Technische Dokumentation

Zum Projekt DriveLink

Angaben zum Dokument	
Projektnummer	SWVSYPRO-SS25-T8
Projektname	DriveLink
Kurzbeschreibung	Ein Softwaresystem zur digitalen Abwicklung von Fahrzeugreservierung, Verwaltung und Nutzerkonten sowie automatisierter Abrechnung
Auftraggeber	MoveSmart
Auftragnehmer	Neurobyte
Autor	Teoman Nazim Sabah (5345335), Amir Mangkok (5345319), Sayed Ahmad Qubaid (5345348), Rida Alil (5201352)
Version	1.0
Stand	30.07.2025

Inhaltsverzeichnis

1 Einführung

Diese technische Dokumentation beschreibt das Projekt DriveLink, ein webbasiertes Carsharing-System. Das System wurde von der Firma Neurobyte im Auftrag der MoveSmart GmbH entwickelt. Ziel war die vollständige Neuentwicklung einer Softwarelösung, die den Einstieg des Kunden in den Carsharing-Markt unterstützt und eine digitale Verwaltung von Fahrzeugen, Mitgliedern und Reservierungen ermöglicht.

1.1 Ziel und Umfang der Dokumentation

Diese technische Dokumentation hat das Ziel, den Entwicklungsprozess des Projektes strukturiert darzustellen und alle relevanten technischen Inhalte nachvollziehbar festzuhalten.

Sie richtet sich an verschiedene Zielgruppen:

- Projektbeteiligte und Entwickler: als Orientierungshilfe und Referenz für spätere Entwicklungsphasen.
- Technische Prüfer und Gutachter: zur Bewertung der Systemarchitektur, eingesetzten Technologien und umgesetzten Funktionen.
- Stakeholder und Endnutzer: zur Übersicht über den Funktionsumfang und den Nutzen der Anwendung.

Der Inhalt umfasst die Beschreibung der Anforderungen, eingesetzten Technologien, der Systemarchitektur und Implementierung sowie die eingesetzten Test- und Verifikationsverfahren. Ziel ist es, eine klare und vollständige Übersicht über alle Projektphasen und Systemkomponenten zu geben. Das Dokument dient sowohl als Nachschlagewerk als auch als Grundlage für mögliche Weiterentwicklungen in der Zukunft.

2 Produkt Backlog

In diesem Abschnitt wird das Produkt Backlog dargestellt . Es enthält alle Anforderungen aus dem Pflichtenheft, die zu Beginn des Projekts als Grundlage für die Entwicklung des ersten Prototyps definiert wurden.

Mitgliederverwaltung

Nr.	Backlog Item / Beschreibung
1	**Anmeldeseite** – Eine Login-Seite, auf der Nutzer ihre Zugangsdaten eingeben. Nach erfolgreicher Anmeldung werden sie zum persönlichen Dashboard weitergeleitet; Anmeldedaten werden sicher gespeichert.
2	**Registrierungsseite** – Gäste registrieren sich über ein Formular; die Daten werden an den Server gesendet und zur Kontoerstellung genutzt.
3	**Account-Infoseite** – Angemeldete Nutzer können persönliche Daten anzeigen und bearbeiten.
4	**Mitgliederverwaltung-Seite** – Admins können Konten anlegen, bearbeiten, einsehen und deaktivieren.
5	**Datenstruktur** – Umsetzung in DB und Modellklassen.
6	**API-Schnittstellen** – Alle Endpunkte für die Mitgliederverwaltung inkl. Geschäftslogik und Sicherheit.

Tabelle 2: Mitgliederverwaltung

Reservierungsverwaltung

Nr.	Backlog Item / Beschreibung
7	**Reservierungsansicht-Seite** Nutzer sehen vergangene und künftige Buchungen mit Details.
8	**Reservierungserstell-Seite** Nutzer wählen Fahrzeug, Zeitraum und Standort, um eine Buchung anzulegen.
9	**Reservierungsänderungs-Seite** Änderungen bestehender Buchungen wie Zeit oder Fahrzeug.
10	**Reservierungsverwaltungs-Seite** Admins können Reservierungen anlegen, bearbeiten, stornieren.
11	**Datenstruktur**DB-Tabellen und für Reservierungen.
12	**API-Schnittstellen** Endpunkte und Logik zur Validierung, Speicherung und Verarbeitung.

Tabelle 3: Reservierungsverwaltung

Tarifverwaltung

Nr.	Backlog Item / Beschreibung
13	**Tarifansichts-Seite** Übersicht aller verfügbaren Tarife für Nutzer.
15	**Datenstruktur** DB-Tabellen für Tarife.
16	**API-Schnittstellen** Endpunkte zur Verwaltung und Anzeige von Tarifen.

Tabelle 4: Tarifverwaltung

Fahrzeugverwaltung

Nr.	Backlog Item / Beschreibung
21	**Fahrzeugansichts-Seite** Übersicht aller Fahrzeuge mit Modell, Preise und Standorte.
22	**Fahrzeugverwaltungs-Seite** Fahrzeuge hinzufügen, ändern, löschen durch Mitarbeitende.
23	**Datenstruktur** – DB-Tabellen und Funktion für Fahrzeuge.
24	**API-Schnittstellen** Endpunkte für Fahrzeugfunktionen.

Tabelle 5: Fahrzeugverwaltung

Integration

Nr.	Backlog Item / Beschreibung
25	**REST-API-Tests** – Tests für jede API-Funktionalität.
26	**Abnahmetests** – Gesamttests zur Sicherstellung der Mindestanforderungen.

Tabelle 6: Integration

Deployment

Nr.	Backlog Item / Beschreibung
27	**Docker** – Dockerfiles und Compose-Datei zur Containerisierung und automatischen Einrichtung aller Systemkomponenten.

Tabelle 7: Deployment

3 Eingesetzte Technologien und Werkzeuge

3.1 Technologien in Frontend

React: React ist ein JavaScript-Framework zur Entwicklung von Benutzeroberflächen im Web. Es hilft dabei, verschiedene Teile einer Webseite wie Buttons, Kopf- und Fußzeilen oder ganze Seiten als wiederverwendbare Bausteine, sogenannte Komponenten, zu erstellen. Jede Komponente kann eigenen Zustand, Logik und Design haben, was die Wartung und Erweiterung der Anwendung erleichtert. React nutzt ein sogenanntes virtuelles DOM, mit dem nur die Teile der Webseite aktualisiert werden, die sich ändern. Dadurch muss die ganze Seite nicht neu geladen werden, was die Benutzererfahrung verbessert und die Serverlast reduziert.

React Router: React Router wird verwendet, um die Navigation in der React-Webseite zu steuern. Es sorgt dafür, dass verschiedene Seiteninhalte über unterschiedliche URLs im Browser angezeigt werden, ohne die ganze Seite neu zu laden. So bleibt die Webseite schnell und benutzerfreundlich. Die Routen (also welche URL welche Seite zeigt) können flexibel und einfach im Code definiert werden – auch mit verschachtelten Seiten oder dynamischen Inhalten, z.B. für Suchmasken oder Formulare.

Vite: Vite ist ein modernes Werkzeug zum Entwickeln von Webanwendungen, besonders geeignet für Frameworks wie React. Es hilft dabei, den Entwicklungsprozess schneller und einfacher zu machen, indem es den Code schnell lädt und Änderungen sofort sichtbar macht (Hot Module Replacement). Vite sorgt dafür, dass man beim Programmieren nicht lange warten muss, bis Änderungen wirksam werden. Für den Einsatz in der fertigen Anwendung (Produktion) ist Vite aber nicht relevant.

3.2 Technologien in Backend

SQLITE: SQLITE ist ein leichtgewichtiges, serverloses Datenbanksystem, das direkt als Datei auf dem lokalen Dateisystem gespeichert wird. Es benötigt keinen seperaten Datenbankserver und ist deshalb gut für kleinere Anwendungen oder lokale Entwicklungsumgebungen geeignet. In der Anwendung ist es gut, um alle wichtigen Daten dauerhaft zu speichern und durch die einfache Integration mit dem Backend kann man die Daten schnell und effizient speichern und aktualisieren, ohne eine externe Datenbank einrichten zu müssen.

Express: Express.js ist eine minimalistische und flexibeleles Node.js Framework zum schnellen Aufbau von Webanwendung und für seine Leistung dowie modulare Struktur bekannt ist. In der Anwendung bildet es das Rückgrat des Backends, indem es für Routing zuständig ist, API-Endpunkte wie /api/users oder /api/cars definieren und Middleware-Funktion wie Authentifizierung, Autorisierung und CORS integriert. Es übernimmt zudem das Handling von HTTP-Anfragen. Darüber hinaus organisiert Express.js die Geschäftlogik, indem es Controller und Datenbankinteraktionen via Sequelize miteinander verbindet. Die Vorteile für das Projekt liegen in der schnellen API-Entwicklung, derhohen Flexibilität und Skalierbarkeit sowie der Möglichkeit,das JavaScript-Ökosystem über den gesamten Stack zu nutzen.

3.3 Entwicklungswerzeuge für die Softwareentwicklung

Github: Github wird für die Verwaltung und die Versionskontrolle des Quellcodes verwendet. Es ist eine Plattform, die auf dem Versionsverwaltungssystem Git basiert und es ermöglicht, den Code zu speichern, Änderungen nachzuverfolgen und die Zusammenarbeit im Team effizient zu organisieren. Die Entwicklungen werden in einem Repositiory abgelegt, sodass Änderungen nachvollziehbar und über Branches und Pull Requests strukturiert halten zu können. Es dient nicht nur als Speicherort für Code, sondern auch als Anlaufstelle für die Koordination der Softwareentwicklung.

Docker: Docker ist eine Plattform, die es ermöglicht Anwendungen mit ihrer Abhängigkeit, Konfigurationen und Laufzeitumgebungen in Containern auszuführen. Diese Container sind eine leichte, isolierte Umgebung die auf jedem System laufen können, auf dem der Docker installiert ist, unabhängig vom Betriebssystem. Dadurch wird sichergestellt, dass die Anwendung überall gleich funktioniert.

REST API: Eine REST API (auch Representational State Transfer Application Programming Interface genannt) ist eine Schnittstelle, die es verschiedenen Softwareanwendungen ermöglicht, über das Internet miteinader zu kommunizieren. Sie folgt bestimmten Anforderungen, die sicherstellen dass die Kommunikation einfach und flexibel ist. REST APIs arbeiten meistens mit dem HTTP-Protokoll und nutzen die Methoden wie GET, POST, PUT und DELETE um Daten anzufordern, zu erstellen, zu ändern oder zu löschen. Sie sind auch ressourcenorientiert welches auch bedeutet, dass jede wichtige Information als Ressource betrachet wird und es hat eine eindeutige URL. Durch diese Sturktur und die Verwendung von Standardprotokollen sind sie ienfach zu nutzen und un verschiedene Systeme integrierbar. Sie sind gut dafür da, um Webanwendungen, mobile Apps oder andere Dienste miteinander zu verbinden und Daten auszutauschen.

HTTP: HHTP auch "Hypertext Transfer Protocol"genannt, ist das grundlegende Protokoll, das den Datenaustausch im Web regelt. Wenn eine Webseite aufgerufen wird, dann wird ein Formular abgesendet oder Daten von einem Server abgerufen. Diese laufen über die Kommunikation des HTTPs. Es legt fest, wie Nachrichten zwischen einem Client und einem Server gesendet werden. Der Client stellt dabei eine Anfrage und der Server antwortet mit den gewünschten Daten. Es arbeitet nach einem Anfrage-Antwort Prinzip. HTTP ist also das Rückrat der Kommunikation im Internet und ermöglicht es uns, über einfache Regeln zwischen Clients und Servern auszutauschen, auch ob es beim Arbeiten mit einer App oder beim Verbinden von Diensten über APIs.

JWT JWT auch "JSON Web Token"wird in der Webentwicklung dafür verwendet, Nutzer sicher zu identifizieren und ihre Zugriffsrechte zu überprüfen. Es kommt dann zum Einsatz, wenn sich zum Beispiel ein Nutzer in einer App anmeldet und der Server danach erkennen muss, wer der Nutzer ist. Nachdem der Nutzer sich eingeloggt hat, erstellt der Server ein JWT und gibt es dem Client zurück. Diesen Token speichert den Client lokal (im localstorage oder als Cookie) und schickt es bei jeder weiteren Anfrage an. Es besteht aus de drei Teilen:

- **Header:** beschreibt wie der Token verschlüsselt ist.
- **Payload:** Ort, wo die Daten stehen

- **Signatur:** dient als Sicherheitsshicht

Dadurch ist das System effizient und flexibel und am besten für moderne Webanwendungen, bei denen Frontend und Backend getrennt arbeiten, geeignet

4 Versionsmanagement

Dieses Kapitel beschreibt die Versionsverwaltung des Projekts. Der gesamte Quellcode liegt in einem zentralen Git-Repository auf GitHub, das als einzige Source of Truth für alle Teammitglieder dient. Die Codebasis entstand aus einem öffentlichen Vite + Node.js Startertemplate, das als erster Commit in den Hauptzweig master übernommen wurde und seither die Grundlage sämtlicher Entwicklungsarbeiten bildet.

4.1 Branching-Strategie

Die Entwicklung folgt einem trunkbasierten Modell (TBM). Alle Teammitglieder integrieren ihre Arbeit in den geschützten Hauptzweig master. Für jede Aufgabe wird daraus ein eigenständiger, kurzlebiger Branch gezogen. Erweiterungen tragen als Namen ausschließlich das jeweilige Feature. Korrekturen werden nach dem betroffenen Problem benannt und enden auf fix, zum Beispiel serverfix oder loginfix. Während der Umsetzung pushen die Beteiligten fortlaufend auf ihren Branch, eröffnen nach erfolgreicher Funktionsprüfung einen Pull Request und führen den Branch nach Review per Squash-Merge wieder in master zurück. Der master bleibt jederzeit integrierbar und unmittelbar auslieferbar.

4.2 Projektmanagement mit GitHub

Der gesamte Quell- und Projekt Datenbestand liegt in einem privaten GitHub Repository, auf das alle Teammitglieder mit Owner-Rechten zugreifen, damit sind Verwaltung und Review Aufgaben gleichmäßig verteilt, während Branch Protection Regeln unerwünschte Force Pushs oder unbeabsichtigtes Löschen absichern. Arbeitseinheiten werden ausschließlich als GitHub Issues erfasst. Deren Fortschritt visualisiert ein GitHub Projects-Board im Kanban-Stil mit den Spalten Backlog, Ready, In Progress, Review und Done. Jede Feature oder Fix Branch mündet in einen Pull Request, mindestens eine fachliche Prüfung verlangt, die Standard-Checks von GitHub durchläuft und nach erfolgreicher Kontrolle per Squash-Merge in den Hauptzweig integriert wird. Kommt es bei gleichzeitiger Arbeit an denselben Dateien zu Konflikten, werden diese vor dem Merge lokal gelöst und der Branch anschließend auf den aktuellen master rebased.

4.3 Repository-Struktur

4.3.1 Einleitung und Architektur

Für dieses Projekt wurde ein Monorepo gewählt. Dies bedeutet, dass alle logisch getrennten, aber funktional zusammengehörigen Anwendungskomponenten in einem einzigen Git-Repository verwaltet werden. Die Entscheidung für ein Monorepo ist strategisch und zielt darauf ab, die Kohäsion des Gesamtsystems zu stärken und Entwicklungsprozesse zu optimieren. Das zentrale Leitprinzip, das die gesamte Struktur durchdringt, ist die Separation of Concerns. Dieses Prinzip manifestiert sich auf der höchsten Ebene durch die klare Aufteilung des Projekts in zwei primäre, voneinander entkoppelte Hauptkomponenten, die die grundlegende Architektur einer modernen Webanwendung widerspiegeln:

1. Backend: Eine auf Node.js und Express basierende REST-API, die für die Geschäftslogik und Datenverwaltung zuständig ist.

2. Frontend: Eine mit Vite und React realisierte Single-Page Application (SPA), die die Benutzeroberfläche darstellt.

Die visuelle Gliederung des Wurzelverzeichnisses stellt sich wie folgt dar und bildet die Grundlage für die nachfolgenden detaillierten Erläuterungen:

Abbildung 1: Repository-Struktur

Die Wahl des Monorepo-Ansatzes ist durch mehrere signifikante Vorteile motiviert. Erstens ermöglicht er ein vereinfachtes Dependency Management. Durch den Einsatz von Workspace-Management-Tools können Abhängigkeiten zentral verwaltet und Versionen über die verschiedenen Komponenten hinweg konsistent gehalten werden. Zweitens fördert er die Code-Wiederverwendung. Beispielsweise können Typdefinitionen für die API-Kommunikation zwischen dem Backend und dem Frontend geteilt werden, was eine konsistente Datenschnittstelle erzwingt und redundanten Code vermeidet. Zusätzlich dazu, ermöglichen Monorepos atomare Commits. Änderungen, die beide Komponenten betreffen – wie die Einführung eines neuen API-Endpunkts im Backend und dessen Nutzung im Frontend – können in einem einzigen, logisch zusammenhängenden Commit erfasst werden. Dies verbessert die Nachvollziehbarkeit in der Versionshistorie erheblich.

4.3.2 Die Backend-Komponente

Die Backend ist als REST-API mit dem Node.js-Framework Express.js realisiert. Express.js zeichnet sich durch seine minimalistische und flexible Natur aus. Diese Flexibilität erfordert eine bewusste und disziplinierte Strukturierung, um die Wartbarkeit und Skalierbarkeit des Projekts sicherzustellen. Für dieses Projekt wurde eine klassische geschichtete Architektur (Layered Architecture) gewählt, die sich in der Praxis bewährt hat und eine klare Trennung der Verantwortlichkeiten fördert. Die Logik ist dabei in verschiedene Schichten aufgeteilt, die sich in der Verzeichnisstruktur widerspiegeln: Routen, Controller, Middleware und Models. Diese Struktur ist intuitiv und erleichtert die Navigation und das Verständnis des Codes.

Die detaillierte Verzeichnisstruktur der Backend-Komponente ist wie folgt aufgebaut:

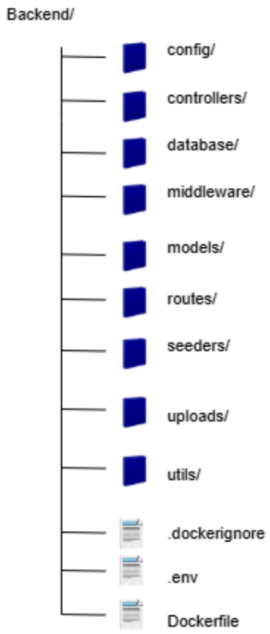

Abbildung 2: Verzeichnisbaum der Backend

Diese Struktur folgt dem Prinzip der 'Separation of Concerns' auf Dateiebene. Jedes Verzeichnis hat eine klar definierte Aufgabe. Die `routes` definieren die Endpunkte der API und leiten Anfragen an die entsprechenden `controllers` weiter. Die Controller enthalten die Kernlogik zur Verarbeitung der Anfrage, interagieren mit den models (der Datenzugriffsschicht) und senden eine Antwort zurück an den Client. `middleware` wird für querschnittliche Anliegen wie Authentifizierung oder Logging verwendet, die auf mehrere Routen angewendet werden können. Diese klare Aufteilung macht den Code modular, testbar und leichter verständlich.

4.3.3 Die Frontend-Komponente

Die Frontend ist als moderne Single-Page Application konzipiert und nutzt die leistungsstarke Kombination aus Vite als Build-Tool und React als UI-Bibliothek. Vite wurde aufgrund seiner extrem schnellen Entwicklungs-Performance durch Hot Module Replacement (HMR) und der Nutzung nativer ES-Module gewählt, was die Entwicklererfahrung erheblich verbessert. Die Architektur des Frontends legt den Schwerpunkt auf Skalierbarkeit und Wartbarkeit. Obwohl die genaue Struktur innerhalb des src-Verzeichnisses flexibel ist, folgt sie typischerweise einer Feature-basierten Organisation, um den Code logisch zu gruppieren.

Die detaillierte Verzeichnisstruktur der Frontend-Komponente sieht wie folgt aus:

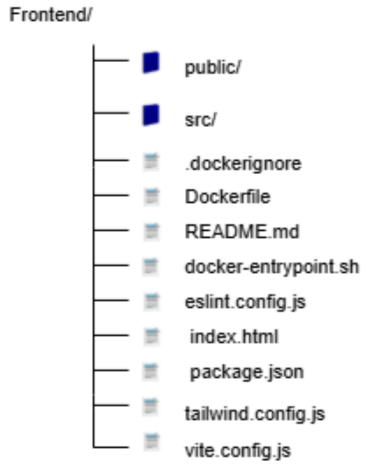

Abbildung 3: Verzeichnisbaum der Frontend

Ein wesentliches Merkmal einer Vite-basierten React-Anwendung ist die explizite Kontrolle über den Aufbau. Die `index.html` im Stammverzeichnis dient als Einstiegspunkt, in den die gebündelten JavaScript-Dateien von Vite injiziert werden. Das Routing wird programmatisch mit einer Bibliothek wie `react-router-dom` gehandhabt. Dies bietet maximale Flexibilität bei der Definition komplexer Navigationsstrukturen. Der `src`-Ordner enthält typischerweise Unterverzeichnisse wie components für wiederverwendbare UI-Elemente, pages oder views für seiten-spezifische Komponenten, hooks für benutzerdefinierte React Hooks und `utils` für Hilfsfunktionen.

4.3.4 Orchestrierung und Konfiguration auf der Wurzelebene

Die auf der Wurzelebene des Repositorys angesiedelten Dateien fungieren als das 'Bindeglied', das die einzelnen Komponenten (`Backend, Frontend`) zu einem kohärenten Gesamtsystem zusammenfügt und die Entwicklungsumgebung standardisiert. Das zentrale Werkzeug für die Orchestrierung der lokalen Entwicklungsumgebung ist Docker Compose. Die `docker-compose.yaml`-Datei dient als deklaratives Manifest. Sie definiert die Anwendungs-Services: `Backend` und `Frontend`, basierend auf den jeweiligen `Dockerfile` in den Unterverzeichnissen, die benötigten Infrastruktur-Container und die notwendigen Netzwerke und Volumes, um eine reibungslose Kommunikation zwischen den Containern und die Persistenz der Daten zu gewährleisten. Durch diesen Ansatz wird sichergestellt, dass jeder Teammitglieder mit einem einzigen Befehl `docker compose up` eine identische, voll funktionsfähige und isolierte Entwicklungsumgebung starten kann. Dies eliminiert die berüchtigten "works on my machine"Probleme und standardisiert den gesamten Entwicklungs.

5 Gesamtarchitektur

Die Carsharing-Plattform „Drive Link“ ist als moderne Webanwendung konzipiert, die über eine URL im Browser erreichbar ist. Die Oberfläche wird durch eine sogenannte Single Page Application (SPA) realisiert, die auf React basiert. Diese Anwendung läuft im Browser des Nutzers und wird beim ersten Aufruf über einen separaten Webserver bereitgestellt. Dabei wird lediglich eine leere HTML-Hülle sowie das JavaScript-Bundle übertragen. Anschließend übernimmt die Anwendung selbst die Darstellung und Navigation zwischen den Seiten, ohne dass die Seite erneut vom Server geladen werden muss. Das System besteht aus mehreren logisch getrennten Containern, die jeweils eine spezifische Aufgabe übernehmen. Die wichtigsten Komponenten sind:

- **Client (React-App):** Präsentationsschicht mit Benutzeroberfläche, läuft im Browser. Die Anwendung verwendet React Router zur clientseitigen Navigation und Vite als modernen Build- und Entwicklungsserver.
- **Server:** Bearbeitet alle API-Anfragen und enthält die Geschäftslogik. Implementiert ist der Server mit Express, einem minimalistischen Webframework für Node.js
- **Datenbank (SQLite):** Speichert dauerhaft Informationen wie Benutzer, Fahrzeuge und Reservierungen. Die Datenbank ist leichtgewichtig und wird direkt vom Server aus angesprochen.
- **Static Content Server:** Stellt Medieninhalte wie Fahrzeugbilder bereit. Diese Inhalte werden containerisiert und über einen dedizierten statischen Server ausgeliefert.

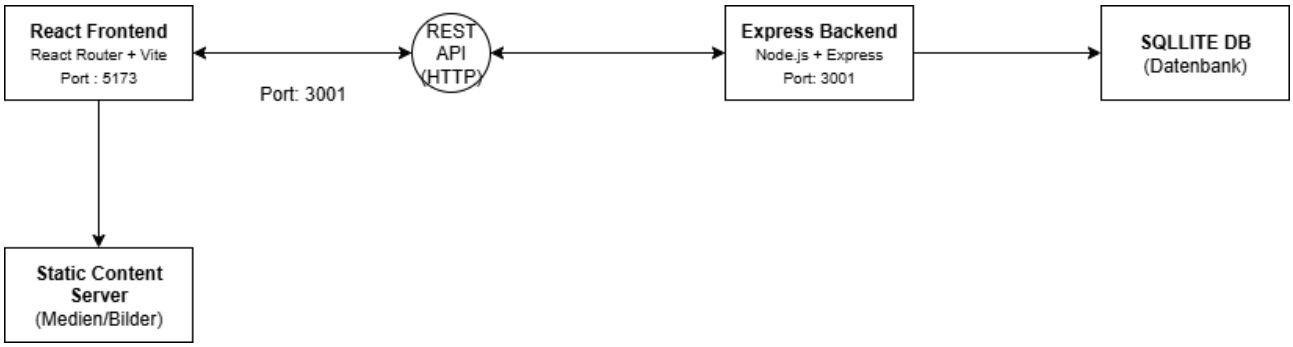

Abbildung 4: Komponentendiagramm vom Gesamtarchitektur

Diese Architektur ermöglicht eine klare Trennung von Frontend, Backend und Datenhaltung und sorgt gleichzeitig für gute Skalierbarkeit und Performance. Die gesamte Plattform wird mithilfe von Docker containerisiert und ermöglicht dadurch eine einfache Bereitstellung in unterschiedlichen Umgebungen.

5.1 Persistenzschicht

Für die dauerhafte Datenspeicherung nutzt „Drive Link“ eine relationale SQLite-Datenbank, in der alle zentralen Informationen strukturiert abgelegt sind. Dazu zählen unter anderem Tabellen für: User, Fahrzeuge, Reservierungen und Tarife. Die Datenbank ist als Datei organisiert `z.B carsharing_db.sqlite`) und wird direkt vom Backend gelesen und beschrieben. Eine

Verbindung erfolgt über den Object-Relational Mapper (ORM) Sequelize, der mit Node.js betrieben wird. Dadurch wird die Interaktion mit den relationalen Datenstrukturen abstrahiert, komplexe SQL-Abfragen lassen sich bequem in Form von JavaScript-Methoden realisieren. Für größere Medieninhalte wie Fahrzeugbilder wird ein separater statischer Server eingesetzt. In der Datenbank werden nur die Pfade oder URLs zu den Bilddateien gespeichert, nicht die Dateien selbst. Das reduziert die Last auf der Datenbank und verbessert die Performance der Anwendung erheblich. Da keine In-Memory-Lösungen wie Redis verwendet werden, erfolgt die Token-Verwaltung (z.B für JWT) lokal, über Token-Verifizierung bei jeder Anfrage, jedoch ohne persistente Blacklist.

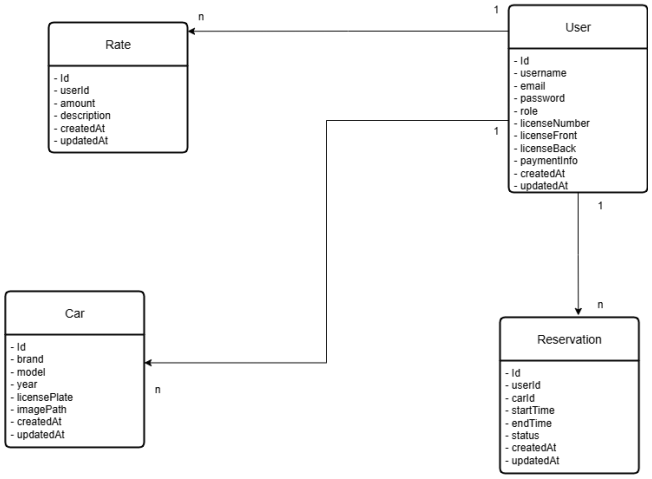

Abbildung 5: ER-Diagramm der Klassen

5.2 Dynamische Sicht

Die dynamische Sicht beschreibt die zeitliche Abfolge und Interaktion der einzelnen Systemkomponenten zur Laufzeit. Die Webanwendung „Drive Link“ wird von der Firma MoveSmart betrieben und wurde vom Unternehmen Neurobyte entwickelt. Sie basiert auf einer komponentenbasierten Architektur mit einem React-Frontend, einem Node.js-Backend, einer SQLite-Datenbank sowie einer JWT-basierten Authentifizierung. Die zentrale Funktion der Anwendung ist die Fahrzeugreservierung, doch es gibt eine Reihe weiterer Funktionen, deren Ablauf sich exemplarisch anhand der Systemarchitektur erklären lässt:

5.2.1 Benutzerauthentifizierung (Login)

Beim Öffnen der Webanwendung ruft der Benutzer die Login-Seite auf und gibt seine Anmeldedaten ein. Das Frontend (React SPA) sendet diese Daten via POST-Anfrage an das Backend. Dort validiert ein Auth-Service die Eingaben, prüft die Zugangsdaten in der SQLite-Datenbank und erstellt bei erfolgreicher Authentifizierung ein JWT (JSON Web Token). Dieses Token wird an das Frontend zurückgegeben und lokal gespeichert (z.B im SessionStorage). Es dient ab diesem Zeitpunkt zur Authentifizierung aller weiteren Anfragen.

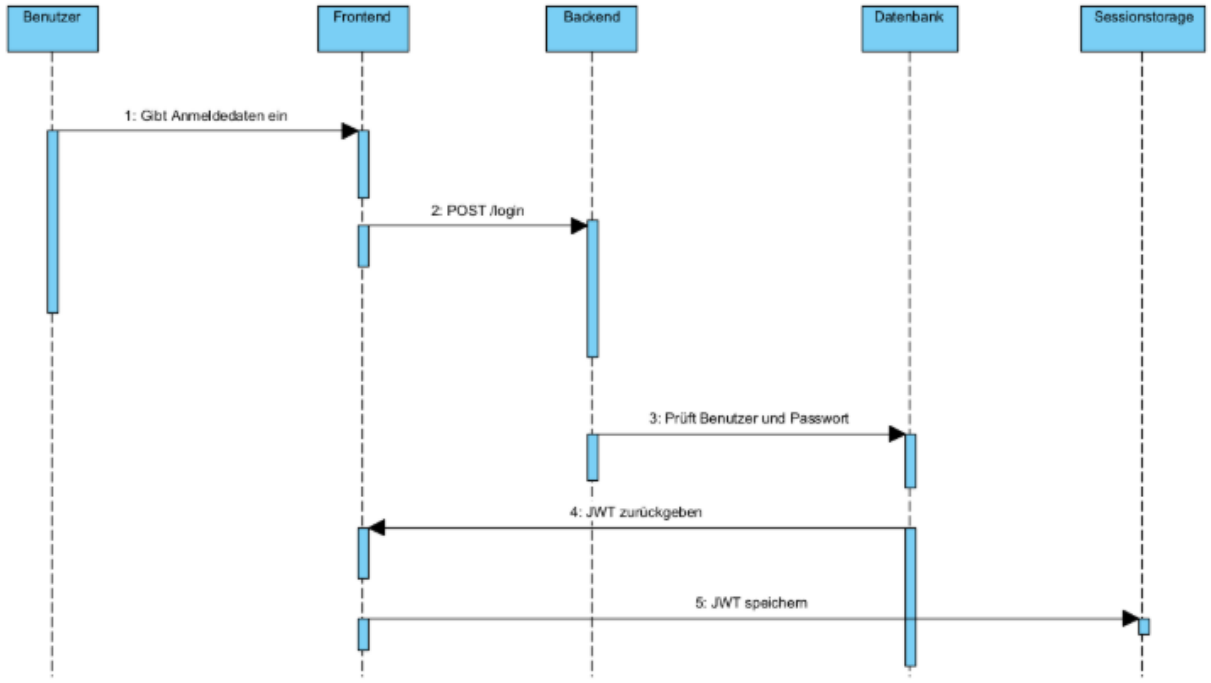

Abbildung 6: Anmeldung Sequenzdiagramm

5.2.2 Zugriff auf geschützte Endpunkte

Funktionen wie Benutzerverwaltung (für Admins), das Anlegen von Fahrzeugen oder das Bearbeiten von Reservierungen sind nur nach vorheriger Authentifizierung möglich. Dabei überprüft eine Middleware bei jedem Request, ob das Token gültig ist. Nicht autorisierte Zugriffe werden sofort mit einem 401-Fehler abgewehrt. Rollenprüfungen (z.B. Admin oder Staff) finden ebenfalls in diesem Schritt statt.

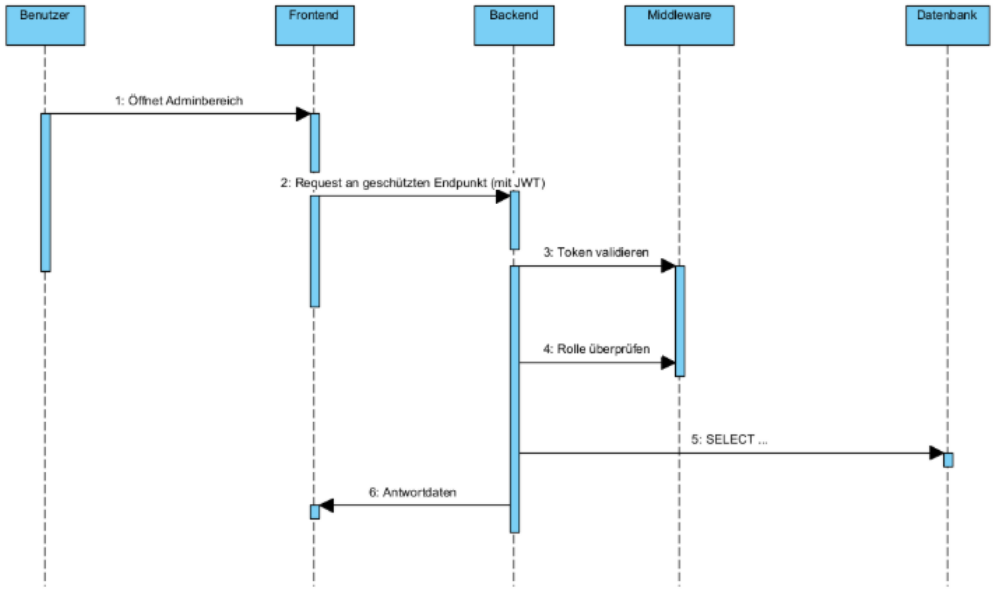

Abbildung 7: Authentifizierung Sequenzdiagramm

5.2.3 Fahrzeugreservierung

Sobald sich der Nutzer für ein Fahrzeug entschieden hat, kann er eine Reservierung anstoßen. Im Frontend wird dazu eine POST-Anfrage an den entsprechenden Service gesendet. Diese enthält alle relevanten Daten (Fahrzeug-ID, Zeitfenster, Nutzer-ID). Das Backend prüft: Ist der Nutzer authentifiziert? Ist das Fahrzeug im gewünschten Zeitraum verfügbar? Liegen Sperrungen oder Zahlungsrückstände vor? Ist alles korrekt, wird eine Transaktion geöffnet, die neue Reservierung in der Datenbank gespeichert und dem Benutzer die Reservierungs-ID zurückgeschickt.

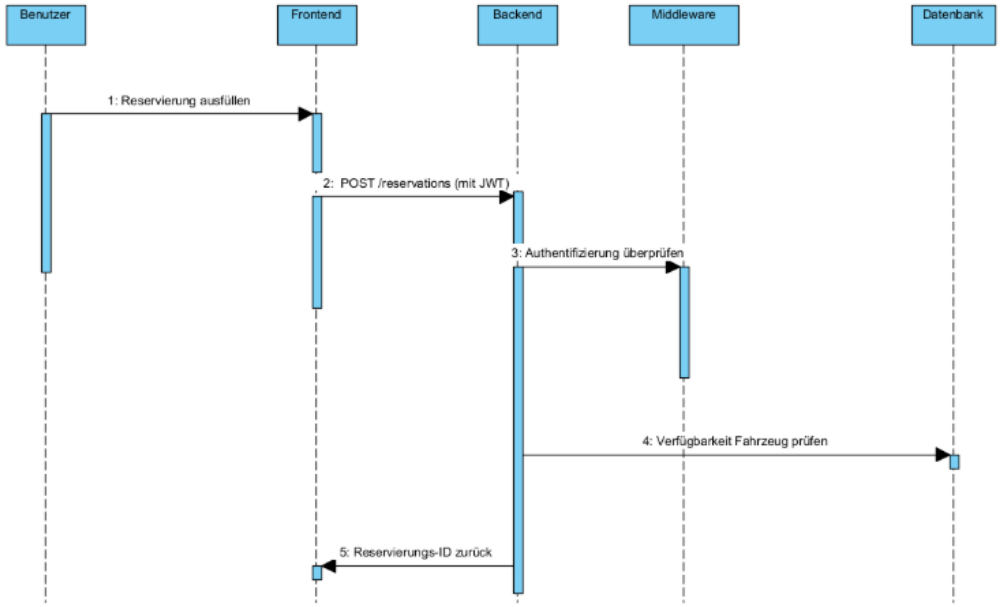

Abbildung 8: Reservierung Sequenzdiagramm

5.2.4 Abmelden (Logout)

Wenn sich der Benutzer abmeldet, wird das lokal gespeicherte Token im Frontend gelöscht. Da kein zentrales Token-Blacklisting implementiert ist, ist die Token-Gültigkeit technisch weiterhin gegeben, aber durch die fehlende Speicherung nicht mehr nutzbar. Alle beschriebenen Abläufe zeigen exemplarisch, wie das System bei „Drive Link“ funktioniert: Das Frontend agiert als leichtgewichtige Benutzeroberfläche, die REST-Anfragen an das Backend stellt. Dort übernehmen spezialisierte Services die Geschäftslogik, Authentifizierung und Datenhaltung. JWT sorgt für eine sichere Zugriffskontrolle, während die SQLite-Datenbank die langfristige Persistenz gewährleistet. Diese dynamische Verzahnung aller Komponenten schafft ein robustes und nachvollziehbares Gesamtsystem.

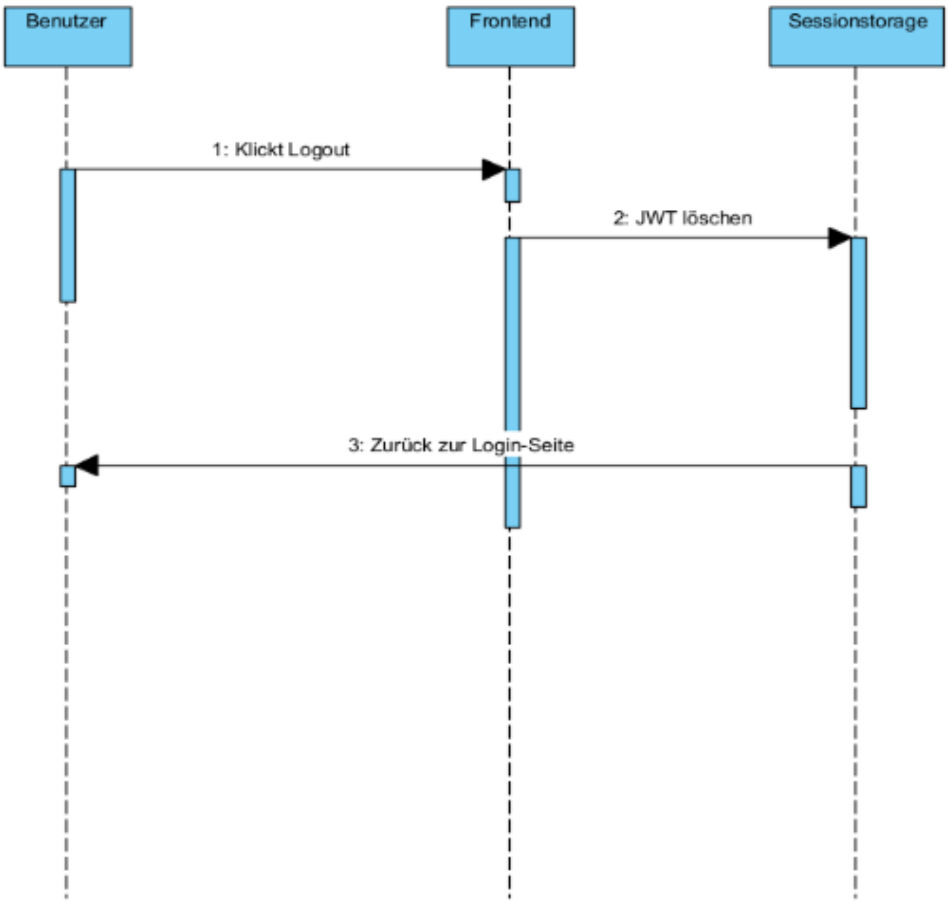

Abbildung 9: Abmeldung Sequenzdiagramm

6 Komponenten und Schnittstellen

6.1 Sitemap

Die Sitemap dient als struktureller Bauplan der Anwendung und zeigt alle verfügbaren Seiten sowie deren hierarchische Beziehung zueinander auf. Die Gliederung orientiert sich an den verschiedenen Benutzerrollen des Systems : öffentlicher Besucher, eingeloggte Kunden, Mitarbeiter und Administratoren. Für jede Seite werden zugehörige URL - Pfad und eine kurze funktionale Beschreibung angegeben, um einen Überblick über den gesamten Funktionsumfang und die Navigationlogik der Applikation zu ermöglichen.

Sitemap

```
/                                   LandingPage: Öffentliche Start und Informationsseite
|---/ about                         AboutUsPage: Seite mit Information über das Unternehmen
|---/ rates                         RatesPages: Übersicht der verfügbaren Tarife und Preise
|---/ login                         LoginPage: Anmeldeseite für alle Benutzerrollen.
|---/ register                      RegisterPage: Mehrstufiger Prozess zur Registrierung neuer Kunden

/app/                               Privater Bereich für eingeloggte Kunden
|---/ home                          HomePage: Persönlicher Startseite nach dem Login.
|---/ reservation                   ReservationPage: Übersicht der eigenen aktuellen Reservierung
|    |--- new                       NewReservationPage: Formular zur Erstellung einer Reservation
|    |--- edit/ : id                EditReservationPage: Formular zur Bearbeitung einer Reservation
|---/ profile

/staff/                             Interner Bereich für eingeloggte Mitarbeiter
|---/ dashboard                     StaffDashboardPage: Dashboard mit
|---/ reservation                   StaffReservationPage: Zentraler Übersichtseite für Mitarbeiter
|---/ cars / edit / : id            EditCarPage: Bearbeitung von spezifischen Fahrzeugdaten
|---/ rates / manage                StaffRatesPage: Verwaltung der Fahrzeugtarife
|---/ users / update / : id         UpdateUserPage: Aktualisierung von Kundendaten durch Mitarbeiter

/admin/                             Adminnistrativer Bereich mit vollen Zugriffsrecht
|---/ dashboard                     AdminDashoardPage: Dashboard mit Systemübersicht
|---/ management / cars             CarManagmentPage: Umfassende Verwaltung des gesamten Fuhrparks
|---/ management / users            UserManagmentPage: Umfassende Verwaltung aller Benutzerkonten
|---/ management / reservations     ReservationManagemntPage: Umfassende Verwaltung der Reservierung
|---/ management / staff / create   CreateStaffPage: Formular zum Anlegen neuer Mitarbeiter
```

Abbildung 10: Sitemap

6.2 Schnittstellen

In diesem Kapitel werden die technischen Schnittstellen der Anwendung beschrieben, die den Datenaustausch zwischen dem Frontend und dem Backend ermöglichen. Die Kommunikation erfolgt über eine zustandslose REST-API im JSON-Format. Die folgende Tabellen geben einen Überblick über alle verfügbaren Endpunkte. Im Anschluss werden diese detailliert beschrieben.

Authentifizierung

Methode	Endpunkt	Kurzbeschreibung	Berechtigung
POST	/auth/register	Registriert einen neuen Endkunden.	Gast

POST	/admin/staff/register	Registriert einen neuen Mitarbeiter.	Admin
POST	/auth/login	Authentifiziert einen Benutzer.	Gast

Tabelle 8: Authentifizierung

Fahrzeuge

Methode	Endpunkt	Kurzbeschreibung	Berechtigung
POST	/cars	Legt ein neues Fahrzeug an.	Admin
GET	/cars	Ruft die Liste aller Fahrzeuge ab.	Gast
GET	/cars/:id	Ruft ein einzelnes Fahrzeug ab.	Gast
PUT	/cars/:id	Aktualisiert ein Fahrzeug.	Admin
DELETE	/cars/:id	Löscht ein Fahrzeug.	Admin

Tabelle 9: Fahrzeuge

Tarife

Methode	Endpunkt	Kurzbeschreibung	Berechtigung
POST	/rates	Legt einen neuen Tarif an.	Admin
GET	/rates	Ruft alle Tarife ab.	Gast
GET	/rates/:id	Ruft einen einzelnen Tarif ab.	Gast
PUT	/rates/:id	Aktualisiert einen Tarif.	Admin
DELETE	/rates/:id	Löscht einen Tarif.	Admin

Tabelle 10: Tarife

Reservierungen

Methode	Endpunkt	Kurzbeschreibung	Berechtigung
POST	/reservations	Erstellt eine neue Reservierung.	Kunde
GET	/reservations	Ruft alle Reservierungen des Nutzers ab.	Kunde/Admin
GET	/reservations/:id	Ruft eine einzelne Reservierung ab.	Kunde/Admin
PUT	/reservations/:id	Aktualisiert eine Reservierung.	Kunde/Admin
DELETE	/reservations/:id	Löscht eine Reservierung.	Kunde/Admin

Tabelle 11: Reservierungen

7 Verifikation

In diesem Abschnitt werden die Strategien und die Methoden zur verifikation des Carshariing-Systems dargelegt. Ziel ist es, die korrekte und vollständige Umsetzung aller im Pflichenheft definierten Anforderungen nachzuweisen. Anschließend werden die Testergebnisse präsentiert und in einer Verifikationsmatrix den ursprünglichen Anforderungen gegenübergestellt

7.1 Teststrategien

Unsere Teststrategie dient der systematischen Verifikation des Gesammtsystems, um die korrekte und vollständige Umsetzung der definierten Anforderungen sicherzustellen. Das Hauptziel ist die vollständige funktionale Abdeckung.

7.2 Testergebnisse

Die Test wurden mit dem Jest-Framework durgeführt. Die 35 Test liefen durch und lieferten ein positives ergebniss wie in der Abbildung zu sehen

```
> backend@1.0.0 test
> jest

 PASS  controllers/staffController.test.js
 PASS  controllers/reservationController.test.js
 PASS  controllers/carController.test.js
 PASS  controllers/userController.test.js
 PASS  controllers/ratesController.test.js
 PASS  controllers/authController.test.js

Test Suites: 6 passed, 6 total
Tests:       35 passed, 35 total
Snapshots:   0 total
Time:        0.698 s, estimated 1 s
Ran all test suites.
```

Abbildung 11: Testergebnisse

ID	Szenario	Testziel	Ablauf (Testschritte)	Erwartetes Ergebnis
MT-01	Ein neuer Gast registriert sich als neues Mitglied.	Die korrekte Validierung von Eingabefeldern wird geprüft. Eine erfolgreiche Registrierung führt zu einem Datenbankeintrag.	1. Webseite öffnen und auf Registrieren klicken. 2. E-Mail ohne @ und Passwort Haus1 eingeben. 3. Korrekte Daten eingeben und Formular abschließen.	1. Fehlermeldungen für die ungültige E-Mail und das zu kurze Passwort erscheinen. 2. Registrierung ist erfolgreich, der Nutzer wird weitergeleitet.
MT-02	Ein registriertes Mitglied will sich anmelden.	Das System soll eine Anmeldung nur mit korrekten und vollständigen Daten zulassen.	1. Auf Anmelden klicken. 2. Ein Feld leer lassen. 3. Ein falsches Passwort eingeben. 4. Korrekte Daten eingeben.	1. Fehlermeldung für das leere Feld erscheint. 2. Fehlermeldung für das falsche Passwort erscheint. 3. Mitglied wird erfolgreich angemeldet.
MT-03	Ein angemeldetes Mitglied möchte sich abmelden.	Das Mitglied soll sich über die Oberfläche erfolgreich abmelden können.	1. Im eingeloggten Zustand auf Abmelden klicken. 2. Prüfen, ob man zur Login-Seite weitergeleitet wird.	1. Die Sitzung des Mitglieds wird beendet. 2. Eine erneute Anmeldung ist möglich, was den Logout bestätigt.
MT-04	Ein Mitglied möchte seine Daten ändern.	Ein Mitglied soll seine persönlichen Daten und sein Passwort ändern können.	1. Zu Mein Account navigieren. 2. Persönliche Daten bearbeiten und speichern. 3. Zum Tab Passwort ändern wechseln und Passwort aktualisieren.	1. Die Änderungen der Daten werden übernommen. 2. Das Passwort wird erfolgreich geändert.
MT-05	Ein Mitglied möchte eine neue Reservierung erstellen.	Ein Mitglied soll ein verfügbares Fahrzeug buchen können.	1. Zur Seite Autos navigieren. 2. Ein Fahrzeug auswählen. 3. Zeitraum und Station festlegen. 4. Reservierung bestätigen.	1. Die Reservierung wird erfolgreich erstellt. 2. Eine Bestätigung wird angezeigt.

ID	Szenario	Testziel	Ablauf (Testschritte)	Erwartetes Ergebnis
MT-06	Ein Mitglied möchte eine Reservierung stornieren.	Das Mitglied soll eine bestehende Reservierung löschen können.	1. Eine bestehende Reservierung aufrufen. 2. Auf Löschen oder Stornieren klicken.	1. Die Reservierung wird aus dem System entfernt. 2. Die Stornierung wird bestätigt.
MT-07	Ein Mitglied möchte eine Reservierung verschieben.	Ein Mitglied soll den Zeitraum einer bestehenden Reservierung anpassen können.	1. Reservierung auswählen und auf Bearbeiten klicken. 2. Den Zeitraum anpassen. 3. Änderungen speichern.	1. Der Reservierungszeitraum wird erfolgreich aktualisiert. 2. Die Änderungen werden bestätigt.

Tabelle 12: Manuelle Testszenarien

7.3 Verifikationsmatrix

7.3.1 Anforderung

Nummer	Status	Test	Anmerkung
FA1	Erfüllt	MT1	
FA2	Erfüllt	Automatisierter Test	
FA3	Erfüllt	Automatisierter Test	
FA4	Geplant		
FA5	Erfüllt	MT5	
FA6	Erfüllt	MT6	
FA7	Erfüllt	MT4	
FA8	Geplant		
FA9	Erfüllt	Automatisierter Test	
FA10	Erfüllt	Automatisierter Test	
FA11	Erfüllt	Automatisierter Test	
FA12	Erfüllt	Automatisierter Test	
FA13	Erfüllt	MT5	
FA14	Geplant		
FA15	Geplant		
FA16	Geplant		
FA17	Geplant		
FA18	Erfüllt	MT3 und MT4	

FA19	Erfüllt	Code-Inspektion	
FA20	Geplant		
FA21	Geplant		
FA22	Geplant		

Tabelle 13: Anforderung

7.3.2 Bedienbarkeits/Operationalitäts-Anforderungen

Nummer	**Status**	**Test**	**Anmerkung**
BA1	Geplant		
BA2	Geplant		
BA3	Geplant		
BA4	Geplant		
BA5	Geplant		
OA1	Erfüllt	MT1 und MT2	

Tabelle 14: Bedienbarkeits/Operationalitäts-Anforderungen

7.3.3 Qualitätsanforderungen

Nummer	**Status**	**Test**	**Anmerkung**
QA1	Geplant		
QA2	Geplant		
QA3	Teilweise Erfüllt	Code-Inspektion	
QA4	Geplant		

Tabelle 15: Qualitätsanforderungen

7.3.4 Schnittstellen Anforderung

Die zentrale Schnittstelle des Systems ist eine REST-API, die den gesamten Datenaustausch zwischen Frontend und Backend regelt und vollständig erfüllt ist. Die Absicherung dieser API erfolgt über eine Token-basierte Authentifizierung mittels JSON Web Tokens (JWT), die ebenfalls erfüllt ist und durch die authMiddleware sichergestellt wird. Auch die Anbindung an die Datenbank ist durch das Sequelize-ORM standardisiert und somit erfüllt.

Andere im Pflichtenheft genannte Schnittstellen, wie die Anbindung an physische RFID-Leser, MQTT für Telemetrie oder externe Zahlungsdienste, sind als geplant zu betrachten, da ihre Implementierung außerhalb des reinen Software-Backends liegt und nicht Teil des aktuellen Projektumfangs war. Ein Datenexport ist teilweise erfüllt, da die Grundlage zum Abrufen eigener Nutzerdaten existiert.