HAW HAMBURG

BACHELORTHESIS
Henning Brockmann

# Entwicklung eines Algorithmus zur Reduktion von Lastspitzen bei Ladevorgängen von Elektrofahrzeugen auf Basis einer empirischen Analyse

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Henning Brockmann

# Entwicklung eines Algorithmus zur Reduktion von Lastspitzen bei Ladevorgängen von Elektrofahrzeugen auf Basis einer empirischen Analyse

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 20. März 2022

**Henning Brockmann**

**Thema der Arbeit**

Entwicklung eines Algorithmus zur Reduktion von Lastspitzen bei Ladevorgängen von Elektrofahrzeugen auf Basis einer empirischen Analyse

**Stichworte**

Elektromobilität, Lastspitze, Optimierung, gemischte-ganzzahlige Optimierungsprogramm

**Kurzzusammenfassung**

Das Ziel der vorliegenden Bachelorabeit war eine Software zu entwickeln die es Betriebshöfen für Elektrofahrzeuge ermöglicht ihre Lastspitze zu senken. Es wird der Verlauf der Entwicklung, von der Untersuchung des fachlichen Umfeldes, über die Ermittlung der Anforderung, bis hin zur Implementation und Evaluation gezeigt. Die geleistet Arbeit zeigt wie groß das monetäre Potenziell der Lastspitzenreduktion ist und wie es sich nutzen lässt.

**Henning Brockmann**

**Title of Thesis**

Development of an algorithm for the reduction of peak loads during charging processes of electric vehicles based on an empirical analysis

**Keywords**

electromobility, peak load, optimization, mixed integer programming

**Abstract**

The aim of this bachelor thesis was to develop a software that enables depots for electric vehicles to reduce their peak loads. It shows the journey, from investigating the technical environment, over the determination of the requirement, up to the implementation and evaluation. The achieved work shows how large the monetary potential of the load peak reduction is and how it can be used.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Akronyme

**API** Application Programming Interface.

**DOM** Document Object Model.

**FHH** Freie und Hansestadt Hamburg.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**JPA** Java Persistence API.

**JSON** JavaScript Object Notation.

**JSX** Java Serialization to XML.

**LAN** Local Area Network.

**REST** Representational State Transfer.

**RFID** Radio Frequency Identification.

**SIM** Subscriber Identity Module.

**SNH** Stromnetz Hamburg GmbH.

**UML** Unified Modeling Language.

**URI** Uniform Resource Identifier.

**YAML** YAML Ain't Markup Language.

# 1 Einleitung

Die Nutzung von erneuerbaren Energien ist ein zentrales Thema unserer Zeit. Um die Abhängigkeit von fossilen Brennstoffen zu minimieren ist die Elektrifizierung des Antriebs ein wichtiger Schritt. Mit dem Wandel entstehen Herausforderungen die gelöst werden müssen um den gewünschten Fortschritt zu ermöglichen. Diese Arbeit widmet sich der Algorithmen-gestützten-Optimierung des Ladeverhaltens von Elektrofahrzeugen.

## 1.1 Problemstellung und Motivation

### 1.1.1 Betriebliches Umfeld

Diese Arbeit entsteht im betrieblichen Umfeld der Stromnetz Hamburg GmbH (SNH). Die SNH ist Eigentümer und Betreiber des Stromverteilungsnetzes sowie grundzuständiger Messstellenbetreiber im Konzessionsgebiet der Freie und Hansestadt Hamburg (FHH). Im Jahr 2022 sind insgesamt 1300 Mitarbeiter (vgl. Stromnetz Hamburg GmbH (2022)) angestellt. Das Unternehmen befindet sich zu 100 Prozent im Besitz der FHH und ist ökologischen, energie- sowie umweltpolitischen Zielen der Stadt verpflichtet. Der 2014 beschlossene Masterplan zur öffentlichen Ladeinfrastruktur führte zu der Entwicklung einer Anwendung, die bundesweit den Betrieb von Ladestationen unterstützt. Dieses Produkt wird unter dem Namen eRound vermarktet und vertrieben.

### 1.1.2 Betriebshöfe für elektrifizierte Fahrzeugflotten

Mit dem Voranschreiten der Energiewende werden die Betriebshöfe der Fahrzeugflotten elektrifiziert. Sie müssen sicherstellen, dass jedes elektrische Fahrzeug in der Zeit, die es im Betriebshof verbringt, die Möglichkeit hat die Batterie auf einen Füllstand zu bringen, sodass es die bevorstehenden Aufgaben erledigen kann. Die Kostenstruktur wandelt sich.

Die Kosten die normalerweise auf fossile Brennstoffe entfallen würden, finden sich nun auf der Abrechnung des Stromanbieters wieder.

### 1.1.3 Stromkosten durch den Lastgang

Der Strommarkt und die Strompreise sind sehr volatil. Sowohl Angebot als auch Nachfrage schwanken ständig. Ein Stromanbieter gibt diese Kosten an seine Kunden weiter. Dabei ist der Lastgang ein wichtiges Instrument. Der Lastgang beschreibt, wann und wie viel Strom ein Kunde verbraucht. Die Kosten pro verbrauchter Kilo Watt Stunde (kWh) hängen vom Profil des Kunden ab. Entscheidende Faktoren sind zu welchem Zeitpunkt Strom verbraucht wird und wie hoch die Lastspitze ist. Dieser Sachverhalt bietet ein Optimierungspotential.

### 1.1.4 Motivation

Durch diese Arbeit kann das Ladeverhalten auf einem Betriebshof so manipuliert werden, dass der Lastgang optimiert wird. Bei gleicher abgenommener Strommenge hätte dies eine Senkung der Stromkosten zur Folge.

> „Eine Reduzierung der Leistungsspitzen hat positive Auswirkungen auf den Netto-Energiepreis, die Netznutzungsentgelte und indirekt auf die Steuern, Umlagen und Abgaben. Sofern vom Betriebsablauf möglich, kann das Kundenunternehmen Änderungen vornehmen, um den Lastgang zu optimieren.“ (Schumacher (2015), Seite 48)

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist die empirische Aufarbeitung der Themen Elektrofahrzeuge, Strommarkt und mathematische Optimierung, sowie die Planung und Entwicklung eines Prototypen einer Applikation die zur Reduktion der Lastspitzen eingesetzt werden kann.

## 1.3 Überblick über die Kapitel der Arbeit

Die Arbeit besteht aus 8 Kapiteln - beginnend mit einer Einleitung. In Kapitel 2 werden wissenschaftliche Grundlagen empirisch aufgearbeitet. Gefolgt von Kapitel 3, indem die aktuelle Situation auf einem Betriebshofs, als auch der aktuelle Aufbau der bereits bestehende Software dargestellt wird. In Kapitel 4 werden mithilfe des Requirements Engeneering die Anforderungen an den Prototypen ausgearbeitet. Im darauf folgenden Kapitel, entsteht aus den Anforderungen ein Konzept, welches auf die Infrastruktur, die Architektur, die Kommunikation, die Datensicherung und die Optimierung eingeht. In Kapitel 6 werden die einzelnen Facetten der Implementation beleuchtet. Bevor diese in Kapitel 7 evaluiert werden. Abgerundet wird die Arbeit im letzten Kapitel mit einem persönlichen Fazit und einem Ausblick auf die Möglichkeiten die diese Arbeit eröffnet hat.

# 2 Grundlagen

Für die Umsetzung dieser Arbeit ist es nötig einige fachliche Grundlagen zu schaffen. Diese werden in diesem Kapitel geschaffen.

## 2.1 Lastspitzenkappung

Bei der Lastspitzenkappung ist das erklärte Ziel, die Lastspitze zu minimieren. Die grundsätzliche Idee ist es, durch eine gleichmäßige Belastung des Stromnetzes, Netzentgelte (Abschnitt 2.2) zu sparen (Kistner (2020)).

Die in Abschnitt 2.5 beschriebene Mobilitätswende steigert die Nachfrage nach elektrischer Energie dramatisch. Damit die Nachfrage nach elektrischer Leistung nicht im gleichen Maße ansteigt, ist eine gleichmäßige Energieabgabe wichtig. Dieses Ziel, der Lastspitzenkappung, steht im direkten Widerspruch mit dem Wunsch des Fahrzeughalters möglichst schnell den Ladezustand der Batterie zu erhöhen. Die Lösung dieses Problems ist ein Kernthema der Mobilitätswende. Ein weit verbreiteter Ansatz ist die Nutzung von stationären Batterie-Systemen (Charing Booster). Die Funktionsweise ähnelt der einer Toiletten-Spülung. Das Reservoir wird gleichmäßig befüllt. Bei Bedarf wird die gesamte Ladung in kurzer Zeit freigegeben werden (Kistner (2020)).

## 2.2 Netzentgeltsystematik

Die Netzentgeltsystematik beschreibt eine Regelung zur Abdeckung der Kosten für die Übertragung und Verteilung des Stroms. Die Kosten für den Betrieb, Erhalt und Ausbau des deutschen Stromnetzes betrugen 2015 rund 22 Milliarden Euro. Die erwartete Entwicklung ist eine Steigerung um 23 Prozent bis 2030. Besonders die zunehmende Nutzung von Wärmepumpen und Elektrofahrzeugen sorgen für eine Erhöhung der maximalen Netzauslastung und des daraus resultierenden Netzausbaubedarfs. Netzentgelte

setzen Anreize für das Verhalten von Netznutzern, um den Netzausbaubedarf zu bremsen (Jeddi und Sitzmann (2019)).

Die Kosten fallen auf sämtlichen Ebenen (Hochspannung / Niederspannung) des Stromnetzes an. Die Berechnung der Kosten erfolgt von oben nach unten für jede einzelne Netz- bzw. Umspannungsebene mithilfe einer Briefmarke. Bei der Briefmarke handelt es sich um den Quotienten der Gesamtkosten und der Jahreshöchstlast der Ebene. Die Briefmarke wird anschließend anhand der Gleichzeitigkeitsfunktion (G-Funktion) in die jeweiligen Netzentgeltkomponenten überführt. Da die Zuordnung der Netzkosten auf die Netznutzer nach dem Verursacherprinzip folgen soll, tragen Netznutzer, die einen höheren Beitrag zu der Jahreshöchstlast der Netzebene leisten, dementsprechend einen höheren Anteil der Kosten (Jeddi und Sitzmann (2019)).

Einfache Haushalte zahlen einen Grundpreis (€/ Monat) sowie einen Arbeitspreis (€/ kWh). Netznutzer mit einem höheren Verbrauch können sich für eine Leistungsmessung (Abschnitt 2.3) entscheiden. Ab einem Jahresverbrauch von 100 MWh wird eine Leistungsmessung zur Pflicht. Nutzer mit einer Leistungsmessung zahlen zusätzlich zum Arbeitspreis einen Leistungspreis (€/ kW). Für Industrieunternehmen gibt es Sonderformen der Netznutzung, die es ermöglichen mit den zuständigen Netzbetreibern individuelle Netzentgelte zu vereinbaren (§ 19 Abs. 2 StromNEV). Eine weitere Möglichkeit der Kostensenkung bieten steuerbare Einrichtungen in der Niederspannung (§ 14a EnWG). Bisher wird diese Möglichkeit vor allem von Einrichtungen zur elektrischen Wärmeversorgung genutzt (Jeddi und Sitzmann (2019)).

In der Tabelle 2.1 werden die Preise für Leistungsgemessene Kunden für das Jahr 2022 aufgeschlüsselt. Besondere Relevanz hat die letzte Zeile. Jedes Kilowatt, um das die Lastspitze gesenkt werden kann, bringt dem Kunden eine Ersparnis von ungefähr 96 Euro. Bei Kunden dessen Lastspitzen mehrere Megawatt messen ist das Potential der Einsparung sehr hoch.

### 2.2.1 Aktuelle Kritik

Durch den Ausbau der erneuerbaren Energien entstehen zunehmend volatile Einspeisungen ins Stromnetz. Daraus resultieren Situationen, in denen Strom am Markt zu negativen oder deutlich überdurchschnittlichen Preisen gehandelt wird. Mit der aktuellen Netzentgeltsystematik gibt es keine Möglichkeit diesem Phänomen entgegenzuwirken bzw. Anreize zu schaffen in diesen Situationen besonders viel Strom zu laden. Aktuell

| **Netz- oder Umspannebene (Jahresbenutzungsdauer)** | **Leistungspreis (€/kW/Jahr)** | **Arbeitspreis (ct/kWh)** |
|---|---|---|
| Hochspannung ($<$ 2500h/Jahr) | 9,22 | 2,37 |
| Hochspannung ($\geq$ 2500h/Jahr) | 59,05 | 0,38 |
| Hochspannung mit Umspannung auf MS ($<$ 2500h/Jahr) | 11,50 | 2,96 |
| Hochspannung mit Umspannung auf MS ($\geq$ 2500h/Jahr) | 73,67 | 0,47 |
| Mittelspannung ($<$ 2500h/Jahr) | 13,73 | 2,92 |
| Mittelspannung ($\geq$ 2500h/Jahr) | 66,62 | 0,81 |
| Mittelspannung mit Umspannung auf NS ($<$ 2500h/Jahr) | 14,77 | 3,72 |
| Mittelspannung mit Umspannung auf NS ($\geq$ 2500h/Jahr) | 91,88 | 0,64 |
| Niederspannung ($<$ 2500h/Jahr) | 18,68 | 4,15 |
| Niederspannung ($\geq$ 2500h/Jahr) | 96,57 | 1,03 |

Tabelle 2.1: Netzentgelte für leistungsgemessene Kunden mit Lastgangzähler 2022 (Quelle: Rheinische NETZGesellschaft mbH (2022))

kann ein Abnehmer sogar bestraft werden, da er Gefahr läuft, dass sich der zu zahlende Leistungspreis erhöht (Jeddi und Sitzmann (2019)).

Die Rolle der Haushalte steigt durch die vermehrte Nutzung von Wärmepumpen und der Elektromobilität. Jedoch werden aktuell noch keine Anreize geschaffen um ein netzdienliches Verhalten zu fördern.

## 2.3 Registrierende Leistungsmessung

Bei einem Verbrauch von mehr als 100 MWh im Jahr bedarf es einer registrierenden Leistungsmessung (RLM). Das Resultat einer solchen Messung ist der Lastgang. Für den Lastgang wird die bezogene Leistung in 15 Minuten Abschnitte zusammengefasst. Die gemessenen Werte werden täglich an den Netzbetreiber und Stromlieferanten übermittelt. Anhand des Lastgangs kann der Stromlieferant evaluieren, wie er seinen Einkauf optimieren kann. Für die Bepreisung gegenüber dem Kunden werden daher Lastgänge aus der Vergangenheit verwendet. Findet keine registrierende Leistungsmessung statt werden Standartlastprofile zur Berechnung genutzt. Dabei hält der Stromversorger mehrere Lastprofile (Tabelle 2.2) für unterschiedliche Kundengruppen vor (Schumacher (2015)).

| | |
|---|---|
| G0 | allgemeines Gewerbe (Mittelwert der Profile G1–G6) |
| G1 | Gewerbe zwischen 8 und 18 Uhr (z. B. Anwaltskanzlei, Verwaltungseinrichtungen) |
| G2 | Gewerbe mit hohem Abendverbrauch (z. B. Gastronomie, Fitnessstudio) |
| G3 | Durchlaufendes Gewerbe (z. B. Betrieb von Kühlhäusern, Serverdienstleistungen) |
| G4 | Ladengeschäft (z. B. Einzelhandel, Friseur) |
| G5 | Bäckerei mit integrierter Backstube |
| G6 | Wochenendbetriebe (z. B. Kinos, Discos) |
| G7 | Sendestationen Mobilfunk |
| L0 | Allgemeine landwirtschaftliche Betriebe (Mittelwert L1 und L2) |
| L1 | Landwirtschaftliche Betriebe mit Viehzucht oder Milchwirtschaft |
| L2 | Übrige landwirtschaftliche Betriebe |
| H0 | Haushaltsprofil (Privathaushalte) |

Tabelle 2.2: Standardlastprofile gemäß BDEW (Schumacher (2015))

## 2.4 Benutzungsstunden

Anders als der Term vermuten lässt, verbirgt sich hinter dem Begriff Benutzungsstunden nicht die Zeit die ein Kunde Strom bezieht. Es ist ein Wert der widerspiegeln soll, wie kontinuierlich der Strom bezogen wird - je höher der Wert desto gleichmäßiger der Bezug. Die Formel kann Abbildung 2.1 entnommen werden. Zur Berechnung werden der Jahresverbrauch und die maximale Leistungsspitze genutzt. Ein hoher Wert sorgt in der Regel für einen günstigeren Energiepreis (Schumacher (2015)).

$$\text{Benutzungsstunden} = \frac{\text{Jahresverbrauch}}{\text{Leistungsspitze}}$$

Abbildung 2.1: Berechnung der Benutzungsstunden

## 2.5 Mobilitätswende

Die Mobilitätswende beschreibt die Elektrifizierung des Antriebsstranges, die voranschreitende Vernetzung, Automatisierung und die dadurch entstehenden Herausforderungen. Das Ziel ist eine effizientere, ökologische und intelligente Mobilität. Dabei geht es um den Klimawandel, die Verkehrssicherheit, die soziale Teilhabe und der Sicherung der Innovationsfähigkeit des Wirtschaftsstandorts Deutschland. Für die erfolgreiche Umsetzung ist es notwendig, dass die Bedürfnisse der Verbraucher berücksichtigt werden (Kagermann (2017)).

## 2.6 Gemischt-ganzzahlige Optimierung

Gemischt-ganzzahlige Optimierung, auch unter dem aus dem englischen stammenden Akronym MIP (Mixed Integer Programming) bekannt, beschäftigt sich mit der Optimierung einer linearen Zielfunktion. Die Lösung der Optimierung wird dabei von linearen Gleichungen und Ungleichungen eingeschränkt. Bei der Lösung wird die Zielfunktion, unter Berücksichtigung der Restriktionen, maximiert oder minimiert. In der Komplexitätstheorie werden diese Probleme als NP-schwer verordnet. In der Praxis wird gemischt-ganzzahlige Optimierung zur Planung von Produktion, Ressourcen, Verkehr oder Touren genutzt. Verglichen mit linearen Optimierungsmodellen sind gemischt-ganzzahlige Modelle sehr schwer zu lösen. Grund hierfür ist der exponentielle Anstieg möglicher Wertekombinationen. Bereits kleine Modelle erreichen dabei die Grenzen des durch Computer in vertretbarer Zeit Lösbaren. Um dem entgegen zu wirken, gibt es spezielle Suchstrategien. Solche Strategien können eine Verkürzung der Rechenzeit bewirken, tun dies aber nicht zwangsläufig (Suhl und Mellouli (2013)).

## 2.7 OCPP

OCPP ist die Abkürtung für Open Charge Point Protocoll[1]. Das Protokoll wird von der E-Laad-Stiftung[2] aus den Niederlanden entwickelt und ist der vorherschende Standart in Europa und Asien, wenn es um die Kommunikation mit einer elektrischen Ladesäule geht. Die aktuell verwendeten Versionen des Protokolls sind veraltete Version 1.5 und der gängige Standard 1.6. In Zukunft soll die Version 2.0 der vorherrschende Standard werden (Vaidya und Mouftah (2018)). Technisch basiert die Kommunikation ab 1.6 auf einer Websocket Verbindung über die Daten im JavaScript Object Notation (JSON) Format gesendet werden. Im Idealfall herrscht eine ständige Verbindung zum Backend. Diese wird meist durch eine Subscriber Identity Module (SIM)-Karte und dem mobilen Datennetz oder einem Anschluss per Local Area Network (LAN) gewährleistet.

[1] https://www.openchargealliance.org/
[2] https://www.elaad.nl/wat-is-ocpp/

# 3 Analyse der aktuellen Situation

Um zu verstehen, welches Problem gelöst werden soll, ist es von signifikanter Bedeutung die aktuelle Situation zu analysieren.

## 3.1 Aufbau und Ablauf auf einem Betriebshof

Der Ursprung dieses Projekts entstand durch die Zusammenarbeit mit einem Betriebshof. Die Ideen und die Umsetzung fußen zu großen Stücken auf den Gegebenheiten die auf diesem Betriebshof herrschen. Dieser lässt sich wie folgt quantifizieren:
Auf dem Betriebshof stehen ca. 100 Ladesäulen die jeweils eine maximale Leistung von 72 kW haben. Zu diesem Betriebshof gehören aktuell ca. 70 Linienbusse. Der Ab- und Zulauf der Busse ist bedarfsabhängig und nicht gleichmäßig. Grundsätzlich kann das Verhalten so beschrieben werden, dass die Busse am Morgen den Hof verlassen und gegen Abend wieder eintreffen um ihre Batterie zu laden. Die Busse haben eine Ladegeschwindigkeit von bis zu 150 kW. Es gibt bereits Möglichkeiten die Ladegeschwindigkeit zu regulieren. Diese sind statischer Natur und unterliegen dem Grundsatz, dass im Zweifel mit maximal möglicher Geschwindigkeit geladen wird, bis die Batterie komplett geladen wurde.

## 3.2 eRound

Bei eRound handelt es sich um eine von der Stromnetz Hamburg GmbH entwickelte Plattform. Über die Plattform wird die Kommunikation mit den unterschiedlichen Partnern geregelt die an dem Ladevorgang eines Elektrofahrzeugs beteiligt werden.

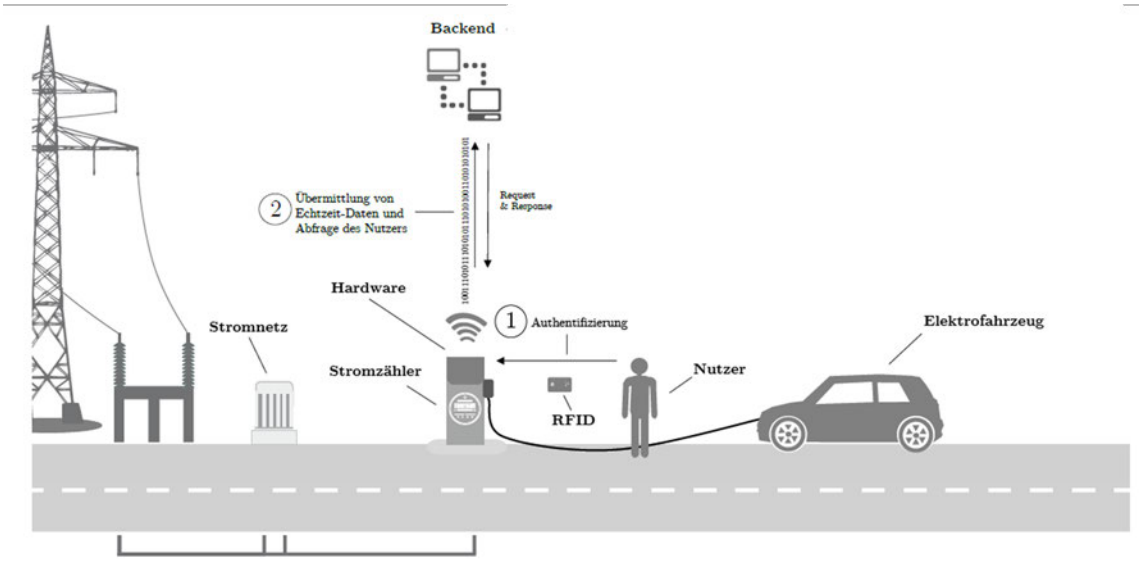


Abbildung 3.1: Grafische Darstellung des Ladevorgangs

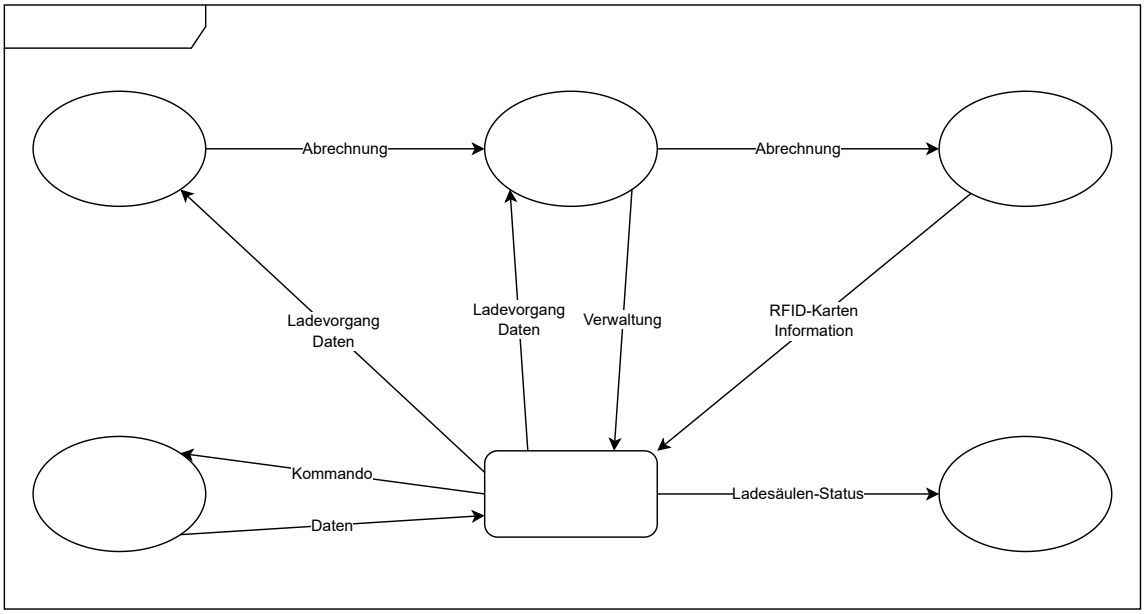


Abbildung 3.2: Technisches Umfeld der Anwendung eRound

### 3.2.1 Technisches Umfeld

Das technische Umfeld von eRound kann der Abbildung 3.2 entnommen werden. Die Identifikation an der Ladestation (Schritt 1 in Abbildung 3.1) erfolgt typischerweise mithilfe einer Radio Frequency Identification (RFID)-Karte. Damit diese akzeptiert wird, muss sie im System hinterlegt werden. Hält der Kunde die RFID-Karte vor die Station sendet diese eine Anfrage an eRound (Schritt 2 in Abbildung 3.1). Gibt eRound die RFID-Karte frei, kann der Ladevorgang gestartet werden. Sobald der Ladevorgang beendet wird, werden die Daten zusammengetragen und an den zuständigen Stromlieferanten und den Ladestationbetreiber gesendet. Der Stromlieferant sendet eine Abrechnung an den Betreiber. Der Betreiber sendet wiederum eine Abrechnung an den Mobilitätsanbieter.

Der Betreiber kann mithilfe der Weboberfläche seine Ladestationen verwalten und ansteuern. Sollte der Betreiber es erlauben, werden aktuelle Informationen publiziert und Navigationsdiensten zur Verfügung gestellt.

Die Person die an der Ladestation eine RFID-Karte vor das Lesegerät hält steht meistens in einem vertraglichen Verhältnis mit dem Mobilitätsanbieter. Mit Abschluss eines Vertrags werden die Konditionen, zu denen geladen werden kann, festgelegt und eine RFID-Karte bereitgestellt.

### 3.2.2 Architektur

Die Architektur der Anwendung kann der Abbildung 3.3 entnommen werden. Die Nutzer des Portals und Navigationssysteme greifen über das Internet auf die Anwendung zu. Die Ladestationen haben in der Regel einen eigenen Tunnel über den sie in das Netzwerk der Anwendung gelangen. Vor dem Eintreffen beim entsprechenden Service durchläuft jede Anfrage Firewall und Proxy. Da alle zentralen Services redundant laufen, wird an dieser Stelle auch die Verteilung der Last durchgeführt. Jeder Service teilt sich seine Datenbanken mit den gleichartigen Services die auf anderen Servern laufen. Die Services unter Docker[1] und Grafana[2] kommunizieren via Kafka[3] miteinander. Neben den Instanzen die

[1] https://docker.com
[2] https://grafana.com/
[3] https://kafka.apache.org/

für den fachlichen Kern der Anwendung notwendig sind, gibt es noch weitere Services. Diese werden zur Fehlersuche und Überwachung genutzt. Der Grafana Service übernimmt dabei einen Teil der Datenverarbeitung und die visuelle Aufbereitung. Eine Besonderheit ist, dass dafür eine Timescale[4] Datenbank verwendet wird. Um die Ausfallwahrscheinlichkeit zu verringern und die stetig steigende Last abzufangen ist der kritische Teil der Infrastruktur immer mehrfach ausgerollt. Es werden keine Cloud-Dienste genutzt. Es werden physische Server im Rechenzentrum angemietet.

[4] https://www.timescale.com/

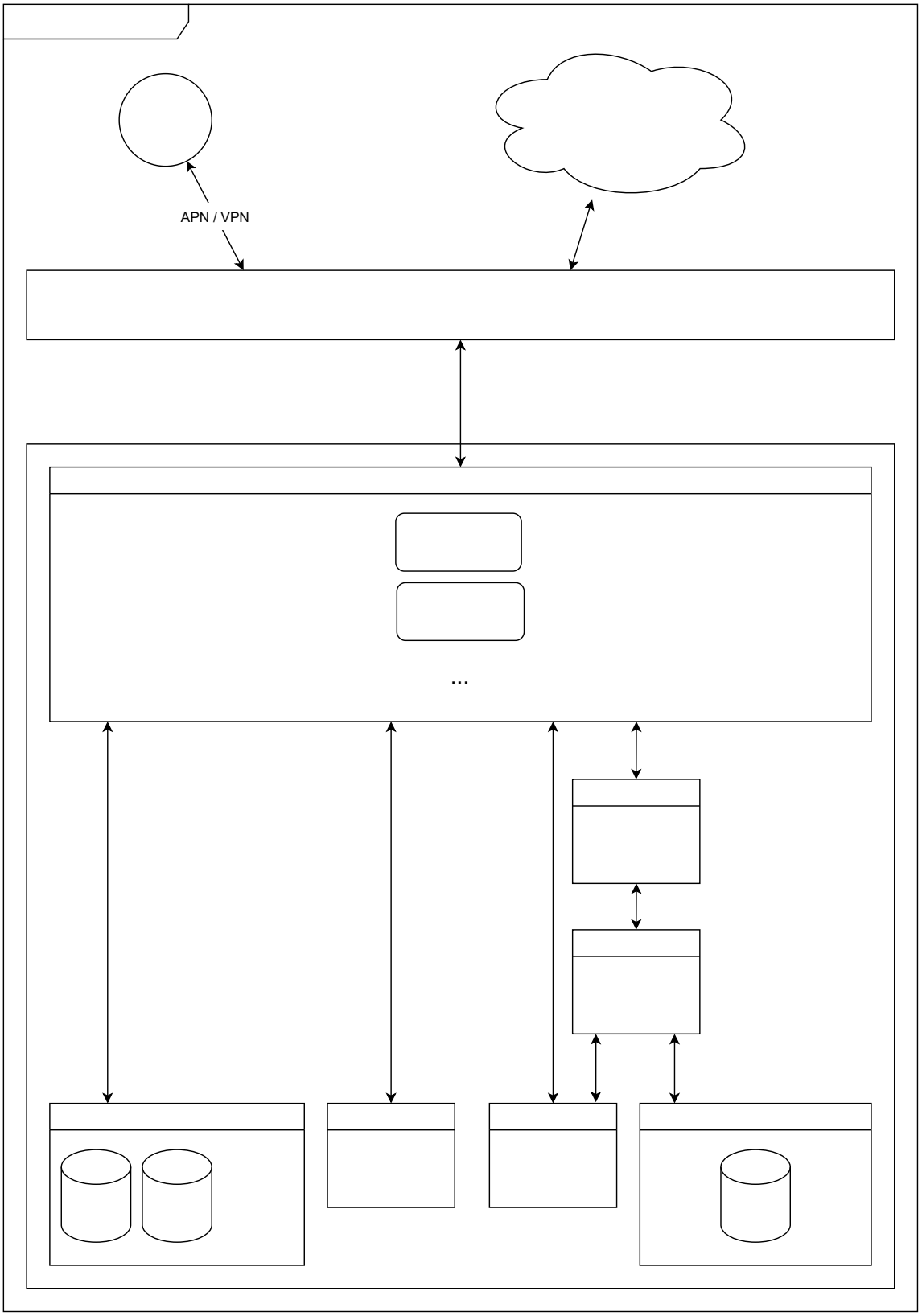


Abbildung 3.3: eRound Architektur

# 4 Anforderungsanalyse

Um eine Anwendung und ihre Architektur zu entwickeln, müssen die Anforderungen bekannt sein und dokumentiert werden. Dabei geht es sowohl um die Anforderungen an die eigene Entwicklung, als auch an extern entwickelte Komponenten. Die Ermittlung erfolgt mithilfe des Requirements Engeneering. Dies sorgt dafür, dass komplexe technische Unterfangen strukturiert analysiert und die Anforderungen fachgerecht dokumentiert werden (Rupp (2021)).

## 4.1 Funktionsweise

Die zu entwickelnde Anwendung soll es ermöglichen, das Ladeverhalten zu optimieren. Dazu soll es möglich sein einen Betriebshof (Depot) zu erstellen. Unter einem Depot werden die geplanten Ladevorgänge (Planned Charges) gesammelt. Eine weitere Zentrale Entität ist das Fahrzeug (Vehicle). Bei einem Betriebshof kann spezifiziert werden für wie lange im Voraus eine Berechnung stattfinden und wie oft eine Anpassung der Ladegeschwindigkeit innerhalb einer Stunde ermöglicht werden soll. Im Fahrzeug kann hinterlegt werden, wie die Ladegeschwindigkeit eingeschränkt ist und welche Kapazität die Batterie hat. Ein geplanter Ladevorgang kombiniert das Fahrzeug, den Betriebshof, sowie die Informationen wann der Ladevorgang gestartet und beendet werden soll, sowie den Umfang der Ladung. Die Pflege dieser Information soll durch Nutzereingaben als auch durch technische Schnittstellen ermöglicht werden. Analog zur Datenpflege soll auch die Ausführung der Berechnung von außen angestoßen werden können. Das Ergebnis der Berechnung soll ebenfalls durch eine Schnittstelle bereitgestellt werden.

## 4.2 Stakeholder

Zentraler Bestandteil des Requirements Engineering ist die Zusammenarbeit und Kommunikation mit den Stakeholdern. Es ist wichtig die Wünsche und Anforderungen der beteiligten Personen zu dokumentieren sowie im Zweifelsfall auch Konflikte zu moderieren (Rupp (2021)).

Da es sich bei der zu entwickelnden Anwendung um einen Prototypen handelt auf dem zukünftige Entwicklungen aufbauen sollen, spielen unternehmensinterne Interessen die Hauptrolle. Der wichtigste Stakeholder ist dabei der **Fachbereich**. Dieser arbeitet kontinuierlich an der Vision des gesamten Produktes. Für die technische Umsetzung spielen die **Entwickler** eine zentrale Rolle. Die zu entwickelnde Anwendung soll sich in die bereits existierende Service-Landschaft eingliedern. Für den **Betrieb** gibt es ein dediziertes Team, sodass auch deren Anforderungen berücksichtigt werden müssen.

Folglich ergeben sich folgende Stakeholder:

- Fachbereich
- Entwickler
- Betrieb

## 4.3 Systemkontext

Im Zuge der Dokumentation der zu entwickelnden Anwendung gibt der Systemkontext Klarheit über die Kontextgrenzen (Rupp (2021)).

Im ersten Schritt kann das Zielsystem relativ frei aufgebaut werden. Es wird lediglich die Kommunikation mit einem Nutzer via Hypertext Transfer Protocol (HTTP) vorausgesetzt. Sollte der Prototyp nach und nach in den produktiven Status überführt werden, ist es wichtig, dass es sich in die bereits existierende Service Landschaft integrieren lässt. Hier würde die Kommunikation sehr wahrscheinlich ausschließlich über den Messeging Service Kafka abgewickelt werden.

## 4.4 Anforderungen

Nach Absprache mit den Stakeholdern und unter Berücksichtigung der in Abschnitt 4.1 beschriebenen Funktionsweise, wurden folgende Anforderungen festgelegt.

### 4.4.1 Nicht-funktionale Anforderungen

Bei der Konzeption eines Systems müssen nicht nur die Funktionalität, sondern auch die umliegenden Faktoren beachtet werden. Dabei kann per Definition nach SOPHIST in folgende Kategorien unterschieden werden (Rupp (2021)):

- Anforderungen an die Benutzungsoberfläche (**BO**)
- Qualitätsanforderungen (**QA**)
- Anforderungen an die Technologie (**TE**)
- Rechtlich-vertragliche Anforderungen (**RE**)
- Anforderungen an durchzuführende Tätigkeiten (**DT**)
- Anforderungen an sonstige Lieferbestandsteile (**SL**)

Mit der gegebenen Kategorisierung und den von den Stakeholdern geäußerten Erwartungen haben sich folgende nicht-funktionale Anforderungen ergeben:

**BO01 - Weboberfläche** Es soll einen Service geben, der einem Nutzer die Verwaltung und Auswertung der Daten über eine Weboberfläche ermöglicht.

**BO02 - Funktionalität** Die Weboberfläche soll einfach und funktional gehalten werden.

**BO03 - Graphische Darstellung** Das Ergebnis der Optimierung soll graphisch dargestellt werden.

**QA01 - Gleichmäßigkeit der Leistung** Bei der Berechnung kann nicht berücksichtigt werden, wie die Ladegeschwindigkeit im vorherigen Zeitabschnitt ist. Es ist daher akzeptiert, dass der Ladeverlauf eines Fahrzeugs abrupten Schwankungen unterliegt.

**QA02 - Einsatzbereich** Da die Volatilität des Stromverbrauchs ein großes Problem in der aktuellen Zeit darstellt, könnte die Idee dieser Arbeit Anwendung in vielen Industrien finden. Die Anforderung an diese Arbeit beschränkt sich auf das Optimieren von Ladevorgängen im Kontext eines Betriebshofs für Elektrofahrzeuge.

**QA03 - Leistungsfähigkeit** Die Berechnung des optimalen Ladeverhaltens muss performant durchführbar sein. Bei hundert Fahrzeugen sollte die Berechnung innerhalb von 10 Sekunden erfolgen.

**QA04 - Präzision der Berechnung** Das Ergebnis der Berechnung des optimalen Ladeverhaltens soll um weniger als 100 Watt genau sein.

**TE01 - Eingliederung in die bestehende Microservice Landschaft** Andere Services basieren auf dem Stack Java / Spring / Maven und werden in einem Docker-Umfeld ausgerollt. Der zu entwickelnde Service soll sich in dieses Bild einfügen.

**TE02 - Schnittstelle** Um eine Kommunikation zwischen den Services zu ermöglichen, sollen Schnittstellen via Kafka und REST implementiert werden.

**TE03 - Asynchronität** Kommunikation zwischen Prozessen / Services möglichst asynchron halten.

### 4.4.2 Funktionale Anforderungen

Funktionale Anforderungen beschreiben die Funktionalität und das Verhalten eines Systems. Zur Feststellung bietet sich die Use-Case-basierten Analyse und die Dokumentation in Form eines Use-Case-Diagramms an (Rupp (2021)).

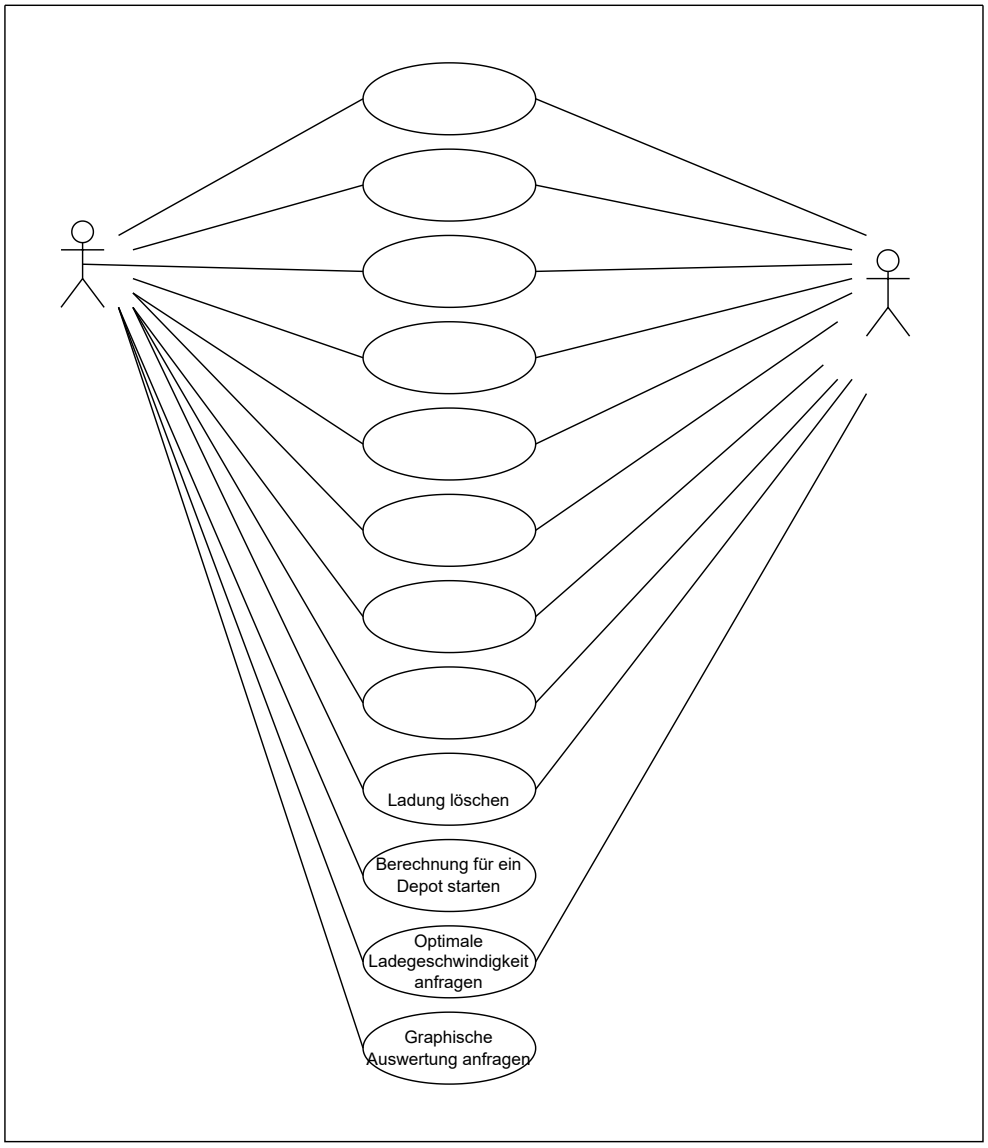


Abbildung 4.1: User Stories als Use-Case-Diagramm

Das in Abbildung 4.1 dargestellte Use-Case-Diagramm gibt einen guten Überblick über den Umfang der Funktionalität und veranschaulicht welche Akteure den Service nutzen. Für die genauere Untersuchung und Nutzung des Prototypen gibt es Zugriff für einen Benutzer in Form einer Weboberfläche. Sollte die Anwendung in der Produktion eingesetzt werden, wird diese Kommunikation von umliegenden Microservices übernommen. Eine Überprüfung der Vertrauenswürdigkeit des Kommunikationspartners ist zu keinem Zeitpunkt notwendig, da immer davon ausgegangen werden kann, dass sich die Anwendung in einem geschützten Umfeld befindet.

## 4.5 Anwendungsdomäne

Die Domäne beschreibt das direkte Umfeld und die Struktur der Problemwelt. Es werden Entitäten festgelegt und ihre Beziehungen zueinander definiert. Um diese zu visualisieren eignen sich Unified Modeling Language (UML)-Klassendiagramme (Rupp (2021)).

Die zentralen Entitäten zur Optimierung des Ladeverhaltens werden in Abbildung 4.2 beschrieben. Ein geplanter Ladevorgang (PlannedCharge) besteht sowohl aus einem Fahrzeug (Vehicle) als auch einem Betriebshof (Depot) und besitzt Attribute die für Ankunft und Abfahrt sowie den zeitlichen Aspekt als auch den Ladezustand des Fahrzeugs beschreiben. Im Fahrzeug wird festgehalten, wie viel Kapazität die Batterie fasst und welchen Einschränkungen die Ladegeschwindigkeit unterliegt. Dem Betriebshof können ebenfalls Beschränkungen der Ladegeschwindigkeit hinterlegt werden. Zudem werden in der Entität des Betriebshofes Parameter zur Berechnung hinterlegt. Es wird die Unterteilung einer Stunde (timeslots_p_H) und die Größe des Gesamtzeitraums (calculationLimit_H) der Berechnung definiert. Die Entität der Ladegeschwindigkeit (ChargingSpeed) repräsentiert die berechnete Ladegeschwindigkeit nachdem die Optimierung stattgefunden hat. In ihr sind Fahrzeug und Betriebshof sowie die Höhe und der Zeitraum der Gültigkeit der Ladegeschwindigkeit hinterlegt.

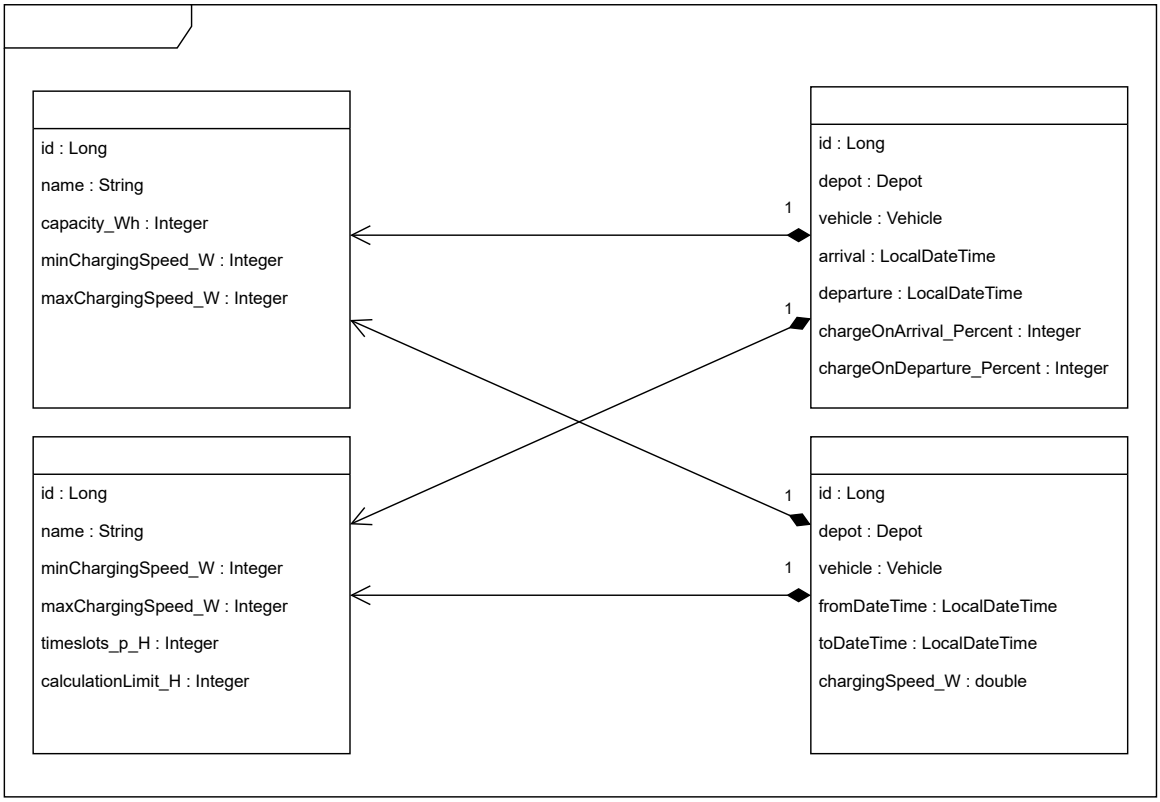


Abbildung 4.2: Klassendiagram der persistenten Entitäten

# 5 Konzeption der Lösung

Mithilfe der dokumentierten Anforderungen kann nun die zu entwickelnde Anwendung konzipiert werden.

## 5.1 Infrastruktur

Die grundlegenden Infrastruktur-Entscheidungen sind durch die schon bestehende Anwendung vorgegeben. Services werden via Docker bzw. Docker Compose ausgerollt. Die zu entwickelnde Anwendung muss ohne viel Aufwand integrierbar sein (TE01).

Docker bietet eine Plattform zum Betrieb von Anwendungen durch Containerisierung. Dadurch entsteht eine Isolation zwischen den einzelnen Services wie bei einer virtuellen Maschine. Im Gegensatz zu einer virtuellen Maschine ist kein eigenes Betriebssystem nötig. Es handelt sich dabei um isolierte Prozesse, die alle den Kernel des unterliegenden Betriebssystems nutzen. Es vereinfacht das Ausrollen neuer bzw. weiterer Services und senkt die Kosten von Hardware und Personal. Das nutzen von Docker bietet viele Vorteile und hat sich etabliert. Dies sorgt dafür, dass viele Software Produkte mit dem Bewusstsein entwickelt werden, dass es containerisiert wird (Rad, Bhatti und Ahmadi (2017)).

Jeder Docker Container basiert auf einem Image. Ein Image ist ein Packet von Software und deren Abhängigkeiten. Die Konfiguration erfolgt über eine Datei - Dockerfile. Es kann sinnvoll sein, dass zur Erstellung eines Images mehrere Phasen durchlaufen werden. Die Ergebnisse der vorherigen Phase können dabei von den folgenden Phasen aufgegriffen werden. Um die Performance zu verbessern, wird bei der Erstellung von Images auf einen Cache zurückgegriffen. Sollte in der Vergangenheit bereits eine Abhängigkeit heruntergeladen oder eine Stage durchlaufen worden sein, wird dieser Prozess abgekürzt und es werden schon bestehende Daten verwendet.

Um zusammenhängende Images bzw. Container strukturiert verwalten, starten und stoppen zu können, gibt es eine Weiterentwicklung - Docker Compose[1]. Compose bietet die Möglichkeit in einer Datei im YAML Ain't Markup Language (YAML)-Format eine komplette Umgebung zu beschreiben. Dadurch, dass diese Datei fest hinterlegt wird, muss beim Betrieb keine lange Folge an Kommandos abgefeuert werden. Es reicht ein einfacher Befehl um die gesamte Umgebung zu starten. Neben den Services lassen sich auch Volumes konfigurieren. Diese werden benötigt um Daten, über die Lebensdauer eines Services hinaus, zu persistieren (Docker, Inc (2022)).

## 5.2 Architektur

Der zu entwickelnde Prototyp soll sich in eine bereits existierende Umgebung integrieren. In Folge dessen ist die Infrastruktur (Abschnitt 5.1) komplett und die Architektur zu Teilen bereits vorgegeben. In diesem Abschnitt wird das darunterliegende Konzept erläutert und die Integration beschrieben.

### 5.2.1 Microservices und Monoliten

Die bereits bestehende Architektur (Abbildung 3.3) basiert auf dem Konzept für Microservices. Diese Architektur findet immer mehr Anklang bei der Entwicklung von modernen Anwendungen. In einer Umfrage mit über 1500 Teilnehmern aus dem Jahr 2020 gaben 61 Prozent der Befragten an, dass ihr Unternehmen seit mehr als 1 Jahr das Konzept der Microservices nutzen (Loukides (2020)). Dies zeigt die zentrale Bedeutung dieses Konzepts bei der Bereitstellung moderner Dienste. Die Idee dahinter ist, dass es viele kleine und unabhängige Services gibt, die alle einen abgegrenzten Bereich der Domäne abbilden.

Die Alternative dazu, die in der Vergangenheit oft zur Anwendung kam, ist die monolithische Architektur. Bei diesem Konzept wird nur eine Einheit ausgeliefert, die die gesamte Geschäftslogik und Kommunikation abbildet. Verglichen mit Microservices vereinfacht dieser zentralisierte Ansatz die Fehlersuche und Datenhaltung. Monolithen sind auch skalierbar - jedoch nur als ganzes. Dies ist problematisch, da nicht jeder Teilbereich gleich viele Ressourcen benötigt. Für kleine Anwendungen ist der Monolith ein gängiges Mittel. Wird die Anwendung jedoch komplexer wird das Konzept der Microservices

[1] https://docs.docker.com/compose/

immer sinnvoller. Je komplexer eine Anwendung wird, desto aufwendiger wird es neue Mitarbeiter einzuarbeiten. Bei Microservices entsteht der Vorteil, dass die Struktur es erleichtert einem Mitarbeiter sein Verantwortungsgebiet zu erklären. Interne Grenzen sind leicht zu definieren. Bei Grenzübertritt muss die Kommunikation klar definiert werden. Was hinter der Systemgrenze passiert unterliegt der Zuständigkeit eines anderen Bereichs. Auch wird die parallele Entwicklung von Features begünstigt. Services können unabhängig voneinander versioniert werden, dadurch können Änderungen schneller in die produktive Umgebung eingebunden werden.

Neben vielen Vorteilen entstehen auch einige Herausforderungen durch eine Microservice-Architektur. Jeder Service hat seine eigene Datenhaltung und besitzt die Hoheit über seine Untermenge des Datensystems. Diese beinhaltet meist nur wenige Entitäten der gesamten Domäne. Jedoch benötigen andere Services auch Teile dieser Daten, sodass die Kommunikation mit steigender Anzahl an Services und steigender Komplexität immer wichtiger wird. Ändert sich ein Datensatz innerhalbt eines Services muss sichergestellt werden, dass diese Änderung an sämtliche Stellen des Gesamtsystems kommuniziert wird. Bei einem Monolithen stellt die Sicherstellung der Konsistenz der Daten keine besondere Herausforderung dar (Jamshidi, Pahl, Mendonça, Lewis und Tilkov (2018)).

### 5.2.2 Sichten

Sichten sind ein gängiges Mittel um die Architektur einer Anwendung zu visualisieren. Ein weit verbreitetes Modell ist das 4+1 Sichtenmodell von Philippe Kruchten (Kruchten (1995)). Dieses nutzt 4 Blickwinkel um eine Anwendung zu beschreiben und legt die zu verwendende Notation fest. Auf die Festlegung der Nutzung bestimmter Diagramme wird dabei verzichtet. Es muss sichergestellt werden, dass jedem Stakeholder die nötigen Informationen transportiert werden (Owens (2014)). Die 4 Sichten laut Kruchten (1995) sind die folgenden:

- Logische Sicht
- Prozesssicht
- Physikalische Sicht
- Entwicklungssicht

Im folgenden werden nun einige Sichten mithilfen von Diagrammen visualisiert:

### Komponentendiagramm

Ein Komponentendiagramm stellt die einzelnen Bausteine und die Kommunikationswege vereinfacht dar. Jede Komponente wird dabei als Blackbox betrachtet. Die Verwendung von spezifischen Begriffen wird, wenn möglich, vermieden.

Zwei der in Abbildung 5.1 dargestellten Komponenten existieren bereits unabhängig von der Entwicklung - eRound-Service und Message Queue. Die zentrale Komponente für die Neuentwicklung trägt den Namen ECLB - kurz für *Electric Charging Load Balancer*. Die Kommunikation untereinander läuft meist synchron über die angegebenen Schnittstellen. Die Kommunikation mit der Komponente Message Queue stellt dabei eine Ausnahme dar und läuft asynchron ab. Die Komponente ECLB-FE ist die Weboberfläche die zur Anpassung von Modellen der Optimierung genutzt werden können. Die Komponente mit dem Namen Gurobi sorgt für eine effiziente Optimierung der Modelle.

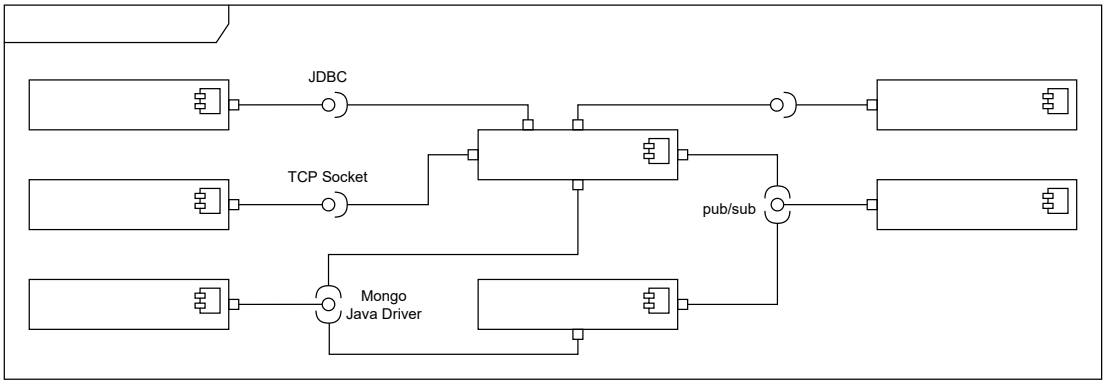


Abbildung 5.1: Komponentendiagramm des Systems aus der Blackbox-Sicht

### Sequenzdiagramm

Mithilfe eines Sequenzdiagramms kann sowohl die logische Sicht als auch die Prozesssicht veranschaulicht werden. Das Diagramm aus Abbildung 5.2 zeigt den Ablauf einer Optimierung die durch einen weiteren eRound Service angestoßen wurde.

Für die Berechnung ist es erforderlich, dass ein JSON Objekt in der MongoDB Datenbank hinterlegt wird. Das Objekt sollte die gleiche Struktur aufweisen wie Anhang A.1. Der Service erhält eine Id, unter der der hinterlegte Datensatz zu finden ist, zurück. Diese Id wird via Kafka veröffentlicht. Die neue Vertikale (ECLB) aboniert den entsprechenden

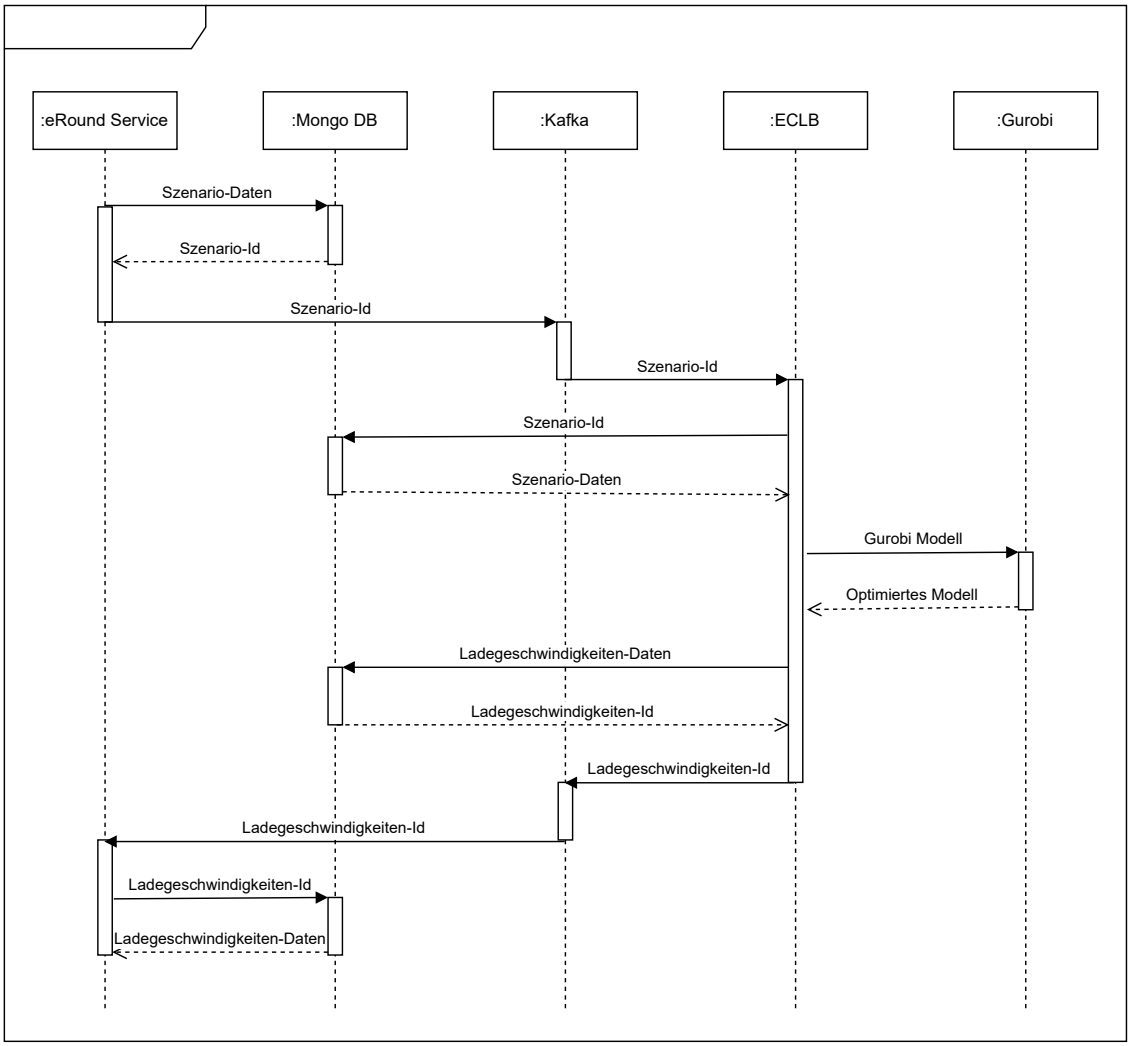


Abbildung 5.2: Sequenzdiagramm für den Prozess der Optimierung angestoßen durch einen eRound Service

Kanal und empfängt die Id. Mithilfe der Id wird der Datensatz aus der MongoDB Datenbank ausgelesen und in eine optimierbares Modell übersetzt. Dieses wird zur Berechnung an den Gurobi Service übermittelt. Das Resultat ist ein optimiertes Modell welches vom ECLB Service in einen JSON Datensatz überführt und in der MongoDB Datenbank hinterlegt wird. Die Struktur dieses Objekts entspricht dem Anhang A.2. Die erhaltene Id wird via Kafka veröffentlicht und vom eRound Service empfangen. Mithilfe der Id wird der entsprechende Datensatz aus der Datenbank gelesen. Im nächsten Schritt steuert der eRound Service die Ladesäulen an.

Für die Erstellung und Manipulation von Modellen soll eine Weboberfläche erstellt werden. Der Ablauf von der Erstellung bis zur Durchführung der Optimierung werden in Abbildung 5.3 dargestellt.

Dem Nutzer stehen auf der Weboberfläche (ECLB-FE) Möglichkeiten bereit Depots, Fahrzeuge und geplante Ladevorgänge zu erstellen. Diese Entitäten werden mit entsprechenden Werten befüllt an das Backend (ECLB) gesendet. Die Werte die in dem jeweiligen Formular befüllt werden müssen, sind dem Klassendiagramm aus Abbildung 4.2 zu entnehmen. Geplante Ladevorgänge bedingen die Entitäten Depot und Fahrzeug und müssen daher zuletzt erstellt werden. Nachdem der Erstellprozess zufriedenstellend durchgeführt wurde, kann die Optimierung angestoßen werden. Der Nutzer erhält eine Erfolgsmeldung.

Wie aus den User Stories aus Abbildung 4.1 zu entnehmen, soll es dem Nutzer ermöglicht werden eine Graphische Auswertung aufrufen zu können. Der dazugehörige Ablauf würde in Abbildung 5.4 visualisiert. Hierfür müssen die Daten aufbereitet werden. Nachdem die Abfrage im Backend eingegangen ist, werden die errechneten Ladegeschwindigkeiten in verwertbare Objekte umgewandelt und ans Frontend übermittelt. Dort findet die tatsächliche visuelle Aufbereitung statt, die dem Nutzer an der Weboberfläche präsentiert wird.

### Klassendiagramm

Ein weiterer Diagrammtyp um die logische Sicht zu visualisieren ist das Klassendiagramm. In Abbildung 4.2 sind die Klassen dargestellt deren Objekte persistiert werden. Da für die Optimierung einige temporäre Objekte erzeugt werden, wurden diese separat dargestellt. Das entsprechende Klassendiagramm ist der Abbildung 5.5 zu entnehmen. Für die Optimierung wird ein Kalkulationsobjekt (Calculation) erstellt. In mehreren

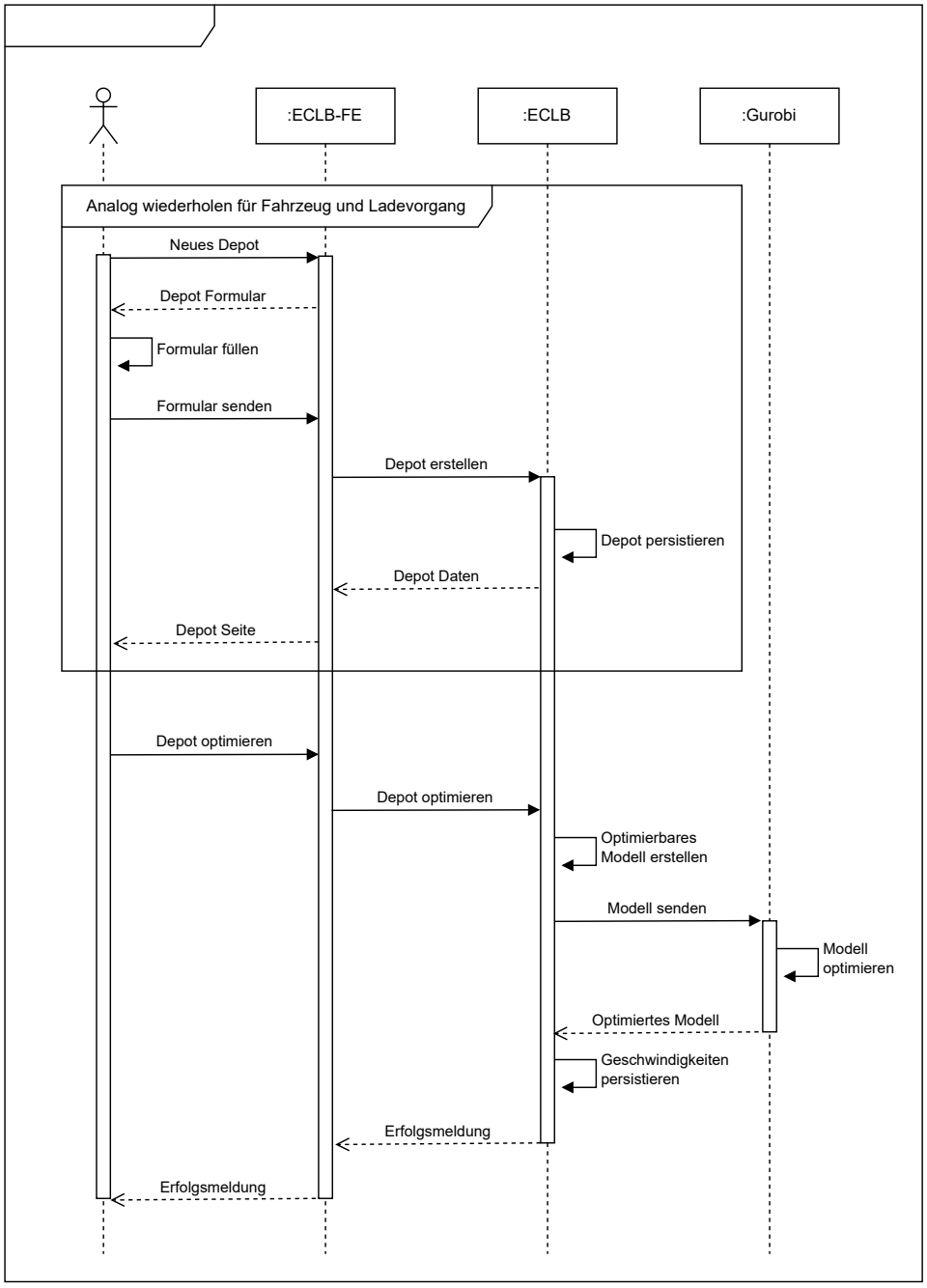


Abbildung 5.3: Sequenzdiagramm für den Prozess der Optimierung erstellt durch einen Nutzer

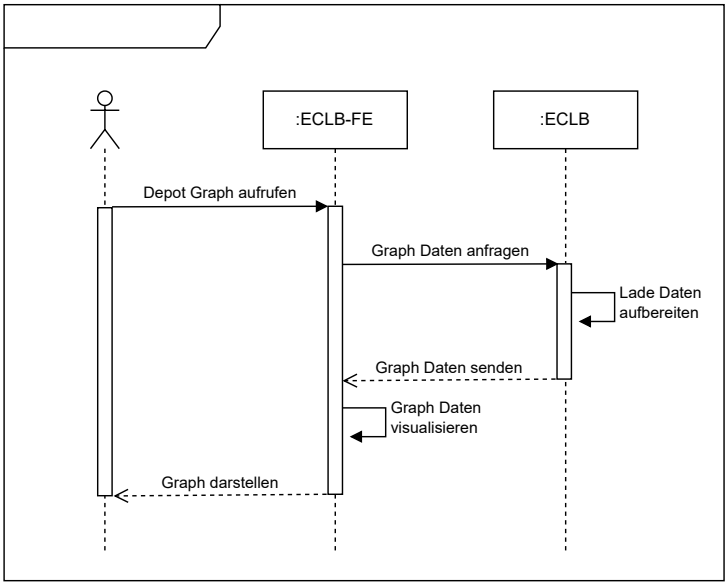


Abbildung 5.4: Sequenzdiagramm für den Aufruf der graphischen Aufbereitung durch den Nutzer

Schritten wird dieses angereichert. Nach der Durchführung aller Schritte befindet sich das Ergebnis in kompakter Form in dem Feld matrix. Das Ergebnis wird im Anschluss in eine Menge von ChargingSpeed-Objekten (Abbildung 4.2) umgewandelt und persistiert. Wie in der Abbildung zu erkennen, sind die Klassen ConsumerFrame, Consumer und Matrix generisch gehalten. Dies sorgt für eine Wiederverwendbarkeit in andere Szenarien.

## 5.3 Kommunikation

Bei der Kommunikation der Komponenten untereinander muss zwischen zwei Arten unterschieden werden: synchron und asynchron. Im folgenden Abschnitt werden die in Abbildung 5.1 dargestellten Schnittstellen beschrieben.

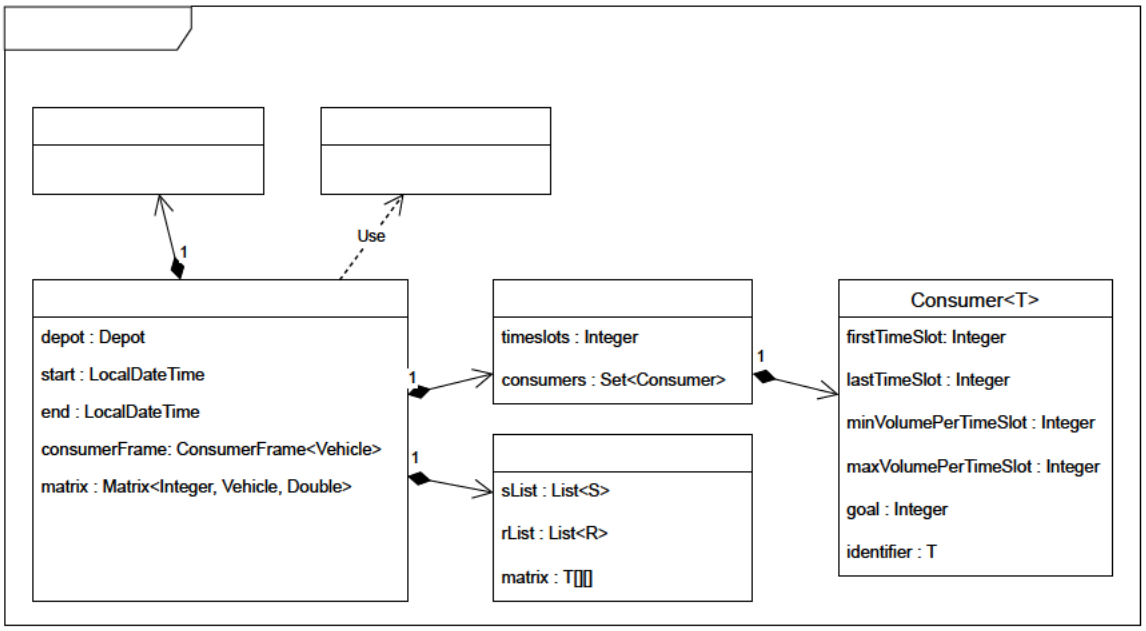


Abbildung 5.5: Klassendiagramm der temporären Entitäten

### 5.3.1 Synchrone Schnittstellen

Die Schnittstellen mit den Bezeichnungen JDBC[2], (Gurobi-)TCP Socket[3], Mongo Java Driver[4] und Rest Application Programming Interface (API) fallen dabei in die Kategorie der synchronen Kommunikation. In der Praxis bedeutet das, dass ein Prozess eine Anfrage stellt, dieser auf die Antwort gewartet und erst im Anschluss mit der weiteren Logik fortfährt. Die ersten drei der aufgeführten Schnittstellen sind dabei produktspezifisch und durch die verwendeten Services alternativlos.

Anders ist es bei der Representational State Transfer (REST) API die für die Kommunikation zweier zu entwickelnden Services zuständig ist. Diese basiert auf dem HTTP Protokoll und ist nach REST entworfen. Jede Entität bekommt eine Uniform Resource Identifier (URI) zugewiesen. Um Daten aufzurufen bzw. zu verändern werden die HTTP-Verben genutzt. Die Gegenseite antwortet mit einem HTTP Statuscode und optional mit einem Datensatz. Ein Merkmal ist die Idempotenz. Ausgenommen bei POST Anfragen

[2]https://www.oracle.com/java/technologies/javase/javase-tech-database.html
[3]https://www.gurobi.com/
[4]https://mongodb.github.io/mongo-java-driver/

sollte das mehrfache Ausführen der gleichen Anfrage immer die selbe Antwort liefern (Patni (2017)).

### 5.3.2 Asynchrone Schnittstellen

Anders als bei der synchronen Kommunikation wird bei der asynchronen Kommunikation nicht auf die Antwort gewartet. Der Wunsch nach dieser Art der Kommunikation wird durch die Anforderung TE03 geäußert. Dies ist bei internen Abläufen interessant, wenn keine direkte Antwort nötig oder die Nachricht an mehrere Services gerichtet ist. Durch die Asynchronität und Verteilung der Daten geht die Transaktionssicherheit verloren. Die Konsistenz der Daten ist möglicherweise verzögert. Dies ist aber in den meisten Anwendungsfällen vertretbar. In dem System wird die Asynchronität genutzt um andere Services über Neuerungen zu informieren.

Innerhalb eines Systems werden Nachrichten anhand eines Topic unterschieden. Nachrichten eines Topic haben das gleiche Format. Services können Nachrichten eines Topic publiziert, welche dann von Services die diesen Topic abonniert haben empfangen wird. In größeren System kann Parallelisierung eingesetzt werden. Dafür werden Topics auf mehrere Partitionen verteilt. Das Weiterreichen der Nachrichten wird durch Broker umgesetzt (Ferreira (2013)).

Die Umgebung, in die der Prototyp integriert werden soll, arbeitet mit dem Messaging Produkt Kafka und dieser wird übernommen.

## 5.4 Datensicherung

Für den zu entwickelnden Prototypen ist es angedacht einige der Daten zu persistieren. Dabei gibt es zwei Arten von Datensätzen. Zum einem geht es um Daten die direkt in die Domäne der Anwendung fallen. Des Weiteren ist geplant, eine Datenbank zu nutzen um die Kommunikation zu unterstützen. Die Größe der Daten die entstehen, sind nicht geeignet um über einen Messenger gesendet zu werden.

### 5.4.1 Relationale Datenbank

Für das Speichern von zentralen Entitäten bieten sich relationale Datenbanken an. Relationale Datenbanken sind sehr flexible in ihrer Datenstruktur und ermöglichen das Abbilden von Entitäten und ihre Beziehungen untereinander. Das Konzept ist lange etabliert und es gibt eine große Auswahl an Produkten (Steiner (2014)).

Da es für den Prototypen nicht wichtig ist, dass die Daten langfristig gespeichert werden und die aktuelle Planung es nahe legt, dass der Service im Umfeld der bestehenden Applikation auch in Produktion keine relevante Rolle bei der Datensicherung übernimmt, wurde sich für eine untypische Lösung entschieden. Bei H2[5] handelt es sich um eine Datenbank die oftmals bei der Entwicklung eingesetzt wird. Sie zeichnet aus, dass die sehr leichtgewichtig ist und im In-Memory-Modus besonders performant ist (h2database.com (2021)).

Sollten sich die Anforderungen in Zukunft ändern, ist es sehr leicht die Technologie auszutauschen. Grund dafür ist die Umsetzung der Spezifikation der Java Persistence API (JPA) in Spring. Diese ermöglicht es relationale Datenbanken untereinander zu tauschen, ohne die Logik der Anwendung anzupassen.

### 5.4.2 Dokumenten Datenbank

Um die Kommunikation durch Kafka (Abschnitt 5.3.2) zu unterstützen, soll eine dokumentenorientierte Datenbank eingesetzt werden. Diesen Typ von Datenbank zeichnet aus, dass kein Schema festgelegt wird sondern ganze Objekte einfach abgelegt werden (Hows, Membrey und Plugge (2014)).

In diesem Fall wird eine MongoDB-Datenbank[6] verwendet. Ein Open-Source Produkt welches das Feld der Dokumenten-Datenbanken dominiert (solid IT gmbh (2022)).

## 5.5 Optimierung

Für die Umsetzung des Prototypen wird ein Programm benötigt, dass in der Lage ist gemischt-ganzzahlige Modelle zu lösen bzw. zu optimieren. Nach Absprache mit dem

[5] http://www.h2database.com/
[6] https://www.mongodb.com/

Fachbereich (Abschnitt 4.2) wurde sich für das Produkt Gurobi[7] entschieden. Da die Nachfrage nach dieser Art von Software relativ klein ist, ist es schwerer einen Überblick über das Angebot zu erlangen. Der Fachbereich hat bereits positive Erfahrungen mit diesem Anbieter gemacht und es werden Lösungen angeboten, die eine Integration in dieses Projekt ermöglichen. Für die Nutzung der Software ist eine Lizenz nötig.

[7] https://www.gurobi.com/

# 6 Umsetzung

Nachdem das Konzept erstellt wurde, wird nun die Umsetzung erläutert. Hier werden zentrale Themen der Umsetzung und aufgetretene Hindernisse angesprochen. Im Anhang befinden sich Ausschnitte aus dem entwickelten Quellcode.

## 6.1 Frameworks

Für die Umsetzung wurden Frameworks zur Hilfe genommen. Es wird erläutert welche gewählt und welche Vorteile dadurch generiert wurden.

### 6.1.1 Frontend

Mit dem Frontend ist eine Weboberfläche gemeint, mittels derer die Nutzer mit dem Backend interagieren können. Die Datensätze können verändert, die Optimierung angestoßen und das Ergebnis visualisiert werden.

Für die Entwicklung des Frontends wurde das JavaScript Framework NextJS[1] verwendet. Dabei handelt es sich um eine Weiterentwicklung basierend auf dem Framework React[2]. Ein zentrales Konzept ist die Unterteilung in Komponenten. Diese werden in der Regel so entworfen, dass sie wiederverwendet werden können. Mithilfe einer Komponente lassen sich auch mehrere Komponenten bündeln - sodass am Ende ein sehr verschachteltes Konstrukt entsteht. Dadurch entsteht eine geringe Code Basis die immer wiederverwendet werden kann. Des Weiteren wird mit einem virtuellen Document Object Model (DOM) gearbeitet. Dadurch können einzelne Seiteninhalte gezielt aktualisiert werden. Die entstandenen Änderungen werden dann auf den tatsächlichen DOM übertragen. Entwickelt

[1] https://nextjs.org/
[2] https://reactjs.org/

wird in Java Serialization to XML (JSX), einer Mischung aus JavaScript und Hypertext Markup Language (HTML). Mithilfe des Codes erstellt React den virtuellen DOM. (Gackenheimer (2015))

Das NextJS übernimmt diese Struktur von React und baut darauf weitere Funktionen auf. Mit seinen zahlreichen Funktionen erleichtert NextJS die Programmierarbeit. Es wurde gewählt, weil es schnell und einfach das Entwickeln von strukturierten und benutzerfreundlichen Anwendungen ermöglicht.

### 6.1.2 Backend

Mit dem Backend ist die Software gemeint, die im Hintergrund läuft und das Kernstück der Entwicklung abbildet. Über das Backend findet der gesamte Datenverkehr statt.

Das Backend wurde in Java, unter Zuhilfenahme des Frameworks Spring[3], entwickelt. Wie aus der Anforderung TE01 zu erkennen, sind andere Services bereits mithilfe dieses Frameworks entwicklt worden. Es ist das Mittel der Wahl im unmittelbaren Umfeld der Applikation.

Spring ermöglicht durch das Anpassen der Konfiguration externe Ressourcen als *Dependecy* einzubinden. Entwicklern ermöglicht dies neue Technologien ohne großen Aufwand zu nutzen und auszuprobieren. Da die externen Ressourcen im Hintergrund geladen werden und teilweise nur konfiguriert werden müssen, kann mit wenigen Zeilen sehr viel Logik entstehen. Für das Verwalten dieser Dependencies gibt es eigene Software (z.B. Maven[4]) und unzählige Datenbanken.

## 6.2 OpenAPI und Swagger

OpenAPI ist ein Standart, der genutzt wird um REST-APIs zu spezifizieren. In die Spezifikation fließen die verfügbaren Endpunkte, die verwendeten HTTP-Verben und das Format der Daten, die empfangen oder gesendet werden können, ein. Diese Spezifikation nutzt Swagger[5] um daraus eine eigene Weboberfläche zu erstellen. Dort werden die genannten Spezifikation visuell aufbereitet und ermöglichen das Testen der Endpunkte.

[3] https://spring.io/
[4] https://maven.apache.org/
[5] https://swagger.io/

Mithilfe von Spring (Abschnitt 6.1.2) ist das Nutzen dieser Möglichkeit extrem einfach. Für dieses Projekt war lediglich das Einbinden der Dependency und eine minimale Konfiguration (Anhang C.1) nötig.

Die Verwendung von OpenAPI bzw. Swagger sorgt dafür, dass eine Dokumentation generiert wird, auf die bei der Entwicklung zurückgegriffen werden kann. Sowohl bei der Entwicklung des Backends als auch beim Frontend schafft dies Anhaltspunkte für den Entwickler die er zur Orientierung nutzen kann.

## 6.3 Optimierung

Das Herzstück der Anwendung ist die Optimierung des Ladeverhaltens. Um eine Optimierung durchführen zu können müssen alle relevanten Größen in einem Modell zusammengefasst werden. Ein Modell besteht aus einer Menge Ungleichungen und einem Ziel. Ein Ziel ist eine Maximierung oder Minimierung eines Wertes. Das Modell für diese Anwendung wird im folgenden Abschnitt beschrieben.

### 6.3.1 Theorie

Zunächst erfolgen die Erläuterungen der verwendeten Variablen des Modells. Für jede Kombination aus Zeitfenster und Fahrzeug gibt es eine Variable. Für die Summe pro Zeitfenster und Fahrzeug gibt es jeweils eine weitere Variable. Die Variablen, die die Summe der Fahrzeuge darstellen, müssen dem Wert des Ladeziels des Fahrzeugs gleichgesetzt werden. Des Weiteren gibt es eine Variable die den Wert der Lastspitze darstellt. Sie muss größer oder gleich sein, wie jede Zeitfenster Summe. Das Ziel des Modells ist die Minimierung der Variable, die die Lastspitze repräsentiert.

Um das Modell zu verkleinern, werden für Zeitfenster-Fahrzeug-Kombinationen (ZFK) an denen das Fahrzeug nicht geladen werden kann, keine Variablen erstellt. Für das Abbilden der Limitierungen des Ladevorgangs sind weitere Ungleichungen nötig. Es muss abgebildet werden, dass die Fahrzeuge entweder gar nicht ($0$) oder mit einen definierten Bereich ($Min$ - $Max$) an Leistung laden. Dafür sind für jede Zeitfenster-Fahrzeug-Kombination zwei weitere Variablen nötig. Die Gleichung zur Modellierung ist in Abbildung 6.1 dargestellt. Die Variablen und ihre Unter- sowie Obergrenzen sind Tabelle 6.1 zu entnehmen.

Die Werte können nur ganze Zahlen annehmen. Der Faktor kann nur die Werte 0 und 1 annehmen.

$$\text{Zeitfenster-Fahrzeug-Kombinationen} = \text{Bereich} - (\ Max \cdot \text{Faktor}\ )$$

Abbildung 6.1: Gleichung zur Modellierung der Teillast

| **Variable** | **Obergrenze** | **Untergrenze** |
|---|---|---|
| ZFK | 0 | Max |
| Bereich | Min | Max |
| Faktor | 0 | 1 |

Tabelle 6.1: Ober- und Untergrenze der der Variablen zur Modellierung der Teillast

### 6.3.2 Beispiel

Die Zusammensetzung des Wertes lässt sich anhand eines Beispiels veranschaulichen. Angenommen ein Fahrzeug kann entweder gar nicht laden oder mit einer Geschwindigkeit im Bereich von 4-6 Watt, dann ergibt sich durch Anwendung der Gleichung (Abbildung 6.1) und der Ober- und Untergrenzen (Tabelle 6.1) eine mögliche Belegung wie in Tabelle 6.2 dargestellt. Da negative Werte (rot hinterlegt) für ZFK ausgeschlossen wurden, entfallen diese Belegungen ebenfalls aus einer möglichen Belegung des Modells. Es bleiben nur die Werte 0, 4, 5 und 6 (grün hinterlegt) - so wie gewünscht.

| **Bereich** | **Faktor** | **ZFK** |
|---|---|---|
| 4 | 0 | 4 |
| 5 | 0 | 5 |
| 6 | 0 | 6 |
| 4 | 1 | -2 |
| 5 | 1 | -1 |
| 6 | 1 | 0 |

Tabelle 6.2: Belegung und Rechnung für den Teillastbereich

## 6.4 Gurobi

Die zentrale Herausforderung dieses Projektes waren die Einbindung und Nutzung von Gurobi. Gurobi ist die gewählte Komponente, die die Optimierung schlussendlich durch-

führt. Der Kern der Software ist in C geschrieben. Um diesen in möglichst vielen Projekten nutzen zu können, werden entsprechende Schnittstellen bereitgestellt.

### 6.4.1 Entwickler Computer

Während der Entwicklung der Applikation arbeitet und testet ein Entwickler auf einem Computer. Gurobi bietet viele Schnittstellen an, durch die es möglich ist, in vielen verschiedenen Programmiersprachen eine Optimierung durchzuführen. Die Anwendung die entwickelt wurde, ist in Java geschrieben. Um das Durchführen einer Optimierung auf dem Computer des Entwicklers zu ermöglichen, muss das gesamte Programm[6] auf dem Rechner des Entwicklers installiert werden - auch wenn die Optimierung auf einem anderen Server stattfindet.

### 6.4.2 Optimierung im Container

Da die Anwendung in Echtbetrieb in einem Container ausgeführt werden soll, muss auch an dieser Stelle dafür gesorgt werden, dass der Gurobi Kern verfügbar ist. Für diesen Fall wurde vom Hersteller ein entsprechendes Image auf Docker Hub[7] publiziert. Die Konfiguration erfolgt durch eine Dockerfile-Datei (Anhang B.1). In den Zeilen 1-4 wird mithilfe von Maven die Java Applikation erstellt. In Zeile 6 wird das angesprochene Gurobi Image geladen. Die Zeilen 8-13 repräsentieren die Installation von Java. In Zeile 16 wird die erstellte Applikation aus der ersten Stage überführt. In den letzten Zeilen wird der Befehl zum Starten der Applikation definiert und welcher Port nach außen freigegeben wird.

Ein wichtiges Detail ist, dass bei der Erstellung der Java Applikation sichergestellt wird, dass die Gurobi Bibliothek eingebunden wird. Dies wurde durch einen Eintrag in die Konfiguration (Anhang B.3, Zeile 71-75) von Maven umgesetzt. Da die zentralen Datenbanken diesen Eintrag nicht hinterlegt haben, muss zudem ein Repository hinzugefügt werden (Zeile 21 -25).

[6] https://www.gurobi.com/downloads/gurobi-software/
[7] https://hub.docker.com/r/gurobi/optimizer

### 6.4.3 Dedizierter Service

Für die Nutzung von Gurobi ist das Erstellen einer dedizierten Instanz nicht notwendig. Es bietet jedoch die Vorteile, dass die Ablage der Lizenz an einem zentralen Ort stattfindet und die genutzten Ressourcen isoliert betrachtet werden können. Dies erleichtert die Überprüfung, ob die verfügbar Hardware bzw. freigegebene Leistung ausreichend ist. Um von diesen Vorteilen Gebrauch machen zu können, wurde ein dedizierter Container konfiguriert (Anhang B.2, Zeile 5-13).

### 6.4.4 Java Bibliothek

Mit Hilfe der Gurobi Bibliothek lassen sich Modelle in Java erstellen und lösen. Zentrale Komponente für die Übersetzung des Modells ist der entwickelte GurobiLoadBalancer (Anhang C.3). Dieser verfügt über eine Methode (C.3, Zeile 12), welche einen ConsumerFrame (Abbildung 5.5) als Parameter erwartet. Nach Erhalt einer Gurobi Umgebung (C.3, Zeile 17) wird ein Modell initialisiert (C.3, Zeile 18). Anschließend werden Variablen und (Un-) Gleichungen erstellt und dem Modell hinzugefügt (C.3, Zeile 21-47). Zudem wird das Ziel des Modells festgelegt (Zeile 25). Anschließend wird das Modell optimiert (C.3, Zeile 49) und die Belegung des gefundenen Optimums ausgelesen (C.3, Zeile 56) und zurückgegeben (C.3, Zeile 63). Innerhalb dieser Methode wird mehrfach die Klasse GurobiTools aufgerufen (C.3, Zeile 30/40/46). Die Implementation befindet sich im Anhang C.2. Diese Klasse beinhaltet Methoden die für eine bessere Übersicht im GurobiLoadBalancer sorgen und die Testbarkeit einer zentralen Stelle verbessern. Der in Abschnitt 6.3.1 angesprochene Bedarf von mehreren Variablen, um Teillast der Fahrzeuge abbilden zu können, wird durch die Methode rangeMinMaxOrZero (C.2, Zeile 23) abgedeckt. Die Methode sumHasToCompareTo (C.2, Zeile 39/54) sorgt dafür, dass eine Variable oder ein Wert mit der Summe von einer Liste von Variablen verglichen wird.

## 6.5 Visualisierung

Das Resultat einer Optimierung ist durch die Menge an Zahlen für einen Menschen nur schwer zu verstehen, geschweige denn zu kontrollieren. Eine visuelle Aufbereitung der Daten erleichtert den Umgang mit dem Ergebnis. Dafür wurde im Frontend eine Ansicht erstellt, die die Daten in einem Liniendiagramm darstellt. Für die Umsetzung

wurde die Javascript Bibliothek ChartJS[8] eingesetzt. Die Bibliothek lässt sich mithilfe des verwendeten Frontend Frameworks einfach hinzufügen und sorgt für eine ansprechende Darstellung. Das Diagramm zeigt die Ladegeschwindigkeit (Y-Achse) in Abhängigkeit von der Zeit (X-Achse). Jede Linie repräsentiert ein Fahrzeug. Zusätzlich gibt es eine weitere Linie, die die Summe aller Fahrzeuge, die Gesamtlast, abbildet. An dieser lässt sich die Lastspitze für den abgebildeten Zeitraum ablesen.

[8] https://www.chartjs.org/

# 7 Evaluation

Nach der Umsetzung des Prototypen folgt die Evaluation. Dazu wird eine Bewertung in Bezug auf die Erfüllung der Anforderungen aus Kapitel 4 vorgenommen.

| **User Story** | **Implementiert** |
|---|---|
| Ein Benutzer möchte ein Depot anlegen | Ja |
| Ein Benutzer möchte ein Depot verändern | Ja |
| Ein Benutzer möchte ein Depot löschen | Ja |
| Ein Benutzer möchte ein Fahrzeug anlegen | Ja |
| Ein Benutzer möchte ein Fahrzeug verändern | Ja |
| Ein Benutzer möchte ein Fahrzeug löschen | Ja |
| Ein Benutzer möchte eine geplante Ladung anlegen | Ja |
| Ein Benutzer möchte ein geplante Ladung verändern | Ja |
| Ein Benutzer möchte ein geplante Ladung löschen | Ja |
| Ein Benutzer möchte eine Berechnung für ein Depot starten | Ja |
| Ein Benutzer möchte eine Optimale Ladegeschwindigkeit anfragen | Ja |
| Ein Benutzer möchte eine Graphische Auswertung anfragen | Ja |
| Ein Microservice möchte ein Depot anlegen | Nein |
| Ein Microservice möchte ein Depot verändern | Nein |
| Ein Microservice möchte ein Depot löschen | Nein |
| Ein Microservice möchte ein Fahrzeug anlegen | Nein |
| Ein Microservice möchte ein Fahrzeug verändern | Nein |
| Ein Microservice möchte ein Fahrzeug löschen | Nein |
| Ein Microservice möchte eine geplante Ladung anlegen | Nein |
| Ein Microservice möchte ein geplante Ladung verändern | Nein |
| Ein Microservice möchte ein geplante Ladung löschen | Nein |
| Ein Microservice möchte eine Optimale Ladegeschwindigkeit anfragen | Nein |

Tabelle 7.1: Überblick über funktionale Anforderungen

Die Evaluation beginnt mit den funktionalen Anforderungen die in Abbildung 4.1 zusammengefasst wurden. Die Anforderungen und der Stand der Umsetzung werden in Tabelle 7.1 aufgeführt. Aus der Übersicht wird ersichtlich, dass die User Stories für den Benutzer ausnahmslos umgesetzt wurden, die User Stories für Microservices hingegen nicht. Diese

Umstände sorgen dafür, dass der Prototyp vollumfänglich dafür genutzt werden kann die Optimierung zu testen und zu visualisieren. Die Integration in die Microservice Landschaft ist nicht ohne Weiteres möglich. Da die Integration des entwickelten Services von der übrigen Umgebung noch nicht vorbereitet wurde, ist noch nicht abschließend geklärt, wie die Kommunikation und die Handhabe der Daten ablaufen soll. Dies sorgt in der Liste dafür, dass es so wirkt als sei noch ein Großteil der Entwicklung ausstehend. Dieser Eindruck täuscht. Jede User Story des Microservice wurde bereits in Form einer Rest-API umgesetzt, um mit dem Frontend kommunizieren zu können. Das Konzept sieht vor, dass die Kommunikation über einen Messanging Dienst abgewickelt wird. Würde die Planung der Integration finalisiert werden, könnten die bestehenden Schnittstellen einfach adaptiert werden. Der Aufwand dafür wäre im Verhältnis zum Ganzen minimal.

Neben den zahlreichen technischen und fachlichen Anforderungen gab es eine Anforderung an die Leistungsfähigkeit (QA03) des Service. Um die Erfüllung dieser Anforderung prüfen zu können, wurde der Service entsprechend getestet. Der Prozessor der für die Tests genutzt wurde war nicht besonders leistungsstark (Intel i5-4690, Einführung 2014). Mithilfe eines Python-Scriptes (Anhang C.4) wurde die Erstellung von Szenarien automatisiert. Die Ergebnisse der Tests können aus Tabelle 7.2 entnommen werden.

| **Anzahl Ladevorgänge** | **Berechnungzeitraum** | **Anpassungen** | **Dauer** |
|---|---|---|---|
| 10 | 24 h | 4 pro h | 0,01 sec |
| 100 | 24 h | 4 pro h | 0,04 sec |
| 100 | 24 h | 12 pro h | 0,46 sec |
| 1000 | 24 h | 12 pro h | 50 sec |

Tabelle 7.2: Lasttest

Die Anforderung war, dass die Berechnung bei 100 Fahrzeugen nicht länger als 10 Sekunden dauert. Die Tests haben gezeigt, dass diese Anforderung erfüllt wird. Die Tests haben aber auch gezeigt, dass bei steigender Komplexität die Berechnungszeit signifikant steigt. Wie in Abschnitt 3.1 beschrieben, ist der Ursprung des Konzepts aus einer Partnerschaft mit einem Betriebshof entstanden. Aktuell werden dort 70 Fahrzeuge geladen. Die Leistung der Anwendung ist dafür ausreichend. Sollte das Modell in Zukunft komplexer oder umfangreicher werden, gibt es einige Faktoren die beeinflusst werden können, um die nötige Leistung zu erbringen. Neben der Nutzung stärkerer Hardware, bietet Gurobi auch die Möglichkeit mehrere Rechner gleichzeitig an einem Modell arbeiten zu lassen. Zudem haben die grafischen Auswertungen der Lasttest gezeigt, dass

es keinen signifikanten Mehrwert bietet, wenn die Anzahl der Anpassungen pro Stunde erhöht werden.

# 8 Schluss

Als Abschluss der Arbeit wird ein persönliches Fazit gezogen und ein Ausblick gewährt.

## 8.1 Fazit

Das Ergebnis der Arbeit stellt eine Prototyp zur Optimierung der Ladevorgänge für Elektrofahrzeuge dar. Neben der Konzeption des Prototypen wurden bereits Vorkehrungen zur Integration in eine bestehende Microservice Landschaft geliefert.

Der Aufwand der betrieben werden musste, bis die Nutzung von Gurobi möglich war, wurde stark unterschätzt. Bei vielen gängigen Frameworks und Bibliotheken gibt es zahlreiche Anleitungen und Tutorials an denen sich der Entwickler orientieren kann. Bei dieser Art von Software gibt es nur wenige solcher Angebote. Die Eigenart von Gurobi und der Mangel an Beispielmaterial haben dafür gesorgt, dass das Aufsetzen des Services sowie der eigenen Entwicklungsumgebung deutliche zeitintensiver waren als geplant.

Diese Arbeit basiert auf einem Spagat, dass die zu entwickelnde Anwendung auf der grünen Wiese entworfen werden kann und sie sich in eine bereits bestehende Microservice Landschaft integrieren soll, die zeitgleich aber noch nicht die entsprechenden Services bereitstellt, die final mit der zu entwickelnden Anwendung kommunizieren sollen. Dies hat dazu geführt, dass es dem entworfenen Konzept zu Beginn an Schärfe mangelte und es während der Umsetzung immer stärker hinterfragt wurde. Die geplante Umsetzung der Kommunikation via Kafka wurde daher nicht umgesetzt. Bevor dies geschehen kann, müssen die umliegenden Services konzipiert werden.

Besonders erfreulich war das Resultat der Arbeit am Frontend. Das Framework NextJS erwies sich als sehr nützlich. Das Ergebnis ist ein funktionales Frontend welches auf einem gut strukturiertem Quellcode basiert. Die einfache Integration von weiteren Software-Paketen war sehr hilfreich. So war die grafische Aufbereitung der Daten keine Große Herausforderung, nachdem das entsprechende Paket von ChartJS eingebunden wurde.

Das Ergebnis der Optimierung ist sehr zufriedenstellend. Zuvor war schwer abzuschätzen, wie das Resultat einer Optimierung aussehen könnte. Doch nach dem Testen der Anwendung war festzustellen, dass bei der Untersuchung der für sinnvoll erachteten Szenarien, nach der Optimierung die Kurve der Gesamtlast nahezu horizontal verlief. Ein besseres Ergebnis ist mit dem aktuellen Modell nicht erwartbar.

Der Titel dieser Arbeit ist rückblickend betrachtet nicht ideal gewählt. Durch diese Arbeit wurde kein Algorithmus entwickelt. Viel mehr wurde ein Modell erstellt, dass durch einen bereits existierenden Algorithmus optimiert wird. Ein weiterer signifikanter Teil der Arbeit beschäftigte sich mit der Datenhaltung und den Kommunikationswegen. Dies findet nicht ausreichend Repräsentation im Titel. Dieser Makel zeigt, wie steil die Lernkurve war, wie wenig Wissen über die Lösung eines solchen mathematischen Problems vorlag und wie aus einer groben Idee eine Anwendung wurde.

## 8.2 Ausblick

Mit der Fertigstellung des Prototypen gibt es eine erste Bestätigung der Umsetzbarkeit und einen Überblick über den damit verbunden Aufwand. Die SNH kann zusammen mit ihren Partnern evaluieren ob und wie der Prototyp eingesetzt werden kann. Die Partner müssen in der Lage sein, die Planung der Ladevorgänge vorzunehmen und diese Informationen an die Anwendung zu senden. Sollte den Partnern keine geeignete Softwarelösung zur Verfügung stehen, müsste in Betracht gezogen werden, einen solchen Service zu entwickeln.

Unabhängig von den Herausforderungen der Partner, muss die SNH für eine produktive Nutzung der Anwendung die bereits bestehende Microservice Landschaft anpassen bzw. erweitern. Es muss ein klares Konzept über die Datenhaltung und interne Kommunikation erstellt werden.

Ein weiterer Aspekt, der im Laufe der Zeit an Bedeutung gewinnen wird, ist die Berücksichtigung der Strompreise an der Börse. Dadurch das die Politik der Bundesrepublik Deutschland aktuell die Nutzung volatiler Energiequellen (Wind und Sonne) priorisiert, ist eine direkte Weitergabe des volatilen Strompreis an den Konsumenten ein geeignetes Mittel um den Verbraucher dazu zu bringen, dann Strom zu nutzen, wenn ein Energieüberschuss herrscht. Dieser Aspekt findet im aktuellen Modell keine Berücksichtigung.

In dem Moment, in dem ein monetärer Vorteil erzielbar wäre, sollte eine Erweiterung des Modells in Betracht gezogen werden.

# Literaturverzeichnis


Docker, Inc. (2022). *Develop with docker.* Zugriff am 28.02.2022 auf https://docs.docker.com/develop/

Ferreira, D. R. (2013). *Enterprise systems integration - a process-oriented approach.* Berlin Heidelberg: Springer Science and Business Media.

Gackenheimer, C. (2015). *Introduction to react.* Apress. Zugriff auf https://doi.org/10.1007/978-1-4842-1245-5 doi: 10.1007/978-1-4842-1245-5

h2database.com. (2021). *H2 - performance comparison.* Zugriff am 11.03.2022 auf http://www.h2database.com/html/performance.html

Hows, D., Membrey, P. & Plugge, E. (2014). *MongoDB basics.* Apress. Zugriff auf https://doi.org/10.1007/978-1-4842-0895-3 doi: 10.1007/978-1-4842-0895-3

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J. & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, *35* (3), 24-35. doi: 10.1109/MS.2018.2141039

Jeddi, S. & Sitzmann, A. (2019, Dezember). Netzentgeltsystematik in deutschland – status-quo, alternativen und europäische erfahrungen. *Zeitschrift für Energiewirtschaft*, *43* (4), 245–267. Zugriff auf https://doi.org/10.1007/s12398-019-00265-6 doi: 10.1007/s12398-019-00265-6

Kagermann, H. (2017). Die mobilitätswende: Die zukunft der mobilität ist elektrisch, vernetzt und automatisiert. In A. Hildebrandt & W. Landhäußer (Hrsg.), *Csr und digitalisierung: Der digitale wandel als chance und herausforderung für wirtschaft und gesellschaft* (S. 357–371). Berlin, Heidelberg: Springer Berlin Heidelberg. Zugriff auf https://doi.org/10.1007/978-3-662-53202-7_27 doi: 10.1007/978-3-662-53202-7_27

Kistner, J. (2020). It-basierte batteriespeichersysteme in der anwendung für industrie und infrastruktur. In O. D. Doleski (Hrsg.), *Realisierung utility 4.0 band 1: Praxis der digitalen energiewirtschaft von den grundlagen bis zur verteilung im smart grid* (S. 423–437). Wiesbaden: Springer Fachmedien Wiesbaden. Zugriff auf https://

doi.org/10.1007/978-3-658-25332-5_25 doi: 10.1007/978-3-658-25332-5_25

Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Software*, *12* (6), 42-50. doi: 10.1109/52.469759

Loukides, M. (2020). *Microservices adoption in 2020.* Zugriff am 10.02.2022 auf https://www.oreilly.com/radar/microservices-adoption-in-2020/

Owens, T. (2014). *Mapping between 4+1 architectural view model and uml.* Zugriff am 10.03.2022 auf https://softwareengineering.stackexchange.com/a/233307

Patni, S. (2017). *Pro restful apis - design, build and integrate with rest, json, xml and jax-rs.* New York: Apress.

Rad, B. B., Bhatti, H. J. & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, *17* (3), 228. Zugriff auf https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance

Rheinische NETZGesellschaft mbH. (2022). *Netznutzungsentgelte strom 2022.* Zugriff am 05.03.2022 auf https://www.rng.de/cms/netzentgelte_strom.html?file=tl_files/rng/downloads/Strom/Netzentgelte/Preisblaetter/Netzentgelte%20Strom%20ab%20dem%2001.01.2022.pdf

Rupp, C. . (2021). *Requirements-engineering und -management das handbuch für anforderungen in jeder situation* (7., aktualisierte und erweiterte Auflage Aufl.; C. H. Verlag, Hrsg.). Hanser. Zugriff auf https://dx.doi.org/10.3139/9783446464308

Schumacher, I. (2015). *Strategien zur strombeschaffung in unternehmen.* Springer Fachmedien Wiesbaden. Zugriff auf https://doi.org/10.1007/978-3-658-07422-7 doi: 10.1007/978-3-658-07422-7

solid IT gmbh. (2022). *Db-engines ranking of document stores.* Zugriff am 14.03.2022 auf https://db-engines.com/en/ranking/document+store

Steiner, R. (2014). *Grundkurs relationale datenbanken.* Springer Fachmedien Wiesbaden. Zugriff auf https://doi.org/10.1007/978-3-658-04287-5 doi: 10.1007/978-3-658-04287-5

Stromnetz Hamburg GmbH. (2022). *Mitarbeiteranzahl der stromnetz hamburg gmbh.* Zugriff am 08.03.2022 auf https://www.stromnetz-hamburg.de/ueber-uns/

unternehmen/arbeitgeber

Suhl, L. & Mellouli, T. (2013). *Optimierungssysteme*. Springer Berlin Heidelberg. Zugriff auf https://doi.org/10.1007/978-3-642-38937-5 doi: 10.1007/978-3-642-38937-5

Vaidya, B. & Mouftah, H. T. (2018). Deployment of secure ev charging system using open charge point protocol. In *2018 14th international wireless communications mobile computing conference (iwcmc)* (S. 922-927). doi: 10.1109/IWCMC.2018.8450489

# A Beispiele

## A.1 JSON Objekt vor der Optimierung


```
{
  "id": 1,
  "minChargingSpeed_W": 0,
  "maxChargingSpeed_W": 3000,
  "timeslots_p_H": 4,
  "calculationLimit_H": 24,
  "chargeList": [
    {
      "id": 3,
      "arrival": "2022-03-01T15:52:00",
      "departure": "2022-03-02T01:52:00",
      "chargeOnArrival_Percent": 20,
      "chargeOnDeparture_Percent": 90,
      "capacity_Wh": 3000,
      "minChargingSpeed_W": 200,
      "maxChargingSpeed_W": 500
    },
    {
      "id": 1,
      "arrival": "2022-03-01T15:09:00",
      "departure": "2022-03-02T07:09:00",
      "chargeOnArrival_Percent": 10,
      "chargeOnDeparture_Percent": 95,
      "capacity_Wh": 1000,
      "minChargingSpeed_W": 50,
      "maxChargingSpeed_W": 200
```

```
    }
  ]
}
```

## A.2 JSON Objekt nach der Optimierung

```
{
  "id": 1,
  "speedList": [
    {
      "id": 1,
      "fromDateTime": "2022-03-01T14:30:00",
      "toDateTime": "2022-03-01T14:45:00",
      "chargingSpeed_W": 0
    },
    {
      "id": 2,
      "fromDateTime": "2022-03-01T14:30:00",
      "toDateTime": "2022-03-01T14:45:00",
      "chargingSpeed_W": 0
    },
    {
      "id": 3,
      "fromDateTime": "2022-03-01T14:30:00",
      "toDateTime": "2022-03-01T14:45:00",
      "chargingSpeed_W": 0
    },
    {
      "id": 4,
      "fromDateTime": "2022-03-01T14:30:00",
      "toDateTime": "2022-03-01T14:45:00",
      "chargingSpeed_W": 300
    }
  ]
}
```

# B Konfiguration

## B.1 Dockerfile

```
FROM maven:3.6.3-jdk-11 as build
WORKDIR /app
COPY . .
RUN mvn -f pom.xml clean package -DskipTests

FROM gurobi/optimizer:9.1.1

RUN apt-get update && \
    apt-get install -y openjdk-11-jdk ca-certificates-java && \
    apt-get clean && \
    update-ca-certificates -f
ENV JAVA_HOME /usr/lib/jvm/java-11-openjdk-amd64/
RUN export JAVA_HOME


COPY --from=build /app/target/*.jar app.jar
ENTRYPOINT ["java","-jar","app.jar"]
EXPOSE 9000
```

## B.2 Docker-Compose

```yaml
version: '3'

services:

  gurobi:
    container_name: gurobi-compute
    image: gurobi/compute:9.5.0
    restart: always
    ports:
      - "61000:61000"
    command: --hostname=gurobi
    volumes:
      - ./gurobi.lic:/opt/gurobi/gurobi.lic:ro

  # Electric Charging Load Balancer
  eclb:
    container_name: eclb
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "9000:9000"
    environment:
      - GUROBI_SERVER=gurobi:61000
      - SERVER_PORT=9000
    depends_on:
      - gurobi

  # Electric Charging Load Balancer User Interface
  nextjs-ui:
    build:
      context: ./eclb-frontend
    ports:
      - "3000:3000"
```

```
35      container_name: eclb-fe
36      stdin_open: true
37      volumes:
38        - ./my-app:/usr/src/app/my-app
39        - /usr/src/app/my-app/node_modules
```

## B.3 Maven - pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
       ↪ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       ↪ https://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.5.5</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <properties>
12         <java.version>11</java.version>
13     </properties>
14
15     <repositories>
16         <repository>
17             <id>jena</id>
18             <name>Jena Bio Repository</name>
19             <url>https://bio.informatik.uni-jena.de/repository/libs-release-oss/
20         </repository>
21     </repositories>
22
23     <dependencies>
24         <dependency>
25             <groupId>org.springframework.boot</groupId>
```

```
26          <artifactId>spring-boot-starter</artifactId>
27      </dependency>
28
29      <dependency>
30          <groupId>org.modelmapper</groupId>
31          <artifactId>modelmapper</artifactId>
32          <version>3.0.0</version>
33      </dependency>
34
35      <dependency>
36          <groupId>org.springdoc</groupId>
37          <artifactId>springdoc-openapi-ui</artifactId>
38          <version>1.5.9</version>
39      </dependency>
40
41      <dependency>
42          <groupId>com.h2database</groupId>
43          <artifactId>h2</artifactId>
44          <scope>runtime</scope>
45      </dependency>
46
47      <dependency>
48          <groupId>org.springframework.boot</groupId>
49          <artifactId>spring-boot-starter-test</artifactId>
50          <scope>test</scope>
51      </dependency>
52
53      <dependency>
54          <groupId>org.projectlombok</groupId>
55          <artifactId>lombok</artifactId>
56          <optional>true</optional>
57      </dependency>
58
59      <dependency>
60          <groupId>org.apache.maven.plugins</groupId>
61          <artifactId>maven-assembly-plugin</artifactId>
```

```
62          <version>2.5</version>
63          <type>maven-plugin</type>
64       </dependency>
65
66       <dependency>
67          <groupId>com.gurobi</groupId>
68          <artifactId>gurobi-jar</artifactId>
69          <version>9.1.1</version>
70       </dependency>
71       <dependency>
72          <groupId>org.springframework.boot</groupId>
73          <artifactId>spring-boot-starter-web</artifactId>
74       </dependency>
75       <dependency>
76          <groupId>org.springframework.boot</groupId>
77          <artifactId>spring-boot-starter-data-jpa</artifactId>
78       </dependency>
79
80    </dependencies>
81
82    <build>
83       <plugins>
84          <plugin>
85             <groupId>org.springframework.boot</groupId>
86             <artifactId>spring-boot-maven-plugin</artifactId>
87             <configuration>
88                <excludes>
89                   <exclude>
90                      <groupId>org.projectlombok</groupId>
91                      <artifactId>lombok</artifactId>
92                   </exclude>
93                </excludes>
94             </configuration>
95          </plugin>
96          <plugin>
97             <artifactId>maven-assembly-plugin</artifactId>
```

```
98              <configuration>
99                 <archive>
100                   <manifest>
101                      <mainClass>fully.qualified.MainClass</mainClass>
102                   </manifest>
103                </archive>
104                <descriptorRefs>
105                   <descriptorRef>jar-with-dependencies</descriptorRef>
106                </descriptorRefs>
107             </configuration>
108          </plugin>
109       </plugins>
110    </build>
111
112 </project>
```

# C Quellcode

## C.1 OpenAPI Konfiguration

```
1
2  @Configuration
3  public class OpenApi {
4
5      @Bean
6      public OpenAPI openAPI() {
7          return new OpenAPI()
8                  .addServersItem(new Server().url("/"));
9      }
10
11 }
```

## C.2 GurobiTools

```
1  /**
2   * GUROBI TOOLBOX
3   * TO MAKE CODE EASIER
4   */
5  public class GurobiTools {
6
7      /**
8       * Creates Variable that can withhold a value between a
            ↪ range or zero
9       *
10      * @param model in which the variable is used
11      * @param min minimum value apart from zero
12      * @param max maximum value
13      * @param name description
14      * @return variable for further usage
15      * @throws GRBException
16      */
17     public static GRBVar rangeMinMaxOrZero(GRBModel model,
            ↪ Double min, Double max, String name) throws
            ↪ GRBException {
18         GRBVar res = model.addVar(0, max, 0.0, GRB.INTEGER,
               ↪ name);
19         GRBVar x1 = model.addVar(min, max, 0d, GRB.INTEGER, name
               ↪ + " x1");
20         GRBVar xMax = model.addVar(0d, 1d, 0d, GRB.INTEGER, name
               ↪ + "xMax");
21
22         GRBLinExpr expr = new GRBLinExpr();
23         expr.addTerm(1.0, x1);
24         expr.addTerm(-max, xMax);
25         model.addConstr(expr, GRB.EQUAL, res, name + "
               ↪ MinMaxZeroExpr");
26
27         return res;
```

```
28      }
29
30      /**
31       * Creates Expression that equals a list of variables to a
            ↪ specific value
32       *
33       * @param model in which the variable is used
34       * @param vars that need to be added up
35       * @param value the comparing value
36       * @param name description
37       * @throws GRBException
38       */
39      public static void sumHasToCompareTo(GRBModel model,
            ↪ Set<GRBVar> vars, char grb, Double value, String name)
            ↪ throws GRBException {
40          checkGRBChar(grb);
41          GRBLinExpr sum = createExprFromVars(vars);
42          model.addConstr(sum, grb, value, name + "sum");
43      }
44
45      /**
46       * Creates Expression that equals a list of variables to a
            ↪ variable
47       *
48       * @param model in which the variable is used
49       * @param vars that need to be added up
50       * @param var the comparing variable
51       * @param name description
52       * @throws GRBException
53       */
54      public static void sumHasToCompareTo(GRBModel model,
            ↪ Set<GRBVar> vars, char grb, GRBVar var, String name)
            ↪ throws GRBException {
55          checkGRBChar(grb);
56          GRBLinExpr sum = createExprFromVars(vars);
57          model.addConstr(sum, grb, var, name + "sum");
```

```
58      }
59
60      private static void checkGRBChar(char grbchar) throws
            ↪ GRBException {
61          if (grbchar != GRB.EQUAL && grbchar != GRB.LESS_EQUAL &&
                ↪ grbchar != GRB.GREATER_EQUAL) {
62              throw new GRBException("Invalid GRB char");
63          }
64      }
65
66      private static GRBLinExpr createExprFromVars(Set<GRBVar>
            ↪ vars) throws GRBException {
67          if (vars.size() == 0) {
68              throw new GRBException("List can not be empty");
69          }
70          GRBLinExpr expr = new GRBLinExpr();
71          for (GRBVar v : vars) {
72              expr.addTerm(1.0, v);
73          }
74          return expr;
75      }
76
77  }
```

## C.3 GurobiLoadBalancer

```
1  @Component
2  public class GurobiLoadBalancer<T> implements
       ↪ LoadBalancerSolver<T> {
3
4      private final GurobiAdministrationService
           ↪ gurobiAdministrationService;
5
6      public GurobiLoadBalancer(GurobiAdministrationService
           ↪ gurobiAdministrationService) {
7          this.gurobiAdministrationService =
               ↪ gurobiAdministrationService;
8      }
9
10     @SneakyThrows
11     @Override
12     public Matrix<Integer, T, Double> solve(ConsumerFrame<T>
           ↪ consumerFrame) {
13         Set<Integer> timeSlots = IntStream.rangeClosed(1,
               ↪ consumerFrame.getTimeslots()).boxed().collect(Collectors.toSet());
14         Set<T> consumers =
               ↪ consumerFrame.getConsumers().stream().map(Consumer::getIdentifier)
15         Matrix<Integer, T, GRBVar> matrix = new
               ↪ Matrix(timeSlots, consumers, GRBVar.class);
16
17         GRBEnv env =
               ↪ gurobiAdministrationService.getEnvironment();
18         GRBModel model = new GRBModel(env);
19
20         // max represents the highest total charging speed for
               ↪ one timeslot
21         GRBVar max = model.addVar(0, Double.MAX_VALUE, 0.0,
               ↪ GRB.INTEGER, "Max");
22         GRBLinExpr maxExpr = new GRBLinExpr();
23         maxExpr.addTerm(1.0, max);
```

```
24          // the goal is to minimize the max value
25          model.setObjective(maxExpr, GRB.MINIMIZE);
26
27          for (Integer ts : timeSlots) {
28              for (Consumer<T> consumer :
                    ↪ consumerFrame.getConsumers()) {
29                  if (consumer.getFirstTimeSlot() <= ts &&
                        ↪ consumer.getLastTimeSlot() >= ts) {
30                      GRBVar grbVar = GurobiTools.rangeMinMaxOrZero(
31                              model,
32                              (double)
                                    ↪ consumer.getMinVolumePerTimeSlot(),
33                              (double)
                                    ↪ consumer.getMaxVolumePerTimeSlot(),
34                              ts + " " + consumer.getIdentifier());
35                      matrix.set(ts, consumer.getIdentifier(),
                            ↪ grbVar);
36                  }
37              }
38              Set<GRBVar> allForTimeSlot =
                    ↪ matrix.getAllByR(ts).stream().filter(Objects::nonNull).collect(
39              if (allForTimeSlot.size() > 0) {
40                  GurobiTools.sumHasToCompareTo(model,
                        ↪ allForTimeSlot, GRB.LESS_EQUAL, max, "MAX TS"
                        ↪ + ts);
41              }
42          }
43
44          for (Consumer<T> consumer :
                ↪ consumerFrame.getConsumers()) {
45              Set<GRBVar> allForConsumer =
                    ↪ matrix.getAllByS(consumer.getIdentifier()).stream().filter(Obje
46              GurobiTools.sumHasToCompareTo(model, allForConsumer,
                    ↪ GRB.EQUAL, (double) consumer.getGoal(), "SUM" +
                    ↪ consumer.getIdentifier());
47          }
```

```
48
49          model.optimize();
50
51          Matrix<Integer, T, Double> result = new
               ↪ Matrix<>(timeSlots, consumers, Double.class);
52
53          for (Integer ts : timeSlots) {
54              for (T consumer : consumers) {
55                  GRBVar grbVar = matrix.get(ts, consumer);
56                  Double speed = grbVar == null ? 0d :
                       ↪ grbVar.get(GRB.DoubleAttr.X);
57                  result.set(ts, consumer, speed);
58              }
59          }
60
61          model.dispose();
62
63          return result;
64      }
65
66  }
```

## C.4 Test Setup Tool

```
import requests
import random
from datetime import datetime, timedelta, date
from json import dumps

def json_serial(obj):
    if isinstance(obj, (datetime, date)):
        return obj.isoformat()
    raise TypeError ("Type %s not serializable" % type(obj))


r = lambda: random.randint(0,255)
t = lambda: random.randint(0,24)
s = lambda: random.randint(6,18)



for x in range(100):
    color = '#{:02x}{:02x}{:02x}'.format(r(), r(), r())
    v_response =
        ↪ requests.post('http://localhost:9000/vehicle/create',
        ↪ json = {
        "name": "test"+str(x),
        "capacity_Wh": 50000,
        "minChargingSpeed_W": 1000,
        "maxChargingSpeed_W": 10000,
        "color": color
        }
    )
    res = v_response.json()
    v_id = res.get('id')

    arrival = datetime.now() + timedelta(hours=t())
    departure = arrival + timedelta(hours=s())
```

```
33
34      p_response =
            ↪ requests.post('http://localhost:9000/plannedcharge/create',
            ↪ json = {
35          "depotId": 3,
36          "vehicleId": v_id,
37          "arrival": dumps(arrival,
                ↪ default=json_serial).strip('"'),
38          "departure": dumps(departure,
                ↪ default=json_serial).strip('"'),
39          "chargeOnArrival_Percent": 10,
40          "chargeOnDeparture_Percent": 90
41          }
42      )
```

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| Ort | Datum | Unterschrift im Original |
|---|---|---|