Abschlussprüfung Winter 2025

Fachinformatiker:in für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

# Auswertungsansicht für Gerätenachrichten im ABUS Cloud-Portal

**Erweiterung des bestehenden ABUS Cloud-Portals um eine Auswertungsansicht für Gerätenachrichten, damit der technische Support Fehler in unterstützten Zutritts- und Alarmsystemen schneller erkennen kann.**


**Prüfungsbewerber:in**

Ilia Bozhkov

Brendelstraße 6

90453 Nürnberg

**Ausbildungsbetrieb:**

conplement AG

Südwestpark 92

90449 Nürnberg


**Abgabedatum:**

15. Dezember 2025

# Inhaltsverzeichnis

## Glossar

**Endnutzer** Die Endnutzer können sowohl Privat- als auch Gewerbekunden sein. Sie sind Kunden von ABUS SC. Privatkunden dürfen ein Gerät selbst installieren und verwalten. Gewerbekunden dagegen müssen sich aus Versicherungsschutzgründen an einen qualifizierten Facherrichter wenden.

**Facherrichter** Ein Facherrichter ist ein Händler oder Vertriebspartner, der die Produkte von ABUS SC an Endkunden verkauft. Sie sind Kunden von ABUS SC.
Ein Facherrichter installiert und ggf. wartet Geräte von ABUS SC bei einem Endkunden, z.B. einem Lebensmittelladen. Bei Problemen mit einem Gerät sind die Facherrichter der erste Ansprechpartner. Sie können alle Gerätedetails im Cloud-Portal sehen und das Gerät aus der Ferne warten. Falls sie nicht weiterhelfen können, erreicht die Anfrage den TS.

**Technischer Support** Der Technischer Support (TS) ist eine Mitarbeiterrolle bei ABUS SC, die Kunden bei technischen Problemen mit ihren Geräten unterstützt. Ein TS hat Zugriff auf das Portal und kann dort Geräte-Details einsehen, Fernwartungsaktionen durchführen etc. Dafür benötigt er aber von dem jeweiligen Facherrichter oder Endnutzer eine Freigabe.

**Produktmanager** Der Produktmanager (PM) ist eine Mitarbeiterrolle bei ABUS SC, die für eine Produktlinie (z.B. Secoris) verantwortlich ist. Ein PM kann ungenutzte Geräte im Portal zurücksetzen/löschen und die Firmware-Versionen verwalten. Außerdem ist er der nächste Ansprechpartner für den TS bei Problemen, die dieser nicht lösen kann.

**MongoDB** MongoDB ist eine dokumentenorientierte NoSQL-Datenbank. Anstatt Daten in Tabellenspalten zu speichern, wie es bei relationalen Datenbanken der Fall ist, speichert MongoDB Daten in flexiblen BSON *(Binary JSON)* Dokumenten, die in Collections organisiert sind [Monb, Mona].

**Collection** Im Kontext der MongoDB ist eine Collection ein Container für Dokumente mit (meist) ähnlicher Struktur, ähnlich wie eine Tabelle in einer relationalen Datenbank [Mona].

**Responsive Web Design** Laut MDN[MDN] ist Responsive Web Design (RWD) *"ein Ansatz im Webdesign, um sicherzustellen, dass Webseiten auf allen Bildschirmgrößen und -auflösungen gut dargestellt werden und gleichzeitig eine gute Benutzerfreundlichkeit gewährleisten"*.

**Routing** Routing im Kontext von *Single-Page Applications* (SPAs) bezieht sich auf den Mechanismus, der es ermöglicht, den Seiteninhalt anhand der URL dynamisch zu ändern, ohne die gesamte Seite neu zu laden. Wie das in Angular funktioniert, wird in [Goo] beschrieben.

**Repository** Das Repository-Muster ist ein Entwurfsmuster im *Domain-Driven Design* (DDD), das eine Abstraktionsschicht zwischen der Domänenschicht und der Datenzugriffsschicht bereitstellt. Es kapselt die Logik zum Abrufen, Speichern und Verwalten von Daten. Repositories werden meist als Interfaces in der Domänenschicht definiert und in der Datenzugriffsschicht implementiert [Micb].

**Controller** Ein Controller ist im Kontext von ASP.NET Web API eine Klasse, die API-Endpunkte definiert und HTTP-Anfragen verarbeitet. Eine Controller-Methode entspricht einem Endpunkt und wird durch Attribute wie `[HttpGet]`, `[HttpPost]` usw. gekennzeichnet [Mica].

## Abkürzungsverzeichnis

**ABUS SC** ABUS Security Center GmbH & Co. KG.

**DI** Dependency Injection.

**PM** Produktmanager.

**Portal** ABUS Cloud-Portal.

**TS** Technischer Support.

# 1 Einleitung

## 1.1 Projektumfeld

Die conplement AG ist ein mittelständischer IT-Dienstleister mit ca. 130 Mitarbeitern, zwei Standorten in Deutschland (Nürnberg und Dortmund) und fast 20 Jahren Markterfahrung. Sie bietet individuelle Lösungen sowie Beratung in Bereichen wie Web, Embedded und Mobile.

Dazu gehört das ABUS Cloud-Portal (Portal) für den Kunden ABUS Security Center GmbH & Co. KG (ABUS SC). ABUS SC ist eine Tochtergesellschaft der ABUS KG mit Sitz in Augsburg, die seit 1999 Sicherheitssysteme für kleine und mittlere Unternehmen sowie Privatpersonen anbietet. Das Produktportfolio umfasst unter anderem Alarmanlagen, Videoüberwachungs- und Zutrittskontrollsystemen.

Das ABUS Cloud-Portal dient dazu, solche Geräte zu verwalten und zu warten. Dort kann man den Status eines Geräts und ggf. dessen Komponenten ansehen, Gerätenachrichten abonnieren und Logs einsehen sowie ein Gerät aus der Ferne konfigurieren. Es gibt vier Rollen, die das Portal in unterschiedlichem Umfang und für unterschiedliche Zwecke nutzen können: Endnutzer, Facherrichter, Technischer Support und Produktmanager. Jede Rolle ist im Glossar detailliert beschrieben.

Das conplement-Entwicklungsteam vom Portal besteht aus einer Projektmanagerin, einem Softwarearchitekten, zwei Fullstack-Entwicklern und einem Auszubildenden (mir), ebenfalls als Fullstack-Entwickler tätig. Das Team arbeitet eng mit dem Product Owner von ABUS SC zusammen und setzt agile Methoden wie Kanban ein, um die Entwicklung sowie den Betrieb effizient zu gestalten.

## 1.2 Projektziel

Das Ziel des Projekts ist die Implementierung eines neuen Features im Portal, das dem TS ermöglicht, die Häufigkeit und Art von Gerätenachrichten für ein bestimmtes Gerät über einen bestimmten Zeitraum zu analysieren. Dies soll auch ohne Freigabe durch den Facherrichter oder Endnutzer möglich sein. Ein PM soll ebenfalls Zugriff auf diese Funktionalität haben, aber die Hauptzielgruppe ist der TS.

Ein Auszug aus den Anforderungen des Kunden beschreibt das Ziel wie folgt (im Original auf Englisch):

> *As role technical support or pm, I want to see diagrams and overviews of system messages on the new "System Analysis" page so that I can identify possible issues and draw conclusions about potential errors.*
>
> Acceptance criteria:
>
> - The page displays a graph/chart visualizing system messages over time (for the exact type of chart please make an appropriate proposal)
> - The system messages can be filtered by time range, allowing me to view data from the past (e.g. last 24h, last 7 days, . . . )
> - The diagram updates when a filter is applied
> - The default filter is last 7 days

Außerdem wurde ein Mockup von dem UI-Designer von ABUS SC erstellt, das als visuelle Referenz für die Frontend-Implementierung dient. Die Anforderungen sind bewusst etwas offen formuliert, um dem Entwicklungsteam die Freiheit zu geben, die beste technische Lösung zu finden.

## 1.3 Projektbegründung

Momentan kann ein TS nur bei Geräten der Produktlinie *Secoris*[1] die Gerätenachrichten ansehen. Dazu muss er jedoch aus Datenschutzgründen eine Freigabe vom Facherrichter oder Endnutzer einholen, was den Support-Prozess verlangsamt. Außerdem hat die Secoris-Log-Tabelle keine grafische Darstellung der Nachrichtenhäufigkeit, was die Analyse und Erkennung von Mustern erschwert.

[1] Für andere Gerätetypen ist die Anzeige von Logs im Portal erst für die Zukunft geplant.

Das neue Feature soll diese Probleme lösen und die Effizienz des technischen Supports verbessern. Die Einsicht beschränkt sich auf die Daten, die für die Störungsbehebung notwendig sind. Somit wird der Datenschutz weiterhin gewährleistet. Durch die schnellere Problemlösung können sowohl monetäre als auch nicht-monetäre Vorteile für ABUS SC entstehen, wie in der 2.1 beschrieben.

### 1.4 Projektschnittstellen

Die technischen Schnittstellen des Projekts sind wie folgt:

- Die Daten für die Analyse stammen aus der bestehenden MongoDB-Datenbank. Sie müssen ausgelesen und aggregiert werden.
- Dies wird im bestehenden Backend passieren, das in C# mit ASP.NET implementiert ist.
- Die Auswertungsansicht muss in das bestehende Frontend vom Portal integriert werden.
- Für die DevOps-Prozesse (Build, Tests, Deployment) werden bestehende Pipelines und Tools in *Azure DevOps* verwendet.

Die personellen Schnittstellen des Projekts sind wie folgt:

- Abstimmung und Code-Reviews mit den beiden Fullstack-Entwicklern im Team.
- Bei Fragen oder Verbesserungsvorschlägen zu den Anforderungen und dem Design wird der Product Owner von ABUS SC kontaktiert. Die Ergebnisse des Projekts werden ebenfalls ihm präsentiert.
- Für das UI-Design wird der UI-Designer von ABUS SC einbezogen.
- Die Nutzer des neuen Features sind TS und PM von ABUS SC.

### 1.5 Projektabgrenzung

Die folgenden Punkte sind nicht Teil des Projekts:

- Datenbank-Änderungen: Sowohl die Datenbank-Struktur als auch die Daten selbst sind bereits vorhanden und müssen nicht geändert werden. Sie werden nur gelesen und ausgewertet.
- Das Portal: Das Projekt umfasst nur die Implementierung des neuen Features in die **bestehende** Anwendung.
- Responsive Web Design: Die bestehende Anwendung ist nicht für mobile Geräte optimiert. Daher wird das neue Feature ebenfalls nur für die Desktop-Ansicht implementiert.
- Diagramme: Die Darstellung der Daten in Diagrammen wird mit einer externen Bibliothek umgesetzt

### 1.6 Projektphasen

Das Projekt wird nach dem *Wasserfallmodell* durchgeführt, da die Anforderungen zu Beginn des Projekts klar definiert sind und sich voraussichtlich nicht ändern werden. Die einzelnen Phasen des Projekts sind wie folgt:

- Analysephase
- Konzeptionsphase
- Implementierungsphase
- Abschlussphase

In der folgenden Tabelle ist eine vollständige Zeitplanung mit den jeweiligen Projektphasen dargestellt. Die Nummerierung der einzelnen Schritte entspricht der Nummerierung in der Zeitplanung des Projektantrags. Die Planungsphase wurde jedoch in zwei separate Phasen aufgeteilt: Analysephase und Konzeptionsphase.

| Nr.* | Aufgabe | Phase | Geplante Zeit (Stunden) |
|---|---|---|---|
| 1.1 | Wirtschaftlichkeitsanalyse | Analyse | 2 |
| 1.2 | Ist-Analyse des Gerätenachrichten-Features im Backend | Analyse | 3 |
| 1.3 | Ist-Analyse der Geräteansicht des technischen Supports im Frontend | Analyse | 1 |
| 1.4 | Planung der technischen Implementierung, Abstimmung mit anderen Entwicklern | Konzeption | 5 |
| 1.5 | Entwurf eines REST-API-Endpunkts | Konzeption | 1 |
| 2.1 | Frontend-Entwicklung | Implementierung | 18 |
| 2.2 | Frontend-Tests | Implementierung | 5 |
| 2.3 | Backend-Entwicklung | Implementierung | 18 |
| 2.4 | Backend-Tests | Implementierung | 8 |
| 2.5 | Manueller Systemtest lokal | Implementierung | 1 |
| 3.1 | Code-Review mit einem anderen Entwickler | Abschluss | 2 |
| 3.2 | Deployment auf Testumgebung | Abschluss | 2 |
| 3.3 | Präsentation der Ergebnisse dem Kunden | Abschluss | 1 |
| 4.1 | Projektdokumentation | Abschluss | 10 |
| 4.2 | Entwicklerdokumentation | Abschluss | 3 |
| | **Gesamt** | | **80** |

Tabelle 1: Zeitplanung des Projekts

*laut Projektantrag

Ergebnisse einiger Phasen sind nicht in eigenen Abschnitten dieser Dokumentation dargestellt, sondern sind in den jeweiligen Phasen integriert (z.B. die Planung der technischen Implementierung umfasst sowohl Backend- als auch Frontend-Architektur und wird im Abschnitt 3 beschrieben).

# 2 Analysephase

## 2.1 Wirtschaftlichkeitsanalyse

Für die conplement AG lohnt sich das Projekt schon deswegen, weil im Vertrag mit der ABUS SC zusätzlich zu den Projektkosten auch ein Gewinnaufschlag und ein Risikozuschlag vereinbart wurde (die genaue Höhe ist vertraulich). Somit sind die Projektkosten gedeckt und es wird zusätzlich ein Gewinn erzielt.

### 2.1.1 Projektkosten

Die Projektkosten setzen sich aus den Personalkosten der beteiligten Mitarbeiter der conplement AG zusammen. Alle verwendeten Ressourcen sind bereits im Stundensatz der Mitarbeiter einkalkuliert, sodass keine zusätzlichen Kosten anfallen.

Für einen Azubi (mich) ist ein Stundensatz von 35€ veranschlagt, während für einen erfahrenen Softwareentwickler ein Stundensatz von 104€ gilt. Das Projekt umfasst insgesamt 80 Stunden, die sich wie folgt aufteilen:

- 80 Stunden Arbeit von mir als Azubi

- 5 Stunden Arbeit von einem erfahrenen Softwareentwickler zur Unterstützung bei der Planung und Code-Reviews

Daraus ergibt sich die folgende Berechnung der Projektkosten:

| **Rolle** | **Stundensatz** | **Stunden** | **Gesamtkosten** |
|---|---|---|---|
| Azubi | 35€ | 80 | 2.800€ |
| Erfahrener Entwickler | 104€ | 5 | 520€ |
| **Gesamt** | | | **3.320€** |

Tabelle 2: Projektkostenübersicht

### 2.1.2 Amortisationsrechnung aus Kundensicht

Genaue Zahlen zur Amortisation des Projekts sind schwer zu ermitteln, da das Projekt keinen direkten Umsatz für ABUS SC generiert. Stattdessen spart das Projekt indirekt Kosten, indem es die Effizienz des TS verbessert.

Basierend auf Gesprächen mit der ABUS SC und Annahmen über die Arbeitsweise des TS lassen sich folgende Schätzungen treffen:

- Der Stundensatz eines TS beträgt ca. 90€ (angenommen).
- Durch das neue Feature wird 30 Minuten weniger für eine Support-Anfrage benötigt (angenommen).
- Bei ungefähr 10% aller Geräte wird pro Jahr eine Support-Anfrage gestellt, die durch das neue Feature vereinfacht wird (angenommen).
- Es sind insgesamt 6.572 Geräte im Portal registriert (Stand: Dezember 2025).

Daraus ergeben sich folgende Berechnungen:

- Anzahl der Support-Anfragen pro Jahr:

$$6.572 \text{ Geräte} \times 10\% = 657{,}2 \approx 657 \text{ Anfragen}$$

- Zeitersparnis pro Jahr durch das neue Feature:

$$657 \text{ Anfragen} \times 0{,}5 \text{ Stunden} = 328{,}5 \text{ Stunden}$$

- Kosteneinsparung pro Jahr:

$$328,5 \text{ Stunden} \times 90€/\text{Stunde} = 29.565€$$

- Kosteneinsparung pro Monat:

$$29.565€/12 = 2.463{,}75€$$

Also wird sich das Projekt für die ABUS SC bereits nach etwa

$$3.320€/2.463{,}75€ \approx 1{,}35 \text{ Monate} \approx 2 \text{ Monate}$$

amortisieren, wenn die Annahmen zutreffen.

Aber auch bei Abweichungen von diesen Annahmen ist es sehr wahrscheinlich, dass sich das Projekt innerhalb weniger Monate amortisiert. Z.B. wenn nur für 5% der Geräte pro Jahr eine Support-Anfrage gestellt wird, bei der das neue Feature nur 15 Minuten spart und der Stundensatz eines TS nur 75€ beträgt, würde sich das Projekt immer noch innerhalb von etwa einem halben Jahr amortisieren.

### 2.1.3 Nicht-monetäre Vorteile für den Kunden

Es gibt auch mehrere nicht-monetäre bzw. schwer berechenbare Vorteile für die ABUS SC durch das neue Feature:

- Weniger Freigabeanfragen und somit mehr Zufriedenheit bei Endnutzer und Facherrichter
- Verringerte Arbeitslast der TS und PM
- Weniger Support-Anfragen, die an das Entwicklungsteam eskalieren
- Neue Sichtweise auf die Gerätedaten und Erkenntnisse, die vorher nicht möglich waren

## 2.2 Ist-Analyse

Das vorhandene System besteht aus einem Backend, einem Frontend und einer MongoDB-Datenbank. Das Backend ist mit *ASP.NET* implementiert und stellt eine *REST-API* bereit, die vom *Angular*-Frontend genutzt wird. Das Backend hat außerdem viele Schnittstellen zu weiteren internen und externen Systemen, wie z.B. einen *Service Bus*, ein *IoT Hub*, Systeme von ABUS SC usw. Das Frontend wird von einem *Node.js*-Server an die Clients ausgeliefert, der gleichzeitig auch als ein *Proxy* für die API-Anfragen fungiert. Eine abstrakte Übersicht der Architektur ist in der Abbildung 6 dargestellt.

Die MongoDB-Datenbank speichert alle Gerätedaten, darunter auch die Gerätenachrichten, die für die Auswertung benötigt werden. Sie sind in einer sog. Collection gespeichert, die den Namen `common-messages` trägt. Jedes Dokument in dieser Collection repräsentiert eine einzelne Gerätenachricht und hat folgendes Format (nur relevante Felder dargestellt):

```
{
    deviceId: Guid,              // ID des Geräts, das die Nachricht gesendet hat
    messageType: "string",       // Typ der Nachricht (z.B. "connectEvent")
    deviceType: "string",        // Typ des Geräts (z.B. "Secoris")
    enqueueTimeUtc: Date,        // Eingangszeit der Nachricht in UTC
    rawPayload: object           // Rohdaten der Nachricht
    ...                          // weitere Felder
}
```

Grundsätzlich soll als Nachrichtenart der `messageType` ausgewertet werden. Allerdings ist dieser Typ nicht immer aussagekräftig genug, da manche Gerätetypen auch viel detailliertere Typen in den `rawPayload`-Daten haben. Daher muss die genaue Auswertung je nach Gerätetyp unterschiedlich implementiert werden.

# 3 Konzeptionsphase

## 3.1 Namensgebung

Das zu implementierende Feature wird im Code als *Device Analysis* bzw. *Message Statistics* bezeichnet. Device Analysis ist der Oberbegriff für alle Arten von Auswertungen, die in der Zukunft möglicherweise hinzugefügt werden. Message Statistics ist die konkrete Auswertung der Gerätenachrichten, die in diesem Projekt implementiert wird. Die beiden Begriffe werden je nach Kontext synonym verwendet.

## 3.2 Backend-Architektur

Das Backend ist in ASP.NET Core als ein *modularer Monolith* aufgebaut. Es besteht aus dem Common-Modul und mehreren Device-Modulen, die keine Abhängigkeiten untereinander haben dürfen. Eine visuelle Darstellung der Architektur ist in der folgenden Abbildung dargestellt:

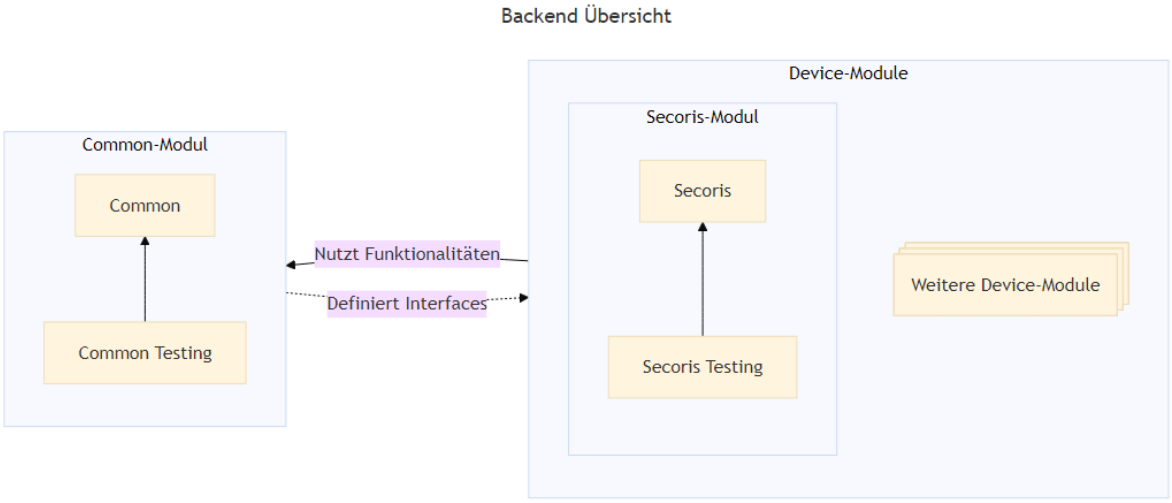


Abbildung 1: Allgemeine Übersicht der Backend-Architektur

Die Module haben folgende Eigenschaften:

- Das `Common-Modul` stellt geräteübergreifende Features und Funktionalitäten bereit, wie z.B. Authentifizierung, Profilverwaltung und Gerätesuche.
- Jedes `Device-Modul` implementiert eigene Controller, Services und Repositories, um die jeweiligen Anforderungen zu erfüllen. Sie können alle Funktionalitäten des Common-Moduls nutzen, aber nicht umgekehrt.
- Wenn ein Feature im Common-Modul gerätespezifische Daten oder Logik benötigt, wird dafür meist das *Strategy-Pattern* verwendet. Dabei wird ein Interface im Common-Modul definiert, das von den Device-Modulen implementiert und in der *ASP.NET Core Dependency Injection (DI)* registriert wird.

Die Device Analysis ist genau so ein Feature. Es wird eine allgemeine Implementierung im Common-Modul geben. Die Device-Module, wenn sie dies brauchen, werden ihre eigenen Implementierungen eines im Common-Modul definierten Interfaces bereitstellen und sich somit im Common-Modul registrieren. Je nachdem ob eine Implementierung für ein Gerätetyp vorhanden ist oder nicht, wird diese oder eine Standard-Implementierung im Common-Modul verwendet. Dies entspricht dem *Strategy-Pattern*.

### 3.3 Rest API

Für die Kommunikation zwischen Frontend und Backend wird ein neuer REST API Endpunkt im Common-Modul des Backends erstellt. Der Endpunkt wird unter dem Pfad `api/core/DeviceAnalysis/{deviceId}/MessageStatistics` implementiert. Dieser Pfad folgt dem bestehenden Muster (api/[Modul]/[Feature]/{deviceId}) und ist leicht verständlich. Außerdem ermöglicht dieses Schema eine einfache Erweiterung in der Zukunft, falls weitere Analyse-Endpunkte neben der MessageStatistics hinzugefügt werden sollen.

Die Device-Id wird als Pfadparameter übergeben. Alle anderen Parameter werden als Query-Parameter übergeben. Das sind:

- `from`: Startdatum der Auswertung im ISO 8601 Format (z.B. 2023-01-01T00:00:00Z)
- `to`: Enddatum der Auswertung im ISO 8601 Format (z.B. 2023-01-07T23:59:59Z)
- `deviceType`: Der Typ des Geräts als *string*. Dies bestimmt, welches Device-Modul die Anfrage verarbeitet. Das Backend kann den Typ aus der Device-Id nicht ableiten, ohne zusätzliche Abfragen in der Datenbank durchzuführen. Daher muss der Client den Typ explizit angeben.

Die Antwort ist ein JSON-Objekt, das die Nachrichtentypen und deren Anzahl im angegebenen Zeitraum

enthält. Ein Beispiel für die Antwortstruktur ist wie folgt:

```
{
    "connectEvent": 150,
    "assignmentChange": 35,
    "firmwareUpdate": 20
}
```

## 3.4 Frontend-Architektur

Das Frontend ist in Angular als eine *Single-Page Application* (SPA) aufgebaut. Ähnlich wie im Backend gibt es Common-Module, Feature-Module und Device-Module.

- Die `Common-Module` enthalten feature- und geräteübergreifende Komponenten, Services und Utilities, die von allen anderen Modulen genutzt werden können.
- Die `Feature-Module` sind für nicht-gerätespezifische aber selbstständige Features zuständig, wie z.B. Suchseite, Profilseite usw.
- Für jeden Gerätetyp gibt es ein eigenes `Device-Modul`, das die gerätespezifischen Komponenten, Services und Routen enthält und sie bei den Feature-Modulen registriert.
- Für nicht-gerätespezifische Komponenten, die aber nur in den Device-Modulen verwendet werden, gibt es ein `Device-Common-Modul`. Darin werden z.B. gemeinsame UI-Komponenten für die Geräte-Detailansicht bereitgestellt.

Die Auswertungsansicht soll in jede Geräte-Detailansicht als Tab eingebaut werden. Daher wird sie einmal im Device-Common-Modul implementiert und von den Device-Modulen genutzt. Allerdings muss das Routing und die Anzeigelogik von Tabs in jedem Device-Modul geändert werden, da jedes Modul seine eigenen Routen und Tabs verwaltet. Damit die Rollen PM und TS ohne Freigabe durch den Facherrichter auf die Auswertungsansicht zugreifen können, muss die Sichtbarkeit der Tabs in allen Device-Modulen überarbeitet werden.

## 3.5 Frontend-UI

Grundsätzlich sind zwei bestehende Ansichten im Frontend relevant: die Suchseite und die Geräte-Detailansicht. Auf den folgenden Abbildungen sind die beiden Ansichten dargestellt:

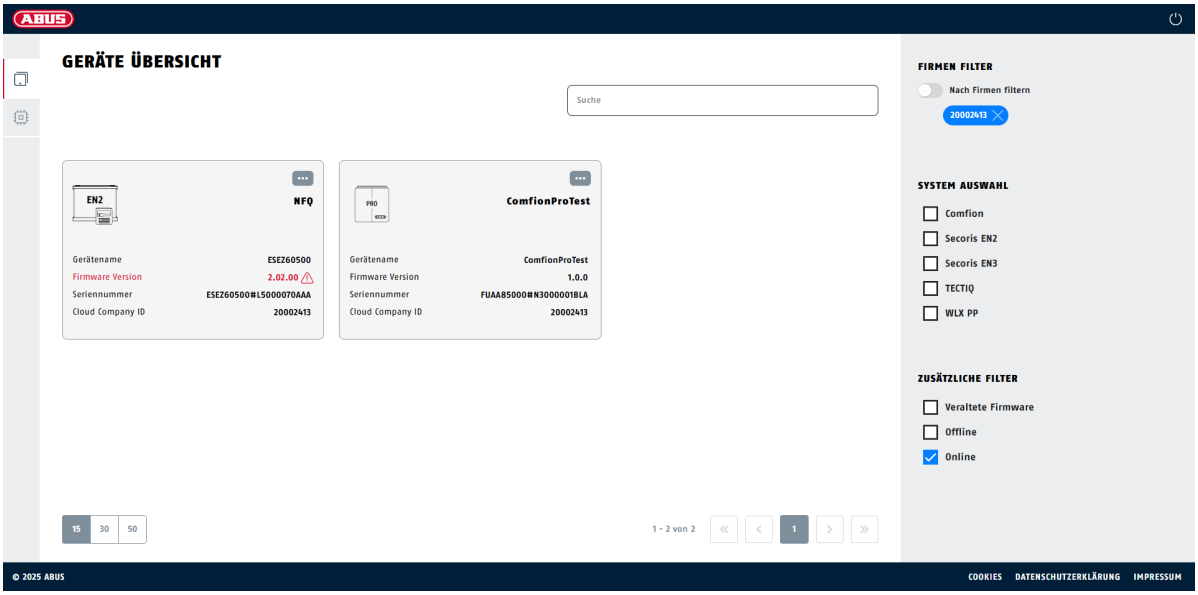


Abbildung 2: Suchseite aus Sicht eines TS (mit Mock-Daten)

Beim Klicken auf eine Gerätekachel kommt man auf die Geräte-Detailansicht (vorausgesetzt, man hat Zugriff auf das Gerät):

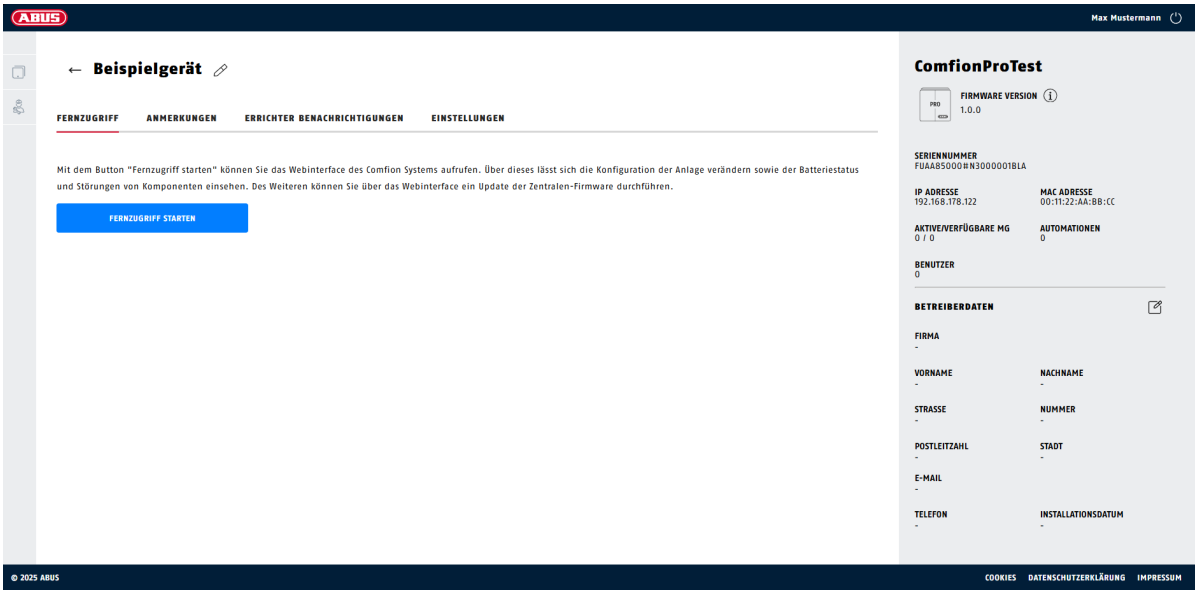


Abbildung 3: Geräte-Detailansicht aus Sicht eines Facherrichters (mit Mock-Daten)

Die Auswertungsansicht wird als ein Tab in der Geräte-Detailansicht implementiert. Der Tab enthält eine Auswahlmöglichkeit für den Zeitraum der Auswertung (letzte 24 Stunden, letzte 7 Tage, letzter Monat usw.) und eine Darstellung der Gerätenachrichten in einem horizontalen Balkendiagramm. An der x-Achse wird die Anzahl der Nachrichten und an der y-Achse der Nachrichtentyp angezeigt.

Für einen PM, der davor auf die Geräte-Detailansicht gar keinen Zugriff hatte, wird dies der einzige Tab sein, den er sehen kann. Für einen TS, der keinen Zugriff auf das Gerät hat, wird dies der einzig *zugängliche* Tab sein. Andere Tabs wie Fernwartung, Logs usw. werden zwar in der Tab-Leiste angezeigt, leiten aber zu einer Seite weiter, die den fehlenden Zugriff erklärt. Falls der TS Zugriff auf das Gerät hat, werden alle Tabs wie gewohnt angezeigt. Wenn ein TS ohne Gerätezugriff auf einen unzugänglichen Tab klickt, wird stattdessen ein Tab angezeigt, das den fehlenden Zugriff erklärt.

Ein Mockup der Auswertungsansicht wurde von dem UI/UX-Designer von ABUS SC bereitgestellt[2]. Die Darstellung des Diagramms wurde allerdings mir überlassen, da sie stark von der gewählten Chart-Bibliothek abhängt.

Die Chart-Bibliothek **Chart.js** wurde ausgewählt, da sie eine gute Balance zwischen Funktionsumfang, Einfachheit und Performance bietet. Außerdem hat sie viel Support von der Community und ist gut dokumentiert. Es gibt zwar einen Angular-Wrapper für Chart.js namens **ng2-charts**, aber sie bietet nichts weiteres als eine `provideCharts()` Methode und ein `BaseChartDirective` für die Chart-Komponenten. Dies ist ziemlich einfach auch ohne den Wrapper zu implementieren, daher wird Chart.js direkt verwendet. So wird eine unnötige Abhängigkeit vermieden.

# 4 Implementierungsphase

## 4.1 Backend

Im Backend wurde ein neuer Controller `DeviceAnalysisController` im Common-Modul erstellt, der den Endpunkt `api/core/DeviceAnalysis/deviceId/MessageStatistics` bereitstellt. Die Hauptaufgabe des Controllers ist es, die passende Implementierung des `IDeviceAnalysisRepository` Interface basierend auf dem `deviceType` Query-Parameter auszuwählen und die Anfrage an diese Implementierung weiterzuleiten.

[2]Das Mockup kann nicht zur Dokumentation beigefügt werden, da es von dem Designer gelöscht wurde, nachdem ein neues Design gefordert wurde. Dies wird im Kapitel 4.2 erläutert.

Das `IDeviceAnalysisRepository` ist, wie der Name schon sagt, ein Repository-Interface. Das heißt, es ist für die Kommunikation mit der Datenbank zuständig. Jede Implementierung des Interfaces ist für einen bestimmten Gerätetyp zuständig und hat eine Methode `IsRepositoryFor(deviceType)`, die überprüft, ob die Implementierung für den angegebenen Gerätetyp zuständig ist.

Die Standard-Implementierung des `IDeviceAnalysisRepository` Interface im Common-Modul führt eine generische Datenbank-Abfrage aus, die alle Nachrichten von dem angegebenen Gerät in dem gewählten Zeitraum aggregiert und die Häufigkeit der Nachrichten pro `messageType` zählt. Diese Implementierung funktioniert für alle Gerätetypen, kann aber nur die generischen Nachrichtentypen verarbeiten. Ihre `IsRepositoryFor(deviceType)` Methode gibt immer `false` zurück, da sie für keinen spezifischen Gerätetyp zuständig ist. Sie wird als ein sog. *Keyed Singleton* in der DI registriert, sodass der Controller sie separat von den gerätespezifischen Implementierungen injizieren kann. Dies wird mit einem Attribut `[FromKeyedServices(key)]` im Konstruktor des Controllers erreicht.

Einige Gerätetypen (derzeit nur *Secoris*) haben spezifischere Nachrichtentypen, die in der Standard-Implementierung nicht berücksichtigt werden können. Sie werden aus dem `rawPayload`-Feld der Nachricht extrahiert, welches gerätespezifisch deserialisiert werden muss. Außerdem müssen bei *Secoris* einige Nachrichtentypen gefiltert werden, da sie für die Auswertung nicht relevant sind. Dafür registriert das *Secoris*-Device-Modul eine eigene Implementierung des `IDeviceAnalysisRepository` Interface in der DI, die diese gerätespezifische Abfrage enthält. Ihre `IsRepositoryFor(deviceType)` Methode gibt `true` zurück, wenn der `deviceType`-Parameter den Wert `"hyen"` hat (der interne Name für *Secoris*-Geräte).

Die Ergebnisse der Datenbank-Abfrage werden in ein sog. *Data Transfer Object* (DTO) umgewandelt und als JSON-Antwort an das Frontend zurückgegeben. Das DTO ist ein einfaches Objekt, das die Nachrichtentypen und deren Häufigkeit als Schlüssel-Wert-Paare enthält.

Eine grafische Darstellung der Backend-Architektur des neuen Features ist in der Abbildung 7 zu sehen.

### 4.2 Frontend

Im Frontend wurde mit dem Routing und der Anzeigelogik in den Device-Modulen begonnen, weil das der Teil war, in dem es die größte Unsicherheit gab. Die Implementierung der Auswertung selbst im Device-Common-Modul war relativ einfach, da es nur eine Komponente mit einem Diagramm und einer Datumsauswahl ist.

Es stellte sich heraus, dass dies die richtige Entscheidung war, da es erhebliche Herausforderungen bei der Integration der Auswertungsansicht in die Geräte-Detailansicht gab. Die Geräte-Detailansichten waren ursprünglich nicht dafür ausgelegt, dass auch Rollen wie TS und PM ohne Freigabe durch den Facherrichter auf sie zugreifen können. An vielen Stellen wurde Gerätezugriff als eine Vorbedingung für die Anzeige von Inhalten angenommen, was zu Problemen führte. Dies wurde in der Planungsphase nicht erkannt, da diese Vorbedingung nur implizit in der bestehenden Codebasis vorhanden war und nicht dokumentiert wurde.

Es zeigte sich zudem, dass das Routing eine erhöhte Komplexität aufweist, siehe dazu Abbildung 8. Dabei muss man beachten, dass das Routing in Angular nicht mit einfachen if/else/switch-Anweisungen gesteuert werden kann, sondern mit sogenannten *Guards*. Es gibt mehrere Arten von Guards in Angular, die jeweils zu einem bestimmten Zeitpunkt im Routing-Prozess ausgeführt werden und dementsprechend verschiedene Daten zur Verfügung haben. Das macht die Implementierung der komplexen Logik noch schwieriger.

Diese Komplexität führte zu mehreren Herausforderungen während der Implementierung:

- Der Code für das Routing und die Anzeigelogik wurde sehr unübersichtlich und schwer wartbar.
- Es war schwierig, alle möglichen Szenarien und Randfälle zu berücksichtigen und zu testen.
- Neue Device-Module in der Zukunft müssen die komplexe Logik ebenfalls implementieren, was zu Fehlern führen kann und die Integration neuer Geräte erschwert.
- Die Implementierung war in der vorgegebenen Zeit nicht vollständig abschließbar.

Aus diesen Gründen wurde beschlossen, das initiale Konzept **nicht** weiter zu verfolgen. Stattdessen wird die Auswertungsansicht als ein Dialogfenster implementiert, der von einem Button in dem Drei-Punkte-

Menü der Gerätekachel auf der Suchseite geöffnet wird. Dies kann man in der folgenden Abbildung sehen (Systemanalyse ist der deutsche Begriff für Device Analysis):


Abbildung 4: Gerätekachel mit geöffnetem Drei-Punkte-Menü und Systemanalyse-Button

Diese Änderung hat mehrere Vorteile:

- Das Menü, die Zugriffslogik und die Registrierung des Buttons sind bereits implementiert, sodass nur noch wenig Setup-Arbeit nötig ist.
- Es gibt keine Änderungen an der Geräte-Detailansicht oder dem Routing, was die Komplexität erheblich reduziert.
- Die Implementierung ist in der restlichen Zeit abschließbar.

Die einzig verbleibende Herausforderung sind die Gerätedaten. In der Geräte-Detailansicht ist eine Übersicht der Gerätedaten bereits in einem Sidebar vorhanden. In einem Dialogfenster dagegen nicht - also müssen die Gerätedaten an das Dialogfenster übergeben und dort angezeigt werden. In der folgenden Abbildung ist dieser Dialog dargestellt:

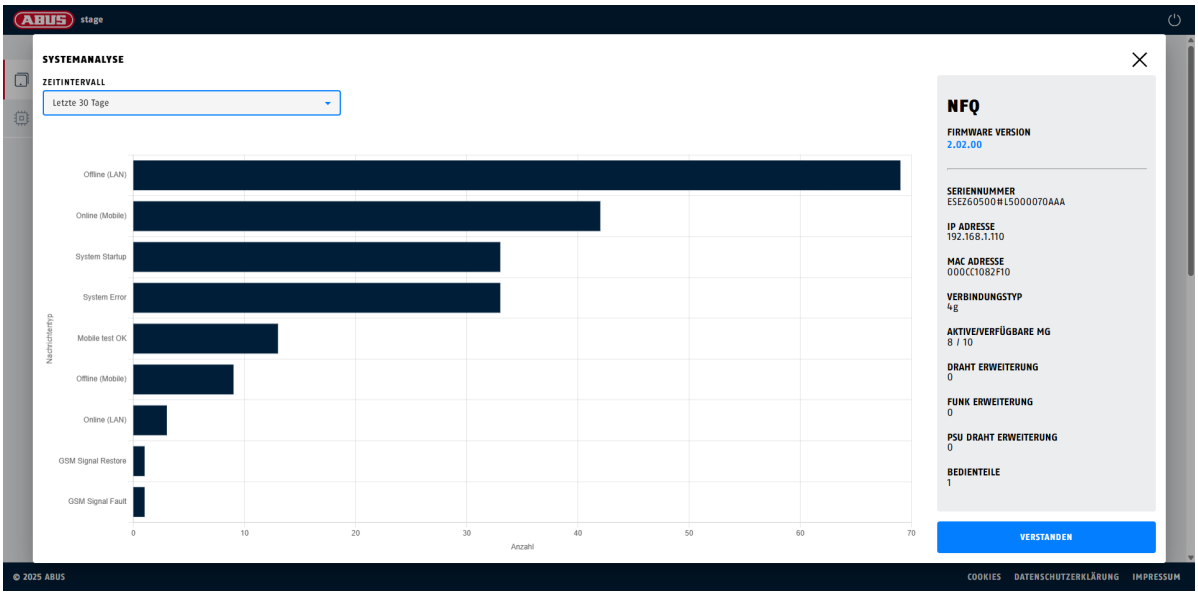


Abbildung 5: Dialogfenster mit der Auswertungsansicht

Die Hauptaspekte der neuen Implementierung sind:

- Die Geräte registrieren in ihren Gerätekachel-Komponenten (siehe Quellcode 1) eine sog. *DeviceAna-*

*lysisAction*, die ein Service im Device-Common-Modul zur Verfügung stellt. Dies erfolgt über eine Methode, die im Quellcode 2 dargestellt ist. Solche *Actions* sind bereits für andere Features wie z.B. Zurücksetzen des Geräts implementiert. Somit können Geräte-Module nur die *Actions* registrieren, die sie tatsächlich unterstützen, und diese ggf. konfigurieren.

- Der Button nimmt als Abhängigkeit ein `DeviceAnalysisProvider`, der die gerätespezifischen Daten für die Auswertungsansicht bereitstellt. Das ist notwendig, da die Komponente im Device-Common-Modul nicht wissen kann, welche Felder angezeigt werden müssen. Dies muss zur Laufzeit entschieden werden. Ein Beispiel dafür ist die Implementierung für Secoris im Quellcode 3.
- Die Grafik wird wie geplant mit `Chart.js` umgesetzt. Es wird ein `Chart`-Objekt in der `ngOnInit()`-Methode (d.h. beim Initialisieren der Komponente) erstellt und konfiguriert. Bei Änderungen der Daten oder des ausgewählten Zeitraums wird das `data`-Attribut des `Chart`-Objekts aktualisiert und die `update()`-Methode aufgerufen, um die Grafik neu zu rendern.

## 4.3 Tests

Die Tests sind für die Qualitätssicherung des neuen Features von entscheidender Bedeutung. Für das gesamte Portal-Projekt gibt es keine konkrete Vorgabe für einen Grad der Testabdeckung. Es wird pragmatisch entschieden, welche Tests für ein Feature notwendig sind, um dessen Qualität sicherzustellen.

Deswegen wird im Backend hauptsächlich auf Integrationstests gesetzt, um maximale Abdeckung bei minimalem Aufwand zu erreichen. Im Frontend werden Unit-Tests für die wichtigsten Komponenten und Services geschrieben, um deren Funktionalität zu überprüfen.

### 4.3.1 Backend

Im Backend werden Integrationstests für alle *Repositories* geschrieben, um sicherzustellen, dass sie die Daten korrekt aus der Datenbank abrufen. Diese Tests haben die folgenden Merkmale:

- Sie sind in *xUnit* geschrieben, da das Backend bereits dieses Test-Framework verwendet.
- Sie verwenden ein *Test Container* — eine kleine, temporäre Instanz der MongoDB-Datenbank, die nur für die Dauer der Tests läuft. Dadurch wird sichergestellt, dass die Tests in einer isolierten Umgebung laufen und nicht von externen Faktoren beeinflusst werden.
- Sie verwenden die Bibliothek *Fluent Assertions*, um die Testergebnisse auf eine lesbare und verständliche Weise zu überprüfen.
- Um Testläufe kurz zu halten, deckt jeder Test so viele Szenarien wie möglich ab. Dies macht sie zwar nicht so isoliert wie Unit-Tests, aber es reduziert die Gesamtanzahl der Tests erheblich.
- Die Tests verwenden ein *Arrange, Act, Assert (AAA)*-Muster, um die Struktur der Tests klar und konsistent zu halten.

Ein Beispiel für einen Integrationstest ist im Quellcode 4 dargestellt.

### 4.3.2 Frontend

Als das erste Konzept für die Frontend-Implementierung verworfen wurde (siehe Abschnitt 4.2), mussten auch viele Frontend-Tests verworfen werden, da sie auf dem ursprünglichen Konzept basierten. Nur eine Handvoll Tests konnte wiederverwendet werden.

Für das neue Konzept werden aus folgenden Gründen **keine** automatisierten Tests geschrieben:

- Die Implementierung ist relativ einfach und besteht entweder aus UI-Komponenten oder aus Registrierungslogik, die bereits in anderen Teilen des Codes getestet wurde. Also wären hier *Unit-Tests* nicht sehr nützlich.
- *Integrationstests* werden im Frontend nur für geschäftskritische Features geschrieben, z.B. die Suchseite. Die Auswertungsansicht ist kein geschäftskritisches Feature, daher rechtfertigt sie keine

*Integrationstests.*

- Die für das Frontend-Testing eingeteilte Zeit wurde schon für die ursprüngliche Implementierung verwendet. Zeit von anderen Aufgaben abzuzweigen, um Tests zu schreiben, wäre aus oben genannten Gründen nicht effizient gewesen.

# 5 Abschlussphase

## 5.1 Code-Review

Code-Reviews fanden sowohl während der Implementierungsphase als auch nach deren Abschluss statt. Während der Implementierung wurden regelmäßig kleinere Code-Reviews durchgeführt, um sicherzustellen, dass die Entwicklung in die richtige Richtung geht und um frühzeitig Feedback zu erhalten.

Nach Abschluss der Implementierung wurde ein *Pull Request* erstellt und von zwei Entwicklern (einem für das Backend und einem für das Frontend) überprüft. Dabei sind keine größeren Probleme festgestellt worden. Es gab nur einige Anmerkungen bezüglich Code-Stil, dupliziertem Code und einigen kleineren Verbesserungen, die schnell behoben wurden.

## 5.2 Deployment

Das Portal hat drei Umgebungen: Dev, Stage und Prod (Produktion). Auf Dev sind meist nur Entwickler aktiv, die dort neue Features implementieren und testen. Auf Stage können auch bestimmte Mitarbeiter von ABUS SC testen, ob alles wie erwartet funktioniert. Prod ist die Live-Umgebung, die von den Endbenutzern genutzt wird.

Nachdem die Implementierung abgeschlossen war, lokal getestet wurde und das Code-Review bestanden hat, wurde die neue Version des Backends und Frontends gebaut, getestet und auf die Dev-Umgebung deployed. Dort wurde nochmal manuell getestet, ob alles wie erwartet funktioniert.

An dieser Stelle wurde ein Bug gefunden. Dies wird in der nächsten Sektion beschrieben. Nachdem der Bug behoben war, wurde die neue Version nochmal auf die Dev-Umgebung deployed und getestet. Diesmal funktionierte alles wie erwartet und der Build wurde auf die Staging-Umgebung deployed.

Auf die Prod-Umgebung wird das neue Feature erst mit dem nächsten Release deployed, da jedes Deployment auf Prod ein Release darstellt und dies darf nur zu bestimmten, von ABUS SC eingeplanten Terminen durchgeführt werden.

## 5.3 Bugfixes

Als das neue Feature in der Dev-Umgebung getestet wurde, wurde ein Bug gefunden:
Das Diagramm konnte nicht geladen werden, da die API-Anfrage an das Backend mit einem 400-Fehler (Bad Request) fehlschlug. Nach einer Untersuchung des Problems wurde festgestellt, dass das Backend die Anfrage ablehnte, weil das Enddatum in der Zukunft lag. Dies lag daran, dass das Backend und das Frontend unterschiedliche Zeitzonen verwendeten. Das Backend verwendete UTC (Coordinated Universal Time), während das Frontend die lokale Zeitzone des Benutzers verwendete (in diesem Fall UTC+1).

Das Backend wurde initial so implementiert, dass es eine Fehlerantwort zurückgibt, wenn das Enddatum in der Zukunft liegt. Dies war eine bewusste Entscheidung, um ungültige Anfragen abzuweisen. Allerdings wurde nach einer genaueren Überlegung gesehen, dass dieses Verhalten kein echtes Problem löst und stattdessen den Endpunkt bloß weniger robust macht. Daher wurde beschlossen, dass das Enddatum im Backend in solchen Fällen automatisch auf das aktuelle Datum gesetzt wird.

Dies hat den Bug behoben und nach einem erneuten Deployment auf die Dev-Umgebung funktionierte das Diagramm wie erwartet. Weitere Bugs wurden nicht gefunden.

### 5.4 Abnahme

Nach einer Demonstration des neuen Features für den Projektbetreuer von ABUS SC wurde eine kleine Änderung am Design des Diagramms gewünscht. Die Standardoption für die Zeitauswahl sollte auf „Letzte 7 Tage“ anstatt „Letztes Jahr“ gesetzt werden und eine Option für „Letzte 24 Stunden“ sollte hinzugefügt werden. Diese Änderung wurde kurzfristig implementiert und wieder auf Dev und Stage deployed.

Danach wurde das neue Feature von dem Projektbetreuer abgenommen. Es gab keine weiteren Änderungswünsche und das Feature wurde für gut befunden.

### 5.5 Dokumentation

Als Entwicklerdokumentation im Backend dient die automatisch generierte `Swagger`-Dokumentation für die Backend-API. Diese ist unter `https://stage.abus-cloud.com/swagger/index.html` erreichbar (siehe Core - `POST /api/core/DeviceAnalysis/{deviceId}/MessageStatistics`). Außerdem wurde auf sprechende Methoden- und Variablennamen geachtet, damit der Code auch ohne zusätzliche Dokumentation verständlich ist.

Im Frontend wurde ebenfalls auf sprechende Namen geachtet. Da das Frontend jedoch relativ einfach und selbsterklärend ist, wurde keine zusätzliche Dokumentation erstellt.

Kundendokumentation ist in Form des abgeschlossenen *Product Backlog Item* in Jira vorhanden. Außerdem wurden zwei Ansprechpartner von ABUS SC eingewiesen, die bei Fragen kontaktiert werden können. Das Feature wird ausschließlich von ABUS-internen Rollen genutzt, daher ist keine weitere Kundendokumentation erforderlich.

## 6 Fazit

### 6.1 Soll-/Ist-Vergleich

| Nr. | Aufgabe | Soll (Std.) | Ist (Std.) | Diff. (Std.) |
|---|---|---|---|---|
| **1** | | **12** | **11** | **-1** |
| 1.1 | Wirtschaftlichkeitsanalyse | 2 | 2 | |
| 1.2 | Ist-Analyse Backend | 3 | 2 | -1 |
| 1.3 | Ist-Analyse Frontend | 1 | 2 | +1 |
| 1.4 | Planung der Implementierung | 5 | 4 | -1 |
| 1.5 | Entwurf REST-API-Endpunkt | 1 | 1 | |
| **2** | | **50** | **52** | **+2** |
| 2.1 | Frontend-Entwicklung | 18 | 26:30 | +8:30 |
| 2.2 | Frontend-Tests | 5 | 5 | |
| 2.3 | Backend-Entwicklung | 18 | 14:30 | -3:30 |
| 2.4 | Backend-Tests | 8 | 5 | -3 |
| 2.5 | Manueller Systemtest lokal | 1 | 1 | |
| **3** | | **5** | **4** | **-1** |
| 3.1 | Code-Review | 2 | 2 | |
| 3.2 | Deployment auf Testumgebung | 2 | 1 | -1 |
| 3.3 | Präsentation der Ergebnisse | 1 | 1 | |
| **4** | | **13** | **13** | |
| 4.1 | Projektdokumentation | 10 | 10 | |
| 4.2 | Entwicklerdokumentation | 3 | 3 | |
| | **Gesamt** | **80** | **80** | |

Tabelle 3: Soll-/Ist-Vergleich der Projektphasen

Das Zeitbudget vom 80 Stunden wurde wie geplant eingehalten. Es gab jedoch Abweichungen in den einzelnen Phasen, am meisten in der Implementierungsphase. Die Frontend-Implementierung hat deutlich

mehr Zeit in Anspruch genommen als geplant, während die Backend-Implementierung weniger Zeit benötigte.

Dafür gab es zwei Hauptgründe. Zum einen gab es Herausforderungen bei der Implementierung der Auswertungsansicht im Frontend. Erst tief in der Implementierung wurde klar, dass die initiale Idee nicht sinnvoll umsetzbar war und eine alternative Lösung gewählt werden musste. Dadurch entstand Mehrarbeit, die nicht eingeplant war.

Der zweite Grund für die Abweichung war die Annahme, dass ich viel mehr Erfahrung im Frontend habe als im Backend und daher für das Backend mehr Zeit brauchen würde. Es zeigte sich jedoch, dass die Backend-Implementierung relativ unkompliziert war. Dadurch benötigte ich dort weniger Zeit, als für die Auswertungsansicht im Portal ursprünglich vorgesehen war — trotz meiner geringeren Erfahrung im Vergleich zum Frontend.

## 6.2 Erkenntnisse

In diesem Projekt war die Entscheidung, das ursprüngliche Konzept für die Auswertungsansicht zu verwerfen und stattdessen einen Dialog zu verwenden, entscheidend für den erfolgreichen Abschluss der Implementierung innerhalb des Zeitrahmens. Hätte ich weiter an dem ursprünglichen Konzept festgehalten, wäre die Implementierung wahrscheinlich nicht rechtzeitig fertig geworden. In der Zukunft sollte ich am besten noch früher im Prozess solche unnötig komplexen Konzepte erkennen und auf deren Vereinfachung bestehen.

Außerdem hat sich die Entscheidung, `ng2-charts` nicht zu verwenden, als richtig erwiesen. Nach dem Abschluss der Implementierung unterstützte die Wrapper-Bibliothek die letzten zwei Versionen von Angular nicht mehr vollständig, was ein Anzeichen für mangelnde Wartung ist. Hätte ich die Bibliothek verwendet, würden in der Zukunft wahrscheinlich Kompatibilitätsprobleme beim Upgrade auf neuere Angular-Versionen auftreten. `Chart.js` dagegen hängt gar nicht von Angular ab und ist daher zukunftssicherer.

## 6.3 Ausblick

Das neue Feature zur Auswertung der Gerätedaten wird mit dem nächsten Release im ersten Quartal 2026 ausgeliefert. Die Ansprechpartner von ABUS SC, die gleichzeitig die Nutzer davon sind, haben das Feature bereits getestet und für gut befunden, was auf eine erfolgreiche Einführung schließen lässt. Das Feature wurde so gebaut, dass es in der Zukunft leicht erweitert werden kann, z.B. durch Hinzufügen neuer Diagrammtypen oder Filteroptionen, auch gerätespezifisch.

# A Quellenverzeichnis

[Goo] Google. Angular Routing (Engl.). https://angular.dev/guide/routing.

[MDN] MDN. Responsive Web Design (Engl.). https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design.

[Mica] Microsoft. Create web APIs with ASP.NET Core (Engl.). https://learn.microsoft.com/en-us/aspnet/core/web-api.

[Micb] Microsoft. The Repository pattern (Engl.). https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design#the-repository-pattern.

[Mona] MongoDB, Inc. Databases and Collections in MongoDB (Engl.). https://www.mongodb.com/docs/manual/core/databases-and-collections/.

[Monb] MongoDB, Inc. What is MongoDB? (Engl.). https://www.mongodb.com/company/what-is-mongodb.

# B Nutzung der künstlichen Intelligenz

Bei der Implementierung und Dokumentation des Projekts wurden verschiedene KI-Tools eingesetzt, um den Entwicklungsprozess zu unterstützen und zu beschleunigen. Die verwendeten Tools waren:

- ChatGPT (OpenAI, https://chatgpt.com/, letzter Aufruf: 12.12.2025)
- GitHub Copilot (GitHub, https://github.com/copilot, letzter Aufruf: 12.12.2025, hauptsächlich als VSCode-Plugin)

## B.1 Verwendungszweck bzw. Einsatzszenario

Die KI-Tools wurden hauptsächlich für folgende Zwecke eingesetzt:

- IntelliSense-ähnliche Code-Vervollständigungen (kein „Vibe Coding“, reine Tipphilfe)
- Formulierungshilfen für die Dokumentation (aber **nicht** die Inhalte selbst)
- Formatierungshilfen (z.B. LaTeX-Syntax)
- Generierung von Mermaid-Diagrammen für die Dokumentation (aus detaillierten textuellen Beschreibungen)

## B.2 Ergänzende Hinweise

Es ist kaum möglich, den genauen Anteil der mit KI-Tools erstellten Inhalte zu beziffern. Genauso wenig ist es möglich, genaue Stellenangaben in der Arbeit zu machen, da die KI-Tools nicht zur Erstellung ganzer Abschnitte oder Kapitel verwendet wurden. Sie dienten als Hilfsmittel während des ganzen Projekts.

**Es ist jedoch sicher, dass die Eigenständigkeit der Arbeit gewahrt bleibt, da alle Inhalte und Entscheidungen ausschließlich von mir stammen. Die KI-Tools wurden nur als eine Arbeitserleichterung bzw. *Quality of Life*-Verbesserung eingesetzt.**

# C Abbildungen

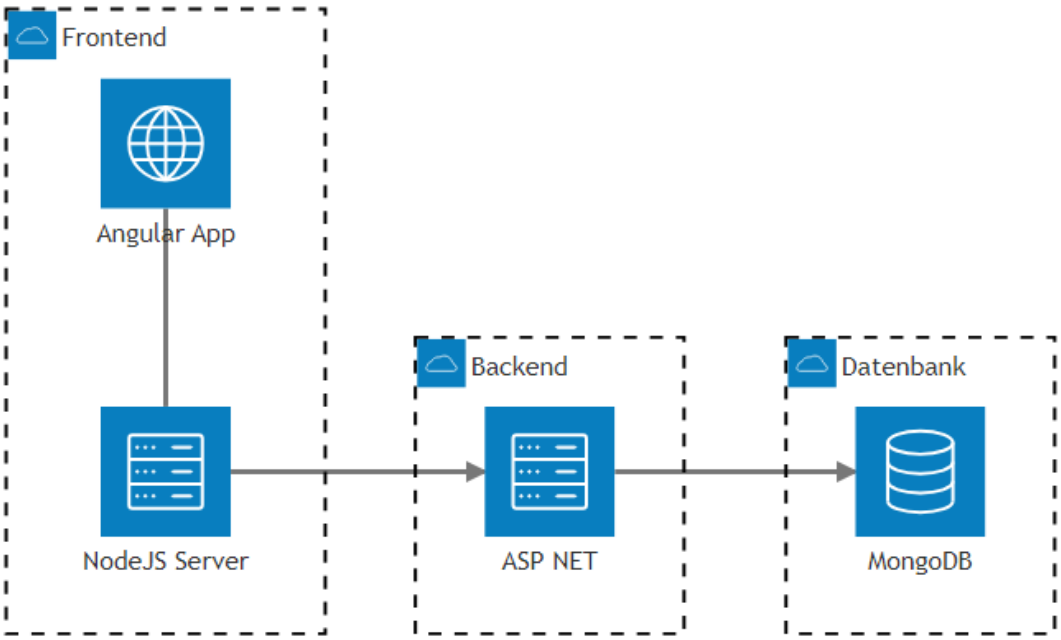


Abbildung 6: Systemübersicht des Portals

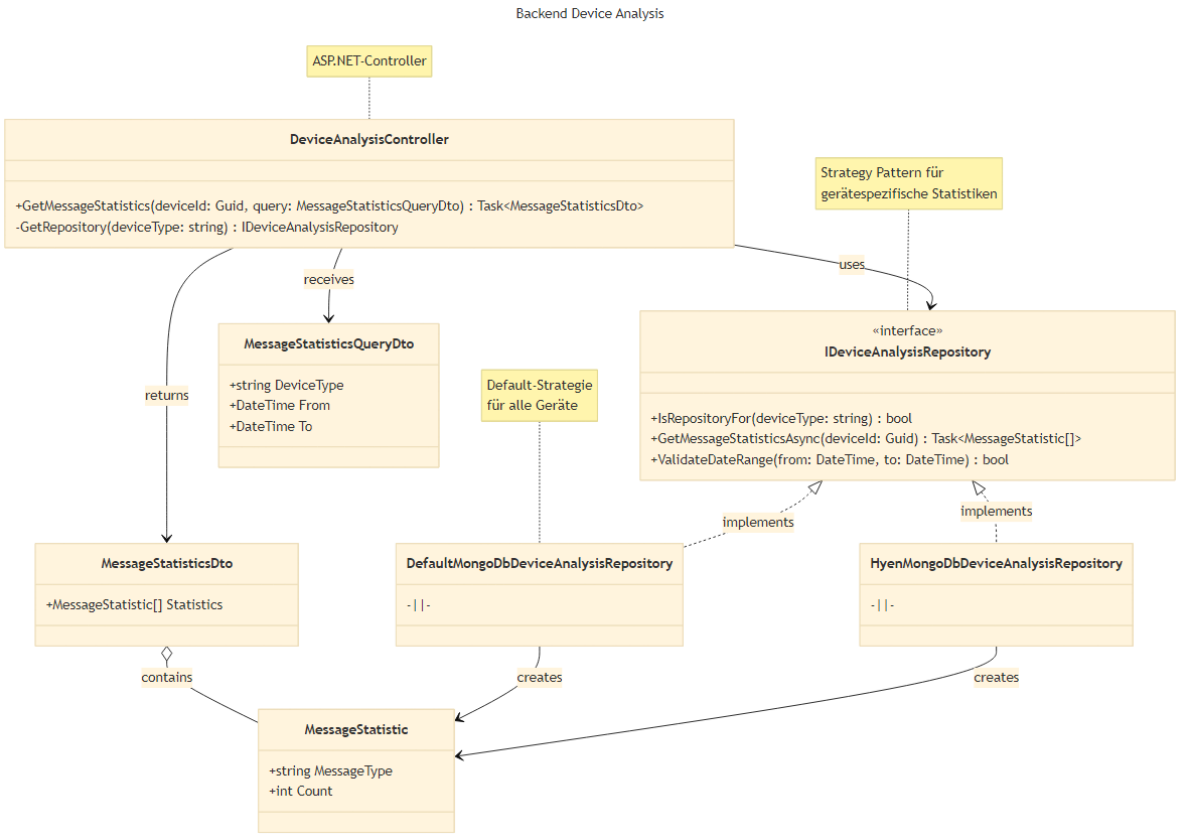


Abbildung 7: Backend-Architektur des neuen Features (vereinfacht dargestellt)

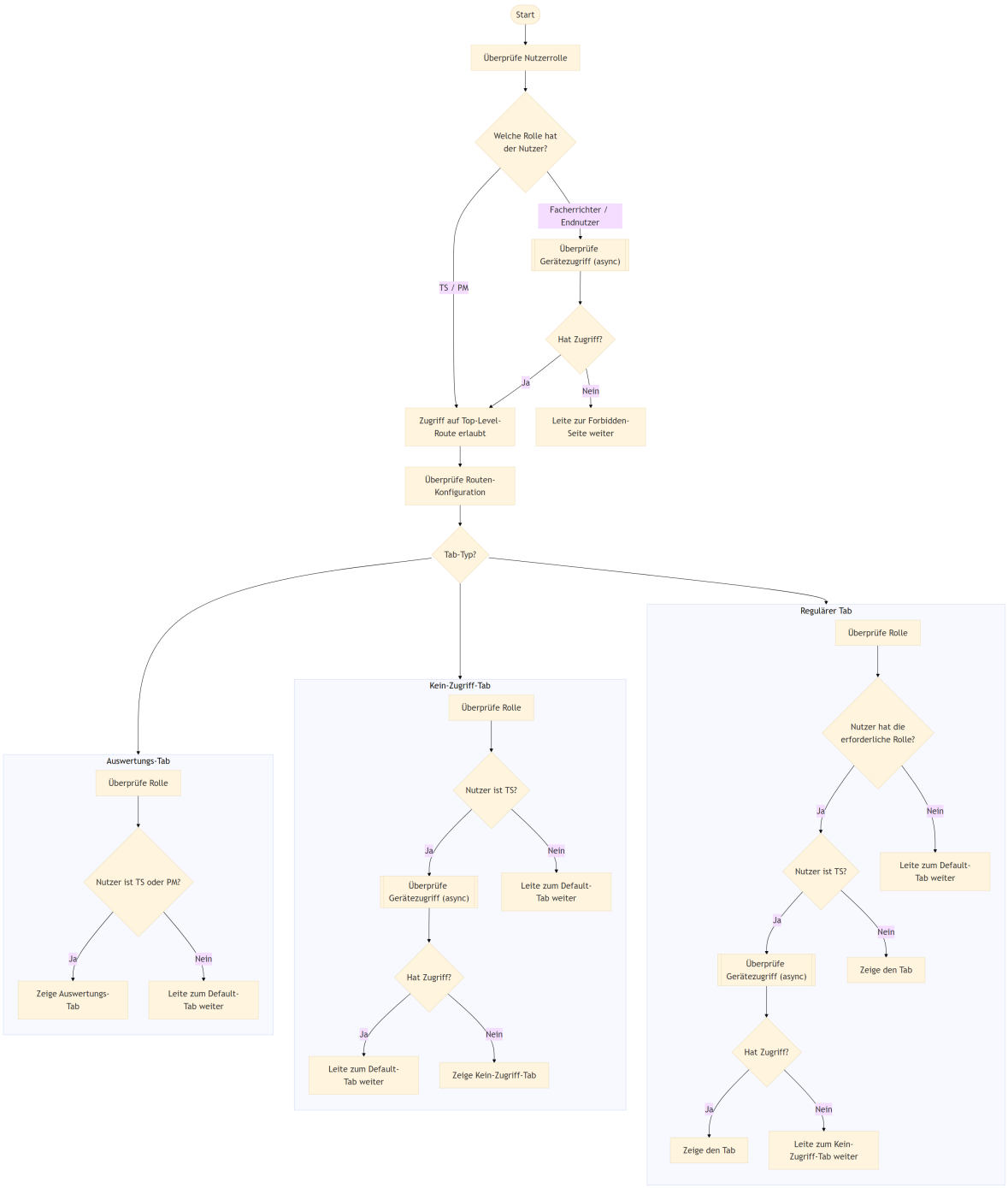


Abbildung 8: Routing-Entscheidungsbaum für den Auswertungs-Tab in einem Device-Modul

# D Code-Beispiele

**Anmerkung:** Die folgenden Codebeispiele sind Auszüge aus dem tatsächlichen Quellcode.

Allerdings wurden deutschsprachige Kommentare und Erklärungen hinzugefügt, um den Code ohne weiteren Kontext verständlich zu machen. Im tatsächlichen Quellcode sind diese Kommentare nicht vorhanden.

Außerdem wurden einige für das Projekt nicht relevante Teile des Codes mit ihren Imports und Abhängigkeiten weggelassen, um den Leser nicht mit irrelevanten Details zu überladen. Einige Klassen- und Variablennamen wurden verkürzt/vereinfacht.

Umlaute in den Kommentaren wurden aus technischen Gründen entfernt (z.B. „ä“ zu „ae“).

```
import {
  DeviceActionsProvider,
  CommonSearchCardComponent,
  SearchResultDeviceCard,
  SearchResultDeviceItem,
} from '@abus-cloud-frontend/device-common';
import { Component, computed, input } from '@angular/core';
import { DEVICE_SECORIS } from '../secoris-consts';
import { SecorisSearchItemContent } from '../types';
import { SecorisDeviceAnalysisProvider } from './secoris-device-analysis.provider';

@Component({
    selector: `abus-secoris-search-result-card`,
    imports: [CommonSearchCardComponent],
    // template ist hier vereinfacht dargestellt, da nur die Aktionen relevant sind
    template: `
      <common-search-card
        [actions]="getActions()"
      >
      </common-search-card>
    `
})
export class SecorisSearchResultCardComponent implements SearchResultDeviceCard {
  type: string = DEVICE_SECORIS;

  searchTerm = input.required<string>();
  searchItem = input.required<SearchResultDeviceItem<SecorisSearchItemContent>>();
  protected device = computed(() => ({...this.searchItem(), ...this.searchItem().
      content}));
  protected access = computed(() => this.searchItem().access);

  constructor(
    private actionsProvider: DeviceActionsProvider,
    private deviceAnalysisProvider: SecorisDeviceAnalysisProvider
  ) {}

  public getActions() {
    return [
      this.actionsProvider.getDeviceAnalysisAction({
        // Allgemeine Geraetedaten
        deviceId: this.device().RegisteredDeviceId,
        deviceType: this.type,
        resolvedDeviceTitle: this.getProjectNameOrFallback(),
        firmwareVersion: this.device().FirmwareVersion,

        // Provider fuer geraetespezifische Daten
        provider: this.deviceAnalysisProvider
      })
    ];
  }

  protected getProjectNameOrFallback(): string {
    // Loest den Projektnamen auf abhaengig von der Nutzerrolle und vorhandenen Daten.
    // Irrelevant fuer das Projekt.
  }
}
```

Quellcode 1: Registrierung des `DeviceAnalysisDeviceActions` in der Secoris-Gerätekachel

```typescript
import { Injectable } from '@angular/core';
import { DeviceAction } from './types';
import { I18NextPipe } from 'angular-i18next';
import { UserRole } from '@abus-cloud-frontend/auth';
import { DeviceAnalysisService } from './device-analysis';
import { DeviceAnalysisConfig } from './device-analysis/types';

@Injectable({
    providedIn: 'root'
})
export class DeviceActionsProvider {

    constructor(
        private i18n: I18NextPipe,
        private deviceAnalysis: DeviceAnalysisService
    ) {}

    getDeviceAnalysisAction(config: DeviceAnalysisConfig): DeviceAction {
        return {
            // Text des Buttons
            title: this.i18n.transform('common.action.open_device_analysis'),
            // Callback-Funktion, die ausgefuehrt wird, wenn der Button geklickt wird
            callback: async () => await this.deviceAnalysis.openAnalysisDialog(config),
            // Rollen, die berechtigt sind, diese Aktion auszufuehren
            roles: [
                UserRole.TechnicalSupport,
                UserRole.ProductManager
            ]
        }
    }

    // Andere Aktionen sind nicht relevant und wurden hier weggelassen (inkl. ihre
        Importe und Abhaengigkeiten)
}
```

Quellcode 2: Provider von `DeviceActions`

```typescript
import { Injectable } from '@angular/core';
import { I18NextService } from 'angular-i18next';
import { lastValueFrom } from 'rxjs';
import { SecorisService } from '@abus-cloud-frontend/api';
import { DeviceAnalysisDeviceSpecificData, DeviceAnalysisDeviceSpecificDataProvider }
    from '@abus-cloud-frontend/device-common';

@Injectable({
    providedIn: 'root'
})
export class SecorisDeviceAnalysisProvider implements
    DeviceAnalysisDeviceSpecificDataProvider {
    constructor(
        private secorisService: SecorisService, // Secoris API Service
        private i18n: I18NextService // Internationalisierung
    ) {}

    async getDeviceAnalysisInfo(deviceId: string): Promise<
        DeviceAnalysisDeviceSpecificData[]> {
        // Holen der Geraeteeinstellungen von der API fuer die gegebene deviceId
        const settings = await lastValueFrom(
            this.secorisService.SettingsDeviceIdGet(deviceId)
        );

        let connectionType;
        if (settings.connectionType != undefined) {
            // Verbindungstyp grossschreiben
            connectionType = settings.connectionType[0].toUpperCase() + settings.
                connectionType.slice(1).toLowerCase()
        }

        // Array mit den geraetespezifischen Daten zurueckgeben
        return [
            {
                title: this.i18n.t('devices.secoris.sidebar_device.serial_number').
                    toUpperCase(),
                content: settings?.serialNumber
            },
            {
                title: this.i18n.t('devices.secoris.sidebar_device.ip_address').
                    toUpperCase(),
                content: settings?.internalIpAddress
            },
            {
                title: this.i18n.t('devices.secoris.sidebar_device.mac_address').
                    toUpperCase(),
                content: settings?.macAddress
            },
            // ...weitere Daten abgekuerzt
        ];
    }
}
```

Quellcode 3: Implementierung des `DeviceAnalysisProvider` für Secoris

```
// using-Anweisungen aus Platzgruenden weggelassen

namespace ABUS.Cloud.Portal.Common.Tests.DeviceAnalysis
{
  public class DefaultDeviceAnalysisRepositoryTests : IAsyncLifetime
  {
    // Fixture = gemeinsame Testumgebung fuer Integrationstests
    private readonly IntegrationTestFixture _fixture;
    // Das zu testende Repository
    private readonly IDeviceAnalysisRepository _repository;
    private static readonly Guid _deviceId = Guid.NewGuid();
    private static readonly DateTime _now = DateTime.UtcNow;
    // die folgenden englischen Kommentare sind auch im Originalcode vorhanden
    private readonly MessageSinkEntry[] _testingData = [
      // Last year
      CreateMessage(_deviceId, "type-a", _now - TimeSpan.FromDays(200)),
      CreateMessage(_deviceId, "type-a", _now - TimeSpan.FromDays(59)),
      CreateMessage(_deviceId, "type-b", _now - TimeSpan.FromDays(32)),
      // Last month
      CreateMessage(_deviceId, "type-b", _now - TimeSpan.FromDays(27)),
      CreateMessage(_deviceId, "type-b", _now - TimeSpan.FromDays(8)),
      // Wrong messages
      CreateMessage(Guid.NewGuid(), "wrong-device", _now - TimeSpan.FromMinutes(1))
    ];

    public DefaultDeviceAnalysisRepositoryTests(IntegrationTestFixture fixture)
    {
      _fixture = fixture;

      // Initialisierung des Repositories ueber Dependency Injection der Test-Fixture
      _repository = _fixture.DefaultServiceProvider
        .GetKeyedService<IDeviceAnalysisRepository>(DeviceAnalysisConstants.
            DEFAULT_REPOSITORY_KEY);
    }

    // ... Test-Setup und Teardown-Methoden ...

    [Fact]
    public async Task Should_get_less_message_statistics_for_last_month()
    {
      // Arrange
      var from = _now - TimeSpan.FromDays(30);

      // Act
      var result = await _repository.GetMessageStatisticsAsync(_deviceId, from, _now);

      // Assert
      // Nur "type-b" Nachrichten im letzten Monat
      result.Should().HaveCount(1);
      // Falsche Geraete-ID soll nicht enthalten sein
      result.Should().NotContain(stat => stat.MessageType == "wrong-device");
      // "type-a" Nachrichten sollen nicht enthalten sein (aelter als 30 Tage)
      result.Should().NotContain(stat => stat.MessageType == "type-a");
      // Es gibt 2 "type-b" Nachrichten im letzten Monat
      result.First(stat => stat.MessageType == "type-b").Count.Should().Be(2);
    }

    // ... weitere Tests ...

    private static MessageSinkEntry CreateMessage(Guid deviceId, string messageType,
        DateTime enqueueTimeUtc)
    {
      // Methodenimplementierung aus Platzgruenden weggelassen.
    }
  }
}
```

Quellcode 4: Integrationstest der Default-Implementierung des `IDeviceAnalysisRepository`