

# The Vintage Garage Community

## Software-Entwurf

Moumani Toko Sharon Lucresse  
Dtzameni Wepadjui Pradelle  
Fokam Fodouop Rodrigue  
Kuidja Nguedong

5. Januar 2025

# Inhaltsverzeichnis

<b>1 Operationelle Verträge</b>	<b>3</b>
1.1 Benutzer . . . . .	3
1.1.1 Usecase 1: "Sich registrieren" . . . . .	3
1.1.2 Usecase 2: "Sich anmelden" . . . . .	4
1.2 Lagerhalter . . . . .	6
1.2.1 Usecase 1: "Lager einfügen" . . . . .	6
1.2.2 Usecase 2: "Lagerinformationen aktualisieren" . . . . .	8
1.2.3 Usecase 3: "Services anbieten" . . . . .	9
1.2.4 Usecase 4: "Verfügbare Plätze anzeigen" . . . . .	10
1.2.5 Usecase 5: "Fahrzeugspezialisierung angeben" . . . . .	11
1.2.6 Usecase 6: "Fahrzeuginformationen bereitstellen" . . . . .	12
1.2.7 Usecase 7: "Reparaturanfragen stellen" . . . . .	13
1.2.8 Usecase 8: "Reparaturtermine vereinbaren" . . . . .	14
1.3 Einlagerer . . . . .	16
1.3.1 Usecase 1: "Nach Stellplätzen suchen" . . . . .	16
1.3.2 Usecase 2: "Anfragen an Lagerhalter stellen" . . . . .	17
1.3.3 Usecase 3: "Angebot annehmen oder ablehnen" . . . . .	18
1.3.4 Usecase 4: "sich Ersatzteile anzeigen lassen" . . . . .	19
1.3.5 Usecase 5: "Abhol- und Rückgabetermin vereinbaren" . . . . .	20
1.3.6 Usecase 6: "Zusätzliche Services buchen" . . . . .	21
1.3.7 Usecase 7: "Ersatzteilreservierungen durchführen" . . . . .	22
1.4 Werkstattinhaber . . . . .	23
1.4.1 Usecase 1: "Werkstatt einfügen" . . . . .	23
1.4.2 Usecase 2: "Reparaturanfragen empfangen" . . . . .	24
1.5 Ersatzteilanbieter . . . . .	25
1.5.1 Usecase 1: "Ersatzteile anbieten" . . . . .	25
1.5.2 Usecase 2: "Ersatzteile löschen" . . . . .	26
<b>2 Interaktionsdiagramme</b>	<b>27</b>
2.1 Benutzer . . . . .	27
2.1.1 Usecase 1: "Sich registrieren" . . . . .	27
2.1.2 Usecase 2: "Sich anmelden" . . . . .	27
2.2 Lagerhalter . . . . .	28
2.2.1 Usecase 1: "Lager einfügen" . . . . .	28
2.2.2 Usecase 2: "Lagerinformationen aktualisieren" . . . . .	28
2.2.3 Usecase 3: "Services anbieten" . . . . .	29
2.2.4 Usecase 4: "Verfügbare Plätze anzeigen" . . . . .	29
2.2.5 Usecase 5: "Fahrzeugspezialisierung angeben" . . . . .	30
2.2.6 Usecase 6: "Fahrzeuginformationen bereitstellen" . . . . .	30
2.2.7 Usecase 7: "Reparaturanfragen stellen" . . . . .	31
2.2.8 Usecase 8: "Reparaturtermine vereinbaren" . . . . .	31
2.3 Einlagerer . . . . .	32
2.3.1 Usecase 1: "Nach Stellplätzen suchen" . . . . .	32

2.3.2	Usecase 2: "Anfragen an Lagerhalter stellen" . . . . .	32
2.3.3	Usecase 3: "Angebot annehmen oder ablehnen" . . . . .	33
2.3.4	Usecase 4: "sich Ersatzteile anzeigen lassen" . . . . .	33
2.3.5	Usecase 5: "Abhol- und Rückgabetermin vereinbaren" . . . . .	34
2.3.6	Usecase 6: "Zusätzliche Services buchen" . . . . .	34
2.3.7	Usecase 7: "Ersatzteilreservierungen durchführen" . . . . .	35
2.4	Werkstattinhaber . . . . .	36
2.4.1	Usecase 1: "Werkstatt einfügen" . . . . .	36
2.4.2	Usecase 2: "Reparaturanfragen empfangen" . . . . .	36
2.5	Ersatzteilanbieter . . . . .	37
2.5.1	Usecase 1: "Ersatzteile anbieten" . . . . .	37
2.5.2	Usecase 2: "Ersatzteile löschen" . . . . .	37
<b>3</b>	<b>Entwurfsklassendiagramm</b>	<b>38</b>
<b>4</b>	<b>Skizze der GUI</b>	<b>39</b>
4.1	Registrierung- und Loginseite . . . . .	39
4.2	Lagerhalter-Funktionalitäten . . . . .	40
4.3	Einlagerer-Funktionalitäten . . . . .	41
4.4	Werkstattinhaber-Funktionalitäten . . . . .	42
<b>5</b>	<b>Annahme und bewusste Abweichungen von der OOD-Methodik</b>	<b>43</b>
5.1	Annahmen . . . . .	43
5.1.1	Technische Einschränkungen . . . . .	43
5.1.2	Funktionale Annahmen . . . . .	43
5.1.3	Architekturentscheidungen . . . . .	43
5.2	Bewusste Abweichungen von der OOD-Methodik . . . . .	44
5.2.1	Reduktion der Komplexität . . . . .	44
5.2.2	Keine vollständige Implementierung aller Patterns . . . . .	44
5.2.3	Zusammenführung von Konzepten . . . . .	44
5.2.4	Abweichung bei der Fehlerbehandlung . . . . .	44

# Kapitel 1

## Operationelle Verträge

### 1.1 Benutzer(Alle Akteure im System)

#### 1.1.1 Usecase 1: “Sich registrieren”

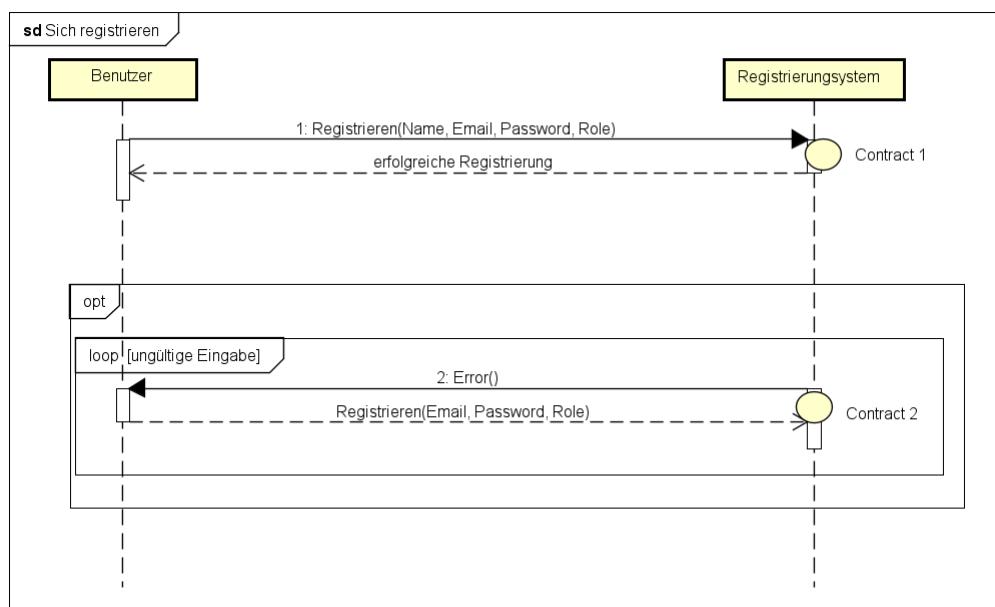


Abbildung 1.1: “Sich registrieren”.

#### Contract 1: sich registrieren

**Operation:** `registrieren(name: String, email: String, password:String, userrolle: enum)`

**Querverweis:** Use Cases “Sich registrieren”

**Vorbedingungen:** Ein neuer Benutzer möchte sich registrieren.

**Nachbedingungen:**

- Eine Instanz von **Benutzer** wurde erzeugt.  
(Erzeugung einer Instanz)
- Die Attribute der **Benutzer**-Instanz (Name, E-Mail, Passwort, userrolle) wurden mit den bereitgestellten Werten belegt.  
(Setzen von Attributwerten)

- Die Benutzer-Instanz wurde in das System (Datenbank) persistiert.  
(Speicherung der Instanz)
- Dem Benutzer wurde eine eindeutige Benutzer-ID (`userID`) zugewiesen.  
(Erzeugung eines eindeutigen Identifikators)
- Der Benutzer hat die Standardrolle basierend auf dem Wert von `userrolle` erhalten.  
(Zuweisung einer Benutzerrolle)
- Ein Bestätigungsereignis wurde ausgelöst, um den Benutzer per E-Mail über die erfolgreiche Registrierung zu informieren.  
(Auslösen eines Events)

**Contract 2:** falsche Registrierung

**Operation:** `error()`

**Querverweis:** Use Cases “Sich registrieren”

**Vorbedingungen:** Ein neuer Benutzer möchte sich registrieren.

**Nachbedingungen:**

- Keine Instanz von Benutzer wurde erzeugt.  
(Keine Erzeugung einer Instanz aufgrund unvollständiger Daten)
- Ein Fehler wurde ausgelöst und an den Benutzer zurückgegeben, der genau angibt, welche Daten fehlen oder ungültig sind.  
(Rückmeldung an den Benutzer)
- Der Benutzer wurde aufgefordert, die fehlenden oder ungültigen Daten erneut einzugeben.  
(Erneute Dateneingabe gefordert)
- Der Registrierungsprozess bleibt offen, bis gültige und vollständige Daten eingegeben werden.  
(Offener Prozess bis zum Abschluss)

### 1.1.2 Usecase 2: “Sich anmelden”

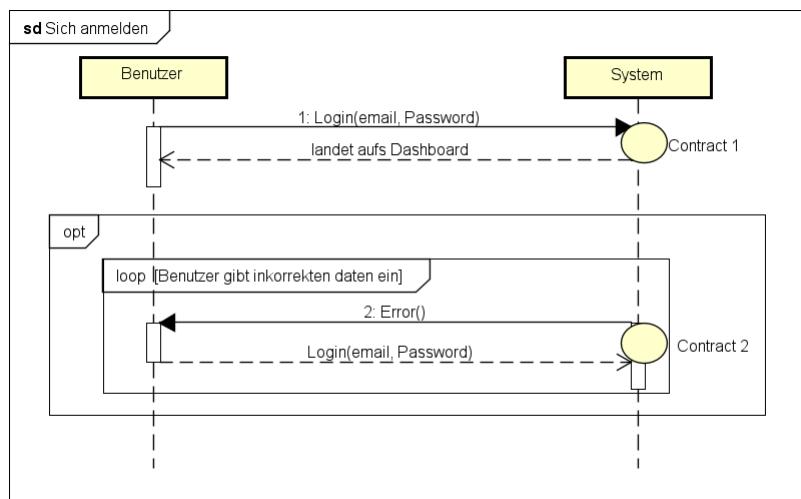


Abbildung 1.2: “Sich anmelden”.

**Contract 1:** sich anmelden

**Operation:** login(email: String, password:String)

**Querverweis:** Use Cases “Sich anmelden”

**Vorbedingungen:**

- Der Benutzer möchte sich anmelden.
- Eine gültige Kombination aus E-Mail und Passwort ist im System registriert.

**Nachbedingungen:**

- Die Authentifizierung des Benutzers wurde erfolgreich abgeschlossen.  
(Überprüfung der Anmeldedaten)
- Eine bestehende Instanz von **Benutzer** wurde im System gefunden.  
(Zugriff auf die Benutzerinstanz)
- Eine Benutzersitzung (**session**) wurde erstellt und mit der **Benutzer**-Instanz verknüpft.  
(Erstellung einer Sitzung)
- Der Benutzer wurde auf sein persönliches Dashboard weitergeleitet.  
(Weiterleitung nach der Anmeldung)

**Contract 2:** falsche Anmeldung

**Operation:** error()

**Querverweis:** Use Cases “Sich anmelden”

**Vorbedingungen:**

- Ein Benutzer möchte sich anmelden.
- Die Eingabedaten (E-Mail und Passwort) stimmen nicht mit den im System gespeicherten Anmeldedaten überein.

**Nachbedingungen:**

- Keine Benutzersitzung wurde erstellt.  
(Keine Authentifizierung und Sitzungserstellung)
- Der Benutzer wurde informiert, dass die eingegebenen Daten ungültig oder nicht korrekt sind.  
(Rückmeldung an den Benutzer)
- Der Benutzer hat die Möglichkeit, die Daten zu korrigieren und erneut einzugeben.  
(Erneute Dateneingabe gefordert)

## 1.2 Lagerhalter

### 1.2.1 Usecase 1: “Lager einfügen”

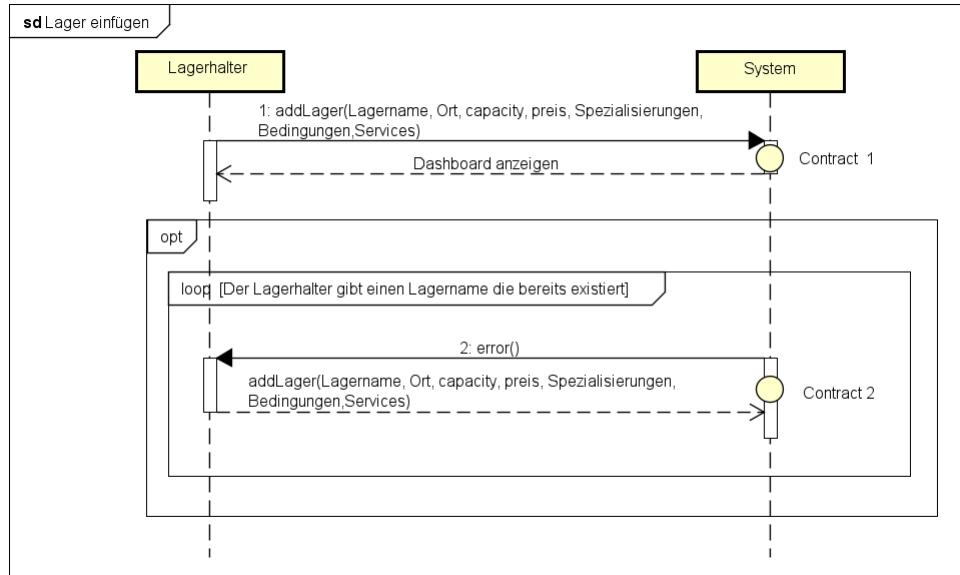


Abbildung 1.3: “Lager einfügen”.

**Contract 1:** Lager einfügen

**Operation:** `addLager(Lagernname: string, Ort: enum, capacity: int, preis: double, Spezialisierungen: string[], Bedingungen: string[], Services: Service[])`

**Querverweis:** Use Cases “Lager einfügen”

**Vorbedingungen:**

- Der Benutzer ist authentifiziert und hat die Rolle eines **Lagerhalters**.  
(Benutzer muss die Rolle eines Lagerhalters besitzen)
- Die Eingabedaten (Lagernname, Ort, Kapazität, Preis, Spezialisierungen, Bedingungen, Services) sind vollständig und gültig.  
(Überprüfung der Eingabedaten)

**Nachbedingungen:**

- Eine Instanz von **Lager** wurde im System erstellt.  
(Erzeugung einer Instanz von Lager)
- Instanzen von **Service** wurden im System erstellt und mit dem Lager geknüft. je nachdem welche service eingegeben wurden.  
(Erzeugung der Instanze von Service)
- Instanzen von **Stellplatz** wurden im System erstellt und mit dem Lager geknüft.  
(Erzeugung der Instanze von Stellplatz)
- Die Attribute der **Lager**-Instanz (Name, Ort, Kapazität, Preis, Spezialisierungen, Bedingungen, Services) wurden mit den bereitgestellten Werten belegt.  
(Setzen von Attributwerten)
- Die **Lager**-Instanz wurde in das System (Datenbank) persistiert.  
(Speicherung der Instanz)

- Dem Lager wurde eine eindeutige ID zugewiesen.  
(Erzeugung eines eindeutigen Identifikators für das Lager)
- Eine Bestätigung des erfolgreichen Hinzufügens des Lagers wurde an den Benutzer zurückgegeben.  
(Bestätigung der Aktion)

**Contract 2:** Lager existiert schon

**Operation:** `error()`

**Querverweis:** Use Cases “Lager einfügen”

**Vorbedingungen:**

- Ein Benutzer möchte ein Lager hinzufügen.
- Das angegebene Lager (mit dem gleichen Namen und im gleichen Ort Attributen) existiert bereits im System.

**Nachbedingungen:**

- Keine Instanz von **Lager** wurde erstellt.  
(Keine Erzeugung einer Instanz aufgrund des Vorhandenseins eines gleichen Lagers)
- Ein Fehler wurde ausgelöst und an den Benutzer zurückgegeben, der darauf hinweist, dass das Lager bereits existiert.  
(Rückmeldung an den Benutzer)
- Der Benutzer wurde aufgefordert, einen anderen Namen oder Ort für das Lager einzugeben.  
(Erneute Dateneingabe gefordert)
- Der Prozess des Lagerhinzufügens bleibt offen, bis gültige und nicht doppelte Daten eingegeben werden.  
(Offener Prozess bis zum Abschluss)

## 1.2.2 Usecase 2: “Lagerinformationen aktualisieren”

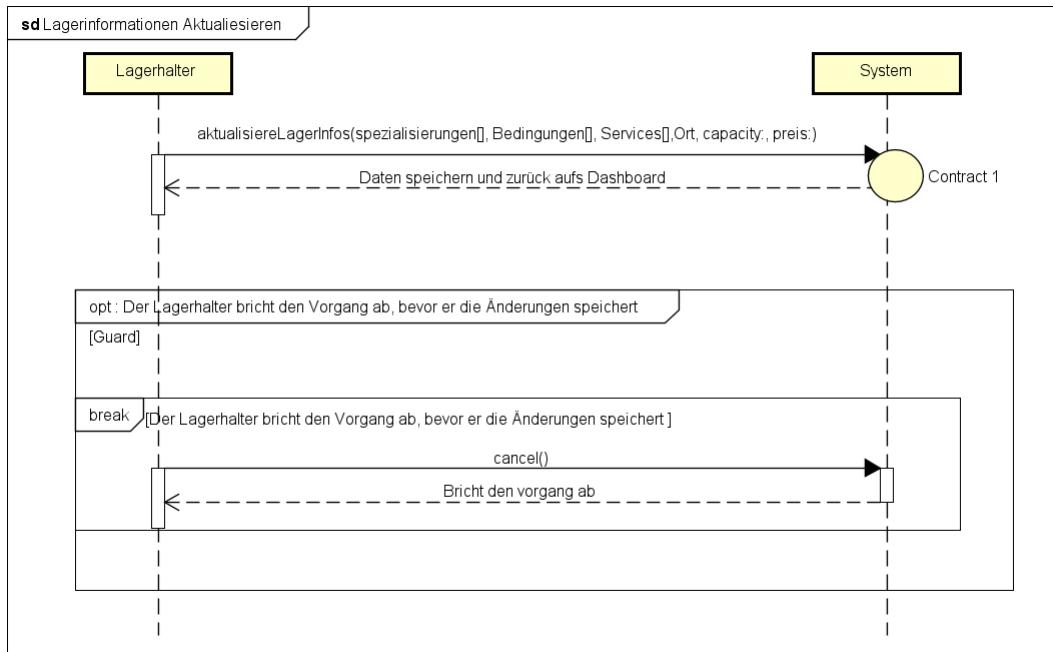


Abbildung 1.4: “Lagerinformationen aktualisieren”.

**Contract 1:** Lagerinformationen aktualisieren

**Operation:** aktualisiereLagerInfos(Ort: enum, capacity: int, preis: double, Spezialisierungen string[], Bedingungen: string[], Services: Service[])

**Querverweis:** Use Cases “Lagerinformationen aktualisieren”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Der Lagerhalter hat mindestens einen Lagerplatz im System erfasst.

**Nachbedingungen:**

- Die Informationen zu einem bestehenden Lager wurden aktualisiert (Veränderung eines Attributwertes eines Objekts)
- Die Informationen zu den verfügbaren Lagerplätzen und deren Kategorien wurden angepasst.
- Neue Spezialisierungen zu Fahrzeugmarken wurden mit dem Lager assoziiert.

### 1.2.3 Usecase 3: “Services anbieten”

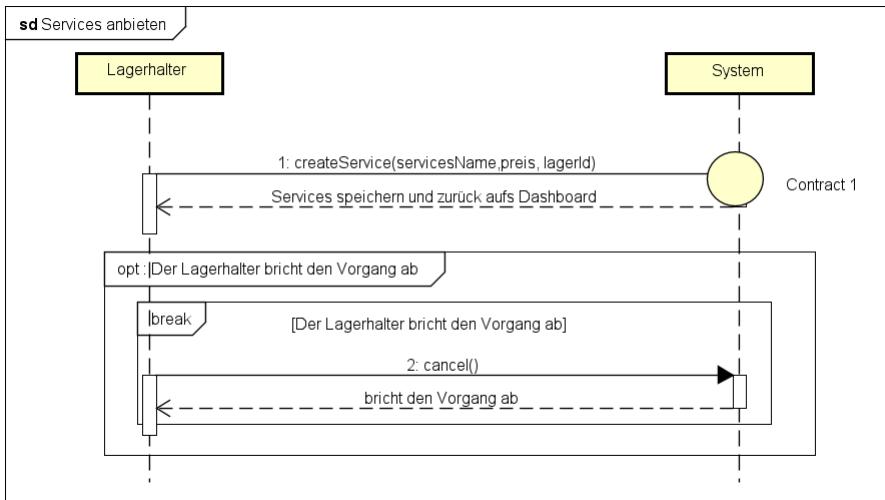


Abbildung 1.5: “Services anbieten”.

#### Contract 1: Services anbieten

**Operation:** `createServices(serviceName: String, preis: Float, lagerId: Integer)`

**Querverweis:** Use Cases “Services anbieten”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert..
- Der Lagerhalter hat mindestens einen Lagerplatz im System erfasst.

**Nachbedingungen:**

- Eine neue Instanz von Service wurde erzeugt.
- Die neue Service-Instanz wurde mit dem entsprechenden Lagerhalter assoziiert.
- Die Attribute serviceName und preis der Service-Instanz wurden mit den übergebenen Werten belegt.

#### 1.2.4 Usecase 4: “Verfügbare Plätze anzeigen”



Abbildung 1.6: “Verfügbare Plätze anzeigen”.

**Contract 1:** Verfügbare Plätze anzeigen

**Operation:** `getAvailablePlaces(lagerId: Integer)`

**Querverweis:** Use Cases “Verfügbare Plätze anzeigen”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Der Lagerhalter hat mindestens einen Lagerplatz im System erfasst.
- Die Datenbank enthält Informationen über alle verfügbaren Stellplätze.

**Nachbedingungen:**

- Eine Liste von Stellplatz-Instanzen wurde aus der Datenbank abgerufen.
- Jeder Stellplatz in der Liste wurde mit einem Attribut `status` versehen, das den Wert frei oder besetzt hat.
- Die freien Stellplätze werden visuell anders hervorgehoben, sodass der Lagerhalter sie leicht erkennen kann.

### 1.2.5 Usecase 5: “ Fahrzeugspezialisierung angeben”

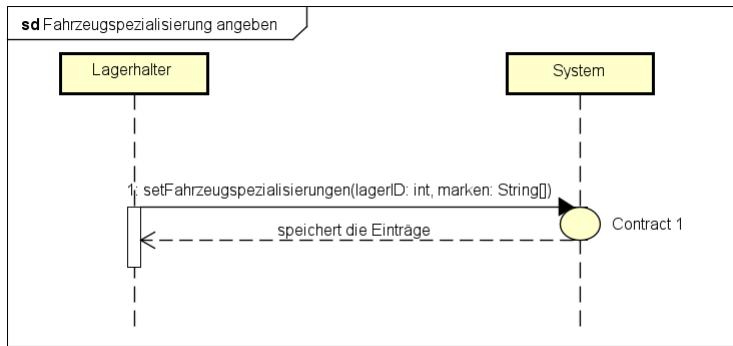


Abbildung 1.7: “ Fahrzeugspezialisierung angeben”.

**Contract 1:** Fahrzeugspezialisierung angeben

**Operation:** `setFahrzeugspezialisierungen(lagerId: Integer, marken: String[])`

**Querverweis:** Use Cases “Fahrzeugspezialisierung angeben”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Der Lagerhalter hat Zugriff auf sein Profil und kann Änderungen daran vornehmen.
- Das System enthält eine Liste gängiger Fahrzeugmarken.

**Nachbedingungen:**

- Eine Instanz von Lagerhalter wurde mit einer aktualisierten Fahrzeugspezialisierung versehen.
- Die angegebenen Fahrzeugmarken werden im Lagerhalter-Profil gespeichert und sind für andere Benutzer sichtbar.

### 1.2.6 Usecase 6: “ Fahrzeuginformationen bereitstellen”

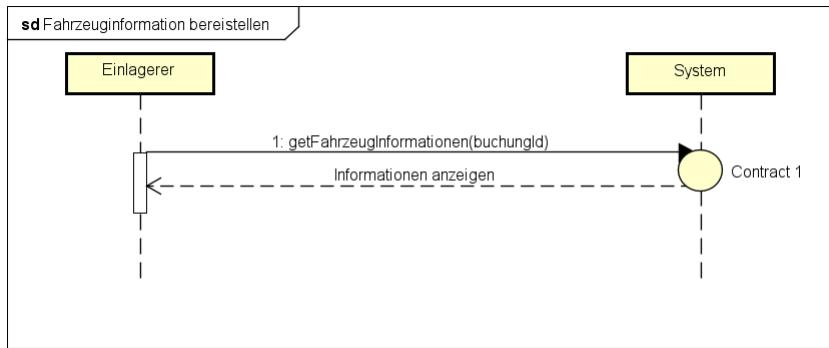


Abbildung 1.8: “ Fahrzeuginformationen bereitstellen”.

**Contract 1:** Fahrzeuginformationen bereitstellen

**Operation:** `getFahrzeugInformationen(buchungId: Integer)`

**Querverweis:** Use Cases “Fahrzeuginformationen bereitstellen”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Ein Stellplatz in einem Lager des Lagerhalters existiert und ist gebucht.
- Es gibt eine gültige BuchungID für den Stellplatz.
- Der Einlagerer hat Zugriff auf die Fahrzeugdetails seiner Buchung (z.B. Zustand, Wartungsstand, Fahrbereitschaft).

**Nachbedingungen:**

- Die Fahrzeuginformationen werden aktualisiert und mit der BuchungId verknüpft.
- Die Informationen sind für berechtigte Einlagerer einsehbar.

### 1.2.7 Usecase 7: “Reparaturanfragen stellen”

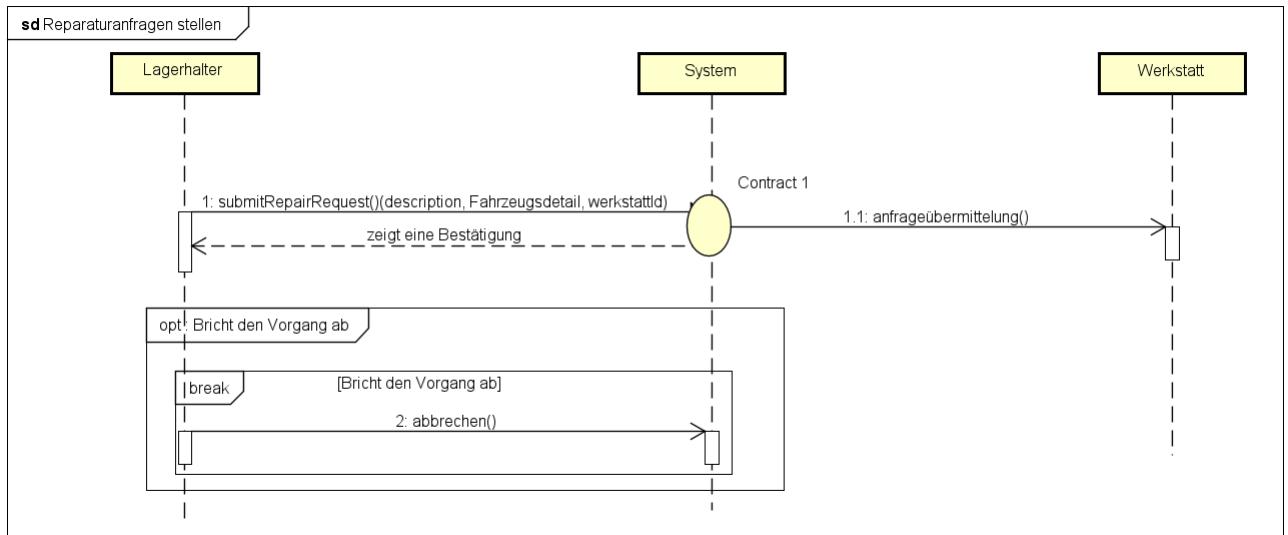


Abbildung 1.9: “Reparaturanfragen stellen”.

**Contract 1:** Reparaturanfragen stellen

**Operation:** submitRepairRequest(description: String, Fahrzeugsdetail: string[], werkstattId: Integer)

**Querverweis:** Use Cases “Reparaturanfragen stellen”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Das Fahrzeug wurde registriert (bei der Stellplatzbuchung) und die Fahrzeuginformationen sind verfügbar.
- Eine Werkstatt wurde registriert und ist bereit, Anfragen zu empfangen.

**Nachbedingungen:**

- Eine Instanz von Reparaturanfrage wird erstellt.
- Die Reparaturanfrage wird mit dem entsprechenden FahrzeugsDetails und der Werkstatt verknüpft
- Die Reparaturanfrage wird in der Datenbank gespeichert.
- Ein Benachrichtigungsereignis wird ausgelöst, um den Werkstattinhaber über die neue Reparaturanfrage zu informieren.
- Der Lagerhalter erhält eine Bestätigung, dass die Reparaturanfrage erfolgreich an die Werkstatt übermittelt wurde.

### 1.2.8 Usecase 8: “ Reparaturtermine vereinbaren”

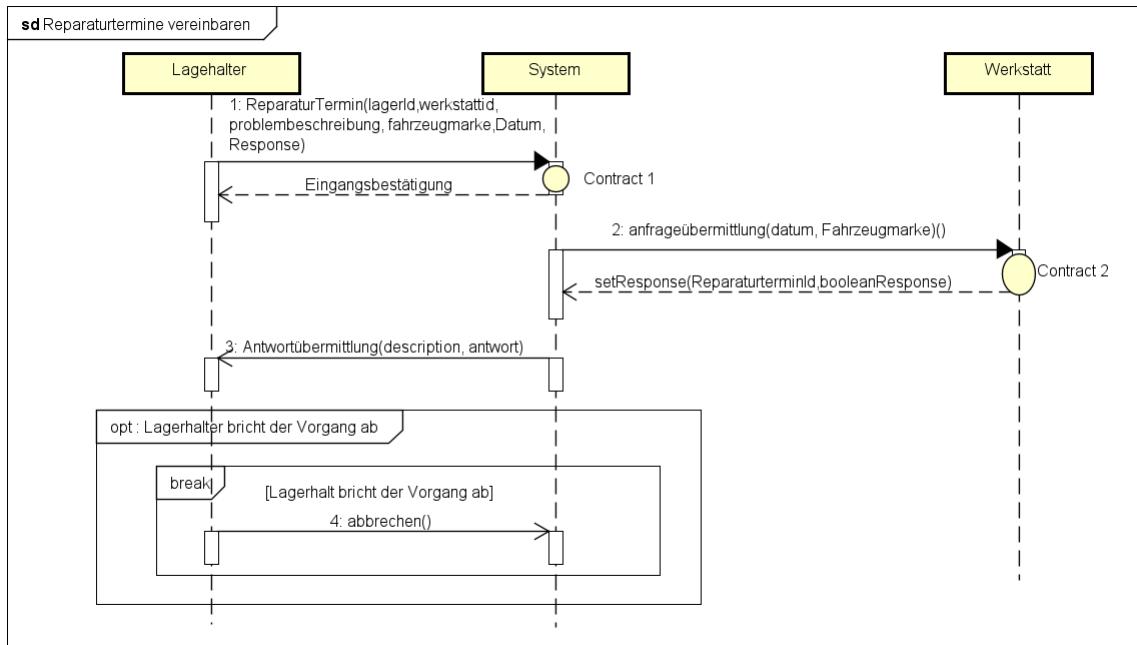


Abbildung 1.10: “ Reparaturtermine vereinbaren”.

**Contract 1:** Reparaturtermine vereinbaren (Anfrage stellen)

**Operation:** ReparaturTermin(datum: Date(), description: String, Fahrzeugsdetail: string[], werkstattId: Integer, lagerId: Integer)

**Querverweis:** Use Cases “Reparaturtermine vereinbaren(Anfrage)”

**Vorbedingungen:**

- Der Lagerhalter ist im System registriert und authentifiziert.
- Das Fahrzeug wurde registriert (bei der Stellplatzbuchung) und die Fahrzeuginformationen sind verfügbar.
- Eine Werkstatt wurde registriert und ist bereit, Anfragen zu empfangen.

**Nachbedingungen:**

- Eine Instanz von Reparaturanfrage wird erstellt.
- Die Reparaturanfrage wird mit dem entsprechenden FahrzeugsDetails und der Werkstatt verknüpft.
- Die Reparaturanfrage wird in der Datenbank gespeichert.
- Ein Benachrichtigungsereignis wird ausgelöst, um den Werkstattinhaber über die neue Reparaturanfrage zu informieren.
- Der Lagerhalter erhält eine Bestätigung, dass die Reparaturanfrage erfolgreich an die Werkstatt übermittelt wurde.

**Contract 2:** Reparaturtermine vereinbaren (Response der Werkstatt)

**Operation:** setResponse( ReparaturterminId: int ,description: String, Antwort:boolean)

**Querverweis:** Use Cases “Reparaturtermine vereinbaren(Response)”

**Vorbedingungen:**

- Eine Reparaturanfrage existiert bereits und wurde von der Werkstatt erhalten.
- Die Werkstatt hat die Reparaturanfrage angenommen und möchte einen Reparaturtermin mit dem Lagerhalter vereinbaren.

**Nachbedingungen:**

- Eine Instanz von ReparaturResponse wird erstellt, um die Rückmeldung der Werkstatt zu speichern.
- Das Attribut Antwort wird angepasst je nach der Antwort der Werkstatt(JA/NEIN)
- Die ReparaturResponse wird in der Datenbank gespeichert.
- Der Lagerhalter die ReparaturResponse.

## 1.3 Einlagerer

### 1.3.1 Usecase 1: “Nach Stellplätzen suchen”

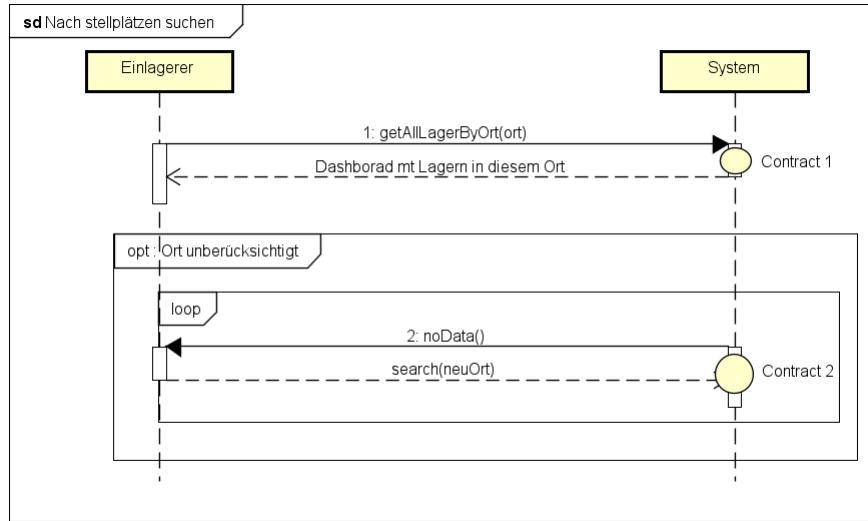


Abbildung 1.11: “Nach Stellplätzen suchen”.

**Contract 1:** Nach Stellplätzen suchen

**Operation:** `search(ort: String)`

**Querverweis:** Use Cases “Nach Stellplätzen suchen”

**Vorbedingungen:**

- Der Benutzer ist bereits im System angemeldet. **Einlagerer**.
- Der Benutzer möchte nach Stellplätzen für sein Fahrzeug suchen.
- Die Suche wird auf ausgewählte Städte beschränkt.

**Nachbedingungen:**

- Das System sucht alle Lager, die sich in der angegebenen Stadt befinden.
- Das System gibt dem Benutzer eine Liste von Lagern zurück, die in der angegebenen Stadt verfügbar sind.
- Der Benutzer kann weitere Details zu jedem Lager (z. B. Adresse, verfügbare Services) einsehen.

### 1.3.2 Usecase 2: “Anfragen an Lagerhalter stellen”

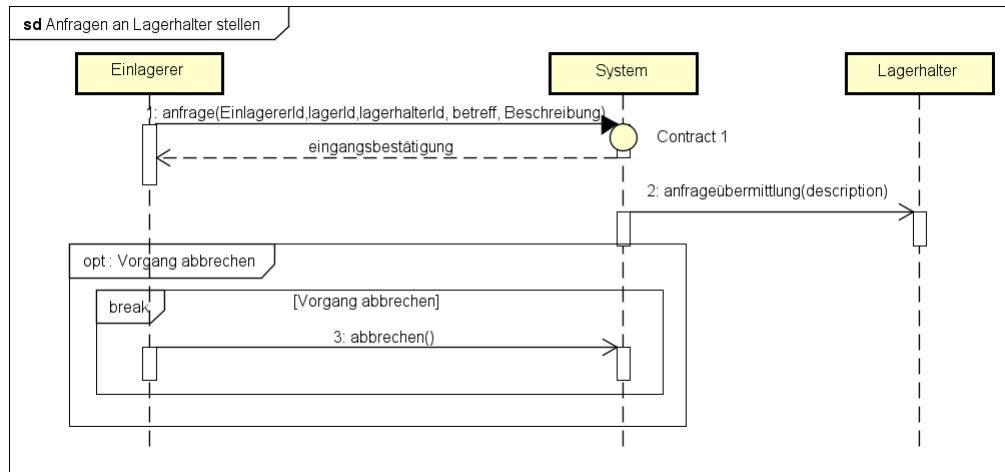


Abbildung 1.12: “Anfragen an Lagerhalter stellen”.

**Contract 1:** Anfragen an Lagerhalter stellen

**Operation:** `anfrage(EinlagererId:Integer, lagerId: Integer, lagerhalterId:Integer, betreff: String, Beschreibung:String)`

**Querverweis:** Use Cases “Anfragen an Lagerhalter stellen”

**Vorbedingungen:**

- Der Einlagerer ist bereits im System angemeldet.
- Der Einlagerer möchte eine Anfrage an einen Lagerhalter für zusätzliche Informationen stellen.

**Nachbedingungen:**

- Eine neue Instanz von Anfrage wird im System erstellt. Die Anfrage wurde mit dem Lagerhalter assoziiert
- Die Anfrage wird in der Datenbank persistent gespeichert und dem entsprechenden Lagerhalter übermittelt.
- Eine Bestätigung wird an den Einlagerer gesendet

### 1.3.3 Usecase 3: “Angebot annehmen oder ablehnen”

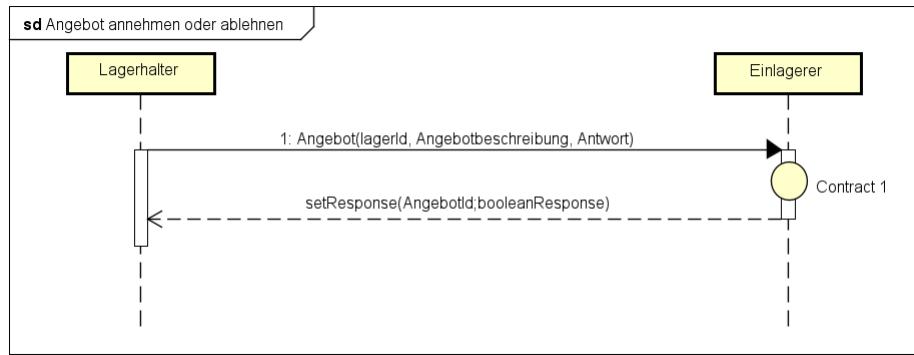


Abbildung 1.13: “Angebot annehmen oder ablehnen”.

**Contract 1:** Angebot annehmen oder ablehnen

**Operation:** `angebot(lagerId: Integer, Angebotbeschreibung: String, Antwort:boolean)`

**Querverweis:** Use Cases “Angebot annehmen oder ablehnen”

**Vorbedingungen:**

- Der Einlagerer hat ein Angebot von einem Lagerhalter erhalten.
- Der Einlagerer möchte dieses Angebot entweder annehmen oder ablehnen.

**Nachbedingungen:**

- Basierend auf dem Wert von Antwort (true für annehmen, false für ablehnen) wird das Angebot entsprechend akzeptiert oder abgelehnt.
- Der Lagerhalter wird über die Entscheidung, ob das Angebot akzeptiert oder abgelehnt wurde, informiert.

### 1.3.4 Usecase 4: “sich Ersatzteile anzeigen lassen”

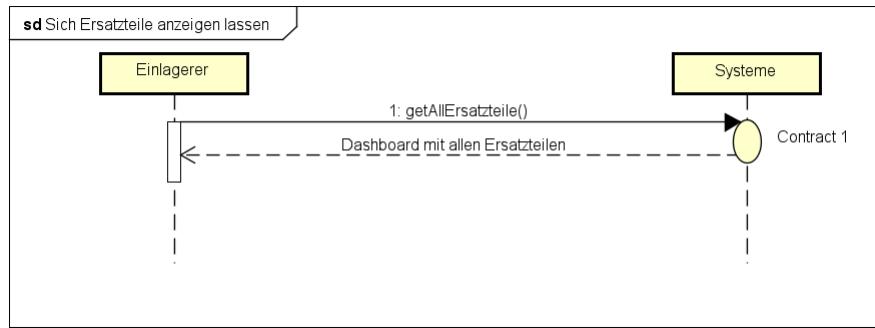


Abbildung 1.14: “sich Ersatzteile anzeigen lassen”.

**Contract 1:** sich Ersatzteile anzeigen lassen

**Operation:** `getAllErsatzteile()`

**Querverweis:** Use Cases “sich Ersatzteile anzeigen lassen”

**Vorbedingungen:**

- Der Einlagerer möchte sich Ersatzteile anzeigen lassen.
- Ersatzteilhändler haben Ersatzteile für verschiedene Fahrzeugtypen in der Datenbank verfügbar.

**Nachbedingungen:**

- Basierend auf den verfügbaren Ersatzteilen in der Datenbank werden alle Ersatzteile angezeigt.
- Alle Ersatzteile werden mit den notwendigen Informationen wie Bezeichnung, Preis und Verfügbarkeit angezeigt.

### 1.3.5 Usecase 5: “Abhol- und Rückgabetermin vereinbaren”

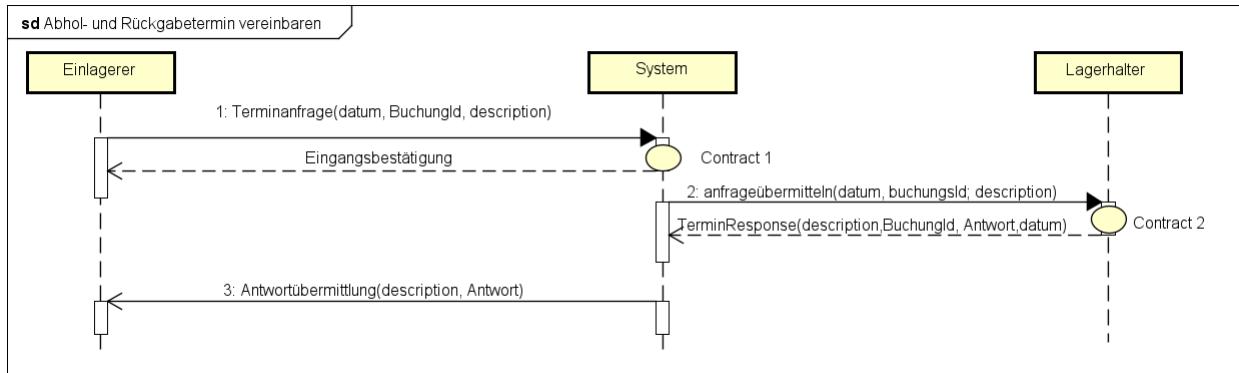


Abbildung 1.15: “Abhol- und Rückgabetermin vereinbaren”.

**Contract 1:** Abhol- und Rückgabetermin vereinbaren (Anfrage)

**Operation:** Terminanfrage(datum: Date(), BuchungId: Integer, description: String)

**Querverweis:** Use Cases “Abhol- und Rückgabetermin vereinbaren”

**Vorbedingungen:**

- Der Lagerhalter bietet eine Möglichkeit zur Buchung eines Abhol- und Rückgabetermins für ein Fahrzeug an.
- Ein Lagerhalter hat ein Fahrzeug, das zur Abholung und Rückgabe bereitsteht.
- Der Einlagerer möchte einen Termin für die Abholung und Rückgabe seines Fahrzeugs vereinbaren.

**Nachbedingungen:**

- Eine Instanz der Anfrage für die Abholung und Rückgabe eines Fahrzeugs wurde erstellt.
- Die Anfrage enthält die vom Einlagerer gewünschten Abhol- und Rückgabetermine.
- Die Anfrage wurde an den Lagerhalter gesendet.

**Contract 2:** Abhol- und Rückgabetermin vereinbaren (Response)

**Operation:** TerminResponse(datum: Date(), BuchungId: Integer, description: String, Antwort : boolean)

**Querverweis:** Use Cases “Abhol- und Rückgabetermin vereinbaren”

**Vorbedingungen:**

- Ein Lagerhalter hat eine Anfrage zur Vereinbarung eines Abhol- und Rückgabetermins für ein Fahrzeug erhalten.
- Der Lagerhalter hat die Anfrage geprüft und eine Entscheidung getroffen, ob der Termin bestätigt oder abgelehnt wird.

**Nachbedingungen:**

- Eine Instanz der Antwort auf die Anfrage wird erstellt, die den Status (bestätigt oder abgelehnt) und einen Kommentar enthält.
- Das Attribut Antwort wird mit dem Status (boolean) des Termins belegt.
- Die Antwort wird an den Einlagerer zurückgesendet, um den Status der Anfrage zu übermitteln.

### 1.3.6 Usecase 6: “Zusätzliche Services buchen”

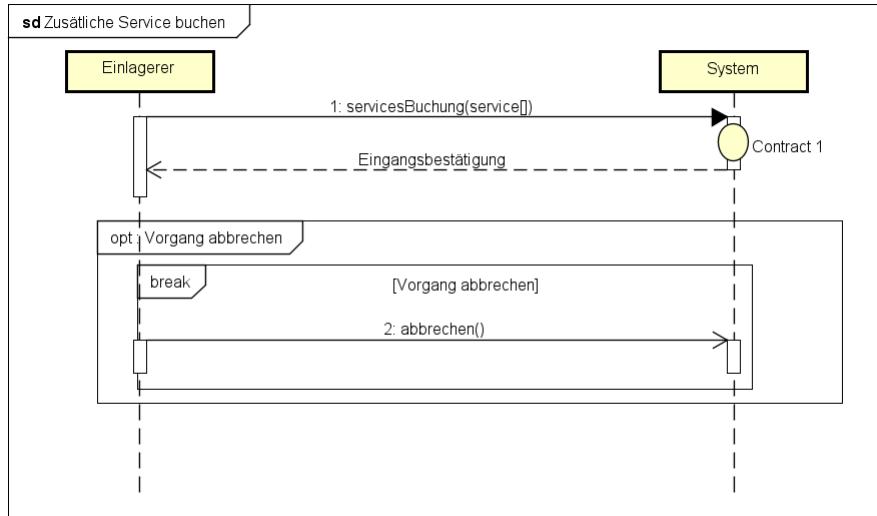


Abbildung 1.16: “Zusätzliche Services buchen”.

**Contract 1:** Zusätzliche Services buchen

**Operation:** serviceBuchung(services : Service[])

**Querverweis:** Use Cases “Zusätzliche Services buchen”

**Vorbedingungen:**

- Der Einlagerer hat bereits einen Stellplatz für sein Fahrzeug reserviert und den Buchungsprozess für zusätzliche Services gestartet.
- Der Einlagerer kann zusätzliche Services zu seiner Buchung hinzufügen (z.B. Reinigung etc.).
- Der Service ist in dem entsprechenden Lager verfügbar.

**Nachbedingungen:**

- Eine Instanz der Buchung für zusätzliche Services wurde erzeugt.
- Die Details des Services wurden der Buchung hinzugefügt.
- Der zusätzliche Service wird mit der bestehenden Buchung verknüpft.
- Die Buchung der zusätzlichen Services wird gespeichert und in der Datenbank aktualisiert.
- Der Buchungsprozess für zusätzliche Services ist abgeschlossen und ein Bestätigungsereignis wird an den Benutzer gesendet.

### 1.3.7 Usecase 7: “Ersatzteilreservierungen durchführen”

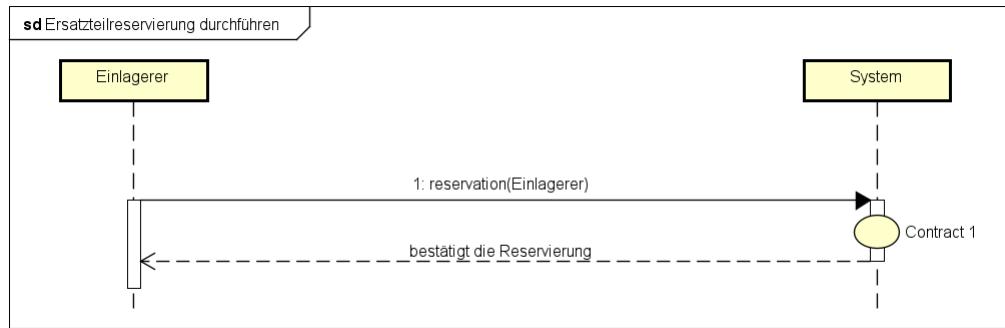


Abbildung 1.17: “Ersatzteilreservierungen durchführen”.

**Contract 1:** Ersatzteilreservierungen durchführen

**Operation:** Reservierung(*einlagerer* : Benutzer)

**Querverweis:** Use Cases “Ersatzteilreservierungen durchführen”

**Vorbedingungen:**

- Der Einlagerer sucht nach Ersatzteilen für sein Fahrzeug.
- Das Ersatzteil ist im Inventar des Anbieters oder Händlers verfügbar.

**Nachbedingungen:**

- Eine neue Instanz der Ersatzteilreservierung wird erstellt.
- Die Menge und die Ersatzteil-ID wurden in der Reservierung gesetzt.
- Die Ersatzteilreservierung wird mit dem Einlagerer verknüpft.
- Die Reservierung wird in der Datenbank gespeichert.
- Ein Bestätigungsereignis wird an den Einlagerer gesendet.

## 1.4 Werkstattinhaber

### 1.4.1 Usecase 1: “Werkstatt einfügen”

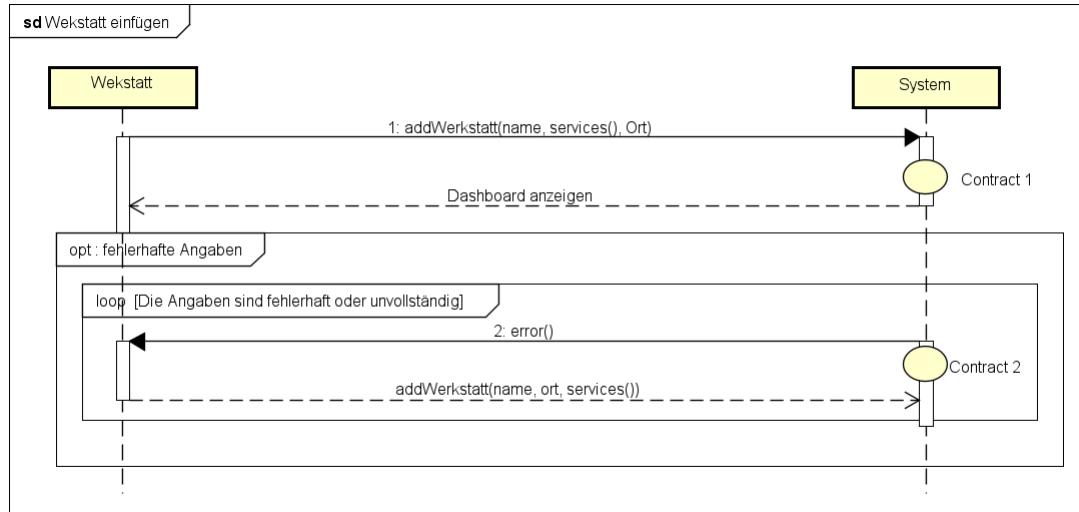


Abbildung 1.18: “Werkstatt einfügen”.

**Contract 1:** Werkstatt einfügen

**Operation:** `addWerkstatt(name: string, Ort: enum, Services: string[])`

**Querverweis:** Use Cases “Werkstatt einfügen”

**Vorbedingungen:**

- Der Benutzer ist authentifiziert und hat die Rolle eines **Werkstattinhaber**.
- Die Eingabedaten (name, Ort, Services) sind vollständig und gültig.

**Nachbedingungen:**

- Eine Instanz von **Werkstatt** wurde erzeugt.
- Die Attribute der **Werkstatt**-Instanz (Name, Ort, Services) wurden mit den bereitgestellten Werten belegt.
- Die **Werkstatt**-Instanz wurde in das System (Datenbank)persistiert.
- Der Werkstatt wurde eine eindeutige ID zugewiesen.
- Eine Bestätigung des erfolgreichen Hinzufügens des Werkstatt wurde an den Benutzer zurückgegeben.

**Contract 2:** unvollständige oder fehlerhafte Daten

**Operation:** `error()`

**Querverweis:** Use Cases “Werkstatt einfügen”

**Vorbedingungen:**

- Ein Werkstattinhaber möchte ein Werkstatt hinzufügen.
- Das angegebene Werkstatt (mit dem gleichen Namen und im gleichen Ort Attributen) existiert bereits im System oder die Eingabefelder sind unvollständig.

## Nachbedingungen:

- Keine Instanz von **Werkstatt** wurde erstellt.
- Ein Fehler wurde ausgelöst und an den Werkstattinhaber zurückgegeben, der darauf hinweist, dass die Daten unvollständig sind.
- Der Benutzer wurde aufgefordert, einen Namen oder Ort für das Werkstatt einzugeben.
- Der Prozess des Werkstatthinzufügens bleibt offen, bis gültige Daten eingegeben werden.

### 1.4.2 Usecase 2: “Reparaturanfragen empfangen”

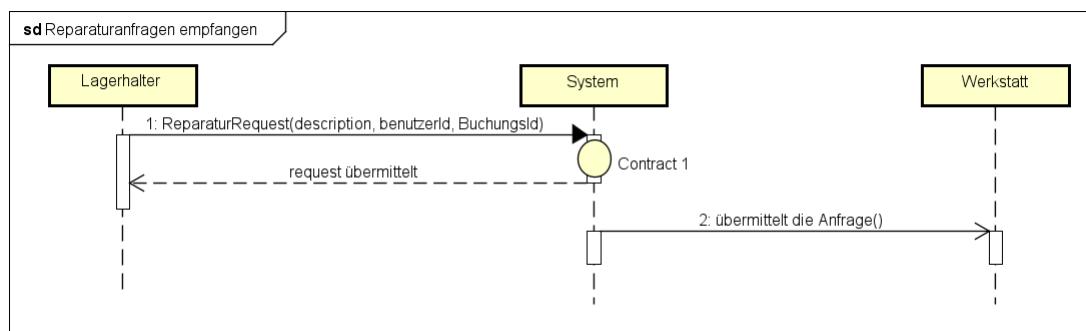


Abbildung 1.19: “Reparaturanfragen empfangen”.

#### Contract 1: Reparaturanfragen empfangen

**Operation:** ReparaturRequest(description: string, benutzerId: Integer, BuchungId: Integer)

**Querverweis:** Use Cases “Reparaturanfragen empfangen”

#### Vorbedingungen:

- Eine Reparaturanfrage wurde von einem Lagerhalter gestellt.
- Die Anfrage enthält Details zur Reparatur oder Wartung eines Fahrzeugs.
- Die Werkstatt ist für Reparaturen oder Wartungsarbeiten zuständig und hat ein Profil im System.

#### Nachbedingungen:

- Eine Reparaturanfrage wird von einem Lagerhalter an die Werkstatt übermittelt.
- Die Reparaturanfrage wird mit der entsprechenden Werkstatt instanziert und verknüpft.
- Eine E-Mail wird an den Lagerhalter geschickt, um ihm die erfolgreiche Übermittlung der Reparaturanfrage zu bestätigen.

## 1.5 Ersatzteilanbieter

### 1.5.1 Usecase 1: “Ersatzteile anbieten”

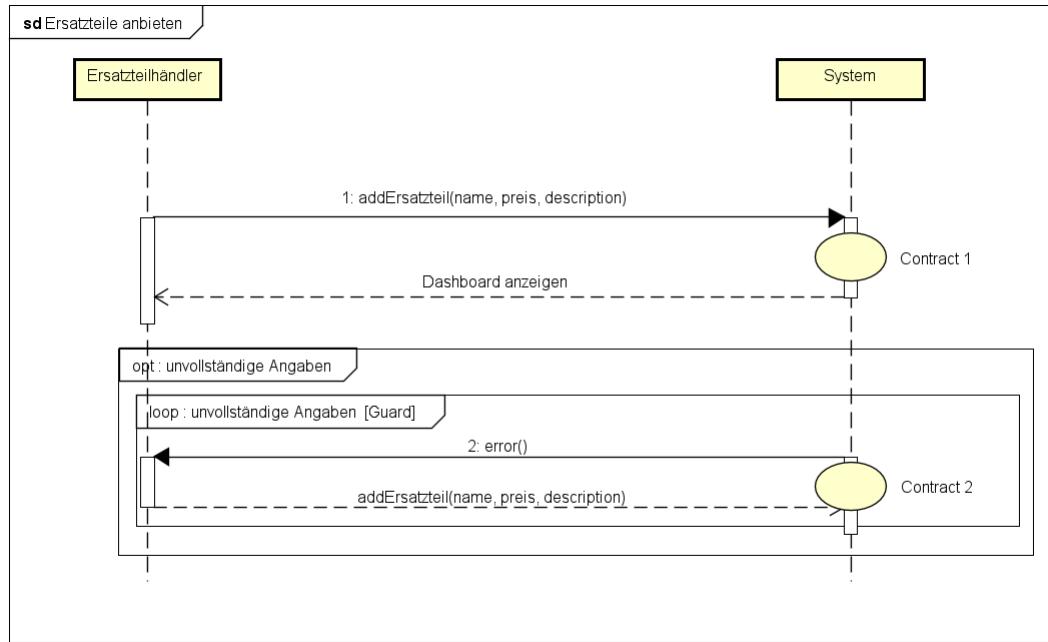


Abbildung 1.20: “Ersatzteile anbieten”.

#### Contract 1: Ersatzteile anbieten

**Operation:** `addErsatzteil(name: string, preis: Float, description: string)`

**Querverweis:** Use Cases “Ersatzteile anbieten”

**Vorbedingungen:**

- Der Benutzer ist authentifiziert und hat die Rolle eines **Ersatzteilanbieter**.
- Die Eingabedaten (name, preis, description) sind vollständig und gültig.

**Nachbedingungen:**

- Eine Instanz von **Ersatzteil** wurde erzeugt.
  - Die Attribute der **Ersatzteil**-Instanz (Name, preis, description) wurden mit den bereitgestellten Werten belegt.
  - Die **Ersatzteil**-Instanz wurde in das System (Datenbank)persistiert.
  - Der Ersatzteil wurde eine eindeutige ID zugewiesen.
- 
- Eine Bestätigung des erfolgreichen Hinzufügens des Ersatzteil wurde an den Benutzer zurückgegeben.

#### Contract 2: unvollständige Angaben

**Operation:** `error()`

**Querverweis:** Use Cases “Ersatzteile anbieten”

**Vorbedingungen:**

- Ein Ersatzteilanbieter möchte einen Ersatzteil hinzufügen.
- die Eingabefelder sind unvollständig.

#### Nachbedingungen:

- Keine Instanz von **Ersatzteil** wurde erstellt.
- Ein Fehler wurde ausgelöst und an den Ersatzteilanbieter zurückgegeben, der darauf hinweist, dass die Daten unvollständig sind.
- Der Benutzer wurde aufgefordert, einen Namen oder Preis für den Ersatzteil einzugeben.
- Der Prozess des Ersatzteilhinzufügens bleibt offen, bis gültige Daten eingegeben werden.

#### 1.5.2 Usecase 2: “Ersatzteile löschen”

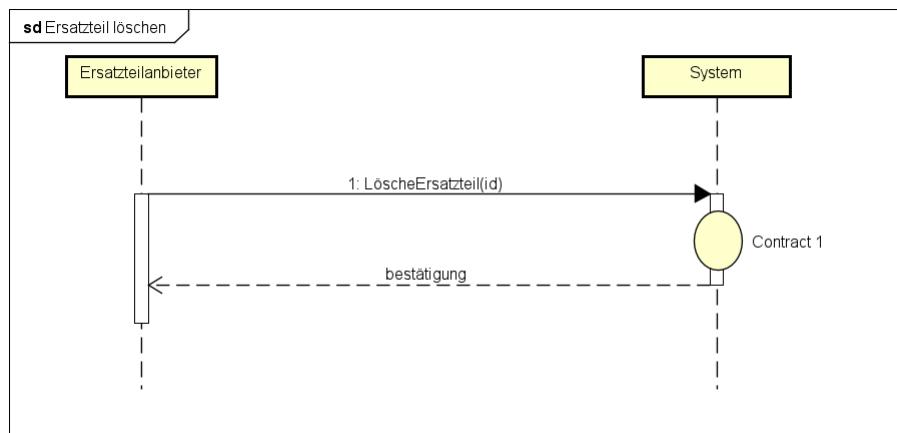


Abbildung 1.21: “Ersatzteile löschen”.

**Contract 1:** Ersatzteile löschen

**Operation:** `deleteErsatzteil(id: Integer)`

**Querverweis:** Use Cases “Ersatzteile löschen”

**Vorbedingungen:**

- Der Ersatzteilhändler hat sich im System registriert und ist berechtigt, Ersatzteile zu verwalten.
- Ein Ersatzteil ist bereits im System gespeichert .

**Nachbedingungen:**

- Das angegebene Ersatzteil wird aus der Ersatzteilliste entfernt.
- Alle Verknüpfungen des Ersatzteils mit Fahrzeugen werden entfernt.
- Eine Bestätigungs Nachricht wird an den Ersatzteilhändler gesendet, dass das Ersatzteil erfolgreich gelöscht wurde.

# Kapitel 2

## Interaktionsdiagramme

### 2.1 Benutzer(Alle Akteure im System)

#### 2.1.1 Usecase 1: “Sich registrieren”

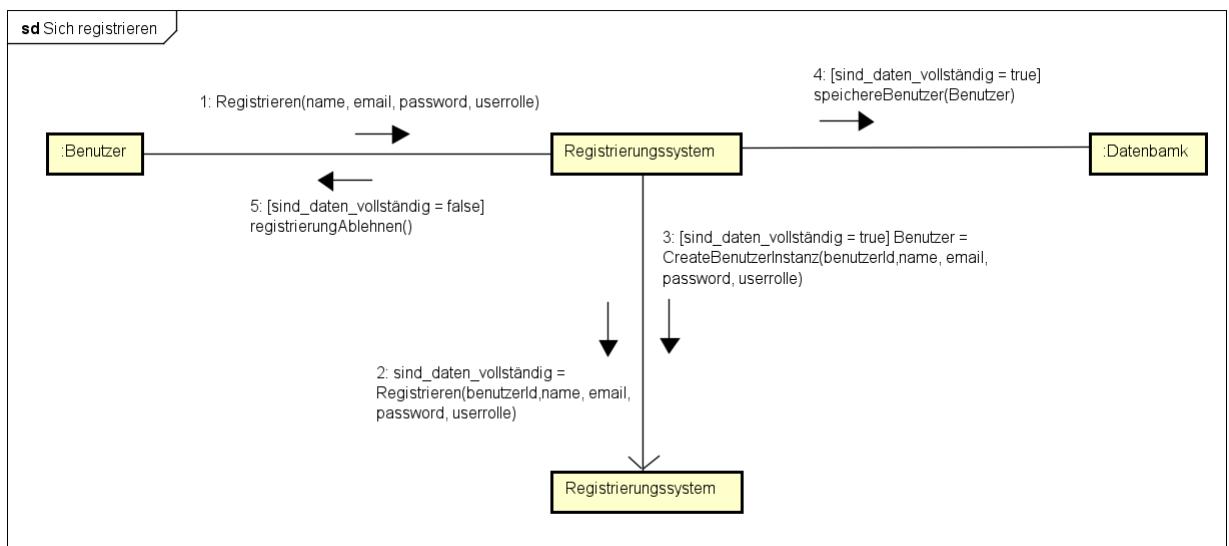


Abbildung 2.1: “Sich registrieren”.

#### 2.1.2 Usecase 2: “Sich anmelden”

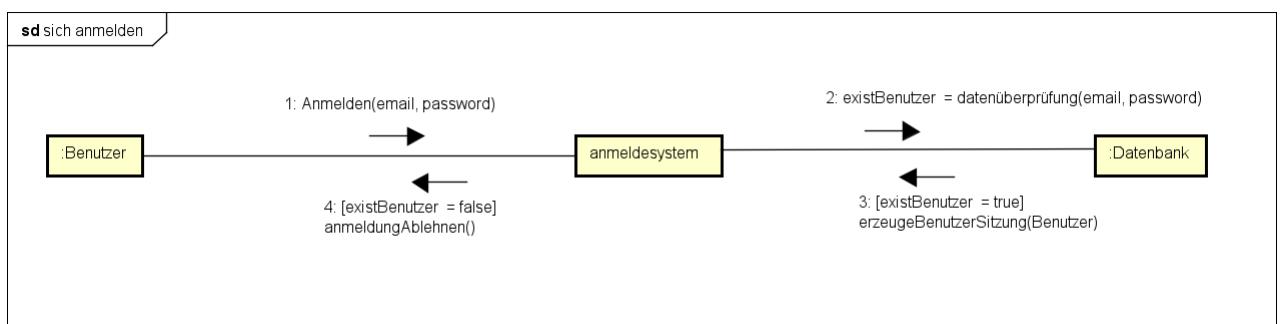


Abbildung 2.2: “Sich anmelden”.

## 2.2 Lagerhalter

### 2.2.1 Usecase 1: “Lager einfügen”

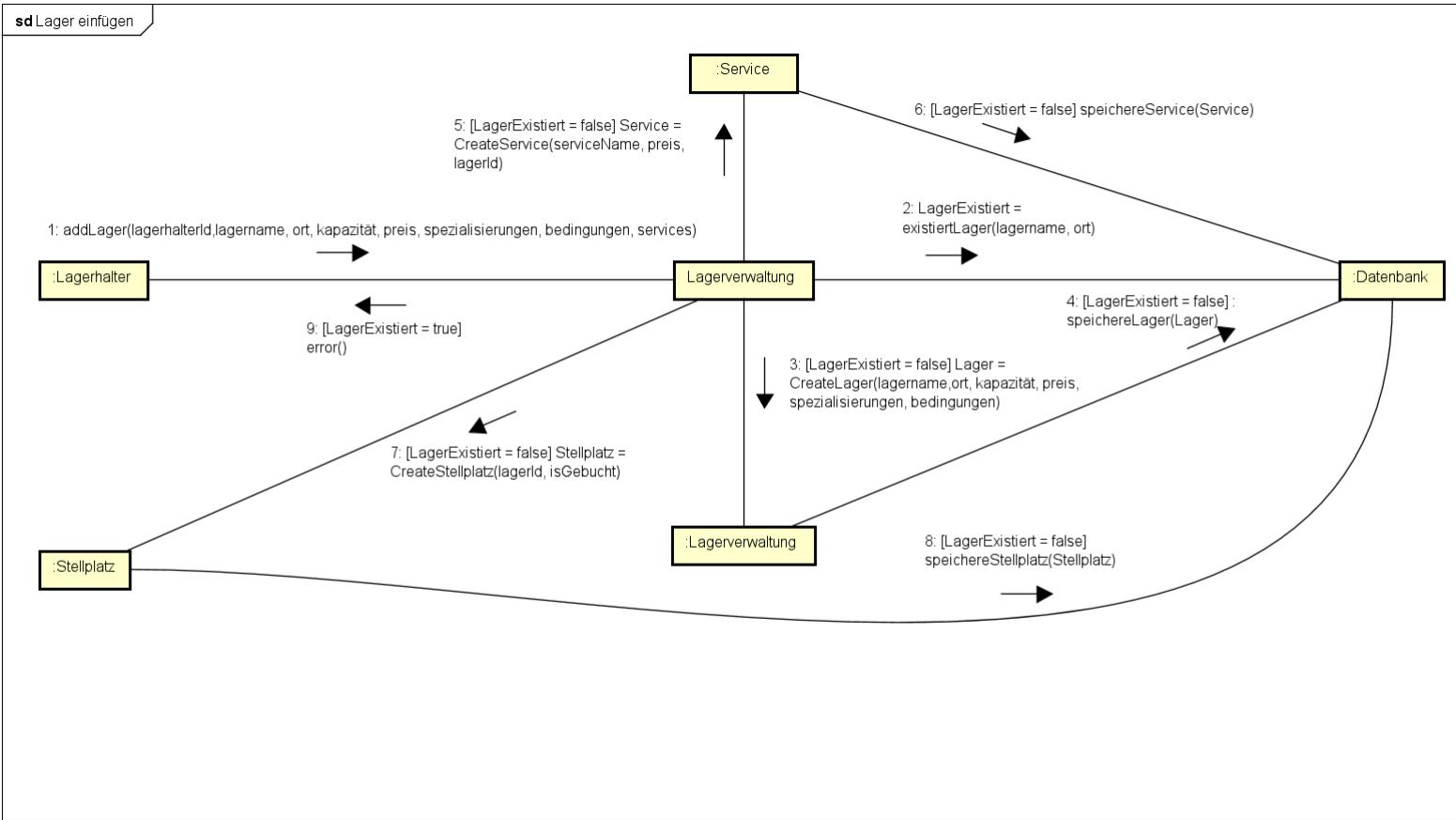


Abbildung 2.3: “Lager einfügen”.

### 2.2.2 Usecase 2: “Lagerinformationen aktualisieren”

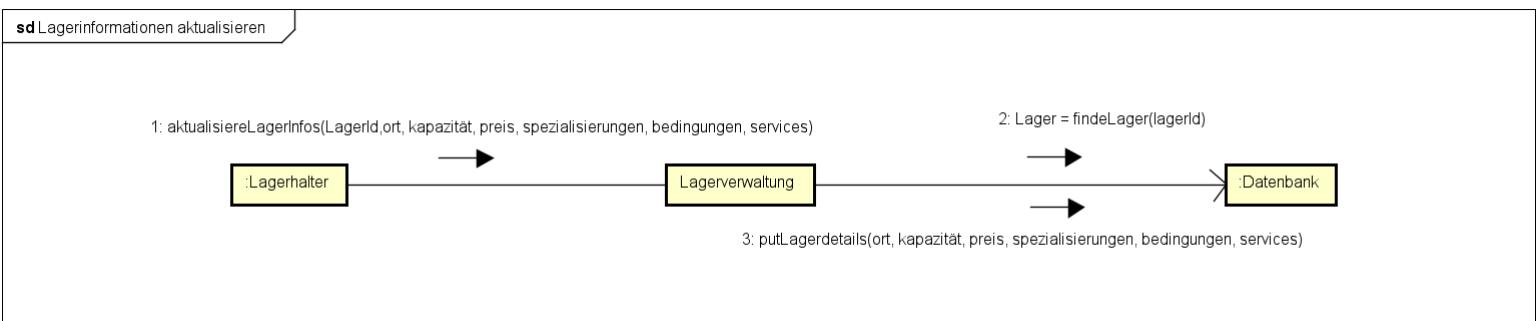


Abbildung 2.4: “Lagerinformationen aktualisieren”.

### 2.2.3 Usecase 3: “Services anbieten”

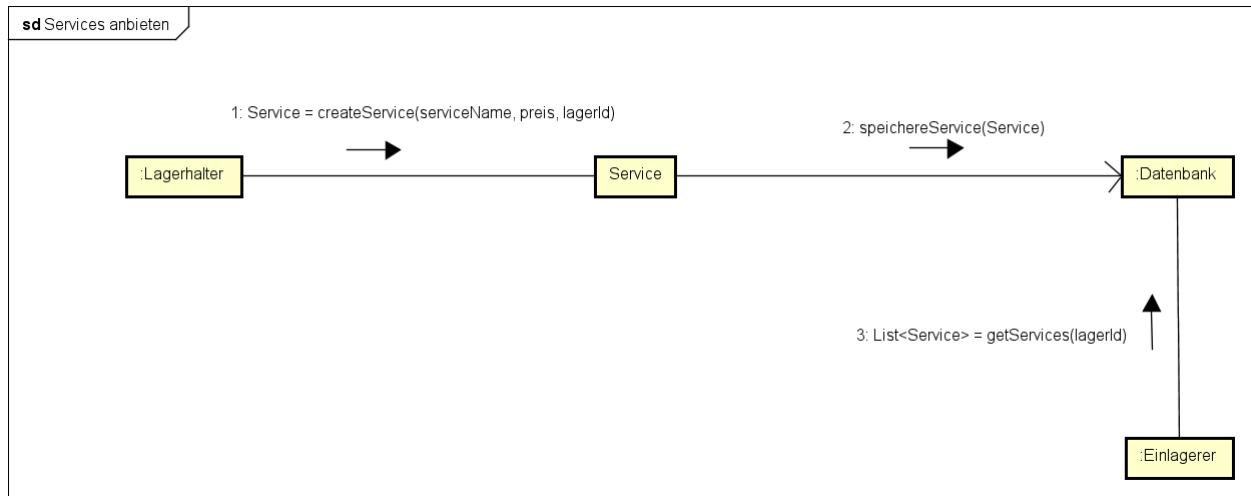


Abbildung 2.5: “Services anbieten”.

### 2.2.4 Usecase 4: “Verfügbare Plätze anzeigen”

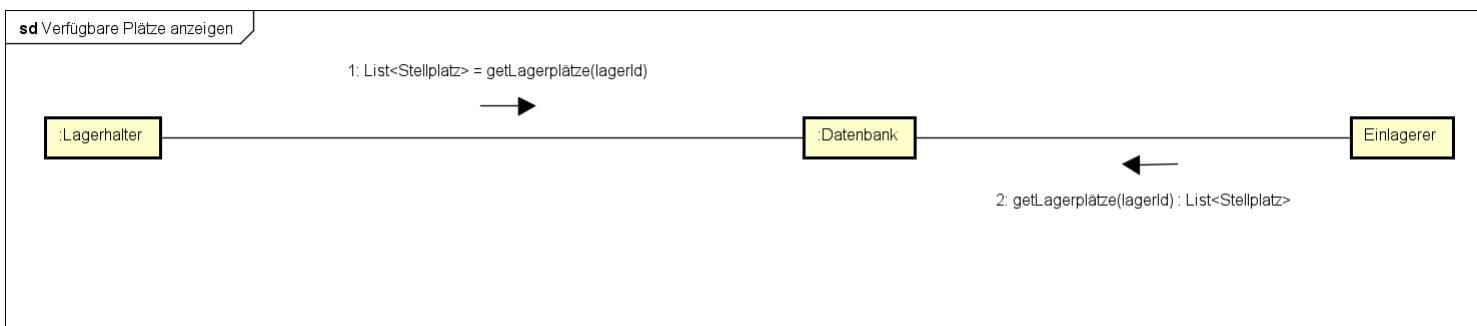


Abbildung 2.6: “Verfügbare Plätze anzeigen”.

## 2.2.5 Usecase 5: “Fahrzeugspezialisierung angeben”



Abbildung 2.7: “Fahrzeugspezialisierung angeben”.

## 2.2.6 Usecase 6: “Fahrzeuginformationen bereitstellen”

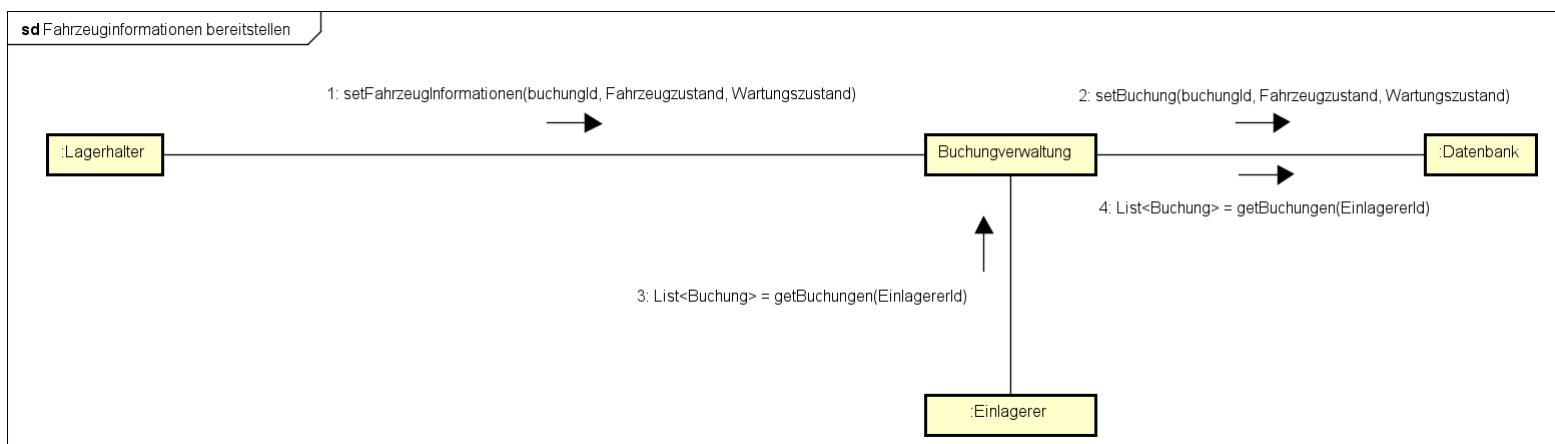


Abbildung 2.8: “Fahrzeuginformationen bereitstellen”.

## 2.2.7 Usecase 7: “Reparaturanfragen stellen”

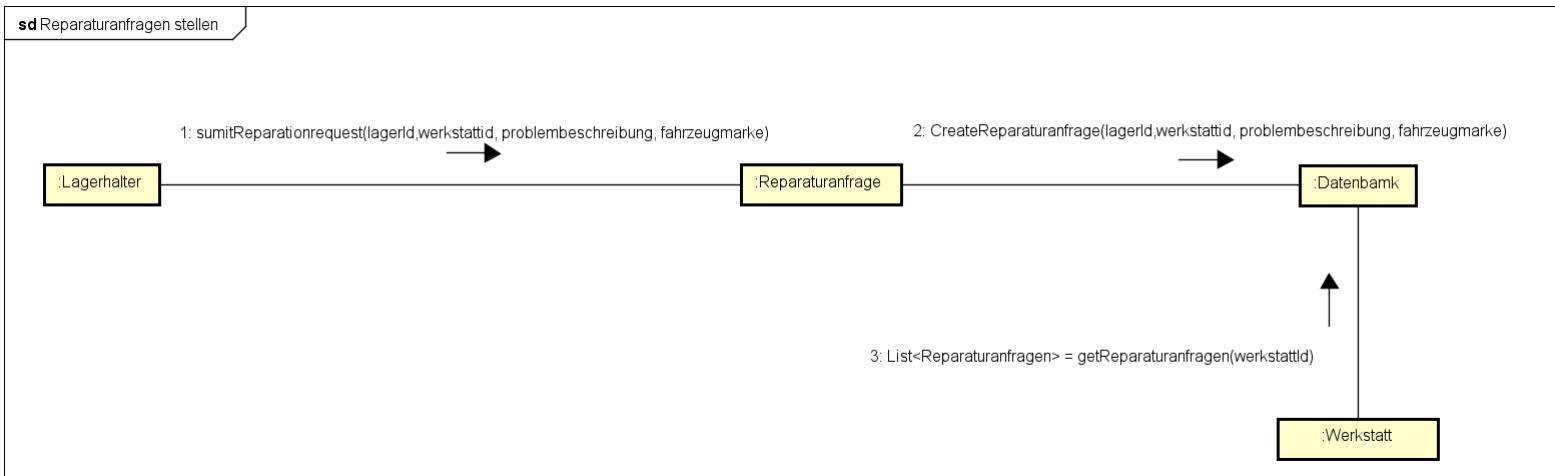


Abbildung 2.9: “Reparaturanfragen stellen”.

## 2.2.8 Usecase 8: “Reparaturtermine vereinbaren”

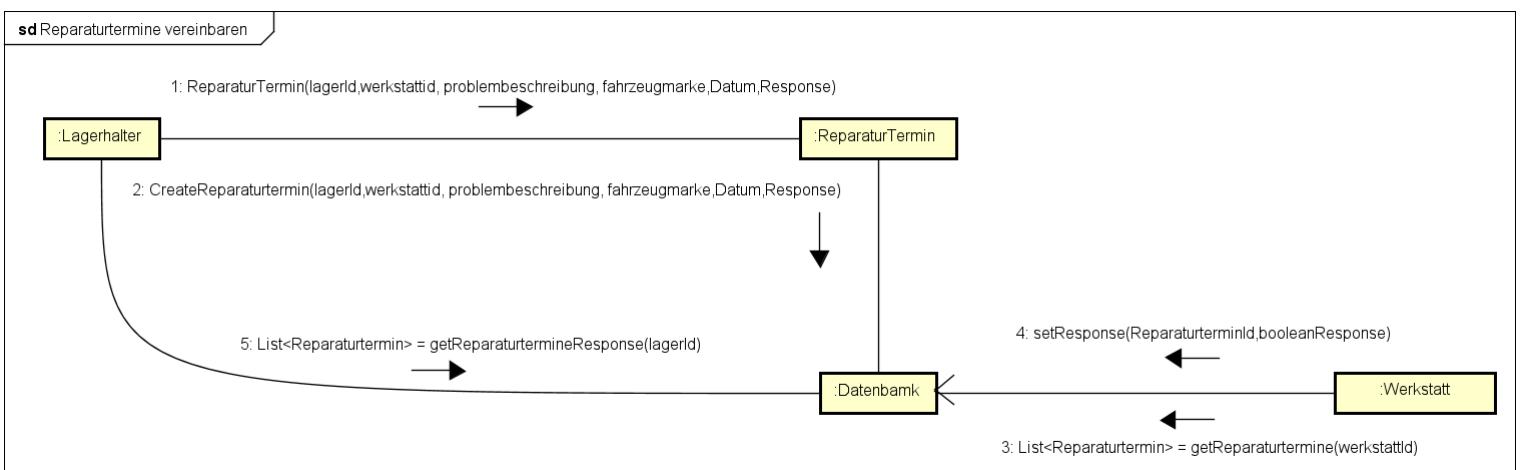


Abbildung 2.10: “Reparaturtermine vereinbaren”.

## 2.3 Einlagerer

### 2.3.1 Usecase 1: “Nach Stellplätzen suchen”

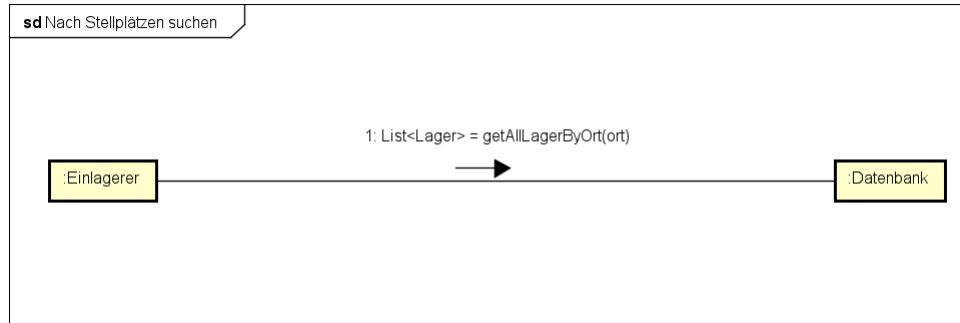


Abbildung 2.11: “Nach Stellplätzen suchen”.

### 2.3.2 Usecase 2: “Anfragen an Lagerhalter stellen”

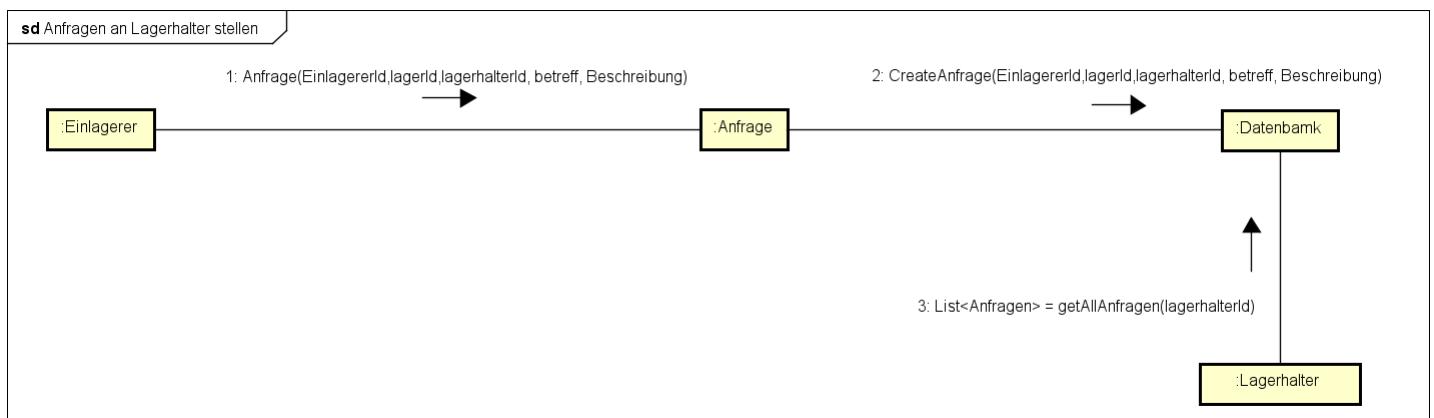


Abbildung 2.12: “Anfragen an Lagerhalter stellen”.

### 2.3.3 Usecase 3: “Angebot annehmen oder ablehnen”

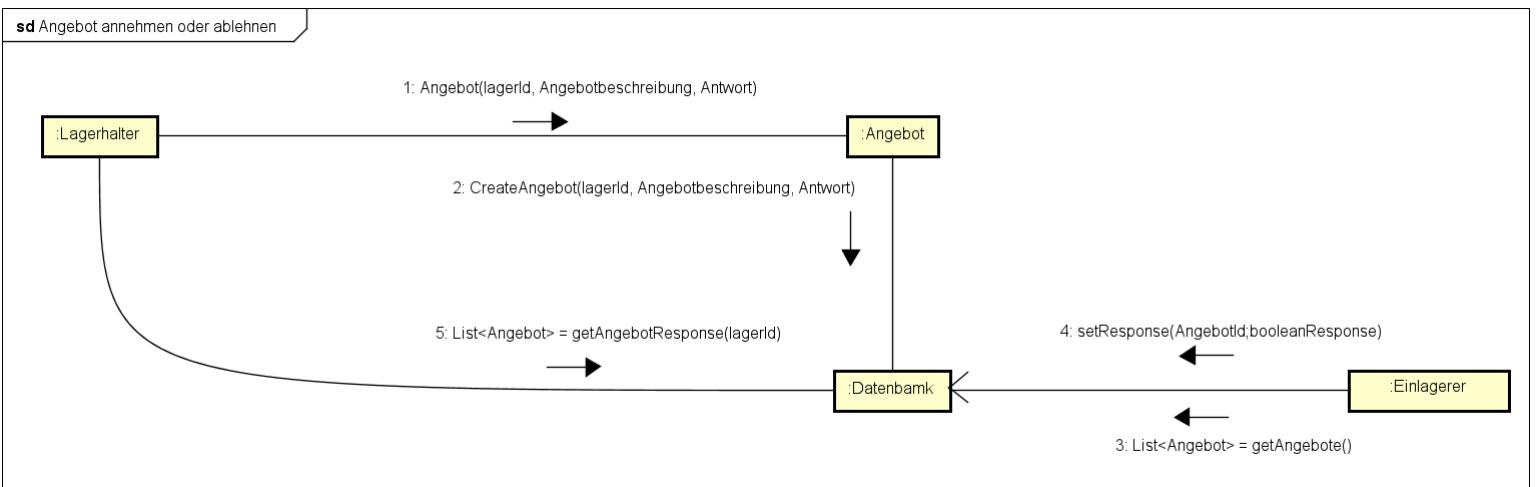


Abbildung 2.13: “Angebot annehmen oder ablehnen”.

### 2.3.4 Usecase 4: “sich Ersatzteile anzeigen lassen”

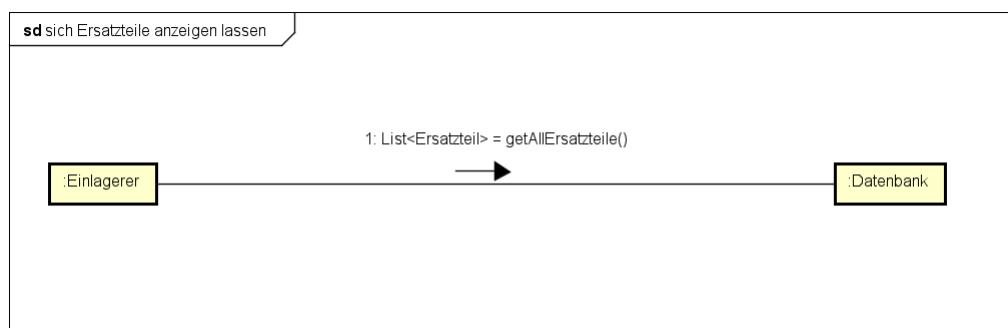


Abbildung 2.14: “sich Ersatzteile anzeigen lassen”.

### 2.3.5 Usecase 5: “Abhol- und Rückgabetermin vereinbaren”

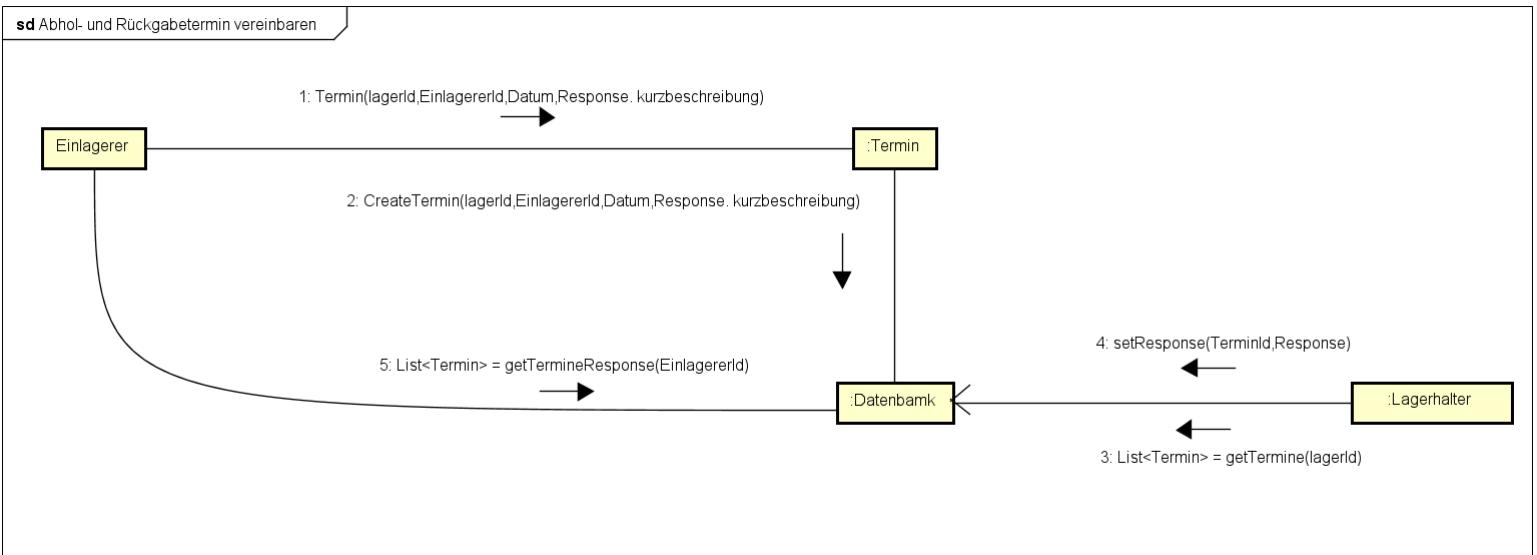


Abbildung 2.15: “Abhol- und Rückgabetermin vereinbaren”.

### 2.3.6 Usecase 6: “Zusätzliche Services buchen”

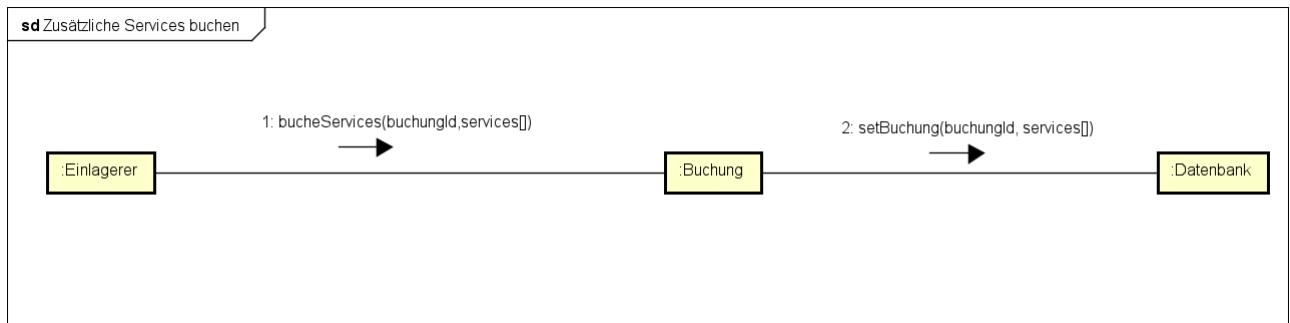


Abbildung 2.16: “Zusätzliche Services buchen”.

### 2.3.7 Usecase 7: “Ersatzteilreservierungen durchführen”

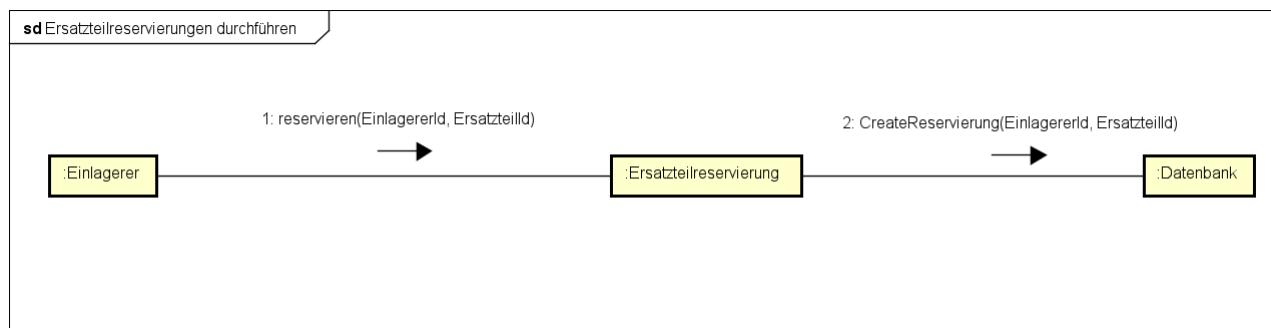


Abbildung 2.17: “Ersatzteilreservierungen durchführen”.

## 2.4 Werkstattinhaber

### 2.4.1 Usecase 1: “Werkstatt einfügen”

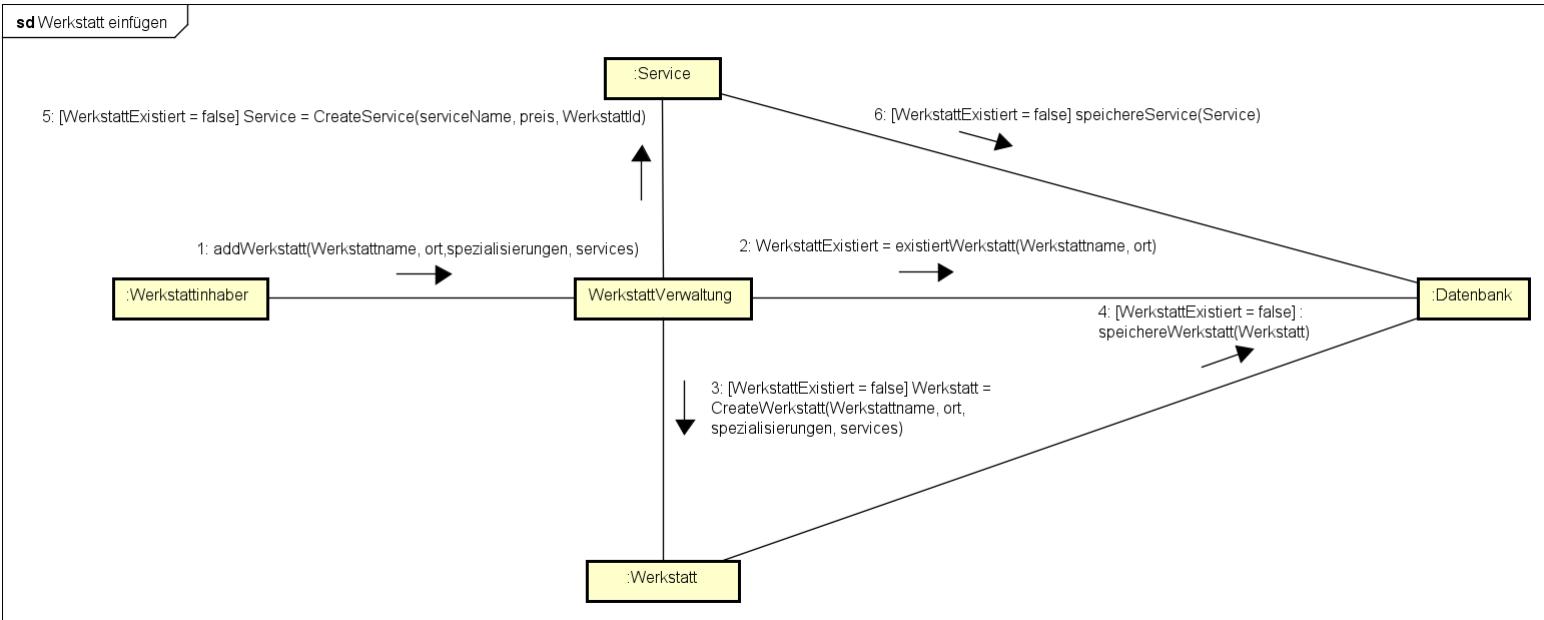


Abbildung 2.18: “Werkstatt einfügen”.

### 2.4.2 Usecase 2: “Reparaturanfragen empfangen”

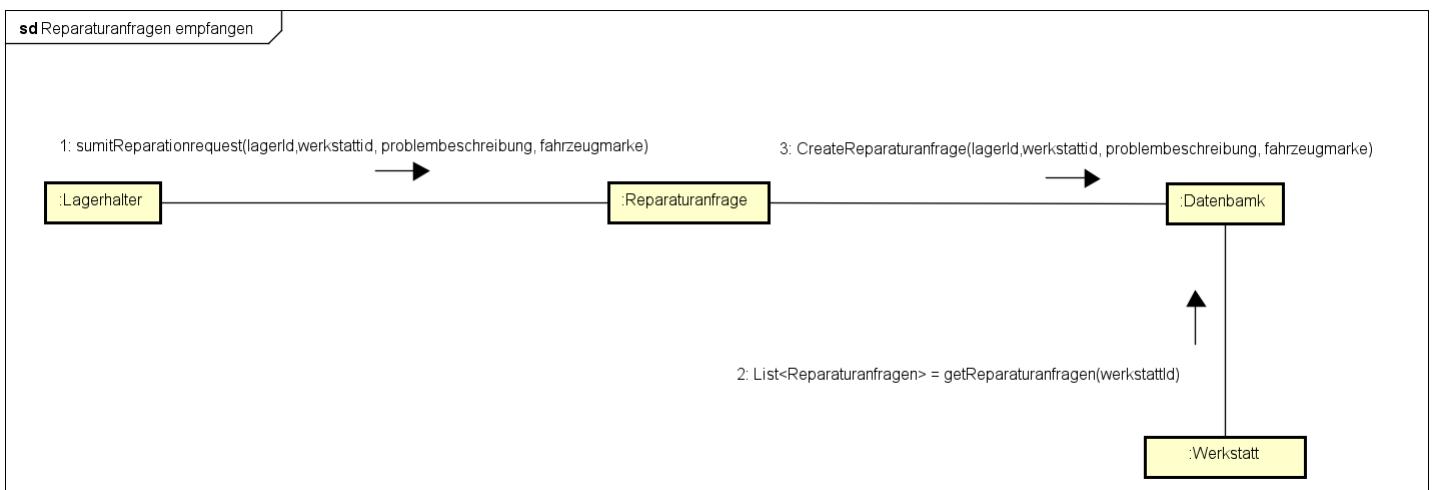


Abbildung 2.19: “Reparaturanfragen empfangen”.

## 2.5 Ersatzteilanbieter

### 2.5.1 Usecase 1: “Ersatzteile anbieten”

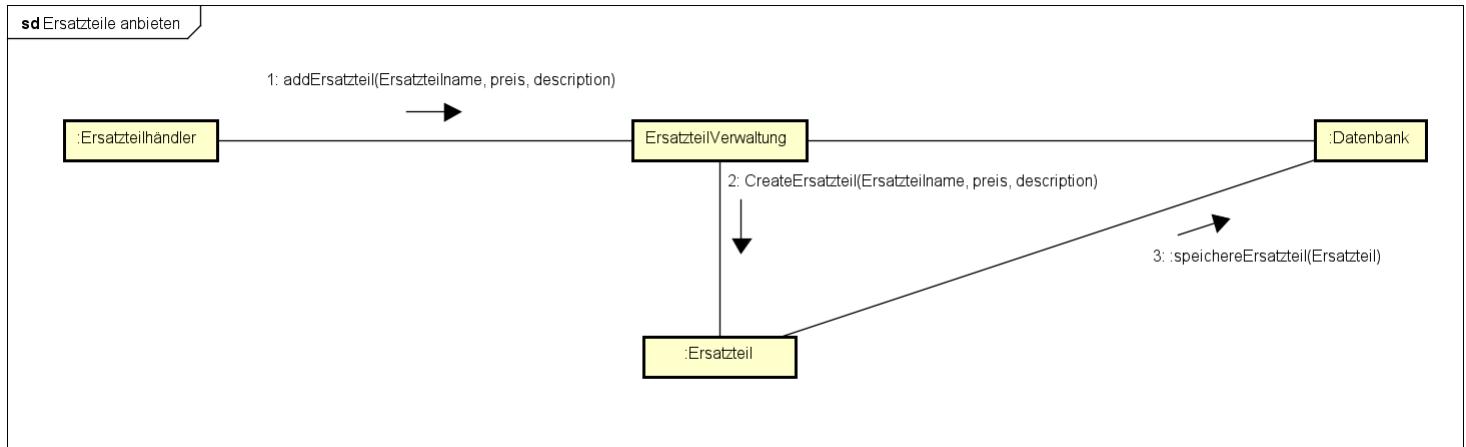


Abbildung 2.20: “Ersatzteile anbieten”.

### 2.5.2 Usecase 2: “Ersatzteile löschen”



Abbildung 2.21: “Ersatzteile löschen”.

# Kapitel 3

## Entwurfsklassendiagramm

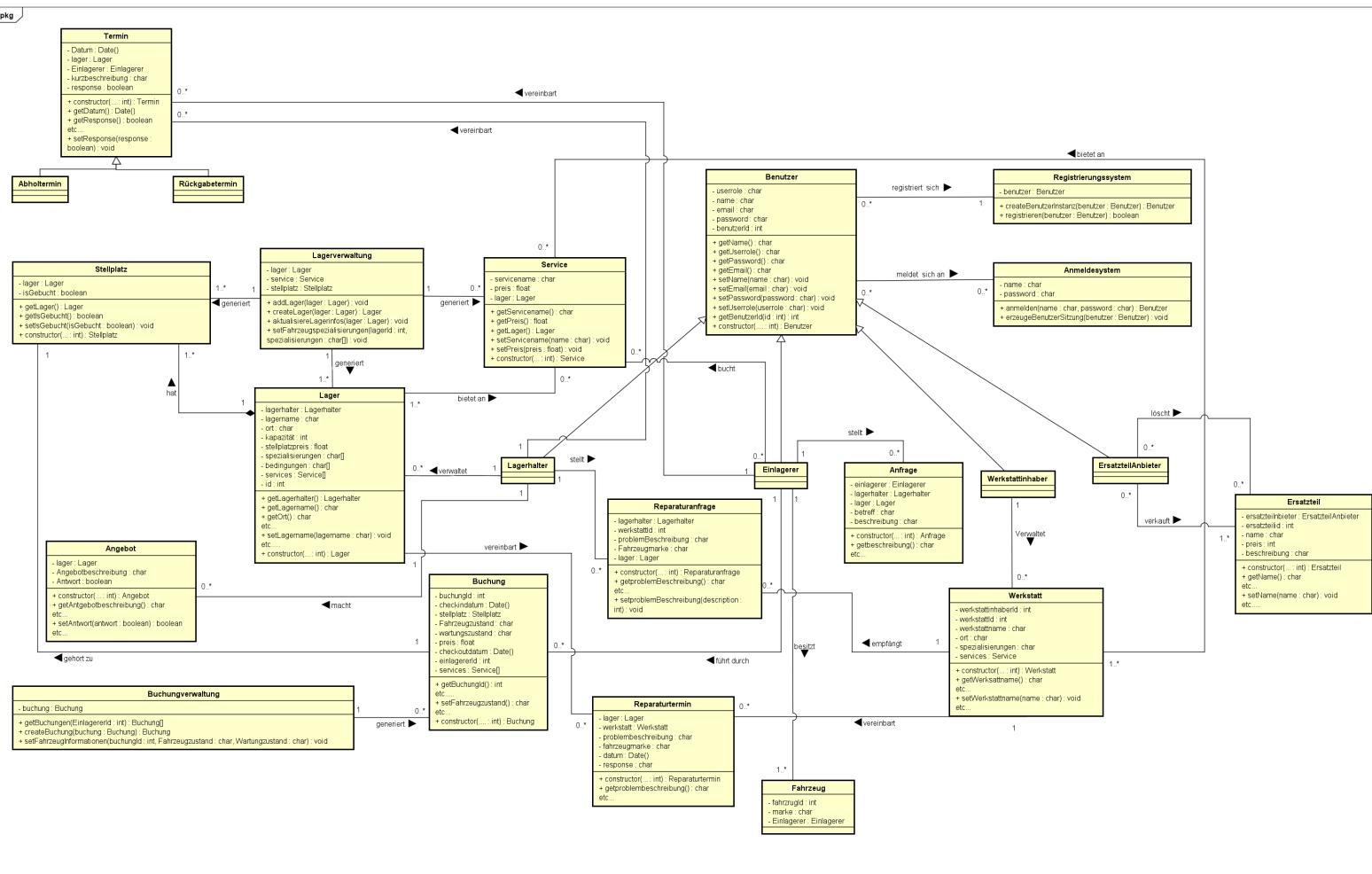


Abbildung 3.1: Entwurfsklassendiagramm.

# Kapitel 4

## Skizze der GUI

### 4.1 Registrierung- und Loginseite

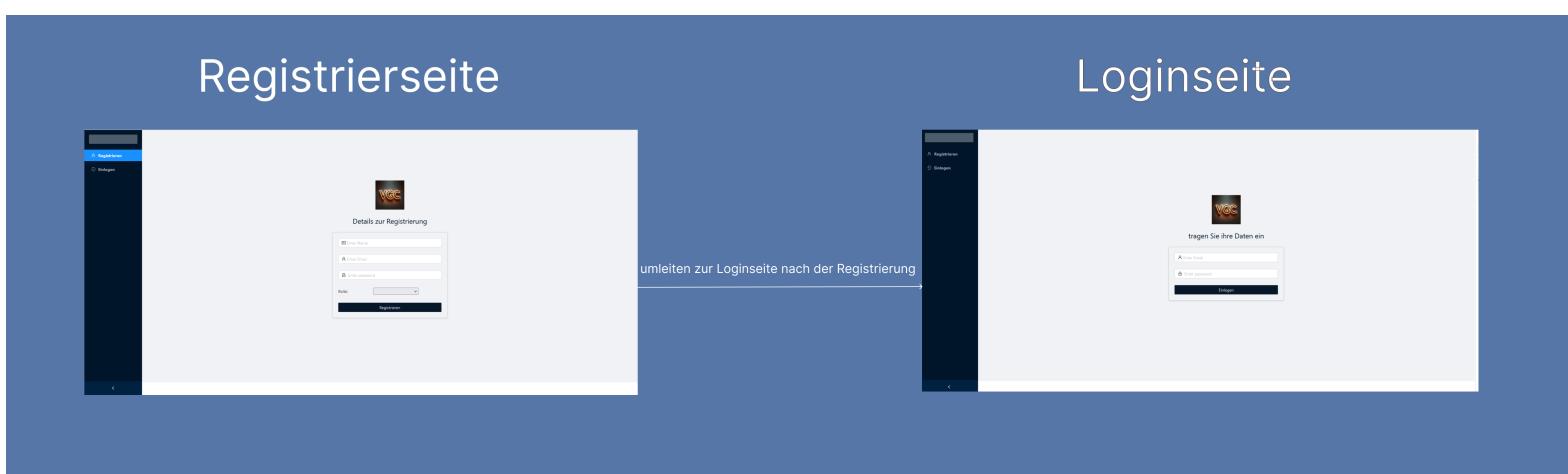


Abbildung 4.1: “Registrierung- und Loginseite”.

## 4.2 Lagerhalter-Funktionalitäten

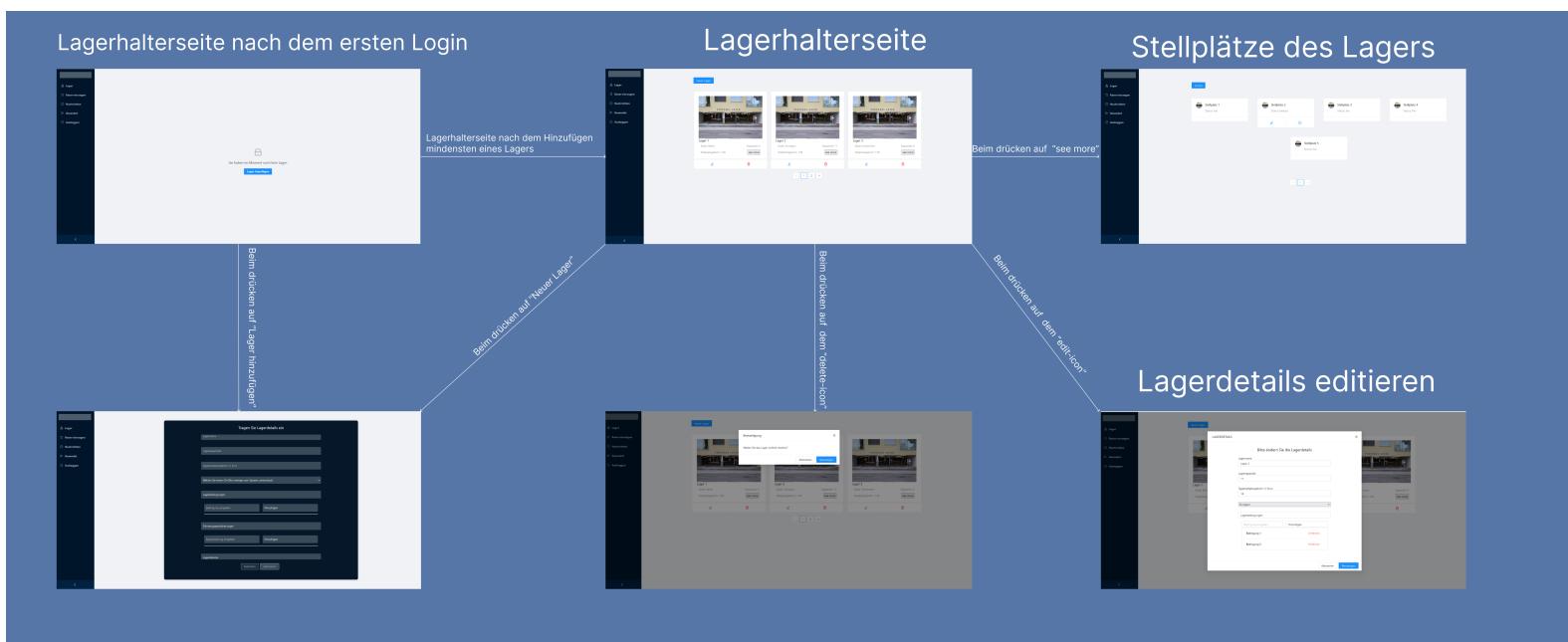


Abbildung 4.2: "Lagerhalter-Funktionalitäten".

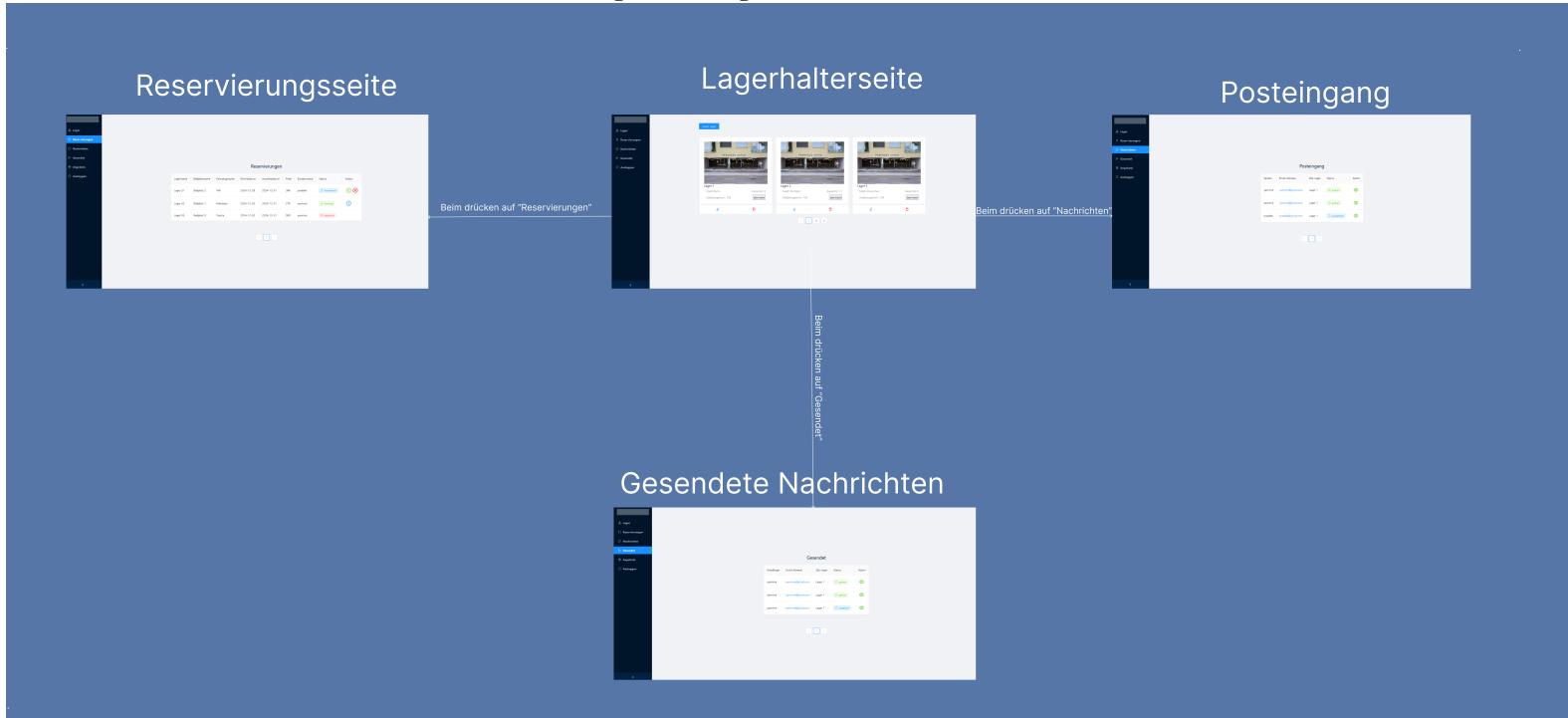


Abbildung 4.3: "Lagerhalter-Funktionalitäten".

## 4.3 Einlagerer-Funktionalitäten

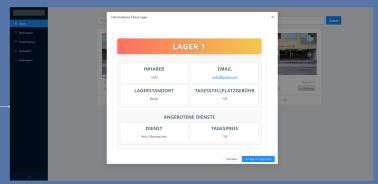
Stellplatz buchen



Einlagererseite



Lagerinformationen



Buchungen



Posteingang



Gesendete Nachrichten

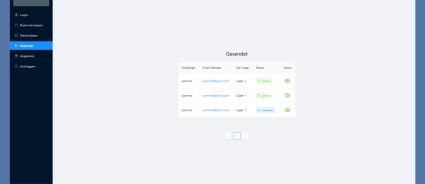


Abbildung 4.4: "Einlagerer-Funktionalitäten".

## 4.4 Werkstattinhaber-Funktionalitäten

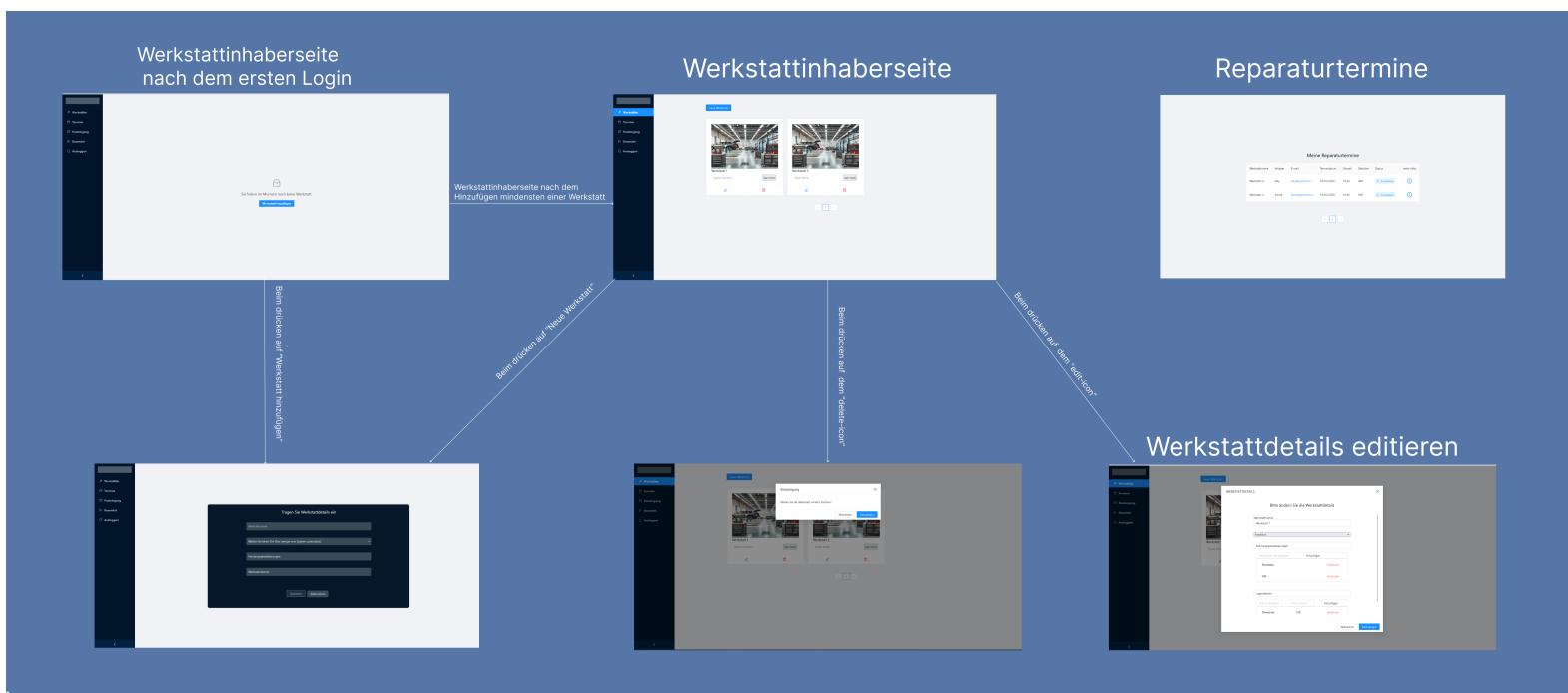


Abbildung 4.5: “Werkstattinhaber-Funktionalitäten”.

# Kapitel 5

## Annahme und bewusste Abweichungen von der OOD-Methodik

### 5.1 Annahmen

Annahmen sind Annahmen über das System, die während der Konzeption getroffen wurden, um die Analyse und das Design zu erleichtern.

#### 5.1.1 Technische Einschränkungen

- Das System wird zunächst nur für ausgewählte deutsche Städte bereitgestellt
- BenutzerLogin erfolgt ausschließlich über E-Mail und Passwort.
- Es wird vorausgesetzt, dass die Nutzer Zugriff auf stabile Internetverbindungen haben.
- Es gibt keine Integration mit externen Zahlungssystemen in der aktuellen Version.

#### 5.1.2 Funktionale Annahmen

- Jeder Benutzer kann mehrere Rollen besitzen (z. B. Lagerhalter und Einlagerer), jedoch nicht gleichzeitig aktiv nutzen.
- Die Funktionalität zum Anzeigen von Stellplätzen ist auf freie Stellplätze beschränkt; belegte Plätze werden standardmäßig ausgeblendet.
- Services, die von einem Lagerhalter angeboten werden, sind nicht dynamisch konfigurierbar und basieren auf einer vordefinierten Liste.
- Buchungen können nur durch registrierte Einlagerer durchgeführt werden.

#### 5.1.3 Architekturentscheidungen

- Es wurde angenommen, dass die Datenbank synchron arbeitet und jederzeit verfügbar ist, um Echtzeitanfragen zu beantworten (z. B. Lagerplatzsuche, Fahrzeugstatusabfragen).
- Sicherheitsmaßnahmen wie Verschlüsselung von Passwörtern sind vorhanden.
- Es wird eine einfache Fehlerbehandlung implementiert, die generische Fehlermeldungen an den Benutzer zurückgibt.
- Die Datenbank speichert nur aktuelle Daten; historische Daten werden nicht erfasst.

## **5.2 Bewusste Abweichungen von der OOD-Methodik**

Die Abweichungen sind gezielte Entscheidungen, die von der Standard-OOD-Methodik abweichen, um das Projekt effizienter zu gestalten.

### **5.2.1 Reduktion der Komplexität**

- Die Klasse "Lager" wurde so erweitert, dass sie sowohl die Lagerdaten als auch die zugehörigen Bedingungen enthält, um die Anzahl der separaten Klassen zu reduzieren.

### **5.2.2 Keine vollständige Implementierung aller Patterns**

- Das Controller-Pattern wurde vereinfacht: Der gleiche Controller verarbeitet mehrere ähnliche Anfragen (z. B. Registrierung und Login).
- Das System implementiert eine separate Validierungsklasse; Validierungen werden nicht direkt im Controller durchgeführt.
- Der Einsatz von Design Patterns wurde auf die wesentlichen reduziert, um die Entwicklungszeit zu optimieren.

### **5.2.3 Zusammenführung von Konzepten**

- Die Benutzerrollenverwaltung ist in die Hauptbenutzerklasse integriert, anstatt eine separate Klasse zu verwenden.

### **5.2.4 Abweichung bei der Fehlerbehandlung**

- Fehler werden dem Benutzer durch generische Nachrichten angezeigt, ohne spezifische Debug-Informationen offenzulegen.
- Es gibt keine automatische Wiederherstellung bei kritischen Fehlern; der Benutzer muss die Aktion erneut versuchen.
- Bei unerwarteten Fehlern wird der Benutzer auf die Startseite weitergeleitet.