hochschule mannheim

OBJECTIVE PARTNER

# Ligenium Track+

## Architekturdokument

**Agile Alligators**

Jannis Seefeld
Baris Kacmaz
Timo Scheuermann

**Version**

2.0

**Zuletzt geändert am**

30.06.2022

Basierend auf dem arc42 Template

## Projektrahmen

### Kunde

objetive partner AG (i. A. von ligenium GmbH)

### Kundenkontakt (E-Mail)

objective partner
*michael.thron@objective-partner.com*
*marian.wensky@objective-partner.com*

ligenium
*angela.grimmer@ligenium.de*
*christoph.alt@ligenium.de*

### Produktname

Ligenium Track+

### Management

Prof. Dr. Peter Knauber
*p.knauber@hs-mannheim.de*

Prof. Kirstin Kohler
*k.kohler@hs-mannheim.de*

# Inhaltsverzeichnis

# Tabellenverzeichnis

# Abbildungsverzeichnis

## Abkürzungsverzeichnis

| | |
|---|---|
| **CBA** | Component Based Architecture |
| **DTO** | Datentransferobjekt |
| **JSON** | JavaScript Object Notation |
| **LT+** | Ligenium Track+ |
| **LT** | Ladungsträger |
| **MVC** | Model-View-Controller |
| **OWASP** | Open Web Application Security Project |

# 1 Einführung und Ziele

## 1.1 Aufgabenstellung

### 1.1.1 Architekturtreiber

Mit dem Wandel zur Industrie 4.0 hat sich die Produktion hinsichtlich Automatisierung und Individualisierung stark verändert. Es wird vom Kunden erwartet, dass Aufträge flexibler und in weniger Zeit abgearbeitet werden können. Dazu benötigen Firmen die Möglichkeit, ihre Produktion spontan und kostengünstig überwachen und steuern zu können. Um dies umzusetzen, sind in den vergangenen Jahren viele sogenannte „X as a Service“-Modelle entstanden, die den Unternehmen die Möglichkeit bieten, ihr Kapital dynamischer zu binden. Auch Ligenium möchte von diesem Trend profitieren und ihre Ladungsträger als Pay-per-Use (Asset as a Service)- Modell anbieten. Hierbei kann Ligenium ihren Kunden einen Mehrwert durch das Sammeln und Auswerten von Sensordaten bieten und kann sich über dieses Alleinstellungsmerkmal am Markt behaupten.

### 1.1.2 Fachliche Aufgabenstellung

Ligenium stellt Ladungsträger her, mit denen ihre Kunden Bauteile sicher und flexibel an den Ort der Verarbeitung transportieren können. Um den Kunden hier einen Mehrwert zu bieten, sollen die Ladungsträger mit diversen Sensoren ausgestattet werden, die beispielsweise Daten zu Position, Erschütterung und Geschwindigkeit sammeln und in einer sogenannten Verwaltungsschale persistieren können. Im Folgenden soll eine Anwendung beschrieben werden, die dem Kunden dabei hilft, die Ladungsträger zu überwachen und mithilfe von Datenauswertung. die Nutzung der Ladungsträger effizienter zu gestalten.

Im Rahmen des Projekts soll ein Demonstrator entstehen, welcher die Arbeit eines Logistikplaners unterstützt. Dazu soll dem Logistikplaner die Möglichkeit gegeben werden, sich die Ladungsträger nach bestimmten Filteroptionen anzeigen zu lassen. Zu diesen Optionen zählen Ladungsträgerart, Lieferung, Kunde und Bauteil. Nach Auswahl der Ladungsträger, werden dem Logistikplaner die Positions-, Beladungs-, Standzeit- und Erschütterungsdaten angezeigt. Über die Festlegung eines Zeitraums können die Daten weiter spezifiziert und dabei Trends abgelesen werden. Die ermittelten Daten werden dem Logistikplaner aufbereitet dargestellt. Hierbei werden die einzelnen Daten in Relation zu ihrer Position in einzelnen Hotspot-Maps angezeigt. Über Linien- und Säulendiagramme kann die zeitliche Entwicklung der Daten nachvollzogen werden.

## 1.2 Qualitätsziele

### 1.2.1 Zuverlässigkeit

Aufgrund des eingangs beschriebenen Trends zur Automatisierung und der dadurch benötigten Flexibilität, ist es wichtig, den Kunden ein zuverlässiges System zur Überwachung bereitzustellen. Hierzu muss beachtet werden, dass eine Vielzahl an Nutzern gleichzeitig auf das System zugreifen möchten und die Verfügbarkeit zu jeder Zeit

sichergestellt sein muss. Je nach Zahl der Kunden, Zahl derer Ladungsträger und Zahl der daran angebrachten Sensoren und deren Senderate vermehren sich die gesammelten Daten stetig. Das System muss in der Lage sein, diese Datenmenge zuverlässig verarbeiten zu können.

### 1.2.2 Sicherheit

Das System muss in der Lage sein, Mandanten voneinander zu trennen und nur die zum Kunden zugehörigen Daten anzuzeigen.

Außerhalb des Projektumfangs sollte bei einer tatsächlichen Umsetzung auch auf folgende Sicherheitsaspekte Wert gelegt werden:

- Sichere Verschlüsselung der gespeicherten Daten, damit ein Angreifer keinen Nutzen aus den erhaltenen Daten ziehen kann
- Backups sollten in regelmäßigen Zeitabschnitten erstellt werden, um im Falle eines Datenverlusts die Daten wieder herstellen zu können
- Damit die gängigsten Sicherheitslücken und -risiken abgedeckt sind, muss das produktive System gegen die OWASP Top 10[1] jährlich getestet werden.

### 1.2.3 Leistungseffizienz

Im Rahmen dieses Projekts kann auf diesen Punkt kein Augenmerk gelegt werden. Bei einer späteren Implementierung kann Leistungseffizienz durch die Nutzung von beispielsweise Kubernetes-Clustern erreicht werden.

### 1.2.4 Wartbarkeit

Eine weitere Anforderung für Systeme in der Industrie 4.0 ist, dass diese schnell und einfach verändert, angepasst oder verbessert werden können. Deshalb sollen die einzelnen Komponenten des Systems als Microservice implementiert werden. Hierdurch kann einerseits sprachenunabhängig entwickelt werden und andererseits können gezielter und regelmäßiger Änderungen an Teilen des Systems vorgenommen werden.

Da aufgrund der zeitlichen Rahmenbedingung eine Microservicearchitektur zu zeitintensiv wäre, wird der Demonstrator lediglich aus einem System bestehen, welches die einzelnen geplanten Microservices innerhalb von Modulen kapselt.

### 1.2.5 Kompatibilität

Da die Verwaltungsschale von objective partner die zentrale Datenhaltungseinheit bildet, stellt die Kompatibilität einen wichtigen Aspekt innerhalb des Projektes dar. Sollte es hier

[1] *https://owasp.org/Top10/*

aufgrund fehlender Verbindungsoptionen zu einer Inkompatibilität kommen, wird der Demonstrator lediglich auf generierte Daten zugreifen können.

Für eine spätere Phase der Entwicklung, wäre die Anbindung an weitere Systeme wie beispielsweise Jira oder SAP denkbar, um unmittelbare Störmeldungen automatisiert zu verteilen.

## 1.3 Stakeholder

Für den Rahmen des Demonstrators, der innerhalb dieses Semesters entsteht, gehen folgende Stakeholder hervor:

- objective partner: Ist ein von ligenium beauftragter IT-Dienstleister zur Konzeption und Umsetzung des gewünschten Softwareprodukts. Im Rahmen dieses Projekts ist objective partner auf die Hochschule Mannheim zugegangen, um einen Demonstrator für das Projekt zu erstellen. Dabei fungiert objective partner als technischer Ansprechpartner.

- ligenium GmbH: Ist ein Systemanbieter für die Entwicklung und Herstellung leichter, nachhaltiger Fördertechnik und Förderhilfsmittel sowie Komponenten aus Holzwerkstoffen für den Maschinen- und Anlagenbau. Um ihren Marktanteil zu vergrößern, hat ligenium sich dazu entschlossen ihren Kunden Ladungsträger auch als „Asset as a Service“-Modell anzubieten. Dabei möchten sie gleichzeitig die bereits im Einsatz befindlichen Ladungsträger mit Sensoren ausstatten, um dadurch wertvolle Daten zu sammeln, welche dem Kunden helfen können, den Produktionsprozess zu optimieren. Sie dienen als fachlicher Ansprechpartner in diesem Projekt.

- Dozenten der Hochschule Mannheim: Dienen als Betreuer und Organisatoren des Projekts. Sie überwachen und leiten den Entwicklungsprozess des Demonstrators.

Für einen späteren produktiven Einsatz wären folgende Stakeholder sinnvoll genauer zu betrachten und mit in den Entwicklungsprozess aufzunehmen:

- Logistikplaner: Ist diejenige Person, welche nach einem erfolgreichen Launch das Tool nutzen wird, um hilfreiche Informationen zu den Ladungsträgern zu erhalten. Mit diesen Informationen soll er in der Lage sein die Produktion anzupassen und Verbesserungen vorzunehmen.

- SAP: Könnte ein weiterer Stakeholder sein, insofern ligenium sich dazu entscheiden sollte, in der Software eine Schnittstelle zu anderen IT-Systemen bereitzustellen. Um hier eine erfolgreiche Bereitstellung zu gewährleisten, sollte SAP mit ins Projekt aufgenommen werden.

# 2 Randbedingungen

## 2.1 Technische Randbedingungen

Wie im Kapitel „Qualitätsziele > Wartbarkeit“ bereits angedeutet, wird auf eine Microservice-Architektur für den Demonstrator verzichtet. Da mit einer geringeren Menge an Daten zu rechnen ist, wird dadurch die Leistungseffizienz ebenfalls nicht beeinflusst. Der Demonstrator selbst stellt dabei ein monolithisches System dar, welches die einzelnen Module[2] voneinander kapselt. Diese können per Dependency Injection[3] auf Funktionen anderer Module innerhalb des Monolithen zugreifen.

Aufgrund der zeitlich festgelegten Rahmenbedingungen und der unterschiedlichen Kenntnisstände im Bereich der Webentwicklung, wird für den Demonstrator TypeScript verwendet. Das Backend entsteht mithilfe des Frameworks NestJs[4], das Frontend wird mittels Vue.js[5] realisiert und als Datenbank wird auf eine MongoDB[6] zurückgegriffen. Ebenso wird für den Demonstrator nur eine Datenbank verwendet und nicht eine eigene Datenbank pro Microservice, wie für die produktive Umsetzung angedacht.

Die Festlegung der Sprache, Frameworks und Datenbanktechnik gilt lediglich für den Demonstrator. Spätere Entwickler können die Sprache, das Framework und die Datenbanktechnik für die zu schreibenden Microservices selbst auswählen. Durch die gewählte Microservice-Architektur können auch unterschiedliche Sprachen, Frameworks und Datenbanksysteme für die einzelnen Microservices verwendet werden.

Da der Demonstrator selbst nicht in einem produktiven Umfeld zum Einsatz kommt, können die zuvor erwähnten Sicherheitsaspekte außer Acht gelassen werden.

## 2.2 Organisatorische Randbedingungen

Der Demonstrator ist innerhalb des Semesters zu entwerfen und implementieren. Hierbei wird der festgelegte Zeitraum von zwölf Wochen zu jeweils 50% für den Entwurf und 50% für die Entwicklung aufgeteilt. Abbildung 1 zeigt einen groben Zeitplan des Projekts:

---

[2] Die Module des Demonstrators stehen exemplarisch für die einzelnen Microservices, des produktiven Kubernetes-Clusters

[3] Bei einer produktiven Umsetzung, würde Dependency Injection durch REST-Schnittstellen und REST-Calls ersetzt werden.

[4] *https://nestjs.com/*

[5] *https://vuejs.org/*

[6] *https://www.mongodb.com/*


*Abbildung 1: Zeitplan des Demonstrators*

Die Entwicklung selbst ist in vier Releases unterteilt. Dabei soll bereits nach dem ersten Release eine Anwendung entstehen, die den angezielten Workflow abbildet. Durch weitere Releases werden zusätzliche Funktionalitäten zum bestehenden System ergänzt. Abbildung 2 stellt die geplanten Releases und darin enthaltenen Features grob dar:

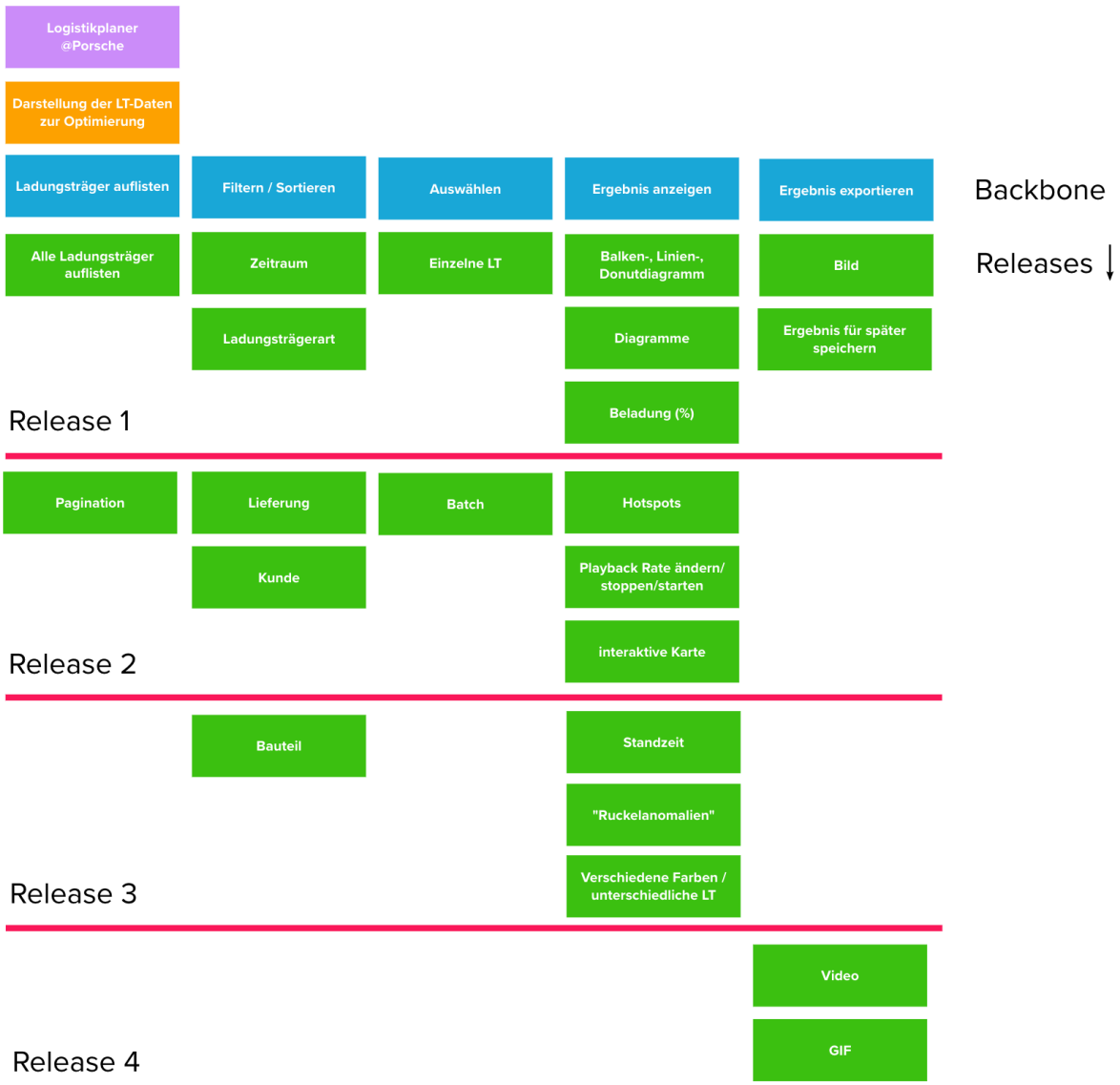


*Abbildung 2: Journey-Map des Demonstrators. Porsche ist hier als illustrierendes Beispiel dargestellt.*

## 2.3 Konventionen

### 2.3.1 Frontend

Innerhalb des Projektteams wurde sich darauf verständigt Ansichten und Komponenten voneinander zu trennen. Dies geschieht einerseits über die Ordnerstruktur und anderseits über die Benennung der Dateien. Hierbei erhalten Dateien für Ansichten kein Präfix, Dateien für Komponenten ein Präfix, welcher sich anhand des Projektnamens ableitet.[7]

### 2.3.2 Backend

Um auch im Backend eine nachvollziehbare Struktur einzuführen, wurden Datentransferobjekte (DTOs), Schemas und Modelle durch separate Ordner voneinander getrennt. Da für den Demonstrator NestJs verwendet wird, können hier mithilfe der Bibliotheken „class-transformer“ und „class-validator“ Validierungsmerkmale festgelegt werden, welche eingehende DTOs automatisch überprüfen. Bei einer späteren Produktiventwicklung muss der Entwickler des jeweiligen Microservices prüfen, ob für die gewählte Entwicklungssprache eine ähnliche Bibliothek vorhanden ist.

Da bei NestJs die Controller für eingehende Anfragen sowie ausgehende Antworten vorgesehen sind, beinhalten diese keine Businesslogik.

Die Dokumentation [8]der Endpunkte erfolgt mittels Swagger[9].

---

[7] Da der Demonstrator innerhalb des Teams unter dem Namen „Agile Alligators“ entwickelt wird, findet hier das Präfix „AA“ Verwendung. Bei einer produktiven Entwicklung kann hier beispielsweise der Produktname „Ligenium Track+ (LT+)“ als Präfix „LT“ verwendet werden.

[8] *https://agilealligators.timos.design/api/*

[9] *https://swagger.io/*

# 3 Kontextabgrenzung

Da der Demonstrator lediglich als ein Datenaufbereitungs- und Darstellungssystem fungiert, ist zum jetzigen Zeitpunkt nur eine Schnittstelle zu einem externen System geplant. Dabei liest der Demonstrator über eine Eventqueue neu erstellte Datensätze aus der Verwaltungsschale aus und bereitet diese für den eigenen Gebrauch auf.

# 4 Lösungsstrategie

## 4.1 Technologieentscheidungen

Die Auswahl der optimalen Technologie für die Entwicklung des Demonstrators ist sehr wichtig. Bei der großen Auswahl an Tools muss die Auswahl eingeschränkt werden, indem die Vor- und Nachteile der Frameworks abgewogen werden. Die festgelegten Kriterien, die die Auswahl unterstützen, sind in Tabelle 1 aufgeführt[10]. Eine sehr große Rolle spielten dabei zum Beispiel die Lernkurve, sowie die Qualität der Dokumentation.

*Tabelle 1: Nutzwertanalyse Frontend-Technologien*

| | Vue | Angular | React | NativeScript | NextJs | Ionic |
|---|---|---|---|---|---|---|
| Anfängerfreundlich/Lernkurve | 5 | 3 | 3 | 2 | 2 | 2 |
| Beliebtheit[11] | 5 | 3 | 4 | 1 | 2 | 1 |
| Architektur | MVC | MVC | CBA | - | - | MVC |
| Dokumentation | 5 | 4 | 4 | 1 | 2 | 2 |

Aus der Analyse geht Vue klar als Frontend-Technologie hervor. Wichtig zu erwähnen ist jedoch, dass sich die Vorauswahl auf JavaScript-Frameworks beschränkt hat, da somit der Austausch von Informationen zwischen Frontend, Backend und Datenbank über ein einheitliches Format (JSON) einfach realisiert werden kann. Deshalb wird für die Datenbank auch MongoDB verwendet.

Bei der Auswahl des Backend-Frameworks wurde ähnlich zur Auswahl der Frontend-Technologie vorgegangen. Auch hier war sowohl die Dokumentation und der Community-Support als auch die Reichweite[12] wichtig. Hierbei ging NestJs führend hervor. Ein weiteres Kriterium war auch die Anzahl der Forks als Indiz für den Verbesserungsbedarf des

[10] Hierbei handelt es sich um eine Nutzwertanalyse, bei der die diversen Frameworks mit einer Punktzahl von 0 (= ungenügend) bis 5 (= sehr gut) bewertet wurden.

[11] Nach Anzahl der Sterne in GitHub (Stand 30. Oktober 2020), da dies eine einfache, aber trotzdem aussagekräftige Metrik für die Bekanntheit und die Wertschätzung eines Projekes ist (Vgl. N. Munaiah, S. Kroh, C. Cabrey et al.: Kuratieren von GitHub für technische Softwareprojekte; Empirical Software Engineering 22, S. 3219–3253; 2017. *https://doi.org/10.1007/s10664-017-9512-6*)

[12] Ein Indiz hierfür war die Anzahl der Watches bei GitHub

Projekts. Im Vergleich zu anderen Frameworks[13] hatte NestJs die kleinste Anzahl, was die gute Qualität des Frameworks verdeutlicht.

Für den Demonstrator wird keine Microservice-Architektur mittels Kubernetes verwendet, da dies den Umfang des Demonstrators überschreiten würde. Die einzelnen Microservices werden über einzelne Module zu einem Monolithen zusammengefasst.

Zu erwähnen ist hierbei jedoch, dass sich dieses Verfahren und die daraus herausgegangene Entscheidung nicht auf ein produktives System übertragen lässt. Hier muss später für jeden einzelnen Microservice eine individuelle Entscheidung getroffen werden, da weitere Aspekte wie der Aufgabenbereich und mögliche Leistungsanforderungen bei der Auswahl mitbetrachtet werden müssen.

## 4.2 Top-Level-Zerlegung des Systems

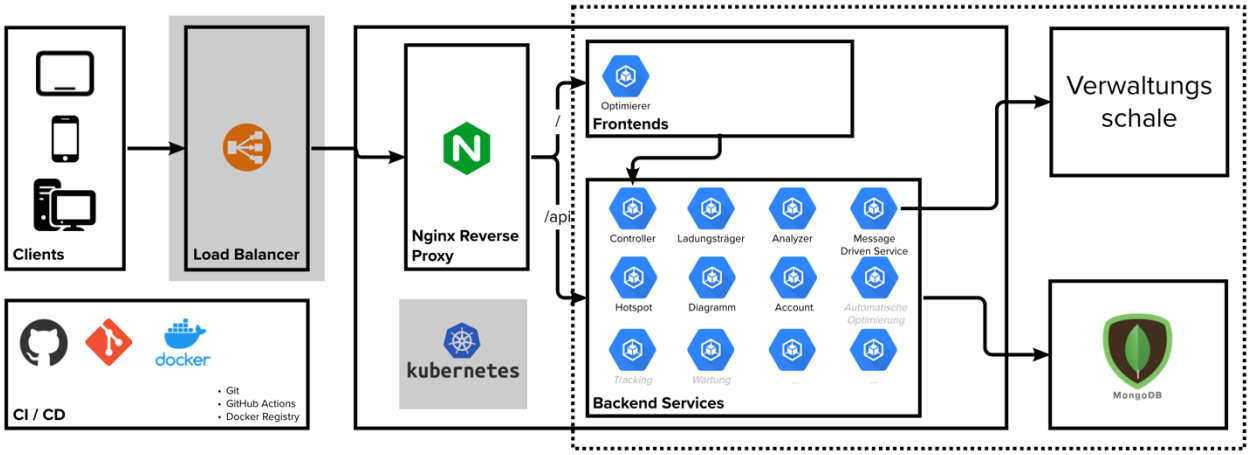


*Abbildung 3: Top-Level-Zerlegung des Systems*

Der Demonstrator ist im Kern ein Monolith und vereint alle benötigten Services/Module. Die ausgegrauten Services stellen hier mögliche Services dar, welche weitere Funktionen für smarte Ladungsträger bereitstellen könnten. Die benötigten Services werden in Kapitel 6 genauer beschrieben.

Innerhalb des Demonstrators werden Daten zwischen den einzelnen Modulen mittels Dependency Injection ausgetauscht. Bei einer produktiven Umsetzung würde dies über REST-Schnittstellen und Sockets geschehen. Diese Kommunikationsverbindungen sind in Abbildung 4 dargestellt:

[13] Vgl. NodeJs (23.600 Forks), Go (14.900 Forks), Express (9.700 Forks), NestJs (5.400 Forks) (Stand 1. Juni 2022)

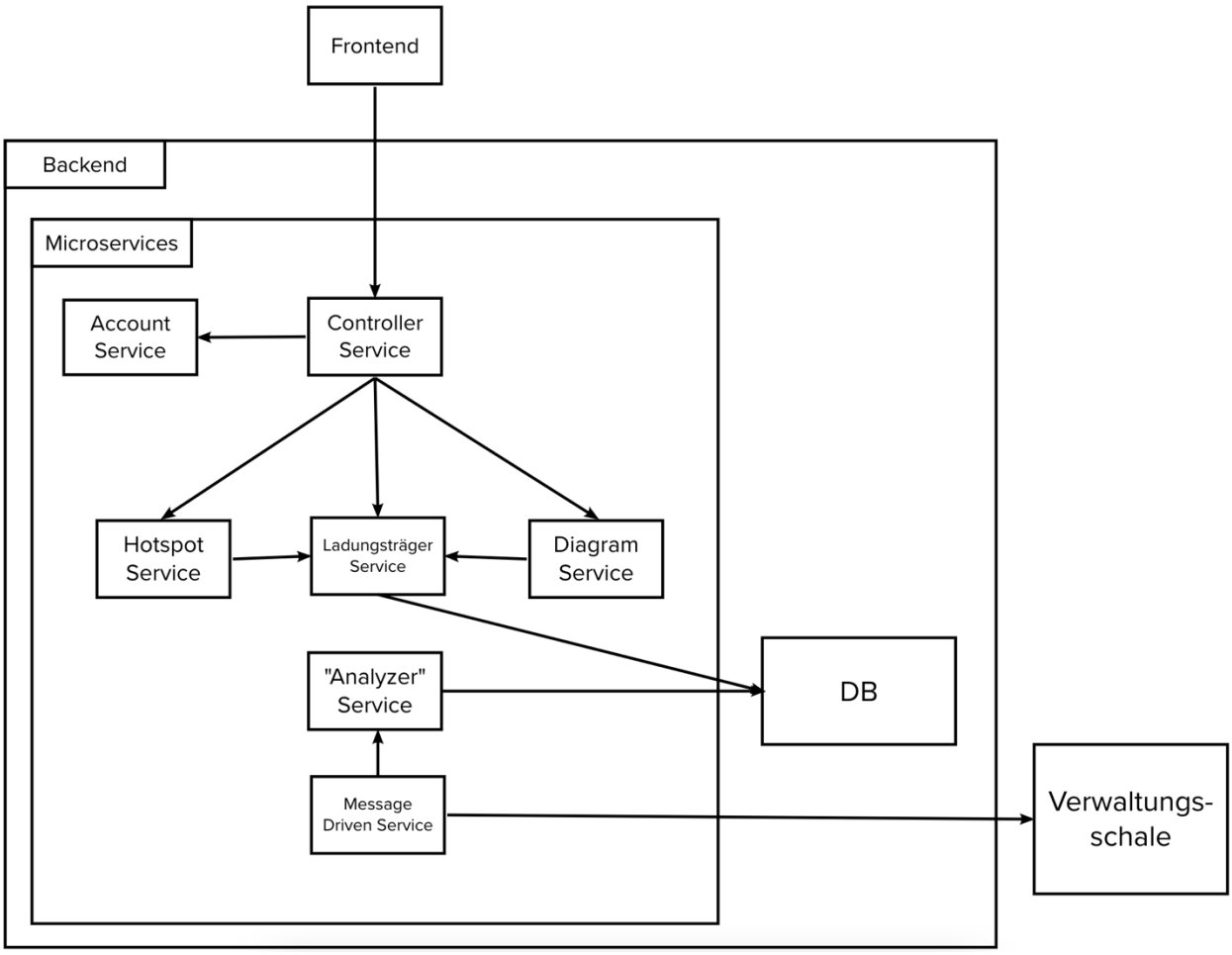


*Abbildung 4: Kommunikationsverbindungen der einzelnen Microservices*

## 4.3 Erreichung der wichtigsten Qualitätsanforderungen

### 4.3.1 Zuverlässigkeit

Um die Zuverlässigkeit zu gewährleisten, wird auf eine Microservice-Architektur mit Kubernetes gesetzt. Hierrüber kann jeder Service individuell nach Bedarf skaliert werden, um so zu jeder Zeit erreichbar zu sein. Über einen Heartbeat kann Kubernetes sicherstellen, dass ein Service für den Nutzer des Systems erreichbar ist.

### 4.3.2 Leistungseffizienz

Durch den gezielten Einsatz der Programmiersprache für jeden einzelnen Service, kann die optionale Leistung für jeden einzelnen Service erzeugt werden. Auch die Einbindung von Fremdbibliotheken erhöht die Leistungseffizienz, da diese zum einen von großen Entwicklerteams bereitgestellt werden, zum anderen aber von vielen anderen Entwicklern genutzt, getestet und optimiert werden, wodurch der Code dieser Bibliotheken automatisch eine hohe Leistungseffizienz aufweist.

### 4.3.3 Wartbarkeit

Die Wartbarkeit ist sowohl im Demonstrator als auch im produktiven System durch die Verwendung von Modulen beziehungsweise Services sichergestellt. Diese können unabhängig voneinander entwickelt, getestet und erweitert werden. Die Verwendung von Bibliotheken erhöht zudem die Wartbarkeit, da diese einfach wiederverwendet und ausgetauscht werden können. Hierdurch erhöht sich auch die Verständlichkeit des gesamten Projekts, da komplizierte Funktionen aus Fremdbibliotheken bezogen werden können. Da diese von vielen Entwicklern genutzt werden, bieten diese auch eine größere Sicherheit in Bezug auf Korrektheit und Funktionalität.

### 4.3.4 Sicherheit

Sicherheit im Produktivsystem wird zum einen dadurch erreicht, dass eine Mandantentrennung eingeführt wird. Durch die Verwendung von Mandanten wird sichergestellt, dass die Nutzer des Systems nur die Daten sehen können, die für sie bestimmt sind und nicht auf Daten anderer Nutzer zugreifen können. Zum anderen wird durch die Verwendung von Kubernetes-Clustern sichergestellt, dass das System ausfallsicher ist und auch gegen Angreifer von außen geschützt ist.

# 5 Bausteinsicht

In diesem Kapitel wird eine Übersicht über das gesamte System und einzelne Bereiche im Detail gegeben.

## 5.1 Kontextabgrenzung

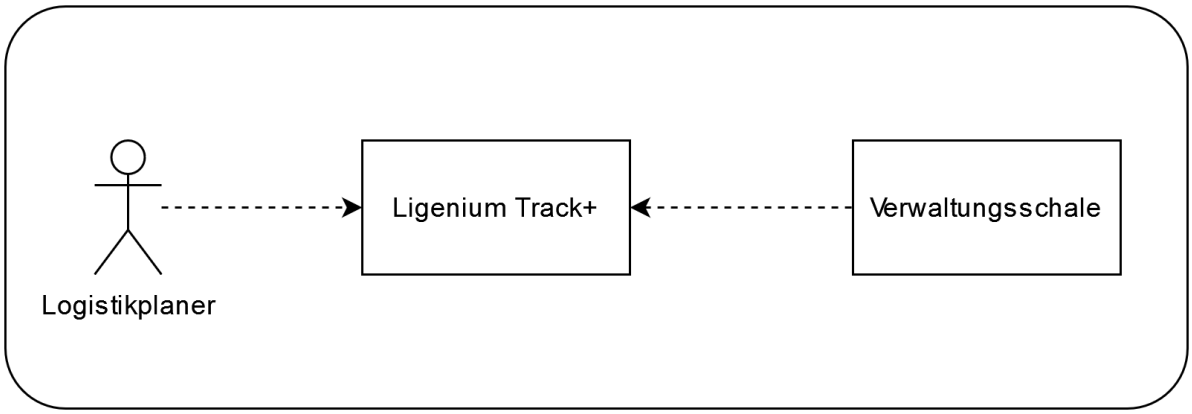


*Abbildung 5: Systemübersicht Bausteinsicht LT+*

Das System hat nach außen nur wenige Interaktionspunkte. Zum einen kann der Logistikplaner über ein Web-UI mit dem System interagieren und Anfragen an das System abschicken. Zum anderen bekommt das System von der Verwaltungsschale regelmäßige Updates über den Zustand der Ladungsträger.

## 5.2 Ebene 1

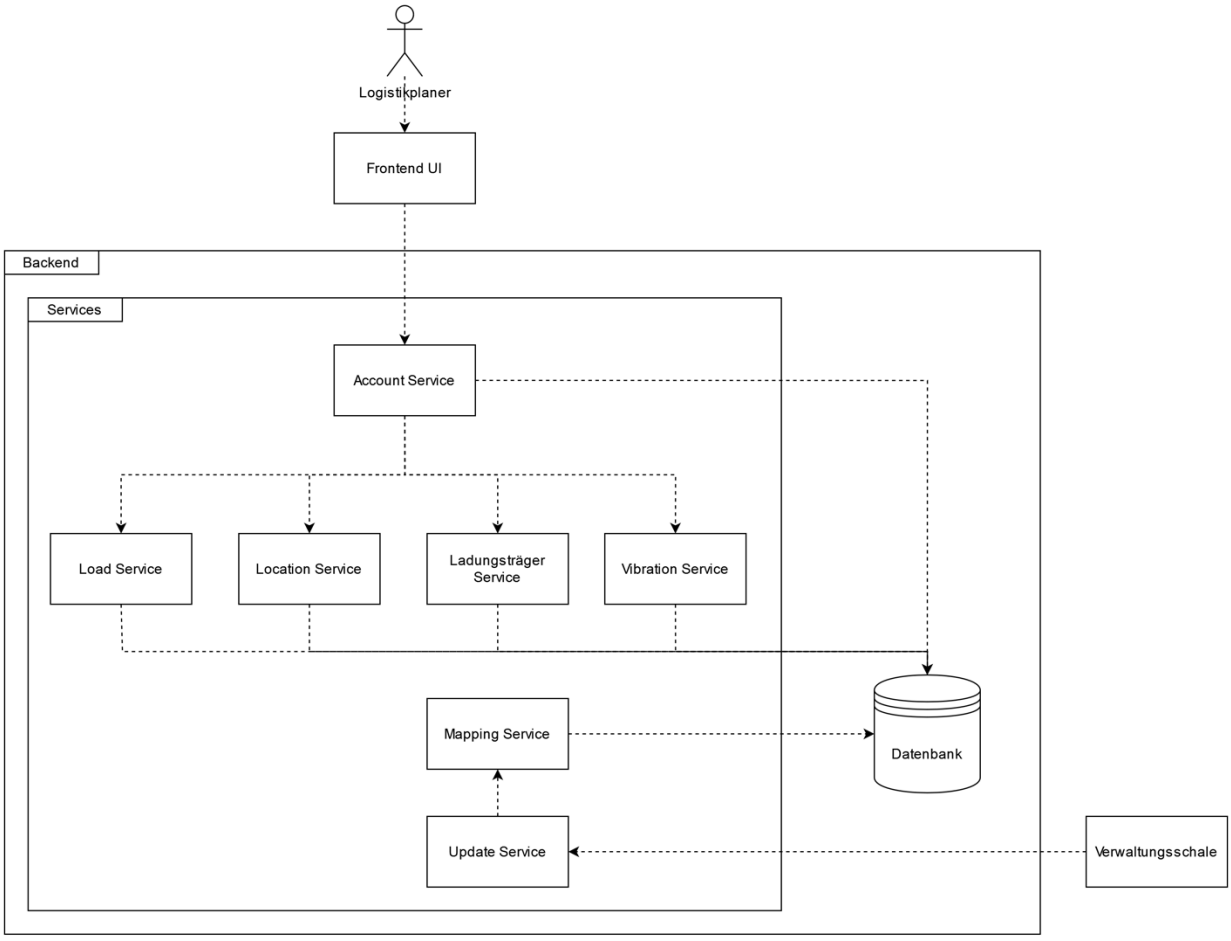


*Abbildung 6: Detaillierte Übersicht über die Bausteine von LT+*

Um eine Aufteilung auf Microservices gewährleisten zu können, sind die einzelnen Services anhand ihrer Zuständigkeiten abgegrenzt. Jeder Service hat seinen eigenen Aufgabenbereich und grenzt sich durch diesen von den anderen Services ab. Teilweise existiert jedoch Zusammenarbeit zwischen einzelnen Services, diese wird in der folgenden Auflistung der Bausteine näher beschrieben.

### 5.2.1 Account Service

Der Account Service dient zur Authentifizierung von Nutzern, die auf das System zugreifen. Er verhindert unautorisierten Zugriff auf das System und ermöglicht die Unterteilung von Nutzern in Rollen, die jeweils eigene Berechtigungen haben.

Der Account Service ist allen anderen Services vorgeschalten, um unautorisierten Zugriff auf diese Services zu verhindern. Zusätzlich kommuniziert er mit der Datenbank, um die gespeicherten Nutzer abzufragen.

### 5.2.2 Ladungsträger Service

Der Ladungsträger Service ist für einfache Abfragen über die Ladungsträger zuständig. Er liefert Auflistungen, die sortiert und gefiltert sein können.

Der Ladungsträger Service erhält nach einer Prüfung der Zugriffsberechtigung durch den Account Service Anfragen direkt aus dem Frontend. Außerdem fragt er die Ladungsträger aus der Datenbank ab.

### 5.2.3 Load Service

Der Load Service ist ein weiterverarbeitender Service. Er bereitet Aggregationen von Ladungsträgern und deren zusätzlichen Informationen soweit auf, dass mit dem Ergebnis im Frontend unterschiedliche Diagramme wie Balken- und Liniendiagramme dargestellt werden können. Zusätzlich bietet er die Möglichkeit, die abgefragten Daten mit einem Standort zu kombinieren und so aufzubereiten, dass im Frontend eine Hotspot Map erstellt werden kann.

Der Load Service wird nach einer Prüfung der Zugriffsberechtigung durch den Account Service direkt aus dem Frontend angesprochen. Er fragt Aggregationen von Informationen über Ladungsträger direkt aus der Datenbank ab.

### 5.2.4 Location Service

Der Location Service erfüllt nahezu die gleiche Aufgabe wie der Load Service. Er bedient jedoch Anfragen zur Beladung der Ladungsträger und bereitet diese Daten für ein Diagramm oder eine Hotspot Map auf. Zusätzlich überprüft er beim Speichern von Positionsdaten, ob der Ladungsträger sich im Vergleich zum letzten Eintrag bewegt hat.

Der Location Service wird nach einer Prüfung der Zugriffsberechtigung durch den Account Service direkt aus dem Frontend angesprochen. Er fragt genau wie der Load Service Aggregationen von Informationen über Ladungsträger direkt aus der Datenbank ab.

### 5.2.5 Vibration Service

Der Vibration Service bedient Anfragen zu Erschütterungsdaten der Ladungsträger und bereitet diese Daten ebenso wie die beiden vorhergehenden Services für ein Diagramm oder eine Hotspot Map auf.

Der Vibration Service wird nach einer Prüfung der Zugriffsberechtigung durch den Account Service direkt aus dem Frontend angesprochen. Er fragt genau wie die beiden vorhergehenden Services Aggregationen von Informationen über Ladungsträger direkt aus der Datenbank ab.

### 5.2.6 Mapping Service

Der Mapping Service bereitet neue Daten aus der Verwaltungsschale auf. Die Informationen werden in einem für dieses System dienlichen Format abgespeichert.

Der Mapping Service erhält vom Update Service Daten der Verwaltungsschale. Er speichert die aufgearbeiteten Daten in der Datenbank.

### 5.2.7 Update Service

Der Update Service ist ein Message Driven Service, der auf Änderungen in der Verwaltungsschale wartet. Erfährt er von einer solchen Änderung, leitet er die darin enthaltenen Daten an den Analyse Service weiter.

## 5.3 Ebene 2

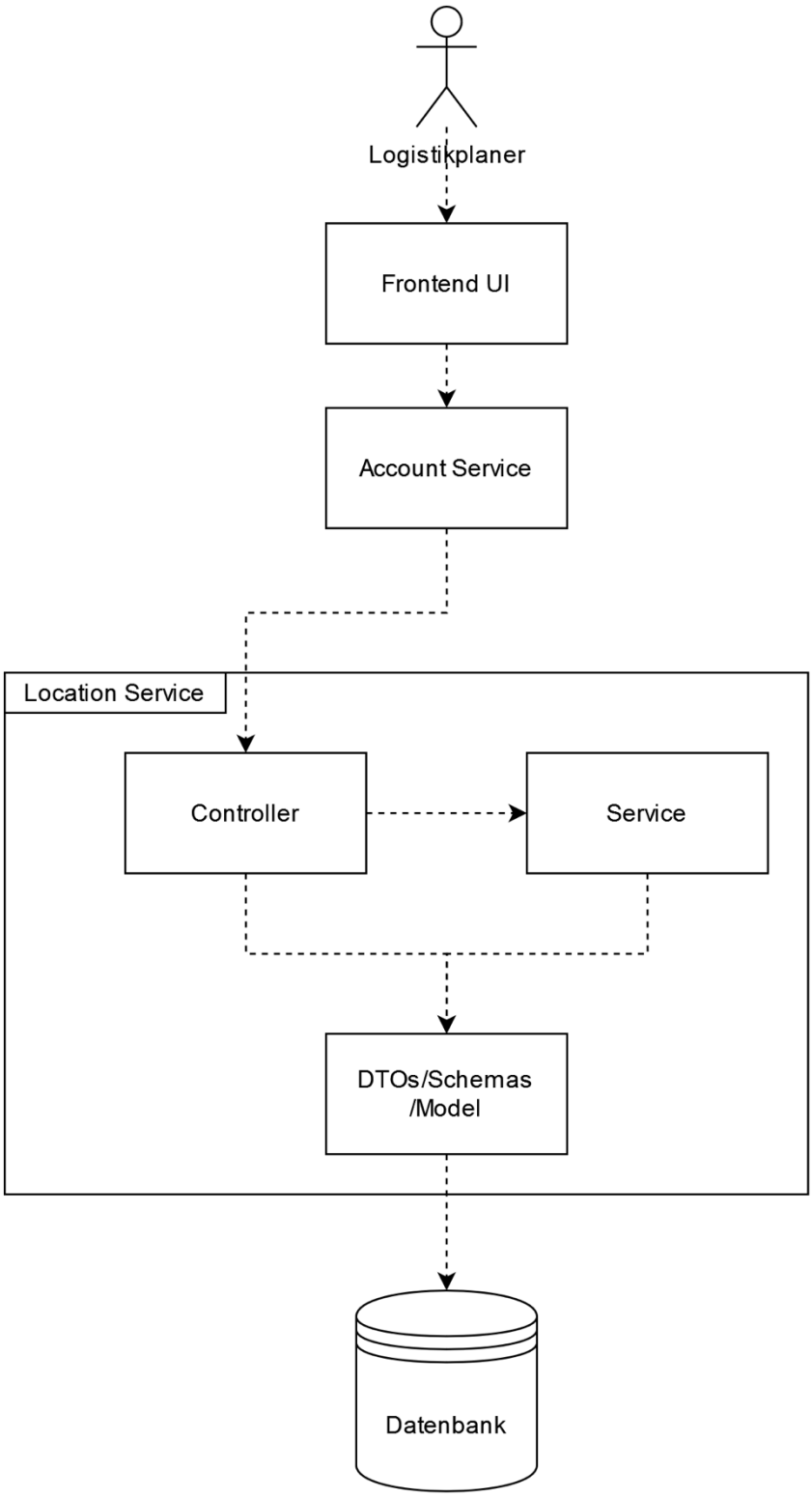
Logistikplaner
Frontend UI
Account Service
Location Service
Controller
Service
DTOs/Schemas
/Model
Datenbank

*Abbildung 7: Darstellung des Location Service in LT+ und Anbindung an angrenzende Komponenten*

### 5.3.1 Aufbau der Services

Der Aufbau der Services ist durch die Verwendung des Frameworks *NestJs* vorgegeben und deshalb in jedem Service gleich. Jeder Service besteht aus einer Controller-Komponente, einer Service-Komponente und das Datenmodell, bestehend aus DTOs, Schemas und Modellen. Deshalb wird dieser Aufbau hier einmal beispielhaft am Location Service gezeigt. Um Verwechslungen zwischen dem Service und der Service–Komponente innerhalb des Service vorzubeugen, wird diese weiterhin als Service–Komponente bezeichnet und nicht abgekürzt.

### 5.3.2 Location Service

Der Nutzer des Systems, in diesem Fall der Logistikplaner, greift über das Frontend UI nach einer Prüfung der Zugriffsberechtigung direkt auf den Location Service zu. Dabei wird die Anfrage vom Controller entgegengenommen. Der Controller implementiert nur wenige Funktionalitäten und fragt deswegen die geforderten Informationen bei der Service-Komponente an. Die Service-Komponente kommuniziert mit der Datenbank, verarbeitet die so erhaltenen Daten weiter und gibt sie an den Controller zurück. Der Controller sendet dem Frontend eine Antwort mit den geforderten Daten. Der Controller und die Service-Komponente nutzen beide gemeinsame Daten-Schemata und DTOs.

# 6 Laufzeitsicht

Im Folgenden findet sich eine Übersicht über den Abfrageablauf von Darstellungen von Daten, sowie ein Einblick in die Aufbereitung von Daten für Diagramme und Hotspot Maps.

## 6.1 Diagramm anzeigen lassen und Ansicht speichern

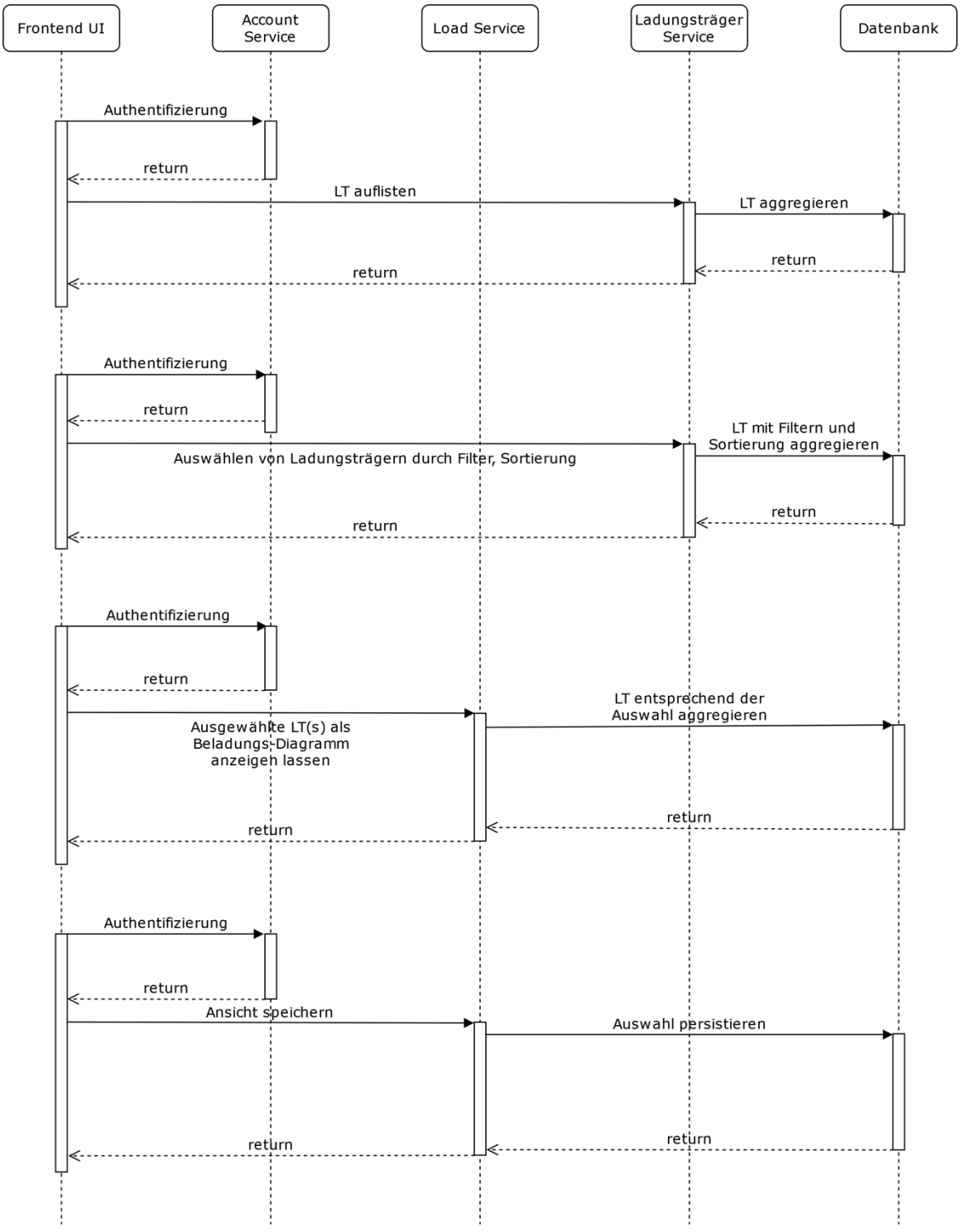


*Abbildung 8: Ablauf mehrerer miteinander verbundenen Anzeigeaktionen*

Der hier gezeigte Ablauf stellt eine quer durch das System verlaufende Aktion dar. Bei jeder Anfrage an das Backend werden zunächst die Zugriffsrechte des Nutzers.

Der Nutzer lässt sich zunächst eine Liste der Ladungsträger anzeigen. Dafür kommuniziert das Frontend mit dem Ladungsträger Service im Backend. Dieser fragt die gewünschten Daten in der Datenbank ab und bereitet sie so auf, dass sie vom Frontend dargestellt werden können.

Anschließend wählt der Nutzer bestimmte Filter- und Sortierkriterien aus und schickt eine neue Anfrage an den Ladungsträger Service. Dieser geht vor wie im Schritt zuvor, dieses Mal berücksichtigt er aber die geforderten Kriterien.

Als nächstes wählt der Nutzer einen Ladungsträger aus der Liste aus, um ihn sich als Diagramm anzeigen zulassen. Ein Beispiel für solch ein Diagramm wäre ein Liniendiagramm mit dem Verlauf der Beladung über einen Zeitraum. Dieses Mal kommuniziert das Frontend direkt mit dem Load Service. Dieser fragt die geforderten Daten für den Ladungsträger aus der Datenbank ab und verarbeitet sie in eine Form, die vom Frontend als Diagramm dargestellt werden kann. Im Frontend hat der Nutzer die Option, das fertige Diagramm herunterzuladen.

Abschließend speichert der Nutzer die Ansicht, wobei eine entsprechende Anfrage an den Load Service geschickt wird, der die ausgewählten Kriterien der Ansicht in der Datenbank speichert.

Dieses Beispiel gilt als Schaubild für die generelle Kommunikation zwischen Services und läuft bei anderen Anfragen, z.B. bei einer Hotspot Map oder anderen Informationen, wie z.B. Beladung oder Erschütterung, ebenfalls so ab.

## 6.2 Abfrage und Aufarbeitung der Daten im Diagramm Service

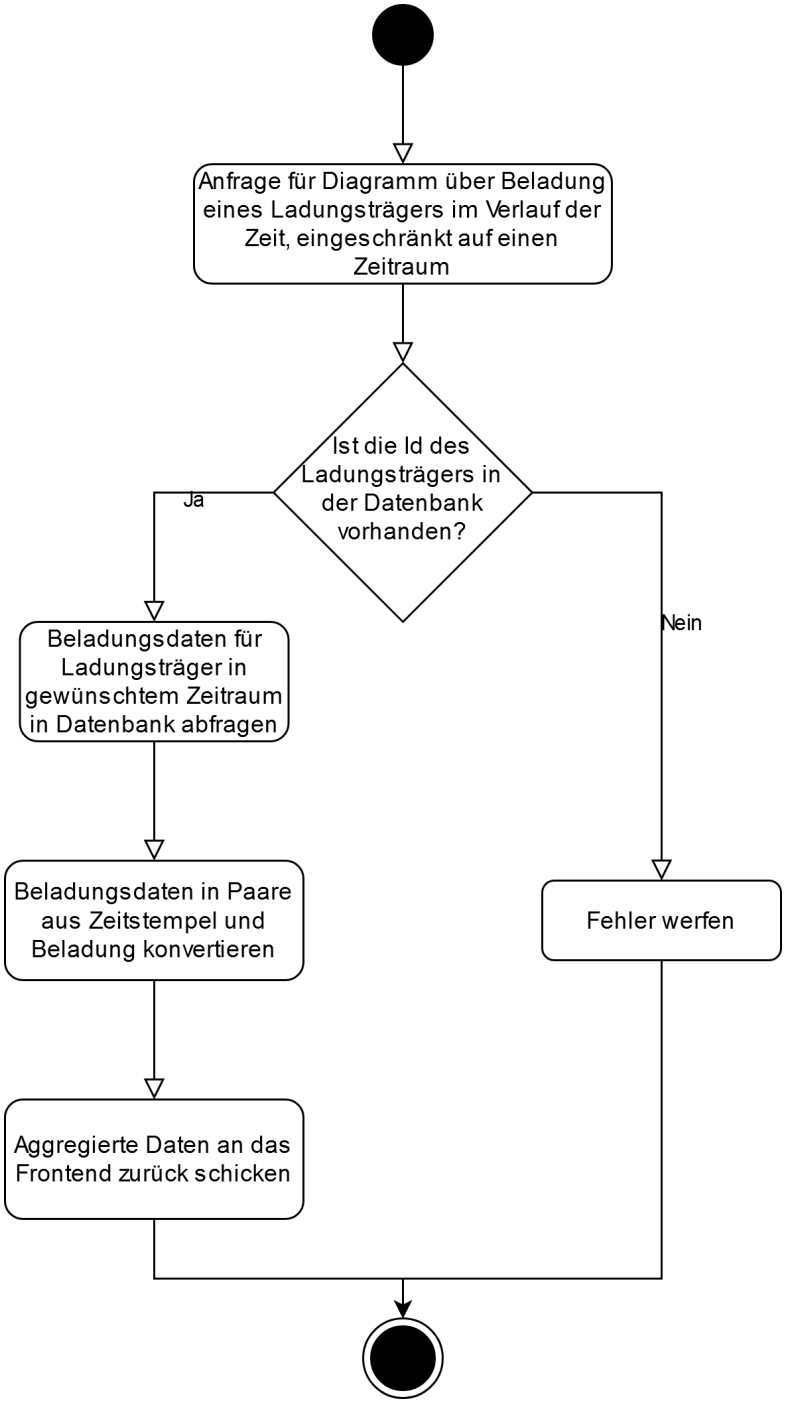


*Abbildung 9: Beispielhafte Darstellung des internen Ablaufs des Load Service*

Dieses Diagramm zeigt den Ablauf einer Anfrage für ein Diagramm über die Beladung eines Ladungsträgers im Verlauf der Zeit innerhalb eines gewählten Zeitraums. Der Ablauf ist beispielhaft für alle Services, die Daten aggregieren und aufbereiten.

# 7 Verteilungssicht

Im Folgenden werden zwei Verteilungssichten aufgeführt. Es wird unterschieden in Produktivbetrieb (Abbildung 10) und Demonstrator (Abbildung 11). Beim Demonstrator handelt es sich um eine vereinfachte Version des Produktivbetriebs, angepasst an die Natur der Veranstaltung, in deren Ramen das Projekt durchgeführt wird.

## 7.1 Infrastruktur Produktion

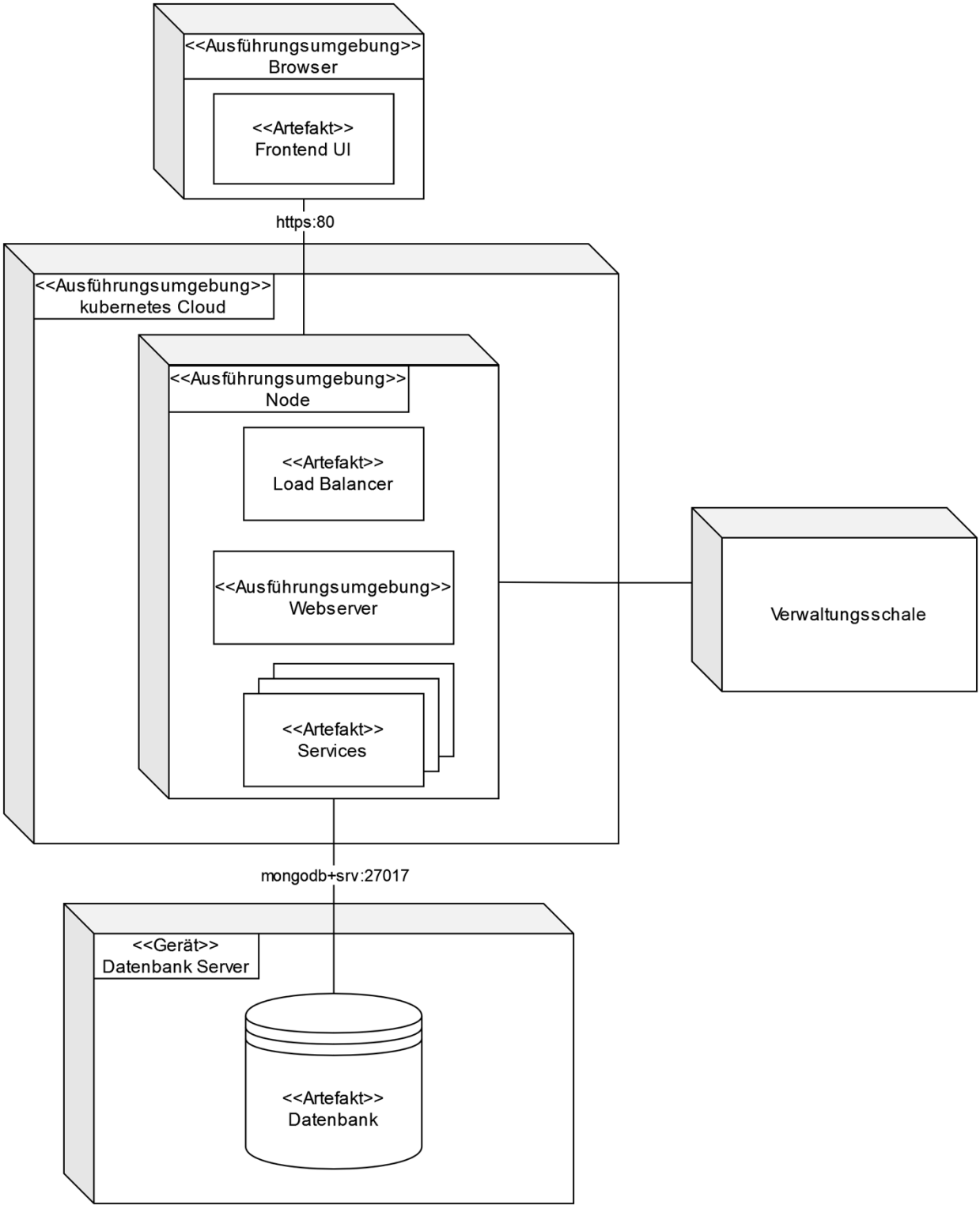


*Abbildung 10: Darstellung der Verteilung des Systems im Produktivbetrieb*

**Begründung**
Die Verwendung eines Load-Balancers wird angestrebt, um variierende Nutzerlast handhaben zu können. Das Deployment der Microservices als kubernetes-Cluster bei einem Anbieter (kubernetes as a service) sorgt für Skalierbarkeit mit der Anzahl der Nutzer. Die in Abbildung 10 aufgeführte Node stellt eine Art Arbeiter da, der einen Teile des Systems bereitstellt. Innerhalb der Node können Teile des Systems repliziert werden, um viel beanspruchte Funktionen mehrfach bereitzuhalten, damit die Anfragelast verteilt werden kann. Die Anzahl der Nodes skaliert automatisch mit der benötigten Leistung. Die einzelnen Nodes befinden sich in der Cloud, können also auf einem Server liegen oder verteilt auf mehreren Servern laufen. Da das Aufsetzen und die Verwaltung eines kubernetes-Clusters komplex ist, viel Zeit und technische Kenntnisse in diesem Bereich voraussetzt, sowie sowieso Kosten für das Bereitstellen anfallen, ist die naheliegendste Lösung die Nutzung eines kubernetes-Anbieters. Dies nimmt die meisten Arbeiten ab zum Beispiel die Verteilung der Nodes auf beliebig viele Server. Die Entscheidung über den Anbieter steht noch aus, eine Möglichkeit wäre die Google Kubernetes Engine, da kubernetes ein Google-Produkt ist. Auch die Datenbank kann als Datenbank Cluster gemietet werden, um Ausfallsicherheit und Lastenverteilung nutzen zu können.

**Qualitäts- und/oder Leistungsmerkmale**
Die Verwendung der Kombination eines Loadbalancers und eines kubernetes-Services ermöglicht maximale Flexibilität was Nutzerlast und gemietete Leistung angeht. Das System kann so automatisch mit der Nutzerlast skalieren, es wird aber nur für so viel Leistung bezahlt, wie auch tatsächlich benötigt wird. Der kubernetes-Cluster läuft verteilt in der Cloud, was dem System eine gewisse Resilienz gegenüber Ausfällen mit physischen Ursachen gibt.

## 7.2 Infrastruktur Demonstrator

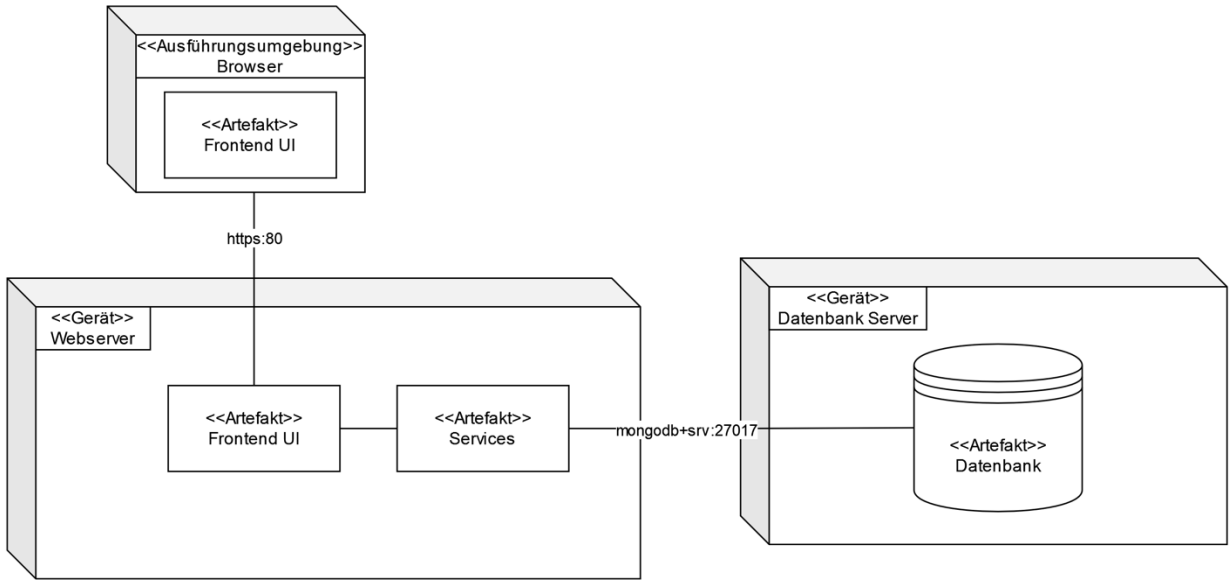


*Abbildung 11: Darstellung der Infrastruktur im Demonstrator*

**Begründung**
Da die Einrichtung eines kubernetes-Clusters aufwändig und kostspielig ist, wird der Demonstrator als einzelner Monolith in einem Docker-Container laufen gelassen. Durch die geringe Nutzerlast beim Testen und Vorführen des Demonstrators kann auf Skalierbarkeit verzichtet werden.

**Qualitäts- und/oder Leistungsmerkmale**
Das Aufsetzen des Systems in einem Docker-Container geht einfach und schnell und lässt sich sogar problemlos automatisieren. Änderungen sind schnell möglich, insgesamt ist es eine sehr einfache und flexible Lösung.

# 8 Querschnittliche Konzepte

Im folgenden Abschnitt werden querschnittliche Konzepte besprochen, welche Einfluss auf das ganze System haben.

## 8.1 User Experience

Die Benutzeroberfläche für das System besteht aus einer Verwaltungsansicht, einem Dashboard, Diagrammansicht, Hotspotansicht, Spitzenansicht und einem Benutzerhandbuch. Bevor der Benutzer diese Ansichten sehen kann, muss er sich zunächst mit einem gültigen Account einloggen. Abbildung 12 stellt diesen Login dar.

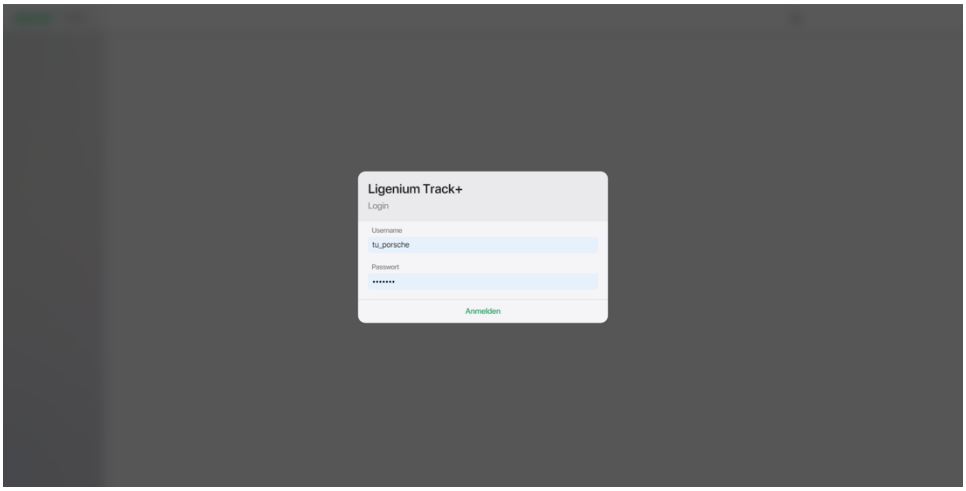


*Abbildung 12: Login des Ligenium Track+*

Nach erfolgreicher Authentifizierung wird die Verwaltungsseite angezeigt.

**Verwaltungsansicht**

In dieser Ansicht steht die Accountverwaltung im Mittelpunkt.

Hierbei sind folgende Funktionen je nach Berechtigung möglich:

- Abmelden
- Passwort ändern
- Passwort zurücksetzen
- Neue Accounts erstellen
- Account löschen
- Account einer Gruppe anordnen
- Berechtigungen eines Accounts anpassen

In Abbildung 13 ist ein Teil dieser Ansicht dargestellt.

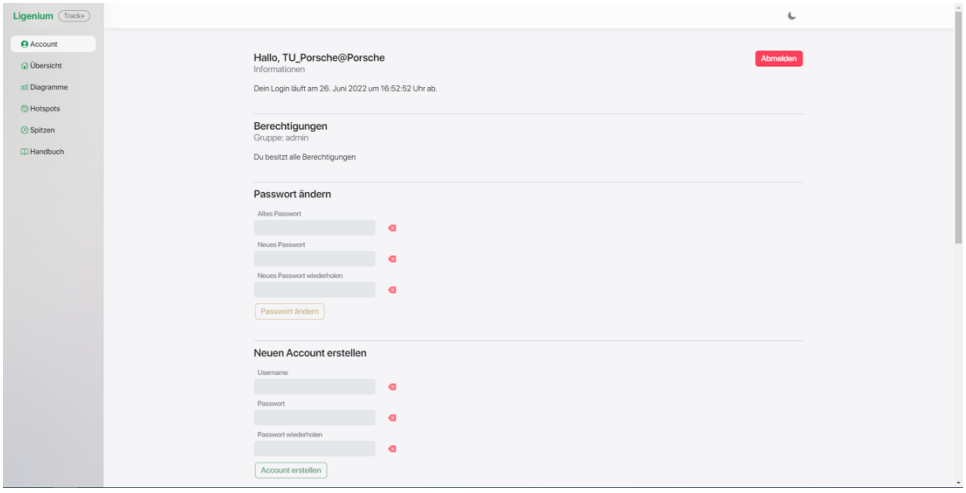


*Abbildung 13: Accountansicht des Ligenium Track+*

Wie man aus der Abbildung erkennen kann, sind die Funktionen zum Passwort ändern sowie das Erstellen eines neuen Accounts möglich.

Zusätzlich ist zu sehen, dass die Navigation innerhalb des Frontend und somit auch zwischen den verschiedenen Ansichten über die Navigationsleiste links erfolgt.

### Dashboard

In dieser Ansicht werden alle vorhandenen Ladungsträger aufgelistet. Diese Seite stellt dem Benutzer eine grobe Übersicht der vorhandenen Ladungsträger dar. Diese Ladungsträger können nach verschiedenen Kriterien gefiltert und sortiert werden, und ermöglichen somit dem Benutzer sich ein Bild der aktuellen Lage zu machen.

In Abbildung 14 ist diese Übersicht mit einigen Testdaten dargestellt.

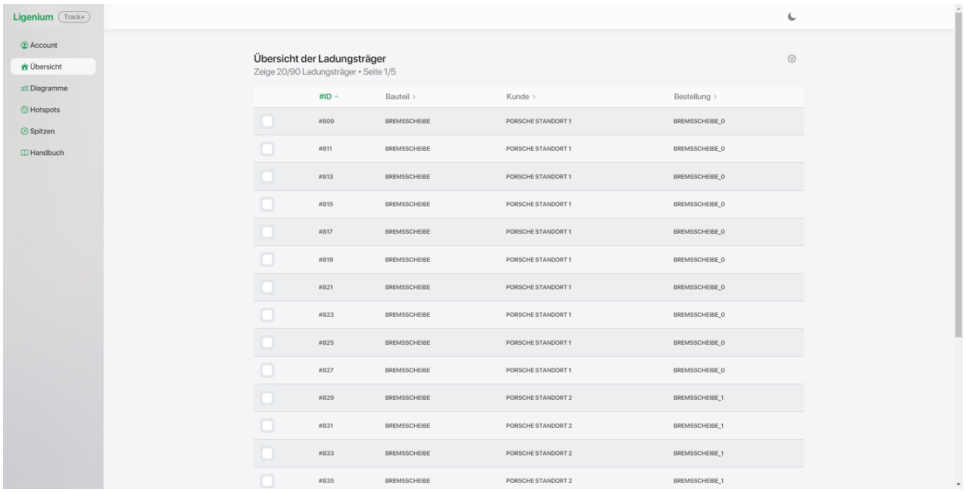


*Abbildung 14: Dashboard mit verfügbaren Ladungsträgern*

Die aufgelisteten Ladungsträger sind aufsteigend nach IDs sortiert, wie man an dem grünen Pfeil nach oben feststellen kann. Des Weiteren kann diese Liste an Ladungsträgern gefiltert werden, indem der Benutzer auf das Filtersymbol oben rechts klickt. Hierbei öffnet sich ein Popup-Fenster mit verschiedenen Filteroptionen.

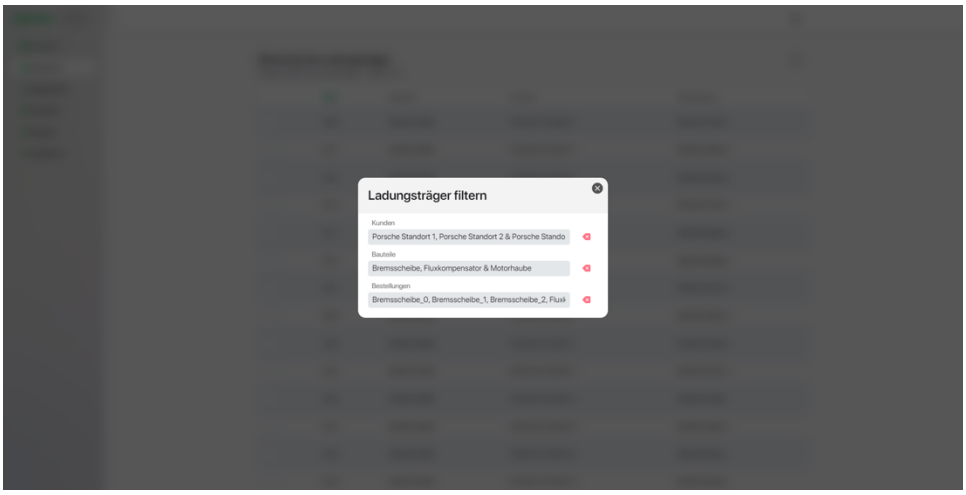


*Abbildung 15: Overlay zur Filterung der Ladungsträger*

In Abbildung 15 werden die aktuellen Filteroptionen dargestellt. Ist eine dieser Optionen ausgewählt wie z.B. “Bremsscheibe_0” so werden nur Ladungsträger von dieser Bestellung dargestellt. Wurde mindestens ein Kriterium ausgewählt, so werden diese in der entsprechenden Zeile aufgelistet. In Abbildung 15 wurden die Bestellungen “Bestellung_0”, “Bestellung_1” und “Bestellung_2” ausgewählt.

Möchte der Benutzer mehr über einen Ladungsträger erfahren, wie z.B. die Beladung des Ladungsträgers in der letzten Woche, so kann die Diagrammseite über die

Navigationsleiste aufgerufen werden. Die gefilterten Ladungsträger im Dashboard werden in den Diagrammen entsprechend visuell repräsentiert.

### Diagrammansicht

Hier können weitere Informationen über einen Ladungsträger abgerufen werden. Über mehrere Diagramme können verschiedene Werte wie z.B. Standzeit oder Beladung des Ladungsträgers visuell dargestellt werden.

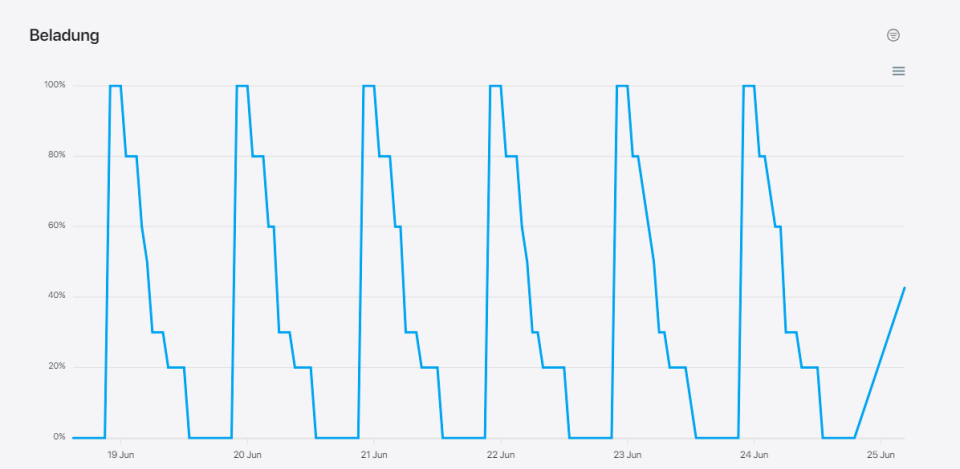


*Abbildung 16: Beladung eines Ladungsträgers*

In Abbildung 16 wird die Beladung des Ladungsträgers mit der ID #001 dargestellt. Je nachdem welchen Zeitraum man auswählt, wird die entsprechende Beladung innerhalb des Zeitraums angezeigt. Wählt man keinen Zeitraum aus, wie in der Abbildung zu sehen, so werden alle vorhandenen Beladungsdaten angezeigt.

Zusätzlich zur Beladung, sollen auch die Voll – und Leerzeiten der Ladungsträger in dieser Ansicht dargestellt werden. Hierzu wird ein gestapeltes Balkendiagramm verwendet, wie in der Abbildung 17 zu sehen ist.

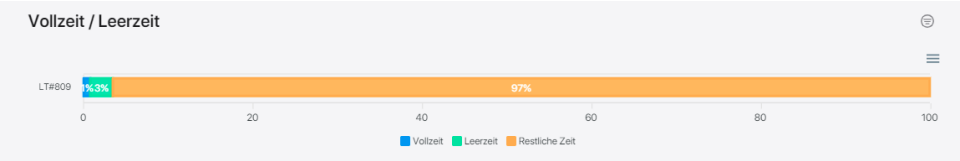


*Abbildung 17: Voll – und Leerzeit eines Ladungsträgers*

In Abbildung 17 wird dargestellt, wie lange ein Ladungsträger in welchem Zustand verbracht hat. Die Farbe Blau repräsentiert den Zustand, in dem der Ladungsträger vollständig beladen war. Analog repräsentieren die Farbe Grün den Leerzustand und Orange die restlichen Zustände zwischen voll und leer.

Damit auch die Standzeiten eines Ladungsträgers beobachtet werden können, folgt dem Diagramm in Abbildung 17 eine separates Balkendiagramm. Dieses Diagramm ist in

Abbildung 18 dargestellt.

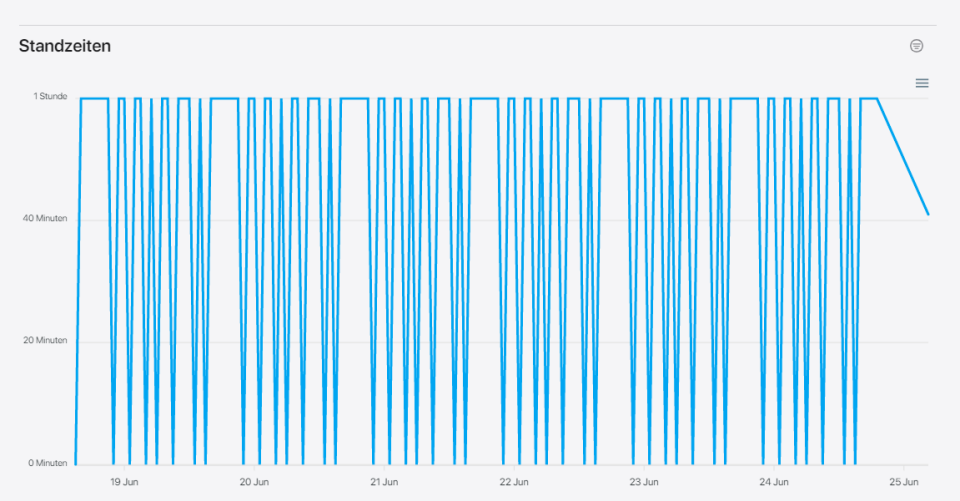


*Abbildung 18: Standzeit eines Ladungsträgers*

In Abbildung 18 sind die Standzeiten bzgl. eines Ladungsträgers dargestellt. Hier wird visualisiert, wie oft und wie lange ein Ladungsträger stehen bleibt.

Ergänzend zu den beschriebenen Diagrammen, werden Erschütterungen der Ladungsträger erfasst und visuell dargestellt. In Abbildung 18 wird das entsprechende Diagramm dargestellt.

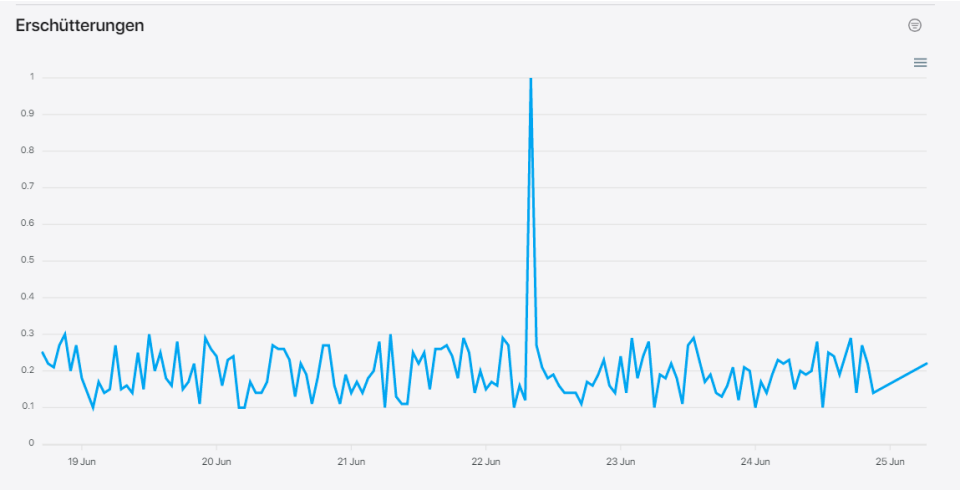


*Abbildung 19: Erschütterungen eines Ladungsträgers*

Hier ist zu erkennen, dass zwischen 22. - und 23. Juni eine größere Erschütterung stattgefunden hat. Sollte sich diese wiederholen, kann dies ein Indikator sein, um diese Erschütterung weiter zu untersuchen.

Neben einzelnen Ladungsträgern, können auch mehrere Ladungsträger gleichzeitig in den Diagrammen dargestellt werden. Diese werden entsprechend farblich unterschieden, wie in der folgenden Abbildung 20.

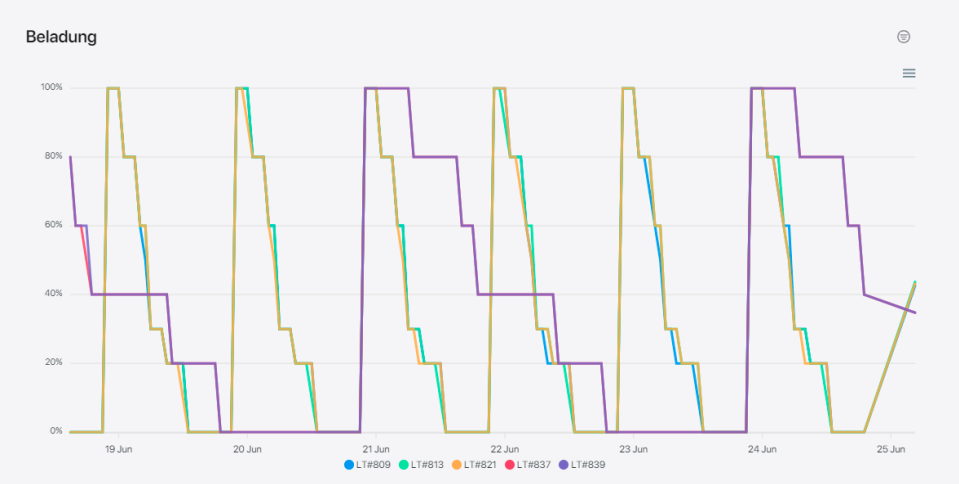


*Abbildung 20: Beladungen von mehreren Ladungsträgern*

Hier sind die Testdaten zu den Ladungsträgern mit den IDs #809 (blau), #813 (grün) und #821 (gelb), #837 (rot), #839 (lila) dargestellt. Dies erlaubt es den Benutzer, die Unterschiede bzgl. Beladung zwischen verschiedenen Ladungsträgern auf einen Blick zu vergleichen und ggf. Anomalien festzustellen.

Analog können auch mehrere Ladungsträger bei den anderen beschriebenen Diagrammen dargestellt werden.

Möchte der Benutzer das Diagramm außerhalb der Applikation festhalten, so kann per Klick auf das Hamburger-Menu des Diagrammes oben rechts, das Diagramm als SVG, CSV oder PNG exportiert werden. Das entsprechende Menü ist in der folgenden Abbildung dargestellt.

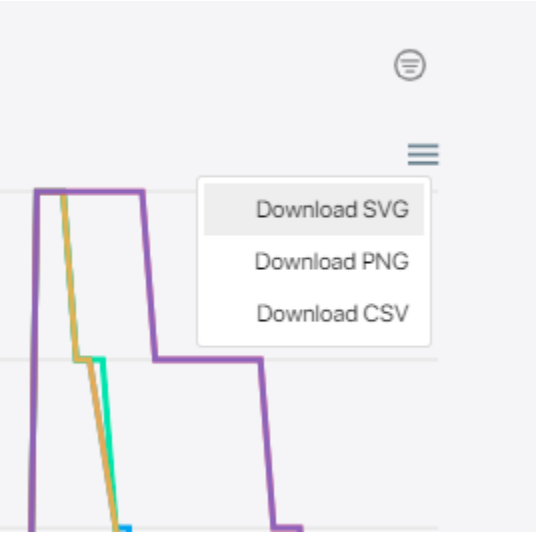


*Abbildung 21: Menüoptionen der Diagramme*

Zusätzlich bietet das Diagramm weitere Funktionalitäten wie Zoom und das entsprechende Zurücksetzten der Zoom-Einstellung.

**Hotspotansicht**

In dieser Ansicht werden die ausgewählten Ladungsträger als Hotspots auf einer Karte dargestellt. Dies soll dem Benutzer als eine zusätzliche, visuelle Unterstützung dienen, um die Ladungsträger mit hohen Standzeiten oder mit längerer niedriger Beladung zu erkennen. Mit der geografischen Position der Ladungsträger, soll es dem Benutzer möglich sein, potenzielle Muster oder Problemstellen zu identifizieren. In Abbildung 22 ist solch eine Hotspot-Map dargestellt.

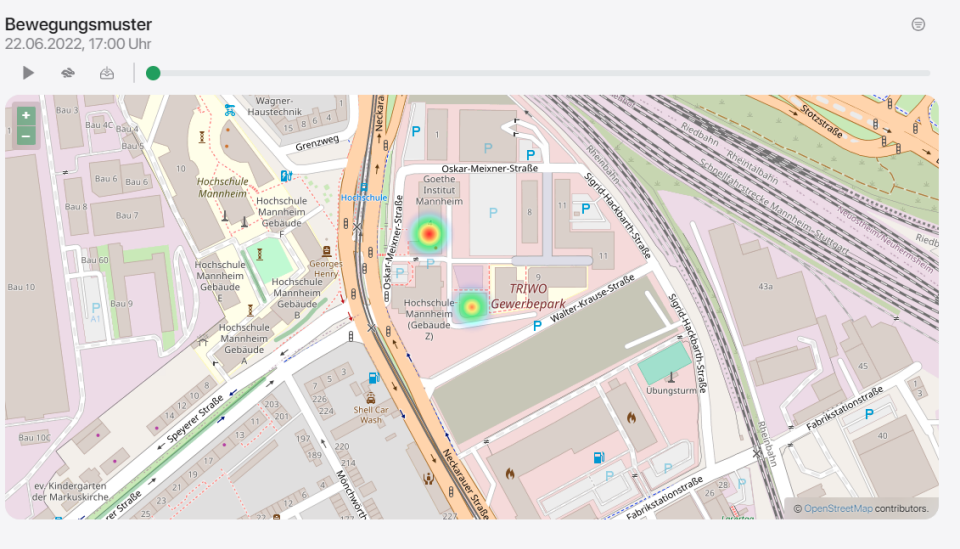


*Abbildung 22: Hotspot-Map der Bewegungsmuster von Ladungsträgern*

In diesem Beispiel werden die Bewegungsmuster der ausgewählten Ladungsträger dargestellt. Die Konzentration der Ladungsträger ist durch die unterschiedliche farbliche Erkennung dargestellt.

Neben dem Bewegungsmuster, werden die Standzeiten der Ladungsträger ebenfalls in Form einer Hotspot-Map angezeigt. In Abbildung 23 ist solch eine Karte dargestellt.

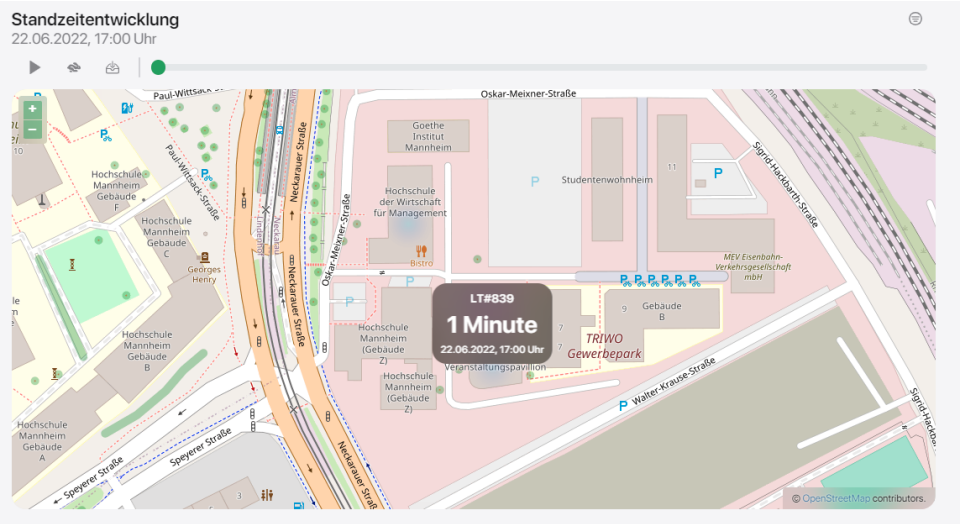


*Abbildung 23: Standzeiten von Ladungsträgern*

Wie man aus Abbildung 23 erkennen kann, sind an den Positionen aus Abbildung 22 nun graue Punkte dargestellt. Diese Punkte visualisieren Ladungsträger. Per Mousehover wird ein Tooltip angezeigt. Dieser Tooltip beschriebt, um welchen Ladungsträger es sich handelt und wie lange dieser Ladungsträger sich bereits an dieser Position befindet.

Analog zu der Darstellung aus Abbildung 23, enthält die Hotspotansicht zwei weitere Diagramme bzgl. der Beladungsentwicklung sowie der Vibrationen von Ladungsträgern.

**Spitzenansicht**

In dieser Ansicht geht es darum, den Benutzer besonders stark bzgl. Analyse und Anomalien-Erkennung bei den Ladungsträgern zu unterstützten. Hier beziehen sich die Diagramme auf genau zwei Aspekte: Spitzen in der Standzeit sowie Spitzen bzgl. Vibrationsstärke.

In Abbildung 24 ist die Hotspot-Map für solche Standzeitspitzen dargestellt.

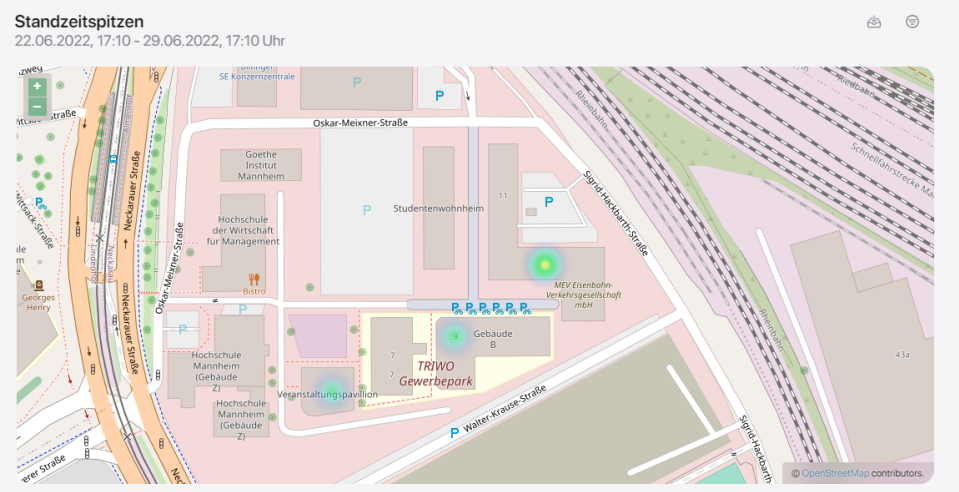


*Abbildung 24: Standspitzen verschiedener Ladungsträger*

Wie man an Abbildung 24 sehen kann, unterscheiden sich die Ladungsträger auch visuell. Je nachdem wie lange ein Ladungsträger steht, desto stärker wird dies visuell ausgedrückt.

Zusätzlich ermöglicht die Hotspot-Map in Abbildung 25 die Darstellung von Erschütterungen bzw. Vibrationen an den Ladungsträgern über einen Schwellenwert. Dies erlaubt es dem Benutzer festzustellen, wann und wo starke Erschütterungen stattgefunden haben.

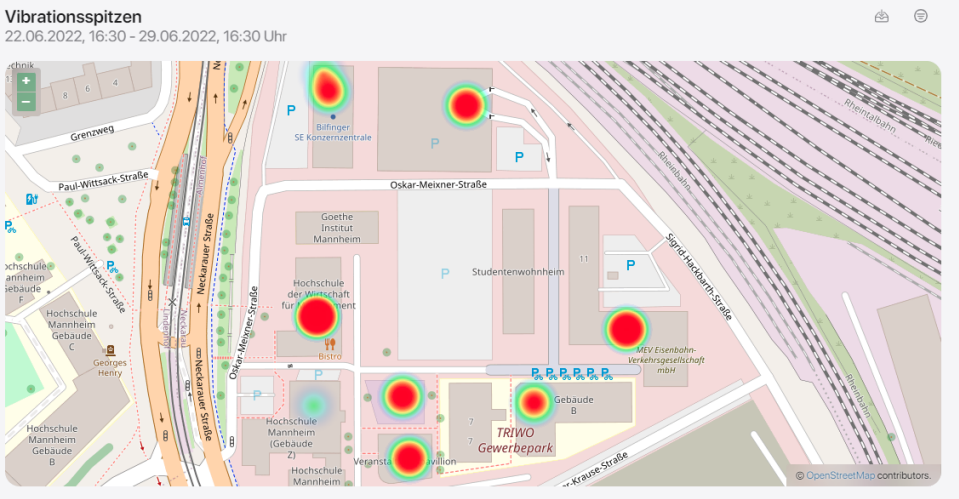


*Abbildung 25: Hotspot-Map der Vibrationsspitzen von Ladungsträgern*

An Abbildung 25 ist an der farblichen Markierung erkennbar, dass es sich um stärkere Erschütterungen handelt. Je höher der Wert der Erschütterung ist, desto drastischer wird der Ladungsträger auf der Karte repräsentiert.

**Benutzerhandbuch**

Diese Ansicht beinhaltet ein Benutzerhandbuch, dass die wesentlichen Elemente des Ligenum Track+ beschreibt. Sie soll insbesondere neue Benutzer unterstützen, um sich zurechtzufinden.

## 8.2 Sicherheitskonzepte

Um die Sicherheit des Systems zu erhöhen, können nur Personen mit einem authentifizierten Account zugreifen. Zusätzlich werden diese Accounts in verschiedene Gruppen sowie Berechtigungen unterteilt. Somit wird der Schaden, den ein Account der nicht zur Admin-Gruppe gehört anrichten kann, minimiert. Nur Personen die administrative Positionen besitzen, sollen auch Accounts von anderen Benutzern beeinflussen können.

Jeder Kunde besitzt seine eigene Account-Gruppe und kann somit auch nur die Daten der Ladungsträger sehen, die ihm zugeordnet wurden. Somit wird die Datensicherheit der verschiedenen Kunden gewährleistet.

## *8.3* Unter der Haube

**Persistenz**

Zum Persistieren der Daten wird hier die Datenbank Mongo-DB verwendet. Die Micro-Service-Architektur benötigt einige Eigenschaften von der Datenbank, um diese effektiv einzusetzen.

Sollte es zu einer Änderung des Datenmodells kommen, so muss das Datenbankmanagementsystem flexibel genug sein, dies zu erkennen und mit dieser Änderung umzugehen. Relationale Datenbanken dagegen, müssen hierbei alle vorhandenen Einträge in der Datenbank auf einmal aktualisieren, was bei MongoDB nicht notwendig ist.

Zusätzlich bietet MongoDB Redundanz, was bei dieser Architektur von Vorteil ist. Micro-Services sind keine langlebigen Prozesse, somit ist es nicht unüblich, dass ein Prozess abstürzt, oder einfach heruntergefahren wird, falls dieser nicht korrekt arbeitet.

MongoDB bietet eine skalierende Datenbanklösung, die insbesondere bei Micro-Services von Vorteil ist. Je nach Notwendigkeit kann die Anzahl der Micro-Services hoch oder runter geregelt werden. Hierbei kann MongoDB bei der Skalierung die Datenbank in verschiedene Partitionen zerlegen und auf verschiedene Knoten des kubernetes-Clusters verteilt werden.

Bei einer Micro-Service-Architektur ist es üblich, eine eigene Datenbank für jeden Service anzulegen. Dies führt zu mehreren Instanzen von Datenbanken, was bei relationalen Datenbanken problematisch ist. Jedoch ist MongoDB in der Lage mit diesen Datenbank-Clustern umzugehen, und bietet somit eine passende Lösung zur Micro-Service-Architektur.

Beschränkung beim Demonstrator

Im Rahmen des Demonstrators verwenden wir eine MongoDB-Instanz im Backend.

**Kommunikation-/Integration**

Die Kommunikation im System erfolgt über HTTPS-Request zwischen dem Frontend und den Micro-Services im Backend.

**Parallelisierung**

Innerhalb des Demonstrators wird JavaScript im Backend und auch im Frontend verwendet. Das Frontend selbst ist in Typescript geschrieben, was dann zu JavaScript kompiliert wird. Da JavaScript single-threaded ist, bietet es keine multi-thread / Parallelisierungsmöglichkeit an. Die Prozesse werden entweder synchron oder via asynchrone Funktionen abgearbeitet.

## 8.4 Entwicklungskonzepte

**Build, Test, Deploy**

Mithilfe von z.B. GitLab Runner kann der Code-Stand gebaut und automatische Tests ausgeführt werden. Vor dem Build können Tests im Frontend in Form von E2E-, Unit-, oder Komponententests durch Verwendung von z.B. cypress oder Puppeteer durchgeführt

werden. Daraufhin kann das Build auf die Registry (z.B. Helm) innerhalb des kubernetes-Clusters hochgeladen werden. Von dieser Registry heraus können die Container bzw. Nodes aktualisiert werden mittels Rolling Update.

Beschränkung beim Demonstrator

Während der Implementierung des Demonstrators werden GitHub Actions verwendet, um ein Build zu erstellen und auf einen Docker-Container hochzuladen. Die Verwaltung der Docker-Container erfolgt mithilfe von Docker Compose.

### Codegenerierung

Zur Codegenerierung wurden Vue-Komponent-Schablonen angelegt, um die Struktur der Vue-Komponenten zu vereinheitlichen.

### Deployment

Zum Deployment des Systems ist ein kubernetes-Cluster notwendig. Dieses Cluster muss zuvor bestehen oder spätestens vor der Inbetriebnahme eingerichtet werden.

### Konfiguration

Die Verbindungsdaten zu der MongoDB werden in einer .env-Datei abgelegt. Je nach Notwendigkeit können diese Daten angepasst werden.

## 8.5 Betriebskonzepte

### Skalierung

Bei der Micro-Service-Architektur kann nach Bedarf die Anzahl der Services hoch- und runterskaliert werden. So kann das System bei einer großen Anfrage von Nutzern hochskaliert werden bzw. Runterskaliert werden, wenn die Auslastung des Systems nachlässt. Kubernetes bietet diese Funktionalität an, und kann mit entsprechender Konfiguration diese Skalierung auch automatisieren.

# 9 Qualitätsanforderungen

## 9.1 Qualitätsbaum

In Tabelle 2 werden die in Kapitel 1.2 beschriebenen Qualitätsziele erneut aufgefasst und in konkretere Unterpunkte aufgeteilt. Dieser Unterpunkte sind mit Nummern versehen und dienen zur Orientierung bzgl. der Qualitätsszenarien.

*Tabelle 2: Qualitätsbaum des Ligenum Track+*

| Qualitätsziel | Qualitätsunterpunkt | ID |
|---|---|---|
| Zuverlässigkeit | Verfügbarkeit | 0 |
| | Fehlertoleranz | 1 |
| Sicherheit | Vertraulichkeit | 2 |
| | Verantwortlichkeit | 3 |
| Leistungseffizienz | Zeitverhalten | |
| | Kapazität | 4 |
| Wartbarkeit | Änderbarkeit | 5 |
| Kompatibilität | Zusammenarbeitsfähigkeit | 6 |

## 9.2 Qualitätsszenarien

In Tabelle 3 folgen Szenarien, die auf verschiedene Unterpunkte der festgelegten Qualitätsziele beschreiben.

*Tabelle 3: Qualitätsszenarien des Ligenum Track+*

| ID | Qualitätsszenario |
|---|---|
| 0,1,4 | Die Verfügbarkeit des Systems und soll auch bei großen Datenmengen und einer Vielzahl von parallelen Nutzern gewährleistet werden. |

| | |
|---|---|
| 2,3 | Der Kunde erhält nur die Daten, die auch für ihn notwendig sind. Kein weiterer Kunde soll in der Lage sein, die Daten der anderen Kunden abzufragen |
| 2,3 | Die Kundendaten sind verschlüsselt abzulegen, um die Sicherheit der Daten zu gewährleisten. |
| 0,1,3 | Das System soll regelmäßig Backups der Kundendaten erstellen, um die Ausfallsicherheit zu gewährleisten. |
| 5 | Weiterentwicklung wie z.B. das Erstellen eines neuen Features, soll durch das System ermöglicht werden |
| 5,6 | Die Kompatibilität zur Verwaltungsschale soll zu jedem Zeitpunkt gewährleistet werden. Insbesondere nach Änderungen am System |

# 10 Risiken und technische Schulden

In Tabelle 4 sind die erkannten Risiken bei der Entwicklung des Demonstrators “Ligenum Track+” dargestellt. Als Skala wird hier die Zahlenreihe zwischen 1 und 5 verwendet. 1 repräsentiert “sehr niedrig” und 5 repräsentiert “sehr hoch”.

*Tabelle 4: Potenzielle Risiken*

| ID | Beschreibung | Wahrscheinlichkeit | Schaden | Präventive Maßnahmen | Reaktive Maßnahmen |
|---|---|---|---|---|---|
| 1 | Weiteres Teammitglied fällt aus | 2 | 5 | Regelmäßige interne Gespräche über Motivation und Belastung, um ggf. Mitglieder temporär zu entlasten | Neuaufteilung der Aufgaben, ggf. Streichungen in Rücksprache mit objective partner |
| 2 | Entwicklungshardware bei einem oder mehreren Teammitgliedern fällt aus | 2 | 4 | Überwachung technischer Gesundheit der Geräte, Ausweichmöglichkeiten sondieren | Ersatz beschaffen, vorhandenen Ersatz nutzen |
| 3 | Wir entwickeln an den Vorstellungen/Wünschen von objective partner vorbei | 2 | 4 | Rücksprache mit objective partner, Validierung der Journey-Map durch objective partner | Neues Gespräch über Vorstellungen, Anpassung des Projektes |
| 4 | Architektur stellt sich als ungeeignet heraus | 1 | 4 | Ausgiebige Recherche im Vorfeld, Nutzung etablierter Architekturmodelle, Validierung durch objective partner | Architektur überarbeiten und anpassen |
| 5 | Verwendet Technologien stellen sich als ungeeignet heraus | 1 | 5 | Ausgiebige Recherche im Vorfeld, Verwendung von Technologien mit vorhandenen Erfahrungen in vergleichbaren Projekten | Technologiewechsel |
| 6 | Zeitplanung zu optimistisch geschätzt | 2 | 2 | Vorsichtige Planung auf Basis von Erfahrungen vergangener Projekte | Zeitplanung überarbeiten, schlimmstenfalls Streichungen |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | in Rücksprache mit objective partner |

# 11 Glossar

| Begriff | Definition |
|---|---|
| *Dashboard* | *Unter Dashboard versteht man in dem System eine Ansicht, die alle verfügbaren Ladungsträger in Form einer Liste darstellt.* |
| *OWASP* | *OWASP ist eine internationale gemeinnützige Organisation, die sich der Sicherheit von Webanwendungen widmet. Eines der Hauptprinzipien von OWASP ist, dass alle ihre Materialien frei verfügbar und auf ihrer Website leicht zugänglich sind, so dass jeder die Möglichkeit hat, die Sicherheit seiner eigenen Webanwendungen zu verbessern. Zu den Materialien, die sie anbieten, gehören Dokumentationen, Tools, Videos und Foren. Ihr vielleicht bekanntestes Projekt ist die OWASP Top 10. Die OWASP Top 10 ist ein regelmäßig aktualisierter Bericht, der die Sicherheitsprobleme im Bereich der Webanwendungssicherheit beschreibt und sich auf die 10 wichtigsten Risiken konzentriert. (https://www.cloudflare.com/en-gb/learning/security/threats/owasp-top-10/) [26.06.2022]* |
| *Schema* | *MongoDB verwendet zum Speichern von Datensätzen sogenannte „Schemas“, welche die Struktur und den Inhalt des Datensatzes beschreiben. Sie sind das Pendant zu Relationen bei relationalen Datenbanken. (Vgl. hierzu https://www.mongodb.com/docs/atlas/app-services/schemas/)* |
| *Verwaltungsschale* | *“Die Verwaltungsschale ist ein herstellerübergreifender und branchenneutraler Standard für die Bereitstellung von Informationen und für die Kommunikation in einheitlicher Sprache. Jedes „Ding“ im „Internet der Dinge“ (IoT) erhält eine eigene Verwaltungsschale. [...] Jedes Asset, wie beispielsweise ein Gerät oder Bauteil, kann über seine eigene Verwaltungsschale weltweit identifiziert und angesprochen werden. Sie stellt Informationen über Eigenschaften und Fähigkeiten des Gegenstands bereit. Über ihre genormten Schnittstellen und eine einheitliche Sprache können die „Dinge“ miteinander kommunizieren.” (https://www.dke.de/de/arbeitsfelder/industry/verwaltungsschale) [14.06.2022]* |