Abschlussprüfung Sommer 2025

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Einrichtung eines LoRaWAN Netzwerks mit App und Datenverarbeitung

Abgabetermin: Bad Neustadt, den 03.06.2025

Prüfungsbewerber:
Bernd Storath
Raiffeisenstr. 4
97618 Wülfershausen a.d. Saale

Ausbildungsbetrieb:
offizium next GmbH
Schulstraße 10
97616 Bad Neustadt a.d. Saale

Inhaltsverzeichnis

Abbildungsverzeichnis III

Tabellenverzeichnis IV

Listings V

Abkürzungsverzeichnis VI

1 Einleitung 1
1.1 Projektumfeld . . . 1
1.2 Projektziel . . . 1
1.3 Projektbegründung . . . 2
1.4 Projektschnittstellen . . . 2
1.5 Projektabgrenzung . . . 3

2 Projektplanung 3
2.1 Projektphasen . . . 3
2.2 Abweichungen vom Projektantrag . . . 3
2.3 Ressourcenplanung . . . 4
2.4 Entwicklungsprozess . . . 4

3 Analysephase 4
3.1 Ist-Analyse . . . 4
3.2 Wirtschaftlichkeitsanalyse . . . 5
3.2.1 „Make or Buy“-Entscheidung . . . 5
3.2.2 Projektkosten . . . 5
3.2.3 Amortisationsdauer . . . 6
3.3 Anwendungsfälle . . . 7
3.4 Lastenheft . . . 7

4 Entwurfsphase 7
4.1 Zielplattform . . . 7
4.2 Architekturdesign . . . 7
4.3 Entwurf der Benutzeroberfläche . . . 9
4.4 Datenmodell . . . 10
4.5 Geschäftslogik . . . 11
4.6 Maßnahmen zur Qualitätssicherung . . . 12
4.7 Deployment . . . 12
4.8 Pflichtenheft . . . 13

5 Implementierungsphase 13

5.1 Implementierung der Datenstrukturen . . . 13
5.2 Implementierung der Benutzeroberfläche . . . 13
5.3 Implementierung der Geschäftslogik . . . 14

6 Abnahmephase 15

7 Einführungsphase 15

8 Dokumentation 16

9 Fazit 16
9.1 Soll-/Ist-Vergleich . . . 16
9.2 Lessons Learned . . . 17
9.3 Ausblick . . . 17

Literaturverzeichnis 18

Eidesstattliche Erklärung 19

A Anhang i
A.1 Detaillierte Zeitplanung . . . i
A.2 Verwendete Ressourcen . . . ii
A.3 Use Case-Diagramm . . . iii
A.4 Lastenheft (Auszug) . . . iv
A.5 Mockups . . . v
A.6 ER-Modell . . . viii
A.7 Komponentendiagramm . . . viii
A.8 Pflichtenheft (Auszug) . . . ix
A.9 Migration: LORA_sensors . . . x
A.10 Model: LORASensors (Auszug) . . . xi
A.11 Screenshots der Anwendung . . . xiii
A.12 Sensorliste (Auszug) . . . xvi
A.13 gRPC-Proxy (Auszug) . . . xviii
A.14 Klasse: LoraDevicesController (Auszug) . . . xx
A.15 Benutzerdokumentation . . . xxiii

Abbildungsverzeichnis

1 Use Case-Diagramm iii
2 Übersicht aller Sensoren v
3 Detailseite eines Sensors vi
4 Historie eines Sensors vii
5 ER-Modell viii
6 Komponentendiagramm viii
7 Übersicht aller Sensoren xiii
8 Detailseite eines Sensors xiv
9 Historie eines Sensors xv

Tabellenverzeichnis

1 Zeitplanung 3
2 Kostenaufstellung 6
3 Soll-/Ist-Vergleich 16

Listings

1 Migration: LORA_sensors x
2 Model: LORASensors (Auszug) xii
3 Sensorliste (Auszug) xviii
4 gRPC-Proxy (Auszug) xx
5 Klasse: LoraDevicesController (Auszug) xxii

Abkürzungsverzeichnis

IDE	Integrated Development Environment
LoRaWAN	Long Range Wide Area Network
IoT	Internet of Things
PHP	Hypertext Preprocessor
MVC	Model View Controller
ORM	Object-Relational Mapping
SPA	Single-Page Application
ERM	Entity-Relationship-Modell
API	Application Programming Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
CRUD	Create, Read, Update, Delete

1 Einleitung

In der folgenden Projektdokumentation wird der Ablauf des Abschlussprojektes, das durch den Autor im Rahmen seiner Abschlussprüfung zum Fachinformatiker mit der Fachrichtung Anwendungsentwicklung durchgeführt wurde, erläutert.

1.1 Projektumfeld

Der Ausbildungsbetrieb offizium next GmbH ist ein zukunftsorientiertes Unternehmen mit Spezialisierung auf Marketing und digitale Transformation. Im Mittelpunkt der Geschäftstätigkeit steht die strategische und technische Beratung von Unternehmen, die ihre Geschäftsprozesse modernisieren, optimieren und digitalisieren möchten. offizium next GmbH versteht sich als Schnittstelle zwischen Technologie und Unternehmensentwicklung und begleitet seine Kunden auf dem Weg in die digitale Zukunft – von der ersten Analyse bis zur erfolgreichen Implementierung maßgeschneiderter Lösungen.

offizium next GmbH plant, s.connect zu vertreiben – ein System zur Einrichtung, Verwaltung und Nutzung von Long Range Wide Area Network (LoRaWAN) Netzwerken. Mit s.connect können Sensordaten effizient erfasst, verarbeitet, analysiert und visualisiert werden. Das System bietet eine ganzheitliche Lösung zur Digitalisierung und Überwachung von Prozessen in verschiedenen Anwendungsbereichen.

1.2 Projektziel

Das Projektziel bestand darin ein ganzheitliches System zur Erfassung, Verwaltung und Visualisierung von Sensordaten zu erstellen. Das System ermöglicht die zentrale Integration und Administration von Gateways und Sensoren über ein webbasiertes Management-Portal. Erfasste Daten werden über ein Backend verarbeitet, gespeichert und für die Visualisierung in einer intuitiven Web- und Mobile-App aufbereitet.

Folgende Aspekte lagen im Fokus:

- zentrale Verwaltung von Gateways und Sensoren
- zuverlässige Datenübertragung und -verarbeitung mithilfe des LoRaWAN Protokolls
- Backend zur Datenaufnahme, Analyse und Bereitstellung von Schnittstellen
- benutzerfreundliche Visualisierung der Sensordaten durch eine App für Android und iOS

Das Projekt zielte darauf ab die bestehenden Systeme (d. h. Backend, Web und App) um eine modulare, erweiterbare und benutzerorientierte Internet of Things (IoT)-Plattform zu erweitern, die in unterschiedlichen Anwendungsfeldern wie Smart City, Umweltmonitoring oder Industrie 4.0 flexibel eingesetzt werden kann.

1.3 Projektbegründung

Im Alltag vieler Unternehmen gehört das manuelle Ablesen von Strom-, Wasser- oder Verbrauchszählern noch immer zum Standard. Auch im Umfeld der offizium next GmbH gibt es Kunden, bei denen beispielsweise Stromzähler unterschiedlicher Stromkreisläufe regelmäßig von Mitarbeitenden vor Ort händisch abgelesen und dokumentiert werden müssen. Diese Vorgänge sind nicht nur zeitintensiv, sondern auch fehleranfällig, da sie auf manuelle Übertragung und Interpretation von Messwerten angewiesen sind.

Mit dem System s.connect sollte genau hier angesetzt werden: Statt wiederkehrende Routinetätigkeiten durch Personal durchführen zu lassen, können die benötigten Verbrauchsdaten automatisiert erfasst, übertragen, gespeichert und aufbereitet werden. Das reduziert nicht nur den Arbeitsaufwand für Mitarbeitende erheblich, sondern sorgt gleichzeitig für eine höhere Datenqualität und eine bessere Nachvollziehbarkeit. Falsche Ablesewerte, vergessene Protokolle oder uneinheitliche Formate werden somit weitestgehend vermieden.

1.4 Projektschnittstellen

Für die Übertragung, Verarbeitung und Darstellung der Sensordaten innerhalb des Systems waren mehrere technische Schnittstellen erforderlich, die unterschiedliche Protokolle und Dienste miteinander verbinden.

Die von den Sensoren erfassten Daten werden über LoRaWAN-Gateways an ChirpStack, eine Open-Source-Netzwerkserver-Plattform für LoRaWAN, übermittelt. Um diese Daten weiterzuverarbeiten, wurde ein Webhook eingerichtet, der die dekodierten Daten automatisch an das Backend-System weiterleitet. Dort erfolgt die Validierung, Speicherung und Aufbereitung der empfangenen Informationen.

Da das entwickelte Backend nur Hypertext Transfer Protocol (HTTP) unterstützt, ChirpStack jedoch ausschließlich über eine gRPC-Schnittstelle verwaltet werden kann, war die Implementierung einer Vermittlungsschicht erforderlich. Diese Komponente übernimmt die Aufgabe, eingehende Anfragen des Backends in entsprechende gRPC-Aufrufe umzuwandeln. So wird eine zentrale und einfache Verwaltung der LoRaWAN-Komponenten direkt über das Backend ermöglicht.

Zusätzlich wurde eine Representational State Transfer (REST)-Application Programming Interface (API) entwickelt, über die sowohl die Webanwendung als auch die mobile App auf die

verarbeiteten Sensordaten zugreifen können. Diese Schnittstelle stellt die Daten in strukturierter Form bereit und bildet die Grundlage für die nutzerfreundliche Visualisierung innerhalb der Benutzeroberflächen.

1.5 Projektabgrenzung

Da der Projektumfang begrenzt ist, ist die Erstellung der Weboberfläche zur einfachen Verwaltung kein Bestandteil des Projekts.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projekts standen insgesamt 80 Stunden zur Verfügung. Eine grobe Zeitplanung mit den Hauptphasen lässt sich aus Tabelle 1 entnehmen. Eine detailliertere Zeitplanung findet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

Projektphase	Geplante Zeit
Analysephase	5 h
Entwurfsphase	14 h
Implementierungsphase	39 h
Qualitätsmanagement	10 h
Einführungsphase	1 h
Erstellen der Dokumentation	11 h
Gesamt	**80 h**

Tabelle 1: Zeitplanung

2.2 Abweichungen vom Projektantrag

Im Projektantrag war vorgesehen, eine Weboberfläche zur einfachen Verwaltung der Gateways und Sensoren zu entwickeln. Aufgrund der begrenzten Zeit konnte diese Umsetzung jedoch nicht realisiert werden. Stattdessen wurde eine Postman Kollektion angelegt um die API aufrufen zu können.

2.3 Ressourcenplanung

Anschließend wurden alle Ressourcen im Anhang A.2: Verwendete Ressourcen auf Seite ii aufgelistet. Die Planung umfasst dabei neben allen Hard- und Softwareressourcen, die im Rahmen des Projektes verwendet wurden, auch das Personal. Da sich bei offizium next GmbH um ein kleines Unternehmen handelt, wurde darauf geachtet, dass möglichst wenig Kosten anfallen. Aus diesem Grund wurde das Projekt größtenteils mit kostenloser Software entwickelt. Bei kostenpflichtigen Services wurde darauf geachtet, nur die jeweils passende und notwendige Leistung zu erwerben.

2.4 Entwicklungsprozess

Vor Beginn der Umsetzung wählte der Autor einen geeigneten Entwicklungsprozess. Die Projektentwicklung orientiert sich grundsätzlich am Wasserfall-Modell, da die Anforderungen von offizium next GmbH weitgehend klar definiert sind und eine strukturierte, schrittweise Umsetzung ermöglichen. Dieses Modell bietet klare Phasen und erleichtert die Planung sowie Nachvollziehbarkeit des Projektfortschritts[1].

3 Analysephase

3.1 Ist-Analyse

Die Firma offizium next GmbH verfolgt das Ziel, eine ganzheitliche IoT-Plattform zu entwickeln. Im Rahmen dieser Ist-Analyse wird untersucht, in welchem Umfang bereits bestehende Systemkomponenten verfügbar sind und wo noch Entwicklungsbedarf besteht.

Aktuell bietet offizium next GmbH ein bestehendes Produkt über eine modulare App an. Diese App ist so konzipiert, dass sie flexibel um neue Funktionen erweitert werden kann, ohne dass eine vollständige Neuentwicklung erforderlich ist. Dies bildet eine solide Grundlage für die geplante Erweiterung zur IoT-Plattform.

Ein zentrales Element, das derzeit noch fehlt, ist jedoch ein System zur Erfassung und Verarbeitung von Sensordaten. Dieses bildet eine essenzielle Voraussetzung für die Realisierung der IoT-Funktionalitäten und muss daher im weiteren Projektverlauf neu entwickelt werden.

[1] Vgl. Christoph Friedrich [2025]

3.2 Wirtschaftlichkeitsanalyse

3.2.1 „Make or Buy"-Entscheidung

Die geplante IoT-Plattform soll zentrale Funktionen für die Datenerfassung und -verarbeitung übernehmen, insbesondere im Zusammenhang mit Sensordaten, die für zukünftige digitale Dienstleistungen eine strategische Rolle spielen. Diese Daten bilden eine wesentliche Grundlage für produktbezogene Auswertungen, Prozessautomatisierung und potenzielle Erlösmodelle.

Zudem ist eine enge Integration mit der bereits bestehenden modularen App notwendig, die unternehmensspezifisch entwickelt wurde. Die App-Architektur erfordert eine hohe Kompatibilität mit bestehenden Schnittstellen sowie eine flexible Erweiterbarkeit, um zukünftige Funktionalitäten effizient einbinden zu können. Eine externe Lösung würde potenzielle Risiken im Hinblick auf Datenschutz, Anpassungsfähigkeit und Abhängigkeit von Drittanbietern mit sich bringen.

Vor diesem Hintergrund wurde entschieden, die erforderlichen Systemkomponenten zur Sensordatenerfassung und -verarbeitung unternehmensintern zu entwickeln. Die Eigenentwicklung ermöglicht maximale Kontrolle über sensible Daten, volle technische Integration in bestehende Systeme sowie eine langfristig strategische Unabhängigkeit.

3.2.2 Projektkosten

Im Folgenden werden die voraussichtlichen Kosten des Projekts kalkuliert. Dabei sind sowohl die Personalkosten für den zuständigen Entwickler sowie weitere projektbeteiligte Mitarbeitende zu berücksichtigen, als auch die Aufwendungen für die unter 2.3 (Ressourcenplanung) aufgeführten Sachmittel und technischen Ressourcen. Alle relevanten Kostenpositionen fließen in die Gesamtkalkulation ein, um eine realistische Einschätzung des finanziellen Projektaufwands zu ermöglichen.

Die exakten Personalkosten können aus datenschutzrechtlichen Gründen nicht offengelegt werden. Daher erfolgt die Kalkulation auf Basis realitätsnaher Durchschnittswerte. Für einen voll ausgebildeten Mitarbeitenden werden die Gesamtkosten für das Unternehmen mit 60,00 € pro Stunde angesetzt. Die Kosten für einen Auszubildenden werden auf Grundlage branchenüblicher Vergütungen und betrieblicher Aufwendungen mit 12,00 € pro Stunde kalkuliert. Zusätzlich wird für die Nutzung von Hard- und Software ein pauschaler Stundensatz von 15,00 € berücksichtigt.

Für das Projekt ist ein Gesamtaufwand von 80 Arbeitsstunden eingeplant. Auf dieser Grundlage lassen sich die voraussichtlichen Gesamtkosten näherungsweise ermitteln.

Eine Aufstellung der Kosten befindet sich in Tabelle 2 und sie betragen insgesamt 2739,20 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	80 h	12 € + 15 € = 27 €	2160 €
Projektbewertung	1 h	25 € + 15 € = 40 €	40 €
Abnahmetest	1 h	25 € + 15 € = 40 €	40 €
Code-Review	2 h	25 € + 15 € = 40 €	80 €
			2320 €

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Da es sich bei der IoT-Plattform um ein neu entwickeltes Produkt handelt, kann die Amortisationsdauer nicht auf Basis von Einsparungen durch die Ablösung eines bestehenden Systems berechnet werden. Stattdessen orientiert sich die Kalkulation an den erwarteten Einnahmen aus dem geplanten monatlichen Abomodell.

Die einmaligen Projektkosten belaufen sich auf insgesamt 2.320 €. Hinzu kommen laufende Betriebskosten von 15 € pro Monat (10 € für Serverbetrieb und 5 € für Datensicherung) Diese Kosten fallen pauschal für das Gesamtsystem an – unabhängig von der Anzahl der Nutzer.

Bei einem monatlichen Verkaufspreis von 20 € pro Kunde ergeben sich folgende Szenarien:

- Bei 1 Kunde: Deckungsbeitrag = 20 € - 15 € = 5 € pro Monat
- Bei 5 Kunden: Deckungsbeitrag = $5 \times 20\,€ - 15\,€ = 85\,€$ pro Monat
- Bei 10 Kunden: Deckungsbeitrag = $10 \times 20\,€ - 15\,€ = 185\,€$ pro Monat

Die Amortisationsdauer bei 10 Kunden berechnet sich wie folgt:

$$\frac{2.320\,€}{185\,€} \approx 12,54 \text{ Monate} \tag{1}$$

Somit wäre die Investition bei zehn gleichzeitig zahlenden Kunden bereits nach etwas über einem Jahr vollständig amortisiert. Mit wachsender Kundenzahl verkürzt sich die Amortisationsdauer entsprechend weiter.

Die Analyse zeigt, dass das Projekt bei realistischer Kundengewinnung wirtschaftlich ist. Die Investitionskosten können innerhalb eines angemessenen Zeitraums amortisiert werden, weshalb die Umsetzung aus wirtschaftlicher Sicht empfohlen wird.

3.3 Anwendungsfälle

Um einen ersten Überblick darüber zu gewinnen, wie Kunden und Mitarbeitende mit der Anwendung interagieren sollen und welche Anwendungsfälle aus Sicht der Endnutzer abgedeckt werden müssen, wurde im Rahmen der Analysephase ein Use-Case-Diagramm erstellt. Dieses ist im Anhang A.3: Use Case-Diagramm auf Seite iii dargestellt.

3.4 Lastenheft

Zum Abschluss der Analysephase wurde gemeinsam mit dem Kunden ein Lastenheft erstellt. Darin sind sämtliche Anforderungen des Auftraggebers an die zu entwickelnde Anwendung festgehalten. Ein Auszug daraus befindet sich im Anhang A.4: Lastenheft (Auszug) auf Seite iv.

4 Entwurfsphase

4.1 Zielplattform

Wie bereits unter 1.2 (Projektziel) erwähnt, sollen bestehende Systeme erweitert werden. Dies macht die Entwicklung einfacher, da bestehende Funktionalitäten aufgegriffen werden können. Daten aus dem System liegen in einer MySQL Datenbank. Auf ein neues Datenbanksystem kann daher verzichtet werden.

Als Programmiersprache im Backend wird Hypertext Preprocessor (PHP) verwendet, da das bestehende System dies vorgibt. Die App verwendet NativeScript-Vue somit wird die App in JavaScript und Vue.js programmiert.

4.2 Architekturdesign

Im Backend der Anwendung kommt das PHP-Framework Laravel zum Einsatz, welches auf dem Model View Controller (MVC)-Architekturmuster basiert. Diese bewährte Architektur wird durch eine zusätzliche Komponentenstruktur erweitert, um eine modulare und wartbare Codebasis zu ermöglichen.

Die Anwendung gliedert sich grundsätzlich in drei zentrale Schichten:

Die Model-Schicht bildet sowohl die zugrunde liegenden Datenstrukturen als auch die fachliche Logik der Anwendung ab. Dabei wird Eloquent verwendet, ein Active Record Object-Relational Mapping (ORM) von Laravel. Das bedeutet, dass die Datenbankoperationen direkt im jeweiligen Model definiert und ausgeführt werden, anstatt ein separates Repository-Pattern zu verwenden.

So sind Funktionen wie das Abrufen, Speichern oder Löschen von Datensätzen eng mit den jeweiligen Datenmodellen verknüpft.

Die View-Schicht ist in einer klassischen Laravel-Anwendung für die Darstellung der Benutzeroberfläche verantwortlich. In diesem Projekt wird jedoch eine moderne Single-Page Application (SPA) bzw. App auf Basis von Vue.js eingesetzt, wodurch die klassische Laravel-View nur noch eine untergeordnete Rolle spielt. Sie wird primär für die Darstellung von E-Mail-Templates oder anderen serverseitig generierten Inhalten verwendet.

Die Controller bilden die Vermittlung zwischen Benutzeranfragen und der Geschäftslogik. Sie nehmen eingehende HTTP-Anfragen entgegen, verarbeiten diese, interagieren mit den entsprechenden Models und geben eine passende Antwort zurück. Darüber hinaus sind die Controller auch für die Validierung von Benutzereingaben zuständig, sowohl hinsichtlich formaler Korrektheit als auch, falls erforderlich, in Bezug auf deren inhaltliche Sinnhaftigkeit.

Durch diese klar strukturierte Trennung der Verantwortlichkeiten wird eine saubere, wartbare und erweiterbare Codebasis sichergestellt, die sowohl die Entwicklung neuer Features als auch die Fehlerbehebung erleichtert.

Ein weiteres zentrales Element der Systemarchitektur ist der gRPC-Proxy, welcher als Vermittler zwischen dem REST-basierten Backend und dem gRPC-basierten ChirpStack-Server fungiert. Seine Hauptaufgabe besteht darin, eingehende REST-Anfragen aus dem Backend entgegenzunehmen und diese in äquivalente gRPC-Aufrufe zu übersetzen. Dabei handelt es sich um eine rein technische Übersetzungsschicht ohne eigene Geschäftslogik. Aus diesem Grund wurde bewusst auf die Implementierung eines komplexen Software-Patterns verzichtet, um die Komponente so einfach und wartungsarm wie möglich zu halten.

Für die Umsetzung des Proxys wird das Webframework h3 eingesetzt. h3 zeichnet sich durch seine hohe Performance, geringe Größe sowie die hervorragende Unterstützung von TypeScript aus. Der Einsatz von TypeScript ermöglicht es, viele potenzielle Fehler bereits beim Kompilieren zu identifizieren und damit die Stabilität und Wartbarkeit des Codes deutlich zu erhöhen. Zur Validierung der eingehenden REST-Anfragen wird die Bibliothek `zod` verwendet. Dadurch kann sichergestellt werden, dass fehlerhafte oder unvollständige Daten nicht an ChirpStack weitergeleitet werden. Stattdessen erhält das Backend unmittelbar eine aussagekräftige Fehlermeldung, was die Nutzerfreundlichkeit und Robustheit der Schnittstelle erhöht.

Die letzte serverseitige Komponente ist ChirpStack, ein Open-Source Netzwerk- und Applikationsserver für LoRaWAN. ChirpStack ermöglicht die Verwaltung von Sensoren und Gateways sowie die Organisation und Steuerung von LoRaWAN-Sitzungen. Zusätzlich bietet die Plattform die Möglichkeit, benutzerdefinierte Decoder-Skripte zu hinterlegen. Diese werden genutzt, um die von den Sensoren gesendeten Binärdaten automatisiert in strukturierte JSON-Objekte umzuwandeln. Somit stellt ChirpStack eine zentrale und flexible Plattform für den zuverlässigen Betrieb und die Integration von LoRaWAN-basierten IoT-Systemen dar.

Die App verwendet NativeScript-Vue, ein Framework, das Webentwicklern ermöglicht, native mobile Apps mit vertrauten Webtechnologien wie JavaScript zu erstellen. Ein großer Vorteil von NativeScript ist, dass der Code einmal geschrieben und anschließend sowohl für iOS als auch Android verwendet werden kann.

NativeScript-Vue kombiniert NativeScript mit dem JavaScript Webframework Vue.js. Vue.js basiert auf einem reaktiven Datenmodell und dem Virtual DOM. Reaktivität bedeutet, dass sich die Benutzeroberfläche automatisch aktualisiert, sobald sich die zugrunde liegenden Daten ändern. Der Virtual DOM optimiert diesen Prozess, indem er eine virtuelle Kopie der Benutzeroberfläche erstellt und nur die tatsächlich veränderten Teile neu rendert. Dadurch werden Updates effizient und ressourcenschonend durchgeführt.

Was NativeScript-Vue besonders macht, ist, dass es keine HTML-Webansicht auf dem Gerät anzeigt. Stattdessen werden alle UI-Elemente tatsächlich nativ gerendert, was zu einer deutlich besseren Performance und einem authentischen Look-and-Feel auf der jeweiligen Plattform führt.

4.3 Entwurf der Benutzeroberfläche

Da das Nutzererlebnis bei mobilen Anwendungen von zentraler Bedeutung ist, wurde frühzeitig ein besonderer Fokus auf die Gestaltung der Benutzeroberfläche gelegt. Zu diesem Zweck wurden zunächst Mockups entwickelt, die anschließend in enger Abstimmung mit dem Kunden mehrfach überarbeitet und verbessert wurden. Die finalen Entwürfe sind im Anhang A.5: Mockups auf Seite v dokumentiert.

Da die Anwendung in eine bestehende App integriert wird, deren grundlegendes Layout bereits definiert ist, mussten diese bei der Gestaltung der Benutzeroberfläche berücksichtigt werden. Insbesondere sind der obere Seitenbereich (Header) sowie das Navigationsmenü fest vorgegeben. Diese Strukturen bilden den äußeren Rahmen des App-Designs und wurden im Entwurf entsprechend berücksichtigt.

Die Startseite der Anwendung bietet eine Übersicht über alle vorhandenen Sensoren. Jeder Sensor wird in einer eigenen Kachel dargestellt. Gemäß der mit dem Kunden abgestimmten Gestaltung ist die Kachel weiß hinterlegt und hebt sich damit optisch von dem grauen Hintergrund ab. Jede dieser Kacheln enthält folgende Informationen:

- den Namen des Sensors
- die Art des Sensors visualisiert durch ein passendes Symbol
- den aktuellen Messwert
- den Batteriestand dargestellt durch ein Symbol

- den Zeitpunkt der letzten Datenübertragung

Wählt der Nutzer einen Sensor aus, wird eine Detailansicht geöffnet. Dort werden die oben genannten Informationen erneut angezeigt, wobei der Batteriestand deutlicher hervorgehoben wird. Falls der Sensor zusätzliche Messwerte liefert, die nicht unmittelbar relevant, aber dennoch informativ sind, werden diese in dieser Detailansicht ebenfalls dargestellt. Darüber hinaus können Nutzer einzelne Sensoren als Favoriten markieren. Diese Favoriten erscheinen auf der Startseite ganz oben, um einen schnellen Zugriff zu ermöglichen.

Ein zentrales Element der Benutzeroberfläche ist die Darstellung historischer Sensordaten. Diese erfolgt mithilfe eines Säulendiagramms, das in drei zeitliche Ansichten unterteilt ist: Tag, Woche und Jahr.

- In der Tagesansicht werden die Daten in zwei-Stunden-Intervallen gruppiert.
- In der Wochenansicht erfolgt die Aggregation der Daten nach Wochentagen.
- In der Jahresansicht wird der Verlauf über die Monate hinweg dargestellt.

Zur besseren Vergleichbarkeit wird in jeder dieser Ansichten zusätzlich der jeweilige Vorzeitraum eingeblendet. Dadurch können Veränderungen im zeitlichen Verlauf unmittelbar erkannt und eingeordnet werden.

Insgesamt verfolgt der Entwurf der Benutzeroberfläche das Ziel, eine klare, übersichtliche und nutzerfreundliche Darstellung aller relevanten Informationen sicherzustellen.

4.4 Datenmodell

Das Datenmodell der Anwendung umfasst die zentralen Entitäten Gateway, Gerät, Sensor, Kategorie, Wert, Geräte-Profil, Hersteller sowie Sensortyp. Diese bilden die Grundlage für die strukturierte Erfassung, Zuordnung und Auswertung der erfassten Sensordaten.

Ein Gerät stellt einen physischen Sensor bzw. ein Sensormodul dar, das über eine eindeutige devEUI identifiziert wird. Es fungiert als logische Einheit, die mit einem oder mehreren Sensoren ausgestattet ist und regelmäßig Messwerte an das System übermittelt. Jedes Gerät ist einem Geräte-Profil zugeordnet, das allgemeine Informationen wie unterstützte Sensortypen oder Kommunikationsverhalten enthält.

Das Geräte-Profil wiederum ist einem bestimmten Hersteller zugeordnet, welcher das Gerät entwickelt oder produziert hat. Durch diese Beziehung können gerätespezifische Merkmale, Abhängigkeiten und Kompatibilitäten nachvollziehbar dokumentiert und verwaltet werden. Ein Geräte-Profil definiert eine Menge an unterstützten Sensortypen. Diese Typen geben an, welche

Arten von Sensoren in einem Gerät dieses Profils verbaut sind. Beim Anlegen eines neuen Geräts im System werden automatisch die zugehörigen Sensoren entsprechend dem Geräte-Profil erstellt.

Jeder Sensor steht für eine konkrete Messeinheit innerhalb eines Geräts. Ein Gerät kann mehrere Sensoren enthalten, da in modernen Umweltsensoren meist mehrere Messgrößen parallel erfasst werden. Typische Beispiele sind Temperatur, Luftfeuchtigkeit, Helligkeit und CO_2-Konzentration. Ein Sensor kann mehreren Kategorien zugeordnet sein, welche thematische oder funktionale Gruppierungen darstellen. Kategorien dienen unter anderem der Strukturierung und Aggregation von Sensordaten. Ein Sensor speichert kontinuierlich Werte, die jeweils einen gemessenen Messwert sowie den zugehörigen Zeitstempel enthalten. Zusätzlich werden Durchschnittswerte berechnet und gespeichert um aggregierte Informationen über bestimmte Zeitintervalle hinweg (z. B. stündlich, täglich) effizient verfügbar zu machen. Diese Durchschnittswerte ermöglichen eine performante Visualisierung historischer Verläufe ohne aufwendig alle Einzelwerte laden zu müssen.

Diese Beziehungen sind im Anhang A.6: ER-Modell auf Seite viii als Entity-Relationship-Modell (ERM) dargestellt.

4.5 Geschäftslogik

Vor Beginn der Implementierung wurde ein Komponentendiagramm erstellt, um das Zusammenspiel der verschiedenen Systembestandteile übersichtlich darzustellen und die zugrunde liegende Architektur zu visualisieren.

Der Einstiegspunkt für den Nutzer ist die mobile App. Diese kommuniziert ausschließlich mit dem Laravel-Backend, welches als zentrale Steuerungseinheit fungiert. Über das Backend können unter anderem Sensordaten sowie deren Historie abgerufen werden.

Darüber hinaus dient das Laravel-Backend als Schnittstelle für interne und externe Benutzer, beispielsweise für Mitarbeitende, die über eine definierte API, mithilfe von Postman, Aktionen wie das Anlegen neuer Gateways durchführen können.

Müssen im Zuge solcher Aktionen auch Änderungen im ChirpStack vorgenommen werden, z. B. das parallele Anlegen eines Geräts oder Gateways, wird die entsprechende Anfrage intern an den gRPC-Proxy weitergeleitet. Dieser übernimmt die Konvertierung der REST-Anfragen in gRPC-Aufrufe und leitet diese an ChirpStack weiter, welches die entsprechenden Daten ebenfalls in seinem System speichert.

Eine visuelle Darstellung dieser Systemkomponenten und ihrer Beziehungen ist im Anhang A.7: Komponentendiagramm auf Seite viii zu finden.

4.6 Maßnahmen zur Qualitätssicherung

Im Rahmen dieses Projekts wurde bewusst auf automatisierte Tests oder den Einsatz spezieller Qualitätssicherungs-Tools verzichtet. Stattdessen erfolgte die Qualitätssicherung in Form manueller Tests sowie durch regelmäßige Code-Reviews während der Entwicklungsphase.

Die manuelle Überprüfung ermöglichte eine praxisnahe Validierung der Funktionalitäten, insbesondere im Hinblick auf die Benutzeroberfläche und die Interaktion mit dem Backend. Zusätzlich trugen die Code-Reviews dazu bei, mögliche Fehler frühzeitig zu erkennen, die Codequalität zu verbessern und die Einhaltung von Projektstandards sicherzustellen.

4.7 Deployment

Für den Betrieb von ChirpStack sowie dem gRPC-Proxy wurde ein eigener Server benötigt, da beide Dienste unabhängig von der Hauptanwendung laufen und bestimmte Netzwerk- sowie Systemressourcen erfordern. Aus diesem Grund wurde ein Cloud-Server bei Ionos angemietet. Da sowohl ChirpStack als auch der gRPC-Proxy vergleichsweise geringe Systemanforderungen aufweisen, genügte eine Konfiguration mit einer einzelnen CPU und 1 GB RAM, um eine stabile Ausführung sicherzustellen.

Als Betriebssystem kam Debian 12 zum Einsatz. Diese Version stellt die derzeit aktuelle stabile Veröffentlichung der Debian-Distribution dar und wurde aufgrund ihrer Zuverlässigkeit, Sicherheit sowie der guten Dokumentation gewählt.[2]

Nach der Grundkonfiguration des Servers erfolgte die Installation von ChirpStack gemäß der offiziellen Dokumentation. Dabei wurden auch alle erforderlichen Abhängigkeiten eingerichtet, darunter der Message-Broker Mosquitto, die In-Memory-Datenbank Redis sowie das relationale Datenbanksystem PostgreSQL.

Für den Betrieb von dem gRPC-Proxy wurde zudem Node.js installiert. Zur Prozessverwaltung und zur Sicherstellung der dauerhaften Verfügbarkeit des Proxys wird `pm2` verwendet. Dieses Tool ermöglicht es, den Dienst automatisch beim Systemstart auszuführen und bei unerwarteten Fehlern oder Abstürzen selbstständig neu zu starten.

Um sowohl den Zugriff auf ChirpStack als auch auf den gRPC-Proxy über das Internet zu ermöglichen, wurde ein Reverse Proxy mit Nginx eingerichtet. In Kombination mit Certbot konnten zudem automatisierte SSL-Zertifikate von Let's Encrypt eingerichtet werden, wodurch eine verschlüsselte und sichere Kommunikation über HTTPS gewährleistet ist.

Diese Infrastruktur gewährleistet einen zuverlässigen und wartungsarmen Betrieb der zentralen Backend-Komponenten des Projektes.

[2] Vgl. Roberto Scipione [2023]

4.8 Pflichtenheft

Basierend auf den erarbeiteten Entwürfen wurde zum Abschluss der Entwurfsphase ein Pflichtenheft erstellt. Dieses baut auf dem zuvor definierten Lastenheft auf und konkretisiert die Umsetzung der in Abschnitt 3.4 (Lastenheft) festgelegten Anforderungen. Es dient sowohl während der Entwicklungsphase als Orientierungshilfe als auch am Projektende zur Überprüfung, ob sämtliche Anforderungen erfüllt wurden. Ein Auszug des Pflichtenhefts ist im Anhang A.8: Pflichtenheft (Auszug) auf Seite ix dargestellt.

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

Die Umsetzung des Datenbankschemas orientierte sich weitgehend an dem in Abschnitt 4.4 beschriebenen ERM. Lediglich kleinere Anpassungen waren notwendig, um spezifische technische Anforderungen und Rahmenbedingungen zu erfüllen.

Zur besseren Strukturierung und Übersichtlichkeit innerhalb der Datenbank wurden die Tabellen eines Moduls mit einem gemeinsamen Präfix versehen. Im vorliegenden Fall wurde das Präfix `LORA` verwendet, wodurch eine eindeutige Zuordnung der Tabellen zum jeweiligen Modul gewährleistet ist.

Darüber hinaus wurden zusätzliche Spalten vorgesehen, beispielsweise zur Speicherung von Informationen über Dezimalstellen oder andere potenzielle Erweiterungen, um künftige Anforderungen flexibel abbilden zu können.

Die Erstellung der Datenbankstruktur erfolgte mithilfe sogenannter Migrationen, die eine nachvollziehbare und versionierte Entwicklung der Datenbank ermöglichen. Die Migration zur Tabelle `LORA_sensors` ist im Anhang A.9: Migration: LORA_sensors auf Seite x aufgeführt.

Parallel zur Definition der Tabellen wurden auch die zugehörigen Model-Klassen implementiert, welche die Verbindung zwischen Anwendung und Datenbank darstellen. Diese Modelle bilden die logische Repräsentation der Datenstrukturen innerhalb der Applikation ab. Ein Auszug des Models `LORASensor` findet sich im Anhang A.10: Model: LORASensors (Auszug) auf Seite xi.

5.2 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wurde auf Grundlage der im Abschnitt 4.3 dargestellten Mockups entwickelt. Für die Umsetzung kam das in Abschnitt 4.2 vorgestellte Framework NativeScript-Vue zum Einsatz, das eine Kombination aus der Vue.js Syntax und nativer App-Entwicklung ermöglicht.

Vue verwendet eine HTML-ähnliche Syntax, die durch eigene Direktiven wie `v-for` ergänzt wird, um Daten dynamisch darzustellen. Beispielsweise kann mit dieser Direktive einfach über eine Liste von Sensoren iteriert werden:

```
1 <li v-for="sensor in sensors"></li>
```

In nativen Anwendungen wie denen, die mit NativeScript erstellt werden, spielt die Performance jedoch eine entscheidende Rolle. Da keine klassischen HTML-Elemente verfügbar sind und eine hohe Effizienz beim Rendering notwendig ist, kann obiges Beispiel nicht direkt verwendet werden. Stattdessen wird die Komponente `CollectionView` eingesetzt. Diese bietet den Vorteil, dass nur die tatsächlich sichtbaren Listenelemente gerendert werden, während Elemente außerhalb des sichtbaren Bereichs verworfen werden. Dadurch wird der Speicherverbrauch reduziert und die Anwendung bleibt auch bei großen Datenmengen performant:

```
1 <CollectionView :items="dataItems"></CollectionView>
```

Für das Design und Styling der Benutzeroberfläche wird CSS verwendet. Einen Eindruck des finalen Designs geben die im Anhang A.11: Screenshots der Anwendung auf Seite xiii enthaltenen Screenshots, die den Zustand der App nach Abschluss der Implementierungsphase dokumentieren.

Ein Auszug aus dem Quelltext für die Liste aller Sensoren befindet sich im Anhang A.12: Sensorliste (Auszug) auf Seite xvi.

5.3 Implementierung der Geschäftslogik

Die Geschäftslogik bildet das zentrale Bindeglied zwischen den fachlichen Anforderungen und der technischen Umsetzung. Sie steuert die Verarbeitung der Anwendungsdaten und definiert die Abläufe zur Umsetzung der funktionalen Anforderungen. Im Rahmen dieses Projekts wurden im ersten Schritt sämtliche in Abschnitt 4.5 beschriebenen Komponenten des Laravel-Servers realisiert.

Für jede relevante Model-Klasse wurde ein zugehöriger Controller erstellt, welcher die standardmäßigen Methoden zur Datenverarbeitung gemäß dem CRUD-Paradigma bereitstellt. In Laravel entspricht dies beispielsweise der Methode `store` für das Erstellen neuer Einträge (Create).

Sobald eine Interaktion mit ChirpStack erforderlich war, wurde die jeweilige Funktionalität gezielt im gRPC-Proxy umgesetzt. Ziel war es, den Proxy schlank zu halten und ausschließlich um notwendige Funktionen zu erweitern. Ein entsprechender Auszug der Proxy-Implementierung ist im Anhang A.13: gRPC-Proxy (Auszug) auf Seite xviii dokumentiert.

Zur Kommunikation zwischen Laravel und dem gRPC-Proxy wurde eine dedizierte Service-Klasse entwickelt. Diese übernimmt sowohl die Authentifizierung als auch den Versand der HTTP-Anfragen an den Proxy. Dadurch wird eine klare Trennung der Verantwortlichkeiten erreicht und die Anbindung an externe Dienste zentralisiert.

Ein Controller-Auszug ist im Anhang A.14: Klasse: LoraDevicesController (Auszug) auf Seite xx dargestellt. Dieser veranschaulicht unter anderem das Anzeigen und Anlegen eines Geräts. Im Rahmen des Anlegens wird dabei auch eine Anfrage an den gRPC-Proxy gesendet, um das entsprechende Gerät innerhalb von ChirpStack zu registrieren.

6 Abnahmephase

Vor der finalen Präsentation der Anwendung im Fachbereich wurde ein Code-Review durch den Ausbilder durchgeführt. Dabei lag der Fokus auf fachlicher und technischer Korrektheit sowie auf der Verständlichkeit des Codes, insbesondere hinsichtlich der Benennung von Variablen und der Komplexität einzelner Methoden. Da der Ausbilder mit dem Projekt vertraut war, war keine zusätzliche Einführung notwendig. Anmerkungen wurden direkt als TODO Kommentare im Code vermerkt und in einer anschließenden Besprechung geklärt. Ziel war es, die Codequalität zu verbessern und die Verständlichkeit für andere Entwickler zu erhöhen.

7 Einführungsphase

Für die finale Einführung der Anwendung wurde der entsprechende Git-Branch mit dem Hauptzweig zusammengeführt. Anschließend erfolgte ein manueller `git pull` auf dem Produktionsserver, wodurch die neu implementierte Funktionalität im produktiven Backend verfügbar wurde.

Auch der gRPC-Proxy wurde über Git auf den aktuellen Stand gebracht. Im Anschluss daran wurde die Konfiguration für den automatischen Start des Dienstes mithilfe von `pm2` eingerichtet, um einen stabilen und dauerhaften Betrieb sicherzustellen.

Auf eine separate Benutzerschulung wurde bewusst verzichtet, da die Anwendung gezielt so gestaltet wurde, dass sie möglichst intuitiv bedienbar ist. Durch die klare Benutzerführung und eine reduzierte Komplexität der Oberfläche sollte eine Einarbeitung ohne zusätzliche Anleitung möglich sein.

8 Dokumentation

Die Dokumentation der Anwendung gliedert sich in die Projektdokumentation sowie ein Benutzerhandbuch. Die Projektdokumentation beschreibt die im Verlauf des Projekts durchlaufenen Phasen und getroffenen Entscheidungen.

Das Benutzerhandbuch richtet sich an Endanwender und soll die Bedienung der Anwendung erleichtern. Es enthält eine Übersicht über die wichtigsten Funktionen sowie eine schrittweise Anleitung zur Navigation innerhalb der Anwendung. Unterstützt durch Screenshots wird die Nutzung anschaulich erklärt. Ein Auszug ist im Anhang A.15: Benutzerdokumentation auf Seite xxiii zu finden.

Auf eine separate Entwicklerdokumentation wurde bewusst verzichtet. Stattdessen wurde der Quellcode umfangreich kommentiert, sodass sich Entwickler bei einer späteren Weiterentwicklung oder Anpassung direkt im Code orientieren können. In der Entwurfsphase wurde ein Komponentendiagramm erstellt, das die Struktur und die Abhängigkeiten der wichtigsten Systemkomponenten visualisiert. Dieses dient als Überblick für neue Entwickler und ist im Anhang A.7: Komponentendiagramm auf Seite viii enthalten.

9 Fazit

9.1 Soll-/Ist-Vergleich

Durch eine sorgfältige Anforderungsanalyse und eine strukturierte Entwurfsphase konnte das Projekt vollständig nach Plan umgesetzt werden. Zu Beginn wurden vergleichbare Anwendungen und Systeme umfassend recherchiert, um bewährte Konzepte und Technologien in die Entwicklung einfließen zu lassen. Diese Vorarbeit bildete die Grundlage für eine effiziente und zielgerichtete Umsetzung.

Wie Tabelle 3 zeigt, konnten alle Phasen des Projekts ohne Abweichungen im vorgesehenen Zeitraum abgeschlossen werden.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	5 h	5 h	
Entwurfsphase	14 h	14 h	
Implementierungsphase	39 h	39 h	
Qualitätsmanagement	10 h	10 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	11 h	11 h	
Gesamt	80 h	80 h	

Tabelle 3: Soll-/Ist-Vergleich

9.2 Lessons Learned

Im Verlauf des Projekts konnte der Prüfling durch eigenständige Recherchen sowie die Entwicklung eigener Lösungsansätze wertvolle praktische Erfahrungen sammeln. Eine zentrale Erkenntnis war die Bedeutung einer fundierten Planung und eines durchdachten Entwurfs. Diese bilden die Grundlage für eine strukturierte und effiziente Umsetzung, wodurch Projekte reibungsloser realisiert und Ressourcen gezielt eingesetzt werden können.

Positiv zu bewerten ist außerdem die Fähigkeit, auftretende Probleme eigenständig zu analysieren und passende Lösungen zu entwickeln. Dies stellt eine wichtige Kompetenz für die erfolgreiche Durchführung zukünftiger Projekte dar.

9.3 Ausblick

Da die Anwendung modular und erweiterbar konzipiert wurde, sind zukünftige Erweiterungen problemlos möglich. Erste Ideen für zusätzliche Funktionen bestehen bereits, beispielsweise die Entwicklung einer Weboberfläche, die das Anlegen und Verwalten von Sensordaten erleichtert. Auch die Möglichkeit, Sensordaten mithilfe benutzerdefinierter Formeln direkt in der Anwendung zu modifizieren, ist angedacht.

Darüber hinaus bestehen Potenziale für eine technische Weiterentwicklung, etwa durch die Integration weiterer Übertragungstechnologien. Ein mögliches Szenario ist die Einbindung von NB-IoT, einem auf Mobilfunk basierenden IoT-Netzwerk, das im Gegensatz zu LoRaWAN keine eigene Gateway-Infrastruktur erfordert. Dadurch ließen sich neue Anwendungsbereiche erschließen und die Flexibilität der Lösung weiter erhöhen.

Literaturverzeichnis

Christoph Friedrich 2025

CHRISTOPH FRIEDRICH: *Wasserfallmodell: Definition, Anwendung, Vor- und Nachteile*. Version: Februar 2025. https://alltena.com/de/blog/wasserfallmodell, Abruf: 20.05.2025

Roberto Scipione 2023

ROBERTO SCIPIONE: *Linux Debian the Stable and Secure Distribution*. Version: Juni 2023. https://www.robertoscipione.com/en/blog/linux-debian-the-stable-and-secure-distribution, Abruf: 22.05.2025

Eidesstattliche Erklärung

Ich, Bernd Storath, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

Einrichtung eines LoRaWAN Netzwerks mit App und Datenverarbeitung

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Bad Neustadt, den 03.06.2025

_____

Bernd Storath

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase		**5 h**
1. Analyse des Ist-Zustands	1 h	
2. Wirtschaftlichkeitsanalyse	1 h	
3. Erstellen eines „Use-Case“-Diagramms	1 h	
4. Erstellen des Lastenhefts	2 h	
Entwurfsphase		**14 h**
1. Entwerfen der Benutzeroberfläche	2 h	
2. Entwerfen der Datenbankstruktur (ERM)	3 h	
3. Einrichten des Servers	2 h	
4. Erstellen eines Komponentendiagramms	3 h	
5. Erstellen des Pflichtenhefts	4 h	
Implementierungsphase		**39 h**
1. Implementieren der Migrationen	1 h	
2. Implementieren der Model-Klassen	1 h	
3. Programmieren des Laravel Backends	17 h	
3.1. Anlegen der Controller	1 h	
3.2. Implementieren der Controller	14 h	
3.3. Implementieren der Service-Klasse	2 h	
4. Programmieren des gRPC-Proxys	4 h	
4.1. Anlegen der Struktur	1 h	
4.2. Entwickeln der gRPC Ebene	1 h	
4.3. Implementieren der Endpunkte	2 h	
5. Programmieren der App	16 h	
5.1. Implementieren der Schnittstelle	1 h	
5.2. Erstellen des Vuex Zustands	1 h	
5.3. Implementieren der Sensorübersicht	4 h	
5.4. Anlegen der Detailseite	4 h	
5.5. Implementieren der Historie	6 h	
Qualitätsmanagement		**10 h**
1. Code-Review durch Autor	4 h	
2. White-Box-Test durch Autor	3 h	
3. Testen der App	1 h	
4. Code-Review der Fachabteilung	2 h	
Einführungsphase		**1 h**
1. Einführung	1 h	
Erstellen der Dokumentation		**11 h**
1. Erstellen der Benutzerdokumentation	2 h	
2. Erstellen der Projektdokumentation	7 h	
3. Kommentieren des Codes	2 h	
Gesamt		**80 h**

A.2 Verwendete Ressourcen

Hardware

- Arbeitsplatz mit Laptop und externem Monitor
- Ionos Debian 12 - Cloudserver für ChirpStack und gRPC-Proxy
- Apple iPhone X - Gerät zum Testen der App auf iOS
- Google Pixel 9 Pro - Gerät zum Testen der App auf Android

Software

- PhpStorm - Integrated Development Environment (IDE) für Front- und Backendsprachen wie z. B. PHP
- macOS 15 - Betriebssystem zur Entwicklung
- ChirpStack - LoRaWAN Open-Source-Netzwerkserver-Plattform
- Visual Studio Code - Quelltext-Editor zum Programmieren und Erstellung der Projektdokumentation
- Podio - Projektmanagementsoftware
- Git - verteilte Versionsverwaltung
- NativeScript-Vue - Framework zum Entwickeln von Apps
- Laravel - PHP Webframework
- h3 - Webframework für JavaScript
- MySQL - Datenbankmanagementsystem
- Postman - Plattform zum Testen einer API

Personal

- Auszubildender - Umsetzung des Projekts
- Anwendungsentwickler – Review des Codes

A.3 Use Case-Diagramm

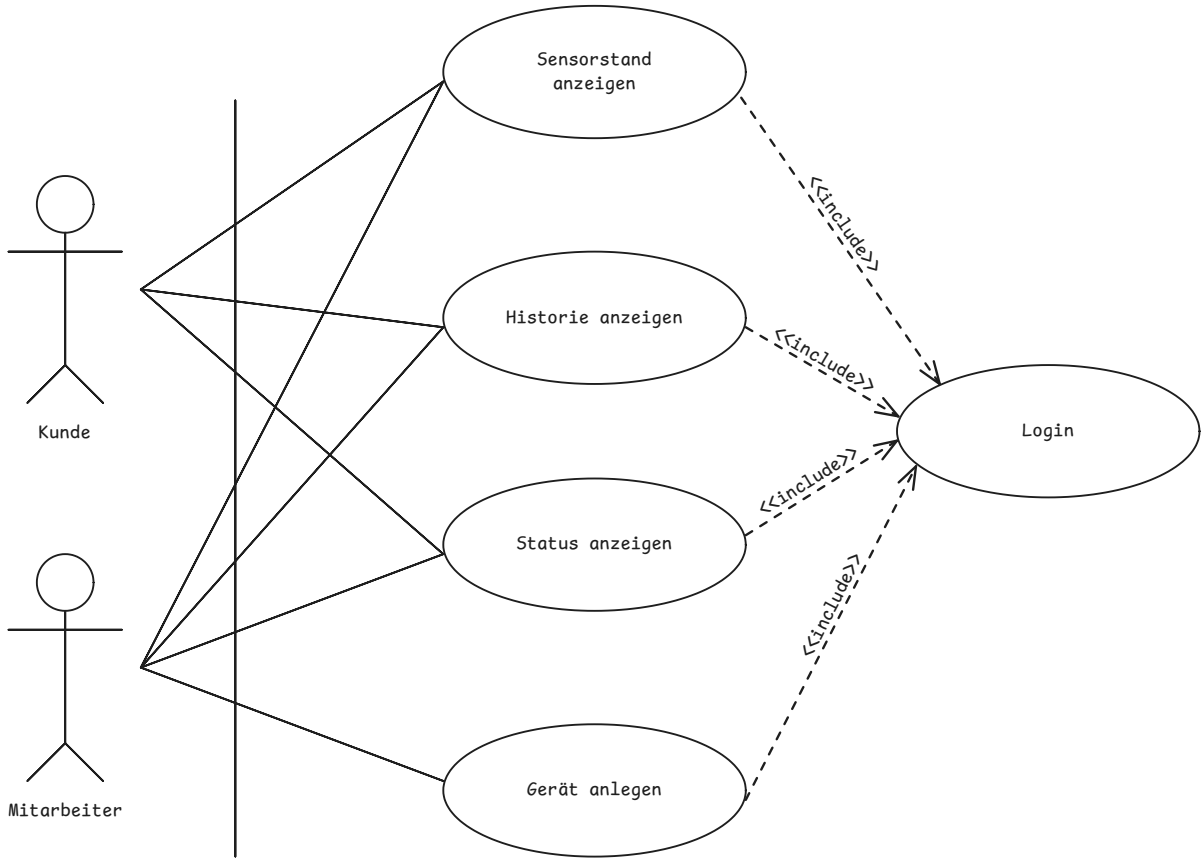


Abbildung 1: Use Case-Diagramm

A.4 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der Sensordaten
   1.1. Das System muss die Verwaltung von Sensoren und Gateways ermöglichen
   1.2. Sensordaten sollen empfangen, aufbereitet und gespeichert werden
   1.3. Es müssen Durchschnittswerte für Stunde, Tag, Monat erstellt werden
   1.4. Die aufbereiten Sensordaten sollen in einer Datenbank gespeichert werden
2. Darstellung der Daten
   2.1. Die App muss eine Liste aller Sensoren mit ihrem Status und dem Sensorstand erstellen
   2.2. Der Status eines Sensors muss visuell hervorgehoben werden: grün (unauffällig), gelb (auffällig), rot (kritisch)
   2.3. Es muss eine Übersicht erstellt werden mit dem Sensorstand und dem Status
   2.4. Die Übersicht muss einen direkten Zugang zur Datenhistorie jedes Sensors bieten
   2.5. Die Historie muss eine Möglichkeit bieten die Genauigkeit der Daten einzustellen
   2.6. Ein Vergleich der aktuellen Werte mit denen eines vorherigen Zeitraums (z. B. Vortag oder Vorwoche) muss möglich sein
3. Sonstige Anforderungen
   3.1. Die App muss auf iOS und Android verfügbar sein
   3.2. Der Code muss modular aufgebaut werden um weitere Sensortypen hinzufügen zu können
   3.3. Das Systen soll jederzeit erreichbar sein
   3.4. Bei Ausfällen muss ein fehlerhafte Durchschnittsberechnung ausgeschlossen werden
   3.5. Die App muss intuitiv bedienbar sein, sodass keine zusätzliche Benutzerschulung notwendig ist

A.5 Mockups

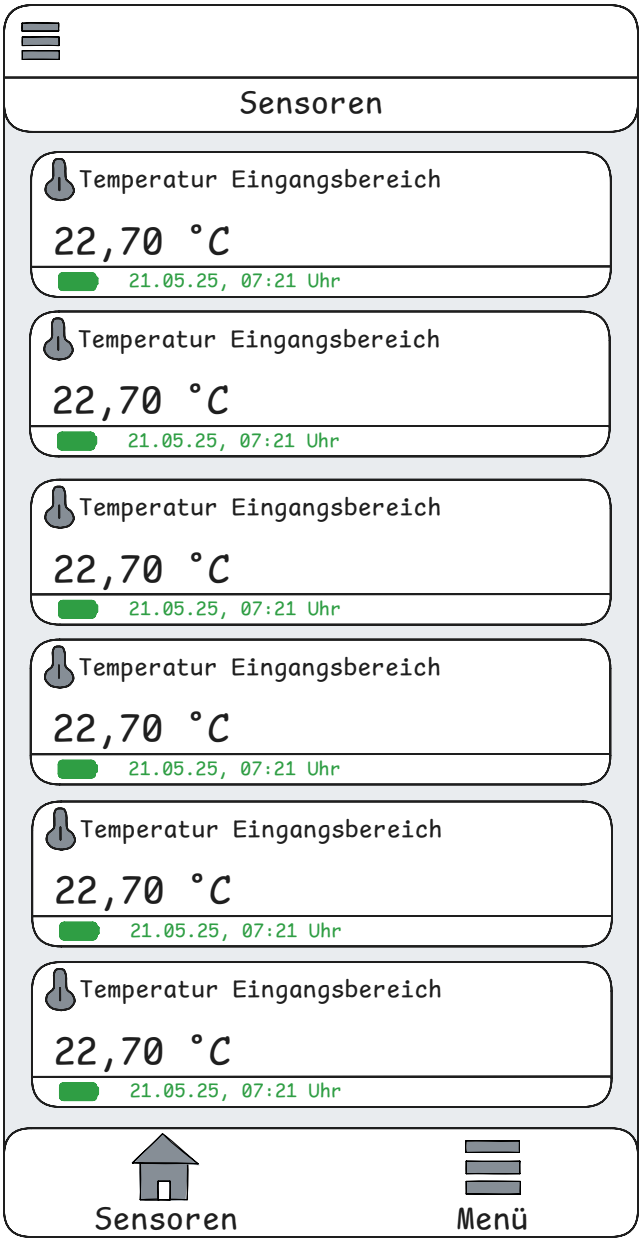


Abbildung 2: Übersicht aller Sensoren

Abbildung 3: Detailseite eines Sensors

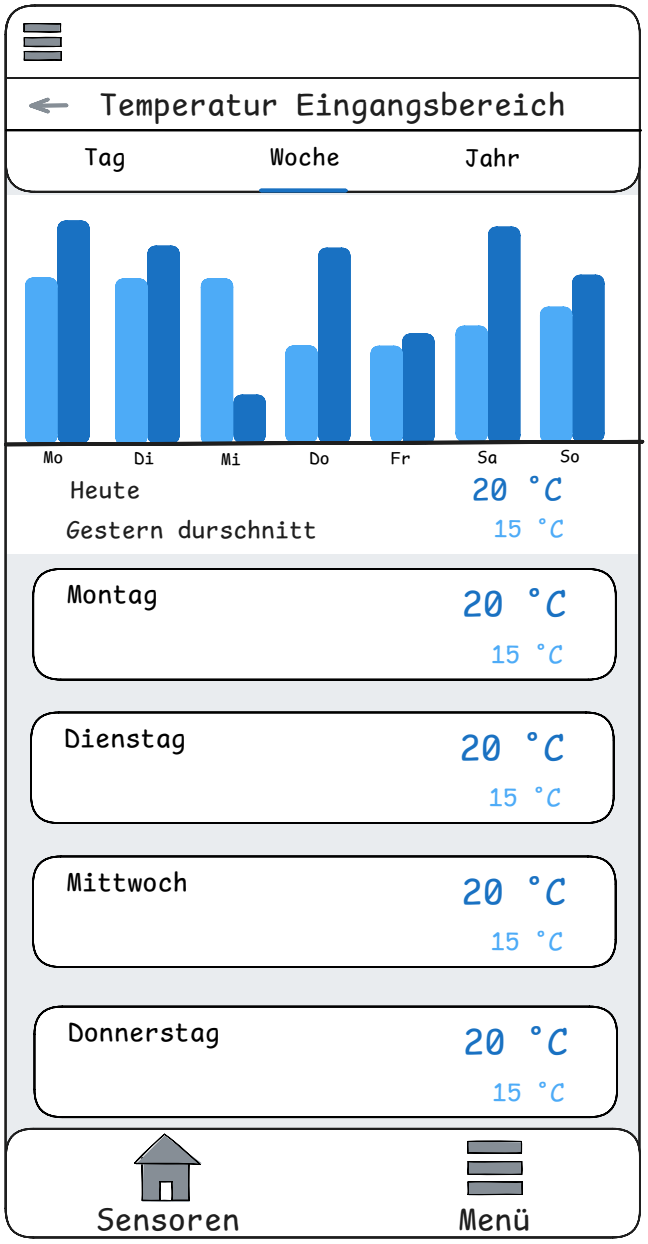


Abbildung 4: Historie eines Sensors

A.6 ER-Modell

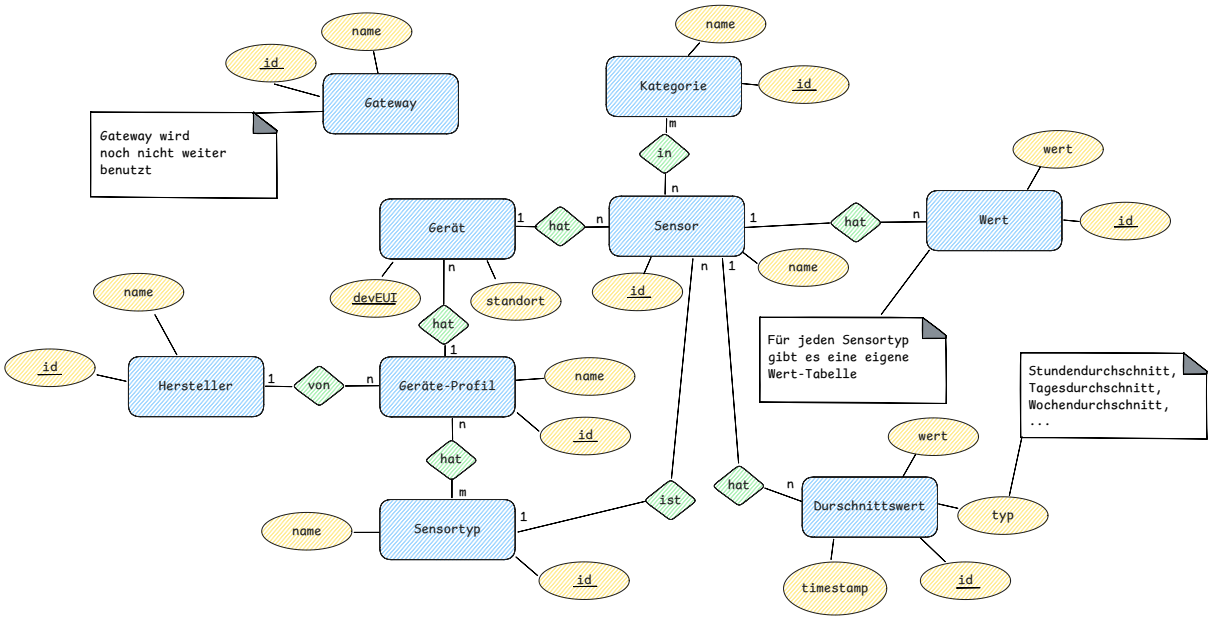


Abbildung 5: ER-Modell

A.7 Komponentendiagramm

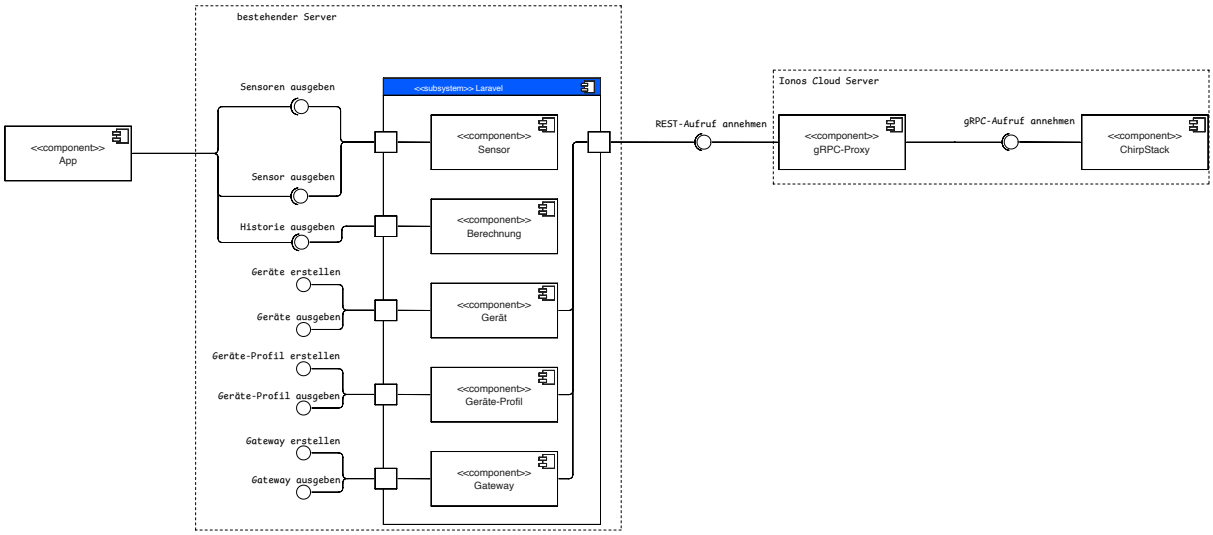


Abbildung 6: Komponentendiagramm

A.8 Pflichtenheft (Auszug)

Es folgt ein Auszug aus dem Pflichtenheft, der die Umsetzung der im Lastenheft definierten Anforderungen beschreibt.

1. Plattform
   1.1. Zur Entwicklung der Anwendung wird PhpStorm und Visual Studio Code verwendet
   1.2. Die App wird mit der Programmiersprache JavaScript programmiert
   1.3. Der gRPC-Proxy wird mit TypeScript programmiert
   1.4. Das Backend wird mit PHP 8.1 entwickelt
   1.5. Als Zielplattform für den Server wird Debian 12 eingesetzt. Der gRPC-Proxy wird dort mit `pm2` ausgeführt
   1.6. Um den gRPC-Proxy in ausführbares JavaScript zu kompilieren wird `make` verwendet
2. Datenbank
   2.1. Die Anbindung der Datenbank erfolgt mit Eloquent
   2.2. Die Daten werden in einer MySQL 8 Datenbank gespeichert
3. Oberfläche
   3.1. Die App muss mit NativeScript-Vue umgesetzt werden
4. Geschäftslogik
   4.1. Die Geschäftslogik wird in Laravel 9 umgesetzt
   4.2. Zur Kommunikation mit gRPC-Proxy wird Guzzle verwendet

A.9 Migration: LORA_sensors

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('LORA_sensors', function (Blueprint $table) {
            $table->id();
            $table->foreignId('lora_device_id')->references('id')->on('LORA_devices');
            $table->foreignId('lora_sensor_type_id')->references('id')->on('LORA_sensor_types');
            $table->string('name')->nullable();
            $table->string('location')->nullable();
            $table->float('min')->nullable();
            $table->float('max')->nullable();
            $table->integer('decimals')->nullable();
            $table->string('prefix')->nullable();
            $table->string('suffix')->nullable();
            $table->text('formula')->nullable();
            $table->json('config')->nullable();
            $table->text('general_info')->nullable();
            $table->boolean('published')->default(false);
            $table->boolean('active')->default(true);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('LORA_sensors');
    }
};
```

Listing 1: Migration: LORA_sensors

A.10 Model: LORASensors (Auszug)

```php
<?php

namespace App\Models\Lora;

use App\Models\Admin\Customer;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class LORASensor extends Model
{
    use HasFactory;

    protected $table = 'LORA_sensors';

    protected $fillable = [
        'name',
        'location',
        'min',
        'max',
        'decimals',
        'prefix',
        'suffix',
        'formula',
        'config',
        'general_info',
        'published',
        'active',

        'lora_device_id',
        'lora_sensor_type_id'
    ];

    public static $validator = [
        'name' => 'string',
        'location' => 'string',
        'min' => 'float',
        'max' => 'float',
        'decimals' => 'integer',
        'prefix' => 'string',
        'suffix' => 'string',
        'formula' => 'string',
        'config' => 'json',
        'general_info' => 'string',
        'published' => 'boolean',
        'active' => 'boolean',
    ];

    protected $casts = [
        'name' => 'string',
```

```
        'location' => 'string',
        'min' => 'float',
        'max' => 'float',
        'decimals' => 'integer',
        'prefix' => 'string',
        'suffix' => 'string',
        'formula' => 'string',
        'config' => 'json',
        'general_info' => 'string',
        'published' => 'boolean',
        'active' => 'boolean',
    ];


    public function lora_device() {
        return $this->belongsTo(LORADevice::class, 'lora_device_id', 'id');
    }

    public function lora_sensor_type() {
        return $this->belongsTo(LORASensorType::class, 'lora_sensor_type_id', 'id');
    }

    public function lora_temperatures() {
        return $this->hasMany(LORATemperature::class, 'lora_sensor_id', 'id');
    }

    public function categoryMapping() {
        return $this->hasMany(LORACategorySensor::class, 'sensor_id', 'id');
    }
}
```

Listing 2: Model: LORASensors (Auszug)

A.11 Screenshots der Anwendung

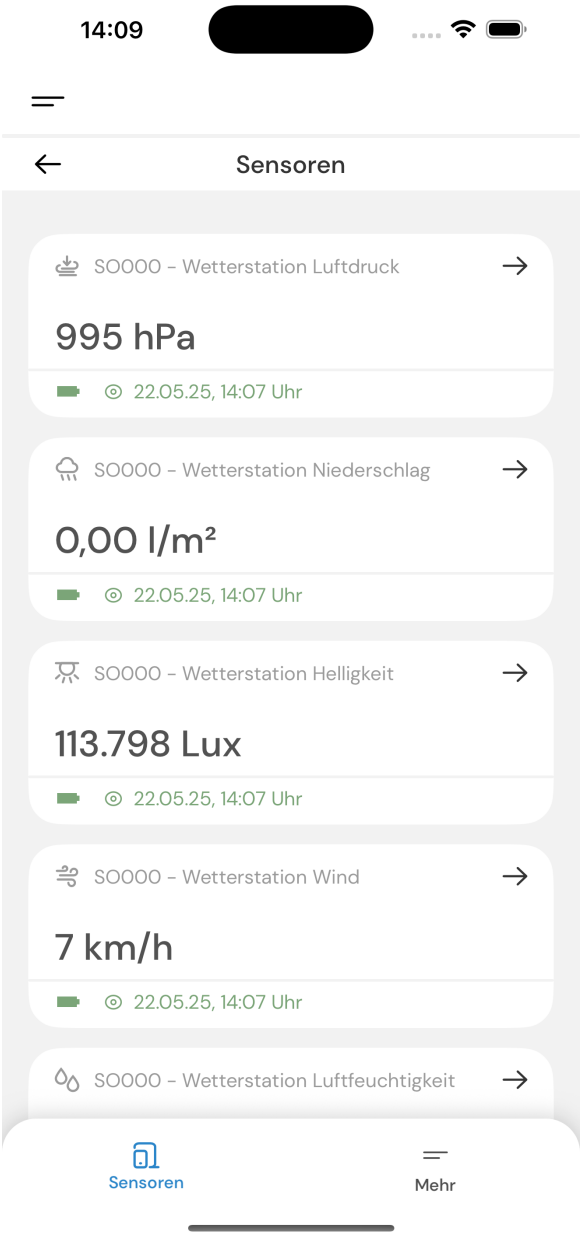


Abbildung 7: Übersicht aller Sensoren

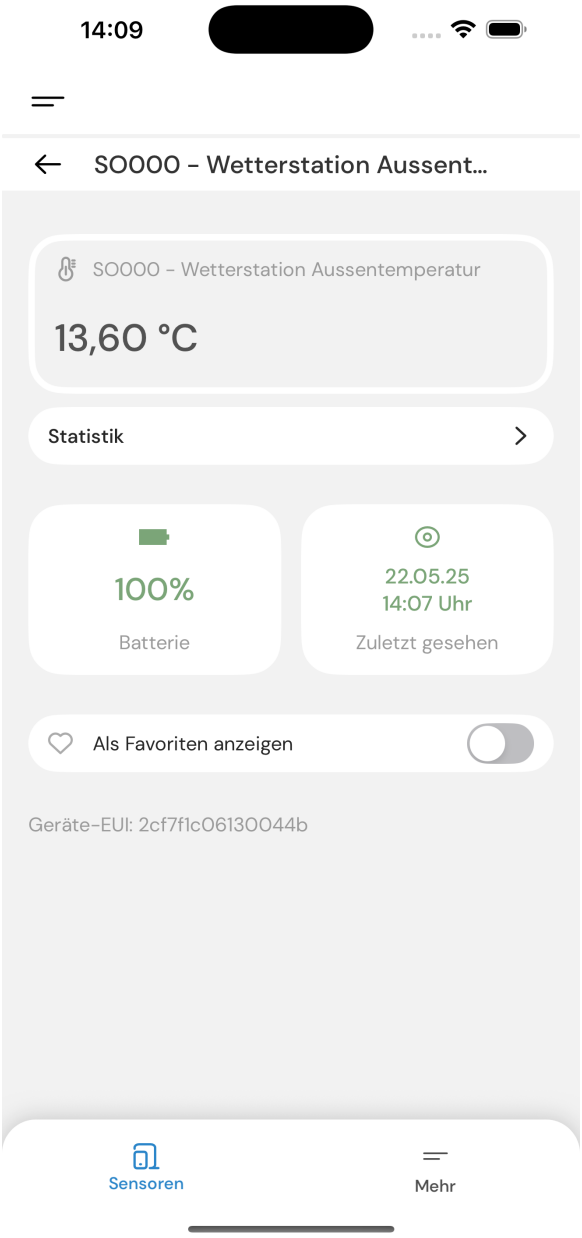


Abbildung 8: Detailseite eines Sensors

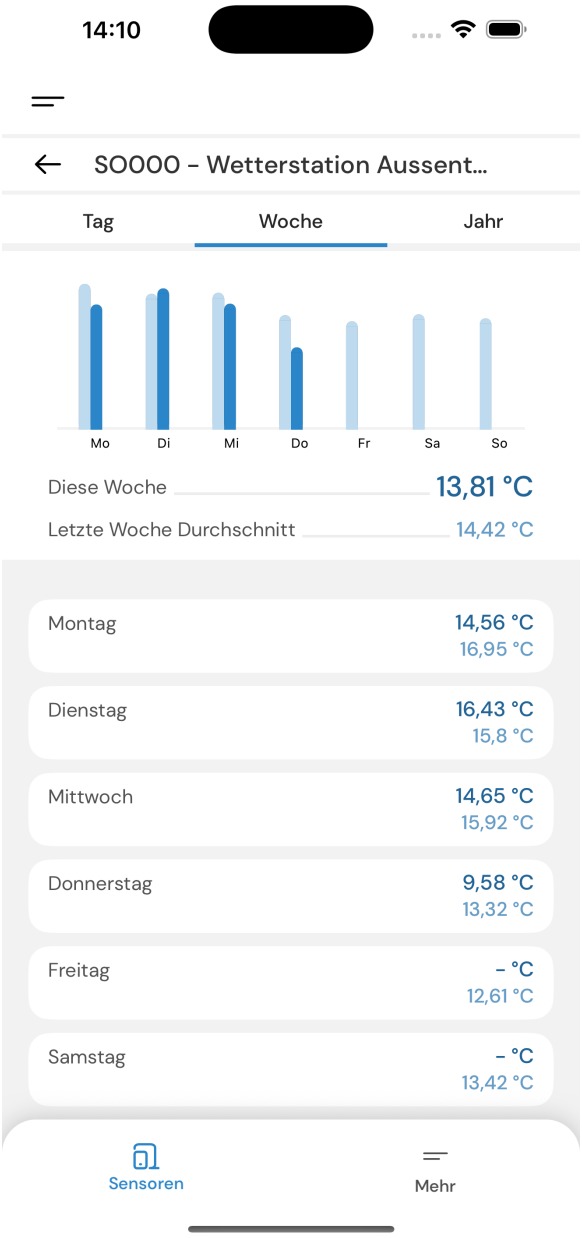


Abbildung 9: Historie eines Sensors

A.12 Sensorliste (Auszug)

```
<template>
    <Page actionBarHidden="true" class="page-interface lora-sensors"
    ↪  enableSwipeBackNavigation="false">
        <GridLayout iosOverflowSafeArea="true">

            <GridLayout rows="*, auto, auto, *" paddingLeft="20" paddingRight="20"
            ↪  paddingTop="30" paddingBottom="100" height="100%" v-if="dataItems.length === 0
            ↪  && !loading">
                <NoDataIcon marginBottom="40" verticalAlignment="bottom"
                ↪  horizontalAlignment="center" row="1" />
                <HTMLLabel class="marginal-text" horizontalAlignment="center"
                ↪  textAlignment="center" width="66%" textWrap="true" row="2" text="Es wurden
                ↪  keine passenden Einträge gefunden." />
            </GridLayout>

            <PullToRefresh @refresh="refresh" indicatorColor="transparent"
            ↪  indicatorFillColor="transparent">
                <CollectionView :items="dataItems" paddingLeft="20" paddingRight="20"
                ↪  paddingTop="30" paddingBottom="100"
                ↪  :itemTemplateSelector="templateSelector"
                ↪  :scrollBarIndicatorVisible="false" @loadMoreItems="loadMore">
                    <v-template name="default">
                        <GridLayout columns="20, 30, 30, *, 30, 20" rows="40, *, 3, 30"
                        ↪  class="list-template" @tap="goToDetails(item)">
                            <HTMLLabel :text="getIconForSensorType(item.lora_sensor_type_id)"
                            ↪  class="mdi-outlined-18px desc-icon" col="1" row="0"
                            ↪  verticalAlignment="top" />
                            <HTMLLabel :text="item.name" col="2" colSpan="2" row="0"
                            ↪  textWrap="true" verticalAlignment="top" class="desc-text" />
                            <HTMLLabel :text="item['data'][0]['data']" class="data-text"
                            ↪  row="1" col="1" colSpan="3" lineBreak="end" maxLines="1" />
                            <Separator col="0" colSpan="6" row="2" marginLeft="0"
                            ↪  marginRight="0" />
                            <CanvasView col="1" row="4" width="18" height="8"
                            ↪  @draw="onDraw($event, item.status)" horizontalAlignment="left"
                            ↪  marginLeft="2" :key="item.id" />
                            <HTMLLabel text="visibility" class="visibility mdi-outlined-12px"
                            ↪  col="2" row="4" :color="item.status.lastSeenCritical ?
                            ↪  '#C15B5B' : '#7BA578'" textAlignment="center" />
                            <HTMLLabel :text="item.lora_device.last_seen_at ?
                            ↪  parseDateTimeToPattern(item.lora_device.last_seen_at) + ' Uhr'
                            ↪  : 'keine Daten vorhanden.'" col="3" row="4"
                            ↪  verticalAlignment="center" verticalTextAlignment="center"
                            ↪  :color="item.status.lastSeenCritical ? '#C15B5B' : '#7BA578'"
                            ↪  height="100%" />
                            <HTMLLabel row="0" col="4" text="arrow_forward_ios"
                            ↪  class="proceed-icon mdi-outlined-18px" verticalAlignment="top"
                            ↪  horizontalAlignment="right" />
                        </GridLayout>
```

```
                    </v-template>

                    <v-template name="skeleton">
                        <LoraListSkeleton :single="true" />
                    </v-template>
                </CollectionView>
            </PullToRefresh>
        </GridLayout>
    </Page>
</template>

<script>
import { navigateToFunction } from "~/utils/helpers/navigationFunctions";
import LORASensorDetailView from
↪  "@/utils/_microapps/_lorawan/sensors/LORASensorDetailView.vue";
import MappingMixin from "../mixins/mapping";
import Separator from "@/utils/_ground/shared/Separator.vue";
import { parseDateTimeToPattern } from "@/utils/helpers/functions";
import LoraListSkeleton from "~/utils/_microapps/_lorawan/LoraListSkeleton.vue";
import paginationMixin from "../mixins/pagination";
import { drawBattery } from "@/utils/_microapps/_lorawan/utils/canvas";
import NoDataIcon from "~/static/NoDataIcon.vue";

export default {
    components: {
        LoraListSkeleton,
        NoDataIcon,
        LORASensorDetailView,
        MappingMixin,
        Separator,
    },
    props: ["divisionTitle"],
    name: "LORASensors",
    mixins: [paginationMixin],
    data() {
        return {
            rowsPerPage: 25,
            MappingMixin,
        };
    },
    computed: {
        url() {
            return "api/lora/sensors/search";
        },
        params() {
            return {
                page: this.page,
                rowsPerPage: this.rowsPerPage,
            };
        },
```

```
        dataItems() {
            if (this.loading) {
                return [{ type: "skeleton" }, { type: "skeleton" }];
            }
            return this.items;
        },
    },
    methods: {
        parseDateTimeToPattern,
        onDraw(event, status) {
            drawBattery(event, status);
        },
        templateSelector(item) {
            if (item.type) {
                return item.type;
            }
            return "default";
        },
        goToDetails(item) {
            let frame = "apps-frame";
            let options = {
                frame: frame,
                transition: "slideLeft",
                props: {
                    item: item,
                },
            };
            let targetSettings = {
                hideFooter: true,
                setPageTitle: item.name,
            };
            navigateToFunction(this, LORASensorDetailView, options, targetSettings);
        },
        getIconForSensorType(sensorType) {
            return MappingMixin.computed.iconsMapping()[sensorType]
        },
    },
    mounted() {
        this.paginate();
    },
};
</script>
```

Listing 3: Sensorliste (Auszug)

A.13 gRPC-Proxy (Auszug)

```
import { promisify } from 'node:util';
import { createError, createRouter, eventHandler, readValidatedBody } from 'h3';
```

```
import { DeviceServiceClient } from '@chirpstack/chirpstack-api/api/device_grpc_pb.js';
import Device from '@chirpstack/chirpstack-api/api/device_pb.js';
import { ChannelCredentials } from '@grpc/grpc-js';

import { ADDRESS, APPLICATION_ID, metadata } from './globals.js';
import { responseHandler } from './utils.js';
import { CreateDeviceSchema } from './schema.js';

const devicesRouter = createRouter();

async function createDevice(
    devEui: string,
    name: string,
    appKey: string,
    deviceProfileId: string
) {
    const client = new DeviceServiceClient(
        ADDRESS,
        ChannelCredentials.createSsl()
    );

    const request = new Device.CreateDeviceRequest();
    const device = new Device.Device();
    device.setDevEui(devEui);
    device.setName(name);
    device.setDeviceProfileId(deviceProfileId);
    device.setApplicationId(APPLICATION_ID);
    request.setDevice(device);
    await responseHandler(
        promisify(client.create.bind(client, request, metadata, {}))()
    );
    const requestK = new Device.CreateDeviceKeysRequest();
    const deviceKeys = new Device.DeviceKeys();
    deviceKeys.setDevEui(devEui);
    // LoraWAN 1.1 renamed AppKey to NwkKey
    deviceKeys.setNwkKey(appKey);
    requestK.setDeviceKeys(deviceKeys);
    return responseHandler(
        promisify(client.createKeys.bind(client, requestK, metadata, {}))()
    );
}

devicesRouter.post(
    '/create',
    eventHandler(async (ev) => {
        const body = await readValidatedBody(ev, CreateDeviceSchema.safeParse);
        if (!body.success) {
            return createError({
                statusCode: 422,
                statusMessage: 'Unprocessable Content',
```

```
                data: { details: body.error.message },
            });
        }
        const response = await createDevice(
            body.data.devEui,
            body.data.name,
            body.data.appKey,
            body.data.deviceProfileId
        );

        return response.toObject();
    })
);
```

Listing 4: gRPC-Proxy (Auszug)

A.14 Klasse: LoraDevicesController (Auszug)

```php
<?php

namespace App\Http\Controllers\Lora;

use App\Http\Controllers\Controller;
use App\Http\Controllers\Interfaces\ChirpstackController;
use App\Models\Lora\LORADevice;
use App\Models\Lora\LORADeviceProfile;
use App\Models\Lora\LORASensor;
use App\Resources\Helpers;
use GuzzleHttp\Exception\GuzzleException;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Log;

class LoraDevicesController extends Controller
{
    private $m = LORADevice::class;

    public function show(Request $request, int $id) {
        $userTypeId = Auth::user()->user_type_id;
        if ($userTypeId !== 1) {
            return response()->json(['message' => 'Not allowed'], 403);
        }

        $device = $this->m::find($id);
        if (!isset($device)) {
            return response()->json(['message' => 'Device not found'], 404);
        }

        return [
```

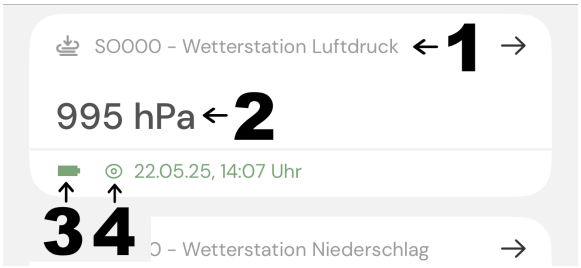
```
32              'id' => $device->id,
33              'published' => $device->published ? 1 : 0,
34              'deviceProfileId' => $device->device_profile_id,
35              'devEui' => $device->devEui,
36              'joinEui' => $device->join_eui,
37              'appKey' => $device->app_key,
38              'serialNumber' => $device->serial_number,
39              'battery' => $device->battery,
40              'lastSeenAt' => $device->last_seen_at,
41              'generalInfo' => $device->general_info,
42          ];
43      }
44
45      public function store(Request $request) {
46          $userTypeId = Auth::user()->user_type_id;
47          if ($userTypeId !== 1) {
48              return response()->json(['message' => 'Not allowed'], 403);
49          }
50
51          $values = $request->all();
52          $inp = Helpers::updateArrayKeys($values, [
53              'published' => 'published',
54              'deviceProfileId' => 'device_profile_id',
55              'devEui' => 'devEui',
56              'joinEui' => 'join_eui',
57              'appKey' => 'app_key',
58              'serialNumber' => 'serial_number',
59              'generalInfo' => 'general_info',
60          ]);
61          $validation = Helpers::requireArrayKeys($inp, ['published', 'device_profile_id',
            ↪ 'devEui', 'app_key']);
62          if ($validation->fails()) {
63              return response()->json(['status' => -1, 'message' => $validation->errors()],
                ↪ 422);
64          }
65
66          $inp['devEui'] = strtolower($inp['devEui']);
67          $chirpstack = new ChirpstackController();
68
69          $deviceProfile = LORADeviceProfile::where('active',
            ↪ 1)->find($inp['device_profile_id']);
70          if (!isset($deviceProfile)) {
71              return response()->json(['message' => 'Invalid Device Profile supplied'], 422);
72          }
73
74          try {
75              if ($chirpstack->addDevice($inp['devEui'], $inp['devEui'], $inp['app_key'],
                ↪ $deviceProfile->uuid) === null) {
76                  Log::warning('addDevice failed '.$inp['devEui']);
77                  return response()->json(['status' => -1, 'message' => 'Device creation
                    ↪ failed'], 500);
```

```
78              }
79          } catch (GuzzleException $e) {
80              return response()->json(['status' => -1, 'message' => $e->getMessage()],
                ↪  $e->getCode());
81          }
82
83          $device = $this->m::create($inp);
84          foreach ($deviceProfile->sensorTypes as $sensorType) {
85              LORASensor::create([
86                  'lora_device_id' => $device->id,
87                  'lora_sensor_type_id' => $sensorType->id,
88              ]);
89          }
90
91          return ['status' => 0, 'id' => $device->id];
92      }
93  }
```

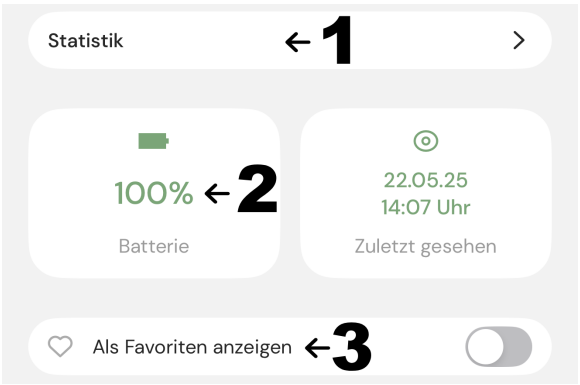
Listing 5: Klasse: LoraDevicesController (Auszug)

A.15 Benutzerdokumentation

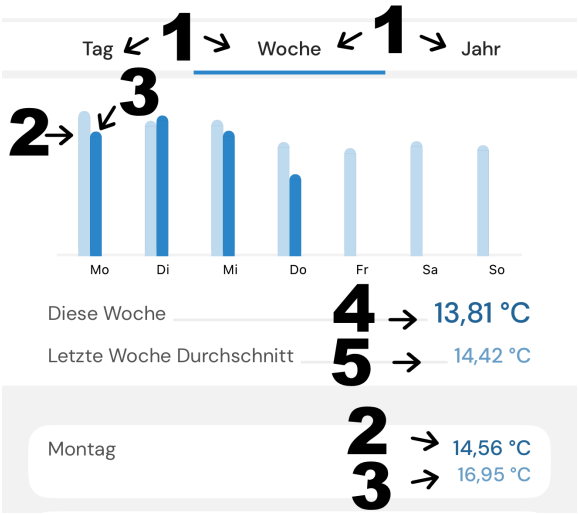
Auszug aus der Benutzerdokumentation:



(1) zeigt den Namen des Sensors.Das davorstehende Icon zeigt die Art des Sensortyps. In diesem Fall ein Luftdrucksensor.
(2) zeigt den aktuellsten Zählerstand
(3) zeigt den aktuellsten Batteriestand
(4) zeigt wann die letzten Daten geschickt wurden



(1) Sie können hier klicken um die Historie anzuzeigen
(2) zeigt den aktuellsten Batteriestand in Prozent
(3) Sie können den Knopf drücken um den Sensor als Favoriten zu markieren



(1) Sie können die Genauigkeit der Daten auswählen
(2) zeigt den Wert des Vorzeitraums (Montag der letzten Woche)
(3) zeigt einen Wert des aktuellen Zeitraums (Montag der aktuellen Woche)
(4) zeigt den Wert des kompletten Zeitraums (komplette aktuelle Woche)
(5) zeigt den Wert des kompletten Vorzeitraums (komplette letzte Woche)