htlperg
innovativ | kreativ | praxisnah

WattPay

Abrechnungssystem für das Laden von Elektrofahrzeugen mit einer KEBA P40 Wallbox

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/07/2025 – 26/03/2026

Projektmitglieder: Milot Jonuzi

Betreuer: OStR Dipl.-Ing. Roland Eggetsberger

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir 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 habe ich die generativen KI-Tools wie ChatGPT oder DeepL zu folgendem Zweck verwendet: Rechtschreib- und Grammatikprüfung.

Perg, 23.03.2026

Ort, Datum

Unterschrift, Milot Jonuzi

Gendererklärung

Aus Gründen der besseren Lesbarkeit und Verständlichkeit wird in dieser Diplomarbeit bei personenbezogenen Bezeichnungen die grammatikalisch männliche Form verwendet.

Diese Formulierung dient ausschließlich der sprachlichen Vereinfachung und schließt ausdrücklich alle Geschlechter gleichermaßen mit ein. Die Verwendung der männlichen Sprachform impliziert in keiner Weise eine Wertung oder Bevorzugung eines Geschlechts und stellt keinerlei Diskriminierung dar. Sämtliche Aussagen dieser Arbeit gelten gleichermaßen für alle Geschlechter im Sinne der Gleichbehandlung und des Respekts gegenüber jeder Person.

Danksagung

Ich bedanke mich bei allen Personen der HTL Perg, die zur Umsetzung dieser Diplomarbeit beigetragen haben.

Ein besonderer Dank geht an Herrn Professor Roland Eggetsberger, der mich während der gesamten Projektphase fachlich begleitet hat. Seine Rückmeldungen waren ehrlich, direkt und haben mir geholfen, meine Arbeit gezielt zu verbessern. Die Zusammenarbeit war stets unkompliziert und professionell, wofür ich sehr dankbar bin.

Ebenso bedanke ich mich bei Herrn Professor Johannes Oppitz, der mir die Möglichkeit gegeben hat, dieses Projekt eigenständig im schulischen Rahmen durchzuführen. Seine offene Art und die verlässliche Betreuung haben mir die nötige Freiheit und das Vertrauen gegeben, um die Arbeit nach meinen Vorstellungen umzusetzen.

Abstract

With the increasing popularity of electric vehicles, transparent and traceable billing of charging processes is becoming increasingly important. The aim of this thesis is therefore to develop a web-based billing system for KEBA P40 charging stations that enables personalized recording of charging processes using RFID tags as well as automated billing at regular intervals.

The technical implementation is based on the evaluation of data provided via the REST API of the KEBA P40 wallbox. For this purpose, communication with the charging station was analysed and a reliable data acquisition process was implemented. By using RFID tags, individual charging processes can be clearly assigned to their respective users. The collected data is stored in a relational database and serves as the basis for subsequent billing.

For user interaction, a web application was developed using React and TypeScript, in which user information, charging processes and billing data are clearly presented. Supabase is used as the backend solution, providing features such as the database, authentication and server-side processing. Billing is automated on a monthly basis and can be exported both as an invoice and as a CSV file. All system components, including the backend, frontend and database, are containerised and managed using Docker Compose.

Kurzfassung

Mit der zunehmenden Verbreitung von Elektrofahrzeugen gewinnt auch die transparente und nachvollziehbare Abrechnung von Ladevorgängen immer mehr an Bedeutung. Ziel dieser Diplomarbeit ist daher die Entwicklung eines webbasierten Abrechnungssystems für KEBA-P40-Ladestationen, das eine personenbezogene Erfassung von Ladevorgängen mittels RFID-Tags sowie eine automatisierte Abrechnung in regelmäßigen Intervallen ermöglicht.

Die technische Umsetzung basiert auf der Auswertung von Daten, welche über die API-REST-Schnittstelle der KEBA-P40-Wallbox bereitgestellt werden. Dazu wurde die Kommunikation mit der Ladestation analysiert und eine zuverlässige Datenerfassung implementiert. Durch den Einsatz von RFID-Tags können die einzelnen Ladevorgänge eindeutig den jeweiligen Nutzern zugeordnet werden. Die erfassten Daten werden in einer relationalen Datenbank gespeichert und dienen als Grundlage für die spätere Abrechnung.

Für die Benutzerinteraktion wurde eine Webanwendung mit React und TypeScript entwickelt, in der Nutzerinformationen, Ladevorgänge und Abrechnungsdaten übersichtlich dargestellt werden. Als Backend kommt Supabase zum Einsatz, welches unter anderem die Datenbank, die Authentifizierung sowie die serverseitige Verarbeitung bereitstellt. Die Abrechnung erfolgt automatisiert auf monatlicher Basis und kann sowohl als Rechnung als auch als CSV-Datei exportiert werden. Sämtliche Systemkomponenten wie Backend, Frontend und Datenbank sind containerisiert und werden über Docker Compose bereitgestellt, was eine einfache Installation und Wartung ermöglicht.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Durch die zunehmende Nutzung von Elektrofahrzeugen, steigt auch der Bedarf an geeigneten Lösungen für die Erfassung und Abrechnung von Ladevorgängen. Besonders im privaten und halböffentlichen Bereich besteht häufig die Notwendigkeit, Ladevorgänge einzelnen Nutzern eindeutig zuzuordnen und regelmäßig abzurechnen. Viele bestehende Ladesysteme unterstützen diese Anforderungen jedoch unzureichend.

Die Keba P40 Ladestation stellt über eine REST-API umfangreiche Daten zu Ladevorgängen zur Verfügung, jedoch fehlt eine integrierte, webbasierte Lösung zur personenbezogenen Abrechnung dieser Daten. Daraus entstand die Motivation, ein eigenständiges Abrechnungssystem zu entwickeln, welches diese Schnittstelle nutzt und eine automatisierte Auswertung der Ladevorgänge ermöglicht.

1.2 Zielsetzung

Ziel des Projekts ist die Entwicklung eines webbasierten Abrechnungssystems für die KEBA P40 Ladestation. Das System soll die Erfassung von Ladevorgängen ermöglichen und diese mithilfe von RFID-Karten eindeutig zu den einzelnen Nutzern zuordnen. Die erfassten Daten sollen gespeichert, ausgewertet und für eine automatisierte Abrechnung, welche monatlich passiert, verwendet werden. Zusätzlich soll eine übersichtliche Weboberfläche bereitgestellt werden, wo der Nutzer die Ladevorgänge und Abrechnungsdaten einsehen kann.

Für die Benutzeroberfläche soll ein Web-Frontend mit React und Typescript entwickelt werden. Als Backend wird Supabase verwendet, das unter anderem Datenbank, Authentifizierung und serverseitige Logik bereitstellt. Die erfassten Daten sollen in einer relationalen Datenbank gespeichert und verarbeitet werden.

1.3 Projektinhalt

Das System besteht aus mehreren Komponenten, die gemeinsam die Erfassung, Verarbeitung und Abrechnung von Ladevorgängen ermöglichen. Das Zusammenspiel dieser Komponenten stellt sicher, dass die Ladevorgänge eindeutig einzelnen Nutzern zugeordnet, gespeichert und für die Abrechnung ausgewertet werden können.

1.3.1 Anbindung der Ladestation und Datenerfassung

Die KEBA P40 Ladestation stellt über eine REST-API Informationen zu laufenden und abgeschlossenen Ladevorgängen bereit. Genau diese Schnittstelle wird benutzt um relevante Daten wie Ladezeit, Energieverbrauch und RFID-Informationen auszulesen. Die erfassten Daten werden dann aufbereitet und an das Backend weitergeleitet.

1.3.2 Backend und Datenverarbeitung

Das Backend übernimmt die zentrale Geschäftslogik des Systems. Es verarbeitet die gelieferten Daten der Ladestation und ordnet die Ladevorgänge den entsprechenden Nutzern zu und speichert diese in einer relationalen Datenbank.

Zusätzlich ist das Backend für die Verwaltung von Nutzern, RFID-Karten und Abrechnungszeiträumen zuständig. Auf Basis der gespeicherten Daten wird die Abrechnung automatisiert durchgeführt. Das Backend stellt eine REST-API bereit, über die das Frontend auf die benötigten Daten zugreifen kann. Für die Umsetzung wird Supabase als Backend-Plattform eingesetzt.

1.3.3 Frontend und Benutzeroberfläche

Das Frontend bildet die Benutzerschnittstelle des Systems und wird als Webanwendung mit React und Typescript umgesetzt. Über diese Oberfläche können Nutzerinformationen, Ladevorgänge und Abrechnungen übersichtlich dargestellt werden. Administratoren haben die Möglichkeit, Nutzer und RFID-Karten zu verwalten sowie Abrechnungen einzusehen und zu exportieren.

1.3.4 Abrechnung und Bereitstellung des Systems

Die Abrechnung der Ladevorgänge erfolgt automatisiert in regelmäßigen Intervallen. Die berechneten Daten können in Form von Rechnungen oder im CSV Format exportiert werden.

Alle Systemkomponenten, einschließlich Frontend, Backend und Datenbank, werden containerisiert und über Docker bereitgestellt. Dadurch wird eine einfache Installation ermöglicht.

1.4 Projektumfeld

1.4.1 Projektteam

Die Diplomarbeit wird als Einzelprojekt durchgeführt von Milot Jonuzi (siehe Abbildung 1), Schüler aus der HTL Perg. Das Thema der Diplomarbeit wurde aufgrund von Interesse an Webentwicklung und Elektromobilität gewählt.

Abbildung 1: Milot Jonuzi

1.4.2 Betreuung

Die Betreuung der Diplomarbeit erfolgt durch Herrn Professor Eggetsberger, der bei organisatorischen Fragestellungen unterstützend zur Seite steht.

1.4.3 Auftraggeber

Die Diplomarbeit wird schulintern durchgeführt. Als Arbeitsauftraggeber fungiert Herr Professor Oppitz, der die fachlichen Anforderungen definiert und die inhaltliche Ausrichtung der Arbeit begleitet, als auch bei technischen Fragestellungen unterstützt.

2 Grundlagen und Methoden

2.1 Grundlegende Fachbegriffe

In diesem Kapitel werden die grundlegenden Fachbegriffe erläutert, die für das Verständnis dieser Diplomarbeit notwendig sind. Sie bilden die fachliche Grundlage für die beschriebenen Konzepte, Technologien und Umsetzungen.

2.1.1 Elektrofahrzeug

Ein Elektrofahrzeug ist ein Fahrzeug, welches ganz oder teilweise mit elektrischer Energie betrieben wird. Die benötigte Energie wird in einer Batterie gespeichert, die über eine externe Stromquelle aufgeladen werden kann. Der Antrieb erfolgt durch einen oder mehrerer Elektromotoren, wodurch während dem Betrieb keine Abgase entstehen.

Es gibt unterschiedliche Arten von Elektrofahrzeugen. Dazu zählen reine Elektrofahrzeuge, bei denen der Antrieb ausschließlich elektrisch erfolgt. sowie Plug-in-Hybridfahrzeuge, die zusätzlich über einen Verbrennungsmotor verfügen. Reine Elektrofahrzeuge werden ausschließlich über Ladeeinrichtungen wie öffentliche oder private Ladestationen aufgeladen.

Ein wesentlicher Vorteil von Elektrofahrzeugen ist die hohe Energieeffizienz und der emissionsarme Betrieb. Da kein Verbrennungsprozess stattfindet, entstehen während der Fahrt weder CO2 noch Stickoxid-Emissionen. Der tatsächliche Umweltvorteil hängt jedoch auch von der Art der Stromerzeugung ab.

Für den Betrieb von Elektrofahrzeugen ist eine geeignete Ladestation notwendig. Diese ermöglich das sichere und kontrollierte Laden. Besonders im privaten und halböffentlichen Bereich ist es wichtig, den Energieverbrauch genau zu kennen, um eine Abrechnung zu ermöglichen.

Im Rahmen dieser Diplomarbeit bilden Elektrofahrzeuge die Grundlage für die untersuchten Ladevorgänge [1].

2.1.2 Ladestation / Wallbox

Eine Ladestation, auch Wallbox genannt, ist eine feste installierte Einrichtung zum Laden von Elektrofahrzeugen (siehe Abbildung 2). Sie verbindet das Fahrzeug mit dem Stromnetz und steuert den gesamten Ladevorgang. Wallboxen werden vor allem im privaten und halböffentlichen Bereich eingesetzt, zum Beispiel in Wohnanlagen oder Betrieben, und ermöglichen ein sicheres Laden von Elektrofahrzeugen.

Abbildung 2: Keba Wallbox P40 [2]

Moderne Ladestationen überwachen den Ladevorgang und erfassen verschiedene Betriebsdaten. Dazu gehören unter anderem der Beginn und das Ende eines Ladevorgangs, die Ladeleistung sowie die geladene Energiemenge. Zusätzlich verfügen viele Wallboxen über Schutzfunktionen, die einen sicheren Betrieb gewährleisten und den Ladevorgang bei Störungen automatisch unterbrechen.

Die Ladestation der KEBA KeContact-Serie bieten Schnittstellen zur Kommunikation mit externen Systemen. Über diese Schnittstellen kann man Lade - und Statusinfromationen abfragen, sowie Änderungen der Einstellungen der Ladestation vornehmen. Dadurch ist eine Einbindung in Abrechnungs - oder Verwaltungssysteme möglich.

In dieser Diplomarbeit wird eine KEBA P40 Ladestation verwendet. Diese stellt über eine REST-API detallierte Informationen zu Ladevorgängen zur Verfügung.[3][4]

2.1.3 Ladevorgang

Ein Ladevorgang beschreibt den Zeitraum, in dem ein Elektrofahrzeug über eine Ladestation mit elektrischer Energie versorgt wird. Der Ladevorgang beginnt, sobald das Fahrzeug angeschlossen und der Ladevorgang freigegeben wurde. Der Ladevorgang endet entweder nach vollständigem Laden, durch manuelles Beenden oder durch eine Unterbrechnung.

Während eines Ladevorgangs überwacht die Ladestation kontinuierlich den aktuellen Zustand. Dabei werden unter anderem die Ladeleistung, die Dauer des Ladevorgangs sowie die geladene Energiemenge erfasst. Diese Werte werden dann von der Ladestation gemessen und für weitere Auswertungen bereitgestellt.

Zusätzlich können Ladestationen Zustandsinformationen liefern, wie etwa ob ein Ladevorgang aktiv, pausiert oder beendet ist. Das ist wichtig, um den Ablauf eines Ladevorgangs nachvollziehen zu können und mögliche Fehler oder Unterbrechnungen zu erkennen. [4]

2.1.4 RFID

RFID steht für Radio Frequency Identification und bezeichnet eine Technologie zur kontaktlosen Identifikation von Objekten oder Personen mithilfe von Funkwellen. Ein RFID-System besteht grundsätzlich aus zwei Komponenten: einem RFID-Tag und einem Lesegerät. Der RFID-Tag enthält eine eindeutige Kennung, die von einem Lesegerät ausgelesen werden kann.

RFID-Tags können in verschiedenen Formen auftreten, beispielsweise als Karte, Schlüsselanhänger oder Aufkleber. Man benötigt keinen direkten Sichtkontakt zum Lesegerät. Sie können innerhalb einer kurzen Distanz automatisch erkannt werden. Abhängig von der Bauart unterscheidet man zwischen passiven RFID-Tags, die ihre Energie aus dem elektromagnetischen Feld des Lesegeräts beziehen, und aktiven RFID-Tags, die über eine eigene Stromversorgung verfügen.

Die RFID-Technologie wird in vielen Bereichen eingesetzt, darunter Zutrittskontrollsysteme, Logistik, Zeiterfassung und Abrechnungssysteme. Durch die eindeutige Identifikation eignet sich RFID besonders für Anwendungen, bei denen man eindeutig bestimmten Nutzern etwas zuordnen muss.

Bei der Ladestation wird RFID verwendet, um Nutzer vor Beginn eines Ladevorgangs zu identifizieren. Der Ladevorgang kann dann eindeutig einer Person zugeordnet werden. [5]

2.1.5 REST-API

REST wurde im Jahr 2000 von Roy Fielding im seiner Dissertation beschrieben. Es handelt sich dabei um einen Architekturstil, der häufig für die Kommunikation zwischen verteilten Softwaresystemen verwendet wird. REST steht für Representational State Transfer und wird vonr allem bei Webanwendungen eingesetzt.

Das Grundprinzip von REST ist der Zugriff auf sogenannte Ressourcen. Eine Ressource kann zum Beispiel ein Benutzer, ein Ladevorgang oder eine Abrechnung sein. Jede Ressource wird über eine eindeutige Adresse angesprochen.

Für die Kommunikation wird meist das HTTP-Protokoll verwendet. Dabei gibt es verschiedene Befehle, mit denen festgelegt wird, was mit den Daten passieren soll.

Die wichtigsten Befehle	
GET	Daten abrufen
POST	Neue Daten erstellen
PUT	Bestehende Daten ändern
DELETE	Daten löschen

Diese Befehle sind einheitlich festgelegt und werden von vielen Systemen unterstützt. Dadurch können unterschiedliche Programme einfach miteinander kommunizieren.

Die Daten werden meist im JSON-Format übertragen. Dieses Format ist einfach aufgebaut und gut lesbar. Deshalb wird es häufig in Webanwendungen verwendet. Zusäztlich sendet der Server bei jeder Anfrage einen Statuscode zurück. Dieser zeigt an, ob die Anfrage funktioniert hat oder ob ein Fehler aufgetreten ist. Typische Beispiele sind:

- 200 - Anfrage erfolgreich
- 400 - Fehlerhafte Anfrage
- 401 - Kein Zugriff erlaubt
- 404 - Daten nicht gefunden
- 500 - Fehler am Server

REST wird oft verwendet, damit man das Frontend und Backend voneinander trennt. Das Frontend zeigt die Daten an, während das Backend diese verarbeitet und speichert.[6]

2.1.6 Backend

Das Backend ist Teil einer Anwendung, der im Hintergrund arbeitet und für den Benutzer nicht direkt sichtbar ist. Es verarbeitet Daten, führt Berechnungen durch und speichert Informationen in einer Datenbank. Das Backend bildet damit die technische Grundlage einer Webanwendung. Zu den Aufgaben gehören unter anderem die Verwaltung von Benutzern, die Verarbeitung von Anfragen sowie die Bereitstellung von Daten für das Frontend.

Häufig wird über Schnittstellen, wie zum Beispiel REST-APIs, mit anderen Systemen kommuniziert. Dadurch können Daten zwischen verschiedenen Anwendungen ausgetauscht werden. Das Backend entscheidet dabei, welche Daten abgerufen, gespeichert oder verändert werden dürfen. [7]

In dieser Diplomarbeit übernimmt das Backend die Verarbeitung der von der Ladestation gelieferten Daten. Es ordnet die Ladevorgänge den jeweiligen Nutzern zu und speichert diese. Außerdem stellt das Backend die Schnittstelle für das Frontend bereit, über welche Lade - und Abrechnungsdaten angezeigt werden können.

2.1.7 Frontend

Das Frontend ist der Teil einer Anwendung, mit dem Benutzer direkt arbeiten. Es umfasst alle sichtbaren Elemente einer Website oder Webanwendung, wie zum Beispiel Texte, Buttons, Formulare und Tabellen. Das Frontend wird über einen Webbrowser angezeigt und ermöglicht die Interaktion mit dem System.

Zu den Hauptaufgaben des Frontends gehört die Darstellung von Informationen sowie die Weiterleitung von Benutzereingaben an das Backend. Aktionen wie das Anzeigen von Daten, das Ausfüllen von Formularen oder das Starten von Funktionen erfolgen über das Frontend.

Das Frontend kommuniziert mit dem Backend über Schnittstellen, wie zum Beispiel REST-APIs. Über diese Schnittstellen werden Daten vom Backend abgefragt. Das Frontend selbst speichert in der Regel keine Daten dauerhaft, sondern stellt diese nur dar. [8]

In diesem Projekt dient das Frontend zur Anzeige von Nutzerdaten, Ladevorgängen und Abrechnungen und ermöglicht eine einfache Bedienung des Systems.

2.1.8 Relationale Datenbank

Eine relationale Datenbank dient zur strukturierten Speicherung von Daten. Die Daten werden in Tabellen gespeichert, die aus Zeilen und Spalten bestehen. Jede Tabelle enthält Informationen zu einem bestimmten Bereich, zum Beispiel zu Nutzern oder Ladevorgängen.

Tabellen können über Beziehungen miteinander verknüpft werden. Dadurch lassen sich zusammengehörige Daten einfach verbinden, ohne sie mehrfach speichern zu müssen.

Relationale Datenbanken verwenden in der Regel eine Abfragesprache wie SQL (Structured Query Language). Mit dieser Sprache können Daten gespeichert, geändert, und abgefragt werden. SQL ermöglicht es, gezielt nach bestimmten Informationen zu suchen oder Daten für Auswertungen zusammenzustellen.

Ein großer Vorteil relationaler Datenbanken ist, dass die gespeicherten Daten zuverlässig und korrekt bleiben. Durch klare Regeln und feste Beziehungen zwischen den Tabellen wird sichergestellt, dass keine falschen oder unvollständigen Daten gespeichert werden. [9]

2.2 Verwendete Technologien

Für die Umsetzung dieser Diplomarbeit wurden mehrere Technologien und Werkzeuge eingesetzt. Sie werden hauptsächlich für die Entwicklung der Webanwendung benötigt. Einige Werkzeuge helfen bei der Programmierung, andere bei der Gestaltung der Benutzeroberfläche oder bei der Dokumentation der Schnittstellen.

2.2.1 NPM

NPM (Siehe Logo 3) bedeutet Node Package Manager. Es ist ein Werkzeug, mit dem man Zusatzpakete für JavaScript-Projekte verwalten kann. In vielen Webprojekten werden Bibliotheken verwendet, die nicht selbst geschrieben werden müssen. Diese Bibliotheken werden über NPM installiert.

Abbildung 3: NPM Logo
[10]

In diesem Projekt wird NPM verwendet, um alle Abhängigkeiten für das Frontend zu verwalten. Dazu gehören zum Beispiel React 6, Tailwind CSS 4. Ein großer Vorteil ist, dass das Projekt auf jeden Computer gleich gestartet werden kann, solange NPM verwendet wird. [11]

2.2.2 Tailwind CSS

Tailwind CSS (Siehe Logo 4) ist für das Design einer Website zuständig. Normalerweise schreibt man CSS-Dateien, um Farben, Abstände oder Schriftgrößen festzulegen. Tailwind arbeitet anders: Es stellt sehr viele fertige Klassen bereit, die direkt im Code verwendet werden können.

Abbildung 4: Tailwind CSS Logo

Mit Tailwind können Abstände, Schriftgrößen, Farben, Rahmen und Schatten, Layouts mit Flexbox oder Grid einfach umgesetzt werden.

Der Vorteil ist, dass man schnell ein einheitliches Design erstellen kann. Außerdem ist es leichter das Design später anzupassen, weil viele Einstellungen direkt beim jeweiligen Element stehen. [12]

2.2.3 shadcn-ui

Shadncn-ui ist eine Sammlung von fertigen Bausteinen für Benuteroberflächen. Diese Bausteine nennt man UI-Komponenten. Beispiele dafür sind Buttons, Dialogfenster, Tabellen, Eingabefelder, Menüs oder Dropdowns

Der Vorteil von shadcn-ui ist, dass viele Teile nicht komplett neu erstellt werden müssen. Die Komponenten sind bereits gut gestaltet und können trotzdem angepasst werden. [13]

In diesem Projekt wird shadcn-ui verwendet, um wichtige Elemente der Oberfläche schneller zu erstellen und ein einheitliches Design im ganzen System zu erreichen.

2.2.4 TypeScript

JavaScript ist die Standard-Sprache für viele Webanwendungen. TypeScript (Siehe Logo 5) ist nun eine Erweiterung und ergänzt JavaScript um sogenannte Typen. Das bedeutet: Variablen und Funktionen bekommen eine klare Beschreibung, welche Art von Daten sie enthalten dürfen.

Abbildung 5: TypeScript Logo [14]

Ein einfaches Beispiel: Eine Variable kann als Zahl definiert werden und man kann dort nicht aus Versehen einen Text speichern.

Viele Fehler werden dabei schon beim Programmieren erkannt, was besonders bei größeren Projekten vorteilhaft ist. Zusätzlich wird der Code verständlicher weil man sofort sieht, welche Daten erwartet werden. [15]

2.2.5 React

React (Siehe Logo 6) ist eine Bibliothek, mit der man Benutzeroberflächen erstellen kann. React arbeitet mit sogenannten Komponenten. Eine Komponente ist ein Baustein der Oberfläche. Zum Beispiel eine Tabelle, ein Formular, eine Liste von Ladevorgängen oder eine einzelne Ansicht für eine Abrechnung

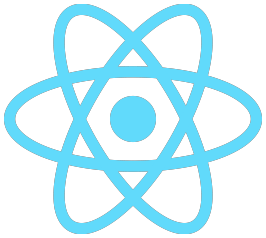

Abbildung 6: React Logo [16]

Diese Komponenten können wiederverwendet werden. Dadurch muss man Code nicht doppelt schreiben. React aktualisiert außerdem automatisch die Oberfläche, wenn sich Daten ändern. Wenn also neue Ladevorgänge in der Datenbank gespeichert werden, kann die Anzeige im Frontend automatisch aktualisiert werden. [17]

2.2.6 Swagger

In Webprojekten kommunizieren Frontend und Backend über eine API. Damit klar ist, welche Endpunkte es gibt und welche Daten gesendet werden müssen, ist eine Dokumentation wichtig. Swagger (Siehe Logo 7) ist nun ein Werkzeug zur Beschreibung und Dokumentation von Schnitstellen. Damit kann man alle API-Endpunkte übersichtlich anzeigen, die Parameter und Antworten beschreiben Und sogar Testanfragen direkt im Browser ermöglichen [19]

Abbildung 7: Swagger Logo [18]

2.3 Verwendete Datenbank

2.3.1 PostgreSQL

PostgreSQL (Siehe Logo 8) ist ein freies und quelloffenes Datenbankmanagementsystem. Es wird verwendet, um Daten zuverlässig zu speichern, verwalten und abzufragen. PostgreSQL gehört zu den sogenannten relationalen Datenbanken (siehe Abschnitt 2.1.8) und wird weltweit in vielen Anwendungen eingesetzt, sowohl in kleinen als auch in großen Unternehmenssystemen.

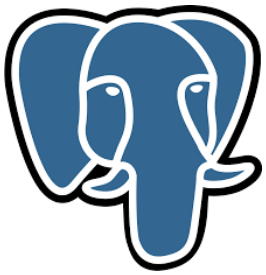

Abbildung 8: PostgreSQL Logo [20]

Ein wichtiger Vorteil von PostgreSQL ist seine Stabilität und Zuverlässigkeit. Das System wird seit vielen Jahren weiterentwickelt und legt großen Wert auf korrekte Datenverarbeitung. Dadurch eignet sich PostgreSQL besonders für Anwendungen, bei denen Daten dauerhaft gespeichert und einwandfrei verarbeitet werden müssen.

PostgreSQL speichert Daten in Tabellen, die aus Zeilen und Spalten bestehen. Diese Tabellen können über Beziehungen miteinander verknüpft werden. Dadurch lassen sich zusammengehörige Daten logisch strukturieren, zum Beispiel Nutzer, Ladevorgänge und Abrechnungen.

Ein weiterer positiver Aspekt ist die gute Unterstützung für Datensicherheit und Datenintegrität. Durch Regeln, Einschränkungen und Transaktionen wird sichergestellt, dass gespeicherte Daten korrekt bleiben und keine fehlerhaften Zustände entstehen. Dies ist besonders wichtig für Anwendungen, bei denen Daten nachvollziehbar und konsistent gespeichert werden müssen. [21]

2.4 Verwendete Entwicklungssysteme

2.4.1 Supabase

Das Backend der Anwendung wurde mit Supabase umgesetzt. Supabase (Siehe Logo 9) ist eine Plattform, welche zentrale Backend-Funktionen schon bereitstellt. Dazu gehören unter anderem eine PostgreSQL Datenbank, automatisch generierte APIs, eine Benutzerverwaltung mit Authentifizierung sowie Edge Functions für serverseitige Logik. Dadurch muss man nicht selbst eine Backendinfrastrukur von Grund auf entwickeln.

Abbildung 9: Supabase Logo
[22]

Ein wichtiger Bestandteil des Backends ist die Authentifizierung. Benutzern ist es möglich, sich sicher anzumelden und ihre Identität zu bestätigen. Supabase bietet dafür ein Authentifizierungssystem. Es werden Funktionen wie die Benutzerregistrierung, die Anmeldung und die Verwaltung von Benutzerkonten unterstützt. Dadurch kann gesteuert werden, welche Benutzer auf bestimmte Funktionen zugreifen dürfen.

Es werden auch Edge Functions angeboten, um serverseitige Logik auszuführen. Das sind kleine Funktionen, die bestimmte Aufgaben übernehmen können und im Backend ausgeführt werden. Durch die Nutzung dieser Funktionen kann ein Teil der Andwendungslogik im Backend umgesetzt werden. So kann man sich im Frontend hauptsächlich auf die Benutzeroberfläche konzentrieren.

Die API ist ein weiterer zentraler Aspekt des Backends. Supabase stellt automatisch eine REST-API bereit. Diese API wird automatisch aus dem Datenbankschema generiert und erlaubt typische CURD-Operationen, also das Erstellen, Lesen, Aktualisieren und Löschen von Daten. Außerdem arbeitet diese API mit den Sicherheitsfunktionen von PostgreSQL zusammen. Zum Beispiel mit Rollen und Row Level Security. Mit Rollen legt man fest, welche Art von Zugriff erlaubt ist und Row Level Security sorgt dafür, dass Benutzer nur die Daten sehen oder ändern können, für die sich auch berechtigt sind.

2.4.2 Visual Studio Code

Visual Studio Code (Siehe Logo 10) ist ein kostenloser und plattformübergreifender Quellcode-Editor von Microsoft. Er wird für die Entwicklung von Software und Webanwendungen verwendet und unterstützt viele verschiedene Programmiersprachen. Die Software läuft auf Windows, macOS und Linux und kann flexibel an unterschiedlichen Projekten angepasst werden.

Abbildung 10: Visual Studio Code Logo

[23]

Ein Vorteil ist, dass man über sogenannte Extensions zusätzliche Funktionen installieren kann für zum Beispiel bestimmte Programmiersprachen, Datenbanken oder Werkzeuge wie Docker. Dadurch kann der Editor an die Anforderungen, des jeweiligen Projekts angepasst werden. Visual Studio Code bietet viele Funktionen, die die Entwicklung erleichtern [24]. Dazu gehören unter anderem:

- Syntaxhervorhebung
- Automatische Code-Vervollständigung
- Fehleranzeige direkt im Editor
- Integrierte Git-Unterstützung
- Ein eingebautes Terminal

2.4.3 Postman

Postman ist ein Werkzeug zum Entwickeln, Testen und Dokumentieren von APIs. Es wird häufig verwendet, um Anfragen an eine REST-API zu senden und die Antworten zu überprüfen. Dadurch können Schnittstellen einfach getestet werden, ohne dass bereits ein fertiges Frontend vorhanden sein muss.

Abbildung 11: Postman Logo

[25]

Mit Postman können verschiedene HTTP-Anfragen erstellen werden, wie zum Beispiel GET, POST, PUT oder DELETE. Dabei können Parameter, Header oder ein JSON-Body definiert werden. Die Antwort des Servers wird anschließend übersichtlich dargestellt. So kann man auch überprüfen, ob die API korrekt funktioniert und die erwarteten Daten liefert. [26]

2.4.4 Figma

Figma ist ein webbasiertes Design-Werkzeug zur Erstellung von Benutzeroberflächen. Es wird verwendet, um Layouts, Mockups und Prototypen für Websites oder Anwendungen zu entwerfen. Figma läuft direkt im Browser und benötigt keine Installation. Dadurch kann man von verschiedenen Geräten auf Projekte zugreifen.

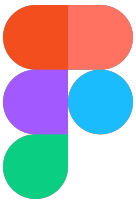

Abbildung 12: Figma Logo [27]

Mit Figma können einzelne Oberflächenelemente wie Buttons, Eingabefelder oder Menüs gestaltet werden und anschließend zu vollständigen Seiten zusammengesetzt werden. Diese Entwürfe dienen als visuelle Vorlage für die spätere Umsetzung im Code. So kann das Design geplant werden, bevor mit der eigentlichen Programmierung begonnen wird. [28]

2.5 Verworfene Optionen

Zu Beginn der Entwicklung der Webanwendung wurde geplant, ein eigenes Backend in Java zu implementieren. Im Laufe der Arbeit wurde die Plattform Supabase 9 entdeckt.

Ein wichtiger Aspekt für die Entscheidung war der deutlich geringere Entwicklungsaufwand, da das Projektteam nur aus einer Person besteht. Dazu kommt, dass Supabase bereits viele grundlegende Funktionen zur Verfügung stellt, wie zum Beispiel Datenbankzugriff, Authentifizierung und API-Bereitstellung. Diese Funktionen hätten sonst bei einer eigenen Backend-Lösung zuerst selbst entwickelt werden müssen.

Die integrierte PostgreSQL-Datenbank ist auch ein weiterer Vorteil, sowie der automatische Zugriff auf die Datenbank mit einer API. Dadurch konnte die Kommunikation zwischen Frontend und Backend vereinfacht werden. Bei einer eigenen Java-Lösung hätte die API zuerst entwickelt und getestet werden müssen. Auch die Authentifizierung war ein entscheidender Faktor. Supabase stellt bereits fertige Lösungen für Benutzerverwaltung und Login bereit.

Zusätzlich wurde zu Beginn überlegt, die Daten der Wallbox über eine UDP-Schnittstelle auszulesen. Diese Möglichkeit besteht jedoch nur bei älteren Modellen der KEBA Wallbox (z. B. der P30-Serie). Die in diesem Projekt verwendete Wallbox unterstützt keine UDP-Schnittstelle mehr, sondern stellt ausschließlich eine REST-API zur Verfügung. Daher wurde die Kommunikation direkt über die vorhandene API umgesetzt.

3 Implementierung

3.1 Technischer Überblick

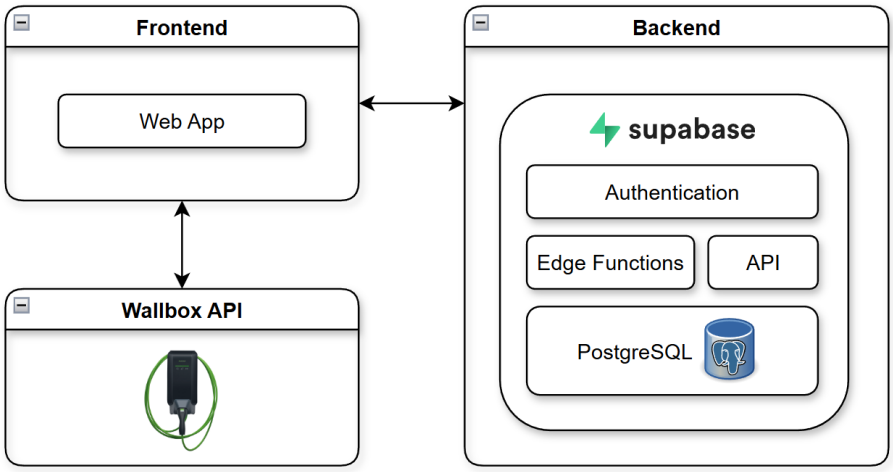

Abbildung 13: Architektur von WattPay

Die Abbildung zeigt wie Wattpay aufgebaut ist. Das System besteht aus dem Frontend, dem Supabase Backend welche gewisse Funktionen wie die Authentifizierung, Edge Functions, einer Backend API und die Datenbank zu Verfügung stellt, und der Wallbox API, mit der die Daten der Wallbox abgefragt werden.

3.2 Backend

3.2.1 Supabase

Für die Umsetzung des Backends wurde Supabase verwendet. Supabase übernimmt dabei vor allem die Benutzerverwaltung und die Anbindung an die Datenbank. Dadurch konnte auf die Entwicklung eines eigenen klassischen Backend-Servers verzichtet werden.

Das Frontend kommuniziert in der Anwendung nicht direkt mit Supabase, um aktuelle Daten von der Wallbox zu erhalten. Stattdessen werden die benötigten Informationen zunächst über die Wallbox API abgerufen und dann im Frontend angezeigt. Anschließend werden diese Daten in der Datenbank gespeichert.

3.2.2 Authentication

In dem Projekt ist die Authentifizierung ein sehr wichtiger Bestandteil. Sie wird verwendet, damit sich Benutzer in der Anwendung mit E-Mail und Passwort anmelden und nur auf ihre eigenen Daten zugreifen können. Nach der Anmeldung erhält der Benutzer eine gültige Sitzung, die im Frontend weiterverwendet wird.

3.3 Datenmodell

Für die Speicherung der Anwendungsdaten wird eine Datenbank basierend auf PostgreSQL verwendet, welche über Supabase verwaltet wird. In ihr werden Benutzerdaten, Statusdaten der Wallbox, Ladevorgänge, Tarife und Rechnungen gespeichert.

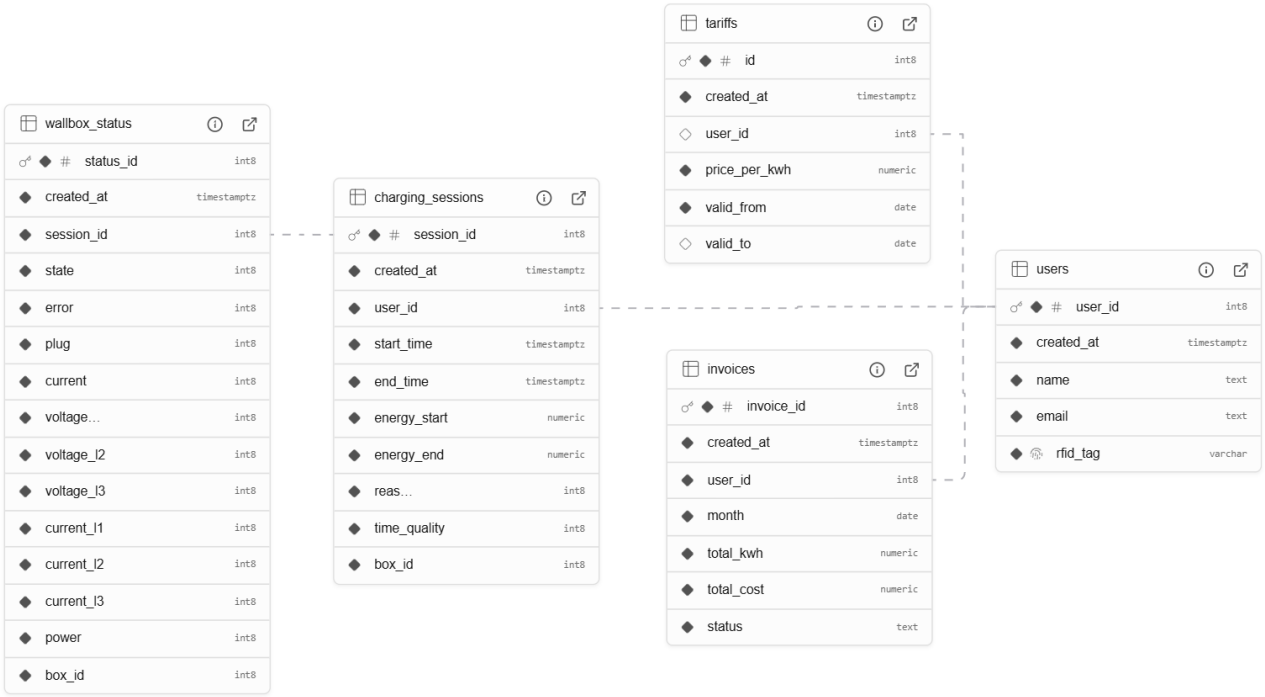

Abbildung 14: Visualisierung des Datenbank Schema

3.3.1 Tabellenstruktur

users

Die Tabelle ”users” speichert die Benutzerdaten. Dazu gehören Benutzer-ID, das Erstelldatum, der Name, die Email-Adresse sowie ein RFID-Tag. Die User-ID fungiert dabei als Primärschlüssel. Über diese Tabelle kann man einen Benutzer eindeutig identifizieren.

charging sessions

Die Tabelle ”charging sessions” speichert Informationen zu einzelnen Ladevorgängen. Für jede Ladesitzung werden unter anderem der zugehörige Benutzer, Start - und Endzeit, der Energiezählerstand zu Beginn und am Ende sowie weitere technische Informationen gespeichert. Jede Sitzung wird über die Session-ID eindeutig identifiziert.

wallbox status

Die Tabelle ”wallbox status” enthält Statusdaten der Wallbox. Dazu gehören beispielsweise der aktuelle Zustand der Wallbox, Fehlercodes, Informationen zum Steckerstatus, Stromstärken, Spannungswerte und die aktuelle Leistung. Außerdem enthält die Tabelle eine Verknüpfung zu einer Ladesitzung über Session-ID. Dadurch können Statuswerte einem bestimmten Ladevorgang zugeordnet werden.

tariffs

Die Tabelle ”tariffs” speichert die Stromtarife der Benutzer. Hier werden neben der Tarif-ID auch die Benutzer-ID, der Preis pro Kilowattstunde sowie der Gültigkeitszeitraum gespeichert. Dadurch können unterschiedliche Tarife zeitlich zugeordnet und bei späteren Berechnungen berücksichtigt werden.

invoices

Die Tabelle ”invoices” enthält die erzeugten Abrechnungsdaten. Gespeichert werden unter anderem der Benutzer, der Abrechnungsmonat, die gesamte geladene Energiemenge, die Gesamtkosten sowie der Status der Rechnung. Damit dient diese Tabelle zur Verwaltung der monatlichen Abrechnung.

3.3.2 Relationen

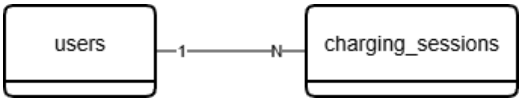

Abbildung 15: Beziehung zwischen ”users” und ”charging sessions”

Die Tabelle ”users” bildet die zentrale Grundlage für mehrere weitere Tabellen. Zwischen ”users” und ”charging sessions” besteht eine 1:n-Beziehung. Das bedeutet, dass ein Benutzer mehrere Ladevorgänge haben kann, während jeder Ladevorgang einem Benutzer zugeordnet ist. Diese Zuordnung erfolgt über das Feld ”user-id”

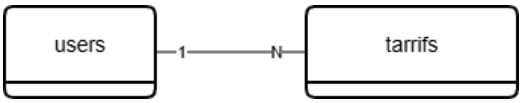

Abbildung 16: Beziehung zwischen ”users” und ”tarrifs”

Auch zwischen ”users” und ”tariffs” besteht eine 1:n-Beziehung. Ein Benutzer kann im Laufe der Zeit mehrere Tarife besitzen, zum Beispiel wenn sich der Strompreis ändert. Jeder Tarifdatensatz gehört jedoch genau zu einem Benutzer.

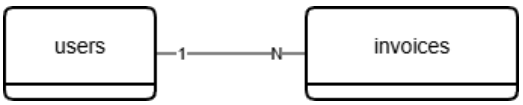

Abbildung 17: Beziehung zwischen ”users” und ”invoices”

Ebenso besteht zwischen ”users” und ”invoices” eine 1:n-Beziehung. Ein Benutzer kann mehrere Rechnungen erhalten, beispielsweise für unterschiedliche Monate. Jede Rechnung ist dabei genau einem Benutzer zugeordnet.

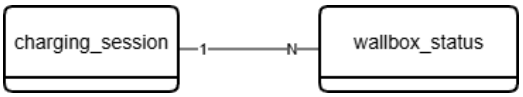

Abbildung 18: Beziehung zwischen ”charging sessions” und ”wallbox status”

Zwischen ”charging sessions” und ”wallbox status” besteht ebenfalls eine Beziehung über das Feld ”session-id”. Dadurch können Statusinformationen der Wallbox einem bestimmten Ladevorgang zugeordnet werden. So ist nachvollziehbar, welche technischen Zustände während einer bestimmten Ladesitzung vorlagen.

3.4 REST-API / Schnittstellen

3.4.1 Kommunikation Frontend - Backend

Die Kommunikation zwischen Frontend und Backend wird vor allem hauptsächlich für die Benutzeranmeldung und den Zugriff auf gespeicherte Daten verwendet.

Benutzer melden sich mit einer E-Mail und einem Passwort an. Die Überprüfung der Anmeldedaten erfolgt über Supabase. Wenn die Anmeldung erfolgreich ist, können Benutzer die geschützten Bereiche der Anwendung verwenden.

Listing 1: Benutzeranmeldung mit Supabase

```
try {
  // Sendet E-Mail und Passwort an Supabase
  const { error } = await supabase.auth.signInWithPassword({
    email,
    password,
  });

  // Wenn ein Fehler auftritt, wird dieser ausgeloest
  if (error) {
    throw error;
  }

  // Zeigt eine Erfolgsmeldung an
  toast({
    title: "Anmeldung erfolgreich",
    description: "Willkommen zurueck!",
  });

  // Leitet den Benutzer auf die Startseite weiter
  navigate("/");
} catch (error: any) {
  // Zeigt eine Fehlermeldung an
  toast({
    variant: "destructive",
    title: "Anmeldung fehlgeschlagen",
    description: error.message,
  });
} finally {
  // Beendet den Ladezustand
  setLoading(false);
}
```

Der Code (Siehe 1) zeigt die Anmeldung eines Benutzers mit Supabase. Dabei werden die eingegebene E-Mail-Adresse und das Passwort an die Methode `signInWithPassword()` übergeben. Wenn die Anmeldung erfolgreich ist, erhält der Benutzer eine Erfolgsmeldung und wird auf die Startseite weitergeleitet. Tritt ein Fehler auf, wird eine Fehlermeldung angezeigt. Am Ende wird der Ladezustand wieder zurückgesetzt.

Außerdem wird die Verbindung auch genutzt, um Daten aus der Datenbank abzurufen oder zu speichern. Dadurch können Informationen nicht nur kurz angezeigt, sondern auch später wieder erneut geladen und im Frontend dargestellt werden.

3.4.2 Kommunikation zur Wallbox API

Die aktuellen Daten der Wallbox werden direkt im Frontend aufgerufen. Dafür wurde eine eigene Client-Klasse erstellt. Diese Klasse bündelt die Kommunikation mit der KEBA-Wallbox API und führt die benötigten HTTP-Anfragen an die Schnittstelle aus.

Die Anfragen an die API werden mit ”fetch” ausgeführt. Das ist eine Funktion in JavaScript, mit der man HTTP-Anfragen an eine API schicken kann. Als Basis wird die Adresse der Wallbox verwendet. Über den API-Client werden verschiedene Endpunkte der Wallbox API angesprochen. Dazu gehören unter anderem der Abruf von Informationen zur Wallbox selbst, Sitzungsdaten, statistischen Werten und Exportdaten.

Beispiele
/v2/wallboxes
/v2/sessions
/v2/sessions/stats
/v2/sessions/export

Listing 2: Zentrale Methode für Anfragen an die Wallbox API

```
private async makeRequest<T>(
  endpoint: string,
  options: RequestInit = {}
): Promise<T> {
  // Setzt die vollstaendige URL aus Basisadresse und Endpunkt zusammen
  const url = `${this.baseURL}${endpoint}`;

  // Erstellt die Header fuer die Anfrage
  const headers: Record<string, string> = {
    'Content-Type': 'application/json',
    ...(options.headers as Record<string, string> || {}),
  };

  // Fuegt bei geschuetzten Endpunkten das Bearer-Token hinzu
  if (!endpoint.includes('/jwt/')) {
    const token = this.accessToken || this.envToken;
    if (token) {
      headers.Authorization = `Bearer ${token}`;
    }
  }

  // Sendet die Anfrage an die Wallbox API
  const response = await fetch(url, {
    ...options,
    headers,
  });

  // Prueft, ob ein Fehler bei der Anfrage aufgetreten ist
  if (!response.ok) {
    const errorData = await response.text();
    throw new Error(`API Error: ${response.status} - ${errorData}`);
  }

  // Gibt die Antwort als JSON zurueck
  return response.json();
}
```

Im Code (Siehe 2) ist die zentrale Methode für die Kommunikation mit der Wallbox API dargestellt. Die Methode `makeRequest()` wird verwendet, um Anfragen an verschiedene Endpunkte der API zu senden. Zuerst wird aus der Basisadresse und dem übergebenen Endpunkt die vollständige URL zusammengesetzt. Anschließend werden die benötigten Header definiert. Dabei wird festgelegt, dass Daten im JSON-Format übertragen werden.

Für geschützte Endpunkte wird zusätzlich ein Bearer-Token in den Header eingefügt. Dadurch kann sich die Anwendung gegenüber der Wallbox API authentifizieren. Die Anfrage wird an die Wallbox API gesendet. Wenn dabei ein Fehler auftritt, gibt die Methode eine Fehlermeldung zurück. Wenn die Anfrage erfolgreich ist, werden die Daten als JSON zurückgegeben. Diese Daten können danach im Frontend angezeigt oder weiterverarbeitet werden.

Der Vorteil dieser Methode ist, dass alle API-Aufrufe auf die gleiche Weise ablaufen. Dadurch ist der Code übersichtlicher, und wichtige Schritte wie Authentifizierung, Fehlerbehandlung und Verarbeitung der Antwort müssen nicht mehrfach geschrieben werden.

Listing 3: Abruf der Wallbox-Daten über die KEBA API

```
1  async getWallboxes() {
2    return this.makeRequest('/v2/wallboxes');
3  }
```

Im Code (Siehe 3) ist eine Methode zu sehen, mit der Daten von der Wallbox API geladen werden. Dabei wird der Endpunkt `/v2/wallboxes` an die Methode `makeRequest()` übergeben. Diese Methode sendet die Anfrage an die API, setzt die nötigen Header und verarbeitet die Antwort.

3.5 Webanwendung

3.5.1 Aufbau der Webb App

Die Webanwendung ist in mehrere getrennte Bereiche unterteilt. Die Navigation erfolgt über eine Seitenleiste. Die fünf Hauptbereiche sind in Dashboard, Ladehistorie, Abrechnung, Live-Daten und den Einstellungen unterteilt.

Das Dashboard dient als Startseite der Anwendung. Für Benutzer dient diese als schneller Überblick über die wichtigsten Informationen. Angezeigt werden der aktuelle Status der Wallbox, die bisher geladene Energie, die Ladezeit sowie zusammengefasste Werte wie Monatsverbrauch, Kosten und die Anzahl der Ladevorgänge.

Der Bereich Abrechnung zeigt die Kosten des geladenen Stroms. Dort sieht der Benutzer die aktuelle Abrechnungsperiode, den gesamten Energieverbrauch und den Preis pro Kilowattstunde. Zusätzlich gibt es eine Übersicht über die Jahreswerte. Rechnungen können außerdem als PDF oder CSV heruntergeladen werden.

Listing 4: Export von Sitzungsdaten als CSV-Datei

```
const handleExport = () => {
  // Startet den Export der Sitzungsdaten
  exportMutation.mutate(
    // Filtert die Daten auf den Zeitraum des letzten Jahres
    { f1: `SESSION_START_DATE=${Date.now() - 365 * 24 * 60 * 60 * 1000}` },
    {
      onSuccess: (blob) => {
        // Erstellt eine temporaere URL fuer die exportierte Datei
        const url = window.URL.createObjectURL(blob);

        // Erstellt einen unsichtbaren Download-Link
        const link = document.createElement('a');
        link.href = url;

        // Legt den Dateinamen der CSV-Datei fest
        link.download = `keba-sessions-${new Date().toISOString().split('T')[0]}.csv`;

        // Fuegt den Link kurz zur Seite hinzu und startet den Download
        document.body.appendChild(link);
        link.click();

        // Entfernt den Link nach dem Download wieder
        document.body.removeChild(link);

        // Gibt die temporaere URL wieder frei
        window.URL.revokeObjectURL(url);
      }
    }
  );
};
```

Der Code (Siehe 4) zeigt den Export von Sitzungsdaten als CSV-Datei. Zuerst wird der Exportvorgang gestartet und auf Daten aus dem letzten Jahr eingeschraenkt. Wenn der Export erfolgreich ist, wird aus der Antwort eine temporaere Datei erstellt. Anschliessend wird automatisch ein Download-Link erzeugt, ueber den die CSV-Datei heruntergeladen wird. Danach wird der Link wieder entfernt und die temporaere Datei freigegeben.

Im Bereich Live-Daten werden aktuelle Informationen der Wallbox angezeigt. Dazu gehören zum Beispiel Strom, Spannung, Leistung, Energieverbrauch und Ladezeit. Zusätzlich werden weitere Informationen wie der Leistungsverlauf, Details zu den einzelnen Phasen, Informationen zur Wallbox und Daten zur aktuellen Ladesitzung dargestellt.

Der Bereich Einstellungen dient zur Verwaltung der persönlichen Daten. Benutzer können dort ihre persönlichen Informationen ansehen, ihr Passwort ändern und Einstellungen für die Abrechnung anpassen.

4 Ergebnis

4.1 Web App

Im Rahmen dieser Arbeit wurde eine Webanwendung entwickelt, die als Benutzeroberfläche für das Laden mit einer KEBA P40 Wallbox dient. Über diese Anwendung können Benutzer Informationen zur Wallbox anzeigen, Ladevorgänge einsehen und Abrechnungen verwalten. In den folgenden Abbildung sind ausschließlich Testdaten zu sehen.

4.1.1 Dashboard

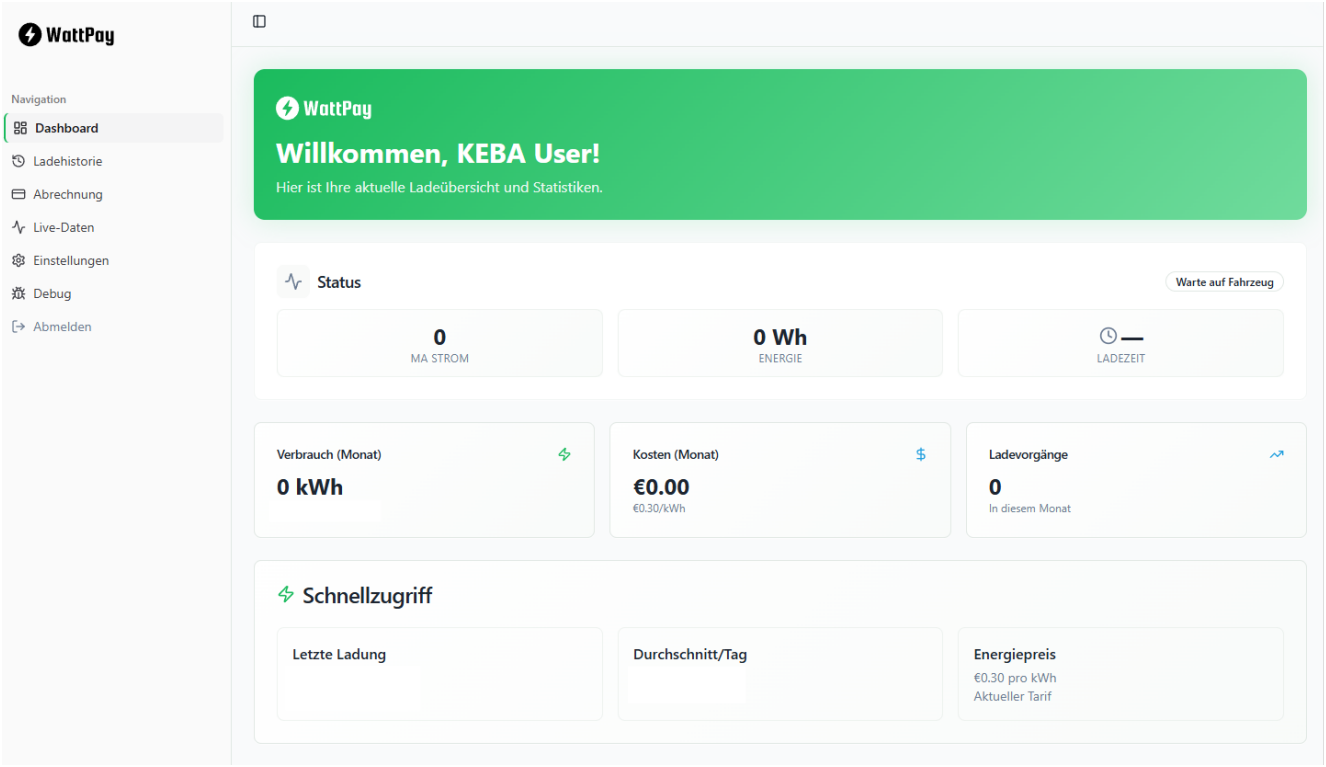

Abbildung 19: Startseite der Anwendung

Das Dashboard ist die Startseite der Anwendung und ist dafür da, um einen schnellen Überblick über die wichtigsten Daten zu bekommen.

Im oberen Bereich sieht man den aktuellen Status der Wallbox, zum Beispiel ob ein Fahrzeug geladen wird oder nicht. Darunter sieht man mehrere Karten mit Kennzahlen wie aktuellem Strom, Energie und Ladezeit.

Außerdem werden zusätzlich Monatswerte dargestellt, wie der gesamte Energieverbrauch, die Kosten und die Anzahl der Ladevorgänge. Im Bereich ”Schnellzugriff” sind weitere Informationen, wie die letzte Ladung, der durchschnittliche Verbrauch pro Tag und der aktuelle Energiepreis zusehen.

4.1.2 Ladehistorie

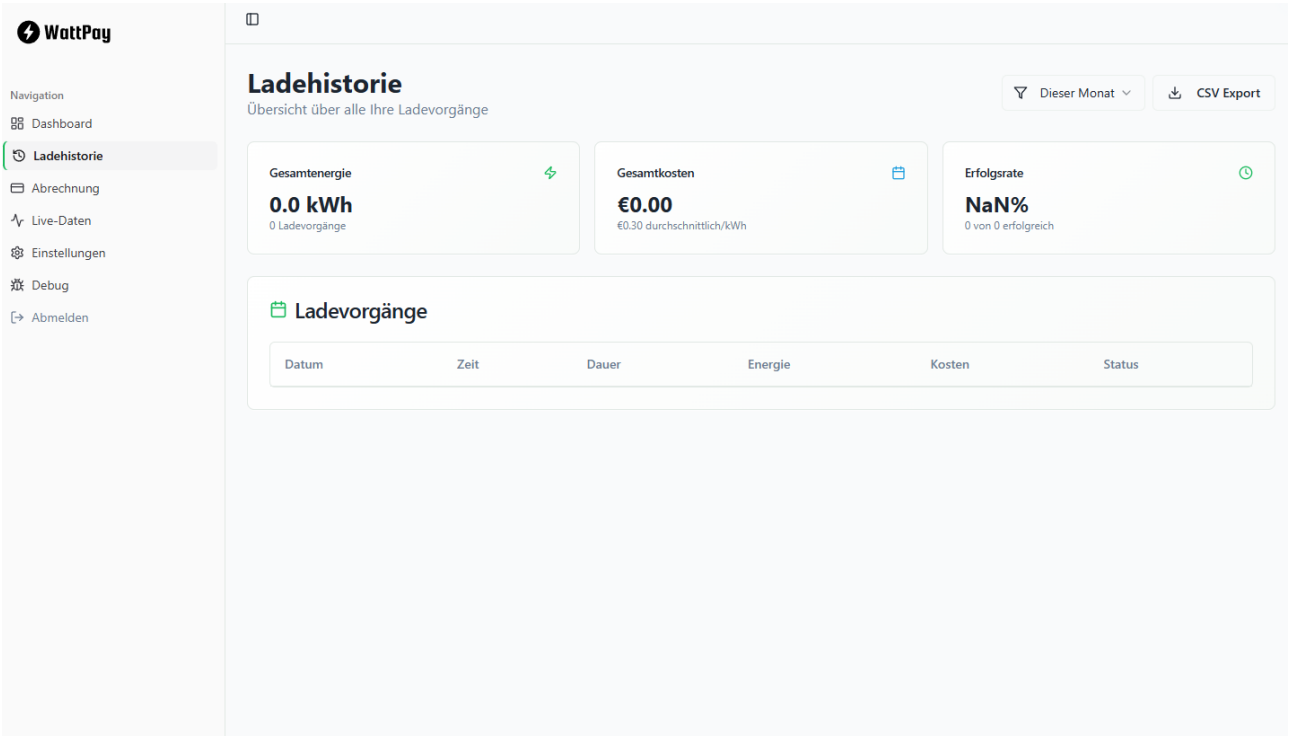

Abbildung 20: Bereich Ladehistorie

In der Ladehistorie werden alle vergangenen Ladevorgänge angezeigt. Die Daten werden in einer Tabelle dargestellt. Für jeden Ladevorgang werden Informationen wie Datum, Dauer, Energieverbrauch, Kosten und Status angezeigt.

Zusätzlich gibt es eine Übersicht über die gesamte geladene Energie und die entstandenen Kosten. Es besteht außerdem die Möglichkeit, die Daten zu filtern oder zu exportieren, zum Beispiel als CSV-Datei.

4.1.3 Abrechnung

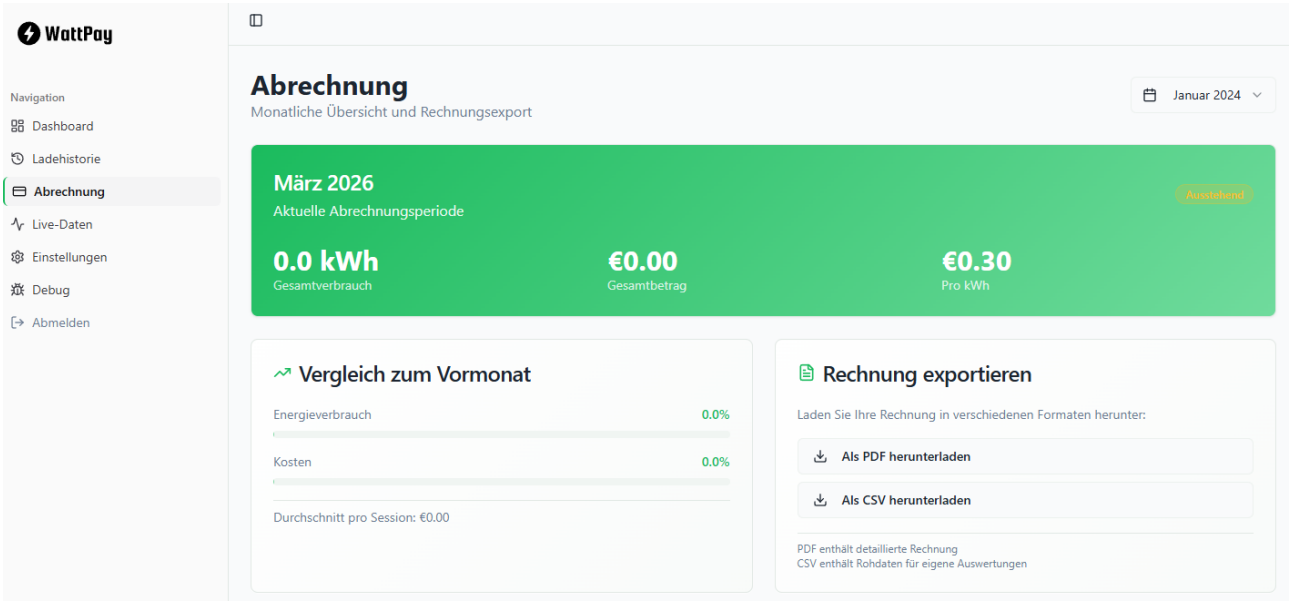

Abbildung 21: Bereich Abrechnung

Im Bereich Abrechnung sieht man eine Übersicht über die Kosten und den Energieverbrauch für einen bestimmten Zeitraum, zum Beispiel einen Monat. Im oberen Bereich wird die aktuelle Abrechnungsperiode angezeigt. Dazu gehören der Gesamtverbrauch, der Gesamtbetrag und der Preis pro Kilowattstunde.

Weiter unten kann man die aktuellen Werte mit dem Vormonat vergleichen, wodurch Veränderungen im Verbrauch und in den Kosten sichtbar werden. Rechnungen könnnen von Benutzern auch als PDF oder CSV-Datei exportiert werden.

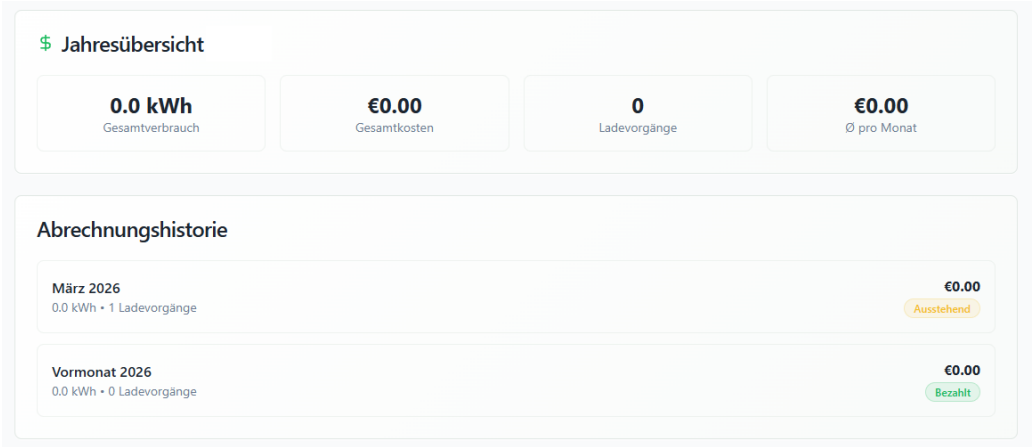

Abbildung 22: Bereich Abrechnung

Zusätzlich sieht man eine Abrechnungshistorie sowie eine Jahresübersicht, in der alte Rechnungen angezeigt werden.

4.1.4 Live-Daten

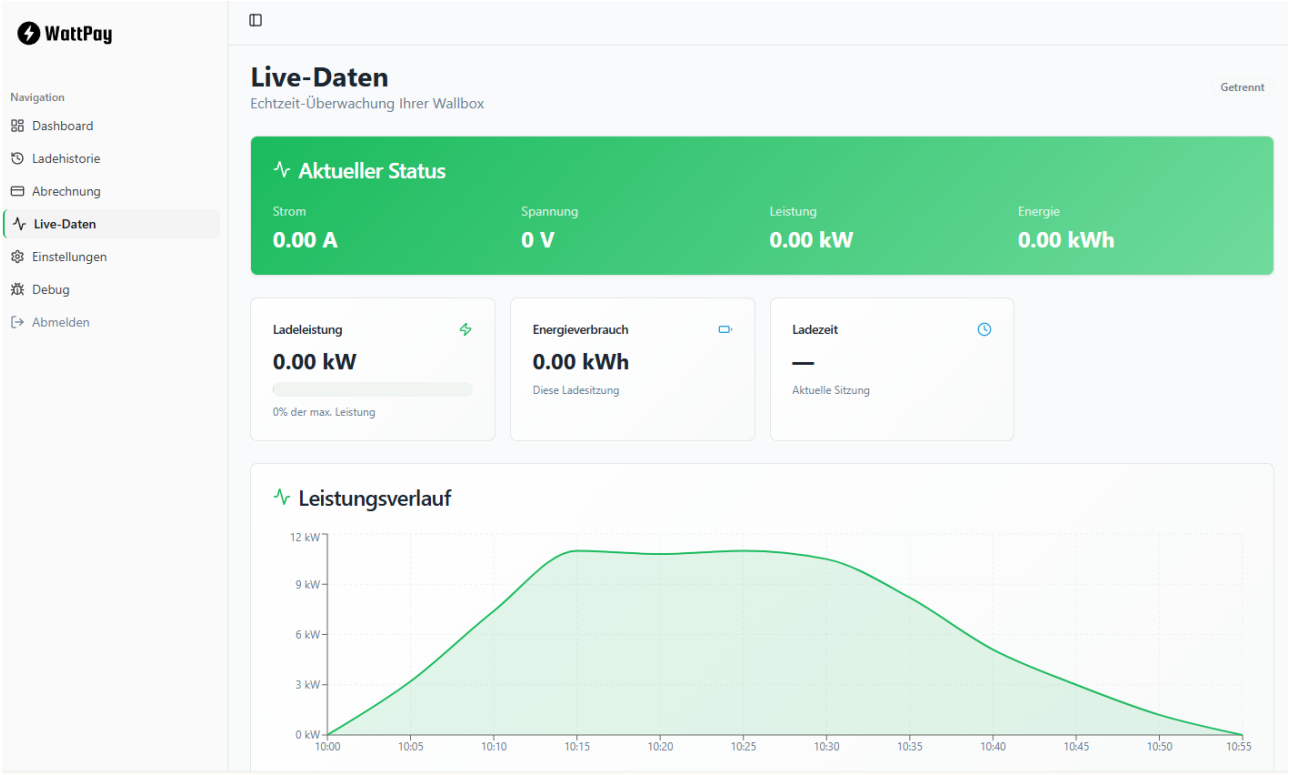

Abbildung 23: Bereich Live-Daten

Es werden auch die technischen Werte im Bereich ”Live-daten” angezeigt. Dazu gehören unter anderem Strom, Spannung, Leistung und Energie. Außerdem werden zusätzliche Informationen wie Ladeleistung, Energieverbrauch der aktuellen Sitzung und Ladezeit dargestellt.

Mithilfe eines Diagramms wird der Leistungsverlauf dargestellt, wodurch Änderungen eines Ladevorgangs zu sehen sind. Zusätzlich gibt es Informationen zu den einzelnen Phasen sowie allgemeine Informationen zur Wallbox.

4.1.5 Einstellungen

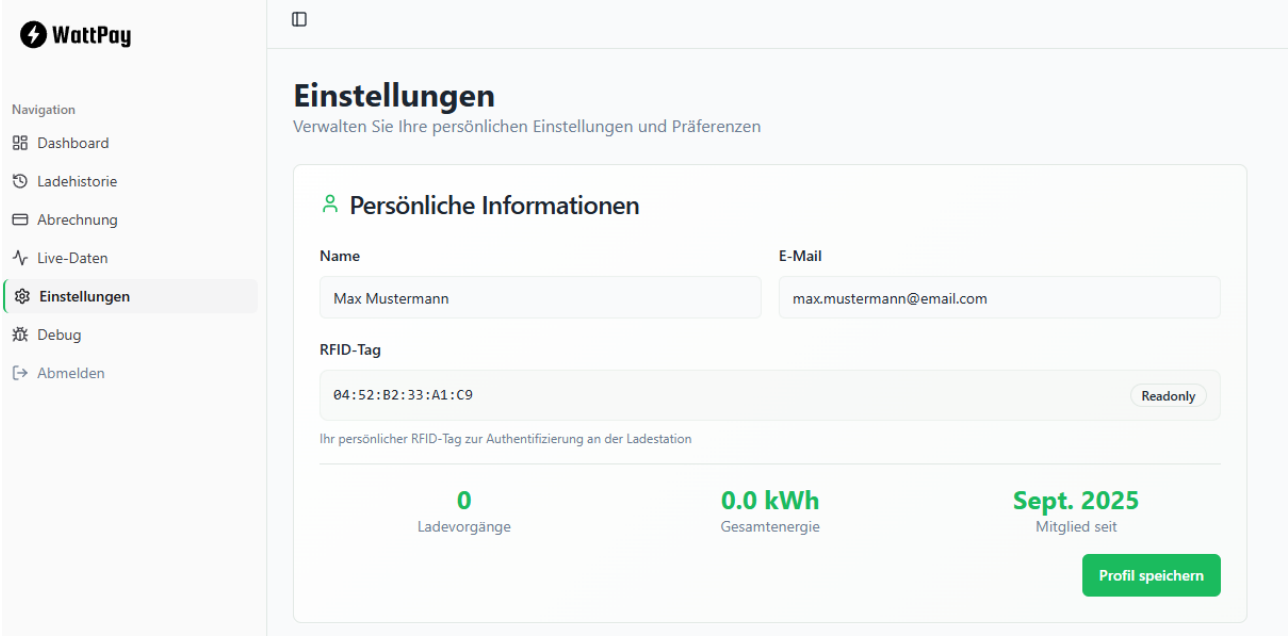

Abbildung 24: Bereich Einstellungen

In den Einstellungen kann der Benutzer persönliche und systembezogene Daten verwalten. Dazu gehören persönliche Informationen wie Name, E-Mail-Adresse und RFID-Tag.

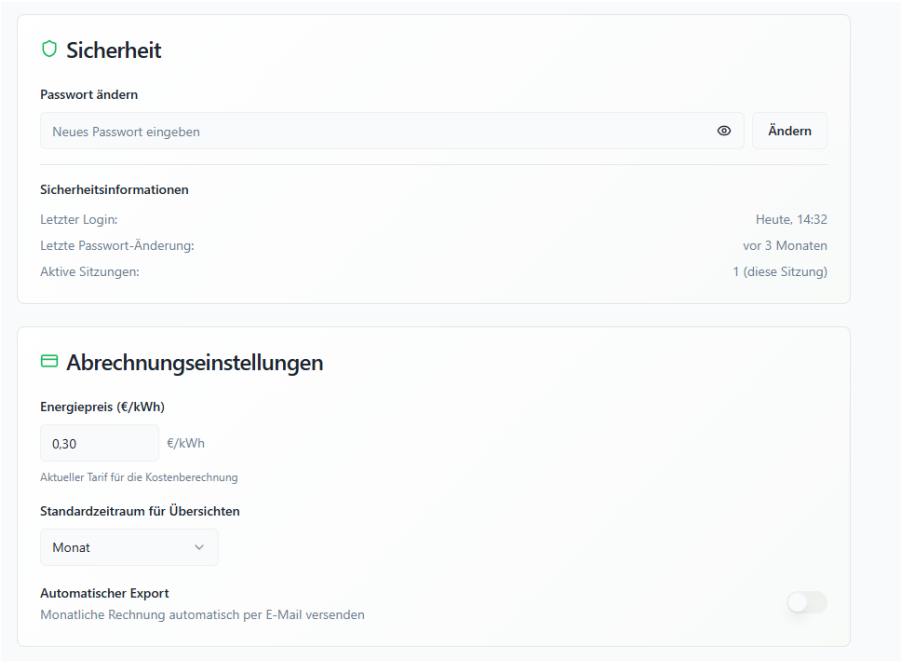

Abbildung 25: Bereich Einstellungen

Außerdem kann das Passwort geändert werden. Zusätzlich werden Sicherheitsinformationen wie letzter Login oder aktive Sitzungen angezeigt.

4.2 Design

Das Design wurde so gestaltet, sodass die wichtigsten Informationen gleich zu sehen und schnell erreichbar sind. Die Oberfläche verwendet ein Layout mit einer festen Navigation auf der linken Seite und dem Inhaltsbereich auf der rechten Seite

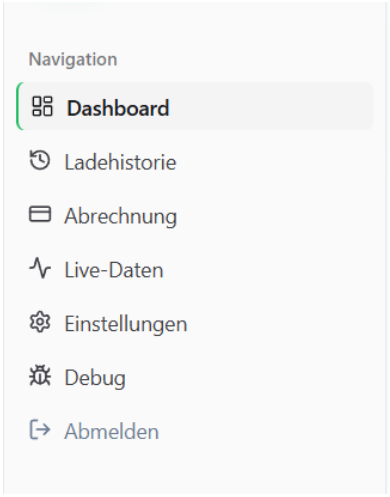

Abbildung 26: Navigationsleiste der Anwendung

Die Navigation erfolgt über eine Seitenleiste. Die einzelnen Menüpunkte sind klar benannt und mit passenden Symbolen versehen. Benutzer können damit zwischen verschiedenen Bereichen wechseln. Zu den wichtigsten Bereichen gehören das Dashboard, die Ladehistorie, die Abrechnung, die Live-Daten sowie die Einstellungen. Zusätzlich stehen auch Funktionen wie Debug oder Abmeldung zur Verfügung.

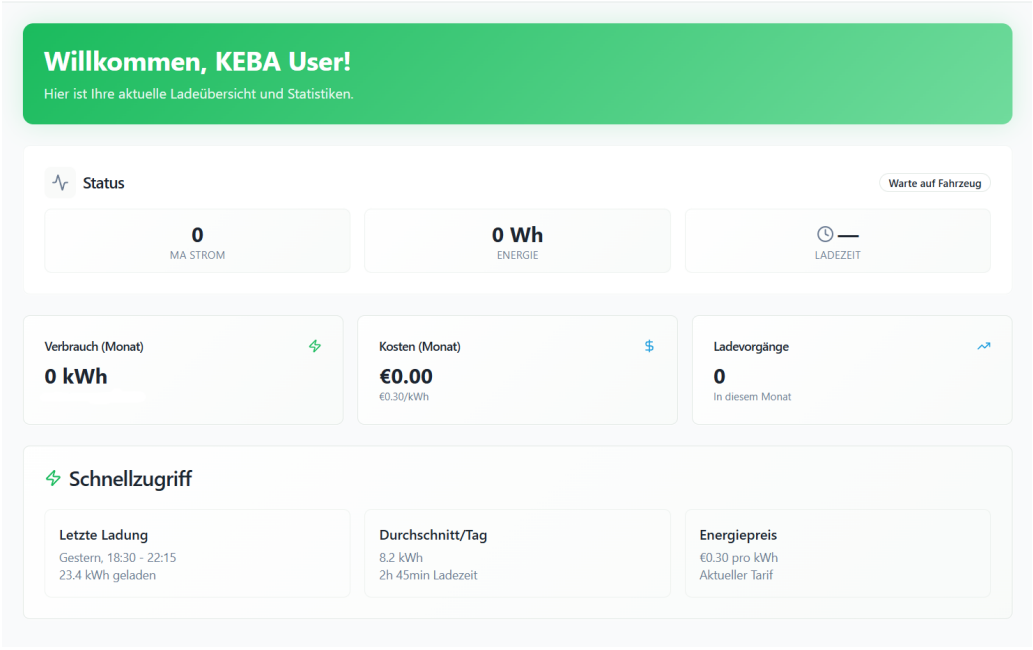

Abbildung 27: Dashboard der Anwendung

Im Hauptbereich der Anwendung werden die Inhalte in Form von Karten dargestellt. Diese sogenannten Cards enthalten jeweils eine bestimmte Information oder Kennzahl, zum Beispiel Energieverbrauch oder Kosten.

Für das Design wurde ein ruhiges Farbschema verwendet. Der Hintergrund ist überwiegend hell, während wichtige Informationen durch farbige Elemente hervorgehoben werden. Statusinformationen wie Warnungen oder Fehler werden durch deutlich sichtbare Farben dargestellt, damit sie sofort auffallen.

Abbildung 28: Leistungsverlauf einer Ladesession

Auch Diagramme und Visualisierungen werden verwendet, um Daten verständlicher darzustellen. Beispielsweise werden Verbrauchswerte oder Leistungsdaten in Diagrammform angezeigt. Dadurch können Benutzer Veränderungen im Zeitverlauf leichter erkennen.

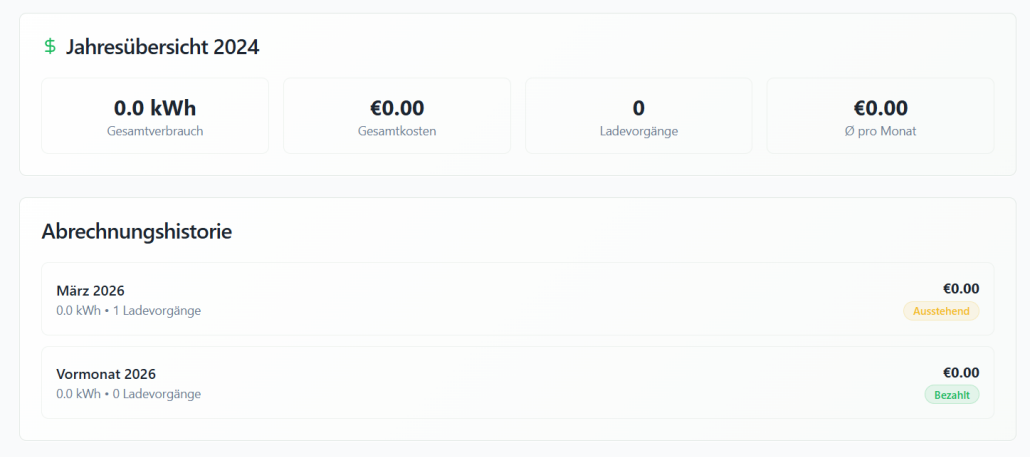

Abbildung 29: Gruppierung der Daten in eigenen Bereichen

Ein weiterer wichtiger Aspekt des Designs ist die klare Strukturierung der Informationen. Zusammengehörige Inhalte werden in eigenen Bereichen gruppiert. Dadurch bleibt die Oberfläche auch bei vielen Daten übersichtlich.

4.3 Rechnungen

In der Webanwendung gibt es auch eine Exportfunktion für Rechnungen. Dafür stehen zwei Formate zur Verfügung: PDF und CSV (Siehe Abbild 30). Damit können Ladevorgänge und die Kosten übersichtlich angezeigt werden.

Abbildung 30: Exportfunktion für die Rechnung

Die Rechnungen basieren auf gespeicherten Daten der Ladevorgänge. Für jeden Ladevorgang werden Informationen wie Datum, Start - und Endzeit, Dauer, geladene Energie, Kosten und Status angezeigt.

Abbildung 31: Rechnung im PDF Format

Die PDF-Datei zeigt die Rechnung in einer übersichtlichen Form. Sie enthält die wichtigsten Informtionen zum Benutzer, zur Ladestation und alle Ladevorgänge in einer Tabelle. Es wird auch der Gesamtbetrag angezeigt. (Siehe Abbildung 31)

	A	B	C	D	E	F	G	H
1	Datum	Startzeit	Endzeit	Dauer	Energie (kWh)	Kosten (€)	Status	
2	05.01.2026	08:15	12:30	4h 15min	32,5	9,75	Abgeschlossen	
3	09.01.2026	18:00	22:45	4h 45min	38,2	11,46	Abgeschlossen	
4	14.01.2026	09:00	11:30	2h 30min	18,7	5,61	Abgeschlossen	
5	18.01.2026	20:00	23:15	3h 15min	27,1	8,13	Abgeschlossen	
6	22.01.2026	07:30	10:00	2h 30min	22	6,6	Unterbrochen	
7	28.01.2026	14:00	18:30	4h 30min	35,8	10,74	Abgeschlossen	
8								
9	Gesamt				174,3	52,29		
10								

Abbildung 32: Rechnung im CSV Format

Die CSV-Datei beinhaltet die gleichen Daten in einer einfachen Tabellenform. Die Datei kann beispielsweise in Excel geöffnet werden. (Siehe Abbildung 32)

5 Resümee

5.1 Zielerreichung der Arbeit

Im Rahmen dieser Arbeit wurde eine Webanwendung zur Auswertung und Abrechnung von Ladevorgängen entwickelt. Ziel war es, Daten einer Wallbox zu erfassen, zu speichern und für den Benutzer übersichtlich darzustellen. Dieses Ziel konnte größtenteils erreicht werden.

Trotz der aufgetretenen Schwierigkeiten konnte eine funktionierende Webanwendung entwickelt werden. Die Anwendung ermöglicht es, Ladevorgänge anzuzeigen, Daten zu speichern und grundlegende Auswertungen durchzuführen.

5.2 Herausforderungen während der Umsetzung

Während der Umsetzung gab es einige technische Herausforderungen. Die Anzeige von Live-Daten hat sich als anspruchsvoll erwiesen, da die Daten nicht immer zuverlässig dargestellt werden konnten. Dies zeigte, dass die Verarbeitung von Echtzeitdaten, insbesondere bei externen Schnittstellen, komplex ist. Dabei konnten wichtige Erkenntnisse im Bereich der API-Kommunikation und des Umgangs mit asynchronen Daten gewonnen werden.

Auch bei der Umsetzung der Rechnungen und Tarife traten Probleme auf. Die korrekte Berechnung von Kosten sowie die Abbildung von Zeiträumen erwiesen sich als komplexer als zunächst angenommen. Diese Erfahrung verdeutlichte die Bedeutung einer gut geplanten Datenstruktur.

Zusätzlich bestand zu Beginn keine Erfahrung im Umgang mit Ladestationen. Es war notwendig, sich intensiv mit der Dokumentation auseinanderzusetzen, um die Schnittstellen und die Verarbeitung der Wallbox-Daten zu verstehen.

5.3 Projektorganisation und Arbeitsweise

Eine weitere Herausforderung bestand darin, dass das gesamte Projekt eigenständig geplant, umgesetzt und getestet wurde. Alle Schritte mussten selbst organisiert werden, von der Planung bis zur fertigen Anwendung. Dies erforderte eine strukturierte Arbeitsweise sowie einen hohen Zeitaufwand.

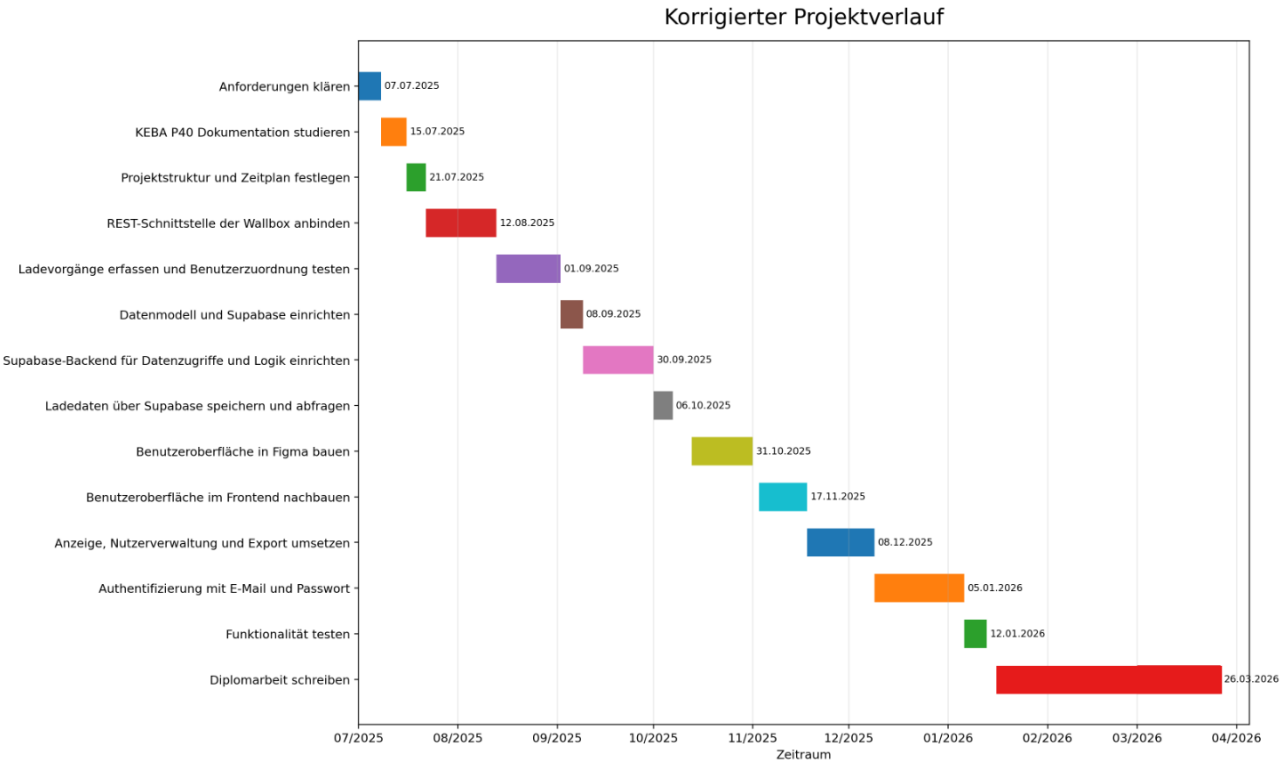

Abbildung 33: Projektverlauf

5.4 Nicht umgesetzte Funktionen

Die ursprünglich geplante Identifizierung über RFID-Karten wurde in der finalen Umsetzung nicht integriert. Dies lag vor allem daran, dass die Anbindung und Verarbeitung dieser Funktion zusätzliche Komplexität mit sich bringt. Dennoch konnte ein grundlegendes Verständnis für solche Identifizierungssysteme gewonnen werden, wodurch eine zukünftige Erweiterung möglich ist.

Auch die geplante Containerisierung der Anwendung wurde nicht umgesetzt, obwohl sie ursprünglich vorgesehen war. Durch die Verwendung von Supabase als Backend-Plattform wird jedoch ein Großteil der Infrastruktur automatisch verwaltet, wodurch eine eigene Containerisierung für zentrale Backend-Funktionen weniger notwendig ist.

5.5 Technische Erkenntnisse

Trotz der nicht umgesetzten Containerisierung konnte ein gutes Verständnis für Containerisierung, Deployment und Systemstrukturen entwickelt werden. Es wurde deutlich, welche Voraussetzungen dafür notwendig sind und in welchen Fällen ihr Einsatz sinnvoll ist. Diese Erkenntnisse bilden eine gute Grundlage für zukünftige Erweiterungen.

Der größte Gewinn dieser Arbeit war der Lernprozess. Es konnten viele praktische Erfahrungen in der Webentwicklung, bei der Arbeit mit APIs und bei der Nutzung von Datenbanken gesammelt werden. Außerdem wurde ein besseres Verständnis für die Herausforderungen bei Echtzeitdaten und Abrechnungssystemen gewonnen.

5.6 Gesamtbewertung und Ausblick

Abschließend kann gesagt werden, dass die Arbeit nicht nur zu einer funktionierenden Anwendung geführt hat, sondern auch die eigenen Fähigkeiten deutlich verbessert hat. Die gewonnenen Erfahrungen sind eine gute Grundlage für zukünftige Projekte.

Literaturverzeichnis

[1] Bundesministerium für Innovation, Mobilität und Infrastruktur, „Allgemeines zu Elektroautos und E-Mobilität," letzter Zugriff am 27.01.2026. Online verfügbar: https://www.oesterreich.gv.at/de/themen/mobilitaet/elektroautos_und_e_mobilitaet/Seite.4320010

[2] EnBW Energie Baden-Württemberg AG, „KEBA KeContact P40 Pro," letzter Zugriff am 27.01.2026. Online verfügbar: https://wallbox-shop.enbw.com/KEBA-KeContact-P40-Pro/SW10025.2

[3] KEBA Energy Automation GmbH, „KeContact P30 – Installationshandbuch," letzter Zugriff am 27.01.2026. Online verfügbar: https://www.keba.com/download/x/bf55ca39e4/kecontactp30_ihde.pdf

[4] ——, „KeContact P30 – Bedienungsanleitung," letzter Zugriff am 27.01.2026. Online verfügbar: https://www.keba.com/download/x/9a9d9fbcb0/kecontactp30_bdde.pdf

[5] Verbraucherzentrale, „RFID einfach erklärt: Wie funktioniert die kontaktlose Datenübertragung?" letzter Zugriff am 27.01.2026. Online verfügbar: https://www.verbraucherzentrale.de/wissen/digitale-welt/apps-und-software/rfid-einfach-erklaert-wie-funktioniert-die-kontaktlose-datenuebertragung-100809

[6] Martin Helmich, „RESTful Webservices (1): Was ist das überhaupt?" letzter Zugriff am 27.01.2026. Online verfügbar: https://www.mittwald.de/blog/webentwicklung-design/restful-webservices-1-was-ist-das-uberhaupt

[7] IONOS, „Was ist ein Backend?" letzter Zugriff am 29.01.2026. Online verfügbar: https://www.ionos.at/digitalguide/websites/webseiten-erstellen/was-ist-ein-backend/

[8] ——, „Was ist ein Frontend?" letzter Zugriff am 29.01.2026. Online verfügbar: https://www.ionos.at/digitalguide/websites/webseiten-erstellen/was-ist-ein-frontend/

[9] Oracle, „Was ist eine relationale Datenbank? (RDBMS)?" letzter Zugriff am 29.01.2026. Online verfügbar: https://www.oracle.com/de/database/what-is-a-relational-database/

[10] Wikimedia Common, „Npm logo," letzter Zugriff am 30.01.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Npm-logo.svg

[11] K. Luke, R. Michael, „About npm?" letzter Zugriff am 29.01.2026. Online verfügbar: https://docs.npmjs.com/about-npm

[12] GeeksforGeeks, „Introduction to Tailwind CSS," letzter Zugriff am 29.01.2026. Online verfügbar: https://www.geeksforgeeks.org/css/introduction-to-tailwind-css/

[13] ShadCn, „Introduction," letzter Zugriff am 29.01.2026. Online verfügbar: https://ui.shadcn.com/docs

[14] Wikimedia Common, „Typescript Logo," letzter Zugriff am 30.01.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Typescript_logo_2020.svg

[15] W3Schools, „TypeScript Introduction," letzter Zugriff am 29.01.2026. Online verfügbar: https://www.w3schools.com/typescript/typescript_intro.php

[16] Wikimedia Commons, „React Logo," letzter Zugriff am 30.01.2026. Online verfügbar: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7/React-icon.svg/3840px-React-icon.svg.png

[17] Meta Open Source, „Learn React Quick Start," letzter Zugriff am 29.01.2026. Online verfügbar: https://react.dev/learn

[18] Wikipedia, „Swagger Logo," letzter Zugriff am 30.01.2026. Online verfügbar: https://de.wikipedia.org/wiki/Datei:Swagger-logo.png

[19] SmartBear Software, „What is Swagger?" letzter Zugriff am 29.01.2026. Online verfügbar: https://swagger.io/docs/specification/v2_0/what-is-swagger/

[20] Wikimedia Commons, „PostgreSQL Logo," letzter Zugriff am 09.02.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Postgresql_elephant.svg

[21] PostgreSQL Global Development Group, „About PostgreSQL," letzter Zugriff am 09.02.2026. Online verfügbar: https://www.postgresql.org/about/

[22] GetLogo.Net., „Supabase Logo," letzter Zugriff am 10.03.2026. Online verfügbar: https://getlogo.net/supabase-logo-vector-svg/

[23] Wikimedia Commons, „Visual Studio Code," letzter Zugriff am 24.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg

[24] Microsoft, „Visual Studio Code Documentation," letzter Zugriff am 20.02.2026. Online verfügbar: https://code.visualstudio.com/docs

[25] UXWing, „Postman," letzter Zugriff am 24.03.2026. Online verfügbar: https://uxwing.com/postman-icon/

[26] Postman, „Postman documentation overview," letzter Zugriff am 20.02.2026. Online verfügbar: https://learning.postman.com/docs/introduction/overview

[27] Wikimedia Commons, „Figma Logo," letzter Zugriff am 24.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Figma-logo.svg

[28] Figma, „Was ist Figma?" letzter Zugriff am 20.02.2026. Online verfügbar: https://help.figma.com/hc/de/articles/14563969806359-Was-ist-Figma

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

Aufgabenbereich	**Verantwortlich**
Projektplanung und Organisation	Eigenständig durchgeführt
Analyse der Wallbox-Schnittstelle	Eigenständig durchgeführt
Backend-Entwicklung (REST-API)	Eigenständig durchgeführt
Datenbankdesign und Supabase-Integration	Eigenständig durchgeführt
Frontend-Entwicklung (Angular)	Eigenständig durchgeführt
Benutzeroberfläche (UI/UX Design)	Eigenständig durchgeführt
Authentifizierung (E-Mail/Passwort)	Eigenständig durchgeführt
Testing und Fehlerbehebung	Eigenständig durchgeführt
Dokumentation der Diplomarbeit	Eigenständig durchgeführt

Tabelle 1: Aufgabenverteilung im Projekt

B Meilensteine

B.1 Geplante Meilensteine

Meilenstein	**Datum**
Anforderungen geklärt	07.07.2025
KEBA P40 Dokumentation studiert	15.07.2025
Projektstruktur und Zeitplan festgelegt	21.07.2025
UDP-Schnittstelle implementiert	12.08.2025
Ladevorgänge erfassen und RFID-Zuordnung getestet	01.09.2025
Datenmodell erstellt	08.09.2025
REST-API programmiert	30.09.2025
Speicherung und Abfrage der Ladedaten implementiert	06.10.2025
Benutzeroberfläche mit Figma gebaut	31.10.2025
Benutzeroberfläche mit Angular nachgebaut	17.11.2025
Anzeige von Ladevorgängen, Nutzerverwaltung, Exportfunktionen implementiert	08.12.2025
Konzeption und ggf. erste Implementierung der PV-Integration fertiggestellt	15.12.2025
Docker-Container für Backend, Frontend, DB erstellt	22.12.2025
Docker Compose eingerichtet	05.01.2026
Funktionalität getestet	12.01.2026
Diplomarbeit geschrieben	06.02.2026

Tabelle 2: Geplante Projektmeilensteine

B.2 Tatsächlich umgesetzte Meilensteine

Meilenstein	**Datum**
Anforderungen geklärt	07.07.2025
KEBA P40 Dokumentation studiert	15.07.2025
Projektstruktur und Zeitplan festgelegt	21.07.2025
REST-Schnittstelle der Wallbox analysiert und angebunden	12.08.2025
Ladevorgänge erfassen und Benutzerzuordnung getestet	01.09.2025
Datenmodell und Supabase-Datenbank eingerichtet	08.09.2025
REST-API für Backend implementiert	30.09.2025
Speicherung und Abfrage der Ladedaten über Supabase realisiert	06.10.2025
Benutzeroberfläche mit Figma gebaut	31.10.2025
Benutzeroberfläche im Code nachgebaut	17.11.2025
Anzeige von Ladevorgängen, Nutzerverwaltung, Exportfunktionen implementiert	08.12.2025
Authentifizierung mit E-Mail und Passwort implementiert	05.01.2026
Funktionalität getestet	12.01.2026
Diplomarbeit geschrieben	26.03.2026

Tabelle 3: umgesetzte Meilensteine Projektmeilensteine

C Diplomarbeitsplakat