

Dokumentation

Fuhrparkmanagement

Firma: Doubleslash

Projekt: Fuhrparkmanagement System

Betreuer: Andreas Heinrich

Kunde: Johannes Mayer

Team: Leonard Deininger, Okan Kizilagil, Cedric Schaaf, Nick Habermann, Edwin Starz


Inhaltsverzeichnis

Zielgruppe, Problem, Eigenschaften.....	1
Erste UI Entwürfe.....	3
Architektur / Technologie.....	4
Softwarearchitektur.....	7
Struktursicht.....	7
Verteilungssicht.....	7
Verhaltenssicht.....	8
Verhaltenssicht.....	8
Projektmanagement.....	9
User Stories & Aufwandsschätzung.....	10
Tatsächlicher Zeitaufwand.....	12
Schnittstellentechnologie.....	14
Fahrzeugverwaltungs-Schnittstelle.....	14
Reservations-Schnittstelle.....	17
Fahrten-Schnittstelle.....	19
Statistik-Schnittstelle.....	23
Übersicht Endpunkte (Swagger UI).....	24
Externer User Management Service (von doubleSlash).....	25
Codegenerierung zu Rest API Schnittstellen mit Swagger.....	28
Datenbankmodell (logisch).....	28
Datenbankmodell (physisch).....	29
CI/CD Pipeline.....	30
Backend-Workflow (mit Gradle).....	30
Frontend-Workflow (mit npm).....	30
Benutzeroberfläche.....	31
Startseite (Kalender).....	31
Fahrt anlegen (Stepper).....	32
Fahrt bearbeiten/löschen.....	36
Navigationsmenü.....	37
Autoverwaltung.....	37
Fahrtenübersicht.....	38
Statistik.....	39
Technische Aspekte.....	40
Login und Authentifizierung.....	40
Validierungen.....	40
Testing.....	41
E-Mail Versand und Empfang.....	41
Beispielablauf: Anfragen/Antworten (Sequenzdiagramm).....	42
Features & Aufteilung.....	43
Herausforderungen.....	45
Installation auf einem Linux-Server (Debian).....	46

Installationshandbuch.....	47
Administrationshandbuch.....	47
Lizenz.....	48
Ausblick: Was kann noch ergänzt werden?.....	49

Zielgruppe, Problem, Eigenschaften

Die Zielgruppe sind alle Mitarbeiter der Firma doubleSlash an den Standorten Stuttgart, Karlsruhe, München und Friedrichshafen.



Jörg Baier ♂


BILDUNG: Bachelor-Abschluss in Informatik
JOB-TITEL: Softwareentwickler bei doubleSlash
ORT: Stuttgart
ALTER: 32

BESCHREIBUNG:

- Jörg ist sportlich aktiv und legt großen Wert auf Fitness und Gesundheit. Er betreibt regelmäßig Sport, sei es Laufen, Radfahren, oder Fußball, um körperlich fit zu bleiben.
- Sein Beruf erfordert viele Meetings und Zusammenarbeiten mit Teams an verschiedenen Standorten. Dies bedeutet, dass er häufig mit dem Firmen-Auto reisen muss, um persönlich an Besprechungen teilzunehmen.

NUTZER ZIELE:

- Er sucht eine zuverlässige Web-Anwendung, um die Firmen internen Fahrzeuge zu buchen, damit er konfliktfrei seine Meetings und Kundentermine an verschiedenen Standorten erreichen kann.



Lea Weiß ♀

BILDUNG: Abitur
JOB-TITEL: Ausbildung – Fachinformatikerin bei doubleSlash
ORT: Karlsruhe
ALTER: 26

BESCHREIBUNG:

- Lea ist leidenschaftlich an der Softwareentwicklung interessiert und möchte ihre Fähigkeiten und Kenntnisse in diesem Bereich vertiefen.
- Sie liebt es, mit ihren Freunden zu reisen und diese Reisen zu organisieren

NUTZER ZIELE:

- Lea Organisiert in der Firma Events und Ausflüge, damit sie besser Planen kann, benötigt Sie ein Managementsystem, um die Fahrzeuge rechtzeitig zu reservieren.

Problem: Buchung der Fuhrpark-Fahrzeuge ist nicht übersichtlich und läuft nicht zentral auf einer Plattform, in der Vergangenheit wurden Fahrzeuge doppelt verbucht, da zum Teil über Outlook und Confluence parallel gebucht wurde.

Gewünschte Eigenschaften:

- **Responsive Webanwendung:** Die Benutzeroberfläche sollte so konzipiert sein, dass verschiedene Geräte und Bildschirmgrößen optimal funktionieren. So ist es auch möglich, dass man diese auch von unterwegs nutzen kann.
- **Zentrale Plattform:** Das System schafft eine zentrale Plattform zur Buchung und Verwaltung von Firmenfahrzeugen. Damit sind auch doppelte Buchungen und sonstige Unstimmigkeiten nicht mehr möglich. Alle Daten sind an einem logischen Ort und nicht mehr verteilt.
- **Statistikerfassung:** Das System erfasst statistische Daten über die Nutzung der Fahrzeuge, um unter anderem die Ressourcenplanung und -verwaltung zu verbessern.

- Erweiterbarkeit: Das System sollte mit dem Aspekt der Erweiterbarkeit entwickelt werden, sodass es auch nach Abschluss des Projekts zu erweiterten Anforderungen noch von einer externen Person weiterentwickelt werden könnte.
- Container-Software-Kompatibilität: Das System kann mittels Container-Software wie Docker virtualisiert werden, um eine effizientere Ausführung (und ggfs. Skalierbarkeit) zu gewährleisten.

Erste UI Entwürfe

Um eine Idee zu bekommen, wie die Anwendung aussehen könnte, und worauf wir achten müssen, haben wir eine einfach gehaltene UI Prototyp Skizze erstellt.

A hand-drawn UI prototype of a login page. The browser address bar shows "fpm.kunagant.de". The page has a header with a hamburger menu icon and the "FPM" logo. The main content area is divided into three vertical sections. The middle section contains a stick figure icon in a circle, followed by two input fields labeled "Benutzername:" and "Kennwort:". Below the password field is a red link that says "Passwort vergessen?".

A hand-drawn UI prototype of a dashboard page. The browser address bar shows "fpm.kunagant.de". The page has a header with a hamburger menu icon, the "FPM" logo, and a circular "OK" button. The main content area is titled "Startseite" and is divided into a left sidebar and a main content area. The sidebar contains four sections: "Neue Fahrt Buchen" with a car icon, "Aktuelle Buchung", "Letzte Fahrten", and "Ranking". The main content area is titled "Übersicht aktueller Fahrten" and contains a table with the following data:

	Von:	Nach:	Zeitraum:	Auto:	Mitfahrer:		Mitfahrt Buchen	Abst. & Credits
(ES)	Stuttgart	München	01.01.2024 - 02.01.2024	Stadler	keine		<input type="checkbox"/>	
(LD)								
(UH)								
(CC)								
(OK)								

Architektur / Technologie

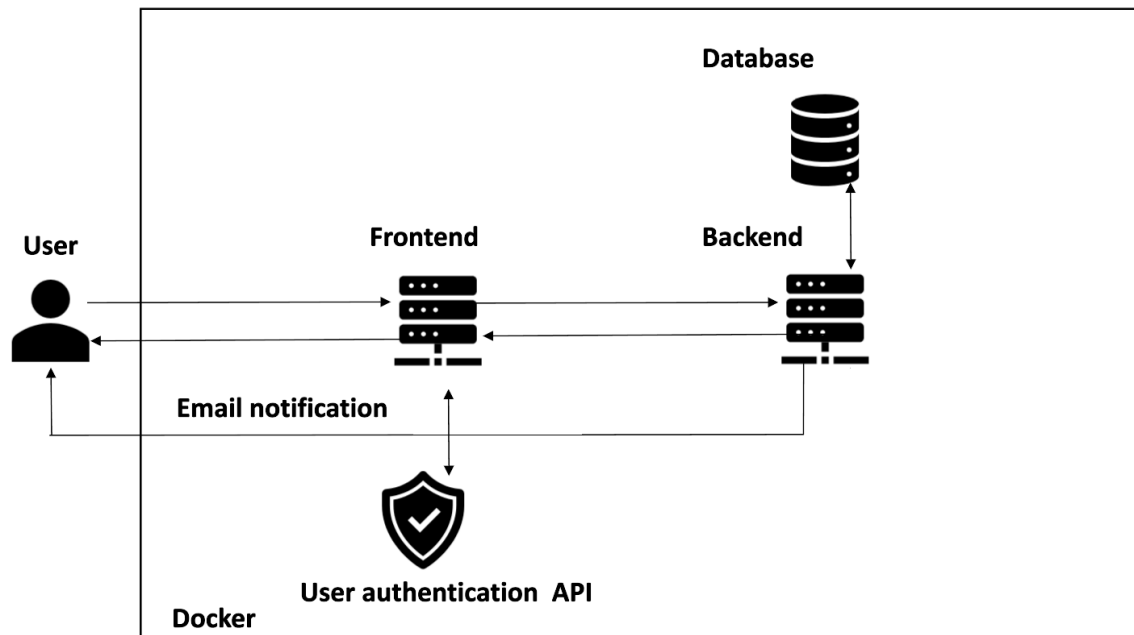
Konkrete Vorgaben, welche Technologien wir zu verwenden haben, wurden von der Firma doubleSlash nicht gestellt, jedoch sind Spring Boot und Angular erwünscht.

Die anderen Architektur-/Technologieentscheidungen, wie z.B. Git, Docker etc. haben wir vor allem an Kriterien wie Einsatzzweck, Praxisbewährung oder auch Erlernbarkeit getroffen.

Wir haben uns auch über die Lizenzen der einzelnen Technologien informiert und überprüft, ob diese die kommerzielle Nutzung oder Veränderungen zulassen.

- Spring Boot als Backend Framework
 - Gradle als Build-Management-Automatisierungs-Tool
 - Lizenz: [Apache 2.0](#)
 - Incremental Builds, Sehr flexibel, Tasks lassen sich einfach erstellen und einbinden
 - Lombok als Code Generierungs Tool zur Vermeidung von Boilerplate-Code
 - Lizenz: [Eigene](#)
 - fusionauth-jwt, um Json Web Tokens (JWTs) auszulesen und zu validieren
 - Lizenz: [Apache 2.0](#)
 - swagger-codegen, um Modelle und Schnittstellen-Methoden aus der Definition der REST-API als Java Code zu generieren
 - Lizenz: [Apache 2.0](#)
 - ModelMapper, um Data Transfer Objects (DTOs) zu den Entity Klassen zu konvertieren
 - Lizenz: [Apache 2.0](#)
 - JUnit für Unit-Tests
 - Lizenz: [Eclipse Public License 2.0](#)
 - H2 Database als In-Memory Testing Datenbank
 - Lizenz: [MPL 2.0](#)
 - Integrationstests mit Spring Boot Test
 - Lizenz: [Apache 2.0](#)
 - Wenig Konfigurationsaufwand, einfache Entwicklung von REST-Anwendungen, Ecosystem mit großer Anzahl von Benutzern, Praxisbewährt, (Erwünscht von doubleSlash)
- Angular als Frontend Framework
 - npm als Abhängigkeits-/Paketmanager
 - Lizenz: [Artistic License 2.0](#)
 - Testing-Frameworks: Jasmine & Karma
 - Lizenz: [MIT](#)
 - Angular Material als generelle Komponenten Bibliothek
 - Lizenz: [MIT](#)
 - [angular-calendar](#) als Komponenten Bibliothek für eine Kalenderansicht
 - Lizenz: [MIT](#)
 - ng-openapi-gen um Modelle und Schnittstellen-Methoden aus der Definition der REST-API als TypeScript Code zu generieren
 - Lizenz: [MIT](#)
 - Lizenz: [MIT](#)
 - Modularisierung durch Komponente-System, in der Praxis weit verbreitet, (Erwünscht von doubleSlash)

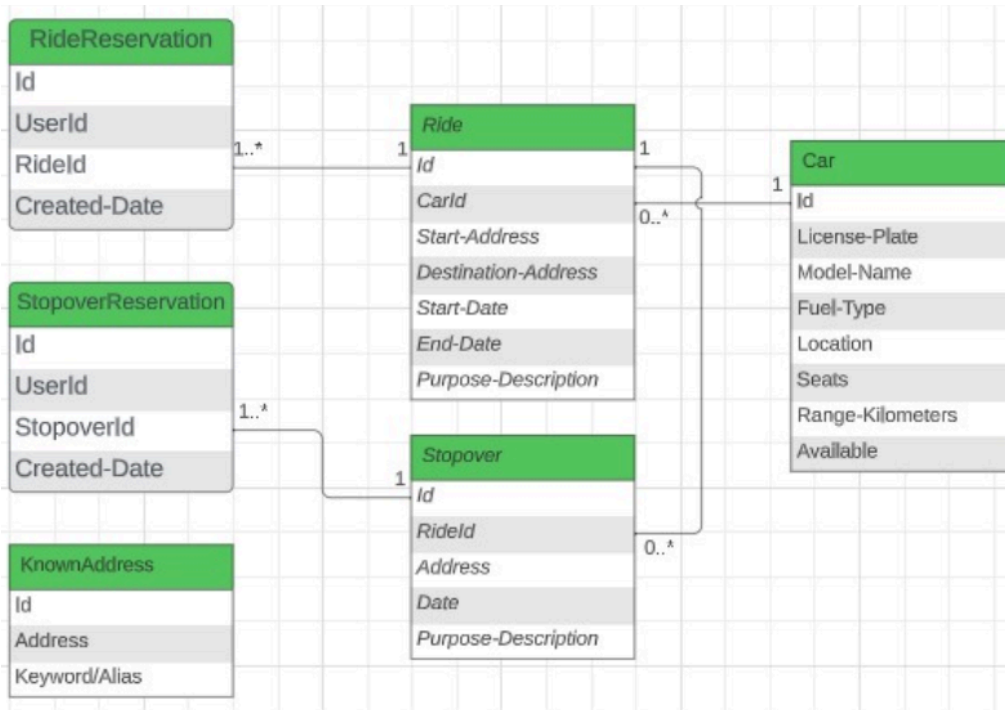
- PostgreSQL als Datenbank-Managementsystem (DBMS)
 - Lizenz: [PostgreSQL Licence](#)
 - Open Source, hat sich in der Praxis bewährt (Robust), gute SQL Unterstützung
- Git als Version Control System (VCS)
 - GitHub Repository als Code-Hosting-Plattform
 - GitHub Kanban Board für das Projektmanagement / Aufgabenverteilung
 - Github Actions als CI/CD Pipeline
 - Zentrale Verwaltung des Quellcodes, des Projektmanagements und der CI/CD Pipeline auf einer Plattform sichert eine bestmögliche Integration der einzelnen Dienste untereinander und verringert den Arbeitsaufwand
 - Basisfunktionen sind kostenfrei mit optionalen Abonnement Modellen für erweiterte Features und mehr Performance (z.B. mehr Speicherplatz)
- Docker zur Containerization und Virtualisierung
 - Lizenz: [Apache 2.0](#)
 - Industriestandard, gute Integration in z.B. IntelliJ IDEA und hat außerdem einen eigene Desktopanwendung zur Verwaltung von Docker Containern (Docker Desktop)
- IntelliJ IDEA Ultimate als Entwicklungsumgebung (IDE)
 - All-in-One Lösung, beinhaltet z.B. auch Git- und Datenbanken Verwaltungs Funktionalität
 - Kostenlose Bildungslizenzen für unter anderem Studierende
- MailHog
 - Lizenz: [MIT](#)
 - Open-Source-Tool für das Testen von E-Mail-Funktionalitäten in Entwicklungsumgebungen.
- bwCloud
 - Kostenloser Server für Studenten, um das tatsächliche Deployment der Anwendung zu testen.
- Swagger/OpenAPI
 - Lizenz: [Apache 2.0](#)
 - REST API Spezifikation und grafische Aufbereitung dieser.



Datenmodell

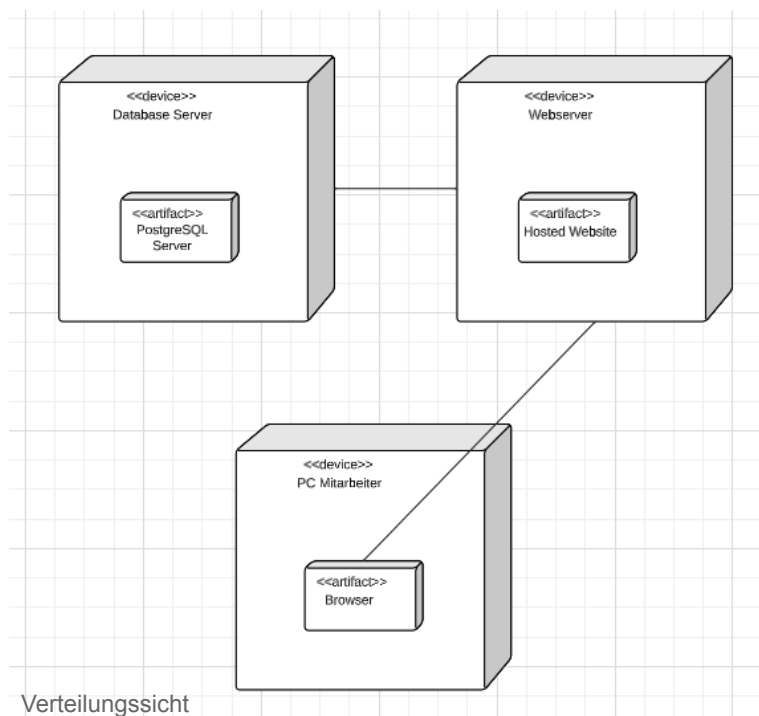
Softwarearchitektur

Struktursicht



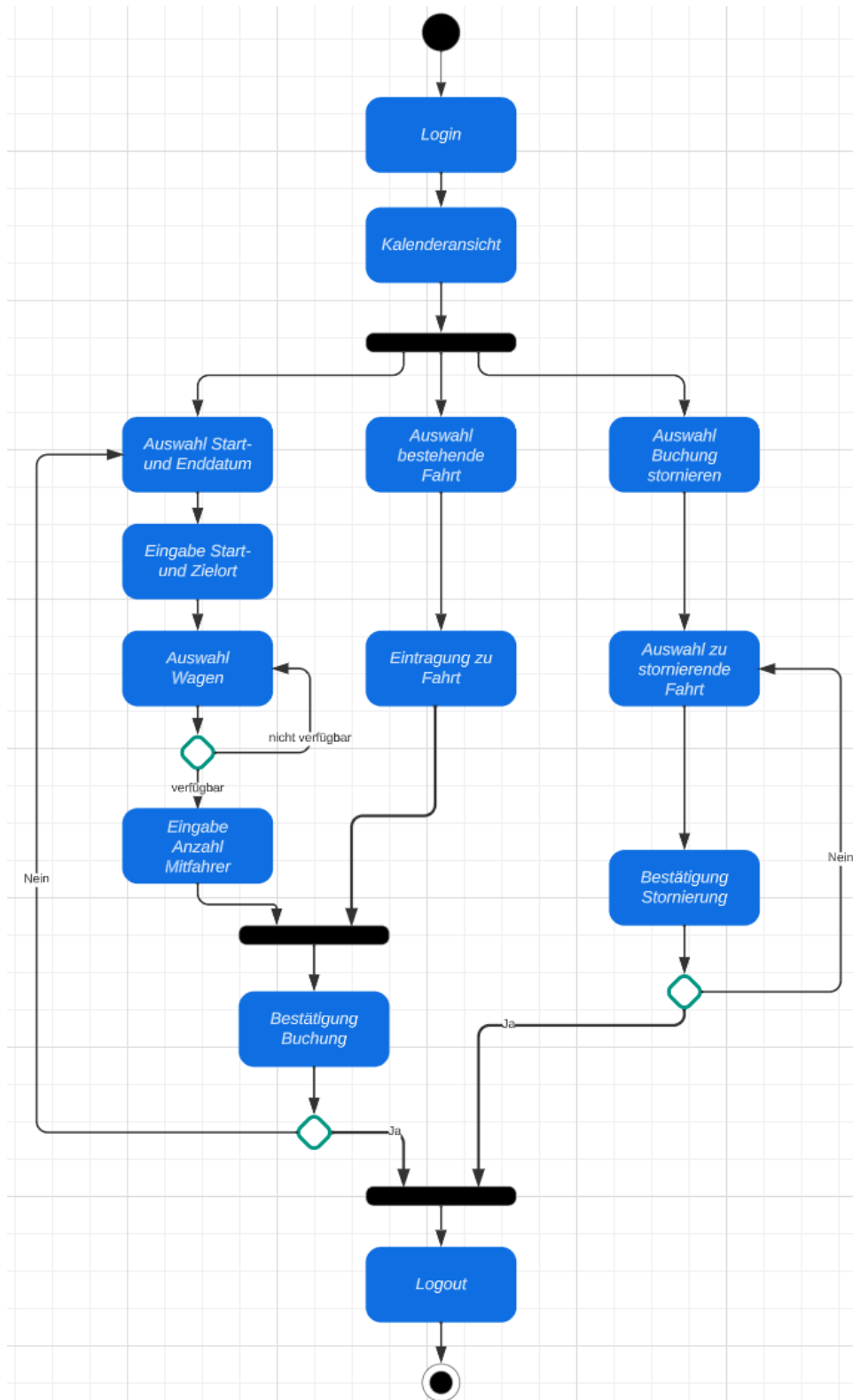
Struktursicht

Verteilungssicht



Verteilungssicht

Verhaltenssicht



Verhaltenssicht

Projektmanagement

Gruppenintern haben wir beschlossen, jede Woche im Team unsere Fortschritte zu präsentieren und zu besprechen (Weekly Standup). Unklarheiten und Fragen werden protokolliert, um diese beim nächsten Weekly Sync zu besprechen.

Nach der internen Präsentation gehen wir den aktuellen Stand mit dem Kunden durch, um Feedback zu erhalten und sicherzustellen, dass die Implementierung und geplanten Features seinen Erwartungen entsprechen.

Unser Backlog ist im Kanban Board von GitHub erfasst. Jedes Teammitglied setzt sich dort Ziele für die kommende Woche, außerdem wird das angelegte Protokoll von allen Teammitgliedern überprüft.

Für weitere Informationen und Absprachen haben wir einen Discord-Server und eine WhatsApp-Gruppe eingerichtet.

Definition of Done (DoD):

- Präsentation für Team vorbereitet (Jeder präsentiert Dienstagabend vor dem Meeting mit Kunden, was er erreicht hat, und sagt, was wichtig zu wissen fürs Team ist)
- Feature Dokumentiert
- Feature manuell getestet, ggfs. Unit-/Integrationstest

User Stories & Aufwandsschätzung

Wir haben geschätzt, was wir denken, wie viel Aufwand die einzelnen Backlog Items, in Stunden, sein könnten. Dies ist uns sehr schwer gefallen, da unterschiedliche Teammitglieder unterschiedliche Ideen hatten, aber auch natürlich, da das Thema inhaltlich komplex und das Schätzen allgemein für uns mit wenig Erfahrung verbunden ist. 15 Stunden pro Woche, pro Person, mal fünf Personen, mal acht Wochen für die Implementierung ergibt ein Zeitkontingent von 600 Stunden. Hier müssen natürlich Puffer für unvorhergesehene Probleme zugerechnet werden und jedes Feature muss dokumentiert und zur Präsentation aufbereitet werden.

- Als Mitarbeiter möchte ich mich mit meinem Doubleslash Account einloggen können.
 - Login Ansicht erstellen: 8 Stunden
 - Session Handling: 10 Stunden
 - Daten über Kunden-API validieren und Token erhalten: 18 Stunden
 - Kundendaten und Token in der Datenbank und Browser halten: 16 Stunden
 - **=> 52h**
- Als Mitarbeiter möchte ich in einer Kalenderansicht einen Zeitraum für meine Buchung auswählen können.
 - Kalenderansicht im Frontend erstellen, die Buchungen über einen oder mehrere Tage zulässt, und start und ende anwählbar macht: 36 Stunden
 - Daten in der Datenbank zu einer Buchung hinzufügen: 24 Stunden
 - **=> 60h**
- Als Mitarbeiter möchte ich bereits gebuchte Fahrten sehen, um zu sehen, wann kein Auto verfügbar ist.
 - Ansicht erstellen, die existierende Buchungen übersichtlich anzeigt: 30 Stunden
 - Möglichkeit, sich bei einer Fahrt hinzuzubuchen: 18 Stunden
 - Möglichkeit, sich bei einer Fahrt zu entfernen: 18 Stunden
 - **=> 66h**
- Als Mitarbeiter möchte ich Firmenstandorte und selbst gesetzte Adressen als Ziel und Startpunkt verwenden können.
 - Adressbuch einbinden, eingegebene Adressen werden automatisch vervollständigt/vorgeschlagen: 36 Stunden
 - Firmenstandorte hinterlegt und direkt auswählbar: 18 Stunden
 - **=> 54h**
- Als Mitarbeiter möchte ich Buchungen löschen können: **=> 18 Stunden**
- Als Mitarbeiter möchte ich Statistiken angezeigt bekommen.
 - OpenStreetView-Berechnung der Kilometer: 30 Stunden
 - Leaderboard-Design: 30 Stunden
 - **=> 60h**
- Als Mitarbeiter möchte ich im Buchungsprozess alle aktuell freien Fahrzeuge auswählen können **=> 30h**
- Als Mitarbeiter möchte ich automatisch einen Outlook-Termin an die Fahrtteilnehmer versenden können.
 - Versenden von ICS-Datei: 24 Stunden
 - Konfiguration SMTP: 22 Stunden

○ => 46h

- Als Mitarbeiter möchte ich Zwischenstopps zu einer Route hinzufügen können: 30 Stunden
- Als Mitarbeiter möchte ich eine Buchung vorzeitig beenden können: 18 Stunden

Insgesamt sind es ca. 386 Stunden für die Features. Puffer ca. 20%: 100 Stunden. Meetings mit dem Kunden: ca. 40 Stunden. Da wir uns in einem agilen Umfeld bewegen kommen für regelmäßige Überarbeitungen der Anforderungen noch Zeit hinzu: ca. 50h.

Aufgaben	Wochen	KW40	KW41	KW42	KW43	KW44	KW45	KW46	KW47	KW48	KW49	KW50	KW51	KW52	KW01	KW02	KW02
Phase 1: Planung																	
Kundenanforderung	2																
Verwendung von Technologien	2																
Zeit und Aufgabenmanagement	2																
Phase 3: Vorbereitung																	
Einrichtung & Einarbeitung der Technologien	2																
Initialisierung	2																
Phase 4: Umsetzung																	
Entwicklung	8																
Tests	8																
Phase 5: Dokumentation																	
Dokumentation fertigstellen	2																

Zeitplan

Tatsächlicher Zeitaufwand

In diesem Kapitel vergleichen wir den tatsächlichen Zeitaufwand mit unseren Aufwandsschätzungen. Es sind auch neue Punkte dazu gekommen, die zeitaufwendig waren und mit denen man im Voraus nicht geplant hatte.

- (NEU)Erstellung eines Git Repositories.
 - Aufwandsschätzung => 0h Tatsächlicher Zeitaufwand => 2h
- (NEU)Erstellung eines ERM.
 - Aufwandsschätzung => 0h Tatsächlicher Zeitaufwand => 15h
- (NEU)Einrichtung von Technologien: SpringBoot, Angular, Docker, PostgreSQL.
 - Aufwandsschätzung => 0h Tatsächlicher Zeitaufwand => 20h
- (NEU)Backend Einrichtung, bevor mit den User-Stories begonnen werden konnte.
 - Aufwandsschätzung => 0h Tatsächlicher Zeitaufwand => 20h
- Als Mitarbeiter möchte ich mich mit meinem Doubleslash Account einloggen können.
 - Aufwandsschätzung => 52h Tatsächlicher Zeitaufwand => 58h
- Als Mitarbeiter möchte ich in einer Kalenderansicht einen Zeitraum für meine Buchung auswählen können.
 - Aufwandsschätzung => 60h Tatsächlicher Zeitaufwand => 58h
- Als Mitarbeiter möchte ich bereits gebuchte Fahrten angezeigt bekommen, um zu sehen, wann kein Auto verfügbar ist.
 - Aufwandsschätzung => 66h Tatsächlicher Zeitaufwand => 29h
- Als Mitarbeiter möchte ich Firmenstandorte und selbst gesetzte Adressen als Ziel und Startpunkt verwenden können.
 - Aufwandsschätzung => 54h Tatsächlicher Zeitaufwand => 2h
- Als Mitarbeiter möchte ich Buchungen löschen können:
 - Aufwandsschätzung => 18h Tatsächlicher Zeitaufwand => 19h
- (NEU) Als Mitarbeiter möchte ich Buchungen bearbeiten können:
 - Aufwandsschätzung => 0h Tatsächlicher Zeitaufwand => 10h
- Als Mitarbeiter möchte ich Statistiken angezeigt bekommen.
 - Aufwandsschätzung => 60h Tatsächlicher Zeitaufwand => 23h
- Als Mitarbeiter möchte ich im Buchungsprozess alle aktuell freien Fahrzeuge auswählen können
 - Aufwandsschätzung => 30h Tatsächlicher Zeitaufwand => 5h

- Als Mitarbeiter möchte ich automatisch einen Outlook-Termin an die Fahrtteilnehmer versenden können.
 - Aufwandsschätzung => **46h** Tatsächlicher Zeitaufwand => **45h**
- Als Mitarbeiter möchte ich Zwischenstopps zu einer Route hinzufügen können.
 - Aufwandsschätzung => **30h** Tatsächlicher Zeitaufwand => **12,5h**
- Als Mitarbeiter möchte ich eine Buchung vorzeitig beenden können.
 - Aufwandsschätzung => **18h** Tatsächlicher Zeitaufwand => **0h**
- Wöchentliche Meetings.
 - Aufwandsschätzung => **40h** Tatsächlicher Zeitaufwand => **50h**
- (NEU)Meetings mit unserem Betreuer.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **20h**
- (NEU)Dokumentationen und Präsentationen.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **112h**
- (NEU)Abschluss Konfiguration bwCloud Hosting.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **30h**
- (NEU)Einarbeitung in SonarQube für Codequalität (Wurde aus zeitlichen Gründen verworfen).
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **10h**
- (NEU)Einarbeitung in Angular.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **15h**
- (NEU)Frontend Logik und optische Anpassungen.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **25h**
- (NEU)Swagger Einrichtung und Testen der Schnittstellen.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **28h**
- (NEU)Abschluss Konfiguration bwCloud Hosting.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **30h**
- (NEU) CI/CD für Back-End und Front-End.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **8h**
- (NEU) Tests für Back-End Komponenten geschrieben.
 - Aufwandsschätzung => **0h** Tatsächlicher Zeitaufwand => **4h**

Unsere Aufwandseinschätzung am Anfang des Projekts lag bei ca. 580 Stunden. Nach dem Projekt und nach dem Auswerten unserer Dokumentationen kommen wir auf einen Zeitaufwand von ca. 650 Stunden.

Schnittstellentechnologie

Fuhrparkmanagement RESTful API (Backend)

Wir haben uns an den [Zalando RESTful API Guidelines](#) orientiert. Der Inhalt der Anfragen und Rückgaben ist im JSON Format, ist fest definiert und manche Werte sind mit Konditionen verknüpft. So muss z.B. "seats" (aus Anfragen, siehe (*)) größer als 0 sein, sonst wird ein HTTP Status Code 400 (Bad Request) zurück gesendet. Generell wird ein HTTP Status Code 403 (Forbidden) zurück gesendet, falls kein Authentifizierungs-Token bei einer Anfrage mitgesendet wird.

Fahrzeugverwaltungs-Schnittstelle

Anfrage

HTTP Schnittstelle	http://localhost:8080/cars
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>[{ "id": 1, "licensePlate": "S-DS-123", "modelName": "BMW 118i", "fuelType": "PETROL", "location": "Stuttgart Feuerbach", "seats": 5, "range": 500, "available": true }]</pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/cars
HTTP Anfragemethode	POST
Parameter	-

Beispiel-Anfrage-Inhalt (*)	<pre>{ "licensePlate": "S-DS-123", "modelName": "BMW 118i", "fuelType": "PETROL", "location": "Stuttgart Feuerbach", "seats": 5, "range": 500, "available": true }</pre>
-----------------------------	--

Rückgaben

HTTP Status Code	201 (Created)
Beispiel-Rückgabe-Inhalt	<pre>{ "id":1, "licensePlate": "S-DS-123", "modelName": "BMW 118i", "fuelType": "PETROL", "location": "Stuttgart Feuerbach", "seats": 5, "range": 500, "available": true }</pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/cars/{id}
HTTP Anfragemethode	PUT
Parameter	id
Beispiel-Anfrage-Inhalt	<pre>{ "licensePlate": "S-DS-123", "modelName": "BMW 118i", "fuelType": "PETROL", "location": "Stuttgart Feuerbach", "seats": 5, "range": 500, "available": true }</pre>

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "id": 1, "licensePlate": "S-DS-123", "modelName": "BMW 118i", "fuelType": "PETROL", "location": "Stuttgart Feuerbach", "seats": 5, "range": 500, "available": true }</pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/cars/{id}
HTTP Anfragemethode	DELETE
Parameter	id
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Reservations-Schnittstelle

Anfrage

HTTP Schnittstelle	http://localhost:8080/reservations
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>[{ "id": 1, "userId": "1", "rideId": 1, "createdAt": "2023-12-06T12:54:02.479Z", "deletedDate": "null" }]</pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/reservations
HTTP Anfragemethode	POST
Parameter	-
Beispiel-Anfrage-Inhalt	<pre>{ "userId": "1", "rideId": 1, }</pre>

Rückgaben

HTTP Status Code	201 (Created)
Beispiel-Rückgabe-Inhalt	<pre>{ "id": 1, "userId": "1", "rideId": 1, "createdAt": "2023-12-06T12:54:02.479Z", }</pre>

	<pre>"deletedDate": "null" }</pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/reservations/{id}
HTTP Anfragemethode	DELETE
Parameter	id
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/reservations/{id}
HTTP Anfragemethode	PUT
Parameter	id
Beispiel-Anfrage-Inhalt	<pre>{ "userId": "1", "rideId": 1, }</pre>

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "id": 1, "userId": "1", "rideId": 1, "createdDate": "2023-12-06T12:54:02.479Z", "deletedDate": "null" }</pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Fahrten-Schnittstelle

Anfrage

HTTP Schnittstelle	http://localhost:8080/rides
HTTP Anfragemethode	POST
Parameter	-
Beispiel-Anfrage-Inhalt	<pre>{ "carId": 1, "startAddress": "Stuttgart Feuerbach", "destinationAddress": "Karlsruhe Stadtmitte", "startDate": "2023-11-20T12:08:00.000Z", "endDate": "2023-11-20T12:11:00.000Z", "purpose": "Geschäftstreffen" }</pre>

Rückgaben

HTTP Status Code	201 (Created)
Beispiel-Rückgabe-Inhalt	<pre>{ "id": 1, "carId": 1, "startAddress": "Stuttgart Feuerbach", "destinationAddress": "Karlsruhe Stadtmitte",</pre>

	<pre> "startDate": "2023-11-20T12:08:00.000Z", "endDate": "2023-11-20T12:11:00.000Z", "purpose": "Geschäftstreffen" } </pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/rides/{id}
HTTP Anfragemethode	PUT
Parameter	id
Beispiel-Anfrage-Inhalt	<pre> { "carId": 1, "startAddress": "Stuttgart Feuerbach", "destinationAddress": "Karlsruhe Stadtmitte", "startDate": "2023-11-20T12:08:00.000Z", "endDate": "2023-11-20T12:11:00.000Z", "purpose": "Geschäftstreffen" } </pre>

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre> { "id": 1, "carId": 1, "startAddress": "Stuttgart Feuerbach", "destinationAddress": "Karlsruhe Stadtmitte", "startDate": "2023-11-20T12:08:00.000Z", "endDate": "2023-11-20T12:11:00.000Z", "purpose": "Geschäftstreffen" } </pre>
HTTP Status Code	400 (Bad Request)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)

Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/rides/{id}
HTTP Anfragemethode	DELETE
Parameter	id
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/rides
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	[{ "id": 1, "carId": 1, "startAddress": "Stuttgart Feuerbach", "destinationAddress": "Karlsruhe" }]

	<pre> Stadtmitte", "startDate": "2023-11-20T12:08:00.000Z", "endDate": "2023-11-20T12:11:00.000Z", "purpose": "Geschäftstreffen" }] </pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Statistik-Schnittstelle

Anfrage

HTTP Schnittstelle	http://localhost:8080/statistics/users
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>[{ "userId": "1", "name": "Martin Beck", "amountReservations": 3 }]</pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8080/statistics/users/{id}
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "userId": "1", "name": "Martin Beck", "amountReservations": 3 }</pre>
HTTP Status Code	404 (Not found)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Übersicht Endpunkte (Swagger UI)

Interaktiv verfügbar: <http://193.197.228.214:8080/swagger-ui/index.html> (Server auf bwCloud)

Swagger powered by SMARTBEAR Explore

Fuhrparkmanagement API 1.0.0 OAS 3.0
Fuhrparkmanagement API

Servers
http://localhost:8080 Authorize

Cars Cars with their attributes

- GET** `/cars` GET cars
- POST** `/cars` POST cars
- GET** `/cars/{id}` GET cars/{id}
- PUT** `/cars/{id}` PUT cars/{id}
- DELETE** `/cars/{id}` DELETE cars/{id}

Rides Combines rides with reservations

- GET** `/rides` GET rides
- POST** `/rides` POST rides
- GET** `/rides/{id}` GET rides/{id}
- PUT** `/rides/{id}` PUT rides/{id}
- DELETE** `/rides/{id}` DELETE rides/{id}

Reservations Reservation

- GET** `/reservations` GET reservations
- POST** `/reservations` POST reservations
- GET** `/reservations/{id}` GET reservations/{id}
- PUT** `/reservations/{id}` PUT reservations/{id}
- DELETE** `/reservations/{id}` DELETE reservations/{id}

Statistics Statistics; driven kilometers

- GET** `/statistics/users` GET statistics/users
- GET** `/statistics/users/{id}` GET statistics/users/{id}

Schemas

- Car >
- Ride >
- Reservation >
- Statistic >

Swagger UI Schnittstellen

Leonard Deininger, Okan Kizilagil, Cedric Schaaf, Nick Habermann, Edwin Starz

Externer User Management Service (von doubleSlash)

Anfrage

HTTP Schnittstelle	http://localhost:8082/usermanagement/v2/authenticate
HTTP Anfragemethode	POST
Parameter	-
Beispiel-Anfrage-Inhalt	<pre>{ "name": "mbeck", "password": "mbeck-pw", }</pre>

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "token": "eyJhbGciOiJIUzI1NiIsInR...", "user": { "id": 1, "name": "mbeck", "givenName": "Martin", "surname": "Beck", "mail": "Martin.Beck@dev.doubleslash.de" } }</pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8082/usermanagement/v2/tokens/renew
HTTP Anfragemethode	POST
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "token": "eyJhbGciOiJIUzI1NiIsInR..."</pre>

	}
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8082/usermanagement/v2/tokens/rev oke
HTTP Anfragemethode	POST
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	-
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8082/usermanagement/v2/users
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	[{ "id": 1, "name": "mbeck", "givenName": "Martin", "surname": "Beck", "mail": "Martin.Beck@dev.doubleslash.de" }]

]
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

Anfrage

HTTP Schnittstelle	http://localhost:8082/usermanagement/v2/users/{id}
HTTP Anfragemethode	GET
Parameter	-
Beispiel-Anfrage-Inhalt	-

Rückgaben

HTTP Status Code	200 (OK)
Beispiel-Rückgabe-Inhalt	<pre>{ "id": 1, "name": "test", "givenName": "test", "surname": "test", "mail": "test@doubleslash.org" }</pre>
HTTP Status Code	403 (Forbidden)
Beispiel-Rückgabe-Inhalt	-

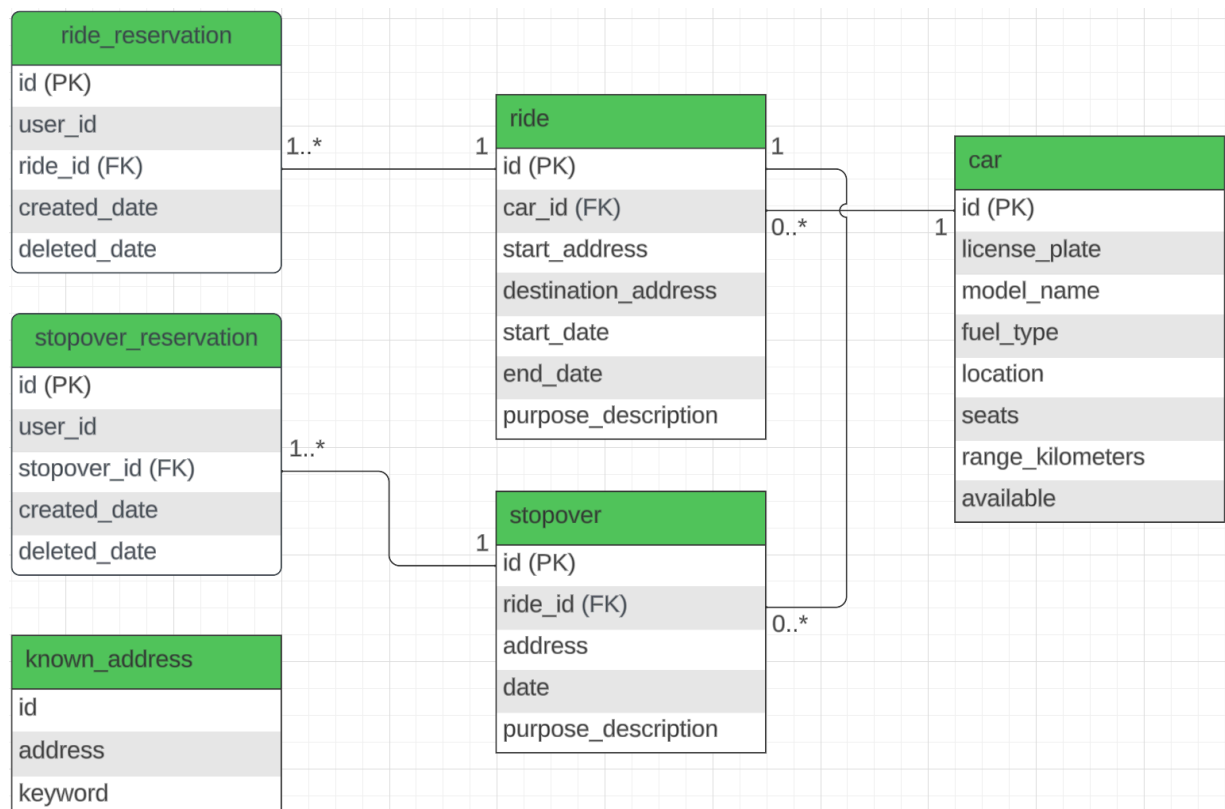
Codegenerierung zu Rest API Schnittstellen mit Swagger

Unsere Schnittstellen definieren wir als [OpenAPI Spezifikation](#) (siehe swagger/swagger-fuhrparkmanagement.yaml im Projekt Repository) und generieren aus dieser für das Backend und Frontend die jeweiligen Modell-Strukturen und Schnittstellen-Methoden (ohne fertige Implementation im Backend).

Im Backend binden wir ein [extra Plugin](#) ein (siehe build.gradle Datei im Projekt Repository), welches eigene Tasks bereitstellt, die die Java-Klassen und Methoden generieren und in den Kompilierungsvorgang inkludieren. Für die eigentliche Implementation auf Backend-Seite vererbt man die API Komponenten und überschreibt die zu implementierenden Schnittstellen-Methoden.

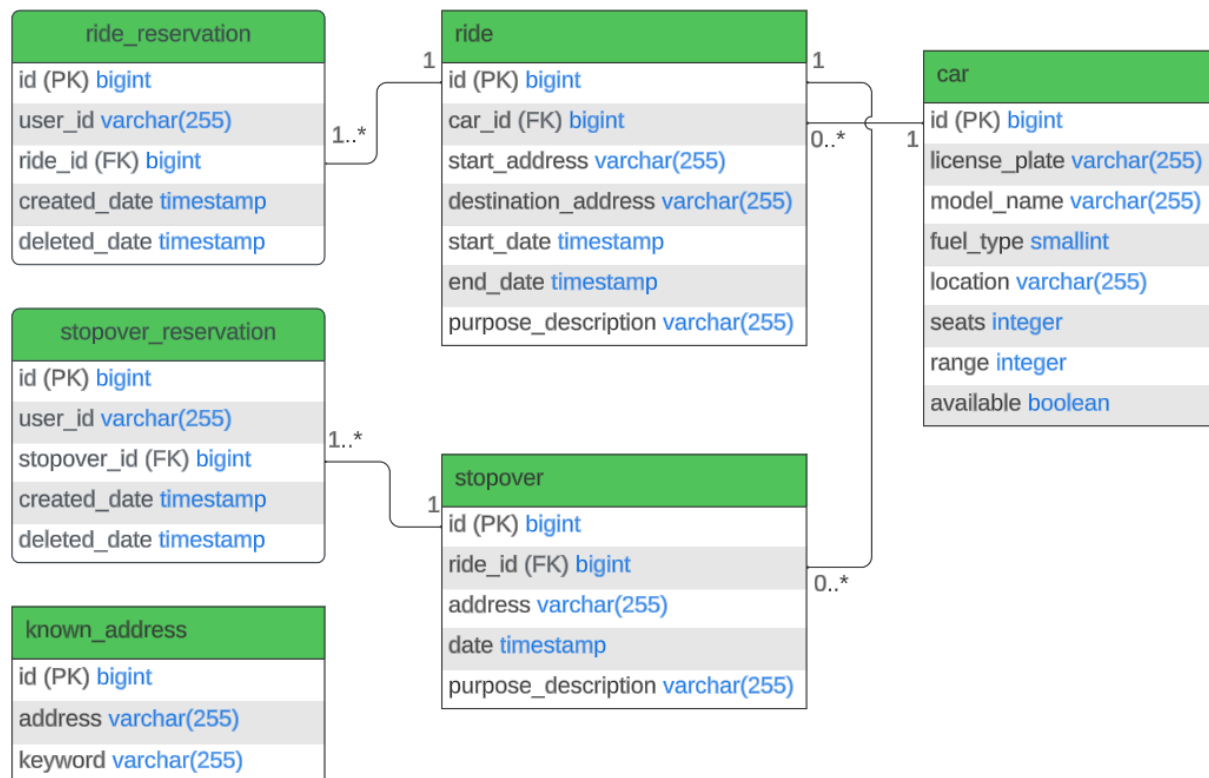
Für das Frontend benutzen wir eine [zusätzliche Abhängigkeit](#), um TypeScript Modell-Interfaces und voll implementierte Service Klassen mit Methoden zu den jeweiligen Schnittstellen zu generieren.

Datenbankmodell (logisch)



Logisches Datenbankmodell

Datenbankmodell (physisch)



Physisches Datenbankmodell

CI/CD Pipeline

Um eine höhere Codequalität zu erreichen, haben wir uns dazu entschieden, unseren Code kontinuierlich zu testen. Dafür verwenden wir bei GitHub sogenannte [Workflows](#), die jeweils verschiedene Jobs mit vordefinierten Schritten ausführen, wenn Commits auf den main-Branch gepusht werden oder ein Pull Request auf diesem eröffnet wird. Derzeit haben wir zwei Workflows (im `.github/workflows/` Ordner) definiert: je einen, um das Backend beziehungsweise das Frontend zu kompilieren und zu testen.

Backend-Workflow (mit Gradle)

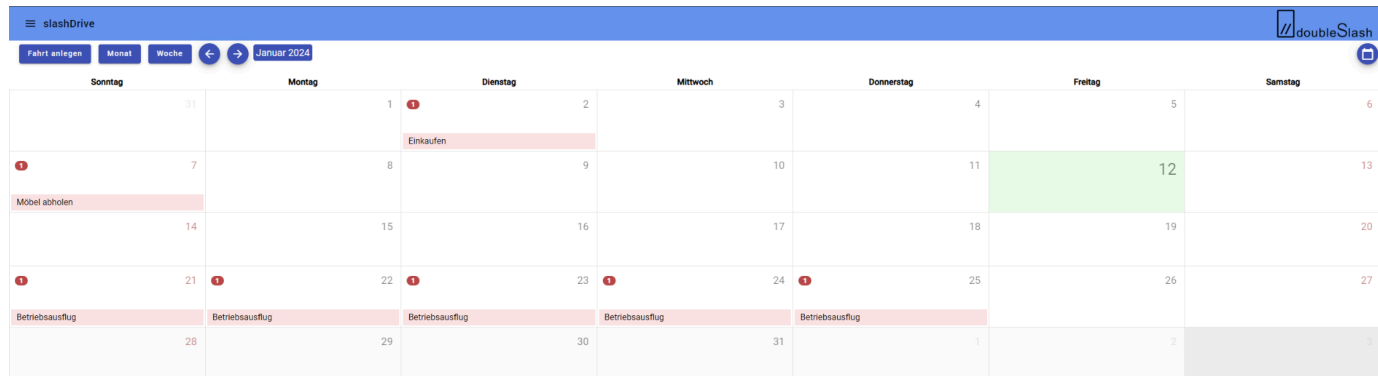
Im ersten Schritt wird der Quellcode aus dem main-Branch abgerufen (mittels `git checkout`). Danach wird das Java Development Kit (JDK) mit der Java Version 17 aufgesetzt. Nun wird der Hashwert der Gradle-Wrapper Jar mit den offiziellen Hashwerten verglichen, um sicherzustellen, dass die Gradle-Wrapper Jar valide ist. Und letztendlich wird der `build`-Task mit dem Gradle-Wrapper ausgeführt, der unter anderem auch die Tests zum Backend ausführt.

Frontend-Workflow (mit npm)

Ebenfalls wird im ersten Schritt der Quellcode aus dem main-branch abgerufen (mittels `git checkout`). Danach wird Node.js mit der Version 18 aufgesetzt. Nun werden mit dem Befehl `$ npm ci` alle Abhängigkeiten im Frontend komplett neu installiert. Schließlich werden zwei npm Run-Scripts ausgeführt, welche das Frontend kompilieren und dann die Tests dafür ausgeführt werden.

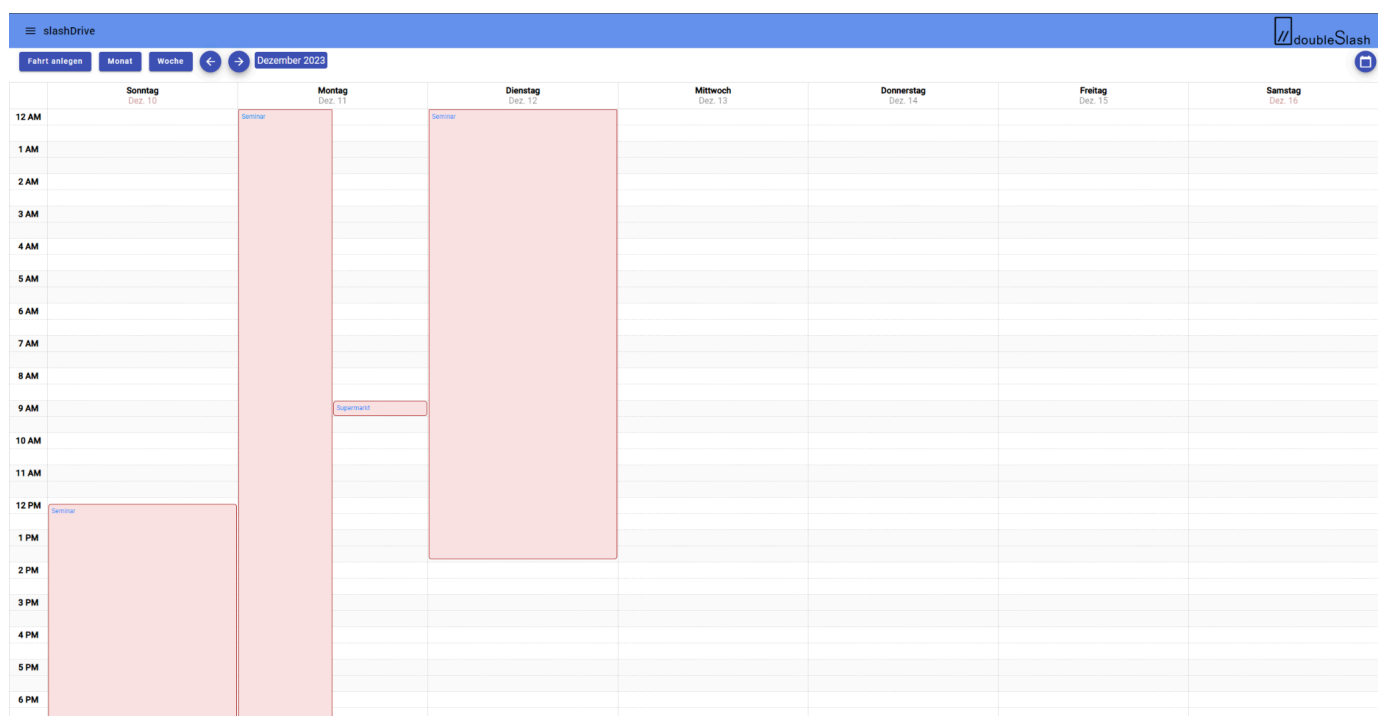
Benutzeroberfläche

Startseite (Kalender)



Kalender Monatsansicht

Auf der Startseite sieht man auf einen Blick die Fahrten dieses Monats, um sich einen Überblick zu verschaffen. Der Kalender lässt sich zwischen Monats- und Wochenansicht umschalten (siehe Bild unten).



Kalender Wochenansicht



Startseite Buttons

Über die Pfeil-Buttons verändert man die Kalenderansicht und je nachdem, ob die Ansicht gerade im Monats- oder Wochen-Modus ist, wird sie um einen Monat/Woche zurück oder

vorwärts verschoben. Der Text neben diesen Buttons zeigt an, welchen Monat die Kalenderansicht gerade anzeigt.

Der Button ganz rechts oben auf der Startseite mit dem Kalender Symbol lässt die Kalenderansicht auf den heutigen Tag springen.

Fahrt anlegen (Stepper)

Über den Button *“Fahrt anlegen”* auf der Startseite lässt sich das Popup zum Anlegen einer neuen Fahrt öffnen. Das Anlegen einer neuen Fahrt ist in drei Schritte unterteilt.

Schritt 1:

Fahrt anlegen 1. Schritt

Im Textfeld (1) gibt man eine Beschreibung zum Zweck der Fahrt an. Dieser kann auch der gleiche Zweck einer anderen Fahrt sein, muss also nicht einzigartig sein. Er dient hauptsächlich dazu, eine grobe Beschreibung des Zwecks der Fahrt dem Benutzer zu übermitteln.

Der Benutzer wählt bei (2) einen Buchungszeitraum aus. Dieser ist standardmäßig auf den aktuellen Tag gesetzt. Außerdem gewährleistet das Auswahlelement, dass der Endtag nicht vor dem Starttag sein kann. Derzeit ist es auch möglich, ein Datum aus der Vergangenheit auszuwählen, um nachträgliches Eintragen einer Fahrt zu ermöglichen.

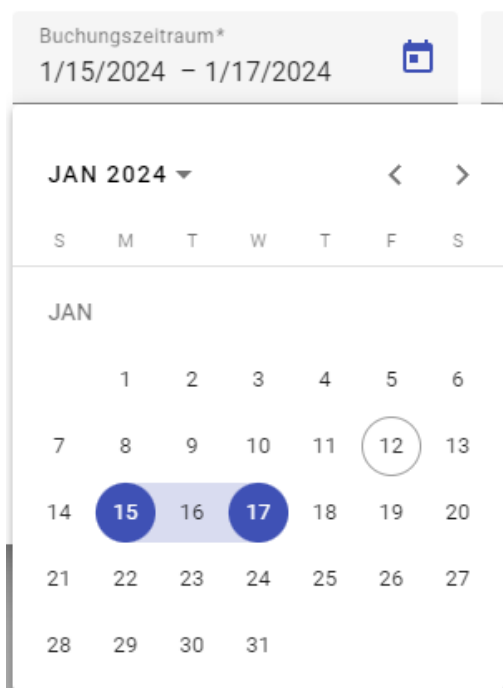
Mithilfe der Uhrzeiteingaben (3) und (4) wählt der Benutzer anschließend die Start- und Endzeit aus. Hierbei wird zudem validiert, wenn der Buchungszeitraum ein und derselbe Tag sind, dass die Endzeit nicht vor der Startzeit sein kann. Sollte dies der Fall sein, wird ein roter Fehlertext unterhalb der beiden Eingabefelder angezeigt.

Die Start- und Zieladressen werden bei (5) und (6) eingetragen. Hierbei werden auch per Autocomplete die Standorte der Firmen Niederlassungen von doubleSlash aufgelistet, um so diesen Vorgang zu vereinfachen. Auch gibt es die Möglichkeit, beliebige andere Adressen einzutragen.

Wenn man mit den Eingaben der Basisinformationen fertig ist, kann man auf den "Weiter"-Button (7) klicken, um zur Auswahl des Autos fortzufahren. Sollten Eingabefelder jedoch noch nicht ausgefüllt sein, erscheint unter diesen eine rote Fehlermeldung (siehe (4)) und das Formular lässt einen nicht fortfahren.

Den Vorgang des Fahrt anlegen lässt sich auch vorzeitig abbrechen, indem man auf den "Abbrechen"-Button (8) klickt.

Der "Fahrt anlegen"-Button (9) ist in dem ersten Schritt (Eingabe der Basisinformationen) noch nicht verfügbar und ist somit ausgegraut und nicht anklickbar.



Buchungszeitraum Auswahl (wird angezeigt, wenn auf (2) im vorheriger Bild geklickt wird)

Schritt 2 (Verfügbares Auto auswählen und Mitfahrer auswählen): Die Auswahlmöglichkeiten im 2. Schritt sind von den Informationen aus dem 1. Schritt abhängig. Die verfügbaren Autos hängen von dem angegebenen Start-Standort ab. Die Anzahl der Mitfahrer, die sich hinzufügen lässt, ist von der Anzahl der Sitzplätze des ausgewählten Autos abhängig.

Fahrt anlegen 2. Schritt

Nun lassen sich alle verfügbaren Autos (1), sprich diese, die am ausgewählten Start Standort ansässig sind, im vorher ausgewählten Buchungszeitraum noch nicht gebucht sind und ausreichend Sitzplätze für die Passagiere haben, auswählen. Hierbei werden Informationen zu den jeweiligen Autos angezeigt, wie etwa der Modellname, das Nummernschild, die Anzahl der Sitzplätze, die Kraftstoffart, sowie die Reichweite in Kilometern. Es ist immer nur ein Auto pro Fahrt möglich auszuwählen.

Im Eingabefeld bei (2) ist es möglich, die Passagiere zur Fahrt einzutragen. Hierbei klickt man auf das Feld und gibt den Namen des Passagiers ein. Dabei werden auch alle noch nicht hinzugefügten Personen per Autocomplete angezeigt und können auch ausgewählt werden (siehe nächstes Bild). Allgemein werden alle Personen aus dem User Management Service von doubleSlash hier mit berücksichtigt. Standardmäßig ist der derzeit eingeloggte Benutzer automatisch in der Liste der Passagiere vertreten. Das Löschen von Passagieren ist über einen Kreuz-Button neben dem jeweiligen Namen der Person möglich. Auch ist es möglich, dass keine Passagiere ausgewählt werden, es somit noch keine Reservierungen für die Fahrt gibt, um diese an einem späteren Zeitpunkt einzutragen. Falls die Anzahl der Passagiere die Anzahl der Sitzplätze eines Autos übersteigt, wird das Auto aus der Liste der verfügbaren Autos entfernt und ist nicht mehr ausgewählt.

Über den “Zurück”-Button (3) kann man die vorherigen Basisdaten noch einmal überarbeiten, falls gewünscht.

Mit dem *Weiter*-Button (4) gelangt man zu einer Übersicht der Fahrt, um möglicherweise aufgetretene Fehler zu erkennen und zu korrigieren.

Der *Fahrt anlegen*-Button (5) wird außerdem nun nicht mehr ausgegraut und ist anklickbar, sobald man ein Auto ausgewählt hat.

The screenshot shows a user interface for selecting passengers. At the top, there is a header 'Passagiere / Mitfahrer'. Below it, two passenger names are listed in a light gray bar: 'Martin Beck' and 'Alex mit Langen Nachnamen', each with a small 'x' icon to its right. Below this bar is a text input field with the placeholder 'Passagier hinzufügen...'. A dropdown menu is open, showing a list of names: 'Michale Schenider', 'Peter Maier', 'Julia Moser', and 'Sabrina Hensle'. The dropdown menu has a blue border and a blue tab on the right side.

Auswahl der Passagiere, siehe (2) im vorherigen Bild

Schritt 3 (Übersicht Daten): Im 3. Schritt sieht man nochmal eine Übersicht aller Fahrtdaten.

The screenshot shows the 'Fahrt anlegen' (Create Trip) form, Step 3. The form has a title 'Fahrt anlegen' and a progress bar with three steps: 'Basisinformationen', 'Auto auswählen', and '3 Fahrt bestätigen'. The third step is highlighted. Below the progress bar, there is a summary of the trip data: 'Fahrt: Seminar', 'vom 15.1.2024 (08:30 Uhr) bis 17.1.2024 (12:20 Uhr)', 'Startadresse: Stuttgart', 'Zieladresse: Ulm', 'Auto: BMW i4 (S-DS-1234)', and 'Passagiere: Martin Beck, Alex mit Langen Nachnamen'. There are three red circles with numbers 1, 2, and 3. Circle 1 is next to the 'Startadresse: Stuttgart' text. Circle 2 is next to the 'Alle Daten zurücksetzen' button. Circle 3 is next to the 'Fahrt anlegen' button. At the bottom left, there are two buttons: 'Zurück' and 'Alle Daten zurücksetzen'. At the bottom right, there are two buttons: 'Abbrechen' and 'Fahrt anlegen'.

Fahrt anlegen 3. Schritt

In der Karte bei (1) sieht man eine Übersicht der eingetragenen Daten zur Fahrt. Hier sollte man aufgetretene Fehler bei der Eingabe der Daten gut erkennen.

Bei dem Button bei (2) kann man alle bisherigen Eingabefelder für alle Schritte auf ihren Standardzustand zurücksetzen.

Schlussendlich wird beim Drücken von (3) die Fahrt angelegt und die Reservierungen für die Passagiere angelegt (Anfragen ans Backend). Auch werden vom Backend aus anschließend die E-Mails an die Passagiere zu ihren Reservierungen geschickt.

Fahrt bearbeiten/löschen

Auf der Startseite kann in der Kalenderansicht auf eine Fahrt geklickt werden. Dies öffnet das "Fahrt bearbeiten" Popup.

Daraufhin sieht der Benutzer alle Informationen, die zu dieser Fahrt gehören. Jede dieser Informationen kann nun angepasst werden und durch den "Aktualisieren"-Button gespeichert werden. Hierbei werden die verfügbaren Autos, je nachdem man den Buchungszeitraum oder die Anzahl der Passagiere verändert, angepasst.

Möchte man stattdessen die gesamte Fahrt löschen, drückt man auf den "Löschen"-Button.

Fahrt bearbeiten

Zweck*

Seminar

Buchungszeitraum*

1/15/2024 – 1/17/2024

MM/DD/YYYY – MM/DD/YYYY

Startzeit*

09:30

Endzeit*

12:20

Startadresse*

Stuttgart

Zieladresse*

Ulm

BMW i4

Nummernschild: S-DS-1234

Sitzplätze: 5

Kraftstoffart: ELECTRIC

Reichweite: 465km

Passagiere / Mitfahrer

Martin Beck

Peter Maier

Passagier hinzufügen...

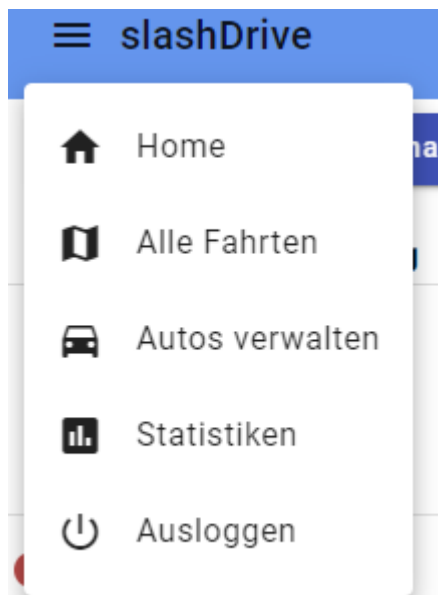
Abbrechen

Löschen

Aktualisieren

Fahrt bearbeitungs Popup

Navigationsmenü



















Sidebar Menü

Wenn man links oben in der Toolbar auf das Icon mit den drei Strichen klickt, öffnet sich das Navigationsmenü. Hier kann man zu allen Seiten der Webanwendung gelangen. Auch findet sich dort der Button zum Ausloggen.

Autoverwaltung

Übersicht:

Nummernschild*	Modell-Name*	Treibstoff* 	Standort* 	Anzahl Sitze*	Reichweite (in KM)*	<div>Auto anlegen</div>	
Nummernschild	Modell-Name	Treibstoff	Standort	Sitze	Reichweite	Verfügbar	Bearbeiten/Löschen
S-DS-1234	BMW i4	Elektrisch	Stuttgart	5	465	Ja	 
FN-DS-1212	BMW i4	Elektrisch	Friedrichshafen	5	465	Ja	 
FN-DS-4444	BMW 318i	Elektrisch	Friedrichshafen	5	570	Ja	 
FN-DS-3333	BMW i3	Elektrisch	Friedrichshafen	4	340	Ja	 
M-DS-707	BMW 740d	Diesel	München	5	670	Ja	 
M-DS-717	BMW i7	Elektrisch	München	5	580	Ja	 
KA-DS-5656	Smart	Elektrisch	Karlsruhe	2	100	Ja	 
FN-DS-2323	BMW 320i	Benzin	Friedrichshafen	5	550	Ja	 

Alle Autos (Tabelle)

Alle Autos werden tabellarisch aufgeführt und ihre Eigenschaften in einer Tabelle angezeigt. Zusätzlich sind in jeder Zeile Buttons zum Bearbeiten und Löschen eines Fahrzeugs.

Oberhalb der Tabelle finden sich Eingabefelder zu den Eigenschaften eines Autos, sprich das Nummernschild, der Modellname, die Treibstoffart (Benzin, Diesel, Elektrisch), der Standort, an dem das Auto ansässig ist, die Anzahl der Sitze sowie die Reichweite in Kilometern. Der Wert in den Feldern zur Anzahl der Sitze und Reichweite kann nicht unter 1 fallen, sonst wird ein roter Fehlertext unter den Feldern angezeigt. Wenn ansonsten

außerdem auch ein Feld nicht ausgefüllt ist, ist der “Auto anlegen”-Button ausgegraut und nicht klickbar. Andernfalls kann man über diesen Button ein Auto anlegen.

Bearbeiten:

Auto bearbeiten

Nummernschild* S-DS-1234	Modell-Name* BMW i4
Treibstoff* Elektrisch	Standort* Stuttgart
Anzahl Sitze* 5	Reichweite (in KM)* 465

☒ Generell verfügbar 1

Abbrechen Aktualisieren 2

Auto Bearbeiten Popup

In diesem Dialog kann man die gleichen Felder wie oben genannt bearbeiten. Zusätzlich kann man noch mit Hilfe einer Checkbox (1) angeben, ob ein Auto generell verfügbar sein soll. Dies ist für den Fall, wenn ein Auto z.B. in der Reparatur ist, beim TÜV ist oder anderweitig aktuell nicht verfügbar sein soll. Somit muss man kein Auto direkt löschen, wenn es nur temporär nicht verfügbar sein soll.

Über den “Aktualisieren”-Button (2) werden die angepassten Eigenschaften des Autos übernommen.

Fahrtenübersicht

In der Fahrtenübersicht findet sich eine Tabelle, in der alle gebuchten Fahrten hinterlegt sind. Hier lassen sich auch direkt sowohl alle Fahrtetails wie der Buchungszeitraum als auch die Mitfahrenden einsehen. Über das Dropdown-Menü kann man diese Tabelle nach Fahrten filtern, in die man selbst involviert ist oder nach allgemein allen.

Fahrten Filter
Alle Fahrten

Zweck	Startadresse	Zieladresse	Startdatum	Enddatum	Fahrzeug	Mitfahrer
Seminar	Stuttgart	München	10. Dezember 2023 von 12:11 Uhr	12. Dezember 2023 bis 13:55 Uhr	BMW i4, S-DS-1234	Martin Beck
Supermarkt	Friedrichshafen	Friedrichshafen	11. Dezember 2023 von 09:00 Uhr	11. Dezember 2023 bis 09:30 Uhr	BMW i3, FN-DS-3333	Peter Maier
Betriebsausflug	Friedrichshafen	Dortmund	21. Januar 2024 von 09:00 Uhr	25. Januar 2024 bis 15:30 Uhr	BMW 320i, FN-DS-2323	Martin Beck Michale Schenider Alex mit Langen Nachnamen Julia Moser
Seminar	Stuttgart	Karlsruhe	20. März 2024 von 09:00 Uhr	21. März 2024 bis 17:30 Uhr	BMW i4, S-DS-1234	Peter Maier
Seminar	Stuttgart	Singen	17. März 2024 von 12:11 Uhr	18. März 2024 bis 12:02 Uhr	BMW i4, S-DS-1234	Peter Maier
Grillen	Stuttgart	Stuttgart	26. März 2024 von 13:30 Uhr	26. März 2024 bis 14:15 Uhr	BMW i4, S-DS-1234	Peter Maier
Möbel abholen	Stuttgart	Stuttgart	7. Januar 2024 von 11:30 Uhr	7. Januar 2024 bis 12:55 Uhr	BMW i4, S-DS-1234	Alex mit Langen Nachnamen
Einlaufen	Stuttgart	Friedrichshafen	2. Januar 2024 von 11:00 Uhr	2. Januar 2024 bis 12:00 Uhr	BMW i4, S-DS-1234	Martin Beck Michale Schenider

Alle Fahrten (Tabelle)



Fahrten Filter

Statistik

Im Statistik-Reiter wird dem Benutzer eine Tabelle angezeigt, auf welcher er sehen kann, welcher Mitarbeiter wie oft ein Poolfahrzeug verwendet hat. Dabei zählt auch das Mitfahren auf einer Fahrt in diese Statistik hinein. Die Tabelle kann man zwischen einer aktuellen Monats- und einer Gesamtstatistik umschalten.

Statistik Zeitraum
Insgesamt

Position	Name	Anzahl Fahrten
1	Peter Maier	4
2	Martin Beck	3
3	Michale Schenider	2
4	Alex mit Langen Nachnamen	2
5	Julia Moser	1
6	Sabrina Hensle	0

Statistik (Tabelle)



Statistik Filter

Technische Aspekte

Login und Authentifizierung

Für das Anmelden benötigt man einen Benutzernamen und ein Kennwort.

Zunächst wird eine Authentifizierungsanfrage an den User Management Service von doubleSlash gesendet. Nach erfolgreicher Antwort (Status Code 200 OK) werden im Körper der Antwort Informationen zum Benutzer, sowie ein *“Long Time Token”* zurückgesendet, welcher eine Gültigkeit von 8 Stunden hat.

Die Daten werden vom Frontend in den Local Storage vom benutzten Browser gespeichert, damit eine erneute Anmeldung nicht benötigt wird. Vorteilhaft ist daran, dass der LocalStorage persistent ist und sich nicht nach jeder Browser-Sitzung zurückgesetzt wird. Außerdem ist keine andere Website berechtigt, Informationen aus diesem zu unserer Website abzurufen und zu verändern.

Mithilfe des *“Long Time Token”* lässt sich ein *“Short Time Token”* über den User Management Service generieren, welcher wiederum nur 10 Minuten gültig ist und nach Ablauf dieser Zeit wieder neu generiert werden muss. Dieser wird benötigt, um z.B. Benutzer von dem User Management Service abzurufen oder Anfragen an das Backend zu senden.

(Für einen beispielhaften Ablauf von Netzwerk-Anfragen und -Antworten, siehe Sequenzdiagramm weiter unten)

Validierungen

Bei der Anlegung oder Bearbeitung einer Fahrt finden generell im Frontend Validierungen statt, die es gar nicht ermöglichen, invalide Anfragen an das Backend zu senden. So ist es z.B. nicht möglich, eine Fahrt mit einem Auto, welches zum Buchungszeitraum schon gebucht ist, anzulegen. Ungeachtet dessen werden Anfragen an das Backend trotzdem sorgfältig geprüft. So werden z.B. Anfragen, die eine bestimmte Ressource per ID bekommen möchten, auf die Existenz dieser Ressourcen zu der angefragten ID überprüft. Sollten diese nicht gefunden werden, werden Exceptions geworfen (*CarNotFoundException*, *RideNotFoundException*, *ReservationNotFoundException*) und vom globalen Exception Handler aufgefangen und eine HTTP Rückgabe mit dem Status Code 404 (Not Found) zurückgesendet. Wenn man auch eine Fahrt über das Backend erstellen möchte, wird z.B.

überprüft, ob das Enddatum des Buchungszeitraums vor dem Startdatum ist. Dieses Szenario ist vom Frontend aus gar nicht möglich, da das Datumsauswahl-Eingabeelement dies gar nicht zulässt. Sollte trotzdem so eine invalide Anfrage an das Backend gestellt werden, wird eine *IllegalRideException* geworfen (und vom *GlobalExceptionHandler* aufgefangen) und die Rückantwort mit dem HTTP Status Code 400 (Bad Request) zurückgesendet. Die gleiche Vorgehensweise findet mit einer Fahrt statt, welche mit anderen kollidiert, sprich das Auto der Fahrt zum gleichen Buchungszeitraum schon von einer anderen Fahrt belegt ist. Auch sollte es vom Frontend aus nicht möglich sein, eine Reservierung für einen Nutzer doppelt anzulegen. Dies wird ebenfalls im Backend noch einmal überprüft und wenn nötig eine *IllegalReservationException* geworfen, welche auch in einem HTTP Status Code 400 (Bad Request) resultiert.

Testing

Für das Schreiben der Tests haben wir uns hauptsächlich auf das Backend konzentriert, da dieses vor allem Daten verarbeitet und diese auch in der Datenbank hinterlegt. Somit hat es mehr schwerwiegende Konsequenzen, wenn das Backend fehlerhaft operiert.

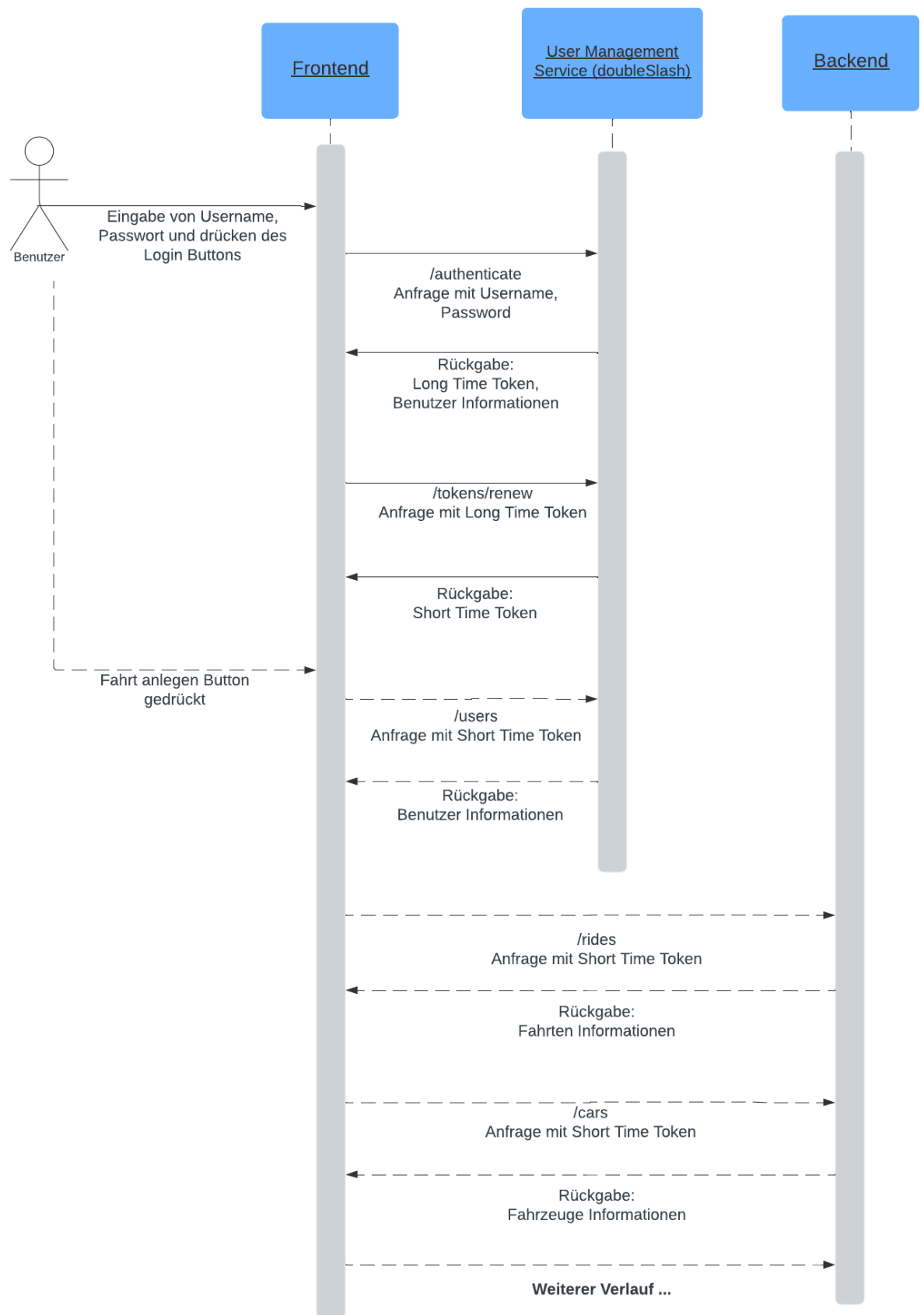
Generell sind wir dort vorgegangen, dass wir die einzelnen Komponenten getrennt voneinander testen, um Folgefehler zu vermeiden. So werden Repositories, also Komponenten, die Entitäten mit Datenbankanbindung verwalten, auf die einzelnen CRUD-Operationen (Create, Read, Update, Delete) getestet und gut gewählte Assertionen überprüft. Die Service Klassen werden ebenfalls auf ihre Funktionalität getestet, also Rückgabewerte verglichen, geworfene Exceptions überprüft, etc. Hierbei werden auch Komponenten, von denen die Services abhängen, durch Mock-Objekte ersetzt, um getrenntes Testen der einzelnen Komponenten zu gewährleisten. Letztendlich gibt es auch Tests für die Controller Klassen, welche Anfragen an die REST API Schnittstellen verarbeiten. Für diesen Zweck werden Anfragen an die Schnittstellen mit Testdaten im Json Format simuliert und das Verhalten des Controllers zu diesen Anfragen überprüft. Auch hier werden Mock-Objekte genutzt.

Im Frontend haben wir hauptsächlich manuelle Tests durchgeführt, aber auch die vom Angular Framework bereits generierten Tests zu den Komponenten genutzt.

E-Mail Versand und Empfang

Zum Testen der durch Spring-Mail versendeten E-Mails haben wir uns für Mail Hog entschieden. Dies ist ein Tool mit grafischer Benutzeroberfläche, welches als SMTP Server fungiert und gesendete E-Mails auffängt und grafisch wiedergibt. Generell senden wir E-Mails an die jeweiligen betroffenen Passagiere, wenn eine Reservierung zu einer Fahrt angelegt wird, Informationen aus einer Fahrt geändert werden oder wenn eine Fahrt gelöscht wird. Auch lassen sich in Mail Hog Anhänge an E-Mails einsehen und herunterladen. Dieses Feature haben wir genutzt, um die angehängten ICS-Dateien auszulesen, die wir auch im Backend generieren. Jede ICS-Datei erzeugt/aktualisiert/löscht einen Eintrag aus dem Kalenderprogramm, indem man die Datei öffnet. Oder im Falle von Outlook wird der Kalendereintrag direkt aus der E-Mail erkannt und kann in den integrierten Kalender importiert werden.

Beispielablauf: Anfragen/Antworten (Sequenzdiagramm)



Features & Aufteilung

- Erste Schritte des Projekts:
 - Erstellung einer WhatsApp Gruppe und Discord Server für die Kommunikation (Edwin)
 - Git Repository erstellt (Edwin)
 - Brainstorming vor Ort bei doubleSlash um Anforderungen zu vervollständigen und Priorisieren (Edwin, Leonard, Okan, Nick, Cedric)
 - Erste Überlegen vom ERM mit doubleSlash (Edwin, Leonard, Okan, Nick, Cedric)
 - User-Stories aus den Anforderungen formuliert (Edwin)
 - Kanban Board erstellt und User-Stories eingepflegt (Edwin, Leonard, Okan)
 - Einrichtung von IntelliJ und einbindung von Git Repository "Fuhrparkmanagement" (Edwin, Leonard, Okan, Nick, Cedric)
 - Erstellung eines erweiterten ERM (Edwin, Leonard, Okan, Nick, Cedric)
- Installation der Technologien:
 - Einrichtung von Spring Boot mit Abhängigkeiten und Angular (Leonard, Edwin, Okan)
 - Installation von Docker/WSL, npm und PostgreSQL (Leonard, Okan)
 - Installations Dokumentation geschrieben für alle Teammitglieder (Okan)
- Backend Einrichtung mit Datenbank:
 - Erstellung der model Klassen anhand des ERM (Leonard, Okan, Edwin)
 - Erstellung von Repository und CRUD Controller (Okan)
 - Einrichtung von Swagger zum Generieren der Schnittstellen (Edwin, Leonard)
- Frontend Einrichtung:
 - Einarbeiten in Frontend (Nick, Cedric)
 - Erstellung erster Components "Startseite, Fahrt anlegen, Statistik" (Nick)
 - Erstellung von Side-Bar und Page-Routing (Nick)
 - Favicon erstellt und dynamische Seitentitel (Okan)
- Login/Logout:
 - Login Component erstellt und Styling verändert (Cedric)
 - Login/Logout funktionalität Token im Browser halten und löschen (Okan)
 - Long Time Token über User Management Service validieren (Leonard)
- Big-Kalender auf der Startseite:
 - Informieren über Kalender Möglichkeiten (Nick)
 - Kalender Component und Big-Kalender implementiert (Nick, Cedric)
 - Kalender Funktionalitäten erstellt (Monat/Woche) (Okan)
 - Buchungsprozess über ein Button erstellt, 3 Schritte Pop-Up Dialog (Leonard, Edwin, Okan)
 - Eingetragene Daten validieren und in Datenbank ablegen (Leonard)
- Alle Fahrten
 - Alle Fahrten Component erstellt und Styling verändert (Nick)
 - Existierende Fahrten formatiert und übersichtlich in einer Tabelle anzeigen (Edwin, Okan)
 - Anfordern der Daten über das Backend (Edwin, Okan)
 - Filter Möglichkeit über ein Dropdown erstellt: Eigene/Alle Fahrten (Okan)
- Autos verwalten

- Autos verwalten Component erstellt und gestylt (Nick, Cedric, Leonard)
- TextFelder erstellt zum eintragen eines Autos (Leonard)
- Verarbeitung der Daten über das Backend (Leonard)
- Existierende Autos übersichtlich in einer Tabelle anzeigen (Leonard)
- Autos Bearbeiten und Löschen (Leonard)
- Statistik
 - Statistik Component erstellt und gestylt (Nick)
 - Leaderboard-Design (Nick, Edwin)
 - Anzahl der Fahrten anzeigen lassen (Edwin, Leonard)
 - DropDown für: Aktueller Monat/ Insgesamt (Edwin, Leonard)
- E-Mail Konfiguration
 - Informieren über Möglichkeiten: Microsoft Graph/Mailhog (Edwin)
 - Einarbeitung und Konfiguration in Mailhog (Edwin)
 - Erstellung von Email Service zum versenden von Mails (Edwin, Leonard)
 - Erstellung von ICS Service zum versenden von Kalendereinträge (Edwin, Leonard, Okan)
- Abschluss Konfiguration:
 - bwCloud einrichtung und erstellung von Debian Server (Leonard, Edwin, Okan)
 - slashDrive Webanwendung auf bwCloud Hosten (Leonard, Edwin, Okan)
 - Erstellung von production compose file (Leonard, Edwin, Okan)
 - slashDrive getestet und für Live-Demo vorbereitet (Leonard, Edwin, Okan)
- Wöchentliche Meetings:
 - Vorbereitung der Wöchentlichen Meetings (Edwin, Leonard, Okan)
 - Protokolle wurden abwechselnd erstellt (Edwin, Leonard, Okan, Nick, Cedric)
- Dokumentationen und Präsentationen:
 - Meilenstein 1 (Okan, Edwin)
 - Zwischenpräsentation (Okan, Edwin, Leonard, Nick, Cedric)
 - Meilenstein 2 (Okan, Edwin, Leonard, Nick, Cedric)
 - Meilenstein 3 (Okan, Edwin, Leonard)
 - Meilenstein X (Okan, Edwin, Leonard, Nick, Cedric)
 - Endpräsentation (Okan, Edwin, Leonard, Nick, Cedric)
- Weitere Aufteilungen:
 - CI/CD Pipeline erstellt auf master branch (Leonard)
 - Tests für Cars, Rides... erstellt, getrenntes Tesing der einzelnen Backend Komponenten (Repositories, Services, Rest API Schnittstellen) (Leonard)
 - Interne Meetings Planen und das Projekt leiten (Edwin, Okan)

Herausforderungen

Natürlich lief über den Zeitraum des Projekts nicht alles so, wie wir es uns von Anfang an kalkuliert hatten. Wir haben von Anfang an nicht jedes Detail vollständig durchgeplant, da sich auch die Anforderungen während des Projekts geändert haben. Auch haben wir einen agilen Ansatz verfolgt, welcher Einarbeitungszeit benötigt hat. Zudem musste man sich erstmal in der Gruppe kennenlernen und eine Gruppendynamik entwickeln. Dafür haben sich aber hervorragend die Projekttag zur Teambildung und Konfliktlösung herausgestellt. Außerdem mussten wir uns erst einmal mit den Technologien vertraut machen. Hierfür mussten wir einige Zeit investieren, da wir mit einigen Technologien noch keine Erfahrung hatten, auch insbesondere im Zusammenspiel dieser.

Neben den technischen Herausforderungen hatten wir auch zum ersten Mal ein Projekt in diesem Ausmaß und einen realen Kunden, was viel Projektmanagement verlangt hat.

Was vor allem zu Beginn nicht so wie geplant lief, waren die einzelnen User-Stories, die auf unserem GitHub Kanban Board standen. Da wir anfangs viel Zeit brauchten, um alles aufzusetzen und uns mit den Technologien vertraut zu machen, waren wir anfangs nicht bis zum nächsten Meeting in der folgenden Woche komplett fertig. So zogen sie sich über mehr als eine Woche hinweg. Das hatte zur Folge, dass im Meeting mit dem Kunden nur verschiedene Teile von verschiedenen User-Stories gezeigt werden konnten, und nicht ganz abgeschlossen waren. Dies konnten wir jedoch lösen, indem wir organisierter gearbeitet haben. Vor allem kamen auch im Entwicklungsprozess wichtigere Features dazwischen, an denen zuerst gearbeitet werden musste. Wir hätten die einzelnen User-Stories mehr aufteilen, und der Priorisierung dieser mehr Bedeutung zuweisen müssen.

Mehr Zeit als gedacht hat auch das Erstellen und das spätere Überarbeiten des Entity-Relationship-Models in Anspruch genommen. Bei unseren ersten Entwürfen haben ein paar Details gefehlt. Nach sorgfältiger Überarbeitung und mehreren Iterationen an Entwürfen haben wir uns dann auf ein Modell festlegen können.

Zum Thema intuitives Design mussten wir uns auch viele Gedanken machen, was wir so ursprünglich gar nicht so sehr gedacht hatten. Wo zum Beispiel ein Button ist, wie klar gemacht wird, was dieser bewirkt, und an welcher Stelle im Buchungsprozess welche Informationen eingetragen werden können.

Nun können wir als Team sagen, dass uns das Projekt und die Überwindung der aufgetretenen Herausforderungen viel Erfahrung übermittelt hat und wir aus diesen viel gelernt haben.

Installation auf einem Linux-Server (Debian)

Da unsere Anwendung mit Docker Desktop mit WSL auf Windows sehr anspruchsvoll ist, haben wir uns dafür entschieden, unsere Anwendung auf einem Server zu installieren. So müssen wir die Anwendung nicht lokal in unserer Entwicklungsumgebung starten, sondern können einfach die Webseite aufrufen und unsere Anwendung dort live demonstrieren. Außerdem testen wir so auch gleich, wie unsere Anwendung später tatsächlich installiert werden würde.

Unter <https://portal.bw-cloud.org/> bekommt man als Student kostenlos einen minimalistischen Server bereitgestellt. Hier haben wir uns eine Instanz erstellt und alle benötigten Ports freigegeben. Da wir lokal Java und Node JS mit Angular installiert haben, auf dem Server aber nicht, mussten wir eine extra `production_compose.yml` Docker Compose Konfigurationsdatei hinzufügen. Diese startet das Frontend und Backend dann auch, sowie die anderen benötigten Services, wie MailHog, PostgreSQL, etc.

Da es jetzt nicht mehr lokal läuft, mussten wir an ein paar Stellen, bei der Kommunikation von Frontend und Backend, etwas anpassen.

Da der Server nur über 1 Gigabyte Arbeitsspeicher verfügt, mussten wir das Frontend lokal bauen und das Image manuell auf den Server hochladen, da dies auf dem Server aufgrund von Ressourcenknappheit nicht geklappt hat.

Nun läuft der Server und ist zum aktuellen Zeitpunkt im Internet erreichbar unter <http://193.197.228.214:4200> .

Installationshandbuch

Git Repository klonen mit:

```
$ git clone git@github.com:edwinst/Fuhrparkmanagement.git
```

Mit Docker Compose die verschiedenen Container (erstellen und) starten:

```
$ docker-compose -f production_compose.yml up -d
```

Administrationshandbuch

Die Standardkonfiguration der veränderbaren Konfigurationsparameter ist wie folgt in der application.yml Datei beschreiben:

application.yml:

```
spring:
  jpa:
    hibernate:
      ddl-auto: update
      generate-ddl: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.PostgreSQLDialect
  datasource:
    url: 'jdbc:postgresql://localhost:5432/fuhrparkmanagement'
    username: 'admin'
    password: 'secret'
  mail:
    host: '127.0.0.1'
    port: 1025
    test-connection: true
    username: 'admin'
    password: 'secret'
    from: "test@fuhrparmanagement.com"
  authentication:
    jwt-secret: 'hAdsJsFnV5TxvkuWKr...'
  user-management-api:
    url: 'http://localhost:8082'
    api-key: 'secret'
```

Möchte man nun einen Konfigurationsparameter ändern, ist dies am besten über Umgebungsvariablen für den jeweiligen Docker Container zu erreichen.

Wenn man z.B. für das Backend das Passwort des Benutzers für die PostgreSQL Datenbank ändern möchte, setzt man z.B in der gewählten Docker Compose Konfigurationsdatei eine Umgebungsvariable mit dem Namen `SPRING_DATASOURCE_PASSWORD`: 'Wert des Passworts' (vergleiche Namensschema mit `spring.datasource.password`: 'secret' in application.yml). So würde der

Konfigurationsabschnitt für den Backend Service in der Docker Compose Datei so aussehen:

```
backend:
  build: .
  environment:
    SPRING_DATASOURCE_PASSWORD: 'Wert des Passworts'
    # Weitere Umgebungsvariablen/Konfigurationsparameter ...
  ports:
    - 8080:8080
```

Lizenz

Das GitHub Repository wurde veröffentlicht und unter der (MIT) Lizenz lizenziert.

MIT License

Copyright (c) 2024

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Ausblick: Was kann noch ergänzt werden?

Abschließend können wir als Team sagen, dass uns das gesamte Projekt sehr viel Spaß gemacht hat und wir froh sind, dass wir im Rahmen unseres Studiums das Softwareprojekt mit doubleSlash absolvieren durften. Für die Weiterentwicklung der App haben wir uns Gedanken gemacht, welche weiteren Features und Quality-of-Life-Changes noch für unsere Anwendung ergänzt werden könnten:

- Beim ersten Einloggen ein Hilfe-Assistent der alle wichtigen Schritte erklärt
- Ein Hilfe Menü um Probleme oder Fragen zu klären
- Mehr Filter für die Fahrtentabelle (Standort, Sitzplätze)
- Mehr Statistiken, z.B. die gefahrenen Kilometer, wie oft welches Auto gefahren wurde, die am meisten gefahrene Strecke, ...
- Autocomplete zu echten Adressen bei der Adresseingabe (OpenStreet Map)
- Farbliche Unterteilung der Fahrten im Kalender
- Zwischenstopps zu einzelnen Fahrten hinzufügen können
- Die Anzeige der Autos mit Bilder erweitern
- Domainname anpassen, z.B.: www.slashDrive.de
- Dark-Mode