

Prüfungsteil A

Prüfling (private Anschrift):
Aland Mariwan
Lortzingstraße 20
14772 Brandenburg an der Havel

Ausbildungsbetrieb:
Gesellschaft für Systemtechnik, Softwareentwicklung
und Datenverarbeitungsservice mbH
Zehlendorfer Straße 5
14513 Teltow

Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):
Fachinformatiker für Anwendungsentwicklung

Projektbezeichnung:
Entwicklung einer Überwachung für Anwendungen

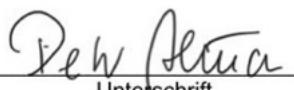
Projektbeginn: 18.02.2023 Projektfertigstellung: 21.04.2023 Zeitaufwand in Std.: 80

Bestätigung der Ausbildungsfirma:

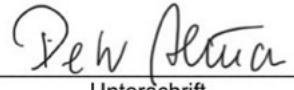
Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation im Zeitraum

vom: 18.02.2023 bis: 21.04.2023 selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

Peter	Altmann	+49 3328 3534500	
Vorname	Name	Telefon	Unterschrift

Ausbildungsverantwortliche(r) in der Firma:

Peter	Altmann	+49 3328 3534500	
Vorname	Name	Telefon	Unterschrift

Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Ort und Datum: Teltow, 13.04.2023 Unterschrift des Prüflings: 

Sommerprüfung 2023

Ausbildungsberuf

Fachinformatiker/Fachinformatikerin (VO 2020) Fachrichtung:
Anwendungsentwicklung

Prüfungsbezirk

Potsdam FI 1196-1 (AP T2V1)

Aland Mariwan

Identnummer: 1494891

E-Mail: kontakt@aland-mariwan.de, Telefon: +49 172 7134842

Ausbildungsbetrieb: GSSD mbH

Projektbetreuer: Peter Altmann

E-Mail: peteraltmann@gssd.de, Telefon: +49 3328 3534500

Thema der Projektarbeit

Entwicklung einer Überwachung für Anwendungen.

1 Thema der Projektarbeit

Entwicklung einer Überwachung für Anwendungen.

2 Geplanter Bearbeitungszeitraum

Beginn: 18.02.2023

Ende: 21.04.2023

3 Ausgangssituation

Ein Kunde von der GSSD verwendet auf seinen Servern verschiedene Anwendungen, die als Dienste oder im Hintergrund laufen. Es muss sichergestellt werden, dass diese Anwendungen ständig funktionieren. Gegenwärtig gibt es ein Überwachungssystem, das nur unvollständige Zustände anzeigt.

4 Projektziel

Ziel ist die Erstellung einer Überwachung für ein System von Anwendungen. Auf einem Dashboard sollen die Status der Anwendungen dargestellt werden.

Die Umsetzung wird als Überwachungssystem realisiert, die es dem Administrator ermöglicht, alle verfügbaren Anwendungen auch in ihrer Zusammenarbeit zu überwachen. Die einzelnen Anwendungen müssen regelmäßig Informationen über ihren Zustand und ihre aktuelle Tätigkeit liefern. Diese Informationen schreiben die Anwendungen in Datenbanktabellen. Ein Backend liest die Daten aus der Datenbank, bewertet sie und zeigt die Ergebnisse im Frontend. Über ein WebSocket werden die benötigten Informationen zwischen Frontend und Backend ausgetauscht. Auf dem Frontend kann der aktuelle Status von Anwendungen von jedem Nutzer überprüft werden, solange dieser eine Verbindung zum Frontend aufbauen kann. Über eine Filterfunktion kann die Ausgabe vom Backend eingeschränkt werden.

Das Überwachungssystem wird mit den Programmiersprachen HTML, VueJs, CSS, Bash, SQL, NodeJs realisiert. Die Anwendung wird vom Administrator überwacht und bei bestimmten Fehlermeldungen (Bsp.: Datenbank-Überlastung oder Anwendungsausfall) wird er auf geeignete Weise aufmerksam gemacht. Somit kann der Administrator reagieren.

5 Zeitplanung

Anlage 1

6 Anlagen

siehe Anlage 1

7 Präsentationsmittel

- PowerPoint
- Beamer
- Notebook

8 Hinweis!

Ich bestätige, dass der Projektantrag dem Ausbildungsbetrieb vorgelegt und vom Ausbildenden genehmigt wurde. Der Projektantrag enthält keine Betriebsgeheimnisse. Soweit diese für die Antragstellung notwendig sind, wurden nach Rücksprache mit dem Ausbildenden die entsprechenden Stellen unkenntlich gemacht.

Mit dem Absenden des Projektantrages bestätige ich weiterhin, dass der Antrag eigenständig von mir angefertigt wurde. Ferner sichere ich zu, dass im Projektantrag personenbezogene Daten (d. h. Daten über die eine Person identifizierbar oder bestimmbar ist) nur verwendet werden, wenn die betroffene Person hierin eingewilligt hat.

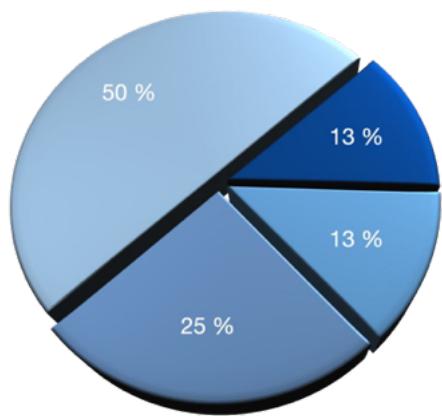
Bei meiner ersten Anmeldung im Online-Portal wurde ich darauf hingewiesen, dass meine Arbeit bei Täuschungshandlungen bzw. Ordnungsverstößen mit „null“ Punkten bewertet werden kann. Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Arbeit im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Es ist mir bewusst, dass Kontrollen durchgeführt werden.

Anlage 1

Projektstrukturplan erstellen

Grafische und tabellarische Darstellung

- Planung und Konzeption
- Programmierung
- Erstellung der Dokumentation
- Abnahme



Die gesamte geschätzte Zeit beläuft sich auf 80h (h = Stunden):

Phase	Zeit in Stunden
Planung und Konzeption	20h

Phase	Zeit in Stunden
Programmierung	40h
Erstellung der Dokumentation	10h
Abnahme	10h
Total	80h

Projektphasen mit Zeitplanung in Stunden

Planung und Konzeption	ca. 20h
Selbstständige Fortbildung	ca. 10h
Absprache mit der internen Entwicklungsabteilung für Schnittstellen und Frontend	ca. 5h
Projektstruktur bilden (Struktogramme und Programmablaufpläne)	ca. 5h
Programmierung	ca. 40h
Entwicklung des Backend	ca. 10h
Entwicklung der WebSocket	ca. 10h
Entwicklung der Frontend	ca. 15h
Gesamte Testphase	ca. 5h
Erstellung der Dokumentation	ca. 10h
Anwenderdokumentation	ca. 2h
Technische Dokumentation	ca. 8h
Abnahme	ca. 10h
Qualitätssicherung	ca. 5h
interne Anwender	ca. 5h

Zeitmitschrift der Projektarbeit

Prüfungsteilnehmer/-in

Name: Aland

Vorname: Mariwan

Login: 1831494891

Datum	Tätigkeit	Zeit in Stunden
06.03.2023	Selbstständige Fortbildung	4h
06.03.2023	Vorbereitung Projektdokumentation	4h
08.03.2023	Zeitmitschrift	1h
08.03.2023	Absprache mit der internen Entwicklungsabteilung für Schnittstellen	6h
08.03.2023	Besprechung mit Auftraggeber über Frontend	1h
13.03.2023	Lastenheft, Konzept Projektstruktur	4h
13.03.2023	Recherche NodeJs Bibliotheken	1h
13.03.2023	Implementation Backend	3h
15.03.2023	Implementation Backend	4,5h
15.03.2023	Besprechung mit Auftraggeber	1h
15.03.2023	Implementation Frontend	2,5h
17.03.2023	Implementation Backend	3h
17.03.2023	Postman Tests	1,5h
17.03.2023	Build-Skripte	0,5h
17.03.2023	Implementation Frontend	3h
27.03.2023	Implementation Frontend	4h
27.03.2023	Implementation Backend	2h
27.03.2023	Postman Tests	2h
29.03.2023	Implementation Frontend	4h
29.03.2023	Implementation Backend	2h
29.03.2023	Postman Tests	2h
31.03.2023	Implementation Backend	1h
31.03.2023	Implementation Frontend	6h
31.10.2023	Build-Skripte, Test	0,5h
31.03.2023	Recherche Cert-Gen und Rate-Limiting Backend	0,5h
03.04.2023	Projektdokumentation	4h
03.04.2023	Backend Tests	2h
03.04.2023	Frontend Tests	2h
05.04.2023	White-Box Text	3h
05.04.2023	Entwurf Diagramme und Darstellungen	3h
05.04.2023	Projektdokumentation	2h



ABSCHLUSSPRÜFUNG SOMMER 2023

FACHINFORMATIKERIN FÜR ANWENDUNGSENTWICKLUNG

DOKUMENTATION ZUR PROJEKTARBEIT

Entwicklung einer Überwachung für Anwendungen

Abgabetermin: Potsdam, den 26.04.2023

Prüfungsbewerber:

Aland Mariwan
Lortzingstraße 20
14772 Brandenburg an der Havel

GSSD

Gesellschaft für Systemtechnik,
Softwareentwicklung und
Datenverarbeitungsservice mbH

Ausbildungsbetrieb:

GSSD mbH
Zehlendorfer Straße 5
14513 Teltow

Inhaltsverzeichnis

Abkürzungsverzeichnis	13
1. Einleitung	1
1.1. Projektbeschreibung	1
1.2. Projektziel	1
1.3. Projektbegründung	1
1.4. Projektschnittstellen	2
1.5. Projektgrenzen	2
1.6. Use-Case	2
1.6.1. Prozessabläufe	2
1.6.2. Anwendungsfälle	3
2. Projektplanung	3
2.1. Projektphasen	3
2.2. Ressourcenplanung	4
2.3. Entwicklungsprozess	4
3. Analysephase	4
3.1. Ist-Analyse	4
3.2. Workflow	5
3.3. Soll-Konzept	5
3.4. Risikoanalyse	6
3.5. Wirtschaftlichkeitsanalyse	6
3.5.1. „Make or Buy“-Entscheidung	6
3.5.2. Projektkosten	7
3.5.3. Amortisationsdauer	7
3.6. Lastenheft	7
4. Entwurfsphase	8
4.1. Zielplattform	8
4.2. Architekturdesign	8
4.3. Entwurf der Benutzeroberfläche	8
4.4. Datenmodell	9
4.5. Maßnahmen zur Qualitätssicherung	9
4.6. Deployment	9
4.7. Pflichtenheft	10
5. Implementierungsphase	10
5.1. Entwicklung des Dashboards	10
5.2. Datenbank	11
5.3. REST-API	11
5.4. Testen der Anwendung	12
5.5. Sicherheitstests	12
6. Abnahmephase	12
6.1. Code-Review	12
6.2. Abnahme	13

7. Dokumentation	13
7.1. Benutzerhandbuch Endanwender-Dokumentation	13
7.2. Entwicklerdokumentation	14
8. Fazit	14
8.1. Soll-Ist-Vergleich	14
8.2. Lessons Learned	14
8.3. Ausblick	15
Literaturverzeichnis	16
A. Anhang	i
A.1. Lastenheft	i
A.2. Pflichtenheft	ii
A.3. Verwendete Ressourcen	iii
A.4. Anforderungen	iv
A.5. Middleware	vi
A.6. Der Grund für die Verwendung von Session	vi
A.7. Der Grund für die Verwendung von Helmet.js	vii
A.8. Fehlercodes und Statuscode-Meldungen	viii
A.9. Use-case Diagramm	ix
A.10. Komponenten Diagramm	x
B. Relationales Datenbankmodell	xi
B.1. Screenshot des Sprint-Boards in GitHub	xii
B.2. Verwendung von Gantt-Diagramm in GitHub für die Projektplanung	xiii
B.3. Screenshot des File-Trees Backend	xiv
B.4. Anmeldung eines neuen Sensors	xv
B.5. Sequenzdiagramm: Sensor schickt Daten	xix
B.6. Dashboard-Seite	xxvii

Tabellenverzeichnis

1. Übersicht der angefallenen Kosten	7
2. Anforderungen an das Überwachungssystem	iv
3. Detaillierte Zeitmitschrift	v

Abbildungsverzeichnis

1. ExpressJS Middleware	vi
2. Fehlercodes und Statuscode-Meldungen	viii
3. Use-case Diagramm	ix
4. Komponenten Diagramm	x
5. Relationales Datenbankmodell	xi
6. Screenshot des Sprint-Boards in GitHub	xii
7. Verwendung von Gantt-Diagramm in GitHub für die Projektplanung	xiii
8. Screenshot des File-Trees Backend	xiv
9. Anmeldung eines neuen Sensors Sequenzdiagramm	xviii

10. Sensor schickt Daten (Sequenzdiagramm)	xix
11. Dashboard-Seite (Frontend)	xxvii

Quellcodesverzeichnis

1. Anmeldung eines neuen Sensors (Backend) JSON-Modell	xv
2. Anmeldung eines neuen Sensors (Backend)	xvi
3. Anmeldung eines neuen Sensors (Backend) Unit-Tests	xvii
4. Sensor schickt Daten (Backend)	xx
5. Sensor schickt Daten (Backend) Unit-Tests	xx
6. Neue Benutzer hinzufügen (Backend Codestück Teil 1)	xxi
7. Neue Benutzer hinzufügen (Backend Codestück Teil 2)	xxii
8. Neue Benutzer hinzufügen (Backend Codestück Teil 1)	xxiii
9. Neue Benutzer hinzufügen (Backend Codestück Teil 2)	xxiv
10. Passwort zurücksetzen (Backend)	xxv
11. Passwort zurücksetzen (Backend)	xxvi
12. VueJs router (Frontend Codestück Teil 1)	xxviii
13. VueJs router (Frontend Codestück Teil 2)	xxix
14. VueJs router (Frontend Codestück Teil 3)	xxx
15. VueJs AllSensors (Frontend Codestück Teil 1)	xxxi
16. VueJs AllSensors (Frontend Codestück Teil 2)	xxxii

Abkürzungsverzeichnis

GSSD Gesellschaft für Systemtechnik, Softwareentwicklung und Datenverarbeitungsservice

GMS GSSD Monitoring System

SQL Structured Query Language

mariaDB Maria Database

HTML Hypertext Markup Language

CSS Cascading Style Sheets

BASH Bourne-Again SHell

API Application Programming Interface

PRs Pull Requests

REST-API Representational State Transfer - Application Programming Interface

UML Unified Modeling Language

CI Continuous Integration

CD Continuous Deployment

TDD Test-Driven Development

BDD Behavior-Driven Development

IDE Integrated Development Environment

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

JSON JavaScript Object Notation

VSCode Visual Studio Code

GIT Global Information Tracker

NPM Node Package Manager

ERM Entity Relationship Modell

MiKTeX Mathematical Institute (Mathematisches Institut) (MI) + TeX

CDI Contexts and Dependency Injection

z.B. zum Beispiel

KI künstlicher Intelligenz

1. Einleitung

In dieser Projektdokumentation präsentiert der Autor den Ablauf seines Abschlussprojekts, welches im Rahmen seiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung durchgeführt wurde. Das Projekt fand bei der Gesellschaft für Systemtechnik, Softwareentwicklung und Datenverarbeitungsservice mbH (GSSD) in Teltow statt. Aktuell beschäftigt die GSSD sieben Mitarbeiter und betreut als IT-Dienstleister zahlreiche kleine und mittlere Unternehmen. Darüber hinaus ist die GSSD Partner verschiedener Anbieter von Warenwirtschaftssoftware und offeriert in diesem Bereich Anpassungs- und kundenspezifische Dienstleistungen. Ein weiterer Geschäftszweig der GSSD besteht im Verkauf von Hardware für Unternehmen und Privatkunden sowie in der dazugehörigen Installation und Wartung der Systeme. Das Hauptziel dieses Projekts bestand darin, die im Rahmen der Ausbildung erworbenen Kenntnisse und Fähigkeiten anzuwenden und eine webbasierte Anwendung zu entwickeln, die den Anforderungen der GSSD gerecht wird. Die Projektdokumentation beschreibt den gesamten Ablauf, von der Planung und Analyse über die Implementierung bis hin zur Bewertung und Dokumentation der Ergebnisse.

1.1. Projektbeschreibung

Die GSSD stellt für einen Kunden ein innovatives Überwachungssystem bereit, das auf dessen Servern laufende Anwendungen, welche als Dienste oder im Hintergrund agieren, effizient überwacht. Das bisherige System lieferte nur ungenügende Informationen, weshalb eine modernere Lösung entwickelt wird, die ein umfassendes Bild der Anwendungslandschaft ermöglicht. Das Herzstück des Projekts ist die Erstellung eines intuitiven Dashboards, das den Anwendungsstatus klar und ansprechend visualisiert. Das Überwachungssystem besteht aus einer leistungsstarken Backend- und einer benutzerfreundlichen Frontend-Anwendung, die es dem Administrator ermöglichen, alle laufenden Anwendungen und deren Interaktionen zu überwachen. Die Anwendungen selbst generieren kontinuierlich Status- und Aktivitätsdaten, die in Datenbanktabellen gespeichert werden. Das Backend verarbeitet diese Daten und bereitet sie für die Darstellung im Frontend auf. Mithilfe von Websockets wird eine nahtlose Kommunikation zwischen Frontend und Backend gewährleistet. Dadurch können Benutzer den Status der Anwendungen jederzeit auf dem Frontend einsehen, solange eine Verbindung besteht. Zudem erlaubt eine intelligente Filterfunktion im Backend, die angezeigten Informationen gezielt zu verfeinern.

1.2. Projektziel

Das Ziel des Projekts besteht darin, dem Kunden ein ansprechendes Dashboard mit Echtzeit-Visualisierung des Anwendungsstatus zur Verfügung zu stellen, um den Administratoren eine umfassende Überwachung der Anwendungen und ihrer Interaktionen zu ermöglichen. Die Anwendungen senden kontinuierlich Daten, die vom Backend verarbeitet und in Datenbanktabellen gespeichert werden. Die Kommunikation zwischen Frontend und Backend erfolgt über Websockets. Die Administratoren haben die Möglichkeit, den Anwendungsstatus abzurufen und neue Sensoren hinzuzufügen. Im Falle kritischer Fehler sorgt ein automatisches Benachrichtigungssystem dafür, dass die Administratoren sofort informiert werden und angemessene Maßnahmen ergreifen können.

1.3. Projektbegründung

Eine effektive Überwachung von Anwendungen und Systemen ist im Bereich der Informationstechnologie von entscheidender Bedeutung, da sie eine frühzeitige Fehlererkennung

ermöglicht und negative Auswirkungen auf Betrieb und die Verfügbarkeit minimiert. Ein zuverlässiges Überwachungssystem ermöglicht dem Administrator, die Leistung und den Zustand von Komponenten kontinuierlich zu überwachen, um schnell auf auftretende Probleme zu reagieren. Dadurch werden die Qualität und Verfügbarkeit der Anwendungen verbessert und die Kundenzufriedenheit erhöht. Die Implementierung eines solchen Systems stellt eine reibungslos funktionierende IT-Infrastruktur und eine hohe Kundenzufriedenheit sicher. Mit einem modernen und intuitiven Überwachungssystem können Administratoren potenzielle Probleme frühzeitig erkennen und beheben, bevor sie zu schwerwiegenden Störungen oder Ausfällen führen. Dies trägt dazu bei, den kontinuierlichen Betrieb der Anwendungen und Dienste zu gewährleisten und den Kunden einen zuverlässigen Service zu bieten.

1.4. Projektschnittstellen

Das Projekt umfasst folgende Schnittstellen:

- Geschäftsführung: Informationen über Projektfortschritte und Entscheidungen, die das Projekt beeinflussen, werden an die Geschäftsführung weitergegeben.
- IT-Abteilung: Die IT-Abteilung ist für die technische Umsetzung zuständig und arbeitet eng mit dem Projektteam zusammen.
- Externe Dienstleister: Schnittstellen müssen klar definiert werden, um eine reibungslose Zusammenarbeit mit externen Systemen und Dienstleistern zu gewährleisten.
- Benutzer: Die Einbindung und Information der Benutzer ist wichtig, um deren Anforderungen gerecht zu werden und eine hohe Akzeptanz des Systems zu erreichen.
- Datenbanken und Anwendungen: Die Kommunikation mit anderen Datenbanken und Anwendungen sind erforderlich, um Informationen effizient verarbeiten und austauschen zu können.
- Netzwerk- und Sicherheitsinfrastruktur: Schnittstellen zu Netzwerk- und Sicherheitsinfrastruktur müssen definiert werden, um mögliche Anpassungen oder Erweiterungen reibungslos durchführen zu können.

1.5. Projektgrenzen

- Die Konzeption und Realisierung von Sensoren fällt nicht in den Zuständigkeitsbereich dieses Projekts. Dennoch werden Spezifikationen und Anforderungen definiert, die den Sensoren ermöglichen, sich korrekt an das Backend-System anzubinden.
- Schulungen oder Weiterbildungen für Nutzer oder Mitarbeiter sind nicht vorgesehen.
- Das Überwachungssystem ist auf ein bestimmtes Team oder eine Abteilung beschränkt, nicht für das gesamte Unternehmen konzipiert.

1.6. Use-Case

1.6.1. Prozessabläufe

- Ablauf 1 beim Überwacher:
 - Überwacher öffnet Dashboard.
 - Dashboard zeigt Anwendungsübersicht und aktuellen Zustand.
 - Überwacher filtert Anwendungsliste nach Kriterien.
 - Bewertung jeder Anwendung wird angezeigt.
- Ablauf 2 beim Administrator:
 - Bei kritischem Zustand oder Problem wird Administrator benachrichtigt.

- Administrator reagiert auf Benachrichtigung (Problem lösen, System normalisieren).
- Administrator passt Überwachungssystem an, beim Anpassen ist eine Anmeldung erforderlich (Bewertungsfaktoren ändern, Anwendungen hinzufügen oder entfernen).
- Administrator öffnet Benachrichtigungseinstellungen im Überwachungssystem.
- Administrator konfiguriert Benachrichtigungsschwellenwerte.
- Überwachungssystem speichert die Konfiguration und sendet automatisch Benachrichtigungen, wenn Schwellenwerte überschritten werden.
- Administrator erhält Benachrichtigungen und ergreift entsprechende Maßnahmen.
- Ablauf beim Entwickler
 - Entwickler öffnet Sensorintegrationsseite im Überwachungssystem.
 - Entwickler gibt erforderliche Informationen für den neuen Sensor ein.
 - Überwachungssystem validiert eingegebene Informationen.
 - Bei erfolgreicher Validierung wird neuer Sensor dem System hinzugefügt und in der Sensorliste angezeigt.
 - Bei fehlerhafter Validierung wird Entwickler benachrichtigt und kann die Eingaben korrigieren.

1.6.2. Anwendungsfälle

Resultierend aus den Prozessabläufen lassen sich die folgende Anwendungsfälle und Abläufe ableiten:

- Administratoren/Überwacher als Akteur (Administratoren/Überwacher überwachen Systemzustand und reagieren auf potenzielle Probleme):
 - Anmelden im Überwachungssystem
 - Öffnen von Dashboard
 - Filtern Anwendungsliste nach Kriterien
- Administratoren als Akteur :
 - Konfigurieren des Überwachungssystems (Bewertungsfaktoren ändern, Anwendungen hinzufügen oder entfernen)
 - Öffnen der Benachrichtigungseinstellungen im Überwachungssystem
 - Konfigurieren Benachrichtigungsschwellenwerte
- Entwickler als Akteur (integriert neue Sensoren ins System):
 - Öffnen von Sensorintegrationsseite im Überwachungssystem
 - Eingeben der erforderlichen Informationen für den neuen Sensor

Das Use-Case-Diagramm ist im Anhang A.9 auf Seite ix zu finden.

2. Projektplanung

2.1. Projektphasen

Im Rahmen des Projektantrags wurden die Projektphasen definiert und in Unterpunkte untergliedert. Für die Durchführung des Projekts stehen insgesamt 80 Stunden zur Verfügung, wobei 20 Stunden für die Planung und Konzeption, 40 Stunden für die Programmierung, 10 Stunden für die Erstellung der Dokumentation und weitere 10 Stunden für die Abnahme des Projekts vorgesehen sind.

Eine grobe Zeitplanung wurde vor Beginn des Projekts durchgeführt, um die verfügbare Zeit den jeweiligen Projektphasen zuzuordnen. Diese grobe Zeitplanung ist in Tabelle 1 des Projektantrags dargestellt.

2.2. Ressourcenplanung

Im Anhang A.3 werden die verwendeten Ressourcen auf Seite iii aufgelistet, die während des Projekts eingesetzt wurden. Die Planung berücksichtigt sowohl Hard- als auch Software-Ressourcen sowie das beteiligte Personal. Um Kosten zu minimieren, wurde bei der Auswahl der verwendeten Software darauf geachtet, dass keine Lizenzgebühren anfallen, erforderliches Fachwissen vorhanden ist und die Architekturrichtlinien der GSSD eingehalten werden. Die Architekturrichtlinien der GSSD legen unter anderem die Nutzung von VSCode als Entwicklungsumgebung und den Jenkins-Server als Werkzeug für die CI fest. Durch die Einhaltung dieser Richtlinien wird sichergestellt, dass das Projekt den Anforderungen der GSSD entspricht und nahtlos in die bestehende Infrastruktur integriert werden kann.

2.3. Entwicklungsprozess

In der GSSD erfolgt die Entwicklung von Kundenprojekten üblicherweise durch den Einsatz der agilen Scrum-Methode. Der Entwicklungsprozess gliedert sich in mehrere ein- oder zweiwöchige Sprints, wobei Github zur Verwaltung der Aufgaben genutzt wird. Die Aufgaben werden den Sprints zugeordnet und während der wöchentlichen Sprintreviews überprüft.

Die umgesetzten Aufgaben werden gemeinsam mit dem Projektleiter kontrolliert, getestet und analysiert, um ein effektives Feedback und kontinuierliche Verbesserungen während des gesamten Entwicklungsprozesses zu gewährleisten. Diese strukturierte und kollaborative Vorgehensweise stellt sicher, dass die Endprodukte den Anforderungen und Erwartungen entsprechen.

3. Analysephase

3.1. Ist-Analyse

Der Kunde der GSSD verwendet auf seinen Servern verschiedene Anwendungen als Dienste oder im Hintergrund. Zur Überwachung dieser Anwendungen hat der Kunde bereits ein Überwachungssystem im Einsatz. Allerdings erweist sich dieses System als äußerst unzureichend und ineffizient in der Darstellung der Zustände und Informationen, die für die effektive Überwachung der Anwendungen erforderlich sind.

Die Ist-Analyse ergab folgende gravierende Schwachstellen im aktuellen Überwachungssystem des Kunden:

- stark eingeschränkte und unvollständige Anzeige von Systemzuständen
- mangelhafte Integration und Erkennung von Sensoren, was zu Informationslücken führt
- fehlende oder stark verzögerte Benachrichtigungen bei Systemproblemen, wodurch das Eingreifen des Administrators erschwert wird
- Schwierigkeiten bei der Skalierbarkeit und Anpassungsfähigkeit an neue Anforderungen, was die Erweiterung und Anpassung des Systems behindert
- unübersichtliche und verwirrende Benutzeroberfläche, die eine effiziente Nutzung erschwert
- hohe Latenzzeiten und Leistungseinbußen bei der Überwachung von Anwendungen

Um die Zuverlässigkeit und Funktionalität der überwachten Anwendungen sicherzustellen, müssen diese Schwachstellen dringend angegangen werden. Die Ergebnisse der Ist-Analyse bilden die Grundlage für die anschließende Planung und Implementierung von Verbesserungsmaßnahmen für das Überwachungssystem.

3.2. Workflow

Der Code definiert sowohl ein Datenmodell als auch einen Sensor mit verschiedenen Eigenschaften wie Name, Typ, Datentyp, Einheit, Minimum- und Maximumwert, Status, Zeitstempel und weiteren Optionen.

Es gibt zwei Arten von Sensoren: echte Sensoren und virtuelle Sensoren. Echte Sensoren senden kontinuierliche Daten an das Backend, während virtuelle Sensoren SQL-Abfragen an das Backend senden, um Daten aus der Datenbank abzurufen.

1. Der Prozess der Sensorregistrierung beginnt damit, dass der Sensorbauer eine API-Anfrage an das Backend sendet und seine Anmeldeinformationen übermittelt. Das Backend prüft, ob der Sensor bereits registriert ist. Wenn nicht, wird der Sensor in der Datenbank registriert und die Anmeldeinformationen werden an den Sensorbauer zurückgesendet.
2. Nach Erhalt der ID des Sensors kann der Sensor regelmäßig Daten an das Backend übermitteln und Konfigurationsdaten abrufen. Das Backend ruft die Konfigurationsdaten des Sensors über die entsprechende ID aus der Datenbank ab und aktualisiert seine Eigenschaften, wie Name, Typ, Datentyp und Einheit.
3. Der Sensor kann über die show-Eigenschaft angeben, ob er seine Daten an das Frontend senden und über saveData, ob er seine Daten in der Datenbank speichern möchte. Wenn der Sensor-Daten senden möchte, aktualisiert er das Datenfeld im Sensor-Modell und sendet es an das Backend.
4. Das Backend prüft, ob die Daten gemäß der in der variablen DataRetentionPeriodInMonths festgelegten Aufbewahrungsfrist gespeichert werden sollen. Wenn der saveData-Wert auf true gesetzt ist, speichert das Backend die Daten in der Datenbank und aktualisiert den Sensor-Status und den Zeitstempel.
5. Wenn der Sensor ein virtueller Sensor ist, kann er über die commandsql-Eigenschaft SQL-Abfragen an das Backend senden, um Daten aus der Datenbank abzurufen. Das Backend führt die Abfrage aus und sendet das Ergebnis an den virtuellen Sensor.
6. Das Frontend empfängt die Daten von den aktiven Sensoren und aktualisiert die Benutzeroberfläche entsprechend. Das Backend überwacht den Status der Sensoren und informiert den Sensorbauer über Probleme, die auftreten könnten, wie z.B. Sensorsausfälle oder Datenbankfehler.
7. Der Sensorbauer kann über die API auch neue Sensoren registrieren, bestehende Sensoren aktualisieren oder löschen sowie weitere Konfigurationen durchführen. Das Backend stellt hierfür entsprechende Funktionen bereit.

Das Komponenten Diagramm ist im Anhang A.10 auf Seite x zu finden.

3.3. Soll-Konzept

In Zusammenarbeit mit Vorgesetzten wurde das folgende Soll-Konzept entwickelt, welches die vom Kunden gestellten Anforderungen an das Überwachungssystem beschreibt. Das Überwachungssystem soll alle relevanten Anwendungen auf den Servern des Kunden kontinuierlich überprüfen und bei Fehlererkennung oder Ausfällen den Administrator zuverlässig benachrichtigen. Dabei legt der Kunde Wert auf eine einfache Konfiguration und Verwaltung des Systems. Um sicherzustellen, dass das Überwachungssystem den Erwartungen des Kunden gerecht wird, müssen zusätzlich bestimmte Leistungskriterien festgelegt werden. Dazu zählen beispielsweise die maximale Anzahl der zu überwachenden Anwendungen, die Reaktionszeit bei Fehlermeldungen und die Verfügbarkeit des Überwachungssystems.

3.4. Risikoanalyse

Die Analyse möglicher Probleme ist entscheidend, um potenzielle Risiken während der Projektumsetzung zu identifizieren und geeignete Gegenmaßnahmen zu planen. Hier sind die identifizierten Probleme und ihre entsprechenden Risikoanalysen:

1. Integration mit vorhandenen Systemen und Anwendungen,
2. Datenschutz und Sicherheit,
3. Skalierbarkeit und Anpassungsfähigkeit,
4. Kompetenz der beteiligten Mitarbeiter,
5. Budget- und Zeitbeschränkungen.

Durch die frühzeitige Identifizierung dieser potenziellen Probleme und die Umsetzung geeigneter Gegenmaßnahmen können die Risiken während der Projektumsetzung minimiert werden, was zu einer erfolgreichen Implementierung der maßgeschneiderten Überwachungslösung beiträgt.

3.5. Wirtschaftlichkeitsanalyse

Die Frage, ob sich die Entwicklung dieses Projekts für die GSSD lohnt, lässt sich relativ einfach beantworten. Vor Beginn des Projekts wurde der Arbeitsaufwand, unter Berücksichtigung des Tagessatzes, berechnet und dem Kunden ein entsprechendes Angebot unterbreitet. Die im Angebot aufgeführte Vergütung deckt die Projektkosten und erwirtschaftet zusätzlich einen finanziellen Gewinn für das Unternehmen. Eine Wirtschaftlichkeitsanalyse sollte daher die Vorteile des Kunden berücksichtigen. Allerdings gestaltet sich dies schwierig, da dem Autor kritische Daten wie beispielsweise täglicher/monatlicher/jährlicher Zeitaufwand oder Mitarbeiterlöhne des Kunden nicht vorliegen und auch nicht eingeholt werden können. Dennoch kann man davon ausgehen, dass der Kunde die Wirtschaftlichkeit dieser Entscheidung selbstständig berechnet hat und das Ergebnis für zufriedenstellend befunden wurde.

3.5.1. „Make or Buy“-Entscheidung

Die „Make or Buy“-Entscheidung ist ein wesentlicher Aspekt für Unternehmen, wenn es darum geht, ob sie eine Komponente oder einen Prozess intern entwickeln („Make“) oder von externen Anbietern kaufen („Buy“) sollen. Bei der Entscheidung für ein maßgeschneidertes Überwachungssystem oder eine vorhandene Lösung (z.B. Checkmk) sind mehrere Faktoren zu berücksichtigen:

- **Kosten:** Die Eigenentwicklung kann hohe Kosten verursachen, während beim Kauf Lizenz- und Supportkosten anfallen. Langfristige Kosteneinsparungen und Effizienz sind jedoch entscheidend.
- **Zeit:** Eigenentwicklungen benötigen mehr Zeit, während der Kauf einer Lösung eine schnellere Implementierung ermöglicht. Anpassungen können jedoch zusätzliche Zeit erfordern.
- **Expertise:** Internes Know-how kann die erfolgreiche Entwicklung eines maßgeschneiderten Überwachungssystems ermöglichen.
- **Anpassungsfähigkeit und Flexibilität:** Eine Eigenentwicklung bietet größere Anpassungsfähigkeit und Flexibilität für unternehmensspezifische Anforderungen.
- **Langfristige Perspektive:** Eine Eigenentwicklung bietet langfristig bessere Kontrolle und Anpassungsfähigkeit, während der Kauf einer Lösung kurzfristige Vorteile bietet.

Unter Berücksichtigung dieser Faktoren könnte der Kunde zu dem Schluss kommen, dass eine maßgeschneiderte Überwachungslösung langfristig bessere Ergebnisse ermöglicht. Daher wäre die „Make“-Entscheidung für die Entwicklung eines eigenen Überwachungssystems vorzuziehen. Der Projektfokus liegt auf der Eigenentwicklung einer maßgeschneiderten Softwarelösung, die den spezifischen Anforderungen und Bedürfnissen des Kunden entspricht. Die Vorteile sind gezielte

Gestaltung von Workflow und Benutzerfreundlichkeit, vollständige Kontrolle über Code und Implementierung, effektive Fehlerbehebung und kontinuierliche Verbesserung. Daher ist eine Eigenentwicklung, die ideale Lösung für dieses Projekt.

3.5.2. Projektkosten

Die Kosten für das Projekt umfassen sowohl Personalkosten als auch sonstige Aufwendungen wie Büroflächen, Stromkosten, Kommunikationskosten usw. Im folgenden ist eine detaillierte Kostenaufstellung für das Projekt dargestellt:

- Personalkosten: Da genaue Personalkosten vertraulich sind, basieren die Berechnungen auf Stundensätzen, die von der Personalabteilung festgelegt wurden. Für Mitarbeiter beträgt der Stundensatz 35,00 €, während der Stundensatz für Auszubildende bei 10,50 € liegt.
- Ressourcenkosten: Die Kosten für die in Abschnitt 2.2 (Ressourcenplanung) aufgeführten Ressourcen betragen pauschal 15,00 € über die Projekt-Laufzeit.
- Sonstige Aufwendungen: Für Büroflächen, Stromkosten, Kommunikationskosten und weitere allgemeine Kosten wird eine Pauschale von 300,00 € veranschlagt.

Die Durchführungszeit des Projekts beträgt 80 Stunden. Die untenstehende Tabelle zeigt die Kostenaufstellung, aufgeteilt nach den einzelnen Projektphasen:

Vorgang	Mitarbeiter	Zeit	kosten pro Stunde	Gesamt
Entwicklungskosten im Rahmen der GMS	1x Auszubildende	80	10,50	840,00
Abnahme der Dokumentation	1x Mitarbeiter	2h	35,00 €/h	70,00 €
Aufsicht bei Projektplanung	1x Mitarbeiter	1h	35,00 €/h	35,00 €
Hilfestellung bei Problemen	1x Mitarbeiter	2h	35,00 €/h	70,00 €
Pauschalkosten				315,00 €
				Gesamt 1.330,00 €

Tabelle 1: Übersicht der angefallenen Kosten

3.5.3. Amortisationsdauer

Die Ermittlung der Amortisationsdauer für den Kunden in diesem Projekt gestaltet sich äußerst komplex, da es weder Vergleichsdaten aufgrund fehlender ähnlicher Anwendungen gibt, noch konkrete Zeitaufwandsangaben vorliegen, wie bereits in Abschnitt 3.2 erwähnt. Für die GSSD hingegen amortisiert sich das Projekt relativ schnell, da die im Vertrag mit dem Kunden vereinbarten Mitarbeiterstunden in Rechnung gestellt und beglichen werden. Zusätzliche Kosten, die nach der Inbetriebnahme anfallen, wie beispielsweise die Anmietung von Servern zur Bereitstellung der produktiven Anwendung, werden direkt vom Kunden getragen. Daher entstehen für die GSSD keine weiteren Kosten, die nicht durch die Zahlungen des Kunden gedeckt wären.

3.6. Lastenheft

Das Lastenheft beschreibt die Anforderungen an die Überwachung eines komplexen Systems aus Sicht des Kunden. Im Anhang A.1 auf Seite i ist das detaillierte Lastenheft des Kunden zu finden.

4. Entwurfsphase

4.1. Zielplattform

Die zu entwickelnde Lösung ist eine Web-Applikation. Das System besteht aus Backend und Frontend, welche über Websockets Daten austauschen. Die Datenbank wurde mit mariaDB realisiert und enthält Tabellen mit Sensor-Daten und Verwaltungsdaten. Das Backend (Node.js, Express) liest und bewertet die Daten und das Frontend (Vue.js) zeigt die ausgewerteten Daten auf dem Dashboard an.

4.2. Architekturdesign

Das Architekturdesign für die Anwendungsüberwachung besteht aus mehreren Komponenten, die gemeinsam eine umfassende Monitoring-Lösung bieten. Die Architektur beinhaltet das Backend und das Frontend.

Das Backend wird in Node.js entwickelt und ist für die Verwaltung der Datenbank und der API-Schnittstellen verantwortlich. Es sammelt und analysiert Daten von verschiedenen Anwendungen und speichert sie in der Datenbank. Das Backend verwendet eine REST-API, um die Daten an das Frontend weiterzugeben.

Das Frontend, das in Vue.js entwickelt wurde, bietet eine grafische Benutzeroberfläche, die den Zustand der Anwendungen visualisiert. Es kommuniziert über WebSocket-Schnittstellen mit dem Backend, um Echtzeitdaten anzuzeigen und interaktive Funktionen wie das Hinzufügen oder Entfernen von Anwendungen sowie das Konfigurieren von Warnmeldungen bereitzustellen.

Das Überwachungssystem setzt sich aus Sensoren zusammen, die in jeder Anwendung integriert sind und Daten über deren Leistung und Verfügbarkeit sammeln. Diese Sensoren senden in regelmäßigen Abständen Daten an das Backend, um analysiert und gespeichert zu werden.

4.3. Entwurf der Benutzeroberfläche

Der Entwurf der Benutzeroberfläche für das Überwachungssystem besteht aus verschiedenen Elementen, um eine intuitive und ansprechende Umgebung für die Benutzer zu schaffen. Hier sind die Hauptkomponenten des Benutzeroberflächenentwurfs:

1. Navigation: Eine Seitenleiste oder ein Menü am seitlichen Rand der Benutzeroberfläche ermöglichen den Benutzern den Zugriff auf verschiedene Abschnitte der Anwendung, z.B. Dashboard, Anwendungsverwaltung und Einstellungen.
2. Dashboard: Das Dashboard ist der zentrale Bereich der Benutzeroberfläche, in dem die aktuellen Statusinformationen der überwachten Anwendungen angezeigt werden. Es kann Kacheln oder Karten enthalten, die für jede Anwendung einen schnellen Überblick über den Zustand, die Leistung und etwaige Warnmeldungen bieten.
3. Anwendungsverwaltung: In diesem Abschnitt können Benutzer neue Anwendungen hinzufügen, vorhandene Anwendungen entfernen oder konfigurieren und die Sensoren für die Überwachung anpassen.
4. Einstellungen: Das ist ein Bereich für die Verwaltung von Benutzerkonten, Systemeinstellungen und Benachrichtigungsoptionen.
5. Filter- und Suchfunktion: Sie bieten eine Möglichkeit für Benutzer, die angezeigten Anwendungen und Daten schnell zu filtern oder nach bestimmten Anwendungen oder Kriterien zu suchen.
6. Warnmeldungen: Eine Liste oder ein Bereich, der die aktiven Warnmeldungen und kritischen Ereignisse für die überwachten Anwendungen anzeigt. Administratoren können die Warnmeldungen ein- oder ausblenden und Details zu jedem Ereignis anzeigen.

7. Diagramme und Statistiken: Für jede Anwendung können detaillierte Diagramme und Statistiken zur Leistung und Verfügbarkeit angezeigt werden. Dies kann in Form von Liniendiagrammen, Balkendiagrammen oder Tortendiagrammen erfolgen, je nach Art der Daten und der gewünschten Darstellung.
8. Responsives Design: Die Benutzeroberfläche sollte so gestaltet sein, dass sie auf verschiedenen Bildschirmgrößen und Geräten gut aussieht und funktioniert, einschließlich Desktop-Computern, Tablets und Mobiltelefonen.

Durch die Kombination dieser Elemente wird eine benutzerfreundliche und effektive Benutzeroberfläche geschaffen, die es den Benutzern ermöglicht, die Überwachungsinformationen für ihre Anwendungen leicht zu überprüfen und zu verwalten.

4.4. Datenmodell

Basierend auf dem Vertrag zwischen der GSSD und dem Kunden sowie den zu berücksichtigenden Anwendungsfällen wurde vom Autor die erforderliche Datenbankstruktur analysiert und mithilfe von Node.js-Migrationen erstellt.

Diese Datenbankstruktur war auch für die Entwicklung des Frontends von entscheidender Bedeutung, da nahezu jede Entität durch eine eigene Unterseite in der Anwendung repräsentiert wird. Ein vereinfachtes ERM, das die Entitäten, Beziehungen und Kardinalitäten zeigt, befindet sich in Anhang B auf Seite xi.

4.5. Maßnahmen zur Qualitätssicherung

Ein zentrales Element der agilen Entwicklung bei der GSSD ist die Durchführung regelmäßiger, technisch fokussierter Meetings. In diesen Sitzungen werden abgeschlossene Arbeitspakete präsentiert, gemeinsam mit dem Projektleiter getestet und gründlich analysiert. Die interdisziplinäre Zusammenarbeit ermöglicht eine fortlaufende Integration der erlangten Erkenntnisse in den Entwicklungsprozess, wodurch die Anwendungsleistung und Codequalität ständig optimiert werden. Die kontinuierliche Qualitätssicherung ermöglicht es, anspruchsvolle Herausforderungen im Entwicklungsprozess effektiv zu bewältigen. Dies resultiert in einer erheblichen Zeitsparnis während der Entwicklung und schafft eine solide Grundlage für eine zukunftssichere und nachhaltige Weiterentwicklung der Anwendung im technischen Bereich.

Um die Qualität des Projekts zu gewährleisten, wurden folgende Maßnahmen umgesetzt:

1. **Unit- und Widget-Tests:** Diese Testverfahren validieren die Richtigkeit der Implementierung und sorgen für eine zuverlässige Funktionalität der Anwendung.
2. **Regelmäßige Abstimmungen:** Die Kommunikation mit dem Entwicklungsleiter dient dazu, Abweichungen frühzeitig zu identifizieren und entsprechende Korrekturmaßnahmen einzuleiten.
3. **GIT-Versionsverwaltung:** Durch den Einsatz von GIT wird die Transparenz der Softwareentwicklung gewährleistet.
4. **CI:** Mithilfe von Jenkins wird nach jedem Push-Vorgang eine automatische Prüfung der Software durchgeführt, um die Qualität ständig zu überwachen und sicherzustellen.

4.6. Deployment

Im Rahmen des Projekts wird die entwickelte Überwachungssoftware auf den Servern des Kunden implementiert und betriebsbereit gemacht. Dabei werden folgende Schritte unternommen:

- Installation der Software: Die entwickelte Anwendung wird auf den vom Kunden bereitgestellten Servern installiert. Dabei werden alle notwendigen Abhängigkeiten und Komponenten eingerichtet.

- Konfiguration von Systemeinstellungen: Systemeinstellungen und -parameter werden gemäß den Anforderungen und der Infrastruktur des Kunden angepasst.
- Integration mit anderen Anwendungen und Systemen: Die Überwachungssoftware wird mit bestehenden Anwendungen und Systemen des Kunden integriert, um eine nahtlose Zusammenarbeit und Informationsaustausch mit den Sensoren zu gewährleisten.
- Durchführung von Tests: Nachdem die Software erfolgreich installiert und konfiguriert wurde, werden Tests durchgeführt, um sicherzustellen, dass sie korrekt funktioniert und keine Fehler oder Probleme auftreten.
- Dokumentation: Die Installations- und Konfigurationsprozesse werden dokumentiert, um dem Kunden eine Referenz und Anleitung für zukünftige Anpassungen oder Updates zu bieten.

Das Deployment der Überwachungssoftware sollte sorgfältig geplant und durchgeführt werden, um mögliche Ausfallzeiten oder Probleme zu minimieren und einen reibungslosen Übergang zu gewährleisten. Hierzu wird ein entsprechender Plan erstellt, der die Aufgaben, Zuständigkeiten und Zeitpläne für das Deployment festlegt. Es werden auch Backup-Pläne und Wiederherstellungsmöglichkeiten eingerichtet, um Datenverluste oder Störungen zu vermeiden. Ein erfolgreicher Abschluss des Deployments stellt sicher, dass das Überwachungssystem effektiv arbeitet und den Kundenbedürfnissen entspricht.

4.7. Pflichtenheft

Nach Abschluss der Entwurfsphase wurde ein detailliertes Pflichtenheft erstellt, in dem die technischen Lösungen zur Erfüllung der Anforderungen des Lastenhefts dokumentiert sind. Es dient als technische Grundlage für die Implementierungsphase und stellt sicher, dass die umgesetzten Funktionen den Kundenerwartungen entsprechen. Ein Auszug des Pflichtenhefts ist im Anhang A.2 auf Seite ii abgebildet.

5. Implementierungsphase

5.1. Entwicklung des Dashboards

In der Entwicklungsphase des Dashboards lag der Fokus auf der Nutzung von wiederverwendbaren und verschachtelbaren Vue-Komponenten, um eine modulare und erweiterbare Benutzeroberfläche zu schaffen. Das Dashboard besteht aus diversen Komponenten, die spezifische Informationen, wie Benutzerstatistiken, Systemzustand oder Leistungskennzahlen, darstellen. Diese Komponenten wurden auf der Dashboard-Seite integriert, um einen umfassenden und aktuellen Einblick in die Leistung des Überwachungssystems zu ermöglichen.

Die einzelnen Dashboard-Komponenten wurden so konzipiert, dass sie sowohl autonom als auch in Kombination miteinander funktionieren. Hierbei wurde die Flexibilität von Vue.js genutzt, um die Anordnung und Darstellung der Komponenten dynamisch an die Erfordernisse des Projekts anzupassen. Ein Beispiel hierfür ist die Implementierung von vuety, einem Material Design Framework für Vue.js, das eine Vielzahl von vordefinierten Komponenten und Layouts bereitstellt.

Dank der Anwendung asynchroner Funktionen und reaktiver Daten innerhalb der Dashboard-Komponenten werden die dargestellten Informationen kontinuierlich aktualisiert und auf dem neuesten Stand gehalten. Dies ermöglicht die Echtzeitüberwachung verschiedener Systemaspekte und trägt zur raschen Identifizierung von Problemen und Leistungseinbußen bei. Im Anhang sind beispielhafte Screenshots des entwickelten Dashboards enthalten, die verschiedene Komponenten und ihre Anordnung innerhalb der Benutzeroberfläche verdeutlichen. Diese Screenshots zeigen, wie das Dashboard eine strukturierte und benutzerfreundliche Visualisierung relevanter

Informationen bietet und gleichzeitig flexibel genug ist, um an die spezifischen Anforderungen des Überwachungsprojekts angepasst zu werden.

5.2. Datenbank

Die verwendete Datenbank für das Projekt ist mariaDB. mariaDB ist ein weit verbreitetes, relationales Datenbankmanagementsystem, das sich durch seine Skalierbarkeit und Flexibilität auszeichnet. Es ist Open-Source und kann kostenfrei genutzt werden.

Die Integration von mariaDB in das Backend ermöglicht die effiziente Verwaltung und Speicherung der gesammelten Daten aus den verschiedenen Anwendungen. Durch die Verwendung von mariaDB können auch komplexe Abfragen und Analysen der gespeicherten Daten durchgeführt werden, um wertvolle Erkenntnisse über die Leistung und Verfügbarkeit der überwachten Anwendungen zu gewinnen.

Die Verbindung zwischen dem Backend und der mariaDB-Datenbank wird über entsprechende Treiber und Bibliotheken in Node.js hergestellt. Die Datenbankstruktur wird so gestaltet, dass sie leicht erweitert und angepasst werden kann, um neue Anwendungen oder Sensoren hinzuzufügen oder um zusätzliche Funktionen und Warnmeldungen zu unterstützen.

Insgesamt bietet die Verwendung von mariaDB als Datenbanklösung für das Projekt eine robuste und skalierbare Grundlage, die den Anforderungen des Kunden gerecht wird und die Möglichkeit bietet, das Überwachungssystem im Laufe der Zeit weiterzuentwickeln und zu optimieren.

5.3. REST-API

In diesem Projekt wird eine REST-API verwendet, um den Datenaustausch zwischen den Sensoren und dem Backend zu ermöglichen. Die REST-API stellt eine standardisierte und leicht zu verwendende Schnittstelle bereit, um Daten zwischen diesen Systemkomponenten auszutauschen.

Die Hauptmerkmale der in diesem Projekt verwendeten REST-API sind:

1. Zustandslosigkeit: Jede Anfrage von Client zu Server enthält alle notwendigen Informationen, sodass der Server den Kontext der Anfrage nicht speichern muss. Dies führt zu einer besseren Skalierbarkeit und Vereinfachung der Serverlogik.
2. Cache-Fähigkeit: Die API-Antworten können gecacht werden, um die Leistung zu verbessern und die Last auf dem Server zu reduzieren.
3. Client-Server-Architektur: Die REST-API trennt die Benutzeroberfläche (Frontend) von der Backend-Logik und der Datenverarbeitung. Dies erlaubt eine unabhängige Entwicklung und Verbesserung der einzelnen Komponenten.
4. Einheitliche Schnittstelle: Die REST-API stellt eine einheitliche und konsistente Schnittstelle bereit, die die Interaktion zwischen den Komponenten vereinfacht.

Im Kontext des Projekts werden die Sensoren die REST-API verwenden, um Daten an das Backend zu senden oder abzufragen. Das Backend empfängt die Daten von den Sensoren, analysiert und speichert sie in der Datenbank.

Für die Kommunikation zwischen dem Backend und dem Frontend wird ein WebSocket verwendet. Dies ermöglicht eine bidirektionale Kommunikation in Echtzeit zwischen den beiden Komponenten. Das Frontend verwendet den WebSocket, um die Daten vom Backend abzurufen und sie in einer benutzerfreundlichen Weise darzustellen.

Die Verwendung der REST-API für die Sensoren und die Websockets für die Kommunikation zwischen Backend und Frontend gewährleistet eine effiziente und zuverlässige Kommunikation, wodurch ein effektives Überwachungssystem für die Anwendungen auf den Servern des Kunden geschaffen wird.

5.4. Testen der Anwendung

Um sicherzustellen, dass die entwickelte Anwendung fehlerfrei funktionierte und die definierten Anforderungen erfüllte, war ein systematischer Testprozess erforderlich. Hierbei kamen verschiedene Testansätze und -ebenen zum Einsatz:

1. Unit-Tests

- Unit-Tests konzentrierten sich auf einzelne Codeeinheiten wie Funktionen oder Klassen. Sie gewährleisteten, dass diese korrekt arbeiteten und die erwarteten Ergebnisse lieferten.
- In der Regel wurden Unit-Tests mit Hilfe von Test-Frameworks wie JUnit Mocha (für JavaScript) erstellt und automatisiert ausgeführt.

2. Integrationstests

- Integrationstests überprüften die korrekte Interaktion zwischen verschiedenen Komponenten der Anwendung, wie zum Beispiel Datenbankzugriffe oder Kommunikation zwischen Backend und Frontend.
- Diese Tests wurden sowohl auf Code- als auch auf Systemebene durchgeführt, abhängig von den zu testenden Komponenten.

3. Systemtests

- Systemtests prüften die Anwendung in ihrer Gesamtheit, um sicherzustellen, dass alle Komponenten ordnungsgemäß zusammenspielten und die Anwendung wie beabsichtigt funktionierte.
- Diese Tests umfassten oft auch Last- und Performance-Tests, um die Leistungsfähigkeit und Stabilität der Anwendung unter verschiedenen Bedingungen zu überprüfen.

4. Akzeptanztests

- Akzeptanztests, auch bekannt als End-to-End-Tests oder Benutzertests, stellten sicher, dass die Anwendung den Anforderungen der Endbenutzer entsprach und alle definierten Use Cases abdeckte.
- Diese Tests wurden manuell durchgeführt oder mithilfe von Test-Tools wie Selenium automatisiert, um Benutzerinteraktionen mit der Anwendung zu simulieren.

Ein effektiver Testprozess umfasste sowohl manuelle als auch automatisierte Tests und folgte den Prinzipien des TDD oder BDD, bei denen Tests als integraler Bestandteil des Entwicklungsprozesses betrachtet wurden. Regelmäßige Code-Reviews und die Verwendung von CI und CD trugen ebenfalls zur Qualitätssicherung der Anwendung bei.

5.5. Sicherheitstests

Die Sicherheit des Überwachungssystems ist von großer Bedeutung, da es sensible Daten über die überwachten Anwendungen verarbeitet. Daher wurden Sicherheitstests durchgeführt, um potenzielle Schwachstellen und Angriffsvektoren zu identifizieren und entsprechende Gegenmaßnahmen zu implementieren. Dazu zählen unter anderem Tests zur Überprüfung der Authentifizierung, Autorisierung und Verschlüsselung.

6. Abnahmephase

6.1. Code-Review

Code-Reviews waren ein wesentlicher Bestandteil des Entwicklungsprozesses, um die Codequalität zu gewährleisten und eine kontinuierliche Verbesserung des Codes zu fördern. Sie dienten dazu, potenzielle Fehler frühzeitig zu erkennen und Best Practices für die Codeentwicklung zu fördern.

Während des Entwicklungsprozesses wurden Code-Reviews durchgeführt, um sicherzustellen, dass:

1. der Code den vereinbarten Programmierstandards und Richtlinien entsprach,
2. der Code gut strukturiert, lesbar und wartbar ist,
3. der Code effizient und performant ist,
4. der Code frei von Sicherheitslücken und Anfälligkeiten ist,
5. der Code keine unbeabsichtigten Seiteneffekte oder Regressionen verursacht.

Um Code-Reviews effektiv zu gestalten, wurden folgende Best Practices angewendet:

1. Die Verwendung von PRs in Git, um Änderungen am Code vor der Integration in den Hauptzweig zu überprüfen. Dies ermöglichte es den Entwicklern, Feedback zu geben und Probleme gemeinsam zu lösen.
2. Die Einhaltung einer Checkliste für Code-Reviews, um sicherzustellen, dass alle wichtigen Aspekte des Codes überprüft wurden.
3. Die Durchführung von periodischen Code-Reviews mit dem Ziel, mögliche Verbesserungen zu realisieren.

Durch das Implementieren dieser Praktiken wurden Code-Reviews zu einem wichtigen Instrument zur Qualitätssicherung und zur Verbesserung des Codes.

6.2. Abnahme

Im Rahmen dieses Projekts fand keine separate Abnahmephase statt. Stattdessen wurden die Anwendung, ihre Benutzeroberfläche sowie die Interaktion zwischen Frontend und Backend kontinuierlich während des Entwicklungsprozesses in gemeinsamen Sprint-Reviews getestet und besprochen. Durch diesen Ansatz konnte eine interne Abnahme des Projekts bereits während der Entwicklungsphase sichergestellt werden.

Trotzdem wurde eine formelle Testphase der Benutzeroberfläche, einschließlich JavaScript-Testfällen, durchgeführt. Aufgrund von Verzögerungen kann diese Phase jedoch nicht ausführlich in dieser Projektdokumentation erklärt und berücksichtigt werden.

7. Dokumentation

Im Rahmen des Projektabschlusses für das spezifische Projekt, das sich auf die Entwicklung einer webbasierten Anwendung für die GSSD konzentriert, wurden zwei zentrale Dokumentationen erstellt, um sowohl Endanwendern als auch Entwicklern eine umfassende Informationsquelle zur Verfügung zu stellen. Beide Dokumente wurden fachlich fundiert und ansprechend gestaltet, um ihren jeweiligen Zielgruppen den bestmöglichen Nutzen zu bieten. Im Folgenden werden die spezifischen Ziele und Inhalte dieser Dokumentationen in Bezug auf das Projekt erläutert:

7.1. Benutzerhandbuch Endanwender-Dokumentation

Das Benutzerhandbuch wurde für die Endanwender der GSSD Anwendung konzipiert und bietet eine systematische Einführung in die Plattform, ihre Funktionen und Bedienung. Hierbei wurden ansprechende visuelle Darstellungen und präzise Anweisungen verwendet, um den Anwendern ein effizientes und angenehmes Nutzungserlebnis zu ermöglichen. Das Handbuch enthält detaillierte Schritt-für-Schritt-Anleitungen zur Verwendung der verschiedenen Module und Funktionen der Anwendung, wie z.B. Datenverwaltung, Berichterstellung und Zusammenarbeit zwischen verschiedenen Nutzergruppen. Zudem werden Best Practices und nützliche Tipps für den optimalen Einsatz der Software innerhalb der GSSD bereitgestellt.

7.2. Entwicklerdokumentation

Die Entwicklerdokumentation richtet sich an Fachleute, die an der Weiterentwicklung, Wartung oder Integration der GSSD-Anwendung beteiligt sind. Diese Dokumentation wurde sorgfältig erstellt, um ein tiefgehendes Verständnis der technischen Aspekte der Software zu vermitteln, einschließlich ihrer Architektur, des Codes, der verwendeten Technologien wie Node.js, Vue.js und MariaDB sowie der eingesetzten Bibliotheken und Frameworks. Um den Entwicklern eine strukturierte Orientierungshilfe zu bieten, wurden Best Practices und Richtlinien für die Projektstruktur, das Einrichten der Entwicklungsumgebung, das Durchführen von Tests und das hinzufügen neuer Funktionen integriert. Die Entwicklerdokumentation soll somit eine solide Grundlage für eine effektive Zusammenarbeit und eine qualitativ hochwertige Weiterentwicklung des GSSD-Projekts schaffen.

8. Fazit

Das vom Autor entwickelte Überwachungssystem erfüllt alle im Projektantrag definierten Anforderungen effizient und zuverlässig. Es bietet eine optimale Überwachung der Anwendungen auf den Servern des Kunden. Die identifizierten Schwachstellen des bestehenden Systems werden behoben. Die Anwendungen sind anpassbar und skalierbar, so dass sie mit den Bedürfnissen des Kunden mitwachsen können.

Während der Projektdurchführung traten einige Herausforderungen auf, wie z.B. die Anpassung des Zeitplans und die Identifizierung der Anforderungen im Lastenheft. Diese wurden jedoch erfolgreich gemeistert und die Erfahrungen können für zukünftige Projekte genutzt werden. Die Zusammenarbeit und Kommunikation mit dem Kunden waren ebenfalls wichtige Aspekte, die während des Projekts berücksichtigt und erfolgreich bewältigt wurden.

8.1. Soll-Ist-Vergleich

Nach Abschluss des Projekts kann festgestellt werden, dass alle im Projektantrag genannten Anforderungen erfolgreich umgesetzt wurden. Jedoch musste der Zeitplan im Verlauf des Projekts angepasst werden, da die Umsetzung des Konzepts mehr Zeit in Anspruch nahm als ursprünglich vermutet. Die Analyse des Lastenhefts und die Ermittlung der Pflichten waren aufgrund der hohen Anzahl an einzelnen Komponenten und des komplexen Prozesses der Rechtevergabe zeitintensiver als erwartet. Die Projektplanungsphase wurde aufgrund der fehlenden Erfahrungen im Vorfeld als aufwendiger eingeschätzt. Die zeitliche Schätzung der weiteren Phasen erwies sich insgesamt als zutreffend. Etwa 9 Stunden Arbeitszeit konnten keiner Projektphase direkt zugeordnet werden, da sie für die Überprüfung der Rahmenbedingungen wie das Layout und die Erstellung erforderlicher Anlagen verwendet wurden.

8.2. Lessons Learned

Während des Projekts wurden verschiedene Erfahrungen gesammelt und wichtige Erkenntnisse gewonnen, die in zukünftigen Projekten genutzt werden können, um den Erfolg und die Effizienz der Projektarbeit zu verbessern. Einige der wichtigsten Lektionen sind:

- **Realistische Zeitplanung:** Die Anpassung des Zeitplans während der Projektumsetzung zeigt, wie wichtig es ist, genügend Zeit für die verschiedenen Phasen und Aufgaben einzuplanen. Dabei sollte auch ausreichend Pufferzeit für unvorhergesehene Probleme oder Verzögerungen berücksichtigt werden.

- **Kommunikation und Zusammenarbeit:** Obwohl das Projekt allein durchgeführt wurde, war eine klare Kommunikation mit den Stakeholdern, insbesondere dem Kunden, entscheidend für den Projekterfolg. Regelmäßige Updates und Feedback-Runden stellen sicher, dass alle Anforderungen korrekt verstanden und umgesetzt werden und ermöglichen eine kontinuierliche Verbesserung während des Entwicklungsprozesses.
- **Umfassende Analyse und Dokumentation:** Eine gründliche Analyse der bestehenden Systeme und der Anforderungen des Kunden ist entscheidend für die erfolgreiche Umsetzung eines Projekts. Eine ausführliche und gut strukturierte Dokumentation hilft dabei, den Projektverlauf nachzuvollziehen und eventuelle Probleme oder Verbesserungsmöglichkeiten zu identifizieren.
- **Fehlerbehebung und Tests:** Eine sorgfältige Fehlerbehebung und umfassende Tests während des gesamten Entwicklungsprozesses stellen sicher, dass die entwickelte Lösung den Anforderungen entspricht und funktioniert, wie sie soll. Frühzeitiges Erkennen und Beheben von Fehlern minimiert das Risiko von Problemen in späteren Projektphasen.
- **Anpassungsfähigkeit und Lernbereitschaft:** Die Fähigkeit, sich an Veränderungen oder neue Anforderungen während des Projekts anzupassen, ist entscheidend für den Projekterfolg. Die Bereitschaft, neue Technologien oder Ansätze zu erlernen und anzuwenden, kann dazu beitragen, die Projekteffizienz zu steigern und die Qualität der entwickelten Lösung zu verbessern.
- **Planung für Skalierbarkeit:** Bei der Entwicklung der Überwachungssoftware war es wichtig, die Skalierbarkeit der Lösung zu berücksichtigen, um zukünftige Erweiterungen oder Anpassungen an neue Anforderungen zu ermöglichen. Das Einplanen der Skalierbarkeit reduziert von Anfang an die Wahrscheinlichkeit, dass umfangreiche Änderungen in späteren Projektphasen oder nach der Fertigstellung erforderlich sind.

Die gewonnenen Erkenntnisse und Lektionen können dazu beitragen, zukünftige Projekte besser zu planen, durchzuführen und abzuschließen, was zu höherer Effizienz, erfolgreichen Projektergebnissen und zufriedenen Kunden führt. Es ist wichtig, diese Lektionen zu dokumentieren und in zukünftigen Projekten zu berücksichtigen, um wiederkehrende Probleme zu vermeiden und die Qualität der Projektergebnisse kontinuierlich zu verbessern.

8.3. Ausblick

Im Ausblick werden potenzielle Erweiterungen des Systems oder zukünftige Entwicklungen aufgezeigt, die auf Kundenfeedback, Marktanforderungen und technologischen Fortschritten basieren. Die Integration von KI in das System ermöglicht es, zukünftige Entwicklungen zu antizipieren und gezielt darauf zu reagieren, um die Wettbewerbsfähigkeit des Systems zu erhalten. In diesem Zusammenhang können auch fortgeschrittene Analysemethoden wie Machine Learning und Predictive Analytics eingesetzt werden, um Einblicke in Kundenverhalten und Geschäftsentwicklungen zu gewinnen und das System entsprechend zu optimieren. Der Ausblick kann auch Informationen über geplante Verbesserungen oder neue Features enthalten, die in zukünftigen Releases des Systems implementiert werden sollen und durch den Einsatz von KI und anderen fortschrittlichen Technologien unterstützt werden.

Literaturverzeichnis

NodeJs v18.16.0

Available at: <https://nodejs.org/>
Accessed March 2023

expressjs v4.18.2

Available at: <https://expressjs.com/>
Accessed March 2023

Mozilla v3

Available at: <https://developer.mozilla.org/>
Accessed March 2023

Axios v1.3.5

Available at: <https://developer.mozilla.org/>
Accessed March 2023

Vuejs v3

Available at: <https://vuejs.org/>
Accessed March 2023

Vuex v3

Available at: <https://Vuex.vuejs.org/>
Accessed March 2023

Vuetify v3

Available at: <https://vuetify.org/>
Accessed March 2023

Vue Router v3

Available at: <https://router.vuejs.org/>
Accessed March 2023

MariaDB v11.1.0

Available at: <https://mariadb.com/>
Accessed March 2023

HelmetJs v6.1.5

Available at: <https://helmetjs.github.io/>
Accessed March 2023

A. Anhang

A.1. Lastenheft

Ziel des zu entwickelnden Systems ist die Überwachung eines komplexen Systems, bestehend aus mindestens 10 Komponenten unterschiedlicher Art. Das System soll dem Benutzer ermöglichen, den aktuellen Zustand des Systems auf einen Blick zu bewerten und auf kommende Probleme hinzuweisen. Das System soll folgende Funktionalitäten bereitstellen:

- Eine einfache und intuitive Anzeige des Ist-Zustands des Systems, welche eine schnelle und ge- naue Beurteilung des Systemzustands ermöglicht.
- Eine Bewertungsanzeige, welche den Benutzer auf kommende Probleme hinweist und eine auto- matische Alarmierung bei kritischen Zuständen des Systems ermöglicht.
- Definition von Schwellwerten für jeden Sensor, um eine effektive Überwachung des Systems zu ermöglichen.
- Zusammenfassung der Sensorwerte in einer geeigneten Form, um eine schnelle Bewertung des Systemzustands zu ermöglichen.
- Langzeitanalyse der Sensorwerte für Trends und Muster im Systemverhalten.
- Interaktive Dashboards für detaillierte Ansichten und individuelle Einstellungen.
- Die Sensoren sollen Datenbank-Einträge abfragen. Hilfsprozesse sollen Sensor-Daten in eine Da- tenbank eintragen, um einen entsprechenden Sensor zu bauen.
- Die zeitliche Auflösung der Sensoren muss den erfassten Daten entsprechend angepasst werden können.
- Das System soll diskrete Zustandsbeschreibungen bereitstellen. Die Anzeige kann durch verschie- dene Darstellungsformen, wie beispielsweise die Größe, erfolgen. Wichtig ist, dass die Anzeige mit einem kurzen Blick zu erfassen ist.
- Das System soll so gestaltet sein, dass es einfach zu bedienen und zu warten ist. Es soll auf einer geeigneten Plattform entwickelt werden, welche eine hohe Verfügbarkeit und Sicherheit gewähr- leistet.

A.2. Pflichtenheft

Basierend auf dem Lastenheft und unter Berücksichtigung der verfügbaren Ressourcen wurden die folgenden Anforderungen im Pflichtenheft festgelegt:

1. Plattform
 - Das Backend wird mit Node.js implementiert.
 - Die Webkomponenten werden unter Verwendung von Vue.js entwickelt.
 - Die Webanwendung ist ausschließlich im Intranet erreichbar.
 - Die Webanwendung wird auf einem Linux-Server innerhalb eines Docker-Containers ausgeführt.
 - GIT wird als System für die Versionskontrolle genutzt.
 - Github dient als gehostete Softwarelösung für die Versionsverwaltung der GSSD.
 - Der Jenkins-Server ist für die CD zuständig.
2. Datenbank
 - Die Datenbankanbindung erfolgt über NodeJs.
 - Daten, die älter als 6 Monate sind, werden gelöscht.
 - Die Daten werden in einer MariaDB-Datenbank gespeichert.
 - Die Repository-Klassen implementieren vorgegebene Interfaces, die die erforderlichen Methoden definieren.
 - Die Persistenzschicht wird mit Hilfe von Testcontainern getestet.
3. Benutzeroberfläche
 - Die Benutzeroberflächen werden mit Vue.js erstellt.
 - Die Benutzeroberflächen werden unter Verwendung von CSS 3 gestaltet und sollen die Corporate Identity der GSSD wahren.
 - Die Benutzeroberflächen sind nicht responsiv gestaltet.
 - Die Benutzeroberfläche greift über WebSocket-Schnittstellen auf das Backend zu, um Daten anzuzeigen und interaktive Funktionen bereitzustellen, wie z.B. das Hinzufügen oder Entfernen von Anwendungen oder das Einstellen von Warnmeldungen.
4. Geschäftslogik
 - Zur Erzeugung von Objekten wird CDI verwendet.
 - Das Framework JUnit 5 wird für automatisierte Tests eingesetzt.

A.3. Verwendete Ressourcen

Hardware

- Laptop
- Büroeigener Server – Bereitstellung der Testumgebung

Software

- Figma – Programm zur Erstellung von Mockups
- VSCode – Code Editor
- Node.js – plattformunabhängige JavaScript-Laufzeitumgebung
- NPM – Package Manager für das Projekt
- Vue.js – JavaScript-Framework für Benutzeroberflächen
- VSCode mit MiKTeX – Entwicklungsumgebung für L^AT_EX
- Draw.io – Website zum Erstellen von Diagrammen
- Lucidchart – Website zum Erstellen von Diagrammen
- GIT – Verteilte Versionsverwaltung
- Jenkins – Buildserver
- Postman – Test und Dokumentation von REST-APIs
- JUnit – Framework zur Durchführung von Unit-Tests
- MiKTeX – Distribution des Textsatzsystems T_EX
- Mockito – Mocking-Framework zur Erstellung von Pseudoklassen
- MariaDB – Datenbanksystem
- Github – Projektmanagement
- Windows 10 – Betriebssystem
- Internet Browser (Firefox, Chrome, IE) – Darstellung der lokalen Umgebung und Zugang auf das Testsystem

Personal

- Auszubildender – Umsetzung des Projektes
- Entwickler, Projektleiter – Umsetzung des Backends der Applikation und Review des Codes
- Geschäftsführer – Beratung, Hilfe bei Fragestellungen, Kommunikation mit dem Kunden

Sonstiges

- Büro
- Arbeitsplatz

A.4. Anforderungen

Anforderung	Beschreibung
Funktional	
Diensteüberwachung	Überwachung der Dienste, die auf den Servern laufen
Alarmierung	Benachrichtigung im Falle von Störungen
Protokollierung	Aufzeichnung von Zustandsänderungen und Störungen
Registrieren	neue Sensor registrieren
Nicht-Funktional	
Benutzerfreundlichkeit	Einfache Bedienbarkeit des Systems
Zuverlässigkeit	Hohe Verfügbarkeit und Genauigkeit des Überwachungssystems
Qualität	
Skalierbarkeit	Möglichkeit zur Erweiterung des Systems
Flexibilität	Anpassung an unterschiedliche Anforderungen des Kunden
Leistung	
Echtzeitüberwachung	Echtzeitüberwachung der Zustände
Ressourcenschonung	Geringer Ressourcenverbrauch auf den überwachten Servern

Tabelle 2: Anforderungen an das Überwachungssystem

Tätigkeit	Zeit in Stunden
Selbstständige Fortbildung	4h
Vorbereitung Projektdokumentation	4h
Zeitschrift	1h
Absprache mit der internen Entwicklungsabteilung für Schnittstellen	6h
Besprechung mit Auftraggeber über Frontend	1h
Lastenheft, Konzept Projektstruktur	4h
Recherche NodeJs Bibliotheken	1h
Implementation Backend	3h
Implementation Backend	4,5h
Besprechung mit Auftraggeber	1h
Implementation Frontend	2,5h
Implementation Backend	3h
Postman Tests	1,5h
Build-Skripte	0,5h
Implementation Frontend	3h
Implementation Frontend	4h
Implementation Backend	2h
Postman Tests	2h
Implementation Frontend	4h
Implementation Backend	2h
Postman Tests	2h
Implementation Backend	1h
Implementation Frontend	6h
Build-Skripte, Test	0,5h
Recherche Cert-Gen und Rate-Limiting Backend	0,5h
Projektdokumentation	4h
Backend Tests	2h
Frontend Tests	2h
White-Box Text	3h
Entwurf Diagramme und Darstellungen	3h
Projektdokumentation	2h

Tabelle 3: Detaillierte Zeitschrift

A.5. Middleware

Middleware bezeichnet eine Art von Anwendungssoftware, die als Vermittler zwischen verschiedenen Systemen oder Komponenten fungiert. Es handelt sich um eine Funktion, die Zugriff auf den Anfrage-Antwort-Zyklus der Anwendung hat. Middleware kann dabei helfen, den Ablauf von Anfragen und Antworten zu steuern, indem sie Daten von der Anfrage entgegennimmt und an die Anwendung weiterleitet, sowie die Antwort von der Anwendung entgegennimmt und an den Client zurücksendet.

Eine Middleware-Funktion enthält in der Regel das Anfrageobjekt, das Antwortobjekt und die Middleware-

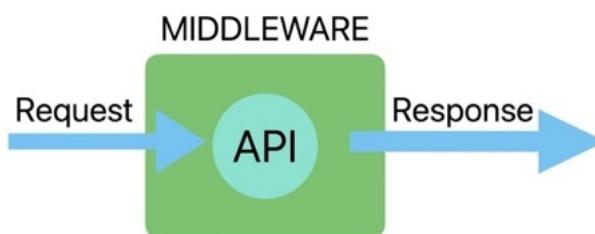


Abbildung 1: ExpressJS Middleware

Funktion selbst. Middleware kann auch die Antwort an den Server senden, bevor die Anfrage abgeschlossen ist. Die nächste Middleware-Funktion wird normalerweise als Variable mit dem Namen "next" dargestellt. Im Grunde ist Middleware eine Funktion, die nur über Routen angewendet werden kann. Middleware kann auch verwendet werden, um auf Anfragen und Antworten zuzugreifen und diese zu ändern.

Middleware kann mehrere Aufgaben erfüllen. Es kann beliebigen Code ausführen, Änderungen an den Anfrage-Antwort-Objekten vornehmen, den Anfrage-Antwort-Zyklus beenden oder die nächste Middleware-Funktion im Stapel aufrufen. Mit diesen Funktionen können wir Middleware modifizieren, um viele Aufgaben auszuführen, wie beispielsweise eine Website für bestimmte Länder zu sperren oder die Authentifizierung eines Benutzers zu überprüfen.

Um Middleware in einer Express-Anwendung zu erstellen, können wir eine separate Middleware-Datei erstellen und diese ausführen. Dazu müssen wir zunächst alle Pakete installieren, die für die Ausführung von Express-Middleware benötigt werden. Anschließend können wir die Middleware-Datei erstellen und ausführen, um die Funktionen der Middleware in unserer Anwendung zu nutzen.

Im Beispiel wird gezeigt, wie eine Middleware-Funktion in Express erstellt werden kann. Zunächst wird eine einfache Express-API erstellt und dann eine Middleware-Datei erstellt. In dieser Datei wird die Middleware-Funktion definiert, die aufgerufen wird, wenn eine Anfrage an die API gestellt wird. Die Middleware-Funktion ruft dann die nächste Middleware-Funktion auf und gibt die Antwort an den Client zurück.

Insgesamt bietet Middleware eine einfache Möglichkeit, um verschiedene Funktionen in eine Anwendung zu integrieren und diese zu verwalten. Durch die Verwendung von Middleware können Entwickler schnell und einfach Funktionen hinzufügen oder entfernen, um die Funktionalität ihrer Anwendungen zu verbessern.

A.6. Der Grund für die Verwendung von Session

In der Webentwicklung ist die Verwendung von Sessions ein wichtiger Aspekt bei der Entwicklung von Überwachungssystemen. Eine Session ermöglicht es, den Benutzer während seiner Interaktion mit dem System zu identifizieren und seine Aktionen zu verfolgen. Dies ist besonders wichtig bei der Überwa-

chung eines komplexen Systems, bei dem viele Komponenten beteiligt sind.

Durch die Verwendung von Sessions kann der Benutzer authentifiziert werden, um sicherzustellen, dass nur autorisierte Benutzer auf das System zugreifen können. Außerdem kann die Session genutzt werden, um die Berechtigungen des Benutzers zu überprüfen und sicherzustellen, dass er nur auf die Bereiche des Systems zugreifen kann, auf die er zugreifen sollte.

Eine Session ermöglicht auch die Verfolgung der Aktivitäten des Benutzers im System, was hilfreich sein kann, um ungewöhnliches Verhalten oder mögliche Sicherheitsbedrohungen zu erkennen. Durch die Aufzeichnung von Benutzeraktivitäten in der Session können Probleme schneller erkannt und behoben werden.

A.7. Der Grund für die Verwendung von Helmet.js

Helmet.js ist ein Middleware-Modul für Express, das verschiedene HTTP-Header für verbesserte Sicherheit konfiguriert. Es ist wichtig für die Webentwicklung, insbesondere für Systeme zur Überwachung, da es dazu beiträgt, Angriffe auf die Anwendung zu verhindern und die Sicherheit zu erhöhen. Einige der Funktionen, die Helmet.js bietet, sind:

- **XSS-Schutz:** Cross-Site-Scripting-Angriffe können durch Einfügen von Skripten in die Benutzeingabe durchgeführt werden. Helmet.js konfiguriert den X-XSS-Protection-Header, um solche Angriffe zu blockieren.
- **HTTP-Header-Steuerung:** HTTP-Header können so konfiguriert werden, dass sie Angriffe verhindern oder abmildern. Beispielsweise kann der X-Content-Type-Options-Header so eingestellt werden, dass der Browser dazu gezwungen wird, nur MIME-Typen zu akzeptieren, die der Server ausdrücklich angibt.
- **Clickjacking-Schutz:** Clickjacking-Angriffe können dazu führen, dass Benutzer auf unerwünschte Links klicken, ohne es zu merken. Helmet.js konfiguriert den X-Frame-Options-Header, um solche Angriffe zu verhindern.
- **Weitere Funktionen:** Weitere Funktionen von Helmet.js sind das Verhindern von MIME-Sniffing, das Verbergen von X-Powered-By-Headern und das Hinzufügen von Strict-Transport-Security-Headern.

A.8. Fehlercodes und Statuscode-Meldungen

In der Webentwicklung ist es von großer Bedeutung, standardisierte Fehlercodes und Statuscode-Meldungen zu verwenden, da sie dazu beitragen, Fehler schnell zu erkennen und zu beheben. Eine Tabelle, die diese Codes und Meldungen enthält (siehe Abbildung 2), ist eine nützliche Referenz, um Fehler zu identifizieren und zu beheben. Die Tabelle sollte alle relevanten Codes enthalten, die von der Anwendung verwendet werden, sowie eine kurze und klare Beschreibung der zugehörigen Meldungen. Es ist auch wichtig, dass die Codes und Meldungen präzise formuliert sind, um eine einfache Verständlichkeit zu gewährleisten.

1	code	Msg
2	:	-----
3	101	Anfrage nicht gültig
4	102	Nicht angemeldet
5	103	Berechtigungen fehlen
6	104	Benutzername oder E-Mail bereits registriert
7	105	E-Mail existiert nicht
8	106	Kennwort Mindestlänge ist 8 Zeichen
9	107	Benutzername hat ungültige Zeichen
10	108	Benutzername/E-Mail oder Passwort falsch
11	109	"UserID" nicht gesetzt
12	110	Benutzer nicht gefunden
13	111	"Sensor_ID" nicht gesetzt
14	112	Sensor nicht gefunden
15	113	Fehlende oder ungültige Sensorinformationen
16	114	Datei nicht gefunden
17	115	Datei nicht lesbar
18	116	Range-Header nicht gesetzt
19	117	Token nicht gefunden
20	201	Daten gesendet
21	202	Benutzer registriert
22	203	Benutzer eingelogt
23	204	Benutzer abgemeldet
24	205	Benutzer geändert
25	206	Benutzer gelöscht
26	207	Sensor hinzugefügt
27	208	Sensor geändert
28	209	Sensor gelöscht
29	210	Datei hochgeladen
30	211	E-Mail gesendet
31	212	Fehler beim Registrieren des Sensors
32	401	DB Error
33	402	Bycrypt Error
34	403	Mail Error
35	404	Unknown Error

Abbildung 2: Fehlercodes und Statuscode-Meldungen

A.9. Use-case Diagramm

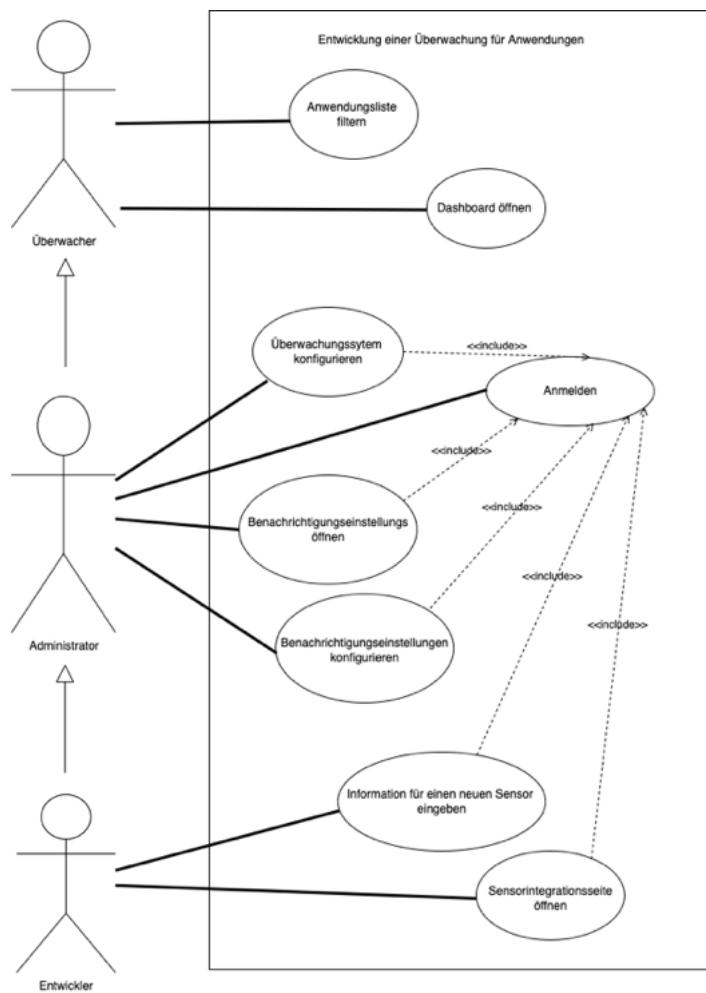


Abbildung 3: Use-case Diagramm

A.10. Komponenten Diagramm

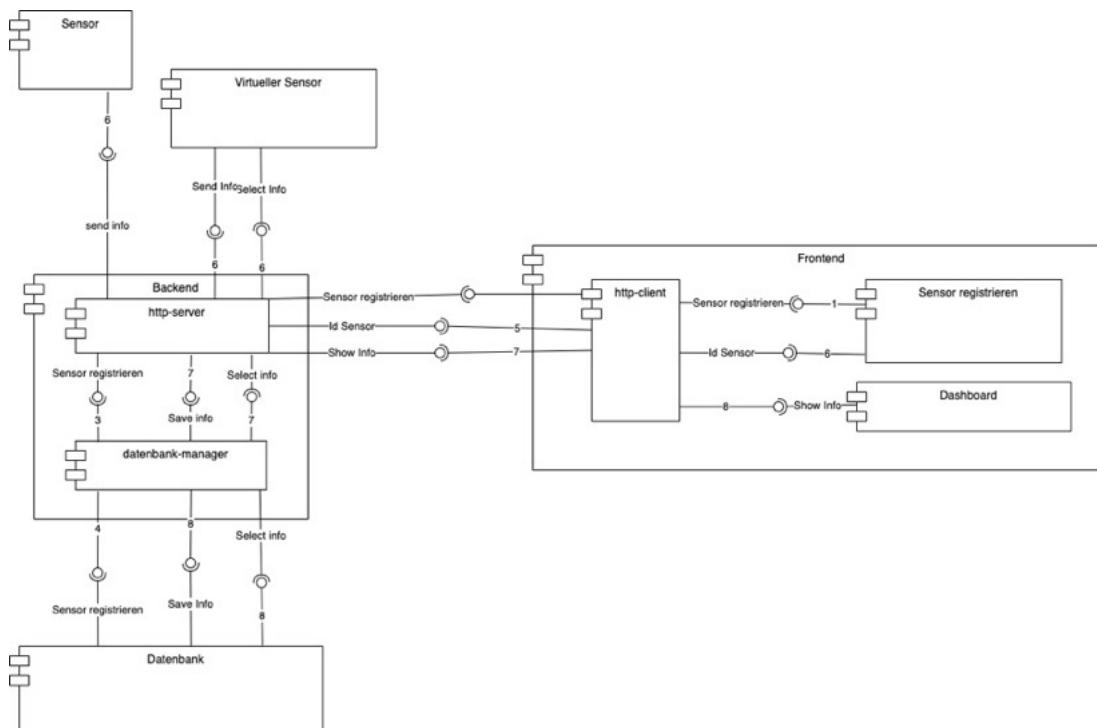


Abbildung 4: Komponenten Diagramm

B. Relationales Datenbankmodell

Für dieses Projekt wurde ein Entity-Relationship-Diagramm (ER-Diagramm) erstellt, um die Struktur der Datenbank und die Beziehungen zwischen den verschiedenen Entitäten zu modellieren. Das ER-Diagramm bildet die Grundlage für das Datenbankschema und hilft bei der Organisation der Daten, die im Rahmen des Überwachungssystems erfasst und verarbeitet werden.

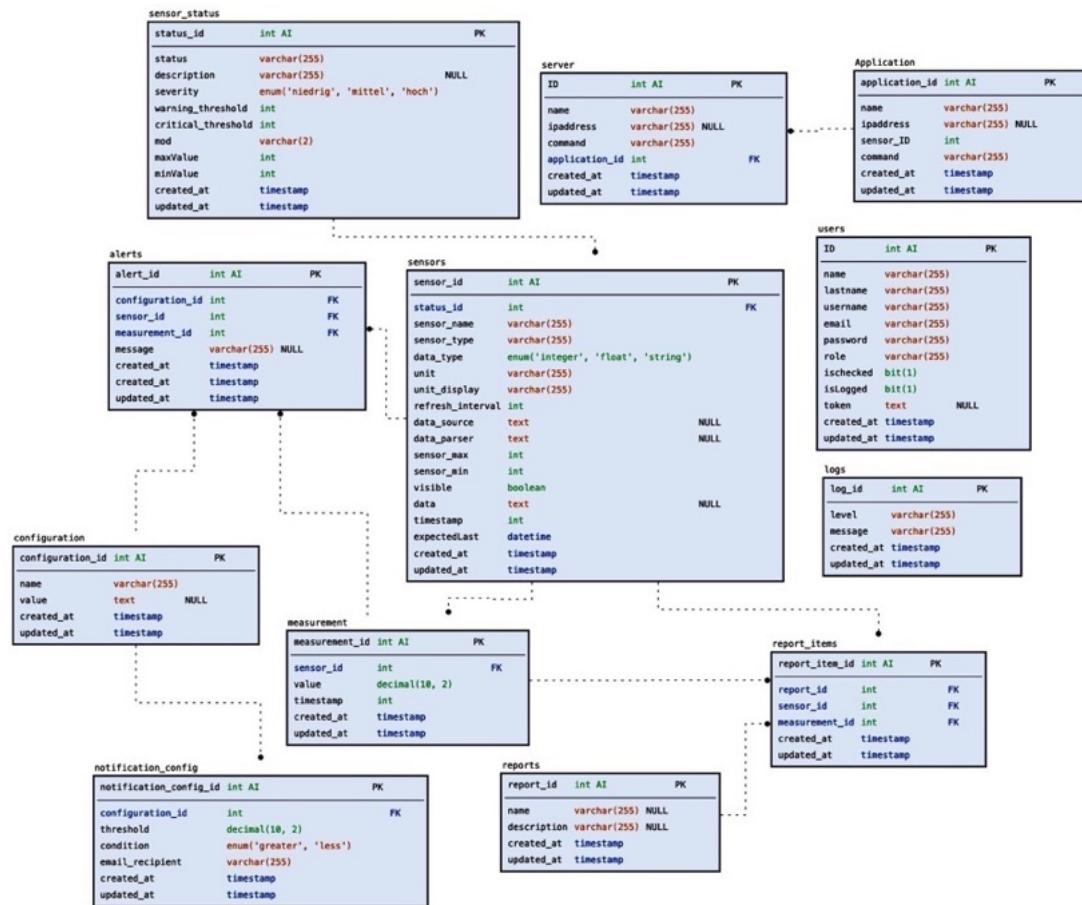


Abbildung 5: Relationales Datenbankmodell

B.1. Screenshot des Sprint-Boards in GitHub

In diesem Projekt wurde ein Sprint-Board in GitHub für die Planung und Verwaltung der Softwareentwicklung eingesetzt. Das Kanban Board ermöglichte eine effektive Organisation der verschiedenen Aufgaben und Phasen des Projekts, indem es den Fortschritt in Echtzeit darstellte. Als Einzelperson war das Kanban Board ein wertvolles Instrument zur Selbstorganisation und zum Nachverfolgen der Aufgaben, um sicherzustellen, dass alle notwendigen Schritte abgeschlossen und Ziele erreicht wurden.

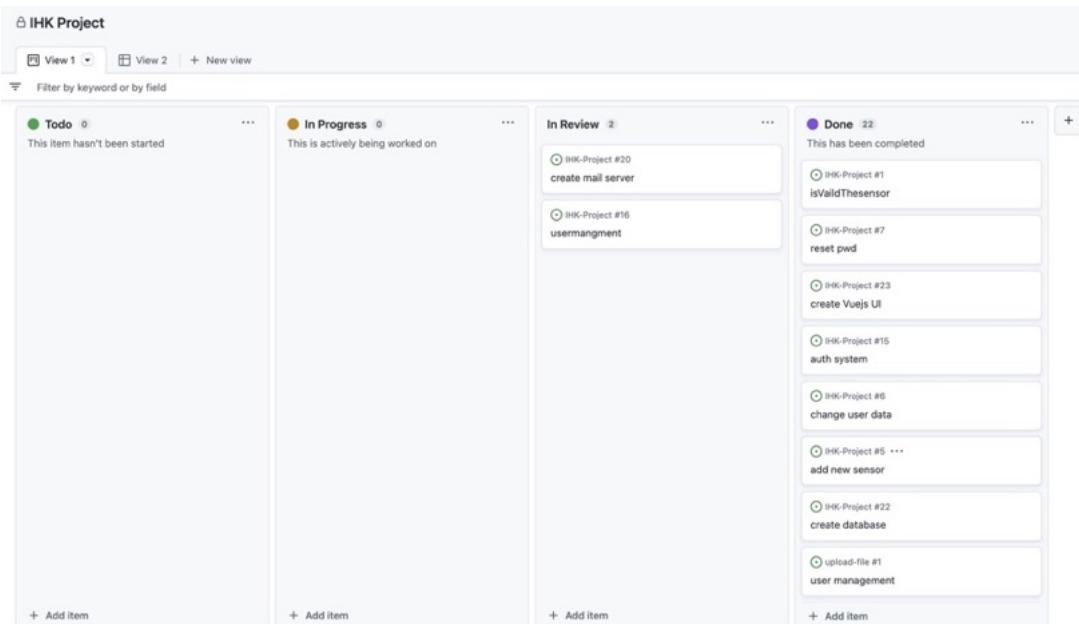


Abbildung 6: Screenshot des Sprint-Boards in GitHub

B.2. Verwendung von Gantt-Diagramm in GitHub für die Projektplanung

Seitdem GitHub die Unterstützung für Gantt-Diagramme in seinem Feature-Set integriert hat, ist die Planung und Verfolgung von Projekten einfacher geworden. Durch die Verwendung von Gantt-Diagrammen in GitHub können die verschiedenen Phasen des Projekts, Meilensteine und Aufgaben übersichtlich dargestellt und der Fortschritt sowie die Abhängigkeiten zwischen den einzelnen Aufgaben verfolgt werden. Dies trägt zur effektiven Planung, Zeitmanagement und zur rechtzeitigen Erreichung der Projektziele bei.

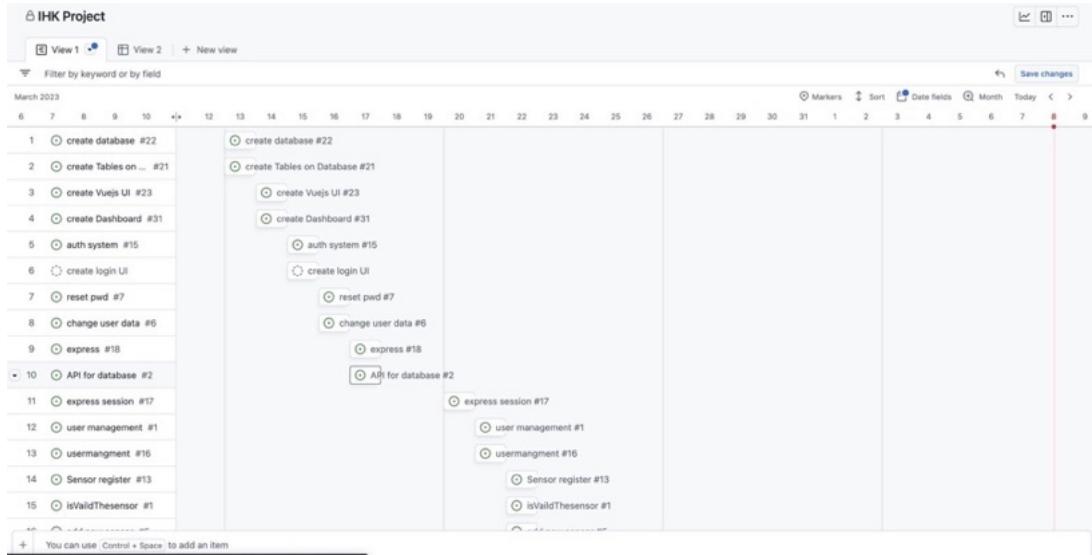


Abbildung 7: Verwendung von Gantt-Diagramm in GitHub für die Projektplanung

B.3. Screenshot des File-Trees Backend

Das Projekt wurde mit einer sorgfältig durchdachten Ordnerstruktur als Back-End entwickelt, um eine hohe Organisationsstruktur und Fehlerfreiheit zu gewährleisten. Durch die präzise Aufteilung der Dateien und Verzeichnisse ist es einfach, Probleme zu finden und zu beheben. Auf diese Weise können wir eine reibungslose Funktionsweise unseres Produkts sicherstellen und die Benutzererfahrung optimieren.

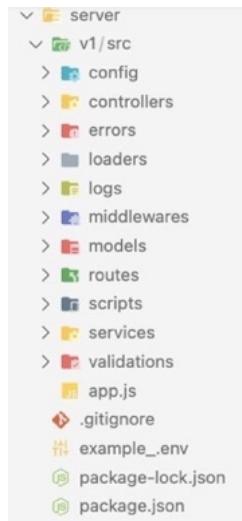


Abbildung 8: Screenshot des File-Trees Backend

B.4. Anmeldung eines neuen Sensors

Im folgenden Abschnitt wird ein JSON beschrieben, das einen Sensor mit dem Datentyp “integer” repräsentiert.

Mit diesem JSON kann der Sensor seine Daten und Einstellungen an das Backend senden, um beispielsweise Daten an das Frontend zu senden oder die Daten in der Datenbank zu speichern. Das Backend verwendet diese Informationen, um den Sensor zu verwalten und seine Konfiguration zu aktualisieren.

```
1  {
2      sensor_name: 'Anrufe pro Stunde',
3      sensor_type: 'process',
4      data_type: 'integer',
5      unit: 'anrufe',
6      unit_display: 'Anrufe/h',
7      sensor_max: 1000,
8      sensor_min: 0,
9      show: true,
10     saveData: true,
11     DataRetentionPeriodInMonths: 6,
12     expectedLast: 'Date',
13 }
```

Quellcode 1: Anmeldung eines neuen Sensors (Backend) JSON-Modell

```

1 // Function to check if all required fields are present
2 const validateSensorData = (sensorData) => {
3   const requiredFields = [
4     'sensor_name',
5     'sensor_type',
6     'data_type',
7     'unit',
8     'unit_display',
9     'sensor_max',
10    'sensor_min',
11    'show',
12  ];
13
14  requiredFields.forEach((field) => {
15    if (!sensorData.hasOwnProperty(field)) {
16      return false;
17    }
18  });
19  return true;
20};
21 // register a new sensor
22 app.post('/registerSensor', async (req, res) => {
23   const sensorData = req.body;
24   // Check that all required fields are present
25   if (!validateSensorData(sensorData)) {
26     res.status(400).json({
27       code: 113,
28       message: 'Fehlende oder ungültige Sensorinformationen',
29     });
30     return;
31   }
32   // Add the sensor
33   try {
34     const connection = await db.getConnection();
35
36     const sql = `
37       INSERT INTO sensors (sensor_name, sensor_type, data_type, unit, unit_display, sensor_max, sensor_min, \`show\`, created_at, updated_at)
38       VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
39     `;
40     const params = [
41       sensorData.sensor_name,
42       sensorData.sensor_type,
43       sensorData.data_type,
44       sensorData.unit,
45       sensorData.unit_display,
46       sensorData.sensor_max,
47       sensorData.sensor_min,
48       sensorData.show,
49       new Date(),
50       new Date(),
51     ];
52     const result = await connection.query(sql, params);
53     const sensorId = result.insertId;
54     await connection.end();
55     // Send back a response that the sensor was successfully registered and the sensor_id return
56     res.status(201).json({
57       message: 'Sensor erfolgreich registriert',
58       code: 207,
59       sensor_id: sensorId,
60     });
61   } catch (error) {
62     console.error('Fehler beim Hinzufügen des Sensors:', error);
63     res.status(500).json({
64       message: 'Fehler beim Registrieren des Sensors',
65       code: 212,
66     });
67   }
68 });

```

```
1 const validateSensorData = require('./validateSensorData');
2
3 describe('validateSensorData', () => {
4   it('should return true if all required fields are present', () => {
5     const sensorData = {
6       sensor_name: 'Test Sensor',
7       sensor_type: 'test',
8       data_type: 'integer',
9       unit: 'test',
10      unit_display: 'Test Units',
11      sensor_max: 100,
12      sensor_min: 0,
13      show: true,
14    };
15    expect(validateSensorData(sensorData)).toBe(true);
16  });
17
18  it('should return false if a required field is missing', () => {
19    const sensorData = {
20      sensor_name: 'Test Sensor',
21      sensor_type: 'test',
22      data_type: 'integer',
23      unit: 'test',
24      unit_display: 'Test Units',
25      sensor_max: 100,
26      sensor_min: 0,
27      // Missing "show" field
28    };
29    expect(validateSensorData(sensorData)).toBe(false);
30  });
31});
32
```

Quellcode 3: Anmeldung eines neuen Sensors (Backend) Unit-Tests

Sensor :Register

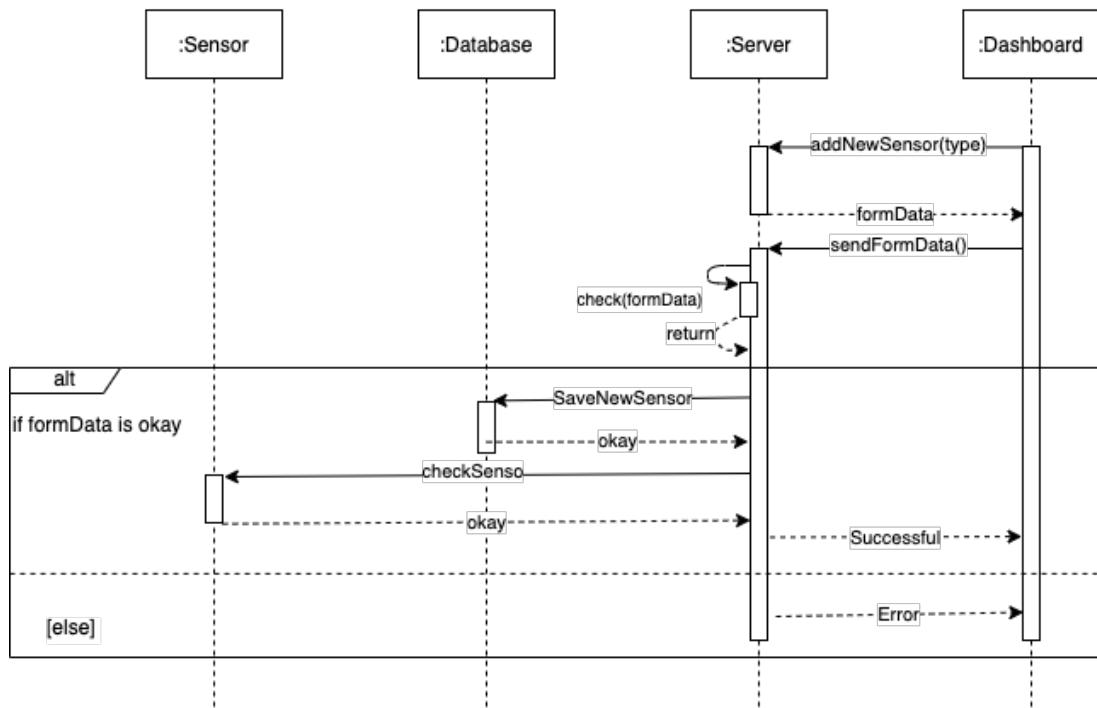


Abbildung 9: Anmeldung eines neuen Sensors Sequenzdiagramm

B.5. Sequenzdiagramm: Sensor schickt Daten

In diesem Sequenzdiagramm wird dargestellt, wie der Sensor Daten an den Server sendet und wie der Server die Daten in der Datenbank speichert. Außerdem wird gezeigt, wie das Dashboard die Daten aktualisiert:

- Der Sensor erfasst Daten und sendet sie an den Server.
- Der Server empfängt die Daten und verarbeitet sie.
- Der Server speichert die verarbeiteten Daten in der Datenbank.
- Das Dashboard fordert regelmäßig aktualisierte Daten von der Datenbank an.
- Der Server sendet die angeforderten Daten an das Dashboard.
- Das Dashboard aktualisiert die angezeigten Daten entsprechend den neuen Daten.

Durch diese Darstellung wird der Datenfluss von der Erfassung durch den Sensor bis zur Anzeige auf dem Dashboard verdeutlicht.

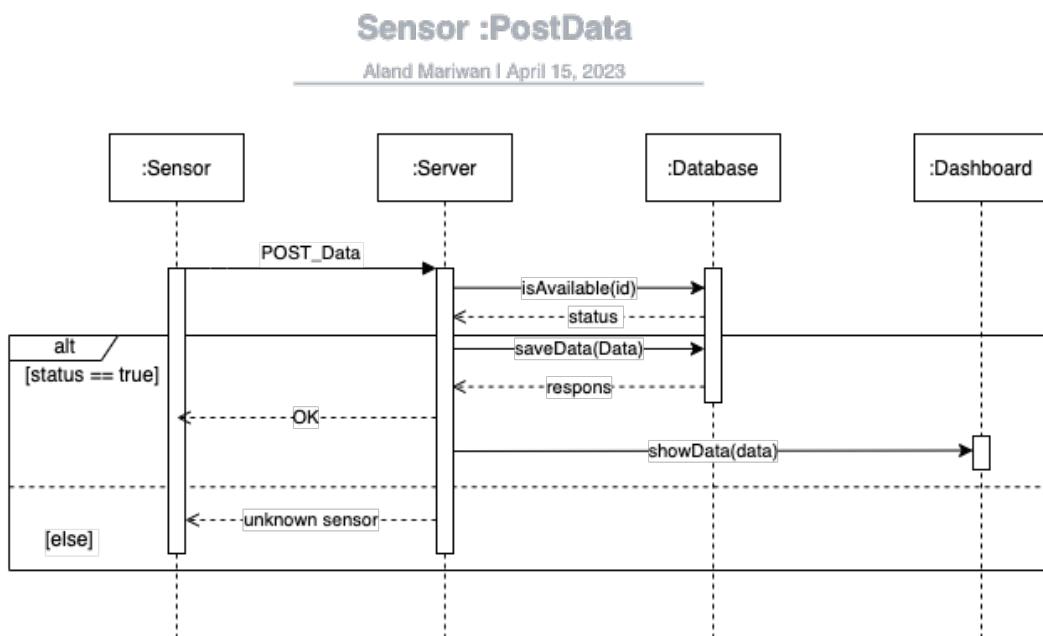


Abbildung 10: Sensor schickt Daten (Sequenzdiagramm)

```

1 router.get('/sensors/:id/data', async (req, res) => {
2   try {
3     const sensorId = req.params.id;
4     const sensor = await db.getSensorById(sensorId);
5     if (!sensor) {
6       res.status(404).json({ code: 112, msg: 'Sensor not found' });
7       return;
8     }
9     const data = await db.getLatestSensorData(sensorId);
10    if (!data) {
11      res.status(204).end();
12      return;
13    }
14    res.status(200).json(data);
15  } catch (error) {
16    console.error('Error getting sensor data:', error);
17    res.status(500).json({ code: 401, msg: 'DB Error' });
18  }
19 });

```

Quellcode 4: Sensor schickt Daten (Backend)

```

1 const request = require('supertest');
2 const app = require('../app');
3
4 describe('GET /sensordata', () => {
5   it('responds with JSON array', async () => {
6     const response = await request(app).get('/sensordata');
7     expect(response.status).toBe(200);
8     expect(response.body).toBeInstanceOf(Array);
9   });
10
11  it('responds with sensor data', async () => {
12    const response = await request(app).get('/sensordata');
13    expect(response.status).toBe(200);
14    expect(response.body[0]).toHaveProperty('sensor_id');
15    expect(response.body[0]).toHaveProperty('data');
16    expect(response.body[0]).toHaveProperty('timestamp');
17  });
18 });
19

```

Quellcode 5: Sensor schickt Daten (Backend) Unit-Tests

```
1 router.post('/register', function (req, res) {
2     let name = decrypt(req.body.name);
3     let lastname = decrypt(req.body.lastname);
4     let username = decrypt(req.body.username);
5     let email = decrypt(req.body.email);
6     let password = decrypt(req.body.password);
7
8     if (
9         name === false ||
10        lastname === false ||
11        username === false ||
12        email === false ||
13        password === false
14     ) {
15         res.status(400).send({
16             msg: 'Anfrage nicht gültig',
17             code: 101,
18         });
19         return;
20     }
21     username = username.toLowerCase();
22     email = email.toLowerCase();
23
24     if (!isEmail(email)) {
25         res.status(400).send({
26             msg: 'E-Mail existiert nicht',
27             code: 105,
28         });
29         return;
30     }
31
32     if (!checkUsername(username)) {
33         res.status(400).send({
34             msg: 'Benutzername hat ungültige Zeichen',
35             code: 107,
36         });
37         return;
38     }
39
40     if (password.length < 8) {
41         res.status(400).send({
42             msg: 'Kennwort Mindestlänge ist 8 Zeichen',
43             code: 106,
44         });
45         return;
46     }
47 }
```

Quellcode 6: Neue Benutzer hinzufügen (Backend Codestück Teil 1)

```

1   db.query(
2     'SELECT * FROM users WHERE username = ? OR email = ?',
3     [username, email],
4     function (err, result) {
5       if (err) {
6         console.error(err);
7         res.status(500).send({
8           msg: 'DB Error',
9           code: 401,
10          err: err,
11        });
12        return;
13      }
14
15      if (result.length != 0) {
16        res.status(500).send({
17          msg: 'Benutzername oder E-Mail bereits registriert',
18          code: 104,
19        });
20        return;
21      }
22
23      bcrypt.hash(password, saltRounds, function (err2, hash) {
24        if (err2) {
25          console.error(err2);
26          res.status(500).send({
27            msg: 'Bcrypt Error',
28            code: 402,
29            err: err2,
30          });
31          return;
32        }
33
34        db.query(
35          'INSERT INTO users (name, lastname, username, email, password ) VALUE (?,?,?,?,?)',
36          [name, lastname, username, email, hash],
37          function (error, response) {
38            if (error) {
39              console.error(error);
40              res.status(500).send({
41                msg: 'DB Error',
42                code: 401,
43                err: error,
44              });
45              return;
46            }
47
48            res.status(200).send({
49              msg: 'Benutzer registriert',
50              code: 202,
51            });
52          },
53        );
54      });
55    },
56  );
57 });

```

Quellcode 7: Neue Benutzer hinzufügen (Backend Codestück Teil 2)

```
1 describe('POST /auth/register', () => {
2   test('should return 400 if request is invalid', async () => {
3     var nameHash = encrypt('invalid');
4     var lastnameHash = encrypt('request');
5     var usernameHash = encrypt('invalid');
6     var emailHash = encrypt('invalid');
7     var passwordHash = bcrypt.hashSync('short', saltRounds); // hash created
8     const response = await request(app)
9       .post('/auth/register')
10      .send({
11        name: nameHash,
12        lastname: lastnameHash,
13        username: usernameHash,
14        email: emailHash,
15        password: passwordHash,
16      })
17      .set('Content-Type', 'application/json');
18
19     expect(response.status).toBe(400);
20     expect(response.body.msg).toBe('Anfrage nicht gültig');
21     expect(response.body.code).toBe(101);
22   });
23
24   test('should return 400 if email is invalid', async () => {
25     var nameHash = encrypt('Aland');
26     var lastnameHash = encrypt('Mariwan');
27     var usernameHash = encrypt('amariwan');
28     var emailHash = encrypt('invalid');
29     var passwordHash = bcrypt.hashSync('short1234568', saltRounds); // hash created
30     const response = await request(app)
31       .post('/auth/register')
32      .send({
33        name: nameHash,
34        lastname: lastnameHash,
35        username: usernameHash,
36        email: emailHash,
37        password: passwordHash,
38      })
39      .set('Content-Type', 'application/json');
40
41     expect(response.status).toBe(400);
42     expect(response.body.msg).toBe('E-Mail existiert nicht');
43     expect(response.body.code).toBe(105);
44   });
45 }
```

Quellcode 8: Neue Benutzer hinzufügen (Backend Codestück Teil 1)

```
1  test('should return 400 if username has invalid characters', async () => {
2    var nameHash = encrypt('Aland');
3    var lastnameHash = encrypt('Mariwan');
4    var usernameHash = encrypt('amariwan.99');
5    var emailHash = encrypt('dev@aland-mariwan.de');
6    var passwordHash = bcrypt.hashSync('short1234568', saltRounds); // hash created
7    const response = await request(app)
8      .post('/auth/register')
9      .send({
10        name: nameHash,
11        lastname: lastnameHash,
12        username: usernameHash,
13        email: emailHash,
14        password: passwordHash,
15      })
16      .set('Content-Type', 'application/json');
17
18    expect(response.status).toBe(400);
19    expect(response.body.msg).toBe('Benutzername hat ungültige Zeichen');
20    expect(response.body.code).toBe(107);
21  });
22
23  test('should return 400 if password is too short', async () => {
24    var nameHash = encrypt('Aland');
25    var lastnameHash = encrypt('Mariwan');
26    var usernameHash = encrypt('amariwan.99');
27    var emailHash = encrypt('dev@aland-mariwan.de');
28    var passwordHash = bcrypt.hashSync('short', saltRounds); // hash created
29    const response = await request(app)
30      .post('/auth/register')
31      .send({
32        name: nameHash,
33        lastname: lastnameHash,
34        username: usernameHash,
35        email: emailHash,
36        password: passwordHash,
37      })
38      .set('Content-Type', 'application/json');
39
40    expect(response.status).toBe(400);
41    expect(response.body.msg).toBe('Kennwort Mindestlänge ist 8 Zeichen');
42    expect(response.body.code).toBe(106);
43  });
44});
```

Quellcode 9: Neue Benutzer hinzufügen (Backend Codestück Teil 2)

```

1 router.post('/forgetpassword', async (req, res) => {
2   const email = decrypt(req.body.email);
3   if (email === false) {
4     res.status(400).send({
5       msg: 'Anfrage nicht gültig',
6       code: 101,
7     });
8     return;
9   }
10
11  const user = await checkEmailOnDB(email);
12  if (user.result === 0) {
13    // FAKE RESPONSE
14    res.status(200).send({
15      msg: 'E-Mail gesendet',
16      code: 211,
17    });
18    return;
19  }
20  if (user.result === 1) {
21    // DB Error
22    console.error(user.err);
23    res.status(500).send({
24      msg: 'DB Error',
25      code: 401,
26      err: user.err,
27    });
28    return;
29  }
30  // Generate a token and send it to the user's email
31  const token = crypto.randomBytes(1000).toString('hex');
32  const transporter = nodemailer.createTransport(config.mailAuth[0]);
33  const html = fs.readFileSync('htmlMail/forgotPassword.html').toString();
34  const mailOptions = {
35    from: {
36      name: 'GMS Passwort vergessen',
37      address: config.mailAuth[0].auth.user,
38    },
39    to: email,
40    subject: 'GMS Passwort Änderung',
41    html: html
42      .replace('${__NAME__}', user.data.lastname)
43      .replace('${__HOST__}', config.frontend_host)
44      .replace('${__TOKEN__}', token),
45  };

```

```
1  transporter.sendMail(mailOptions, async (err, info) => {
2    if (err) {
3      console.error(err);
4      res.status(400).send({
5        msg: `Mail Error`,
6        code: 403,
7        err: err,
8      });
9      return;
10   }
11   const isSetUserTokenOnDB = await setUserTokenOnDB(token, email);
12   if (isSetUserTokenOnDB.result === 0) {
13     res.status(400).send({
14       msg: 'Benutzer nicht gefunden',
15       code: 110,
16     });
17     return;
18   }
19   if (isSetUserTokenOnDB.result === 1) {
20     //DB Error
21     console.error(user.err);
22     res.status(500).send({
23       msg: 'DB Error',
24       code: 401,
25       err: user.err,
26     });
27     return;
28   }
29   res.status(200).send({
30     msg: 'E-Mail gesendet',
31     code: 211,
32   });
33 });
34});
```

Quellcode 11: Passwort zurücksetzen (Backend)

B.6. Dashboard-Seite

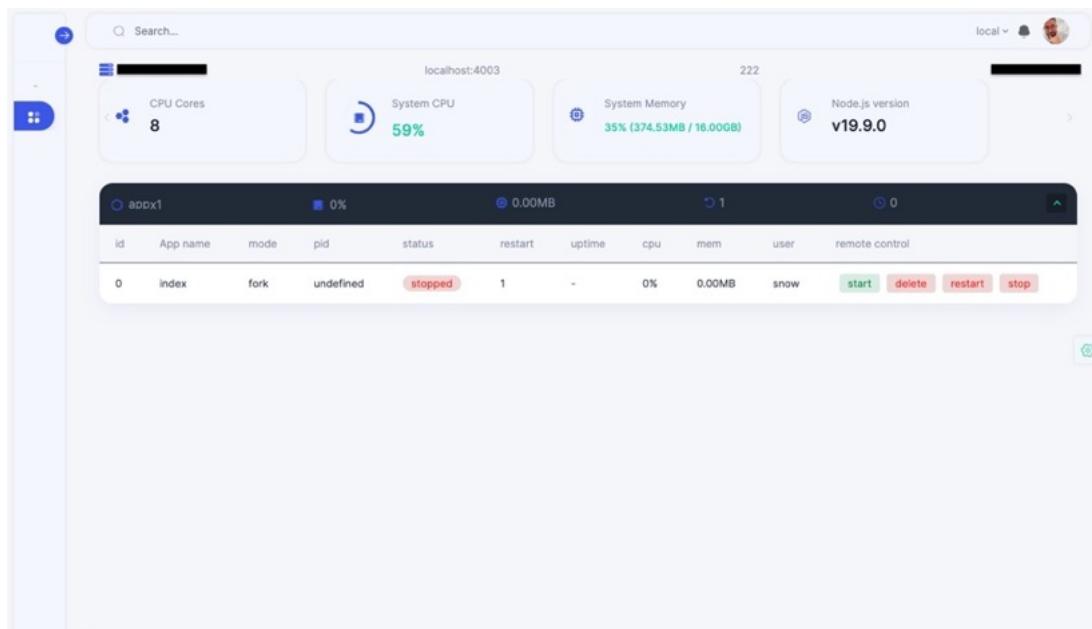


Abbildung 11: Dashboard-Seite (Frontend)

```
1 import { createRouter, createWebHistory } from 'vue-router'
2
3
4 // Auth Default Routes
5 const authChildRoutes = (prefix) => [
6   {
7     path: 'login',
8     name: prefix + '.login',
9     meta: { auth: false, name: 'Login' },
10    component: () => import('@/views/auth/default/SignIn.vue')
11  },
12  {
13    path: 'register',
14    name: prefix + '.register',
15    meta: { auth: false, name: 'Register' },
16    component: () => import('@/views/auth/default/SignUp.vue')
17  },
18  {
19    path: 'reset-password',
20    name: prefix + '.reset-password',
21    meta: { auth: false, name: 'Reset Password' },
22    component: () => import('@/views/auth/default/ResetPassword.vue')
23  },
24  {
25    path: 'varify-email',
26    name: prefix + '.varify-email',
27    meta: { auth: false, name: 'Varify Email' },
28    component: () => import('@/views/auth/default/VarifyEmail.vue')
29  },
30  {
31    path: 'lock-screen',
32    name: prefix + '.lock-screen',
33    meta: { auth: false, name: 'Lock Screen' },
34    component: () => import('@/views/auth/default/LockScreen.vue')
35  }
36 ]
```

Quellcode 12: VueJs router (Frontend Codestück Teil 1)

```
1 // Dashboard routes
2 const dashboardRoutes = (prefix) => [
3   {
4     path: '',
5     name: prefix + '.dashboard',
6     meta: { auth: true, name: 'Home', isBanner: false },
7     component: () => import('@/views/dashboards/IndexPage.vue')
8   }
9 ]
10 // Default routes
11 const defaultChildRoutes = (prefix) => [
12   {
13     path: '',
14     name: prefix + '.dashboard',
15     meta: { auth: true, name: 'Home', isBanner: false },
16     component: () => import('@/views/dashboards/IndexPage.vue')
17   },
18   // Spacial Pages
19   {
20     path: '/server',
21     name: prefix + '.server',
22     meta: { auth: true, name: 'Server', isBanner: false },
23     component: () => import('@/views/spacial-pages/serverPage.vue')
24   },
25   {
26     path: '/sensor',
27     name: prefix + '.sensor',
28     meta: { auth: true, name: 'Sensor', isBanner: false },
29     component: () => import('@/views/spacial-pages/sensorPage.vue')
30   },
31   {
32     path: '/setting',
33     name: prefix + '.setting',
34     meta: { auth: true, name: 'Setting', isBanner: false },
35     component: () => import('@/views/spacial-pages/settingPage.vue')
36   },
37   {
38     path: '/timeline',
39     name: prefix + '.timeline',
40     meta: { auth: true, name: 'Timeline', isBanner: false },
41     component: () => import('@/views/spacial-pages/TimelinePage.vue')
42   }
43 ]
```

Quellcode 13: VueJs router (Frontend Codestück Teil 2)

```

1 const usersPages = (prefix) => [
2   // Users Pages
3   {
4     path: '/sensor-list',
5     name: prefix + '.sensor-list',
6     meta: { auth: true, name: 'Sensor List', isBanner: false },
7     component: () => import('@/views/sensor/ListPage.vue')
8   },
9   {
10    path: '/sensor-add',
11    name: prefix + '.sensor-add',
12    meta: { auth: true, name: 'Sensor Add', isBanner: false },
13    component: () => import('@/views/sensor/AddPage.vue')
14  },
15  {
16    path: '/user-profile',
17    name: prefix + '.user-profile',
18    meta: { auth: true, name: 'User Add', isBanner: false },
19    component: () => import('@/views/user/ProfilePage.vue')
20  },
21  {
22    path: '/privacy-setting',
23    name: prefix + '.user-privacy-setting',
24    meta: { auth: true, name: 'Privacy Setting', isBanner: false },
25    component: () => import('@/views/user/PrivacySetting.vue')
26  },
27  {
28    path: '/charts',
29    name: prefix + '.charts',
30    meta: { auth: true, name: 'Charts', isBanner: false },
31    component: () => import('@/views/charts/charts.vue')
32  },
33  {
34    path: '/validation',
35    name: prefix + '.validation',
36    meta: { auth: true, name: 'Validation', isBanner: false },
37    component: () => import('@/views/forms/ValidationPage.vue')
38  },
39  // Table Pages
40  {
41    path: '/table',
42    name: prefix + '.table',
43    meta: { auth: true, name: 'Table', isBanner: false },
44    component: () => import('@/views/tables/Table.vue')
45  },
46  // Admin Pages
47  {
48    path: '/admin-permissions',
49    name: prefix + '.admin-permissions',
50    meta: { auth: true, name: 'Admin Permissions', isBanner: false },
51    component: () => import('@/views/admin/AdminPage.vue')
52  }
53 ]
54 ]

```

Quellcode 14: VueJs router (Frontend Codestück Teil 3)

```

1 import Footer from './views/Footer.vue';
2 import { useToast } from 'vue-toastification';
3 export default {
4   components: { Footer },
5   data() {
6     return {
7       sensorsInfo: {}, // store all sensors of main pointe here
8       mainPointInfo: {},
9       addSensorForm: {
10         // to add a new sensor to this main pointer
11         type: 'Pick Sensor',
12         locationX: null,
13         locationY: null,
14         reportInterval: null,
15       },
16     };
17   },
18   created() {
19     this.getMainPointInformation();
20     this.getSensorFromMainPoint();
21   },
22   methods: {
23     async getMainPointInformation() {
24       var path = window.location.pathname.split('/');
25       await this.$appAxios({
26         url: '/mainpoints/' + path[2],
27         method: 'GET',
28       }).then((response) => {
29         // console.log(response.data);
30         this.mainPointInfo = { ...response };
31       });
32     },
33     async getSensorFromMainPoint() {
34       var path = window.location.pathname.split('/');
35       await this.$appAxios({
36         url: '/sensors/all/' + path[2],
37         method: 'GET',
38       })
39         .then((response) => {
40           //   console.log(response);
41           this.sensorsInfo = { ...response };
42         })
43         .catch((error) => {
44           // do nothing
45         });
46       //   console.log(this.sensorsInfo);
47     },
48     async deleteSensor(id) {
49       await this.$appAxios({
50         url: `/sensors/${id}`,
51         method: 'DELETE',
52       }).then((deletedItem) => {
53         // this.getMainPoints();
54         this.sensorsInfo.data = this.sensorsInfo.data.filter((item) => item._id !== id);
55         if (this.sensorsInfo.data.length === 0) {
56           this.$router.go(); // refresh page and show code-mock-up
57         }
58         const toast = useToast();
59         toast.success('Sensor has been deleted!');
60         return { toast };
61       });
62     },
63   },

```

Quellcode 15: VueJs AllSensors (Frontend Codestück Teil 1)

```

1   1
2   2   async createSensor() {
3   3     if (
4   4       this.addSensorForm.locationX = null ||
5   5       this.addSensorForm.locationY = null ||
6   6       this.addSensorForm.reportInterval = null
7   7     ) {
8   8       const toast = useToast();
9   9       toast.error('Please fill in all fields !');
10 10      return { toast };
11 11    }
12 12    if (this.addSensorForm.type = 'Pick Sensor') {
13 13      const toast = useToast();
14 14      toast.error('Please choose a sensor type ! ');
15 15      return { toast };
16 16    }
17
18    const mainPointRadius = this.mainPointInfo.data[0].radius;
19    const mainPointLocationX = this.mainPointInfo.data[0].locationX;
20    const mainPointLocationY = this.mainPointInfo.data[0].locationY;
21    const locationX = this.addSensorForm.locationX;
22    const locationY = this.addSensorForm.locationY;
23    const distance = Math.sqrt(
24      Math.pow(mainPointLocationX - locationX, 2) + Math.pow(mainPointLocationY - locationY, 2)
25    );
26    console.log(distance);
27    if (distance > mainPointRadius) {
28      const toast = useToast();
29      toast.error('Sensor coordinate values are outside the radius of the main point !');
30      return { toast };
31    }
32    var path = window.location.pathname.split('/');
33    await this.Sappaxios({
34      url: '/sensors',
35      method: 'POST',
36      data: {
37        type: this.addSensorForm.type,
38        locationX: this.addSensorForm.locationX,
39        locationY: this.addSensorForm.locationY,
40        reportInterval: this.addSensorForm.reportInterval * 1000,
41        mainPoint_id: path[2],
42      },
43    }).then((sensors) => {
44      // console.log(sensors);
45      this.getSensorFromMainPoint();
46      const toast = useToast();
47      toast.success('Created sensor successfully !');
48      return { toast };
49    });
50  },
51  ),
52  );

```

Quellcode 16: VueJs AllSensors (Frontend Codestück Teil 2)