htlperg
innovativ | kreativ | praxisnah

Way2Work

Unternehmensinterne Plattform für Fahrgemeinschaften

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/07/2025 – 26/03/2026

Projektmitglieder: Kevin Stocker
Christof Hundegger

Betreuer: Prof. Ing. Patrick Praher, MSc

1 Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung der Arbeit wurden die generativen KI-Tools ChatGPT, Google Gemini und DeepL Write zu folgendem Zweck verwendet: Rechtschreib- und Grammatikprüfung, Formulierungshilfen zur sprachlichen Verbesserung und Umformulierung einzelner Textpassagen.

Die inhaltliche Konzeption, technische Umsetzung, Programmierung sowie die fachliche Ausarbeitung der Arbeit erfolgten eigenständig durch die Autoren.

Perg, 26.03.2026

Ort, Datum

Unterschrift, Kevin Stocker

Perg, 26.03.2026

Ort, Datum

Unterschrift, Christof Hundegger

2 Danksagung

Wir bedanken uns bei sämtlichen Personen der HTL Perg und der Firma Count IT GmbH, die uns bei dieser Diplomarbeit unterstützt und diese Kooperation ermöglicht haben.

Unser besonderer Dank gilt unserem Diplomarbeitsbetreuer Herrn Professor Ing. Patrick Praher, MSc, der uns bei all unseren Fragen rund um das Verfassen der Diplomarbeit stets kompetent und engagiert zur Seite stand.

Ebenso gilt unser Dank Herrn Matthias Eder, der diese Zusammenarbeit mit der Firma Count IT ermöglicht hat. Weiters möchten wir uns herzlich bei unserem Projektbetreuer Matthias Furtlehner bedanken. Durch seine tatkräftige Unterstützung und sein Fachwissen konnten wir den praktischen Teil unseres Projekts erfolgreich umsetzen.

Ein ganz besonderer Dank richtet sich abschließend an unsere Eltern und Familien, die uns nicht nur durch das Korrekturlesen der Arbeit geholfen haben, sondern uns während unserer gesamten Schullaufbahn und Projektdauer stets ein starker mentaler Rückhalt waren.

3 Abstract

The diploma thesis focuses on the development of „Way2Work“, an internal company web application designed to facilitate carpooling in daily commuting. In cooperation with Count IT GmbH, a platform was created that automatically connects employees, reduces individual traffic, lowers CO_2 emissions, and at the same time promotes social interaction.

To achieve these goals, a modern system architecture based on .NET 9 was implemented. It consists of a Blazor-based web frontend and a C# backend that functions as a REST API. Data is stored using Microsoft SQL Server. A key technical focus was the integration of open-source components (Nominatim, OSRM, Vroom) for geocoding and automated route optimization. The application is deployed in a scalable, container-based Azure environment.

Employees access Way2Work via Single Sign-On (Entra ID). An intelligent matching system suggests suitable carpools based on users' working hours and routes. In addition, an integrated reward system grants points for verified shared rides, creating a gamification incentive for long-term use while supporting the company in achieving its sustainability goals.

4 Kurzfassung

Die Diplomarbeit befasst sich mit der Entwicklung von „Way2Work“, einer unternehmensinternen Webanwendung zur gezielten Vermittlung von Fahrgemeinschaften im Berufsverkehr. In Kooperation mit der Count IT GmbH entstand eine Plattform, die Mitarbeiter automatisiert vernetzt, den Individualverkehr reduziert, CO_2-Emissionen senkt und gleichzeitig den sozialen Austausch fördert.

Zur Erreichung dieser Ziele wurde eine moderne Systemarchitektur basierend auf .NET 9 umgesetzt. Diese gliedert sich in ein Blazor-basiertes Web-Frontend und ein C#-Backend, das als REST-API fungiert. Die Datenspeicherung erfolgt über einen Microsoft SQL Server. Ein technischer Schwerpunkt lag auf der Integration von Open-Source-Komponenten (Nominatim, OSRM, Vroom) für das Geocoding und die automatisierte Routenoptimierung. Der Betrieb der Anwendung erfolgt in einer skalierbaren, containerbasierten Azure-Umgebung.

Der Zugriff auf Way2Work erfolgt für Unternehmensangehörige über ein Single Sign-On (Entra ID). Durch ein intelligentes Matching-System werden den Nutzern passgenaue Fahrgemeinschaften basierend auf ihren Arbeitszeiten und Routen vorgeschlagen. Ein integriertes Reward-System belohnt nachgewiesene gemeinsame Fahrten mit Punkten, wodurch ein Gamification-Anreiz zur langfristigen Nutzung der Plattform geschaffen und das Unternehmen bei der Erreichung seiner Nachhaltigkeitsziele unterstützt wird.

Inhaltsverzeichnis

5 Einführung

Way2Work ist eine unternehmensinterne Plattform, die Mitarbeitende im Berufsverkehr gezielt zu Fahrgemeinschaften vernetzt. Im Fokus stehen dabei ein geringer organisatorischer Aufwand, Datenschutz im Unternehmenskontext sowie eine nutzerfreundliche Umsetzung, die die langfristige Nutzung durch Anreize unterstützt. Dieses Kapitel beschreibt die Motivation, Zielsetzung, den Projektinhalt sowie das Projektumfeld als Grundlage für die weiteren Kapitel.

Abbildung 1: Logo der Anwendung Way2Work

5.1 Motivation

Der Berufsverkehr ist in vielen Regionen stark durch Individualverkehr geprägt: Mitarbeitende pendeln häufig alleine mit dem eigenen Fahrzeug, obwohl Start- und Zielorte sowie Arbeitszeiten in Unternehmen oft Überschneidungen aufweisen. In der Praxis entstehen Fahrgemeinschaften dennoch selten, da die Abstimmung in der Regel informell erfolgt (z. B. über persönliche Kontakte) und damit unzuverlässig, aufwendig oder zu wenig transparent ist.

5.1.1 Herausforderungen im Unternehmenskontext

Öffentliche Mitfahrplattformen adressieren den allgemeinen Markt, sind jedoch für den Einsatz innerhalb einer Organisation nur bedingt geeignet. Typische Hürden sind fehlende interne Zugriffskontrolle, Datenschutzbedenken bei sensiblen Daten wie Wohnorte und Zeitfenster, mangelnde Verbindlichkeit sowie keine direkte Einbettung in Unternehmensprozesse. Für Unternehmen wie Count IT, die Nachhaltigkeits- bzw. ESG-Ziele haben, ergibt sich daraus ein Bedarf

an einer Lösung, die organisationsintern funktioniert und es Unternehmen ermöglicht Anreize für Fahrgemeinschaften in Form von Goodies zu schaffen.

5.1.2 Motivation für Way2Work

Way2Work wurde gemeinsam mit dem Kooperationspartner Count IT entwickelt, um dieses Potenzial durch eine digitale Plattform nutzbar zu machen. Durch automatisiertes Matching, klare Benutzerführung und ein Reward-System soll die Hürde zur Bildung von Fahrgemeinschaften reduziert werden. Neben dem ökologischen Nutzen (Reduktion von CO_2) soll Way2Work auch einen sozialen Mehrwert schaffen, indem standort- oder abteilungsübergreifender Austausch im Unternehmen erleichtert wird. [1]

5.2 Zielsetzung

Ziel der Diplomarbeit ist die Konzeption und Umsetzung einer Webanwendung, die Fahrgemeinschaften innerhalb eines Unternehmens automatisiert berechnet, übersichtlich darstellt und organisatorisch unterstützt. Die Anwendung soll so gestaltet sein, dass ausschließlich Mitarbeitende der jeweiligen Organisation Zugriff erhalten. Zudem sollen alle verwendeten Routing-Technologien Open Source sein.

[1] Die Abschnitte Einführung, Motivation sowie Herausforderungen im Unternehmenskontext wurden unter Einsatz generativer KI (ChatGPT) sprachlich überarbeitet. (siehe Anhang C)

5.3 Projektinhalt

Dieses Kapitel beschreibt den inhaltlichen und technischen Aufbau des Projekts. Zunächst wird ein kompakter Überblick über das Gesamtsystem gegeben. Darauf aufbauend werden die wesentlichen architektonischen Komponenten und deren Zusammenspiel näher erläutert.

5.3.1 Projektinhalt – Überblick

Das System besteht aus einer relationalen Datenbank, einem Web-API Backend als REST-API sowie einem Blazor Web-Frontend. Die Anwendung ist als gemeinsame *.NET 9 Solution* umgesetzt, wodurch Frontend, Backend und gemeinsame Bibliotheken konsistent versioniert und entwickelt werden können. Für die lokale Orchestrierung der beteiligten Komponenten wird Aspire (6.4.1) eingesetzt. Zusätzlich werden Open-Source-Komponenten für Geocoding, Routing und Optimierung integriert. Der Betrieb erfolgt in einer containerbasierten Azure-Umgebung (6.6.5).

5.3.2 Systemarchitektur und Komponenten

Datenbank und Datenmodell

Die Datenhaltung erfolgt in Microsoft SQL Server als relationalem Datenbankmanagementsystem (DBMS). Das Datenmodell ist normalisiert, um konsistente und robuste Berechnungen zu ermöglichen. Direkte Datenbankzugriffe aus dem Frontend erfolgen nicht, da sämtliche Operationen über das Backend laufen.

Backend und Schnittstellen

Das Backend wurde in C#/.NET 9 umgesetzt und als REST-API bereitgestellt. Es verwaltet Stammdaten, verarbeitet die Onboarding-Eingaben und führt die Berechnung sowie Persistierung potenzieller Fahrgemeinschaften durch. Architektonisch wird das CQRS-Prinzip eingesetzt; MediatR unterstützt die Trennung von Lese- und Schreiboperationen. Der Datenzugriff erfolgt über Entity Framework Core, die API ist über OpenAPI dokumentiert.

Routing, Geocoding und Optimierung

Für die technische Umsetzung der Routen- und Optimierungsaspekte werden Open-Source-Komponenten kombiniert: Nominatim für Geocoding, OSRM für Routenberechnung und

VROOM für Optimierung. Die Herausforderung liegt in der konsistenten Integration dieser Bausteine in eine Backend-Anwendung.

Frontend, Routing und User Flow

Das Web-Frontend wurde mit Blazor und MudBlazor umgesetzt. Es bildet den vollständigen User Flow ab: Onboarding, Matching-Übersicht, Detailansichten mit Karte sowie Interaktion mit Fahrgemeinschaften. Die Kartenvisualisierung erfolgt mit MapLibre. Das Frontend konsumiert ausschließlich die dokumentierten Backend-Endpunkte.

Deployment und Betrieb

Die Anwendung wird in einer Microsoft Azure-basierten Container-Umgebung betrieben. Diese Struktur ermöglicht eine produktionsnahe Bereitstellung, klare Trennung der Komponenten sowie die Skalierbarkeit einzelner Dienste.

Datenschutz und Security

Da Wohnorte und Pendelinformationen sensible personenbezogene Daten darstellen, sind Zugriffssicherheit und Datenschutz zentrale Anforderungen. Der Zugriff erfolgt ausschließlich über Microsoft Entra ID mittels SSO, wodurch nur Mitarbeitende der jeweiligen Organisation Zugang erhalten. Sensible Detailinformationen (insbesondere Adressen) werden nicht global sichtbar gemacht, sondern nur den jeweils beteiligten Personen einer konkreten Fahrgemeinschaft angezeigt.

5.3.3 Funktionale Schwerpunkte

Onboarding und Profilverwaltung

Das Onboarding ist dreistufig aufgebaut:

- (1) Erfassung von Wohnort(en),
- (2) optionales Hinterlegen von Fahrzeugdaten,
- (3) Definition eines Wochenplans mit Zeitfenstern und Toleranzen für Hin- und Rückfahrt.

Änderungen an Profil, Schedule oder Fahrzeugdaten sind jederzeit möglich und fließen in die Berechnungslogik ein.

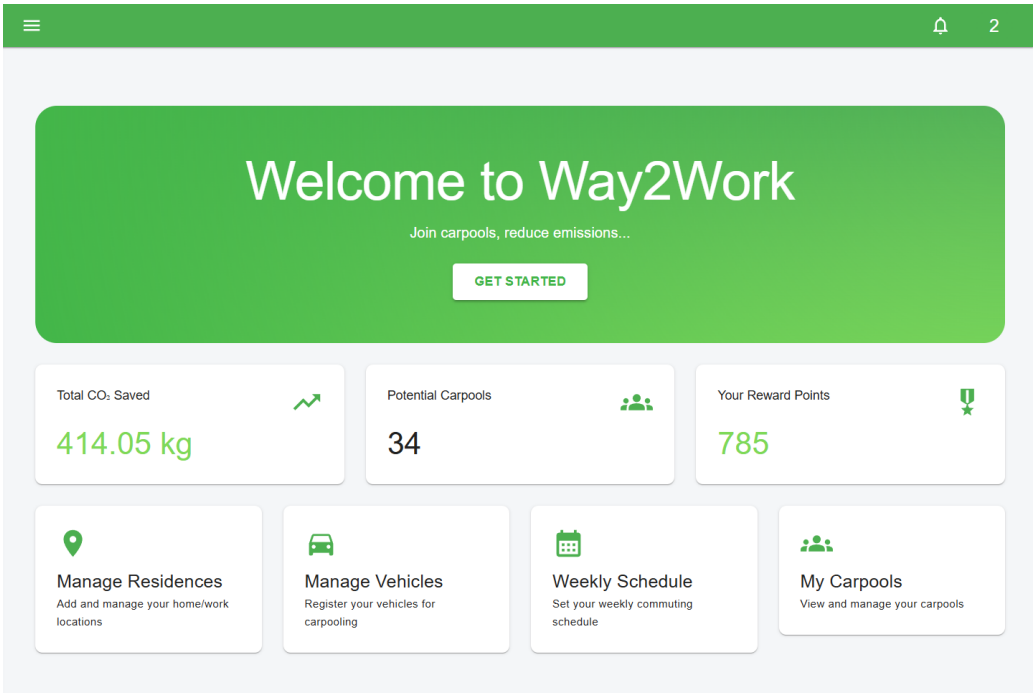


Abbildung 2: Way2Work Dashboard

Matching und Fahrgemeinschafts-Workflow

Basierend auf den Eingaben werden potenzielle Fahrgemeinschaften automatisch berechnet und je Tag dargestellt. Nutzer können Vorschläge ansehen, Fahrgemeinschaften auswählen, beitreten oder ablehnen. Eine Detailansicht visualisiert Route sowie Ein-/Ausstiegspunkte auf einer Karte.

Fahrtbestätigung und Nachweislogik

Nach einer gemeinsamen Fahrt kann die Fahrt durch den Fahrer bestätigt werden. Mitfahrer können keine Bestätigung durchführen; die Logik ist rollenbasiert umgesetzt. Auf Basis der Bestätigung werden CO_2-Werte sowie Reward-Punkte aktualisiert.

Reward-System

Way2Work beinhaltet ein Reward-System als Gamification-Element. Für bestätigte Fahrten werden Punkte gutgeschrieben; die Einlösung erfolgt organisatorisch über eine verantwortliche Stelle im Unternehmen. Damit wird ein Anreiz geschaffen, Fahrgemeinschaften nicht nur einmalig, sondern dauerhaft zu nutzen.

Backoffice und Auswertung

Für administrative Zwecke existiert eine Auswertungsansicht, die vergangene sowie potenzielle Fahrgemeinschaften gesamthaft darstellt. Dies dient der technischen Kontrolle im Betrieb.

5.4 Projektumfeld

Dieses Kapitel beschreibt das Umfeld, in dem das Projekt umgesetzt wurde. Dabei werden der Kooperationspartner, das Projektteam sowie die organisatorischen Rahmenbedingungen vorgestellt.

5.4.1 Kooperationspartner / Auftraggeber

Das Projekt wurde in Kooperation mit der Count IT GmbH umgesetzt. Count IT stellte die grundlegende Zielsetzung, praxisnahe Anforderungen sowie organisatorische Rahmenbedingungen bereit. Die Endabnahme erfolgte durch eine verantwortliche Stelle des Kooperationspartners.

Abbildung 3: Logo Count IT GmbH

5.4.2 Projektteam und Rollen

Das Projektteam bestand aus zwei Personen:

- Kevin Stocker: Frontend-Implementierung, Reward-System sowie Projektplanung und Koordination mit dem Kooperationspartner.
- Christof Hundegger: Backend-Implementierung inkl. Routing-/Optimierungslogik.

Das Datenmodell wurde gemeinsam geplant und abgestimmt.

5.4.3 Betreuung und Rahmenbedingungen

Die Umsetzung erfolgte im Rahmen der Diplomarbeit in Verbindung mit einem vierwöchigen Praktikum bei Count IT in Hagenberg. Die Qualitätssicherung und Abnahme wurden laufend durch Reviews unterstützt; Zwischenergebnisse wurden regelmäßig abgestimmt.

6 Theoretische und fachpraktische Grundlagen und Methoden

In diesem Kapitel werden die fachlichen Grundlagen des Projekts beschrieben. Außerdem werden zentrale Begriffe geklärt und die verwendeten Technologien, Werkzeuge und Methoden in ihren Grundzügen vorgestellt.

6.1 Grundlegende Fachbegriffe

Im Folgenden werden jene Fachbegriffe beschrieben, die für das Verständnis des Projekts wesentlich sind.

6.1.1 Fahrgemeinschaften und Nachhaltigkeit

Bei Fahrgemeinschaften benutzen mehrere Personen ein Fahrzeug, um ähnliche oder gleiche Strecken zurückzulegen. Dadurch kann der Individualverkehr reduziert, und die Fahrzeuge besser ausgelastet werden. Vor allem im Berufsverkehr sind Fahrgemeinschaften nützlich, da sie die Anzahl der Alleinfahrer reduzieren, und einen Beitrag zur nachhaltigen Gestaltung des Verkehrs liefern.

Gerade im Hinblick auf die Umwelt können Fahrgemeinschaften nützlich sein, weil sie den CO_2 Ausstoß verringern. Auch weniger Staus und ein besserer Verkehrsfluss können die Folge sein, weil sich das allgemeine Verkehrsaufkommen reduziert. Außerdem hat das nutzen von Fahrgemeinschaften auch wirtschaftliche Effekte, da sich die Kosten für den Treibstoff und die Fahrzeugnutzung aufteilen lassen. Fahrgemeinschaften stellen einen wichtigen Bestandteil nachhaltiger Mobilitätskonzepte dar, und ergänzen den bestehenden Verkehr sinnvoll.

6.1.2 Routing und Optimierung

Unter dem Begriff Routing versteht man in der Informatik und Verkehrstelematik die Ermittlung des optimalen Weges zwischen zwei oder mehreren Punkten innerhalb eines Netzwerks.

Verkehrssysteme werden dafür in der Regel als mathematische Graphen abgebildet, in denen Kreuzungen die Knoten (Nodes) und die Straßen die verbindenden Kanten (Edges) darstellen. Klassische Routing-Algorithmen, wie sie zum Beispiel von OSRM genutzt werden, suchen in diesen Strukturen nach der effizientesten Verbindung. Dabei ist “effizient“ nicht zwingend gleichbedeutend mit der absolut kürzesten Distanz: In der Praxis berücksichtigt das Routing vielmehr eine Kombination aus Fahrtzeit, Straßentypen und Geschwindigkeitsbegrenzungen, um die realistischste und beste Route zu finden.

Darauf aufbauend befasst sich die Routenoptimierung mit viel komplexeren Planungsaufgaben, die meist unter dem Begriff *Vehicle Routing Problem* (VRP) zusammengefasst werden [1]. Während das einfache Routing lediglich den Weg von A nach B berechnet, geht es bei der Optimierung darum, mehrere Fahrzeuge und Passagiere logistisch so zu koordinieren, dass alle Nebenbedingungen erfüllt sind. Für die automatisierte Bildung von Fahrgemeinschaften ist dieses Konzept essenziell: Die Algorithmen müssen unterschiedliche Startorte, gewünschte Ankunftszeiten, Toleranzfenster und die Sitzplatzkapazitäten der Fahrzeuge so aufeinander abstimmen, dass eine bestmögliche Auslastung erreicht wird, ohne den Fahrern zu große Umwege aufzuzwingen.

6.1.3 CO_2-Berechnung und Punkteverteilung

Ein wesentliches Ziel bei der Entwicklung der Plattform war es, den ökologischen Nutzen für die Nutzer direkt greifbar zu machen. Nur wenn man sieht, wie viele Emissionen durch eine gemeinsame Fahrt tatsächlich eingespart werden, entsteht ein Bewusstsein für die positiven Umwelteffekte.

Bei der theoretischen Konzeption der CO_2-Berechnung ergab sich aber eine Herausforderung: Um die exakte Einsparung zu ermitteln, müsste man theoretisch wissen, mit welchem konkreten Fahrzeug (und welchem individuellen Verbrauch) jeder einzelne Mitfahrer gefahren wäre, wenn er alleine zur Arbeit angereist wäre. Da diese Daten nicht immer vorliegen – oder manche Mitfahrer gar kein eigenes Auto besitzen –, wurde ein pragmatischen Berechnungsansatz gewählt.

Die Grundannahme dieses Ansatzes lautet: Jeder Mitfahrer, der in die Fahrgemeinschaft einsteigt, hätte andernfalls die Strecke von seinem Wohnort bis zur Firma alleine mit einem Fahrzeug derselben Emissionsklasse zurückgelegt. Die Einsparung der gesamten Fahrgemeinschaft ergibt sich also aus der Summe dieser theoretisch vermiedenen Einzelfahrten. Der Fahrer selbst wird

dabei logischerweise nicht als Einsparung gerechnet, da sein Auto die Strecke ohnehin fährt. [2]

Mathematisch lässt sich dieser Ansatz durch folgende Formel abbilden:

$$E_{\text{saved}} = \sum_{i=1}^{n} (d_i \times e_{\text{vehicle}})$$

Wobei die Variablen Folgendes bedeuten:

- n entspricht der Anzahl der Mitfahrer.
- d_i ist die individuelle Wegstrecke des jeweiligen Mitfahrers i zum Arbeitsplatz (in Kilometern).
- e_{vehicle} ist der spezifische CO_2-Ausstoß des für die Fahrgemeinschaft genutzten Fahrzeugs (in kg/km), welcher im Profil des Fahrers hinterlegt ist.

Konzept der Punkteverteilung (Gamification) Neben dem reinen CO_2-Wert war es wichtig, einen direkten Anreiz für die aktive Beteiligung an Fahrgemeinschaften zu schaffen. Aus theoretischer und praktischer Sicht erfordert die Rolle des Fahrers deutlich mehr Aufwand als die des Mitfahrers: Der Fahrer stellt das eigene Fahrzeug zur Verfügung, trägt die Kosten für den Verschleiß und übernimmt die Verantwortung für die Fahrtätigkeit.

Deshalb erhält der Fahrer bei einer erfolgreich abgeschlossenen Fahrt standardmäßig den doppelten Punktewert (200 Punkte) im Vergleich zu seinen Mitfahrern (jeweils 100 Punkte). Diese logische Punkteverteilung bildet die Berechnungsgrundlage für das anschließende Reward-System.

6.1.4 Reward-System

Ein Reward-System (Belohnungssystem) bezeichnet die Gesamtheit an Mechanismen, mit denen eine Organisation gewünschte Leistungen oder Verhaltensweisen durch Anreize (z. B. monetäre oder nicht-monetäre Belohnungen) fördert. In der Literatur wird ein Reward System unter anderem als Gesamtheit der monetären und nicht-monetären Gegenleistungen beschrieben, die eine Organisation ihren Mitarbeitenden im Austausch für erbrachte Arbeit zur Verfügung stellt [2]. Ziel eines Reward-Systems ist es, Motivation und Zielerreichung durch klar definierte Anreizstrukturen zu unterstützen [3].

[2] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

In der vorliegenden Arbeit wird dieses Prinzip in Form eines Punktesystems umgesetzt: Für gemeinsames Fahren werden Reward-Punkte gesammelt, die intern bei Count IT gegen Goodies (z. B. Merchandise) oder Gutscheine eingelöst werden können. Dadurch wird umweltfreundliches Verhalten gezielt belohnt. [3]

6.2 Verwendete Technologien

In diesem Abschnitt werden die im Projekt eingesetzten Technologien und Werkzeuge vorgestellt. Dabei wird kurz erläutert, welche Aufgabe sie im System übernehmen und warum sie für die Umsetzung gewählt wurden.

6.2.1 .NET 9 und C#

.NET ist ein von Microsoft entwickeltes Open-Source-Framework zur Erstellung moderner, plattformübergreifender Anwendungen. Als Programmiersprache kommt dabei das objektorientierte und stark typisierte C# zum Einsatz. Diese Sprache bietet Funktionen wie eine automatische Speicherbereinigung (Garbage Collection) und Language Integrated Query (LINQ), wodurch sich komplexe Datenverarbeitungen effizient und sehr gut lesbar umsetzen lassen.

Die eigentliche Ausführung der Programme übernimmt die sogenannte Common Language Runtime (CLR). Sie kümmert sich im Hintergrund um essenzielle Aufgaben wie die Speicherverwaltung und die Sicherheit. Ein zentraler Bestandteil des Ökosystems für Webanwendungen ist das Framework ASP.NET Core, mit dem sich hochperformante REST-APIs entwickeln lassen, die flexibel auf verschiedenen Betriebssystemen wie Windows oder Linux laufen [4].

(a) .NET Logo

(b) C# Logo

Abbildung 4: Logos von .NET und C#

[3] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

6.2.2 Blazor

Blazor ist ein modernes Webframework von Microsoft zur Entwicklung interaktiver Webanwendungen auf Basis von C# und .NET. Im Gegensatz zu klassischen Webframeworks wie Angular oder React, bei denen JavaScript bzw. TypeScript eine zentrale Rolle im Frontend übernehmen, ermöglicht Blazor die vollständige Umsetzung der logischen Bestandteile in C#. [5]

Blazor basiert auf einem komponentenorientierten Architekturmodell. Jede Komponente ist ein eigenständiger Teil der Benutzeroberfläche, der Darstellung und Logik kapselt. Inhalt und Styling werden in sogenannten `.razor`-Dateien definiert, der Logische Teil ist meistens in eine `.razor.cs`-Datei ausgelagert. [6]

Je nach Architektur unterscheidet man mehrere Ausführungsmodelle. In dieser Arbeit wird Blazor als serverseitige Webanwendung eingesetzt. Dabei läuft die Anwendungslogik auf dem Server, während der Browser lediglich die gerenderte Oberfläche darstellt. Benutzerinteraktionen werden über eine dauerhafte Verbindung (SignalR) an den Server übertragen, wo die Zustandsänderungen verarbeitet werden. [7]

Abbildung 5: Blazor Logo

6.2.3 GeoJSON

GeoJSON ist ein Datenformat zur Kodierung vieler verschiedener geografischer Datenstrukturen. Es verwendet die JavaScript Object Notation (JSON) und definiert eine Reihe von JSON-Objekten [8].

GeoJSON unterstützt die verschiedensten Geometrietypen, wie etwa `Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, und GeometryCollection` [9]. Diese werden in Abbildung 6 gezeigt.

Es gibt zum einen sogenannte `Features`, welche einzelne Objekte darstellen, und zum anderen `FeatureCollections`, welche eine Liste von `Features` speichern.

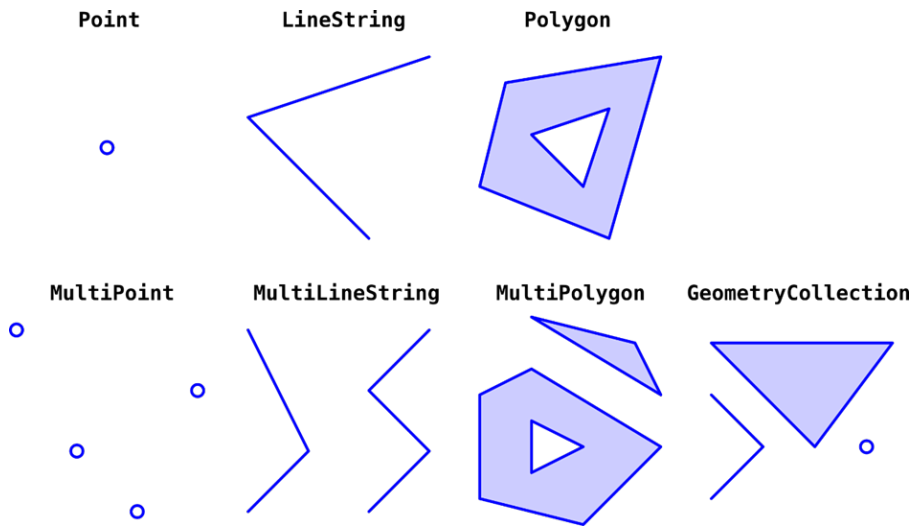


Abbildung 6: Übersicht über die unterstützten GeoJSON-Geometrietypen. [10]

GeoJSON verwendet standardmäßig das Koordinatenreferenzsystem `World Geodetic System 1984 (WGS 84)`. Die Koordinaten werden dabei in Dezimalgrad angegeben und in der Reihenfolge Längengrad (Longitude) gefolgt von Breitengrad (Latitude) gespeichert [11].

6.2.4 Microsoft Entra ID (Authentifizierung)

Microsoft Entra ID ist ein cloudbasierter Zugriffsverwaltungsdienst von Microsoft, der zur zentralen Verwaltung von Benutzeridentitäten und zur sicheren Authentifizierung in Anwendungen eingesetzt wird. Entra ID ermöglicht es Organisationen, Benutzerkonten, Gruppen und Zugriffsrichtlinien zentral zu verwalten und stellt eine einheitliche Basis für cloudbasierte und lokale Anwendungen bereit [12].

Nach erfolgreicher Anmeldung stellt Entra ID der Anwendung die relevanten Identitätsinformationen zur Verfügung, sodass ein sicherer Zugriff auf geschützte Bereiche der Anwendung ermöglicht wird. Die eigentliche Benutzerverwaltung sowie die Absicherung der Zugangsdaten erfolgen vollständig durch Entra ID, wodurch sicherheitsrelevante Aufgaben aus der Anwendung ausgelagert werden.

Im Rahmen dieser Arbeit wird Microsoft Entra ID zur Authentifizierung der Benutzer im Frontend eingesetzt. Ziel ist es sicherzustellen, dass ausschließlich berechtigte Mitarbeiter des Unternehmens Count IT Zugriff auf die entwickelte Anwendung erhalten. Da alle Mitarbeiter bereits über ein unternehmensinternes Benutzerkonto mit Firmen-E-Mail-Adresse verfügen, bietet sich Entra ID als bestehende und etablierte Authentifizierungslösung an. [13]

Abbildung 7: Entra ID Logo

6.2.5 SQL Server

Nahezu jede moderne Applikation benötigt eine Datenbank, um Informationen dauerhaft zu speichern und strukturiert verarbeiten zu können. In dieser Arbeit wird dafür Microsoft SQL Server eingesetzt. SQL Server ist ein relationales Datenbankmanagementsystem (RDBMS), bei dem Daten in Tabellen organisiert und über die Abfragesprache SQL verwaltet werden [14].

Durch die Verwendung eines relationalen Datenbanksystems können Daten konsistent gespeichert und über Beziehungen (z. B. Primär- und Fremdschlüssel) logisch miteinander verknüpft werden. Zudem ermöglicht SQL Server die Durchführung von Transaktionen, wodurch mehrere zusammengehörige Änderungen zuverlässig und vollständig ausgeführt werden [15].

Für die Entwicklung kann SQL Server lokal oder containerbasiert betrieben werden. Im produktiven Einsatz lässt sich die Datenbank auch einfach über Microsoft Azure hosten, wodurch Wartung und Betrieb weitgehend vereinfacht werden.

Abbildung 8: Microsoft SQL-Server Logo

6.3 Verwendete Entwicklungssysteme

In diesem Abschnitt wird die technische Umgebung beschrieben, in der das Projekt umgesetzt wurde. Dazu zählen das verwendete Betriebssystem, die eingesetzten Entwicklungswerkzeuge sowie die lokale Containerumgebung.

6.3.1 Windows 11 Entwicklungsumgebung

Für die Umsetzung der Anwendung wurden vom Projektpartner Entwicklungsrechner mit Windows 11 bereitgestellt. Windows 11 diente als zentrale Arbeitsumgebung für die Implementierung, das lokale Testen sowie das Debugging der Anwendung. Durch die breite Unterstützung aktueller Entwicklungswerkzeuge im Microsoft-.NET-Ökosystem (z. B. Visual Studio, SQL Server Management Studio und lokale Laufzeitumgebungen) eignet sich Windows 11 besonders für die Entwicklung von Anwendungen auf Basis eines Microsoft-Technologie-Stacks. [**?**]

Abbildung 9: Logo Windows 11

6.3.2 Visual Studio 2022

Visual Studio 2022 ist eine umfassende integrierte Entwicklungsumgebung (IDE) von Microsoft, die speziell für das .NET-Ökosystem optimiert ist. Sie bietet eine Vielzahl an intelligenten Werkzeugen, darunter eine fortschrittliche Code-Vervollständigung (IntelliSense), tiefgreifende Debugging-Möglichkeiten sowie integrierte Tools zur Code-Analyse und Projektverwaltung. Diese Funktionen erleichtern die Fehlersuche und ermöglichen einen sehr effizienten und strukturierten Entwicklungsprozess [16].

Da Visual Studio nahtlos mit den von uns eingesetzten Technologien wie C#, ASP.NET Core und Blazor harmoniert, diente es als zentrale Arbeitsumgebung für die gesamte Implementierung des Front- und Backends.

Abbildung 10: Visual Studio Logo

6.3.3 SQL Server Management Studio 21

Das SQL Server Management Studio (SSMS) ist eine von Microsoft entwickelte integrierte Umgebung für die Verwaltung, Konfiguration und Administration von SQL-Infrastrukturen. Das Tool bietet eine umfangreiche grafische Benutzeroberfläche, um relationale Datenbanken

effizient zu verwalten, Tabellenstrukturen anzupassen und komplexe SQL-Abfragen zu schreiben und auszuführen [17].

Neben dem reinen Datenzugriff stellt SSMS leistungsstarke Werkzeuge für die Überwachung, Fehlerbehebung und Leistungsoptimierung zur Verfügung. Sowohl lokale SQL Server-Instanzen als auch cloudbasierte Azure SQL-Datenbanken lassen sich damit zentral steuern. Durch den integrierten Abfrage-Editor und visuelle Design-Tools wird die tägliche Arbeit mit dem Datenbankschema und den gespeicherten Daten für Entwickler und Administratoren deutlich vereinfacht.

Abbildung 11: SQL Server Management Studio Logo

6.3.4 Podman Container Umgebung

Podman ist eine Open-Source Container-Engine zur Verwaltung und Ausführung von Containern. Ein wesentliches Alleinstellungsmerkmal gegenüber klassischen Containerlösungen ist die sogenannte daemonlose Architektur. Das bedeutet, dass Podman keinen zentralen Hintergrunddienst (Daemon) mit Administratorrechten benötigt, um Container zu steuern. Stattdessen werden diese als reguläre Benutzerprozesse ausgeführt, was Podman zu einer besonders sicheren und ressourcenschonenden Alternative macht.

Die Containerisierung an sich ermöglicht es, Softwareanwendungen gemeinsam mit all ihren Abhängigkeiten, Bibliotheken und Konfigurationen strikt isoliert vom restlichen Betriebssystem zu betreiben. Dies schafft reproduzierbare Umgebungen und die einzelnen Komponenten verhalten sich stets identisch, völlig unabhängig von der lokalen Systemkonfiguration des Entwicklerrechners. Vor allem bei der Arbeit mit verteilten Systemen erleichtert diese Isolation den lokalen Betrieb und das Testen verschiedener Dienste erheblich [18] [4].

[4]Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

Abbildung 12: Podman Logo

6.4 Verwendete Bibliotheken und Frameworks

In diesem Abschnitt werden die Bibliotheken und Frameworks vorgestellt, die bei der Entwicklung des Projekts verwendet wurden. Dabei wird jeweils kurz erklärt, welche Aufgabe sie im System übernehmen und wie sie die Umsetzung unterstützen.

6.4.1 Aspire (.NET Cloud Orchestrierung)

Aspire (*.NET Aspire*) ist ein von Microsoft bereitgestelltes Framework bzw. Tooling im .NET-Ökosystem zur Entwicklung und zum lokalen Betrieb verteilter Anwendungen.Es unterstützt insbesondere das strukturierte Zusammenspiel mehrerer Komponenten (z. B. Web-Frontend, Backend-Services und Infrastrukturkomponenten wie Datenbanken oder Container) und erleichtert dadurch die Entwicklung und das Debugging komplexerer Systemlandschaften [19].

Unter Orchestrierung versteht man die koordinierte Verwaltung mehrerer Dienste einer Anwendung (Start, Konfiguration und Zusammenspiel), sodass alle Komponenten gemeinsam lauffähig sind. Aspire stellt dafür mit dem sogenannten *AppHost* einen zentralen Einstiegspunkt bereit. In diesem werden die benötigten Ressourcen und Dienste deklarativ beschrieben, wodurch Konfigurationen konsistent an einem Ort verwaltet werden können. Das gesamte System kann anschließend mit einem einzigen Startvorgang ausgeführt werden. Zusätzlich wird ein Dashboard bereitgestellt, das eine Übersicht über die laufenden Komponenten und deren Status bietet [19].

In dieser Arbeit wird der AppHost verwendet, um die verschiedenen Anwendungsteile sowie zusätzliche containerisierte Dienste Routing- bzw. Geocodingdienste gemeinsam zu starten und lokal zu debuggen. [20]

Abbildung 13: Aspire Logo

6.4.2 Swagger

Swagger ist ein Open-Source-Framework, das weitverbreitet für den Entwurf, die Dokumentation und die Nutzung von RESTful-APIs eingesetzt wird. Es basiert maßgeblich auf der OpenAPI-Spezifikation, einem standardisierten, maschinenlesbaren und sprachunabhängigen Format zur Beschreibung von Programmierschnittstellen. Durch diesen Standard wird sichergestellt, dass sowohl Menschen als auch andere Softwaresysteme die Struktur und die Funktionen einer API vollständig verstehen können, ohne Zugriff auf den Quellcode zu benötigen.

Das bekannteste und am häufigsten genutzte Werkzeug aus diesem Ökosystem ist die Swagger UI. Diese Anwendung generiert aus der zugrundeliegenden API-Spezifikation automatisch eine interaktive, webbasierte Dokumentationsoberfläche. Über diese grafische Ansicht lassen sich alle verfügbaren Endpunkte, Parameter und Datenmodelle übersichtlich darstellen. Zudem bietet Swagger UI die Möglichkeit, direkt aus dem Browser heraus Testanfragen an die API zu senden und die Antworten zu analysieren, was den Entwicklungs- und Integrationsprozess erheblich vereinfacht [21] [5].

Abbildung 14: Swagger Logo

6.4.3 Entity Framework Core

Entity Framework Core (EF Core) ist ein modernes, plattformübergreifendes Object-Relational Mapping (ORM) Framework von Microsoft. Es vereinfacht den Datenzugriff innerhalb von .NET-Anwendungen erheblich, da Entwickler nicht mehr zwingend manuelle SQL-Abfragen schreiben müssen. Stattdessen können relationale Datenbanken direkt über objektorientierte C#-Klassen abgefragt und manipuliert werden.

Das Kernkonzept des Frameworks beruht auf sogenannten Entities (Entitäten), welche die Tabellen der Datenbank repräsentieren, sowie dem `DbContext`, der als zentrale Schnittstelle fungiert und gängige Lese- und Schreiboperationen (CRUD) automatisiert.

Für die Modellierung und Erstellung der Datenbankarchitektur unterscheidet man klassischerweise drei Ansätze: Bei *Database-First* existiert die Datenbank bereits und der Programmcode wird daraus abgeleitet. Bei *Model-First* wird das Datenmodell zunächst grafisch in einem Designer ent-

[5] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

worfen. Beim *Code-First*-Ansatz hingegen stehen die C#-Klassen im Zentrum der Entwicklung, und das Datenbankschema wird direkt aus dem geschriebenen Code generiert.

In dieser Diplomarbeit wurde der **Code-First**-Ansatz gewählt. Da die Datenbankstruktur somit vollständig im Code abgebildet ist, wird die Entwicklung deutlich übersichtlicher und weniger fehleranfällig. Ein essenzielles Werkzeug hierfür sind die integrierten EF Core Migrationen, die es ermöglichen strukturelle Änderungen an den Entitäten zu erfassen und diese dann versioniert, kontrolliert und reproduzierbar auf die Zieldatenbank anzuwenden [22] [6].

Abbildung 15: Entity Framework Core Logo

6.4.4 AutoMapper

AutoMapper ist eine populäre Open-Source-Bibliothek im .NET-Ökosystem, die das sogenannte Object-to-Object-Mapping automatisiert. In modernen, schichtenbasierten Softwarearchitekturen ist es gängige Praxis, Daten beim Übergang zwischen verschiedenen logischen Ebenen in separaten Objekten zu kapseln. Ein typisches Beispiel hierfür ist die Umwandlung von internen Datenbank-Entitäten (Entities) in externe Datenübertragungsobjekte (Data Transfer Objects, DTOs) für eine API-Antwort.

Ohne ein solches Werkzeug müssten Entwickler für jede dieser Objektumwandlungen repetitiven und fehleranfälligen Code schreiben, um die Werte der einzelnen Eigenschaften (Properties) manuell zuzuweisen. AutoMapper löst dieses Problem durch einen konventionsbasierten Ansatz. Eigenschaften mit identischen oder namensverwandten Bezeichnern werden bei der Konfiguration automatisch erkannt und einander zugewiesen. Durch diesen Automatisierungsschritt wird der Code deutlich kompakter, die Lesbarkeit steigt und der Wartungsaufwand bei Modelländerungen wird minimiert [23].

Abbildung 16: AutoMapper Logo

[6] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

6.4.5 MudBlazor

MudBlazor ist eine Open-Source-Komponentenbibliothek für Blazor, die eine Vielzahl vorgefertigter UI-Komponenten bereitstellt. Dazu zählen unter anderem Buttons, Karten, Dialoge, Tabellen sowie weitere Elemente zur Gestaltung moderner Benutzeroberflächen [24]. Durch die Verwendung einer UI-Bibliothek müssen grundlegende Bedienelemente nicht selbst implementiert werden. Dadurch wird der Entwicklungsaufwand reduziert und ein konsistentes Erscheinungsbild der Anwendung unterstützt.

Neben den Komponenten bietet MudBlazor ein integriertes Theming-System sowie vordefinierte CSS-Hilfsklassen (Utility Classes) zur schnellen Anpassung von Layout und Darstellung [24]. Dieses Prinzip ist in seiner Zielsetzung vergleichbar mit Utility-First-Ansätzen wie Tailwind CSS, da Layout- und Stilregeln über wiederverwendbare Klassen direkt in der UI-Struktur angewendet werden.

Abbildung 17: MudBlazor Logo

6.4.6 MapLibre GL JS

MapLibre GL JS ist eine Open-Source-JavaScript-Bibliothek zur Darstellung interaktiver Karten in Webanwendungen. [25] Sie basiert auf WebGL, einer Webtechnologie, die es ermöglicht, Grafiken direkt über die Grafikkarte (GPU) im Browser zu rendern. Dadurch können auch komplexe und performante Kartenvisualisierungen flüssig dargestellt werden.

MapLibre GL unterstützt unter anderem das GeoJSON-Format (6.2.3), wodurch sich räumliche Daten strukturiert laden und darstellen lassen.

In der vorliegenden Arbeit wird MapLibre GL JS zur Visualisierung von GeoJSON-Daten im Frontend eingesetzt. Die Bibliothek erlaubt eine flexible und performante Darstellung geografischer Informationen und eignet sich insbesondere für Webanwendungen mit interaktiven Karten.

Abbildung 18: MapLibre Logo

6.5 Routing-Engines und deren Funktionsweise

6.5.1 OSRM – Open Source Routing Machine

Die Open Source Routing Machine (OSRM) ist eine hochperformante, in C++ geschriebene Routing-Engine, die speziell für die Berechnung von Wegen in komplexen Straßennetzwerken entwickelt wurde. Als Datengrundlage greift OSRM auf das detaillierte und frei verfügbare Kartenmaterial von OpenStreetMap (OSM) zurück. Dies ermöglicht es, Routenberechnungen völlig unabhängig von kommerziellen Anbietern durchzuführen und das Straßennetz flexibel als lokalen Dienst zu hosten.

Technisch nutzt OSRM fortschrittliche graphenbasierte Algorithmen wie Multi-Level Dijkstra (MLD) oder Contraction Hierarchies (CH), um den Suchraum bei der Routenfindung massiv zu verkleinern. Dadurch können selbst umfangreiche Distanzmatrizen oder streckengenaue Navigationsanweisungen innerhalb von Millisekunden berechnet werden. Neben der reinen Distanz und der geschätzten Fahrtzeit liefert die Engine auch die exakte Geometrie der berechneten Route, was für eine spätere visuelle Darstellung auf digitalen Karten unerlässlich ist [26].

Abbildung 19: OSRM Logo

6.5.2 VROOM – Routing-Optimierung

VROOM (Vehicle Routing Open-Source Optimization Machine) ist eine hochperformante, Open-Source Optimierungs-Engine, die in C++ entwickelt wurde. Sie ist darauf spezialisiert, komplexe logistische Herausforderungen wie das Vehicle Routing Problem (VRP) oder das Traveling Salesperson Problem (TSP) algorithmisch zu lösen. Im Gegensatz zu reinen Routing-Diensten, die lediglich den besten Weg zwischen zwei Punkten suchen, beantwortet VROOM übergeordnete logistische Fragen, wie beispielsweise: Welches Fahrzeug sollte welche Stationen in welcher Reihenfolge abfahren, um die Gesamteffizienz zu maximieren?

Um diese Zuteilung berechnen zu können, greift VROOM typischerweise auf externe Routing-Engines (wie beispielsweise OSRM) zurück, um im Vorfeld eine Matrix der exakten Reisezeiten oder Distanzen zwischen allen relevanten Punkten zu erstellen. Auf Basis dieser Matrix und unter strenger Berücksichtigung definierter Nebenbedingungen – wie etwa begrenzten Fahrzeugkapazitäten, maximal zulässigen Reisezeiten oder spezifischen Abhol-Zeitfenstern – ermittelt VROOM die bestmögliche Verteilung. Ziel der mathematischen Optimierung ist es in der Regel, die Gesamtzeit, die zurückgelegte Strecke oder die Anzahl der benötigten Fahrzeuge auf ein Minimum zu reduzieren [27] [7].

Abbildung 20: VROOM Logo

6.5.3 Nominatim – Geocoding und Adressauflösung

Nominatim ist eine quelloffene Suchmaschine, die speziell für die Suche nach Daten innerhalb der OpenStreetMap (OSM) entwickelt wurde. Ihr primärer Einsatzzweck ist das sogenannte Geocoding. Geocoding beschreibt den Prozess, bei dem menschenlesbare Adressdaten wie Straße, Hausnummer, Postleitzahl und Ort in eindeutige geografische Koordinaten (Breiten- und Längengrade) umgewandelt werden. Da Routing-Algorithmen und digitale Karten ausschließlich mit solchen exakten Koordinaten rechnen können, ist dieser Übersetzungsschritt eine zwingende technische Voraussetzung für geografische Anwendungen.

Zusätzlich zur normalen Adressauflösung unterstützt Nominatim auch das Reverse Geocoding. Dabei wird der umgekehrte Weg gegangen: Anhand einer gegebenen Koordinate auf der Karte ermittelt das System die nächstgelegene bekannte Adresse oder das entsprechende Gebäude. Da Nominatim vollständig auf dem freien Kartenmaterial von OpenStreetMap basiert, bietet es eine transparente und lizenzkostenfreie Möglichkeit, diese essenziellen Geodienste als eigenen Service zu betreiben, ohne auf kommerzielle Anbieter angewiesen zu sein [28].

Abbildung 21: Nominatim Logo

[7] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

6.6 Sonstige verwendete Software

Neben den bereits beschriebenen Technologien, Bibliotheken und Frameworks kamen im Projekt weitere Softwarelösungen zum Einsatz. Diese unterstützten insbesondere die Kommunikation, Projektorganisation, Dokumentation, Diagrammerstellung sowie die Bereitstellung und Verwaltung der Cloud-Umgebung.

6.6.1 Microsoft Teams

Microsoft Teams ist ein Online-Kommunikations- und Kollaborationstool des Unternehmens Microsoft. Es bietet Funktionen wie Chat, Audio- und Videokonferenzen sowie die gemeinsame Bearbeitung und den Austausch von Dateien. Durch die Integration in Microsoft 365 unterstützt Teams die Zusammenarbeit in Organisationen, insbesondere bei verteilten Teams und Projekten. [29]

Da Count IT Microsoft Teams als primäres Kommunikationsmedium verwendet, erfolgte die projektbezogene Abstimmung während der Diplomarbeit überwiegend über Teams.

Abbildung 22: Microsoft Teams Logo

6.6.2 Azure DevOps

Azure DevOps ist eine cloudbasierte Plattform von Microsoft, die Werkzeuge zur Planung, Entwicklung und Bereitstellung von Software bereitstellt [30]. Zu den zentralen Komponenten zählen unter anderem Azure Repos zur Quellcodeverwaltung , Azure Boards zur Aufgaben- und Projektplanung sowie Azure Pipelines zur automatisierten Build- und Deployment-Unterstützung. Dadurch können Entwicklungsprozesse strukturiert organisiert und Änderungen nachvollziehbar versioniert werden.

Abbildung 23: Azure DevOps Logo

6.6.3 Clockify

Clockify ist ein webbasiertes Tool zur Zeiterfassung. Es ermöglicht unter anderem die Protokollierung von Arbeitszeiten sowie die Auswertung von Tätigkeiten und Projektmitgliedern. Im Rahmen der Vorbereitung und des Schreibens dieser Diplomarbeit wurde Clockify zur Dokumentation der aufgewendeten Zeiten verwendet. [31]

Während des Praktikums erfolgte die Zeiterfassung jedoch über das interne Zeiterfassungssystem von Count IT.

Abbildung 24: Clockify Logo

6.6.4 Draw IO

Draw.io ist ein kostenloses Tool zur Erstellung verschiedener Diagrammtypen wie UML-Diagrammen, Flussdiagrammen und Systemübersichten. [32] Im Rahmen dieser Diplomarbeit wurde es zur Erstellung der in diesem Dokument dargestellten Übersichts- und Architekturdiagramme verwendet.

Abbildung 25: draw.io Logo

6.6.5 Azure Portal

Microsoft Azure ist eine Cloud-Computing-Plattform von Microsoft, die eine Vielzahl von Diensten zur Bereitstellung und zum Betrieb von Anwendungen bereitstellt [33]. Das Azure Portal ist die webbasierte Verwaltungsoberfläche (Unified Console), über die Azure-Ressourcen erstellt, konfiguriert, überwacht und verwaltet werden können [34].

Im Rahmen dieser Diplomarbeit wurde das Azure Portal projektbezogen genutzt, um cloudseitige Konfigurationen vorzunehmen. Dazu zählten insbesondere die Verwaltung von Identitäts- und Zugriffseinstellungen über Microsoft Entra ID [35] sowie die Anlage und Verwaltung der benötigten Hosting-Umgebung für containerisierte Anwendungsteile (z. B. für das Aspire-Projekt) [36].

Count IT verwendet das Azure Portal zudem generell für Deployment und Administration von Azure-Ressourcen. [8].

Abbildung 26: Logo Microsoft Azure

[8]Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

7 Implementierung

In diesem Kapitel wird die konkrete technische Umsetzung des Systems beschrieben. Dabei wird sowohl die Gesamtarchitektur als auch die Implementierung der einzelnen Komponenten, vom Backend über das Frontend bis hin zur Datenhaltung und dem Deployment, erläutert.

Die Anwendung wurde als verteiltes System innerhalb einer gemeinsamen Solution umgesetzt. Die einzelnen Bestandteile kommunizieren dabei über klar definierte Schnittstellen und folgen einer modularen, erweiterbaren Architektur.

7.1 Architektur des Systems

Zu Beginn wird die grundlegende Systemarchitektur vorgestellt, um einen Gesamtüberblick über die Struktur und das Zusammenspiel der einzelnen Komponenten zu geben. Die Architektur bildet die Basis für die anschließende detaillierte Beschreibung der Implementierung.

Die in Abbildung 27 dargestellte Architektur zeigt den Aufbau der Anwendung innerhalb einer containerisierten Umgebung in Microsoft Azure.

Im Zentrum der Architektur steht die REST-API, welche als zentrale Schnittstelle zwischen Frontend, Datenbank und den angebundenen Routing-Diensten fungiert. Das Frontend (Blazor Web App) kommuniziert ausschließlich über diese API, wodurch eine klare Trennung zwischen Benutzeroberfläche und Geschäftslogik gewährleistet wird.

Innerhalb der Container-Umgebung sind mehrere spezialisierte Dienste integriert: Der OSRM-Dienst übernimmt die Berechnung von Routen und Distanzen, während Vroom für die Optimierung von Fahrgemeinschaften zuständig ist. Nominatim wird für die Umwandlung von Adressen in geografische Koordinaten (Geocoding) verwendet. Diese Dienste sind voneinander getrennt, können jedoch über die API gemeinsam genutzt werden.

Die persistente Datenspeicherung erfolgt über eine SQL-Datenbank, welche über den SQL Server bereitgestellt wird. Zusätzlich ist Microsoft Entra ID für die Authentifizierung eingebunden, wodurch ein sicherer Zugriff auf die Anwendung gewährleistet wird.

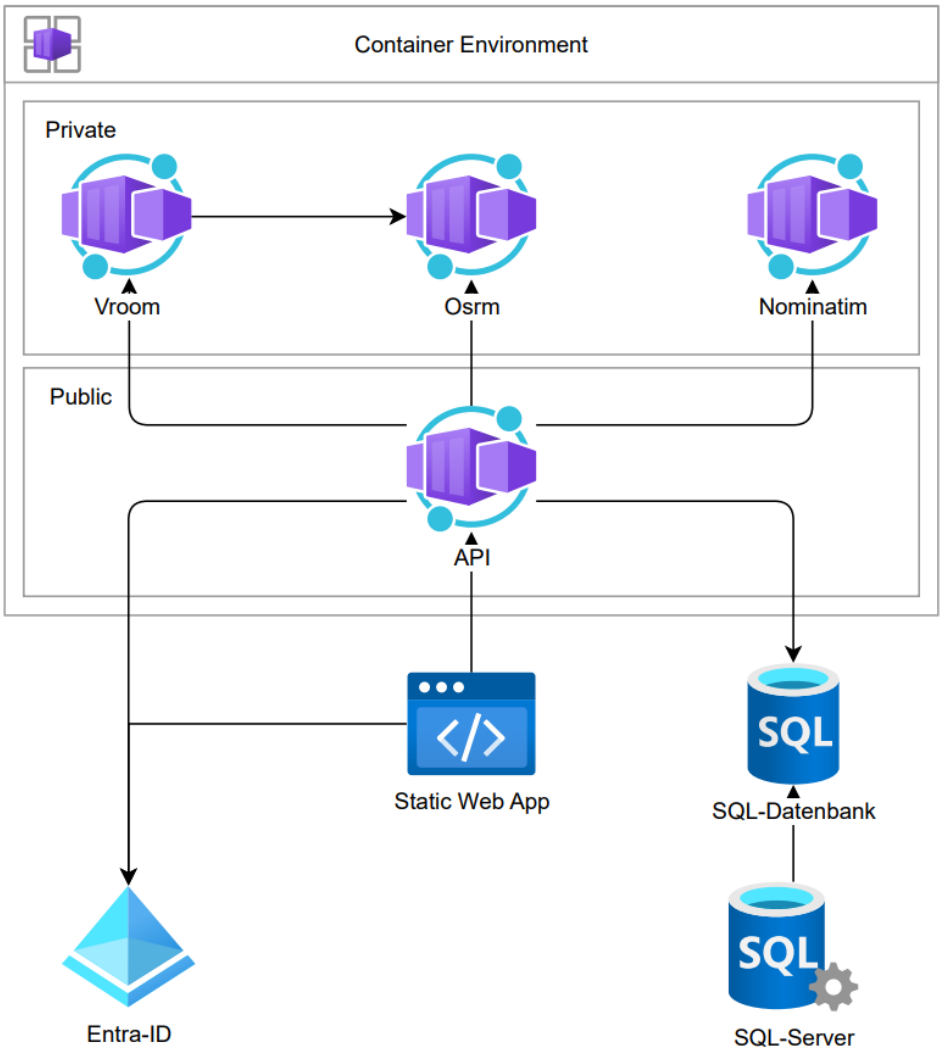


Abbildung 27: Architekturübersicht der Microsoft Azure-Container-Umgebung

7.1.1 Projektstruktur

Die Solution `CIT.Way2Work` ist als verteilte Anwendung konzipiert und nutzt *.NET Aspire* für die Orchestrierung der verschiedenen Dienste. Die zugrundeliegende Architektur folgt einem sauberen Schichtenmodell [37] und sorgt damit für eine strikte Trennung von Verantwortlichkeiten. Diese logische Trennung wird zusätzlich durch den konsequenten Einsatz des CQRS-Patterns (Unterabschnitt 7.2.1) unterstützt.

Im Folgenden werden die einzelnen Projekte der Solution und deren spezifische Aufgaben näher erklärt:

Projekt `CIT.Way2Work.AppHost` Dieses Projekt dient als zentraler Einstiegspunkt und Orchestrator für die gesamte verteilte Anwendung, basierend auf dem `IDistributedApplicationBuilder`. Es konfiguriert und startet sowohl die benötigten Container als auch die restlichen Teilprojekte:

- **Container-Dienste:** Hierzu zählen eine Azure SQL Edge Instanz (`sql`) für die persistente Datenspeicherung, der *Open Source Routing Machine* Container (`osrm`) für Routing und Distanzberechnungen, die Optimierungs-Engine *Vroom* (`vroom`) für Fahrgemeinschaften sowie der Geocoding-Dienst *Nominatim* (`nominatim`).
- **Projekt-Referenzen:** Der AppHost startet die API (`cit-way2work-api`), den Client (`cit-way2work-client`) und den Migrations-Service (`cit-way2work-data-migrationservice`) in der korrekten Startreihenfolge.

Projekt `CIT.Way2Work.API` Hierbei handelt es sich um das Backend der Anwendung, welches als ASP.NET Core Web API implementiert wurde.

- **Architektur:** Zur Umsetzung des CQRS-Patterns kommt die Bibliothek `MediatR` zum Einsatz. Commands (schreibende Operationen) und Queries (lesende Operationen) werden über klar definierte Handler verarbeitet.
- **Dienste:** Externe Services wie der `NominatimGeocodingService`, der `VroomRoutingService` und der `OsrmRoutingService` sind über typisierte `HttpClient`-Instanzen sauber angebunden.

Projekt `CIT.Way2Work.Client` Das Frontend der Anwendung ist als moderne Blazor Server Web App (`InteractiveServerComponents`) realisiert.

- **UI-Framework:** Für ein ansprechendes und responsives Design wird die Komponentenbibliothek `MudBlazor` verwendet.
- **Kommunikation & State:** Ein eigens entwickelter CQRS-Client übersetzt lokale Befehle direkt in HTTP-Aufrufe an die API. Die Statusverwaltung des aktuell angemeldeten Benutzers übernimmt ein zentraler `AppState`-Service.

Projekt `CIT.Way2Work.Domain.Contract` Dieses Projekt enthält die geteilten Vertragsdefinitionen (Contracts), die sowohl vom Client als auch von der API verwendet werden. Dies ist ein wesentlicher Baustein, um die Typsicherheit zwischen Frontend und Backend zu garantieren. Die Contracts sind dabei logisch in `Commands`, `Queries` und `Notifications` unterteilt. [Glossar Erklärung Contract]

Projekt `CIT.Way2Work.Domain.Logic` Hier ist das eigentliche Herzstück der Anwendung beheimatet: die Geschäftslogik. Dieses Projekt implementiert die `IRequestHandler` für die definierten Befehle und Abfragen (wie etwa den `UserQueryHandler`). Außerdem finden sich hier domänenspezifische Services, zum Beispiel der `CarpoolSuggestionService`, der für die Ermittlung der Fahrgemeinschaften zuständig ist.

Projekt `CIT.Way2Work.Data` Die Datenzugriffsschicht (Data Access Layer) kapselt die gesamte Kommunikation mit der Datenbank. Sie beinhaltet den `Way2WorkContext` für das Entity Framework Core, definiert die Datenbank-Entitäten und stellt Konfigurationen sowie AutoMapper-Profile für das Mapping zwischen Entitäten und DTOs (Data Transfer Objects) bereit.

Projekt `CIT.Way2Work.Data.MigrationService` Um das automatische Deployment zu erleichtern, wird ein bereitgestellter Worker-Service genutzt. Dieser wendet beim Start der Aspire-Umgebung vollautomatisch alle noch ausstehenden EF Core Migrationen auf die SQL-Datenbank an. Dadurch wird sichergestellt, dass das Datenbankschema ohne manuelle Eingriffe von Entwicklerseite immer synchron zur aktuell ausgeführten Code-Version bleibt.

Projekt `CIT.Way2Work.ServiceDefaults` Dieses Hilfsprojekt bündelt Standardkonfigurationen für OpenTelemetry (Logging, Metriken und Tracing) sowie Health-Checks. Damit wird eine

umfassende Observability der verteilten Dienste gewährleistet, was bei einer containerisierten Anwendung unerlässlich ist. [9]

7.2 Design Patterns

Um eine zukunftssichere, wartbare und vor allem übersichtliche Codebasis zu schaffen, stützt sich die Architektur von Way2Work maßgeblich auf etablierte Software-Entwurfsmuster. Im Zentrum unserer Implementierung stehen dabei das CQRS-Pattern (Command Query Responsibility Segregation) sowie das Mediator-Pattern. Diese sorgen für eine lose Kopplung der Komponenten und eine klare Trennung der Verantwortlichkeiten.

7.2.1 CQRS-Aufbau und Datenfluss

In traditionellen Systemarchitekturen wird häufig dasselbe Datenmodell sowohl für das Lesen als auch für das Schreiben von Daten verwendet. Bei komplexeren Domänen führt das schnell zu unübersichtlichem Code und Kompromissen beim Datenbankzugriff. Das CQRS-Pattern [38] löst dieses Problem durch eine strikte architektonische Trennung von Datenmanipulation und Datenabfrage.

Wie in Abbildung 28 dargestellt, teilt CQRS die Logik in zwei völlig unabhängige Bereiche auf:

- **Commands (Schreibseite):** Ein Command repräsentiert die klare Absicht, den Zustand des Systems zu verändern (beispielsweise `CreateUser` oder `UpdateCarpoolStatus`). Hier findet die Validierung der Eingaben und die Ausführung der eigentlichen Geschäftslogik statt. Ein Command liefert im Idealfall keine komplexen Datenstrukturen zurück, sondern lediglich eine Bestätigung oder die ID des neu erstellten Objekts.
- **Queries (Leseseite):** Queries dienen im Gegensatz dazu ausschließlich dem Abrufen von Daten (wie `GetUsers` oder `GetCarpool`). Sie verändern niemals den Zustand der Anwendung (sie sind seiteneffektfrei) und können so optimal auf schnelle Lesezugriffe zugeschnitten werden.

Ein wesentlicher Vorteil dieses Patterns ist die Möglichkeit, die Datenhaltung auch physisch zu trennen. So könnten für Lese- und Schreibzugriffe zwei unterschiedliche, jeweils für ihren Zweck optimierte Datenbanken verwendet werden, die im Hintergrund synchronisiert werden. Auch

[9] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

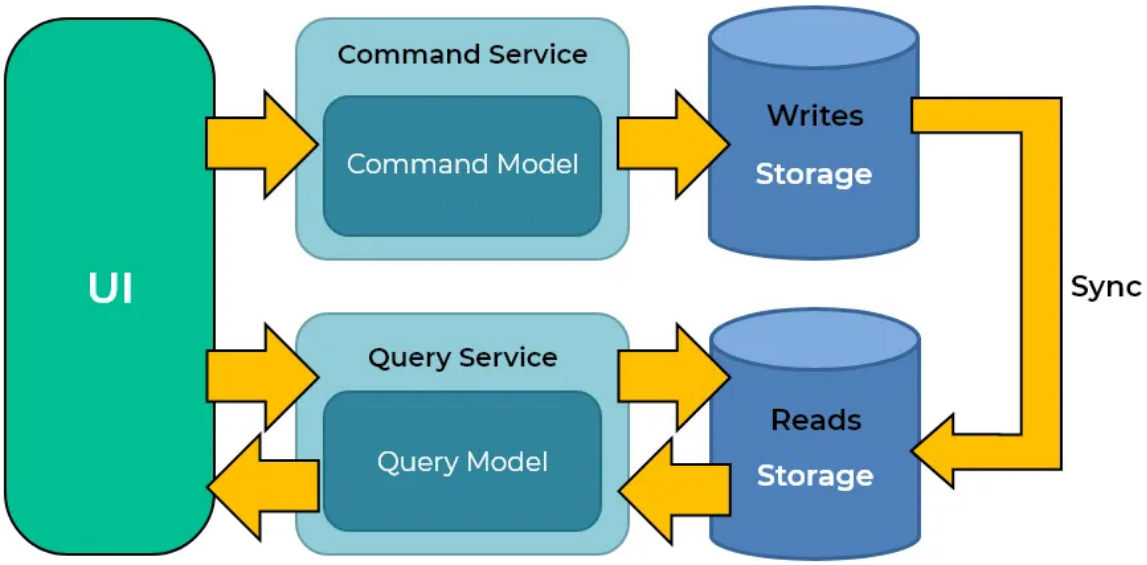


Abbildung 28: CQRS Pattern Overview [39]

wenn in Way2Work aktuell eine gemeinsame relationale Datenbank genutzt wird, schafft die Architektur bereits die Voraussetzung für eine solche Skalierung.

Umsetzung im Projekt Diese theoretische Trennung spiegelt sich direkt in unserer Projektstruktur wider. Im Projekt `CIT.Way2Work.Domain.Contract` werden die Schnittstellen für sämtliche Operationstypen vertraglich über spezifische Interfaces definiert: Befehle implementieren das `ICommand<T>`-Interface und Abfragen das `IQuery<T>`-Interface (z. B. als `record CreateUser(...) : ICommand<User>`).

Die eigentliche Ausführung passiert im Projekt `CIT.Way2Work.Domain.Logic`, wo die zugehörigen Handler implementiert sind. Nimmt beispielsweise der `UserCommandHandler` den Befehl `CreateUser` entgegen, so kümmert er sich um das Mapping auf die Datenbank-Entität, speichert diese über das Entity Framework (`Way2WorkContext`) und gibt das fertige Domänen-Modell zurück.

7.2.2 Das Mediator-Pattern

Damit die Trennung durch CQRS im Code sauber nutzbar bleibt und Komponenten (wie API-Controller, Blazor-Seiten und Service-Klassen) nicht starr aneinander gekoppelt sind, setzen wir ergänzend das Mediator-Pattern ein. Der Mediator fungiert dabei als zentrale Vermittlungsstelle.

Funktionsweise und Vorteile Anstatt dass ein API-Endpunkt eine Service-Klasse direkt instanziiert oder per Dependency Injection anfordert, erstellt er lediglich ein Anfrage-Objekt (einen Command oder eine Query) und übergibt dieses an den Mediator. Der Mediator ermittelt zur Laufzeit dynamisch, welcher Handler für diese spezifische Anfrage zuständig ist, und leitet sie dorthin weiter. Nach der Verarbeitung durch den Handler wird das Ergebnis an den Aufrufer durchgereicht.

Dieser Ansatz bringt für unser Projekt entscheidende Vorteile:

- **Entkopplung:** Der Aufrufer (z. B. das Frontend oder der API-Controller) muss weder wissen, wer die Anfrage verarbeitet, noch wie die konkrete Implementierung der Logik aussieht.
- **Single Responsibility Principle:** Jeder Handler hat exakt eine Aufgabe und ist nur für einen spezifischen Anwendungsfall verantwortlich.
- **Erweiterbarkeit:** Neue Features lassen sich unkompliziert hinzufügen, indem einfach ein neuer Command und der dazugehörige Handler geschrieben werden. Bestehender Code muss dafür nicht verändert werden.

Technische Umsetzung mit MediatR Technisch haben wir dies mit der populären .NET-Bibliothek **MediatR** gelöst. Dabei greifen wir auf eine zusätzliche Abstraktionsschicht (`CIT.CCL.CQRS.Media` zurück, die von unserem Projektpartner Count IT zur Verfügung gestellt wurde. Diese stellt sicher, dass das Pattern über alle Firmensysteme hinweg einheitlich und standardisiert angewendet wird.

Jeder Handler in unserer Geschäftslogik implementiert das entsprechende Interface (z. B. `IRequestHandler<TRequest, TResponse>`). Die Registrierung aller Handler im Dependency Injection Container erfolgt dabei vollautomatisch beim Start der API (in der `Program.cs`).

Ein weiteres wichtiges Konzept, das wir über MediatR nutzen, sind **Notifications** (unter Verwendung des `INotification`-Interfaces). Damit können wir domain-interne Events auslösen (etwa `CarpoolRelevantChanged`), auf die dann mehrere andere Handler völlig unabhängig voneinander reagieren können, ohne den eigentlichen Ablauf zu blockieren. [10]

[10] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

7.3 Datenbank und Migration

Die Datenbank in unserer Diplomarbeit ist ein Microsoft SQL Server und dieser wird im Projekt vollständig containerisiert betrieben. Es ist daher keine separate lokale Installation eines SQL Servers erforderlich. Stattdessen wird der Datenbank-Container, wie auch die übrigen Services, über *.NET Aspire* orchestriert und zur Laufzeit bereitgestellt.

7.3.1 SQL Server Konfiguration

Der SQL Server wird im Projekt `CIT.Way2Work.MigrationService` als Ressource eingebunden. Über den Aspire-Resource-Namen `sqldb-way2work` wird der Connection String automatisch in die Anwendung injiziert. Dadurch entfällt eine manuelle Konfiguration von Host, Port und Credentials im Quellcode und die Anwendung kann in unterschiedlichen Umgebungen konsistent betrieben werden.

Die Registrierung des Entity-Framework-Kontexts erfolgt in `Program`.cs.
Mit `AddSqlServerDbContext<Way2WorkContext>(...)` wird der `DbContext` an den durch Aspire bereitgestellten SQL Server angebunden. Zusätzlich wird ein `HostedService` registriert, der beim Start der Applikation die Datenbankmigrationen ausführt und so sicherstellt, dass das Schema dem aktuellen Stand der Anwendung entspricht.

7.3.2 Data Migration Service

Für die Datenbank wurde ein **Code-First**-Ansatz mit Entity Framework Core gewählt. Das Datenbankschema ergibt sich damit direkt aus den C#-Entitäten und deren Data Annotations. Änderungen am Datenmodell werden über EF Core Migrationen versioniert und reproduzierbar ausgerollt.

Neue Migrationen werden über die **NuGet Package Manager Console** erzeugt:

Listing 1: Erstellen einer EF Core Migration

```
1 dotnet ef migrations add <NameDerMigration>
```

Die Migrationen werden in der Entwicklung typischerweise mit `dotnet ef database update` angewendet. Im produktionsnahen Betrieb übernimmt dies der `Data Migration Service` automatisiert beim Start.

Kernstück ist die Datei `DatabaseMigrator.cs`. Sie sorgt beim Start der Anwendung dafür, dass die Datenbank erreichbar ist und automatisch auf den neuesten Stand gebracht wird, indem alle noch fehlenden Migrationen angewendet werden.

Damit wird sichergestellt, dass die Datenbank beim Hochfahren der Umgebung automatisch auf dem erwarteten Schema-Stand ist.

7.3.3 Datenmodell

Das Datenbankschema (Abbildung 29) ist als normalisiertes, relationales Modell ausgeführt. Zur besseren Konsistenz und Lesbarkeit wurden die Tabellen- und Spaltennamen in Englisch gehalten. Primärschlüssel werden durchgehend als `uniqueidentifier` realisiert.

Entität `CompanyInformations` In dieser Tabelle werden Stammdaten des Unternehmens gespeichert, insbesondere Adresse, Koordinaten (Latitude/Longitude). Zusätzlich wird ein kumulierter Wert für `TotalCo2Savings` gespeichert welche im Dashboard als Kennzahl angezeigt wird. Die Entität steht unabhängig im Modell hat also keine Fremdschlüsselbeziehungen und dient als globale Konfiguration bzw. Auswertungsbasis.

Entität `Users` Die Tabelle `Users` bildet die zentrale Benutzer-Entität ab und speichert Basisinformationen wie `DisplayName` und `Email`. Über `Points` wird der aktuelle Reward-Punktestand geführt. Von `Users` gehen mehrere 1:n-Beziehungen aus, da ein Benutzer mehrere Wohnorte, Fahrzeuge und Schedule-Einträge besitzen kann.

Entität `Residences` `Residences` speichert die vom Benutzer erfassten Wohnorte. Über eine 1:n-Beziehung ist jeder Wohnort genau einem Benutzer (`UserId`) zugeordnet. Beim Erstellen einer Residence werden die genauen Adressdaten wie Ort, Straße und Hausnummer erfasst. Vor dem Speichern wird die Adresse mittels Nominatim in Geokoordinaten umgerechnet.

Entität `Vehicles` In `Vehicles` werden die vom Benutzer verwalteten Fahrzeuge gespeichert (Name, Typ, Sitzplätze). Optional wird über `Co2PerKm` ein Emissionswert hinterlegt, der für CO_2-Berechnungen genutzt werden kann, um die Berechnung der eingesparten Emissionen genauer zu machen, wenn man bei jemandem mitfährt. Auch hier besteht eine 1:n-Beziehung von `Users` zu `Vehicles`.

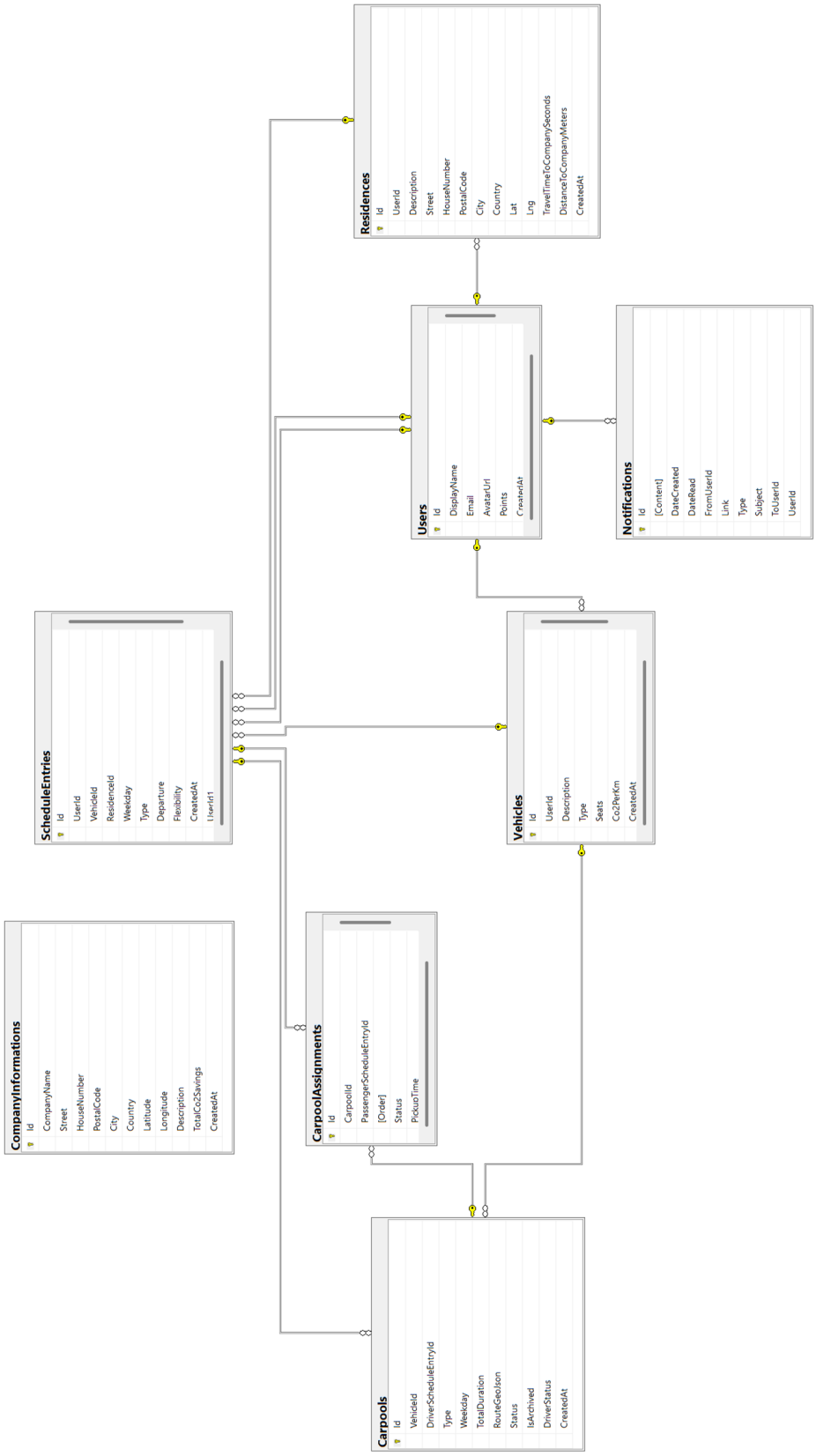


Abbildung 29: Finales Entity-Relationship-Diagramm (ERD)

Entität `ScheduleEntries` Die Tabelle `ScheduleEntries` bildet den wöchentlichen Fahrplan ab und ist die Grundlage für das Matching. Ein `ScheduleEntry` gehört über `UserId` genau zu einem Benutzer. Über `ResidenceId` kann der Start-/Wohnort referenziert werden (Wohnort als Ausgangspunkt), und über `VehicleId` wird optional angegeben ob man Fahrer sein kann oder wenn diese Id NULL ist, ist man nur mitfahrer. Mit `Weekday`, `Type` (z. B. Hin-/Rückfahrt), `Departure` und `Flexibility` werden Zeitfenster und Toleranzen technisch abgebildet.

Entität `Carpools` `Carpools` repräsentiert eine berechnete oder verwaltete Fahrgemeinschaft für einen bestimmten Wochentag und Fahrt-Typ. Ein Carpool referenziert das Fahrerfahrzeug (`VehicleId`) sowie den Fahrplan-Eintrag des Fahrers (`DriverScheduleEntryId`). Damit bestehen n:1-Beziehungen zu `Vehicles` und `ScheduleEntries`. Für die Kartenansicht wird die Route als `RouteGeoJson` gespeichert; `TotalDuration` hält die Gesamtdauer. Der Zustand wird über Felder wie `Status`, `DriverStatus` und `IsArchived` abgebildet, um den Lebenszyklus (Vorschlag, aktiv, abgeschlossen, archiviert) nachvollziehbar zu machen.

Entität `CarpoolAssignments` `CarpoolAssignments` ist die Zuordnungstabelle zwischen einer Fahrgemeinschaft und den Mitfahrern. Sie verknüpft einen `Carpool` mit einem Fahrplan-Eintrag eines Mitfahrers (`PassengerScheduleEntryId`). Da ein Carpool mehrere Mitfahrer enthalten kann und ein ScheduleEntry (insbesondere im Matching-Kontext) potenziell in mehreren Vorschlägen auftauchen kann, wird hier eine n:m-Beziehung zwischen `Carpools` und `ScheduleEntries` modelliert. Die Tabelle enthält zusätzlich fachliche Attribute wie `Status` (z. B. angefragt/akzeptiert/abgelehnt), `PickupTime` sowie `Order` zur Reihenfolge der Abholpunkte.

Entität `Notifications` In `Notifications` werden Benachrichtigungen gespeichert (Inhalt, Typ, Betreff, Link) sowie Zeitpunkte für Erstellung und Lesen (`DateCreated`, `DateRead`). Eine Benachrichtigung hat typischerweise einen Sender (`FromUserId`) und einen Empfänger (`ToUserId`) und referenziert damit Benutzer. Zusätzlich existiert eine optionale FK-Spalte `UserId`, die als technische Zuordnung/Navigationsbeziehung geführt wird. Damit lässt sich Kommunikation im Anfrage-Workflow persistent nachvollziehen. Notifications werden allerdings in der Aktuellen App noch nicht genutzt, bzw. sind noch nicht vollständig implementiert.

Entität `__EFMigrationsHistory` Diese Tabelle wird von Entity Framework Core intern genutzt, um angewendete Migrationen inklusive Produktversion zu protokollieren. Sie ist nicht

fachlich Teil des Domänenmodells, aber notwendig für die automatische Datenbankmigration beim Start. [11].

7.4 Backend – API und Routing

Nachdem im vorherigen Abschnitt das statische Datenmodell und die Struktur der Datenbankentitäten definiert wurden, widmet sich dieses Kapitel der dynamischen Verarbeitung dieser Daten. Die Backend-API bildet das logische Herzstück von Way2Work: Sie nimmt die Anfragen des Frontends entgegen, führt die Geschäfts- und Routing-Logik aus und interagiert über das Entity Framework mit der zuvor beschriebenen Datenbank.

7.4.1 Aufbau der API-Schichten

Die Architektur unserer API ist logisch in drei Hauptebenen unterteilt: die Präsentationsschicht (API), die Logikschicht (Domain Logic) und die Datenzugriffsschicht (Data). Durch die konsequente Anwendung des zuvor beschriebenen CQRS-Patterns in Kombination mit MediatR weicht unsere Implementierung bewusst von klassischen MVC-Architekturen (Model-View-Controller) ab und setzt stattdessen auf Handler.

Präsentationsschicht (API-Ebene) Diese Schicht dient als reiner Einstiegspunkt (Entry Point) für externe Anfragen. Eine Besonderheit unserer Umsetzung ist, dass wir auf das manuelle Schreiben von klassischen Controller-Klassen verzichten. Stattdessen nutzt das Projekt die Bibliothek `CIT.CCL.CQRS.MediatR.AspNetCore`. Diese Bibliothek ermöglicht es, HTTP-Endpunkte beim Start der Anwendung (in der `Program.cs`) automatisch auf Basis der registrierten Commands und Queries zu generieren. Ein HTTP-GET-Request an die Route `/api/users` wird so beispielsweise dynamisch an den passenden `UserQueryHandler` geroutet, ohne dass dafür expliziter Controller-Code gewartet werden muss.

Logikschicht (Service-Ebene) In dieser Schicht findet die eigentliche Geschäftsverarbeitung statt. Die Logik ist dabei grob in zwei Kategorien unterteilt: MediatR-Handler und spezialisierte Domain-Services. Jeder spezifische Anwendungsfall (Use Case) wird durch genau einen **MediatR-Handler** repräsentiert, der das Interface `IRequestHandler` implementiert. Listing 2 zeigt

[11] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

beispielhaft, wie ein solcher Handler einen eingehenden Befehl zur Benutzererstellung verarbeitet, über AutoMapper in eine Datenbankentität umwandelt und persistiert.

Listing 2: Beispiel eines CommandHandlers für die Benutzererstellung

```
public class UserCommandHandler(Way2WorkContext dbContext, IMapper mapper)
    : IRequestHandler<CreateUser, User>
{
    public async Task<User> Handle(CreateUser request, CancellationToken cancellationToken)
    {
        // 1. Mapping des Commands auf die Datenbank-Entität
        var entityToSave = mapper.Map<Data.Entities.User>(request.User);
        entityToSave.CreatedAt = DateTime.UtcNow;

        // 2. Persistierung in der Datenbank via EF Core
        var entity = (await dbContext.AddAsync(entityToSave, cancellationToken)).Entity;
        await dbContext.SaveChangesAsync(cancellationToken);

        // 3. Rückgabe des erstellten Domain-Models
        return mapper.Map<User>(entity);
    }
}
```

Für domänenspezifische Aufgaben, die zu komplex für einen einfachen Handler sind oder externe Systeme einbinden, kommen zusätzliche **Domain-Services** zum Einsatz. Ein zentrales Beispiel hierfür ist der `CarpoolSuggestionService`. Dieser kapselt die aufwendige Geschäftslogik zur täglichen Ermittlung und Generierung von Fahrgemeinschaftsvorschlägen und delegiert Teilaufgaben an den `VroomRoutingService`.

Datenzugriffsschicht (Data-Ebene) Die unterste Schicht ist für die Persistierung und das Laden der Daten zuständig. Dies erfolgt vollständig über das ORM-Framework (Object-Relational Mapper) *Entity Framework Core*. Der `Way2WorkContext` repräsentiert dabei die Datenbanksitzung und definiert die `DbSet`-Eigenschaften für sämtliche Entitäten wie `Users`, `Carpools` oder `ScheduleEntries`. Neben der reinen Bereitstellung der Tabellen werden hier über die überschriebene Methode `OnModelCreating` auch essenzielle Beziehungen und Einschränkungen konfiguriert. So wird beispielsweise über ein `DeleteBehavior.Restrict` technisch verhindert, dass das versehentliche Löschen eines Fahrplans (Schedule Entry) automatisiert alle damit verknüpften Fahrgemeinschaften (Carpools) aus der Datenbank entfernt (Cascading Deletes).

7.4.2 Kommunikation mit Routing-Containern

Die rechenintensiven Routing- und Optimierungsalgorithmen werden in unserer Architektur bewusst nicht direkt innerhalb der .NET-Anwendung ausgeführt. Stattdessen haben wir diese komplexen Aufgaben an Microservices ausgelagert, die als eigenständige Container bereitgestellt werden. Zum Einsatz kommen dabei OSRM für die reinen Distanzberechnungen und die Routenführung sowie VROOM für die eigentliche mathematische Optimierung der Fahrgemeinschaften.

Orchestrierung über .NET Aspire Die Bereitstellung und Verwaltung dieser Dienste übernimmt das `AppHost`-Projekt mithilfe von *.NET Aspire*. Dies garantiert eine absolut konsistente Umgebung – von der lokalen Entwicklung bis hin zum produktiven Deployment. Der OSRM-Container wird dabei so konfiguriert, dass er über einen Bind-Mount auf lokal vorprozessierte Kartendaten (in unserem Fall das Straßennetz von Oberösterreich) zugreift. Er startet standardmäßig mit dem performanten MLD-Algorithmus (Multi-Level Dijkstra). Der Vroom-Container wird parallel dazu hochgefahren und über Umgebungsvariablen direkt mit dem OSRM-Container verknüpft. Dadurch weiß Vroom, dass es für die Berechnung der zugrundeliegenden Distanzmatrizen auf den lokalen OSRM-Dienst zurückgreifen soll.

Anbindung in der API Innerhalb der Geschäftslogik (API) erfolgt die Kommunikation mit diesen Containern asynchron über typisierte `HttpClient`-Instanzen. Diese werden beim Start der Anwendung per Dependency Injection registriert und mit den entsprechenden Basis-URLs der Container vorkonfiguriert.

Für die Erstellung der Fahrgemeinschaften ist primär der `VroomRoutingService` zuständig. Er bereitet die Optimierungsanfragen auf, indem er die verfügbaren Fahrzeuge, die zu transportierenden Personen (in Vroom als *Jobs* bezeichnet) sowie deren zeitliche Flexibilität in ein JSON-Format serialisiert. Dieser Payload wird an den Vroom-Endpunkt gesendet, welcher nach der Berechnung die optimalen Zuordnungen und Abholreihenfolgen zurückliefert.

Nachdem Vroom die optimale Reihenfolge ermittelt hat, fehlt jedoch noch die exakte Wegstrecke für die visuelle Darstellung auf der Karte. An dieser Stelle kommt der `OsrmRoutingService` ins Spiel. Er nimmt die sortierten Koordinaten der Route entgegen und fragt bei OSRM die genaue Geometrie ab. Listing 3 zeigt die praktische Umsetzung dieses Aufrufs, bei dem am Ende ein fertiges GeoJSON-Objekt aus der HTTP-Antwort extrahiert wird.

Listing 3: Abruf der Routengeometrie (GeoJSON) im OsrmRoutingService

```
1  public class OsrmRoutingService(HttpClient httpClient) : IOsrmRoutingService
2  {
3      public async Task<string> GetRouteGeoJsonAsync(List<double[]> coordinates)
4      {
5          // 1. Formatierung der Koordinaten für die URL (lon,lat;lon,lat)
6          var coordString = string.Join(";", coordinates.Select(c => $"{c[0]},{c[1]}"));
7
8          // 2. Abfrage der Routengeometrie über die OSRM API
9          var url = $"/route/v1/car/{coordString}?overview=full&geometries=geojson";
10         var response = await httpClient.GetAsync(url);
11         response.EnsureSuccessStatusCode();
12
13         // 3. Extraktion des GeoJSON-LineStrings aus der JSON-Antwort
14         using var doc = JsonDocument.Parse(await response.Content.ReadAsStringAsync());
15         return doc.RootElement
16                   .GetProperty("routes")[0]
17                   .GetProperty("geometry")
18                   .GetRawText();
19     }
20 }
```

7.4.3 Routenoptimierung und -speicherung

Der Kernbestandteil und die größte technische Herausforderung dieser Diplomarbeit ist die automatische Ermittlung von optimalen Fahrgemeinschaften. Dieser Prozess wird typischerweise immer dann angestoßen, wenn ein Mitarbeiter Änderungen an seinem Wochenplan (Weekly-Schedule) vornimmt, woraufhin die potenziellen Fahrgemeinschaften für den jeweiligen Tag neu berechnet werden. Die Orchestrierung dieses gesamten Ablaufs, vom Sammeln der Pendlerdaten über die Optimierung bis hin zur Speicherung, übernimmt der `CarpoolSuggestionService`.

Das System stützt sich dabei auf das Zusammenspiel von zwei spezialisierten Diensten: VROOM für die mathematische Zuteilung und Reihenfolge sowie OSRM für die tatsächliche Streckenberechnung. Beide Systeme ergänzen sich perfekt: Vroom benötigt im Hintergrund eine OSRM-API für Distanzmatrizen, und das Ergebnis von Vroom muss im Anschluss noch einmal durch einen OSRM-Request verarbeitet werden, da Vroom lediglich die logische Reihenfolge der abzuholenden Personen liefert, nicht aber die exakte Wegstrecke für die Kartendarstellung.

Datenaufbereitung und Constraints (Vroom) Da Vroom ein generisches Routing-Problem (Vehicle Routing Problem) löst, müssen die domänenspezifischen Daten der Mitarbeiter zunächst transformiert werden. Der Service filtert alle relevanten Zeitplaneinträge (`ScheduleEntries`) für den jeweiligen Wochentag und unterteilt diese in Fahrer (in Vroom als *Vehicles* bezeichnet) und reine Mitfahrer (in Vroom als *Jobs* bezeichnet). Dabei werden essenzielle Nebenbedingungen (Constraints) berechnet und an Vroom übergeben:

- **Kapazität:** Die Anzahl der verfügbaren Sitzplätze im Fahrzeug des Fahrers (abzüglich des Fahrers selbst).
- **Zeitfenster:** Ein Toleranzbereich, der sich aus der gewünschten Ankunfts- oder Abfahrtszeit und der vom Benutzer hinterlegten zeitlichen Flexibilität ergibt.
- **Maximale Reisezeit:** Um unzumutbare Umwege für den Fahrer zu vermeiden, wird die Reisezeit dynamisch limitiert. So wird beispielsweise konfiguriert, dass die Fahrt maximal 30 % länger dauern darf als die direkte Strecke zum Unternehmen.

Durchführung der Optimierung Diese aufbereiteten Listen von Fahrzeugen und Jobs sowie der Unternehmensstandort als festes Ziel werden in ein JSON-Anfrageobjekt verpackt und an den Vroom-Container gesendet. Als primäres Optimierungsziel wird dabei definiert, die Anzahl der benötigten Fahrzeuge zu minimieren (`minimize: "vehicles"`). Vroom berechnet daraufhin

die effizientesten Zuteilungen und liefert eine Liste von Routen zurück, in der genau definiert ist, welcher Fahrer welche Mitfahrer in welcher Reihenfolge abholt.

Routenberechnung und Persistierung Die von Vroom zurückgelieferten Ergebnisse müssen nun in das Domänenmodell von Way2Work übersetzt werden. Um Duplikate zu vermeiden, prüft der Service vorab, ob für eine exakt gleiche Konstellation bereits ein Vorschlag in der Datenbank existiert.

Für jede gefundene Fahrgemeinschaft werden die sortierten Koordinaten der Abholpunkte an den `OsrmRoutingService` übergeben. Dieser liefert einen exakten GeoJSON-LineString zurück, der es später im Frontend ermöglicht, die Straßenroute präzise auf der MapLibre-Karte einzuzeichnen. Abschließend wird die Fahrgemeinschaft in der Entität `Carpool` samt ihren Zuweisungen (`CarpoolAssignments`) in der Microsoft SQL-Datenbank persistiert. Listing 4 zeigt, wie diese Entität im Code zusammengesetzt und über das Entity Framework gespeichert wird.

Listing 4: Zusammensetzen und Speichern einer berechneten Fahrgemeinschaft

```
var newCarpool = new Carpool
{
    DriverScheduleEntryId = driverEntry.Id,
    VehicleId = driverEntry.VehicleId!.Value,
    Weekday = weekday,
    Type = type,
    Status = CarpoolStatus.Suggestion, // Wird initial als Vorschlag markiert

    // Abruf der exakten Geometrie für das Frontend via OSRM
    RouteGeoJson = await osrmRoutingService.GetRouteGeoJsonAsync(routeCoordinates),

    // Zuweisung der Mitfahrer
    Assignments = passengerIds.Select(pId => new CarpoolAssignment
    {
        PassengerScheduleEntryId = pId,
        Status = CarpoolParticipantStatus.Pending
    }).ToList()
};

dbContext.Carpools.Add(newCarpool);
await dbContext.SaveChangesAsync(cancellationToken);
```

7.5 Frontend – Blazor WebApp

Die Webanwendung von Way2Work wurde als Blazor Web App innerhalb der .NET-9-Solution umgesetzt. Ziel des Frontends war es, eine moderne, klar strukturierte und für verschiedene Endgeräte geeignete Benutzeroberfläche bereitzustellen, über die Funktionen wie Matching, Kartenansicht, Fahrgemeinschaftsverwaltung, Reward-System und Authentifizierung benutzerfreundlich genutzt werden können. Die Herausforderung lag dabei weniger in einzelnen komplexen Algorithmen, sondern vielmehr in der sauberen Kombination mehrerer Technologien wie Blazor, MapLibre, GeoJSON und Microsoft Entra ID.

7.5.1 Komponentenaufbau und Layout

Das Frontend wurde komponentenbaspiert aufgebaut. Die einzelnen Bereiche der Anwendung sind in eigenständige Razor-Komponentent gegliedert, wodurch Seitenlogik, Layout und wiederverwendbare UI-Elemente sauber voneinander getrennt bleiben.

Routing Für das Routing der Anwendung wird die in Blazor integrierte Router-Komponente verwendet. Der folgende Codeausschnitt aus der Datei `Routes.razor` zeigt, wie Seiten anhand der Route geladen und innerhalb des Hauptlayouts dargestellt werden.

Listing 5: Routing der Anwendung

```
<Router AppAssembly="typeof(Program).Assembly">
    <Found Context="routeData">
        <AuthorizeRouteView RouteData="routeData"
                            DefaultLayout="typeof(Layout.MainLayout)" />
        <FocusOnNavigate RouteData="routeData" Selector="h1" />
    </Found>
</Router>
```

Gemeinsame Basiskomponente Neben dem Routing wurde auch eine gemeinsame Basiskomponente eingeführt. Diese bündelt wiederkehrende Aufgaben wie das Senden von Commands, die Anzeige von Erfolgsmeldungen sowie den Zugriff auf zentrale Dienste. Abbildung 30

Listing 6: Zentrale Basiskomponente für Commands

```
[Inject] protected ISender Sender { get; set; } = null!;
[Inject] protected ISnackbar Snackbar { get; set; } = null!;

protected async Task<TResult?> ExecuteCommand<TCommand, TResult>(
    TCommand command, string? successMessage = null)
    where TCommand : ICommand<TResult>
{
    var result = await Sender.Send(command);
    if (successMessage is not null)
        _ = Snackbar.Add(successMessage, Severity.Success);
    return result;
}
```

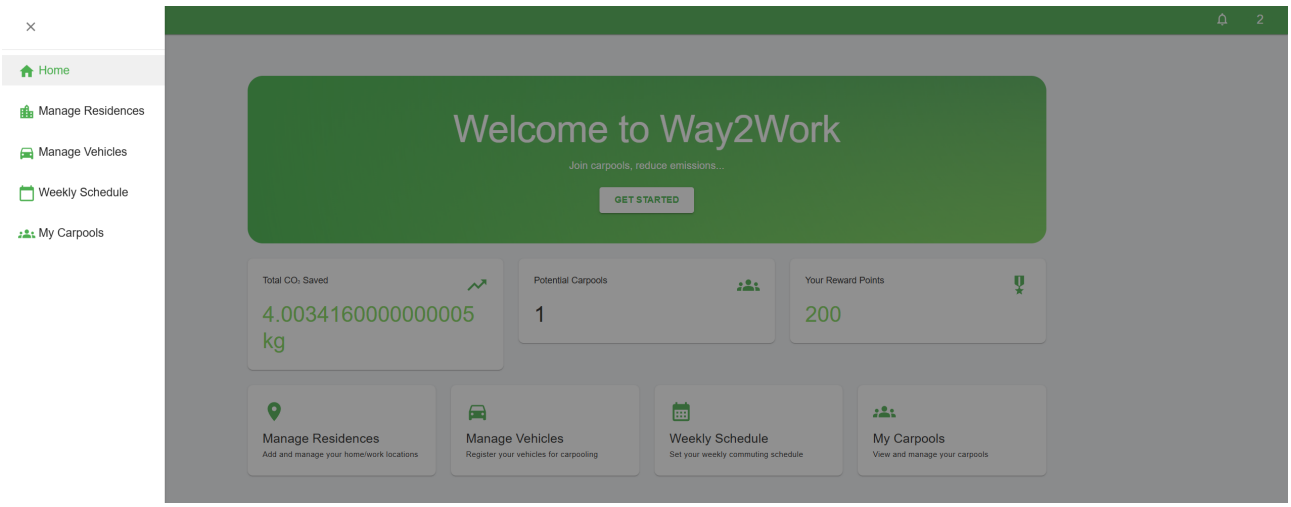


Abbildung 30: Layout

Die Klasse `MainComponent` dient damit als technische Grundlage für die restlichen Seiten des Frontends. Gerade bei den Formularen und Verwaltungsseiten ist dies vom Vorteil, da Backend-Aufrufe, Erfolgsmeldungen und weitere Standardabläufe nicht in jeder Komponente erneut implementiert werden müssen. Dadurch bleiben die einzelnen Seiten übersichtlich und konsistent aufgebaut.

Zentrales Seitenlayout Das grundlegende Layout der Anwendung wird durch die Komponente `MainLayout` definiert. Diese stellt die gemeinsame Struktur für alle Seiten bereit und enthält zentrale UI-Elemente wie Navigation, Seitenleiste und Inhaltsbereich.

Der folgende Codeausschnitt aus der Code-Behind-Datei zeigt, dass im Layout insbesondere der globale Anwendungszustand eingebunden und die Umschaltung des Darkmode verarbeitet wird.

Listing 7: Zentrales Seitenlayout der Anwendung

```
1  [Inject]
2  public AppState AppState { get; set; } = null!;
3
4  [CascadingParameter]
5  private Task<AuthenticationState> AuthenticationStateTask { get; set; } = default!;
6
7  private MudThemeProvider? _mudThemeProvider;
8
9  protected override async Task OnInitializedAsync()
10 {
11     AppState.MajorUpdateOccured += OnAppStateChanged;
12 }
13
14 private void OnDarkModeChanged(bool args) => AppState.ToggleDarkMode();
```

Die Komponente `MainLayout` stellt damit zentrale Funktionen wie den Zugriff auf den globalen Anwendungszustand (`AppState`) sowie die Steuerung des Darkmode bereit. Änderungen im App-State führen dabei zu einer Aktualisierung der Oberfläche

Die eigentliche visuelle Struktur des Layouts ist in der zugehörigen Razor-Datei definiert.

Listing 8: Grundstruktur des Hauptlayouts

```
@inherits LayoutComponentBase

<MudThemeProvider Theme="@AppState.Theme"
                  IsDarkMode="@AppState.IsDarkMode"
                  IsDarkModeChanged="OnDarkModeChanged" />

<MudDialogProvider />
<MudSnackbarProvider />

<AuthorizeView>
    <Authorized>
        <div class="page">
            <AppBar />
            <article class="content">
                @Body
            </article>
        </div>
    </Authorized>

    <NotAuthorized>
        <RedirectToLogin />
    </NotAuthorized>
</AuthorizeView>
```

Innerhalb des Layouts werden mehrere globale UI-Komponenten registriert. Der `MudThemeProvider` steuert das Erscheinungsbild der Anwendung sowie die Umschaltung zwischen hellem und dunklem Darstellungsmodus. Zusätzlich werden über `MudDialogProvider` und `MudSnackbarProvider` globale Dialog- und Benachrichtigungssysteme bereitgestellt.

Die Darstellung der eigentlichen Inhalte erfolgt innerhalb eines `AuthorizeView`. Dadurch wird sichergestellt, dass geschützte Seiten nur für authentifizierte Benutzer sichtbar sind. Ist ein Benutzer angemeldet, wird das Hauptlayout mit Navigationsleiste und Inhaltsbereich gerendert. Andernfalls erfolgt eine automatische Weiterleitung zur Anmeldeseite.

7.5.2 Entra ID Authentifizierung

Für unsere Diplomarbeit wurde uns Microsoft Entra ID als Authentifizierungsverfahren verwendet, um die Anwendung ind das Unternehmensumfeld von Count IT einzubinden.

Innerhalb der Anwendung wird der Authentifizierungszustand ausgelesen und mit dem internen Benutzermodell verknüpft. Der folgende Code zeigt, wie anhand der Claims ein Benutzer ermittelt beziehungsweise bei Bedarf neu angelegt wird.

Listing 9: Verarbeitung des authentifizierten Benutzers

```
var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();
var userClaims = authState.User;

var displayName = userClaims.FindFirst(c => c.Type == "name")?.Value
    ?? userClaims.Identity?.Name
    ?? "Unknown";
var email = userClaims.Identity?.Name ?? "Unknown@countit.at";

var user = await RunQuery(new GetUserByEmail(email));
```

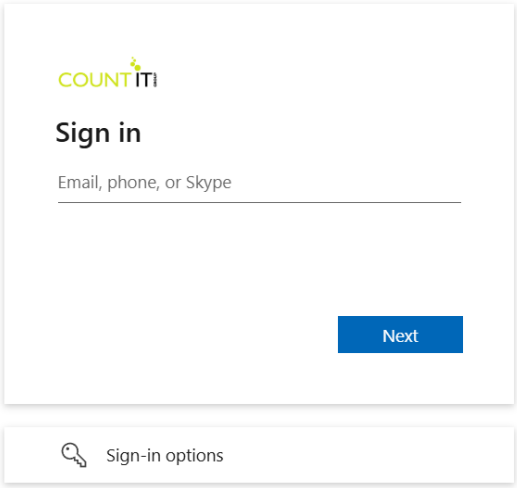


Abbildung 31: Login Screen

```
10  user ??= await ExecuteCommand<CreateUser, User>(
11      new CreateUser(new UserBase { DisplayName = displayName, Email = email })
12  );
13  AppState.SetUser(user);
```

Nach erfolgreicher Anmeldung wird zunächst der aktuelle Authentifizierungszustand ausgelesen. Abbildung 31 Anschließend werden Anzeigename und E-Mail-Adresse aus den vorhandenen Claims bestimmt. Auf Basis der E-Mail-Adresse wird geprüft, ob der Benutzer bereits in der Anwendungsdatenbank existiert. Falls dies nicht der Fall ist, wird automatisch ein neuer Benutzer erzeugt und im globalen App-State gespeichert. Damit wird die externe Authentifizierung direkt mit dem internen Domänenmodell der Anwendung verknüpft.

Zusätzlich wurde eine Weiterleitung für nicht authentifizierte Benutzer vorgesehen.

Listing 10: Weiterleitung auf die Login-Seite

```
1  protected override void OnInitialized()
2  {
3      var uri = NavigationManager.Uri.ToLowerInvariant();
4      if (!uri.Contains("/login") && !uri.Contains("/signin-oidc"))
5          NavigationManager.NavigateTo("/login", forceLoad: true);
6  }
```

Die gezeigte Logik prüft beim Initialisieren der Komponente, ob sich der Benutzer bereits auf einer Anmelde- oder Callback-Seite befindet. Ist dies nicht der Fall, wird eine Weiterleitung auf die Login-Seite ausgelöst. Dadurch wird verhindert, dass nicht authentifizierte Benutzer geschützte Bereiche der Anwendung direkt aufrufen können.

7.5.3 Dateneingabe-Formulare

Die für den Matching-Algrorythmus benötigten Benutzerdaten werden in einem dreistufigen Onboarding-Prozess erfasst. Die Dateneingabe wurde bewusst auf drei unabhängige Formulare aufgeteilt um die Benutzerführung zu vereinfachen und um die Eingaben trennen.

Die Grundstruktur der Formulare ist in allen Schritten sehr ähnlich. Die Eingaben werden über ein `MudForm` validiert, mit dem aktuell angemeldeten Benutzer verknüpft und anschließend über Commands an das Backend übertragen.

Der Folgende Code Block zeigt die Beispielhafte Validierung und Verknüpfung mit dem User.

Listing 11: Grundmuster eines Eingabeformulars

```
1  await _form.Validate();
2
3  if (!_form.IsValid || AppState.User == null)
4      return;
5
6  _entity.UserId = AppState.User.Id;
7  await ExecuteCommand<CreateEntity, Entity>(
8      new CreateEntity(_entity), successMessage);
```

Formulare [12]

- Im ersten Schritt werden die Wohnsitzdaten des Benutzers erfasst. Dazu gehören insbesondere Bezeichnung, Adresse und Ortsdaten. Abbildung 37
- Im zweiten Schritt werden Fahrzeugdaten wie Beschreibung, Typ, Sitzplätze und gegebenenfalls CO_2-Werte hinterlegt. Abbildung 38
- Der dritte Schritt dient der Wochenplanung, in der verfügbare Wohnsitze, Fahrzeuge und Fahrzeiten kombiniert werden. Erst durch diese zeitliche Zuordnung entsteht die Grundlage für das Matching von Fahrgemeinschaften, welches nach dem Speichern angestoßen wird. Abbildung 39

7.5.4 Integration von MapLibre und GeoJSON

Um die ermittelten potentiellen Fahrgemeinschaften welche als GeoJSON gespeichert werden ordentlich im Frontend für die Nutzer darzustellen wurde MapLibre verwendet. Da es sich dabei um eine JavaScript-basierte Kartenbibliothek handelt, erfolgt die Einbindung im Blazor-Frontend über JavaScript-Interop.

[12] Die Strukturierung dieses Abschnitts, insbesondere die Aufbereitung in übersichtliche Stichpunkte, wurde unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

Die `MapComponent`-Komponente zeigt, wie die Karte aus dem Blazor-Frontend heraus initialisiert wird.

Listing 12: Initialisierung der Kartenkomponente

```
[Parameter] public string? GeoJson { get; set; }

protected override async Task OnAfterRenderAsync(bool firstRender)
{
    if (firstRender && !string.IsNullOrEmpty(GeoJson))
        await JSRuntime.InvokeVoidAsync("initializeMap", GeoJson, AppState.IsDarkMode);
}

private async Task CenterMap() => await JSRuntime.InvokeVoidAsync("centerMap");
```

Die Komponente erhält die GeoJSON-Daten der Fahrgemeinschaft als Parameter und übergibt diese nach dem ersten Rendern an die JavaScript-Funktion `initializeMap`. Dadurch bleibt die Kartenlogik vom restlichen Seitenaufbau getrennt. Zusätzlich steht mit `CenterMap` eine eigene Funktion zur Verfügung, um den sichtbaren Kartenausschnitt an die geladenen Geodaten anzupassen.

Die eigentliche Initialisierung der Karte erfolgt in JavaScript.

Listing 13: MapLibre mit GeoJSON-Datenquelle

```
const map = new maplibregl.Map({
    container: 'map',
    style: styleUrl,
    center: [0, 0],
    zoom: 1,
});

map.addSource('route', { type: 'geojson', data: geoJsonData });
map.addLayer({ id: 'route-line', type: 'line', source: 'route' });
map.addLayer({ id: 'route-points', type: 'circle', source: 'route' });
```

Zunächst wird eine MapLibre-Karte erstellt und anschließend mit einer GeoJSON-Datenquelle verknüpft. Auf Basis dieser Quelle werden separate Ebenen für Linien und Punkte angelegt.

7.5.5 Reward- und CO_2-Anzeige

Auf der Startseite werden Kennzahlen angezeigt, die sowohl ökologische Auswirkungen als auch nutzerbezogene Belohnungen visualisieren. Dazu zählen insbesondere die gesamte CO_2-Einsparung, die Anzahl potenzieller Fahrgemeinschaften und der aktuelle Punktestand des Benutzers.

Die visuelle Darstellung dieser Werte erfolgt über einzelne Kennzahlenkarten im Dashboard. Abbildung 2

Listing 14: KPI-Anzeige auf dem Dashboard

```
<MudGrid Class="mb-6">
    <MudItem xs="12" md="4">...</MudItem>
```

```
    <MudItem xs="12" md="4">...</MudItem>
    <MudItem xs="12" md="4">...</MudItem>
</MudGrid>
```

Die Kennzahlen werden jeweils in eigenen Elementen dargestellt. Durch die Unterteilung in drei Karten sind die wichtigsten Informationen unmittelbar auf der Startseite sichtbar. Siehe Abbildung 32

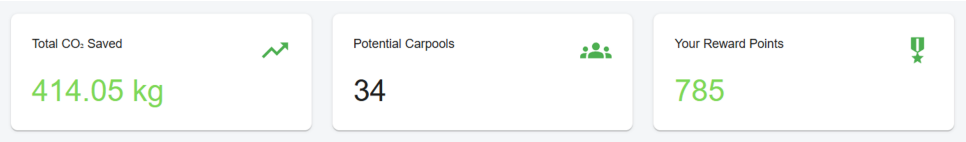


Abbildung 32: Kennzahlen Startseite

Insbesondere die Kombination aus CO_2-Wert und Reward-Punkten unterstützt die Zielsetzung der Anwendung, nachhaltiges Verhalten nicht nur funktional, sondern auch motivierend darzustellen.

Die Werte selbst werden beim Laden der Seite aus Queries und dem globalen App-State übernommen.

Listing 15: Laden der Reward- und CO_2-Werte

```
if (AppState.User != null)
{
    var carpools = await RunQuery(new GetCarpools(AppState.User.Id, null, null)) ?? [];
    _potentialCarpoolsCount = carpools.Count();
}

_totalCo2Saved = AppState.CompanyInformation?.TotalCo2Savings ?? 0;
```

Beim Initialisieren der Startseite werden zunächst die zum Benutzer gehörenden Fahrgemeinschaften geladen und gezählt. Zusätzlich werden die Unternehmensinformationen aus dem App-State verwendet, um die gesamte CO_2-Einsparung anzuzeigen. Die Reward-Punkte werden direkt über den aktuell geladenen Benutzer eingebunden. Dadurch wird die Startseite automatisch mit den für den Benutzer relevanten Kennzahlen befüllt.

7.5.6 Darkmode

Ein weiterer Aspekt des Frontends ist die Unterstützung eines Darkmode. Dadurch kann die Anwendung nicht nur optisch ansprechender gestaltet, sondern auch an unterschiedliche Nutzungssituationen angepasst werden. Die Umschaltung zwischen hellem und dunklem Darstellungsmodus erfolgt zentral über den globalen Anwendungszustand.

(a) Toggle im Lightmode

(b) Toggle im Darkmode

Abbildung 33: Vergleich des Darkmode-Toggles in beiden Darstellungsmodi

Im Layout wird dafür der `MudThemeProvider` verwendet, der das aktuelle Theme sowie den Status des Darkmode verwaltet.

Listing 16: Steuerung des Darkmode im Layout

```
1  private MudThemeProvider? _mudThemeProvider;
2
3  private void OnDarkModeChanged(bool args) => AppState.ToggleDarkMode();
```

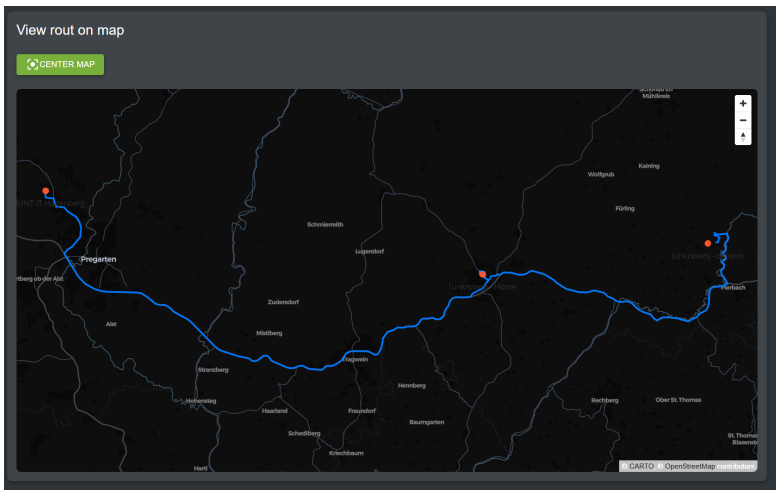
Die Umschaltung des Darstellungsmodus wird damit direkt an den globalen App-State gekoppelt. Änderungen wirken sich dadurch konsistent auf die gesamte Benutzeroberfläche aus.

Auch die Kartenansicht berücksichtigt den aktuellen Darstellungsmodus. Dafür wird in der JavaScript-Logik abhängig vom Parameter `darkmode` ein passender Kartenstil geladen.

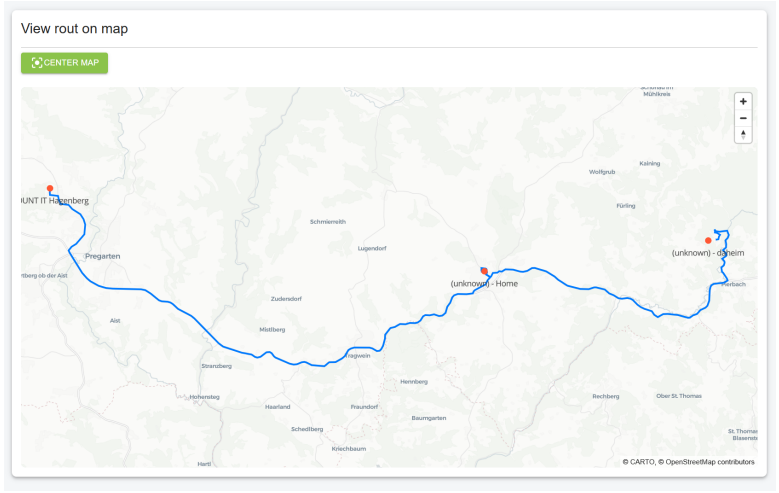
Listing 17: Darkmode-abhängiger Kartenstil

```
1  const styleUrl = darkmode
2      ? 'https://tiles.basemaps.cartocdn.com/gl/dark-matter-gl-style/style.json'
3      : 'https://tiles.basemaps.cartocdn.com/gl/positron-gl-style/style.json';
```

Abhängig vom aktuellen Darstellungsmodus wird damit entweder ein heller oder ein dunkler Kartenstil verwendet. Dadurch passt sich die Kartenansicht optisch an das restliche Erscheinungsbild der Anwendung an, ohne dass die Kartenkomponente selbst grundlegend verändert werden muss. Abbildung 34



(a) Kartenansicht im Darkmode

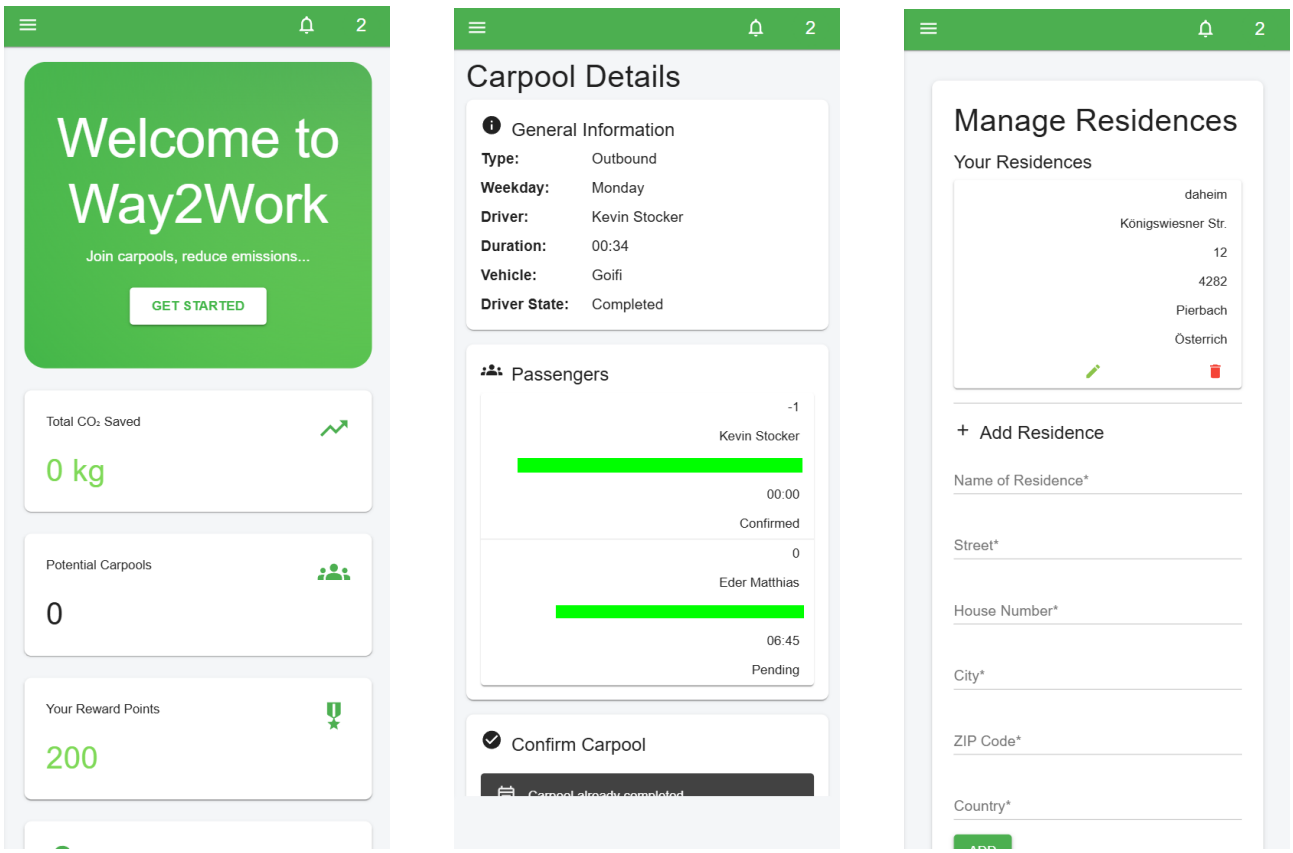


(b) Kartenansicht im Lightmode

Abbildung 34: Vergleich der Kartenansicht im Darkmode und Lightmode

7.5.7 Mobile Resvonsive Layout

Da die Anwendung nicht ausschließlich auf Desktop-Geräten verwendet wird, wurde die Oberfläche responsiv aufgebaut. Dabei kommen sowohl die Breakpoints des MudBlazor-Grid-Systems als auch eigene CSS-Regeln für kleinere Bildschirmgrößen zum Einsatz. Dadurch können Inhalte auf Smartphones untereinander dargestellt und auf größeren Bildschirmen kompakter angeordnet werden. Abbildung 35



(a) Startseite (b) Fahrgemeinschaften (c) Formularansicht

Abbildung 35: Responsive Darstellung ausgewählter Bereiche der Anwendung auf mobilen Endgeräten

Ein Beispiel dafür zeigt die Grid-Struktur des Dashboards.

Listing 18: Responsive Grid-Struktur im Dashboard

```
<MudGrid Class="mb-6">
    <MudItem xs="12" md="4">...</MudItem>
    <MudItem xs="12" md="4">...</MudItem>
    <MudItem xs="12" md="4">...</MudItem>
</MudGrid>
```

Die Angabe `xs="12"` bewirkt, dass ein Element auf kleinen Bildschirmen die gesamte verfügbare Breite einnimmt. Mit `md="4"` wird festgelegt, dass dieselben Elemente ab mittleren Bildschirmgrößen nebeneinander in drei Spalten dargestellt werden. Dadurch bleiben Inhalte auf Smartphones gut lesbar, während auf größeren Geräten eine kompaktere Darstellung möglich ist.

Dasselbe Prinzip wird auch in Formularen verwendet.

Listing 19: Responsive Formularfelder

```
<MudGrid>
    <MudItem xs="12" md="6">
        <MudTextField @bind-Value="_newVehicle.Description"
                      Label="@Translation.Vehicles_Description" Required="true" />
    </MudItem>
```

```
6       <MudItem xs="12" md="6">
7           <MudSelect T="VehicleType" @bind-Value="_newVehicle.Type"
8                      Label="@Translation.Vehicles_Type" Required="true">
9               ...
10          </MudSelect>
11      </MudItem>
12  </MudGrid>
```

Auch hier werden die Eingabefelder auf kleinen Geräten untereinander dargestellt und auf größeren Bildschirmen nebeneinander angeordnet. Dieses Vorgehen verbessert die Bedienbarkeit erheblich, da Formulare sowohl auf Desktop-Geräten als auch auf Smartphones strukturiert und gut nutzbar bleiben.

Für die Kartenansicht wurden zusätzlich eigene responsive CSS-Regeln definiert.

Listing 20: Responsive Kartenansicht

```
1   .map-responsive {
2       width: 100%;
3       max-width: 100vw;
4       height: calc(90vh - (200px + 30px));
5   }
6
7   @media (max-width: 900px) {
8       .map-responsive {
9           width: 100% !important;
10          border-radius: 0 !important;
11      }
12  }
```

Die Klasse `map-responsive` legt die Grundgröße der Kartenansicht fest. Über die Media Query wird das Verhalten auf kleineren Displays angepasst, sodass die Karte auch auf mobilen Geräten die verfügbare Breite optimal ausnutzt. Das responsive Layout ist damit ein wesentlicher Bestandteil der Frontend-Implementierung von Way2Work.

7.5.8 Language File

Für die sprachabhängige Darstellung der Benutzeroberfläche wurde ein zentrales Language File auf Basis von `.resx`-Dateien verwendet. Die Sprachdateien wurden im `Resources`-Bereich des Projekts verwaltet und mit dem Visual-Studio-Werkzeug *ResX Manager* bearbeitet. Neben der Standardsprache Englisch wurden auch deutsche und österreichische Sprachvarianten hinterlegt. Die österreichische Variante stellt dabei ein kleines zusätzliches Detail innerhalb der Anwendung dar.

Die Einbindung der Texte erfolgt über die Klasse `Translation`. Dadurch können Übersetzungen sowohl im Razor-Markup als auch im Code-Behind direkt verwendet werden.

Listing 21: Einbindung von Übersetzungen im Razor-Markup

```
1   <MudText Typo="Typo.h2">@Translation.Home_Welcome_Title</MudText>
2   <MudButton Href="/residence">@Translation.Home_Welcome_Button</MudButton>
```

Im Code-Behind werden dieselben Sprachressourcen beispielsweise für Fehlermeldungen oder Erfolgsmeldungen verwendet.

Listing 22: Verwendung der Übersetzungen im Code-Behind

```
1 if (AppState.User == null)
2 {
3     Snackbar.Add(Translation.Validation_NoUser, Severity.Error);
4     return;
5 }
```

7.6 Azure Deployment

Das Deployment der Anwendung erfolgt in die Microsoft Azure Cloud und orientiert sich an modernen Cloud-Native-Prinzipien. Das bedeutet, dass die Architektur speziell für die Cloud entworfen wurde: Sie nutzt isolierte Container, dynamisch skalierbare Ressourcen und vollständig verwaltete Plattformdienste (PaaS), anstatt auf klassischen, starren Serverstrukturen zu laufen. Als primäre Hosting-Umgebung dienen *Azure Container Apps* (ACA) für die Ausführung der Microservices und *Azure SQL* für die persistente Datenhaltung.

7.6.1 Vom AppHost in die Cloud

Wie bereits in den theoretischen Grundlagen (siehe Kapitel 6.4.1) erklärt, definiert das Projekt `CIT.Way2Work.AppHost` die gesamte Infrastruktur und das Zusammenspiel der Dienste. Während dieser AppHost bei der lokalen Entwicklung primär dazu dient, Container auf dem Entwicklerrechner zu starten, spielt er beim Deployment eine weitaus mächtigere Rolle: Er fungiert als automatische Vorlage für die Cloud-Infrastruktur (Infrastructure as Code).

Automatisierte Containerisierung Beim Deployment-Prozess analysieren die Azure-Tools den AppHost. Für die eigenen .NET-Projekte (wie die API, den Blazor-Client und den MigrationService) müssen keine klassischen `Dockerfile`-Skripte geschrieben oder gewartet werden. Das .NET SDK verpackt diese Projekte zur Laufzeit automatisch in lauffähige Container-Images. Gemeinsam mit den fertigen Images der Drittanbieter-Dienste (OSRM, Vroom, Nominatim) werden diese in die Cloud übertragen und dort als nahtlos vernetzte Azure Container Apps gestartet.

Intelligente Ressourcen-Abstraktion Ein enormer Vorteil dieses Deployment-Ansatzes ist die intelligente Abstraktion der Infrastruktur beim Wechsel von der lokalen Umgebung in die Cloud. Startet man die Anwendung lokal, provisioniert Aspire beispielsweise einen simplen

SQL-Container für schnelle Tests. Beim Deployment erkennt das System jedoch die produktive Zielumgebung und erstellt stattdessen eine echte, hochverfügbare und verwaltete *Azure SQL*-Datenbank.

Die dafür benötigten Verbindungsinformationen (Connection Strings) und Passwörter müssen von den Entwicklern zu keinem Zeitpunkt manuell kopiert oder konfiguriert werden. Das Deployment-Tooling injiziert diese sensiblen Daten vollautomatisch und sicher als *Secrets* direkt in die Umgebungsvariablen der jeweiligen Container Apps.

7.6.2 CI/CD

Um eine gleichbleibend hohe Softwarequalität und schnelle Release-Zyklen zu garantieren, erfolgt die Auslieferung der Software weitgehend automatisiert über Pipelines in Azure DevOps.

Continuous Integration (CI) Die Datei `gated-checkin.yaml` definiert unsere CI-Pipeline. Dieser Prozess wird automatisch bei jedem Pull Request auf den Hauptzweig (`main`-Branch) angestoßen. Die Pipeline lädt den aktuellen Quellcode herunter, stellt alle NuGet-Pakete wieder her (`dotnet restore`), kompiliert die gesamte Solution (`dotnet build`) und führt vorhandene Tests aus. Dieser *Gated Check-in* stellt sicher, dass fehlerhafter Code gar nicht erst in den Hauptentwicklungszweig integriert wird.

Automatisches Deployment (CD) Der eigentliche Deployment-Prozess in die Cloud nutzt die *Azure Developer CLI* (`azd`) in Kombination mit .NET Aspire. Dieser Prozess läuft in mehreren automatisierten Schritten ab: Zunächst generiert Aspire aus dem C#-Code des `AppHost` ein vollständiges JSON-Manifest aller benötigten Ressourcen. Das CLI-Tool übersetzt dieses Manifest anschließend in sogenannte Bicep-Templates [was sind das für Templates??] (Infrastructure as Code). Daraufhin werden die benötigten Azure-Ressourcen (Container Apps Environment, SQL Server, Log Analytics Workspace) in der Cloud erstellt oder aktualisiert. Im letzten Schritt werden die Container-Images gebaut, in eine sichere *Azure Container Registry* (ACR) hochgeladen und als neue Revisionen gestartet.

Durch diesen stark automatisierten Ansatz entfällt die fehleranfällige, manuelle Pflege komplexer Deployment-Skripte vollständig, da sich die gesamte Cloud-Infrastruktur direkt aus dem C#-Code ableitet. [13]

[13] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

8 Ergebnis

In diesem Kapitel werden die im Rahmen der Diplomarbeit erzielten Ergebnisse vorgestellt. Dabei werden sowohl die umgesetzten Funktionen des Frontends als auch die technischen Ergebnisse des Backends vorgestellt und anhand ausgewählter Ansichten und Komponenten gezeigt.

8.1 Ergebnis Frontend

Im Folgenden werden die zentralen Ergebnisse der Frontend-Umsetzung anhand der wichtigsten Benutzeroberflächen und Funktionen der Webanwendung dargestellt.

Authentifizierung Die Authentifizierung der Benutzer erfolgt über Microsoft Entra ID. Dabei werden Benutzer auf die Microsoft-Anmeldeseite weitergeleitet und können sich mit ihrem bestehenden Unternehmenskonto anmelden. Nach erfolgreicher Authentifizierung wird der Zugriff auf die geschützten Bereiche der Anwendung freigegeben.

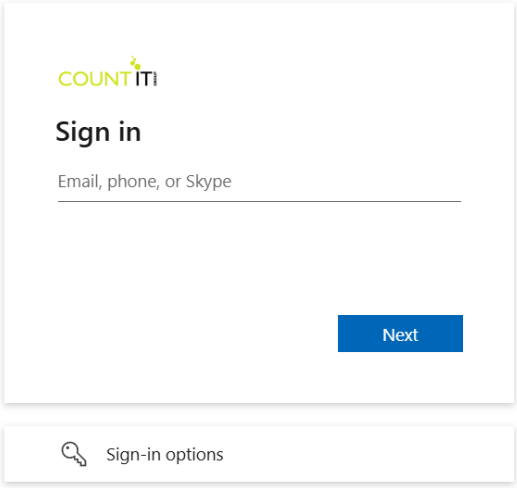


Abbildung 36: Anmeldemaske von Microsoft Entra ID zur Authentifizierung der Benutzer mit dem Unternehmenskonto.

Onboarding und Profilverwaltung Das Onboarding umfasst die Erfassung von Wohnort(en), optionalen Fahrzeugdaten sowie eines wöchentlichen Schedules inklusive Zeitfenstern und Toleranzen. Die Profilverwaltung ermöglicht jederzeitige Änderungen, die anschließend in die Berechnungslogik einfließen. Die drei Onboarding-Schritte umfassen die Erfassung der Wohnorte (Abbildung 37), optional der Fahrzeuge (Abbildung 38) sowie den wöchentlichen Fahrplan mit Zeitfenstern (Abbildung 39).

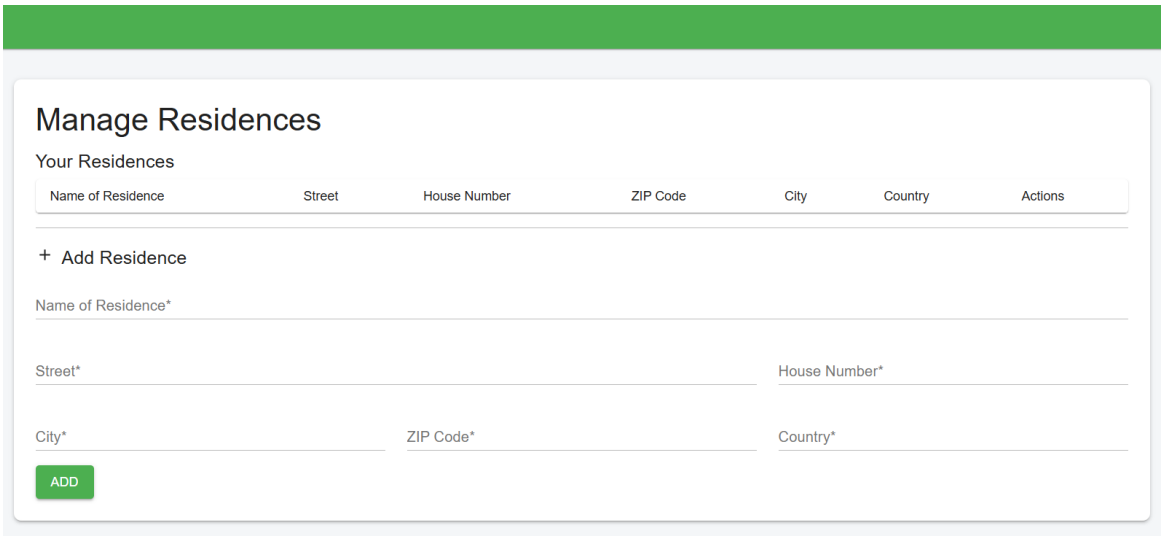


Abbildung 37: Onboarding-Schritt 1: Verwaltung der Wohnorte (Residences) als Basis für die spätere Fahrgemeinschaftsberechnung.

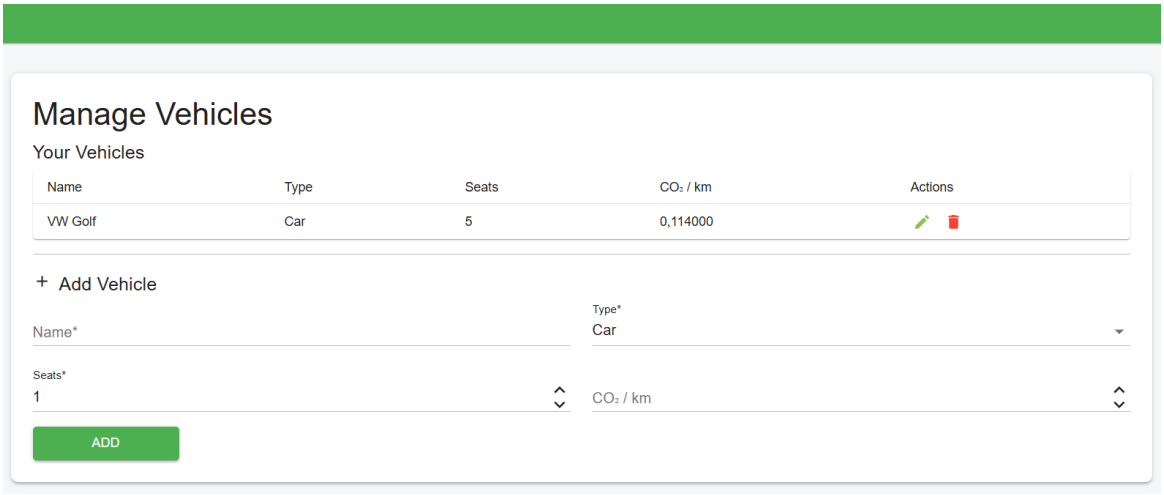


Abbildung 38: Onboarding-Schritt 2: Verwaltung der Fahrzeuge inklusive Emissionswerten; optional für Nutzer ohne eigenes Fahrzeug.

Matching und Kartenansicht Potenzielle Fahrgemeinschaften werden automatisiert berechnet und im Frontend dargestellt. Eine Kartenansicht mit MapLibre visualisiert den Streckenverlauf sowie Ein- und Ausstiegspunkte. Die berechneten Vorschläge werden in einer Übersicht je Tag dargestellt (Abbildung 40) und können in einer Detailansicht inklusive Kartenvisualisierung nachvollzogen werden (Abbildung 41).

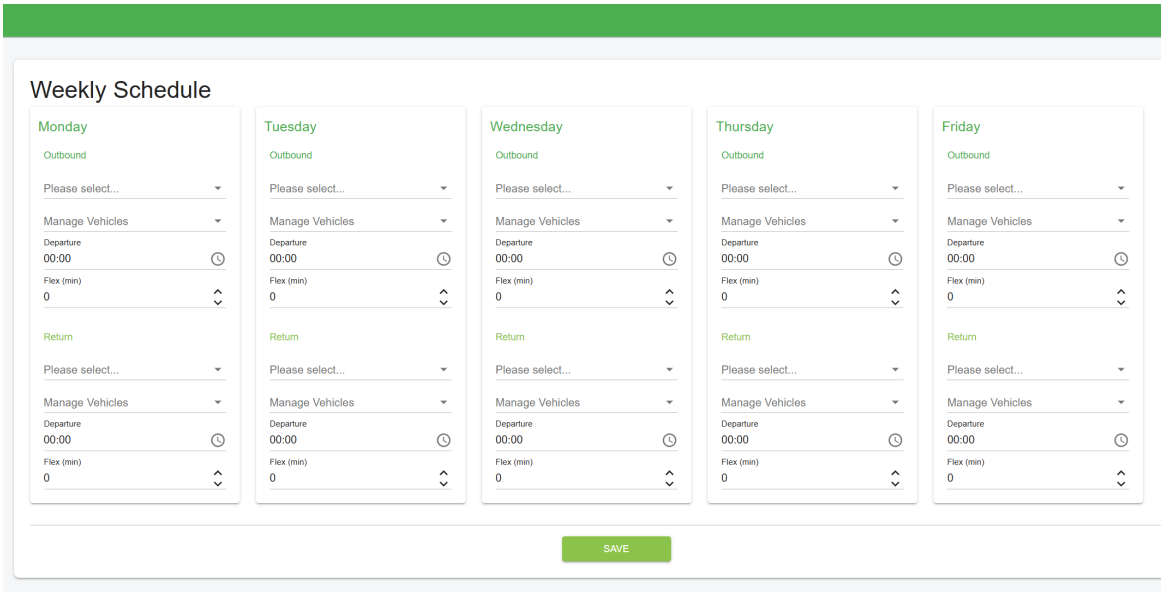


Abbildung 39: Onboarding-Schritt 3: Wochenschedule mit Zeitfenstern und Toleranzen für Hin- und Rückfahrten.

Anfrage- und Fahrgemeinschafts-Workflow Benutzer können vorgeschlagenen Fahrgemeinschaften beitreten oder diese ablehnen. Die Detailansicht liefert die für die Entscheidung notwendigen Informationen (Route, Beteiligte, Zeitfenster). Die Auswahl und Verwaltung einer Fahrgemeinschaft erfolgt über die Matching-Übersicht (Abbildung 40) sowie die Detailansicht (Abbildung 41).

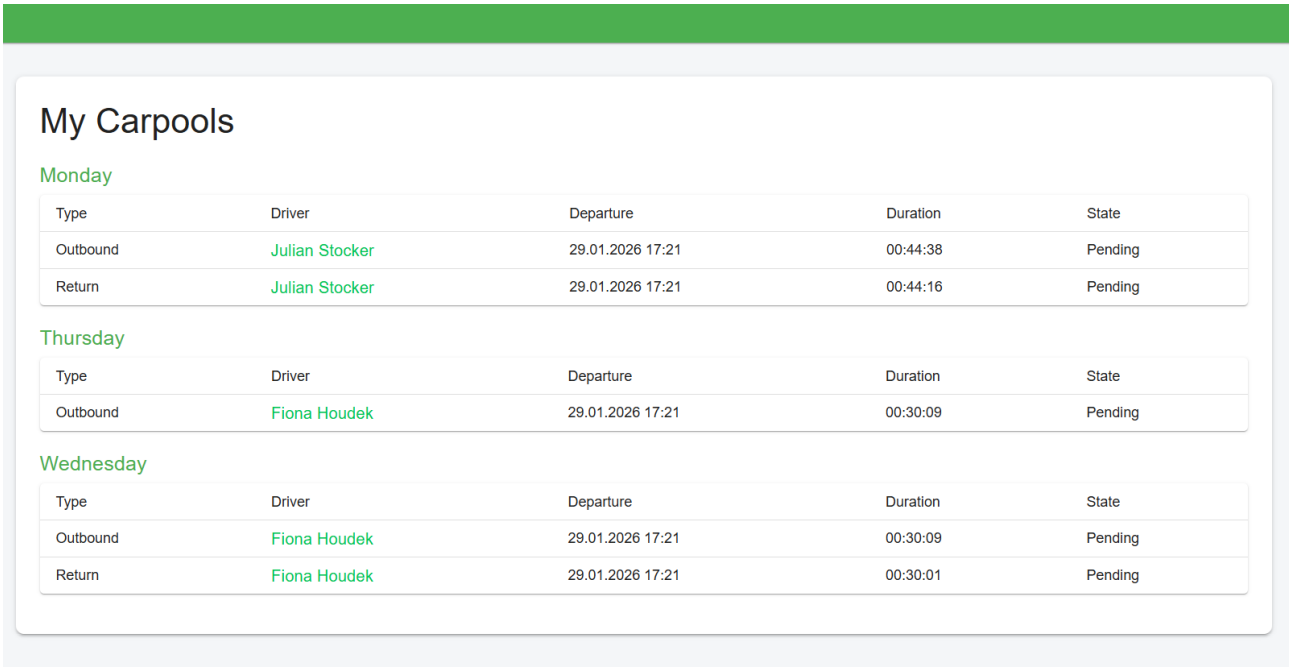


Abbildung 40: Übersicht der berechneten potenziellen Fahrgemeinschaften (Matching-Ergebnisse) je Wochentag.

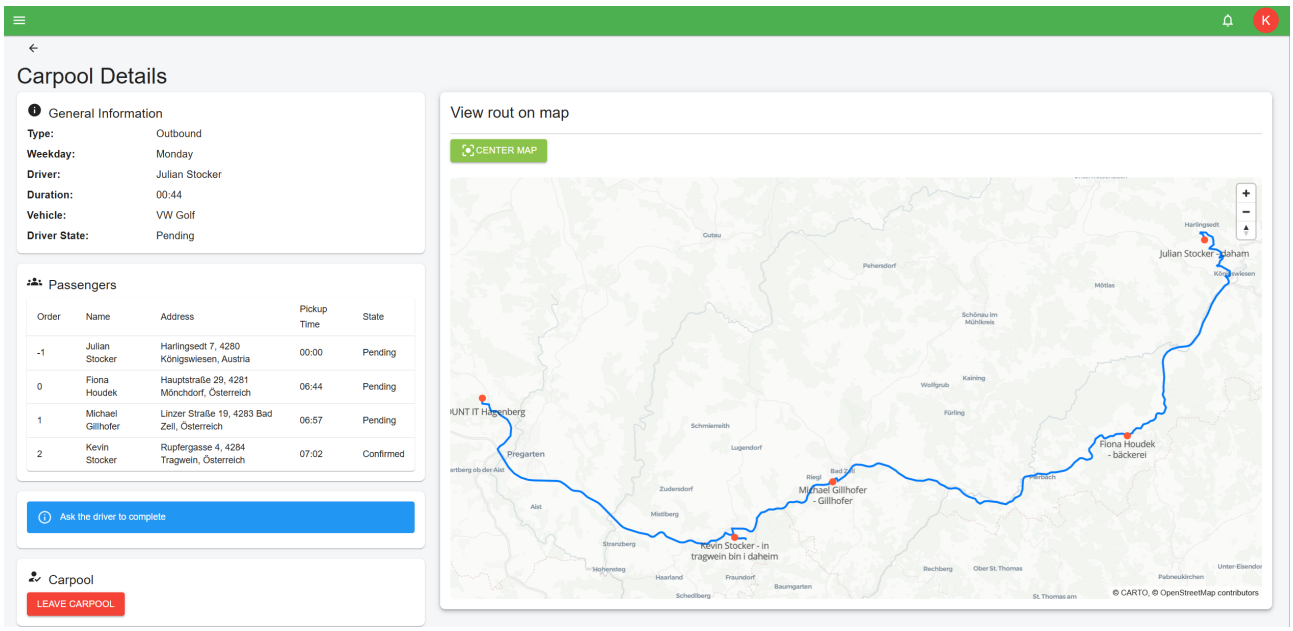


Abbildung 41: Detailansicht einer Fahrgemeinschaft mit Beteiligten und Kartenvisualisierung des Streckenverlaufs (MapLibre).

Fahrtbestätigung und Nachweislogik Die Durchführung einer gemeinsamen Fahrt wird durch den Fahrer bestätigt. Auf dieser Grundlage werden die gesammelten Werte (z. B. eingespartes CO_2) und die Reward-Punkte aktualisiert. Die Fahrtbestätigung ist rollenbasiert umgesetzt: Mitfahrer können keine Bestätigung durchführen (Abbildung 42), während der Fahrer die Fahrt abschließen kann (Abbildung 43); der bestätigte Status wird anschließend im System angezeigt (Abbildung 44).

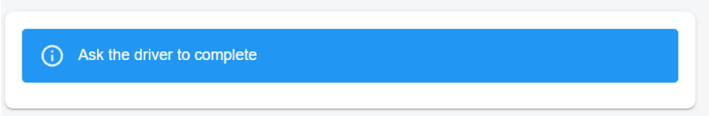


Abbildung 42: Mitfahreransicht: Fahrtbestätigung ist nicht möglich, da diese Rolle nur dem Fahrer vorbehalten ist.

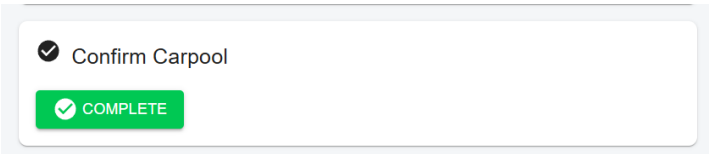


Abbildung 43: Fahreransicht: Option zur Fahrtbestätigung nach Durchführung der gemeinsamen Fahrt.

Reward-System und Einlösung Für bestätigte gemeinsame Fahrten werden Reward-Punkte gutgeschrieben. Die Einlösung erfolgt aktuell organisatorisch über eine verantwortliche Stelle im Unternehmen: Die Punkte werden geprüft, ein Gutschein/Goodie ausgegeben und anschließend im System entsprechend abgezogen.

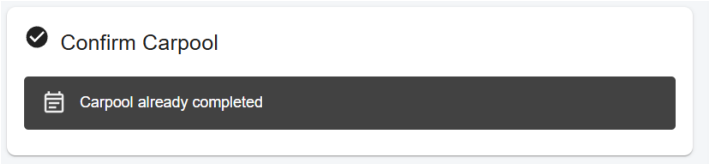


Abbildung 44: Status nach Bestätigung: Die Fahrt ist als abgeschlossen markiert; CO_2- und Reward-Werte werden aktualisiert.

Backoffice / Admin-Funktionen Für administrative Zwecke existiert ein gesonderter Zugriff (Endpunkt/Ansicht), der es ermöglicht, vergangene sowie potenzielle Fahrgemeinschaften einzusehen. Dies dient sowohl der Auswertung der Nutzung (Akzeptanz in der Belegschaft) als auch der technischen Kontrolle im Betrieb. Für Auswertungs- und Kontrollzwecke steht eine administrative Gesamtübersicht zur Verfügung (Abbildung 45).

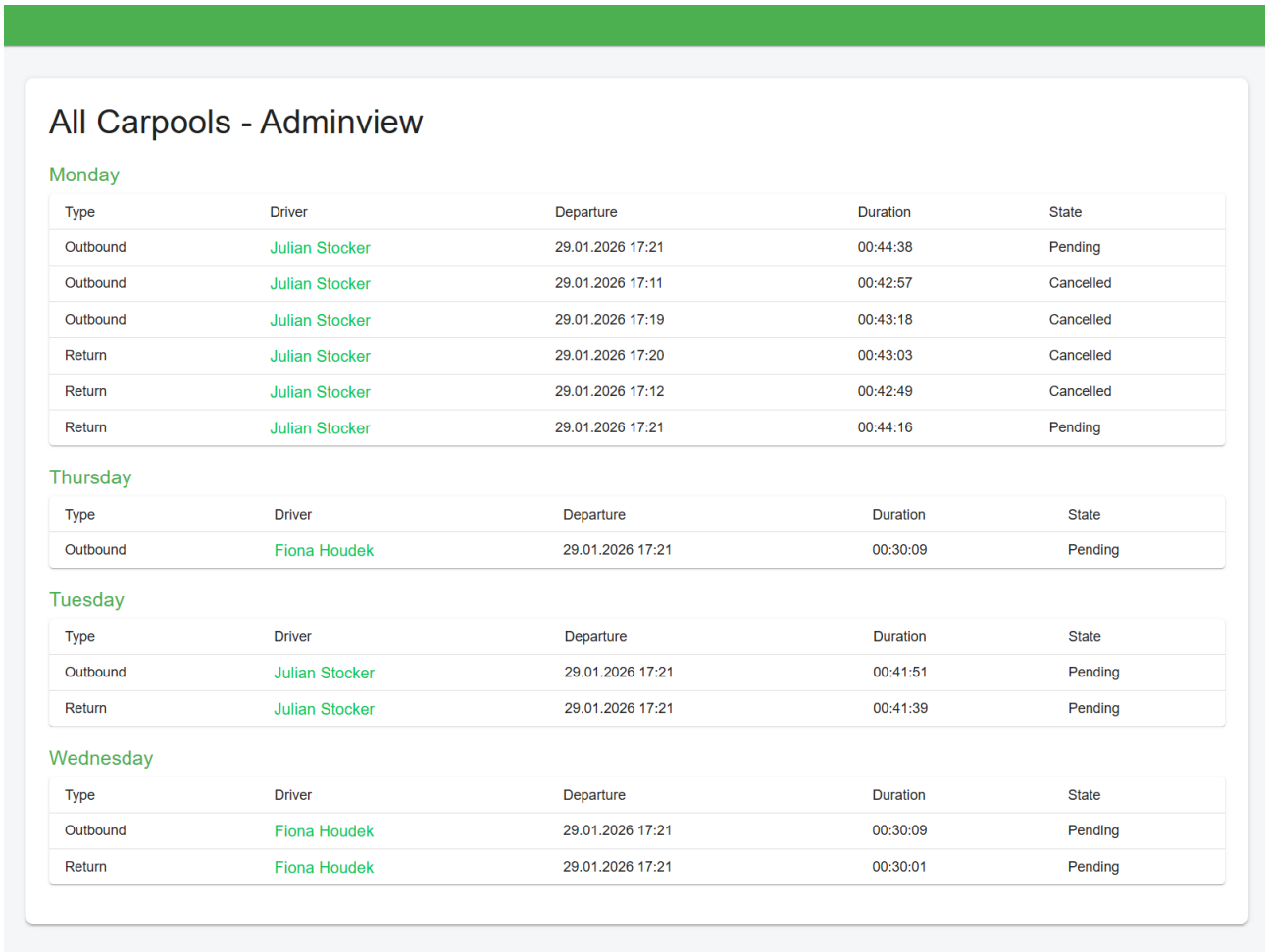


Abbildung 45: Administrative Auswertung: Gesamtübersicht aller Fahrgemeinschaften zur Kontrolle und zur Nutzungsanalyse.

8.2 Ergebnis Backend

Im Folgenden werden die zentralen Ergebnisse der Backend-Umsetzung anhand der bereitgestellten Schnittstellen, der laufenden Systemumgebung und der integrierten Geodienste dargestellt.

Orchestrierung und Systemübersicht Die gesamte Backend-Umgebung lässt sich über *.NET Aspire* mit einem einzigen Befehl starten. Dabei werden automatisch alle benötigten Dienste hochgefahren und miteinander vernetzt: die API, die Azure-SQL-Datenbank, der Datenbank-Migrationsdienst sowie die drei Geodienst-Container (OSRM, VROOM und Nominatim). Der Zustand aller Dienste ist über das Aspire-Dashboard (Abbildung 46) in Echtzeit einsehbar, das Logs, Traces und Metriken anwendungsübergreifend bündelt.

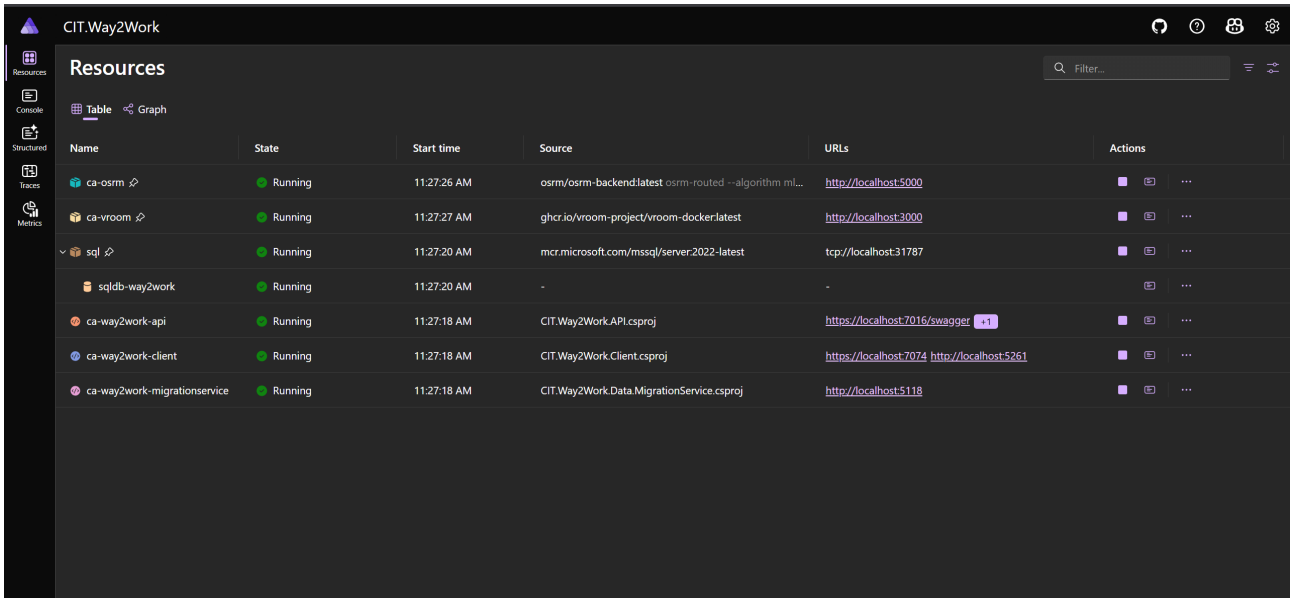


Abbildung 46: Das .NET Aspire Dashboard bündelt Logs, Metriken und den Status aller Container (API, Datenbank, OSRM, Vroom, Nominatim) in einer zentralen Echtzeit-Ansicht.

API-Endpunkte Die API stellt insgesamt 29 Endpunkte bereit, die sich aus 14 schreibenden Commands und 15 lesenden Queries zusammensetzen. Die Endpunkte werden automatisch aus den CQRS-Definitionen generiert – klassische Controller-Klassen entfallen vollständig. Dabei bestimmt der Typ des Commands die HTTP-Methode: erzeugende Operationen verwenden `POST`, idempotente Aktualisierungen `PUT` und Löschvorgänge `DELETE`; sämtliche Queries sind als `GET`-Anfragen realisiert.

Die Endpunkte decken folgende Bereiche ab:

- **Benutzerverwaltung:** Anlegen, Aktualisieren, Löschen und Abfragen von Benutzern – sowohl per ID als auch per E-Mail-Adresse.
- **Fahrzeugverwaltung:** CRUD-Operationen für Fahrzeuge inklusive Abfrage aller Fahrzeuge eines bestimmten Benutzers.
- **Wohnortverwaltung:** CRUD-Operationen für Wohnorte. Beim Anlegen oder Ändern eines Wohnorts werden automatisch die Koordinaten über Nominatim aufgelöst sowie Fahrtzeit und Distanz zum Firmenstandort über OSRM berechnet und gespeichert.
- **Fahrplanverwaltung:** CRUD-Operationen für wöchentliche Fahrplaneinträge (Wochentag, Fahrtrichtung, Abfahrtszeit und Flexibilität). Jede Änderung an Benutzerdaten, Fahrzeugen, Wohnorten oder Fahrplänen löst automatisch eine Neuberechnung aller Fahrgemeinschaftsvorschläge aus.
- **Fahrgemeinschaften:** Abfrage von Fahrgemeinschaften – filterbar nach Benutzer, Wochentag und Fahrtrichtung – sowie Detailabfrage inklusive Teilnehmerliste. Die Antwort enthält neben den Teilnehmerdaten und Abholzeiten auch ein vollständiges GeoJSON-Objekt mit Streckenverlauf und beschrifteten Haltepunkten, das vom Frontend direkt auf der Karte dargestellt wird.
- **Statusänderungen:** Endpunkte zum Bestätigen oder Ablehnen einer Fahrgemeinschaft durch Fahrer und Mitfahrer sowie zur Fahrtbestätigung durch den Fahrer nach erfolgter gemeinsamer Fahrt.
- **Unternehmensdaten:** Abfrage der Firmeninformationen inklusive Standortkoordinaten und kumulierten CO_2-Einsparungen.

Tabelle 1 zeigt eine Übersicht der bereitgestellten Endpunkte, gruppiert nach Domänenbereich.

Die vollständige, interaktive Dokumentation aller Endpunkte inklusive Parametertypen und Antwortstrukturen wird über Swagger UI (Abbildung 47) automatisch aus dem Code generiert. Diese Oberfläche diente während der Entwicklung als technischer Vertrag zwischen Backend und Frontend und ermöglichte die parallele Arbeit beider Teams.

Lokales Routing- und Geocoding-Ökosystem Das Backend arbeitet in Bezug auf Geodaten vollständig autark: Drei lokal betriebene Container ersetzen externe Web-APIs. Nominatim löst Adressen in Koordinaten auf und umgekehrt, OSRM berechnet exakte Fahrtrouten im Straßennetz und liefert GeoJSON-Geometrien für die Kartenvisualisierung, und VROOM optimiert

Tabelle 1: Übersicht der API-Endpunkte, gruppiert nach Domänenbereich und HTTP-Methode.

Bereich	**Methode**	**Endpunkt**	**Beschreibung**
Benutzer	POST	/CreateUser	Neuen Benutzer anlegen
	PUT	/UpdateUser	Benutzerdaten aktualisieren
	DELETE	/DeleteUser	Benutzer löschen
	GET	/GetUsers	Alle Benutzer abfragen
	GET	/GetUser	Benutzer per ID abfragen
	GET	/GetUserByEmail	Benutzer per E-Mail abfragen
Fahrzeuge	POST	/CreateVehicle	Fahrzeug anlegen
	PUT	/UpdateVehicle	Fahrzeugdaten aktualisieren
	DELETE	/DeleteVehicle	Fahrzeug löschen
	GET	/GetVehicle	Fahrzeug per ID abfragen
	GET	/GetVehiclesForUser	Fahrzeuge eines Benutzers
Wohnorte	POST	/CreateResidence	Wohnort anlegen (inkl. Geocoding)
	PUT	/UpdateResidence	Wohnort aktualisieren
	DELETE	/DeleteResidence	Wohnort löschen
	GET	/GetResidences	Alle Wohnorte abfragen
	GET	/GetResidencesForUser	Wohnorte eines Benutzers
Fahrpläne	POST	/CreateScheduleEntry	Fahrplaneintrag anlegen
	PUT	/UpdateScheduleEntry	Fahrplaneintrag aktualisieren
	DELETE	/DeleteScheduleEntry	Fahrplaneintrag löschen
	GET	/GetScheduleEntries	Alle Fahrpläne abfragen
	GET	/GetScheduleEntry	Fahrplan per ID abfragen
	GET	/GetScheduleEntriesForUser	Fahrpläne eines Benutzers
Fahrgemein-	PUT	/UpdateCarpoolStatus	Status ändern (bestätigen/ablehnen)
schaften	POST	/CompleteCarpool	Fahrt als durchgeführt markieren
	GET	/GetCarpools	Fahrgemeinschaften (filterbar)
	GET	/GetCarpool	Detail per ID abfragen
	GET	/GetCarpoolParticipants	Teilnehmer einer Fahrgemeinschaft
Unternehmen	GET	/GetCompanyInformations	Firmendaten und CO_2-Statistik

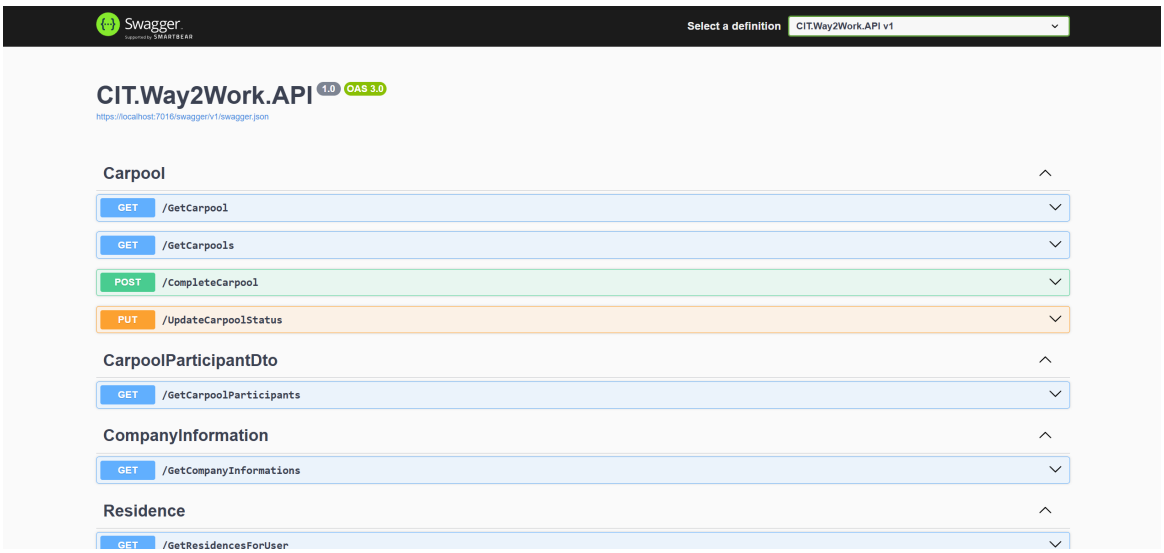


Abbildung 47: Die automatisch generierte OpenAPI-Dokumentation (Swagger UI) zeigt alle verfügbaren Endpunkte inklusive Parameter und Antwortstrukturen.

die Zuordnung von Mitfahrern zu Fahrzeugen inklusive Abholreihenfolge. Als Datengrundlage dienen OpenStreetMap-Daten des Bundeslandes Oberösterreich.

Das Zusammenspiel dieser Dienste ermöglicht es dem Backend, dem Frontend pro Fahrgemeinschaft ein vollständiges Ergebnisobjekt zu liefern: Abholzeiten für jeden Mitfahrer, Gesamtdauer und -distanz, CO_2-Einsparung sowie ein fertiges GeoJSON-`FeatureCollection`-Objekt, das den Streckenverlauf und die beschrifteten Haltepunkte enthält. Das Frontend kann diese Daten ohne weitere Berechnung direkt auf der interaktiven Karte darstellen.

Datenbank Das Datenbankschema wurde über Entity Framework Core im Code-First-Ansatz in eine Azure-SQL-Datenbank überführt. Es bildet die Kernbereiche Benutzer, Fahrzeuge, Wohnorte, Fahrpläne, Fahrgemeinschaften und deren Teilnehmerzuordnungen über relationale Verknüpfungen ab. Migrationen werden über einen eigenen, von der API entkoppelten Migrationsdienst ausgeführt.

9 Resümee

Dieses Kapitel fasst die im Rahmen der Diplomarbeit gewonnenen Erkenntnisse zusammen, reflektiert den Projektverlauf und gibt einen Ausblick auf mögliche Weiterentwicklungen.

9.1 Zusammenfassung und Erkenntnisse

Im Zuge der Umsetzung des Projekts konnte ein umfassender Einblick in einen realitätsnahen Softwareentwicklungsprozess gewonnen werden. Durch die Zusammenarbeit mit der Firma Count IT wurden zentrale Aspekte moderner Softwareentwicklung praxisnah erlebt, insbesondere der strukturierte Einsatz von Versionskontrolle, Pull Requests sowie Code-Quality-Standards.

Ein wesentliches Ergebnis der Arbeit ist die erfolgreiche Entwicklung einer nahezu produktionsreifen Anwendung innerhalb eines vergleichsweise kurzen Zeitraums. Trotz begrenzter Vorerfahrung mit dem verwendeten Technologie-Stack – insbesondere im Bereich Blazor – konnte durch die Nutzung bestehender Frameworks und Open-Source-Lösungen eine funktionale und skalierbare Applikation realisiert werden.

Die Arbeit zeigt deutlich, dass moderne Entwicklungswerkzeuge und Bibliotheken es ermöglichen, auch mit begrenzten Ressourcen leistungsfähige Softwarelösungen umzusetzen. Gleichzeitig wurde ersichtlich, wie wichtig ein solides technisches Fundament ist, um sich effizient in neue Technologien einarbeiten zu können.

9.2 Reflexion des Projektverlaufs

Rückblickend stellte sich insbesondere die strukturierte Projektplanung als entscheidender Erfolgsfaktor heraus. Bereits vor Beginn der eigentlichen Implementierungsphase wurde ein klarer Plan definiert, wodurch eine zielgerichtete Umsetzung innerhalb des vorgegebenen Zeitrahmens ermöglicht wurde.

Darüber hinaus konnten wertvolle Erfahrungen im Bereich Projektmanagement gesammelt werden. Dazu zählen unter anderem die Organisation von Aufgaben, die Abstimmung im Team sowie der Umgang mit Entwicklungsprozessen wie Code-Reviews und Versionsverwaltung.

Eine besondere Herausforderung bestand darin, sich parallel zur Umsetzung in neue Technologien einzuarbeiten. Dennoch konnte gezeigt werden, dass durch eigenständiges Lernen und vorhandene Grundkenntnisse komplexe Systeme in kurzer Zeit verstanden und angewendet werden können.

Obwohl die entwickelte Anwendung noch Optimierungspotenzial aufweist, stellt sie ein solides und funktionsfähiges Ergebnis dar, das sich für einen produktiven Einsatz eignet.

9.3 Persönliche Erkenntnisse und Ausblick

Die Arbeit hat verdeutlicht, dass die im Rahmen der Ausbildung an der HTL vermittelten Grundlagen eine wichtige Basis für die Umsetzung realer Softwareprojekte darstellen. Insbesondere die Fähigkeit, sich eigenständig in neue Technologien einzuarbeiten und Probleme strukturiert zu lösen, konnte weiter gestärkt werden.

Zusätzlich konnte ein tieferes Verständnis für den gesamten Softwareentwicklungsprozess gewonnen werden – von der Planung über die Implementierung bis hin zur Qualitätssicherung.

Ein weiterer Meilenstein des Projekts ist die Teilnahme am Wettbewerb *Jugend Innovativ*, bei dem das Projekt in der Kategorie *Sustainability* zu den Top-Projekten zählt. Dies unterstreicht die Relevanz und das Potenzial der entwickelten Lösung.

Für die Zukunft bietet das Projekt zahlreiche Erweiterungsmöglichkeiten. Eine naheliegende Weiterentwicklung besteht darin, die Plattform auch anderen Unternehmen zur Verfügung zu stellen, beispielsweise weiteren Firmen am Standort Hagenberg. Darüber hinaus wäre eine Erweiterung auf mehrere Unternehmensstandorte denkbar, sodass Fahrgemeinschaften standortübergreifend organisiert und verwaltet werden können.

Diese Skalierung würde den praktischen Nutzen der Anwendung erheblich steigern und den Einsatz in größeren Organisationen ermöglichen. Zusätzlich könnten bestehende Funktionen weiter optimiert sowie neue Features zur Verbesserung der Benutzerfreundlichkeit und Effizienz ergänzt werden.

Abschließend kann festgehalten werden, dass das Projekt nicht nur fachlich, sondern auch persönlich einen bedeutenden Lern- und Entwicklungsschritt darstellt. [14].

[14] Die sprachliche Formulierung sowie einzelne erklärende Passagen dieses Abschnitts wurden unter Einsatz generativer KI (ChatGPT) unterstützt (siehe Anhang C).

Glossar

API Application Programming Interface

Blazor Web-Framework von Microsoft zur Entwicklung interaktiver Webanwendungen mit C# und .NET

CO_2 Kohlenstoffdioxid

Common Language Runtime (CLR) Die virtuelle Maschine und zentrale Laufzeitumgebung des .NET-Frameworks. Sie ist für die Ausführung von .NET-Programmen verantwortlich und übernimmt wichtige Hintergrunddienste. Dazu zählen die Speicherverwaltung (Garbage Collection), die Just-In-Time-Kompilierung (JIT) von Zwischencode in maschinenlesbaren Code sowie die Durchsetzung von Typensicherheit und Sicherheitsrichtlinien.

Contraction Hierarchies (CH) Eine Vorberechnungstechnik zur extrem schnellen Ermittlung kürzester Wege in großen Graphen. Bei diesem Verfahren werden die Knoten des Netzwerks nach ihrer Wichtigkeit sortiert. Weniger wichtige Knoten werden „kontrahiert“ (zusammengezogen) und durch direkte Abkürzungskanten (Shortcuts) zwischen den verbleibenden wichtigen Knoten ersetzt. Bei der eigentlichen Wegesuche wird der Suchraum dadurch drastisch verkleinert, was Antwortzeiten im Millisekundenbereich ermöglicht.

CQRS Command Query Responsibility Segregation

DBMS Database Management System

Entity Framework Core Object-Relational Mapper (ORM) für .NET zur Anbindung relationaler Datenbanken über ein objektorientiertes Datenmodell

ESG Environmental, Social and Governance

MapLibre Open-Source Kartenbibliothek (MapLibre GL) zur Darstellung und Interaktion mit Vektorkarten im Web

MediatR .NET-Bibliothek zur Umsetzung des Mediator-Patterns; häufig in CQRS-Architekturen zur Entkopplung von Requests und Handlern verwendet

Microsoft Azure Cloud-Plattform von Microsoft zur Bereitstellung und zum Betrieb von Diensten (z. B. Webanwendungen, Container, Datenbanken) in einer skalierbaren Infrastruktur

Microsoft Entra ID Identitäts- und Zugriffsmanagement von Microsoft (ehem. Azure AD) für Authentifizierung/Autorisierung; unterstützt u. a. SSO und Mandantenverwaltung

MudBlazor UI-Komponentenbibliothek für Blazor mit Material-Design-orientierten Komponenten (z. B. Tabellen, Dialoge, Formulare)

Multi-Level Dijkstra (MLD) Ein fortschrittlicher Routing-Algorithmus, der auf dem klassischen Dijkstra-Algorithmus aufbaut. MLD unterteilt das Straßennetzwerk in verschiedene hierarchische Zellen (Partitionierung) und berechnet die Wege zwischen den Zellgrenzen im Voraus. Dadurch muss bei einer Suchanfrage nicht das gesamte Netzwerk durchsucht werden, was die Routenberechnung enorm beschleunigt. MLD eignet sich besonders gut für Netzwerke, in denen sich Kantengewichte (wie z. B. aktuelle Verkehrsdaten) häufig ändern.

Nominatim Open-Source-Geocoding-Dienst auf Basis von OpenStreetMap zur Umwandlung von Adressen in Koordinaten (und umgekehrt)

OpenAPI OpenAPI Specification

OSRM Open Source Routing Machine; Routing-Engine zur Berechnung von Routen, Distanzen und Fahrzeiten auf Basis von OpenStreetMap-Daten

REST Representational State Transfer

SignalR SignalR ist eine Bibliothek von Microsoft zur Echtzeitkommunikation zwischen Client und Server. Sie ermöglicht dauerhafte Verbindungen, über die Daten und Ereignisse in Echtzeit übertragen werden können, und wird in Blazor Server für die Synchronisation von Benutzerinteraktionen verwendet.[40]

SSO Single Sign-On

Traveling Salesperson Problem (TSP) Das „Problem des Handlungsreisenden“ ist eines der klassischsten Probleme der Graphentheorie und theoretischen Informatik. Die Aufgabe besteht darin, die kürzestmögliche Route zu finden, auf der eine vorgegebene Menge von Stationen genau einmal besucht wird, bevor man wieder zum Ausgangspunkt zurückkehrt.

Vehicle Routing Problem (VRP) Ein bekanntes kombinatorisches Optimierungsproblem aus der Logistik. Es befasst sich mit der Frage, wie eine Flotte von Fahrzeugen optimal koordiniert werden kann, um eine bestimmte Menge an Stationen (oder Kunden) zu bedienen. Dabei müssen oft komplexe Nebenbedingungen wie Fahrzeugkapazitäten oder strenge Zeitfenster eingehalten werden.

VROOM Open-Source-Optimierungs-Engine (Vehicle Routing Problem) zur Berechnung optimierter Touren/Zuordnungen; kann u. a. OSRM nutzen

Literaturverzeichnis

[1] P. Toth und D. Vigo, *Vehicle Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics (SIAM), 2014.

[2] ACCA Global, „Reward schemes for employees and managers," 2025, Zugriff am: 25.01.2026. Online verfügbar: https://www.accaglobal.com/gb/en/student/exam-support-resources/professional-exams-study-resources/p5/technical-articles/reward-schemes-for-employees-and-management.html

[3] M. Armstrong, „A Handbook of Employee Reward Management and Practice," 2012, Zugriff am: 25.01.2026. Online verfügbar: https://hrproblog.wordpress.com/wp-content/uploads/2012/05/a_handbook_of_employee_reward_management_and_practice.pdf

[4] Microsoft. (2026) .NET documentation. Zugriff am: 19.02.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/core/introduction

[5] Microsoft, „ASP.NET Core Blazor overview," 2025, Zugriff am: 04.01.2026. Online verfügbar: https://learn.microsoft.com/aspnet/core/blazor

[6] Microsoft, „Introduction to Razor Pages in ASP.NET Core," 2025, Online; accessed 2026-01-04. Online verfügbar: https://learn.microsoft.com/aspnet/core/razor-pages

[7] Microsoft, „ASP.NET Core Blazor hosting models," 2025, Zugriff am: 04.01.2026. Online verfügbar: https://learn.microsoft.com/aspnet/core/blazor/hosting-models

[8] Internet Engineering Task Force, „The GeoJSON Format (RFC 7946)," 2025, Zugriff am: 04.01.2026. Online verfügbar: https://datatracker.ietf.org/doc/html/rfc7946

[9] GeoJSON Project, „GeoJSON – A format for encoding geographic data structures," 2025, Zugriff am: 04.01.2026. Online verfügbar: https://geojson.org

[10] GeoBGU. (o.J.) GeoJSON – Geometry Types. Abbildung: simple_feature_types.png Zugriff am 20.02.2026. Online verfügbar: https://geobgu.xyz/web-mapping/geojson-1.html

[11] National Geospatial-Intelligence Agency, „World Geodetic System 1984," 2014, Zugriff am: 04.01.2026. Online verfügbar: https://earth-info.nga.mil/index.php?dir=wgs84

[12] Microsoft, „What is Microsoft Entra ID?" 2025, Zugriff am: 04.01.2026. Online verfügbar: https://learn.microsoft.com/en-us/entra/fundamentals/what-is-entra

[13] Microsoft, „Secure an ASP.NET Core Blazor Web App with Microsoft Entra ID," 2025, Online; accessed 2026-01-04. Online verfügbar: https://learn.microsoft.com/aspnet/core/blazor/security/blazor-web-app-with-entra

[14] Microsoft, „What is SQL Server?" 2025, Zugriff am: 04.01.2026. Online verfügbar: https://learn.microsoft.com/sql/sql-server/what-is-sql-server?view=sql-server-ver17

[15] Microsoft Azure, „What is a relational database?" 2025, Zugriff am: 04.01.2026. Online verfügbar: https://azure.microsoft.com/resources/cloud-computing-dictionary/what-is-a-relational-database

[16] Microsoft. (2026) Visual Studio 2022 IDE. Zugriff am: 22.02.2026. Online verfügbar: https://visualstudio.microsoft.com/

[17] Microsoft Corporation. (2024) Was ist SQL Server Management Studio (SSMS)? Zugriff am: 14.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/sql/ssms/sql-server-management-studio-ssms

[18] R. Hat. (2026) Podman documentation. Zugriff am: 28.02.2026. Online verfügbar: https://podman.io/

[19] Microsoft, „.NET Aspire documentation," 2026, Zugriff am: 25.01.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/aspire/

[20] Aspire Contributors, „What is the AppHost?" 2026, Zugriff am: 25.01.2026. Online verfügbar: https://aspire.dev/get-started/app-host/

[21] SmartBear Software. (2024) What is Swagger? Zugriff am: 14.03.2026. Online verfügbar: https://swagger.io/docs/specification/2-0/what-is-swagger/

[22] Microsoft. (2026) Entity Framework Core documentation. Zugriff am: 24.02.2026. Online verfügbar: https://learn.microsoft.com/en-us/ef/core/

[23] J. Bogard. (2024) AutoMapper Documentation. Zugriff am: 14.03.2026. Online verfügbar: https://docs.automapper.org/

[24] MudBlazor Team, „MudBlazor Documentation," 2025, Zugriff am: 25.01.2026. Online verfügbar: https://mudblazor.com/

[25] MapLibre contributors. (2026) MapLibre GL JS Documentation. Zugriff am: 25.03.2026. Online verfügbar: https://maplibre.org/maplibre-gl-js/docs/

[26] Project OSRM. (2026) Open Source Routing Machine. Zugriff am: 14.03.2026. Online verfügbar: http://project-osrm.org/docs/v5.24.0/api/#

[27] J. Coupey, J.-M. Nicod, und C. Varnier, *Vroom v1.14, Vehicle Routing Open-source Optimization Machine*, Verso (https://verso-optim.com/), Besançon, France, 2024, http://vroom-project.org/.

[28] OpenStreetMap Foundation. (2026) Nominatim - OpenStreetMap Wiki. Zugriff am: 14.03.2026. Online verfügbar: https://wiki.openstreetmap.org/wiki/Nominatim

[29] Microsoft. (2026) Microsoft Teams – Group chat software. Zugriff am: 25.03.2026. Online verfügbar: https://www.microsoft.com/en/microsoft-teams/group-chat-software

[30] Microsoft, „What is Azure DevOps?" 2025, Zugriff am: 25.01.2026. Online verfügbar: https://learn.microsoft.com/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

[31] CAKE.com Inc. (2026) Clockify – Time Tracking Software. Zugriff am: 25.03.2026. Online verfügbar: https://clockify.me/

[32] JGraph Ltd. (2026) draw.io – Diagramming Tool. Zugriff am: 25.03.2026. Online verfügbar: https://www.diagrams.net/

[33] Microsoft Azure, „What is Azure?" 2025, Zugriff am: 25.01.2026. Online verfügbar: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure

[34] Microsoft, „What is the Azure portal?" 2024, Zugriff am: 25.01.2026. Online verfügbar: https://learn.microsoft.com/en-us/azure/azure-portal/azure-portal-overview

[35] Microsoft, „How to register an app in Microsoft Entra ID," 2025, Zugriff am: 25.01.2026. Online verfügbar: https://learn.microsoft.com/en-us/entra/identity-platform/quickstart-register-app

[36] Microsoft, „Azure Container Apps overview," 2025, Zugriff am: 25.01.2026. Online verfügbar: https://learn.microsoft.com/en-us/azure/container-apps/overview

[37] Microsoft Corporation. (2023) Gängige Webanwendungsarchitekturen - Clean Architecture. Zugriff am: 14.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[38] M. Fowler. (2011) CQRS - Command Query Responsibility Segregation. Zugriff am: 14.03.2026. Online verfügbar: https://martinfowler.com/bliki/CQRS.html

[39] E. Kurbegovic. (2023) CQRS Software Architecture Pattern: The Good, The Bad, and The Ugly. Zugriff am: 22.03.2026. Online verfügbar: https://medium.com/@emer.kurbegovic/cqrs-software-architecture-pattern-the-good-the-bad-and-the-ugly-efe48e8dcd14

[40] Microsoft, „Introduction to ASP.NET Core SignalR," 2024, Zugriff am: 04.01.2026. Online verfügbar: https://learn.microsoft.com/aspnet/core/signalr/introduction

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

A.1 Kevin Stocker

Inhaltsverzeichnis Stocker

A.2 Christof Hundegger

Inhaltsverzeichnis Hundegger

B Planung und Realisierung

B.1 Projektorganisation

Die ersten Abstimmungen mit der Firma Count IT GmbH fanden bereits im Jänner 2025 statt. Nach einem Vor-Ort-Termin und dem Vergleich mehrerer möglicher Unternehmen wurde Count IT von uns gewählt. Zu Beginn war das konkrete Projektthema noch nicht vollständig definiert. Im Februar konkretisierte sich die Idee, im Kontext der von Count IT verfolgten ESG-Ziele eine Plattform zur Bildung von Fahrgemeinschaften zu entwickeln. Ziel war es, eine vollständig neue, auf Open-Source-Technologien basierende Lösung von Grund auf zu konzipieren und umzusetzen.

Bereits vor Beginn des vierwöchigen Praktikums wurde ein grobes Umsetzungskonzept ausgearbeitet. Dieses umfasste eine erste Architekturdefinition, ein vorläufiges Entity-Relationship-Diagramm (ERD) sowie eine Aufgabenverteilung im Team. Die Backend-Entwicklung wurde von Christof Hundegger übernommen, während Kevin Stocker für die Umsetzung des Frontends verantwortlich war.

Als technische Ansprechpartner standen während des Praktikums Matthias Furtlehner (technische Betreuung und Projektsetup) sowie Matthias Eder (Abteilungsleitung Custom Solutions, Abnahmen und wöchentliche Statusgespräche) zur Verfügung.

B.2 Meilensteine

Die Umsetzung wurde anhand definierter Meilensteine geplant. Sämtliche Meilensteine wurden im vorgesehenen Zeitrahmen erreicht.

Meilenstein	**Datum**
Die Anforderungsanalyse ist abgeschlossen	21.06.2025
Das Datenmodell ist erstellt, normalisiert und in MS SQL Server implementiert.	04.07.2025
Die REST-API ist in C# entwickelt und alle erforderlichen Endpunkte werden bereitgestellt.	11.07.2025
Die Basisversion der Car Sharing Library ist entwickelt und erste Fahrgemeinschaften sind berechnet.	11.07.2025
Das Frontend ist mit Blazor realisiert und an die REST-API angebunden.	18.07.2025
Der konfigurierbare Algorithmus zur Berechnung optimaler Fahrgemeinschaften ist implementiert.	18.07.2025
Die Integration zwischen Backend und Frontend wurde erfolgreich getestet.	21.07.2025
Das Reward-System ist implementiert und in die Fahrgemeinschaftslogik eingebunden.	21.07.2025

Tabelle 2: Meilensteine und geplante Erreichungsdaten

B.3 Projektverlauf

Der offizielle Projektstart im Rahmen des Praktikums erfolgte am 1. Juli 2025. Die Dauer betrug vier Wochen mit jeweils 38,5 Wochenstunden pro Person. Insgesamt wurden somit 308 Arbeitsstunden im Praktikum erbracht (2 Personen $\times$ 4 Wochen $\times$ 38,5 Stunden).

Die erste Woche diente primär dem Projektsetup sowie der finalen Ausarbeitung der Datenstruktur. Anschließend erfolgte die parallele Entwicklung von Backend und Frontend. Durch die klare Aufgabenverteilung konnte effizient und ohne größere Verzögerungen gearbeitet werden.

Nach Abschluss des Praktikums wurden kleinere Nacharbeiten und Fehlerbehebungen im Umfang von etwa fünf Stunden durchgeführt.

Die Verschriftlichung der Diplomarbeit erfolgte im Zeitraum von November 2025 bis März 2026.

Zur Dokumentation der Arbeitszeiten wurde außerhalb des Praktikums das Tool Clockify verwendet, wodurch eine transparente und nachvollziehbare Zeiterfassung gewährleistet war. Während des Praktikums erfolgte die Zeiterfassung über das interne Zeiterfassungssystem von Count IT.

C Einsatz von generativer KI

C.1 Allgemeine Verwendung

Im Rahmen dieser Diplomarbeit wurden verschiedene generative KI-Werkzeuge unterstützend eingesetzt, darunter ChatGPT, Google Gemini sowie DeepL Write.

Die Nutzung erfolgte ausschließlich zur sprachlichen Überarbeitung, zur Strukturierung von Inhalten sowie zur Formulierung einzelner erklärender Passagen und Überleitungen.

Die fachlichen Inhalte, Konzepte, Implementierungen sowie alle wesentlichen inhaltlichen Entscheidungen wurden eigenständig erarbeitet.

C.2 Verwendung in den einzelnen Kapiteln

- Einführung: Sprachliche Überarbeitung eines bestehenden Projektberichts.
- Motivation / Herausforderungen: Anpassung an wissenschaftlichen Stil.
- Grundlagen / Reward-System: Unterstützung bei Formulierung und Verständlichkeit.
- Diverse Abschnitte: Erstellung von Überleitungen.
- Formulare: Strukturierung von Inhalten in Stichpunkte.
- Glossar: Zusammenfassung und Verdichtung von Definitionen.
- CO2-Berechnung und Punkteverteilung: Unterstützung beim Formulieren.
- Swagger: Sprachliche Überarbeitung.
- Entity Framework Core: Formulierungshilfen.
- VROOM – Routing-Optimierung: Textkürzung.
- Azure Portal: Unterstützung bei Formulierung und Verständlichkeit.
- Projektstruktur: Übersichtliche Aufbereitung der Struktur.
- Datenmodell: Sprachliche korrektur
- Design Patterns: Unterstützung beim Formulieren
- Azure Deployment: Sprachliche Überarbeitung.
- Resümee: Sprachliche Übararbeitung

C.3 Verwendete Prompts

C.3.1 Sprachliche Überarbeitung bestehender Texte

Ziel: Umformulierung bestehender Inhalte in einen wissenschaftlichen Stil.

Prompt:

> Formuliere folgenden Text für eine Diplomarbeit um. Achte auf einen sachlichen, wissenschaftlichen Stil und eine klare Struktur. Der Inhalt soll erhalten bleiben, jedoch sprachlich verbessert werden.
>
> + Zusätzlicher Kontext sowie relevante Informationen aus dem jeweiligen Kapitel werden bei Bedarf bereitgestellt.

C.3.2 Strukturierung von Inhalten in Stichpunkte

Ziel: Aufbereitung von Fließtext in übersichtliche Aufzählungen.

Prompt:

> Strukturiere folgenden Text in übersichtliche Stichpunkte. Achte auf eine klare, logische Gliederung und eine sachliche, präzise Formulierung.

C.3.3 Erstellung von Überleitungen und erklärenden Passagen

Ziel: Verbesserung der Lesbarkeit und Struktur.

Prompt:

> Erstelle eine kurze, sachliche Überleitung für eine Diplomarbeit zwischen folgenden Abschnitten. Der Stil soll klar, präzise und wissenschaftlich sein.
>
> Zusätzlich wurden als Kontext das jeweilige Kapitel, die Hauptüberschriften sowie vereinzelte relevante Textstellen mitgegeben, für die die Überleitung erstellt wurde.

C.3.4 Zusammenfassung und Verdichtung von Definitionen

Ziel: Erstellung kurzer, prägnanter Glossareinträge.

Prompt:

> Fasse folgende technische Definition kurz und prägnant für ein Glossar zusammen. Achte auf eine verständliche, sachliche und kompakte Formulierung.

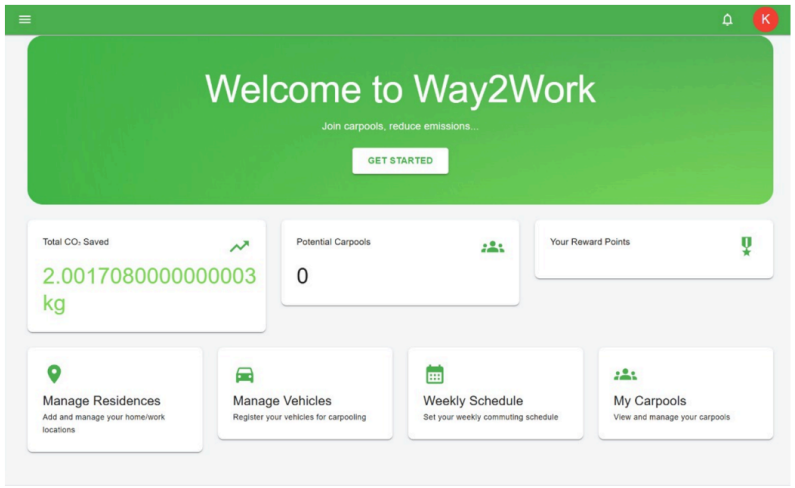
C.4 Abgrenzung der Eigenleistung

Die Verwendung von generativer KI beschränkte sich auf unterstützende Tätigkeiten im sprachlichen Bereich. Es wurden keine Inhalte ungeprüft übernommen.

Alle fachlichen Inhalte, Implementierungen sowie die konzeptionelle Ausarbeitung der Anwendung wurden eigenständig durchgeführt.

way2work

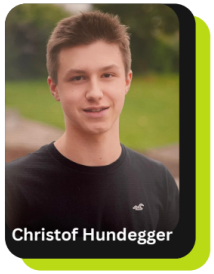
smarte Fahrgemeinschaften



Unser System bildet automatisiert Fahrgemeinschaften für Mitarbeitende, um CO_2-Emissionen zu reduzieren, Fahrzeiten zu optimieren und den Pendelverkehr effizienter zu gestalten.

🌍 weniger CO_2 | 🤝 soziale Vernetzung | 💰 Kostenersparnis | 🚗 weniger Autos

Team



Technologien

.NET Aspire

OSRM
OPEN SOURCE ROUTING MACHINE

COUNT IT GROUP
inspire your business

htlperg

E Logo

Abbildung 48: Logo Way2Work