



LABORAUFGABE 04

TECHNISCHE DOKUMENTATION

BENJAMIN BISSENDORF
(5131381)

JONAS EHLERS
(5128684)

HOWHANNES OGANESIAN
(5010012)

SIMON PFENNIG
(5128655)

FYNN SCHERDIN
(5129128)

<i>Fakultät</i>	4 Elektrotechnik und Informatik
<i>Lehrstuhl</i>	Hochschule Bremen
<i>Lehrveranstaltung</i>	Softwaretechniken 2
<i>Lehrperson</i>	Prof. Jasminka Matevska
<i>Studierendenjahrgang</i>	DSI 2020
<i>Abgabedatum</i>	25. Juli 2022

Inhaltsverzeichnis

1	Einführung	1
2	Product-Backlog	2
3	Entwicklung des Gesamtsystems	8
3.1	Architektur und Technologien	8
3.2	Technologien und Werkzeuge	10
3.2.1	Versionsmanagement	10
3.2.2	Entwicklungsumgebungen	10
3.2.3	Sicherheitstechnologien	11
3.2.4	Progressive Web App	11
3.3	Darstellung der Persistenzschicht	12
3.3.1	ERD - Gesamtsystem	12
3.3.2	ERD - Nutzerverwaltung	13
3.3.3	ERD - Buchungssystem	14
3.4	Dynamische Sicht	15
3.4.1	Ablauf Buchung	15
3.4.2	Ablauf Verfügbarkeitsprüfung	17
3.4.3	Kommunikation und Ablauf während einer Anmeldung	18
4	Beschreibung der implementierten Komponenten/Module und Schnittstellen	19
4.1	Komponenten	19
4.2	Schnittstellen	21
5	Verifikation	28
5.1	Beschreibung der durchgeführten Unit-und System Tests	28
5.2	Automatische Tests	28
5.3	Manuelle Tests	29
5.4	Verifikationsmatrix	32
6	Anhang	37

1 Einführung

Die Technische Dokumentation umfasst alle notwendigen Informationen für den ersten funktionsfähigen und verifizierten Prototyp. In der Technischen Dokumentation werden alle umgesetzten Anforderungen, sowie die dazu verwendeten Technologien und Werkzeuge beschrieben. Ebenfalls umfasst diese die komplette Architekturbeschreibung des Gesamtsystems und deren Komponenten.

2 Product-Backlog

Die folgende Tabelle bildet den Product-Backlog zu Projektstart ab. Eine Auflistung der erfüllten Anforderungen befindet sich im Abschnitt 5.4. Die Zeit wird in Arbeitstagen für eine Person angegeben, wobei eine tägliche Arbeitszeit von ca. 2 Stunden angenommen wird.

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1100	Frontend Entwicklung		
1110	Fahrzeugverwaltung		
1111	Fahrzeugübersicht entwickeln	Eine Seite auf der Mitarbeiter alle Fahrzeuge anzeigen, bearbeiten und erstellen können ist implementiert.	2 Tage
1112	Reinigungs- und Wartungsabwicklung entwickeln	Eine Seite auf der Mitarbeiter alle Wartungen oder Reinigungen von Fahrzeuge anzeigen, bearbeiten und anlegen können ist implementiert.	1 Tag
1113	Testen	Alle Komponenten-, Integrations- und Manuellen Tests sind definiert und erfolgreich ausgeführt worden.	4 Tage
1120	Mitgliederverwaltung		
1121-1	Startseite aufsetzen	Eine Seite, die Besucher der Seite empfängt ist implementiert. Sie ist die Startseite der Anwendung und bietet Wege, um auf die Seiten zu gelangen, auf welchen man sich registrieren oder anmelden kann.	2 Tage
1121-2	Anmeldeseite aufsetzen	Eine Seite zum Anmelden an bestehenden Konten wurde implementiert. Diese ermöglicht es seine Nutzerdaten anzugeben, diese am Server zu prüfen und bei erfolgreicher Anmeldung die Information zur Anmeldung zu speichern, sowie auf die Übersichtsseite zu gelangen.	1 Tag
1122	Nutzerdatenänderungsseite entwickeln	Eine Seite auf der Nutzer ihre persönlichen Daten einsehen und bearbeiten können wurde implementiert.	1 Tag
1123	Testen	Alle Komponenten-, Integrations- und Manuellen Tests sind definiert und erfolgreich ausgeführt worden.	3 Tage

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1130	Ausleihsystem		
1131	Zusammenstellungsseite erstellen	Eine Seite auf der Nutzer eine Buchung zusammenstellen, auf Verfügbarkeit prüfen, den Preis ermitteln, und buchen können ist implementiert.	2 Tage
1132	Buchungsänderungsseite erstellen	Eine Seite auf der Nutzer eine bereits getätigte Buchung ändern können ist implementiert.	2 Tage
1133	Buchungsdetailsseite implementieren	Eine Seite auf der Nutzer Details zu einer bereits getätigten Buchung einsehen, diese starten bzw. beenden oder stornieren können ist implementiert.	3 Tag
1134	Buchungsübersicht implementieren	Eine Seite auf der Nutzer alle ihre Vergangenen, Aktuellen und Aktuellen Buchungen einsehen können ist implementiert	2 Tage
1135	Testen	Alle Komponenten-, Integrations- und manuellen Tests sind definiert und erfolgreich ausgeführt worden.	5 Tage
1140	Datenverwaltung		
1141	Mitarbeiterverwaltungsseite erstellen	Eine Seite auf der Administratoren Mitarbeiter anlegen, löschen, ihre Daten anzeigen und ändern können ist implementiert.	1 Tag
1142	Fahrzeugverwaltungsseite erstellen	Eine Seite auf der Mitarbeiter alle Fahrzeuge anzeigen, bearbeiten und erstellen können ist implementiert.	1 Tag
1143	Mitgliederverwaltungsseite erstellen	Eine Seite auf der Mitarbeiter die Daten von Mitgliedern einsehen und diese Autorisieren und Sperren können ist implementiert.	1 Tag
1144	Tarifdatenverwaltungsseite erstellen	Eine Seite auf der Mitarbeiter alle Tarife anzeigen, bearbeiten und erstellen können ist implementiert.	1 Tag
1145	Testen	Alle Komponenten-, Integrations- und manuellen Tests sind definiert und erfolgreich ausgeführt worden.	2 Tage

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1200	Backend Entwicklung		
1210	Fahrzeugverwaltung		
1211	Datenstruktur entwerfen	Modellklassen in Verbindung zur Fahrzeugverwaltung mit den entsprechenden Beziehungen sind erstellt und Tabellen wurden erfolgreich generiert.	2 Tage
1212	API Endpunkte definieren	Die Schnittstellen der Fahrzeugverwaltung sind in der technischen Dokumentation definiert. Die API-Controller sind nach dieser Spezifikation unter Einhaltung der Route, der Parameter und des Ergebnisses aufgebaut und erreichbar. Die Logik dahinter muss nicht implementiert sein.	4 Tage
1213	Anbindung der Endpunkte	Implementierung der Logik des Fahrzeugverwaltungssystem. Die Verbindung der Logik ist mit den definierten API-Controllern realisiert.	1 Tag
1214	Testen	Alle Komponenten-, Integrations- und manuellen Tests sind definiert und erfolgreich ausgeführt worden.	2 Tage
1220	Mitgliederverwaltung		
1221	Datenstruktur entwerfen	Modellklassen in Verbindung zur Mitgliederverwaltung mit den entsprechenden Beziehungen sind erstellt und Tabellen wurden erfolgreich generiert.	2 Tage
1222	API Endpunkte definieren	Die Schnittstellen der Mitgliederverwaltung sind in der technischen Dokumentation definiert. Die API-Controller sind nach dieser Spezifikation unter Einhaltung der Route, der Parameter und des Ergebnisses aufgebaut und erreichbar. Die Logik dahinter muss nicht implementiert sein.	2 Tage
1223	Anbindung der Endpunkte	Implementierung der Logik für die zuvor definierten API Endpunkte.	1 Tag

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1224	Anmeldesystem implementieren	Implementierung einer token-basierten Authentifizierung aller Rest Endpunkte und Integration in ein rollenbasiertes Autorisierungssystem.	4 Tage
1225	Testen	Alle Komponenten-, Integrations- und manuellen Tests sind definiert und erfolgreich ausgeführt worden	2 Tage
1230	Ausleihsystem		
1231	Datenstruktur entwerfen	Modellklassen in Verbindung zum Ausleihsystem mit den entsprechenden Beziehungen sind erstellt und Tabellen wurden erfolgreich generiert.	2 Tage
1232	API Endpunkte definieren	Die Schnittstellen des Ausleihsystems sind in der technischen Dokumentation definiert. Die API-Controller sind nach dieser Spezifikation unter Einhaltung der Route, der Parameter und des Ergebnisses aufgebaut und erreichbar. Die Logik dahinter muss nicht implementiert sein.	1 Tag
1233	Anbindung der Endpunkte	Die in 1232 definierten Endpunkte wurden implementiert. Sie bieten lediglich die Schnittstelle, dessen Logik in 1234 implementiert wird.	2 Tage
1234	Verfügbarkeitsprüfung implementieren	Die Logik der Verfügbarkeitsprüfung ist implementiert. Sie erlaubt aus einer Kombination von Fahrzeugklasse, Abholstation, Abgabestation, und Zeitraum zu berechnen, ob eine Fahrt gebucht werden kann.	5 Tage
1235	Testen	Alle Komponenten-, Integrations- und manuellen Tests sind definiert und erfolgreich ausgeführt worden.	5 Tage
1240	Datenverwaltung		
1241	Datenstruktur entwerfen	Modellklassen in Verbindung zur Datenverwaltung mit den entsprechenden Beziehungen sind erstellt und Tabellen wurden erfolgreich generiert.	2 Tage

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1242	API Endpunkte definieren	Die Schnittstellen der Datenverwaltung sind in der technischen Dokumentation definiert. Die API-Controller sind nach dieser Spezifikation unter Einhaltung der Route, der Parameter und des Ergebnisses aufgebaut und erreichbar. Die Logik dahinter muss nicht implementiert sein.	1 Tag
1243	Anbindung der Endpunkte	Die in 1242 definierten Endpunkte wurden implementiert. Sie bieten die Möglichkeit entsprechende Daten in die Datenbank zu schreiben beziehungsweise zu lesen.	2 Tage
1244	Rechnungsdatenexport implementieren	Es besteht die Möglichkeit die für die Buchhaltung benötigten Rechnungsdaten automatisch aus der Datenbank einzulesen und in einem geeigneten Format zu exportieren.	2 Tage
1245	Testen	Alle Komponenten-, Integrations- und Manuellen Tests sind definiert und erfolgreich ausgeführt worden.	2 Tage
1300	Integration		
1310	Projektintegration		
1311	Projekt aufsetzen	Projekte für Front und Backende im Repo erstellt und mit einem minimalen Funktionstest ausgestattet.	5 Tage
1312	Frameworks integrieren	In beiden Projekten wurden alle genutzten Frameworks(Unit- und Integrationstest), bis auf die Datenbankanbindung eingebunden und mit einem minimalen Funktionstest auf Funktionalität geprüft.	6 Tage
1313	Datenbank verbinden	Das Backende wurde mithilfe des Datenbank-Frameworks mit einer Datenbank verbunden und ein Testmodell in der Datenbank erstellt und an einen Controller angebunden.	4 Tage

Item-ID	Backlog-Item	Definition of Done	Geschätzte Dauer
1314	Softwarearchitektur entwerfen	Es wurde entschieden welche Frameworks genutzt werden sollen und wie diese miteinander interagieren, sowie eine Teststrategie entwickelt.	4 Tage
1320	Entwicklungsumgebung aufsetzen		
1321	CI/CD aufsetzen	Das Projektrepo wurde auf die CI/CD Umgebung Gitlab verschoben und eine Pipeline zur automatisierten Überprüfung der Tests und Compilieren des Quellcodes.	2 Tage
1322	Docker aufsetzen	Automatisierte Erstellung und Bereitstellung von Dockercontainern aus Compilaten.	2 Tage
1330	Deployment		
1331	Initialkonfiguration erstellen	Erstellung einer installionsfertigen Konfiguration.	1 Tag
1332	Abnahmetests durchführen	Alle vorherigen Tests sind erfolgreich durchgeführt worden. Außerdem wurde das Gesamtsystem anhand von Testszenarien erneut erfolgreich getestet.	6 Tage
1333	Auf Zielsystem aufspielen	Das fertige System befindet sich lauffähig auf dem dafür vorgesehenen Zielsystem	2 Tage

Tabelle 1: Backlog

3 Entwicklung des Gesamtsystems

3.1 Architektur und Technologien

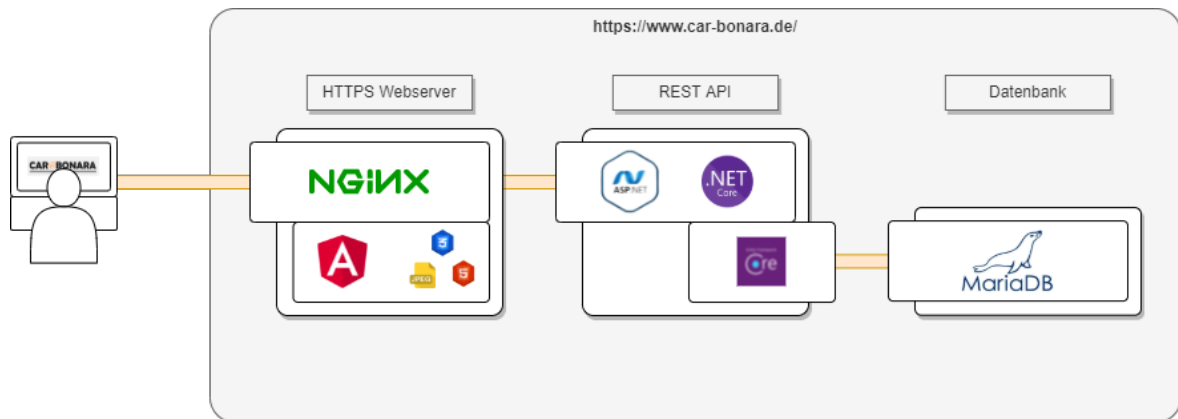


Abbildung 1: Schematische Architektur des Systems mit ihren Technologien

Nach Anforderung des Kunden handelt es sich bei Carbonara um eine Webanwendung, welche von Benutzern, also Kunden und Mitarbeitern gleichermaßen über einen Internet-Browser wie Chrome, Firefox oder Safari aufgerufen und bedient werden kann. Nach Absprache des Kunden wurde sich darauf geeinigt hierbei einen spezielleren Weg zu gehen: Durch Vorerfahrungen mit den betroffenen Technologien wurde sich entschieden eine s.g. Single-Page-Application (SPA) zu entwickeln, welche zum Darstellen von Informationen oder nach Nutzerinteraktionen Anfragen an einen verarbeitenden Server sendet. In Abb. 1 ist dieser Aufbau mit den verwendeten Technologien umrissen. Im Folgenden wird dies weiter erläutert.

Vom Benutzer aus gedacht ist die erste und einzige offensichtliche Berührung mit unserer Anwendung der Webserver im „Frontend“. Über eine sichere HTTPS-Verbindung können über verschiedene Anfragen Daten und Dateien des Servers erhalten werden. Dabei handelt es sich um den Webserver „NGINX“, welcher gleichzeitig auch als Reverse-Proxy dient. Da es für die Art der anfallenden Anfragen entscheidend ist, sollte noch einmal kurz erklärt werden was eine SPA ist: Der Name ist bereits ein Hinweis darauf, dass zum Öffnen der Anwendung einmal eine einzige, dafür etwas größere Seite eingeladen wird. Nach dem Einlesen werden weitere statische Inhalte, wie Bilder und Styledateien eingeladen, wonach die Anwendung aufgebaut und benutzbar ist. Interagiert und navigiert der Nutzer nun innerhalb der Anwendung, wird innerhalb der Stammseite der Inhalt entsprechend mit den am Anfang eingeladenen Daten ausgetauscht. Erst bei tatsächlicher inhaltlicher Interaktion mit dem System, bei dem dynamische Daten benötigt oder ein Nutzer Änderungen an Daten vornehmen möchte, muss weiter mit dem Server kommuniziert werden. Daraus ableitbar sind die beiden Anfragetypen, welche Nginx entgegennehmen muss: Statische Inhalte, wie die Webseite und ihre Ressourcen, die sich nicht verändern, sowie dynamische Inhalte, die vom Server berechnet werden müssen. Erstere werden direkt beim Nginx gespeichert und sind meist durch Caching-Verfahren besonders schnell

erreichbar. Eine Anfrage an die dynamischen Daten kann jedoch nicht selbst beantwortet werden. Hierbei dient die Reverse-Proxy-Funktion, mit der Nginx die Anfrage an die hier in der Grafik betitelte „REST API“ weiterleitet und darauf wartet dessen Antwort herauszugeben.

Angular ist ein von Google entwickeltes Framework für Single-Page-Applications, welches hier für die Anwendung zum Einsatz kommt. Typisch wäre hierbei, dass neben der Beschreibung der Darstellung der Seiten mit HTML und CSS auch Javascript als defacto-Standard für dynamische Seitenprogrammierung im Internet genutzt würde. Doch hier hat man sich für Typescript, einem typisierten Javascript (bzw. ECMAScript) Ableger, entschieden, der als Mischung aus Javascript, Java und C bezeichnet werden könnte. Weiterhin ist Angular komponentenbasiert, wodurch alle darstellbaren Einzelteile der Anwendung für sich ihren Aufbau, ihr Aussehen und ihr Verhalten mit klar definierten Abhängigkeiten beschreiben. Eine Komponente kann außerdem durch diese Modularisierung selbst andere Komponenten in ihrer Darstellung einbinden. Als SPA besitzt Angular als Hauptseite auch eine Komponente, die nun aus der aufgerufenen URL mit vorher definierten Routen heraus entscheidet welche Komponente nun in ihr eingebunden wird. Als fertige Anwendung entsteht so ein komplexes, aber übersichtliches und mächtiges Gesamtsystem, welches nach den Wünschen der Kunden auch komplexe Anforderungen umsetzbar macht. Für das (kontinuierliche) Ausliefern werden dann alle Komponenten durch den Typescript-Compiler gebündelt und als einzelnes Modul mit wenigen Dateien in Javascript und HTML übersetzt, welches dann über den e.g. Nginx-Webserver als statischer Inhalt ausgeliefert werden kann.

Für dynamische Anfragen ist eine Komponente erforderlich, die diese entgegennimmt und verarbeitet. Bei Carbonara ist dies ein in C programmiertes Backend, welches als „dotnet 6.0“-„ASP.NET“ Projekt mit seinen REST-Controllern eine einheitliche API bereitstellt. Intern nehmen die verschiedenen Controller diese Anfragen nach einer Authentifizierungs- und Autorisierungsprüfung entgegen und stellen Informationen über übergebene Parameter bereit. Entweder werden daraufhin bei simpleren Anfragen direkt Ergebnisse, bspw. aus der Datenbank, geliefert, oder es werden weiter Services aufgerufen, die ein Ergebnis berechnen und/oder Manipulationen an der Datenbank vornehmen. Letztendlich wird ein Ergebnis optionalerweise auch mit Inhalt, sowie einem Statuscode zurückgegeben, welches dann vom Nginx an den Aufrufer als Antwort gesendet wird.

Für die Kommunikation mit der Datenbank wird nicht direkt mit SQL-Befehlen im Programmcode gearbeitet. Für ein angenehmeres typisiertes Programmieren, um Fehlerfälle zu minimieren und um eine SQL-Injection auszuschließen, wird ein sogenannter Object-Relational-Mapper verwendet, der aus vorher angegebenen Typen, ihren Attributen und Beziehungen sich automatisiert mit einer Datenbank verbindet und - wenn noch nicht vorhanden - selbstständig Tabellendefinitionen generiert, sowie anschließend anlegt. Mit EntityFramework, welches diese Aufgabe in Carbonara übernimmt, stehen bekannte semantische Strukturen aus C bereit, mit denen komfortabel auf die Daten der Datenbank durch schnell zu erstellende und leicht lesbare Queries zugegriffen werden kann, die sowohl typisierte Parameter entgegennehmen als auch typisierte Antworten liefern. Für die Persistenz selbst wurde MariaDB als Datenbankmanagementsystem ausgewählt, da dieses als beliebter quelloffener Nachfolger des bewährten MySQL-Servers keine Lizenzen benötigt und gleichzeitig schnell und ausgereift ist.

3.2 Technologien und Werkzeuge

3.2.1 Versionsmanagement

Das Entwicklungsteam arbeitet mit all seinen erstellten Artefakten vollständig unter Versionskontrolle.

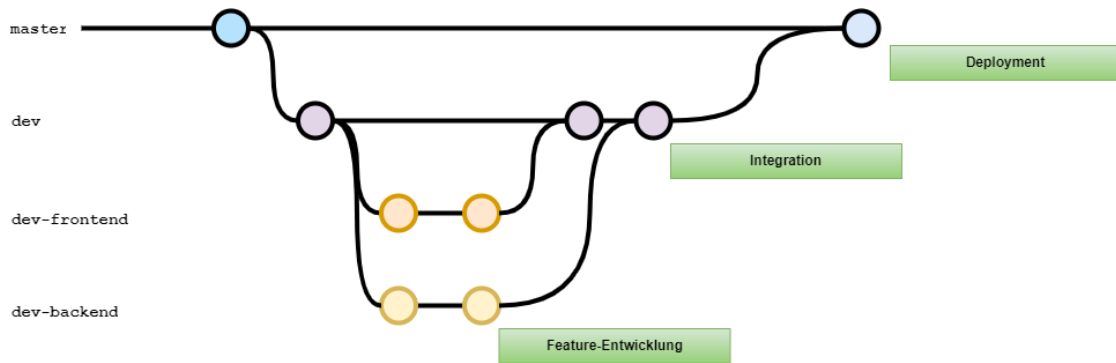


Abbildung 2: Repository Struktur

Wie in Abbildung 2 zu erkennen, haben wir unser Repository in 4 Branches aufgeteilt: Die Entwicklung fand hauptsächlich auf den Branches „dev-backend“ und „dev-frontend“ statt. Dort wurden jeweils Features des Frontends, bzw. des Backends unabhängig anhand der vorher definierten Schnittstellen entwickelt. Nachdem ein Feature auf beiden Branches entwickelt wurde, haben wir beide Branches auf den Branch „dev“ gemerged und die Features dort integriert. Nachdem das Feature getestet wurde, haben wir es auf den Branch „master“ gemerged, wo die neuen Features über eine CI-CD-Pipeline deployed wurden. Dafür wurde Gitlab als Hub des Repositories eingesetzt, welches nach dem Merge auf den Master-Branch automatisch die kontinuierliche Integration, sowie anschließend das kontinuierliche Ausliefern anstößt. Der beim Anbieter IONOS gehostete Server erhält die Information und kann so innerhalb weniger Minuten die neue Version ausliefern und anbieten.

3.2.2 Entwicklungsumgebungen

Durch die Aufteilung der Technologien im Front- und Backend, sowie der daraus resultierenden Unterschiede in der Programmiersprache und Umgebung, werden hier auch unterschiedliche Entwicklungsumgebungen eingesetzt. Für das C-Projekt im Backend wird das Visual Studio 2022 in der Community-Edition verwendet, da eine Integration vieler nützlicher Features zum Testen und Programmieren bietet. Im Frontend verwenden wir für die Angular-Anwendung den Text-Editor Visual Studio Code mit mehreren Erweiterungen für eine Git-, Typescript- und Angular-Unterstützung. Für das Kompilieren wird die Angular CLI eingesetzt, die für die Entwicklung gleichzeitig mit Webpack auch einen Webserver bereitstellt, der mit jeder Änderung im Code direkt auch das Kompilat aktualisiert.

Für die Tests wurde das Test-Framework xUnit eingesetzt, welches eine problemlose Integration im Visual Studio bietet.

3.2.3 Sicherheitstechnologien

Für die Authentifizierung und Autorisierung verwenden wir durch den Server ausgegebene Json-Web-Tokens. Diese sind zustandslose, ablaufende Sicherheitsschlüssel, die mit jeder API-Anfrage mitgesendet werden müssen. Dort kann der Server dem Token entnehmen um welchen Nutzer es sich bei der Anfrage handelt und welche Rechte dieser besitzt. Durch Einschränkungen der API kann dann begrenzt werden welche Nutzergruppen verschiedene Endpunkte aufrufen können. Hierzu ist eine initiale Anmeldung erforderlich, bei der sich der Nutzer mit der Eingabe einer Email und eines Passwortes authentifiziert und als Antwort einen solchen Token erhält. Dieser Token wird gespeichert und nun mit jeder weiteren Anfrage mitgeliefert. Zum Abmelden reicht es aus den Token zu löschen oder die Ablaufzeit verstreichen zu lassen. Mittels kryptografische Verfahren ist es nur dem Server möglich diese Token auszustellen, da ein gefälschter Token die interne Prüfung bei jeder Abfrage nicht bestehen würde. Innerhalb des Token werden dann die erforderliche Daten wie die eindeutige Nummer des Benutzers, seine Rollen, der Aussteller und die Ablaufzeit. Durch den Einsatz von Token muss nicht mit jeder Anfrage die Kombination aus Email und Passwort gesendet werden und gleichzeitig muss der Server nicht alle bestehenden Sessions verwalten. Bei der Implementierung griffen wir auf ausreichend getestete Bibliotheken zurück, die eine semi-automatische Ausstellung durchführen. Schematisch haben wir uns dabei an die RFC6749 gehalten, wichen jedoch entscheidend ab: Normalerweise findet eine Ausstellung eines Refresh-Tokens statt, welcher eine lange Laufzeit hat und zum Anfordern eines Access-Tokens verwendet werden kann. Der Access-Token, mit jeweils kurzer Laufzeit, wird dann bei jeder Anfrage gereicht, um fortlaufend die Rolleninformationen im Token zu aktualisieren. Auf diese Trennung haben wir nach einer Kosten-/Nutzenanalyse verzichtet. Für eine sichere Übertragung von Nutzerdaten wird vom Live-Server ausschließlich HTTPS als Kommunikationsprotokoll zugelassen. Eine HTTP-Anfrage wird automatisch auf HTTPS umgeleitet.

3.2.4 Progressive Web App

Wie bereits beschrieben, handelt es sich bei der Webanwendung um eine SPA. In ihr wird das Grafikframework Angular-Materialdesign eingesetzt, wodurch eine bessere Optik erzielt wurde. Hierdurch ist die Seite responsiv, kann also sowohl auf großen Bildschirmen, wie Laptops oder PCs, als auch auf kleinen Bildschirmen, wie sie bei Tablets oder Smartphones vorkommen, benutzt werden. Mit diesem Verhalten lag es nah die Anwendung in eine Progressive Web App zu verwandeln, welche es möglich macht auf diesen Geräten durch den Browser installiert zu werden. Somit wäre theoretisch auch ein offline-Verhalten möglich, welches in zukünftigen Versionen ergänzt werden kann.

3.3 Darstellung der Persistenzschicht

3.3.1 ERD - Gesamtsystem

Das folgende Diagramm zeigt ein Entity-Relationship-Diagramm des Gesamtsystems. Da dieses relativ komplex ist haben wir es im folgenden in 2 Teildiagramme aufgeteilt, welche zum einen die Nutzerverwaltung und zum anderen das Buchungssystem abbilden und diese genauer beschrieben.

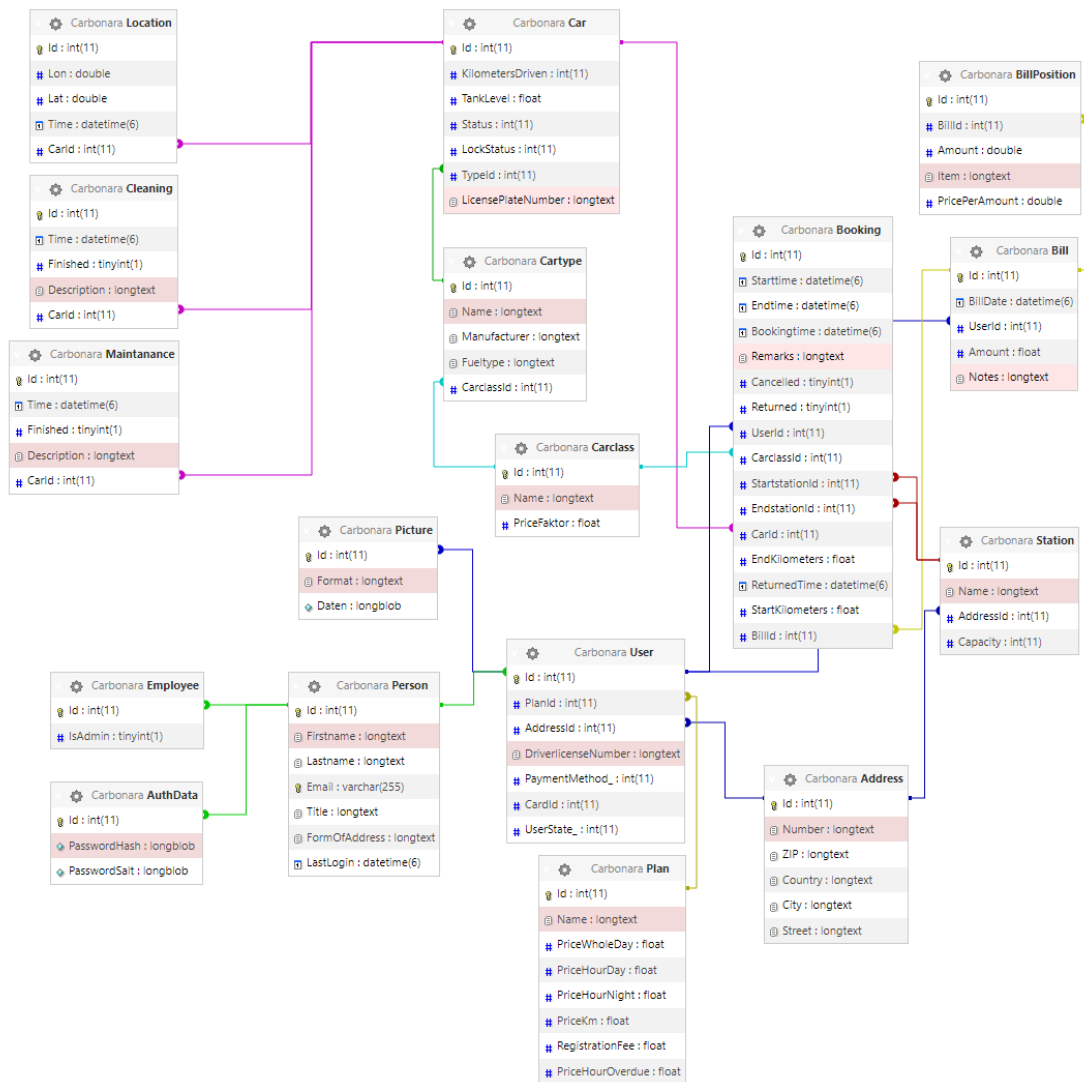


Abbildung 3: ERD Gesamtsystem

3.3.2 ERD - Nutzerverwaltung

Das folgende Diagramm bildet die Nutzerverwaltung ab. Herzstück dieser ist die Person. Diese enthält persönliche Informationen, welche sowohl für Nutzer als auch für Mitarbeiter gebraucht werden. Beide Relationen Employee und User haben jeweils einen Fremdschlüssel, welcher auf eine Person verweist und gleichzeitig als Primärschlüssel dient. Somit ist eine Vererbung realisiert, in der beide Klasse von Person erben. Außerdem ist jeder Person in der Relation AuthData ein gehashtes und gesalzenes Passwort zugeordnet, welches zusammen mit der E-Mail zur Authentifizierung dient. Einem Nutzer sind außerdem eine Adresse, ein Tarif (Plan) und ein Bild zugeordnet. Ein Tarif enthält Informationen zu den Konditionen, zu welchem der Nutzer eine Buchung buchen kann, während das Bild den Führerschein des Nutzers zeigen soll, welchen er während der Registrierung hochgeladen hat, damit ein Mitarbeiter den Kunden verifizieren kann. Letztere Anforderung ist allerdings noch nicht erfüllt.

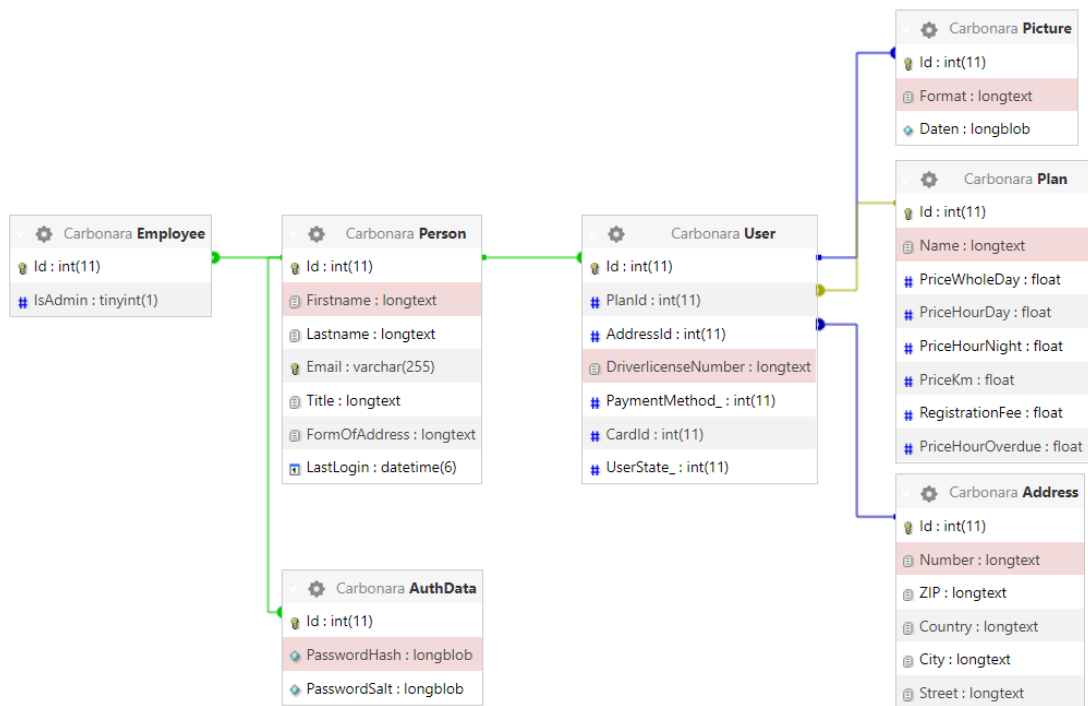


Abbildung 4: ERD Nutzerverwaltung

3.3.3 ERD - Buchungssystem

Das nächste ERD beschreibt das Buchungssystem. Die Zentrale Relation ist dabei die Buchung. Wenn eine Buchung erstellt wird, wird ihr immer eine Start-, und Endstation sowie ein Zeitraum, ein Nutzer und eine Fahrzeugklasse zugeordnet. Jede Station hat dabei eine bestimmten Adresse. Erst mit antreten wird einer Buchung zusätzlich zu ihrer Fahrzeugklasse auch ein bestimmtes Fahrzeug zugeordnet, dieses hat, über die Relation Cartype eine Zuordnung zu einer Fahrzeugklasse. Bei Buchungsantritt, wird also ein der Fahrzeugklasse entsprechendes Auto ausgewählt und zugeordnet. Ist eine Buchung beendet wird eine Rechnung erstellt und der Buchung zugeordnet. Dazu wird die Information des Tarifes des Nutzers benötigt, welcher der Buchung zugeordnet ist. Für die einzelnen Preiselemente wird dann eine Entsprechende Position auf der Rechnung erstellt und in dieser gespeichert.

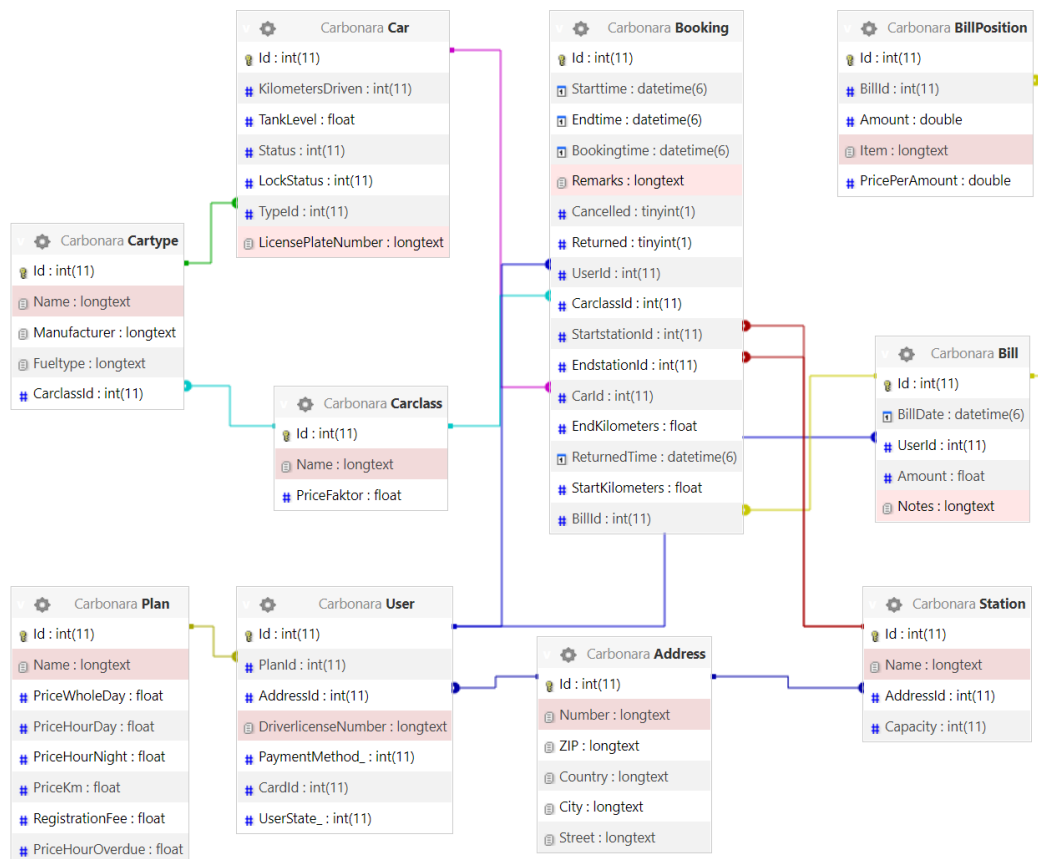


Abbildung 5: ERD Buchungssystem

3.4 Dynamische Sicht

Im folgenden werden einige der wichtigsten Abläufe der Systems dargestellt.

3.4.1 Ablauf Buchung

Das folgende Diagramm zeigt den Ablauf eines Buchungsvorgang durch einen Kunden. Nicht abgebildet ist hier die Möglichkeit des Kunden den Vorgang jederzeit abubrechen, indem er die Seite verlässt.

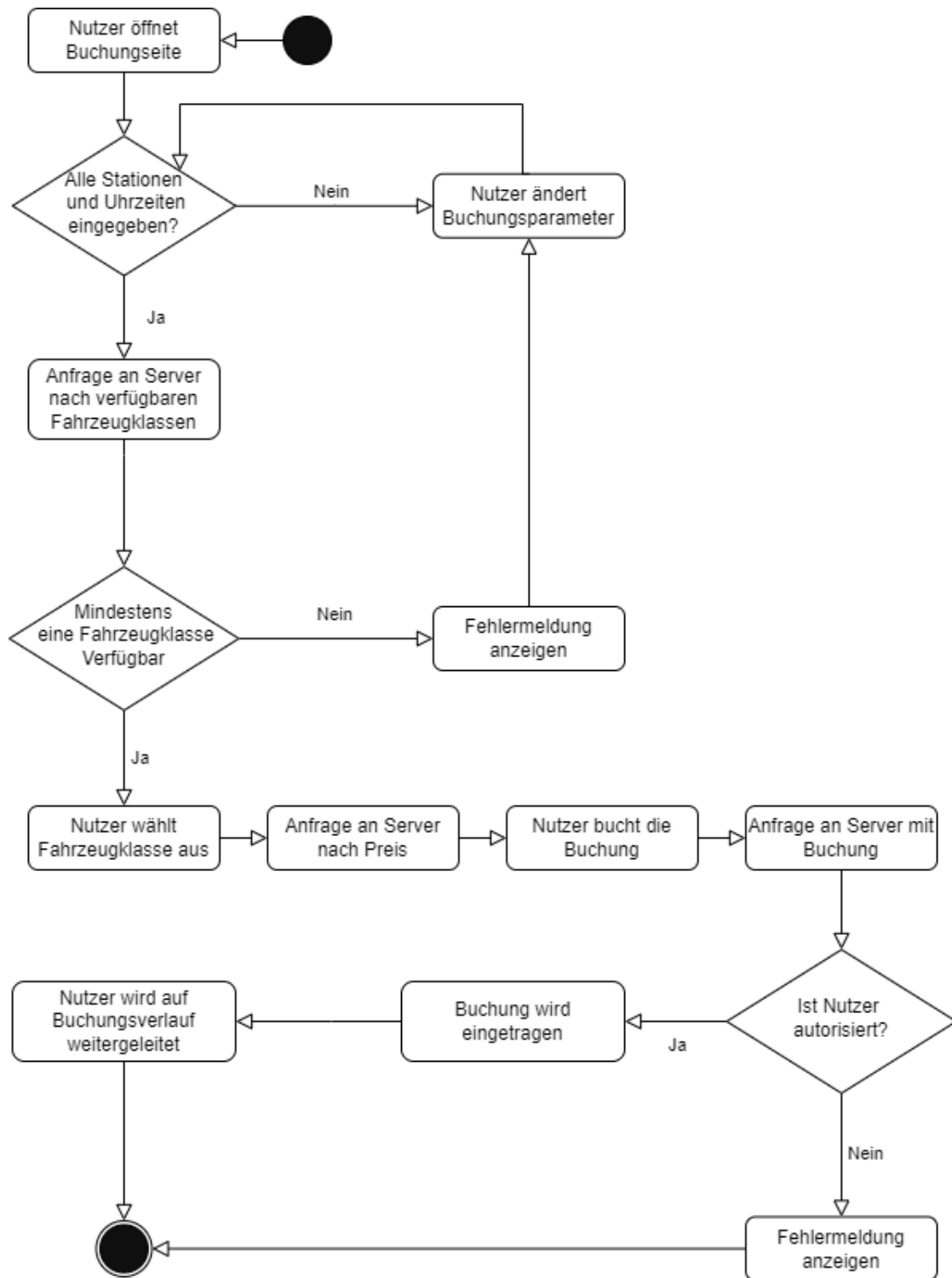


Abbildung 6: Ablauf einer Buchung

3.4.2 Ablauf Verfügbarkeitsprüfung

Das folgende Diagramm zeigt den Ablauf einer Verfügbarkeitsprüfung. In dieser wird festgestellt, ob eine von einem Kunden zusammengestellte Buchung durchführbar ist. Die Abfrage der Verfügbaren Fahrzeugklassen für eine Buchung benutzt diese ebenfalls, und überprüft für jede Fahrzeugklasse, ob die Buchung möglich wäre. Analog zu diesem Ablauf ist die Überprüfung, ob eine Buchung storniert oder geändert werden kann. Nur werden bei dieser Überprüfung keine Einträge in die Liste hinzugefügt sondern die der Buchung entsprechenden entfernt bzw. geändert.

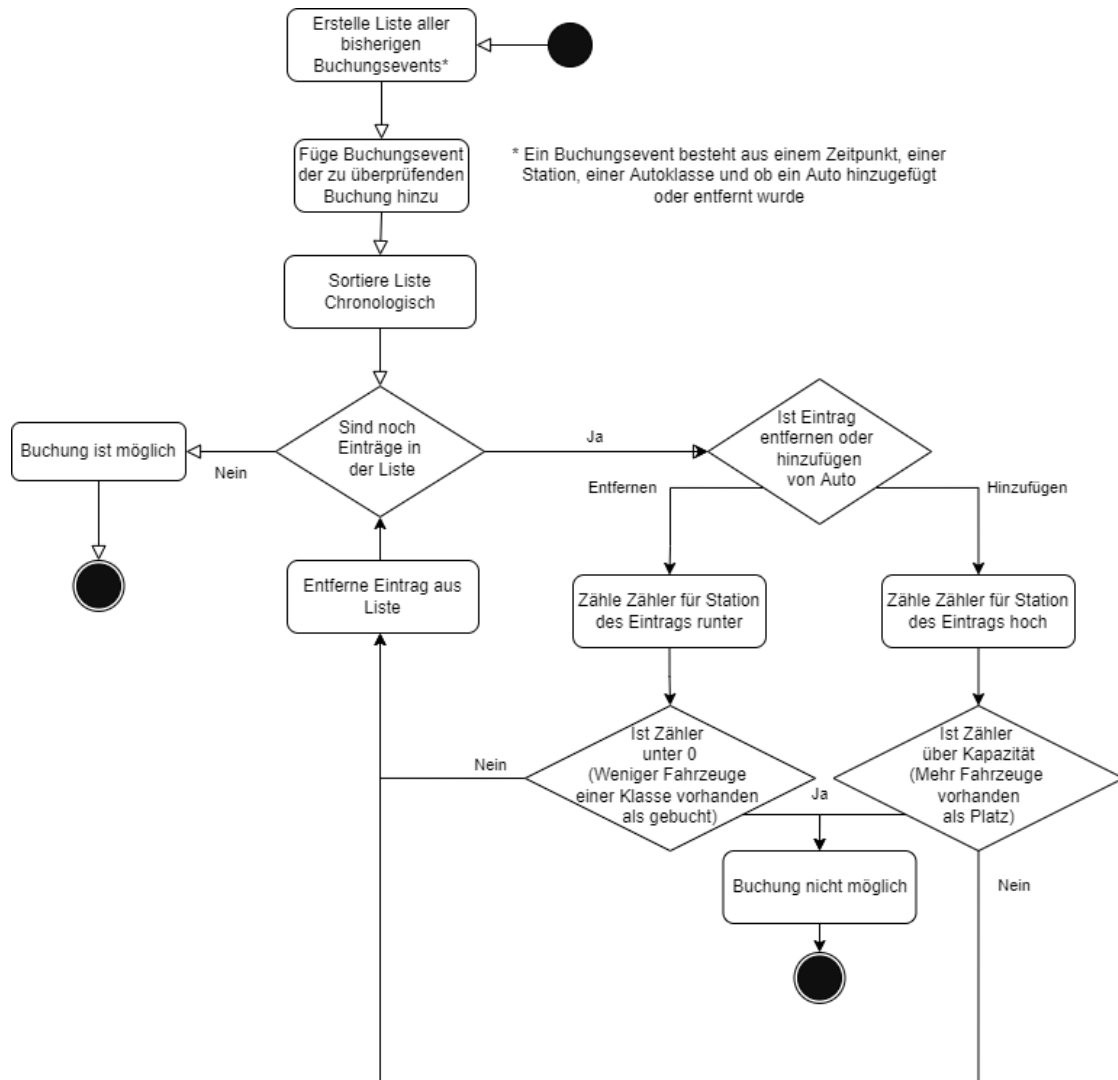


Abbildung 7: Ablauf einer Verfügbarkeitsprüfung

3.4.3 Kommunikation und Ablauf während einer Anmeldung

Das folgende Diagramm beschreibt den Ablauf einer Anmeldung in der Anwendung. Ausgehen vom Gast öffnet dieser die Anmeldeseite im Browser, um dort seine Daten einzugeben. Bestätigt er seine Eingabe, startet der Browser eine Anfrage an den Authentifizierungsserver, in der diese Daten verschlüsselt übertragen werden. Dort werden die Daten entgegengenommen und der Salt für den entsprechenden Nutzer aus der Datenbank erfragt, ist aber eventuell null. Nun wird weiter der Passworthash des Benutzers aus der Datenbank anhand seiner Email erfragt und mit dem neu generierten Hash aus der Anfrage verglichen. Hierbei findet nun eine Aufteilung statt: Existiert der Benutzer nicht, bzw. ist die Kombination ungültig, sendet der Server eine Antwort mit dem Statuscode 401, woraus der Browser ableiten kann, dass die eingegebenen Daten falsch waren. Ansonsten wird ein neuer JsonWebToken für diesen Benutzer und seinen Rollen erstellt, der 12 Stunden gültig bleibt und dem Benutzer solange als Zugangsschlüssel dient. Dieser wird dann in einer Antwort mit dem Statuscode 200 an den Browser gesandt, welcher diesen speichert und interpretiert. Anschließend kann der Benutzer auf seine Startseite navigiert werden.

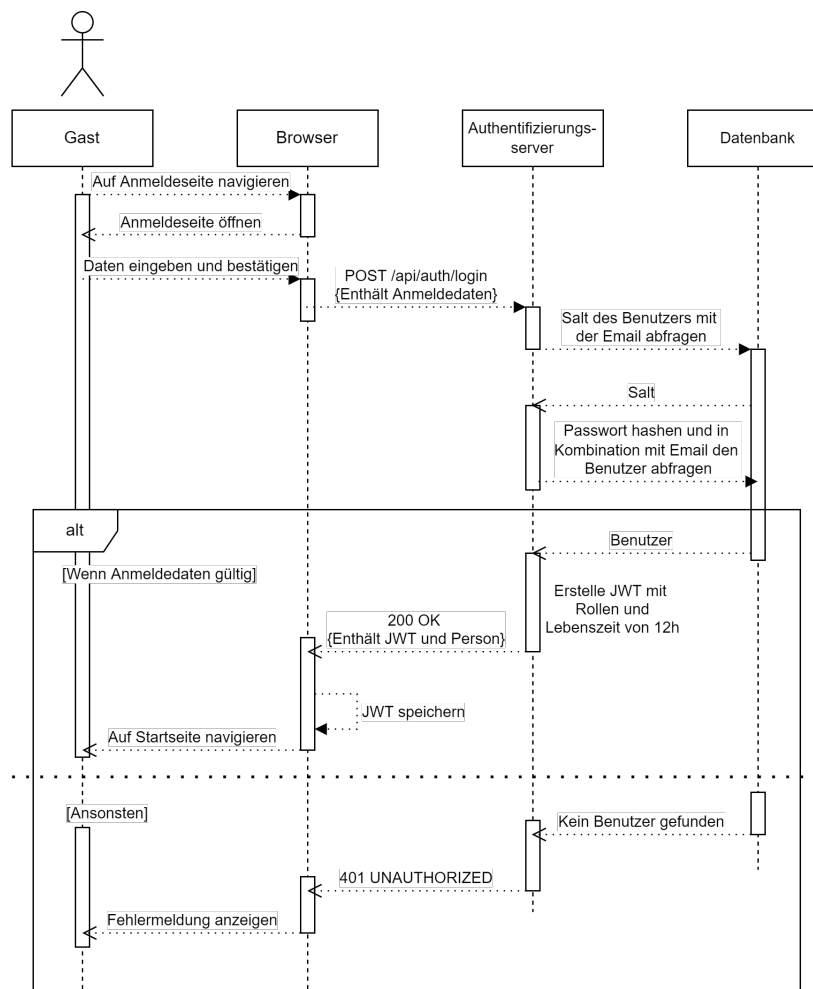


Abbildung 8: Ablauf eines Anmeldevorgangs

4 Beschreibung der implementierten Komponenten/Module und Schnittstellen

4.1 Komponenten

In der nachstehenden Grafik 9 ist eine schematische Darstellung der einzelnen Komponenten und Schnittstellen zu sehen. Sie zeigt, welche Komponenten des Frontends mit welchen Komponenten des Backends kommunizieren. Die meisten Komponenten kommunizieren über mehr als eine Schnittstelle miteinander. Eine vollständige Auflistung und Beschreibung der zwischen Front- und Backend genutzten Schnittstellen kann in Tabelle 3 eingesehen werden. Die Grundsätzliche Architektur sieht vor, dass es einen Controller im Backend der einen Bestimmten Aufgabensatz übernimmt. Im Backend gibt es jeweils einen Service, welcher diesen Controller anspricht und so dem Frontend die entsprechenden Dienste zur Verfügung stellt. Folgende Komponenten wurden genutzt:

Frontend-Service	Backend-Controller	Aufgaben
Network	Ping	Pings austauschen, um den Frontend zu signalisieren, dass das Backend erreichbar ist
Authentication	Auth	Login und Authentifizierung von Nutzern und Mitgliedern, Übernahme von Nutzern durch Mitarbeiter
Admin	Admin	Verwalten (Anlegen, Löschen, Ändern, Einsehen) von Mitarbeitern durch Administratoren
Booking	Booking	Buchungen, ihre Verfügbarkeit, Preis, Buchung und Stornierung, Rechnungen anzeigen
Car	Car	Auf- und Zuschließen von Autos, Aktuelles Auto einer Buchung anzeigen
UserManagement	UserManagement	Verwalten, Freischalten von Nutzern durch Mitarbeitern
User	User	Registrieren, Ändern von Nutzerdaten, Schlüsselkarten anfragen
Database	Station-Database	Anlegen, Einsehen und Ändern von Stationen durch Mitarbeiter
Database	Plan-Database	Anlegen, Einsehen und Ändern von Tarifen durch Mitarbeiter
Database	Cartype-Database	Anlegen, Einsehen und Ändern von Fahrzeugmodellen durch Mitarbeiter
Database	Carclass-Database	Anlegen, Einsehen und Ändern von Fahrzeugklassen durch Mitarbeiter
Database	Car-Database	Anlegen, Einsehen und Ändern von Fahrzeugen durch Mitarbeiter

Tabelle 2: Komponentenbeschreibung

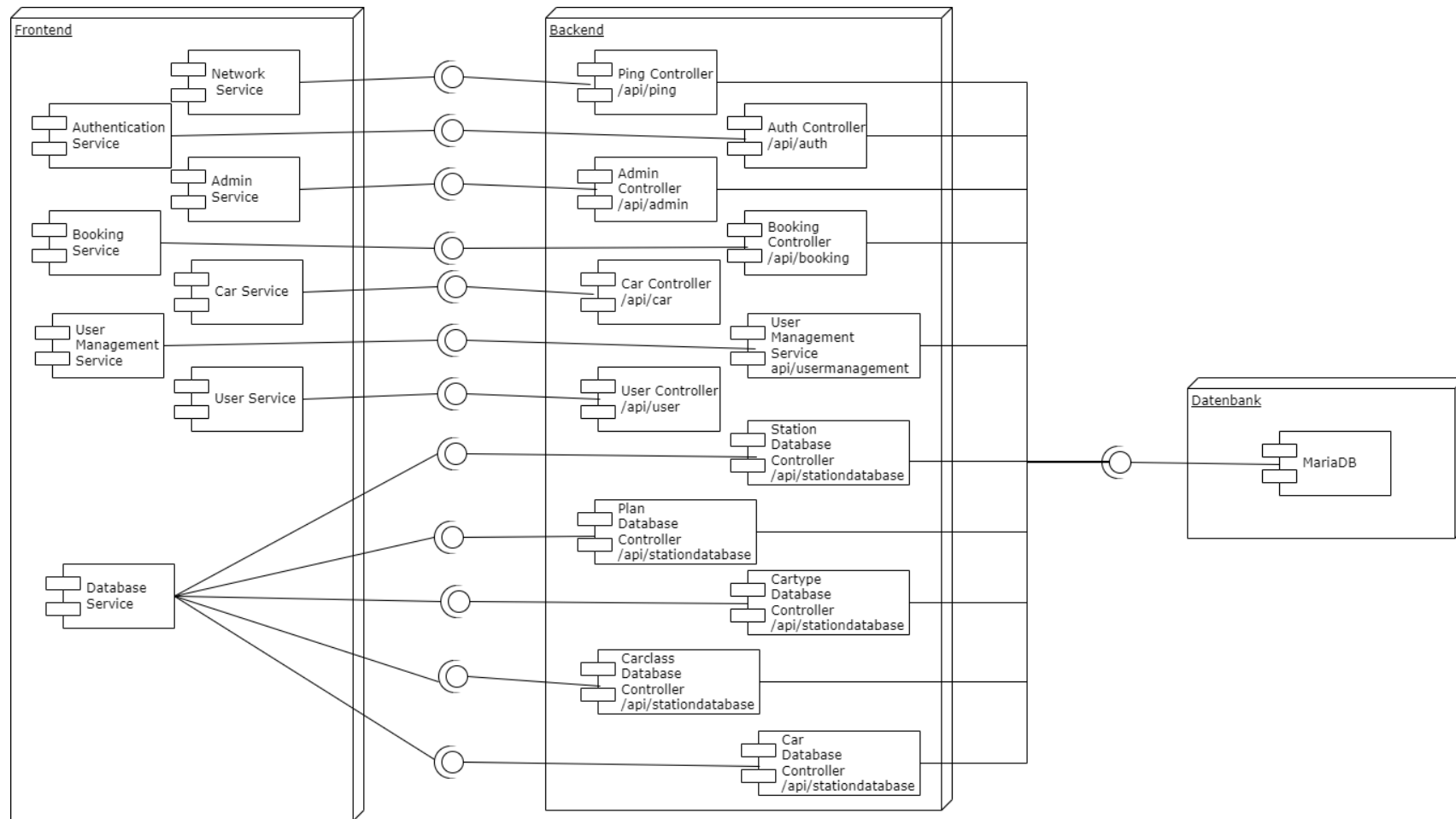


Abbildung 9: Komponentendiagramm

4.2 Schnittstellen

Die Nachfolgende Tabelle enthält alle REST-API Schnittstellen, die das Backend zur Verfügung stellt. Die Autorisierung lässt sich wie folgt Interpretieren.

ANONM : Dies ist ein nicht angemeldeter Benutzer (Gast).

SGDIN : Ein angemeldeter Benutzer, welcher noch nicht Autorisiert wurde. Dieser hat keine Berechtigung ein Fahrzeug zu buchen.

AUTHU : Ein angemeldeter Benutzer, welcher von einem Mitarbeiter Autorisiert wurde und dadurch Fahrzeuge buchen kann.

EMPLE : Ein Mitarbeiter mit anderen Berechtigungen als ein Benutzer, um seine Arbeitstätigkeiten auszuführen.

ADMIN : Ein Mitarbeiter mit der zusätzlichen Berechtigung Mitarbeiter zu verwalten.

Methode	Endpunkt	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
PUT	/api/booking/ available	BookingDTOIn	Float (Preis der Buchung). Ist negativ, wenn nicht verfügbar.	Schnittstelle zum Abrufen, ob eine Buchung zum Reservieren verfügbar wäre.	ANONM
PUT	/api/booking/ availablecarclasses	BookingDTOIn (Ohne Carclass)	CarclassDTO[]	Gibt Fahrzeugklassen zurück, die für die geplante Buchung verfügbar sind.	ANONM
POST	/api/booking/ book	BookingDTOIn	Status 200 400	Bucht angegebene Buchung, wenn alle Voraussetzungen erfüllt sind.	AUTHU
PUT	/api/booking/ isChangePossible/ {bookingId:int}	BookingDTOIn (Neue Buchung)	Boolean 403 wenn verbotener Zugriff	Prüft, ob Buchung änderbar ist.	AUTHU
PUT	/api/booking/ change/{bookingId:int}	BookingDTOIn (Neue Buchung)	Status 200 400	Ändert eine Buchung. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben.	AUTHU
PUT	/api/booking/ cancel/{bookingId:int}	-	Status 200 400 403	Storniert eine Buchung. Ist dies nicht möglich, ist der Antwortstatus 400.	AUTHU
GET	/api/booking/ history	-	BookingDTOOut[]	Gibt eine Liste aller Buchungen des angemeldeten Nutzers zurück.	SGDIN
GET	/api/booking/ getBill/{bookingId:int}	-	BillDTO null	Gibt die der Buchung zugehörige Rechnung oder null zurück wenn noch keine Rechnung zugeordnet wurde.	SGDIN
GET	/api/booking/ bills	-	BillDTO[]	Gibt alle dem Benutzer zugeordneten Rechnungen zurück.	SGDIN

Methode	Endpunkt	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
PUT	/api/booking/ start/{bookingId:int}	-	CarDTO 400	Aktiviert eine Fahrt, die ein Fahrzeug zuweist und entsperren macht. Das zugewiesene Auto wird als Ergebnis zurückgegeben. Ist die fahrt bereits abgelaufen wird die Fahrt direkt beendet und eine Rechnung angelegt und Status 400 zurückgegeben.	AUTHU
PUT	/api/booking/ finish/{bookingId:int}	-	Status 200 400	Beende eine bereits gestartete Fahrt und erstellt eine Rechnung.	AUTHU
POST	/api/auth/ login	LoginRequest	LoginDTO 401	Anmelden eines Mitglied- oder eines Mitarbeiterkontos. Gibt ein LoginDTO bei Erfolg und Status 401 bei falschen Anmeldedaten zurück.	ANONM
PUT	/api/auth/ change	[oldPassword, newPassword]	200 401	Ändert das Passwort, wenn das alte Passwort mit dem gespeicherten übereinstimmt.	SGDIN
GET	/api/auth/ impersonate/{userid:int}	-	LoginDTO 400	Übernehmen eines Mitgliedes durch einen Mitarbeiter. Gibt einen entsprechenden JWT zurück. Ist kein Mitglied mit dieser Id vorhanden Status. 400	AUTHU

Methode	Endpunkt	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
GET	/api/admin/ get/{id:int}	-	EmployeeDTO 400	Erhalte einen Mitarbeiter mit der angegebenen ID.	ADMIN
GET	/api/admin/ getall	-	EmployeeDTO[]	Alle Mitarbeiter abfragen.	ADMIN
POST	/api/admin/ add	EmployeeDTO	-	Legt neuen Mitarbeiteraccount an.	ADMIN
PUT	/api/admin/ change/{id:int}	EmployeeDTO	200 400	Ändert Mitarbeiter Informationen.	ADMIN
DELETE	/api/admin/ delete/{id:int}	-	200 400	Löscht Mitarbeiter Account.	ADMIN
GET	/api/carclassdatabase/ get/{id:int}	-	CarclassDTO 400	Autoklasse abfragen.	ANONM
GET	/api/carclassdatabase/ getall	-	CarclassDTO[]	Alle Autoklassen abfragen.	ANONM
POST	/api/carclassdatabase/ add	CarclassDTO	200	Autoklasse einpflegen.	EMPLE
PUT	/api/carclassdatabase/ change/{id:int}	CarclassDTO	200 400	Autoklasse ändern.	EMPLE
GET	/api/cardatabase/ get/{id:int}	-	CarDTO Status 400	Auto abfragen.	EMPLE
GET	/api/cardatabase/ getall	-	CarDTO[]	Alle Autos abfragen.	EMPLE
POST	/api/cardatabase/ add	AddCarDTO	Status 200	Auto einpflegen. Es wird eine Initiale Buchung zum Startzeitpunkt an der Startstation eingepflegt.	EMPLE
PUT	/api/cardatabase/ change/{id:int}	CarDTO	Status 200 400	Auto ändern.	EMPLE

Method	Endpoint	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
GET	/api/cartypedatabase/ get/{id:int}	-	CartypeDTO Status 400	Autotyp abfragen.	ANONM
GET	/api/cartypedatabase/ getall	-	CartypeDTO[]	Alle Autotypen abfragen.	ANONM
POST	/api/cartypedatabase/ add	CartypeDTO	Status 200	Autotyp einpflegen.	EMPLE
PUT	/api/cartypedatabase/ change/{id:int}	CartypeDTO	200 400	Autotyp ändern.	EMPLE
GET	/api/plandatabase/ get/{id:int}	-	PlanDTO 400	Tarif abfragen.	ANONM
GET	/api/plandatabase/ getall	-	PlanDTO[]	Alle Tarife abfragen.	ANONM
POST	/api/plandatabase/ add	PlanDTO	200	Tarif einpflegen.	EMPLE
PUT	/api/plandatabase/ change/{id:int}	PlanDTO	200 400	Tarif ändern.	EMPLE
Method	Endpoint	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
GET	/api/stationdatabase/ get/{id:int}	-	StationDTO 400	Station abfragen.	ANONM
GET	/api/stationdatabase/ getall	-	StationDTO[]	Alle Stationen abfragen.	ANONM
POST	/api/stationdatabase/ add	StationDTO	200	Station einpflegen.	EMPLE
PUT	/api/stationdatabase/ change/{id:int}	StationDTO	200 400	Station ändern.	EMPLE
GET	/api/ping	-	200	Sendet einen Ping an den Server.	ANONM

Methode	Endpunkt	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
PUT	/api/user/update/	UserDTO (Veränderte Daten)	Status 200 400	Ändert die Daten von dem derzeit eingeloggten User. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben.	SGDIN
PUT	/api/user/register/	RegistrationComposite (UserDTO (Neue Daten für die Registrierung) + Passwort)	Status 200 400	Registriert einen neuen Account für den Gast mit den eingegebenen Werten. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben	ANONM
PUT	/api/user/getnewcard/	-	Status 200 400	Ändert die Karte bzw. KartenId von dem derzeit eingeloggten User. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben.	SGDIN
GET	/api/user/current/	-	Status 200 400	Gibt die Daten des aktuellen angemeldeten Users zurück. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben.	SGDIN
GET	/api/usermanagement/getuserlist/	-	UserDTO[]	Gibt alle User in Form von UserDTO[] zurück. Existieren keine Nutzer ist das Array leer.	EMPLE
PATCH	/api/usermanagement/userstatus/	UserDTO (neuer Userstatus)	Status 200 400	Setzt den Satus eines Benutzers. Wenn hierbei ein Fehler unterläuft, wird Status 400 zurückgegeben.	EMPLE

Tabelle 3: API Endpunkte

Methode	Endpunkt	Body-Parameter	Rückgabe	Beschreibung	Autorisierung
GET	/api/car/ car/{bookingId:int}	-	CarDTO 400	Erhält das einer Buchung zugeordnete Auto oder Status 400 wenn noch keins zugeordnet ist.	AUTHU
PUT	/api/car/ lock/{bookingId:int}	-	200 400	Schließt das der Buchung zugeordnete Auto ab oder liefert 400 wenn noch keins zugeordnet ist.	AUTHU
PUT	/api/car/ unlock/{bookingId:int}	-	200 400	Schließt das der Buchung zugeordnete Auto auf oder liefert 400 wenn noch keins zugeordnet ist.	AUTHU
GET	/api/car/ km/{Id:int}	-	float 400	Gibt den aktuelle Kilometerstand des Fahrzeugs aus oder 400 wenn der Id kein Fahrzeug zugeordnet ist.	EMPLE
GET	/api/car/ booking/{Id:int}	-	BookingDTOOut Null	Gibt die dem Fahrzeug aktuell zugeordnete Buchung aus. Wenn keine Vorhanden ist wird Null ausgegeben.	EMPLE

5 Verifikation

5.1 Beschreibung der durchgeführten Unit-und System Tests

Für das Backend wurden Automatisierte Tests mithilfe von XUnit geschrieben. Der aktuelle Prototyp besitzt mit den Tests eine über 95% Abdeckung und 77% Branch Abdeckung. Mit den Tests wurden jeweils die Methoden der Schnittstellen von der Schnittstellenbeschreibung getestet. Dies beinhaltet ebenfalls die im Hintergrund laufenden Services der Methoden, die mithilfe von System Tests abgedeckt wurden konnten. Für die System Tests wurde eine Testdatenbank verwendet um eine echte Abfrage zu simulieren. Das Testziel war das Testen der systemkritischen Funktionalitäten der Anwendung. Das Frontend dagegen wurde mit manuellen Tests beschrieben in 5.3 Manuelle Tests getestet.

5.2 Automatische Tests

Zum Testen des Backends wurden 110 Automatisierte Tests verwendet. Zum Zeitpunkt der Abgabe sind alle Tests erfolgreich durchgelaufen. Abzüglich der ausgenommenen Klassen, zu den beispielsweise die Entity-Framework-Migrationsdaten und einige angelegte aber für den Prototypen nicht benötigte Klassen zählen, wurde dabei eine Zeilenabdeckung von 95% erreicht, was einer Abdeckung von 1789 der 1864 der abdeckbaren Zeilen entspricht. Dabei wurde jede Klasse zu mindestens 85.7% abgedeckt. Außerdem wurde eine Zweigabdeckung von 275 der 354 Zweige erreicht, was wiederum einer Zweigabdeckung von durchschnittlich 77% führt. Die niedrigste Abdeckung lag dabei bei 50%. Die meisten der nicht abgedeckten Zeilen sind dabei Autorisierungs-, bzw. Null-Abfragen. Da diese Abfragen sehr einfach und oft deckungsgleich mit bereits an anderer Stelle getesteten Abfragen sind, haben wir ihre Fehler in den Automatischen Tests als nicht schwerwiegend eingeschätzt.

Im Anhang Abbildung 10 befindet sich eine Kopie des jüngsten Codeabdeckungsberichts.

5.3 Manuelle Tests

Um die Funktionalität der Anwendung zusätzlich zu testen wurden Testszenarien entwickelt, welche alle Anforderungen abdecken und das Frontend mit einbeziehen. Die in der folgenden Tabelle definierten Szenarien wurden durchgespielt und überprüft, ob sie wie beschrieben ablaufen.

Nummer	Ablauf	Erfüllt
MT-1	Ohne Angemeldet zu sein, ruft man die URL auf. Dort drückt man auf den „Anmelden“-Knopf, welcher einen zu der Anmeldeseite navigiert. Dort drückt man den „Registrieren“-Knopf und wird zu der Registrierungsseite geleitet. Auf dieser gibt man die persönlichen Daten ein. Bevor der Name nicht eingegeben wurde, kann man nicht zum nächsten Schritt navigieren. Nachdem man einen Namen eingegeben hat, navigiert man zum nächsten Schritt und gibt dort eine gültige E-Mail und ein Passwort ein. Auch hier wird auf gültige Werte überprüft und die Navigation erst anschließend freigegeben. Anschließend gibt man eine gültige Adresse ein. Daraufhin daran wählt man aus, dass der Führerschein nachgereicht wird und gibt eine Führerscheinnummer an. Im nächsten Schritt wird eine Zahlungsmethode und anschließend ein Tarif ausgewählt. In der Tarifauswahl lässt sich eine Übersicht aller möglichen Tarife anzeigen. Anschließend drückt man auf Registrieren und wird automatisch angemeldet und auf eine Übersichtsseite weitergeleitet,	✓
MT-2	Ein Nutzer mit einem bereits existierenden Account navigiert zur Anmeldeseite. Dort gibt er seine E-Mail und sein Passwort ein und wird angemeldet und auf die Übersichtsseite geleitet. Anschließend drückt er auf Abmelden und wird abgemeldet und als Gast zur Willkommenseite geleitet.	✓
MT-3	Ein Gast navigiert zur Tarifsübersichtsseite. Dort werden ihm alle möglichen Tarife und ihre Konditionen angezeigt.	✓
MT-4	Ein angemeldeter Nutzer navigiert zur Tarifsübersichtsseite. Dort werden ihm alle möglichen Tarife und ihre Konditionen angezeigt.	✓
MT-5	Ein angemeldeter Nutzer navigiert zur Benutzerseite. Dort werden ihm die Daten angezeigt, welche er bei der Registrierung angegeben hat. Er kann die Daten dort ändern, eine erneute Validierung wird ebenfalls durchgeführt. Sollten alle Daten valide sein, drückt der Nutzer den „Änderungen speichern“-Knopf. Beim erneuten Laden der Seite sind die geänderten Daten weiterhin vorhanden.	✓
MT-6	Ein angemeldeter Nutzer ändert seine E-Mail. Nachdem er sich ausgeloggt hat, kann er sich nicht länger mit der alten, dafür mit der neuen E-Mail anmelden.	✓
MT-7	Ein angemeldeter Nutzer drückt den Knopf zum Ändern eines Passworts in der Benutzerseite. Gibt er dort ein neues, valides Passwort zusammen mit seinem alten ein, wird dieses geändert, nachdem er auf den „Passwort ändern“-Knopf gedrückt hat. Er kann sich fortan nur mit dem neuen Passwort anmelden. Gibt er sein altes Passwort falsch ein, wird es nicht geändert.	✓

Nummer	Ablauf	Erfüllt
MT-8	Ein angemeldeter Nutzer navigiert zur „Buchung anlegen“-Seite. Dort wählt er aus den verfügbaren Stationen eine Start- und Endstation aus. Er wählt außerdem einen Start- und Endzeitpunkt aus. Anschließend wird ihm eine Auswahl an verfügbaren Fahrzeugklassen angezeigt. Er wählt eine aus, woraufhin er einen Preis für die gewählte Buchung an seinen Tarif angezeigt wird. Er drückt anschließend Buchen. Daraufhin wird er auf die Buchungsübersichtsseite geleitet, wo die neue Buchung eingetragen ist.	✓
MT-9	Ein angemeldeter Nutzer stellt eine Buchung zusammen, für die allerdings keine Fahrzeugklassen verfügbar sind. Er kann diese Buchung entsprechend nicht buchen.	✓
MT-10	Ein Gast stellt eine gültige Buchung zusammen. Im wird darauf angezeigt, dass er nur als Mitglied buchen kann und ihm wird ein Link zur Registrierungsseite bereitgestellt.	✓
MT-11	Ein Nutzer mit einer für die Zukunft gebuchte Buchung öffnet die Buchungsübersicht. Er sieht dort seine Buchung unter Zukünftige Buchung. Durch einen Klick auf die Buchung sieht er Details zur Buchung und kann diese durch einen Knopfdruck stornieren.	✓
MT-12	Ein Nutzer storniert eine Buchung. Sie wird nach einem Neuladen von nun an unter Vergangene Buchungen in der Buchungsübersicht angezeigt.	✓
MT-13	Ein Nutzer versucht eine Buchung zu stornieren, welche bereits angefangen hat. Dies ist nicht möglich.	✓
MT-14	Ein Nutzer navigiert zu einer Buchung, dessen Startzeit angefangen hat aber dessen Endzeit noch nicht eingetreten ist. Diese Buchung ist unter aktuelle Buchungen eingeordnet, wird hervorgehoben und kann durch Knopfdruck gestartet werden. Daraufhin wird das Nummernschild des zugeordneten Fahrzeug angezeigt. Außerdem kann der Nutzer das Auto per Knopfdruck auf und zuschließen.	✓
MT-15	Ein Nutzer öffnet eine aktuelle, gestartete Buchung. Per Knopfdruck kann er sie Beenden, woraufhin sie als vergangene Buchung eingeordnet wird. Er kann sich nun per Knopfdruck die Rechnung anzeigen lassen, in der die genutzte Zeit angegeben ist.	✓
MT-16	Ein Nutzer beendet nach der eigentlichen gebuchten Zeit seine Buchung. In der Rechnung wird ihm dann die Verspätung angezeigt.	✓
MT-17	Ein Nutzer startet nach der eigentlichen gebuchten Zeit seine Buchung. Die Buchung wird direkt als beendet markiert und eine Rechnung ohne Verspätung erstellt.	✓
MT-18	Ein Mitarbeiter loggt sich mit seinen Anmeldedaten ein und wieder aus.	✓
MT-19	Ein angemeldeter Mitarbeiter navigiert zur Fahrzeugseite. Dort sieht er alle eingetragenen Fahrzeuge. Er drückt den Hinzufügen-Knopf und trägt die Daten eines neuen Fahrzeugs ein und speichert diese. Es wird danach in der Übersicht angezeigt. Dort öffnet der Mitarbeiter die Detailansicht und sieht die eben eingegebenen Daten. Er drückt den Bearbeiten Knopf, ändert die Daten und speichert diese. Nach erneuten Laden der Seite sind die Daten weiterhin erhalten.	✓
MT-20	Der Test wird nach M-19 analog für Fahrzeugklassen durchgeführt.	✓
MT-21	Der Test wird nach M-19 analog für Fahrzeugtypen durchgeführt.	✓
MT-22	Der Test wird nach M-19 analog für Tarife durchgeführt.	✓
MT-23	Der Test wird nach M-19 analog für Stationen durchgeführt.	✓

Nummer	Ablauf	Erfüllt
MT-24	Ein angemeldeter Mitarbeiter navigiert zur Benutzerübersicht. Dort sieht er die Daten aller Benutzer. Durch einen Knopfdruck kann er den Status eines Nutzers auf Unautorisiert, Autorisiert, oder Gesperrt setzen. Der Betroffene Nutzer versucht eine Buchung zu Buchen. Dies ist nur möglich, wenn er Autorisiert ist.	✓
MT-25	Ein angemeldeter Mitarbeiter navigiert zur Benutzerübersicht. Dort öffnet er die Detailansicht des Nutzers und drückt „Identität Übernehmen“. Sein Account wechselt zu dem des Nutzers. Er kann alle Aktionen als dieser Nutzer ausführen (Tests MT-1 bis MT-17)	✓
MT-26	Ein angemeldeter Mitarbeiter, welcher einen Nutzer übernimmt, meldet sich ab. Nachdem er sich erneut als normaler Benutzer anmeldet übernimmt er ihn nicht länger.	✓
MT-27	Ein angemeldeter Mitarbeiter, welcher einen Nutzer übernimmt, drückt in der Seitenleiste „Identität verlassen“. Daraufhin ist er wieder als er selbst angemeldet.	✓
MT-28	Ein angemeldeter Administrator navigiert zur Mitarbeiterübersicht. Dort sieht er alle Mitarbeiter, Er drückt den Hinzufügen-Knopf und trägt die Daten eines neuen Mitarbeiters ein und speichert diese. Dieser wird danach in der Übersicht angezeigt. Dort öffnet der Administrator die Detailansicht und sieht die eben eingegebenen Daten. Er drückt den Bearbeiten Knopf, ändert die Daten und speichert diese. Nach erneuten Laden der Seite sind die Daten weiterhin erhalten. Er öffnet die Detailansicht erneut und drückt auf Löschen. Die Daten des Mitarbeiters werden daraufhin gelöscht.	✓
MT-29	Die Tests MT-1 bis MT-28 werden in einem Browser „Mozilla Firefox v100.0.2“ durchgeführt.	✓
MT-30	Die Tests MT-1 bis MT-28 werden in einem Browser „Google Chrome v102.0.5005.61“ durchgeführt.	✓
MT-31	Die Tests MT-1 bis MT-28 werden in einem Browser „Chromium v98.0.4758.107“ durchgeführt.	✓
MT-32	Die Tests MT-1 bis MT-28 werden in einem Browser „Microsoft Edge v101.0.1210.53“ durchgeführt.	✓
MT-33	Die Tests MT-1 bis MT-28 werden in einem Browser „Safari v15.4“ durchgeführt.	✓
MT-34	Für einem Nutzer sind die nur für Mitarbeiter vorgesehenen Navigations-elemente nicht sichtbar	✓
MT-35	Als Nutzer werden die nur den Mitarbeitern verfügbaren Seiten über die URL aufgerufen. Dies sorgt für eine Weiterleitung auf die Startseite	✓

Tabelle 4: Manuelle Tests

5.4 Verifikationsmatrix

Die Nachstehende Tabelle zeigt die im Pflichtenheft definierten Anforderungen und ihren Status zum aktuellen Zeitpunkt. Die der Status der Einträge lässt sich wie folgt interpretieren.

Erfüllt : Die Anforderung wurde erfüllt und erfolgreich getestet

Teils erfüllt : Die Anforderung wurde nur in Teilen erfüllt

Out of Scope : Die Anforderung liegt nicht im Rahmen des Prototypen und wurde ignoriert

Nicht Erfüllt : Die Anforderung wurde nicht Erfüllt

Nummer	Status	Test	Anmerkung
FA-M1	Erfüllt	MT-1	
FA-M2	Erfüllt	MT-1	
FA-M3	Erfüllt	MT-1	
FA-M4	Erfüllt	MT-1	
FA-M5	Out of Scope	-	
FA-M6	Erfüllt	MT-1	
FA-M7	Out of Scope	-	
FA-M8	Erfüllt	MT-1	
FA-M9	Erfüllt	MT-1	
FA-M10	Out of Scope	-	
FA-M11	Out of Scope	-	
FA-M12	Out of Scope	-	
FA-M13	Out of Scope	-	
FA-M14	Erfüllt	MT-1	
FA-M15	Erfüllt	MT-1	
FA-M16	Erfüllt	MT-5	
FA-M17	Erfüllt	MT-6, MT-7	
FA-M18	Erfüllt	MT-2	
FA-M19	Erfüllt	MT-18	Die Anforderung wurde geändert, ein Mitarbeiter meldet sich nun ebenfalls per E-Mail an.

Nummer	Status	Test	Anmerkung
FA-M20	Erfüllt	MT-25, MT-27	
FA-M21	Erfüllt	MT-18, MT-2	
FA-M22	Out of Scope	-	
FA-M23	Out of Scope	-	
FA-M24	Erfüllt	MT-28	Die Anforderung wurde geändert, ein Mitarbeiter meldet sich nun ebenfalls per E-Mail an.
FA-M25	Erfüllt	MT-24	
FA-M26	Teils erfüllt	MT-24	Die Keycard-Anforderung wurde nicht vollständig umgesetzt, da sie außerhalb des Rahmen des Prototypen liegt
FA-M27	Teils erfüllt	-	Die Schnittstelle existiert liefert allerdings nur Testdaten
FA-M28	Out of Scope	-	
FA-M29	Erfüllt	MT-1	
FA-A1	Erfüllt	MT-10	
FA-A2	Erfüllt	MT-8	
FA-A3	Out of Scope	-	
FA-A4	Erfüllt	MT-8	
FA-A5	Erfüllt	MT-8	
FA-A6	Erfüllt	MT-8	
FA-A7	Erfüllt	MT-8	
FA-A8	Teils erfüllt	-	Die entsprechende Schnittstelle ist im Backend vorhanden. Es gibt für Nutzer allerdings keine Möglichkeit sie anzusprechen. Eine Buchung kann allerdings Storniert und neu erstellt werden.
FA-A9	Out of Scope	-	
FA-A10	Out of Scope	-	
FA-A11	Teils erfüllt	MT-12, MT-13	Eine Stornierung ist Grundsätzlich immer vor beginn des Buchungszeitraums möglich
FA-A12	Erfüllt	MT-11, MT-14, MT-15	
FA-A13	Erfüllt	MT-11	

Nummer	Status	Test	Anmerkung
FA-A14	Erfüllt	MT-15	
FA-A15	Out of Scope	-	
FA-A16	Teils erfüllt	MT-15	Eine Prüfung des Standorts findet nicht statt. Eine Fahrt kann aber beendet werden
FA-A17	Out of Scope	-	
FA-A18	Erfüllt	MT-16	
FA-A19	Erfüllt	MT-16	
FA-A20	Erfüllt	MT-15	
FA-A21	Out of Scope	-	
FA-A22	Teils erfüllt	-	Die Entsprechende Schnittstelle wird angesprochen, ändert aber in Ermangelung eines echten Autos nur einen Datenbankeintrag
FA-F1	Out of Scope	-	
FA-F2	Out of Scope	-	
FA-F3	Out of Scope	-	
FA-F4	Out of Scope	-	
FA-F5	Out of Scope	-	
FA-F6	Teils erfüllt	MT-14	Die Entsprechende Schnittstelle wird angesprochen, ändert aber in Ermangelung eines echten Autos nur einen Datenbankeintrag
FA-F7	Teils erfüllt	MT-14	Die Entsprechende Schnittstelle wird angesprochen, ändert aber in Ermangelung eines echten Autos nur einen Datenbankeintrag
FA-F8	Out of Scope	-	
FA-D1	Erfüllt	MT-23	
FA-D2	Out of Scope	-	
FA-D3	Out of Scope	-	
FA-D4	Teils erfüllt	MT-19, MT-21	Die zu einem Auto gespeicherten Daten wurden geändert.
FA-D5	Erfüllt	MT-19, MT-21	
FA-D6	Erfüllt	MT-19, MT-21	
FA-D7	Erfüllt	MT-22	
FA-D8	Out of Scope	-	

Nummer	Status	Test	Anmerkung
FA-D9	Out of Scope	-	
FA-D10	Out of Scope	-	
FA-D11	Out of Scope	-	
FA-D12	Out of Scope	-	
FA-D13	Out of Scope	-	
FA-D14	Out of Scope	-	
QA-1	Erfüllt	-	Der SLA des Hosters der Anwendung verspricht eine Verfügbarkeit von 99,9%
QA-2	Out of Scope	-	
QA-3	Erfüllt	-	www.car-bonara.de verlangt eine Übertragung mit HTTPS
QA-4	Erfüllt	-	Bei 100 parallel laufenden Anfragen kam es maximal zu einer Antwortzeit von 281ms. Durchschnittlich betrug sie 226ms ¹
QA-5	Erfüllt	MT-34, MT-25	Ausgenommen wurden Schnittstellen, die explizit anonym aufrufbar sein dürfen. Zusätzlich zu dem Navigationsverbot wird das Backend keine Anfragen ohne gültige Authentifizierung bearbeiten
QA-6	Erfüllt	-	Bei 9 parallel laufenden Anfragen kam es maximal zu einer Antwortzeit von 172ms. Durchschnittlich betrug sie 156ms ²
QA-7	Erfüllt	MT-29, MT-33	
BA-1	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.
BA-2	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.
BA-3	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.
BA-4	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.

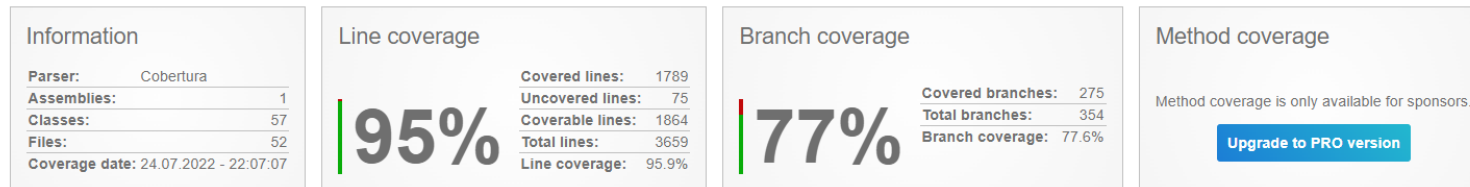
¹Angefragt wurde die URL <https://www.car-bonara.de/api/stationdatabase/getall>

²Siehe vorherige Fußnote

Nummer	Status	Test	Anmerkung
BA-5	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.
BA-6	Erfüllt		Eine Analyse der Benutzeroberfläche wurde in Verbindung mit den Manuellen Tests durchgeführt. Es wurde festgestellt das die Bedingung erfüllt ist.

Tabelle 5: Verifikationsmatrix

6 Anhang



Risk Hotspots

No risk hotspots found.

Coverage

Collapse all | Expand all

By namespace, Level: 3

Grouping: ●

Filter:

Name	Line coverage					Branch coverage		
	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage
CarbonaraWebAPI	1789	75	1864	4136	95.9%	275	354	77.6%
CarbonaraWebAPI.Util	3	0	3	10	100%	0	0	
CarbonaraWebAPI.Util.Extensions	3	0	3	10	100%	0	0	
CarbonaraWebAPI.Model.DAO	263	1	264	612	99.6%	0	0	
CarbonaraWebAPI.Model.DAO.User	38	0	38	109	100%	0	0	
CarbonaraWebAPI.Model.DAO.Station	13	0	13	28	100%	0	0	
CarbonaraWebAPI.Model.DAO.Plan	31	0	31	54	100%	0	0	
CarbonaraWebAPI.Model.DAO.Person	29	0	29	54	100%	0	0	
CarbonaraWebAPI.Model.DAO.Employee	11	0	11	28	100%	0	0	
CarbonaraWebAPI.Model.DAO.CarType	15	0	15	31	100%	0	0	
CarbonaraWebAPI.Model.DAO.CarClass	11	0	11	25	100%	0	0	
CarbonaraWebAPI.Model.DAO.Car	19	0	19	40	100%	0	0	
CarbonaraWebAPI.Model.DAO.Booking	34	0	34	58	100%	0	0	
CarbonaraWebAPI.Model.DAO.BillPosition	12	1	13	57	92.3%	0	0	
CarbonaraWebAPI.Model.DAO.Bill	16	0	16	57	100%	0	0	
CarbonaraWebAPI.Model.DAO.AuthData	8	0	8	22	100%	0	0	
CarbonaraWebAPI.Model.DAO.Address	26	0	26	49	100%	0	0	
CarbonaraWebAPI.Infrastructure.Jwt	5	0	5	22	100%	0	0	
CarbonaraWebAPI.Infrastructure.Jwt.TokenManagement	5	0	5	22	100%	0	0	
CarbonaraWebAPI.Data	19	0	19	49	100%	0	0	
CarbonaraWebAPI.Data.AppDbContext	19	0	19	49	100%	0	0	
CarbonaraWebAPI	67	4	71	107	94.3%	2	4	50%
CarbonaraWebAPI.Startup	67	4	71	107	94.3%	2	4	50%

Abbildung 10: Abdeckungsbericht Teil 1




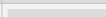
























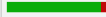
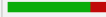






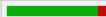
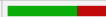























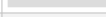

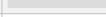



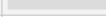


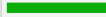
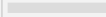


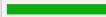
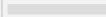



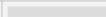






CarbonaraWebAPI.Controllers	444	37	481	1513	92.3%		126	172	73.2%	
CarbonaraWebAPI.Controllers.RegistrationComposite	7	0	7	188	100%		0	0		
CarbonaraWebAPI.Controllers.PingController	3	0	3	20	100%		0	0		
CarbonaraWebAPI.Controllers.LoginRequest	7	0	7	118	100%		0	0		
CarbonaraWebAPI.Controllers.StationDatabaseController	23	2	25	63	92%		2	4	50%	
CarbonaraWebAPI.Controllers.CarDatabaseController	27	3	30	70	90%		5	8	62.5%	
CarbonaraWebAPI.Controllers.CarController	51	8	59	119	86.4%		25	38	65.7%	
CarbonaraWebAPI.Controllers.AdminController	34	3	37	81	91.8%		8	12	66.6%	
CarbonaraWebAPI.Controllers.UserController	60	4	64	188	93.7%		10	14	71.4%	
CarbonaraWebAPI.Controllers.CarclassDatabaseController	24	1	25	62	96%		3	4	75%	
CarbonaraWebAPI.Controllers.CartypeDatabaseController	24	1	25	62	96%		3	4	75%	
CarbonaraWebAPI.Controllers.PlanDatabaseController	24	1	25	63	96%		3	4	75%	
CarbonaraWebAPI.Controllers.BookingController	103	10	113	257	91.1%		46	58	79.3%	
CarbonaraWebAPI.Controllers.AuthController	32	3	35	118	91.4%		16	20	80%	
CarbonaraWebAPI.Controllers.UserManagementController	25	1	26	104	96.1%		5	6	83.3%	
CarbonaraWebAPI.Services	698	31	729	1191	95.7%		132	160	82.5%	
CarbonaraWebAPI.Services.ServiceStarter	18	3	21	38	85.7%		1	2	50%	
CarbonaraWebAPI.Services.CarService	28	4	32	70	87.5%		4	6	66.6%	
CarbonaraWebAPI.Services.CarDatabaseService	57	2	59	91	96.6%		6	8	75%	
CarbonaraWebAPI.Services.ExternalPartnerService	18	3	21	42	85.7%		3	4	75%	
CarbonaraWebAPI.Services.AuthService	112	4	116	207	96.5%		38	46	82.6%	
CarbonaraWebAPI.Services.BookingService	277	15	292	435	94.8%		70	84	83.3%	
CarbonaraWebAPI.Services.AdminService	44	0	44	75	100%		2	2	100%	
CarbonaraWebAPI.Services.CarclassDatabaseService	29	0	29	49	100%		2	2	100%	
CarbonaraWebAPI.Services.CartypeDatabaseService	33	0	33	54	100%		2	2	100%	
CarbonaraWebAPI.Services.PlanDatabaseService	39	0	39	59	100%		2	2	100%	
CarbonaraWebAPI.Services.StationDatabaseService	43	0	43	71	100%		2	2	100%	
CarbonaraWebAPI.Model.DTO	290	2	292	632	99.3%		15	18	83.3%	
CarbonaraWebAPI.Model.DTO.PlanDTO	33	0	33	49	100%		0	0		
CarbonaraWebAPI.Model.DTO.PersonDTO	30	0	30	45	100%		0	0		
CarbonaraWebAPI.Model.DTO.LoginDTO	12	0	12	24	100%		0	0		
CarbonaraWebAPI.Model.DTO.EmployeeDTO	10	0	10	27	100%		0	0		
CarbonaraWebAPI.Model.DTO.CartypeDTO	16	0	16	31	100%		0	0		
CarbonaraWebAPI.Model.DTO.CarDTO	20	0	20	40	100%		0	0		
CarbonaraWebAPI.Model.DTO.CarclassDTO	12	0	12	28	100%		0	0		
CarbonaraWebAPI.Model.DTO.BookingDTOIn	6	0	6	68	100%		0	0		
CarbonaraWebAPI.Model.DTO.BillPositionDTO	8	0	8	46	100%		0	0		
CarbonaraWebAPI.Model.DTO.AddCarDTO	11	0	11	21	100%		0	0		
CarbonaraWebAPI.Model.DTO.AddressDTO	18	0	18	42	100%		1	2	50%	
CarbonaraWebAPI.Model.DTO.BookingDTOOut	36	2	38	68	94.7%		6	8	75%	
CarbonaraWebAPI.Model.DTO.BillDTO	20	0	20	46	100%		2	2	100%	
CarbonaraWebAPI.Model.DTO.StationDTO	23	0	23	39	100%		2	2	100%	
CarbonaraWebAPI.Model.DTO.UserDTO	35	0	35	58	100%		4	4	100%	

Abbildung 11: Abdeckungsbericht Teil 2