine Gesamtnote zusammen. Hier wird nun aufgestellt ab wie vielen Punkten welche Gesamtnote vergeben wird. Diese Note wird darüber entscheiden welches API in dem Projekt CarSharing verwendet wird. Insgesamt sind 200 Punkte verfügbar, daraus ergibt sich folgende Tabelle:

Bewertung	Punkte
5	153 bis 175
4	132 bis 152
3	111 bis 131
2	89 bis 110
1	0 bis 88

Tabelle 7.1: Bewertung

# Kapitel 8

## Bewertung der Client APIs

### 8.1 Jersey

#### 8.1.1 Einleitung

Jersey ist ein RESTful Web Service welches als Referenzimplementierung für JAX-RS gesehen wird. Es werden auch diverse SPIs (Service Provider Interface) unterstützt, welche es den Entwicklern ermöglichen sich ihre Jersey Erfahrungen genau so zu gestalten wie sie dies möchten.

#### Codebeispiel

Hier sieht man wie ein Jersey REST-Endpunkt aussehen kann:

```
1 // Diese Klasse ist nun unter dem URI "/helloworld" erreichbar
2 @Path("/beispiel")
3 public class JerseyBeispiel {
4     // Diese Methode verarbeitet Get requests
5     @GET
6     // Diese Methode produziert auch noch etwas, in diesem Falle "text/
7     // plain"
8     @Produces("text/plain")
9     public String getNachricht() {
10         // gibt in diesem Falle einen String zurueck
11         return "Hallo Jersey";
12     }
13 }
```

Listing 8.1: Jersey Server Beispiel

Es wird in JAX-RS alles über Annotationen geregelt:

`@Path` legt das Mapping von Klassen und Methoden fest.

`@Get/@Put` und so weiter, erlauben an einem Pfad HTTP Methoden.

`@Produces/@Consumes` legen fest welche MIME-Typen als Anfrage erwartet werden beziehungsweise welche als Antwort gesendet werden.

### 8.1.2 Bewertung von Jersey

1. Gibt es Hilfen zum Erlernen des APIs, wenn ja, welche sind vorhanden?

In diesem Punkt wird Jersey mit der vollen Punktzahl bewertet, da es sehr viele ausführliche Tutorials gibt und darüber hinaus auch noch zahlreiche Codebeispiele zur Verfügung stehen.

**Bewertung:** 5

2. Wie gut ist die API dokumentiert?

Die API ist umfassend dokumentiert, es gibt einen 290 Seiten langen User Guide [Ho:Web04, Jersey User Guide] in dem jede Funktionalität bis in kleinste Detail beschrieben ist. Dieser Guide inkludiert weiters ein sehr gutes *Getting Started* Kapitel, womit man sehr schnell eine *Hello World* Applikation zusammengestellt hat. Ebenfalls existiert eine Javadoc für diese API.

**Bewertung:** 5

3. Braucht man eine Lizenz für die API, wenn ja, unter welchen Kosten ist die API verfügbar?

In diesem Punkt erreicht Jersey ebenfalls die volle Punktzahl, denn es ist völlig gratis zu verwenden und benötigt nicht einmal einen API-key.

**Bewertung:** 5

4. Gibt es eine Community für dieses API, wenn ja, ist diese Community aktiv und/oder hilfsbereit?

Es existiert eine ziemlich große Community, welche auf Plattformen wie Stackoverflow präsent sind und bei jedem Problem helfen. Jedoch werden hier zwei Punkte abgezogen, da von den Entwicklern zu diesem Zeitpunkt (12.1.2019) seit dem 13.April 2018 keine offizielle Meldung mehr kundgegeben wurde.

**Bewertung:** 3

### 8.1.3 Entwicklung

1. Kann die API im Produktivbetrieb verwendet werden?

Jersey ist ein bereits über lange Zeit erprobtes API. Es wurde bereits von vielen Benutzern im Produktiveinsatz getestet. Weiters wird es schon von einigen Firmen verwendet, also ist kein Grund erkennbar warum es in dem Projekt Cars-haring nicht genau so funktionieren sollte.

**Bewertung:** 5

2. Welche Java Version ist die minimale die von dem API unterstützt wird?

Jersey funktioniert in jeder Version ab Java SE 7, diese wurde am 7.7.2011 veröffentlicht. Bei diesem Punkt verdient sich die API wieder alle Punkte, da es heutzutage kaum noch Systeme gibt, die eine ältere Version als Java SE 7 verwenden. Dies wird belegt von der [Ho:Web03, Plumbr Java Statistik 2017], welche mit folgender Grafik die Anzahl an verwendeten Java Versionen aufzeigt.

**Bewertung:** 5

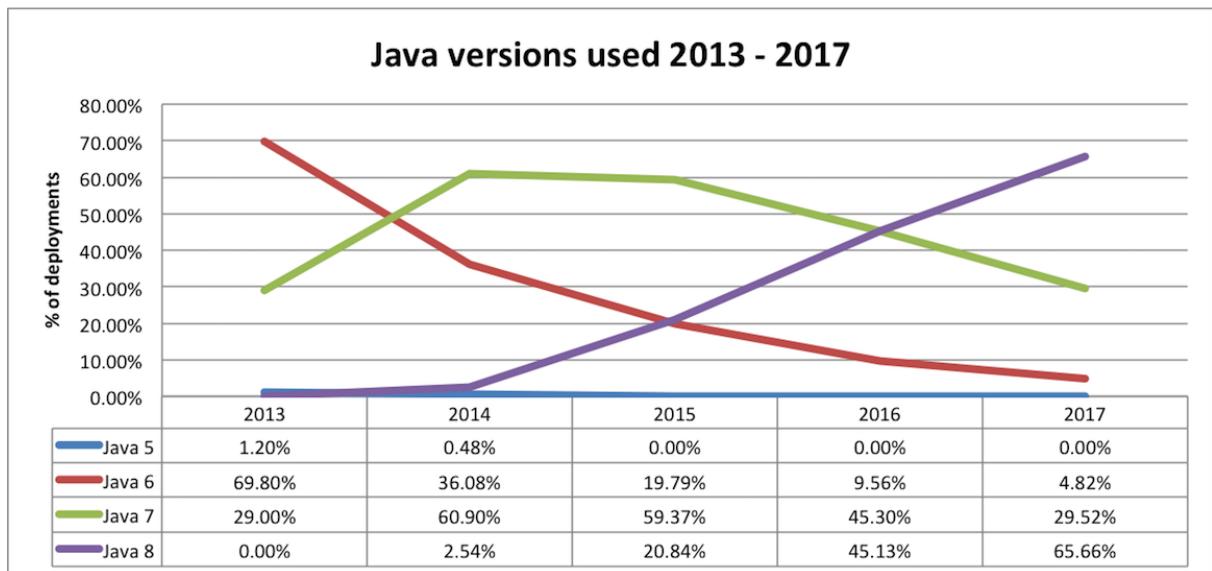


Abbildung 8.1: Java Statistik 2017

### 3. Gibt es als Hilfe bei der Modellierung grafische Hilfsmittel?

Nein, es gibt keine Tools die einem bei der Erstellung des Servers oder des Clients grafisch unterstützen. Bei diesem Unterpunkt bekommt die API keine Punkte, da es kaum grafische Beispiele und keine sonstigen visuellen Hilfen gibt.

**Bewertung:** 0

### 4. Wie komplex ist die Entwicklung des Clients?

Die Entwicklung eines Clients mit Jersey wäre im Projekt CarSharing sehr einfach da der REST Server bereits hiermit entwickelt wurde. Dadurch könnte einem zu jeder RESTResource automatisch ein WebClient generiert werden. Jedoch muss der Client im späteren Produktivbetrieb auf Android Geräten als App funktionieren und es gibt keine Möglichkeit sich den Client hierfür generieren zu lassen. Weiters besteht kein Vorwissen über die manuelle Entwicklung eines Jersey Clients. Die API beachtet auch die besonderen Bedürfnisse von Android nicht. Dies kann man am besten an der dürfig vorhandenen Unterstützung der Asynchronität erkennen. In dem nachfolgenden Codeexemplar wird gezeigt dass sich der Entwickler in Jersey zuerst einen request zusammenbauen muss und dies danach manuell in eine Queue hinzufügen muss.

Der folgende Codeabschnitt zeigt einen Jersey Client in Android:

```
1 public class MainActivity extends AppCompatActivity {
2     EditText editText;
3     TextView text;
4     Button btn;
5     String url ="http://192.168.1.3/JavaRESTfullWS/rest/DemoService/";
6
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_main);
11
12        editText = (EditText) findViewById(R.id.editText);
13        text = (TextView) findViewById(R.id.text);
14        btn = (Button) findViewById(R.id.btn);
15
16        btn.setOnClickListener(new View.OnClickListener() {
17            @Override
18            public void onClick(View v) {
19                StringRequest request = new StringRequest(Request.
20                    Method.GET, url+editText.getText().toString(), new
21                    Response.Listener<String>() {
22                        @Override
23                        public void onResponse(String s) {
24                            text.setText(s);
25                        }
26                    });
27
28        }
29    }
30}
```

```
24     }, new Response.ErrorListener() {
25         @Override
26         public void onErrorResponse(VolleyError
27             volleyError) {
28             text.setText("Some error occurred!!!");
29         }
30     });
31     RequestQueue rQueue = Volley.newRequestQueue(
32         MainActivity.this);
33     rQueue.add(request);
34 }
35 }
36 }
```

Listing 8.2: Jersey Client

Durch die mangelnden Features in Android und dadurch dass sich der Client nicht generieren lässt, verliert die API hier 2 Punkte.

**Bewertung:** 3

### 5. Welche Android SDK ist die minimale die von dem API unterstützt wird?

Die niedrigste SDK Version die verwendet werden kann ist 21. Diese ist jedoch ziemlich neu und wird daher nicht von allen Geräten unterstützt, dies wird durch viele Statistiken belegt. In dem unten angeführten Tabelle kann man erkennen, dass 8.3% aller Geräte weltweit eine zu niedrige SDK Version verwenden und daher Jersey nicht unterstützen.

API Level ▾	Name	OS Version	H1 2018	H2 2017	H1 2017
API 26+	Oreo	8.0 or newer	9.4 %	0.5 %	0.0 %
API 24+	Nougat	7.0 or newer	57.1 %	37.8 %	11.1 %
API 23+	Marshmallow	6.0 or newer	77.3 %	69.0 %	57.9 %
API 21+	Lollipop	5.0 or newer	91.7 %	87.2 %	81.2 %
API 19+	KitKat	4.4 or newer	97.0 %	95.3 %	92.8 %
API 16+	Jelly Bean	4.1 or newer	99.2 %	98.7 %	97.7 %
API 14+	Ice Cream Sandwich	4.0 or newer	99.5 %	99.2 %	98.8 %

Abbildung 8.2: Android Statistik 2018

[Ho:Web19, Android Statistik]

Weiters ist es sehr schwierig diese Information herauszufinden, da dies fast nirgendwo dokumentiert ist. Für diese schlechte Dokumentation und die sehr hohe minimale SDK werden hier 3 Punkte abgezogen.

**Bewertung: 2**

### 6. Wie erprob ist die Verwendung der APIs in Verbindung mit Android?

Jersey wurde erst in einer relativ neuen Android Version in das System integriert, daher gibt es kaum noch Berichte wie sich die API im Produktivbetrieb verhält. Ein weiterer Einflussfaktor ist, dass Jersey sehr selten in der Verbindung mit Android genutzt wird. Daher werden hier nur 2 Punkte vergeben.

**Bewertung: 2**

### 8.1.4 Punkte von Jersey

Bewertung	Punkte	API
5	153bis 175	
4	132 bis 152	
3	111 bis 131	Jersey: 114
2	89 bis 110	
1	0 bis 88	

Tabelle 8.1: Bewertung von Jersey

Die API macht einige Dinge sehr gut und andere nur mittelmäßig, daher ist der Punktestand sehr gerechtfertigt. Weiters ist die Verwendung von Jersey in Kombination mit Android noch relativ neu und nicht völlig erprobt. Daher kann Jersey für nicht Android-Apps bestens empfohlen werden, jedoch ist es nicht besonders passend für das Projekt CarSharing.

## 8.2 Resteasy

### 8.2.1 Einleitung

Resteasy ist ein JBoss Projekt, welches als RESTful Webservice auf der Serverseite oder auch als Client genutzt werden kann. Die API ist eine vollständige, zertifizierte Implementierung von JAX-RS.

#### Codebeispiel

Hier sieht man wie ein Resteasy Endpunkt aussehen kann:

```

1  @Path("/cars")
2  public class CarService {
3
4      private Map<String, Car> inventory = new HashMap<String, Car>();
5
6      @GET
7      @Path("/getinfo")
8      @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
9      public Car carByVin(@QueryParam("vin") String vin) {
10          return inventory.get(vin);
11      }
12
13      @POST
14      @Path("/addCar")
15      @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
16      public Response addMovie(Car car) {
17          inventory.put(car.getVin(), car);
18          return Response.status(Response.Status.CREATED).build();
19      }
20 }
```

Listing 8.3: Resteasy Server Beispiel

Anhand dieses Beispiels kann man nun erneut, wie bereits bei dem Listing 8.1.1 beschrieben die standardmäßigen JAX-RS Annotationen erkennen. Man sieht hier zwei Methoden: In der ersten wird einem das dementsprechende Car Objekt zu der angegebenen Vin<sup>1</sup> zurückgeschickt. In der zweiten Funktion übergibt man ein Car Objekt entweder in JSON oder XML, welches nun dem Inventar hinzugefügt wird und eine Statusmeldung zurücksendet.

<sup>1</sup>Vehicle identification number

### 8.2.2 Bewertung von Resteasy

1. Gibt es Hilfen zum Erlernen des APIs, wenn ja, welche sind vorhanden?

In diesem Punkt wird Resteasy mit der vollen Punktzahl bewertet, da es sehr viele ausführliche Tutorials gibt und darüber hinaus auch noch zahlreiche Codebeispiele zur Verfügung stehen.

**Bewertung:** 5

2. Wie gut ist die API dokumentiert?

Hier bekommt die API alle Punkte, da die Dokumentation umfassend ist. Auf der Website der Entwickler [Ho:Web10, Resteasy Website] ist jegliche Information sofort aufzufinden. Weiters gibt es einen sehr umfangreichen User Guide [Ho:Web11, Resteasy User Guide], welcher 60 Kapitel mit über 300 Seiten beinhaltet. Neben diesem existiert noch eine eigene Wiki [Ho:Web12, Resteasy Wiki] in welcher noch einmal die gesamte Verwendung der API ausführlichst erklärt ist. Zu dieser bereits sehr ausführlichen Dokumentation existiert ebenfalls eine gute Java Dokumentation [Ho:Web13, Resteasy Javadoc] für die API.

**Bewertung:** 5

3. Braucht man eine Lizenz für die API, wenn ja, unter welchen Kosten ist die API verfügbar?

In diesem Punkt erreicht Resteasy die volle Punktzahl, denn es ist völlig gratis zu verwenden und benötigt nicht einmal einen API-key.

**Bewertung:** 5

4. Gibt es eine Community für dieses API, wenn ja, ist diese Community aktiv und/oder hilfsbereit?

Resteasy hat eine sehr aktive Community, auf der Website Stackoverflow [Ho:Web14, Resteasy Stackoverflow] sind derzeit (16.3.2019) 2.099 Diskussionen offen. Weiters gibt es von JBoss einen offiziellen *Issue Tracker* [Ho:Web15, Resteasy Issue Tracker] mit dessen Hilfe schwerwiegende Probleme direkt mit den Entwicklern besprochen werden können. Es wird außerdem von JBoss ein kommerzieller Entwicklungs-, Produktions- und Trainings-Support zur Verfügung gestellt. Daher gibt es hier keinen Grund nicht die volle Punktzahl zu vergeben.

**Bewertung:** 5

### 8.2.3 Entwicklung

1. Kann die API im Produktivbetrieb verwendet werden?

Resteasy ist bereits ein relativ altes API, welches über viele Jahre in diversen Betrieben erprobt wurde. Weiters gibt es offiziellen Support für Betriebe, die es im Produktivbetrieb verwenden. Daher ist kein Grund erkennbar, warum die API nicht perfekt in dem Projekt Carsharing funktionieren sollte.

**Bewertung:** 5

2. Welche Java Version ist die minimale die von dem API unterstützt wird?

Resteasy funktioniert in jeder Version ab Java SE 5, diese wurde am 30.9.2004 veröffentlicht. Bei diesem Punkt verdient sich die API wieder alle Punkte, da es heutzutage keine Systeme mehr gibt die eine ältere Version als Java SE 5 verwenden. Dies wird belegt von der Plumbr Java Statistik 8.1.

**Bewertung:** 5

3. Gibt es als Hilfe bei der Modellierung grafische Hilfsmittel?

Es gibt keine Tools die eine grafische Modellierung erlauben, jedoch gibt es viele UML Diagramme, welche es ermöglichen sich die Architektur besser vorstellen zu können, daher bekommt die API hier nur zwei Punkte Abzug.

**Bewertung:** 3

4. Wie komplex ist die Entwicklung des Clients?

Die Struktur der Clients in Resteasy ist etwas komplexer als bei dem bisher bewerteten API. Diese zusätzliche Komplexität wird jedoch drastisch durch die gute Dokumentation und die Vielzahl an Tutorials reduziert. Die Entwicklung sollte weiter erleichtert werden, da bereits eine gewisse Vertrautheit mit den JAX-RS Annotationen vorhanden ist.

Der folgende Codeabschnitt zeigt einen Resteasy Client:

```
1 @Path("/cars")
2 public interface ServicesInterface {
3
4     @GET
5     @Path("/getinfo")
6     @Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML
7                 })
8     Car carByVin(@QueryParam("vin") String vin);
9
10    @POST
11    @Path("/addCar")
```

```

11  @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML
12      })
13  Response addCar(Car car);
14
15  @PUT
16  @Path("/updateCar")
17  @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML
18      })
19  Response updateCar(Car car);
20
21  @DELETE
22  @Path("/deleteCar")
23  Response deleteCar(@QueryParam("vin") String vin);
24
25  @XmlAccessorType(XmlAccessType.FIELD)
26  @XmlType(name = "car", propOrder = { "vin", "title" })
27  public class Car {
28
29      protected String vin;
30      protected String title;
31
32      // getters and setters
33  }

```

Listing 8.4: Resteasy Client

Die Clients sehen hierbei jenen von Retrofit sehr ähnlich. Einer der Unterschiede ist jedoch die hier zusätzlich nötigen Annotationen in den Entitätsklassen.

**Bewertung:** 5

5. Welche Android SDK ist die minimale die von dem API unterstützt wird?

Die niedrigste SDK Version die verwendet werden kann ist 19. Dies ist eine relativ neue Version und verhindert somit die Rückwertskompatibilität auf einige ältere Geräte. Laut der Android Version Statistik 8.2 sind ungefähr 3% aller Geräte nicht mit der API kompatibel. Weiters muss eindeutig angemerkt werden, dass die API keinen offiziellen Android Support hat. Resteasy werden dafür 3 Punkte abgezogen.

**Bewertung:** 2

6. Wie erprobt ist die Verwendung der APIs in Verbindung mit Android?

Die API wurde nie für Android entwickelt und es ist auch keine offizielle Version dafür vorhanden. Es existieren diverse Ports, ob diesen Vertrauen entgegengebracht werden sollte, ist jedoch fragwürdig. Durch diese inoffizielle Verwendung in Android fehlt auch eine genaue Dokumentation. Weiters existieren kaum Tutorials wie man einen Resteasy Client in Android entwickelt. Dies

würde die Entwicklung enorm erschweren und das Projekt Carsharing müsste sich auf die bereits erwähnten dubiosen Ports verlassen. Daher bekommt die API hier keinen Punkt.

**Bewertung:** 0

Resteasy ist ein tolles API, dass sehr ausführlich dokumentiert ist und hochwertigen professionellen Support zur Verfügung stellt. Die API wirkte wie die perfekte Wahl für das Projekt Carsharing bis es zu dem Punkt kam, an welchem aufgedeckt wurde, dass Android nicht unterstützt wird. Dies macht Resteasy faktisch nutzlos für das Projekt, daher kann es nur für jede Applikation empfohlen werden die nicht unter Android laufen.

#### 8.2.4 Punkte von Resteasy

Bewertung	Punkte	API
5	153bis 175	
4	132 bis 152	Resteasy: 135
3	111 bis 131	Jersey: 114
2	89 bis 110	
1	0 bis 88	

Tabelle 8.2: Bewertung von Resteasy

## 8.3 Restlet

### 8.3.1 Einleitung

Restlet ist eine API mit der ein RESTful Webservice als Serverseite oder auch ein Client erzeugt werden kann. Weiters bieten die Entwickler eine Integrierte Cloud an, in welcher Daten gespeichert werden können. Das Besondere an dieser API ist, dass die Modellierung und das Testen großteils in einem grafischen Client erfolgen. Es werden einem außerdem viele mühsame Prozesse direkt von den Entwicklern erleichtert. Ein Beispiel hiervon ist, dass einem automatisch eine Referenzdokumentation zu der entwickelten Software generiert wird. Ebenfalls können zu den REST Endpunkten Clients in diversen Programmiersprachen generiert werden.

#### Codebeispiel

Hier sieht man wie ein Restlet JAX-RS Endpunkt aussehen kann:

```

1 @Path("car")
2 public class CarResource {
3
4     @GET
5     @Produces("text/plain")
6     public String getCarVin() {
7         return Car.getVin();
8     }
9
10    @GET
11    @Produces(MediaType.TEXT_XML)
12    public String getVinXML() {
13        return "<?xml version=\"1.0\"?>"+<car>"+Car.getCarVin()+"</car>";
14    }
15 }
```

Listing 8.5: Resteasy Server Beispiel

Hier kann man nun erneut wie bereits bei dem Listing 8.1.1 beschrieben die standardmäßigen JAX-RS Annotationen erkennen. Man sieht in dem oben aufgezeigten Source Code zwei Methoden. Die erste gibt einem sehr simpel die Vin<sup>2</sup> des Autos zurück. In der zweiten Funktion bekommt man die Vin im XML Format.

<sup>2</sup>Vehicle identification number

### 8.3.2 Bewertung von Restlet

1. Gibt es Hilfen zum Erlernen des APIs, wenn ja, welche sind vorhanden?

In diesem Punkt wird Restlet mit der vollen Punktzahl bewertet da, es sehr viele ausführliche Tutorials gibt und darüber hinaus auch noch zahlreiche Codebeispiele zur Verfügung stehen.

**Bewertung:** 5

2. Wie gut ist die API dokumentiert?

Die Dokumentation von Restlet wurde in 3 Teile gegliedert: Studio, Client und Cloud. Jeder dieser Unterpunkte ist außerordentlich gut dokumentiert und darüber hinaus gibt es direkt auf der Homepage [Ho:Web16, Restlet Homepage] multiple Tutorials die jede Funktionalität bis in das kleinste Detail beschreiben. Ebenfalls existiert ein alles umfassender User Guide [Ho:Web17, Restlet User Guide], welcher alle Eigenschaften des APIs beschreibt und ausführliche Beispiele zu allen Funktionen liefert.

**Bewertung:** 5

3. Braucht man eine Lizenz für die API, wenn ja, unter welchen Kosten ist die API verfügbar?

Die Entwickler von Restlet stellen eine bedingte gratis Lizenz zur Verfügung. Diese darf selbstverständlich für nicht kommerzielle Zwecke eingesetzt werden und inkludiert nur einen einzigen User mit einem Projekt. *Genauere Preise folgen sobald die Entwickler geantwortet haben.* Unter diesem Punkt müssen ganz klar drei Punkte abgezogen werden, da die anderen in dieser Diplomarbeit verglichenen APIs gratis sind, obwohl sie auf einem ähnlichen hohen Niveau entwickelt wurden.

**Bewertung:** 2

4. Gibt es eine Community für dieses API, wenn ja, ist diese Community aktiv und/oder hilfsbereit?

Restlet hat von allen verglichenen APIs die inaktivste Community. Derzeit (18.3.2019) sind auf Stackoverflow [Ho:Web18, Restlet Stackoverflow] 1.300 Diskussionen aktiv. Obwohl die API im kommerziellen Gebrauch kostenpflichtig ist, gibt es keinen offiziellen Support. Restlet verliert durch die oben genannten Gründe vier Punkte.

**Bewertung:** 1

### 8.3.3 Entwicklung

1. Kann die API im Produktivbetrieb verwendet werden?

Restlet ist ein sehr professionell aufgebautes API, es wird vor allem für Betriebe vermarktet. Weiters ist es schon relativ lange auf dem Markt da es am 27.10.2008 veröffentlicht wurde, dadurch haben die Entwickler bereits einiges an Erfahrung gesammelt. Es ist kein Grund erkennbar warum die API nicht bestens in einem Produktivbetrieb funktionieren sollte.

**Bewertung:** 5

2. Welche Java Version ist die minimale die von dem API unterstützt wird?

Restlet funktioniert in jeder Version ab Java SE 6, diese wurde am 11.12.2006 veröffentlicht. Bei diesem Punkt verdient sich die API wieder alle Punkte, da heutzutage kaum noch Systeme existieren, die eine ältere Version als Java SE 6 verwenden. Dies wird belegt von der Plumbr Java Statistik 8.1.

**Bewertung:** 5

### 3. Gibt es als Hilfe bei der Modellierung grafische Hilfsmittel?

Die Modellierung in Restlet erfolgt großteils mithilfe von grafischen Tools. Diese werden von den offiziellen Entwicklern zur Verfügung gestellt und können direkt zur Implementierung verwendet werden. Weiters ist es möglich mit diesen Tools die bereits entwickelten Services zu testen. Die grafischen Oberflächen sind alle sehr einfach gestaltet und lassen sich intuitiv bedienen. Es gibt hier drei Programme: R-Client, R-Studio und R-Cloud. Das erste Programm ist R-Client, dies ermöglicht es einem seine bereits entwickelten Services zu testen.

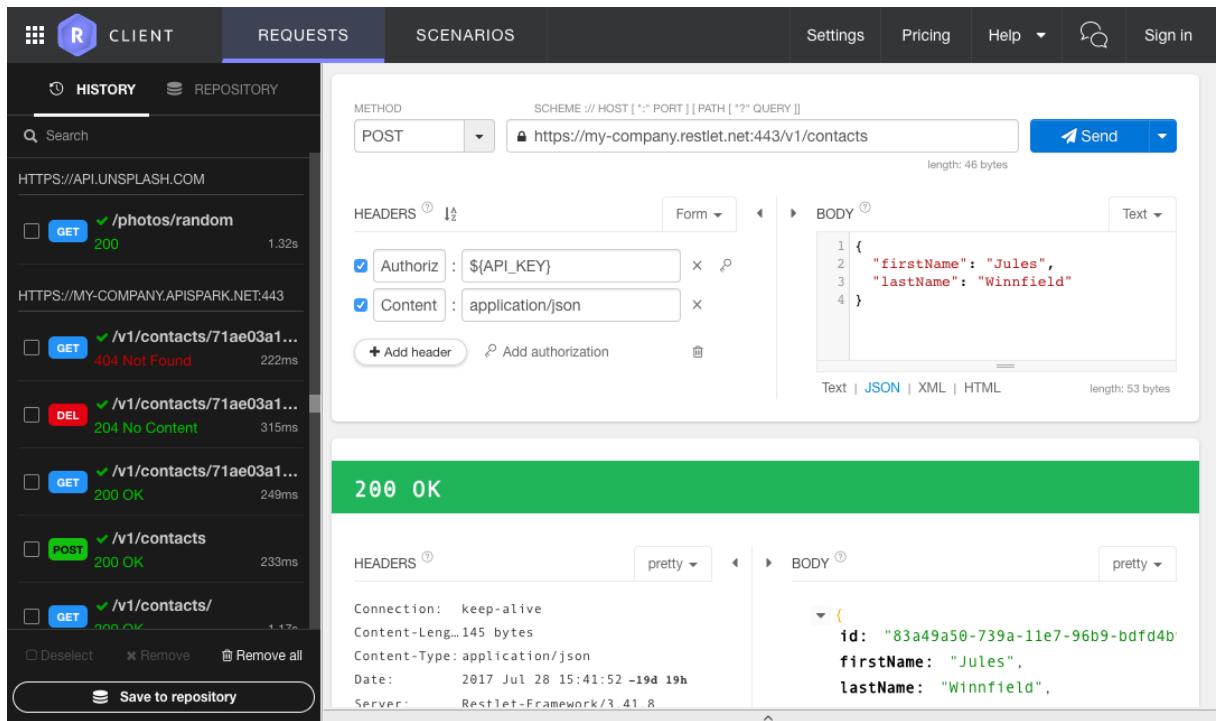


Abbildung 8.3: R-Client

[Ho:Web21, R-Client]

Das zweite Programm ist R-Studio, dies ermöglicht es einem seine Services zu verwalten. Man kann seine GitHub Versionen, Dokumentationen und noch viel mehr verwalten.

The screenshot shows the R-Studio interface for managing APIs. On the left, there's a sidebar with a tree view of API endpoints for the Spotify Web API, including methods like GET /me, GET /users/{user\_id}, and various operations on playlists. The main area displays a specific endpoint: "Check if Users Follow a Playlist" (GET /users/{user\_id}/playlists/{playlist\_id}/followers/contains). This section includes a "Try in Client" button, a description, and a detailed "REQUEST" section with "QUERY PARAMETERS". The "REQUEST" section shows a parameter "ids" with a description: "A comma-separated list of users ids". Below the REQUEST section, there are buttons for "OAS / Swagger 2.0" and "RAML 1.0". The bottom right corner features the "Restlet" logo.

Abbildung 8.4: R-Client

[Ho:Web22, R-Studio]

Das dritte Programm ist R-Cloud, dies ermöglicht es einem komplette REST-Endpunkte zu entwickeln ohne eine einzige Zeile Code zu schreiben. Weiters kann man seinen Service auch direkt in der Entwickler eigenen Cloud hosten. Wenn man sich für dies entscheidet, werden einem weitere analytische Funktionen zur Verfügung gestellt. Zum Beispiel bekommt man wöchentliche Reports, welche einem Statistiken der API Verwendung aufzeigen.

The screenshot shows the R-Cloud interface for managing APIs. At the top, there's a navigation bar with 'CLOUD', 'DASHBOARD', 'ANALYTICS', 'Pricing', 'Help', and 'Sign in'. Below that is the main dashboard for 'My Company API'. It features a sidebar with 'General information', 'Endpoints', and 'Resources'. Under 'Resources', there's a tree view for '/companies/' and '/companies/{companyId}'. Each endpoint has a list of methods (GET, POST, PUT, DELETE) with their descriptions. To the right, there's a detailed configuration panel for the '/companies/{companyId}' endpoint, showing fields for 'Relative path' (/companies/{companyid}), 'Name' (Company), 'Description', and 'Section' (My Company API\_Data). There's also a 'Try it out' button and a 'swagger' link. At the bottom right, the 'Restlet' logo is visible.

Abbildung 8.5: R-Client

[Ho:Web23, R-Cloud]

Bei diesem Punkt verdient die API alle Punkte.

**Bewertung:** 5

#### 4. Wie komplex ist die Entwicklung des Clients?

Die Entwicklung des Clients ist sehr einfach gestaltet und wird durch ein grafisches Tool unterstützt. Weiters ist es möglich sich vollständige Clients aus bereits vorhandenen REST Endpunkten generieren zu lassen. Diese können entweder in Java oder in Android generiert werden. Dadurch wird die Entwicklung massiv beschleunigt und auch um vieles vereinfacht. Es muss jedoch in Betracht gezogen werden, dass wenn man sich weigert den Client grafisch zu implementieren, die Syntax sehr fremd wirkt.

Der folgende Codeabschnitt zeigt einen sehr einfachen Restlet Android Client:

```

1 // Es muss zuerst ein Client generiert werden mit der URL
2 ClientResource cr = new ClientResource("http://privatevoid.net/
3   carsharing/rest");
4
5 CarRessource car = cr.wrap(CarRessource.class);
6
7 Car car = resource.retrieve();
8
9 resource.store(car);

```

Listing 8.6: Resteasy Client

Wie man hier erkennt, sieht die Syntax sehr fremd aus. Dies kann sehr viel Einarbeitung erfordern, wodurch sich das Projekt verzögern würde. Weiters muss erwähnt werden, dass die REST Endpunkte in Restlet standardmäßig GAE<sup>3</sup> Server verwenden, zu welchen gar kein Vorwissen besteht. Die API verliert hier 2 Punkte. **Bewertung:** 3

#### 5. Welche Android SDK ist die minimale die von dem API unterstützt wird?

Die niedrigste SDK Version die verwendet werden kann ist 19. Dies ist eine relativ neue Version und verhindert somit die Rückwertskompatibilität auf einige ältere Geräte. Laut der Android Version Statistik 8.2 sind ungefähr 3% aller Geräte nicht mit der API kompatibel. Weiters muss hier angemerkt werden, dass es sehr schwierig war diese Information zu finden. Aufgrund der oben genannten Punkte verliert die API hier einen Punkt.

**Bewertung:** 4

#### 6. Wie erprobt ist die Verwendung der APIs in Verbindung mit Android?

Restlet wurde das erste Mal am 26.1.2006 für Android veröffentlicht. Die API unterstützt das System nun schon sehr lange und es gibt kaum Meldungen von schwerwiegenden Problemen. Da die API bereits so lange in der Verbindung mit Android funktioniert, haben die Entwickler schon einiges an Erfahrung gesammelt, daher ist kein Grund erkennbar warum die API nicht bestens als Android Client im Projekt Carsharing funktionieren sollte.

**Bewertung:** 5

---

<sup>3</sup>Google App Engine

Restlet ist ein sehr professionelles API, welches schon lange existiert und bereits in vielen Einsatzgebieten erprobt wurde. Durch das automatische Generieren der Clients würde dem Projekt Carsharing viel Arbeit abgenommen werden. Es ist jedoch unerklärlch wieso die Entwicklung so stark von grafischen Oberflächen abhängt. Weiters werden viele Bibliotheken/Services genutzt, über welche absolut gar kein Vorwissen besteht. Die API wirkt daher nicht besonders ideal für das Projekt.

### 8.3.4 Punkte von Resteasy

Bewertung	Punkte	API
5	153bis 175	
4	132 bis 152	Restlet: 144
3	111 bis 131	Jersey: 114 und Resteasy: 129
2	89 bis 110	
1	0 bis 88	

Tabelle 8.3: Bewertung von Resteasy

## 8.4 Retrofit

### 8.4.1 Einleitung

Retrofit ist ein REST Client API, welches Entwicklern ermöglicht, in Android oder Java, Nachrichten mit anderen RESTful Web Services auszutauschen. Standardmäßig wird in Retrofit das Format JSON verwendet, welches durch einen Gson Konverter serialisiert wird. Es steht dem Entwickler jedoch frei sich andere Formate mit anderen Konvertern zu konfigurieren. Retrofit unterstützt folgende Konverter:

1. Gson
2. Jackson
3. Moshi
4. Protobuf
5. Wire
6. Simple XML

#### Codebeispiel

Hier sieht man wie ein Retrofit Client aussehen kann:

```
1 public interface CarClient {  
2  
3     @GET("car")  
4     Call<List<CarCurrent>> getCars();  
5  
6     @GET("car/location/{locId}")  
7     Call<List<CarCurrent>> getCars(@Path("locId") int locId);  
8  
9     @GET("car/{vin}")  
10    Call<CarCurrent> getCar(@Path("vin") String vin);  
11  
12    @GET("car/{fuel_min}/{fuel_max}")  
13    Call<List<CarCurrent>> getCars(@Path("fuel_min") int min, @Path("fuel_max") int max);  
14  
15 }
```

Listing 8.7: Retrofit Client Beispiel

Die Annotationen, welche in Retrofit verwendet werden, halten sich an den JAX-RS Standard, jedoch sind die Clients hierbei keine Klassen sondern Interfaces. Weiters geben die Methoden keine standardmäßigen Datentypen zurück, sondern sogenannte *calls*. Diesen kann mittels der Funktion *enqueue* ein Callback Objekt übergeben werden, welches dann vollkommen automatisch mittels eines *onResponse listeners* asynchron auf die Antwort des REST Endpunktes wartet. Die Funktionalität der Retrofit Clients wird detailliert anhand des Kapitels 9 erläutert.

### 8.4.2 Bewertung von Retrofit

1. Gibt es Hilfen zum Erlernen des APIs, wenn ja, welche sind vorhanden?

In diesem Punkt wird Retrofit mit der vollen Punktzahl bewertet da, es sehr viele ausführliche Tutorials gibt und darüber hinaus auch noch zahlreiche Codebeispiele zur Verfügung stehen.

**Bewertung:** 5

2. Wie gut ist die API dokumentiert?

Hier verliert Retrofit einen Punkt, da die Webseite der Entwickler [Ho:Web05, Retrofit Website] einen Mangel an Informationen aufweist. Diese können jedoch außerhalb der Webseite ausfindig gemacht werden. Was man den Entwicklern jedoch zugutehalten muss ist, dass eine sehr ausführliche Java Dokumentation vorhanden ist [Ho:Web06, Retrofit Java documentation].

**Bewertung:** 4

3. Braucht man eine Lizenz für die API, wenn ja, unter welchen Kosten ist die API verfügbar?

In diesem Punkt erreicht Retrofit die volle Punktzahl, denn es ist völlig gratis zu verwenden und benötigt nicht einmal einen API-key.

**Bewertung:** 5

4. Gibt es eine Community für dieses API, wenn ja, ist diese Community aktiv und/oder hilfsbereit?

Retrofit hat eine sehr große und aktive Community, welche auch sehr hilfsbereit zur Verfügung steht. Dies ist besonders bemerkbar auf Seiten wie Stackoverflow [Ho:Web07, Retrofit Stackoverflow] auf welcher derzeit (15.3.2019) 5.561 aktive Diskussionen offen sind. Aufgrund dessen wird hier ebenfalls die volle Punktzahl vergeben.

**Bewertung:** 5

### 8.4.3 Entwicklung

1. Kann die API im Produktivbetrieb verwendet werden?

Retrofit ist noch relativ neu, da es erst am 13.05.2013 veröffentlicht wurde, jedoch ist es von multiplen Quellen [Ho:Web08, Retrofit Ranking] zu der besten API in der Verbindung mit Android gekürt worden. Weiters wird Retrofit regelmäßig gewartet und es erscheinen kaum Berichte über ernste Probleme. Daher würde ich keinen Grund sehen warum die API im Produktivbetrieb eines Carsharing Unternehmens nicht bestens funktionieren sollte.

**Bewertung:** 5

2. Welche Java Version ist die minimale die von dem API unterstützt wird?

Retrofit funktioniert in jeder Version ab Java SE 6, diese wurde am 11.12.2006 veröffentlicht. Bei diesem Punkt verdient sich die API wieder alle Punkte, da heutzutage kaum noch Systeme existieren, die eine ältere Version als Java SE 6 verwenden. Dies wird belegt von der Plumbr Java Statistik 8.1.

**Bewertung:** 5

3. Gibt es als Hilfe bei der Modellierung grafische Hilfsmittel?

Es gibt keine Tools die einem eine grafische Modellierung erlauben, jedoch gibt es viele UML Diagramme, welche es einem ermöglichen sich die Architektur besser vorstellen zu können, daher bekommt die API hier nur zwei Punkte Abzug.

**Bewertung:** 3

4. Wie komplex ist die Entwicklung des Clients?

Retrofit gestaltet die Entwicklung so einfach wie möglich. Dies wird weiter vereinfacht, da das Team bereits sehr viel Vorwissen zu der API besitzen. Es ist jedoch unmöglich sich Teile des Clients generieren zu lassen. Weiters sollte die Kommunikation zwischen dem Jersey REST Endpunkt im Projekt Carsharing und dem Retrofit Client absolut problemlos verlaufen, da die beiden APIs bekannt sind für ihre gute Kombinierbarkeit. Diese Kommunikation wird noch genauer in dem Kapitel Verwendung des ausgewählten Client APIs im Projekt 9 beschrieben. Retrofit verdient hier durch die Schlichtheit der Entwicklung und durch die simple Kommunikation mit Jersey alle Punkte.

**Bewertung:** 5

5. Welche Android SDK ist die minimale die von dem API unterstützt wird?

Die niedrigste SDK Version die verwendet werden kann ist 10. Dies ist eine sehr alte Version, was wiederum bedeutet, dass die Applikation auf fast allen Geräten

ohne Kompatibilitätsproblemen funktioniert. Laut der Android Verwendungsstatistik 8.2 bedeutet dies, dass nur 0.5% aller Geräte weltweit nicht unterstützt werden. Hier beeindruckt Retrofit wirklich sehr stark und verdient sich erneut die volle Punktzahl.

**Bewertung: 5**

6. Wie erprobst ist die Verwendung der APIs in Verbindung mit Android?

Retrofit wurde für Android entwickelt daher ist es bestens in das System integriert. Weiters erscheinen kaum Berichte von ernsthaften Problemen mit Retrofit im Produktivbetrieb. Ein weiterer Vorteil der API ist, dass sie eine sehr niedrige Latenzzeit hat.

Durch die tolle Integration der API in Android und durch die automatisierte Verwaltung der Asynchronität bekommt Retrofit auch hier alle Punkte.

**Bewertung: 5**

	<b>One Discussion</b>	<b>Dashboard (7 requests)</b>	<b>25 Discussions</b>
<b>AsyncTask</b>	941 ms	4,539 ms	13,957 ms
<b>Volley</b>	560 ms	2,202 ms	4,275 ms
<b>Retrofit</b>	312 ms	889 ms	1,059 ms

Abbildung 8.6: Latenzbenchmark

[Ho:Web09, Latenzzeiten]

Die API erfüllt alle Voraussetzungen für das Projekt Carsharing und ist darüber hinaus noch sehr einfach zu verwenden. Das bereits bestehende Wissen hilft ebenfalls erheblich weiter, da sich der Aufwand dadurch um ein Vielfaches verringert. Es gibt keinen Grund warum in diesem Projekt nicht Retrofit als REST Client API verwendet werden sollte.

#### 8.4.4 Punkte von Retrofit

Bewertung	Punkte	API
5	153bis 175	Retrofit: 164
4	132 bis 152	Resteasy: 135 und Restlet: 144
3	111 bis 131	Jersey: 114
2	89 bis 110	
1	0 bis 88	

Tabelle 8.4: Bewertung von Retrofit

## 8.5 Zusammenfassung der APIs

Die Kriterien und die tatsächlichen Bewertungen der APIs wurden auf die Anforderungen eines Android-Clients im Projekt Carsharing abgestimmt. Dadurch kann diese Bewertung auch nur hierfür herangezogen werden. Wäre der Client in einer anderen Umgebung implementiert worden, wären die Kriterien eventuell völlig anders ausgefallen und ebenso die Bewertungen.

### 8.5.1 Jersey

#### Bewertung: 3 (114 Punkte)

Jersey ist ein sehr mittelmäßiges Client API, wobei es nichts wirklich herausragend gut oder schlecht handhabt. Es ist anzunehmen, dass Jersey in Verbindung mit Java bestens funktioniert, da diese Kombination bereits seit langem verwendet wird und auch erprobt ist. Weiters ist die API zum Entwickeln eines REST Endpunktes sehr angenehm, jedoch ist es nicht passend für das Projekt Carsharing. Dies liegt vor allem an der noch relativ jungen Integration von Android und der schlechten Dokumentation dieser Funktionalität. Da der Client im Projekt auf Android entwickelt wird, wäre gerade dies besonders wichtig gewesen.

### 8.5.2 Resteasy

#### Bewertung: 4 (135 Punkte)

Resteasy ist faktisch eine perfekte Client API. Es ist außerordentlich gut dokumentiert und stellt darüber hinaus noch hochwertigen, professionellen Support zur Verfügung. Weiters hätte dem Projektteam viel Arbeit abgenommen werden können, durch die Möglichkeit sich die Clients automatisch generieren zu lassen. Die API wirkte wie die perfekte Wahl für das Projekt Carsharing, jedoch wurde im Laufe der Recherche aufgedeckt, dass Android von Resteasy nicht unterstützt wird. Dies macht die API nutzlos für das Projekt, da die Clients auf Android Geräten funktionieren müssen.

### 8.5.3 Restlet

**Bewertung: 4 (144 Punkte)**

Restlet machte den professionellsten Eindruck von den vier Client APIs. Weiters ist dies ausführlich dokumentiert und es stehen zahlreiche Tutorials zur Verfügung. Dem Projekt Carsharing wäre außerdem viel Arbeit abgenommen worden, da die Clients automatisch für Android generiert werden können. Das Problem bei dieser API war jedoch die völlig neue Entwicklungsumgebung. Die Entwicklung erfolgt in diesem API großteils über grafische Tools und die verwendeten Bibliotheken/Services sind dem Projektteam völlig fremd. Diese großen Umstellungen machten die API sehr unattraktiv für das Projekt Carsharing.

### 8.5.4 Retrofit

**Bewertung: 5 (164 Punkte)**

Retrofit wirkte von Anfang an mit Abstand wie die beste Wahl. Die API ist sehr gut dokumentiert und darüber hinaus auch noch sehr einfach zu verwenden. Das bereits bestehende Vorwissen des Projektteams zu dieser API vereinfacht die Entwicklung um ein Vielfaches. Weiters ist Retrofit für Android entwickelt worden und hat dadurch auch die beste Integration in das System. Daher fiel die Wahl des REST Client APIs im Projekt Carsharing auf Retrofit.

# Kapitel 9

## Verwendung des ausgewählten Client APIs im Projekt

In dem Projekt Carsharing wird eine Applikation entwickelt, welche eine Gewinnsteigerung bei einem CarSharing Unternehmen bewirken und das Arbeitsleben der Mitarbeiter erleichtern soll. Im Wesentlichen soll dabei das Umparken von Autos optimiert werden. Dieses Umparken ist notwendig, wenn Fahrzeuge an wenig frequentierten Orten abgestellt wurden. Diese Fahrzeuge müssen an Orte überstellt werden, an welchen eine deutlich höhere Kundenfrequenz zu erwarten ist. Die Mitarbeiter werden über eine mobile Applikation benachrichtigt sobald ihnen ein neues Auto zugewiesen wurde. Weiters werden sie mithilfe dieser App auch navigiert und können sich damit selbst Autos zuweisen.

Zum Datenaustausch zwischen der Website und der mobilen Applikation wurde serverseitig ein REST-Endpunkt mit Jersey implementiert. Auf der Clientseite kommt Retrofit als REST-Cient zum Einsatz. Die Implementierung dieses Clients wird in diesem Kapitel erläutert.

Zunächst wurde ein GenericService entwickelt, welches die diversen Clients kreiert. Hierbei musste zuerst ein *OkHttpClient* initialisiert werden, mit welchem schlussendlich ein Retrofit Client erstellt wurde. Die Methode *getClient* wurde generisch implementiert, da es insgesamt fünf Clients gibt und dadurch ist dies wesentlich effizienter. Ebenfalls musste hier ein Gson Converter angegeben werden, da die Timestamps sonst falsch übertragen wurden.

```
1 public class GenericService {  
2     private static final Map<Class<?>, Object> CLIENT_CACHE = new HashMap<  
3         <>();
```

```
4
5  /*
6   * Diese Methode generiert einen Client durch welchen mit dem REST
7   * Endpunkt kommuniziert werden kann
8   * Diese Funktion ist generisch da im Projekt CarSharing diverse
9   * Clients in verwendung sind
10  * Es wird hier ebenfalls ziwschen den Clients der Android Applikation
11  * und der Website unterschieden mit Hilfe von keys
12  */
13 public static <T> T getClient(Class<T> cls, final String key) {
14     synchronized (CLIENT_CACHE) {
15         if (CLIENT_CACHE.containsKey(cls)) {
16             return (T) CLIENT_CACHE.get(cls);
17         }
18     }
19
20     String API_BASE_URL = "https://carsharing.privatevoid.net/server/
21         rest/";
22     Gson gson = new GsonBuilder().setLenient().setDateFormat("yyyy-MM-
23         dd HH:mm:ss").create();
24
25     /*
26      * Hier wird der HttpClient kreiert
27      */
28     OkHttpClient okClient = new OkHttpClient.Builder().addInterceptor(
29         new Interceptor() {
30             @Override
31             public Response intercept(Chain chain) throws IOException {
32                 Request r = chain.request();
33                 HttpUrl url = r.url().newBuilder().addQueryParameter("key",
34                     key).build();
35                 r = r.newBuilder().url(url).build();
36                 return chain.proceed(r);
37             }
38         }).build();
39
40     /*
41      * Hier wird nun die URL festgelegt unter wlcher der REST Endpunkt
42      * Erreichbar ist.
43      * Weiters wird hier der Converter angegeben
44      */
45     Retrofit.Builder builder = new Retrofit.Builder().baseUrl(
46         API_BASE_URL)
47         .addConverterFactory(GsonConverterFactory.create(gson));
48     Retrofit retrofit = builder.client(okClient).build();
49
50     T client = retrofit.create(cls);
51
52     synchronized (CLIENT_CACHE) {
53         CLIENT_CACHE.put(cls, client);
54     }
55 }
```

```
46         return client;
47     }
48 }
```

Listing 9.1: GenericService

## 9.1 Login

Am Beginn eines jeden Arbeitstages muss sich jeder Mitarbeiter mindestens einmal in der mobile Applikation anmelden. Dies geschieht durch Senden der E-Mail-Adresse und des Passwortes an den Server. Dieser validiert nun die zwei Parameter und sendet als Antwort ein AndroidLogin Objekt an die App. In diesem vom Server gesendeten Objekt stehen diverse wichtige Daten, wie zum Beispiel eine Liste der zur Zeit verfügbaren Autos oder der eingeloggte Mitarbeiter. Dies ist in folgendem Listing beschrieben.

```
1 public class AndroidLogin implements Serializable {
2
3     private boolean successful;
4     private User user;
5     private Job currentJob;
6     private CarCurrent cc;
7     private List<CarCurrent> cars;
8
9     .
10    .
11    .
12 }
```

Listing 9.2: AndroidLogin Klasse

Das Anmelden wird durch das Klicken auf den blauen Login-Button gestartet. Sobald dieser geklickt wurde, wird ein UserClient 9.1 über die Methode `getClient` im GenericService 9 angelegt.

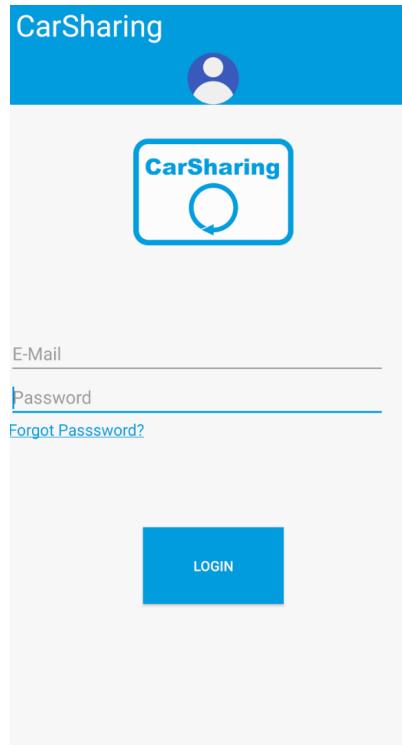


Abbildung 9.1: Login Screen der mobilen Applikation

In der mobilen Applikation wird beim Login die Funktion `checkAndroidLogin` im UserClient aufgerufen. Diese bekommt eine E-Mail-Adresse, ein Passwort und einen Token für das *Firebase Cloud Messaging*. Es wird nun ein Call typischer auf AndroidLogin 9.1 zurückgeliefert. Die Exekution dieses Calls wird in dem Listing des Login Calls 9.1 beschrieben.

```
1 public interface UserClient {  
2  
3     @GET("user/{userid}")  
4     Call<User> getUser(@Path("userid") int userId);  
5  
6     @GET("user/current/{userid}")  
7     Call<UserCurrent> getUserCurrent(@Path("userid") int userId);  
8  
9     @GET("user/byemail/{email}")  
10    Call<User> getUser(@Path("email") String email);  
11  
12    @GET("user/current/byemail/{email}")
```

```

13 Call<UserCurrent> getUserCurrent(@Path("email") String email);
14
15 @POST("user/updateLoc/{userId}/{lat}/{lng}")
16 Call<Boolean> updateUserPosition(@Path("userId") int userId, @Path("lat")
17     ) BigDecimal lat, @Path("lng") BigDecimal lng);
18
19 @GET("user/androidLogin/{email}/{pwd}/{token}")
20 Call<AndroidLogin> checkAndroidLogin(@Path("email") String email, @Path
21     ("pwd") String pwd, @Path("token") String token);
22
23 @GET("user/androidTokenRegister/{email}/{token}")
24 Call<Boolean> registerAndroidToken(@Path("email") String email, @Path(""
25     ) String token);
26
27 .
28 .
29 .
30 .
31 .
32 .
33 .
34 .
35 .
36 .
37 .
38 .
39 .
40 .
41 .
42 .
43 .
44 .
45 .
46 .
47 .
48 .
49 .
50 .
51 .
52 .
53 .
54 .
55 .
56 .
57 .
58 .
59 .
60 .
61 .
62 .
63 .
64 .
65 .
66 .
67 .
68 .
69 .
70 .
71 .
72 .
73 .
74 .
75 .
76 .
77 .
78 .
79 .
80 .
81 .
82 .
83 .
84 .
85 .
86 .
87 .
88 .
89 .
90 .
91 .
92 .
93 .
94 .
95 .
96 .
97 .
98 .
99 .
100 .
101 .
102 .
103 .
104 .
105 .
106 .
107 .
108 .
109 .
110 .
111 .
112 .
113 .
114 .
115 .
116 .
117 .
118 .
119 .
120 .
121 .
122 .
123 .
124 .
125 .
126 .
127 .
128 } 
```

Listing 9.3: UserClient

Die Methode `redirectLogin` überprüft zuallererst ob die mobile Applikation Zugriffsrechte auf den Standort des Smartphones hat. Sollte dies nicht der Fall sein, so wird noch einmal um Zugriffsrechte auf den Standort gebeten. Sollten diese dann immer noch verweigert werden, steckt der Benutzer im Anmeldefenster fest. Sobald die App alle Rechte hat die sie benötigt, werden die E-Mail-Adresse und das Passwort, sofern die beiden Felder nicht leer sind, ausgelesen und in Strings gespeichert. Nun wird in Zeile 13 ein Call typsicher auf `AndroidLogin` 9.1 angefordert. Dieser wird mithilfe der `enqueue` Funktion "gestartet", welche ein Callback Objekt, in diesem Fall als anonyme innere Klasse, übergeben bekommt. Diese Methode startet nun völlig automatisch einen asynchronen Task, welcher auf eine Antwort des REST-Endpunktes wartet. Die innere Klasse hat nun zwei Listener Methoden, einmal den `onResponse` und den `onFailure`. Sobald eine Antwort empfangen wird, werden die Daten aus dem response Objekt ausgelesen und in das DataBean gespeichert. Falls dem Mitarbeiter bereits ein Auto zugewiesen ist, wird er direkt zu der Navigations Activity 9.3 weitergeleitet. Andernfalls wird ihm die Übersicht 9.2 aller Fahrzeuge angezeigt.

```

1 public void redirectLogin(View v) {
2     if (ContextCompat.checkSelfPermission(this, Manifest.permission.
3         ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
4         UserClient loginClient = GenericService.getClient(UserClient.
5             class, "API_KEY");
6         EditText emailEdit = (EditText) findViewById(R.id.email_input);
7         EditText passwordEdit = (EditText) findViewById(R.id.
8             password_input);
9         final String email = emailEdit.getText().toString(); 
```

```
7     String password = passwordEdit.getText().toString();
8     if (email.matches("") || password.matches("")) {
9         Toast.makeText(Login.this, "Username or password is empty!",
10             Toast.LENGTH_LONG).show();
11     } else {
12         String token = Application.currentToken == null ? "null" :
13             Application.currentToken;
14         Toast.makeText(this, "Logging in...", Toast.LENGTH_LONG).show();
15     }
16     Call<AndroidLogin> loginCall = loginClient.
17         checkAndroidLogin(email, password, token);
18     loginCall.enqueue(new Callback<AndroidLogin>() {
19         @Override
20         public void onResponse(Call<AndroidLogin> call,
21             Response<AndroidLogin> response) {
22             if (response.body() != null) {
23                 AndroidLogin log = response.body();
24                 if(log.isSuccessful()) {
25                     Log.e(TAG, "LoginCall: " + log);
26                     Job curJ = log.getCurrentJob();
27                     User u = log.getUser();
28                     DataBean.setCarList(log.getCars());
29                     if (curJ == null) {
30                         Log.i(TAG, "User: " + u.getEmail() + " has no curjob");
31                         Intent imain = new Intent(Login.this,
32                             Main_drawer.class);
33                         DataBean.setUser(u);
34                         Login.this.startActivity(imain);
35                         finish();
36                     } else {
37                         Intent icur = new Intent(Login.this,
38                             CurTask.class);
39                         Log.i(TAG, "LOGTEST User: " + u.
40                             getEmail() + " Job: " + curJ.getId()
41                             + " CC:" + log.getCc());
42                         DataBean.setUser(u);
43                         DataBean.setCurCar(log.getCc());
44                         DataBean.setCurJob(curJ);
45                         Login.this.startActivity(icur);
46                         finish();
47                     }
48                 } else{
49                     Toast.makeText(Login.this, "Email or
50                         password incorrect!", Toast.LENGTH_LONG)
51                         .show();
52                     return;
53                 }
54             } else {
55                 Log.e(TAG, "LoginCall exception: " + response.
```

```

46                               code());
47                         }
48
49             @Override
50             public void onFailure(Call<AndroidLogin> call,
51                     Throwable t) {
52                 Log.e(TAG, "LoginCall failed: " + t.getMessage());
53                 Log.e(TAG, "LoginCall failed: " + t.getCause());
54                 Log.e(TAG, "LoginCall failed: " + t.
55                         getLocalizedMessage());
56                 Log.e(TAG, "LoginCall failed: " + t.getStackTrace()
57                         );
58             }
59         }  

60     }  

61 }
```

Listing 9.4: Login Call



Abbildung 9.2: Übersicht der mobilen Applikation

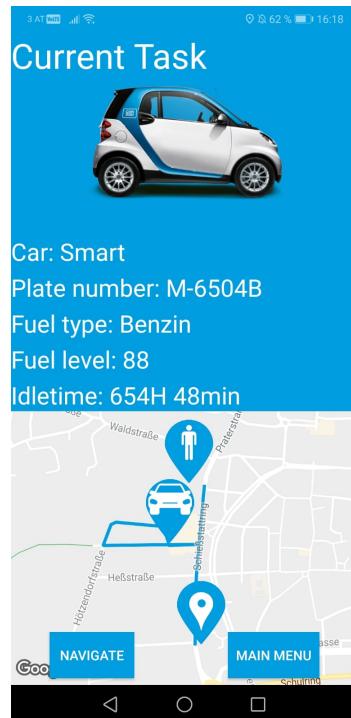


Abbildung 9.3: Navigation mithilfe der mobilen Applikation

## 9.2 Zuweisung eines Autos

In dieser Sektion wird nur die manuelle Selbstzuweisung eines Autos erläutert, da die Zuweisung über die Website durch *Firebase Cloud Messaging* gelöst wurde. Dies wurde so implementiert, da diese Technologie dem Projektteam sehr viel Arbeit abnahm.

Wenn sich ein Mitarbeiter ein Auto zuweist, wird ein Job Objekt 9.2 erstellt und dieses wird an den Server gesendet. Die Daten eines jeden Jobs stehen in einer Datenbank und man kann auf alle derzeit offenen Jobs und auf alle historisierten zugreifen. Ein Job beschreibt alle wichtigen Daten die für das Umparken eines Autos notwendig sind. Die Job Klasse ist in dem nachfolgenden Listing beschrieben.

```

1 public class Job implements Serializable {
2
3     private Integer id;
4     private Date startTime;
5     private Date endTime;
6     private Car vin;
7     private User uid;
8     private double destLat;
9     private double destLng;
10
11     .
12     .
13     .
14 }
```

Listing 9.5: Job Klasse

Mithilfe des JobClients 9.2 lassen sich diverse Daten an den Server senden oder von dem Server abfragen. Für die Zuweisung und für die Fertigstellung eines Jobs sind jedoch nur zwei Methoden essentiell: *createJob* und *finishJob*. Der Zuweisungsprozess startet sobald der Mitarbeiter auf ein Auto in der Übersicht 9.2 oder auf eines in der Liste 9.5 klickt. Durch das Starten der *Assign Car* Activity wird automatisch ein Job-Client über das GenericService 9 erzeugt. Dieser ermöglicht es nun einen Call über die Methode *createJob* zu generieren, welcher danach durch die Funktion *enqueue* "gestartet" wird.

```

1 public interface JobClient {
2
3     @GET("job{userId}")
4     Call<List<Job>> getJobsForUser(@Path("userId") int id);
5
6     @GET("job/finished/{userId}")
```

```

7   Call<List<Job>> getFinishedJobsForUser(@Path("userId") int id);
8
9 @GET("job/current/{userId}")
10 Call<Job> getCurrentJobForUser(@Path("userId") int id);
11
12 @POST("job/create/{userId}/{vin}/{destLat}/{destLng}")
13 Call<Job> createJob(@Path("userId") int uid, @Path("vin") String vin,
14                      @Path("destLat") double destLat, @Path("destLng") double destLng);
15
16 @POST("job/finish/{jobId}")
17 Call<Boolean> finishJob(@Path("jobId") int jid);
18 }
```

Listing 9.6: JobClient

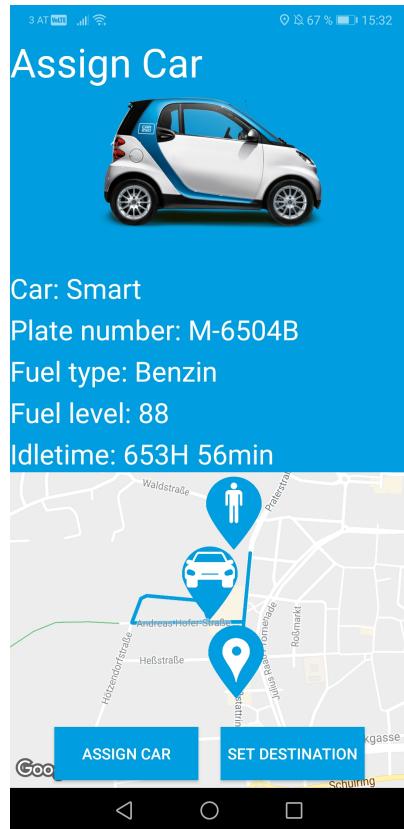


Abbildung 9.4: Zuweisung eines Autos mithilfe der mobilen Applikation

Fuellevel
Car: Smart Platenumber: C-9504D Fuellevel: 100
Car: Smart Platenumber: C-9204A Fuellevel: 100
Car: Smart Platenumber: J-1504Y Fuellevel: 100
Car: Smart Platenumber: O-0310T Fuellevel: 100
Car: Smart Platenumber: L-5573K Fuellevel: 100
Car: Smart Platenumber: X-1637Q Fuellevel: 100
Car: Smart Platenumber: X-2262N Fuellevel: 100
Car: Smart Platenumber: I-0629X Fuellevel: 100
Car: Smart Platenumber: B-7967G Fuellevel: 100
Car: Smart Platenumber: E-5979S Fuellevel: 100

Abbildung 9.5: Liste der Autos

Mithilfe der *assignJob* Methode wird dem Mitarbeiter nun der Job und somit auch das Auto endgültig zugewiesen. Als erstes wird überprüft ob überhaupt ein Benutzer angemeldet ist, oder ob bei dem Anmeldevorgang etwas schiefging. Danach wird überprüft ob der Mitarbeiter sich bereits für eine Zielposition entschieden hat. Dies geschieht zuerst durch ein langes Klicken auf die Karte in der *AssignCar* Activity 9.4, danach erscheint dort ein Marker, welcher das Ziel kennzeichnet. Nach der Zielsetzung muss das Ziel über den Button *SET DESTINATION* auch noch festgelegt werden. Sobald die Destination fest steht, wird der Call ausgeführt und sendet nun alle für den Job relevanten Daten an den Server. Die Applikation wartet nun asynchron auf eine Antwort des Servers und sobald dieser das fertige Job Objekt zurücksendet, wird der neu erstellte Job in das *DataBean* gespeichert. Hiermit ist der Zuweisungsprozess abgeschlossen.

```
1 public void assignJob(View v) {
2     if(DataBean.getUser() != null) {
3         if (dest != null) {
4             final Intent i = new Intent(this, CurTask.class);
5             JobClient jclient = GenericService.getClient(JobClient.
6                 class, "API_KEY");
7             Call<Job> callJob = jclient.createJob(DataBean.getUser().
8                 getId(), DataBean.getCurCar().getCar().getVin(), dest.
9                 lat, dest.lng);
10            callJob.enqueue(new Callback<Job>() {
11                @Override
12                public void onResponse(Call<Job> call, Response<Job>
13                    response) {
14                    Log.i(TAG, "Job creation status: " + response.code
15                        ());
16                    if (response.body() != null) {
17                        DataBean.setCurJob(response.body());
18                        AssignCar.this.startActivity(i);
19                        AssignCar.this.finish();
20                        Log.i(TAG, "Job creation: uid:" + DataBean.
21                            getUserId() + " vin: " + DataBean.
22                            getCurCar().getCar().getVin());
23                        Log.i(TAG, "Job creation status: " + response.
24                            code());
25                        Log.i(TAG, "Job creation: " + response.body());
26                    } else {
27                        Log.e(TAG, "Job creation rejected : " +
28                            response.code());
29                        Toast.makeText(AssignCar.this,"User already has
30                            a job",Toast.LENGTH_LONG).show();
31                    }
32                }
33
34                @Override
35                public void onFailure(Call<Job> call, Throwable t) {
```

```
26             Log.e(TAG, "Job creation failed");
27         }
28     });
29
30     } else {
31         Toast.makeText(this, "No destination set", Toast.
32             LENGTH_LONG).show();
33     } else{
34         Log.e(TAG, "assignJob DataBean user is null");
35     }
36 }
```

Listing 9.7: Create Job Call

Jedes Auto das auf der Karte in der Übersicht 9.2 dargestellt wird, hat einen sogenannten Geofence [Ho:Web20, Create and monitor geofences] zugewiesen. Diese Geofences benachrichtigen den Mitarbeiter, sobald er nah genug an seinem Zielort ist. Wenn der Benutzer am Ziel angekommen ist muss der Job abgeschlossen werden, dies passiert durch die Methode *finishJob*. Hierfür wird ebenfalls ein JobClient 9.2 über ein GenericService 9 erzeugt, über welchen dann ein Call generiert wird. Diesem Call wird nun die Id des aktiven Jobs übergeben und dieser überträgt diese asynchron an den Server. Nun weiß der Server, dass dieser Job abgeschlossen ist und stellt den Mitarbeiter für weitere Jobs wieder frei. Ebenfalls muss der Job aus dem *DataBean* gelöscht werden. Nun kann der gesamte Zuweisungsprozess von neuem beginnen.

```
1 protected void onHandleIntent(Intent intent) {
2     .
3     .
4     .
5     JobClient client = GenericService.getClient(JobClient.class,
6             "API_KEY");
7     Call<Boolean> jCall = client.finishJob(DataBean.getCurJob() .
8             getId());
9     jCall.enqueue(new Callback<Boolean>() {
10         @Override
11         public void onResponse(Call<Boolean> call, Response<Boolean>
12             > response) {
13             if(response.body()){
14                 Log.i(TAG, "Job finished succ");
15                 DataBean.setCurJob(null);
16                 DataBean.setCurCar(null);
17                 GeofenceTransitionsIntentService.this.startActivity
18                     (i);
19                 Toast.makeText(GeofenceTransitionsIntentService.
20                     this, "Job finished", Toast.LENGTH_LONG).show();
21             } else{
22             }
```

```

17             Log.e(TAG, "Job finished failed");
18     }
19 }
20
21 @Override
22 public void onFailure(Call<Boolean> call, Throwable t) {
23     Log.e(TAG, "Job finished Call failed");
24     Log.e(TAG, "Job finished failure: " + t.getMessage());
25     t.printStackTrace();
26 }
27 });
28 .
29 .
30 }
31

```

Listing 9.8: Create Job Call

## 9.3 Fazit der Implementierung von Retrofit

Die Entwicklung mit Retrofit unter Android verlief nahezu problemlos. Es kamen kurzzeitig kleine Kompatibilitätsprobleme auf, jedoch wurden diese schnell beseitigt. Das einzige länger bestehende Problem war die korrekte Übertragung des Datums. Es kostete dem Projektteam einiges an Arbeitszeit bis herausgefunden wurde, dass es für solche Situationen einen *Gson converter* gibt, welcher das Problem zur Gänze behob.

```

1 Gson gson = new GsonBuilder().setLenient().setDateFormat("yyyy-MM-dd HH:mm:ss").create();

```

Listing 9.9: Gson converter

Ein weiterer erwähnenswerter Punkt ist, dass die automatische Asynchronität von Retrofit meistens zwar sehr hilfreich war, jedoch verursachte diese auch einige Crashes. Dies war meistens dadurch bedingt, dass Android andere Methoden bereits ausführen wollte, obwohl einige angeforderte Daten noch nicht fertig übertragen waren. Diese Fehler sind zwar nicht schwer zu beheben, jedoch dauert es oft sehr lange sie zu finden, da sie nicht immer auftreten. Die Wahrscheinlichkeit des Fehlerauftretens ist nämlich abhängig von der Geschwindigkeit der Internetverbindung. Daher funktioniert das Programm manchmal ohne Probleme und andere Male stürzt es sofort ab, was oft für Verwirrung sorgte.

Alles in allem war Retrofit ein sehr angenehmes und problemloses API, dies vereinfachte die Entwicklung immens. Die Entscheidung fiel definitiv auf das richtige API.

# Kapitel 10

## Einführung in die Webentwicklung

### 10.1 Übersicht und Geschichte der Webentwicklung

Als Webentwicklung bezeichnet man die Softwareentwicklung von Webanwendungen, Webservices oder anderer komplexer Websites.

Die in der Webentwicklung genutzten Technologien veränderten sich parallel zur Entwicklung des Internets. Von 1992 bis 1999 dominierten statische HTML<sup>1</sup>-Seiten. In den frühen 2000er Jahren kamen dynamische Webseiten auf, die mit Server Side Scripting in Programmiersprachen wie PHP realisiert wurden. Bei den serverseitigen Programmiersprachen dominieren heute noch (Februar 2019) PHP und ASP.NET.

1997 wurde JavaScript standardisiert. Ab 2005 begann JavaScript die serverseitigen Scriptsprachen zu überflügeln. Ein weiterer Schub kam durch die Einführung von AJAX<sup>2</sup> (Asynchrone Datenübertragung zwischen Browser und Server). 2009 wurde Node.js veröffentlicht, wodurch serverseitiges JavaScript populär wurde und heute weit verbreitet ist.

Bedeutungslos gewordene Technologien sind Adobe Flash und Java-Applets. Flash spielte vom Anfang der 2000er bis etwa 2010 eine wichtige Rolle, um multimediale und interaktive Inhalte im Web auszuliefern. Java-Applets wurden 1997 eingeführt und 2015 offiziell für veraltet erklärt.

---

<sup>1</sup>Hypertext Markup Language

<sup>2</sup>Asynchronous JavaScript and XML

## 10.2 Statische und dynamische Webseiten

Wenn man im Internet über einen Browser eine Webseite aufruft, dann wird diese von dem Server geladen, auf dem sie gespeichert ist. Die Webseite enthält den Quellcode und der Browser stellt dem Betrachter den Inhalt der Webseite so dar, wie es im Quellcode festgelegt wurde. Dazu werden evtl. weitere Dateien wie z.B. Bilder, Videos, CSS<sup>3</sup>- oder sonstige Scriptdateien nachgeladen, die in die Webseite eingebunden sind.

Der Betrachter bekommt eine geordnete und anschauliche Webseite zu sehen und nicht den Quellcode selbst. Dieser bleibt im Hintergrund, kann aber bei Bedarf angezeigt werden.

Der Quellcode, den der Browser verarbeitet, ist in HTML oder XHTML<sup>4</sup> geschrieben. Das sind keine klassischen Programmiersprachen, sondern Auszeichnungssprachen zur Strukturierung und Beschreibung von elektronischen Dokumenten, wie das bei Webseiten der Fall ist.

Eine Webseite kann direkt in HTML geschrieben werden. In dem Fall spricht man von statischen Webseiten und das kann ohne die Verwendung einer klassischen Programmiersprache verwirklicht werden. Man nennt sie deshalb statisch, weil die Inhalte wie z.B. Überschriften, Texte und Bilder und sonstige Inhalte fest definiert sind und wenn man eine statische Webseite aufruft, dann bleibt der Inhalt so, wie es vom Ersteller festgelegt wurde.

Die folgende Grafik zeigt den Vorgang an, wenn eine Webseite aufgerufen wird.



Abbildung 10.1: Aufruf einer Webseite

<sup>3</sup>Cascading Style Sheets

<sup>4</sup>Extensible Hypertext Markup Language

## 10.3 Dynamische Webseiten

Anders ist es, wenn man Webseiten mit dynamischen Inhalten erstellen möchte. Dynamische Inhalte bedeuten, dass sich die Inhalte einer Webseite bei jedem Aufruf theoretisch ändern können. Beispielsweise könnte man festlegen, dass auf der Webseite an einer bestimmten Position immer die aktuelle Anzahl von Besuchern angezeigt wird.

Da sich die Anzahl jede Sekunde ändern kann, braucht man eine Lösung, die bei jedem Aufruf der Webseite danach schaut, wie die aktuelle Anzahl ist und an die entsprechende Stelle die Anzahl schreibt und erst dann an den Benutzer ausliefert.

Durch die Dynamik einer Webseite erhält man eine Vielzahl an Möglichkeiten. Häufig arbeitet auf dem Server im Hintergrund ein Datenbankserver mit einer oder mehreren Datenbanken. Beim Aufruf einer Webseite mit einer Datenbankverbindung wird die Datenbank abgefragt und die Ergebnisse der Datenbankabfragen werden in den Quellcode der Webseite eingefügt und danach ausgeliefert.

Solche Aufgaben lassen sich nicht mit statischem HTML realisieren sondern man muss diese mit einer Programmiersprache implementieren. Im Internetbereich hat sich vor allem PHP in Zusammenarbeit mit dem Datenbankserver MySQL durchgesetzt. Es gibt noch weitere Programmiersprachen, mit denen man dynamische Webseiten erstellen kann, z.B. Perl oder Python.

All diese Programmiersprachen haben die Gemeinsamkeit, dass sie den Code serverseitig ausführen. Dazu benötigt man auf dem Server einen entsprechenden Interpreter. Bei PHP heißt dieser PHP-Interpreter.

Die folgende Grafik beschreibt den Aufruf einer dynamischen Webseite.

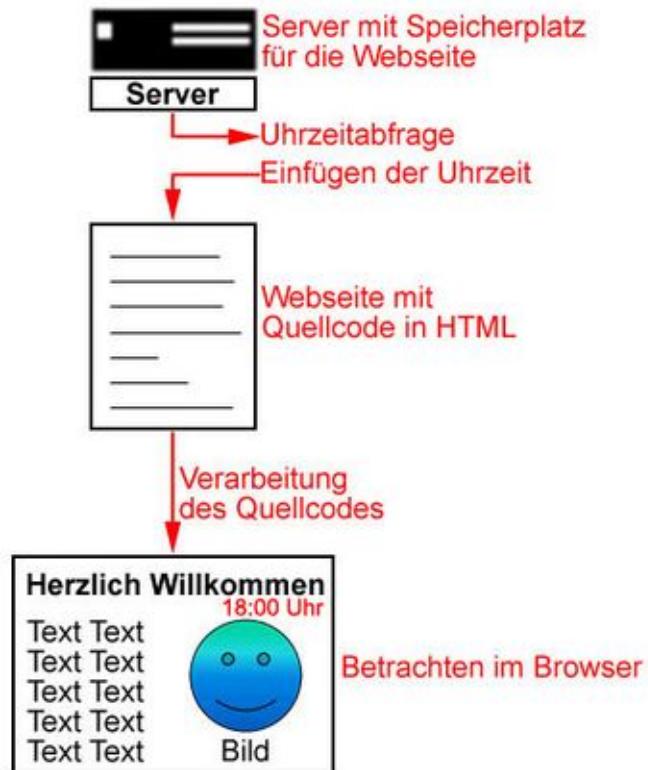


Abbildung 10.2: Aufruf einer dynamischen Webseite

## 10.4 Unterschied zwischen serverseitigen und clientseitigen Programmiersprachen

Der Unterschied liegt darin, dass bei serverseitigen Programmiersprachen die Dynamik auf dem Server geschieht und die fertige Seite an den Benutzer ausgeliefert wird, während bei clientseitigen Programmiersprachen die Dynamik erst auf dem Rechner des Benutzers stattfindet.

### Serverseitige Programmiersprache

- Der Benutzer möchte eine Webseite aufrufen und sendet die Anfrage an den Server, auf dem die Webseite gespeichert ist. Auf dem Server bekommt die Anfrage ein Serverdienst bzw. Webserver, der für die Auslieferung von Webseiten zuständig ist.
- Der Webserver erkennt, dass die Webseite keine statische Webseite ist und Programmiercode enthält. Der Webserver weiß mit dem Programmiercode jedoch

nichts anzufangen. Denn, er ist nur zuständig für die Auslieferung der Webseiten.

- Daher schickt der Webserver die Webseite durch einen 'Interpreter', z.B. PHP-Interpreter. Dieser kann die Anweisungen im Programmiercode verstehen.
- Der Webserver übermittelt die fertige Webseite an den Benutzer.

### **Clientseitige Programmiersprache**

- Der Benutzer möchte eine Webseite aufrufen und sendet die Anfrage an den Server, auf dem die Webseite gespeichert ist. Auf dem Server bekommt die Anfragen ein Serverdienst bzw. Webserver, der für Auslieferung von Webseiten zuständig ist.
- Die Webseite enthält keinen serverseitigen Code, sondern clientseitigen. Da der Webserver nur für serverseitigen Code zuständig ist wird die Webseite an den Benutzer ausgeliefert, ohne sie durch einen Interpreter zu schicken.
- Der Benutzer erhält die Webseitem die in der Regel eine Mischung aus HTML und dem clientseitigen Code enthält.
- Neben dem HTML-Code verarbeitet der Browser diesmal zusätzlich den clientseitigen Programmcode.

## **10.5 Vor- und Nachteile bei client-und serverseitiger Programmierung**

Wenn man eine Webseite mit clientseitigem Programmcode erstellt, dann liegt der Nachteil darin, dass die Verarbeitung des Codes vom Anwender unterbunden werden kann. Theoretisch könnte man mit clientseitigem Code schadhafte Vorgänge auf dem Computer des Anwenders ausführen. Daher haben alle gängigen Browser die Möglichkeit, die Verarbeitung von clientseitigem Code auszuschalten. Vorteilhaft ist, dass man z.B. mit clientseitiger Programmierung die technische Umgebung des Benutzers berücksichtigen kann. Wenn man z.B. mit serverseitiger Programmierung die Uhrzeit des Servers abfragt und in die Webseite einfügt, der Benutzer aber nicht in der selben Zeitzone ist, würde aus Benutzersicht eine 'falsche' Uhrzeit angezeigt werden. Mit clientseitiger Programmierung könnte man die Zeitzone des Benutzers berücksichtigen und die 'richtige' Uhrzeit anzeigen.

Ein weiterer Nachteil bei serverseitiger Programmierung ist, dass die Ausführung des Programmcodes nur bei jedem Aufruf der Webseite erfolgt. Im Falle einer Uhrzeitabfrage würde z.B. die aktuelle Uhrzeit des Servers zwar eingefügt, jedoch nicht mehr aktualisiert werden. Der Benutzer müsste theoretisch jede Sekunde die Webseite neu anfordern, um die aktuelle Uhrzeit angezeigt zu bekommen. Es gibt zwar Technologien (AJAX), um Daten vom Server zu laden, ohne die Webseite neu aufzurufen. Hierfür ist jedoch der Einsatz beider Programmierarten (client- und serverseitig) notwendig und funktioniert nicht, wenn der Benutzer die Verarbeitung von clientseitigem Code deaktiviert hat. Der Vorteil bei rein serverseitiger Programmierung ist, dass man nicht darauf angewiesen ist, dass der Benutzer die Ausführung von clientseitigem Programmcode einschaltet. Denn, der Code wird auf dem Server verarbeitet und das hängt nicht von den Einstellungen des Benutzers ab. Insbesondere bei Vorgängen, die keinesfalls von den Einstellungen des Benutzers abhängen dürfen und Manipulationen ausgeschlossen werden sollen, setzt man auf serverseitige Programmierung.

Bespielsweise könnte man die Benutzereingaben eines Kontaktformulars mit einer clientseitigen Programmiersprache prüfen und z.B. die Länge der Postleitzahl auf 5 Ziffern beschränken. Das Problem dabei ist, dass die Ausführung von clientseitigem Code vom Anwender deaktiviert werden kann. Der Benutzer könnte in diesem Fall z.B. die Ausführung des clientseitigen Codes deaktivieren und in das Formularfeld 10 Ziffern oder schadhaften Code eintippen und das Formular an den Server senden. Clientseitige Programmiersprachen sind in solchen Fällen nicht empfehlenswert und es muss serverseitig geprüft werden, was der Anwender in welcher Form eingetippt hat und bei Falscheingaben muss die weitere Verarbeitung mit einer Fehlermeldung gestoppt werden. Da der Anwender keinen Einfluss darauf hat, was auf dem Server geschieht, wird somit eine Möglichkeit zur Falscheingabe oder Manipulation unterbunden. In der Praxis sieht es meistens so aus, dass Webseiten mit einer Mischung aus HTML, serverseitigem und clientseitigem Code erstellt werden. In HTML wird meistens das Grundgerüst und unter Zuhilfenahme von CSS das Aussehen der Webseite bzw. die Positionierung der Inhalte festgelegt. Clientseitige Programmiersprachen werden für 'harmlose' Vorgänge verwendet, z.B. um schöne Effekte bei Bildern zu realisieren. Serverseitige Programmiersprachen verwendet man immer dann, wenn man sicherstellen will, dass das, was geschehen soll, auch tatsächlich nur in der gewünschten Form geschieht. AJAX-Technologie, um Inhalte vom Server nachzuladen, ohne die Webseite neu aufzurufen, verwendet man z.B. bei Inhalten, die dynamisch vom Server bereitgestellt werden sollen. So braucht nur der Teil vom Server geladen zu werden, der vom Benutzer betrachtet wird. Das Ressourcen auf dem Server und ist für den Benutzer angenehmer, da die Webseite nicht neu geladen werden muss. Bei rein serverseitiger Programmierung würde der Benutzer beim Aufruf der Webseite den gesamten Inhalt laden.

Ein weiterer Aspekt für serverseitige Implementierung ist, dass bei clientseitigen Sprachen die Businesslogik vom Benutzer der Seite eingesehen werden kann. Dies ist möglicherweise Sicherheitskritisch.

## 10.6 PHP

PHP ist eine Scriptssprache mit einer an C und Perl angelehnten Syntax, die hauptsächlich zur Erstellung dynamischer Webseiten oder Webanwendungen verwendet wird.

Erstmals wurde PHP im Jahr 1995 veröffentlicht. Als Entwickler von PHP gilt Rasmus Lerdorf. Mittlerweile wurden verschiedene Versionen der Skriptsprache veröffentlicht. Die aktuellste ist PHP 7.3, welche am 6. Dezember 2018 erschienen ist.

PHP wurde Anfang 2013 auf etwa 244 Millionen Webseiten eingesetzt und Ende 2017 von 83% aller Webseiten als serverseitige Programmiersprache verwendet. PHP ist die am häufigsten verwendete Programmiersprache zum erstellen von Webseiten.

### 10.6.1 Vorteile von PHP

- PHP ist leicht zu lernen - Umsteiger werden auf den ersten Blick die Ähnlichkeiten mit Java oder C++ erkennen und sich schon nach geringem Zeitaufwand wohl fühlen. Neulinge können schnell in die Programmierung einsteigen und bald erste Ergebnisse im Browser betrachten.
- PHP ist frei verfügbar - Jeder kann den Quellcode von PHP einsehen und nach seinen Vorstellungen verbessern oder weiterentwickeln.
- PHP ist vielseitig - Es sind PHP-Versionen für Unix, Windows, Mac OS und einige andere Plattformen verfügbar. PHP hat standardmäßig viele Schnittstellen zu anderen Programmiersprachen und Datenbanken.
- PHP ist populär - Mittlerweile verwenden viele Millionen Internetseiten PHP. In unterschiedlichsten Einsatzgebieten und Webseiten hat sich PHP bewährt.
- PHP ist gut dokumentiert - Die einfache Erlernbarkeit von PHP ist wohl vor allem der guten Internetpräsenz zu verdanken. Allerdings gibt es auch sehr viele Bücher und Foren zum Thema PHP.

### 10.6.2 Nachteile von PHP

- PHP hat bislang keinen Anwendungsserver - Das heißt, alle Variablen müssen bei jedem Aufruf wieder neu geladen bzw. berechnet werden.

- PHP ist populär - Was oft ein Vorteil ist kann auch ein Nachteil sein. Schwachstellen in PHP oder PHP-Skripten können schneller und wirksamer ausgenutzt werden, weil viele potentielle Opfer verfügbar sind.
- PHP ist leicht zu erlernen - Das kann auch ein Nachteil sein, weil viele PHP-Projekte ohne das nötige Hintergrundwissen verwirklicht werden. Die Folge davon ist, dass viele Webseiten gravierende Sicherheitsmängel aufweisen.
- PHP ist eine rein interpretierte Sprache und die erzeugten Programme damit vergleichsweise langsam in der Ausführung.
- PHP hat im Gegensatz zu anderen Skriptsprachen keine Möglichkeiten der Programmierung von Threads.

## 10.7 JavaScript

JavaScript ist eine Skriptsprache, welche ursprünglich 1995 von Netscape für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern.

JavaScript an sich ist sehr kompakt und trotzdem flexibel. Viele Entwickler haben noch weitere Tools für die Arbeit mit JavaScript geschrieben, um noch mehr Effizienz mit wenig Aufwand aus JS herauszuholen. Diese Funktionen sind:

- Programmierschnittstellen (APIs), die in Browsern implementiert wurden, um diese um JS-Funktionen zu erweitern, z.B. das dynamische Erstellen von HTML oder das Einstellen eines CSS-Styles, Erzeugen von 3D-Grafiken und vieles mehr...
- Drittanbieter-APIs, die es Entwicklern ermöglichen, Funktionen von anderen Seiten in eigene Seiten einzubinden, z.B. von Twitter oder Facebook
- Drittanbieter-Frameworks und Bibliotheken, die man zu HTML hinzufügen kann, die es ermöglichen, Webseiten schneller zu erstellen.

### 10.7.1 Vorteile von JavaScript

- Problemlösung beim Validieren - JavaScript ist eine hervorragende Lösung, die beim Validieren von Eingabeformularen auf der Clientseite implementiert wird.

Wenn der Benutzer vergisst, seinen Namen in ein Formular einzugeben, wird es eine JavaScript-Validierungsfunktion bemerken und das Formular nicht an den Server schicken. Es ist möglich die Validierung serverseitig zu erledigen doch mittels JavaScript ist es viel sicherer.

- Ein weiteres Einsatzgebiet für das JavaScript hervorragend geeignet ist, ist die Erstellung von dynamischen Effekten wie z.B. Diaschows
- Schnell - Ein weiter Vorteil ist, dass das Skript nicht auf dem Webserver ausgeführt wird, sondern mittels eines Browsers auf Seiten des Benutzers. Das spart nicht nur Rechenleistung beim Server, sondern auch Kommunikationswege. Ständiges Anfragen und Antworten zwischen Client und Server fällt damit weg. Hierfür läuft in dem Browser eine sogenannte 'JavaScript Engine', die das Skript interpretiert. So kann auf Clientseite eine Dynamik implementiert werden, ohne dass er mit dem Server interagieren muss.
- Fortlaufende Weiterentwicklung - JavaScript wird gemeinsam von Netscape weiterentwickelt. Dem Entwickler werden durch neue Versionen laufend neue Möglichkeiten geboten.

### 10.7.2 Nachteile von JavaScript

- Unterschiedliche Browserversionen - Durch die hohe Anzahl an unterschiedlichen Browsern und dem Fakt, dass beinahe jeder Browser Eigenheiten beim Umgang mit JS aufzeigt, ist es fast unmöglich, einen Code zu erstellen, der auf allen Browsern die identische Ergebnisse liefert.
- Von vielen Usern deaktiviert - In Anbetracht der Gefahren, die von JavaScript ausgehen können, haben viele Benutzer JavaScript generell deaktiviert. Beim Internet Explorer gibt es die Möglichkeit, nur vertrauenswürdigen Seiten die Möglichkeit von JS zu geben. Alle anderen Seiten können davon keinen Gebrauch machen. Besonders ärgerlich ist es dann, wenn die Bedienung einer Webseite JavaScript voraussetzt
- Cross-Site-Scripting(XSS) - Eine der bekanntesten Attacken stellt das Cross-Site-Scripting (Codeeinschleusung) dar. Dabei kann ein Angreifer Webseiten mit einem Skript infizieren. Wenn ein Internetnutzer diese infizierten Webseiten aufruft, wird in seinem Browser das Skript im Hintergrund ausgeführt, ohne dass er dies merkt. Durch diese Schwachstelle kann der Angreifer geheime Informationen ausspähen, die zwischen dem Client und dem Server ausgetauscht werden. Moderne Browser haben Gegenmaßnahmen ergriffen, die Risiken minimieren

sollen. So bieten einige Browser Mechanismen an, mit denen XSS-Angriffe erkannt und verhindert werden sollen aber es gilt, dass das auch keinen 100%igen Schutz gibt.

### 10.7.3 Node.js

Node.js ist eine Open-Source-Entwicklungsplattform für die serverseitige Ausführung von JavaScript Code. Node.js setzt auf Google V8 JavaScript Virtual Machine auf. Normalerweise laufen JavaScript-Engines im Webbrowser. Sie decken also die Client-Seite in einer Client/Server-Anwendung ab. Node.js-Bibliotheken sind dagegen für die Entwicklung von serverseitigen JavaScript-Anwendungen ausgelegt.

Node.js läuft auf einem HTTP-Server und beschäftigt nur einen Thread pro Prozess. Node.js-Anwendungen sind ereignisbasiert und laufen asynchron ab. Der Code, der auf der Node.js-Plattform erstellt ist, folgt nicht dem traditionellen Ablauf: empfangen, verarbeiten, senden, warten, empfangen. Stattdessen werden eingehende Requests in einem fortlaufenden Event Stack so verarbeitet, das immer ein Request nach dem anderen abgesetzt wird, ohne dass auf eine Antwort gewartet wird.

Dies ist anders als bei den Mainstream-Modellen, wo größere, komplexere Prozesse mit mehreren Threads gleichzeitig ablaufen. Hier wartet jeder Thread auf seine entsprechende Antwort, bevor die Bearbeitung weitergeht.

Nach Angaben des Erfinders Ryan Dahl, ist einer der großen Vorteile von Node.js, dass es nicht den Input/ Output blockiert. Einige Entwickler stehen Node.js kritisch gegenüber. Die Kritik richtet sich dagegen, dass, falls ein einziger Prozess sehr viele CPU-Zyklen benötigt, eine Blockade entsteht, die die ganze Anwendung zum Absturz bringen kann. Die Befürworter von Node.js sagen dagegen, dass die CPU-Bearbeitungszeit zu vernachlässigen sei, da Node.js auf vielen kleinen Prozessen basiert.

#### Vorteile von Node.js

- Blitzschnelle Ausführung von Webanwendungen
- Skalierbarkeit gegeben - Es lassen sich Applikationen schreiben, welche mit einer großen Anzahl von Anfragen gut umgehen kann.

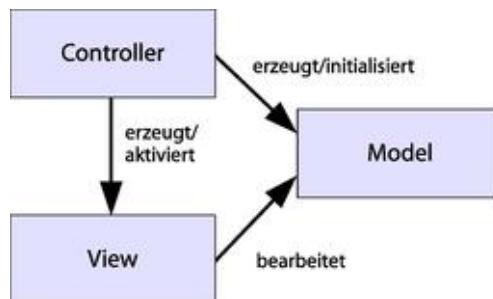
#### Nachteile von Node.js

- Nicht für rechenintensive Aufgaben geeignet - Falls es um Aufgaben geht, in denen viel Rechnerkapazität verwendet wird, dann ist Node.js die falsche Wahl.
- Oftmals fehlende Rückwärtskompatibilität - Es kann sein, dass man seinen bestehenden Code anpassen muss, wenn es eine neue Version gibt. Dieses Problem gibt es bei vielen JS Technologien.
- Keine fertige Technologie - Node.js gibt es erst seit 2009, deshalb sollte man von einer unausgereiften Technologie sprechen. Im Vergleich, PHP gibt es nun seit mehr als 20 Jahren. Sollte man Node.js in Projekten verwenden, sollte man mit höherem Programmieraufwand rechnen.

## 10.8 Java

### 10.8.1 Geschichte der Webentwicklung mit Java

Im Java-bereich war 1997 die Servlet-Technologie der erste Schritt zur dynamischen Generierung von HTML-Seiten am Server. Diese Technologie beruht im Wesentlichen darauf, dass in den Java-Code Befehle eingebunden werden, die HTML Seiten erzeugen. Doch für komplexere HTML-Seiten war diese Vorgehensweise nicht optimal, weswegen die JavaServer Pages entwickelt wurden. HTML ist hier die treibende Kraft und in die einzelnen Tags der HTML sind in sogenannten Scriptlets die Aufrufe der Java-Methoden zur Ausgabe der dynamischen Teile der HTML-Seite eingebunden. Dieser Ansatz erleichterte die Erstellung von komplexen HTML-Seiten mit viel eingebautem JavaScript-Code und einer hohen Anzahl an CSS-Auszeichnungen ungemein. Das Problem der JSP-Technologie ist, dass die Entwickler begannen, immer mehr Code in die einzelnen JSP Seiten aufzunehmen, bis erneut eine hochkomplexe Mischung aus HTML-Tags und Java-Code entstand. Diese Mischung war genauso schlecht wartbar wie die in Servlet-Code eingebaute HTML-Generierung. Zur Lösung dieses Problems traten Webframeworks auf den Plan. Der Entwickler wird bei Benutzung eines Frameworks dazu angehalten, möglichst große Teile der Layoutbeschreibung in einer SeitendeklarationsSprache wie JSP zu erstellen und gleichzeitig möglichst wenig Funktionalität im Sinne von Anwendungslogik zwischen die Elemente der SeitendeklarationsSprache einzufügen. Model-View-Controller (MVC): Ein klarer Schnitt zwischen den Bereichen Modell, Ansicht und Steuerungslogik ist also notwendig - dieses Entwicklungsmuster wird auch Model-View-Controller-Muster (MVC) genannt und ist in der folgenden Abbildung dargestellt.

Abbildung 10.3: *Model-View-Controller*

Das Ziel von MVC ist eine höhere Flexibilität für Änderungen und Erweiterungen und eine größere Wiederverwendbarkeit der einzelnen Komponenten.

Im Fall von JavaServer Faces kommt MVC in der folgenden Ausprägung zum Einsatz: Der Controller wird durch das Faces Servlet implementiert. Die Views sind Facelets, d. h. XHTML-Dokumente, die einige besondere Tag-Bibliotheken verwenden. In JSF können auch andere View-Technologien zum Einsatz kommen, worauf hier nicht weiter eingegangen werden soll. Die Models schließlich werden als POJOs (Plain Old Java Objects) bereitgestellt. Alle Anwendungs-Requests werden vom Faces Servlet behandelt. Aufgrund des Status der Anwendung wird entschieden, welche Beans und Views verwendet werden sollen. Die Antwort wird schließlich von einer View erzeugt und im Browser angezeigt.

JavaServer Faces (JSF) wurde nicht zuletzt als Technologie entwickelt, um die vielfältigen Ansätze zur Entwicklung von Webanwendungen unter Java zu standardisieren. Die Spezifikation dient dazu, den Entwickler in folgenden Bereichen zu unterstützen:

- Komponenten: JSF erlaubt es, vollständige Webanwendungen in einfacher Form aus Komponenten aufzubauen. Darüber hinaus kann man Komponenten selbst erstellen und beliebig wiederverwenden.
- Datentransfer: JSF macht es sehr einfach möglich, Daten von der Applikation in die Benutzerschnittstelle (und wieder zurück) zu transferieren.
- Zustandsspeicherung: JSF ermöglicht die automatische Speicherung des Zustands der Applikation am Server oder am Client.
- Ereignisbehandlung: Vom Benutzer am Client generierte Ereignisse können am Server behandelt werden. Dazu werden Ereignisbehandlungsmethoden mit den einzelnen Komponenten verknüpft.

Durch die strikte Trennung der Schichten der Applikation im Sinne der MVC-Architektur können die einzelnen an der Applikation beteiligten Personen (z.B. Webdesigner, Komponentenentwickler und Applikationsentwickler) unabhängig voneinander arbeiten.

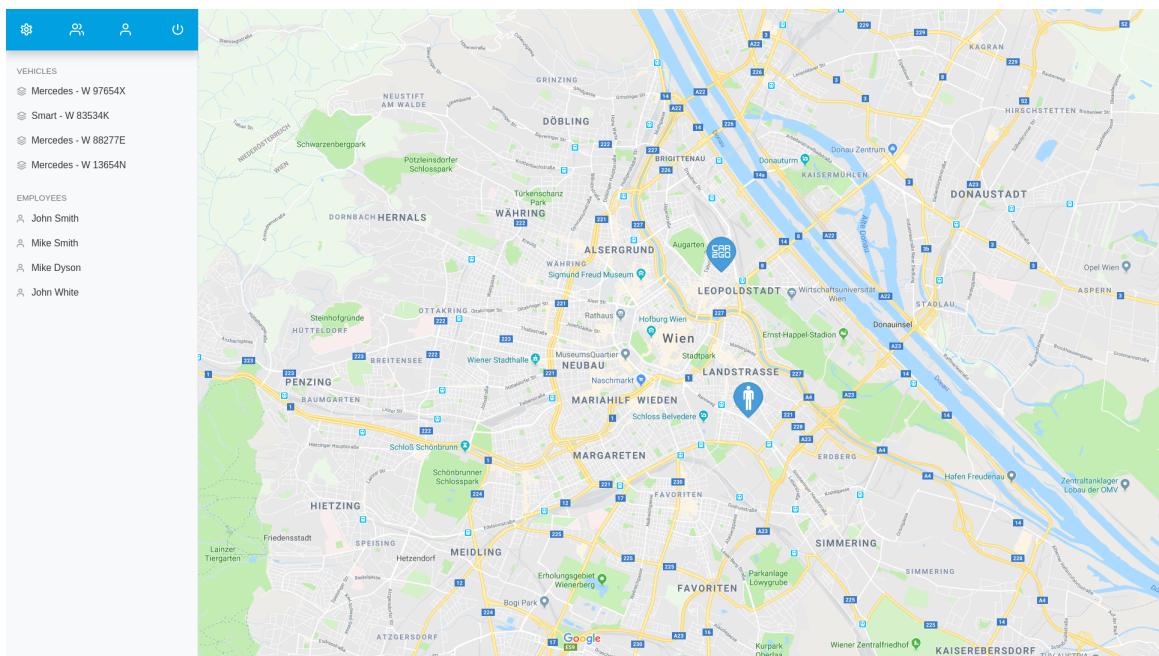
# **Kapitel 11**

## **Auswahl einer geeigneten Technologie**

In diesem Kapitel werden die größten Anforderungen des Projekts in allen drei Technologien implementiert, bewertet und danach wird ein Fazit gezogen, welche Technologie für das Projekt die geeignetste ist.

### **11.1 Anforderungen**

Die Hauptfunktionalität der Webseite besteht darin, dem Benutzer auf einer Karte die Positionsdaten von Autos und Mitarbeitern in Form von Markern anzuzeigen. Weiters werden diese Daten auch in zwei Tabellen aufgelistet und es könnte wie folgt aussehen:

Abbildung 11.1: *Hauptseite*

## 11.2 Implementierung der Karte mit JSF

Die Verwendung dieser Komponente wird in der Sektion 16.1 beschrieben.

## 11.3 Implementierung der Karte mit JavaScript

Das folgende Beispiel zeigt die Implementierung einer Karte durch das Google Maps JavaScript-API.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Simple Map</title>
5   <meta name="viewport" content="initial-scale=1.0" />
6   <meta charset="utf-8" />
7   <style>
8     /* Man sollte die Hoehe der Map immer explizit definieren,
9      * um die groesse des div Elements, welches die Map
10     * beinhaltet zu setzen */
11    #map {
12      height: 100%;
13    }
14    /* Optional */
15    html,
16    body {
17      height: 100%;
18      margin: 0;
19      padding: 0;
20    }
21  </style>
22 </head>
23 <body>
24   <div id="map"></div>
25   <script>
26     var map;
27     function initMap() {
28       map = new google.maps.Map(document.getElementById('map'), {
29         center: { lat: 48.206, lng: 16.358 },
30         zoom: 15
31       });
32     }
33   </script>
34   <script src="https://maps.googleapis.com/maps/api/js?
35           key=API_KEY&callback=initMap" async defer></script>
36 </body>
37 </html>
```

Listing 11.1: Implementation einer Karte mit JS

Dieser Code erzeugt eine Karte die wie folgt aussieht:

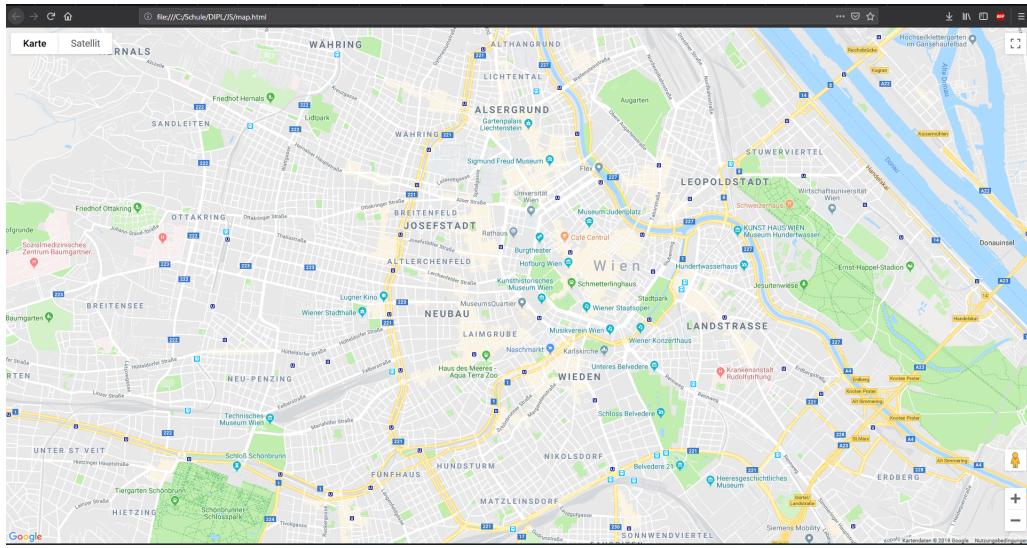


Abbildung 11.2: Einfache Implementierung einer Karte

In diesem Beispiel gibt es einige Anmerkungen zu machen:

1. Die Applikation wird durch den `<!DOCTYPE html>` Tag als HTML5 definiert.
2. Es wird ein `div` Element erzeugt, welches die Map enthält.
3. Es wird eine JavaScript Funktion definiert, welche die Map in dem `div` Element erstellt.
4. Es wird das JavaScript-API durch den `script` Tag geladen.

Diese Schritte werden nun etwas genauer erklärt.

### 11.3.1 Die Applikation als HTML5 definieren

Es wird empfohlen, einen `doctype` bei Webapplikationen zu setzen. Der Grund dafür ist, dass Internetbrowser den Inhalt, der mit diesem `doctype` definiert ist, im sogenannten `standards mode` rendern. Das heißt, dass die Applikation mit verschiedenen Browsern kompatibel sein sollte. Falls ein Browser diesen `doctype` nicht kennt, ignoriert er diesen.

### 11.3.2 Map DOM Elemente

Um die Karte auf einer Internetseite anzuzeigen, muss man dafür einen Platz reservieren. Dies geschieht, indem man ein div Element erzeugt und eine Referenz des Elements im DOM<sup>1</sup> des Browsers erhält.

Im obigen Beispiel wurde CSS verwendet, um die Höhe des div Elements auf 100% zu setzen. Dies wurde getan, weil divs für den Wert ihrer Höhe den Wert übernehmen, welches das in ihnen befindende Element besitzt und leere divs besitzen im Normalfall die Höhe 0. Aus diesem Grund muss man die Höhe eines <div> immer explizit definieren.

### 11.3.3 Das Map Objekt

Die JavaScript Klasse, welche eine Karte repräsentiert, ist die Klasse Map. Objekte dieser Klasse definieren eine einzelne Karte auf einer Seite. Es ist möglich mehrere Instanzen dieser Klasse zu erzeugen. Dabei wird jedes Objekt eine separate Karte auf der Seite erzeugen.

Wenn man eine neue Map Instanz erzeugt, muss man ein <div> Element als Container für die Karte definieren. Eine Referenz auf dieses Element erhält man über die `document.getElementById()` Methode.

### 11.3.4 Optionen des Map Elements

Es gibt zwei erforderliche Optionen für jede Karte: `center` und `zoom`.

#### 11.3.5 Center

Das `center` Attribut gibt den Punkt an, auf den die Karte beim Initialisieren zentriert ist.

---

<sup>1</sup>Document Object Model

## Zoomlevel

Die Standardauflösung in welcher die Karte angezeigt wird, wird durch das `zoom` Attribut festgelegt. Eine Karte, wo der `zoom` auf 0 gestellt ist, entspricht einer Karte der Erde die vollständig herausgezoomt ist. Um eine Karte der gesamten Erde als einzelnes Bild anzuzeigen, würde man entweder eine riesige Karte oder eine kleine Karte mit niedriger Auflösung benötigen. Aus diesem Grund sind Karten auf Google Maps und im Maps JavaScript-API in Kacheln und Zoomlevel aufgeteilt. Bei einem niedrigen Zoomlevel decken wenige Kacheln einen Großteil der Karte ab, während bei einem höherem Zoomlevel die Auflösung dieser Kacheln besser ist und sie eine kleinere Fläche der Karte abdecken. Die folgende Liste zeigt das ungefähre Detailniveau, welches man bei jedem Zommlevel erwarten kann:

- 1: Welt
- 5: Kontinente
- 10: Stadt
- 15: Straßen
- 20: Gebäude

### 11.3.6 Laden des Maps JavaScript-API

Um das Maps JavaScript-API zu laden, muss ein `script` Tag wie in obigem Beispiel verwendet werden. In diesem Tag muss ein API Key angegeben werden, den man bei Google erwerben kann. In dieser Arbeit wurde der verwendete Key immer durch den Text `API_KEY` ersetzt. Die URL in diesem Tag ist der Ort, an dem die JavaScript Datei liegt, welche alle Symbole und Definitionen, die zur Verwendung des Maps JavaScript-APIs benötigt werden, lädt. Dieser `script` Tag muss vorhanden sein. Das `async` Attribut lässt zu, dass der Browser den Rest der Seite rendernt, während das Maps JavaScript-API lädt. Wenn das API bereit ist, führt es die Methode aus, die im `callback` Parameter definiert ist.

### 11.3.7 Erzeugen von Custom Markern

Ein Marker identifiziert einen Standort auf einer Karte. Das Aussehen von Markern kann verändert werden, und in diesem Fall werden sie als `icons` bezeichnet. Man

kann ein eigenes `icon` im Konstruktor des Markers setzen, oder durch einen Aufruf der `setIcon()` Methode.

Der `google.maps.Marker` Konstruktor bekommt ein einzelnes Marker `options` Objekt, welches die Eigenschaften des Markers enthält.

Die folgenden Attribute sind insbesondere wichtig und bei der Erzeugung von Markern oft gesetzt:

- `position` (erforderlich) definiert die Position des Markers
- `map` (optional) spezifiziert die Map auf welcher der Marker platziert wird. Falls bei der Erzeugung eines Markers keine Map spezifiziert ist, wird der Marker erstellt, jedoch nicht auf einer Karte angezeigt. Man kann den Marker dann mit der `setMap()` Methode auf einer Karte anzeigen.

Das folgende Beispiel zeigt, wie Custom Marker auf einer Karte angezeigt werden können:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Simple Map</title>
6   <meta name="viewport" content="initial-scale=1.0" />
7   <meta charset="utf-8" />
8   <style>
9     /* Man sollte die Hoehe der Map immer explizit definieren,
10      * um die groesse des div Elements, welches die Map
11      * beinhaltet zu setzen */
12   #map {
13     height: 100%;
14
15   }
16
17   /* Optional */
18   html,
19   body {
20     height: 100%;
21     margin: 0;
22     padding: 0;
23   }
24   </style>
25 </head>
26
27
28 <body>
29   <div id="map"></div>
```

```
30 <script>
31
32     function initMap() {
33         var myLatLng = { lat: 48.206, lng: 16.358 };
34
35         var locations = [
36             ['Auto 1', 48.203, 16.358],
37             ['Auto 2', 48.200, 16.335],
38             ['Auto 3', 48.209, 16.340],
39             ['Auto 4', 48.206, 16.365],
40             ['Auto 5', 48.204, 16.360]
41         ];
42
43         var map = new google.maps.Map(document.getElementById('map'), {
44             zoom: 15,
45             center: myLatLng
46         });
47
48         var marker, i;
49         var icon = {
50             url: 'http://chittagongit.com//images/google-maps-bus-icon/google-
51                 maps-bus-icon-14.jpg', //url des Bildes welches als Marker
52                 verwendet wird
53             scaledSize: new google.maps.Size(50,50) // scaled die groesse des
54                 Markers auf 50x50
55         };
56
57         for (i = 0; i < locations.length; i++) {
58             marker = new google.maps.Marker({
59                 position: new google.maps.LatLng(locations[i][1], locations[i]
60                     [2]),
61                 map: map,
62                 icon: icon,
63                 title: locations[i][0]
64             });
65         }
66     }
67 </script>
68
69 <script src="https://maps.googleapis.com/maps/api/js?key=
70     AIzaSyCktxoF4cOvmITlfJi3g_VvzZT50WNc5yc&callback=initMap"
71     async defer></script>
72 </body>
73
74 </html>
```

Listing 11.2: Implementation einer Karte mit Custom Markern

Dieser Code erzeugt die folgende Karte:

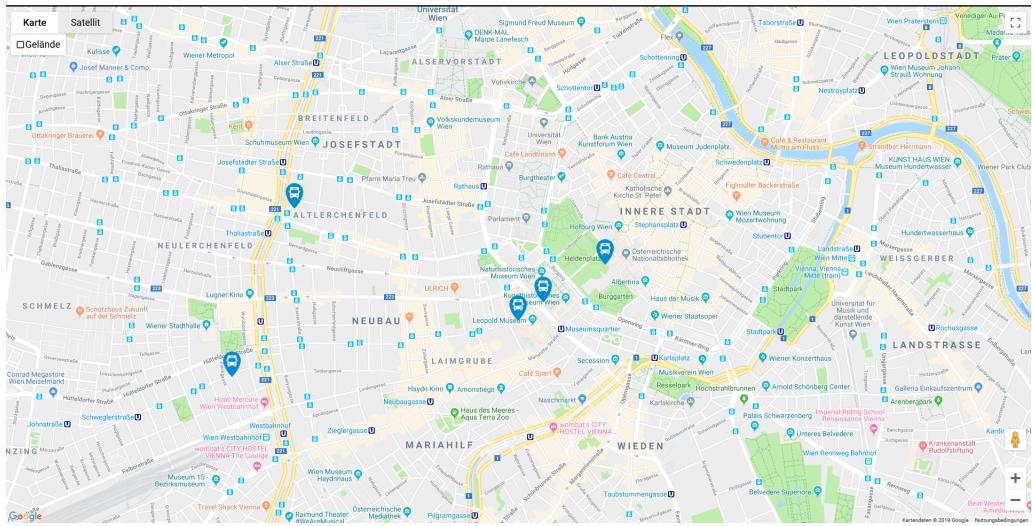


Abbildung 11.3: Karte mit Custom Marker

In obigem Beispiel wird zunächst ein Array definiert, welches fünf Autos beinhaltet. Danach wird ein `icon` Objekt definiert, wo durch das URL Attribut das Aussehen des Markers verändert und anschließend durch das `scaledsize` Attribut die Größe des Markers angepasst wird. Darauf folgt eine Schleife, welche das Array abarbeitet und die Marker auf der Karte platziert.

### 11.3.8 Bewertung

Google hat mit dem Maps JavaScript API ein API erstellt, welches erstaunlich gut ist. Die Dokumentation ist sehr gut, es ist einfach zu benutzen, der Code ist in den meisten Fällen selbsterklärend und Fehler können mit Hilfe des Internets sehr schnell gelöst werden.

## 11.4 Implementierung einer Karte mit PHP

Dieser Abschnitt zeigt, wie man Informationen aus einer MySQL Datenbank liest und auf einer Google Maps Karte, mit Hilfe des Maps JavaScript APIs, anzeigt. Die Karte erhält die benötigten Informationen durch eine XML Datei, welche zwischen der Datenbank und der Karte liegt. Es werden dabei PHP Statements verwendet, um die

Informationen der Marker aus der Datenbank zu lesen und in die XML Datei zu schreiben.

### 11.4.1 Erstellen der Datenbank und der XML Datei

Es wurde eine Datenbank erstellt, welche die Attribute `id`, `name`, `lat` und `long` der Marker speichert.

Es gibt drei verschiedene Arten, um SQL Daten mit Hilfe von PHP als XML auszugeben:

- Die DOM XML Methoden von PHP benutzen um XML auszugeben
- Die DOM Methoden von PHP benutzen um XML auszugeben
- PHPs echo verwenden, um XML auszugeben

Im folgenden Beispiel wurde die `echo` Methode verwendet um die XML Datei zu generieren. Wenn man nur die `echo` Methode verwendet, sollte man auch eine Hilfsfunktion, wie die `parseToXML` verwenden, um einige Sonderzeichen korrekt zu encoden um dadurch XML freundlich zu sein.

Die folgende PHP Datei öffnet eine Verbindung zu einer MySQL Datenbank und gibt den XML Output im Browser aus.

```
1 <?php
2 require("phpsqlajax_dbinfo.php");
3
4 function parseToXML($htmlStr)
5 {
6 $xmlStr=str_replace('<', '&lt;', $htmlStr);
7 $xmlStr=str_replace('>', '&gt;', $xmlStr);
8 $xmlStr=str_replace('"', '&quot;', $xmlStr);
9 $xmlStr=str_replace("'", '&#39;', $xmlStr);
10 $xmlStr=str_replace("&", '&amp;', $xmlStr);
11 return $xmlStr;
12 }
13
14
15 // Öffnet eine Verbindung zu einem MySQL server
16 $connection=mysqli_connect ('localhost', $username, $password);
17 if (!$connection) {
18 die('Not connected : ' . mysqli_error());
19 }
```

```

21 // Setzt die aktive MySQL Datenbank
22 $db_selected = mysqli_select_db($database, $connection);
23 if (!$db_selected) {
24     die ('Can\'t use db : ' . mysqli_error());
25 }
26
27
28 // Selektiert alle Zeilen der Tabelle
29 $query = "SELECT * FROM markers WHERE 1";
30 $result = mysqli_query($query);
31 if (!$result) {
32     die('Invalid query: ' . mysqli_error());
33 }
34
35 header("Content-type: text/xml");
36
37 // Beginnen die XML Datei zu schreiben
38 echo "<?xml version='1.0' ?>";
39 echo '<markers>';
40 $ind=0;
41 // Durch die Zeilen iterieren und einen Knoten fuer jede Zeile erzeugen
42 while ($row = @mysqli_fetch_assoc($result)){
43     echo '<marker >';
44     echo 'id="' . $row['id'] . '" ';
45     echo 'name="' . parseToXML($row['name']) . '" ';
46     echo 'lat="' . $row['lat'] . '" ';
47     echo 'lng="' . $row['lng'] . '" ';
48     echo '/>';
49     $ind = $ind + 1;
50 }
51 // Ende der XML Datei
52 echo '</markers>';
53 ?>
```

Listing 11.3: Datenbankverbindung mit PHP

Der Code im obigen Beispiel, verbindet sich mit der Datenbank und führt den SELECT \* Befehl auf. Danach wird über das Ergebnis iteriert und für jeden Marker ein Knoten erzeugt. Durch das iterieren, wird der Wert des name Attributs an die `parseToXML` Methode geschickt welche überprüft, ob im String Sonderzeichen vorhanden sind. Das Skript endet indem es den schließenden `markers` Tag setzt.

Um sicherzugehen, dass das PHP Skript eine richtige XML Datei liefert, sollte man sich den Output dieser Datei anschauen.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <markers>
3     <marker id="1" name="Auto 1" lat="48.203" lng="16.358"/>
4     <marker id="2" name="Auto 2" lat="48.200" lng="16.340"/>
5     <marker id="3" name="Auto 3" lat="48.206" lng="16.363"/>
```

```

6 <marker id="5" name="Auto 5" lat="48.213" lng="16.352"/>
7 <marker id="4" name="Auto 4" lat="48.210" lng="16.355"/>
8 </markers>
```

Listing 11.4: Ausgabe der XML Datei

Falls der Browser die Daten nicht anzeigt, und man das Problem debuggen will, sollte man die Zeile, wo der Header auf `content-type: text/xml` gesetzt wird entfernen, da dadurch der Browser versucht XML auszugeben und es dem Programmierer schwer macht die debugging Nachrichten zu sehen.

### 11.4.2 Erstellen der Karte

In diesem Abschnitt wird gezeigt, wie mit JavaScript und der XML Datei eine Karte erzeugt wird. Um eine XML Datei zu laden, kann man sich das `XMLHttpRequest` Objekt zur Hilfe nehmen, welches vom Browser zur Verfügung gestellt wird. Dieses Objekt erlaubt es, eine Datei abzurufen, welche auf der selben Domäne liegt wie die anfragende Seite und basiert auf AJAX.

Zunächst definiert man eine Methode um die Datei zu laden. Die Funktion erhält zwei Parameter:

1. `url` spezifiziert den Pfad zu einer XML Datei oder einem PHP Skript welches die Datei generiert, falls man möchte, dass die XML Datei dynamisch aktualisiert wird, wenn sich die Daten ändern. In diesem Beispiel wird eine statische XML Datei aufgerufen.
2. `callback` gibt die Methode an, welche das Skript aufruft, wenn das XML wieder zu JavaSkript zurückkehrt.

Der nachfolgende Code enthält die Deklaration dieser Methode.

```

1 function downloadUrl(url,callback) {
2     var request = window.ActiveXObject ?
3         new ActiveXObject('Microsoft.XMLHTTP') :
4         new XMLHttpRequest();
5
6     request.onreadystatechange = function() {
7         if (request.readyState == 4) {
8             request.onreadystatechange = doNothing;
9             callback(request, request.status);
10        }
11    };
12}
```

```

12
13 request.open('GET', url, true);
14 request.send(null);
15 }
```

Listing 11.5: Deklaration der downloadURL Methode

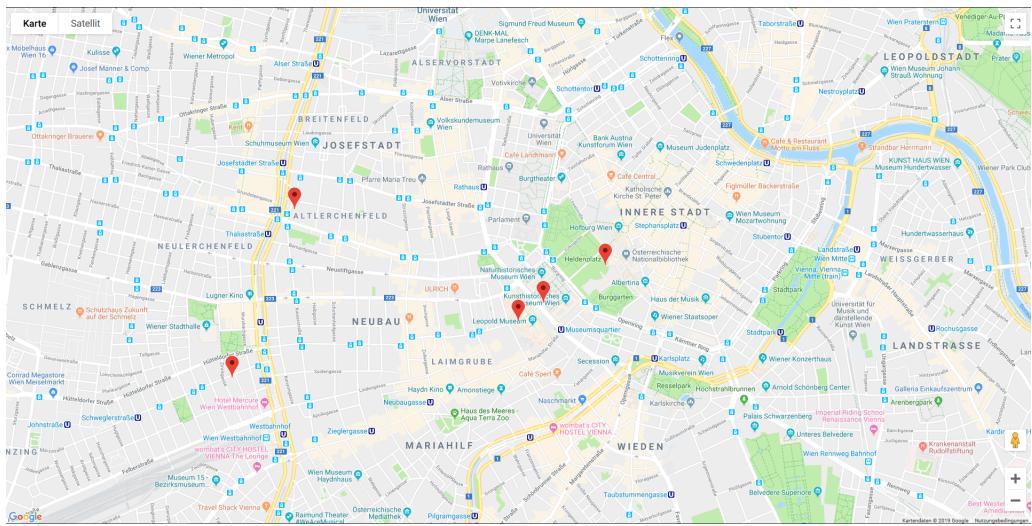
Nun kann die Methode aufgerufen werden. In diesem Beispiel wird eine Statische XML Datei verwendet um die Daten der Marker zu laden.

```

1 // Hier den Namen der XML bzw PHP Datei angeben
2 downloadUrl('marker.xml', function(data) {
3     var xml = data.responseXML;
4     var markers = xml.documentElement.getElementsByTagName('marker'
5         );
6     Array.prototype.forEach.call(markers, function(markerElem) {
7         var id = markerElem.getAttribute('id');
8         var name = markerElem.getAttribute('name');
9         var point = new google.maps.LatLng(
10             parseFloat(markerElem.getAttribute('lat')),
11             parseFloat(markerElem.getAttribute('lng'))));
12
13         var infowincontent = document.createElement('div');
14         var strong = document.createElement('strong');
15         strong.textContent = name
16         infowincontent.appendChild(strong);
17         infowincontent.appendChild(document.createElement('br'));
18
19         var marker = new google.maps.Marker({
20             map: map,
21             position: point,
22         });
23     });
});
```

Listing 11.6: Laden der Marker

Dieser Code erzeugt eine Karte die wie folgt aussieht:

Abbildung 11.4: *phpMap*-Implementierung einer Karte mit PHP

Das Aussehen der Marker kann wie bereits im JavaScript Beispiel beschrieben geändert werden.

### 11.4.3 Bewertung

Die Implementierung dieser Karte beinhaltet nur ein PHP Skript und basiert sonst auf dem Maps JavaScript API. Dadurch trifft die Bewertung die bereits etwas weiter oben genannt wurde auch hier zu. Dennoch kann es sein, dass diese Variante langsamer ist, wie wenn sie nur mit JavaScript implementiert wird. Der Grund dafür ist, dass der XMLHttpRequest asynchron ist und die callback Methode die downloadURL Methode basierend auf der Größe der XML Datei ausführt. Das heißt, je größer die XML Datei ist, umso länger kann es dauern, bis alles geladen hat.

## 11.5 Anzeigen von Daten in Form von Tabellen mit JSF

In diesem Abschnitt wird mit der DataTable-Komponente von PrimeFaces eine Tabelle erzeugt und mit Daten gefüllt.

Das Projekt besteht einerseits aus der `index.xhtml` Datei, welche die DataTable-Komponente beinhaltet und andererseits aus der `carService.java` Datei, welche die Programmlogik beinhaltet

## Die index.xhtml Datei

```

1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
   w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
4    xmlns:h="http://xmlns.jcp.org/jsf/html"
5    xmlns:p="http://primefaces.org/ui">
6    <h:head>
7      <title>Facelet Title</title>
8    </h:head>
9    <h:body>
10      <p:dataTable var="car" value="#{carService.createCars()}">
11        <p:column headerText="Id">
12          <h:outputText value="#{car.id}" />
13        </p:column>
14
15        <p:column headerText="Brand">
16          <h:outputText value="#{car.brand}" />
17        </p:column>
18
19        <p:column headerText="Color">
20          <h:outputText value="#{car.color}" />
21        </p:column>
22
23        <p:column headerText="Lat">
24          <h:outputText value="#{car.lat}" />
25        </p:column>
26
27        <p:column headerText="Lng">
28          <h:outputText value="#{car.lng}" />
29        </p:column>
30      </p:dataTable>
31    </h:body>
32  </html>

```

Listing 11.7: datatable-bsp-index.xhtml

Um PrimeFaces verwenden zu können muss es über einen XML-Namespace importiert werden. Dies geschieht in Zeile 4. Das Einbinden der DataTable-Komponente erfolgt in Zeile 11.

Der DataTable hat in diesem Beispiel zwei Attribute, `var` und `values`. `var` ist der Name der Variable, welche verwendet wird um auf die einzelnen Daten zu verweisen. `value` gibt die Datenquelle an, welche in diesem Fall eine Liste ist, die aus der Klasse CarService geladen wird.

Id	Brand	Color	Lat	Lng
1	Mercedes	Orange	48.209	16.358
2	Jaguar	White	28.209	16.987
3	Honda	Blue	90.239	116.218
4	Renault	White	0.509	128.358
5	Fiat	Silver	148.798	-16.358

Abbildung 11.5: JSF-DataTable-Beispiel - Tabelle mit JSF

Das oben dargestellte Bild zeigt, wie der DataTable nach der Implementation aussieht.

### 11.5.1 Bewertung

PrimeFaces bietet mit dem DataTable eine Komponente an, die noch viel mehr kann, als in obigem Beispiel gezeigt. Die Dokumentation ist gut, die meisten Attribute sind ausreichend erklärt und die Komponente lässt sich leicht verwenden. Ein Nachteil ist, dass PrimeFaces teilweise verbuggt sind und die Fehlermeldungen (falls welche vorhanden sind) dem Nutzer nicht in jedem Fall weiterhelfen. Dennoch lassen sich die meisten Probleme durch etwas Recherche im Internet relativ schnell lösen.

## 11.6 Anzeigen von Daten in Form von Tabellen mit JavaScript

Die folgende Sektion zeigt anhand eines einfachen Beispieles, wie Tabellen in JavaScript erzeugt werden können.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Tables</title>
6 </head>
7
8 <body>
9   <script>
10  var body = document.getElementsByTagName("body")[0];
11
12  //Definieren der Tabelle
13  var ourTable = document.createElement("table");
14  var ourTableBody = document.createElement("tbody");
15
16  //Schleife, welche die Tabelle erzeugt
17  for (var i = 0; i < 2; i++) {
18    //Erzeugen des tr tags

```

```

19      var row = document.createElement("tr");
20
21      //Erzeugen des td tags und dem Inhalt der Zellen
22      for (var j = 0; j < 2; j++) {
23          var cell = document.createElement("td");
24          var cellText = document.createTextNode("Zelle in Zeile" + i
25              + ", Spalte " + j);
26          cell.appendChild(cellText);
27          row.appendChild(cell);
28      }
29      ourTableBody.appendChild(row);
30
31  ourTable.appendChild(ourTableBody);
32
33  body.appendChild(ourTable);
34  // Den Rand der Tabelle definieren
35  ourTable.setAttribute("border", "2");
36
37  function set_border() {
38
39      myBody = document.getElementsByTagName("body")[0];
40
41      myBodyElements = myBody.getElementsByTagName("table");
42
43      myP = myBodyElements[1];
44      myP.style.border = "10px";
45  }
46  </script>
47</body>
48</html>

```

Listing 11.8: javascript-databatable.html

In obigem Beispiel wird zunächst die Tabelle definiert, danach werden in einer Schleife eine Zeile und in einer weiter Schleife zwei Spalten und der Zelleninhalt erstellt. Danach wird noch ein Rand für die Tabelle erzeugt. Dieser Code generiert eine Tabelle die wie folgt aussieht.

Zelle in Zeile0, Spalte 0	Zelle in Zeile0, Spalte 1
Zelle in Zeile1, Spalte 0	Zelle in Zeile1, Spalte 1

Abbildung 11.6: jsTable-Beispiel - Tabelle in JavaScript

### 11.6.1 Bewertung

Die Tabelle in JavaScript ist eine Komponente die leicht zu verwenden ist, sobald man weiß, wie sie richtig verwendet wird. Da die Dokumentation zu wünschen lässt, fällt es Anfängern oftmals schwer die Komponente zu verwenden, weil auch die Beispiele die es im Internet zu finden gibt oftmals schon fortgeschritten sind und teilweise nicht funktionieren.

## 11.7 Anzeigen von Daten in Form von Tabellen mit PHP

In dieser Sektion wird anhand eines einfachen Beispieles gezeigt, wie man mit einem PHP Skript Daten von einer Datenbank ließt und in einfacher Tabellenform anzeigt.

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $servername = "localhost";
7 $username = "username";
8 $password = "password";
9 $dbname = "myDB";
10
11 // Verbindung aufbauen
12 $conn = new mysqli($servername, $username, $password, $dbname);
13 // Verbindung ueberpruefen
14 if ($conn->connect_error) {
15     die("Connection failed: " . $conn->connect_error);
16 }
17
18 // Daten beschaffen
19 $sql = "SELECT id, firstname, lastname FROM MyGuests";
20 $result = $conn->query($sql);
21
22 // Daten pruefen
23 if ($result->num_rows > 0) {
24     // Jede Zeile ausgeben
25     while($row = $result->fetch_assoc()) {
26         echo "<br> id: " . $row["id"] . " - Name: " . $row["firstname"] . " " .
27             $row["lastname"] . "<br>";
28     }
29 } else {
30     echo "0 results";
31 }
32 // Verbindung schliessen
33 $conn->close();
```

```
33  ?>
34
35  </body>
36  </html>
```

Listing 11.9: php-database.php

In obigem Beispiel wird zunächst in Zeile 12 eine Verbindung zur Datenbank aufgebaut. In Zeile 19 werden die Daten durch das **SELECT** Statement von der Datenbank gelesen. In Zeile 23 wird überprüft ob das Statement Werte zurückgeliefert hat und anschließend wird in einer Schleife jede Zeile einzeln ausgegeben und danach wird in Zeile 32 die Verbindung zur Datenbank getrennt.

```
id: 1 - Name: John Doe
id: 2 - Name: Mary Moe
id: 3 - Name: Julie Dooley
```

Abbildung 11.7: *phpTable*-Beispiel - Tabelle in PHP

### 11.7.1 Bewertung

Daten mit PHP in einer Tabelle darstellen mag für den Laien anfangs etwas mühsam sein, doch sobald man sich ein bisschen damit beschäftigt, hat man das Konzept relativ schnell verstanden und kann ohne größere Probleme arbeiten. Die Dokumentation ist zwar nicht schlecht, doch kommt nicht an die PrimeFaces Dokumentation ran.

## 11.8 Finale Auswahl

Die Implementierung der Webseite im Projekt wird zum größten Teil in Java geschehen. Der Grund dafür ist, dass zumal die Diplomanden mit Java die größten Erfahrungen haben und dadurch erwarten, dass bei der Implementierung keine großen Schwierigkeiten auftreten. Weiters ist die PrimeFaces DataTable-Komponente ein sehr gutes Werkzeug, welches sich leicht implementieren und um die erforderlichen Funktionalitäten im Projekt erweitern lässt. Bei der Karte wird jedoch auch etwas JavaScript verwendet, da die Implementierung der geforderten Funktionalitäten in Java nur sehr umständlich oder teilweise gar nicht umsetzbar sind.

# Kapitel 12

## Verwendung im Projekt

### 12.1 Login

Dadurch, dass die Webseite nur von Dispatchern und der Chefetage verwendet werden kann, musste ein Login-System implementiert werden.

#### 12.1.1 Die Oberfläche

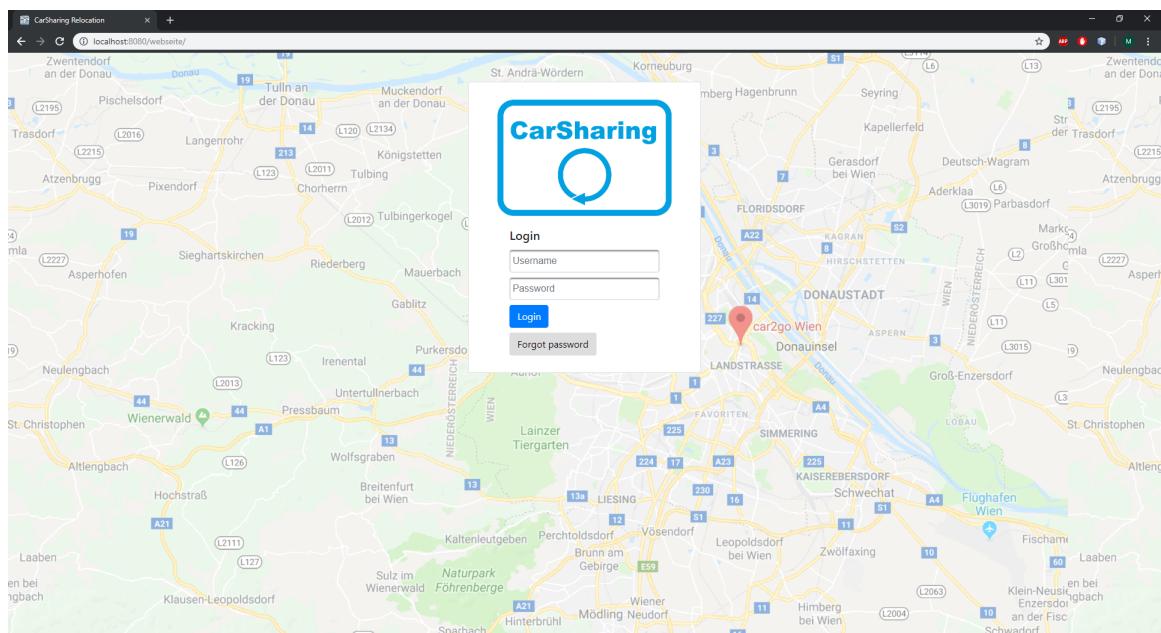


Abbildung 12.1: *login.xhtml* - Login der Webseite

Das oben dargestellte Bild zeigt die Login-Seite der Webseite. Sie besteht aus zwei Eingabekomponenten für die Eingabe von Benutzername und Passwort und zwei Buttons die zum Einloggen oder zum zurücksetzen des Passworts dienen.

Die Einbindung dieser Komponenten sind im folgenden Listing zu sehen.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html>
3 <html xmlns="http://www.w3.org/1999/xhtml"
4      xmlns:h="http://java.sun.com/jsf/html"
5      xmlns:f="http://java.sun.com/jsf/core"
6      xmlns:p="http://primefaces.org/ui">
7
8     <h:head>
9         <meta charset="utf-8"/>
10        <title>CarSharing Relocation</title>
11        <h:outputStylesheet library="css" name="bootstrap.min.css"/>
12        <h:outputStylesheet library="css" name="login_style.css"/>
13
14    </h:head>
15    <h:body>
16        <div class="container">
17            <div class="card px-5 mx-auto mt-5"
18                style="width: 40vh;">
19                
23            <div class="card-body">
24                <h5 class="card-title">Login</h5>
25                <h:form class="form mx-auto">
26                    <p:inputText id="loginText"
27                        class="form-control mb-2"
28                        placeholder="Username"
29                        value="#{login.username}"
30                        required="true"/>
31                    <p:inputText id="pwdText"
32                        class="form-control mb-2"
33                        type="password"
34                        placeholder="Password"
35                        value="#{login.pwd}"
36                        required="true"/>
37                    <span/>
38                    <h:commandButton class="btn btn-primary mb-2"
39                        type="submit"
40                        value="Login"
41                        action="#{login.checkLogin()}" />
42                </h:form>
43                <h:form>
44                    <h:commandButton class="btn mb-2"
45                        value="Forgot password"
```

```
46          </h:form>
47      </div>
48  </div>
49  </div>
50  </h:body>
51</html>
```

Listing 12.1: login.xhtml

## 12.1.2 Der Controller

Im folgenden Listing ist der Controller abgebildet.

```
1 package htlstp.bradaric.projekt_design_v1.controller;
2
3 //imports
4
5 /**
6 * 
7 * @author Matej
8 */
9 @Named("login")
10 @SessionScoped
11 public class Login implements Serializable {
12
13     private String username;
14     private String pwd;
15
16     @Inject
17     private UserController uc;
18
19     public String checkLogin() {
20         UserClient user = GenericService.getClient(UserClient.class, "REST_KEY");
21         try {
22             ;
23             if (user.checkLogin(username, pwd).execute().body()) {
24                 uc.setCurrentUser(user.getUser(username).execute().body());
25                 return "main.xhtml";
26             }
27         } catch (IOException ex) {
28             ex.printStackTrace();
29
30         } finally {
31             username = "";
32             pwd = "";
33     }
```

```
34     FacesContext.getCurrentInstance().addMessage(null, new FacesMessage
35         ("username or password wrong"));
36     return "";
37 }
38
39 public String logout() {
40     ((HttpSession) FacesContext.getCurrentInstance().getExternalContext
41         ().getSession(false)).invalidate();
42     return "index";
43 }
44
45 public String getUsername() {
46     return username;
47 }
48
49 public void setUsername(String username) {
50     this.username = username;
51 }
52
53 public String getPwd() {
54     return pwd;
55 }
56
57 public void setPwd(String pwd) {
58     this.pwd = pwd;
59 }
60 }
```

Listing 12.2: loginController.java

Im obigen Listing ist der Controller abgebildet. Dieser beinhaltet zwei Methoden die für das Login bzw. das Logout zuständig sind. In der `checkLogin()` Methode werden Userdaten über ein Webservice, welches im Projekt entwickelt wurde, geladen. Dieses Service ist mit einem Key geschützt und dieser ist im Listing durch den Text `REST_KEY` ersetzt worden. Danach werden die vom Benutzer eingegebenen Daten mit denen aus der Datenbank verglichen und im Erfolgsfall wird der Benutzer auf die Hauptseite weitergeleitet. In der `logout()` Methode wird die Session des Benutzers terminiert und er wird auf die Login-Seite weitergeleitet.

## 12.2 Die Hauptseite

Die Hauptseite besteht aus drei Teilen. Der Karte, zwei Tabellen und einer Navigationsleiste.

### 12.2.1 Karte

Die Implementation der Karte wird in der Sektion 16.2 erläutert.

### 12.2.2 Tabellen

Auf der Hauptseite befinden sich zwei Tabellen. Eine beinhaltet alle verfügbaren Autos (Die, die keinem Mitarbeiter zugeteilt sind) und die andere alle Mitarbeiter die zurzeit keinen Auftrag haben.

#### Die Oberfläche

VEHICLES	
1	2
3	4
5	6
7	8
 N-8065V	
 C-9504D	
 R-8644W	
 I-3206D	
 S-7898V	
 I-7291G	
 C-9204A	
 A-1252W	
 D-0322F	
 C-5787B	
 G-3973H	
 C-5753F	
 F-8255Z	
 M-6504B	
 J-1504Y	
EMPLOYEES	
 HEINZL	

Abbildung 12.2: *main.xhtml* - Tabellen auf der Hauptseite

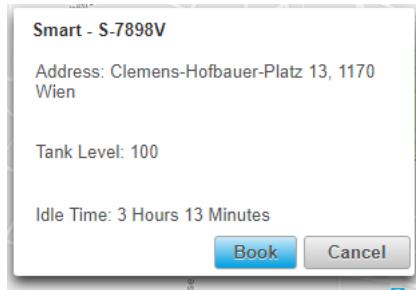
Das folgende Listing enthält den Code, zur Erstellung der Tabellen.

```
1 <h:body>
2     <h2 class="sidebar-heading">
3         <div>Vehicles</div>
4     </h2>
5
6     <div id="flex-vehicle">
7
8         <h:form id="vehicle-form">
9             <p:dataTable value="#{carCon.currentCars}" var="vehicle" id="table"
10                scrollable="true" scrollHeight="500"
11                paginator="true" rows="40" rowHover="true">
12                    <p:column id="table-column" headerText="Vehicles"
13                        toggleable="false">
14                        <p:commandLink action="#{carCon.setCurrentCar(
15                            vehicle)}"
16                            oncomplete="PF('dialogVehicle').show();">
17                            <h:graphicImage url="resources/images/
18                                car_black_small.png" />
19                            <p:commandLink value="#{vehicle.car.plateNumber}
20                                " styleClass="vehicleName"
21                                action="#{carCon.setCurrentCar(vehicle)}"
22                                oncomplete="PF('dialogVehicle').show();"
23                                >
24                                <f:ajax render="dlgV" />
25                            </p:commandLink>
26                            <f:ajax render="dlgV" />
27                        </p:commandLink>
28                    </p:column>
29                </p:dataTable>
30            </h:form>
31        </div>
32        <h2 class="sidebar-heading">
33            <div>employees</div>
34        </h2>
35        <div id="flex-employees">
36            <h:form id="employee-form">
37                <p:dataTable value="#{econ.employees}" var="employee" id="table">
38                    <p:column>
39                        <p:commandLink>
40                            <h:graphicImage url="resources/images/person-
41                                dark.svg" class="bild" />
42                            <p:commandLink value="#{employee.vuser.nachname}
43                                " oncomplete="PF('dialogEmployee').show();"
44                                action="#{econ.setCurrEmployee(employee)}">
45                                <f:ajax render="dlgE" />
46                            </p:commandLink>
47                        </p:commandLink>
48                    </p:column>
49                </p:dataTable>
50            </h:form>
```

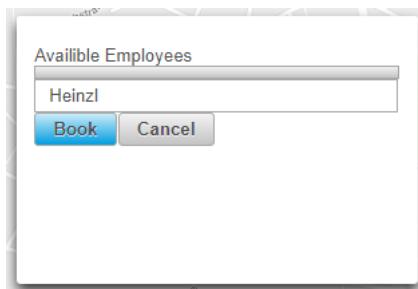
```
42 |     </div>
43 |     </div>
44 | </h:body>
```

Listing 12.3: main.xhtml

Wenn nun auf ein Auto in der Liste geklickt wird, geschieht das selbe, wie wenn der Benutzer auf einen Marker auf der Karte klickt. Es wird ein Dialog geöffnet, welcher genauere Daten über das Fahrzeug anzeigt.

Abbildung 12.3: *Detailansicht eines Autos*

Durch Klicken des ?Book? Buttons wird dieses Auto ausgewählt und es wird ein weiterer Dialog geöffnet, indem die verfügbaren Mitarbeiter in einer Liste dargestellt werden. Aus dieser Liste kann der Dispatcher einen Mitarbeiter auswählen und durch Klicken des ?Book? Buttons einem Auto zuteilen.

Abbildung 12.4: *Verfügbare Mitarbeiter*

Danach kann der Dispatcher durch einen Klick auf der Karte den neuen Standort des Autos auswählen und durch Klicken des ?Add? Buttons wird ein neuer Job erzeugt, den der zugeteilte Mitarbeiter sofort auf seinem Smartphone sieht.

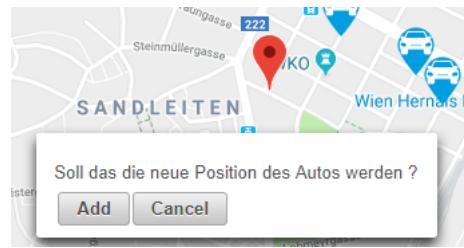


Abbildung 12.5: Neuen Standort auswählen

Das folgende Listing beinhaltet die Definition eines dieser Dialoge.

```

1<!--
2    Dialog, wenn man auf ein Auto klickt.
3-->
4
5<p:dialog id="dlgV" header="#{carCon.currentCar.car.model} - #{carCon.currentCar.car.plateNumber}" widgetVar="dialogVehicle"
6    dynamic="true" class="dialogClass" resizable="false">
7    <h:panelGrid columns="3">
8        <h:outputLabel value="Address: #{carCon.currentCar.address}" />
9        <p>
10       <h:outputLabel value="Tank Level: #{carCon.currentCar.fuelLevel}" />
11       <p>
12       <h:outputLabel value="Idle Time: #{carCon.currentCar.hourDifference} Hours #{carCon.currentCar.minuteDifference} Minutes"/>
13    </h:panelGrid>
14    <p:commandButton id="bookBtn" value="Book" type="Button"
15        styleClass="BtnBook" onclick="PF('dialogVehicle').hide(); PF('aCarDialog').show(); #{carCon.setCurrentCar(carCon.currentCar)}"/>
16    <p:commandButton id="cancelBtn" value="Cancel" type="Button"
17        styleClass="BtnCancel" onclick="PF('dialogVehicle').hide();"/>
18</p:dialog>
19<p:resizable for="dlgV" minHeight="200" minWidth="400" maxHeight="200" maxWidth="400"/>
```

Listing 12.4: main.xhtml

### 12.2.3 Navigationsleiste

Die Navigationsleiste enthält vier Menüpunkte.

Abbildung 12.6: *Navigationsleiste*

- Der erste leitet den Benutzer auf die Hauptseite weiter.
- Der zweite öffnet die Benutzerverwaltung.
- Der dritte öffnet die Benutzerdatenpflege.
- Der vierte terminiert die aktive Session, loggt den Benutzer aus und leitet ihn auf die Login-Seite weiter.

## 12.3 Benutzerverwaltung

Die Benutzerverwaltung besteht aus einer Tabelle, welche alle Benutzer auflistet und einem Button, um neue Mitarbeiter hinzuzufügen.

### User Management

#	First	Last	Email	Type	Edit	Delete
1	Lukas	Heinzel	l.heinzel.stp@gmail.com	Relocator		
2	Max	Suchy	max@privatevoid.net	Dispatcher		
3	Oliver	Hovorka	o.hovorka@htlst.p.at	Viewer		

Abbildung 12.7: *Benutzerverwaltung*

Durch Klicken des Edit Buttons können Name, E-Mail und Typ des betreffenden Benutzers geändert werden.

## User Management

#	First	Last	Email	Type	Edit	Delete
1	Lukas	Heinzl	l.heinzl.stp@gmail.com	Relocator		
2	Max	Suchy	max@privatevoid.net	Relocator ▾		
3	Oliver	Hovorka	o.hovorka@htlstp.at	Viewer		

[Add User](#)

Abbildung 12.8: Benutzerverwaltung Editieren eines Benutzers

Durch bestätigen werden die neuen Daten in die Datenbank geschrieben. Wird der Button ?Delete? gedrückt, öffnet sich ein Dialog, indem der User das Löschen eines anderen Benutzers bestätigen muss.

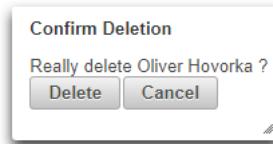


Abbildung 12.9: Benutzerverwaltung Löschen eines Benutzers

Durch Klicken des ?Delete? Buttons wird in der Datenbank ein Flag gesetzt, dass der User gelöscht wurde.

Durch Klicken des ?Add User? Buttons, der sich unter der Tabelle befindet, wird ein Dialog geöffnet, indem der Name, die E-Mail-Adresse und der Typ des Mitarbeiters erfasst werden.

An input form titled "Add User". It contains fields for "First Name \*", "Last Name \*", "E-Mail \*", and a dropdown menu for "User Type" set to "Relocator". At the bottom are "Add User" and "Cancel" buttons.

Abbildung 12.10: Benutzerverwaltung Hinzufügen eines Benutzers

Das folgende Listing beinhaltet die Implementation dieser Tabelle.

```
1 <h:form id="userAdm_form">
2   <p: dataTable value="#{econ.allEmployees}"
3     var="e"
4     id="table_userAdm"
5     editable="true">
6
7   <p:ajax event="rowEdit" listener="#{econ.onRowEdit}" update=":
8     userAdm_form:table_userAdm"/>
9
10  <p:column headerText="#" class="tColumn" >
11    <h:outputText value="#{e.id}" />
12
13  </p:column>
14
15  <p:column headerText="First" class="tColumn" >
16    <p:cellEditor>
17      <f:facet name="output"><h:outputText value="#{e.vorname}" />
18      </f:facet>
19      <f:facet name="input"><p:inputText id="modelInput2" value="#{e.vorname}" style="width:100%"/></f:facet>
20    </p:cellEditor>
21  </p:column>
22
23
24  <p:column headerText="Last" class="tColumn" >
25    <p:cellEditor>
26      <f:facet name="output"><h:outputText value="#{e.nachname}" /></f:facet>
27      <f:facet name="input"><p:inputText id="modelInput3" value="#{e.nachname}" style="width:100%"/></p:inputText></f:facet>
28    </p:cellEditor>
29  </p:column>
30
31
32  <p:column headerText="Email" class="tColumn" >
33    <p:cellEditor>
34      <f:facet name="output"><h:outputText value="#{e.email}" /></f:facet>
35      <f:facet name="input"><p:inputText id="modelInput4" value="#{e.email}" style="width:100%"/></p:inputText></f:facet>
36    </p:cellEditor>
37  </p:column>
38
39
40  <p:column headerText="Type" class="tColumn" >
41    <p:cellEditor>
42      <f:facet name="output"><h:outputText value="#{e.userType.type}" /></f:facet>
43      <f:facet name="input">
44        <h:selectOneMenu value="#{e.userType.type}">
```

```

43      <!-- itemValue = id in der Datenbank -->
44      <f:selectItem itemLabel="Relocator" itemValue="1" /
45          >
46          <f:selectItem itemLabel="Dispatcher" itemValue="2" /
47              >
48              <f:selectItem itemLabel="Viewer" itemValue="3" />
49      </h:selectOneMenu></f:facet>
50  </p:cellEditor>
51 </p:column>
52 <p:column headerText="Edit" class="tColumn" >
53
54     <p:rowEditor editTitle="Edit">
55         </p:rowEditor>
56     </p:column>
57
58     <p:column headerText="Delete" >
59         <p:commandButton value="Delete"
60             onclick="PF('dialogUserDel').show();"
61             action="#{econ.setDelEmployee(e)}"/>
62     </p:column>
63 </p:dataTable>
64 <p:commandButton value="Add User"
65             onclick="PF('dialogAddUser').show()"/>
66 </h:form>
67 </div>

```

Listing 12.5: benutzerverwaltung.xhtml

## Der Controller

```

1 package htlstp.bradaric.projekt_design_v1.controller;
2 //Imports
3
4 /**
5  *
6  * @author Matej
7  */
8 @Named("econ")
9 @SessionScoped
10 public class EmployeeController implements Serializable {
11
12     private List<UserCurrent> employees = new ArrayList<>();
13     private List<User> allEmployees = new ArrayList<>();
14     private UserCurrent currEmployee = null;
15     private UserCurrent selectedEmployee;
16     private User delEmployee;
17     private String newFName;
18     private String newLName;
19     private int newType;

```

```
20  private String newMail;
21
22
23  @PostConstruct
24  public void init() {
25      UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
26      Call<List<UserCurrent>> u = client.getAvailableUsers();
27      try {
28          employees = u.execute().body();
29      } catch (Exception ex) {
30          ex.printStackTrace();
31      }
32  }
33
34
35  public List<UserCurrent> getEmployees() {
36      UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
37      Call<List<UserCurrent>> u = client.getAvailableUsers();
38      try {
39          employees = u.execute().body();
40      } catch (Exception ex) {
41          ex.printStackTrace();
42      }
43      return employees;
44  }
45
46  public List<User> getAllEmployees() {
47      if (!allEmployees.isEmpty()) {
48          return allEmployees;
49      }
50      UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
51      Call<List<User>> u = client.getUsers();
52
53      try {
54          allEmployees = u.execute().body();
55      } catch (Exception ex) {
56          ex.printStackTrace();
57      }
58      allEmployees.sort(new Comparator<User>() {
59          @Override
60          public int compare(User o1, User o2) {
61              return o1.getId() - o2.getId();
62          }
63      });
64      return allEmployees;
65  }
66
67  }
```

```

68     public void delUser() {
69         UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
70         try {
71             boolean reg = client.updateUserActivated(delEmployee.getId(), false).execute().body();
72         } catch (IOException ex) {
73             ex.printStackTrace();
74         }
75     }
76
77     public void onRowEdit(RowEditEvent event) {
78         UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
79         try {
80             client.updateEmail(((User) event.getObject()).getId(), ((User) event.getObject()).getEmail()).execute();
81         } catch (IOException ex) {
82             Logger.getLogger(EmployeeController.class.getName()).log(Level.SEVERE, null, ex);
83         }
84     }
85
86     public void addUser() {
87         UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
88         User newUser = new User();
89         UserType nT = new UserType(getNewType());
90         newUser.setEmail(newMail);
91         newUser.setNachname(newLName);
92         newUser.setUserType(nT);
93         newUser.setVorname(newFName);
94
95         try {
96             client.createUser(newUser).execute();
97         } catch (IOException ex) {
98             ex.printStackTrace();
99         }
100    }
101
102    //Getter und Setter
103
104}
105

```

Listing 12.6: benuzerverwaltungController.java

In obigem Listing ist der Controller der Benutzerverwaltung abgebildet. In der `init()` Methode werden über das Webservice alle verfügbaren Mitarbeiter aus der Datenbank gelesen. In der `getallEmployees()` Methode werden alle Mitarbeiter aus der Daten-

bank gelesen und nach ID sortiert. In der `delUser()` Methode wird ein Flag in der Datenbank auf false gesetzt, und somit bedeutet, dass dieser Benutzer gelöscht wurde. In der `addUser()` Methode wird ein neuer Benutzer mit den vom User eingegebenen Attributen erzeugt und in die Datenbank geschrieben.

## 12.4 Benutzerdatenpflege

Diese Seite bietet dem Benutzer eine Möglichkeit, seine E-Mail-Adresse oder sein Passwort zu ändern, indem im ersten Eingabefeld die neue E-Mail-Adresse und in den anderen beiden das neue Passwort angegeben wird. Es besteht die Möglichkeit, nur die E-Mail zu ändern, indem die zwei Passwortfelder freigelassen werden.

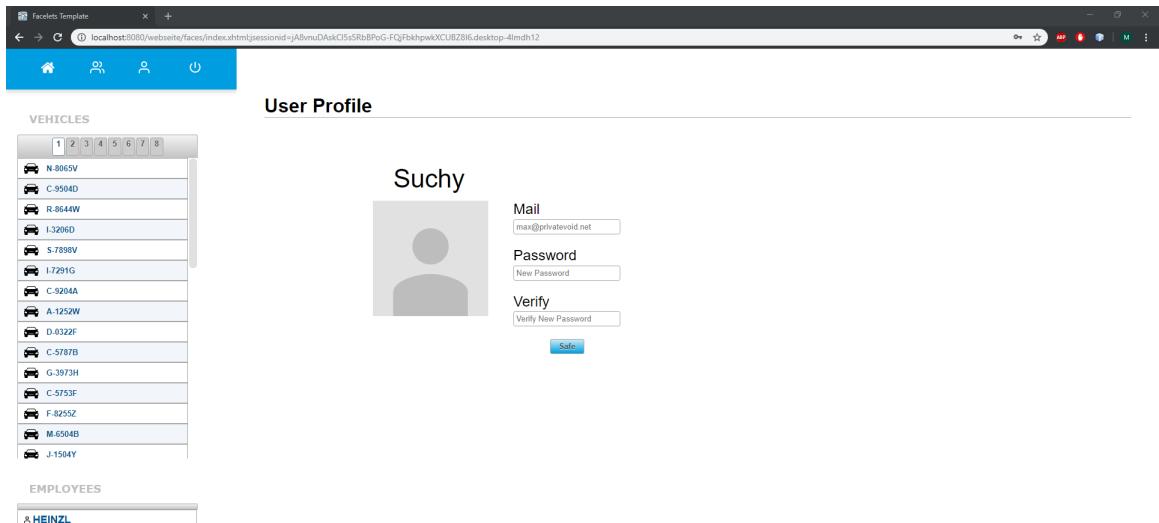


Abbildung 12.11: Benutzerprofil - Ändern der E-Mail oder des Passworts

### Der Controller

Das folgende Listing zeigt den Controller der Benutzerdatenpflege.

```

1 package ht1stp.bradaric.projekt_design_v1.controller;
2
3 //Imports
4
5 /**

```

```
6  *
7  * @author Matej
8  */
9 @Named("userCon")
10 @SessionScoped
11 public class UserController implements Serializable {
12
13     private String newMail;
14     private String newPwd1;
15     private String newPwd2;
16
17     private User currentUser;
18
19
20
21     public String updateUser() {
22         UserClient client = GenericService.getClient(UserClient.class, "REST_KEY");
23         try {
24             client.updateEmail(currentUser.getId(), newMail).execute();
25         } catch (IOException ex) {
26             Logger.getLogger(UserController.class.getName()).log(Level.SEVERE, null, ex);
27         }
28
29         if (newPwd1.equals(newPwd2)) {
30             try {
31                 client.updateUserPasswordWithoutReset(currentUser.getId(),
32                     newPwd1).execute();
33             } catch (IOException ex) {
34                 Logger.getLogger(UserController.class.getName()).log(Level.SEVERE, null, ex);
35             }
36         }
37     }
38
39
40     //Getter und Setter
41 }
```

Listing 12.7: profilController.java

In der Methode `updateUser()` wird die E-Mail und bei korrekter Eingabe auch das Passwort des Benutzers in der Datenbank überschrieben.

# Kapitel 13

## Grundlagen von Routenberechnungen

### 13.1 Überblick

Positionierungssysteme sind heute nicht mehr aus dem Alltag wegzudenken. Ursprünglich für das Militär entwickelt, werden sie heute auch zivil verwendet. Das *GPS* ist zum Beispiel seit dem 1. Mai 2000 zivil nutzbar. Dies reicht von Freizeitaktivitäten, wie GeoCaching, bis zum gewerblichen Einsatz. Auf letzteren wird in den folgenden Kapiteln näher eingegangen.

Dieses Kapitel beschreibt das Grundprinzip dieser Systeme. Weiters wird die Geschichte und Zukunftsentwicklung beschrieben. Zuletzt werden diverse Einsatzgebiete genannt und deren technischen Anforderungen erläutert.

Das am meisten verwendete System ist das us-amerikanische *NAVSTAR GPS*<sup>1</sup>. Als Konkurrenz wurde von Russland *GLONASS*<sup>2</sup> entwickelt. In jüngerer Vergangenheit wurde von der Europäischen Weltraumorganisation (ESA) begonnen ein eigenes System namens *Galileo* zu entwickeln. In der Volksrepublik China wird ebenfalls an einem System gearbeitet. Dieses trägt den Namen *Beidou*. Im weiteren Verlauf des Kapitels wird nur Bezug auf *GPS* und *Galileo* genommen.

Zur Erstellung dieses Kapitels wurden folgende Quellen verwendet:  
[He:Web01, Informationen zu GPS auf kowoma.de] [He:Web06, Was ist Galileo?, ESA]

---

<sup>1</sup>Navigational Satellite Timing and Ranging - Global Positioning System

<sup>2</sup>Global Navigation Satellite System

## 13.2 Funktionsprinzip

A global navigation satellite system (GNSS) is a type of satellite navigation that provides global coverage. A GNSS is defined by a constellation of orbiting satellites working together with a network of ground control stations and receivers that calculate ground positions through an adapted version of trilateration. [He:Web02, Definition von GNSS auf techopedia.com]

### 13.2.1 Trilateration

Das fundamentale Prinzip der satellitenbasierten Positionsbestimmung ist die Trilateration. Diese beschreibt, wie die Position eines Objektes anhand des Orts von und der Distanz zu drei anderen Objekten errechnet werden kann.

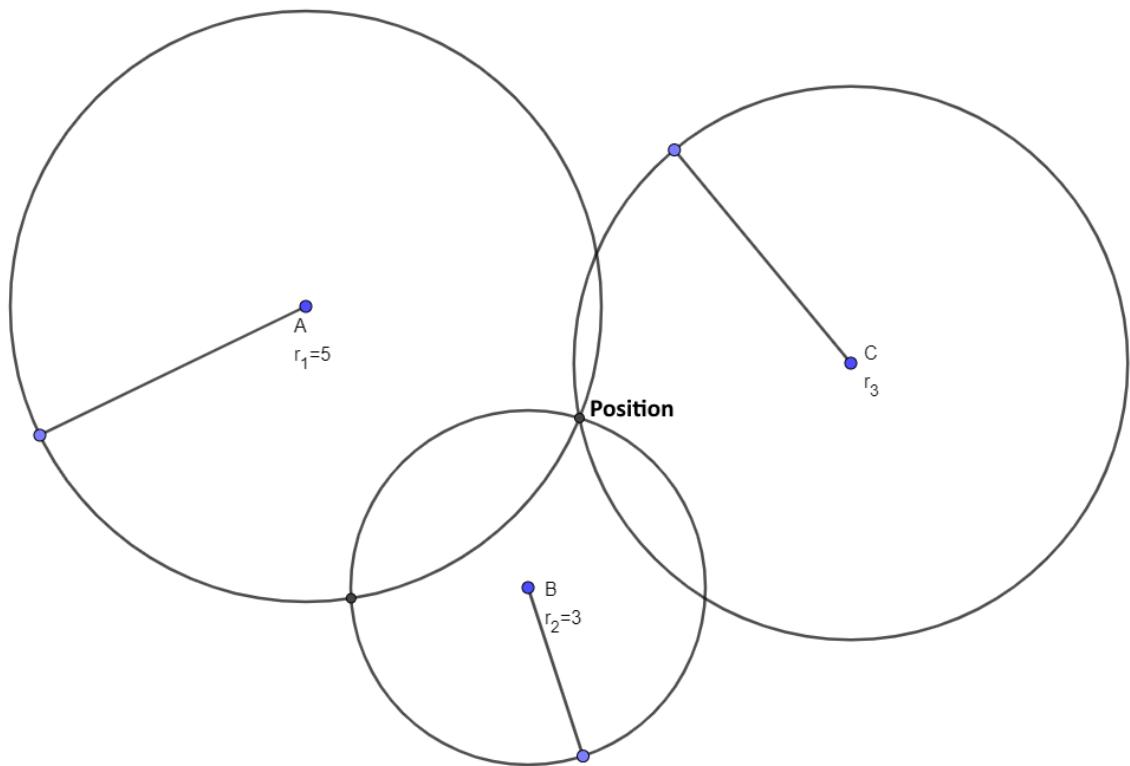


Abbildung 13.1: Trilateration in der Ebene

Die oben abgebildete Grafik stellt die Trilateration im zweidimensionalen Raum dar. Wird nur eine Distanz zu einem Objekt mit bekanntem Ort verwendet, so kann sich das

zu lokalisierende Objekt auf einer Kreisbahn befinden. Im dreidimensionalen Raum wäre dies eine Kugeloberfläche. Werden zwei Objekte und Distanzen zur Positionsbestimmung herangezogen, so ergibt sich im Raum eine Kreisbahn als Lösung. Diese entspricht zwei möglichen Positionen in der Ebene. Sobald drei Messungen verwendet werden ergibt sich eine zweideutige Lösung im Raum sowie eine eindeutige in der Ebene. Eine genaue Beschreibung der Trilateration ist unter [He:Web03, Funktionsbeschreibung von Trilateration auf [howstuffworks.com](http://howstuffworks.com)] zu finden.

### 13.2.2 Multilateration und Distanzberechnung

Alle GNSS verwenden Funksignale zur Distanzmessung. Dabei befinden sich Atomuhren in allen Satelliten für eine genaue Zeitmessung. Der Uhrzeitunterschied zwischen den Satelliten und dem Empfänger wird mit der Ausbreitungsgeschwindigkeit der Signale multipliziert. Diese beträgt annähernd Lichtgeschwindigkeit.

Da viele Empfänger keine genauen Uhren besitzen kann es zu Messungenauigkeiten kommen. Aus diesem Grund werden zur Positionsbestimmung mehr als drei Satelliten eingesetzt. Dabei spricht man von Multilateration. *GPS* hat 31 Satelliten im Betrieb, um eine flächendeckende Erreichbarkeit zu gewährleisten. Das europäische *Galileo* wird mit 30 Satelliten betrieben.

### 13.2.3 Positionen der Satelliten

Jeder Satellit hat seine Bahndaten gespeichert. Diese werden periodisch mit den Bodenstationen abgeglichen. Anhand der Bahndaten und Uhrzeit des Satelliten kann dessen genaue Position errechnet werden. Die Positionsdaten beziehen sich auf ein spezielles Koordinatensystem. Bei GPS wird hierfür *WGS84* verwendet [He:Web04, Beschreibung von WGS84 auf [gisgeography.com](http://gisgeography.com)].

### 13.2.4 Positionsbestimmung

Der Empfänger erhält die Daten von mindestens vier Satelliten. Diese bestehen aus der Satellitenidentifikation, aktueller Uhrzeit und Position. Als nächstes wird die Differenz zwischen den Uhrzeiten der Satelliten und der des Empfängers gebildet. Diese werden mit der Signalgeschwindigkeit multipliziert. Daraus ergeben sich die Distanzen zu den Satelliten. Auf Grund der Positionen kann nun der Ort des Empfängers errechnet werden. Dazu sind die ersten drei Satelliten in Verwendung. Der vierte wird für die

Korrektur der ungenauen Uhrzeit des Empfängers benötigt. Um weitere Messfehler beheben zu können müssen mehr als vier Satelliten zur Verfügung stehen.

## 13.3 Geschichte

### 13.3.1 NAVSTAR GPS

Das GPS nahm seinen Anfang im Jahr 1958, in welchem die US-Marine das *Transit* System entwickelte. Es war als *NNSS*<sup>3</sup> ab dem Jahr 1964 für das Militär verfügbar. Ab 1967 konnte es auch zivil eingesetzt werden. Das *Transit* System wurde 1996 außer Betrieb genommen.

Im Jahr 1973 wurde beschlossen ein Nachfolgesystem für *Transit* zu entwickeln. Die ersten Tests der Empfänger wurden 1977 durchgeführt. Dabei wurden sogenannte Pseudoliten<sup>4</sup> eingesetzt. In den Jahren 1978 bis 1985 wurden die ersten Satelliten in den Weltraum geschossen. Ab dem Jahr 1980 wurden alle Satelliten für die Erkennung von Atombombenexplosionen ausgerüstet.

Das GPS kann seit 1983 auch zivil über eine separate Frequenz genutzt werden. Dieses System hat jedoch eine geringere Genauigkeit als das des Militärs. Diese künstliche Ungenauigkeit wird als *selective availability (SA)* bezeichnet. Die SA wurde während des Golfkrieges (1990 bis 1991) abgeschaltet, da zu wenige militärische Empfänger zur Verfügung standen.

Von 1989 bis 1994 wurden die Satelliten der 2. Generation in den Orbit gebracht. Im Jahr 1993 wurde die *Erste Betriebsbereitschaft (Initial Operational Capability, IOC)* [He:Web05, Geschichte von GPS auf kowoma.de] erreicht. Dabei wurde bekanntgegeben, dass die zivile Nutzung des GPS kostenlos sein wird. Bis zum Jahr 2000 war diese jedoch weiterhin von der SA betroffen.

Am 17.7.1995 wurde das System in den Vollbetrieb genommen. Seit 2005 werden die Satelliten der 2. Generation vorübergehend durch neue ersetzt. Der genaue Ablauf wird in der Sektion *Zukunftsentwicklung* näher beschrieben.

---

<sup>3</sup>Navy Navigation Satellite System

<sup>4</sup>Pseudosatellit - Sender, der sich auf der Erde befindet

### 13.3.2 Galileo

Galileo ist dem us-amerikanischen GPS sehr ähnlich, jedoch nicht dazu kompatibel. Entwickelt wird es von der Europäischen Weltraumorganisation (ESA) und der Europäischen Union (EU).

Zum Testen der Grundfunktion des Systems wurde im Jahr 2006 die Inbetriebnahme von zwei Testsatelliten, sowie einiger Bodenstationen vorbereitet. Die ersten Testsignale der Satelliten wurden im Jahr 2007 bzw. 2008 gesendet. Durch eine Kooperation mit Russland, aus dem Jahr 2003, wurden deren GLONASS-Satelliten zum Test von Galileo verwendet. Dabei wurde auch getestet, ob die Systeme zueinander kompatibel sind.

Die ersten beiden Satelliten, welche für den Normalbetrieb gedacht sind, wurden im Oktober 2011 in den Weltraum befördert. Ein Jahr darauf starteten zwei weitere Satelliten. Im Jahr 2013 wurden diese vier Sender mit den Bodenstationen verknüpft. Es wurde ein erster Systemtest durchgeführt. Im Oktober des selben Jahres wurde der Test erfolgreich abgeschlossen.

Die nächsten zwei Satelliten wurden im August 2014 in Betrieb genommen. Gefolgt wurden sie von dem nächsten Paar im März 2015. Im Zeitraum von September 2015 bis Mai 2016 wurden drei weiter Paare im Orbit positioniert. Der erste Start mit vier Satelliten fand im November 2016 statt.

Galileo ist seit dem 15. Dezember 2016 für die Öffentlichkeit zugänglich [He:Web07, Start des Galileo-Betriebs auf zdnet.de]. Das System ist in fünf Dienste aufgeteilt [He:Web08, Übersicht zu Galileo auf kowoma.de]:

- **Allgemeiner Dienst**

Dieser Dienst ist kostenlos für den zivilen Gebrauch verfügbar.

- **Sicherer-Dienst**

Als *Safety-of-Life Service (SoL)* bezeichnet, steht er für wichtige Einsatzgebiete, wie dem Flugverkehr, zur Verfügung. Es werden automatisch Nachrichten über Einschränkungen des Systems gesendet.

- **Kommerzieller Dienst**

Dieser Dienst ist gegen eine Gebühr verwendbar. Er hat eine Genauigkeit von unter einem Meter. Ähnlich wie beim SoL wird die konstante Verfügbarkeit garantiert.

- **Regulierter Dienst**

Dieser Dienst ist nur für Polizei, Militär und ähnliche Organisationen zugänglich. Er besitzt grundsätzlich die selben Anforderungen, wie der *SoL* und kommerzielle Dienst.

- **Such- und Rettungsservice**

Dieser Service kann Notsignale empfangen. Es soll damit möglich sein, die Position von Unglücksorten automatisch an Rettungsleitstellen weiterzuleiten

## 13.4 Zukunftsentwicklung

In der nahen Zukunft wird die Satellitenflotte von GPS erneuert werden. Beim Galileo-System befinden sich die letzten Satelliten vor der Installation. Die genauen Pläne der Weltraumorganisationen werden in dieser Sektion kurz beschrieben. Zur Erstellung wurden folgende Quellen verwendet: [He:Web09, Zukunftsentwicklung von GPS auf skyrocket.de] [He:Web10, Galileo Fact-Sheet der ESA]

### 13.4.1 GPS-3

Die Modernisierung von GPS begann im Jahr 2005. Die vorhandenen Satelliten der zweiten Generation wurden durch temporäre *GPS-2RM* Satelliten ersetzt. Im Jahr 2008 wurde bekannt gegeben, dass Lockheed Martin<sup>5</sup> die nächste Generation von GPS-Satelliten bauen wird. Dies ist bereits die dritte Generation. Aus diesem Grund wird das Modernisierungsvorhaben auch als GPS-3 bezeichnet.

Das erneuerte System ermöglicht die Verbindung der Satelliten untereinander. Damit kann die gesamte Konstellation von einer Bodenstation aus aktualisiert werden. Weiters wird die Leistung und Verfügbarkeit des Militärcodes verbessert.

Der erste Satellit der dritten Generation wurde im Jahr 2018 installiert. Bis 2021 sollen die ersten acht in Betrieb sein. Ein Zeitplan für die restlichen ist aktuell<sup>6</sup> nicht bekannt.

<sup>5</sup>us-amerikanisches Rüstungsunternehmen

<sup>6</sup>Stand: 17.11.2018

### 13.4.2 Fertigstellung von Galileo

Die Satelliten mit den Nummern 23 bis 26 wurden am 25.07.2018 in den Weltraum geschossen. Sie befinden sich in der Testphase. Die letzten Satelliten sollen bis 2020 installiert werden [He:Web11, Pressemitteilung der Europäischen Kommission]. Danach besteht das Galileo-System aus einer Konstellation von 30 Satelliten.

Ein weiterer Entwicklungsschritt ist die Verfügbarkeit von Galileo-kompatiblen Empfängern. Im Mobilfunkbereich wird es bereits von den neuesten Smartphones unterstützt. Auch in der Luftfahrt wird Galileo teilweise schon verwendet. Unter anderem wird es von europäischen Flughäfen, wie zum Beispiel Frankfurt/Main, eingesetzt.

Eine genaue Auflistung von Galileo-kompatiblen Produkten ist unter [He:Web12, Einsatzgebiete von Galileo, ESA] zu finden.

## 13.5 Einsatzgebiete

### 13.5.1 Navigation

Die Navigation ist eines der Haupteinsatzgebiete von Positionierungssystemen. Ursprünglich wurden diese von Bodenstationen bedient. Heute werden sie über Satellitensysteme betrieben.

Eine häufige Anwendung ist das Navigieren im Straßenverkehr. Dazu werden dezidierte Geräte benötigt, welche im Fahrzeug installiert werden müssen. Ursprünglich hatten diese das Kartenmaterial für bestimmte Bereiche abgespeichert. Durch die Verfügbarkeit von Kartendaten im Internet, wie zum Beispiel durch *OpenStreetMap*, können die Daten nach Bedarf automatisch aktualisiert werden.

Mittlerweile haben viele moderne Fahrzeuge bereits eine Navigationsfunktion eingebaut. Weiters werden vermehrt Smartphones zum Navigieren verwendet. Ein großer Softwarevertreter für Smartphones ist *Google Maps*, das auf allen Android-Geräten vorinstalliert ist.

### 13.5.2 Einsatz im Projekt

Der Einsatz von Positionierungssystemen im Projekt CarSharing gliedert sich in drei Teile:

- **Tracken der Fahrzeuge**

Sobald ein Fahrzeug abgestellt wurde registriert es sich automatisch beim System. Es überträgt dabei seine aktuelle Position.

- **Tracken der Mitarbeiter**

Die Mitarbeiter verwenden eine mobile Android-Applikation. Diese sendet nach Aktivierung periodisch die aktuelle Position des Mitarbeiters an einen Server.

- **Routenberechnung**

Den Mitarbeitern soll der schnellste Weg zum zugeordneten Fahrzeug angezeigt werden. Dazu wird auf den Smartphones eine weitere Applikation benötigt.

Das Verwaltungssystem der Fahrzeuge wird vom Projektpartner zur Verfügung gestellt. Die Android-Applikation wird vom Projektteam entwickelt. Diese benötigt die Fahrzeugdaten aus dem ersten Punkt. Für die Routenberechnung kommt eine externe Applikation zur Verwendung. Möglicher Kandidaten werden später beschrieben.

# Kapitel 14

## Vorstellung diverser Kartenservices

### 14.1 Übersicht

Das Navigieren mit Karten ist bereits Alltag. Ursprünglich musste man sich Faltkarten aus Papier besorgen. Diese waren jedoch schnell veraltet, zeigten nur bestimmte Informationen und waren auf ein Gebiet beschränkt.

Heute ist es durch Kartensoftware möglich jeden Punkt der Erde abzubilden. Durch die Satellitenfotografie werden die Kartendaten kontinuierlich aktualisiert. Mit einer Internetverbindung kann man sich Echtzeitdaten, wie zum Beispiel die aktuelle Verkehrslage, darstellen lassen. Es können Daten zu POIs<sup>1</sup>, wie zum Beispiel Parks, Wahrzeichen oder Geschäfte, hinzugefügt und bearbeitet werden. Auch die Integration von ÖPNV<sup>2</sup>-Daten hilft beim Navigieren.

Dieses Kapitel beschäftigt sich mit solchen, digitalen Kartendiensten. Konkret wird auf das OpenSource-Projekt *OpenStreetMap* und auf den weit verbreiteten Dienst *Google Maps* Bezug genommen. Weiters werden *Here Maps*<sup>3</sup> und *Sygic Maps*<sup>4</sup> zum Vergleich herangezogen.

Zu den genannten Diensten wird ein kurzer Überblick zu deren Entstehungsgeschichte gegeben. Es werden auch ein paar Informationen zu den Herstellern genannt. Hauptsächlich wird das Angebot an Funktionen beschrieben. Zum Schluss werden die Verfügbarkeit und die möglichen Benutzungskosten erläutert.

---

<sup>1</sup>Point of Interest

<sup>2</sup>öffentlicher Personennahverkehr

<sup>3</sup>Navigationsdienst, welcher in Fahrzeugen verwendet wird

<sup>4</sup>Offline Navigationsdienst

## 14.2 OpenStreetMap

OpenStreetMap is an open initiative to create and provide free geographic data.

The goal of the foundation is to support but not control the OpenStreetMap project. It is dedicated to encouraging the growth, development and distribution of free geospatial data and to providing geospatial data for anybody to use and share. [He:Web13, FAQ der OSM Foundation]

### 14.2.1 Geschichte

Zur Erstellung dieses Abschnitts wurde folgende Quelle verwendet: [He:Web14, Geschichte von OSM im OSM Wiki]

OpenStreetMap wurde im Juli 2004 von Steve Coast gestartet. Es bietet weltweit kostenlose Kartendaten an. Zwei Jahre nach dem Beginn wurde die *OpenStreetMap Foundation* gegründet.

Seit 2007 wird jährlich die *State of the Map* Konferenz abgehalten. Im Jahr darauf wurde der Import der *TIGER*<sup>5</sup>-Daten abgeschlossen. Am 6. Jänner 2013 überstieg die Benutzeranzahl zum ersten Mal eine Millionen.

Laut [He:Web15, OSM Statistik auf openstreetmap.org] hat OpenStreetMap aktuell<sup>6</sup> folgende Datenmengen:

<b>Benutzer</b>	5.025.033
<b>GPS-Punkte</b>	6.475.049.635
<b>Knoten</b>	4.863.622.432
<b>Wege</b>	541.783.031
<b>Relationen</b>	6.323.249

Tabelle 14.1: Statistik zu OpenStreetMap

### 14.2.2 OpenStreetMap Foundation

Die *OpenStreetMap Foundation* ist eine Non-Profit-Organisation mit Sitz in London. Ihre Aufgabe ist die Verwaltung und Finanzierung des OpenStreetMap-Projekts. Jede

<sup>5</sup>Geodaten der USA

<sup>6</sup>Stand: 18.11.2018

natürliche Person kann, für einen Jahresbetrag von 15€, der Foundation beitreten. Unternehmen können das Projekt ab einem Betrag von 500€ pro Jahr unterstützen<sup>7</sup>. Sie definiert konkret folgende Aufgabenbereiche [He:Web17, Aufgaben der OSM Foundation auf osmfoundation.org]:

- **Rechtlicher Vertreter**

Die *OpenStreetMap Foundation* fungiert als Rechtlicher Vertreter für das OpenStreetMap Projekt.

- **Bereitstellung der Infrastruktur**

Sie ist für die Verfügbarkeit der Server und Dienste zuständig.

- **Verwalten von Spenden**

Die Foundation nimmt Spenden für das OpenStreetMap Projekt entgegen und teilt diese den Teilprojekten nach Bedarf zu.

- **Koordinierung des Gesamtprojekts**

Die *OpenStreetMap Foundation* kümmert sich um die Kommunikation zwischen den Teilprojekten, sowie um die Koordinierung derselben.

### 14.2.3 Preismodell

Die *OpenStreetMap Foundation* stellt alle ihre Dienste kostenlos zur Verfügung. Wenn ein Dienst verwendet wird, so muss ein Copyright-Hinweis<sup>8</sup> sichtbar sein. Neben den Services der *OSM Foundation* gibt es zahlreiche Dienste von Drittherstellern. Diese greifen auf die OpenStreetMap-Daten zu und verwenden diese weiter (zum Beispiel für die Routenberechnung). Die Hersteller dieser Dienste können für diese Gebühren verlangen.

### 14.2.4 Funktionsumfang

#### Verwendung durch den Menschen

Die Kartendaten von OpenStreetMap sind auf <https://www.openstreetmap.org> zugänglich. Hier können neben Straßen- und Gebäudedaten unter anderem Informationen zu POIs, öffentlichen Verkehrsmitteln sowie Einrichtungen abgefragt werden. Weiters sind Restaurants und Geschäfte auf der Karte eingetragen. über die Suchleiste kann nach

<sup>7</sup>[He:Web16, Informationen zur Mitgliedschaft auf osmfoundation.org]

<sup>8</sup>[He:Web18, Copyrightinformationen auf openstreetmap.org]

bestimmten Orten gesucht werden. Auch die Routenberechnung zwischen zwei Orten oder Koordinaten ist möglich.



Abbildung 14.1: St. Pölten auf OpenStreetMap

### Einbetten in eine eigene Website

Auf der Hauptseite von OpenStreetMap gibt es die Möglichkeit die aktuelle Kartenansicht zu exportieren. Dies kann ein statisches Bild sein oder ein `iframe`-Tag. Um die Karte bearbeiten zu können, zum Beispiel, um Marker hinzuzufügen, wird ein externes API benötigt. Auf der offiziellen OpenStreetMap-Wiki<sup>9</sup> werden hierfür drei APIs vorgeschlagen:

- Leaflet
- OpenLayers
- Google Maps API

### Externer Dienst: Routenberechnung mit OSRM

Auf der Hauptseite von OpenStreetMap können verschiedene Dienste für die Routenberechnung ausgewählt werden. Für diesen Abschnitt wird der Dienst OSRM<sup>10</sup>, welcher standardmäßig ausgewählt ist, für die Beschreibung herangezogen.

<sup>9</sup>[He:Web19, OSM Wiki-Eintrag zu JavaScript-Frameworks]

<sup>10</sup>Open Source Routing Machine

Das OSRM-Projekt bietet ein kostenloses Webservice an. Die wichtigste Funktion ist die Routenberechnung. Dieser können mehrere Koordinaten übergeben werden und sie retourniert die berechnete Route als eine Liste von Wegpunkten. Eine genaue Dokumentation ist unter [He:Web20, OSRM Webservice Dokumentation] zu finden.

### 14.2.5 Verwendungsbeispiele

#### Einbetten in eine eigene Website

Das folgende Beispiel zeigt eine eingebettete OpenStreetMap in einer HTML-Datei. Für die Erstellung des `iframe`-Tags wurde die Exportfunktion auf der Hauptseite von OpenStreetMap verwendet.

```
1 <html>
2   <head>
3     <title>OSM iframe Beispiel</title>
4   </head>
5
6   <body>
7     <iframe width="425" height="350" frameborder="0" scrolling="no"
8       marginheight="0" marginwidth="0" src="https://www.openstreetmap.org/
9         export/embed.html?bbox=15.619318485260012%2C48.20361842453503%2C15
.628277063369751%2C48.20691832589249&layer=mapnik" style="border
: 1px solid black"></iframe><br/><small><a href="https://www.
openstreetmap.org/#map=18/48.20527/15.62380">Größere Karte anzeigen<
/a></small>
10  </body>
11 </html>
```

Listing 14.1: osm\_iframe.html

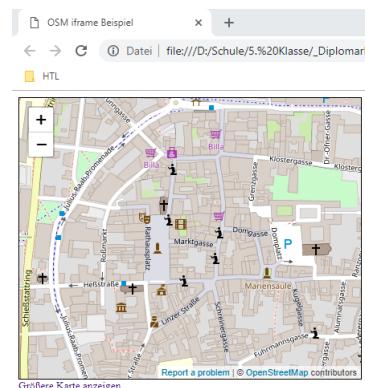


Abbildung 14.2: iframe Einbettung von OpenStreetMap

## Verwendung von OpenLayers

Dieses Beispiel verwendet das OpenLayers-API, um eine OpenStreetMap-Karte mit einem Marker anzuzeigen. Das Zentrum der Karte sowie der Marker zeigen auf die HTL St. Pölten. Zur Erstellung dieses Beispiels wurde folgende Vorlage verwendet: [He:Web21, Marker Beispiel zu Open Layers]

```
1 <html>
2   <head>
3     <title>OSM OpenLayers Beispiel</title>
4     <script src=".//openlayers/OpenLayers.js"></script>
5   </head>
6   <body>
7     <div id="mapdiv"></div>
8     <script>
9       var map = new OpenLayers.Map("mapdiv");
10      map.addLayer(new OpenLayers.Layer.OSM());
11
12      var lonLat = new OpenLayers.LonLat(15.6183001, 48.2070236)
13        .transform(
14          new OpenLayers.Projection("EPSG:4326"),
15          map.getProjectionObject()
16        );
17
18      var zoom = 18;
19      var markers = new OpenLayers.Layer.Markers("Markers");
20      map.addLayer(markers);
21      markers.addMarker(new OpenLayers.Marker(lonLat));
22      map.setCenter(lonLat, zoom);
23    </script>
24  </body>
25</html>
```

Listing 14.2: osm\_openlayers.html

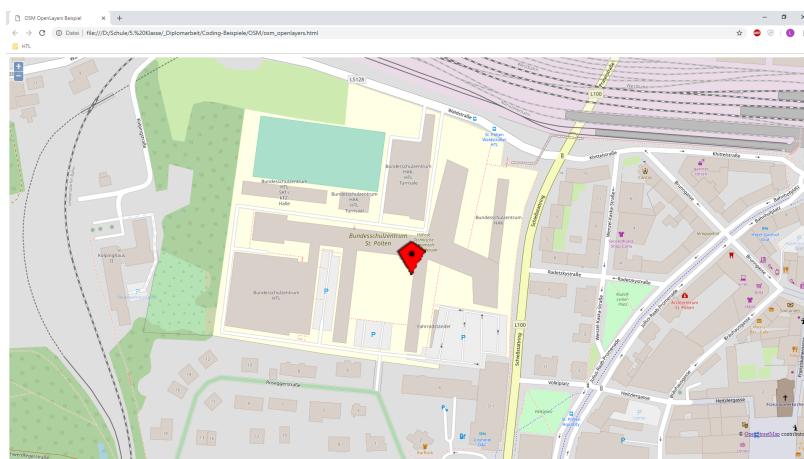


Abbildung 14.3: Verwendung von OpenLayers mit eigenem Marker

## 14.3 Google Maps

### 14.3.1 Entstehungsgeschichte

Zur Erstellung dieses Abschnitts wurde folgende Quelle verwendet: [He:Web22, Geschichte von Google Maps auf theguardian.com]

Google Maps wurde am 8. Februar 2005 in den Vereinigten Staaten veröffentlicht. Die Idee stammte von den dänischen Brüdern Lars und Jens Eilstrup Rasmussen. Google übernahm zwei Unternehmen und schuf ein Team von 50 Personen, welches sich um die Entwicklung von Google Maps und Google Earth kümmerte.

Im Jahr 2005 wurden Navigationsfunktionen für Autos und den öffentlichen Verkehr hinzugefügt. Eine weitere wichtige Neuerung war das Anzeigen von Satellitenfotos. Ein Jahr darauf wurde Google Street View veröffentlicht. Diese Funktion ermöglicht es durch ausgewählte Städte in einem virtuellen Fahrzeug zu fahren. Das Erstellen der Street View Bilder ermöglichte das Erfassen von lokalen Straßeninformationen, wie Einbahnstraßen oder Geschwindigkeitsbeschränkungen.

Weite Verbreitung fand Google Maps mit der Veröffentlichung des ersten iPhones. Durch eine Partnerschaft zwischen Google und Apple war auf jedem Gerät Google Maps vorinstalliert. Die Funktionalität wurde um *Turn-By-Turn Navigation* und Echtzeit Verkehrsdaten erweitert.

### 14.3.2 Google LLC

Die Google LLC<sup>11</sup> ist ein IT-Unternehmen mit Sitz in Mountain View, Kalifornien. Das Unternehmen wurde am 4. September 1998 von Larry Page und Sergey Brin gegründet. Bekannt wurde es durch die gleichnamige Suchmaschine. Die Google Suche hat am PC einen Marktanteil von 75,53% und auf Mobilgeräten von 86,61%<sup>12</sup>.

Im Jahr 2015 wurde das Unternehmen reorganisiert<sup>13</sup>. Dabei wurde die Muttergesellschaft *Alphabet Inc.* gegründet. Die Umstrukturierung soll dazu führen, dass die einzelnen Geschäftsbereiche unabhängiger voneinander arbeiten können. Die Hauptdienste, wie die Suche oder Google Maps, werden weiterhin unter dem Namen Google betrieben.

<sup>11</sup>Limited Liability Company

<sup>12</sup>[He:Web23, Marktanteil von Google Maps auf statista.com]

<sup>13</sup>[He:Web24, Informationen auf tagesschau.de]

### 14.3.3 Preismodell

Alle Informationen zu den Kosten stammen von [He:Web25, Preisrechner von Google]. Die Verwendung von Google Maps auf Mobilgeräten ist sowohl für statische als auch dynamische Karten kostenlos. Auch das Einbinden von Karten auf einer Website, ohne zusätzlicher Funktionalität, wird gratis zur Verfügung gestellt. Während der Entwicklung kann Google Maps in eingeschränkter Funktionalität verwendet werden. Dabei wird die Karte mit einem "Development"-Wasserzeichen versehen. Werden andere Dienste benötigt, so wird pro Aufruf eine Gebühr verlangt. Google stellt jeden Monat eine Gutschrift von 170€ zur Verfügung. Die folgende Tabelle zeigt die Monatskosten für alle Google Maps Dienste der Kategorie *Maps* bei einer Rate von 1.000 Aufrufen pro Monat.

Dienst	Kosten
Embed Advanced	12€
Static Maps	2€
Dynamic Maps	6€
Static Street View	6€
Dynamic Street View	12€
<b>Vorläufige Kosten</b>	<b>38€</b>
Gutschrift	170€
<b>Tatsächliche Kosten</b>	<b>0€</b>
Gutschrift Puffer	132€

Tabelle 14.2: Kostentabelle für Google Maps

### 14.3.4 Funktionsumfang

#### Google Maps im Webbrowser

Dieser Dienst ist über <https://www.google.at/maps> erreichbar. Hier kann nach Orten gesucht, Routen zwischen zwei Positionen berechnet, sowie Informationen zu POIs eingesehen werden. Weiters sind in ausgewählten Gebieten Daten, wie zum Beispiel Fahrpläne, zu den öffentlichen Verkehrsmitteln vorhanden. Es kann zwischen der Standard- und der Satellitendarstellung gewechselt werden. In größeren Städten, wie zum Beispiel Wien, kann man die Umgebung auch in 3D betrachten. Die Routenberechnung unterstützt auch eine Navigationsfunktion. Diese berücksichtigt bei der Zeitberechnung auch die aktuelle Verkehrslage. Diese kann jedoch auch unabhängig von der Navigation eingeblendet werden. All diese Funktionen sind auch in der mobilen Applikation von Google Maps verfügbar.

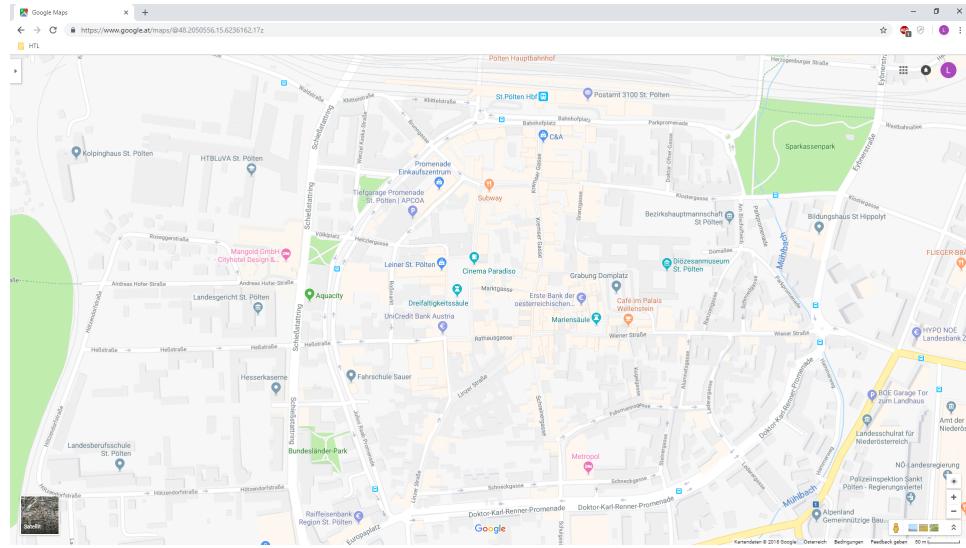


Abbildung 14.4: St. Pölten auf Google Maps

### Einbetten in eine eigene Website

Um Google Maps auf einer Website anzeigen zu können wird ein API-Key benötigt. Danach kann mit einem `iframe`-Tag die Karte eingebunden werden. Die Funktionalität ist dabei nur auf das Markieren eines einzigen Ortes beschränkt. Um Funktionen, wie Navigation oder Street View, verwenden zu können, muss bezahlt werden. Eine genaue Anleitung ist unter <https://developers.google.com/maps/documentation/embed/guide> zu finden.

### Zugriff über Webservices

Die Google Maps Platform bietet Webservices zur Abfrage von Daten an. Sie unterstützen die gesamte Funktionalität von Google Maps. Die Basis-URL dieser Webservices ist <https://maps.googleapis.com/maps/api/>. Danach folgen der Name des jeweiligen APIs, sowie die Anfrageparameter. Als Ergebnisformat kann JSON<sup>14</sup> oder XML gewählt werden.

Auch für die Verwendung dieser Services ist ein gültiger API-Key notwendig. Je nach verwendetem API wird pro Aufruf, wie in der vorherigen Sektion beschrieben, eine Gebühr verlangt. Google stellt für alle Google Maps Webservices Client-Bibliotheken zur Verfügung. Diese sind für folgende Umgebungen vorhanden: Java, Python, Go und Node.js. Eine genaue Dokumentation ist unter [He:Web26, GM-Client-Bibliotheken auf [developers.google.com](https://developers.google.com)] zu finden.

<sup>14</sup>JavaScript Object Notation

### 14.3.5 Verwendungsbeispiele

#### Einbetten in eine eigene Website

Dieses Beispiel zeigt die Verwendung eines `iframe`-Tags für die Einbettung einer Google Maps-Karte. Dabei wurde als anzugebender Ort die HTL St. Pölten gewählt. Wie zuvor beschrieben wird für die Benutzung ein API-Key benötigt. Für dieses und die folgenden Beispiele wurde der API-Key des Projektteams verwendet. Dieser ist in den hier abgebildeten Listings jedoch nicht ersichtlich.

```
1 <html>
2   <head>
3     <title>GM iframe Beispiel</title>
4   </head>
5
6   <body>
7     <iframe
8       width="600"
9       height="450"
10      frameborder="0" style="border:0"
11      src="https://www.google.com/maps/embed/v1/place?key=API_KEY
12      &q=HTL+StPölten, St.Pölten" allowfullscreen>
13   </iframe>
14 </body>
15 </html>
```

Listing 14.3: gm\_iframe.html

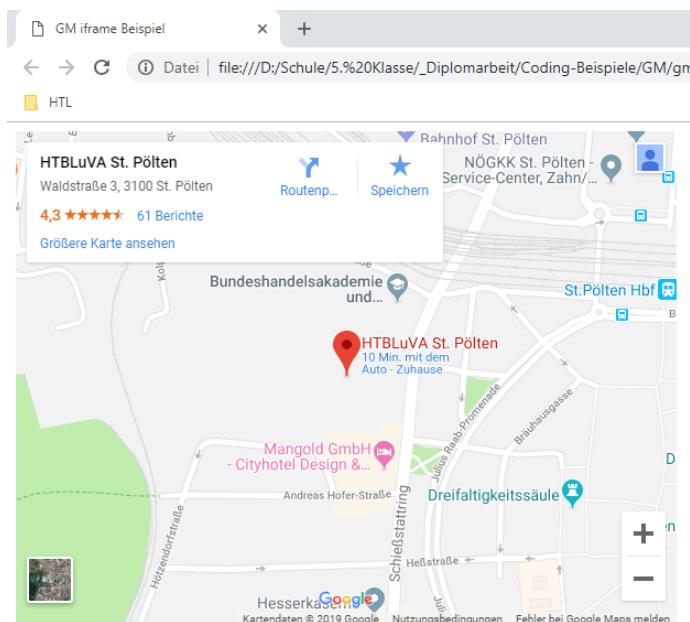


Abbildung 14.5: iframe Einbettung von Google Maps

## Routenberechnung über das Webservice

Das folgende Beispiel zeigt die Verwendung des *Directions*-API. Als zu berechnende Route wurde der Weg zwischen der HTL St. Pölten und dem Hauptbahnhof herangezogen. Als Bewegungsmodus wurde *zu Fuß gehen / walking* verwendet.

- **Anfrage**

```
https://maps.googleapis.com/maps/api/directions/json?key=API_KEY&
origin=48.2033875,15.617148&destination=48.206458,15.6222906&
mode=walking
```

- **Antwort**

- Legs

Beinhaltet die Distanz, benötigte Zeit, Start- und Endadresse dieses Abschnitts.

```
1 "distance": {
2     "text": "0,6 km",
3     "value": 561
4 },
5 "duration": {
6     "text": "7 Minuten",
7     "value": 425
8 },
9 "end_address": "Brunngasse 13, 3100 St. Pölten, Österreich",
10 "end_location": {
11     "lat": 48.2063843,
12     "lng": 15.6223814
13 },
14 "start_address": "Schießstattring 8, 3100 St. Pölten, Österreich",
15 "start_location": {
16     "lat": 48.2043556,
17     "lng": 15.6171113
18 }
```

Listing 14.4: gm\_route\_legs.json

- Steps

Jeder Abschnitt (Leg) beinhaltet mehrere Steps. Ein Step entspricht einem Abschnitt in der Turn-By-Turn Navigation. Jeder Step beinhaltet eine HTML-Beschreibung für die Navigation. Hier ein Auszug aus der Antwort:

*"html\_instructions": "Auf Heßstraße nach Osten Richtung Schießstattring"*

## 14.4 Here Maps

### 14.4.1 Entstehungsgeschichte

Für die Erstellung dieser Sektion wurden folgende Quellen verwendet: [He:Web27, Verkauf von HereMaps auf venturebeat.com] [He:Web28, Verkauf von HereMaps auf digitaltrends.com]

Here Maps besteht aus mehreren Softwareteilen von unterschiedlichen Herstellern. Diese Unternehmen wurden nach und nach von Nokia aufgekauft und deren Software vereint. Im Jahr 2001 trat Nokia dem *TellMaris* Konsortium bei, welches sich mit 3D Karten Technologie beschäftigte. Nokia kaufte die Software im Jahr 2006. Im selben Jahr übernahm es das Berliner Unternehmen Gate5 und dessen Software *smart2go*. Diese wurde als kostenloser Download auf Nokia Handys und Windows Phones angeboten.

Ein Jahr später kaufte Nokia *NAVTEQ*<sup>15</sup>. Die hierbei erworbenen Daten dienen als Grundlage für Nokia's Kartendienste. In den Jahren darauf kaufte Nokia weitere Unternehmen auf. Zwischenzeitlich waren die Kartendienste als *Ovi Maps* bekannt. Der Name wurde auf *Nokia Maps* im Jahr 2011 geändert. Bereits ein Jahr darauf wurde der Name auf *Here Maps* geändert. Die Software wurde schließlich auch für Android und iOS veröffentlicht.

Here Maps ist vor allem bei Autoherstellern beliebt und soll 80% der Navigationssysteme, welche in den Fahrzeugen ab Werk verbaut sind, betreiben. Auch die deutschen Fahrzeughersteller Audi, BMW und Daimler verwenden Here Maps in ihren Fahrzeugen.

Aus diesem Grund beschlossen die drei Unternehmen im Jahr 2015 das *HERE Global B.V.* Konsortium zu bilden und Here Maps von Nokia zu kaufen. Ein Grund, warum sich die drei Hersteller zum Kauf entschieden haben ist, dass Uber<sup>16</sup> die Software kaufen wollte. Die Fahrzeughersteller wollten jedoch selbst bestimmen, wie es mit dem Kartendienst weitergeht. Der Verkauf von Here Maps wurde am 05.12.2015 erfolgreich abgeschlossen.

---

<sup>15</sup>Anbieter von Geodaten

<sup>16</sup>Vermittlung von Privatfahrer und Taxis

### 14.4.2 HERE Global B.V.

HERE Global B.V. ist ein Unternehmen mit Sitz in Eindhoven, Niederlande. Es befindet sich im Mehrheitsbesitz von Audi, BMW und Daimler. Diese hielten anfänglich jeweils ein Drittel am Unternehmen. Im Jahr 2017 wollte eine asiatische Investorengruppe 10% der Anteile übernehmen. Die USA legte jedoch Veto ein und die Investition ist gescheitert.<sup>17</sup> Später, im selben Jahr, stieg Intel mit einem Anteil von 15% ein.<sup>18</sup> Im Jänner 2018 wurden jeweils 5% der Anteile von den Autozulieferern Bosch und Continental übernommen.<sup>19</sup>

### 14.4.3 Preismodell

Die Daten der Preise wurden folgender Quelle entnommen: [He:Web32, Preismodell von HereMaps]

Die Verwendung der Website ist kostenlos. Um gezielt auf Funktionalität zugreifen zu können muss ein Konto angelegt werden. Bei der Verwendung wird zwischen Klein-, Mittel- und Großkunden unterschieden. Es gibt folgende Optionen:

- **Freemium**

Die Verwendung dieses Plans ist grundsätzlich kostenlos. Er erlaubt 250.000 Transaktionen<sup>20</sup> pro Monat. Die Applikation, welche auf ein HERE-API zugreift, kann auf maximal 5.000 Geräten installiert werden. Sind alle 250.000 Transaktionen im aktuellen Monat verbraucht, so wird ein Dollar pro 1.000 weiteren Transaktionen abgebucht.

- **Pro**

Dieser Plan erlaubt 1.000.000 Transaktionen pro Monat. Die Anzahl an Installationen ist weiterhin auf 5.000 beschränkt. Zusätzlich steht ein Support per Email zur Verfügung und es wird eine 99,9%ige Up-Time garantiert. Dieser Plan ist für 449\$ pro Monat zu erwerben.

- **Für Großkunden**

Dieser Plan erlaubt über 1.000.000 Transaktionen pro Monat. Es sind auch mehr als 5.000 Installationen gestattet. Der Support ist per Email und Telefon erreichbar. Die Up-Time wird weiterhin zu 99,9% garantiert. Die Gebühren und der konkrete Funktionsumfang müssen mit dem Hersteller ausgehandelt werden.

<sup>17</sup>[He:Web29, Verkauf von HereMaps auf bloomberg.com]

<sup>18</sup>[He:Web30, Einstieg von Intel in HereMaps auf reuters.com]

<sup>19</sup>[He:Web31, Einstieg von Continental und Bosch in HereMaps auf cnet.com]

<sup>20</sup>entspricht einem oder mehreren Serveraufrufen, je nach API

Für den Freemium-Plan wird grundsätzlich keine Kreditkarte benötigt. Sobald das Gratis-Limit erreicht wurde, muss eine gültige Kreditkarte angegeben werden, um die Services weiterhin verwenden zu können. Es ist gestattet den Freemium-Plan zu verwenden, obwohl man seine eigene Applikation nur gegen eine Gebühr zur Verfügung stellt.

#### 14.4.4 Funktionsumfang

##### Here Maps im Webbrower

Der Dienst ist über <https://wego.here.com> erreichbar. Die Funktionalität ist sehr ähnlich zu Google Maps im Webbrower. Funktionen, wie das Anzeigen von POIs, müssen über ein Menü aktiviert werden und sind nicht standardmäßig aktiviert. Es können Daten zur Verkehrslage sowie zu den öffentlichen Verkehrsmitteln angezeigt werden.

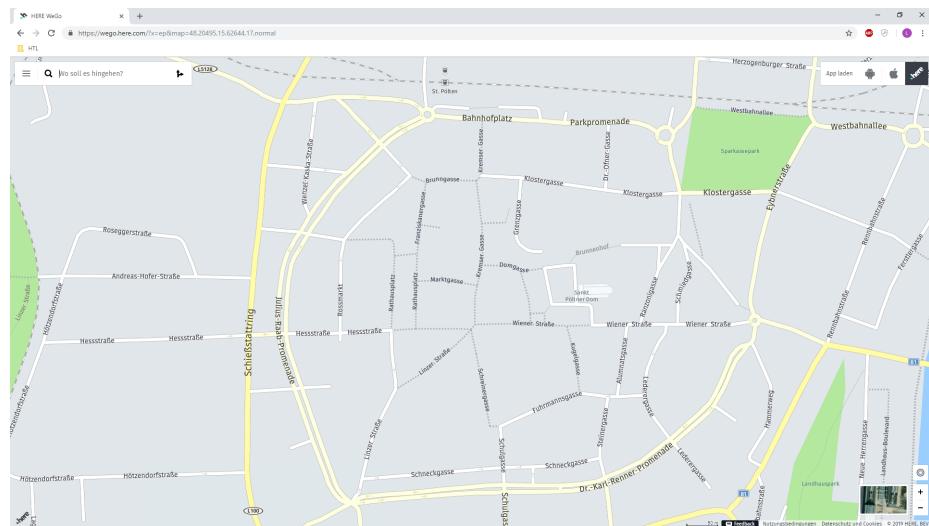


Abbildung 14.6: St. Pölten auf Here Maps

##### Zugriff über Webservices

Eine Liste aller Webservices ist unter [He:Web33, Webservice Dokumentation auf developer.here.com] zu finden. Im folgenden werden nur für das Projekt relevante Services beschrieben.

##### Map Tile API

Dieses API ermöglicht das Abrufen von Kartenteilen. Jedes Teil wird durch ein Zeilen-Spalten-Wertepaar identifiziert. Weiters können die Bildgröße (in Pixel) und das Bildformat angegeben werden. Es gibt mehrere Teiltypen, welche unterschiedliche Informationen beinhalten. Durch das Abfragen benachbarter Kartenteile kann ein Kartenausschnitt erstellt werden.

### **Routing API**

Mit diesem API können Routeninformationen abgefragt werden. Dabei werden Start und Ende als Wegpunkte mit Koordinaten angegeben. Weiters können verschiedene Modi gesetzt werden. Diese definieren das Verkehrsmittel (z.B. Auto oder zu Fuß), die gewünschte Berechnungsmethode (kürzester Weg oder kürzeste Zeit) und die Beachtung der Verkehrslage.

### **Traffic API**

Dieses API ermöglicht die Abfrage von Verkehrsstaus und Unfällen. Als Anfrageparameter wird die ungefähre Position des zu überprüfenden Gebiets benötigt. Das API liefert als Ergebnis alle verfügbaren Daten, welche sich in dem Gebiet um die Abfrageposition befinden.

## **14.5 Sygic Maps**

### **14.5.1 Entstehungsgeschichte**

I was looking for the next challenge. I felt that navigation might be the future. It was pretty difficult to start with, because I had no experience in that space, but that's what drove me. [He:Web34, Michal Štencl, Gründer von Sygic]

Für die Erstellung dieser Sektion wurden folgende Quellen verwendet: [He:Web34, Interview mit Michal Štencl auf bbc.com] [He:Web35, Artikel auf cnet.com] [He:Web36, Informationen zu Sygic auf sygic.com]

Die Idee zu Sygic Maps entstand im Jahr 2004. Michal Štencl begann mit der Entwicklung an dem Navigationsdienst. Zu dieser Zeit waren einige Betriebssysteme am mobilen Markt vorhanden. Da der mobile Sektor noch jung war, war nicht bekannt, welches System sich durchsetzen würde. Aus diesem Grund beschloss Štencl seine App plattformunabhängig zu entwickeln.

Als im Jahr 2007 das erste iPhone auf den Markt kam, war Sygic die erste Navigationsapp, welche auf dem Gerät verfügbar war. Diesen Vorsprung hatte man, obwohl bereits bei der Vorstellung des iPhones eine App des Navigationsgeräteherstellers TomTom präsentiert wurde. Die erste Version von Sygic auf dem iPhone war nur in Australien und Ostasien verfügbar. Die Versionen für Europa und Nordamerika folgten später.

Sygic Maps ist mit 200 Millionen Nutzern eine der meistverwendeten Navigationsapps weltweit. Mittlerweile bietet man auch spezialisierte Apps für Lastkraftwagen und Taxis an. An einer Fahrrad-Navigation wird ebenfalls gearbeitet.

### 14.5.2 Sygic a.s.

Die Sygic a.s.<sup>21</sup> ist ein Softwarehersteller mit Sitz in Bratislava, Slowakei. Das Unternehmen wurde im Jahr 2004 von Michal Štencl, Martin Kališ und Peter Pecho gegründet. Die Vision des Unternehmens ist es, jeden, der unterwegs ist, sicher ans Ziel zu führen [He:Web36, Informationen zu Sygic auf [sygic.com](http://sygic.com)].

### 14.5.3 Preismodell

Die Daten der Preise wurden aus folgender Quelle entnommen: [He:Web37, Onlineshop auf [sygic.com](http://sygic.com)]

Die Verwendung von Sygic Maps am PC ist kostenlos. Die mobile Applikation wird über ein Freemium-Modell vertrieben. Es ist eine sieben Tage gültige Testversion, welche den vollen Funktionsumfang besitzt, kostenlos verfügbar. Nach Ablauf der Testversion sind nur noch die Grundfunktionen verfügbar (siehe später).

Die Premiumlizenz muss einmalig erworben werden und ist ein Leben lang gültig. Beim Erwerb kann zwischen iOS und Android sowie zwischen den Kontinenten unterschieden werden. Es stehen drei Pakete zur Auswahl: Region, Region + Verkehr und Welt + Verkehr. Für die folgende Tabelle wurde als Region Europa und als Betriebssystem Android gewählt.

	Europa	Europa + Traffic	Welt + Traffic
Offline Karten	Ja	Ja	Ja
Tempolimit	Ja	Ja	Ja
Navigation mit Sprachanweisung	Ja	Ja	Ja
Benzinpreise	Ja	Ja	Ja
Fahrbahnassistent	Ja	Ja	Ja
Routen in Echtzeit teilen	Ja	Ja	Ja
Aktuelle Verkehrsinformationen	Nein	Ja	Ja
Weltweite Kartenabdeckung	Nein	Nein	Ja
Preis	49,99€	79,99€	99,99€

Tabelle 14.3: Kostentabelle für Sygic Maps

<sup>21</sup>Akciová spoločnosť - Österreich: AG

### 14.5.4 Funktionsumfang

#### Sygic Maps im Webbrowser

Der Dienst ist über <https://maps.sygic.com> erreichbar. Hier kann man nach Orten suchen, Wegbeschreibungen abrufen und Informationen zu POIs anzeigen lassen. Weiters können für das aktuell sichtbare Gebiet Reiseführer, Hotels, mögliche Touren und lokale Autovermieter angezeigt werden. Wie auf dem nachfolgendem Bild zu sehen ist, werden auch aktuelle Informationen zur Witterung angezeigt. In diesem Fall wird auf Glatteis hingewiesen.

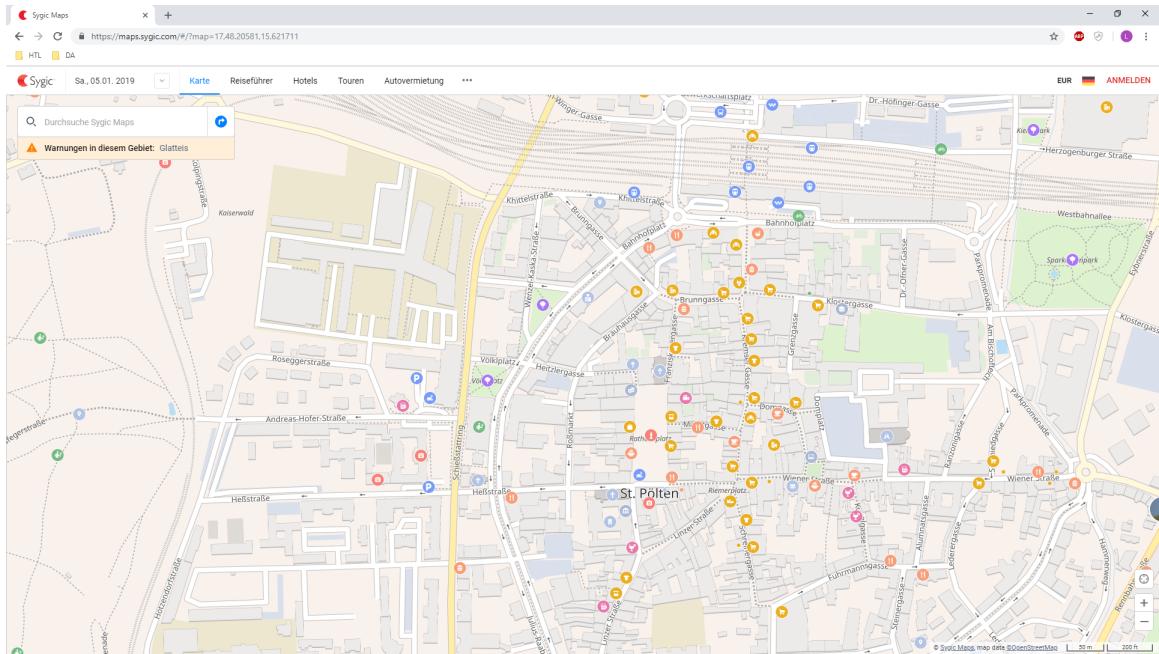


Abbildung 14.7: St. Pölten auf Sygic Maps

#### Zugriff über Webservices

Die Dokumentation der Webservices ist unter [He:Web40, Webservice Dokumentation auf [sygic.com](http://sygic.com)] zu finden.

Sygic stellt einen Großteil ihrer Funktionalität als Webservices zur Verfügung. Für den Zugriff wird ein API-Key benötigt. Ein Test-Key kann auf der Dokumentationsseite der Webservices angefordert werden. Die Kosten für den Erwerb eines Keys sind nur auf Anfrage verfügbar. Sygic bietet für die Verwendung ihrer Webservices SDKs für iOS und Android an. Weiters stellen sie eine JavaScript-Bibliothek zur Verfügung.

#### Maps API

Dieses API ermöglicht das Anzeigen von dynamischen Karten. Der Entwickler kann konfigurieren, welche Informationen angezeigt werden. So können nach Bedarf Informationen zu POIs oder der aktuellen Verkehrslage ein- und ausgeblendet werden.

## Routing API

Mit diesem API können Routen zwischen zwei Koordinaten berechnet werden. Zusätzlich zum Start- und Endpunkt können Zwischenpunkte angegeben werden. Weiters kann konfiguriert werden, dass bestimmte Straßentypen, wie zum Beispiel Autobahnen, oder ganze Gebiete zur Routenberechnung nicht herangezogen werden sollen.

Auch die Beachtung der Verkehrslage kann (de)aktiviert werden. Es kann zwischen verschiedenen Fahrzeugtypen und Fußgänger unterschieden werden. Um auf Gewichtsbeschränkungen reagieren zu können, kann das Gewicht des Fahrzeugs eingestellt werden. Weiters kann die Maximalgeschwindigkeit des Fahrzeugs angegeben werden, um die Zeitberechnung genauer durchführen zu können. Zu guter Letzt kann bei der Berechnungsmethode zwischen *kürzester Zeit* und *kürzester Weg* unterscheiden werden.

## Sygic Maps als mobile Applikation

Sygic Maps wurde von Anfang an als Navigationsapp konzipiert. Ihre Benutzeroberfläche wurde echten Navigationsgeräten nachempfunden. Das Hauptmerkmal der App ist, dass alle Karten auf das Gerät heruntergeladen werden. Dies ermöglicht die Navigation ohne eine Verbindung zum Internet. Sie stellt folgende Funktionalität zur Verfügung:

- **Grundfunktionen<sup>22</sup>**

- Offline-Karten
- Kartenaktualisierungen
- Automatische Streckenführung
- Hinweise auf Geschwindigkeitskontrollen
- Parkplatzinformationen
- Interessante Orte (POI)

- **Premiumfunktionen<sup>23</sup>**

- Offline 3D-Karten
- Navigation mit Sprachanweisung
- Geschwindigkeitsbegrenzungen
- Echtzeit-Verkehrsinformationen

---

<sup>22</sup>[He:Web38, Grundfunktionen der Sygic-App]

<sup>23</sup>[He:Web39, Premiumfunktionen der Sygic-App]

# Kapitel 15

## Bewertung und Auswahl

### 15.1 Bewertungskriterien

In diesem Kapitel werden die oben genannten Dienste miteinander verglichen. Als erstes werden die Anforderung des Projekts erläutert. Danach werden die Dienste in den jeweiligen Kategorien bewertet. Die Teilbewertungen werden am Ende des Kapitels zu einer Gesamtbewertung zusammengefasst. Aus dieser wird der Dienst mit der besten Bewertung ausgewählt. Der ausgewählte Dienst wird danach im Projekt verwendet. Im folgenden werden die oben genannten Dienste nur noch mit ihren Abkürzungen erwähnt: OpenStreetMap (OSM), Google Maps (GM), Here Maps (HM), Sygic Maps (SM).

Folgende Kriterien werden zur Bewertung herangezogen:

- **Funktionsumfang**

Hier werden die für das Projekt benötigten Funktionen angeführt und mit den verfügbaren Funktionen der Dienste verglichen.

- **Preis**

In diesem Punkt werden die (möglichen) Kosten für die Verwendung des jeweiligen Dienstes analysiert.

- **Integration**

In diesem Punkt werden die verfügbaren Zugriffsmethoden analysiert und mit den Anforderungen im Projekt verglichen.

- **Verfügbarkeit**

Hier wird die vom Hersteller garantierte Verfügbarkeitsrate analysiert.

## 15.2 Anforderungen

### 15.2.1 Website

Die Website im Projekt dient als Übersicht, auf welcher die Positionen der Mitarbeiter und Fahrzeuge angezeigt werden. Dabei handelt es sich um eine statische Karte, auf welcher die Positionen mit Marker eingezeichnet sind. Die Marker werden von der Software neu platziert, wenn sich die Positionen ändern. Die Karte muss dabei nicht neu geladen werden. Das bedeutet, dass die Karte im Idealfall nur einmal pro Login geladen werden muss. Die Website wird mit der JSF<sup>1</sup>-Technologie entwickelt. Der ausgewählte Kartendienst sollte dazu kompatibel sein.

Im Projekt wurde die Annahme getroffen, dass die Website im Realbetrieb von vier Dispatchern<sup>2</sup> bedient wird. Weiters wurde angenommen, dass zwei Personen aus der Chefetage sich über die Website einen Überblick über den Betrieb machen werden.

Es wird angenommen, dass jeder Dispatcher maximal zehn Kartenanfragen pro Tag versendet. Dies entspricht 40 täglichen Anfragen. Um die beiden Personen der Chefetage zu erfassen, wird diese Zahl auf 50 Anfragen pro Tag aufgerundet. Dadurch ergeben sich maximal 1.400 Anfragen im Monat (gerechnet mit 28 Tagen).

### 15.2.2 Mobile Applikation

Die mobile Applikation im Projekt wird von den Mitarbeitern verwendet, welche die Fahrzeuge umparken. Sie zeigt eine statische Karte, auf welcher die Positionen der Fahrzeuge eingezeichnet sind. Weiters kann der Benutzer seinen aktuellen Auftrag einsehen. Dabei wird auf einer weiteren Karte die Position des Benutzers und des zugewiesenen Fahrzeugs angezeigt. Hierbei wird auch die Route vom Benutzer zum Fahrzeug angezeigt. Letztendlich wird die Route vom Fahrzeug zur neuen Position angezeigt.

Die mobile Applikation wird unter Android entwickelt. Sie wird voraussichtlich von 15 Mitarbeitern verwendet werden. Es wird angenommen, dass jeder Mitarbeiter täglich bis zu zehn Fahrzeuge umparken kann. Dies entspricht 150 Aufträgen pro Tag. Jeder Auftrag besteht aus drei Anfragen für statische Karten sowie aus zwei Routenberechnungen. Dadurch ergeben sich maximal 12.600 Anfragen im Monat (gerechnet mit 28 Tagen) für statische Karten und maximal 8.400 Anfragen für Routenberechnungen.

---

<sup>1</sup>Java Server Faces

<sup>2</sup>teilen Fahrzeuge den Mitarbeitern zu

### 15.2.3 Notwendige Funktionen

Für das Projekt ist es nötig eine Karte auf einer Website sowie in einer mobilen Android-Applikation darstellen zu können. Weiters ist es nötig Routen zwischen den Mitarbeitern und den Fahrzeugen berechnen und anzeigen zu können. Als letztes muss eine Eventschnittstelle zur Verfügung stehen, mit der die Karte vom Benutzer bzw. der jeweiligen Software manipuliert werden kann (z.B. Daten auslesen und hinzufügen).

### 15.2.4 Wunschkriterien

Es wäre optimal, wenn konfiguriert werden kann, was auf der Karte sichtbar ist. Das bedeutet, dass nicht relevante Daten, wie zum Beispiel POIs, ausgeblendet werden können.

### 15.2.5 Integration

Wie bereits beschrieben wird die Website im Projekt mit JSF entwickelt. Die Einbindung von Karten sollte wenn möglich damit kompatibel sein. Weiters wird im Projekt die JSF-Erweiterung *PrimeFaces* verwendet.

Für die mobile Applikation muss eine native Einbindung des Dienstes gewährleistet sein. Das bedeutet, es muss möglich sein, den Dienst direkt über ein API ansprechen zu können. Die Einbindung sollte ohne großen Aufwand möglich sein.

### 15.2.6 Verfügbarkeit

Die im Projekt entwickelte Software unterstützt das Unternehmen im täglichen Geschäft. Aus diesem Grund muss die Software tagsüber jederzeit zur Verfügung stehen. Da das Anzeigen von Karten und die Berechnung von Routen ein wichtiger Teil der Software ist, werden an die Anbieter die gleichen Verfügbarkeitsanforderungen gestellt.

## 15.3 Funktionsumfang

Die Bewertung der Funktionalität wird über eine Gewichtung durchgeführt. Die Mussfunktionen haben jeweils eine Gewichtung von 25%. Die Eventschnittstelle hat nur eine Gewichtung von 15%. Die Wunschkriterien werden mit 10% gewichtet.

Für die Bewertung wird folgender Schlüssel verwendet:

- **Website- und Androideinbindung, Routenberechnung**

- **3 Punkte**  
Native Unterstützung durch den Hersteller
- **2 Punkte**  
Verfügbar unter Verwendung von Drittsoftware
- **1 Punkt**  
Verwendung über selbst entwickelte Software
- **0 Punkte**  
Nicht Verfügbar

- **Eventschnittstelle und erweiterte Konfiguration**

- **2 Punkte**  
Native Unterstützung durch den Hersteller
- **1 Punkt**  
Verfügbar unter Verwendung von Drittsoftware
- **0 Punkte**  
Nicht Verfügbar

Funktion	Gewichtung	OSM		GM		HM		SM	
Websiteeinbindung	25%	3	75	3	75	3	75	3	75
Androideinbindung	25%	2	50	3	75	3	75	3	75
Routenberechnung	25%	2	50	3	75	3	75	3	75
Eventschnittstelle	15%	2	30	2	30	2	30	2	30
erweiterte Konfiguration	10%	1	10	2	20	2	20	2	20
<b>Gesamt</b>	<b>100%</b>	<b>215</b>		<b>275</b>		<b>275</b>		<b>275</b>	

Tabelle 15.1: Bewertung der Funktionalität

Aus der Tabelle ergibt sich, dass OSM zwar die benötigten Funktionen zur Verfügung stellen kann, dafür jedoch Software von Drittherstellern benötigt wird. Die anderen drei Hersteller bieten alle Funktionen, auch die optionalen, selbst an, wodurch sie alle die gleiche, maximale Punkteanzahl erreichen.

## 15.4 Preis

Aus den Anforderungen ergeben sich insgesamt maximal 14.000 Anfragen für statische Karten und 8.400 Anfragen für Routenberechnungen im Monat. Diese Information wird benötigt, da die meisten Preismodelle volumenbasiert sind. Im folgenden werden die voraussichtlichen Kosten der einzelnen Dienste angeführt. Danach werden diese verglichen und vom günstigsten zum teuersten Anbieter sortiert.

### 15.4.1 OpenStreetMap

Wie bereits in der Sektion 14.2.3 auf Seite 165 beschrieben, kann OSM kostenlos verwendet werden. Es wird angenommen, dass die benötigte Drittsoftware ebenfalls kostenlos genutzt werden kann. Das bedeutet, dass für den Einsatz von OSM im Projekt, keine Kosten entstehen würden.

### 15.4.2 Google Maps

Wie bereits in der Sektion 14.3.3 auf Seite 170 beschrieben, verwendet Google ein volumenbasiertes Preismodell. Mithilfe des dort angeführten Preisrechners ergeben sich monatliche Kosten von 60€. Da Google jedoch eine Gutschrift von 170€ zur Verfügung stellt, wäre die Verwendung von GM im Projekt kostenlos. Dabei bestünde ein 110€ großer Puffer. Für die Verwendung von GM wird ein Google Konto benötigt. Dabei muss eine Kreditkarte angegeben werden, auch wenn keine Kosten entstehen.

### 15.4.3 Here Maps

Wie bereits in der Sektion 14.4.3 auf Seite 175 beschrieben, stellt HM drei Preispläne zur Verfügung. Insgesamt werden 22.400 Anfragen im Monat gesendet. Das Limit des kostenlosen Plans von HM liegt bei 250.000 Anfragen pro Monat. Aus diesem Grund wäre der Einsatz von HM im Projekt kostenlos. Für die Verwendung des kostenlosen Plans wird ein Here Account benötigt. Hier muss jedoch keine Kreditkarte angegeben werden.

### 15.4.4 Sygic Maps

Für die Verwendung der SM-APIs wird ein API-Key benötigt. Es kann ein kostenloser Key angefordert werden. Informationen zu den Verwendungsgebühren werden nur auf Anfrage bekanntgegeben. Es können daher keine Aussagen bezüglich der möglichen Kosten für den Einsatz im Projekt getätigt werden.

### 15.4.5 Bewertung

Um die vier Dienste in der Kategorie *Preis* zu bewerten, werden Punkte von eins bis vier vergeben. Jede Punkteanzahl kann nur einmal vorkommen und je höher sie ist desto besser. Die Dienste werden wie folgt bewertet:

- **OpenStreetMap**

OSM wird mit vier Punkten bewertet. Begründet wird dies dadurch, dass der gesamte Dienst kostenlos ist und ohne ein Konto verwendet werden kann.

- **Here Maps**

HM wird mit drei Punkten bewertet. Es wird zwar ein Here Konto benötigt, um auf die Services zugreifen zu können aber man muss keine Zahlungsinformationen angeben.

- **Google Maps**

GM wird mit zwei Punkten bewertet. Ähnlich wie bei HM wird ein Konto benötigt, jedoch verlangt Google unter allen Umständen die Angabe einer Kreditkarte.

- **Sygic Maps**

SM wird mit der niedrigsten Punkteanzahl von einem Punkt bewertet. Der Grund dafür ist, dass ohne Anfrage keine Preisinformationen zur Verfügung stehen. Für die Anforderung eines Test-Keys wird jedoch wie bei GM und HM ein Konto benötigt.

Abschließend ist zu erwähnen, dass der Umfang des Projekts zu klein ist, um in der Kategorie *Preis* einen Unterschied zu erzielen. Es ist davon auszugehen, dass unter der Verwendung von SM ebenfalls keine Kosten entstehen würden.

## 15.5 Integration

Die Anforderungen wurden bereits in der Sektion 15.2.5 auf Seite 183 genauer beschrieben. Im folgenden werden die Stärken und Schwächen der einzelnen Dienste beschrieben. Anschließend werden die Dienste anhand dieser Analyse bewertet. In der Sektion *Funktionsumfang* wurde auf die allgemeine Möglichkeit der Einbindung in eine Website sowie in Android eingegangen. In dieser Sektion wird auf die Kompatibilität mit den im Projekt bereits vorhandenen Technologien Bezug genommen.

### 15.5.1 OpenStreetMap

OSM stellt kein eigenes Framework für die Integration mit JSF zur Verfügung. Über die Fremdbibliothek *BootsFaces* kann OSM in JSF eingebunden werden. Da bereits *PrimeFaces* im Projekt in Verwendung ist kann *BootsFaces* nur erschwert verwendet werden. Der Grund dafür ist, dass die beiden Frameworks nicht komplett zu einander kompatibel sind, wie unter [He:Web41, Integration von PrimeFaces] beschrieben ist.

Auch für Android stellt OSM kein eigenes API zur Verfügung. Im OSM-Wiki findet sich unter <https://wiki.openstreetmap.org/wiki/Frameworks> eine Liste von APIs, welche auch unter Android funktionsfähig sind.

### 15.5.2 Google Maps

Das im Projekt verwendete Framework *PrimeFaces* stellt die *gmap*-Komponente nativ zur Verfügung. Über diese ist es möglich eine GM-Karte anzeigen zu lassen und mit dieser zu interagieren. Dies ermöglicht eine einfache Integration von GM mit der vorhandenen Software im Projekt.

Mit dem *Maps SDK for Android* stellt Google auch ein eigenes API für die Verwendung von GM unter Android zur Verfügung. Mit diesem API ist es möglich Karten anzeigen zu lassen und mit diesen zu Interagieren. Google stellt für dieses API eine umfangreiche Dokumentation unter [He:Web42, Dokumentation des *Maps SDK for Android*] zur Verfügung.

### 15.5.3 Here Maps

HM stellt, wie OSM, kein eigenes Framework für die Integration mit JSF zur Verfügung. Da es sich bei HM um eine nicht quelloffene Software handelt stehen auch keine Frameworks von Drittherstellern zur Verfügung. Es besteht also nur die Möglichkeit über das JavaScript-API von HM zuzugreifen. Damit wird die Verwendung von HM mit JSF im Projekt erschwert.

Mit dem *HERE Android SDK* stellt HM ein eigenes API für die Verwendung von HM unter Android zur Verfügung. Die gebotene Funktionalität ist ähnlich zu der von Google. Daher stellt die Verwendung von HM unter Android im Projekt kein Problem dar.

### 15.5.4 Sygic Maps

Die Situation bei SM ist gleich wie bei HM. Es gibt keine Möglichkeit SM direkt in JSF einzubinden. Man kann hier nur über das JavaScript-API von SM zugreifen.

Unter Android stellt auch Sygic ein eigenes API zur Verfügung. Dieses ist ähnlich wie bei GM und HM aufgebaut. Aus diesem Grund ist die Verwendung von SM unter Android kein Problem.

### 15.5.5 Bewertung

Die Bewertung teilt sich in zwei Bereiche auf. Der erste bezieht sich auf die Integration mit JSF, der zweite auf die Verwendung in Android. Das Bewertungsschema wird nachstehend angeführt. Jeder Dienst erhält nach diesem Schema zwei Punkte, jeweils einen pro Bereich. Die Punkte werden mit einer Gewichtung zusammengezählt. Je höher die Gesamtpunkteanzahl desto besser.

Das Bewertungsschema sieht wie folgt aus:

- **2 Punkte**

Integration durch vorhandene Software (z.B. JSF, PrimeFaces oder API des Herstellers)

- **1 Punkt**

Integration durch Verwendung von Drittsoftware

- **0 Punkte**

Keine direkte Integration möglich

Bereich	Gewichtung	OSM		GM		HM		SM	
JSF	40%	1	40	2	80	0	0	0	0
Android	60%	1	60	2	120	2	120	2	120
<b>Gesamt</b>	<b>100%</b>	<b>100</b>		<b>200</b>		<b>120</b>		<b>120</b>	

Tabelle 15.2: Bewertung der Integration

Aus der Tabelle ergibt sich, dass OSM, unter Verwendung von Drittsoftware, in das Projekt integriert werden kann. GM kann im Projekt ohne Probleme verwendet werden. HM und SM bieten zwar beide ein Android API an, können jedoch nicht einfach in die JSF-Technologie integriert werden.

## 15.6 Verfügbarkeit

Wie bereits in den Anforderungen beschrieben, ist die im Projekt entwickelte Software essentiell für den reibungslosen Betrieb des Unternehmens. Deshalb muss die Software zu den Geschäftszeiten<sup>3</sup> des Unternehmens stets zur Verfügung stehen. Aus diesem Grund muss auch der im Projekt verwendete Kartendienst stets erreichbar sein. Im nachfolgenden werden die Verfügbarkeitsraten der jeweiligen Hersteller angeführt.

### 15.6.1 OpenStreetMap

Da es sich bei OSM um einen Dienst handelt, welcher freiwillig und kostenlos zur Verfügung gestellt wird, kann keine Verfügbarkeitsrate garantiert werden. Auf der OSM-Wiki ist unter [He:Web43, Kommerzielle Anbieter von OSM] eine Liste von Unternehmen zu finden, welche die Daten von OSM verwenden und kommerziell anbieten. Diese Unternehmen können Verfügbarkeitsraten für ihre Services angeben.

### 15.6.2 Google Maps

Google bietet für alle Dienste, die sie zur Verfügung stellen, eine gute Serverinfrastruktur. Dementsprechend hoch ist auch die Verfügbarkeit. Laut [He:Web44, Verfügbarkeitsrate von Google] weist Google insgesamt eine Verfügbarkeitsrate von 99,9% auf. Als ausfallreichster Dienst wird Google Maps für Handys mit 97,8% Verfügbarkeit angegeben. Das entspricht einer Ausfallzeit von acht Tagen im Jahr.

<sup>3</sup>9 Uhr bis 5 Uhr, Montag bis Freitag

### 15.6.3 Here Maps

HM gibt auf [He:Web32, Preismodell von Here Maps] eine Verfügbarkeit von 99,9% an. Diese gilt für die zu bezahlenden Pläne. Für den kostenlosen Plan wird keine Verfügbarkeitsrate garantiert. Da sich das Projekt im Umfang des kostenlosen Plans befindet, kann hier keine Verfügbarkeit garantiert werden.

### 15.6.4 Sygic Maps

SM gibt für seine Services keine Verfügbarkeitsrate an. Unter [He:Web45, Terms & Conditions von SM] wird unter Punkt 7 *Compensation* angegeben, dass der Benutzer bei Nichtverfügbarkeit von SM entschädigt wird. Konkret wird dem Benutzer, bei einer Verfügbarkeitsrate zwischen 99% und 99,9% in einem Monat, die Lizenz für SM für eine Woche kostenlos zur Verfügung gestellt. Liegt die Verfügbarkeitsrate in einem Monat unter 99%, so wird dem Benutzer die Lizenz für einen Monat kostenlos zur Verfügung gestellt.

### 15.6.5 Bewertung

Um die vier Dienste in der Kategorie *Verfügbarkeit* zu bewerten, werden Punkte von eins bis vier vergeben. Jede Punkteanzahl kann nur einmal vorkommen und je höher sie ist desto besser. Die Dienste werden wie folgt bewertet:

- **OpenStreetMap**

OSM wird mit der niedrigsten Punkteanzahl von einem Punkt bewertet. Begründet wird dies dadurch, dass der Dienst auf freiwilliger Basis zur Verfügung gestellt wird und deshalb keine Verfügbarkeit garantiert werden kann.

- **Google Maps**

GM wird mit vier Punkten bewertet. Google gilt allgemein als zuverlässiger Serviceanbieter. Dies zeigt auch der oben angeführte Verfügbarkeitstest.

- **Here Maps**

HM wird mit zwei Punkten bewertet. Es wird zwar für den kostenlosen Plan keine Verfügbarkeit garantiert, jedoch ist anzunehmen, dass auch die kostenlosen Dienste eine hohe Verfügbarkeit besitzen.

- **Sygic Maps**

SM wird mit drei Punkten bewertet. Die Situation ist ähnlich wie bei HM. SM bietet jedoch explizit eine Entschädigung an, sobald die Verfügbarkeit unter ein gewisses Level fällt.

Abschließend ist zu erwähnen, dass der Umfang des Projekts zu klein ist, um in der Kategorie *Verfügbarkeit* einen merkbaren Unterschied zu erzielen.

## 15.7 Auswahl

Im folgenden werden die Dienste bewertet. Zuerst werden die Teilbewertungen angeführt. Danach werden diese gewichtet und addiert. Die Gesamtpunkteanzahl ist die Endbewertung der Dienste. Der Dienst mit der höchsten Punkteanzahl wird im Projekt verwendet.

### 15.7.1 Teilergebnisse

Die folgende Tabelle zeigt die Ergebnisse der vier Teilbewertungen.

Kategorie	OSM	GM	HM	SM
Funktion	215	275	275	275
Preis	4	2	3	1
Integration	100	200	120	120
Verfügbarkeit	1	4	2	3

Tabelle 15.3: Teilbewertungen der Dienste

Um die Teilbewertungen in Relation zu setzen, werden die Punkte in den Kategorien *Funktion* und *Integration* durch Hundert dividiert. Die neue Punkteverteilung sieht damit wie folgt aus:

Kategorie	OSM	GM	HM	SM
Funktion	2,15	2,75	2,75	2,75
Preis	4	2	3	1
Integration	1	2	1,2	1,2
Verfügbarkeit	1	4	2	3

Tabelle 15.4: Teilbewertungen der Dienste in Relation

## 15.7.2 Endbewertung

In der nachfolgenden Tabelle sind die vier Kategorien und deren Gewichtung für die Gesamtbewertung eingetragen. Als Punkte dienen die Ergebnisse der Teilbewertungen. Die gewichteten Punkte werden summiert und dienen als endgültige Gesamtpunkteanzahl. Der Dienst, welcher die höchste Gesamtpunkteanzahl erreicht, wird im Projekt eingesetzt.

Kategorie	Gewichtung	OSM		GM		HM		SM	
Funktion	35%	2,15	75,25	2,75	96,25	2,75	96,25	2,75	96,25
Preis	20%	4	80	2	40	3	60	1	20
Integration	25%	1	25	2	50	1,2	30	1,2	30
Verfügbarkeit	20%	1	20	4	80	2	40	3	60
<b>Gesamt</b>	<b>100%</b>	200,25		<b>266,25</b>		226,25		206,25	

Tabelle 15.5: Gesamtbewertung der Dienste

## 15.7.3 Analyse

Der Dienst mit der höchsten Gesamtpunkteanzahl ist *Google Maps*. Er erzielte in den beiden höchst gewichteten Kategorien *Funktion* und *Integration* jeweils die höchstmögliche Teilbewertung.

An zweiter Stelle befindet sich *Here Maps*. Der Dienst erzielte ebenfalls die höchste Teilbewertung in der Kategorie *Funktion* musste jedoch in den restlichen Kategorien Einbußen hinnehmen. In der Kategorie *Preis* war Here Maps jedoch gegenüber Google Maps überlegen.

An dritter Stelle ist *Sygic Maps*. Auch dieser Dienst erreichte die maximale Punkteanzahl in der Kategorie *Funktion*. In Bezug auf Verfügbarkeit hatte nur Google Maps eine bessere Bewertung. In den anderen Kategorien konnte Sygic Maps jedoch nicht so gut abschneiden.

An vierter und letzter Stelle befindet sich *OpenStreetMap*. Der Unterschied der Gesamtpunkteanzahl zwischen OSM und Sygic Maps ist sehr gering. OSM konnte in der Kategorie *Preis* die Bestwertung erzielen, da es grundsätzlich kostenlos zur Verfügung gestellt wird. Jedoch war der Dienst in allen anderen Kategorien unterlegen.

# Kapitel 16

## Verwendung des ausgewählten Dienstes im Projekt

### 16.1 Die gmap-Komponente

Die *gmap*-Komponente wird nativ mit PrimeFaces mitgeliefert. Sie dient als Schnittstelle zwischen JSF und dem Google Maps API. Das bedeutet, dass sie die Daten, welche über JSF bereitgestellt werden, zur Laufzeit in JavaScript umwandelt und damit auf das API von Google zugreift. Damit ist es auch möglich, unabhängig von JSF und PrimeFaces, direkt über JavaScript auf die dargestellte Karte zuzugreifen.

#### 16.1.1 Die wichtigsten Attribute

Die Liste der Attribute wurde aus [He:Web46, PrimeFaces User Guide] entnommen. Der User Guide dient grundsätzlich als Quelle für die nachfolgenden Abschnitte.

##### **widgetVar**

Über dieses Attribut kann der clientseitige Name für die JavaScript-Umgebung festgelegt werden. Dies ermöglicht unter anderem den direkten Zugriff auf das Google Maps API. Der Zugriff funktioniert wie folgt (Beispiel aus dem User Guide übernommen):

```
1 var gmap = PF('yourWidgetVar').getMap();  
2 //gmap is a google.maps.Map instance
```

Listing 16.1: use\_widgetvar.js

**model**

Dieses Attribut wird verwendet, um der Karte ein Modell hinzufügen zu können. Dieses beinhaltet Marker, Linien, Polygone etc. Der Attributwert beinhaltet den Zugriff auf ein Objekt vom Typ *org.primefaces.model.MapModel*. Dieses muss in einem JavaBean definiert sein. Der Zugriff erfolgt über die *ExpressionLanguage*<sup>1</sup> von JSF. Wie ein Modell erstellt werden kann ist im Verwendungsbeispiel ersichtlich.

**type**

Gibt an, in welcher Ansicht die Karte dargestellt werden soll. Die verfügbaren Typen sind:

- **ROADMAP**

Dies ist die Standardeinstellung bei Google Maps. Es werden keine Satellitenbilder angezeigt.

- **SATELLITE**

Hier wird anstelle der schematischen Kartendarstellung die Umgebung mit Satellitenbildern dargestellt. Es werden jedoch keine Informationen wie Straßennamen, POIs und ähnliches angezeigt.

- **HYBRID**

Diese Einstellung kombiniert die bereits genannten Ansichten.

- **TERRAIN**

Diese Ansicht stellt die selben Informationen wie *ROADMAP* dar. Zusätzlich werden Höhenunterschiede eingezeichnet.

**UI-Attribute**

Die *gmap*-Komponente definiert vier Attribute, mit denen das Aussehen beziehungsweise die Funktionalität verändert werden kann. Die Attribute mit ihren Funktionen lauten wie folgt:

- **streetView**

Definiert, ob die StreetView-Ansicht vom Benutzer aktiviert werden kann.

- **disableDefaultUI**

Definiert, ob das Default-UI sichtbar ist oder nicht. Dazu gehören der Button um in den Vollbildmodus zu wechseln, sowie die Zoombuttons.

- **navigationControl**

Dieses Attribut hat aktuell<sup>2</sup> keine Funktionalität. Unter [He:Web47, Dokumentation des Google Maps APIs] gibt es keinen Eintrag zu einem *navigation Control*.

<sup>1</sup>spezielle Syntax, um auf Javaobjekte in xhtml-Dateien zugreifen zu können

<sup>2</sup>17.02.2019

- **mapTypeControl**

Definiert, ob der Button, mit dem die Kartenansicht geändert werden kann, angezeigt werden soll.

### 16.1.2 Verknüpfung mit dem GM-API

Um die *gmap*-Komponente verwenden zu können muss folgendes Skript in der jeweiligen xhtml-Seite aufgenommen werden:

```
1 <script
2   type="text/javascript"
3   src="https://maps.google.com/maps/api/js?key=API_KEY"></script>
```

Listing 16.2: use\_gmap\_script.html

Anstelle von *API\_KEY* ist ein gültiger Google Maps API-Key einzusetzen. In allen folgenden Beispielen wird der Projekt-Key verwendet. Dieser ist in den hier abgebildeten Listings jedoch nicht ersichtlich. Wird kein (gültiger) Key angegeben, so ist die Funktionalität eingeschränkt. Weiters wird ein Wasserzeichen eingeblendet, wie im folgenden Bild zu sehen ist.

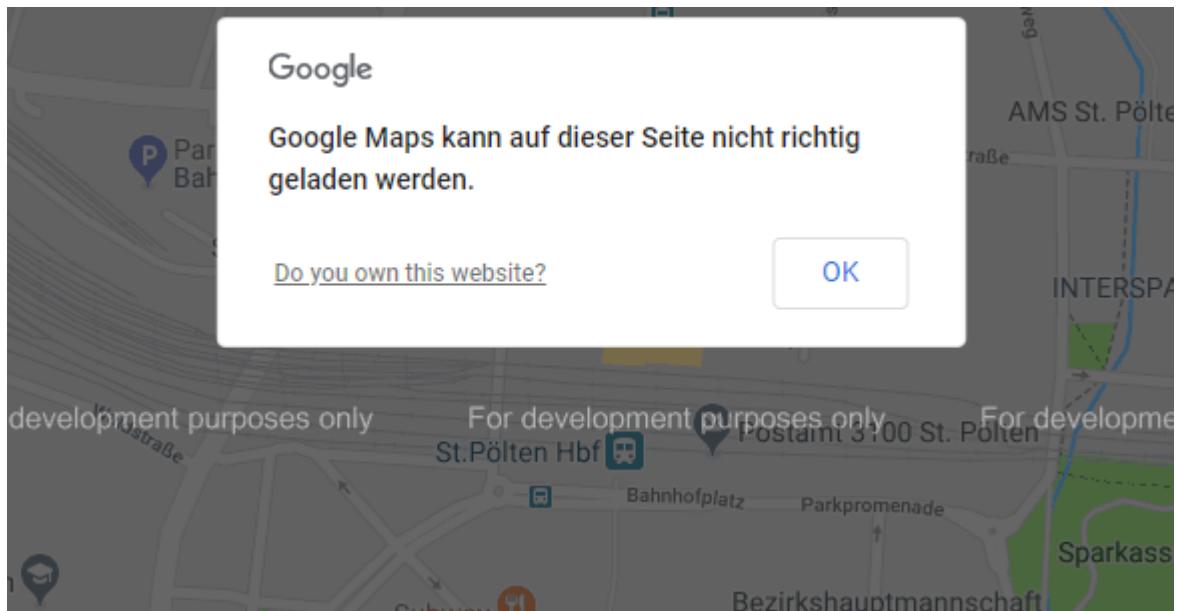


Abbildung 16.1: Verwendung der *gmap*-Komponente ohne API-Key

### 16.1.3 Verwendungsbeispiel

In diesem Beispiel wird eine Übersicht über die Möglichkeiten der *gmap*-Komponente gezeigt. Der Fokus liegt dabei auf der Interaktion zwischen dem Benutzer und der Karte sowie auf der Darstellung von Markern auf der Karte. Für die Erstellung dieses Beispiels wurde die JSF-Version 2.2 und die PrimeFaces-Version 5.0 verwendet. Als JavaEE-Server wird GlassFish in der Version 4.1.1 verwendet.

#### Projektstruktur

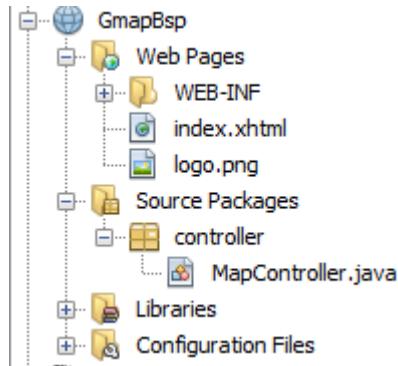


Abbildung 16.2: *gmap*-Beispiel - Projektstruktur

Das Projekt besteht einerseits aus der `index.xhtml` Datei, welche die eigentliche *gmap*-Komponente beinhaltet und andererseits aus der `MapController.java` Datei, welche die Programmlogik beinhaltet.

#### Die `index.xhtml` Datei

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3   xmlns:h="http://xmlns.jcp.org/jsf/html"
4   xmlns:p="http://primefaces.org/ui">
5   <h:head>
6     <title>Facelet Title</title>
7   </h:head>
8   <h:body>
9     <h:form>
10    <p:growl id="growl" life="2000" />
11    <script type="text/javascript"
12      src="https://maps.google.com/maps/api/js?key=API_KEY"></script>
13
14    <p:gmap id="map" center="48.2048547, 15.6244005"
15      zoom="16"
16      type="ROADMAP"
17      style="top:0px;left:0px;position:absolute;width:100%;height:100%">
```

```

18    model="#{mapCon.model}"
19    disableDefaultUI="true">
20
21    <!-- click -->
22    <p:ajax event="overlaySelect"
23      listener="#{mapCon.onMarkerSelect}"
24      update="growl" />
25
26    <!-- drag -->
27    <p:ajax event="markerDrag"
28      listener="#{mapCon.onMarkerDrag}"
29      update="growl" />
30  </p:gmap>
31  </h:form>
32 </h:body>
33 </html>

```

Listing 16.3: gmap\_bsp-index.xhtml

Um PrimeFaces verwenden zu können muss es über einen XML-Namespace importiert werden. Dies geschieht hier in Zeile vier. Das Einbinden des Google Maps-API erfolgt in Zeile 11. Darunter ist die *gmap*-Komponente definiert.

Beim Laden der Seite ist die Karte auf die angegebenen Koordinaten (St. Pölten) zentriert. Um die Karte übersichtlicher zu machen wurde als Ansichtstyp *ROADMAP* verwendet und das Standard-UI ausgeblendet. Über das *style*-Attribut wurde die Karte auf die Größe des Browserfensters gesetzt.

Über das *model*-Attribut wurde die *gmap*-Komponente mit dem im Controller definierten Modell verknüpft. Weiters sind zwei Listener definiert. Der erste führt eine Methode im Controller aus, sobald ein Marker auf der Karte selektiert wird. Der zweite Listener reagiert auf das Verschieben von Markern.

## Der MapController

```

1 package controller;
2
3 // imports
4
5 @Named("mapCon")
6 @SessionScoped
7 public class MapController implements Serializable {
8
9     private MapModel model = new DefaultMapModel();
10
11    @PostConstruct
12    public void init() {
13        model.addOverlay(new Marker(new LatLng(48.2070236, 15.6183001), "HTL

```

```

14     STP", "HTL St. Pölten", "/GMap_Referat/logo.png"));
15     Marker ma = new Marker(new LatLng(48.207858, 15.6239906), "STP Hbf", "St. Pölten Hauptbahnhof");
16     ma.setDraggable(true);
17     model.addOverlay(ma);
18 }
19
20 public MapModel getModel() {
21     return model;
22 }
23
24 public void setModel(MapModel model) {
25     this.model = model;
26 }
27
28 public void onMarkerSelect(OverlaySelectEvent e) {
29     if (!(e.getOverlay() instanceof Marker)) {
30         return;
31     }
32     Marker ma = (Marker) e.getOverlay();
33     FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("Clicked: " + ma.getData()));
34 }
35
36 public void onMarkerDrag(MarkerDragEvent e) {
37     Marker ma = e.getMarker();
38     FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("Dragged: " + ma.getData()));
39 }
40 }
41 }
```

Listing 16.4: MapController.java

Im Controller befindet sich eine Instanz der Klasse *DefaultMapModel*. Diese wird der xhtml-Seite mittels Getter und Setter zur Verfügung gestellt. Beim Erzeugen des Controllers werden in der `init()`-Methode zwei Marker erstellt. Hierfür wird folgender Konstruktor verwendet:

```
public Marker(LatLng latlng, String title, Object data, String icon);
```

*latlng* beinhaltet die Koordinaten des Markers. *title* ist jener Text, welcher angezeigt wird, wenn man mit der Maus über den Marker fährt. Mit dem *data*-Parameter kann der Marker intern Zusatzinformationen kapseln, welche vom Benutzer nicht einlesbar sind. Über den *icon*-Parameter kann dem Marker ein eigenes Anzeigebild zugewiesen werden.

**Hinweis:** Da die *gmap*-Komponente nur das Google Maps-API kapselt muss als Icon-Pfad der gesamte URL-Pfad angegeben werden, inklusive Context-Pfad!

Der zweite Marker wird explizit als verschiebbar gekennzeichnet. Als letztes werden zwei Methoden deklariert, mit denen auf die jeweiligen Events reagiert werden kann. In beiden Methoden wird lediglich das im Marker gekapselte *data*-Objekt ausgegeben. Dazu wird eine globale *FacesMessage* erstellt, welche auf der Website über die *growl*-Komponente angezeigt wird.

### Die Oberfläche



Abbildung 16.3: *gmap*-Beispiel - Oberfläche

Im Bild sind die beiden Marker zu sehen. Dabei ist auch das HTL-Logo als eigenes Markericon sichtbar. Beim Klicken auf einen der beiden Marker wird folgende Meldung rechts oben angezeigt:

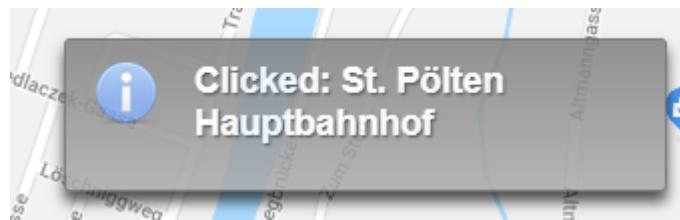


Abbildung 16.4: *gmap*-Beispiel - Klick auf einen Marker

Durch Klicken und Ziehen kann ein Marker verschoben werden, sofern dies beim Erstellen des Markers aktiviert wurde. Da der *MapController* in diesem Beispiel für jede User-Session angelegt wird, bleiben die geänderten Markerpositionen nach erneutem laden der Seite erhalten. In diesem Beispiel ist nur der Marker des Hauptbahnhofs verschiebbar. Im folgenden Bild ist die Nachricht des Eventhandlers sowie die neue Position des Markers erkennbar.

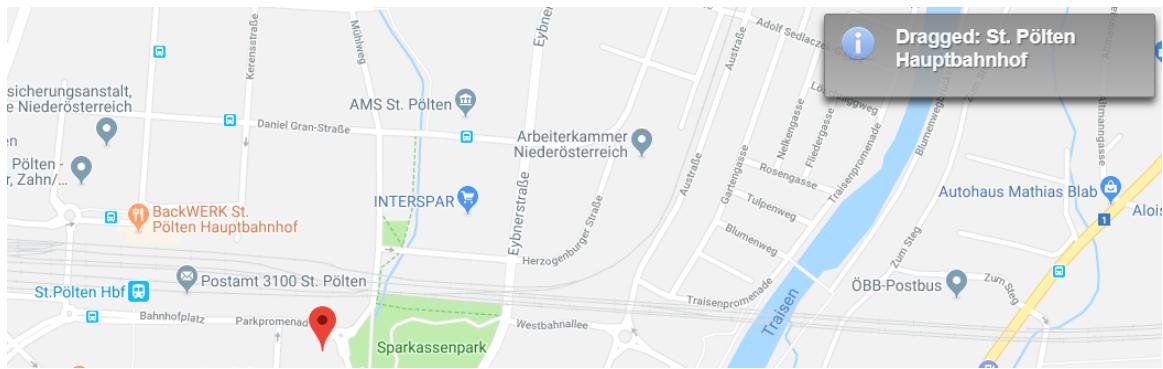


Abbildung 16.5: gmap-Beispiel - Verschiebung eines Markers

## 16.2 Verwendung der gmap-Komponente im Projekt

Die genauen Anforderungen wurden bereits in der Sektion 15.2.1 auf Seite 182 beschrieben. In dieser Sektion werden die konkreten Verwendungsgebiete im Projekt gezeigt und erklärt.

### 16.2.1 Die Oberfläche

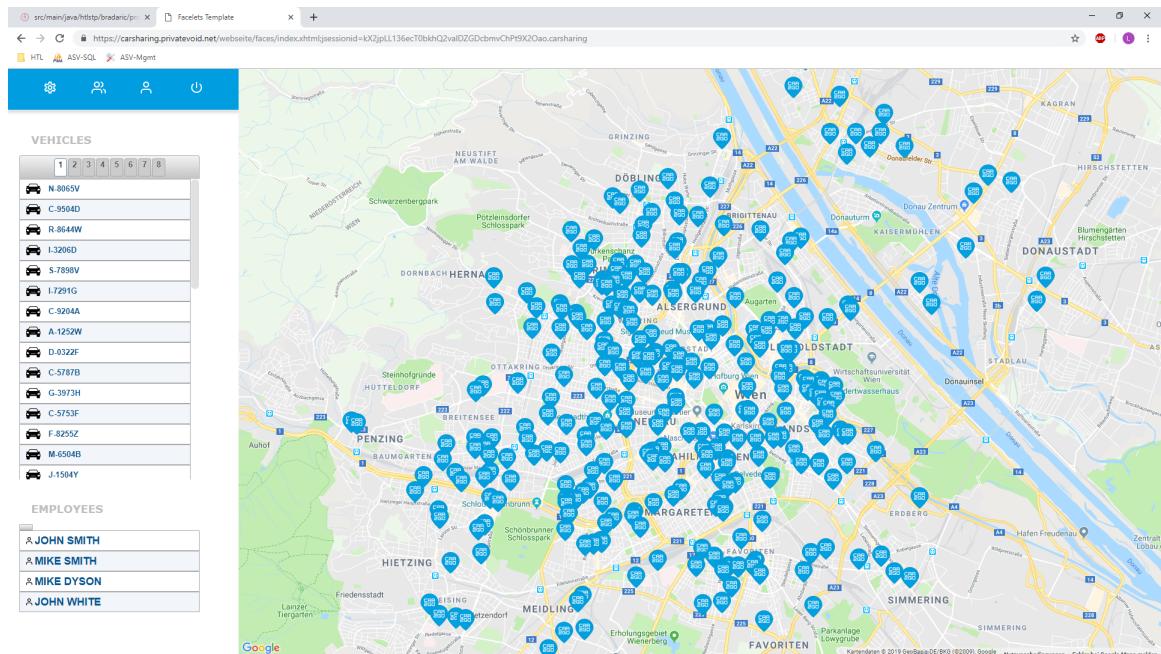


Abbildung 16.6: Website - Übersicht

Das oben dargestellte Bild zeigt die Hauptseite der Website. Der wichtigste Bestandteil ist die *gmap*-Komponente. Auf der Karte werden die aktuellen Positionen der Fahrzeuge mittels Marker angezeigt. Wird ein Marker durch Klicken ausgewählt, so wird ein Dialog angezeigt, in welchem dem ausgewählten Fahrzeug ein Mitarbeiter zugewiesen werden kann.

Die Einbindung der *gmap*-Komponente ist dem folgenden Listing zu entnehmen:

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3   xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
4   xmlns:h="http://java.sun.com/jsf/html"
5   xmlns:f="http://java.sun.com/jsf/core"
6   xmlns:p="http://primefaces.org/ui">
7
8 <h:head>
9   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
10  <h:outputStylesheet name="/css/default.css"/>
11  <h:outputStylesheet name="/js/jsDialog.js"/>
12  <h:outputStylesheet name="/css/cssLayout.css"/>
13  <title>Facelets Template</title>
14 </h:head>
15
16 <h:body>
17  <!-- Sidebar -->
18
19  <div id="content" class="left_content">
20    <p:gmap id="map"
21      center="48.220778, 16.3100205"
22      zoom="15"
23      type="ROADMAP"
24      style="width:100%;height:100%;"
25      widgetVar="map"
26      disableDefaultUI="true"
27      disableDoubleClickZoom="true"
28      streetView="false"
29      styleClass="gmapStyle"
30      model="#{Mcon.simpleModel}">
31      <p:ajax event="overlaySelect" listener="#{Mcon.onMarkerSelect}"
32          update="dlgV"/>
33      <script type="text/javascript" src="https://maps.google.com/maps/api/
34          js?key=API_KEY"></script>
35    </p:gmap>
36  </div>
37
38  <!-- Dialoge -->
39 </h:body>
40 </html>
```

Listing 16.5: main.xhtml

Die Karte verwendet als Ansichtstyp *ROADMAP* und ist beim Laden auf Wien zentriert. Für eine bessere Übersicht ist das Standard-UI deaktiviert. Als Modell wird eine *DefaultMapModel*-Instanz im *MarkersView*-Controller verwendet. Wird ein Marker selektiert, so wird im selben Controller die Methode `onMarkerSelect()` aufgerufen.

### 16.2.2 Der Controller

Im folgenden Listing ist der Controller abgebildet. Dieser beinhaltet die Erzeugung und Verwaltung des *MapModels* sowie die Listener-Methode für das *overlaySelect*-Event. Für die Beschaffung der Fahrzeugdaten wird ein Webservice verwendet, welches ebenfalls im Projekt entwickelt wurde. Dieses Service ist mit einem Key geschützt. Der Key ist im Listing nicht sichtbar und wurde durch den Text *REST\_KEY* ersetzt.

```
1 package controller;
2
3 // imports
4
5 @Named("Mcon")
6 @SessionScoped
7 public class MarkersView implements Serializable {
8
9     private MapModel simpleModel;
10    private CarCurrent currentCar;
11
12    @PostConstruct
13    public void init() {
14        CarClient client = GenericService.getClient(CarClient.class, "REST_KEY"
15            );
16        Call<List<CarCurrent>> c = client.getCars();
17        simpleModel = new DefaultMapModel();
18        try {
19            List<CarCurrent> cars = c.execute().body();
20            for (CarCurrent ca : cars) {
21                LatLng coord1 = new LatLng(ca.getLatitude().doubleValue(), ca.getLongitude()
22                    .doubleValue());
23                Marker m = new Marker(coord1, ca.getCar().getVin(), ca, "resources/
24                    images/marker-car.png");
25                simpleModel.addOverlay(m);
26            }
27        } catch (IOException ex) {
28            LatLng coord5 = new LatLng(0, 0);
29            simpleModel.addOverlay(new Marker(coord5, ex.getMessage()));
30            ex.printStackTrace();
31        }
32    }
33}
```

```

31 public MapModel getSimpleModel() {
32     return simpleModel;
33 }
34
35 public CarCurrent getCurrentCar() {
36     return currentCar;
37 }
38
39 public void onMarkerSelect(OverlaySelectEvent event) {
40     marker = (Marker) event.getOverlay();
41     this.currentCar = (CarCurrent) marker.getData();
42
43     RequestContext.getCurrentInstance().execute("PF('dialogVehicle').show()
44         ;");
45 }
```

Listing 16.6: MarkersView.java

In der `init()` Methode werden alle verfügbaren Fahrzeuge über das Webservice geladen. Für jedes Fahrzeug wird ein Marker erstellt. Als Titel dient die Fahrzeug-VIN<sup>3</sup>. Das gesamte Fahrzeug-Objekt wird im Marker gekapselt. Weiters werden die Marker mit einem eigenen Icon ausgestattet. Sollte ein Fehler auftreten, so wird zur Signalisierung ein Marker an den Koordinaten (0, 0) mit der Fehlermeldung als Titel angezeigt.

In der `onMarkerSelect()` Methode wird auf das `overlaySelect`-Event reagiert. Dabei wird das ausgewählte Fahrzeug über die Methode `getData()` aus dem Marker geladen und in einer Instanzvariable gespeichert. Danach wird ein Dialog angezeigt, in welchem die Daten des ausgewählten Fahrzeugs zu sehen sind.

## 16.3 Maps SDK for Android

Die genaue Dokumentation ist unter [He:Web42, Dokumentation der Maps SDK for Android] zu finden. In dieser Sektion wird die grundlegende Funktionsweise des SDKs beschrieben. Für die Verwendung wird ein Google-Konto mit gültigem API-Key benötigt.

### 16.3.1 Einfaches Beispiel

Dieses Beispiel ist der offiziellen Dokumentation entnommen und beinhaltet die Basiskonfiguration für jede Android-Activity, welche eine Google Maps-Karte beinhaltet.

---

<sup>3</sup>Vehicle Identification Number

Wird die Applikation mit Android Studio erstellt, so wird die Konfiguration fast zur Gänze automatisch durchgeführt.

## Die Layoutdatei

```
1 <fragment xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   android:id="@+id/map"
6   tools:context=".MapsActivity"
7   android:name="com.google.android.gms.maps.SupportMapFragment" />
```

Listing 16.7: activity\_maps.xml

Die zwei wichtigsten Attribute sind `android:id` und `tools:context`. Das erste Attribut legt fest, unter welchem Namen die Karte im Java-Code angesprochen werden kann. Das zweite, in welchem Kontext (Activity) auf die Karte zugegriffen werden kann.

## Die Activity

```
1 import android.os.Bundle;
2 import android.support.v4.app.FragmentActivity;
3 import com.google.android.gms.maps.CameraUpdateFactory;
4 import com.google.android.gms.maps.GoogleMap;
5 import com.google.android.gms.maps.OnMapReadyCallback;
6 import com.google.android.gms.maps.SupportMapFragment;
7 import com.google.android.gms.maps.model.LatLng;
8 import com.google.android.gms.maps.model.MarkerOptions;
9
10 public class MapsActivity extends FragmentActivity implements
11     OnMapReadyCallback {
12
13     private GoogleMap mMap;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_maps);
19         SupportMapFragment mapFragment = (SupportMapFragment)
20             getSupportFragmentManager()
21             .findFragmentById(R.id.map);
22         mapFragment.getMapAsync(this);
23     }
24
25     @Override
26     public void onMapReady(GoogleMap googleMap) {
27         mMap = googleMap;
```

```

26
27     // Add a marker in Sydney, Australia, and move the camera.
28     LatLng sydney = new LatLng(-34, 151);
29     mMap.addMarker(new MarkerOptions().position(sydney).title("Marker
30         in Sydney"));
31     mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
32 }
```

Listing 16.8: MapsActivity.java

In der Methode `onCreate()`, welche bei jeder Erzeugung der Activity aufgerufen wird, wird auf das SupportMapFragment mittels der zuvor festgelegten Id zugegriffen. Über diese wird die eigentliche Karte über den Aufruf der Methode `getMapAsync()` geladen. Dabei wird die Activity selbst als Callback-Klasse angegeben und muss dafür das Interface `OnMapReadyCallback` implementieren.

In der Methode `onMapReady()` wird die geladene Google Maps Instanz in einer Instanzvariable gespeichert. Weiters wird ein Marker mit dem Titel "Marker in Sydney" angelegt und in Sydney platziert. Als letztes wird auch das Zentrum der Karte auf die selben Koordinaten gesetzt.

### 16.3.2 Zeichnen von Routen

Für die einfache Verwendung des *Directions-API* muss folgendes in der Projektkonfiguration aufgenommen werden:

```

1 dependencies {
2     compile 'com.google.maps.android:android-maps-utils:0.5+'
3 }
```

Listing 16.9: Einbinden des Direction-APIs

Dadurch werden Utility-Klassen für das Maps SDK zur Verfügung gestellt. Eine dieser Klassen ist die Klasse *PolyUtil*. Sie stellt die Methode `decode()` zur Verfügung, mit welcher die Berechnungsergebnisse des *Directions-API* in Polylines<sup>4</sup> umgewandelt werden können.

Um auf das *Directions-API* zugreifen zu können, muss in der Activity eine Methode geschrieben werden, in der ein *GeoContext* erzeugt wird. Sie über gibt ihm den benötigten

<sup>4</sup>Liste von Koordinaten, welche eine Linie bilden

API-Key und kann noch weitere Einstellungen vornehmen. Nun kann mit folgendem Code eine Route berechnet und angezeigt werden:

```

1 private GeoApiClientContext getGeoContext() {
2     GeoApiClientContext geoApiClientContext = new GeoApiClientContext();
3     return geoApiClientContext.setQueryRateLimit(3).setApiKey(getString(R.string.
4         google_maps_key))
5     .setConnectTimeout(1, TimeUnit.SECONDS)
6     .setReadTimeout(1, TimeUnit.SECONDS)
7     .setWriteTimeout(1, TimeUnit.SECONDS);
8 }
9
10 public void makeRoute() {
11     DateTime now = new DateTime();
12     try {
13         LatLng htl = new LatLng(48.206112, 15.618411);
14         LatLng bahnhof = new LatLng(48.207607, 15.624878);
15
16         DirectionsResult result = DirectionsApi.newRequest(getGeoContext())
17             .mode(TravelMode.DRIVING).origin(htl)
18             .destination(bahnhof).departureTime(now)
19             .await();
20
21         List<LatLng> decodedPath = PolyUtil.decode(result.routes[0].
22             overviewPolyline.getEncodedPath());
23         gmap.addPolyline(new PolylineOptions().addAll(decodedPath).color(Color.
24             rgb(0, 0, 0)));
25     } catch (Exception e) {
26         e.printStackTrace();
27     }
28 }
```

Listing 16.10: DrawRoute.java

Hier wird eine Route zwischen der HTL St. Pölten und dem Hauptbahnhof berechnet. Als Fortbewegungsmethode wurde Autofahren eingestellt. Nachdem die Route berechnet wurde, wird sie in eine Polyline umgewandelt und auf der Karte angezeigt.

### 16.3.3 Verwendung von Geofences

Geofences werden verwendet, um überprüfen zu können, ob sich der Benutzer in einem bestimmten Gebiet befindet. Google stellt diese Funktionalität über ihr *Location-Services-API* zur Verfügung. Eine genaue Beschreibung des APIs ist unter [He:Web48, Dokumentation des LS-API] zu finden.

Um Geofences in einer Android App verwenden zu können, muss eine Berechtigung

für den Standortzugriff angefordert werden. Die offizielle Dokumentation empfiehlt dazu die Berechtigung `ACCESS_FINE_LOCATION`. Ein Geofence kann nun wie folgt erzeugt werden:

```
1 geofenceList.add(new Geofence.Builder()
2     // Set the request ID of the geofence. This is a string to identify this
3     // geofence.
4     .setRequestId(entry.getKey())
5
6     .setCircularRegion(
7         entry.getValue().latitude,
8         entry.getValue().longitude,
9         Constants.GEOFENCE_RADIUS_IN_METERS
10    )
11    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
12    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
13        Geofence.GEOFENCE_TRANSITION_EXIT)
14    .build());
```

Listing 16.11: CreateGeofence.java

Das obige Beispiel wurde der offiziellen Dokumentation entnommen. Dabei wird ein Geofence angelegt, dessen ID und Position über ein im Listing nicht sichtbares `entry`-Objekt festgelegt wird. Weiters wird der Geofence so konfiguriert, dass er sowohl auf den Eintritt, als auch auf den Austritt aus dem markierten Bereich reagiert.

Um auf einen Eintritt in/Austritt aus einem Geofence zu reagieren wird ein Intent-Service benötigt. Dieser muss in der Manifest-Datei registriert werden. Im folgenden Beispiel wird das Event nur im Log ausgegeben und nicht weiter behandelt:

```
1 public class GeofenceTransitionsIntentService extends IntentService {
2     // ...
3     protected void onHandleIntent(Intent intent) {
4         GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
5         if (geofencingEvent.hasError()) {
6             String errorMessage = GeofenceErrorMessages.getErrorString(this,
7                 geofencingEvent.getErrorCode());
8             Log.e(TAG, errorMessage);
9             return;
10        }
11
12        // Get the transition type.
13        int geofenceTransition = geofencingEvent.getGeofenceTransition();
14
15        // Test that the reported transition was of interest.
16        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
17            geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {
```

```
19     // Get the geofences that were triggered. A single event can trigger
20     // multiple geofences.
21     List<Geofence> triggeringGeofences = geofencingEvent.
22         getTriggeringGeofences();
23
24     // Get the transition details as a String.
25     String geofenceTransitionDetails = getGeofenceTransitionDetails(
26         this,
27         geofenceTransition,
28         triggeringGeofences
29     );
30
31     // Log the transition details.
32     Log.i(TAG, geofenceTransitionDetails);
33 } else {
34     // Log the error.
35     Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
36         geofenceTransition));
37 }
38 // ...
39 }
```

Listing 16.12: GeofenceTransitionsIntentService.java

Zuerst wird überprüft, ob ein Fehler aufgetreten ist. Wenn ja, wird dieser im Log festgehalten und die Ausführung der Methode beendet. Ansonsten wird überprüft, um welchen Übergang es sich handelt. Dadurch kann auf einen Eintritt in einen Geofence anders reagiert werden als auf einen Austritt. Zum Schluss werden die Details zum Event geladen und im Log festgehalten.

## 16.4 Verwendung des Maps SDKs im Projekt

Die genauen Anforderungen wurden bereits in der Sektion 15.2.2 auf Seite 182 beschrieben. In dieser Sektion werden die konkreten Verwendungsgebiete im Projekt gezeigt und erklärt.

### 16.4.1 Gliederung der App

Die mobile Applikation benötigt in folgenden Activities eine Karte:

- **Main**

Beinhaltet eine Google Maps Karte, auf der die Position des Benutzers, sowie der Fahrzeuge anzeigen werden.

- **AssignCar**

Hier wird neben dem Benutzer nur das ausgewählte Fahrzeug angezeigt. Weiters wird die Route zwischen den beiden Markern eingezeichnet. Durch einen langen Klick kann die Destination des Fahrzeuges eingegeben werden.

- **CurTask**

Diese Activity ist ähnlich zur *AssignCar*-Activity. Hier kann jedoch keine Destination gesetzt werden, da sie schon vorhanden ist.

### 16.4.2 Die Main-Activity

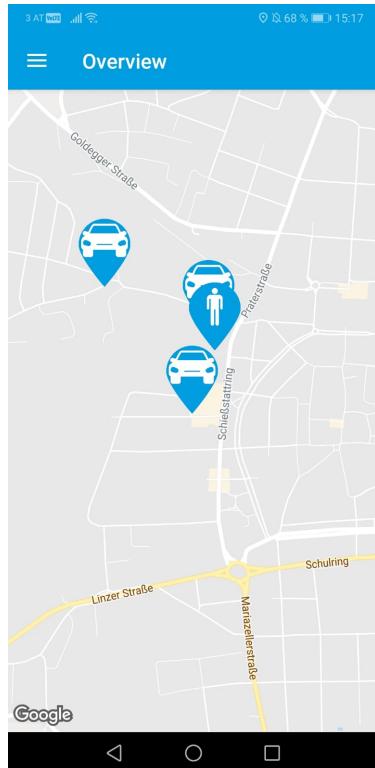


Abbildung 16.7: App - Übersicht

Nachdem sich der Benutzer in der App eingeloggt hat, wird ihm diese Activity angezeigt. Sie dient als Übersicht, über die vorhandenen Fahrzeuge im System. Wird auf einen Fahrzeug-Marker geklickt, so wird man auf die *AssignCar*-Activity weitergeleitet. Wurde dem Benutzer bereits ein Fahrzeug zugewiesen, so wird nur dieses Fahrzeug angezeigt. Beim Klick auf den Fahrzeug-Marker wird man auf die *CurTask*-Activity weitergeleitet.

```

1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main_drawer);
5
6      // ...
7
8      mGoogleApiClient = new GoogleApiClient.Builder(this)
9          .addConnectionCallbacks(this)
10         .addOnConnectionFailedListener(this)
11         .addApi(LocationServices.API)
12         .build();
13
14      mLocationRequest = LocationRequest.create()
15          .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
16          .setInterval(5 * 1000)    // 10 seconds, in milliseconds
17          .setFastestInterval(1 * 1000); // 1 second, in milliseconds
18
19      SupportMapFragment mapFragment = (SupportMapFragment)
20          getSupportFragmentManager()
21          .findFragmentById(R.id.mapView);
22      mapFragment.getMapAsync(this);
23
24      BitmapDrawable bitmapdraw = (BitmapDrawable) ResourcesCompat.getDrawable(
25          getResources(), R.drawable.person_marker, null);
26      Bitmap b = bitmapdraw.getBitmap();
27      smallMarkerPerson = Bitmap.createScaledBitmap(b, 200, 200, false);
28  }

```

Listing 16.13: Main\_drawer.java onCreate()

In der `onCreate()` Methode wird das *LocationServices*-API initialisiert. Es wird benötigt, um auf den Standort des Benutzers zugreifen zu können. Die Callback-Methode befindet sich ebenfalls in der Activity und wird später behandelt. Als letztes wird eine Google Maps-Instanz erzeugt und das Marker-Icon geladen.

```

1  @Override
2  public void onMapReady(GoogleMap googleMap) {
3      if (googleMap == null) {
4          Log.e(TAG, "GOOGLEMAP IS NULL");
5          return;

```

```

6      }
7
8      gmap = googleMap;
9      gmap.setOnMarkerClickListener(this);
10
11     // ...
12
13    for (CarCurrent c : DataBean.getCarList()) {
14        LatLng pos = new LatLng(c.getLatitude().doubleValue(), c.getLongitude().
15            doubleValue());
16        Marker m = gmap.addMarker(new MarkerOptions()
17            .position(pos)
18            .title(c.getCar().getVin())
19            .icon(BitmapDescriptorFactory.fromBitmap(smallMarker)));
20        m.setTag(c);
21    }
}

```

Listing 16.14: Main\_drawer.java onMapReady()

In dieser Methode wird die Google Maps-Instanz gespeichert und der *OnMarkerClickListener* registriert. Weiters werden die Fahrzeuge zur Karte hinzugefügt. Unter anderem kapseln die Marker die jeweiligen Fahrzeug-Objekte.

```

1 private void handleNewLocation(Location location) {
2     if (gmap == null) {
3         Log.e(TAG, "GOOGLEMAP IS NULL HANDLENEWLOC");
4         return;
5     }
6
7     if (userMarker != null) {
8         userMarker.remove();
9     }
10
11    // UserPosition am Server updaten
12    double currentLatitude = location.getLatitude();
13    double currentLongitude = location.getLongitude();
14    LatLng latLng = new LatLng(currentLatitude, currentLongitude);
15    userMarker = gmap.addMarker(new MarkerOptions()
16        .position(latLng).title("User")
17        .icon(BitmapDescriptorFactory.fromBitmap(smallMarkerPerson)));
18}
19
20@Override
21public void onLocationChanged(Location location) {
22    Log.i(TAG, "LOCATION CHANGED MAIN");
23    handleNewLocation(location);
24}

```

Listing 16.15: Main\_drawer.java handleNewLocation()

Wird eine neue Position vom *LocationServices-API* erfasst, so wird diese an die Methode `onLocationChanged()` gesendet. Diese ruft die `handleNewLocation()` Methode auf. In dieser wird der User-Marker, falls dieser bereits vorhanden ist, entfernt. Danach wird ein Marker an der neuen Position gesetzt.

### 16.4.3 Die AssignCar-Activity

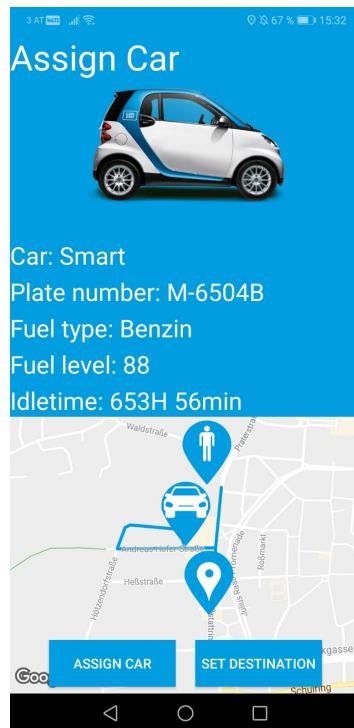


Abbildung 16.8: App - Auftragszuweisung

Die `onMapReady()` Methode besitzt grundsätzlich die selbe Funktionalität wie in der Main-Activity. Anstelle eines *OnMarkerClickListener* wird nun ein *OnMapLongClickListener* verwendet. Auch die `handleNewLocation()` Methode wurde nur geringfügig verändert. Nach dem Update der User-Position wird nun auch die `makeRoute()` Methode aufgerufen.

Wie bereits in der Sektion 16.3.2 auf Seite 205 beschrieben, wird ein GeoContext für das Berechnen der Routen benötigt. Je nachdem, ob eine Destination gesetzt ist oder nicht, werden zwei verschiedene Methoden für die Berechnung der Route(n) aufgerufen. Im folgenden Listing ist jene Methode zu sehen, welche die Destination berücksichtigt.

```

1 public void makeRouteWithDest() {
2     DateTime now = new DateTime();
3     try {
4         LatLng userloc = new LatLng(userMarker.getPosition().latitude,
5             userMarker.getPosition().longitude);
6         LatLng carPos = new LatLng(carMarker.getPosition().latitude, carMarker.
7             getPosition().longitude);
8
9         DirectionsResult resultCar = DirectionsApi.newRequest(getGeoContext())
10            .mode(TravelMode.DRIVING).origin(userloc)
11            .destination(carPos).departureTime(now)
12            .await();
13
14         DirectionsResult resultDest = DirectionsApi.newRequest(getGeoContext())
15            .mode(TravelMode.DRIVING).origin(carPos)
16            .destination(dest).departureTime(now)
17            .await();
18         // add polylines
19     } catch (ApiException | InterruptedException | IOException e) {
20         e.printStackTrace();
21     }
22 }
```

Listing 16.16: AssignCar.java makeRouteWithDest()

Zuerst wird die Route vom Benutzer zum ausgewählten Fahrzeug berechnet. Danach von diesem zur Destination. Die beiden Routen werden danach mittels Polylines auf der Karte dargestellt.

Um die Destination zu setzen wird ein *OnMapLongClickListener* verwendet. Dieser hat folgende Funktionalität.

```

1 @Override
2 public void onMapLongClick(LatLng latLng) {
3     if(!fixed) {
4         dest = new com.google.maps.model.LatLng(latLng.latitude, latLng.
5             longitude);
6         if (gmap != null) {
7             if (destMarker != null) {
8                 destMarker.remove();
9             }
10            destMarker = gmap.addMarker(new MarkerOptions()
11                .position(latLng)
12                .title("Dest")
13                .icon(BitmapDescriptorFactory.fromBitmap(destIcon)));
14            makeRouteWithDest();
15        } else{
16    }}
```

```

16    Toast.makeText(this, "Destination already set", Toast.LENGTH_LONG).show()
17
18 }

```

Listing 16.17: AssignCar.java onMapLongClick()

Sollte die Destination noch nicht gesetzt worden sein, so werden die Koordinaten gespeichert und die Routen neu gezeichnet. Die explizite Angabe des Paketpfades in Zeile vier ist notwendig, da es sich um zwei gleichnamige Klassen in verschiedenen Pakten handelt.

#### 16.4.4 Die CurTask-Activity

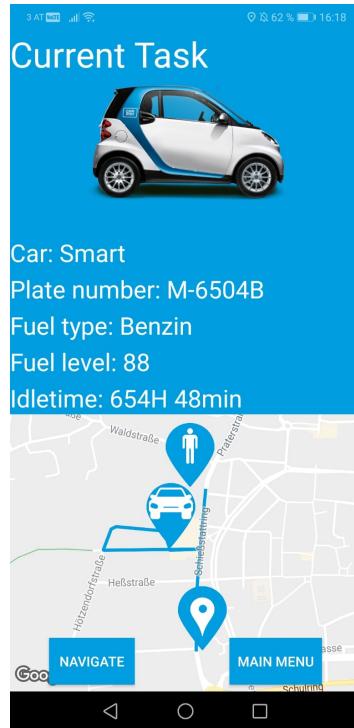


Abbildung 16.9: App - Aktueller Auftrag

In dieser Activity wird der dem Benutzer zugewiesene Auftrag angezeigt. Für die tatsächliche Navigation zum Fahrzeug und zur Destination wird auf die Google Maps-App weitergeleitet. Die Activity ist für die Erstellung eines GeoFences zuständig. Dieser notifiziert die App, sobald der Benutzer in der Nähe der Destination ist. Für das Erstellen werden zwei Objekte benötigt, welche wie folgt erzeugt werden.

```

1  private PendingIntent getGeofencePendingIntent() {
2      // Reuse the PendingIntent if we already have it.
3      if (geofencePendingIntent != null) {
4          return geofencePendingIntent;
5      }
6      Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);
7      geofencePendingIntent = PendingIntent.getService(this, 0, intent, 0);
8      return geofencePendingIntent;
9  }
10
11 private GeofencingRequest getGeofencingRequest() {
12     GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
13     builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
14     builder.addGeofence(DataBean.getGeofence());
15     return builder.build();
16 }
```

Listing 16.18: CurTask.java Erstellen der GeoFence Objekte

Die erste Methode erstellt den benötigten Intent, welcher als Callback für die GeoFence-Events dient. In der zweiten Methode wird der *GeoFenceRequest* erstellt, welcher angibt, auf welches Event und welchen GeoFence reagiert werden soll. Der GeoFence selbst wird in der *onCreate()* Methode registriert.

```

1 @SuppressLint({"MissingPermission", "NewApi"})
2 @Override
3 protected void onCreate(Bundle savedInstanceState) {
4     super.onCreate(savedInstanceState);
5
6     geofencingClient = LocationServices.getGeofencingClient(this);
7     Geofence geofence = new Geofence.Builder()
8         .setRequestId(DataBean.getCurJob().getId() + "")
9         .setCircularRegion(
10             DataBean.getCurJob().getDestLat(),
11             DataBean.getCurJob().getDestLng(),
12             DataBean.getGeoFenceRadius()
13         )
14         .setExpirationDuration(Geofence.NEVER_EXPIRE)
15         .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER)
16         .build();
17
18     DataBean.setGeofence(geofence);
19
20     geofencingClient.addGeofences(getGeofencingRequest(),
21         getGeofencePendingIntent())
22         .addOnSuccessListener(this, new OnSuccessListener<Void>() {
23             @Override
24             public void onSuccess(Void aVoid) {
25                 Log.i(TAG, "Geofences registered successfully");
26             }
27         });
28 }
```

```

25    }
26  })
27 .addOnFailureListener(this, new OnFailureListener() {
28     @Override
29     public void onFailure(@NonNull Exception e) {
30         Log.e(TAG, "Geofences not registered");
31         e.printStackTrace();
32     }
33 });
34 }
```

Listing 16.19: CurTask.java onCreate()

Zuerst wird über das *LocationServices*-API ein GeoFence erzeugt. Dieser enthält die ID des Jobs, sowie dessen Koordinaten. Auf den GeoFence wird nur reagiert, wenn man ihn betritt. Danach wird er unter der Verwendung der oben angeführten Methoden dem GeoFence-Client hinzugefügt.

Um auf die GeoFence-Events reagieren zu können, auch, wenn die App nicht im Fokus ist, wird ein *IntentService* benötigt. Dieser muss in der Manifest-Datei und im GeoFence-Client registriert werden. Die Behandlung der Events sieht wie folgt aus.

```

1 // package & imports
2
3 public class GeofenceTransitionsIntentService extends IntentService {
4
5     // ...
6
7     protected void onHandleIntent(Intent intent) {
8         // gleich wie im Verwendungsbeispiel
9
10        if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER) {
11            List<Geofence> triggeringGeofences = geofencingEvent.
12                getTriggeringGeofences();
13            StringBuilder sb = new StringBuilder();
14            for(Geofence g : triggeringGeofences){
15                sb.append(g.getRequestId());
16            }
17
18            Intent i = new Intent(this, Main_drawer.class);
19            PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, i,
20                0);
21
22            // erstellen einer Notification
23
24            // send job finished to server
25
26            Toast.makeText(this,"Geofence entered: " + sb.toString(),Toast.
27 LENGTH_LONG).show();
28        }
29    }
30}
```

```
25     Log.i(TAG, "Geofence entered: " + geofenceTransition);
26 } else {
27     // Log the error.
28     Log.e(TAG, "GeofenceError later");
29 }
30 }
31 }
```

Listing 16.20: GeofenceTransitionsIntentService.java

Die Klasse ist sehr ähnlich zu der aus dem Beispiel. Es wird ein Intent auf die *Main-Activity* erstellt. Dieser wird in einer Notification verwendet. Weiters wird dem Server mitgeteilt, dass der Auftrag abgeschlossen wurde.

## 16.5 Fazit

Die Verwendung von Google Maps im Projekt verlief grundsätzlich reibungslos. Jedoch gab es anfänglich Probleme bei der Einbindung der *gmap*-Komponente. Auf Grund einer zu neuen *PrimeFaces*-Version wurde die Karte beim Laden der Seite nicht dargestellt. Es wurden jedoch weder im Browser noch im Server-Log ein Fehler angezeigt. Beim Recherchieren im Internet wurde als möglicher Lösungsweg die Verwendung einer älteren Version von *PrimeFaces* gefunden. Dies konnte das Problem tatsächlich beheben.

# Kapitel 17

## Kapitelzuordnung

<b>Kapitelbezeichnung</b>	<b>Seite</b>	<b>Verantwortliche Person</b>
Individuelle Zielsetzungen	1f	Alle
Webserver als Kommunikations-schnittstelle für Apps	2ff	Maximilian Suchy
Sichere Kommunikation	36ff	Maximilian Suchy
Service Management	45ff	Maximilian Suchy
Umsetzung im Projekt	53ff	Maximilian Suchy
Grundlagen von REST	60ff	Oliver Hovorka
Kriterien und deren Gewichtung	63ff	Oliver Hovorka
Bewertung der Client APIs	67ff	Oliver Hovorka
Verwendung des ausgewählten Client APIs im Projekt	94ff	Oliver Hovorka
Einführung in die Webentwicklung	106ff	Matej Bradarić
Auswahl einer geeigneten Technologie	119ff	Matej Bradarić
Verwendung im Projekt	138ff	Matej Bradarić
Grundlagen von Routenberechnungen	155ff	Lukas Heinzl
Vorstellung diverser Kartenservices	163ff	Lukas Heinzl
Bewertung und Auswahl	181ff	Lukas Heinzl
Verwendung des ausgewählten Dienstes im Projekt	193ff	Lukas Heinzl

# Abbildungsverzeichnis

2.1	Reverse Proxying mit SSL-Offloading . . . . .	19
2.2	Cherokee Admin Interface . . . . .	23
2.3	GlassFish Reverse Proxy mit Cherokee . . . . .	24
2.4	Deployment über GlassFish Admin Console . . . . .	30
2.5	Deployment über GlassFish Admin Console . . . . .	33
3.1	Erzeugung asymmetrischer Schlüsselpaare . . . . .	38
3.2	Asymmetrische Verschlüsselung . . . . .	39
3.3	Authentifizierung mittels Signatur . . . . .	39
3.4	Symmetrische Verschlüsselung . . . . .	40
8.1	Java Statistik 2017 . . . . .	70
8.2	Android Statistik 2018 . . . . .	73
8.3	R-Client . . . . .	83
8.4	R-Client . . . . .	84
8.5	R-Client . . . . .	85
8.6	Latenzbenchmark . . . . .	91
9.1	Login Screen der mobilen Applikation . . . . .	98

9.2	Übersicht der mobilen Applikation . . . . .	101
9.3	Navigation mithilfe der mobilen Applikation . . . . .	101
9.4	Zuweisung eines Autos mithilfe der mobilen Applikation . . . . .	103
9.5	Liste der Autos . . . . .	103
10.1	<i>Aufruf einer Webseite</i> . . . . .	108
10.2	<i>Aufruf einer dynamischen Webseite</i> . . . . .	110
10.3	<i>Model-View-Controller</i> . . . . .	118
11.1	<i>Hauptseite</i> . . . . .	121
11.2	<i>Einfache Implementierung einer Karte</i> . . . . .	123
11.3	Karte mit Custom Marker . . . . .	128
11.4	<i>phpMap</i> -Implementierung einer Karte mit PHP . . . . .	133
11.5	<i>JSF-DataTable</i> -Beispiel - Tabelle mit JSF . . . . .	135
11.6	<i>jsTable</i> -Beispiel - Tabelle in JavaScript . . . . .	136
11.7	<i>phpTable</i> -Beispiel - Tabelle in PHP . . . . .	138
12.1	<i>login.xhtml</i> - Login der Webseite . . . . .	139
12.2	<i>main.xhtml</i> - Tabellen auf der Hauptseite . . . . .	143
12.3	<i>Detailansicht eines Autos</i> . . . . .	145
12.4	<i>Verfügbare Mitarbeiter</i> . . . . .	145
12.5	<i>Neuen Standort auswählen</i> . . . . .	146
12.6	<i>Navigationsleiste</i> . . . . .	147
12.7	<i>Benutzerverwaltung</i> . . . . .	147
12.8	<i>Benutzerverwaltung</i> Editieren eines Benutzers . . . . .	148

12.9	<i>Benutzerverwaltung</i> Löschen eines Benutzers . . . . .	148
12.10	<i>Benutzerverwaltung</i> Hinzufügen eines Benutzers . . . . .	148
12.11	<i>Benutzerprofil</i> - Ändern der E-Mail oder des Passworts . . . . .	153
13.1	Trilateration in der Ebene . . . . .	156
14.1	St. Pölten auf OpenStreetMap . . . . .	166
14.2	iframe Einbettung von OpenStreetMap . . . . .	167
14.3	Verwendung von OpenLayers mit eigenem Marker . . . . .	168
14.4	St. Pölten auf Google Maps . . . . .	171
14.5	iframe Einbettung von Google Maps . . . . .	172
14.6	St. Pölten auf Here Maps . . . . .	176
14.7	St. Pölten auf Sygic Maps . . . . .	179
16.1	Verwendung der <i>gmap</i> -Komponente ohne API-Key . . . . .	195
16.2	<i>gmap</i> -Beispiel - Projektstruktur . . . . .	196
16.3	<i>gmap</i> -Beispiel - Oberfläche . . . . .	199
16.4	<i>gmap</i> -Beispiel - Klick auf einen Marker . . . . .	199
16.5	<i>gmap</i> -Beispiel - Verschiebung eines Markers . . . . .	200
16.6	Website - Übersicht . . . . .	200
16.7	App - Übersicht . . . . .	209
16.8	App - Auftragszuweisung . . . . .	212
16.9	App - Aktueller Auftrag . . . . .	214

# Tabellenverzeichnis

2.1	Bewertungstabelle NGINX und Cherokee . . . . .	24
2.2	Bewertungstabelle GlassFish und WildFly . . . . .	35
7.1	Bewertung . . . . .	67
8.1	Bewertung von Jersey . . . . .	74
8.2	Bewertung von Resteasy . . . . .	79
8.3	Bewertung von Resteasy . . . . .	87
8.4	Bewertung von Retrofit . . . . .	92
14.1	Statistik zu OpenStreetMap . . . . .	164
14.2	Kostentabelle für Google Maps . . . . .	170
14.3	Kostentabelle für Sygic Maps . . . . .	178
15.1	Bewertung der Funktionalität . . . . .	184
15.2	Bewertung der Integration . . . . .	189
15.3	Teilbewertungen der Dienste . . . . .	191
15.4	Teilbewertungen der Dienste in Relation . . . . .	191
15.5	Gesamtbewertung der Dienste . . . . .	192

# Listings

2.1	Beispiel für eine REST-Anfrage . . . . .	5
2.2	Beispiel für eine REST-Antwort mit JSON . . . . .	5
2.3	Beispiel für eine REST-Antwort mit XML . . . . .	5
2.4	Beispiel für eine XML-RPC-Anfrage . . . . .	6
2.5	Beispiel für eine SOAP-Anfrage . . . . .	7
2.6	Beispiel JAX-RS Application, language=Java . . . . .	9
2.7	Beispiel JAX-RS Resource Class . . . . .	10
2.8	JAX-RS: Clientseitig erzeugte Objekte am Server . .	11
2.9	JAX-RS: GenericEntity . . . . .	12
2.10	JAXB-Annotationen in einer Geschäftsklasse . . . . .	13
2.11	JAXB JSON-Darstellung . . . . .	14
2.12	JAXB XmlAdapter . . . . .	15
2.13	Jackson ObjectMapper . . . . .	16
2.14	Jackson POJO . . . . .	16
2.15	Jackson JSON-Darstellung POJO . . . . .	16
2.16	Jackson ObjectNode . . . . .	17
2.17	Jackson JSON-Darstellung ObjectNode . . . . .	18
2.18	Jackson Features . . . . .	18

<b>2.19</b>	Konfiguration: NGINX HTTP Reverse Proxy . . . . .	21
<b>2.20</b>	Konfiguration: NGINX HTTP Reverse Proxy mit AJP . .	22
<b>2.21</b>	Beispielausgabe für asadmin list-jobs . . . . .	28
<b>3.1</b>	Serverblock NGINX ohne HTTPS . . . . .	43
<b>3.2</b>	Serverblock NGINX mit HTTPS . . . . .	44
<b>3.3</b>	Serverblock NGINX mit HTTPS und HSTS . . . . .	45
<b>4.1</b>	Initskript /etc/init.d/example . . . . .	49
<b>4.2</b>	Initskript /etc/init.d/stuntman . . . . .	50
<b>4.3</b>	Unit /lib/systemd/system/sshd.service . . . . .	52
<b>5.1</b>	Initskript für GlassFish . . . . .	57
<b>5.2</b>	Initskript für WildFly . . . . .	57
<b>5.3</b>	Unit File für WildFly . . . . .	58
<b>8.1</b>	Jersey Server Beispiel . . . . .	68
<b>8.2</b>	Jersey Client . . . . .	71
<b>8.3</b>	Resteasy Server Beispiel . . . . .	75
<b>8.4</b>	Resteasy Client . . . . .	77
<b>8.5</b>	Resteasy Server Beispiel . . . . .	80
<b>8.6</b>	Resteasy Client . . . . .	86
<b>8.7</b>	Retrofit Client Beispiel . . . . .	88
<b>9.1</b>	GenericService . . . . .	95
<b>9.2</b>	AndroidLogin Klasse . . . . .	97
<b>9.3</b>	UserClient . . . . .	98
<b>9.4</b>	Login Call . . . . .	99

9.5	Job Klasse . . . . .	102
9.6	JobClient . . . . .	102
9.7	Create Job Call . . . . .	104
9.8	Create Job Call . . . . .	105
9.9	Gson converter . . . . .	106
11.1	Implementation einer Karte mit JS . . . . .	122
11.2	Implementation einer Karte mit Custom Markern . . .	126
11.3	Datenbankverbindung mit PHP . . . . .	129
11.4	Ausgabe der XML Datei . . . . .	130
11.5	Deklaration der downloadURL Methode . . . . .	131
11.6	Laden der Marker . . . . .	132
11.7	datatable-bsp-index.xhtml . . . . .	134
11.8	javascript-datatable.html . . . . .	135
11.9	php-datatable.php . . . . .	137
12.1	login.xhtml . . . . .	140
12.2	loginController.java . . . . .	141
12.3	main.xhtml . . . . .	144
12.4	main.xhtml . . . . .	146
12.5	benutzerverwaltung.xhtml . . . . .	149
12.6	benutzerverwaltungController.java . . . . .	150
12.7	profilController.java . . . . .	153
14.1	osm_iframe.html . . . . .	167
14.2	osm_openlayers.html . . . . .	168

14.3	gm_iframe.html . . . . .	172
14.4	gm_route_legs.json . . . . .	173
16.1	use_widgetvar.js . . . . .	193
16.2	use_gmap_script.html . . . . .	195
16.3	gmap_bsp_index.xhtml . . . . .	196
16.4	MapController.java . . . . .	197
16.5	main.xhtml . . . . .	201
16.6	MarkersView.java . . . . .	202
16.7	activity_maps.xml . . . . .	204
16.8	MapsActivity.java . . . . .	204
16.9	Einbinden des Direction-APIs . . . . .	205
16.10	DrawRoute.java . . . . .	206
16.11	CreateGeofence.java . . . . .	207
16.12	GeofenceTransitionsIntentService.java . . . . .	207
16.13	Main_drawer.java onCreate() . . . . .	210
16.14	Main_drawer.java onMapReady() . . . . .	210
16.15	Main_drawer.java handleNewLocation() . . . . .	211
16.16	AssignCar.java makeRouteWithDest() . . . . .	213
16.17	AssignCar.java onMapLongClick() . . . . .	213
16.18	CurTask.java Erstellen der GeoFence Objekte . . . . .	215
16.19	CurTask.java onCreate() . . . . .	215
16.20	GeofenceTransitionsIntentService.java . . . . .	216

# Literaturverzeichnis

[Su:Web1] <https://aplawrence.com/Basics/unix-startup-scripts-1.html>

Andrew Smallshaw: Unix and Linux startup scripts

20.11.2018

[Su:Web2] <http://www.ac.uk.pgp.net/pgpnet/secemail/q4/node4.html>

Piete Brooks: Security: Privacy, Authenticity and Integrity

07.12.2018

[Su:Web3] <http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html>

The Apache Tomcat Connectors - AJP Protocol Reference

08.02.2019

[Su:Web4] <http://xmlrpc.scripting.com/spec.html>

Dave Winer: XML-RPC Specification

20.02.2019

[Su:Web5] <https://www.w3.org/TR/soap12/>

W3C: SOAP 1.2 Specification

20.02.2019

[Su:Web6] <https://www.freedesktop.org/software/systemd/man/systemd.unit.html>

freedesktop.org: Systemd Unit Configuration

26.11.2018

[Su:Web7] <https://wiki.gentoo.org/wiki/Handbook:X86/Working/Initscripts>

Gentoo Handbook: Initscripts, Dependencies

21.11.2018

[Su:Web8] <https://github.com/FasterXML/jackson-databind/wiki/JacksonFeatures>

Jackson GitHub Wiki: Jackson Features

29.03.2019

[Su:JavaEE1] <https://javaee.github.io/glassfish/doc/5.0/reference-manual.pdf>

Java EE GitHub: GlassFish Reference Manual

08.01.2019

[Su:JavaEE2] <https://javaee.github.io/glassfish/doc/5.0/administration-guide.pdf>  
Java EE GitHub: GlassFish Administration Guide  
08.01.2019

[Su:JavaEE3] <https://javaee.github.io/glassfish/doc/5.0/application-deployment-guide.pdf>  
Java EE GitHub: GlassFish Application Deployment Guide  
08.01.2019

[Su:WildFly1] [http://docs.wildfly.org/14/Admin\\_Guide.html](http://docs.wildfly.org/14/Admin_Guide.html)  
WildFly 14.0.0 Final Admin Guide  
09.01.2019

[Su:JSR-366] [https://download.oracle.com/otndocs/jcp/java\\_ee-8-final-spec/](https://download.oracle.com/otndocs/jcp/java_ee-8-final-spec/)  
JSR-000366 Java Platform, Enterprise Edition 8 Specification  
09.01.2019

[Su:JSR-339] [https://download.oracle.com/otndocs/jcp/jaxrs-2\\_0\\_rev\\_A-mrel-spec/](https://download.oracle.com/otndocs/jcp/jaxrs-2_0_rev_A-mrel-spec/)  
JSR-000339 Java API for RESTful Web Services 2.0 rev A  
10.01.2019

[Su:JSR-222] [https://download.oracle.com/otndocs/jcp/jAXB-2\\_3-mrel3-spec/](https://download.oracle.com/otndocs/jcp/jAXB-2_3-mrel3-spec/)  
JSR-000222 Java Architecture for XML Bindings 2.3  
12.01.2019

[Ho:Web01] [https://www.ics.uci.edu/fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/fielding/pubs/dissertation/fielding_dissertation.pdf)  
REST - Definierende Dissertation  
19.11.2018

[Ho:Web02] [https://www.ics.uci.edu/fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/fielding/pubs/dissertation/fielding_dissertation.pdf)  
Zitat-Dr. Fielding optionale Funktionalität S.84  
19.11.2018

[Ho:Web03] <https://plumbr.io/blog/java/java-version-and-vendor-data-analyzed-2017-edition>  
Java versions used 2013-2017  
5.2.2019

[Ho:Web04] <https://jersey.github.io/documentation/latest/index.html>  
Jersey User Guide  
14.2.2019

[Ho:Web19] <https://mobiforge.com/research-analysis/android-statistics-2018-sdks-versions-across-all-continents>  
Android Statistik  
15.3.2019

[Ho:Web05] <https://square.github.io/retrofit/>

Retrofit Website

14.3.2019

[Ho:Web06] <https://square.github.io/retrofit/2.x/retrofit/>

Retrofit Javadocumetation

14.3.2019

[Ho:Web07] <https://stackoverflow.com/questions/tagged/retrofit?sort=active&pageSize=15>

Retrofit Stackoverflow

15.3.2019

[Ho:Web08] <https://infinum.co/the-capsized-eight/top-10-android-libraries-every-android-developer-should-know-about>

Retrofit Ranking

15.3.2019

[Ho:Web09] <http://instructure.github.io/blog/2013/12/09/volley-vs-retrofit/>

Latenzzeiten

15.3.2019

[Ho:Web10] <https://resteasy.github.io/index.html>

Resteasy Website

16.3.2019

[Ho:Web11] [http://docs.jboss.org/resteasy/docs/3.6.3.Final/userguide/html\\_single/index.html#preface](http://docs.jboss.org/resteasy/docs/3.6.3.Final/userguide/html_single/index.html#preface)

Resteasy User Guide

16.3.2019

[Ho:Web12] <https://developer.jboss.org/wiki/ResteasyWIKI>

Resteasy Wiki

16.3.2019

[Ho:Web13] <http://docs.jboss.org/resteasy/docs/3.0.9.Final/javadocs/index.html>

Resteasy Javadoc

16.3.2019

[Ho:Web14] <https://stackoverflow.com/questions/tagged/resteasy?sort=active&pageSize=15>

Resteasy Stackoverflow

16.3.2019

[Ho:Web15] <https://jira.jboss.org/projects/RESTEASY/issues/RESTEASY-2187?filter=allopenissues>

Resteasy Issue Tracker  
16.3.2019

[Ho:Web16] <https://restlet.com/documentation/>  
Restlet Homepage  
18.3.2019

[Ho:Web17] <https://restlet.com/open-source/documentation/user-guide/2.3/introduction/overview>  
Restlet User Guide  
18.3.2019

[Ho:Web18] <https://stackoverflow.com/questions/tagged/restlet?sort=active&pageSize=15>  
Restlet Stackoverflow  
18.3.2019

[Ho:Web20] <https://developer.android.com/training/location/geofencing>  
30.3.2019

[Ho:Web21] <https://restlet.com/modules/client/>  
30.3.2019

[Ho:Web22] <https://restlet.com/modules/studio/>  
30.3.2019

[Ho:Web23] <https://restlet.com/modules/cloud/>  
30.3.2019

[Br:Web01] <https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/>  
Informationen zu Cleient und Serverseitiger Programmierung  
04.11.2018

[Br:Web02] <https://conceptainc.com/blog/php-server-side-scripting-designed-web-development-review/>  
Informationen zu PHP  
04.11.2018

[Br:Web03] <https://conceptainc.com/blog/choosing-a-javascript-library-for-web-apps-polymer-review/>  
Informationen zu JavaScript  
06.11.2018

[Br:Web04] <https://www.upwork.com/hiring/development/how-scripting-languages-work/>

Informationen zu Clientseitigem Skripting  
06.11.2018

[Br:Web05] <https://www.baeldung.com/java-servers>  
Webentwicklung mit Java  
19.11.2018

[Br:Web06] <http://www2.math.uni-wuppertal.de/schaefer/jv/haupt/node102.html>  
Webentwicklung mit Java  
19.11.2018

[Br:Web07] <https://www.dev-insider.de/was-ist-php-a-578773/>  
Informationen zu PHP  
12.12.2018

[Br:Web08] <https://www.edv-lehrgang.de/dynamische-websites-mit-php/>  
Informationen zur Entwicklung von dynamischen Webseiten  
20.12.2018

[Br:Web09] <http://www.mathematik.uni-ulm.de/sai/ws01/portalsem/wiede/>  
Informationen zu JavaScript  
04.01.2019

[Br:Web10] <https://www.datenschutzbeauftragter-info.de/skriptsprachen-und-javascript-einfach-erklaert/>  
Informationen zu JavaScript  
04.01.2019

[Br:Web11] <https://whatis.techtarget.com/de/definition/Nodejs>  
Informationen zu Node.js  
05.01.2019

[Br:Web12] <https://www.yuhiro.de/vorteile-und-nachteile-von-node-js/>  
Informationen zu Node.js  
05.01.2018

[Br:Web13] <https://developers.google.com/maps/documentation/javascript/markers>  
Google Maps JavaScript API  
06.01.2018

[Br:Web14] <https://developers.google.com/maps/documentation/javascript/mysql-to-maps>  
Google Maps API mit PHP  
07.02.2019

[He:Web01] <http://www.kowoma.de/gps/>

Informationen zu GPS

04.11.2018

[He:Web02] [https://www.techopedia.com/definition/16847/global-navigation-satellite-](https://www.techopedia.com/definition/16847/global-navigation-satellite-system-gnss)

system-gnss

Global Navigation Satellite System (GNSS)

03.11.2018

[He:Web03] <https://electronics.howstuffworks.com/gadgets/travel/gps1.htm>

Funktionsbeschreibung von Trilateration

08.12.2018

[He:Web04] <https://gisgeography.com/wgs84-world-geodetic-system/>

WGS84 15.11.2018

[He:Web05] <http://www.kowoma.de/gps/Geschichte.htm>

Geschichte von GPS

04.11.2018

[He:Web06] [http://www.esa.int/Our\\_Activities/Navigation/Galileo/What\\_is\\_Galileo](http://www.esa.int/Our_Activities/Navigation/Galileo/What_is_Galileo)

Galileo, Europäische Weltraumorganisation

15.11.2018

[He:Web07] [https://www.zdnet.de/88284753/satellitennavigationssystem-galileo-](https://www.zdnet.de/88284753/satellitennavigationssystem-galileo-stellt-erste-dienste-bereit/)

stellt-erste-dienste-bereit/

Start des Galileo-Betriebs

15.11.2018

[He:Web08] <http://www.kowoma.de/gps/galileo/Uebersicht.htm>

Übersicht zu Galileo

15.11.2018

[He:Web09] [https://space.skyrocket.de/doc\\_sdat/navstar-3.htm](https://space.skyrocket.de/doc_sdat/navstar-3.htm)

Zukunft von GPS, GPS-3

17.11.2018

[He:Web10] [http://esamultimedia.esa.int/docs/galileo/Galileo\\_factsheet\\_2018.pdf](http://esamultimedia.esa.int/docs/galileo/Galileo_factsheet_2018.pdf)

Status der Galileo-Satelliten

17.11.2018

[He:Web11] [http://europa.eu/rapid/press-release\\_IP-16-4366\\_de.htm](http://europa.eu/rapid/press-release_IP-16-4366_de.htm)

Fertigstellung von Galileo

17.11.2018

[He:Web12] <https://www.usegalileo.eu/DE/>

Einsatz von Galileo

17.11.2018

[He:Web13] <https://wiki.osmfoundation.org/wiki/FAQ>

OpenStreetMap Foundation

18.11.2018

[He:Web14] [https://wiki.openstreetmap.org/wiki/History\\_of\\_OpenStreetMap](https://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap)

OpenStreetMap Geschichte

18.11.2018

[He:Web15] [https://www.openstreetmap.org/stats/data\\_stats.html](https://www.openstreetmap.org/stats/data_stats.html)

OpenStreetMap Statistiken

18.11.2018

[He:Web16] <https://wiki.osmfoundation.org/wiki/Membership>

OpenStreetMap Foundation, Mitgliedschaft

26.12.2018

[He:Web17] <https://blog.osmfoundation.org/about/>

OpenStreetMap Foundation, Aufgaben

26.12.2018

[He:Web18] <https://www.openstreetmap.org/copyright>

OpenStreetMap, Copyright

27.12.2018

[He:Web19] [https://wiki.openstreetmap.org/wiki/Deploying\\_your\\_own\\_Slippy\\_Map](https://wiki.openstreetmap.org/wiki/Deploying_your_own_Slippy_Map)

OpenStreetMap, JavaScript APIs

27.12.2018

[He:Web20] <http://project-osrm.org/docs/v5.5.1/api>

Open Source Routing Machine, Webservice Dokumentation

27.12.2018

[He:Web21] [https://wiki.openstreetmap.org/wiki/OpenLayers\\_Marker\\_Example](https://wiki.openstreetmap.org/wiki/OpenLayers_Marker_Example)

Open Layers, Marker Beispiel

27.12.2018

[He:Web22] <https://www.theguardian.com/technology/2015/feb/08/google-maps-10-anniversary-iphone-android-street-view>

Google Maps, Entstehungsgeschichte

30.12.2018

[He:Web23] <https://de.statista.com/statistik/daten/studie/222849/umfrage/marktanteile-der-suchmaschinen-weltweit/>  
Google Suche, Marktanteil  
30.12.2018

[He:Web24] <https://www.tagesschau.de/wirtschaft/google-167.html>  
Google, Umstrukturierung  
Artikel "Google wird zu Alphabet"  
30.12.2018

[He:Web25] <https://mapsplatformtransition.withgoogle.com/calculator>  
Google Maps, Preisrechner  
31.12.2018

[He:Web26] <https://developers.google.com/maps/documentation/directions/client-library>  
Google Maps Platform, Client Libraries  
01.01.2019

[He:Web27] <https://venturebeat.com/2015/08/03/why-3-car-giants-just-bought-nokias-mapping-division-for-3b/>  
Here Maps, Geschichte  
02.01.2019

[He:Web28] <https://www.digitaltrends.com/business/audi-bmw-and-daimler-have-agreed-to-spend-2-7-billion-on-nokias-here-mapping-tech/>  
Here Maps, Verkauf an Audi, BMW und Daimler  
02.01.2019

[He:Web29] <https://www.bloomberg.com/news/articles/2017-09-26/chinese-bid-for-stake-in-here-maps-denied-by-u-s-security-panel>  
Here Maps, asiatische Investition scheitert  
02.01.2019

[He:Web30] <https://www.reuters.com/article/us-intel-here-equity/intel-to-take-stake-in-german-mapping-firm-here-in-automated-driving-push-idUSKBN14N0RN>  
Here Maps, Intel kauft 15% der Anteile  
02.01.2019

[He:Web31] <https://www.cnet.com/roadshow/news/continental-bosch-help-boost-hd-mapping-with-here-acquisitions/>  
Here Maps, Bosch und Continental kaufen 5% der Anteile  
02.01.2019

[He:Web32] <https://developer.here.com/plans>

Here Maps, Preismodell

02.01.2019

[He:Web33] <https://developer.here.com/develop/rest-apis>

Here Maps, Webservices

02.01.2019

[He:Web34] <https://www.bbc.com/news/business-28371752>

Sygic Maps, Entstehungsgeschichte

04.01.2019

[He:Web35] <https://www.cnet.com/roadshow/pictures/first-iphone-nav-app-but-not-from-tomtom/>

Sygic Maps, Entstehungsgeschichte 2

04.01.2019

[He:Web36] <http://www.sygic.com/de/about>

Sygic Maps, Erfolge

04.01.2019

[He:Web37] <https://eshop.sygic.com/de/>

Sygic Maps, Preise

05.01.2019

[He:Web38] <https://www.sygic.com/de/gps-navigation>

Sygic Maps, Grundfunktionen

05.01.2019

[He:Web39] <https://www.sygic.com/de/gps-navigation/premium/>

Sygic Maps, Premiumfunktionen

05.01.2019

[He:Web40] <https://www.sygic.com/de/developers/maps-api-services/>

Sygic Maps, Webservices - Dokumentation

05.01.2019

[He:Web41] <https://showcase.bootsfaces.net/integration/PrimeFaces.jsf>

Integration von BootsFaces mit PrimeFaces

06.02.2019

[He:Web42] <https://developers.google.com/maps/documentation/android-sdk>

Dokumentation der Android-SDK von Google Maps

06.02.2019

[He:Web43] [https://wiki.openstreetmap.org/wiki/Commercial\\_OSM\\_Software\\_and\\_Services](https://wiki.openstreetmap.org/wiki/Commercial_OSM_Software_and_Services)

Kommerzielle Anbieter von OSM

07.02.2019

[He:Web44] <https://www.uptrends.com/de/news/dienste-von-google-und-yahoo-gleichermassen-erreichbar>

Verfügbarkeitsraten von Google und Yahoo

07.02.2019

[He:Web45] <https://www.sygic.com/de/company/terms-conditions-for-sygic-maps>

Terms & Conditions, Sygic Maps 07.02.2019

[He:Web46] [https://www.primefaces.org/docs/guide/primefaces\\_user\\_guide\\_6\\_2.pdf](https://www.primefaces.org/docs/guide/primefaces_user_guide_6_2.pdf)

PrimeFaces User Guide, Version 6.2

17.02.2019

[He:Web47] <https://developers.google.com/maps/documentation/javascript/controls>

Google Maps API Dokumentation, Controls

17.02.2019

[He:Web48] <https://developer.android.com/training/location/geofencing#java>

Dokumentation des LocationService-API, Android Developers

03.03.2019

**DIPLOMARBEIT  
DOKUMENTATION**

Namen der Verfasser/innen	Matej Bradaric Lukas Heinzl Oliver Hovorka Maximilian Suchy
Jahrgang / Klasse Schuljahr	2018/2019 5AHIF
Thema der Diplomarbeit	Optimierung der Fahrzeugrückstellung für ein Car-Sharing-Unternehmen
Kooperationspartner	Car2Go Hintere Zollamtstraße 9, 1030 Wien Ansprechpartner: Alexander Hovorka (alexander.hovorka@gmail.com)

Aufgabenstellung	Ein CarSharing Unternehmen möchte innerbetriebliche Abläufe optimieren. Konkret sollen Fahrzeuge, die an wenig frequentierten Orten abgestellt sind, an sogenannte "Hotspots" umgeparkt werden. Dabei ist besonders auf Rentabilität zu achten. Zur Optimierung dieser Abläufe ist eine Softwarelösung zielführend. Ein System zur Lokalisierung der Fahrzeuge (GPS-Position) ist bereits vorhanden.
------------------	--

Realisierung	Zur Entwicklung der Webservices wurde die RESTful-Architektur herangezogen und mittels Jersey implementiert. Als Nachrichtenprotokoll wird standardmäßig JSON verwendet.  Die Webservices werden auf einem WildFly-Server (Version 15.0.1.Final)  Die öffentlich zugängliche Website wurde mithilfe von JSF entwickelt. Sie ist unter carsharing.privatevoid.net erreichbar.  Als Client wurde eine mobile Applikation unter Android entwickelt. Diese bekommt ihre Daten über ein REST-Service, für welches Retrofit als Client API verwendet wird.
--------------	--

Ergebnisse	<ul style="list-style-type: none"><li>• Ist-Analyse der Firma Car2Go abgeschlossen</li><li>• Finales Datenbankmodell</li><li>• Datenbank implementiert in PostgreSQL</li><li>• Technologievergleiche durchgeführt</li><li>• Webservices entwickelt und bereitgestellt</li><li>• Client-Android-Applikation fertiggestellt</li><li>• Website zur Verwaltung fertig entwickelt</li></ul>
------------	--

Typische Grafik, Foto etc.  
(mit Erläuterung)

The screenshot shows a web-based monitoring or tracking system. On the right is a map of Vienna, Austria, with many small blue circular icons scattered across it, representing locations. On the left, there is a sidebar with two sections: 'VEHICLES' and 'EMPLOYEES'. The 'VEHICLES' section contains a list of vehicle identifiers: K-0909, C-0940, K-0848, L-0903, K-1808, Z-0946, C-0944, K-1234W, K-0420P, C-0940U, K-0943H, C-0938F, K-0935F, K-1804, and L-0945Y. The 'EMPLOYEES' section lists four names: JOHN SMITH, MIKE SMITH, MIKE DYSON, and JOHN WHITE.

Hier sieht man einen Ausschnitt der Webapplikation. Zu sehen sind die Positionen der Fahrzeuge und Mitarbeiter. Diese sind sowohl auf der Karte als auch in der Liste sichtbar.

Teilnahme an Wettbewerben, Auszeichnungen	spusu
---	-------

Möglichkeiten der Einsichtnahme in die Arbeit	Die Arbeit liegt an der HTBLuVA St.Pölten/Abteilung Informatik (Waldstraße 3 3100 St. Pölten) auf.
---	--

Approbation (Datum / Unterschrift)	Prüfer/in	Abteilungsvorstand / Direktor/in
---------------------------------------	-----------	----------------------------------

	<b>COLLEGE OF ENGINEERING ST. PÖLTEN</b>	
Department:	Computer Science	

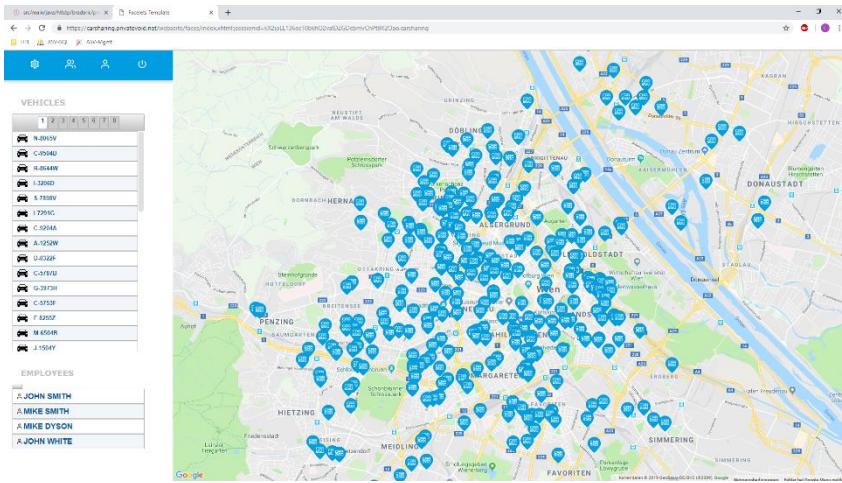
## **DIPLOMA THESIS DOCUMENTATION**

Names of the authors	Matej Bradaric Lukas Heinzl Oliver Hovorka Maximilian suchy
Year / Class Schoolyear	2018/2019 5AHIF
Topic of the diploma thesis	Optimizing the relocation services for a car sharing business
Cooperation partner	Car2Go Hintere Zollamtstraße 9, 1030 Wien Contact: Alexander Hovorka (alexander.hovorka@gmail.com)

Tasks	This project's objective is the development of a mobile application, which shall raise the profit of our partner Car2Go and ease the daily work routine of the employees. Simply put, our project will optimize the relocation of cars. This is necessary because cars are sometimes parked at less populated spots and these cars generate less profit, to solve this problem the cars shall then be moved to more highly frequented spots.
-------	--

Realisation	<p>To develop the web services the RESTful-architecture has been used and was implemented with Jersey. JSON was used as a standard message protocol.</p> <p>The web services are provided using a WildFly-server (version 15.0.1.Final).</p> <p>The website was developed using JSF and it is accessible under the domain carsharing.privatevoid.net.</p> <p>The Client was developed as a mobile application for Android Devices. This app receives its' data through a Rest-Service which was implemented through Retrofit.</p>
-------------	---

Results	<ul style="list-style-type: none"> <li>● Analysis of Car2Go complete</li> <li>● Final database mode</li> <li>● Implemented database using PostgreSQL</li> <li>● Comparison between the technologies</li> <li>● Web services developed and deployed</li> <li>● Client-android-application implemented</li> <li>● Website developed using JSF</li> </ul>
---------	--

Typical graphic, photo (with explanation)	 <p>Here you can see the start screen of the website. It shows all the available cars and employees. This is visualized on the map and in a list.</p>
---	--

Participation in competitions, Awards	spusu
---------------------------------------	-------

Accessability of diploma thesis	This diploma thesis is available for access at the HTBLuVA St. Pölten/Department Informatik (Waldstraße 3, 3100 St. Pölten).
---------------------------------	--

Approval (Date / Sign)	Examiner	Head of Department / College
---------------------------	----------	------------------------------

Begleitprotokoll Bradraic Matej		
KW	Dauer (in h)	Details
40	5	Einführung in LaTeX, Recherche zu LaTeX
43	13	Recherche zu JavaScript und PHP
44	14	Recherche und Erstellung des ersten Kapitels
47	8	Recherche und Einbindung der Google Map mit JavaScript
50	10	Recherche und Einbindung der Google Map mit PHP
3	10	Schreiben des zweiten Kapitels begonnen
5	3	Recherche
6	5	Recherche, Korrekturarbeiten
7	17	Recherche, Erneutes schreiben des ersten Kapitels
8	8	Recherche zu PHP, Korrekturarbeiten
9	10	Recherche zu PHP und JavaScript, Korrekturarbeiten
10	13	Fertigstellen des ersten Kapitels, Korrekturarbeiten des ersten Kapitels
11	10	Schreiben des dritten Kapitels
12	20	Fertigstellen des zweiten Kapitels, Korrekturarbeiten des zweiten Kapitels
13	30	Feritgstellen der Diplomarbeit, Korrekturarbeiten des dritten Kapitels.

Begleitprotokoll Lukas Heinzl		
KW	Dauer (in h)	Details
34	20	Recherche zu Gmap und Maps SDK for Android
40	3,00	Diplomandenseminar
41	1,00	Inhaltsverzeichnis
43	0,50	Diplomarbeitsbesprechung
43	2,00	Bearbeitung der Diplomarbeit: Ändern des Titels, Diplomandenvorstellung und der Kapitel
44	11,50	Recherche + 1.1 Überblick geschrieben 1.3.1 Geschichte NAVSTAR GPS geschrieben
46	12,00	1.2 Funktionsprinzip + 1.3.2 Geschichte Galileo geschrieben 1.4 Zukunftsentwicklung geschrieben 1.5 Einsatzgebiete + 2.1 Übersicht + 2.2.1 Geschichte (OpenStreetMap) geschrieben
48	1,00	Diplomarbeitsbesprechung, Überarbeitung der Diplomarbeit lt. Besprechung
49	1,00	Fertigstellung der Überarbeitung
52	20,50	2.2.2 OpenStreetMap Foundation geschrieben 2.2 OpenStreetMap komplett fertiggestellt, UTF-8 Kodierung umgestellt, Umlaute ersetzt, 2.3.1 Entstehungsgeschichte (Google Maps) + 2.3.2 Google LLC geschrieben, 2.3.3 Preismodell (Google Maps) geschrieben + 2.3.4 Funktionsumfang (Google Maps) angefangen
1	13,50	2.3 Google Maps komplett fertiggestellt Recherche zu HereMaps + 2.4 Here Maps komplett geschrieben, Recherche zu Sygic Maps + 2.5.1 Entstehungsgeschichte (Sygic Maps) geschrieben 2.5 Sygic Maps + 2. Kapitel komplett fertiggestellt
4	5,66	Diplomarbeitsbesprechung Überarbeitung der Diplomarbeit lt. Besprechung
6	15,00	Überarbeitung der Literaturverweise, Formatierung 3. Kapitel komplett fertiggestellt, Überarbeitung der Diplomarbeit lt. korrigiertem PDF
7	8,33	4.1 Die gmap-Komponente geschrieben
8	8,00	4.2.1 Die Oberfläche geschrieben + 4.2.2 Der Controller begonnen, Recherche und Analyse des Gm-APIs für Android
9	8,50	4.2.2 fertiggestellt + 4.3 (Maps SDK for Android) ohne 4.3.3 (Geofences) geschrieben, 4.3.3 (Verwendung von Geofences) geschrieben + 4.3 abgeschlossen
10	10,50	4.4 (Verwendung des Maps SDKs im Projekt) geschrieben - 4. Kapitel komplett fertiggestellt
12	0,50	Diplomarbeitsbesprechung
13	20,00	Diplomarbeitsbesprechung Überarbeitung lt. Besprechung angefangen + Vorbereitung auf Zusammenführung, Überarbeitung lt. Besprechung fertiggestellt, Kapitelzuordnung begonnen + Diplomarbeit in das gemeinsame Dokument transferiert, Danksagung geschrieben, Inhalt aller Diplomarbeiten zusammengeführt
14	10	Zusammenführung der Diplomarbeiten

Begleitprotokoll Oliver Hovorka		
KW	Dauer (in h)	Details
40	2,00	Meeting über Latex und generelle Information zur Diplomarbeit
41	11	Recherche zu dem Thema REST APIs Formulierung des Inhaltsverzeichnisses
43	16	Schreiben des ersten Kapitels (Einleitung) Recherche für das zweite Kapitel (Kriterien)
43	10,00	Recherche für das zweite Kapitel (Kriterien)
46	15,00	Schreiben des zweiten Kapitels (Kriterien)
49	1,00	Diplomarbeitsbesprechung (ersten 10 Seiten)
1	10	Recherche für Jersey, Beginn des Bewertens der APIs
6	5,00	Fertigstellung der Bewertung von Jersey
11	35	Recherche für Retrofit, Bewertung von Retrofit, Recherche für Resteasy, Bewertung von Resteasy, Recherche für Restlet
12	45	Bewertung Restlet, Korrekturlesen + Verbessern, Listings auswählen und Formatieren, Besprechung der Formaiterung und des Drucks der Fertigen DA, Beschreibung der Login Funktionalität, Beschreibung des Zuweisungsprozesses
13	30	Formatierun der Diplomarbeit, Fehler des Zusammenfügens ausbessern, Fehler des Zusammenfügens ausbessern, Fertigstellende Überprüfung

180,00

Begleitprotokoll Maximilian Suchy		
KW	Dauer (in h)	Details
46	8	Recherche GlassFish, REST
47	10	Recherche und Beschreibung System V, OpenRC
48	9	Recherche und Beschreibung systemd
49	8	Recherche und Beschreibung Verschlüsselung
51	9	Recherche Java EE
2	11	Recherche, Dokumentation von GlassFish
3	12	Recherche WildFly, Dokumentation GlassFish
4	8	Recherche und Beschreibung WildFly
5	15	Recherche und Dokumentation REST, JAX-RS
6	14	Beschreibung Reverse Proxying, NGINX und Cherokee
7	12	Bewertung GlassFish und WildFly, NGINX und Cherokee
8	14	Dokumentation Geschichtliches zu REST
9	12	Dokumentation der bisherigen Umsetzung
10	6	Dokumentation eingesetzter Hard- und Software
11	12	Fehleranalysen, Probleme bei der Entwicklung
12	11	Dokumentation JAX-RS, JAXB
13	13	Dokumentation REST im Projekt, Jackson, Finalisierung

184

<b>Besprechungsprotokoll Matej Bradaric</b>		
<b>Datum</b>	<b>Person(en)</b>	<b>Details</b>
21.12.2018	Bradaric	Beprechung des ersten Kapitels.
19.3.2018	Bradaric	Besprechung über den aktuellen Zustand der Diplomarbeit
20.3.2018	Alle	Besprechung des finalen Layouts.

Besprechungsprotokoll Lukas Heinzl		
Datum	Person(en)	Details
23.10.2018	Heinzl	Besprechung des Inhaltsverzeichnis Der Aufbau und Inhalt passt. Im ersten Kapitel soll das Augenmerk auf GPS und Galileo liegen. Zu Ändern waren die Seitenangaben, da sie nicht vorhanden waren.
28.11.2018	Heinzl	Besprechung des ersten Kapitels Die Form der Diplomarbeit ist in Ordnung. Es wurden einige Rechtschreib- und Grammatikfehler besprochen. Zu Ändern ist die Grafik zur Trilateration und der Name des ersten Kapitels. Die Sektion "Geocaching" ist zu Entfernen, da sie nicht zum Thema passt. Alle Quellen sollen einen Präfix haben, um sie bei der Zusammenfügung der Teildiplomarbeiten unterscheiden zu können. Der Fokus liegt als nächstes bei der Vorstellung und Evaluierung der Kartendienste.
23.01.2019	Heinzl	Besprechung des zweiten Kapitels Einige Rechtschreib-/Grammatikfehler sind vorhanden und müssen ausgebessert werden. Der Aufbau des 2. Kapitels ist in Ordnung. Als nächstes muss die Bewertung und Auswahl geschrieben werden. Dabei ist zuerst eine Beschreibung der Methodik zu verfassen und anschließend die Vergleiche nach den einzelnen Kriterien.
20.03.2019	Alle	Besprechung des finalen Layouts
26.03.2019	Heinzl	Besprechung der finalen Diplomarbeitsversion Einige Fehler wurden besprochen. Es ist ein Fazit im Praxiskapitel zu schreiben. Weiters sollten Screenshots der Android-App hinzugefügt werden. Ansonsten ist die Diplomarbeit abgeschlossen.

Besprechungsprotokoll Oliver Hovorka		
Datum	Person(en)	Details
03.12.2018	Hovorka	Es wurde der Inhalt der ersten 10 Seiten der Diplomarbeit besprochen. Der Inhalt war relativ gut jedoch passte die Formatierung nicht ganz. Die Tabellen in den letzten drei Seiten sollen entfernt werden und durch numbered Lists ersetzt werden. Weiters fehlte zu den Kriterien jeweils eine Einleitung und es sind einige Logikfehler vorhanden.
12.02.2018	Hovorka	Es wurde der Kriterienkatalog und die Bewertung von Jersey besprochen. Es sind einige Änderungen notwendig, wie die Trennung von Client und Server APIs, weiters muss einiges Umformuliert werden. Beim schreiben der nächsten Seiten wird vor allem auf die Verwendung der APIs in Android geachtet.
19.03.2018	Hovorka	Es wurde die Bewertung der aller APIs besprochen. Weiters wurde über das Kapitel Verwendung im Projekt gesprochen.
20.03.2018	Alle	Besprechung des Finalen Layouts

Besprechungsprotokoll Maximilian Suchy		
Datum	Person(en)	Details
05.12.2018	Suchy	Der bis jetzt gelieferte Inhalt ist strukturlos, da viele Kapitel auf einmal begonnen wurden. Der Fokus sollte auf die Fertigstellung von Kapitel 1 gelegt werden. Hierzu sollen laut Inhaltsverzeichnis verschiedene Applikationsserver bewertet und miteinander verglichen werden. Zusätzlich soll der Nutzen eines Reverse Proxy wie nginx für das Projekt erläutert werden.
02.02.2019	Suchy	Inhalt und Aufbau sind grundsätzlich OK. Zusätzlich zu GlassFish sollte auch WildFly dokumentiert werden und die beiden miteinander verglichen werden. Datenbanken sollten in ihrem eigenen Kapitel eher weniger projektbezogen beschrieben werden. Stattdessen soll ein neues Kapitel "Umsetzung" hinzugefügt werden, in dem der Einsatz der Software im Projekt näher erläutert wird. Außerdem wurde empfohlen, weniger über Service Management zu schreiben.
20.03.2019	Alle	Gespräch zur Finalisierung
26.03.2019	Suchy	Im REST-Teil soll noch eine ausführliche Beschreibung der angewandten Technologien und Funktionen erarbeitet werden. Hierbei wären Listings mit Code aus dem Projekt sinnvoll. Der Vergleich Jersey/RESTEasy ist aufgrund der Ähnlichkeit dieser beiden Komponenten zu vernachlässigen. Das Kapitel "Datenbanken" ist aus zeitlichen Gründen eventuell vollständig zu entfernen. Dies wird noch evaluiert.