

Verteilte Systeme

**im Sommersemester 2024
an der Hochschule Osnabrück**

**Dozent:
Prof. Dr. Heinz-Josef Eikerling**

Projektbericht

CarConnect

erstellt von

Autor: Moahammed Al-Ozair, Matrikel-Nummer:1006657, mohammed.al-ozair@hs-osnabrueck.de
Autor: Nabeel Elamaireh, Matrikel-Nummer:970217, nabeel.elamaireh@hs-osnabrueck.de

Datum: 17.08.2024

Zusammenfassung

In dieser Arbeit wird die Entwicklung des CarConnect-Systems beschrieben, einer Plattform zur Verwaltung und Buchung von Fahrzeugen. Das System wurde unter Verwendung moderner Technologien konzipiert, wobei RMI (Remote Method Invocation) zur Server-Kommunikation und JWT (JSON Web Token) für die Authentifizierung eingesetzt wurden, um eine hohe Sicherheit und Skalierbarkeit zu gewährleisten. Die Echtzeitinteraktion der Benutzer wurde durch die Integration von WebSocket ermöglicht, was eine effiziente und verzögerungsfreie Kommunikation sicherstellt. Im Rahmen dieser Arbeit wurden die Ziele und die Planung des Projekts detailliert dargelegt sowie die eingesetzten Technologien umfassend erläutert. Die Ergebnisse der Implementierung werden präsentiert und in Bezug auf die ursprünglichen Anforderungen diskutiert, wobei insbesondere die Leistungsfähigkeit und Effizienz des Systems hervorgehoben werden.

Inhalt

1	Einleitung	5
2	Konzept	8
2.1	Authentifizierung.....	8
2.2	Fahrzeugverwaltung	10
2.3	Buchungssystem.....	10
2.4	Benachrichtigungssystem.....	11
2.5	Benutzeroberfläche (UI/UX)	11
3	Schnittstellen und Implementierung	12
4	Installation und Test	17
4.1	Installation.....	17
4.2	Test	17
5	Fazit & Ausblick	19
6	Literaturverzeichnis	20

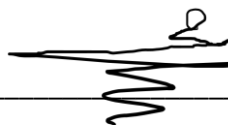
Erklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbstständig und ohne Verwendung anderer als die angegebenen Hilfsmittel angefertigt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Osnabrück, den 17. August 2024

Nabeel Elamaireh

Mohammed Al-Ozair



(Unterschriften der Autoren)

Arbeitsaufteilung

<i>Autor</i>	<i>Projektbeitrag</i>
Mohammed Al-ozair, Nabeel Elamaireh	Implementierung des Authentifizierungssystems, einschließlich der Methoden zur Registrierung und Benutzervalidierung.
Nabeel Elamaireh	Entwicklung der Backend-logik für die Fahrzeugverwaltung, einschließlich der Methoden zur Registrierung und Aktualisierung.
Mohammed Al-ozair	Integration von RMI zur Kommunikation zwischen den verschiedenen Serverkomponenten.
Nabeel Elamaireh	Implementierung des Buchungssystems, das Benutzer die Suche und Buchung von Fahrzeugen ermöglicht.
Mohammed Al-ozair	Implementierung der Benutzeroberfläche (UI) für die Registrierung und Anmeldung, einschließlich des Handling von JWT-Tokens.
Nabeel Elamaireh	Entwicklung des Benachrichtigungssystem, das Websockets für die Echtzeitkommunikation verwendet.
Mohammed Al-ozair	Integration der Datenbank (PostgreSQL) und Implementierung der Datenbankverbindungen, einschließlich der Nutzerdatenverwaltung.
Nabeel Elamaireh	Durchführung von umfangreichen Tests, einschließlich der Validierung von Authentifizierung, Buchung und Benachrichtigungssystemen.
Mohammed Al-ozair, Nabeel Elamaireh	Gemeinsame Planung und Design der Systemarchitektur, einschließlich der Verteilung der Verantwortlichkeiten zwischen den Serverkomponenten.
Mohammed Al-ozair	Verfassung der Kapitels: 1 Einleitung, 2 Konzept, 2.1 Authentifizierung, 2.3 Buchungssystem
Nabeel Elamaireh	Verfassung der Kapitels: 2.2 Fahrzeugverwaltung, 2.4 Benachrichtigungssystem, 2.5 Benutzeroberfläche (UI/UX)
Mohammed Al-ozair, Nabeel Elamaireh	Verfassung der Kapitels: 3 Schnittstellen und Implementierung, 5 Fazit & Ausblick

Abkürzungsverzeichnis

CSS Cascading Style Sheets

EE Enterprise Edition

http Hypertext Transfer Protocol

IoT Internet of Things

JWT JSON Web Tokens

RMI (Remote Method Invocation)

UI User Interface

UX User Experience

1 Einleitung

Die vorliegende Hausarbeit befasst sich mit der Entwicklung und Implementierung des CarConnect-Systems, einer innovativen Plattform zur Verwaltung und Buchung von Fahrzeugen. Ziel dieser Arbeit ist es, eine benutzerfreundliche und effiziente Lösung zu schaffen, die es Fahrzeugbesitzern ermöglicht, ihre ungenutzten Fahrzeuge freiwillig zur Verfügung zu stellen, um anderen zu helfen. Das CarConnect-System zielt darauf ab, moderne Technologien wie WebSocket und RMI zu nutzen, um eine sichere, skalierbare und intuitive Plattform zu bieten. Das System wurde konzipiert, um die Kommunikation zwischen Nutzern in Echtzeit zu ermöglichen und gleichzeitig eine zuverlässige Authentifizierung zu gewährleisten. Hierfür werden verschiedene Sicherheitsmechanismen integriert, die den Schutz der Benutzerdaten sicherstellen sollen. Die Skalierbarkeit des Systems ist ein zentraler Aspekt, um eine hohe Leistung auch bei einer wachsenden Zahl von Nutzern zu gewährleisten. Diese Arbeit beschreibt detailliert den Entwicklungsprozess von der Konzeption bis zur Implementierung und beleuchtet dabei besondere Herausforderungen sowie deren Lösungen.

Durch den Einsatz moderner Technologien und eine durchdachte Software-Architektur wird gezeigt, wie ein innovatives Fahrzeugmanagementsystem entwickelt werden kann, das nicht nur technische Effizienz, sondern auch gesellschaftlichen Mehrwert bietet. Die Implementierung des CarConnect-Systems stellt einen bedeutenden Fortschritt in der Entwicklung von Plattformen zur Fahrzeugverwaltung dar, die den Bedürfnissen und Anforderungen moderner Nutzer gerecht werden.

1.1 Projektziel

Das Hauptziel dieses Projekts besteht in der Entwicklung eines Systems zur effizienten Verwaltung von Fahrzeugdaten. Das System soll eine zentrale Plattform bereitstellen, die es ermöglicht, Daten über Fahrzeugnutzung zu erfassen, zu speichern und zu analysieren. Es wird angestrebt, durch die Implementierung fortschrittlicher Technologien wie WebSocket für die Echtzeitkommunikation und robuste Datenmanagement-Techniken eine zuverlässige und benutzerfreundliche Lösung anzubieten. Die Hauptaugenmerke liegen auf der Sicherstellung einer hohen Datenintegrität, einer effizienten Performance und einer benutzerfreundlichen Schnittstelle.

1.2 Systemkonzept

Das Systemkonzept der CarConnect-Anwendung basiert auf einer verteilten Architektur, die verschiedene Softwarekomponenten und -technologien integriert. Das Backend der Anwendung wird durch Java EE realisiert und umfasst die Implementierung von HTTP-Servlets zur effizienten Verarbeitung von Benutzeranfragen sowie RMI zur Kommunikation zwischen den Serverkomponenten. Die Verwendung von HTTP-Servlets ermöglicht dabei eine zuverlässige und sichere Authentifizierung zwischen dem Server und den Clientanwendungen. Dies ist besonders wichtig, da der Authentifizierungsserver in ständiger Kommunikation mit den anderen Serverkomponenten steht, um die Identität der Benutzer zu überprüfen und einen sicheren Zugriff auf die verschiedenen Funktionen des Systems zu gewährleisten [OSer].

Für die Echtzeitkommunikation zwischen den Benutzern und dem System wurde WebSocket-Technologie integriert. Diese ermöglicht eine sofortige und bidirektionale Kommunikation, wodurch Updates, wie die Bestätigung oder Ablehnung von Buchungsanfragen und Änderungen der Fahrzeugverfügbarkeit, direkt und ohne Verzögerung an die Benutzer gesendet werden können. Durch die Kombination dieser bewährten Technologien wird sichergestellt, dass das CarConnect-System

sowohl in Bezug auf Zuverlässigkeit als auch auf Skalierbarkeit und Benutzerfreundlichkeit den Anforderungen gerecht wird.

1.3 Anwendungsbereich

Das CarConnect-System ist für den Einsatz in verschiedenen Umgebungen konzipiert. Es kann von Einzelpersonen genutzt werden, die ihre Fahrzeuge temporär zur Verfügung stellen möchten, sowie von Gemeinschaften oder Organisationen, die eine gemeinsame Nutzung von Fahrzeugen fördern wollen. Darüber hinaus eignet sich die Plattform für den Einsatz in urbanen Gebieten, wo der Bedarf an flexiblen und kurzfristigen Mobilitätslösungen besonders hoch ist. Durch die Bereitstellung einer solchen Plattform wird die effiziente Nutzung von Fahrzeugen gefördert und gleichzeitig ein Beitrag zur nachhaltigen Mobilität geleistet.

1.4 Zielgruppen

Die Hauptzielgruppen des CarConnect-Systems sind Privatpersonen, die ihre ungenutzten Fahrzeuge anderen zur Verfügung stellen möchten, sowie Personen, die temporär ein Fahrzeug benötigen. Darüber hinaus richtet sich die Plattform an Gemeinschaften und Organisationen, die die gemeinsame Nutzung von Fahrzeugen fördern möchten. Auch Unternehmen, die ihren Fuhrpark effizienter nutzen wollen, können von der Plattform profitieren. Durch die benutzerfreundliche Gestaltung und die intuitive Bedienung ist die Plattform für eine breite Zielgruppe attraktiv und leicht zugänglich.

1.5 Funktionale Anforderungen

Funktionale Anforderungen definieren die Aktionen, die ein System ausführen soll, um die Erwartungen der definierten Zielgruppen zu erfüllen. Im Rahmen dieses Projekts umfassen die funktionalen Anforderungen verschiedene Schlüsselaktionen und Prozesse, die die Anwendung ermöglichen. Diese Anforderungen werden gemäß der Anforderungsschablone von Rupp [RS14] dargestellt.

A1.1: Wenn Benutzer nicht eingeloggt sind, sollte das System ihnen die Option zur Anmeldung bieten.

A1.2: Bevor sich Benutzer anmelden, sollte das System ihnen die Option zur Registrierung bieten.

A1.3: Sobald Benutzer ihre Login-Daten eingegeben haben, sollte das System fähig sein, einen Server zur Authentifizierung zu verwenden.

A1.4: Wenn die Login-Daten der Benutzer falsch sind, sollte das System eine Fehlermeldung anzeigen.

A1.5: Wenn bei der Registrierung ein bereits existierender Benutzername gewählt wird, sollte das System eine entsprechende Nachricht anzeigen.

A1.6: Nach erfolgreicher Validierung muss das System ein Sitzungstoken erstellen und es an den Client senden.

A1.7: Nach erfolgreicher Anmeldung muss das System fähig sein, die Startseite an die Nutzer zu liefern.

A2.1: Bei einer Anfrage nach den aktuellen Fahrzeugen muss das System die vorhandenen Fahrzeuge laden.

A2.2: Das System muss den Benutzer die Möglichkeit bieten, ein neues Fahrzeug zu registrieren.

A2.3: Beim Erstellen eines neuen Fahrzeugs wird das System das neue Fahrzeug in der Fahrzeugliste anzeigen.

A2.4: Das System muss den Benutzer die Möglichkeit bieten, ein Fahrzeug zu bearbeiten oder zu löschen.

A3.1: Wenn ein Benutzer ein Fahrzeug bucht, muss das System die Buchungsanfrage verarbeiten und eine Bestätigung senden.

A3.2: Das System muss die Verfügbarkeit der Fahrzeuge aktualisieren und anzeigen.

A3.3: Das System muss den Benutzer die Möglichkeit bieten, Buchungen zu stornieren.

A3.4: Das System muss Benachrichtigungen über Buchungsänderungen und Systemupdates in Echtzeit bereitstellen.

1.6 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beziehen sich auf die allgemeinen Eigenschaften und Beschränkungen eines Systems, die dessen Leistung, Zuverlässigkeit, Benutzerfreundlichkeit und andere Aspekte beeinflussen. Hier sind einige Beispiele nach [Kle18] für nichtfunktionale Anforderungen, die für dieses Projekt berücksichtigt werden:

Speicher- und Laufzeiteffizienz: Das System soll mindestens 1000 gleichzeitige Benutzer unterstützen, ohne dass Verzögerungen von mehr als 100 Millisekunden auftreten. Dies wird durch regelmäßige Leistungstests unter Volllast überprüft.

Zuverlässigkeit: Das System soll eine Verfügbarkeit von mindestens 99,9% über einen Zeitraum von einem Monat aufweisen. Ausfallzeiten aufgrund von Serverproblemen oder Softwarefehlern sollten maximal 30 Minuten pro Monat betragen.

Benutzbarkeit: Die durchschnittliche Zeit, die ein neuer Benutzer benötigt, um sich im System zurechtzufinden und seine erste Buchung vorzunehmen, sollte unter fünf Minuten liegen. Dies wird durch Benutzertests und Bewertungen der Benutzeroberfläche gemessen.

Skalierbarkeit: Das System soll in der Lage sein, die Anzahl der unterstützten Benutzer linear zu erhöhen, ohne dass die Antwortzeiten des Servers um mehr als das Doppelte steigen. Dies wird durch Skalierungstests unter realistischen Lastbedingungen überprüft.

Kompatibilität: Die Anwendung soll auf den aktuellen Versionen der gängigsten Webbrowser (Chrome, Firefox, Safari) sowie auf den Betriebssystemen Windows und MacOS lauffähig sein.

2 Konzept

Das Konzept der Software- und Verteilungsarchitektur für CarConnect basiert auf einer modularen Struktur, die verschiedene Komponenten und Technologien integriert, um eine skalierbare und zuverlässige Anwendung zu gewährleisten. Im Folgenden werden die Hauptkomponenten des Systems und deren Funktionen detailliert beschrieben.

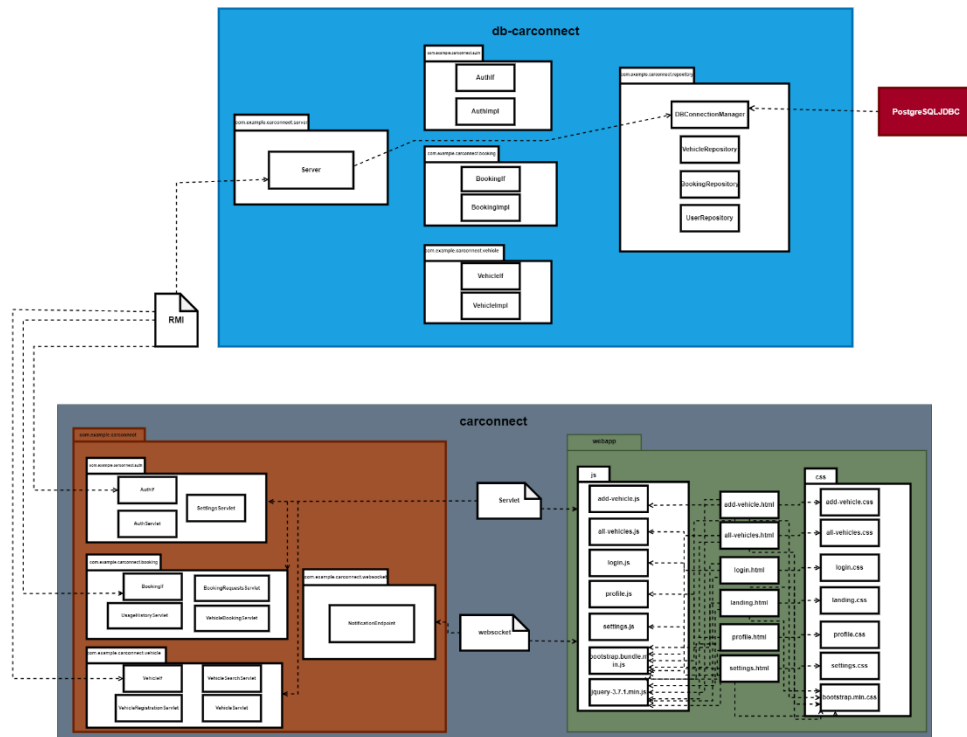


Abbildung 1: Klassendiagramm Back-und Frontend

2.1 Authentifizierung

Die Authentifizierung in der CarConnect-Anwendung ist von zentraler Bedeutung, um die Sicherheit und Integrität der Benutzerdaten zu gewährleisten. Der Authentifizierungsprozess wird in zwei Hauptkomponenten unterteilt: die Nutzung von Remote Method Invocation (RMI) zur Verbindung des Authentifizierungsservers mit dem Hauptserver und die Verwendung von (JWT) zur fortlaufenden Sicherstellung der Benutzerintegrität.

Der Authentifizierungsprozess beginnt, wenn die Benutzer ihre Anmeldedaten (Benutzername und Passwort) in einem Formular eingeben. Diese sensiblen Daten werden durch den SHA256-Algorithmus gehasht, um sicherzustellen, dass die Passwörter weder während der Übertragung noch in der Datenbank im Klartext vorliegen. Die gehashten Anmeldedaten werden dann per HTTP-Post an den Authentifizierungsendpunkt des Servers gesendet. Ein dahinterliegendes Servlet extrahiert die Daten aus dem HTTP-Request und initiiert eine Authentifizierungssitzung beim per RMI angeschlossenen Authentifizierungsserver.

Der Authentifizierungsserver generiert eine eindeutige Sitzungsnummer für den Authentifizierungsversuch und übermittelt diese dem Hauptserver. Anschließend führt der Hauptserver eine weitere Hash-Berechnung durch, die den Sitzungsschlüssel und den ursprünglichen Hash kombiniert. Diese Informationen werden zur Validierung an den Authentifizierungsserver gesendet, der die Benutzerdaten aus der Datenbank abrufen und den berechneten Hash mit dem gespeicherten Hash vergleicht.

Im Erfolgsfall meldet der Authentifizierungsserver dem Hauptserver, dass die Authentifizierung erfolgreich war. Daraufhin wird ein JWT generiert und an den Client übermittelt. Nachfolgende Anfragen werden basierend auf der Gültigkeit dieses Tokens behandelt, was die Notwendigkeit wiederholter Authentifizierungsanfragen eliminiert, und die Effizienz des Systems steigert.

Die Verwendung von RMI ermöglicht eine zentrale Authentifizierungsquelle, die auch von anderen Diensten genutzt werden kann, um auf dieselben Authentifizierungsdaten zuzugreifen. Dies reduziert die Last auf dem Authentifizierungsserver und beschleunigt die Bearbeitung nachfolgender Anfragen. Zudem gewährleistet die Verwendung von JWT, dass nur authentifizierte Benutzer auf die geschützten Bereiche der Anwendung zugreifen können.

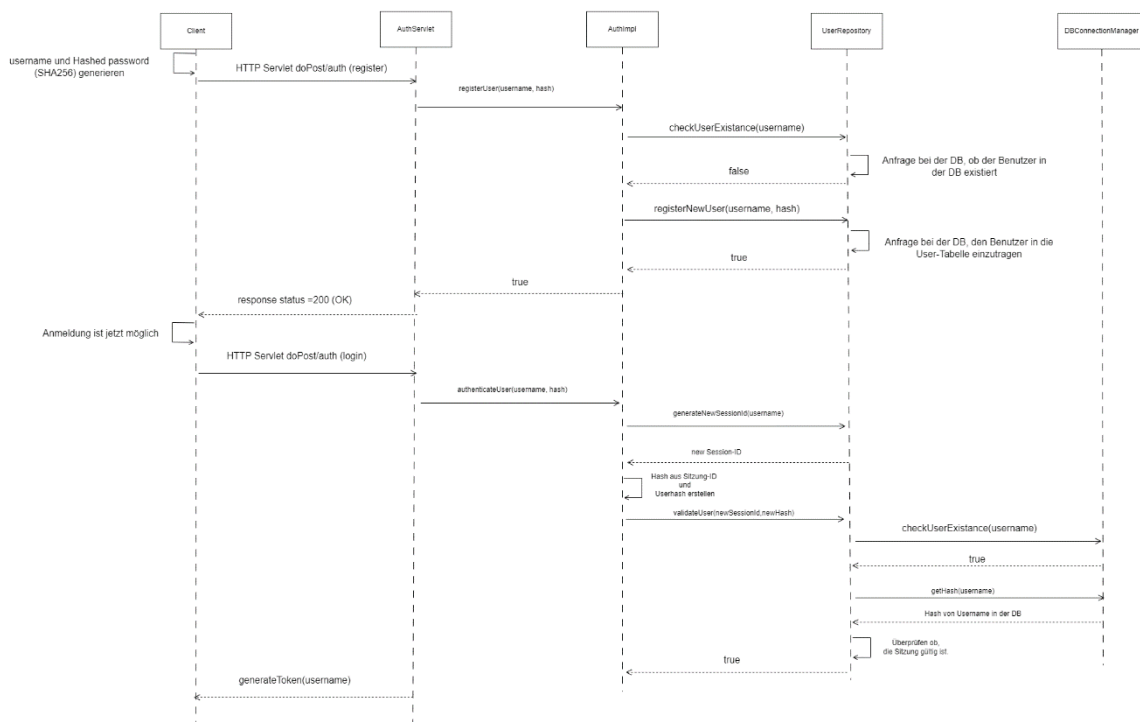


Abbildung 2: Sequenzdiagramm

2.2 Fahrzeugverwaltung

Die Fahrzeugverwaltung in der CarConnect-Anwendung umfasst die Registrierung, Aktualisierung, Löschung und Suche von Fahrzeugen. Diese Funktionen wurden entwickelt, um eine benutzerfreundliche und effiziente Verwaltung der Fahrzeugdaten zu ermöglichen.

Die Registrierung eines neuen Fahrzeugs erfolgt durch die Eingabe relevanter Informationen wie Marke, Modell, Baujahr und Standort. Diese Daten werden über ein Formular gesammelt und an den Server übermittelt, der die Eingaben validiert und in der Datenbank speichert. Die Aktualisierung der Fahrzeugdaten ermöglicht es den Benutzer, bestehende Informationen zu ändern und zu speichern. Der Server prüft die Eingaben und aktualisiert die Datenbank entsprechend.

Die Löschung eines Fahrzeugs entfernt alle zugehörigen Daten aus dem System. Der Server stellt sicher, dass die Daten vollständig und korrekt gelöscht werden, um Inkonsistenzen zu vermeiden. Die Suchfunktion ermöglicht es den Benutzer, Fahrzeuge nach verschiedenen Kriterien wie Marke, Modell und Baujahr zu durchsuchen. Die Suchergebnisse werden in einer übersichtlichen Tabelle angezeigt, die eine einfache Navigation und Auswahl ermöglicht.

Technologisch wird die Verwaltung der Fahrzeugdaten mittels Java EE und HTTP-Servlets umgesetzt, die die Anfragen verarbeiten und die Datenbank verwalten. Die Benutzeroberfläche wird mit HTML, CSS und JavaScript gestaltet, um eine intuitive und reaktionsschnelle Interaktion zu gewährleisten.

2.3 Buchungssystem

Das Buchungssystem von CarConnect wurde entwickelt, um die Echtzeitbuchung von Fahrzeugen zu ermöglichen. Es bietet Funktionen zur Buchung, Überprüfung der Verfügbarkeit, Verwaltung von Stornierungen sowie zur Genehmigung oder Ablehnung von Buchungsanfragen durch den Fahrzeugbesitzer.

Der Buchungsprozess beginnt mit der Auswahl eines Fahrzeugs und der Eingabe des gewünschten Buchungszeitraums durch den Benutzer. Diese Informationen werden an den Server gesendet, der die Verfügbarkeit des Fahrzeugs für den angegebenen Zeitraum überprüft. Nach der Überprüfung erhält der Fahrzeugbesitzer eine Benachrichtigung über die Buchungsanfrage und kann diese entweder bestätigen oder ablehnen. Bei einer Bestätigung wird die Buchung in der Datenbank gespeichert, und die Verfügbarkeit des Fahrzeugs wird entsprechend aktualisiert.

Der Fahrzeugbesitzer hat auch die Möglichkeit, seine Fahrzeugangebote zu verwalten, indem er bestehende Buchungen bearbeiten oder löschen kann. Bei einer Bearbeitung oder Löschung werden die entsprechenden Daten in der Datenbank aktualisiert und die Verfügbarkeit des Fahrzeugs wird neu berechnet.

Die technologische Umsetzung des Buchungssystems erfolgt durch die Verwendung von RMI (Remote Method Invocation) zur Kommunikation zwischen den Serverkomponenten. Dies ermöglicht eine effiziente Verarbeitung von Buchungsanfragen und die dynamische Aktualisierung der Fahrzeugverfügbarkeit. Die Benutzeroberfläche wird mit JavaScript gestaltet, um eine reibungslose und interaktive Benutzererfahrung zu gewährleisten, die eine Echtzeit-Interaktion mit dem Backend ermöglicht.

2.4 Benachrichtigungssystem

Das Benachrichtigungssystem von CarConnect ist darauf ausgelegt, Benutzer in Echtzeit über wichtige Ereignisse und Aktionen zu informieren. Es integriert sich nahtlos in die anderen Systemkomponenten und stellt sicher, dass die Benutzer sofortige Rückmeldungen erhalten, beispielsweise bei der Bestätigung oder Ablehnung von Buchungsanfragen, der Änderung der Fahrzeugverfügbarkeit oder anderen wichtigen Systemereignissen.

Technologisch wird das Benachrichtigungssystem durch WebSockets implementiert, die eine bidirektionale Kommunikation zwischen dem Server und den Clients ermöglichen. Dadurch können Updates ohne manuelle Seitenaktualisierung direkt an die Benutzeroberfläche gesendet werden. Das System stellt sicher, dass alle relevanten Parteien (z.B. Fahrzeugbesitzer und Buchende) über den Status ihrer Aktionen informiert bleiben, wodurch die Benutzererfahrung verbessert und Verzögerungen minimiert werden.

Die Integration der WebSocket-Technologie in das CarConnect-Benachrichtigungssystem ermöglicht eine Echtzeitkommunikation zwischen dem Server und den Clients. Diese Technologie erlaubt es, dynamische und interaktive Benutzererfahrungen zu schaffen, indem Informationen wie die Genehmigung oder Ablehnung von Buchungsanfragen, Änderungen der Fahrzeugverfügbarkeit und andere wichtige Ereignisse unmittelbar an die Benutzeroberfläche übermittelt werden. Durch die WebSockets können Benachrichtigungen ohne Verzögerung und ohne manuelle Seitenaktualisierung direkt an die betroffenen Benutzer gesendet werden. Dies verbessert nicht nur die Benutzerfreundlichkeit, sondern sorgt auch dafür, dass alle relevanten Informationen in Echtzeit bereitgestellt werden, was für die effiziente Nutzung des CarConnect-Systems unerlässlich ist [Fe11].

2.5 Benutzeroberfläche (UI/UX)

Die Benutzeroberfläche (UI) von CarConnect wurde entwickelt, um eine benutzerfreundliche und zugängliche Erfahrung (UX) zu bieten. Das Design folgt den Prinzipien der Klarheit, Konsistenz und Reaktionsfähigkeit. Es ermöglicht Benutzern, alle wesentlichen Funktionen schnell und einfach zu finden. Die Oberfläche passt sich nahtlos an verschiedene Geräte und Bildschirmgrößen an und ist barrierefrei gestaltet. Für die technologische Umsetzung werden HTML5, CSS3, JavaScript, jQuery und Bootstrap genutzt. Diese Technologien ermöglichen eine dynamische und responsive Benutzeroberfläche, die eine reibungslose Interaktion mit dem Backend gewährleistet.

Die Hauptkomponenten der UI sind das Dashboard, die Buchungsseite, das Benachrichtigungspanel und die Einstellungen. Das Dashboard bietet einen zentralen Überblick über alle wichtigen Funktionen, während die Buchungsseite eine einfache Suche und Buchung von Fahrzeugen ermöglicht. Das Benachrichtigungspanel zeigt Echtzeit-Updates, und die Einstellungen ermöglichen die Anpassung von Profilinformationen und Benachrichtigungen.

Der Benutzerfluss wurde so gestaltet, dass alle wichtigen Aktionen effizient durchgeführt werden können. Nach jeder Aktion erhält der Benutzer sofortiges Feedback, was eine nahtlose und zufriedenstellende Benutzererfahrung gewährleistet.

3 Schnittstellen und Implementierung

Nach Abschluss der Phasen der Anforderungsanalyse, des Designs und der Planung begann die Implementierungsphase des CarConnect-Systems. Diese Phase umfasste die tatsächliche Entwicklung der Software, in der die zuvor definierten Anforderungen und Entwürfe in funktionalen Code umgesetzt wurden. In den folgenden Abschnitten wird die Implementierung des Projekts detailliert beschrieben, einschließlich der verwendeten Technologien und Methoden sowie der Herausforderungen und Lösungen, die während dieser Phase auftraten.

Die Implementierung begann mit der Entwicklung des Authentifizierungssystems, das als Grundlage für die restlichen Funktionalitäten des Systems dient. Zunächst wurde das Authentifizierungs-Interface entwickelt, welches im Backend des Systems implementiert ist und grundlegende Methoden zur Verwaltung von Benutzersitzungen bereitstellt. Wie in Snippet 1 zu sehen, ist die zentrale Methode, „generateNewSessionId“, die die Generierung einer neuen Sitzungsschlüssel übernimmt, die einem Benutzernamen zugeordnet wird. Dieser Sitzungsschlüssel wird für die nachfolgende Validierung von Benutzern verwendet.

```
package com.example.carconnect.auth;

@Override
public long generateNewSessionId(String username) {
    for (Map.Entry<Long, String> entry : sessionToUserMap.entrySet()) {
        if (entry.getValue().equals(username)) {
            return entry.getKey();
        }
    }
    long newSessionId = new Date().getTime();
    sessionToUserMap.put(newSessionId, username);
    return newSessionId;
}
```

Snippet 1: Methode zur Generierung einer neuen Sitzungsschlüssel

Eine weitere zentrale Methode des Authentifizierungssystems ist die validateUser-Methode, die für die Überprüfung der Authentizität eines Benutzers verantwortlich ist. Diese Methode nutzt den zuvor generierten Sitzungsschlüssel und einen Hash-Wert, um den Benutzer zu authentifizieren. Wie im Snippet 2 gezeigt, wird zunächst der Benutzername anhand der Sitzungsschlüssel extrahiert und überprüft, ob dieser in der Benutzerdatenbank existiert. Anschließend wird ein neuer Hash-Wert generiert, indem der Sitzungsschlüssel mit dem in der Datenbank gespeicherten Benutzer-Hash kombiniert wird. Dieser neue Hash wird schließlich mit dem vom Benutzer übergebenen Hash verglichen, um die Gültigkeit der Sitzung zu bestätigen. Bei einer erfolgreichen Validierung wird die Sitzung als authentisch betrachtet, andernfalls wird der Zugriff verweigert.

```

package com.example.carconnect.auth;
@Override
public boolean validateUser(long sessionId, String hash) {
    String username = sessionToUserMap.get(sessionId);
    System.out.println("Validating user: " + username);
    if (username == null) {
        System.out.println("Username not found for session ID: " + ses-
sessionId);
        return false;
    }
    if (!userRepository.checkUserExistence(username)) {
        System.out.println("User does not exist: " + username);
        return false;
    }
    String userHash = userRepository.getHash(username);
    if (userHash == null) {
        System.out.println("Hash not found for user: " + username);
        return false;
    }
    try {
        MessageDigest digester = MessageDigest.getInstance("SHA-256");
        byte[] encodedHash = digester.digest((sessionId + userHash).get-
Bytes(StandardCharsets.UTF_8));
        String newHash = bytesToHex(encodedHash);
        sessionToUserMap.remove(sessionId Validierung
        System.out.println("Expected hash: " + newHash + ", Received
hash: " + hash);
        return newHash.equals(hash);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace(); return false;
    }
}

```

Snippet 2: Methode zur Validierung eines Benutzers

Die registerUser-Methode schließlich dient der sicheren Registrierung neuer Benutzer. Diese Methode ist so gestaltet, dass sie zunächst überprüft, ob der gewünschte Benutzername bereits in der Datenbank vorhanden ist. Ist dies der Fall, wird die Registrierung abgelehnt. Andernfalls wird der Benutzername zusammen mit einem sicheren Passwort-Hash in der Datenbank gespeichert. Diese Authentifizierungsmethoden sind von zentraler Bedeutung für die Sicherheit und Integrität des CarConnect-Systems. Sie gewährleisten, dass nur autorisierte Benutzer auf die Funktionen der Plattform zugreifen können und dass alle Benutzerdaten sicher und zuverlässig verarbeitet werden. Der entsprechende Code ist im Snippet 3 dargestellt.

```

package com.example.carconnect.auth;
@Override
public boolean registerUser(String username, String hash) {
    System.out.println("Registering user: " + username);
    boolean userExists = userRepository.checkUserExistence(username);
    System.out.println("User exists: " + userExists);
    if (userExists) {
        return false;
    }
    boolean registrationResult = userRepository.regis-
terNewUser(username, hash);
    System.out.println("Registration result: " + registrationResult);
    return registrationResult;
}

```

Snippet 3: Methode zur Registrierung eines neuen Benutzers

Im Frontend wurde die Authentifizierung durch die Entwicklung eines entsprechenden Formulars in der login.html realisiert. Bei der Eingabe von Benutzerdaten und dem Absenden des Formulars wird eine HTTP-Post-Anfrage an das „/auth“ AuthServlet gesendet. Das Servlet verarbeitet die Anfrage, indem es den Benutzernamen und den Passwort-Hash extrahiert und die Authentifizierung entweder durch Anmeldung oder Registrierung durchführt. Bei erfolgreicher Authentifizierung wird ein JWT generiert und an den Benutzer zurückgesendet. Wie im Snippet 4 zu sehen ist, wird in der doPost-Methode zunächst die Aktion „register“ oder „login“ identifiziert und die entsprechenden Schritte eingeleitet, um entweder die Registrierung durchzuführen oder das Token nach einer erfolgreichen Anmeldung zu erstellen.

```
@Override

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String action = request.getParameter("action");
    try {
        if ("register".equals(action)) {
            String username = request.getParameter("username");
            String password = request.getParameter("password");
            boolean registerResult = authIf.registerUser(username,
hashPassword(password));
            jsonResponse.put("success", registerResult);
        } else if ("login".equals(action)) {
            // ... (Anmeldeprozess)
            String token = generateToken(username);
            jsonResponse.put("token", token); }
    } catch (RemoteException | NoSuchAlgorithmException e) {
        jsonResponse.put("error", e.getMessage());}
    response.getWriter().write(jsonResponse.toString());}
```

Snippet 4: doPost-Methode im AuthServlet

Nach der erfolgreichen Authentifizierung erhält der Benutzer ein JWT, das als Identitätsnachweis bei zukünftigen Anfragen dient. Das JWT wird im lokalen Speicher (Local Storage) des Browsers gespeichert und bei jedem weiteren API-Aufruf automatisch im Header der HTTP-Anfragen übermittelt, um den Benutzer zu identifizieren. Diese Token-basierte Authentifizierung bietet eine sichere und skalierbare Methode, um Benutzersitzungen ohne serverseitige Speicherung von Sitzungsdaten zu verwalten.

Die Token-Generierung ist im Snippet 5 dargestellt. Der Token wird mit einer geheimen Schlüssel (SECRET_KEY) signiert, die nur dem Server bekannt ist. Dies stellt sicher, dass nur der Server gültige Token erstellen kann.

```
private String generateToken(String username) {
    Algorithm algorithm = Algorithm.HMAC256(SECRET_KEY);
    return JWT.create()
        .withSubject(username)
        .withIssuedAt(new Date())
        .sign(algorithm);}
```

Snippet 5: generateToken-Methode in AuthServlet

Nach der Implementierung des Authentifizierungssystems wurde die Fahrzeugverwaltung entwickelt, um Benutzern die Registrierung, Aktualisierung und Löschung von Fahrzeugen zu ermöglichen. Diese Funktionen sind essenziell, da sie den Grundstein für das Buchungssystem legen.

Die Methode `registerVehicle`, wie in Snippet 6 gezeigt, ermöglicht es Benutzern, ein neues Fahrzeug im System zu registrieren. Diese Methode nimmt die Fahrzeugdaten entgegen und speichert sie in der Datenbank. Vor der Speicherung wird überprüft, ob der Benutzer, der das Fahrzeug registrieren möchte, auch tatsächlich existiert und berechtigt ist, Fahrzeuge zu registrieren.

```
package com.example.carconnect.vehicle;

@Override
public boolean registerVehicle(String ownerUsername, String make, String
model, int year, String location) throws RemoteException {
    return vehicleRepository.registerVehicle(ownerUsername, make, model, year,
location);
}
```

Snippet 6: Methode zur Registrierung eines Fahrzeugs

Die Implementierung des Buchungssystems stellt eine zentrale Komponente von CarConnect dar. Es ermöglicht Benutzern, Fahrzeuge in Echtzeit zu buchen, die Verfügbarkeit zu prüfen und Buchungen zu verwalten. Das Buchungssystem basiert auf einer serverseitigen Logik, die über RMI (Remote Method Invocation) realisiert wird.

Die Methode `bookVehicle`, wie in Snippet 7 dargestellt, ist für die eigentliche Buchung eines Fahrzeugs verantwortlich. Sie überprüft die Verfügbarkeit des Fahrzeugs im angegebenen Zeitraum und speichert die Buchung bei Erfolg in der Datenbank. Zusätzlich wird der Buchungsstatus aktualisiert, um zukünftige Buchungen entsprechend zu berücksichtigen.

```
package com.example.carconnect.booking;

@Override
public boolean bookVehicle(String username, int vehicleId,
String startTime, String endTime) throws RemoteException {
    return bookingRepository.bookVehicle(username, vehicleId,
startTime, endTime);
}
```

Snippet 7: Methode zur Buchung eines Fahrzeugs

Ein weiterer wichtiger Aspekt des Buchungssystems ist die Möglichkeit für den Fahrzeugbesitzer, Buchungsanfragen zu verwalten. Wie in Snippet 8 zu sehen, können Buchungsanfragen entweder genehmigt oder abgelehnt werden. Diese Entscheidung aktualisiert den Status der Buchung in der Datenbank und informiert den Benutzer über das Ergebnis.

```

package com.example.carconnect.booking;

@Override
public boolean approveBookingRequest(int requestId) throws RemoteException {
    boolean result = bookingRepository.updateBookingRequestStatus(requestId,
"APPROVED");
    if (result) {
        int vehicleId = bookingRepository.getVehicleIdFromBookingRequest(requestId);
        vehicleRepository.updateVehicleAvailability(vehicleId, false);
    }
    return result;
}

```

Snippet 8: Methode zur Genehmigung einer Buchungsanfrage

Das Benachrichtigungssystem wurde entwickelt, um Benutzer in Echtzeit über wichtige Ereignisse zu informieren, wie z.B. über die Genehmigung oder Ablehnung von Buchungsanfragen. Dies wird durch die Integration von WebSockets ermöglicht, die eine bidirektionale Kommunikation zwischen Server und Client erlauben.

Die Methode `sendNotification`, wie in Snippet 9 dargestellt, ermöglicht es dem Server, Benachrichtigungen an alle verbundenen Clients zu senden. Diese Benachrichtigungen enthalten Informationen über den Status von Buchungen, Änderungen an Fahrzeugdaten und andere relevante Ereignisse. Dadurch wird sichergestellt, dass alle Benutzer stets auf dem neuesten Stand sind.

```

package com.example.carconnect.websocket;

public static void sendNotification(String message) {
    for (NotificationEndpoint endpoint : connections) {
        synchronized (endpoint) {
            try {
                endpoint.session.getBasicRe-
mote().sendText(message);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

Snippet 9: Methode zur Sendung von Benachrichtigungen

4 Installation und Test

Um das CarConnect-System in Betrieb zu nehmen, müssen mehrere Schritte durchgeführt werden. Zunächst ist die Installation eines Tomcat-Servers erforderlich, auf dem die Webanwendung ausgeführt werden kann. Zusätzlich muss eine PostgreSQL-Datenbank installiert und gestartet werden, da diese als Backend-Datenbank für die Anwendung dient.

4.1 Installation

Nach der Installation von PostgreSQL muss die Datenbank konfiguriert werden. Zunächst wird eine neue Datenbank erstellt, und die notwendigen Tabellen werden angelegt. Für die Authentifizierung und andere Funktionen des Systems wird beispielsweise eine Tabelle für Benutzer benötigt. Diese kann mit folgendem SQL-Befehl erstellt werden:

```
CREATE TABLE USERS (  
    id SERIAL PRIMARY KEY,  
    username VARCHAR(255) NOT NULL,  
    hash VARCHAR(255) NOT NULL  
);
```

Snippet 10: Erstellung der Benutzertabelle in der PostgreSQL-Datenbank

Sobald die Datenbank eingerichtet ist, können die CarConnect-Projekte in einer IDE wie IntelliJ geöffnet werden. Maven übernimmt automatisch das Laden der notwendigen Abhängigkeiten. In der Klasse DBConnectionManager des DBServer-Projekts müssen die Verbindungsoptionen für die PostgreSQL-Datenbank angepasst werden. Standardmäßig läuft PostgreSQL auf Port 5432, und die Anmeldeinformationen müssen entsprechend konfiguriert werden.

Das Backend-System, das die Authentifizierung und Buchungslogik enthält, wird durch Starten der main-Methode in der Server-Klasse im DBServer-Projekt gestartet. Parallel dazu muss das Webprojekt in einem Tomcat-Server gestartet werden. Dazu wird in IntelliJ eine neue Konfiguration für einen "Tomcat Server local" erstellt, wobei der Standardport 8080 verwendet wird. Es ist wichtig, den JMX-Port anzupassen, da RMI standardmäßig auch den Port 1099 nutzt.

Sobald beide Projekte erfolgreich gestartet sind, ist das System unter der Adresse [http://localhost:8080/ carconnect_war_exploded /login.html](http://localhost:8080/carconnect_war_exploded/login.html) erreichbar. Für den Zugriff von anderen Rechnern im Netzwerk kann die IP-Adresse des Servers verwendet werden.

4.2 Test

Um die Funktionalitäten des CarConnect-Systems zu validieren, wurde das System in einer Testumgebung umfassend geprüft. Die Tests wurden hauptsächlich in einer lokalen Netzwerkumgebung durchgeführt und umfassten mehrere Szenarien, die den typischen Anwendungsfällen entsprechen.

Die Tests wurden in einer Umgebung durchgeführt, in der mehrere Benutzer in verschiedenen Rollen, wie Fahrzeugbesitzer und Buchender, agierten. Jeder Benutzer startete die Anwendung über den Browser auf seinem jeweiligen Endgerät, um die verschiedenen Funktionen des Systems zu testen.

Zunächst wurde die Registrierung neuer Benutzer getestet. Dabei konnte sich jeder Testbenutzer erfolgreich registrieren und anschließend mit den angegebenen Anmeldeinformationen einloggen. Nach dem erfolgreichen Login wurde überprüft, ob ein JSON Web Token (JWT) generiert und korrekt im lokalen Speicher des Browsers abgelegt wurde, um zukünftige Sitzungen zu authentifizieren.

In einem weiteren Schritt haben Fahrzeugbesitzer ihre Fahrzeuge über die Weboberfläche registriert. Die registrierten Fahrzeuge wurden korrekt in der Datenbank gespeichert, und die Benutzeroberfläche aktualisierte sich entsprechend, um die neu hinzugefügten Fahrzeuge anzuzeigen.

Ein weiterer wichtiger Testfall war die Suche nach verfügbaren Fahrzeugen und die anschließende Buchung dieser Fahrzeuge. Die Benutzer konnten erfolgreich nach Fahrzeugen filtern und eine Buchung für den gewünschten Zeitraum vornehmen. Der Buchungsstatus wurde in Echtzeit aktualisiert, und der jeweilige Fahrzeugbesitzer wurde umgehend über die neue Buchungsanfrage informiert. Darüber hinaus wurde die Möglichkeit für Fahrzeugbesitzer getestet, eingehende Buchungsanfragen zu genehmigen oder abzulehnen. Diese Funktionalität wurde umfassend getestet, indem Anfragen sowohl genehmigt als auch abgelehnt wurden. Bei Genehmigung wurde das Fahrzeug für den gebuchten Zeitraum als nicht verfügbar markiert, und der Buchungsstatus wurde entsprechend in der Datenbank aktualisiert.

Das Benachrichtigungssystem spielte eine wesentliche Rolle, indem es die Benutzer in Echtzeit über wichtige Ereignisse wie die Genehmigung oder Ablehnung von Buchungen informierte. Die Tests zeigten, dass die Benachrichtigungen sofort an alle verbundenen Benutzer gesendet wurden und dass die Benutzeroberfläche korrekt und zeitnah aktualisiert wurde.

Zusammenfassend wurden alle Tests erfolgreich durchgeführt, und das System erfüllte sämtliche spezifizierten Anforderungen. Die Authentifizierungs- und Buchungsprozesse funktionierten reibungslos, und das Benachrichtigungssystem stellte sicher, dass alle Benutzer über relevante Ereignisse informiert wurden. Während der Tests wurden keine kritischen Fehler festgestellt, die die Funktionalität des Systems beeinträchtigen würden.

5 Fazit & Ausblick

Die Entwicklung des CarConnect-Systems stellt einen bedeutenden Schritt in Richtung einer benutzerfreundlichen und effizienten Plattform zur Fahrzeugverwaltung und -buchung dar. Das Projektziel, eine interaktive und robuste Plattform zu schaffen, die moderne Technologien wie WebSocket und RMI nutzt, wurde weitgehend erreicht. Die Plattform zielt darauf ab, Menschen zu unterstützen, die ihre ungenutzten Fahrzeuge freiwillig zur Verfügung stellen möchten, um anderen zu helfen.

Zu den gelungenen Aspekten des Projekts zählt die erfolgreiche Implementierung eines sicheren und effizienten Authentifizierungssystems mittels JWT und RMI. Die Fahrzeugverwaltung und das Buchungssystem arbeiten reibungslos und ermöglichen eine intuitive Interaktion der Benutzer mit der Plattform. Die Echtzeit-Kommunikation über WebSockets wurde ebenfalls erfolgreich integriert, was die Benutzererfahrung erheblich verbessert. Dennoch gab es auch Herausforderungen. Zu Beginn traten Probleme bei der WebSocket-Integration auf, insbesondere hinsichtlich der Handhabung von Verbindungsabbrüchen und Reconnect-Strategien. Diese Schwierigkeiten wurden jedoch durch iterative Tests und Verbesserungen behoben. Des Weiteren war die Skalierbarkeit des Systems ein anfängliches Problem, welches durch optimierte Datenbankabfragen und Lastverteilungslösungen gelöst wurde.

Im Vergleich zu anderen bekannten Ansätzen bietet das CarConnect-System eine gut integrierte Lösung, die sowohl Echtzeit- als auch skalierbare Authentifizierungsmechanismen vereint. Während einige Systeme auf einfache Token-basierte Authentifizierung setzen, bietet die Kombination aus RMI und JWT eine höhere Sicherheit und Flexibilität. Zudem ermöglicht die WebSocket-Integration eine nahtlose Benutzerinteraktion, die in vielen traditionellen HTTP-basierten Systemen fehlt.

In der Zukunft bietet das CarConnect-Konzept verschiedene Erweiterungsmöglichkeiten. Eine mögliche Weiterentwicklung ist die Integration von Machine Learning-Algorithmen zur Vorhersage von Fahrzeugverfügbarkeiten und Benutzerpräferenzen. Dies könnte die Effizienz der Buchungssysteme weiter verbessern. Ein weiterer Aspekt könnte die Erweiterung der Plattform um zusätzliche Module wie ein detailliertes Wartungs- und Reparaturmanagement für die Fahrzeuge sein. Hierbei könnten Sensoren und IoT-Technologien genutzt werden, um den Zustand der Fahrzeuge in Echtzeit zu überwachen und Wartungsarbeiten proaktiv zu planen. Alternativlösungen könnten die Nutzung von Blockchain-Technologien zur Verbesserung der Datensicherheit und -integrität umfassen. Diese könnten insbesondere bei der Authentifizierung nützlich sein und die Plattform noch sicherer und vertrauenswürdiger machen.

Zusammenfassend lässt sich sagen, dass das CarConnect-System eine solide Basis für ein modernes Fahrzeugmanagement und Buchungssystem bietet. Mit den vorgeschlagenen Erweiterungen und Optimierungen kann die Plattform weiterentwickelt werden, um den sich ständig ändernden Anforderungen und Erwartungen der Benutzer gerecht zu werden. CarConnect bietet somit eine wertvolle Möglichkeit, ungenutzte Fahrzeuge effizient zu nutzen und Gemeinschaften zu unterstützen.

6 Literaturverzeichnis

- [RS14] C. Rupp, SOPHIST GROUP, Requirements-Engineering und- Management, 6. Auflage, Hanser, München Wien, 2014.
- [Kle18] S. Kleuker, Grundkurs Software-Engineering mit UML, 4. aktualisierte Auflage, Springer Vieweg, Wiesbaden, 2018.
- [Fe11] Melnikov, Alexey, and Ian Fette. 'The WebSocket Protocol'. Request for Comments. RFC Editor, December 2011.
- [JWT] Michael B. Jones, John Bradley, and Nat Sakimura. "JSON Web Token (JWT). 2015.
- [OSer] Oracle, Programming WebLogic HTTP Servlets, Version 8.1, 2003
https://docs.oracle.com/cd/E13222_01/wls/docs81/servlet/overview.html