



Projektdokumentation

Murtaj Drin
murtadri@students.zhaw.ch
W.BA.WIN.21HS.VZc

ZHAW School of Management and Law
Software Engineering 2
Frühlingssemester 2023

Inhaltsverzeichnis

1	Einleitung	3
1.1	Explore-Board	4
1.2	Create-Board	5
1.3	Evaluate-Board	6
1.4	Erkenntnisse aus dem Pitch	6
2	Anforderungen	7
2.1	Use-Case Diagramm	7
2.2	ER-Modell	8
2.3	BPMN-Diagramm.....	9
2.4	Skizze des UIs.....	10
2.4.1	User-Ansicht	10
2.4.2	Administrator-Ansicht	12
3	Implementation	14
3.1	Beschreibung des Frontends	14
3.2	Klassendiagramm	21
3.3	Drittsystem	22
4	Testing.....	23
4.1	Modul- und Integrationstests.....	23
4.1.1	CarController.java.....	23
4.1.2	UserController.java	24
4.1.3	ServiceController.java	24
4.2	User-Tests.....	26
5	Fazit	28

1 Einleitung

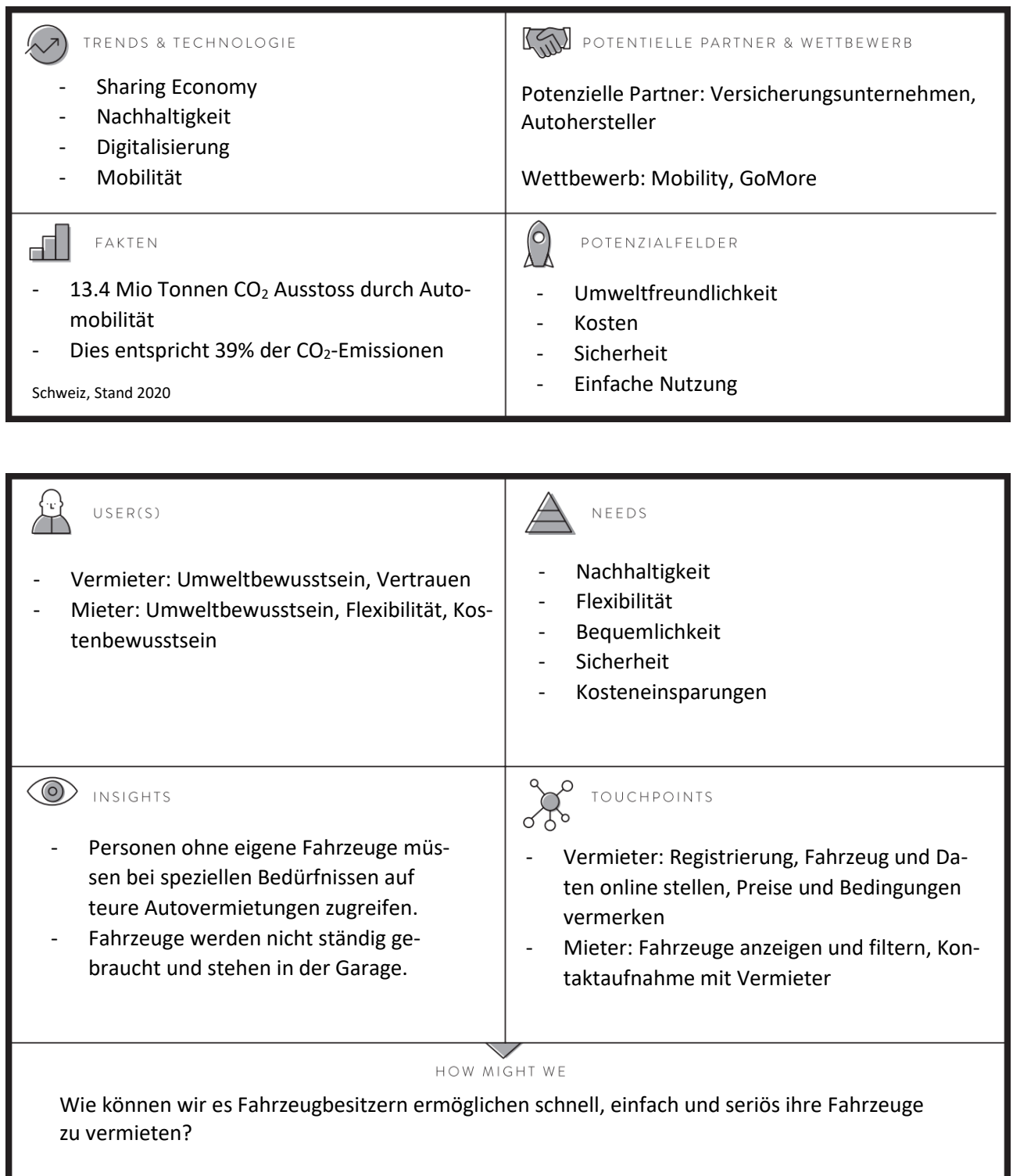
Peer2Vehicle ist eine innovative Online-Plattform, die als Bindeglied zwischen privaten Fahrzeugbesitzern und potenziellen Fahrzeugmietern fungiert. In der heutigen Gesellschaft sind die Strassen oft überfüllt, was verschiedene Probleme wie Umweltverschmutzung, Verkehr und eine erhöhte Nachfrage an Parkplätzen verursacht. Die Kernidee von Peer2Vehicle besteht darin, diese problematische Situation zu lösen.

Unsere Plattform ermöglicht es privaten Autobesitzern, Fahrzeuge auf einfache und effiziente Weise zu vermieten. Anstatt ungenutzt geparkt zu bleiben, können diese Fahrzeuge von anderen genutzt werden, was die Anzahl der Fahrzeuge auf den Strassen reduziert und einen Beitrag zu einer nachhaltigeren Zukunft leistet. Darüber hinaus bietet Peer2Vehicle Fahrzeugbesitzern die Möglichkeit, durch die Vermietung ihrer Fahrzeuge zusätzliche Einnahmen zu erzielen.

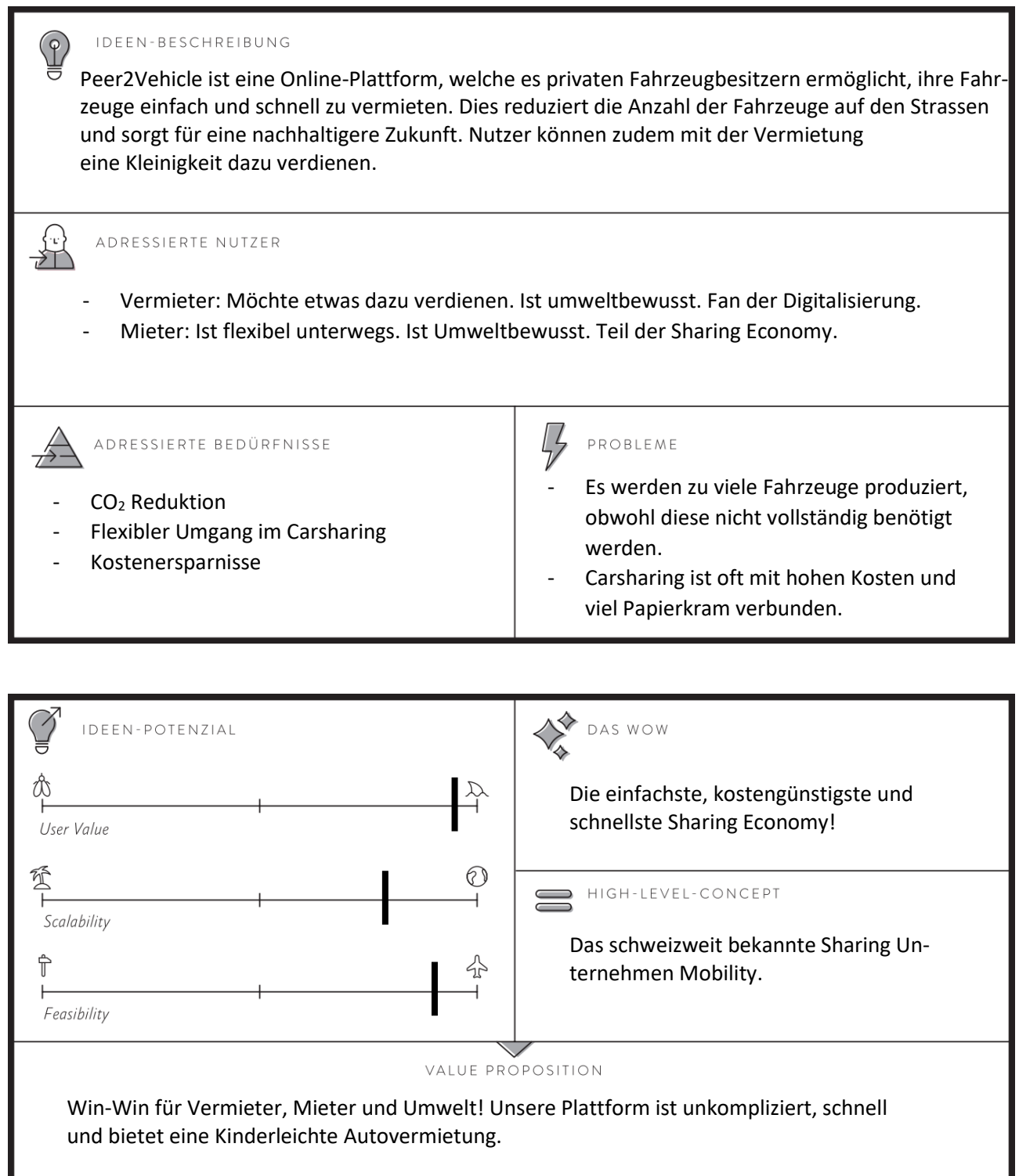
Unsere Zielgruppe besteht sowohl aus Autobesitzern, die einen Mehrwert aus ihren nicht genutzten Fahrzeugen ziehen möchten, als auch aus Personen, die ein Auto vorübergehend benötigen, aber keine dauerhafte Investition wünschen. Peer2Vehicle stellt eine kosteneffektive und umweltfreundliche Lösung für beide Gruppen dar, die zu einer klassischen Win-Win-Situation führt.

Peer2Vehicle ist mehr als nur eine Mietplattform. Es ist ein Schritt hin zu nachhaltiger Mobilität und einer effizienteren Nutzung von Ressourcen. Ich bin davon überzeugt, dass wir durch die Verbesserung der Zusammenarbeit und die gemeinsame Nutzung von Ressourcen einen grossen Schritt in Richtung einer besseren und nachhaltigeren Zukunft machen können.





1.1 Explore-Board



1.2 Create-Board



1.3 Evaluate-Board

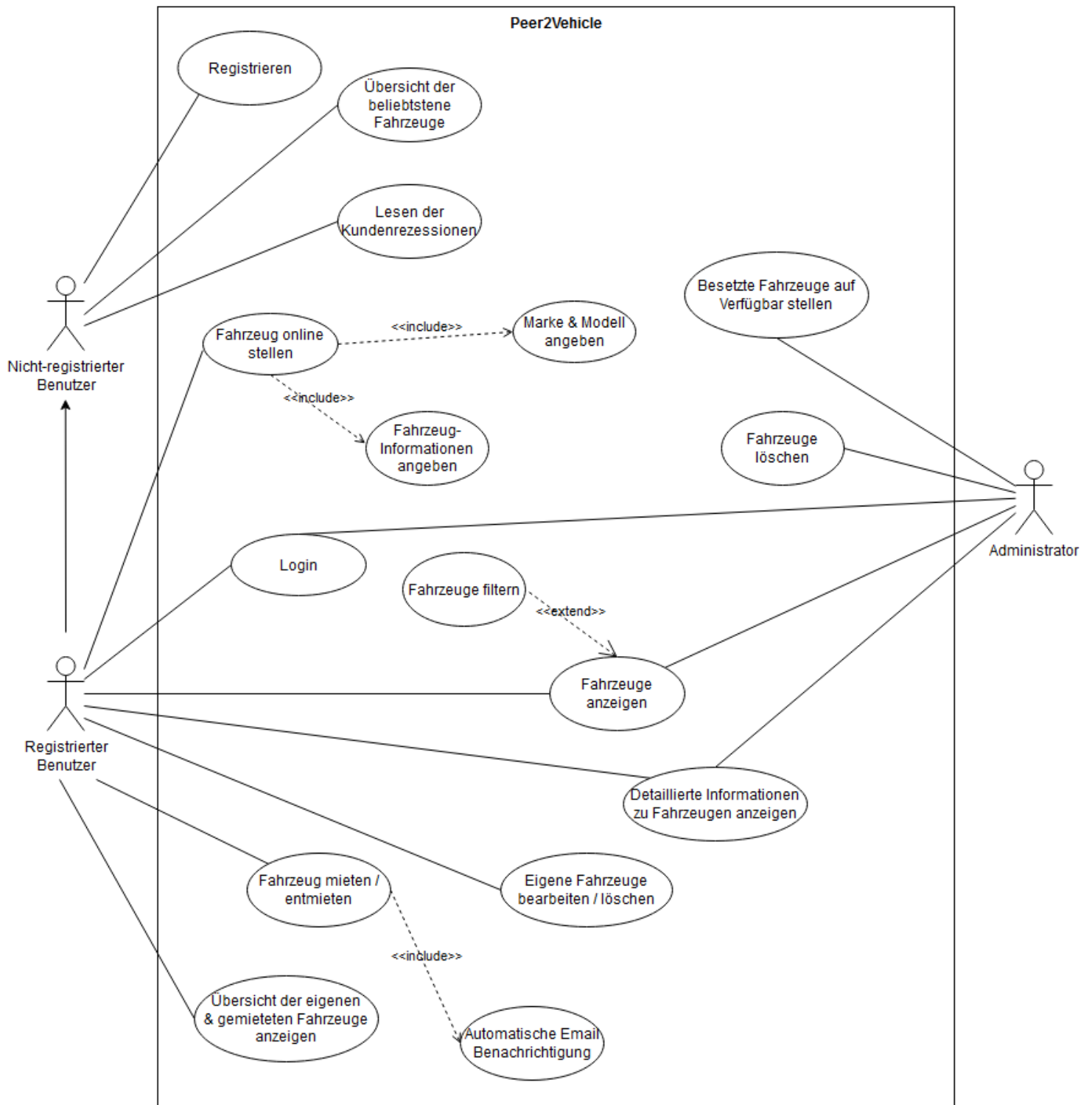
 Kanäle <ul style="list-style-type: none">- Facebook- Instagram- Twitter- Youtube-Ads- E-Mail-Marketing- Flyer an Orten mit hohem Verkehrsaufkommen	 Unfairer Vorteil <ul style="list-style-type: none">- Unglaublich schnelle Vermietung- Einfacher Prozess ohne Papierkram- Starkes Netzwerk- Partnerschaften mit wichtigen Akteuren
 KPI <ul style="list-style-type: none">- Anzahl registrierter Nutzer- Anzahl Fahrzeuge- Anzahl abgeschlossene Vermietungen- Gesamtumsatz	 Einnahmequellen <ul style="list-style-type: none">- Transaktionsgebühr für jede erfolgreich abgeschlossene Vermietung- Werbeflächen auf der Plattform für zusätzlichen Gewinn

1.4 Erkenntnisse aus dem Pitch

- Die Idee kam bei allen gut an.
- Ich konnte alle überzeugen, da allen bewusst ist, wie wichtig der Umweltschutz in der heutigen Zeit ist.
- Es gab vom Publikum keine Verbesserungsvorschläge, einzig die Problematik mit der Versicherung müsste man noch genauer anschauen.

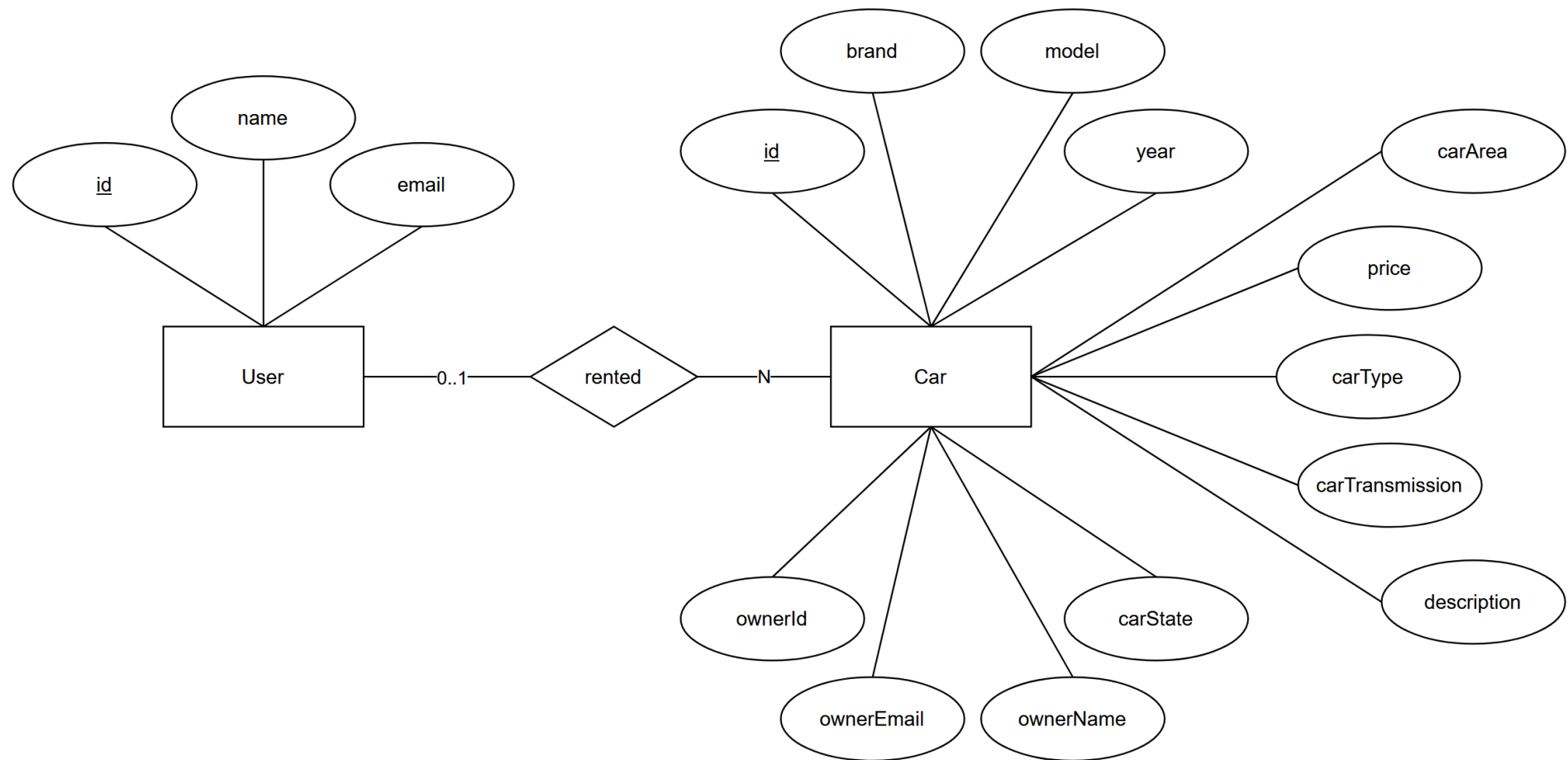
2 Anforderungen

2.1 Use-Case Diagramm



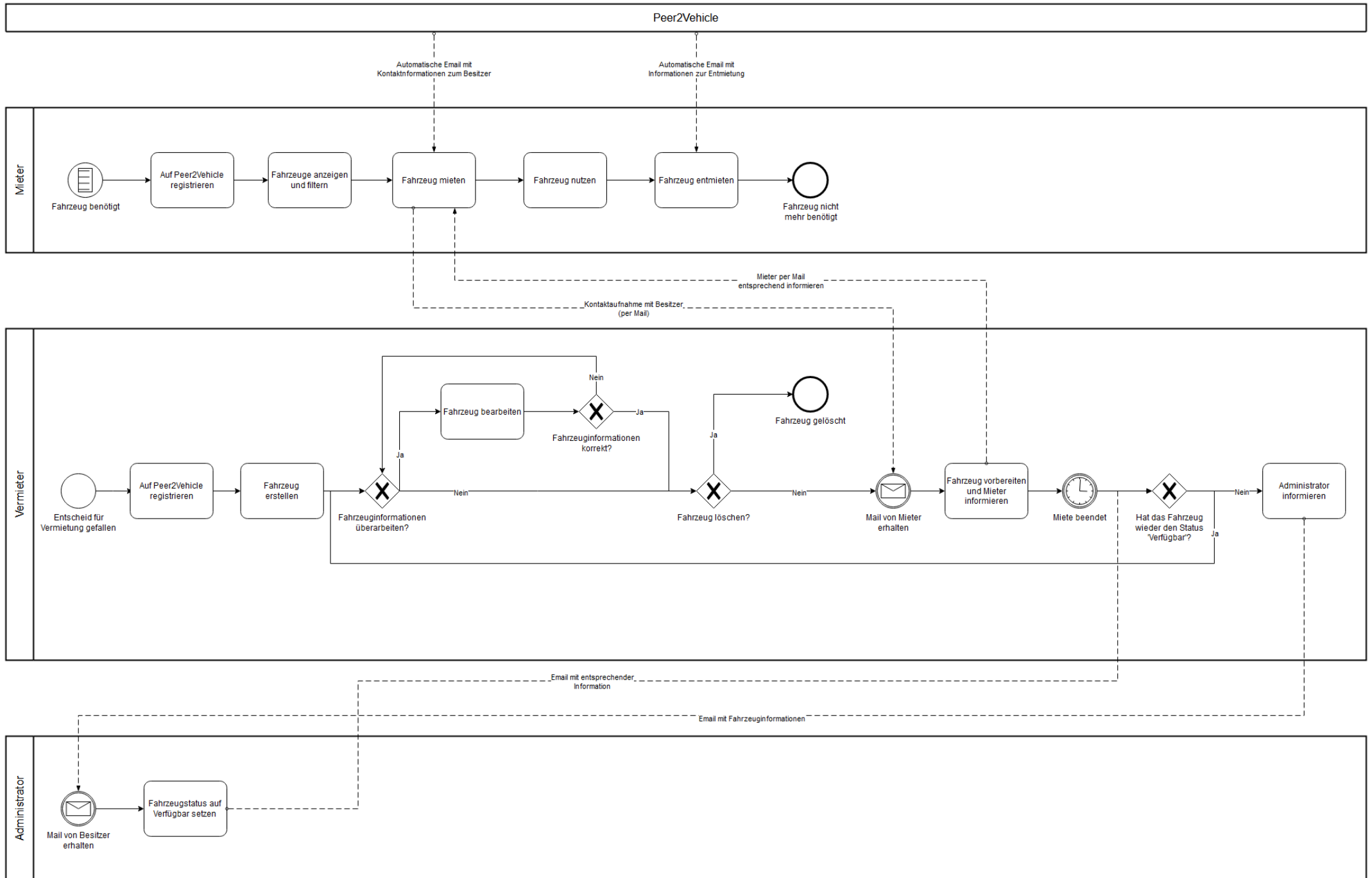
Das Use-Case Diagramm besteht aus drei Actors. Der **Nicht-registrierte Benutzer** kann die Homepage mit den beliebtesten Fahrzeugen und verschiedenen Kundenrezessionen sehen. Er kann sich über Auth0 auf der Plattform registrieren und ist dann ein registrierter Benutzer. Als **registrierter Benutzer** kann man Fahrzeuge online stellen, anzeigen, filtern, bearbeiten, löschen und mieten. Der **Administrator** hat die Möglichkeit alle Fahrzeuge zu löschen und besetzte Fahrzeuge auf Verfügbar zu setzen (falls ein Benutzer das gemietete Fahrzeug fälschlicherweise nicht entmietet).

2.2 ER-Modell



Der **User** besteht aus ID, Name und E-Mail. Das **Fahrzeug** hat mehrere verschiedene Attribute. OwnerName, OwnerEmail und OwnerId sind dabei die Attribute vom Fahrzeugbesitzer, also von dem User, der das Fahrzeug erstellt hat. Ein User kann ein oder mehrere Fahrzeuge, welche nicht von ihm erstellt worden sind, mieten. Bei der Miete wird der Name, die Emailadresse und die ID des entsprechenden Users in der Entität Car abgespeichert. Dies ist im Klassendiagramm weiter unten genauer ersichtlich.

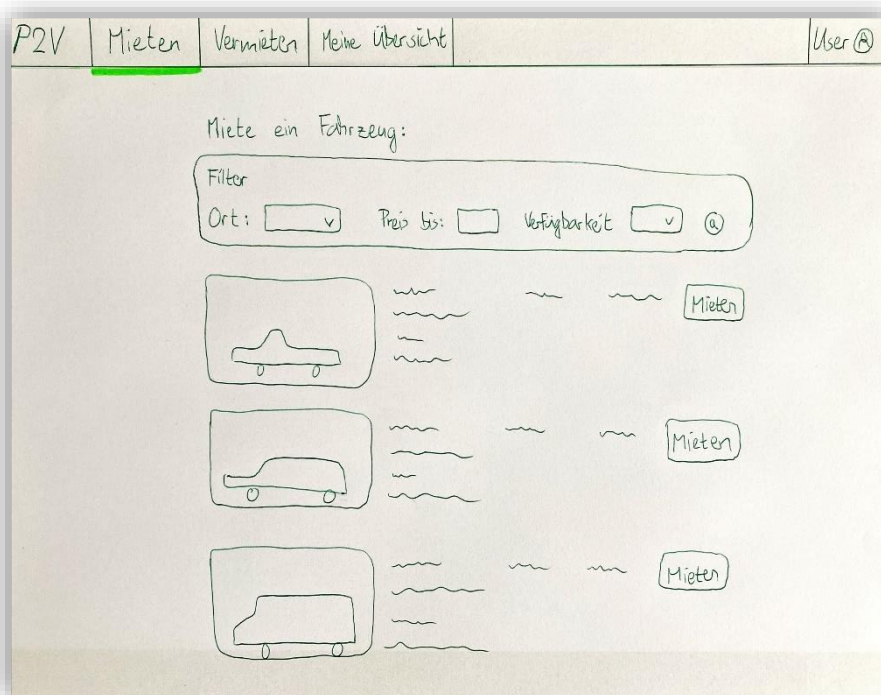
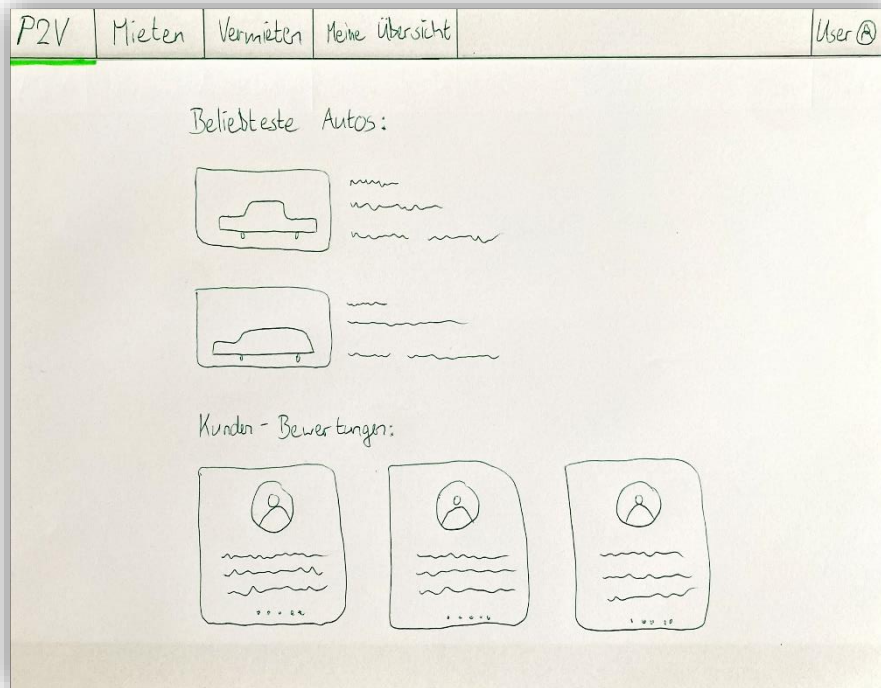
2.3 BPMN-Diagramm



Das BPMN-Modell zeigt einen kompletten Mietprozess von Peer2Vehicle. Dabei fungieren hauptsächlich Vermieter und Mieter. Beides sind gemäss dem Use-Case Diagramm Registrierte Benutzer. Der Administrator kommt nur dann zum Einsatz, wenn ein Fahrzeug fälschlicherweise noch den Status «Besetzt» hat. Der Administrator hat zusätzlich die Möglichkeit beliebige Fahrzeuge zu löschen, dies wurde im BPMN-Modell jedoch nicht dargestellt, da es in diesem Prozess nicht dazu kommt.

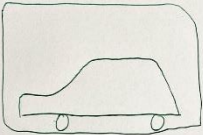
2.4 Skizze des UIs

2.4.1 User-Ansicht



P2V	<u>Mieten</u>	Vermieten	Meine Übersicht	User @
-----	---------------	-----------	-----------------	--------

Fahrzeug - Details



~~~~~

~~~~~

~~~~~

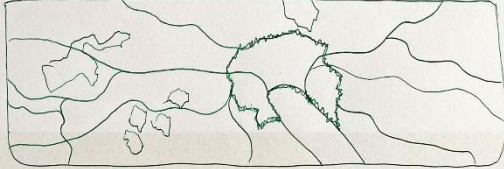
~~~~~

~~~~~

~~~~~

Mieten

Map:



P2V	Mieten	<u>Vermieten</u>	Meine Übersicht	User @
-----	--------	------------------	-----------------	--------

Vermiete dein Auto

Marke

Getriebe

Leistung

Modell

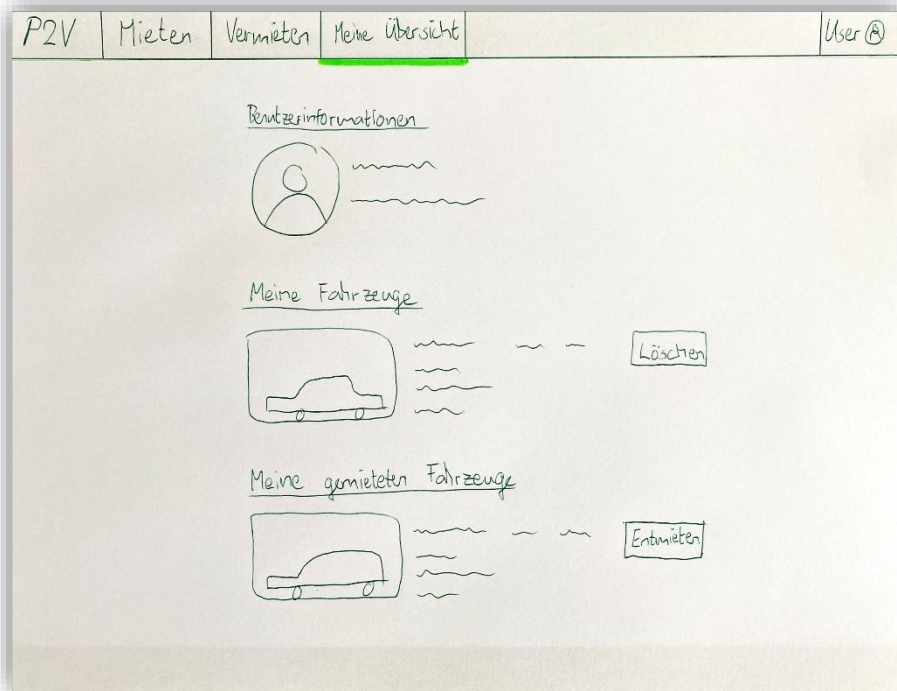
Treibstoff

Ort

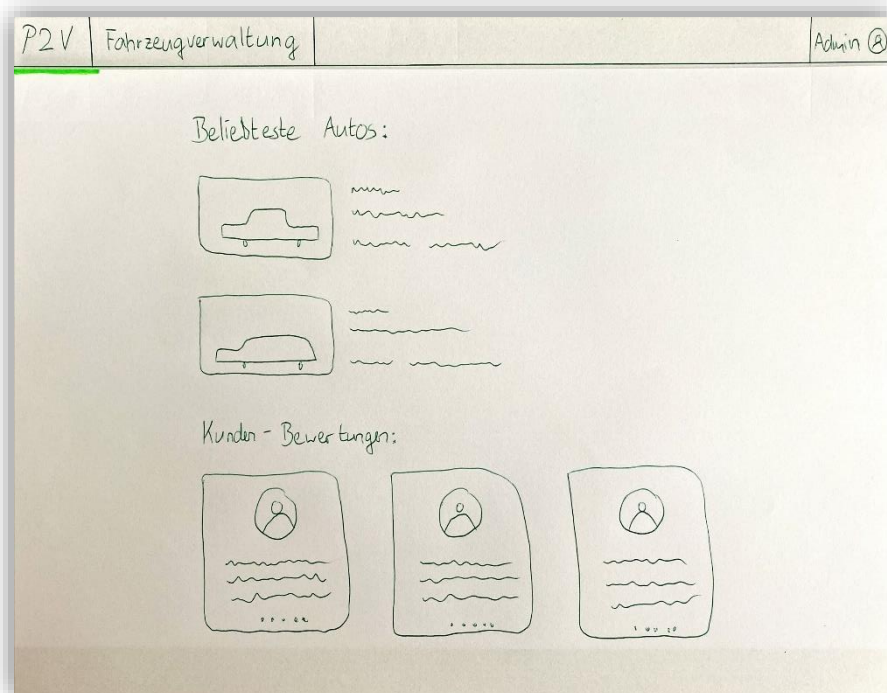
Preis

Beschreibung

Erstellen



2.4.2 Administrator-Ansicht



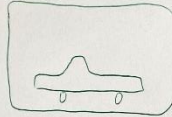
Fahrzeugverwaltung:

Filter

Ort: v

Preis bis:

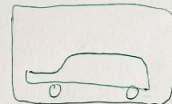
Verfügbarkeit v @



~~~~~  
~~~~~  
~~~~~

~~~~~  
~~~~~  
~~~~~

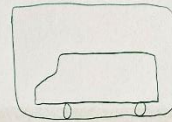
Löschen



~~~~~  
~~~~~  
~~~~~

~~~~~  
~~~~~  
~~~~~

Löschen

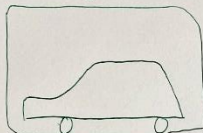


~~~~~  
~~~~~  
~~~~~

~~~~~  
~~~~~  
~~~~~

Löschen

Fahrzeug - Details



~~~~~  
~~~~~  
~~~~~  
~~~~~

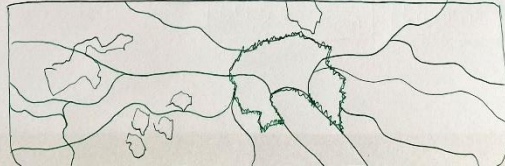
~~~~~

~~~~~  
~~~~~  
~~~~~

Verfügbar setzen

Löschen

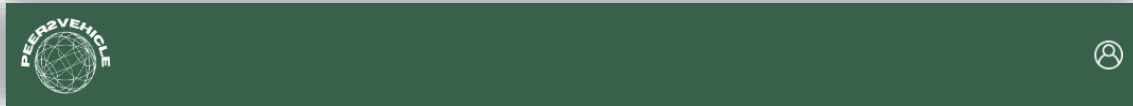
Map:



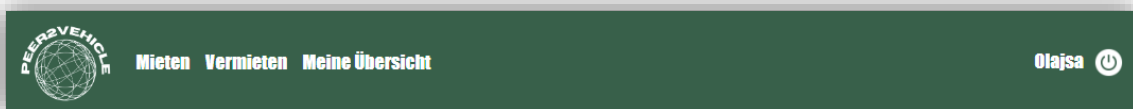
3 Implementation

3.1 Beschreibung des Frontends

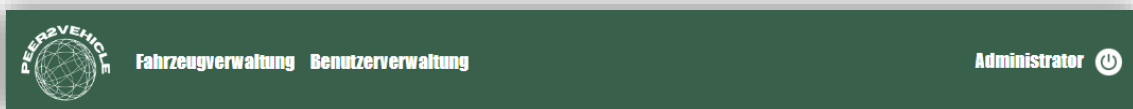
Das Frontend ist in einem schlichten grünen Stil aufgebaut. Als nicht registrierter Benutzer habe ich in der **Navigationsleiste** noch keine Menüpunkte:



Als registrierter Benutzer kann ich zwischen den Punkten «Mieten», «Vermieten» und «Meine Übersicht» wählen:



Der Administrator hat die Punkte «Fahrzeug- und Benutzerverwaltung» zur Auswahl. Dabei ist die Benutzerverwaltung eine Verlinkung zur entsprechenden Auth0-Oberfläche.



Unten auf der Seite wird der **Footer** mit verschiedenen Informationen zu Peer2Vehicle dargestellt. Diese Leiste ist auf jeder Seite zu sehen.



Auf der **Startseite** werden eine Beschreibung zu Peer2Vehicle, die beliebtesten Fahrzeuge und verschiedene Kundenrezensionen angezeigt. Diese Seite ist für alle 3 Actors zugänglich (Nicht-registrierter Benutzer, registrierter Benutzer und Administrator).

Verbessere die Welt mit Peer2Vehicle

Die derzeitige Situation in unseren Städten ist von überschüssigen parkierten Autos geprägt. Tatsächlich sind sie 96% der Zeit ungenutzt, und wenn sie verwendet werden, sind sie in den meisten Fällen nur von einer Person besetzt. Autos sind eine bedeutende Quelle für CO2-Emissionen. Eine effizientere Nutzung der vorhandenen Fahrzeuge durch gemeinsames Teilen könnte die Anzahl der Autos in unseren Städten signifikant reduzieren und damit einen wichtigen Beitrag zum Schutz unseres Planeten leisten.

Unsere beliebtesten Vehicles



Tesla Model Y

Umweltfreundlich, Zuverlässig und Schnell!

140 CHF/Tag



Ford Mustang

Ein perfektes Fahrzeug für Sommer- und Cabrio-Liebhaber.

130 CHF/Tag



Fiat Doblo

Toller und sparsamer Transportwagen, der Dir beim nächsten Umzug beiseitesteht.

100 CHF/Tag

Was sagen unsere Nutzer?



Denis

Tätowierer

"Als Tätowierer reise ich oft zu verschiedenen Tattoo-Events. Die umweltfreundliche Ausrichtung passt perfekt zu meiner Philosophie, meinen ökologischen Fussabdruck zu reduzieren."



Olajsa

Anwältin

"Ich bin beeindruckt von der Benutzerfreundlichkeit und Schnelligkeit von Peer2Vehicle. Als Anwältin mit einem vollen Terminkalender schätze ich die Flexibilität und Effizienz, die mir P2V bietet."



Lena

Floristin





"Als Floristin liebe ich die Natur, und Peer2Vehicle ermöglicht es mir, umweltbewusster zu handeln. Die Auswahl an Fahrzeugen ist grossartig und bietet mir ein passendes Auto."



Im Reiter **Mieten** werden dem registrierten Benutzer alle Fahrzeuge angezeigt. Diese können nach Ort, Preis und Verfügbarkeit gefiltert werden. In der rechten Spalte der Tabelle habe ich die Möglichkeit Fahrzeuge zu Mieten, Entmieten oder, wenn es mein Fahrzeug ist, zu löschen. Ist ein Fahrzeug besetzt, dann wird entsprechend kein Mietbutton angezeigt.


Miete ein Vehicle

Ort:
Preis bis:
Verfügbarkeit:

Vehicle	Ort	Preis	Verfügbarkeit	
 <div> VW ID4 2022 Elektrisch Single </div>	Altstätten	200 CHF/Tag	Besetzt	
 <div> Mini Paceman 2016 Diesel Automat </div>	Altstätten	160 CHF/Tag	Verfügbar	<input type="button" value="Miete starten"/>
 <div> Audi Q8 2019 Benzin Automat </div>	Windisch	220 CHF/Tag	Verfügbar	<input type="button" value="Löschen"/>
 <div> Audi etron 2022 Elektrisch Single </div>	Winterthur	190 CHF/Tag	Besetzt	<input type="button" value="Miete beenden"/>

Wenn auf ein Bild geklickt wird, öffnet sich die **Detailansicht** des entsprechenden Fahrzeuges. Hier werden, zu den Informationen aus der Tabelle, zusätzlich die Beschreibung und eine Map mit dem Standort des Fahrzeuges angezeigt. Die Buttons, die hier angezeigt werden, sind dieselben wie aus der vorherigen Tabelle.

Vehicle Informationen



Mini Paceman

Preis: 160 CHF/Tag

2016

Diesel

Automat

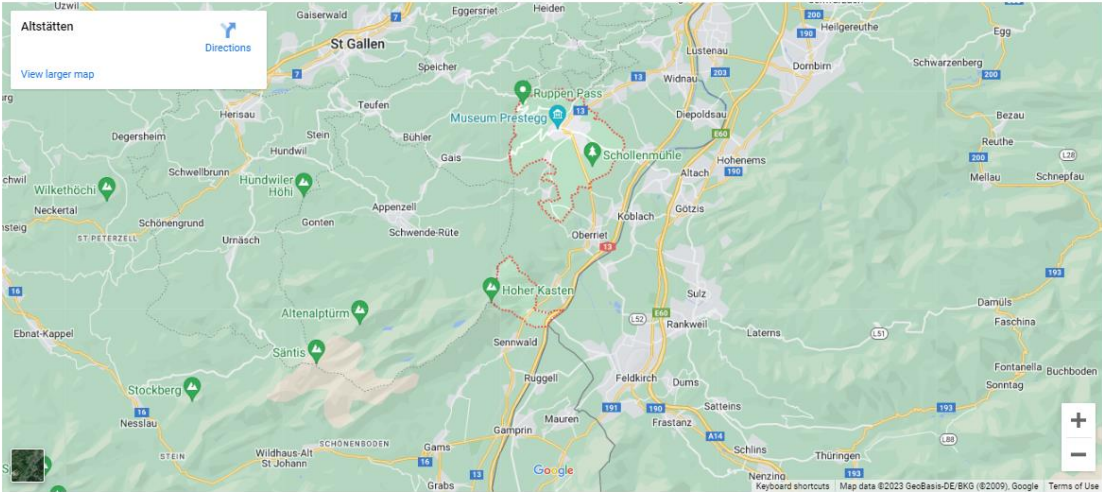
Altstätten

Verfügbar

Besitzer/in Denis sagt:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Scelerisque mauris pellentesque pulvinar pellentesque habitant. Facilisi morbi tempus iaculis urna. Pulvinar etiam non quam lacus suspendisse faucibus interdum posuere lorem. Lacus sed turpis tincidunt id aliquet risus feugiat in. cursus turpis massa tincidunt dui ut ornare lectus sit amet.

Miete starten



Im Reiter **Vermieten** kann ich als registrierter Benutzer ein Fahrzeug zum Vermieten online stellen. Je nachdem welches Modell man auswählt, wird ein entsprechendes Bild im Frontend angezeigt.

Vermiete dein Vehicle

Marke

Modell

Getriebe

Treibstoff

Jahrgang

Ort


Preis in CHF/Tag

Beschreibung

Erstellen



Im Reiter **Meine Übersicht** erhalte ich Informationen zu meinem Benutzerkonto, zu meinen Fahrzeugen und zu meinen gemieteten Fahrzeugen. Von hier aus hat man ebenso die Möglichkeit auf die Detailansicht der Fahrzeuge zu gelangen. Auch die Buttons funktionieren von dieser Ansicht aus. Wenn man in der Navigationsleiste auf den Benutzernamen klickt, kommt man ebenso zu dieser Ansicht.

Account Details




Benutzername: Olajsa
Email-Adresse: olajsa@mail.com

Meine Vehicles

Vehicle	Ort	Preis	Verfügbarkeit	
 <div>Audi Q8 📅 2019 ⛽ Benzin ⚙️ Automat</div>	Windisch	220 CHF/Tag	Verfügbar	<button>Löschen</button>
 <div>Ford Mustang 📅 2016 ⛽ Benzin ⚙️ Geschaltet</div>	Windisch	160 CHF/Tag	Verfügbar	<button>Löschen</button>





Meine gemieteten Vehicles

Vehicle	Ort	Preis	Verfügbarkeit	
 <div>Audi A3 📅 2015 ⛽ Diesel ⚙️ Automat</div>	Winterthur	100 CHF/Tag	Besetzt	<button>Miete beenden</button>

Als Administrator gelange ich zur **Fahrzeugverwaltung**. Es wird dieselbe Tabelle, wie im Reiter «Mieten» angezeigt, nur hat man hier als Admin die Möglichkeit alle Fahrzeuge zu löschen.

Fahrzeugverwaltung


Ort: Preis bis: Verfügbarkeit: Go!

Vehicle	Ort	Preis	Verfügbarkeit
 <div>VW ID4 📅 2022 🚗 Elektrisch 🚗 Single</div>	Altstätten	200 CHF/Tag	Besetzt
 <div>Mini Paceman 📅 2016 🚗 Diesel 🚗 Automat</div>	Altstätten	160 CHF/Tag	Verfügbar
 <div>Audi Q8 📅 2019 🚗 Benzin 🚗 Automat</div>	Windisch	220 CHF/Tag	Verfügbar
 <div>Audi etron 📅 2022 🚗 Elektrisch 🚗 Single</div>	Winterthur	190 CHF/Tag	Verfügbar

1 2 3

Wie beim normalen User gelangt man auch als Administrator mit einem Klick auf das Fahrzeug zur **Detailansicht**. Hier hat man wieder die Möglichkeit das Fahrzeug zu löschen. Wenn das Fahrzeug besetzt ist, kann man hier zusätzlich den Status des Fahrzeuges wieder auf Verfügbar setzen.

Vehicle Informationen

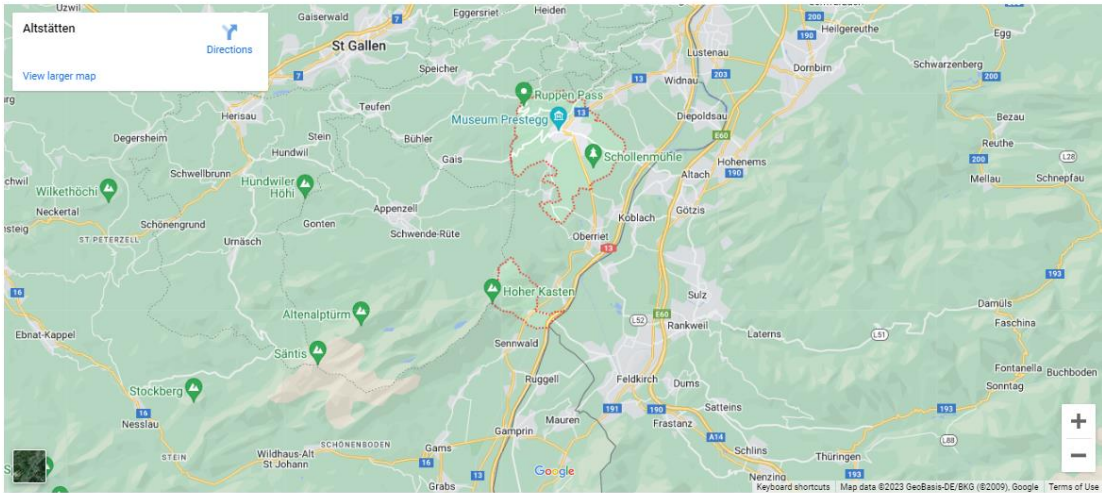


VW ID4
Preis: 200 CHF/Tag
📅 2022
🔌 Elektrisch
🚗 Single
📍 Altstätten
⦿ Besetzt

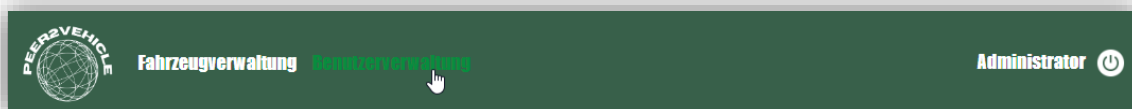
Besitzer/in Denis sagt:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Scelerisque mauris pellentesque pulvinar pellentesque habitant. Facilisi morbi tempus iaculis urna. Pulvinar etiam non quam lacus suspendisse faucibus interdum posuere lorem. Lacus sed turpis tincidunt id aliquet risus feugiat in. Cursum turpis massa tincidunt dui ut ornare lectus sit amet.

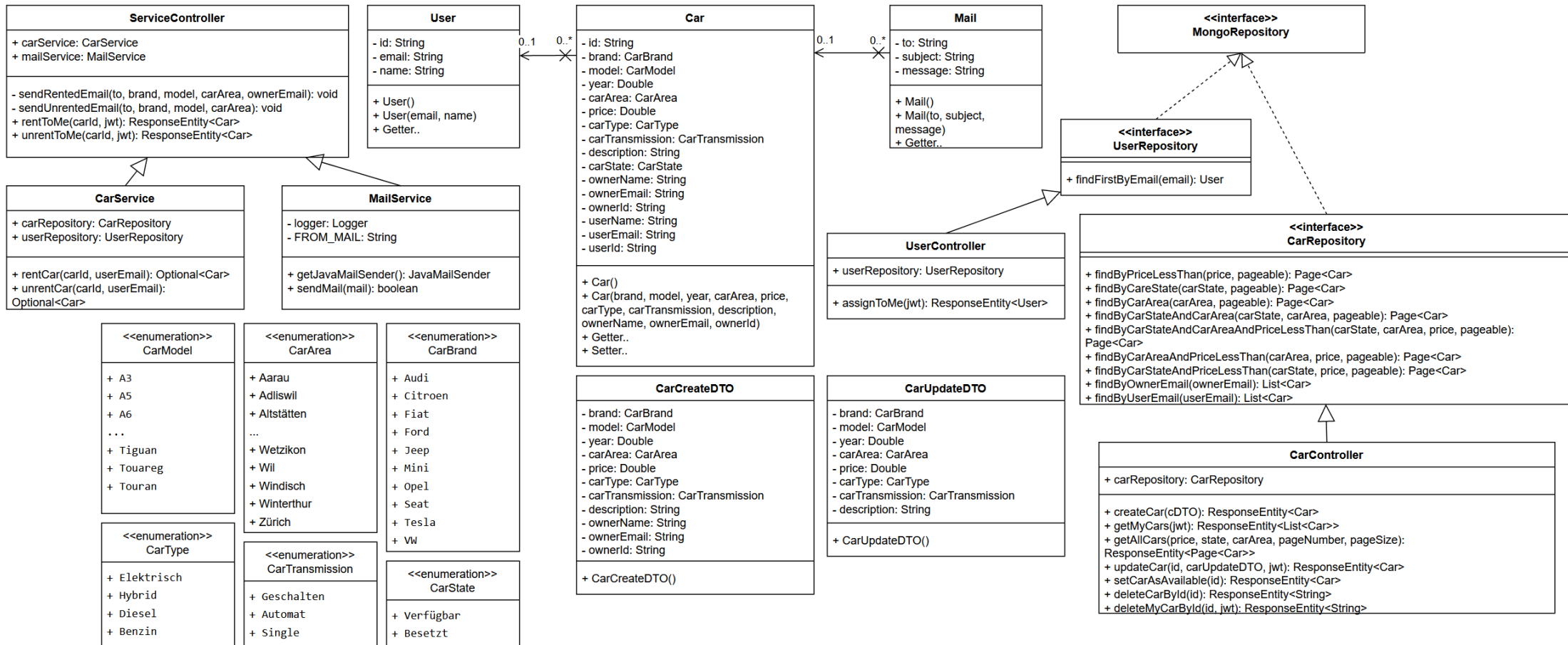
[Verfügbar setzen](#) [Löschen](#)



Mit einem Klick auf Benutzerverwaltung öffnet sich ein neuer Tab mit der Auth0-Oberfläche.



3.2 Klassendiagramm



In meinem Projekt habe ich 6 **Enumeration**-Klassen. Bei den Klassen **CarType**, **CarTransmission** und **CarState** gibt es nur die entsprechenden Möglichkeiten. Bei **CarBrand**, **CarModel** und **CarArea** habe ich mich für Enums entschieden, da ich im Frontend je nach gewähltem Modell und je nach gewähltem Ort ein Bild und eine entsprechende Map anzeige. Zudem habe ich zwei verschiedene **DTOs** (Data Transfer Objects). **CarCreateDTO** und **CarUpdateDTO** stellen dar, welche Attribute bei den entsprechenden Methoden aus dem **CarController** benötigt werden. Meine beiden Repositories **CarRepository** und **UserRepository** sind die beiden **DAOs** (Data Access Object). Sie legen fest, wie der Zugriff auf die entsprechenden Daten geregelt ist.

3.3 Drittsystem

Bei Peer2Vehicle erhält der Benutzer beim Mieten und Entmieten der Fahrzeuge eine E-Mail mit den entsprechenden Informationen. Ich nutze dazu als Drittsystem die Hilfsklasse **JavaMailSender** vom Spring Framework. In `MailService.java` werden anhand der Klasse `JavaMailSenderImpl` die Daten des Absenders festgelegt. Anhand `SimpleMailMessage` wird hier auch der Aufbau bzw. das Grundgerüst der E-Mail geregelt.

Im Folgenden wird der Prozess des Mailversands kurz erläutert:

Ein Benutzer mietet ein Fahrzeug. Dabei wird die Funktion `rentCar` im `CarService` aufgerufen. Bei erfolgreichem Mieten wird im `ServiceController` die Funktion `rentToMe` aufgerufen. Diese bezieht das entsprechende Fahrzeug mit den Informationen aus dem `CarService`. Es wird überprüft, ob das Fahrzeug vorhanden ist. Ist dies der Fall, dann wird ein entsprechendes Mail versendet.

Da ich für das Mieten und Entmieten zwei verschiedene Emails versende, habe ich im `ServiceController` die Funktionen `sendRentedEmail` und `sendUnrentedEmail`:

```
@PreAuthorize("isAuthenticated()")
private void sendRentedEmail(String to, CarBrand brand, CarModel model, CarArea carArea, String ownerEmail) {
    String subject = String.format("Miete von %s %s gestartet", brand, model);
    String message = String.format(
        "Lieber Kunde\n\nDas Fahrzeug %s %s aus %s wurde soeben für Dich gemietet.\nDu kannst den Besitzer /  
die Besitzerin nun unter %s kontaktieren.\n\nWir wünschen eine gute Fahrt!\n\nLiebe Grüße\nDein P2V-Team",
        brand, model, carArea, ownerEmail);
    Mail mail = new Mail(to, subject, message);
    mailService.sendMail(mail);
}

@PreAuthorize("isAuthenticated()")
private void sendUnrentedEmail(String to, CarBrand brand, CarModel model, CarArea carArea) {
    String subject = String.format("Miete von %s %s beendet", brand, model);
    String message = String.format(
        "Lieber Kunde\n\nDie Miete für das Fahrzeug %s %s aus %s wurde soeben beendet.\n\nWir freuen uns wie-  
der von Dir zu hören!\n\nLiebe Grüße\nDein P2V-Team",
        brand, model, carArea);
    Mail mail = new Mail(to, subject, message);
    mailService.sendMail(mail);
}
```

Beispiele:

Miete von Seat Ateca gestartet Posteingang x

peer2vehicle@outlook.com
an mich ▾

Lieber Kunde

Das Fahrzeug Seat Ateca aus Frauenfeld wurde soeben für Dich gemietet.
Du kannst den Besitzer / die Besitzerin nun unter thomas@mail.com kontaktieren.

Wir wünschen eine gute Fahrt!

Liebe Grüße
Dein P2V-Team

Miete von Audi A3 beendet Posteingang x

peer2vehicle@outlook.com
an mich ▾

Lieber Kunde

Die Miete für das Fahrzeug Audi A3 aus Winterthur wurde soeben beendet.

Wir freuen uns wieder von Dir zu hören!

Liebe Grüße
Dein P2V-Team

4 Testing

4.1 Modul- und Integrationstests

Für die Modul- und Integrationstests habe ich die drei Testklassen CarController.java, UserController.java und ServiceController.java erstellt.

4.1.1 CarController.java

Mit JUnit und MockMvc teste ich in dieser Klasse die Methoden des CarControllers.

```
@Test
@Order(1)
@WithMockUser
public void testCreateCar() throws Exception {
    // create a test car and convert to json
    Car car = new Car();
    car.setBrand(CarBrand.Audi);
    car.setModel(CarModel.A3);
    car.setYear(2020.04);
    car.setCarArea(CarArea.Winterthur);
    car.setPrice(120.50);
    car.setCarType(CarType.Diesel);
    car.setCarTransmission(CarTransmission.Automat);
    car.setDescription(TEST_STRING);
    car.setOwnerName("Mustermann");
    car.setOwnerEmail("muster@mail.com");
    car.setOwnerId("1");
    var jsonBody = ow.writeValueAsString(car);

    // POST json to service with authorization header
    mvc.perform(post("/api/car")
        .contentType(MediaType.APPLICATION_JSON)
        .content(jsonBody)
        .header(HttpHeaders.AUTHORIZATION, "Bearer token"))
        .andDo(print())
        .andExpect(status().isCreated())
        .andReturn();
}
```

Diese Methode testet das Anlegen eines neuen Autos. Es wird ein Car-Objekt mit den nötigen Daten erstellt und in ein JSON-Format umgewandelt. Dieses JSON wird dann mittels eines POST-Requests an die "/api/car"-API gesendet. Es wird erwartet, dass der Statuscode der Antwort 201 Created ist, was darauf hinweist, dass das Auto erfolgreich erstellt wurde.

```
@Test
@Order(2)
@WithMockUser
public void testGetAllCar() throws Exception {
    var json = getAllCars();

    // assert list of cars is not empty
    // and result contains test string
    assertFalse(json.isEmpty());
    assertTrue(json.contains(TEST_STRING));
}
```

Diese Methode testet das Abrufen aller Autos. Ein GET-Request wird an die "/api/car"-API gesendet und die Antwort wird in ein JSON-Format umgewandelt. Es wird überprüft, ob das JSON nicht leer ist (was darauf hindeutet, dass Autos abgerufen wurden) und ob es den TEST_STRING enthält (was darauf hindeutet, dass das im testCreateCar() erstellte Auto abgerufen wurde).

```
@Test
@Order(3)
@WithMockUser
public void testDeleteCars() throws Exception {
    // analyse json response and delete all test data cars
    var json = getAllCars();
    JsonNode jsonNode = mapper.readTree(json);
    var content = jsonNode.get("content");
    for (var x : content) {
        var id = x.get("id");
        var description = x.get("description");
        if (description.asText().equals(TEST_STRING)) {
            carRepository.deleteById(id.asText());
        }
    }

    // reload cars and assert no test data
    json = getAllCars();
    System.out.println(json);
    assertFalse(json.contains("\n" + TEST_STRING + "\n"));
}
```

Diese Methode testet das Löschen von Autos. Zuerst wird ein GET-Request an die "/api/car"-API gesendet, um alle Autos abzurufen. Dann werden alle Autos durchlaufen und jedes Auto, dessen Beschreibung gleich dem TEST_STRING ist, wird gelöscht. Schließlich wird erneut ein GET-Request an die "/api/car"-API gesendet, um alle Autos abzurufen, und es wird überprüft, ob das JSON den TEST_STRING nicht mehr enthält (was darauf hindeutet, dass das Auto erfolgreich gelöscht wurde).

```
private String getAllCars() throws Exception {
    var result = mvc.perform(get("/api/car")
        .param("pageSize", String.valueOf(Integer.MAX_VALUE))
        .contentType(MediaType.TEXT_PLAIN))
        .andDo(print())
        .andExpect(status().isOk())
        .andReturn();
    return result.getResponse().getContentAsString();
}
```

Diese Methode ist eine Hilfsmethode, die dazu dient, alle Autos abzurufen. Sie sendet einen GET-Request an die "/api/car"-API und gibt die Antwort als JSON-Zeichenkette zurück. Der Parameter pageSize wird auf den maximal möglichen Integer-Wert gesetzt, um sicherzustellen, dass alle Autos abgerufen werden. Es wird erwartet, dass der Statuscode der Antwort 200 OK ist, was darauf hinweist, dass die Anfrage erfolgreich war.

4.1.2 UserController.java

Ebenfalls mit JUnit und MockMvc teste ich in den UserController.

```
@Test
@Order(1)
@WithMockUser
public void testDeleteUser() throws Exception {
    User result = userRepository.findFirstByEmail(TEST_EMAIL);
    if (result != null) {
        userRepository.deleteById(result.getId());
    }

    result = userRepository.findFirstByEmail(TEST_EMAIL);
    assertNull(result);
}
```

Die Methode testDeleteUser() überprüft die Löschfunktionalität von Benutzern. Zunächst wird ein Benutzer mit einer bestimmten Test-E-Mail-Adresse gesucht und gelöscht. Danach wird erneut versucht, denselben Benutzer zu finden. Der Test ist erfolgreich, wenn kein Benutzer gefunden wird, was darauf hinweist, dass der Löschvorgang wie erwartet funktioniert hat.

4.1.3 ServiceController.java

Mit JUnit und Mockito teste ich hier die Methoden vom ServiceController.

```
@Test
public void rentToMe_Success() {
    // Arrange
    String ownerEmail = "test@test.com";
    String carId = "carId1";

    Car car = new Car();
    car.setId(carId);
    car.setBrand(CarBrand.Audi);
    car.setModel(CarModel.A3);
    car.setYear(2020.04);
    car.setCarArea(CarArea.Winterthur);
    car.setPrice(120.50);
    car.setCarType(CarType.Diesel);
    car.setCarTransmission(CarTransmission.Automat);
    car.setDescription("Test Beschreibung");
    car.setOwnerName("Mustermann");
    car.setOwnerEmail(ownerEmail);
    car.setOwnerId("1");

    when(jwt.getClaimAsString("email")).thenReturn(ownerEmail);
    when(carService.rentCar(anyString(), anyString()))
        .thenReturn(Optional.of(car));

    // Act
    ResponseEntity<Car> response = serviceController.rentToMe(carId, jwt);

    // Assert
    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(car, response.getBody());
}
```

Dieser Test überprüft, ob ein Benutzer erfolgreich ein Auto mieten kann. Es wird ein simulierter Benutzer erstellt, für den dann ein Auto zur Miete verfügbar gemacht wird. Der Test schaut, ob die Antwort des Servers korrekt ist, indem er prüft, ob der Server-Statuscode "OK" ist und das gemietete Auto in der Antwort des Servers korrekt zurückgegeben wird.


```

@Test
public void rentToMe_Failure() {
    // Arrange
    String userEmail = "test@test.com";
    String carId = "carId1";
    when(jwt.getClaimAsString("email")).thenReturn(userEmail);
    when(carService.rentCar(anyString(), anyString()))
        .thenReturn(Optional.empty());

    // Act
    ResponseEntity<Car> response = serviceController
        .rentToMe(carId, jwt);

    // Assert
    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
}

```

Diese Methode testet den Fehlerpfad der rentToMe Funktion im ServiceController. Wenn kein Auto für den gemockten Nutzer "test@test.com" zum Mieten verfügbar ist, wird geprüft, ob der Statuscode der Antwort 400 (BAD REQUEST) ist.

```

@Test
public void unrentToMe_Success() {
    // Arrange
    String ownerEmail = "test@test.com";
    String carId = "carId1";

    Car car = new Car();
    car.setId(carId);
    car.setBrand(CarBrand.Audi);
    car.setModel(CarModel.A3);
    car.setYear(2020.04);
    car.setCarArea(CarArea.Winterthur);
    car.setPrice(120.50);
    car.setCarType(CarType.Diesel);
    car.setCarTransmission(CarTransmission.Automat);
    car.setDescription("Test Beschreibung");
    car.setOwnerName("Mustermann");
    car.setOwnerEmail(ownerEmail);
    car.setOwnerId("1");

    when(jwt.getClaimAsString("email")).thenReturn(ownerEmail);
    when(carService.unrentCar(anyString(), anyString()))
        .thenReturn(Optional.of(car));

    // Act
    ResponseEntity<Car> response = serviceController.unrentToMe(carId, jwt);

    // Assert
    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(car, response.getBody());
}

```

Dieser Test überprüft das erfolgreiche Zurückgeben eines gemieteten Autos. Ein simulierter Benutzer gibt ein gemietetes Auto zurück. Der Test prüft, ob der Server korrekt auf diese Aktion reagiert, indem er den Statuscode "OK" zurückgibt und das zurückgegebene Auto korrekt in der Serverantwort widerspiegelt.

```

@Test
public void unrentToMe_Failure() {
    // Arrange
    String userEmail = "test@test.com";
    String carId = "carId1";
    when(jwt.getClaimAsString("email")).thenReturn(userEmail);
    when(carService.unrentCar(anyString(), anyString()))
        .thenReturn(Optional.empty());

    // Act
    ResponseEntity<Car> response = serviceController
        .unrentToMe(carId, jwt);

    // Assert
    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
}

```

Dieser Test überprüft das Szenario, in dem ein Benutzer versucht, ein Auto zurückzugeben, das er nicht gemietet hat. Der Test stellt sicher, dass der Server in diesem Fall einen Fehler-Statuscode "BAD REQUEST" zurückgibt.

4.2 User-Tests

Mein System lasse ich durch 4 Personen aus meinem Umfeld testen, welche alle aus einem Nicht-IT-Bereich kommen und potenzielle Nutzer von Peer2Vehicle sein könnten. Dabei handelt es sich um meine Eltern, meine Schwester und meine Freundin. Pro Person werden 2 User-Tests durchgeführt. Ich beschreibe den Testpersonen ihre jeweiligen Aufgaben, beobachte sie bei der Durchführung und mache mir währenddessen Notizen in den folgenden Tabellen.

User-Test Nr.	1.1
Testperson	Vater
Aufgabe:	Die Testperson soll von der Startseite aus zum Login-Fenster navigieren und sich mit einem neuen Konto auf der Plattform registrieren.
Ergebnisse:	Die Testperson hat das Icon für den Login oben rechts schnell ausfindig gemacht und angeklickt. Im Login-Fenster von Auth0 hatte die Testperson Schwierigkeiten mit den Begriffen «Sing up» und «Sign in». Erst nach einer gewissen Zeit, konnte sich die Testperson über «Sign up» schliesslich ein Konto erstellen.
Massnahmen:	Auf Auth0 habe ich die Standardsprache auf Deutsch umgestellt, sodass klar wird was «Registrieren» und «Anmelden» ist.

User-Test Nr.	1.2
Testperson	Vater
Aufgabe:	Die Testperson soll sich vom angemeldeten Konto erfolgreich abmelden.
Ergebnisse:	Da sich nach dem Anmelden das Login Icon zu einem «Off-Button» ändert konnte sich die Testperson ohne Mühe vom System abmelden.
Massnahmen:	Keine Massnahmen erforderlich.

User-Test Nr.	2.1
Testperson	Mutter
Aufgabe:	Die Testperson soll von der Startseite aus zur Ansicht aller Fahrzeuge navigieren und die Fahrzeug-Liste nach Ort, Preis und Verfügbarkeit filtern.
Ergebnisse:	Die Testperson konnte problemlos die Liste der Fahrzeuge ausfindig machen, da die Navigationsliste gross genug ist und die Beschriftungen eindeutig gewählt sind. Beim Filtern war die Auswahl für Ort und Verfügbarkeit klar. Beim Preis gab es Verwirrung, da die Testperson davon ausging, dass der Preisfilter alle Fahrzeuge mit exakt dem gewählten Mietpreis anzeigt. Jedoch zeigt der Filter alle Fahrzeuge an die günstiger, als der gewählte Preis sind.
Massnahmen:	Im Frontend habe ich bei der Filterschaltfläche die Beschriftung «Preis» auf «Preis bis» geändert.

User-Test Nr.	2.2
Testperson	Mutter
Aufgabe:	Die Testperson soll die Miete für ein gewünschtes Fahrzeug starten.
Ergebnisse:	Da in der Tabelle bei den verfügbaren Fahrzeugen rechts der Button «Miete starten» steht, konnte die Testperson die Aufgabe erfolgreich durchführen.
Massnahmen:	Keine Massnahmen erforderlich.

User-Test Nr.	3.1
Testperson	Schwester
Aufgabe:	Die Testperson soll ein eigenes Fahrzeug anbieten.
Ergebnisse:	Über die Navigationsleiste ist die Testperson direkt zur Oberfläche «Vermieten» gewechselt. Das Erstellen des Fahrzeuges hat ohne weiteres funktioniert, jedoch ist der Testperson aufgefallen, dass man Modelle auswählen kann, welche nicht zur gewählten Marke passen.
Massnahmen:	Im Frontend habe ich bei der Erstellung eines Fahrzeuges die Dropdown-Listen für das Modell so angepasst, dass diese erst wählbar ist, nachdem eine Marke gewählt wurde. Die Modelle, welche dann in der Liste angezeigt werden, sind jene welche der gewählten Marke entsprechen. Es kommen nun also bei einem VW nur die Modelle Golf, ID3, ID4 etc.

User-Test Nr.	3.2
Testperson	Schwester
Aufgabe:	Die Testperson soll die Angaben zum eigenen Fahrzeug anpassen.
Ergebnisse:	Die Testperson hat schnell gemerkt, dass sie zur Oberfläche «Meine Übersicht» wechseln muss. Dort konnte sie das zuvor erstellte Fahrzeug sehen. Zunächst wusste die Testperson nicht wie man das Fahrzeug bearbeiten kann. Erst als die Person herausgefunden hat, dass man mit einem Klick auf das Fahrzeug zur Detailansicht kommt, konnte sie die Fahrzeuginformationen erfolgreich anpassen.
Massnahmen:	Ich habe mir überlegt, in der Liste einen Button mit einem Info-Icon zu erstellen, damit es klarer wird, wo man klicken muss, um auf die Detailansicht zu kommen. Jedoch meinte die Testperson nach einer Diskussion, dass dieser Button die Oberfläche verunschönert und es genug klar ist, dass man auf das Bild des Fahrzeuges klicken muss.

User-Test Nr.	4.1
Testperson	Freundin
Aufgabe:	Die Testperson soll das eigene Fahrzeug nicht mehr zum Mieten anbieten.
Ergebnisse:	Es war für die Testperson sofort klar, dass man für diese Aufgabe zur Oberfläche «Meine Übersicht» wechseln muss. Sie konnte also ohne weiteres zur geforderten Oberfläche wechseln und das entsprechende Fahrzeug mit dem Löschenbutton löschen.
Massnahmen:	Keine Massnahmen erforderlich.

User-Test Nr.	4.1
Testperson	Freundin
Aufgabe:	Die Testperson soll ein gemietetes Fahrzeug entmieten.
Ergebnisse:	Auch hier war es gleich klar, dass die Testperson zur Oberfläche «Meine Übersicht» wechseln muss. Auf der Übersicht konnte sie mit dem Button «Miete beenden» das Fahrzeug erfolgreich entmieten.
Massnahmen:	Keine Massnahmen erforderlich.

5 Fazit

Aktuell ist der Prototyp von Peer2Vehicle auf dem Stand, dass man Fahrzeuge ganz einfach mieten und entmieten kann. Auch das Erstellen der Fahrzeuge funktioniert. Für eine reale Implementierung der Software müssten noch drei wichtige Punkte ergänzt werden:

1. Bilder:
Beim Erstellen der Fahrzeuge sollte ein Upload von eigenen Bildern möglich sein. Ich habe dazu ein wenig recherchiert und konnte herausfinden, dass dies in MongoDB mit GridFS möglich ist. Im Rahmen dieser Arbeit habe ich mich aber dazu entschieden dies wegzulassen, da die Implementation meinen zeitlichen Rahmen übertreten würde.
2. Entmieten:
Im Moment ist der Prototyp so aufgebaut, dass Nutzer ein Fahrzeug mieten und, nachdem sie es wieder abgeben, auf der Plattform wieder entmieten. Für den Administrator habe ich eine Funktion programmiert, welche es ermöglicht ein besetztes Fahrzeug wieder auf «Verfügbar» zu setzen, für den Fall, dass ein Mieter das Entmieten auf der Plattform vergisst. Dies ist in der realen Welt etwas umständlich. Ich würde hierzu einen Timer einbauen, sodass man beim Mieten wählen kann wie viel Tage man das Auto benötigt. Nach Ablauf dieser Zeit wird das Fahrzeug dann automatisch wieder als «Verfügbar» angezeigt.
3. Rechtliches:
Um eine Carsharing-Plattform in dieser Form Publik zu machen, gibt es einige rechtliche Aspekte, die vorher geklärt werden müssen. Vor allem die Versicherung und Haftung bei Unfällen müsste klar definiert sein.

Das Projekt empfand ich persönlich als sehr lehrreich und spannend. Es gab viele Dinge, die man selbstständig mit Internet-Recherchen herausfinden musste, was das Ganze noch interessanter gestaltet hat. Die allgemeine Selbstständigkeit bezüglich Auswahl der Idee und Projektaufbau hat mir besonders gut gefallen.