Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Anja **Stricker**

Entwurf und Implementierung eines konfigurierbaren Alarmsystems für Monitoring-Systeme unter Berücksichtigung von Hochsicherheitsbereichen

Design and implementation of a configurable alarm system for monitoring systems respecting high security areas

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Anja **Stricker**

Entwurf und Implementierung eines konfigurierbaren Alarmsystems für Monitoring-Systeme unter Berücksichtigung von Hochsicherheitsbereichen

Design and implementation of a configurable alarm system for monitoring systems respecting high security areas

Bearbeitungszeitraum: von 25. Juli 2022
bis 23. Dezember 2022

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dr.-Ing. Ulrich Schäfer

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Stricker, Anja**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Entwurf und Implementierung eines konfigurierbaren Alarmsystems für Monitoring-Systeme unter Berücksichtigung von Hochsicherheitsbereichen

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 12. Dezember 2022

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin (Name, Vorname): **Stricker, Anja**
Studiengang: Medieninformatik
Aufgabensteller, Professor: Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in (Firma): Mühlbauer GmbH & Co. KG
Betreuer in Firma: Alexander Bauer
Ausgabedatum: 25. Juli 2022
Abgabedatum: 23. Dezember 2022

Titel:

Entwurf und Implementierung eines konfigurierbaren Alarmsystems für Monitoring-Systeme unter Berücksichtigung von Hochsicherheitsbereichen

---

Zusammenfassung:

Die Bedeutung und der Nutzen von Echtzeitdaten in der Industrie ist in den letzten Jahren erheblich gestiegen. Durch das Sammeln und Auswerten dieser Daten können industrielle Prozesse überwacht und effizient kontrolliert werden. Dies ermöglicht eine Vielzahl an Vorteilen wie beispielsweise schnellere Fehlerbehebungen, vermiedene Ausfallzeiten und Kosteneinsparungen. Diese Vorteile können durch die Entwicklung eines Alarmsystems ermöglicht werden. Die vorliegende Arbeit beschäftigt sich deshalb mit dem Entwurf und der Implementierung eines konfigurierbaren Alarmsystems. Außerdem wird der Sicherheitsstandard von Hochsicherheitsbereichen berücksichtigt, um das Alarmsystem auch in diesem Umfeld einsetzen zu können.

Ziel dieser Arbeit ist, eine Grundlage für ein unabhängiges und effektives Alarmsystem zu bilden. Dieses soll durch den modularen Aufbau um beliebige Kommunikationsmethoden zur Alarmierung erweitert werden können. Zudem werden grundlegende Terminologien effektiver Alarmierung erläutert. Um den Anforderungen sicherer und reaktiver Alarmierung gerecht zu werden, wird eine ausführliche Analyse zu geeigneten Alarmierungsmethoden durchgeführt. In dieser werden die Methoden in einer Nutzwertanalyse verglichen, um den Nutzen der einzelnen Methoden auflisten zu können. Abschließend wird am Beispiel der Smart Factory Softwarelösung MB PALAMAX® Pico die Implementierung des Alarmsystems beschrieben und getestet.

Schlüsselwörter: Alerting, Notification, WebSocket, E-Mail, SIP, React.js, C#

Title:

Design and implementation of a configurable alarm system for monitoring systems respecting high security areas

Abstract:

The importance and benefits of real-time data in the industry have increased significantly over the last few years. By collecting and evaluating this data, industrial processes can be monitored and controlled efficiently. This enables a variety of benefits such as faster troubleshooting, avoided downtime, and cost savings. These benefits can be achieved through the development of an alarm system. Therefore, this thesis deals with the design and implementation of a configurable alarm system. In order to be able to use the alarm system in high security areas, the safety standard of these areas is also taken into account.

The aim of this work is to form a basis for an independent and effective alarm system. Thanks to its modular structure, it should be possible to expand the system by any communication methods for alerting. Fundamental terminologies of effective alerting are also explained. In order to meet the requirements of secure and reactive alerting, a detailed analysis of suitable alerting methods will be performed. The methods are compared with each other in order to be able to list the benefits of the respective methods. Finally, the implementation of the alarm system is described and tested using the MB PALAMAX® Pico smart factory software solution as an example.

Keywords: Alerting, Notification, WebSocket, E-Mail, SIP, React.js, C#

Inhaltsverzeichnis

Abkürzungsverzeichnis

MQTT Message Queuing Telemetry Transport

TTS Text-to-Speech

SMS Short Message Service

GSM Global System for Mobile Communications

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

IETF Internet Engineering Task Force

APNs Apple Push Notification service

FCM Firebase Cloud Messaging

REST Representational State Transfer

POC Proof of Concept

VoIP Voice over IP

CLI Command Line Interface

SDK Software Development Kit

TCP Transmission Control Protocol

HTTP Hypertext Transfer Protocol

UAC User Agent Client

UAS User Agent Server

SDP Session Description Protocol

JSON JavaScript Object Notation

JWT *JavaScript Object Notation (JSON)* Web Token

UDP User Datagram Protocol

DTMF Dual Tone Multifrequency

CRUD Create, Read, Update, Delete

API Application Programming Interface

DLL Dynamic Link Library

Kapitel 1

Einleitung

Am 26. April 1986 um 01:23 Uhr ereignete sich die Nuklearkatastrophe in Tschernobyl. Ein Desaster, welches weltweit gesundheitliche Langzeitfolgen mit sich brachte, die heute noch spürbar sind. Durch grob fahrlässige Fehler der Betriebsmannschaft geriet ein Reaktor bei einem bis dahin nicht durchgeführten Inbetriebsetzungsversuch außer Kontrolle und explodierte. [1, S. 83]

Doch dies ist nicht das einzige Desaster, welches durch Fehleinschätzungen von Menschen erzeugt wurde. Ein weiteres sehr bekanntes Ereignis in der Industrie ist beispielsweise das der Piper Alpha, eines der größten produzierenden Öl-Plattformen in der Nordsee. Im Jahre 1988 wurde dieses durch eine Explosion zerstört, welche durch das Ausführen von nicht abgesprochenen gleichzeitigen Wartungsarbeiten ausgelöst wurde. [2, S. 103]

Es gibt noch zahlreiche weitere Unfälle, die auf das Versagen von Menschen zurückzuführen sind. Das hohe Unfallrisiko kann jedoch durch den Einsatz eines effektiven Monitoring- und Alarmsystems vermindert werden [3, S. 2].

Am Beispiel des Tschernobyl Ereignisses hätte das bestehende Sicherheitssystem die Einhaltung der betrieblichen Reaktivitätsreserve überwachen können. Dieser Wert hätte gezeigt, dass der Reaktor abgeschaltet werden müsste, da dieser den Minimalwert unterschritten hatte. Dies wird auch im Journal „Der Unfall von Tschernobyl 1986“ [1, S. 86] erwähnt, welches zur Aufarbeitung des Tschernobyl Ereignisses dient. Dieser beschreibt neben den Fakten zum Unfallablauf und den Unfallursachen auch eine Auflistung an Ertüchtigungsmaßnahmen für alle Reaktoren gleichen Typs. Unter dem Punkt „Maßnahmen zur Verbesserung der Betriebsführung“ wird unter anderem eine Alarmierung beim Unterschreiten des zulässigen ORM-Wertes empfohlen. „Der ORM-Wert ist das Reaktivitätsäquivalent aller [...] in den Kern eingefahrenen Steuerstäbe. Er wird als Vielfaches des Reaktivitätsäquivalents eines mittleren, voll eingefahrenen Steuerstabes angegeben. “ [1, S. 82]. Durch die gezielte Alarmierung zu diesem Wert, hätte die Betriebsmannschaft das Problem erkennen und den Versuch abbrechen können.

Um die Sicherheit in der Industrie immer weiter verbessern zu können, ist es wichtig

von bereits geschehenen Fehlern zu lernen und Präventionsmaßnahmen zu ergreifen. Unter anderem ist deswegen in den letzten Jahren die Bedeutung und der Nutzen von Echtzeitdaten in der Industrie erheblich gestiegen. Der gezielte Einsatz dieser Daten ermöglicht eine Vielzahl an Vorteilen. Durch effiziente Überwachung von Produktionen können Fehlerquellen an Maschinen schneller erkannt und behoben werden, was zu einer verlängerten Lebensdauer der Maschinen, vermiedenen Ausfallzeiten und Kosteneinsparungen führt.

Unter dem Begriff der Überwachung in Bezug auf Industriedaten versteht man den Prozess, sich über den aktuellen Systemzustand zu informieren. Dabei unterscheidet man zwischen der proaktiven und der reaktiven Überwachung [4, S. 1]. Die proaktive Überwachung bietet eine visuelle Darstellung des Systems und wird meist unter dem Begriff „Monitoring“ verbunden. Die reaktive Überwachung befasst sich mit der automatisierten Benachrichtigung oder Alarmierung, um über gravierende Änderungen im System benachrichtigt zu werden. Eine proaktive Überwachung kann Aufschluss darüber geben, wie Produkte effizienter hergestellt werden können. Ein Produktionsstillstand kann mithilfe von reaktiver Überwachung durch gezielte Alarmierung schnellere Reaktionen erzielen und somit größere Kosten verhindern.

Die Implementierung einer solchen Überwachung bringt viele Herausforderungen mit sich. Die vorliegende Arbeit gibt darüber Aufschluss, was die Kerndisziplinen der effizienten und reaktiven Überwachung sind, wie diese umgesetzt und wie bekannte Herausforderungen in der Alarmierung gelöst werden können. Außerdem soll die Nutzung des Alarmsystems am Beispiel eines bestehenden proaktiven Monitoring-Systems beschrieben werden.

1.1 Motivation

Die Firma Mühlbauer GmbH & Co. KG in Roding ist einer der führenden Technologiepartner für Smart Cards, elektronische Pässe, RFID und Solar-Backends. Zu den weiteren Geschäftsfeldern gehören das Sortieren von Mikrochips, Carrier Tape Anlagen sowie Automatisierungs-, Kennzeichnungs- und Traceability-Systeme. Der betriebsinterne Geschäftsbereich für Präzisionsteile und Systeme stellt hochpräzise Komponenten her, sowohl für die Produktion seiner eigenen Produkte als auch für andere Hersteller in sensiblen Industrien, wie z. B. Luft- und Raumfahrt, Motorsport, Medizin- und Halbleiterindustrie.

Das Portfolio der Mühlbauer GmbH & Co. KG umfasst neben Maschinen auch eigens entwickelte Softwarelösungen. Eine dieser Softwarelösungen ist MB PALAMAX®, eine Smart Factory Solution im Bereich Industrie 4.0, welche von einer internen Softwareabteilung entwickelt wird und zur Überwachung von Produktionsumgebungen dient. Die Software ermöglicht die Erfassung kompletter Produktionsdaten in Echtzeit. Durch Auswertung gesammelter Daten kann die Verfügbarkeit, Leistung, Qualität und Effektivität einer gesamten Produktionshalle gesteigert werden. Eine weitere Softwarelösung ist MB PALAMAX® Pico, welche die wichtigsten Kernfunktionen der MB PALAMAX® Software beinhaltet, und für weniger leistungsfähige Computer und mobile Geräte

entwickelt worden ist.

Um den höher werdenden Anforderungen hinsichtlich der Maschinenverfügbarkeit in der Industrie nachkommen zu können, müssen Überwachungssysteme um reaktive Alarmierungen ergänzt werden. Der Sinn einer Alarmierung ist die Erhöhung der Aufmerksamkeit auf ein Ereignis, welches eine Reaktion erfordert. In der Industrie hat die Schnelligkeit der Reaktion einen hohen Wert. Probleme, die nicht schnell beseitigt werden, können zu Kostensteigerungen und Gewinnminderung führen.

Diese Arbeit befasst sich mit der Frage, wie eine effiziente und reaktive Alarmierung in mehreren unterschiedlichen Monitoring-Systemen eingebaut werden kann. Die Umsetzung einer solchen Lösung soll jedoch ohne externe Dienste möglich sein, damit auch Systeme in autarken, unabhängigen Hochsicherheitsbereichen das Alarmsystem benutzen können. Neben der Evaluation von geeigneten Kommunikationswegen wird auch die Konzeption und Implementierung eines reaktiven Alarmsystems beschrieben. Zusätzlich wird, am Beispiel des bestehenden Monitoring-Systems MB PALAMAX® Pico, die Integration des Alarmsystems geschildert. Dies umfasst neben der Konzeption der Erweiterung auch die anschließende Umsetzung für das Monitoring-System, um das Alarmsystem voll umfassend verwenden zu können.

1.2 Ist-Analyse

In der Mühlbauer GmbH & Co. KG existieren zwei Arten von Monitoring-Systemen. Sowohl für die industrielle Maschinenproduktion, als auch eines für die Überwachung von IT Systemen. Da das Alarmsystem so konzipiert werden soll, dass es von mehreren unterschiedlichen Monitoring-Systemen genutzt werden kann, eignet sich die Betrachtung beider Varianten in der Ist- und Soll-Analyse.

Systemüberwachung

Der Ausfall von internen Anlagen oder Servern führt zur negativen Beeinflussung von geschäftlichen Prozessen. Um solche Ausfälle rechtzeitig erkennen und lösen zu können, besitzt die IT-Abteilung zur Überwachung dieser Systeme ein Monitoring-System. Dieses ist in der Lage, den zuständigen Mitarbeitern mittels E-Mail, eine Benachrichtigung über den Ausfall zu senden.

E-Mails können nur über Mitarbeiter Notebooks abgerufen werden oder über eine Weboberfläche. Der Nutzen von E-Mail Applikationen auf Smartphones ist nicht erlaubt und ermöglicht so keine auditive Signalisierung über den Empfang einer neuen E-Mail. Während Mitarbeiter im Gebäude unterwegs sind, können diese demnach nicht über den Empfang einer wichtigen E-Mail informiert werden.

Industrielle Produktionsüberwachung

Aufkommende Störungen an Maschinen können visuell über Ampelsysteme erkannt werden. Diese sind an den Maschinen fest montiert und können mit den Farben Grün, Gelb und Rot drei unterschiedliche Status anzeigen. Auch die Erkennung von

Störungen über die Weboberfläche von MB PALAMAX® oder MB PALAMAX® Pico ist möglich. Ein beispielhafter Ausschnitt der Weboberfläche von MB PALAMAX® Pico kann im Anhang A.2 gesichtet werden. In der Live-Daten-Ansicht der Weboberfläche können aktuell auftretende Fehler in der „Active Errors“ Karte überwacht werden.

Über die Maschinensoftware kann der genaue Status der Maschine eingesehen werden, sowie die detaillierte Fehlerbeschreibung und -herkunft. Die Abbildung 1.1 zeigt die Maschinensoftware, in welchem ein offener Fehlerdialog zu sehen ist.

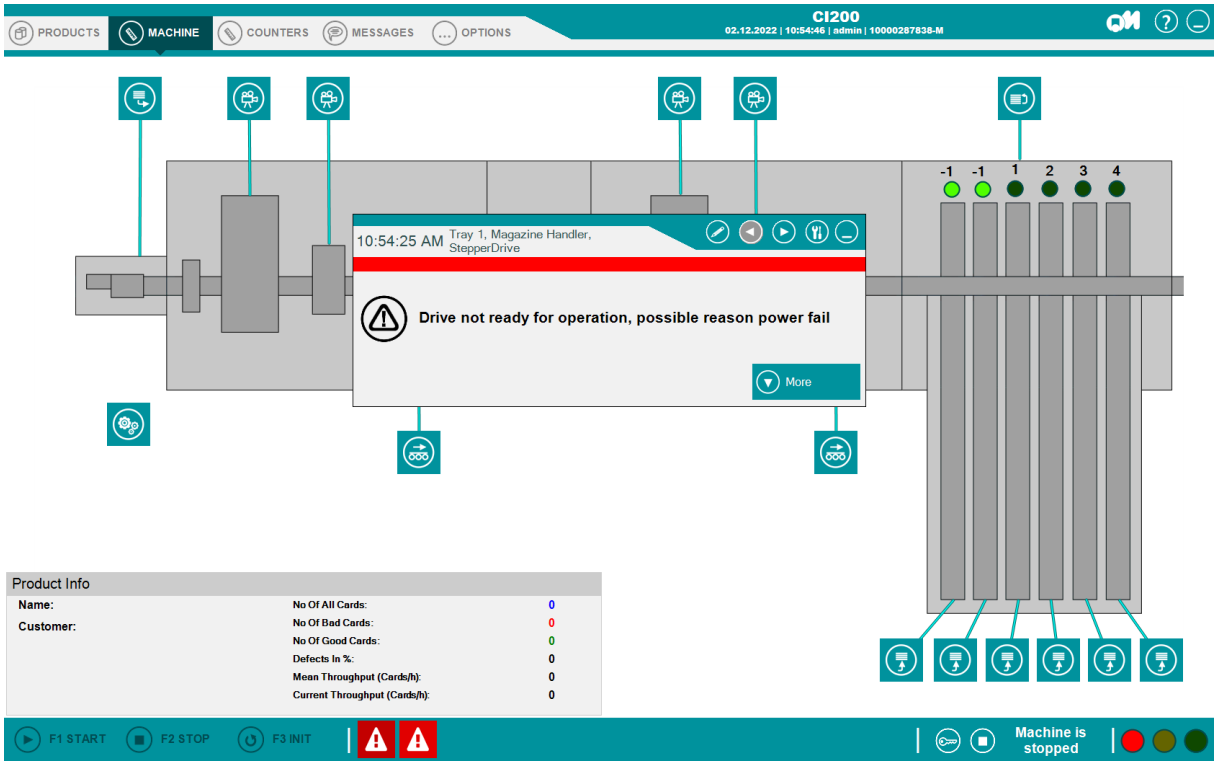


Abbildung 1.1: Ausschnitt der Maschinensoftware mit offenem Fehlerdialog

Da die Maschinen unabhängig produzieren können, wird die ständige Anwesenheit eines Bedieners nicht benötigt. Aus diesem Grund befinden sich nicht immer Personen in Reichweite, um Störungen frühzeitig erkennen und darauf reagieren zu können. Ebenso ist keine permanente Überwachung über die Weboberfläche möglich, da dies enorme Mitarbeiterkosten zur Folge hätte.

1.3 Anforderungskontext

Es soll ein Alarmsystem entwickelt werden, welches die Herausforderung von reaktiver Überwachung lösen und von beliebigen Monitoring-Systemen genutzt werden kann. Die Anwendungsfälle sind hierbei die globale Systemüberwachung und die industrielle Produktionsüberwachung. Einige Kunden der Firma Mühlbauer GmbH & Co. KG besitzen Produktionshallen in Hochsicherheitsbereichen, weshalb die Nutzung von externen Diensten streng untersagt ist. Die Herausforderung in dieser Arbeit liegt

darin, ein internes System zu entwickeln, welches die internen Kommunikationsmöglichkeiten effizient nutzen kann, ohne den Zugriff auf externe Dienste.

In einer IT-Abteilung müssen unter anderem alle Server und Telefonanlagen überwacht werden, welche zum Betrieb im Unternehmen essenziell sind. Bei Ausfällen muss schnell gehandelt werden, da sonst der Arbeitsablauf der Mitarbeiter und der generelle Betrieb beeinträchtigt werden. Daher ist im Falle eines Ausfalls die schnelle Reaktion eines IT-Mitarbeiters nötig, um die negativen Folgen zu minimieren. Wenn der Empfänger des Alarms jedoch nicht erreichbar ist, soll die Alarmierung eine andere erreichbare Person finden, bis sich jemand dem Problem annehmen kann. Mit dieser Funktionalität soll das Alarmsystem dabei unterstützen, schnelle Reaktionen bei besonders schwerwiegenden Ereignissen zu erzielen, um die daraus resultierenden Probleme schneller beseitigen zu können. Für diese Art der Alarmierung muss ein geeigneter Kommunikationsweg evaluiert werden.

Um die Alarmierung in Hochsicherheitsbereichen gewährleisten zu können, wird die Nutzung von externen Diensten nicht gestattet. Das System darf lediglich auf internen Servern betrieben werden und trotz dieser Einschränkungen eine effektive Alarmierung gewährleisten. Auch für diesen Anwendungszweck soll es möglich sein, die Reaktionsgeschwindigkeit zu steigern, um lange Maschinenausfälle zu vermeiden.

Außerdem soll das Alarmsystem eine Schnittstelle bereitstellen, über welches ein Monitoring-System die Informationen über laufende Alarmierungen abfragen kann. Mit dieser Funktionalität wäre es einer leitenden Person möglich, die Alarmierungen kontrollieren und koordinieren zu können, falls diese nicht von einem zuständigen Mitarbeitenden wahrgenommen werden. Diese Funktionalität ist für beide Anwendungsfälle nützlich, um einen Überblick über ausgefallene Systeme zu erhalten.

Kapitel 2

Grundlagen

In diesem Kapitel werden grundlegende Begriffe definiert, welche im Folgenden die Grundlagen der Arbeit bilden.

2.1 Projektbezeichnungen

Die Ausarbeitung dieser Bachelorarbeit befasst sich mit zwei unterschiedlichen Anwendungen, welche mehrmals erwähnt und daher kurz in diesem Abschnitt vorgestellt werden.

2.1.1 MB PALAMAX® Pico

MB PALAMAX® Pico ist eine von Mühlbauers Smart Factory Softwarelösungen, welche die Erfassung von Produktionsdaten in Echtzeit ermöglicht. Das modulare Softwaresystem ermöglicht sowohl die Analyse als auch die Steigerung der Verfügbarkeit, Leistung, Qualität und Effektivität. Dabei kann entweder eine einzelne Maschine oder die gesamte Produktionsstätte überwacht werden.

Die MB PALAMAX® Pico Smart Factory Software bietet eine effiziente Schnittstelle für Manager und Bediener, die mit der Prozessüberwachung und der statistischen Analyse des Produktionsprozesses beauftragt sind. Außerdem unterstützt sie die Planung zur Optimierung und Kontrolle der Kostentreiber im Produktionsprozess. Die Software kann sich an weniger leistungsstarke Computer anpassen, indem nicht benötigte Kalkulationen in einer Konfiguration deaktiviert werden.

Das Backend dieser Anwendung wurde in C# mit ASP.Net Core entwickelt, das Frontend mit der React.js Bibliothek erstellt. Die Maschinenwerte werden von einem eigenständigen System mit dem Namen MB PALAMAX® Bridge über das Protokoll *Message Queuing Telemetry Transport (MQTT)* versendet. Sie dient als Kommunikationsschnittstelle und wurde ebenfalls von einem internen Softwareteam in der Mühlbauer GmbH & Co. KG entwickelt. Die Daten von der MB PALAMAX® Bridge werden von MB PALAMAX® Pico empfangen, verarbeitet und in einer PostgreSQL Datenbank

gesichert. In der Weboberfläche werden diese Daten entsprechend visualisiert. Zudem besitzt die Anwendung die Möglichkeit, einen Bericht für einen ausgewählten Zeitraum an Daten zu erstellen. Abbildung 2.1 zeigt einen, mit Testdaten befüllten, Ausschnitt der Statistik-Seite von Webanwendung MB PALAMAX® Pico. Eine weitere Abbildung, welche die Live-Daten-Ansicht darstellt, ist im Anhang A.2 zu finden.

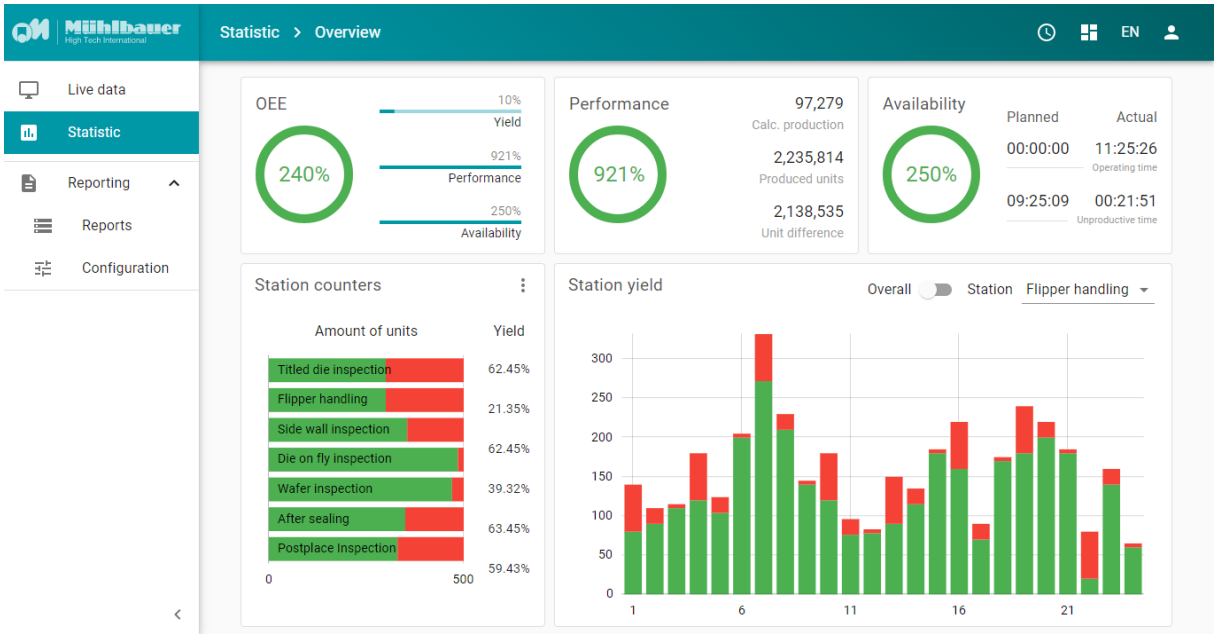


Abbildung 2.1: Ausschnitt der MB PALAMAX® Pico Statistik-Daten Ansicht, welche mit Testdaten befüllt ist.

Diese Anwendung ist nicht als Teil der Bachelorarbeit erarbeitet worden. Im Rahmen dieser Arbeit wird das System lediglich um neue Funktionalitäten erweitert, um das Alarmsystem nutzen zu können.

2.1.2 Alarmsystem

In dieser Ausarbeitung soll für die Nutzung reaktiver Alarmierung eine eigenständige Software entwickelt werden, welche in den folgenden Abschnitten als das „Alarmsystem“ bezeichnet wird. Um dieses System entwickeln zu können, wird in dieser Arbeit neben einer Recherche über mögliche Kommunikationswege, auch die Konzeption und Umsetzung für die Entwicklung eines reaktiven Alarmsystems beschrieben.

2.2 Alarm

Um ein Alarmsystem umsetzen zu können, müssen die grundlegenden Aspekte eines Alarms bekannt sein, welche in diesem Abschnitt beschrieben werden. Neben Definitionen von speziellen Terminologien werden hier ebenfalls Möglichkeiten zur Alarmierungen aufgezählt und erläutert.

2.2.1 Begriffsdefinition

Das Ziel einer Alarmierung ist, die Aufmerksamkeit auf eine spürbare Leistungsverschlechterung zu lenken. Dabei soll im angemessenen Zeitrahmen reagiert werden, um die Quelle des Problems zu isolieren und identifizieren. Die damit verbundenen Auswirkungen sollen so schnell wie möglich vermindert werden.

Slawek Ligus beschreibt in seinem Buch „Effective Monitoring and Alerting“ drei Hauptgründe, durch die eine Benachrichtigung erfolgt: [4, S. 47]

- Abnehmende Qualität des Produktes
- Verlust der Maschinenverfügbarkeit
- Reaktionszeit erhöhen

Um die abnehmende Qualität eines Produktes wahrnehmen zu können, ist die Überwachung einzelner Schwellenwerte nötig. Sobald ein Wert oder eine Kombination von Werten nicht mehr im normalen Bereich liegen, soll eine Alarmierung gesendet werden.

Die Maschinenverfügbarkeit kann durch die Überwachung des Maschinenstatus kontrolliert werden. Sobald dieser beispielsweise in einem Fehlerzustand ist, kann die Benachrichtigung dabei helfen, einen Maschinenbediener auf das Problem aufmerksam zu machen, um es zu beheben.

Wenn kein Alarmierungs- oder Monitoring-System besteht, können Maschinenbediener nur durch manuelle Kontrolle auf Probleme aufmerksam gemacht werden. Damit schwerwiegende Probleme rechtzeitig gelöst werden, ist die Erhöhung der Reaktionszeit besonders wichtig.

2.2.2 Cry Wolf Effect

Nicht alle Fehler verdienen die Aufmerksamkeit eines Bedieners. Würde jede einzelne Nachricht als Alarm behandeln werden, wäre kein Überblick mehr über die Relevanz und Priorisierung der Fehler möglich. Bei besonders großen Datenmengen kann es schnell zu Fehlalarmen führen, welche dem Nutzer fälschlicherweise gemeldet werden. Es kommt zum sogenannten „Cry Wolf Effect“.

Der „Cry Wolf Effect“ bezieht sich auf die Tendenz der Nutzer, der Automatisierung oder den Fehlermeldungen nicht mehr zu vertrauen, nachdem es sich wiederholt um Fehlalarme gehandelt hat. Der Begriff ist abgeleitet von der Fabel „Der Hirtenjunge und der Wolf“, in welcher der Junge einen Wolf nachahmte. [5]. Hier ist die Automatisierung der Junge und die Benutzer sind die Dorfbewohner. Nach den wiederholten Fehlalarmierungen weigerten sich die Dorfbewohner weiterhin auf die Rufe des Jungen zu reagieren. Infolgedessen kam keiner zur Hilfe, als tatsächlich ein Wolf die Schafe angegriffen hat.

Wenn Schwellenwerte für Alarmierungen nicht richtig konfiguriert wurden, kann es manchmal zu Ausreißern kommen, welche einen Fehlalarm auslösen. Wenn dieses

Szenario sehr häufig am Tag auftritt, ist der Empfänger schnell dazu verleitet, die Alarmierung zu ignorieren. Daher ist es wichtig, Schwellenwerte bedacht zu setzen oder diese in Aggregation mit anderen Werten zu verwenden, um das Risiko einer Fehlalarmierung zu senken. Diese Probleme können demnach mit einer angepassten Alarmkonfiguration vermindert werden.

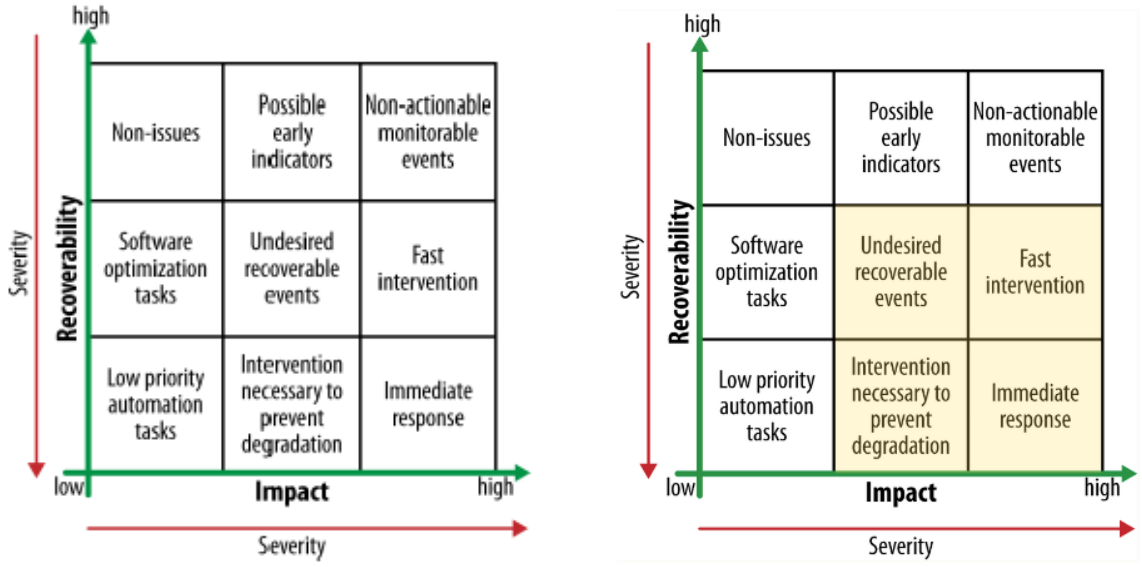
2.2.3 Klassifizierung nach Schweregrad

Um dennoch eine mögliche Informationsflut an Alarmierungen kontrollieren zu können, muss zur jeweiligen Klassifizierung ein sinnvoller Kommunikationsweg genutzt werden, um die Reaktion zur richtigen Zeit zu erhalten. Darauf weist auch der Autor Slawek Ligus in seinem Buch hin [4, S. 49], und klärt auf, wie diese umgesetzt werden kann. Dafür muss zunächst eine geeignete Klassifizierung erstellt werden. Der Autor empfiehlt hier die Betrachtung zweier Ausprägungen eines Alarms:

Die Wiederherstellbarkeit (Recoverability) beschreibt das Potenzial eines Systems, seinen Zustand vor dem Ausfall ohne Eingreifen des Bedieners wiederherzustellen. Die Wiederherstellbarkeit bemisst sich nach der Wahrscheinlichkeit, dass ein bestimmter Fehler behoben wird und die für seine Behebung erforderliche Zeit.

Die Auswirkung (Impact) beschreibt den negativen Effekt auf den Betrieb eines Systems, welcher sich in der Ressourcennutzung und der Erfahrung des Endbenutzers widerspiegelt. Die Auswirkung bemisst sich nach unvorhergesehene Kosten, die in verschiedenen Größenordnungen anfallen.

Der Autor hat mithilfe dieser zwei Hauptausprägungen die folgende Klassifizierungsmatrix definiert (siehe Abbildung 2.2a).



(a) Klassifizierungsmatrix von Slawek Ligus

(b) Klassifizierungsmatrix mit markierten Alarmierungsbereich

Abbildung 2.2: Die Klassifizierungsmatrix von Slawek Ligus [4, S. 49] zeigt verschiedene Klassifizierungen von Ereignissen und wie diese interpretiert werden können

Nicht alle Bereiche in dieser Matrix benötigen eine Alarmierung. Die wichtigsten vier Felder befinden sich im unteren rechten Bereich (siehe Abbildung 2.2b). Durch Einhalten dieser Klassifizierungen ergibt sich der Vorteil, dass nur wichtige Ereignisse zu Alarmierungen werden und somit einer Informationsflut entgegenwirken. Je mehr Klassifizierungen erstellt werden können, desto deutlicher können Alarmierungen zugeordnet werden.

2.2.4 Alarmierungsmethoden

In der heutigen Zeit gibt es viele unterschiedliche Kommunikationswege, um alarmiert zu werden. Jeder Kommunikationsweg hat Vor- und Nachteile, welche in der folgenden Auflistung näher erklärt werden.

Ampelsystem Unter den visuellen Alarmen gibt es aktuell nur eine Variante, die in der Industrie weit verbreitet ist. Dabei handelt es sich um die Alarmierung über ein Ampelsystem. Ampelsysteme helfen Bedienern und Prozessmanagern, den aktuellen Maschinenstatus direkt erkennen zu können. Die drei Farben (Grün, Gelb, Rot) stehen hierbei oft für „Running“, „Warning“ und „Error“. Einige Ampelsysteme besitzen zusätzlich ein blaues Farbelement für die Anzeige „Service“.

Um diese Statusmeldung sehen zu können, muss man sich jedoch physikalisch in der Nähe der Maschine befinden. Im Falle von besonders großen Produktionsumgebungen oder mehreren Etagen kann ein aufgetretener Fehler im „Error“ Status verzögert wahrgenommen werden.

Als Lösung für dieses Problem eignet sich der Nutzen von zusätzlichen Alarmierungsmethoden, welche beispielsweise über kompakte Geräte, wie beispielsweise einem Pager oder Smartphone, empfangen werden können.

SMS Die Alarmierung über *Short Message Service (SMS)* basiert auf der textuellen Alarmierung. Die Alarmierung kann dem Empfänger eine kurze Information über das Ereignis mitteilen. Durch den Empfang mit Ton oder Vibration wird zudem die Aufmerksamkeit auf das Ereignis erhöht. Da *SMS* über mobile Geräte empfangen werden, kann der Empfänger zu jeder Zeit an einem beliebigen Ort alarmiert werden.

Das Versenden von *SMS* ist mit relativ geringen Kosten verbunden, jedoch wird hierfür meist ein externer Dienst benötigt. Eine offline Lösung wäre nur mit einem *Global System for Mobile Communications (GSM)* Modem möglich, welches sich jedoch mit einem öffentlichen Funkmast verbindet und somit den Anforderungen eines Hochsicherheitsbereiches nicht mehr standhalten kann.

E-Mail Die häufigste Form von Benachrichtigung ist eine E-Mail Nachricht, da sie keine Kosten verursacht. Das Versenden und Empfangen von E-Mails kann sowohl in Online- als auch Offline-Systemen problemlos erfolgen. Da dieses Medium jedoch im Leben eines Menschen täglich im Gebrauch ist, tritt der „Cry Wolf Effect“ sehr oft auf. Das bedeutet, dass die empfangenen Informationen übersehen

oder ignoriert werden können. Deswegen sollten eher Benachrichtigungen, statt konkreten Alarmierungen, auf diesem Übertragungsweg gesendet werden, da diese eher übersehen werden dürfen als schwerwiegende Alarmierungen.

Telefonie Ein Anruf besitzt ähnliche Funktionalitäten wie die einer *SMS*. Der Anruf wird auf einem tragbaren Mobiltelefon empfangen und spielt eine Melodie oder eine Vibration ab, um die Aufmerksamkeit auf sich zu ziehen. Bei einem Telefonat wird die Information über eine Stimme oder Töne übertragen.

Um Informationen ohne die Hilfe einer menschlichen Stimme empfangen zu können, werden Ereignisbeschreibung beispielsweise über *Text-to-Speech (TTS)*. mitgeteilt. Durch Abfangen der Wahltasten besteht zusätzlich die Möglichkeit, eine Antwort zur Alarmierung zu empfangen. Somit wäre es möglich zu erkennen, ob der Empfänger sich der Alarmierung annehmen kann, oder ob dieser an einen anderen Empfänger weitergeleitet werden muss. Letzteres Verfahren nennt man auch „Eskalation“.

Der größte Vorteil dieser Alarmierungsmethode ist die geringe Wartezeit auf eine Antwort. Daher eignet sich diese Methode besonders für die Alarmierung über schwerwiegende Ereignisse. Im Gegensatz zur Übertragung von Informationen über *SMS* können für die Telefonie interne Telefonanlagen benutzt werden, welche die Sicherheit der Übertragung gewährleisten können.

Web Push-Notification Über die bereits bestehenden Monitoring-Systeme könnten „Push-Notifications“ versendet werden, welche jedoch nur während der kontinuierlichen Überwachung der Seite einsehbar sind. Da Maschinenbediener überwiegend in Bewegung sind und nicht an einem Monitoring-System verharren, wäre diese Alarmierungsmethode nicht ausreichend, um eine schnelle Reaktion auf das Ereignis zu erhalten.

Jedoch erweist sich diese Methode in speziellen Fällen als nützlich. Dies ist beispielsweise beim Testlauf einer Maschine der Fall. Hierbei befindet sich die bedienende Person überwiegend an der Maschine, um die Ergebnisse der Maschine zu prüfen. Die „Push-Notification“ erleichtert die Suche nach fehlerhaften Werten und weist effizienter auf das Problem hin.

Mobile Push-Notification Durch den Einsatz von modernen Firmenhandys, wäre es möglich eine eigene mobile Applikation zu entwickeln, welche die Alarmierungsmethoden von *SMS* und Telefonie vereint. Alarmierungen könnten über das interne Netzwerk versendet werden, und die App könnte durch Signaltöne und Vibrationen ebenfalls die Aufmerksamkeit auf das Ereignis erhöhen.

Der Vorteil gegenüber anderen Methoden ist, dass das Ereignis detailreicher an die Empfänger gesendet werden kann. Durch eine Eigenentwicklung kann der Sicherheitsstandard eines Hochsicherheitsbereichs gesichert werden. Zudem bietet es eine Vielzahl an Anwendungsmöglichkeiten an, wie beispielsweise eine mobile Ansicht über die Historie aller aktuellen Alarmierungen. Diese Methode ist somit durchaus empfehlenswert. Der zeitliche Aufwand hinter dieser Neuentwicklung sollte jedoch nicht außer Acht gelassen werden.

2.3 Alarm Konfiguration

Damit ein Alarm konfiguriert werden kann, müssen Bedingungen erstellt werden. Die genaue Definition einer Bedingung, und wie diese kombiniert werden können, wird in den nachfolgenden Abschnitten erklärt.

2.3.1 Definition einer Bedingung

Eine Bedingung vergleicht zwei Werte mit einer gewählten Operation und gibt zurück, ob die Bedingung eingetroffen ist oder nicht. Ein Wert kann entweder ein Echtzeitwert oder ein konstanter Wert, wie eine Zahl oder ein Wort, sein.

Der Vergleich beider Werte kann mit unterschiedlichen Operatoren erfolgen, welche im Folgenden aufgelistet werden.

Vergleich mit logischem **Gleich ==** Operator:

$$Bedingung => Echtzeitwert \texttt{ == } konstante\ Zahl$$

Vergleich mit logischem **Nicht Gleich !=** Operator:

$$Bedingung => Echtzeitwert \texttt{ != } konstante\ Zahl$$

Aggregation mit logischem **Kleiner <** Operator:

$$Bedingung => Echtzeitwert \texttt{ < } Echtzeitwert$$

Aggregation mit logischem **Größer >** Operator:

$$Bedingung => Echtzeitwert \texttt{ > } konstantes\ Wort$$

Ein Vergleich mit zwei konstanten Werten ist dabei nicht sinnvoll und sollte vermieden werden.

Zur Auswertung der Bedingung sind die nötigen Echtzeitwerte und Konstanten benötigt. Für eine durchgehende Überwachung sollte die Auswertung idealerweise bei jeder Echtzeitdatenänderung erfolgen.

2.3.2 Aggregation von Bedingungen

Durch die Aggregation von Bedingungen können Ereignisse, zu denen eine Alarmierung erfolgen soll, genauer spezifiziert werden. Damit werden zudem doppelte und fehlerhafte Alarmierungen vermieden, da sie den genauen Fall der Alarmierung besser beschreiben. Für die Aggregation von binären Ereignisfunktionen können folgende Operatoren verwendet werden.

Aggregation mit logischem **UND**:

$$Alarm = Bedingung1 \wedge Bedingung2$$

Aggregation mit logischem **ODER**:

$$Alarm = Bedingung1 \vee Bedingung2$$

Aggregation mit **beiden** Operatoren:

$$Alarm = Bedingung1 \vee Bedingung2 \wedge Bedingung3$$

2.3.3 Alarmunterdrückung

Im Falle einer Wartung der Produktionslinie können Ereignisse stattfinden, die von dem Überwachungssystem als Alarm gewertet werden können, obwohl das Verhalten beabsichtigt war. Die Alarmierung muss in solchen Fällen temporär abgeschaltet werden, um die Erzeugung von Fehlalarmen entgegenzuwirken.

Dieses Problem könnte bereits während der Konfiguration eines Alarms vermieden werden. Der Nutzer könnte beispielsweise direkt wählen, dass Ausschläge nicht während des Maschinenstatus „Service“ ausschlagen darf.

Vergleich mit logischem **Nicht Gleich !=** Operator:

$$Keine\ Wartung => Maschinenstatus\ \textbf{!=}\ Service$$

Aggregation mit logischem **UND**:

$$Alarm = Bedingung1 \wedge Keine\ Wartung$$

2.4 Schlechte Alarmierungen

Die Wichtigkeit der effizienter Alarmierung wird am Ereignis der Raffinerie Texaco Milforc Haven nochmals deutlich. 1994 standen die dortigen Zuständigen vor 275 unterschiedlichen Alarmen, welche nicht eingeordnet werden konnten. Bei dieser Menge an Alarmierungen ist es nicht möglich, die konkrete Ursache rechtzeitig finden. Nach erfolgloser Fehlersuche kam es schließlich zur Explosion [6, S. 34].

Wären die Alarmierungen besser klassifiziert gewesen, hätten die 275 verschiedenen Alarmierungen besser eingegliedert werden können. Durch genaue Alarm Aggregation wäre es den Zuständigen zudem möglich gewesen, Fehlerquellen besser identifizieren zu können. Dieses Beispiel zeigt zudem auch, dass die Existenz eines Alarmsystems zwar Probleme lösen, aber auch neue erzeugen kann, wenn diese zu Hindernissen werden.

Kapitel 3

Verwandte Arbeiten

Verschiedene andere Arbeiten und Technologien sind mit den in der vorliegenden Arbeit verwendeten Konzepten verwandt. Jedoch deckt keine der nachfolgend aufgelisteten Softwarelösungen alle Anforderungen ab. Dies zeigt, dass die eigene Entwicklung eines Alarmsystems erforderlich ist. Die genauen Unterschiede werden zu jedem Punkt näher geschildert.

3.1 Zenduty

Die Alarmierungsplattform „Zenduty“ bietet eine Alarmierung mit ausführlicher Konfiguration von Bedingungen an. Zudem besitzt es die Möglichkeit, Alarmierungen für gewählte Nutzergruppen eskalieren zu lassen. Für jeden Alarmempfänger in der Nutzergruppe können Erreichbarkeiten konfiguriert werden, damit die Anwendung nur die Personen alarmiert, welche zur Zeit der Alarmierung in Bereitschaft sind.

Um den „Cry Wolf Effect“ zu vermeiden, können mehrere Bedingungen verkettet werden, um die Ereignisse, die zu einer Alarmierung führen, stärker eingrenzen zu können. Die Analysen in der eigenen Weboberfläche bieten zudem einen Überblick über unterschiedliche Statistiken zu vergangenen Alarmierungen.

Neben all den Funktionalitäten bietet diese Anwendung jedoch keine On-Premise Lösung, weshalb sie für den Anforderungskontext dieser Arbeit nicht infrage kommt. Eine On-Premise Lösung ist vor allem zur Einhaltung der Datensicherheit erforderlich und erfüllt somit eine der wichtigsten Anforderungen an das System nicht. [7]

3.2 ALERT von Micromedia International

Die Anwendung mit dem Namen „ALERT“ von Micromedia International bietet eine Alarmierung in Text- und Stimmformaten an. Die Softwarelösung ist On-Premise verfügbar und kann auf eigenen Servern benutzt werden. Zudem bietet es eine umfangreiche Schnittstelle für die Datenerfassung und Alarmweiterleitung.

Die Konfiguration der Alarmierung bietet mit dieser Software keine Lösung für die Prävention eines möglichen „Cry Wolf Effects“. Es können nur einzelne Variablen überwacht werden, weshalb eine Aggregation hier nicht möglich ist. Daher würde diese Anwendung den Anforderungen von reaktiver Überwachung nicht entsprechen. [8]

Um dennoch den großen Funktionsumfang nutzen zu können, wäre es möglich diese Anwendung stattdessen zur Nachrichtenverteilung zu nutzen. Die Verarbeitung der Echtzeitdaten und die Erstellung von komplexen Bedingungen kann durch den Einsatz einer eigenständigen Software geregelt werden, welche anschließend das „ALERT“ System für die jeweilige Alarmierung nutzen kann.

Kapitel 4

Analyse

In diesem Kapitel werden alle Alarmierungsmethoden aus Kapitel 2.2.4 evaluiert, um eine Wahl an geeigneten Kommunikationswegen zu finden, welche zum Bewältigen der gestellten Anforderung an das System benötigt werden. Außerdem werden die Ergebnisse eines durchgeführten *Proof of Concept (POC)* beschrieben, welche die Wahl der gewählten Alarmierungsmethoden bestätigen soll.

4.1 Methodenauswahl zur Alarmierung

Da das Alarmsystem eine Neuentwicklung ist, und im Rahmen der Bachelorarbeit nicht alle möglichen Alarmierungsmethoden implementiert werden können, muss zunächst eine Evaluation durchgeführt werden. In dieser soll eine geeignete Methode angewendet werden, um die Wahl von Alarmierungsmethoden zu unterstützen. Dabei soll die Umsetzbarkeit und der Preis der Methode berücksichtigt werden.

Nachdem eine Auswahl von bestimmten Alarmierungsmethoden getroffen wurde, kann zusätzlich eine Empfehlung für weitere Alarmierungsmethoden gegeben werden, welche nachträglich implementiert werden können. Generell sollte das Alarmsystem so entwickelt werden, dass eine nachträgliche Implementierung einer zusätzlichen Alarmierungsmethode problemlos verläuft.

4.1.1 Bedingungen

Die Mindestanforderung an das System lautet, eine Alarmierungsmethode für hohe Schweregrade und eine für niedrige Schweregrade umzusetzen. Da das System in Hochsicherheitsbereichen eingesetzt werden soll, sollen nur Möglichkeiten in Betracht gezogen werden, die ohne die Hilfe von externen Diensten genutzt werden können. Außerdem werden Methoden mit geringen oder keinen Kosten bevorzugt.

Da das System bereits nach Abschluss der Bachelorarbeit im produktiven Umfeld benutzt werden soll, wird ein großer Wert auf möglichst geringen Implementations- und Konfigurationsaufwand gelegt.

4.1.2 Evaluation

Ziel der Evaluation ist das Finden von Kommunikationswegen, die im Rahmen der Bachelorarbeit implementiert werden können, und zudem einen essenziellen Grundstein für die Alarmierung bilden. Da die Maschinen der Mühlbauer GmbH & Co. KG bereits die Alarmierung über ein Ampelsystem nutzen, wird diese Methode in der Evaluation nicht weiter betrachtet.

Trennen nach Schweregrad

Da die Mindestanforderung besagt, dass sowohl für hohe als auch für niedrige Schweregrade jeweils eine Methode gewählt werden soll, können die Alarmierungsmethoden zunächst in die jeweiligen Bereiche eingeteilt werden, um nachfolgende Analysen getrennt zu erstellen.

Wie bereits im Abschnitt 2.2.3 beschrieben wurde, erfolgen besonders wichtige Alarmierungen nur in den folgenden vier Fällen. Ereignisse, die

- eine sofortige Reaktion, oder
- schnelles Eingreifen benötigen,
- unerwünscht, aber wiederherstellbar sind, oder
- eine Intervention erfordern, um eine Verschlechterung zu verhindern.

Diese Alarmierungen gelten zu denen für einen hohen Schweregrad, daher sollen die Methoden die Möglichkeit besitzen, eine besonders kurze Reaktionszeit beim Empfänger zu erzeugen. Alarmierungen für einen geringen Schweregrad sollen den Empfänger lediglich informieren und dabei nicht denselben Kommunikationsweg nutzen, um die Gefahr eines „Cry Wolf Effects“ entgegenzuwirken.

Nach diesen Kriterien sollen auch die Alarmierungsmethoden gegliedert werden, welche in Abbildung 4.1 zu sehen sind.

hoher Schweregrad	niedriger Schweregrad
Telefonie	E-Mail
SMS	Web-Push-Notification
Mobile-Push-Notification	Mobile-History

Tabelle 4.1: Trennung der Alarmierungsmethoden nach Schweregrad

On Premise Prüfung

Eine weitere Anforderung besagt, dass es streng untersagt ist, eine Methode zu wählen, die nur durch den Einsatz von externen Diensten ermöglicht werden kann. In Hochsicherheitsbereichen ist es wichtig, die Sicherheit der Daten gewährleisten zu können. Im Gegensatz zu Cloud Lösungen, sind On-premise Systeme weniger anfällig für Preiserhöhungen, Datenverluste oder externe Sicherheitsbedrohungen, wie der Autor Cameron Fisher in seinem Artikel beschreibt. [9, S. 11].

Da nicht davon ausgegangen werden kann, dass alle Alarmierungsmethoden in einem On-premise System genutzt werden können, muss dies zunächst evaluiert werden. Die folgende Aufzählung zeigt die Ergebnisse der Recherche.

SMS Zum Versenden von *SMS* kann ein *SMS* Gateway genutzt werden, welcher auf einem internen Server installiert und konfiguriert werden kann. Durch On-premise Einsatz sind die genutzten Telefonnummern und Nachrichten sicher, da die Nachrichten auf dem internen Server gespeichert werden.

E-Mail E-Mails können über einen intern betriebenen *Simple Mail Transfer Protocol (SMTP)* Server versendet werden.

Telefonie Um On-premise Telefonate durchführen zu können, ist eine eigene Telefonanlage nötig, mit welcher die Telefonate getätigt werden können. Die Informationen müssen dabei mithilfe des *Session Initiation Protocol (SIP)* Protokolls übermittelt werden.

Web Push-Notification Wenn eine Weboberfläche auf einem internen Server betrieben wird, können auch die „Push-Notifications“ darüber gesendet werden.

Mobile Push-Notification Wenn die Applikation im Vordergrund läuft, können „Push-Notifications“ von einem selbst betriebenen Dienst empfangen werden. Wenn Benachrichtigungen auch außerhalb der App-Nutzung empfangen werden sollen, muss im Fall von iOS Geräten der *Apple Push Notification service (APNs)* genutzt werden [10]. Für Android wird der Nutzen von *Firebase Cloud Messaging (FCM)* empfohlen [11]. Im Falle von Android, ist es möglich einen eigenen Dienst für das Versenden von „Push-Notifications“ aufzusetzen. Dabei muss bedacht werden, dass dies den Aufwand der Umsetzung erhöhen würde.

Mobile History Die Daten zu vergangenen und aktuellen Alarmierungsereignissen kann über das interne Netzwerk empfangen werden, wenn sich sowohl das mobile Gerät als auch das Alarmsystem im selben Netzwerk befindet. Mit diesen Daten kann eine Historie in der mobilen App dargestellt werden.

Bis auf die Nutzung von „Push-Notifications“ auf iOS Geräten, bleiben nach dieser Recherche alle Alarmierungsmethoden bestehen, da diese das On-premise Kriterium erfüllen. Somit können diese in den nachfolgenden Analysen weiterhin betrachtet werden.

Nutzwertanalyse

Um die Entscheidungsfindung über die Wahl von geeigneten Alarmierungsmethoden zu vereinfachen, kann eine Nutzwertanalyse aufgestellt werden. Die Nutzwertanalyse dient der systematischen Entscheidungsvorbereitung durch Bewertung und Auswahl optimaler Alternativen und eignet sich besonders, wenn der Gesamtnutzen aus unterschiedlichen Teilnutzen zusammengesetzt ist [12, S. 43].

Mit der zuvor erstellten Tabelle 4.1 kann für jede Spalte eine eigene Nutzwertanalyse erstellt werden. Durch Betrachtung der einzelnen Attribute kann eine Kennzahl gebil-

det werden, welche den Nutzen einer Methode darstellt [12, S. 43]. Die Formel für die Berechnung des Gesamtwertes einer Methode lautet:

$$v(a) = \sum_{r=1}^{m} w_r v_r(a_r) \tag{4.1}$$

Dabei steht w_r für die Gewichtung und $v_r(a_r)$ für die Bewertung der Ausprägung a_r. Eine Bewertung kann von 0,0 bis 1,0 vergeben werden. Je höher der Wert, desto besser ist die Bewertung.

Da in den Bedingungen in Kapitel 4.1.1 vorgeschrieben ist, den Aufwand der Umsetzung und die Kosten zu berücksichtigen, können diese Kriterien als Attribute gewählt werden, welche in den nachfolgenden Bewertungen herangezogen werden. Der Aufwand wird in Stunden angegeben, und beruhen auf einer groben Schätzung. Der Preis der Umsetzung einer Methode wird in Euro angegeben.

Zunächst wird eine Nutzwertanalyse für die Alarmierungsmethoden erstellt, welche für Ereignisse mit hohem Schweregrad benutzt werden können. Hierbei handelt es sich um die Methoden „Telefonie“ und „mobile Push-Notification“. Der größte Aufwand in der Umsetzung der Telefonie-Methode liegt darin, einen geeigneten Dienst zu implementieren welcher Anrufe tätigen kann. Um mobile Benachrichtigungen umsetzen zu können, bedarf es der Entwicklung einer neuen mobilen Applikation. Der Aufwand letzterer Methode ist demnach größer, was in einer niedrigeren Bewertung resultiert.

Um geeignete Dienste zum Tätigen von Anrufen nutzen zu können, ist der Kauf von Lizenzen erforderlich. Diese befinden sich für eine einzelne Lizenz in einer Preisspanne von 125 € bis 700 €. Die Umsetzung einer mobilen Applikation für Android-Geräte ist mit keinen Kosten verbunden, wenn diese nicht im Google Play Store vertrieben werden. Möchte man die Applikation auf iOS Geräten nutzen, ist der Kauf einer Apple Developer Lizenz nötig. Daher wird auch für diese Methode eine Preisspanne von 0 € bis 95 € angegeben. Die Umsetzung der mobilen Applikation erhält folglich die bessere Bewertung von 0,8. Die einzelnen Attribute und Bewertungen der Methoden werden nun in der nachfolgenden Tabelle 4.2 gegenübergestellt.

Nutzwertanalyse für hohen Schweregrad				
Methode	Aufwand a_1	Bewertung $v_1(a_1)$	Preis a_2	Bewertung $v_2(a_2)$
Telefonie	50	0,7	125 - 700€	0,5
Mobile-Push	100	0,4	0 - 99€	0,8

Tabelle 4.2: Nutzwertanalyse für hohen Schweregrad

Die zweite Nutzwertanalyse wird für Alarmierungsmethoden aufgestellt, welche für Ereignisse niedrigen Schweregrades genutzt werden können. Hierzu zählen die Methoden „E-Mail“, „Web Push-Notification“ und „mobile History“.

Das Versenden von E-Mails bedarf lediglich einer geeigneten Bibliothek und einem internen *SMTP* Server. Der Aufwand beträgt demnach nur wenige Stunden und wird dementsprechend mit 1,0 bewertet. Um Push-Notifications in der Weboberfläche anzeigen zu können, kann im Falle von React.js eine Bibliothek eingebunden werden welche bereits gängige Funktionalitäten liefert. Da diese jedoch nur für wenige Sekunden sichtbar sind, muss eine Möglichkeit implementiert werden, um alle vergangen Benachrichtigungen in einer Liste betrachten zu können. Durch das Hinzufügen dieser Funktion kann ein besserer Vergleich zwischen den beiden Methoden vollzogen werden, da diese somit einen ähnlicheren Umfang besitzen. Der Aufwand für die Umsetzung der Benachrichtigung über die Weboberfläche ist somit höher und erhält eine schlechtere Bewertung. Die Umsetzung einer mobilen Historie erfordert, wie auch bei der Umsetzung von mobilen Push-Notifications, die Entwicklung einer eigenständigen App. Der Aufwand für die Umsetzung dieser Methode ist im Vergleich zu den anderen am höchsten, und erhält daher die niedrigste Bewertung.

Bei der Betrachtung der Preise für die einzelnen Methoden fällt auf, dass alle Methoden ohne Kosten umgesetzt werden können. Im Falle der mobilen Applikation gelten die gleichen Preise wie in der vorherigen Nutzwertanalyse, daher wird hier erneut eine Preisspanne angegeben. Somit erhält diese Methode die Bewertung von 0,5 und die restlichen eine 1,0. Die gesammelten Werte werden nun in der zweiten Nutzwertanalyse 4.3 dargestellt.

Nutzwertanalyse für niedrigen Schweregrad				
Methode	Aufwand a_1	Bewertung $v_1(a_1)$	Preis a_2	Bewertung $v_2(a_2)$
E-Mail	12	1,0	0€	1,0
Web-Push	20	0,5	0€	1,0
Mobile-History	100	0,0	0 - 99€	0,5

Tabelle 4.3: Nutzwertanalyse für niedrigen Schweregrad

Um einen Gesamtwert für die einzelnen Methoden berechnen zu können, wird eine Gewichtung w_r benötigt. Der Product Owner entschied hierbei die folgende Gewichtung, wobei w_1 für den Aufwand und w_2 für den Preis steht. Die gewählten Gewichtungen müssen aufsummiert 1,0 ergeben, um die Nutzwertanalyse korrekt ausführen zu können.

$$w_1 = 0.6 \quad w_2 = 0.4 \tag{4.2}$$

Mit dieser Gewichtung kann nun der Gesamtwert für jede Methode berechnet werden. Am Beispiel der Methode „Telefonie“ wird nachfolgend die Berechnung des gewichteten Aufwands x_1 und Preises x_2 beschrieben.

$$x_1 = w_1 * v_1(a_1) \tag{4.3}$$
$$x_1 = 0.6 * 0.7 \tag{4.4}$$
$$x_1 = 0.42 \tag{4.5}$$
$$x_2 = w_2 * v_2(a_2) \tag{4.6}$$
$$x_2 = 0.4 * 0.5 \tag{4.7}$$
$$x_2 = 0.2 \tag{4.8}$$

Der Gesamtwert dieser Methode wird durch das Addieren der beiden errechneten Werte berechnet und ergibt 0.62. Die berechneten Gewichtungen und die Gesamtwerte sind in den nachfolgenden Tabellen 4.4 und 4.5 dargestellt.

Gesamtwert für hohen Schweregrad			
Methode	Aufwand gewichtet	Preis gewichtet	Gesamtwert
Telefonie	0,42	0,2	0,62
Mobile-Push	0,24	0,32	0,56

Tabelle 4.4: Nutzwertanalyse - Gesamtwert für hohen Schweregrad

Gesamtwert für niedrigen Schweregrad			
Methode	Aufwand gewichtet	Preis gewichtet	Gesamtwert
E-Mail	0,6	0,4	1,0
Web-Push	0,3	0,4	0,7
Mobile-History	0,0	0,2	0,2

Tabelle 4.5: Nutzwertanalyse - Gesamtwert für niedrigen Schweregrad

Mit den errechneten Gesamtwerten ist erkennbar, dass für die Alarmierung mit hohem Schweregrad, die Methode über Telefonie den höchsten Wert mit 0,62 aufweist. Bei der Alarmierung mit niedrigem Schweregrad besitzt die E-Mail Methode den höchsten Gesamtwert von 1,0.

4.1.3 Ergebnis

Auf Basis der Evaluation kann nun eine Entscheidung über die Wahl der Alarmierungsmethode getroffen werden. Schwerwiegende Ereignisse müssen über eine Methode empfangen werden, welche die kürzeste Reaktionszeit hervorbringen kann. Hierfür eignen sich die Methoden über Telefonie und mobile „Push-Notifications“.

Wenn die Zeit für die Entwicklung einer eigenen mobilen Applikation aufgebracht werden kann, ist dies sehr zu empfehlen. Die Möglichkeiten sind grenzenlos und können neue Wege in Richtung intelligente Alarmierung eröffnen. Leider ist diese Methode mit einem zu hohen Aufwand verbunden. Daher empfiehlt sich die Umsetzung dieser Methode im Anschluss zur Bachelorarbeit. Die Implementierungsdauer für den Einsatz der Telefonie ist hingegen geringer und wird somit zur Methode für schwerwiegende Alarmierungen gewählt.

Weniger schwerwiegende Ereignisse sollten dennoch nicht außer Acht gelassen werden. Um einem „Cry Wolf Effect“ entgegenzuwirken, sollte hierbei eine weitere Methode gewählt werden, die nicht der schwerwiegenden Methode gleicht. Um eine mögliche Informationsflut zu senken, könnten Daten kumuliert gesendet werden. Dies setzt eine Methode voraus, welche viele Daten visualisieren kann. Dieses Problem lässt sich mit dem Einsatz der E-Mail Methode lösen, welche zudem den höchsten Gesamtwert in der Nutzwertanalyse erzielte.

4.2 Proof of Concept

Im Evaluationsergebnis ist der Einsatz von *SIP* für schwerwiegende Ereignisse empfohlen worden. Da mit dieser Methode jedoch ein großer Aufwand verbunden ist, wurde vom Product Owner definiert einen *POC* durchzuführen.

Im *POC* soll evaluiert werden, was für die programmatische Telefonie, unter der Benutzung des *SIP* Protokolls, im Vorfeld nötig ist. Außerdem soll geprüft werden, wie die Eskalation von Alarmierungen mit der Telefonie ermöglicht werden kann. Nicht jeder Kunde besitzt dieselbe Telefonanlage, daher muss das System unabhängig von Telefonanlagen sein und diese generisch benutzen können.

4.2.1 Telefonie über SIP

Es gibt zahlreiche externe Dienste, die eine Alarmierung über Telefonie erlauben. Wenn jedoch die Nutzung solcher Dienste nicht erlaubt ist, muss eine eigene Lösung hierzu erstellt werden. Die Lösung hierfür ist die Nutzung von *SIP*.

Definition

SIP ermöglicht es eine Kommunikation zwischen mehreren Teilnehmern zu kontrollieren. Neben vielen Internetprotokollen wurde auch *SIP* von der *Internet Engineering Task Force (IETF)* entwickelt. Die eigentliche Kommunikation findet nach erfolgreicher Verbindung über ein Transportprotokoll statt. Das Kommunikationsmodell basiert auf der Client-Server-Architektur. Der Client wird im SIP Kontext auch *User Agent Client (UAC)* genannt und der Server *User Agent Server (UAS)*. Der *UAC* frägt den Verbindungsaufbau an und der *UAS* vermittelt die Verbindung zum Ziel. Je nach Nutzungsweise können weitere Komponenten im SIP Kontext benutzt werden, wie beispielsweise der Proxyserver. Proxyserver sind vor allem dann sinnvoll, wenn Anrufe über Websockets gesendet werden sollen. Der Proxyserver kann diese entgegennehmen und diese anschließend und den *UAC* weiterleiten. Werden Anrufe in einer Domain genutzt, benötigt man einen Registrar Server, welche für die Registrierung der Anfragen zuständig ist. [13, S. 7-11]

Nutzung

Um Anrufe programmatisch über *SIP* tätigen zu können, muss ein *UAC* erstellt werden, welche seine Adresse beim *SIP* Registrar registrieren kann. Dafür existieren zahlreiche Bibliotheken in den unterschiedlichsten Programmiersprachen. Diese sind zudem überwiegend kostenlos, wie beispielsweise die Bibliotheken in der nachfolgenden Aufzählung.

- **drachtio** ist ein Node.js Framework für *SIP* Server Applikationen. Um Anrufe tätigen zu können, wird ein drachtio Server benötigt. [14]
- **JsSIP** ist eine Javascript *SIP* Bibliothek, welches Anrufe über eine Websocket Verbindung tätigen kann. Zur Nutzung dieser Bibliothek wird ein *SIP* Proxy mit

einer Websocket-Anbindung vorausgesetzt. [15]

- **PJSIP** ist ein Open-Source-Projekt, welches in der Sprache C geschrieben ist. Mit der Bibliothek PJSUA2 können die Funktionalitäten von PJSIP auch in den Sprachen Python, Java, und C++ genutzt werden. [16]

Um größeren Entwicklungsaufwand zu vermeiden, kann eine fertige *SIP Command Line Interface (CLI)* oder *SIP Software Development Kit (SDK)* genutzt werden, welche bereits alle Funktionalitäten zur Telefonie bereitstellt und lediglich konfiguriert werden muss. Fertige *SIP CLI* und *SIP SDK* sind meist nur in Verbindung mit Lizenzen nutzbar und sind dementsprechend nicht kostenfrei.

- **SipCli** ist eine Kommandozeilenanwendung, welche einen User Agent unter Windows zur Verfügung stellt. [17]
- **Ozeki VoIP SIP SDK** ist ein Software Development Kit. Die Webseite von Ozeki bietet eine ausführliche Dokumentation mit Code Beispielen an. Diese ist leider ebenfalls nicht plattformunabhängig und wurde nur für Windows Plattformen entwickelt. [18]

Es existieren auch On-premise Dienste, mit denen man Anrufe über Websockets tätigen kann, welche beispielsweise von Bibliotheken wie JsSIP und drachtio benötigt werden. Solche Dienste sind sinnvoll, wenn Anrufe beispielsweise über eine Weboberfläche gesendet werden sollen. In der nachfolgenden Aufzählung werden einige bekannte aufgezählt.

- **OpenSIPS** ist ein Open Source *SIP* Proxyserver [19]
- **Routr** ist ein leichtgewichtiger *SIP* Proxy [20]
- **OverSIP** ist ein *SIP* Proxy und *SIP* Server [21]
- **drachtio-server** ist ein speziell für drachtio angepasster Proxyserver [22]

Testen von SipCli

Mit der SipCli von KaplanSoft ist es möglich, Audio in Form von .wav Dateien und *TTS* wiederzugeben. Die eingebaute *TTS* Funktionalität erlaubt die Wiedergabe eines beliebigen englischen Textes. Man kann außerdem zwischen verschiedenen Stimmprofilen wechseln. Zudem können sogenannte Szenarios definiert werden, die je nach Nutzereingabe verschiedene Aktionen ausführen kann, wie beispielsweise das Aufrufen von Web API Schnittstellen oder das Ausführen von Datenbankzugriffen.

Die Konfiguration erfolgt über die Konfigurationsdatei SipCLI.ini. In dieser können neben den benötigten SIP Informationen auch Szenarien beschrieben werden. Um diese nicht selbständig in einem Editor verfassen zu müssen, kann der Szenario-Editor von KaplanSoft genutzt werden, welche über eine grafische Benutzeroberfläche verfügt.

Die Nutzung von SipCli ist sehr einfach und lässt sich in jeder Sprache nutzen, da der Aufruf über Kommandozeile erfolgt. Die kostenlose Version dieser *CLI* bietet jedoch nur 3 Sekunden Wiedergabedauer an, danach wird das Telefonat automatisch beendet.

Möchte man den vollen Funktionsumfang ohne Einschränkungen nutzen, so muss eine kostenpflichtige Lizenz gekauft werden.

Testen von Ozeki VoIP SIP SDK

Durch die gute Dokumentation ist es sehr einfach einen komplexen SIP Client zu entwickeln. Im Gegensatz zu SipCli kann mit der SDK von Ozeki komplexere Funktionalitäten eingebaut und kontrolliert werden.

Mit der Testversion von Ozeki können bereits alle Funktionalitäten vor Kauf der Lizenz genutzt werden. Dies erwies sich in den Tests als besonders hilfreich, um die *TTS* Funktionen über einen längeren Zeitraum testen zu können.

Kostenvergleich

Eine Lizenz für SipCLI kostet umgerechnet ungefähr 125 € und muss, laut Nutzungsbedingungen, für jedes System gekauft werden. Die Nutzung der Ozeki Lösung kostet zwar 697 €, muss jedoch nur einmalig gezahlt werden, wenn es für ein einzelnes Produkt entwickelt wird.

Da das Alarmsystem in Zukunft als ein einzelnes Produkt für mehrere Systeme vertrieben werden soll, wäre der Kauf der Ozeki Lizenz für diesen Anwendungsfall geeigneter. Soll das Alarmsystem nur einem einzelnen System genutzt werden, würde sich der Kauf der Lizenz für SipCLI lohnen.

4.2.2 Ergebnis

Der *POC* war erfolgreich und zeigte, dass die Nutzung von *SIP* in unserem Unternehmen möglich ist. Die Implementierung und Konfiguration erwies sich jedoch als große Herausforderung, da es ein umfassendes Verständnis über die Funktionsweise der allgemeinen Telefonie und deren Protokolle benötigt.

Viele Bibliotheken sind veraltet oder werden nicht mehr gewartet. Eine eigene Implementation für *SIP* anzufertigen ist wegen des Aufwandes nicht zu empfehlen und sollte nur mit Kenntnissen in der Telefonie vollzogen werden.

Um die *SIP* Funktionalität in der Entwicklungsphase ohne großen Aufwand testen zu können, empfiehlt sich daher die Nutzung der umfangreichen Ozeki VoIP *SIP SDK*. Je nachdem, welche Anforderungen der Kunde an das System hat, kann auch eine andere Methode gewählt werden.

Kapitel 5

Alarmsystem

Die Anforderung an die Ausarbeitung dieser Bachelorarbeit lässt sich in zwei Themenbereiche gliedern, da neben der Entwicklung des Alarmsystems auch die Erweiterung eines Monitoring-Systems erläutert werden soll. Der erste Themenbereich wird in diesem Kapitel beschrieben, welches sich der Konzeption und Umsetzung des Alarmsystems widmet. Die Ergebnisse der Analyse aus Kapitel 4 stellen die Grundlagen für dieses Kapitel auf.

5.1 Konzeption

Um die Umsetzung des Alarmsystems detailliert planen zu können, muss zunächst ein Konzept erstellt werden. Daher befasst sich dieses Kapitel mit der Konzeption des Alarmsystems, welches in Fachkonzept und Grobarchitektur aufgeteilt ist.

5.1.1 Fachkonzept

Da das zentrale Alarmsystem von mehreren Produkten gleichzeitig benutzt werden soll, soll es als eigenständiges Produkt entwickelt werden. Damit ist es möglich, das System auf einem einzigen Server zu betreiben, wodurch alle Anfragen über die gleiche Schnittstelle kommuniziert werden.

Damit das Alarmsystem von mehreren unterschiedlichen Monitoring-Systemen genutzt werden kann, sollte es sich nicht um die Datenhaltung von Metriken kümmern, da es diese sonst selbständig auswerten müsste. Vielmehr sollte es als Kommunikationsschnittstelle genutzt werden. Das bedeutet, dass die Monitoring-Systeme ihre Metriken eigenständig halten und auswerten. Sobald die Bedingung einer Alarmkonfiguration erfüllt ist, wird die Funktionalität des Alarmsystems genutzt, um eine reaktive Alarmierung auszulösen.

Telefonie

Durch den zentralen Einsatz kann die Verwaltung von Service-Nummern erleichtert werden. Würden beispielsweise mehrere Alarmsysteme die gleiche Service-Nummer benutzen wollen, kann es zu Problemen in der Ausführung kommen. Es ist nämlich zu beachten, dass bei der Nutzung der Telefonie mit einer Nummer nur ein Telefonat gleichzeitig geführt werden kann. In diesem Szenario müssten die einzelnen Alarmsysteme einen Weg finden, die Verfügbarkeit der Leitung anzufragen, um diese nach der Reihe zu benutzen.

Dieses Problem lässt sich durch den zentralen Einsatz eines einzelnen Alarmsystems lösen, da dieser die alleinige Befugnis über die Nutzung der Service-Nummer erlangt und diese somit besser koordinieren kann. Da mehrere Monitoring-Systeme auf ein einzelnes Alarmsystem zugreifen können, um Anrufe zu tätigen, reicht bereits eine einzelne Telefonleitung. Somit kann der Einsatz vieler Service-Nummern vermieden werden. Es wäre dennoch möglich, mehrere Service-Nummern zu nutzen, um die Last der Alarmierungen zu verteilen. Es wäre auch denkbar, eine Service-Nummer für ein einzelnes Monitoring-System zu nutzen. Dadurch würde sich der Vorteil ergeben, dass die benachrichtigten Personen besser erkennen können von welchem System der Alarm ausgegangen ist. Dies würde jedoch eine erweiterte Konfiguration des Alarmsystems erfordern und wird deswegen nicht in der Ausarbeitung dieser Arbeit berücksichtigt.

Um die Nutzung der Nummern kontrollieren zu können, wird zudem eine spezielle Verarbeitungslogik benötigt, da mit einer Nummer nur ein Telefonat gleichzeitig geführt werden kann. Demnach kann im Falle einer gleichzeitigen Alarmierung über eine einzelne Telefonleitung nur ein Anruf verarbeitet werden. Um das Problem zu lösen, muss eine sequentielle Verarbeitung sichergestellt werden, welche dem Verlust von Alarmierungen entgegenwirkt. Für die sequentielle Verarbeitung kann die Funktionalität einer Warteschlange benutzt werden. Alarmierungen werden dafür zuerst in einer Liste eingereiht, welche anschließend nacheinander abgearbeitet werden. Mit dieser Logik ist es möglich, dass gleichzeitige Alarmierungen beim Alarmsystem eintreffen und kontrolliert abgearbeitet werden.

Im Weiteren ist eine Eskalationslogik gefragt, welche es ermöglichen soll, nicht angenommene Telefonate an einen anderen Empfänger weiterzuleiten. Um eine gute Eskalationslogik implementieren zu können, müssen vorher alle Fälle berücksichtigt werden, um eine geeignete Bearbeitungsabfolge zu definieren. Mögliche Fälle sind hierbei:

- **Der erste Empfänger nimmt den Anruf an** - Der erste Empfänger in der Nutzergruppe nimmt den Anruf an. Es wird kein anderer Empfänger für diesen Alarm benachrichtigt und der Alarmierungsprozess kann beendet werden.
- **Der erste Empfänger nimmt den Anruf nicht an** - Der erste Empfänger in der Nutzergruppe hat den Anruf nicht angenommen. Es werden in Reihenfolge alle weiteren Empfänger für diesen Alarm benachrichtigt, bis einer der Empfänger den Anruf annimmt und der Alarmierungsprozess beendet werden kann.

- **Keiner nimmt den Anruf an** - Die gesamte Nutzergruppe hat den Anruf nicht angenommen. Eskalierende Anrufe, auf die kein Empfänger reagiert, müssen kontrolliert werden, damit es nicht zu einer Endlosschleife kommt. Um diesem Problem entgegenwirken zu können, muss nach einer abgeschlossenen Eskalationsrunde der Alarm neu eingereiht werden, damit diese zu einem späteren Zeitpunkt erneut durchgeführt werden kann, ohne andere Alarmierungsprozesse zu blockieren.
- **Keiner nimmt den Anruf in der zweiten Runde an** - Die gesamte Nutzergruppe hat den Anruf erneut nicht angenommen. Um diese Art der Endlosschleife zu verhindern, muss bestimmt werden, wie oft eine Wiederholung der Eskalation eintreten kann und was danach geschehen muss.

Jeder dieser Fälle muss in der Implementierung berücksichtigt werden, damit ein reibungsloser Ablauf zu gewährleisten ist.

Job Status

Aus der Sicht des Alarmsystems können Alarmierungen als einzelne Jobs angesehen werden, da diese einen Zweck in einem Zeitraum erfüllen müssen, welcher nach der Bearbeitung beendet ist. Damit das jeweilige Monitoring-System ihre zugehörigen laufenden Jobs einsehen kann, wird eine weitere Kommunikationsschnittstelle zum Alarmsystem benötigt. Da die Überwachung der laufenden Jobs in Echtzeit gewünscht ist, um Statusänderungen persistent überwachen zu können, muss eine entsprechende Kommunikationsübertragung gewählt werden.

5.1.2 Grobarchitektur

Damit das Alarmsystem von allen möglichen Monitoring-Systemen genutzt werden kann, empfiehlt sich die Erstellung einer eigenständigen Web-API. Damit diese im internen Netzwerk genutzt werden kann, muss das System auf einem Server im selben Netzwerk installiert werden.

Alarmdaten

Ein Alarm entsteht erst, wenn die Bedingung eines konfigurierten Alarms im betreffenden Monitoring-System ausgelöst wird. Sobald dies geschieht werden die erforderlichen Daten zur Alarmierung an das Alarmsystem kommuniziert. Damit das Alarmsystem die einzelnen Alarmierungen kontrollieren kann, ist die Definition der Alarmdaten erforderlich. Das Format der Alarmdaten gibt an, wie die Alarmierung über die REST-Schnittstelle ausgeführt werden kann.

Um einen Alarm eindeutig identifizieren zu können, wird eine ID benötigt, welches durch das Anlegen in der Datenbank des Alarmsystems automatisch generiert werden kann. Damit der Alarm einem Ereignis zugeordnet werden kann, ist zudem eine Beschreibung der Ursache nötig. Um den Zeitpunkt des ausgelösten Ereignisses angeben zu können, wird ein Zeitstempel benötigt. Der Zeitstempel wird im Millisekunden

Format abgespeichert. Um wiederkehrende Alarmierungen für den Nutzer sichtbar zu machen, eignet sich die Vergabe eines Alarmnamens durch den Nutzer. In der Tabelle 5.1.2 wird ein Beispiel an mehreren Alarmdaten abgebildet.

AlarmId	AlarmName	Zeitpunkt des Ereignisses	Ursache
1	Papierstau	1359491000	Sensorfehler
2	Maschinenstopp	1204152772	Sensorfehler
3	Steuerungsfehler	1291615863	Maschinenfehler

Tabelle 5.1: Beispiel Alarmdaten

Job Daten

Da ein Alarm nicht im Alarmsystem, sondern in dem jeweiligen Monitoring-System definiert wird, sollen diese Daten im Monitoring-System gehalten werden. Sobald ein Monitoring-System eine Alarmierung beginnen will, müssen Alarmdaten an das Alarmsystem gesendet werden, um einen Job zu generieren.

Die Jobs werden vom Alarmsystem bearbeitet und deswegen in der eigenen Datenbank gehalten. Auch hier ist eine ID nötig, um den Job identifizieren zu können. Sobald die Alarmierungsanfrage im Alarmsystem eingetroffen ist, wird sofort ein zugehöriger Job erstellt. Dieser erhält zu einem einen Zeitstempel, um den Beginn der Alarmierung festzuhalten. In einem zweiten Zeitstempel kann die Änderung der Alarmdaten festgehalten werden. Dies ist beispielsweise der Fall, wenn der Status der Alarmierung geändert wird. Damit der Status des Jobs überprüft werden kann, wird das Attribut „JobStatus“ benötigt. In der nachfolgenden Auflistung werden alle möglichen Status erklärt.

1. **Queued** - Sobald ein neuer Job angelegt wird, enthält dieser den initialen Status „Queued“. Der Job wird in die Warteschlange eingereiht und wird erst dann ausgeführt, wenn dieser die erste Warteschlangenposition erreicht hat.
2. **Running** - Befindet sich der Job an der ersten Warteschlangenposition, wird dieser ausgeführt und behält den Status „Running “ solange bis die Ausführung beendet ist.
3. **Finished** - Wurde der Job ausgeführt, wird der Status auf „Finished“ gesetzt.

Die Abbildung in 5.1.2 zeigt einen Beispieldatensatz an Job-Daten.

JobId	JobStatus	Erstellung	Änderung	Ursache
1	Queued	1359491000	1359495000	Papierschacht Sensor Fehler
2	Running	1204152772	1204172772	Arm Sensor Fehler
3	Finished	1291615863	1291645863	Maschinenfehler

Tabelle 5.2: Beispiel Jobdaten

Job Status Abfrage

Um die Informationen über laufende Jobs an die Monitoring-Systeme versenden zu können, wird der Einsatz von Websockets empfohlen.

Bei einem Websocket bleibt die Verbindung zwischen dem Client und dem Server bestehen. Dadurch kann die offene Verbindung aktiv genutzt werden, um ständig neue Informationen zu versenden, ohne einen neuen Verbindungsaufbau abzuwarten. Dabei handelt es sich um eine asynchrone Übertragung, welche auf dem *Transmission Control Protocol (TCP)* Protokoll basiert. [23, S. 7]

Durch Websockets wird somit die gewünschte Echtzeitkommunikation aus der Anforderung hergestellt, welche durch den Einsatz von herkömmlichen *Hypertext Transfer Protocol (HTTP) Representational State Transfer (REST)* Schnittstellen nicht möglich wäre.

Sobald die Verbindung zum Websocket aufgebaut wird, kann der Nutzer die Job-Daten empfangen, bis die Verbindung abgebrochen wird. Damit keine großen Datenmengen gesendet werden, sollten nicht bei jeder Änderung alle Job-Daten erneut gesendet werden. Stattdessen sollte lediglich der Job mit der Änderung gesendet werden, damit im Monitoring-System dieses angepasst werden kann. Deswegen ist unter anderem die Nutzung einer eindeutigen JobID unerlässlich.

Mit der aktuellen Implementierung wären jedoch alle Monitoring-Systeme dazu in der Lage, alle Alarmierungen, die von jeglichen Monitoring-Systemen erstellt wurden, einzusehen. Um dies zu vermeiden, muss die Job-Klasse ein weiteres Feld erhalten, welche die ID des Monitoring-Systems hält. Diese Funktionalität wird aus Zeitgründen nicht in dieser Ausarbeitung implementiert, ist jedoch nach Fertigstellung der Arbeit geplant.

Job Handler

Damit einzelne Alarmierungen kontrolliert werden können, kann ein „Job Handler“ entwickelt werden. Da bei der Alarmierung über die Telefonie nur ein Telefonat gleichzeitig getätigt werden kann, ist in diesem Fall eine sequentielle Abarbeitung nötig. Um unterschiedliche Logiken zu den verschiedenen Benachrichtigungsmethoden implementieren zu können, empfiehlt sich die Erstellung von spezifischen „Job Handlern“ für die jeweilige Methodenart.

Telefonie Job Handler

Sobald ein Monitoring-System einen Alarm auslöst, wird dieser über die *REST* Schnittstelle des Alarmsystems empfangen und an den richtigen „Job Handler“ übergeben.

Bei der *SIP* Telefonie ist zu beachten, dass keine simultanen Gespräche mit einem Client geführt werden kann. Daher müssen Anrufe in einer Warteschlange nach Prioritäten eingereiht werden, damit diese nacheinander bearbeitet werden können. Der „Job Handler“ übergibt die Daten an den *SIP* Client und tätigt den Alarm über Telefonie. Wenn der Empfänger den Anruf entgegengenommen hat, wird der Job-Status auf „Finished“ gesetzt und abgeschlossen. Diese Abfolge ist ebenso in Abbildung 5.1 zu

sehen. Sollte der Empfänger den Anruf nicht annehmen, wird der Anruf vom „Job Handler“ für den nächsten Empfänger vorbereitet.

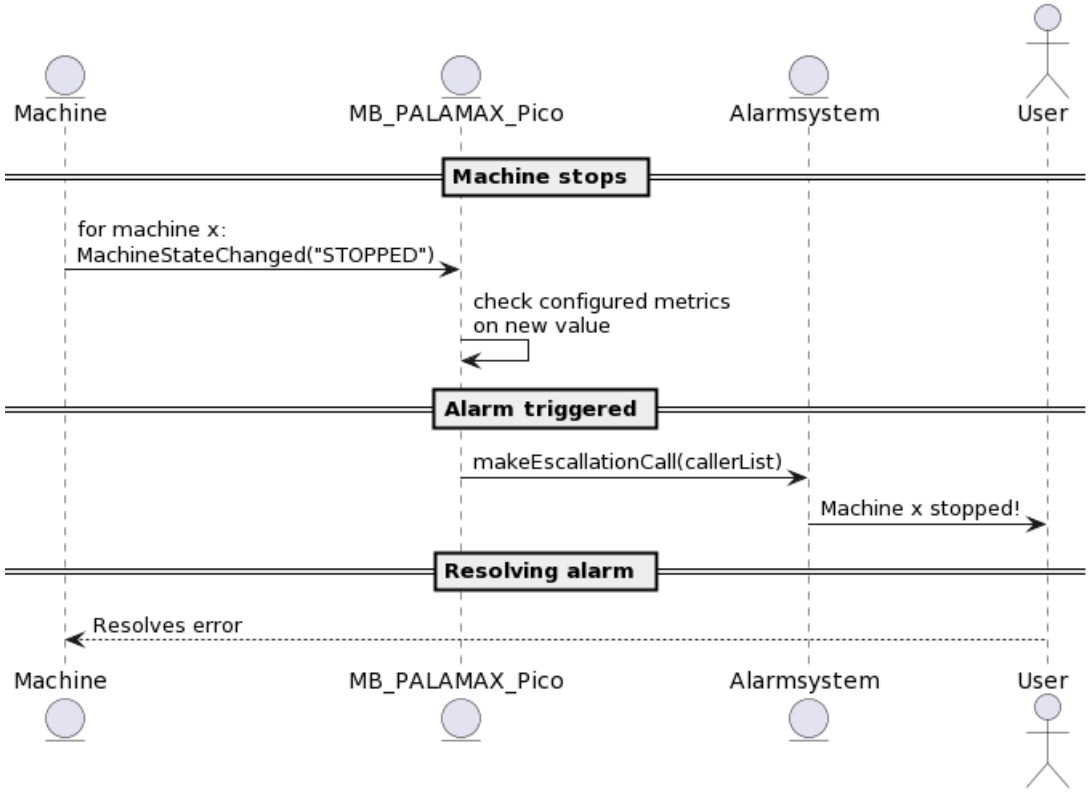


Abbildung 5.1: Der Alarmierungsprozess

Konfiguration

Viele Kommunikationswege, wie beispielsweise das Versenden von E-Mails, benötigen zusätzliche Informationen, um eine Verbindung aufbauen zu können. Um eine E-Mail versenden zu können, werden Informationen zur Verbindung zum *SMTP* Server benötigt. Da sich diese Informationen selten ändern, ist eine *REST* Schnittstelle für diesen Fall unnötig, hier sollte eher eine Konfigurationsdatei verwendet werden, die beim Programmstart eingelesen wird.

Auch für die Nutzung der *SIP* Telefonie wird eine Konfiguration benötigt. Neben der IP-Adresse des Registrars ist auch die registrierte Client-ID erforderlich, um ein Telefonat über die Telefonanlage absetzen zu können.

5.1.3 Systementwurf

Damit das Alarmsystem von beliebigen Monitoring-Systemen genutzt werden kann, wird das Alarmsystem als eigenständiges Produkt konzipiert. Im Rahmen der Bachelorarbeit wird das System als ASP.Net Web *Application Programming Interface (API)* entwickelt und basiert somit auf der Programmiersprache C#. Damit Alarmierungen

im jeweiligen Monitoring-System konfiguriert werden können, benötigt das Alarmsystem keine eigene Weboberfläche. Es soll jedoch die benötigten Informationen über REST Schnittstellen bereitstellen.

Anwendungsszenarien

In der Abbildung 5.2 ist eine mögliche Netzwerkstruktur abgebildet, mit der es möglich ist eine Alarmierung für jede abgebildete Maschine zu gewährleisten. In diesem Beispiel existiert das Monitoring-System MB PALAMAX® Pico, welches auf alle Maschinen in unterschiedlichen Produktionsumgebungen zugreift. Die Metriken zur Alarmierung können also für alle angeschlossenen Maschinen erstellt werden.

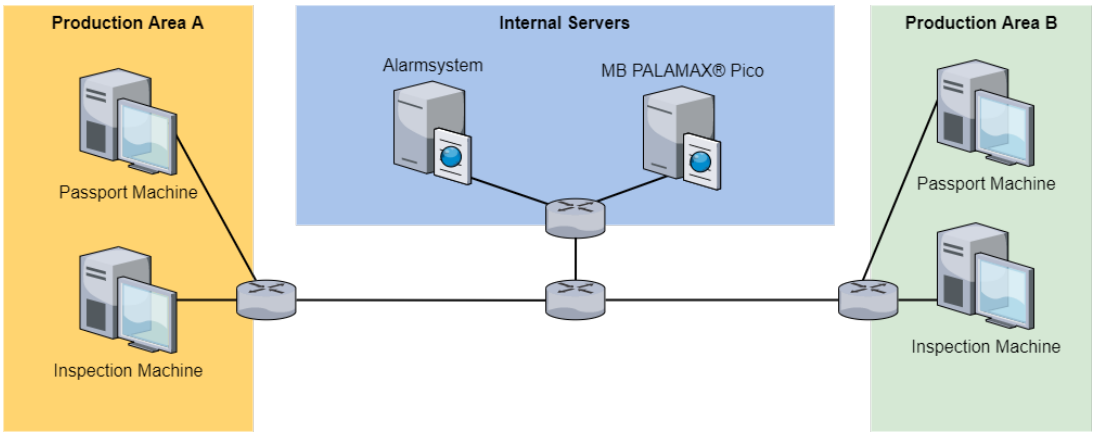


Abbildung 5.2: Netzwerkdiagramm mit Alarmsystem

Da MB PALAMAX® Pico jedoch auch direkt auf den einzelnen Produktionsmaschinen installiert werden kann, besteht ein weiteres Szenario, welches in der Abbildung 5.3 zu sehen ist.

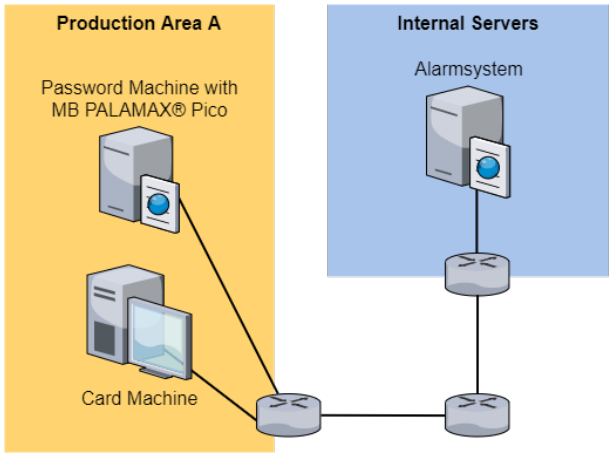


Abbildung 5.3: Netzwerkdiagramm mit Alarmsystem und mehreren Monitoring-Systemen

In beiden Szenarien ist der Einsatz eines zentralen Alarmsystems möglich, da sich jedes Monitoring-System im selben Netz befindet und die Anfragen über REST versenden kann.

Datenbank

Damit im Falle eines Neustarts des Systems laufende Jobs wieder aufgenommen werden können, eignet sich die Speicherung von Jobs in einer Datenbank. Diese können dann zum Programmstart eingelesen werden. Mithilfe dieser Daten ist es zusätzlich möglich, eine Historie an vergangenen Jobs einsehen zu können.

Die Datenstruktur eines Jobs wurde bereits in Kapitel 5.1.2 beschrieben und wird dementsprechend auch in der Job-Tabelle abgebildet. In Abbildung 5.4 wird das Datenbankmodell dargestellt.

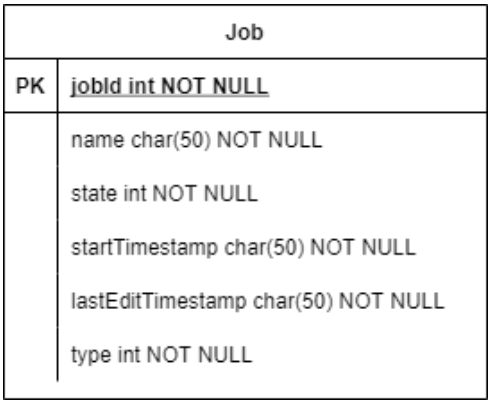


Abbildung 5.4: Job Tabelle

Neben der Job-Tabelle wird keine weitere benötigt, da die restlichen Informationen zur Alarmierung über die *REST* Schnittstellen empfangen werden.

REST Schnittstellen

Um die Alarmierungen an das Alarmsystem senden zu können, sollen die benötigten Informationen über *REST* Schnittstellen bereitgestellt werden.

Die *REST* Schnittstellen befolgen das „Single Responsibility“ Prinzip. Dieses Prinzip ist eines von fünf weiteren, welche zusammen die SOLID Prinzipien nach Robert Cecil Martin bilden. Das „Single Responsibility“ Prinzip besagt, dass eine Klasse nur eine einzige Verantwortlichkeit besitzen darf [24, S. 49-58]. Demnach wird für jede Alarmierungsmethode eine eigene Schnittstelle erzeugt. Dies ist unter anderem wichtig, um die unterschiedlichen Übergabeparameter richtig verarbeiten zu können. Während bei der Telefonie die Telefonnummer von einer oder mehreren Personen benötigt wird, werden bei der Methode über E-Mail die E-Mail-Adressen der Nutzer benötigt.

Somit wird im Alarmsystem eine Schnittstelle für die Telefonie und eine für das Senden von E-Mails implementiert.

Zur Telefonie wird der CallController mit den folgenden API-Schnittstellen erstellt.

- **/simpleCall** - Setzt einen einfachen Anruf ab
- **/escalationCall** - Beginnt einen Eskalationsanruf

Für das Versenden von E-Mails dient der E-Mail Controller.

- **/simpleMail** - Sendet Daten sofort per Mail

Authentifizierung

Damit die Funktionalitäten des Alarmsystems nicht von unbefugten Personen und Services genutzt werden kann, muss eine sichere Authentifizierung sichergestellt werden. Einer der meistgenutzten Authentifizierung Methoden ist die Nutzung von JSON *Web Token (JWT)*.

5.2 Technologieauswahl

Dieser Abschnitt befasst sich mit den verschiedenen Methoden, welche für die Umsetzung des Alarmsystems benötigt werden. Da das System plattformunabhängig betrieben werden soll und in der Softwareabteilung bereits viele Softwarelösungen in C# und ASP.Net geschrieben wurden, soll das Alarmsystem mit den gleichen Technologien umgesetzt werden. Die nachfolgenden Methoden beziehen sich daher auf die Programmiersprache C#.

Um die Suche nach Technologien zu vereinfachen, kann die Nutzung von *Awesome* Listen auf Github nützlich sein. [25].

5.2.1 Versenden von Mails

Für das Versenden von E-Mails in C# können unterschiedliche Bibliotheken benutzt werden. Einige aus der *Awesome* Liste für .NET Core sind beispielsweise:

- FluentEmail [26]
- MailKit [27]
- MimeKit [28]

Da die Anforderung lediglich die Funktionalität zum Versenden von E-Mails vorschreibt, genügt eine Bibliothek, die schnell eingebunden werden kann. MailKit ist hierbei die am häufigsten benutzte Bibliothek zum Versenden von E-Mails in C#.

5.2.2 Job Scheduling

Damit Methoden wie die *SIP* Telefonie benutzt werden können, wird ein Job Scheduling System benötigt. Es gibt viele Scheduling Bibliotheken für C#, die in der nachfolgenden Tabelle gegenüber gestellt werden.

- **FluentScheduler** - Diese Bibliothek wurde zuletzt vor zwei Jahren aktualisiert und könnte somit Probleme in der Kompatibilität aufweisen. [29]
- **QuartzNet** - Der Quartz Enterprise Scheduler wird regelmäßig aktualisiert und ist somit eine bessere Wahl. [30]
- **Hangfire** - Wird vor allem zum Anwenden von „fire-and-forget“ Logiken genutzt. Außerdem können mit dieser Bibliothek verzögerte und wiederkehrende Aufgaben kontrolliert werden. [31]
- **Chroniton** - Auch mit Chroniton können mehrere Anwendungsfälle zum Kontrollieren von geplanten Jobs behandelt werden. Der letzte Commit ist jedoch vor über 6 Jahren erstellt worden und könnte somit ebenfalls veraltet sein. [32]

Jede dieser Technologien wäre für den aktuellen Funktionsumfang geeignet, jedoch werden einige nicht mehr regelmäßig aktualisiert oder überhaupt nicht mehr gewartet. Es ist zu beachten, dass in Zukunft komplexe Scheduling-Logiken gebraucht

werden könnten, die beispielsweise die Alarmierung nach einem Abwesenheitsplan gewährleisten soll.

Für diesen Funktionsumfang empfiehlt sich der Einsatz der Quartz Bibliothek, da diese nicht nur alle aktuellen Anforderungen erfüllt, sondern auch die zukünftigen decken kann. Zwar würde die Implementierung dieser Technologie anfänglich mehr Zeit in Anspruch nehmen als die vergleichbaren Bibliotheken, jedoch kann diese Zeit in Zukunft eingespart werden, da somit der Wechsel der Technologie vermieden wird. Ein weiterer Vorteil ist, dass diese Bibliothek noch heute (Stand 19.11.2022) regelmäßige Commits erhält und somit weniger anfällig für Kompatibilitätsprobleme ist.

5.2.3 Datenbankzugriff

Da das Alarmsystem eine Datenbank mit einer Tabelle für Jobs hält, wird eine Technologie benötigt, welche die Datenbankzugriffe kontrolliert.

In einer .NET Core Umgebung wird hierfür überwiegend Entity Framework Core benutzt, welches in der Kurzschreibweise EF Core genannt wird. Mit EF Core ist es .NET-Entwicklern möglich, Datenbankabfragen mit .NET-Objekten zu erstellen. Dabei müssen keine eigenen Zugriffscodes geschrieben werden, was eine schnellere Integration der Datenbank gewährleistet [33].

5.3 Technische Umsetzung

Im folgenden Kapitel wird mithilfe des zuvor erstellen Konzepts aus Kapitel 5.1 auf die technische Umsetzung des Alarmsystems eingegangen. Zudem werden spezielle Logiken im Detail erläutert, welche zusätzlich in Form von Code-Ausschnitten dargestellt werden.

5.3.1 Job

Für die Umsetzung der Job Logik muss neben der Erstellung eines geeigneten Objekt-Modells, die Art der Job-Status-Abfrage und die Logik des Job Handlers erläutert werden.

Job-Modell

Das Job-Modell wurde wie in der Abbildung 5.5 implementiert. Da es mehrere Job-Typen gibt, wurde hierfür das Enum „JobType“ erstellt. Ein Job-Status kann ebenfalls einer von mehreren vordefinierten Werten sein. Daher wurde hierfür ebenfalls das Enum „JobState“ erstellt.

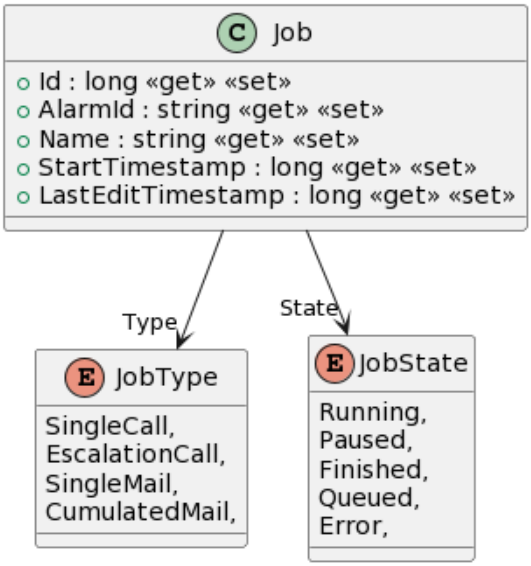


Abbildung 5.5: Job Modell

Job-Status

Wenn ein Monitoring-System gestartet wird, soll es eine initiale Websocket Verbindung zum Alarmsystem aufbauen, welches während der gesamten Laufzeit über aufrechterhalten werden soll. Sobald die Verbindung besteht, kann das Monitoring-System Job Änderungen über den Websocket empfangen. Jegliche Job-Änderungen sollen folglich im Alarmsystem über den Websocket gesendet werden.

Job-Handler

Die Job-Handler basieren auf dem Quartz-NuGet-Paket. Mithilfe von Quartz ist es möglich, zeitbasierte Jobs auszuführen. Diese können zu bestimmen Zeiten gestartet und beendet werden. Diese Funktionalität eignet sich besonders, um kumulierte Daten über E-Mail zu senden, da somit ein Zeitstempel zum Senden festgelegt werden kann, für welchen die Kumulierung der Daten beendet wird und eine E-Mail mit dem Ergebnis versendet wird.

Für jede Kommunikationsvariante wird ein eigener Controller erstellt. Eingehende *REST*-Anfragen zu ausgelösten Alarmen erstellen einen eigenständigen Job, der entweder sofort ausgeführt werden kann oder in einer Warteschleife eingereiht wird. Wurde der Job ausgeführt, wird dieser von Quartz automatisch verworfen.

Die *REST*-Schnittstelle des Alarmsytems erzeugt bei jedem Aufruf einen asynchronen Thread. Die Jobs, die so gestartet werden würden, wären also asynchron. Damit Jobs dennoch sequentiell arbeiten können, muss der Job-Handler Multi-Thread sicher sein.

Über der öffentlichen Methode „AddJob“ können Alarmierungen in Form von Jobs hinzugefügt werden. Da die Alarmierungsdaten über *REST* empfangen werden, und dies asynchron geschieht, muss das Hinzufügen von Jobs „thread-sicher“ implementiert werden. Dies kann durch den Einsatz einer „lock“ Anweisung erreicht werden. Wenn gerade kein Job ausgeführt wird, wird dieser direkt im Scheduler angelegt. Falls bereits einer ausgeführt wird, wird der Job stattdessen der QueuedJobs Liste hinzugefügt. Diese Logik ist in der Methode „AddJob“

```
1  public void AddJob(IJobDetail jobDetail)
2  {
3      lock (jobqueueLock)
4      {
5          if (CurrentJob == null)
6          {
7              CurrentJob = jobDetail;
8              ITrigger trigger = TriggerBuilder.Create()
9                 .WithIdentity(jobDetail.Key.Name + DateTime.Now.Millisecond)
10                .ForJob(jobDetail.Key.Name)
11                .StartNow()
12                .Build();
13
14             _scheduler.ScheduleJob(jobDetail, trigger);
15         }
16         else
17         {
18             QueuedJobs.Add(jobDetail);
19         }
20     }
21 }
```

Listing 5.1: AddJob Methode, mit welcher Jobs erzeugt und der Warteschlange hinzugefügt werden

Der JobHandler implementiert das IJobListener Interface, welches vom Quartz NuGet Paket bereitgestellt wird. Das Interface definiert die zwei Methoden JobToBeExecuted und JobWasExecuted, welche implementiert werden müssen. Der JobHandler ist mit diesen zwei Methoden in der Lage, Funktionen vor und nach dem Ausführen eines Jobs ausführen.

- **JobToBeExecuted** - Mit der Methode „JobToBeExecuted“ kann der Status des Jobs auf Running gesetzt werden.
- **JobWasExecuted** - Mit „JobWasExecuted“ kann nach einem erfolgreich ausgeführten Job ein neuer Job aus der QueuedJobs Liste begonnen werden. Der ausgeführte Job erhält nun den neuen Status „Finished“. Hat ein Empfänger den Anruf nicht angenommen, zählt der Job als nicht erfolgreich. Ist dies der Fall, wird der Job erneut am Ende der Liste eingereiht

Die genaue Umsetzung der JobToBeExecuted B.2 und der JobWasExecuted Methoden B.5 ist im Anhang unter dem Kapitel B.2 als Code zu sehen. Die genutzten Hilfsmethoden werden ebenfalls im Anhang dargestellt.

5.3.2 Telefonie

Je nach Art der intern genutzten *SIP* Hardware muss ein passender Client zum Erzeugen von Telefonaten gewählt werden. Je nach Kunde muss die Schnittstelle also im kleinen Maße angepasst werden. Für den Prototypen eignet sich ein Test Client, wie beispielsweise der von Ozeki. Ozeki stellt eine sehr mächtige *SIP Voice over IP (VoIP) SDK* zur Verfügung, mit der zahlreiche Funktionalitäten der Telefonie genutzt werden können.

Es gibt viele verschiedene Möglichkeiten, die *SIP* Telefonie bereits ohne Kauf einer Lizenz ausprobieren zu können. Die Ozeki *SIP VoIP SDK* für Entwickler bietet eine umfassende Tutorial Reihe und einen angenehmen Einstieg in die Thematik rund um *SIP*. Die bereitgestellte *SDK* bietet neben der Registrierung über *User Datagram Protocol (UDP)* auch die Möglichkeit zur Eingabeaufnahme über *Dual Tone Multifrequency (DTMF)*.

Es ist jedoch zu beachten, dass mit einer einzelnen registrierten *SIP* Nummer keine gleichzeitigen Telefonate geführt werden können. Da es vorkommen kann, dass Alarmierungen zeitgleich stattfinden könnten, müssen diese immer sequentiell abgearbeitet werden. Damit die sequentielle Abarbeitung sichergestellt werden kann, wird eine Logik benötigt, die Alarme als einzelne Jobs betrachtet, welche in einer Warteschlange eingereiht werden.

Da die Alarmierungen über *REST* Anfragen kommen, welche in asynchronen Threads verarbeitet werden, muss besonders viel Wert auf eine „Thread“ sichere Logik gelegt werden. Dies kann unter der Einhaltung des „Singleton Patterns“ erreicht werden und durch den Einsatz von „lock“-Anweisungen.

Eskalationslogik

Damit Telefonate eskaliert werden können, müssen bei der Konfiguration eines Alarms mindestens 2 Empfänger gewählt werden. Sobald ein Empfänger den Anruf nach der konfigurierten Zeit nicht annimmt, wird der nächsten Empfänger in der Eskalationsliste angerufen. Sobald eine Person in dieser Eskalationsliste den Anruf entgegennimmt, wird der Eskalationsprozess beendet und der Job wird aus der Warteschlange entfernt. Diese Logik wird in Abbildung 5.6 in Form eines Sequenzdiagramms dargestellt.

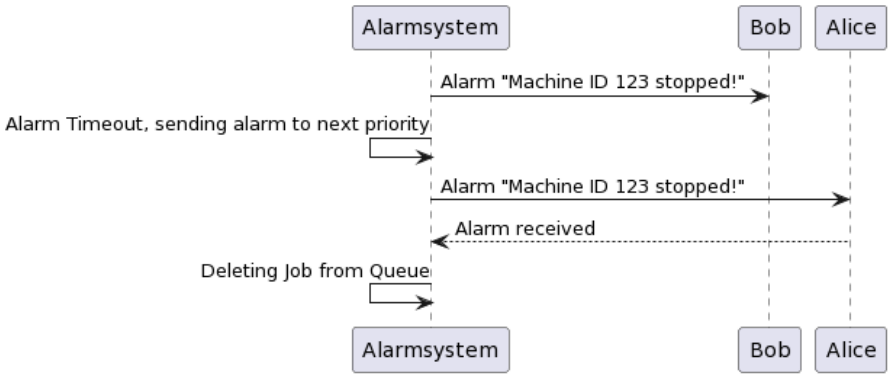


Abbildung 5.6: Eskalationslogik als Sequenzdiagramm

Wenn jedoch keiner in der Eskalationsliste den Anruf entgegennimmt, muss die Alarmierung erneut in die Warteschlange eingereiht werden. Das erneute Einreihen vermeidet eine Endlosschleife und ermöglicht, dass andere Alarmierungen zuerst getätigt werden, bevor die fehlgeschlagene Alarmierung erneut ausgeführt wird.

5.3.3 E-Mail

E-Mails können mit dem Mailkit Package gesendet werden. Hierfür muss dem Alarmsystem ein eigener E-Mail-Account zugewiesen werden, mit welcher E-Mails versendet werden können. Die nötigen Daten zum Authentifizieren beim *SMTP* Server sollten in einer Konfigurationsdatei gesichert werden und beim Programmstart ausgelesen werden. Die Konfigurationsdatei benötigt zum Authentifizieren des Nutzers die E-Mail-Adresse, den Nutzernamen und das zugehörige Passwort. Um die Verbindung zum *SMTP* Server herstellen zu können, wird zudem der Host und der Port benötigt. Eine beispielhafte Konfiguration kann aus dem Listing 5.2 entnommen werden.

```
1 {
2     "Username": "domain\\username",
3     "Mail": "username@domain.de",
4     "Password": "screenshotPassword",
5     "Host": "outlook.company.de",
6     "Port": 465
7 }
```

Listing 5.2: E-Mail Konfigurationsdatei in JSON

Zum Versenden einer E-Mail wird die Methode „SendEmailAsync“ definiert. Diese Methode wird im Listing 5.4 dargestellt und wird nachfolgend beschrieben. In den Zeilen [4-6] wird der Absender, der Empfänger und der Betreff im „email“ Objekt gesetzt. Damit die E-Mails nicht als reine Textnachrichten ankommen, kann ein E-Mail Template verwendet werden. Um ein E-Mail Template nutzen zu können, müssen Platzhalter festgelegt werden, welche bei der Erzeugung der E-Mail durch die tatsächlichen Daten ausgetauscht werden kann. Die Logik zum Ersetzen der Platzhalter ist in den Zeilen [10-13] zu sehen. Das Listing 5.3 zeigt den Auszug eines möglichen Templates, welches mit der „SendEmailAsync“ Methode genutzt werden kann. Die Abbildung A.1 im Anhang zeigt eine erzeugte E-Mail, welche auf einem Template für MB PALAMAX® Pico basiert.

```
1 <p><strong>Dear FullNameValue!</strong></p>
2 <p><strong>Topic:</strong> TopicName</p>
3 <p><strong>Monitored Values:</strong> MonitoredValues</p>
4 <p><strong>Message:</strong></p><p>MessageValue</p>
5 <p>Best regards, MB Alarmsystem</p>
```

Listing 5.3: Auszug eines möglichen E-Mail-Templates

Wenn der Inhalt befüllt ist, kann dieser dem „email“ Objekt hinzugefügt werden. Zum Schluss muss nur noch die Verbindung zum *SMTP* Server aufgebaut werden. Danach kann die E-Mail versenden und die Verbindung zum Server abgebaut werden. Die Umsetzung ist in den Zeilen [17-21] zu finden.

```
1  public async Task SendEmailAsync(MailRequest mailRequest)
2  {
3      var email = new MimeMessage();
4      email.Sender = MailboxAddress.Parse(_mailSettings.Mail);
5      email.To.Add(MailboxAddress.Parse(mailRequest.ToEmail));
6      email.Subject = mailRequest.Subject;
7
8      var builder = new BodyBuilder();
9      var body = CreateEmailBody();
10     body = body.Replace("FullNameValue", mailRequest.Name);
11     body = body.Replace("TopicName", mailRequest.Subject);
12     body = body.Replace("MonitoredValues", mailRequest.MonitoredValues);
13     body = body.Replace("MessageValue", mailRequest.Body);
14     builder.HtmlBody = body;
15     email.Body = builder.ToMessageBody();
16
17     using var smtp = new SmtpClient();
18     smtp.Connect(_mailSettings.Host, _mailSettings.Port, SecureSocketOptions.None);
19     smtp.Authenticate(_mailSettings.Username, _mailSettings.Password);
20     await smtp.SendAsync(email);
21     smtp.Disconnect(true);
22 }
```

Listing 5.4: Funktion zum Senden einer E-Mail

5.3.4 Software Architektur

Die getroffenen Entscheidungen zur Software-Architektur bauen auf den Prinzipien von Microsoft auf [34].

Die Software-Architektur des Alarmsystems muss bei mehreren aktiven Jobs die Ressourcen gerecht verteilen können. Wenn beispielsweise nur eine SIP-Leitung verwendet werden kann, müssen sich alle Alarme, die einen Anruf tätigen wollen, sich diese Leitung teilen. Eine Leitung kann nicht mehrere Anrufe gleichzeitig tätigen, aus diesem Grund müssen die Telefonie-Jobs sequentiell abgearbeitet werden. Im Falle von mehreren SIP-Leitungen, müssen diese dennoch sequentiell benutzt werden, jedoch können hier Jobs auf die einzelnen Leitungen aufgeteilt werden und somit mehr Alarme in kürzerer Zeit aussenden.

Die REST Schnittstelle des Alarmsystems erzeugt bei jedem Aufruf einen asynchronen Thread. Die Jobs, die so gestartet werden würden, wären also asynchron. Damit Jobs dennoch sequentiell arbeiten können, muss die Ausführungslogik Multi-Thread sicher sein. Der Thread kann durch Nutzen von lock-Anweisungen gesichert werden. Die lock-Anweisung ist ein gängiger Befehl in C#, welche an einer bestimmen Stelle im Code die Ausführung sperrt.

Authentifizierung

Die Implementierung einer sicheren Authentifizierung ist für das Alarmsystem essenziell. Unbefugte Zugriffe könnten sogenannte „Nuisance Alarme“ auslösen, welche einen „Cry Wolf Effect“ erzeugen könnten, und somit die Arbeit der Alarmempfänger behindern. Um sich gegen solche Zugriffe abzusichern, ist der Einsatz von *JWT* empfohlen.

In C# können einzelne *REST* Funktionen oder Schnittstellen durch das Setzen des [Authorize] Tags von unbefugten Zugriffen gesichert werden. Dabei wird von jedem eingehenden *REST* Aufruf das Mitsenden eines gültigen *JWT*-Tokens verlangt. Wenn dieser ungültig ist, erhält der Client einen „Unauthorized“ Fehler.

5.4 Zusammenfassung

In diesem Kapitel wurde die Konzeption und Umsetzung des Alarmsystems beschrieben. Zudem wurden diverse Anwendungsfälle erläutert, interne Datenstrukturen festgelegt und Rahmenbedingungen an das System erstellt. Das vorgestellte Konzept beschreibt einen Lösungsweg um Alarmierungen über unterschiedliche Kommunikationswege ausführen zu können. Im Kapitel zur Technologieauswahl wurden verschiedene Bibliotheken zur Umsetzung der gewählten Methoden verglichen und gewählt. In der anschließenden Umsetzung wurden die wichtigsten Implementierungsdetails zur Alarmierungslogik erklärt und beschrieben. Der Nutzen des Alarmsystems soll im nachfolgenden Kapitel unter Beweis gestellt werden. Hierfür wird das Alarmsystem in ein bestehendes Monitoring-System implementiert.

Kapitel 6

Implementierung in ein bestehendes Monitoring-System

Durch den generischen Aufbau des Alarmsystems kann dieser von jeglichen Monitoring-Systemen benutzt werden. Dieses Kapitel behandelt die Implementierung des Alarm-Systems am Beispiel des bestehenden Monitoring-Systems MB PALAMAX® Pico. Neben der Konzeption wird auch dessen Umsetzung geschildert.

6.1 Konzeption

In diesem Abschnitt geht es, um die Konzeption der Erweiterung für das bestehende Monitoring-System MB PALAMAX® Pico, um die Funktionalitäten des Alarmsystems zu nutzen. Neben konzeptionellen Entscheidungen der internen Abläufe in der Anwendung werden hier auch die Designentscheidungen zum Erweitern der Weboberfläche erläutert und in Mockups visualisiert.

6.1.1 Fachkonzept

Damit das Alarmsystem in MB PALAMAX® Pico genutzt werden kann, muss es so erweitert werden, dass der Nutzer, über die erweiterte Weboberfläche, selbständig Alarmmetriken anlegen und verwalten kann. Die angelegten Metriken sollen von MB PALAMAX® Pico regelmäßig ausgewertet werden und im Fall einer erfüllten Alarmbedingung, die voreingestellte Aktion an das Alarmsystem weiterleiten, damit der Alarm ausgelöst werden kann. Um im Falle eines fehlerbedingten Neustarts nicht alle angelegten Konfigurationen zu verlieren, sollten diese persistent gespeichert werden.

Aus den vorhergegangenen Anforderungen bilden sich die folgenden Funktionalitäten, welche in MB PALAMAX® Pico eingebaut werden sollen.

- Betrachten, Anlegen, Bearbeiten und Löschen von Alarmkonfigurationen
- Übersicht über laufende Alarmierungen

- Auswerten der Bedingungen bei jeglichen Echtzeitwertänderungen

Bei der Erstellung einer Alarmbedingung ist zu beachten, dass Aktionen immer einen oder mehrere Empfänger beinhalten können, an welche die Alarmierungen adressiert werden. Da gleiche Empfänger mehrfach in anderen Konfigurationen gewählt werden können, bietet sich die persistente Haltung von Empfängern an. Somit muss ein Empfänger lediglich einmalig angelegt werden und bereichert den Nutzer bei der Konfigurationen durch die erhebliche Zeitersparnis.

Da das Betrachten, Anlegen, Bearbeiten und Löschen Aktionen sind, die von einem Nutzer ausgeführt werden sollen, ist die Erstellung von neuen Ansichten für die jeweiligen Punkte in der Weboberfläche nötig. Da diese Punkte den Oberbegriff der Alarmierung gemeinsam haben, empfiehlt sich hier die Nutzung einer visuellen Gruppierung, damit der Nutzer nicht von den neuen Funktionalitäten überwältigt wird. Diese kann beispielsweise in der Navigationsleiste eingesetzt werden, welche einen Schnellzugriff zu allen alarmbezogenen Seiten gewährt.

Damit Metriken persistent gespeichert und verwaltet werden, wird eine Datenbank benötigt. Da in MB PALAMAX® Pico bereits eine Datenbank existiert, muss diese lediglich um die neuen Daten erweitert werden. Zur Erweiterung wird eine neue Tabelle für die Haltung von allen angelegten Empfängern benötigt. Damit wird die Liste an Funktionalitäten um den folgenden Punkt erweitert: „Betrachten, Anlegen, Bearbeiten und Löschen von Alarmempfängern“.

Um konfigurierte Alarmbedingungen unterschieden zu können, soll der Nutzer einen Namen vergeben können. Damit Metriken regelmäßig evaluiert werden, muss eine angemessene Logik entwickelt werden, welche diese in allen möglichen Aggregationen zulässt. Außerdem sollen die Konfigurationen über die bestehende Weboberfläche verwaltet werden können, um dem Nutzer die Kontrolle darüber zu geben.

Da die Alarmkonfigurationen Empfängerdaten benötigen, empfiehlt sich die zusätzliche Haltung von Nutzerdaten mit entsprechenden Kommunikationsinformationen für die jeweilige Methode wie beispielsweise die E-Mail-Adresse und Telefonnummer. Neben den Metriken soll in dem Dialog zusätzlich die gewünschte Aktion und die damit verbundenen Empfänger gewählt werden können.

Damit verlängert sich die Liste an umzusetzenden Funktionalitäten nach der Evaluierung:

- Betrachten, Anlegen, Bearbeiten und Löschen von Alarmkonfigurationen
- Betrachten, Anlegen, Bearbeiten und Löschen von Alarmempfängern
- Übersicht über laufende Alarmierungen
- Auswerten der Bedingungen bei jeglichen Echtzeitwertänderungen

6.1.2 Grobarchitektur

Um die benötigten Funktionalitäten nutzen zu können, werden neue Schnittstellen für die Kommunikation über *REST* benötigt. Die jeweiligen Schnittstellen werden mithilfe von Entity Framework die benötigten *Create, Read, Update, Delete (CRUD)* Operationen durchführen.

Job Status

Um alle aktiven Jobs einsehen zu können, muss eine Verbindung zum Alarmsystem Websocket aufgebaut werden. Die daraus erhaltenen Daten sollen in dem dazugehörigen neuen Register aufgelistet werden. Da die Daten durch die mitgesendeten eindeutigen IDs identifiziert werden können, kann der Status eines Jobs durch neue Nachrichten im Websocket erneuert werden.

Durch die Echtzeitansicht der aktuellen Jobs ist es dem User möglich, über einzelne Aktionen, die Jobs zu pausieren, priorisieren oder abbrechen zu können.

Metriken

Da das System in Zukunft um neue Variablen und Kommunikationswege erweitert werden könnte, soll sich der Dialog automatisch auf diese Änderungen anpassen können, damit der Benutzer immer die aktuellste Konfiguration erstellen kann.

Damit die Metriken regelmäßig ausgewertet werden können, wird eine Funktionalität benötigt, welche die festgelegten Metriken aus der Datenbank ausliest und überprüft, sobald neue Maschinenwerte erfasst werden. Für die ausgelösten Metriken sollen anschließend die konfigurierte Aktion ausgeführt werden, in dem der Trigger an das Alarmsystem übermittelt wird. Dieses Szenario wird zur Vereinfachung in Abbildung 6.1 dargestellt.

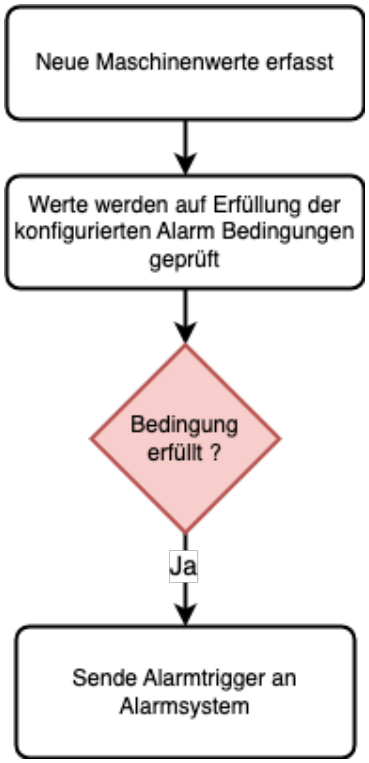


Abbildung 6.1: Auswerten der Metriken

Um Metriken besonders generisch serialisieren und deserialisieren zu können, kann eine eigenständige Bibliothek entwickelt werden. Diese sollte dazu in der Lage sein mehrere Kombinationen von Metriken erstellen zu können. Eine fertige externe Lösung zu dieser Logik existiert zum Stand dieser Bachelorarbeit nicht. Daher musste zunächst evaluiert werden, wie dieses Problem am effizientesten gelöst werden kann, um anschließend eine geeignete Lösung zu entwickeln.

6.1.3 Systementwurf

Das Monitoring-System MB PALAMAX® Pico wurde in der Programmiersprache C# entwickelt und baut, ebenso wie das Alarmsystem, auf dem ASP.NET Core Framework auf. Die Weboberfläche dieser Anwendung ist mit der Bibliothek React.js entwickelt worden. Zum Verwalten der PostgreSQL Datenbank wird auch hier EF-Core genutzt.

Datenbankmodell

Um die im Fachkonzept erwähnte persistente Speicherung von Alarmkonfigurationen und Empfängern zu gewährleisten, muss das bestehende Datenbankmodell abgeändert werden. Ein Empfänger wird nur im Kontext der Alarmierung benötigt und muss somit nur einen Namen zur Identifizierung besitzen und die jeweils geforderten Informationen zur Alarmierung. Da in der Technologieauswahl im Kapitel 5.2 die Alarmierung über Telefonie und E-Mail gewählt wurde, benötigt die Tabelle „User“ die E-Mail-Adresse und die Telefonnummer als weitere Attribute.

Eine Alarmierung besteht aus einem Namen, der Bedingung und dem Aktionstypen. Da in einer Datenbank keine komplexen Objekte abgelegt werden können, die genaue Bedingung jedoch abgespeichert werden muss, eignet sich die Speicherung als JSON-Objekt, welches in der Datenbank als „string“ abgelegt werden kann. Für die Alarmierung wird die Tabelle mit dem Namen „Alert“ angelegt.

Da eine Alarmierung mehrere Empfänger beinhalten, und ein Empfänger zu mehreren Alarmierungen zugeteilt sein kann, wird eine Zwischentabelle benötigt, um diese Beziehung abzubilden. Die Zwischentabelle wird von EF-Core automatisch angelegt. Der Tabellenname setzt sich dabei immer aus den beiden verknüpften Tabellennamen zusammen und lautet somit „AlertUser“.

Das alte Modell bleibt vollständig erhalten und wird lediglich um neue Tabellen erweitert. Der Datenbankzusatz wird in der Abbildung 6.2 dargestellt.

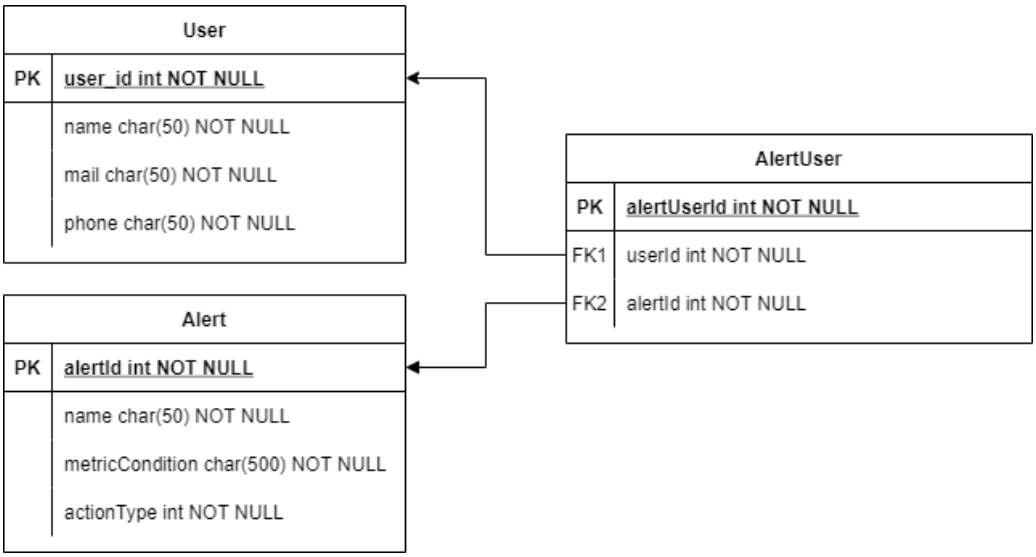


Abbildung 6.2: MB PALAMAX® Pico Datenbankmodell Erweiterung

6.1.4 Entwurf der Benutzeroberfläche

Damit die neuen Funktionalitäten genutzt werden können, muss die Weboberfläche von MB PALAMAX® Pico um die erforderlichen Bedienelemente erweitert werden. Die neuen Elemente sollen für den Nutzer intuitiv sein und die grundlegenden Entwurfsprinzipien guter Benutzerfreundlichkeit erfüllen [35, S. 203-204]. Daher widmet sich dieser Teilabschnitt dem Entwurf der Benutzeroberfläche, in welchem Designentscheidungen getroffen und Mockups dargestellt werden.

Nutzerkonfiguration

Wie bereits im Fachkonzept 6.1.1 beschrieben, wird das Betrachten, Anlegen, Bearbeiten und Löschen von Nutzern benötigt. Um diese Funktionalitäten nutzen zu können, sollte hierfür eine eigene Seite erstellt werden, um das Entwurfsprinzip „Strukturierung der Benutzungsschnittstelle“ zu erfüllen. Dieses besagt, dass Bedienelemente zusammengefasst visualisiert werden sollen, damit diese besonders effizient von einem Menschen verarbeitet werden können [35, S. 209].

Zum Betrachten der „User“ eignet sich die Auflistung in einer seitenbasierten Tabelle. Die einzelnen Elemente können über diese Tabelle bearbeitet und gelöscht werden, indem in der letzten Spalte die Aktionsknöpfe aufgelistet werden. Da das Anlegen von Nutzern unabhängig von einzelnen Elementen in der Tabelle ist, sollte der Aktionsknopf außerhalb der Tabelle platziert werden. Dafür eignet sich eine Position oberhalb des Tabellenkopfes, da dieser im angenehmen Sehbereich liegt. Der Aufbau der Seite wird in Abbildung 6.9 dargestellt.

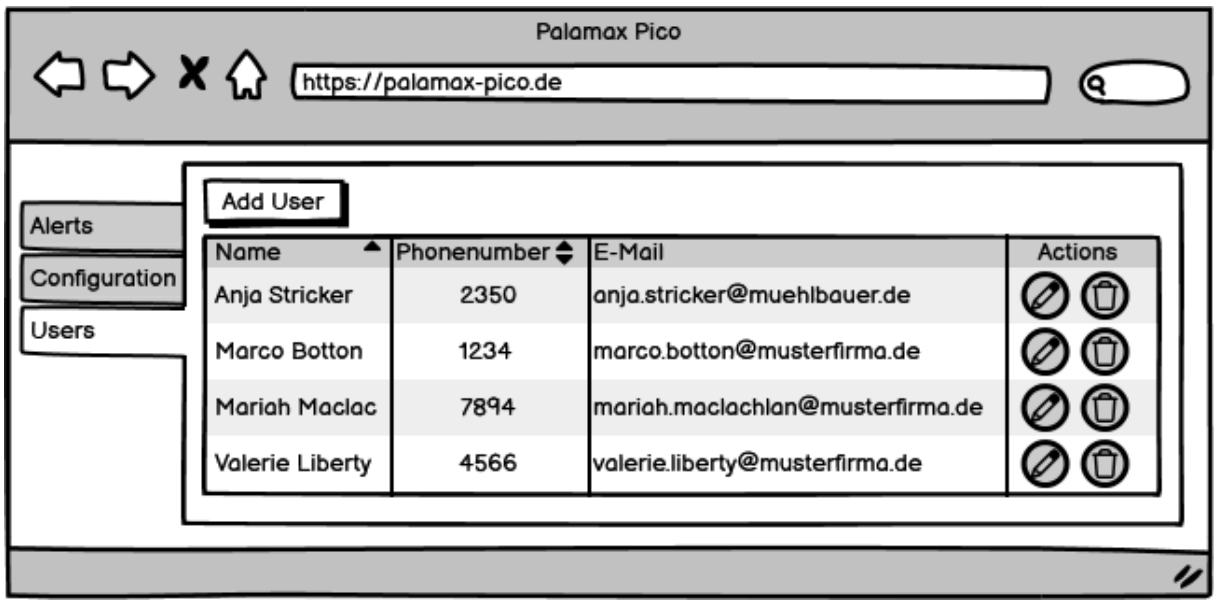


Abbildung 6.3: Nutzerkonfigurations-Auflistung

Damit der Nutzer einen „User“ anlegen und bearbeiten kann, wird ein Dialog benötigt. Hierbei wird für jedes der drei Attribute ein Textfeld angelegt. Für die Einhaltung des Entwurfsprinzips „Berücksichtigung von Fehlern“ muss die Eingabe der E-Mail-

Adresse durch eine Validierung kontrolliert werden, damit der Nutzer darauf hingewiesen wird, wenn eine ungültige Eingabe erfolgt ist. Dabei sollte sich das betreffende Textfeld rot färben, und den Aktionsknopf zum Bestätigen der Aktion ausblenden.

Abbildung 6.4: Nutzer Anlegen Dialog

Alarmkonfiguration

Das Anzeigen von Alarmkonfigurationen kann ebenfalls im Tabellenformat erfolgen, welche in Abbildung 6.5 veranschaulicht wird. Auch hier wird die Funktionalität zum Löschen der einzelnen Einträge durch die Aktionsleiste ermöglicht. Die Bedingung einer Alarmkonfiguration sollte nicht in der Tabelle angezeigt werden, da die Darstellung zu komplex wäre und den Nutzer nur unnötig kognitiv belasten würde. Da es dem Nutzer dennoch wichtig ist, die Bedingung nach der Erstellung einsehen zu können, kann ein weiterer Aktionsknopf angelegt werden.

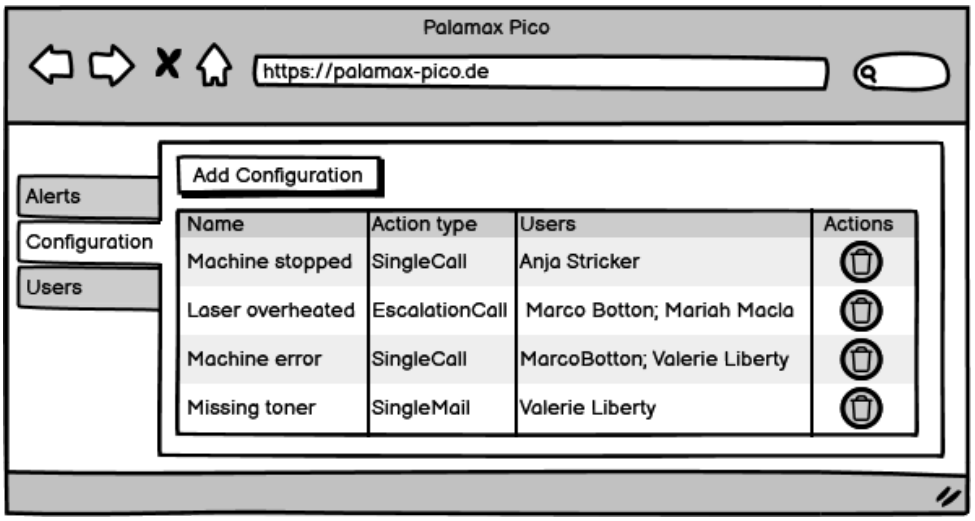


Abbildung 6.5: Auflistung von Alarmkonfigurationen als Mockup

Durch das Betätigen des Knopfes kann in einem Detail-Dialog die konfigurierte Bedingung neben den restlichen Informationen zur Alarmierung dargestellt werden.

Um eine Konfiguration anzulegen, kann über den hervorgehobenen Anlegen-Knopf ein Dialog geöffnet werden. Da das Anlegen einer Bedingung das Anzeigen vieler Bedienelemente voraussetzt, muss eine visuelle Trennung erstellt werden. Dies kann durch den Einsatz eines Stepper-Dialogs gelöst werden. Die drei Schritte „Metric“, „Action“ und „Name“ werden im oberen Bereich des Dialogs angezeigt. Jeder Schritt kann durch das Betätigen des „Next“ Knopfs beendet werden.

Damit der Nutzer die angelegten Alarmierungen leichter unterscheiden kann, kann im ersten Bereich des Stepper-Dialogs ein Name für die Alarmierung vergeben werden. Die geplante Umsetzung ist in der Abbildung 6.6 zu sehen.

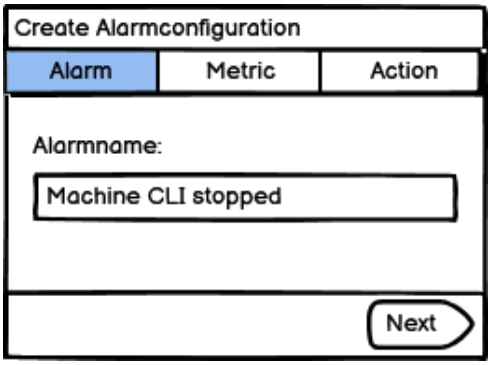


Abbildung 6.6: Vergabe des Alarmnamens im Alarmkonfigurations-Dialog Mockup

Eine Bedingung besteht aus einem Echtzeitwert und einem konstanten Wert, die mit einem Operator verknüpft werden. Durch den Einsatz einer Dropdown-Liste kann ein beliebiger Echtzeitwert gewählt werden. Eine weitere Dropdown-Liste kann für die Wahl des Operators genutzt werden. Der konstante Wert kann durch ein Textfeld eingegeben werden. Um diese Bedingung nun mit einer weiteren verknüpfen zu können, kann ein Aktionsknopf betätigt werden. Dadurch wird eine neue leere Bedingung hinzugefügt und ein Verknüpfungsoperator zwischen den beiden Bedingungen anlegt. Dadurch besteht die Möglichkeit, unendlich viele Bedingungen zu verknüpfen. Damit alle betrachtet werden können, besitzt der innere Bereich des Dialogs eine Bildlaufleiste. Der genaue Aufbau ist in der Abbildung 6.7 dargestellt.

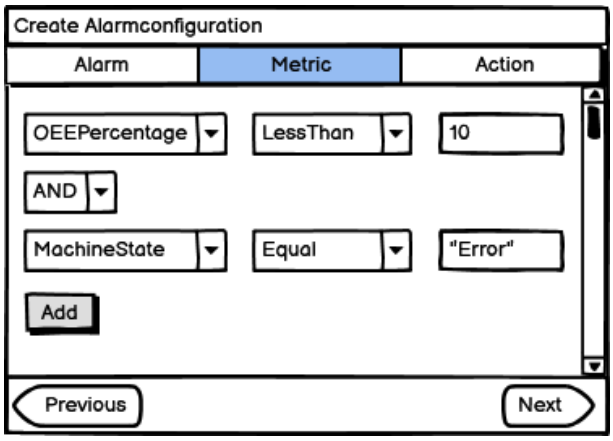


Abbildung 6.7: Konfiguration der Alarmbedingungen als Mockup

Für das Konfigurieren einer Aktion für die Alarmierung, wird neben der Wahl des Aktionstyps die Liste an Empfängern benötigt. Die Wahl des Aktionstyps kann erneut durch den Einsatz einer Dropdown-Liste getätigt werden. Da die Empfänger in der „User-Seite“ angelegt werden, müssen in diesem nur die benötigten Empfänger gewählt werden. Durch Betätigen des „Add User“ Buttons, wird ein Hilfs-Dialog geöffnet, welche in einer Dropdown-Liste alle verfügbaren Empfänger auflistet. Durch den Klick auf die gewünschte Person wird diese der Empfängerliste hinzugefügt. Zum Schluss wird mit dem Aktionsknopf „Save Metric“ die Konfiguration gesichert, und der Dialog schließt sich. Das Mockup 6.8 zeigt dies mit Beispieldaten.

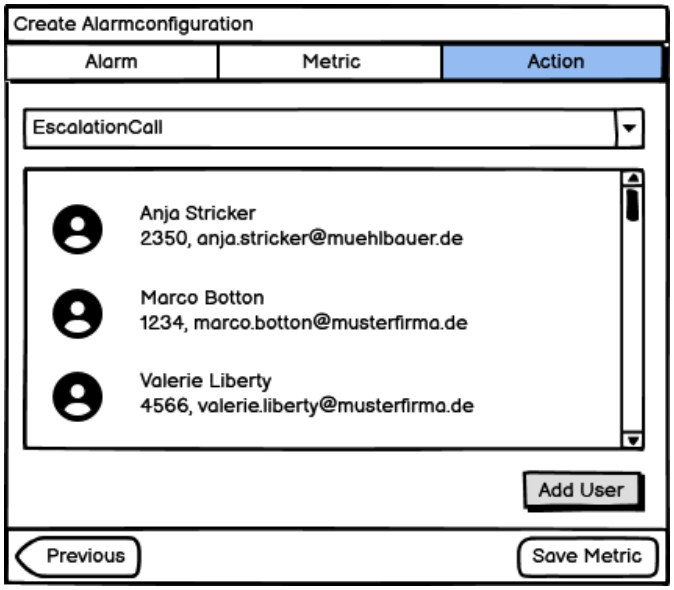


Abbildung 6.8: Wählen der Alarmaktionen im Alarmkonfigurations-Dialog Mockup

Job Status

Da die Überwachung der laufenden Jobs keine Funktionalitäten zum Anlegen und Bearbeiten bieten kann, können erweiterte Tabellenfunktionen in dieser Seite wegfallen. Es genügt eine Auflistung aller aktiven und vergangenen Jobs mit den zugehörigen Informationen. Da der primäre Fokus dieser Übersichtsseite die Überwachung der Status ist, kann dieser hervorgehoben werden, um die Aufmerksamkeit darauf zu lenken und Änderungen schneller wahrzunehmen. In der nachfolgenden Abbildung 6.9 wird die Auflistung der Jobs dargestellt.

AlarmId	Name	State	Type	Start Timestamp	Last Edit Timestamp
2	Machine CLI stopped	Running	SingleCall	1282787629	1578688195
8	Laser overheated	Queued	EscalationCall	1845261978	1674085060
21	Machine error	Queued	SingleCall	1468320938	1446408190
8	Missing toner	Finished	SingleMail	961817149	1848661940

Abbildung 6.9: Job Status Auflistung

6.2 Technische Umsetzung

Durch die erstellte Konzeption kann im folgenden Kapitel auf die technische Umsetzung des Alarmsystems und der dazugehörigen Subsysteme eingegangen werden. Neben der Erstellung von neuen Dialogen und Ansichten für die Weboberfläche werden spezielle interne Logiken vorgestellt.

Da die Weboberfläche durch den Einsatz der „MUI“ Gestaltungsbibliothek umgesetzt worden ist, werden die neuen Bedienelemente ebenfalls mit dieser Bibliothek angefertigt [36]. Diese liefert zusätzliche Funktionalitäten für Tabellen mit, die beispielsweise das Sortieren für jede Spalte erlaubt oder eine Kopfzeile mit generellen Tabellenaktionen anbietet. Zusätzlich werden bei der Umsetzung der neuen Ansichten die Designvorgaben der Firma Mühlbauer & Co. KG zur Bewahrung der Corporate Identity befolgt. Die Umsetzung der neuen Seiten und Dialoge für die Weboberfläche basiert auf den Mockups aus Kapitel 6.1.4 und wird in den nachfolgenden Unterkapiteln beschrieben.

6.2.1 Nutzer Konfiguration

Auf Basis der erstellen Mockups wurde die Nutzeransicht und Nutzerkonfiguration entwickelt. Die Abbildung 6.10 zeigt die Umsetzung der Nutzer Auflistung. Im oberen Bereich befinden sich die generellen Aktionsmöglichkeiten. Der erste Knopf „Add User“ öffnet einen Dialog, mit welchem ein neuer Nutzer angelegt werden kann. Die nachfolgenden Aktionsknöpfe werden von „MUI“ bereitgestellt und ermöglichen einzelne Spalten ein- und auszublenden, die Zeilenhöhe anzupassen und die aktuell angezeigte Tabellenseite zu exportieren. Zum Anzeigen der Tabelle wird die „DataGrid“ Komponente genutzt [37].

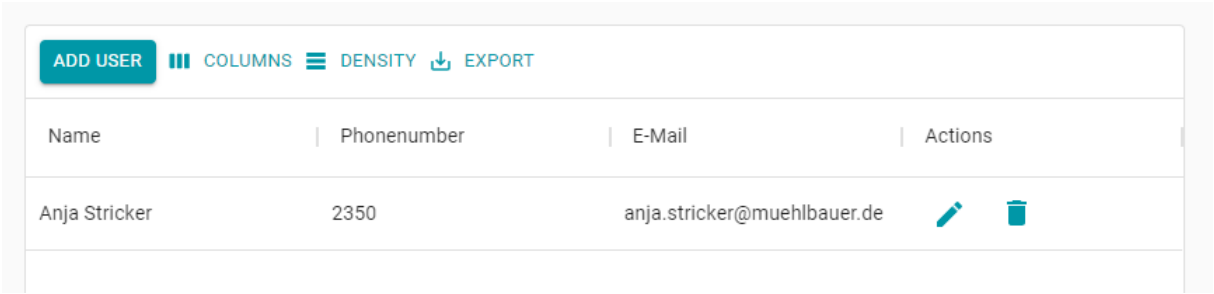


Abbildung 6.10: Nutzer Konfiguration

Betätigt der Nutzer den „Add User“ Knopf, so öffnet sich der Dialog zum Anlegen eines Nutzers. Der Dialog wird mithilfe der „Dialog“ Komponente umgesetzt [38]. Beim Öffnen des Dialogs wird ein sogenannter „Backdrop Shadow“ erzeugt, welches den Dialog hervorhebt und einen Schatten über die restlichen Elemente im Hintergrund legt. Dies ist in der Abbildung 6.11 zu sehen. Um einen Nutzer anlegen zu können, muss ein Name, eine Telefonnummer und eine E-Mail vergeben werden. Durch das Betätigen des „Add“ Knopfes wird der Nutzer angelegt und der Dialog geschlossen.

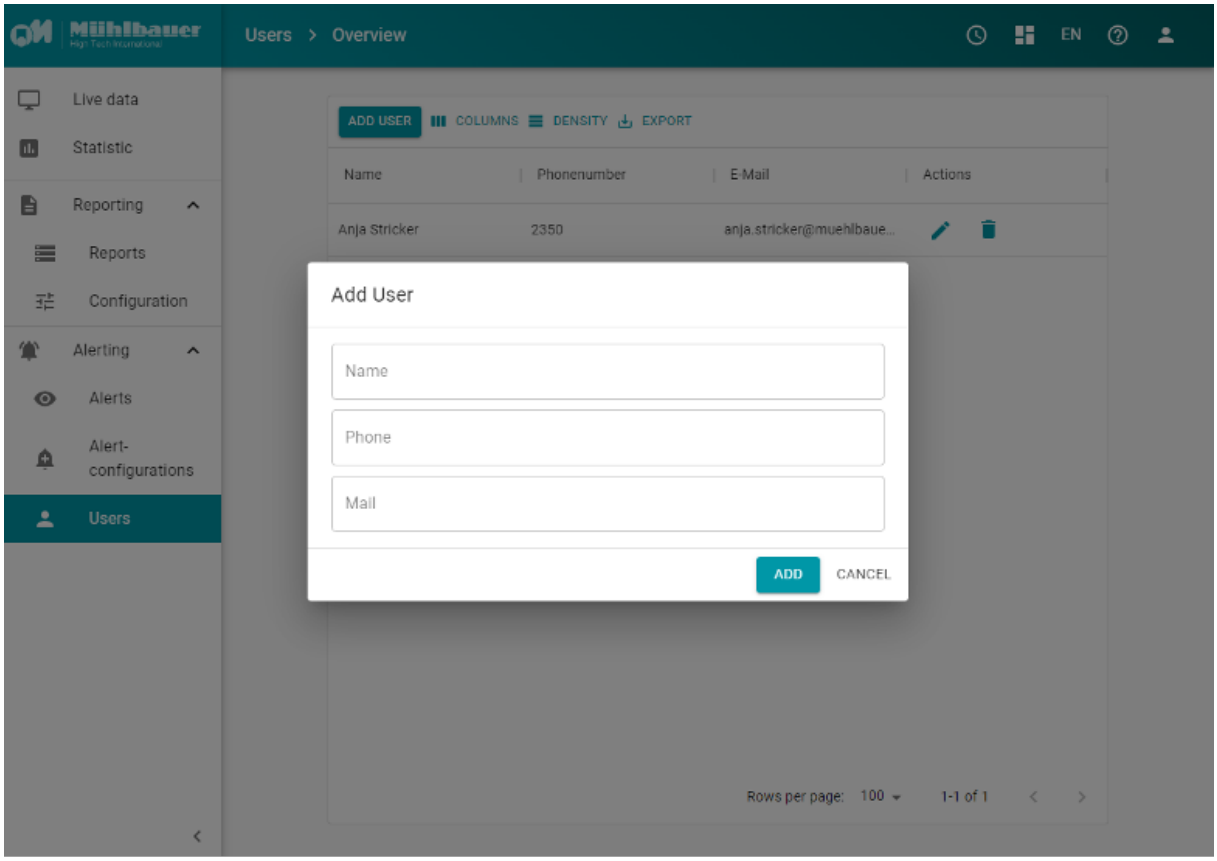


Abbildung 6.11: Nutzer Dialog

6.2.2 Alarm Konfiguration

In diesem Abschnitt wird die Umsetzung zur Konfiguration und Überwachung eines Alarms beschrieben. Dabei werden sowohl die Programmabläufe, als auch die grafische Umsetzung der zusätzlichen Ansichten beschrieben.

Damit die erstellten Konfigurationen angelegt, betrachtet und gelöscht werden können, muss eine weitere Übersichtsseite erstellt werden. Hierfür eignet sich erneut der Einsatz einer seitenbasierten Tabelle. Hierbei werden neben dem Namen der Alarmierung, der Typ des Alarms und die Liste an zu alarmierenden Nutzern angezeigt. Der Nutzer hat zudem die Möglichkeit Konfigurationen, über den Interaktionsknopf in der entsprechenden Zeile, zu löschen.

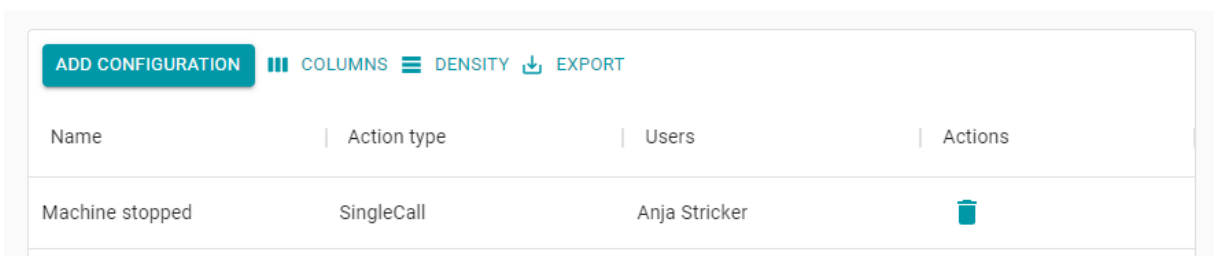


Abbildung 6.12: Alarm Konfigurations-Übersicht

Für den Nutzer soll das Anlegen von Metriken so einfach wie möglich sein. Damit nicht nur einfache Bedingungen, sondern auch komplexe Aggregationen möglich sind, muss eine besondere Funktionalität entwickelt werden. Außerdem soll es möglich sein, Metriken beliebig oft verketten zu können. Weiterhin muss bei der Implementierung darauf geachtet werden, wie die Konfigurationsdaten vom Frontend an das Backend gesendet werden, und wie diese sinnvoll konvertiert werden können, damit diese sowohl in der Datenbank abgespeichert, als auch vom System zur Überprüfung von Werten genutzt werden kann. Es stellt sich also das Problem von effektiver Serialisierung und Deserialisierung.

In C# gibt es sogenannte „Expression Trees“ welche für diesen Nutzen eingesetzt werden können. „Expression Trees“ stellen Code in einer baumartigen Datenstruktur dar, bei der jeder Knoten ein Ausdruck ist. Der Ausdruck kann ein Methodenaufruf oder eine binäre Operation, wie beispielsweise `x < y`, sein. Der Code, der als „Expression Tree“ dargestellt wird, kann kompiliert und ausgeführt werden. Dies ermöglicht die Erstellung von dynamischen Abfragen und kann somit zur Definition von Alarmierungsbedingungen genutzt werden. [39]

Eine einzelne Bedingung zählt als „BinaryExpression“, da sie zwei Werte mit einem binären Operator vergleicht. Will man nun mehrere „Expressions“ durch ein **UND** oder **ODER** verketten, werden diese zu „BooleanExpressions“. Die Abbildung 6.13 zeigt dies in Form eines „Binary Trees“. Der erste Knoten ist ein Verknüpfungs-Operator und besitzt als rechten Kindknoten die „BinaryExpression“ `B < 10`. Der linke Kindknoten ist eine BooleanExpression, da sie die zwei „BinaryExpressions“ `P < 20` und `C == 3` mit einem **ODER** verknüpft.

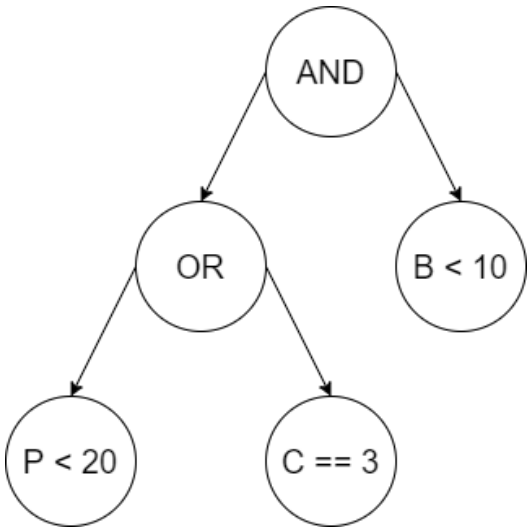


Abbildung 6.13: BooleanExpression in Form eines Binary Trees

Da das Auswerten von Alarm-Bedingungen als eine Metrik bezeichnet werden kann, wird die Klasse `MetricCondition` genannt. Diese kann entweder aus einer einzelnen oder einer verketteten Bedingung bestehen. Eine einzelne Bedingung wird als `MetricBinaryCondition` bezeichnet, da diese einen binären Vergleichsoperatoren enthält.

Wenn der Ausdruck eine `MetricBinaryCondition` enthält, so wird diese mit einem logischen Operator verknüpft. Dies wird als `MetricBooleanCondition` bezeichnet. Die Beziehung zwischen den Klassen wird in dem nachfolgenden Code-Beispiel 6.1 dargestellt.

```
1  public class MetricCondition
2  {/* empty */}
3
4  public class MetricBinaryCondition : MetricCondition
5  {
6      public MetricValue Left { get; set; }
7      public MetricValue Right { get; set; }
8      public Operation Operation { get; set; }
9  }
10
11 public class MetricBooleanCondition : MetricCondition
12 {
13     public MetricCondition Left { get; set; }
14     public MetricCondition Right { get; set; }
15     public LinkedType LinkedOperation { get; set; }
16 }
```

Listing 6.1: Metrik Klassen

Wenn der Nutzer nur eine Bedingungszeile im Konfigurationsdialog befüllt, entsteht lediglich eine `MetricBinaryCondition`. Beim Befüllen von mindestens zwei Bedingungszeilen wird eine `MetricBooleanCondition` erzeugt. Demnach ist es dem Nutzer möglich, zwei unterschiedliche Typen einer Bedingung zu erstellen. Diese beiden Varianten muss vom Backend korrekt behandelt werden. Deswegen wird eine Funktion benötigt, welche je nach Typ der Kondition, eine geeignete Metrik erstellt. Dies kann, wie im unterstehenden Code-Beispiel 6.2 beschrieben, implementiert werden.

```
1  static Expression MakeExpressionByType(
2      ParameterExpression parameterExpression,
3      MetricCondition metricCondition
4      )
5  {
6      if (metricCondition is MetricBinaryCondition metricBinaryCondition)
7          return MakeBinaryExpression(parameterExpression, metricBinaryCondition);
8      else if (metricCondition is MetricBooleanCondition metricBooleanCondition)
9          return MakeBooleanExpression(parameterExpression, metricBooleanCondition);
10
11     throw new NotImplementedException();
12 }
```

Listing 6.2: Diese Funktion erstellt eine Expression je nach Komplexität der Bedingung

Die Funktion „MakeBooleanExpression“ aus dem Listing 6.3 wird im Falle einer verketteten Metrik benutzt. Dabei können beide Werte sowohl eine `BooleanExpression` als auch eine `BinaryExpression` sein. Die korrekte Erzeugung dieser Expressions erfolgt

in der „MakeExpressionByType“ Methode. In „MakeBooleanExpression“ muss die linke und rechte Bedingung lediglich mit dem gewählten logischen Operator verknüpft werden.

```
var left = MakeExpressionByType(parameterExpression, metricBooleanCondition.Left);
var right = MakeExpressionByType(parameterExpression, metricBooleanCondition.Right);

return metricBooleanCondition.LinkedOperation switch
{
  LinkedType.And => Expression.And(left, right),
  LinkedType.Or => Expression.Or(left, right),
  _ => throw new NotImplementedException(),
};
```

Listing 6.3: Die MakeBooleanExpression Methode erzeugt BooleanExpressions durch die Verknüpfung von zwei Bedingungen durch einen logischen Operator

Ist die Bedingung eine `BinaryExpression`, so muss für die linke Seite der Bedingung der genaue Typ definiert werden, um einen korrekten Ausdruck zu bilden. Ein nicht korrekter Ausdruck wäre beispielsweise der Vergleich zwischen zwei unterschiedlichen Typen. Dieser Ausdruck `,,False‘‘ == false` würde einen `InvalidOperationException` Fehler werfen, da der string „False“ nicht mit dem Boolean `false` verglichen werden kann, dieser Fehler kann durch eine Typumwandlung vermieden werden. Nach der Typumwandlung können die beiden Werte der Bedingung durch den gewählten logischen Operator verglichen werden. Die Umsetzung dieser Logik kann dem Listing 6.4 entnommen werden.

```
var leftType = metricBinaryCondition.Left.Type switch {
    Type.Number => typeof(double),
    Type.String => typeof(string),
    Type.Boolean => typeof(bool),
    _ => throw new NotImplementedException(),
};

var left = Expression.Convert(MakeMetricValueExpression(
        parameterExpression, metricBinaryCondition.Left), leftType);

return metricBinaryCondition.Operation switch {
    Operation.GreaterThan => Expression.GreaterThan(left, right),
    Operation.LessThan => Expression.LessThan(left, right),
    Operation.Equal => Expression.Equal(left, right),
    _ => throw new NotImplementedException(),
};
```

Listing 6.4: Die MakeBinaryExpression verknüpft den linken umgewandelten Parameter und den rechten Wert mit einem binären Vergleichsoperator

In TypeScript gibt es keine ExpressionTrees. Es muss also ein Datenformat gefunden werden, welches nach dem Empfang in der REST Schnittstelle in einen entsprechenden

Expression Tree umgewandelt werden kann. Zudem muss beachtet werden, dass diese Information in einer geeigneten Form in der Datenbank abgelegt werden muss um diese auch nach bei einem Programmneustart laden zu können. Ebenfalls zu beachten ist, wie die Auswertung einer Bedingung mithilfe der ExpressionTrees vollzogen werden kann. Damit die Klasse möglichst dynamisch Daten evaluieren kann, ist es sinnvoll eine Auswertung über Pfade zu implementieren. In der Konfiguration der Bedingung muss daher nur der Pfad übergeben werden, welcher zur Auswertung herangezogen werden kann. Wenn die Echtzeitdaten in Form eines JsonDocuments übergeben werden, kann durch den Pfad die Variable gefunden werden und in den ExpressionTree übergeben werden. Da es keine automatische Typenerkennung gibt, muss die Konvertierung zuvor in der Funktion definiert werden, welches im nachfolgenden Code-Beispiel 6.5 beschrieben ist.

```
1  public static object AccessJsonDocumentValue(string[] path, JsonElement jsonDocument)
2  {
3      if (path.Length == 0)
4          return jsonDocument.ValueKind switch
5          {
6              JsonValueKind.Number => jsonDocument.GetDouble(),
7              JsonValueKind.True => true,
8              JsonValueKind.False => false,
9              JsonValueKind.String => jsonDocument.GetString(),
10             _ => throw new NotImplementedException()
11         };
12     else
13         return AccessJsonDocumentValue(
14             path.Skip(1).ToArray(),
15             jsonDocument.GetProperty(path.First()));
16 }
```

Listing 6.5: Umwandeln der JsonDocument Daten

Neben der Erstellung der Metriken durch eine Konfiguration muss auch eine Funktion zur regelmäßigen Auswertung von Metriken mit Echtzeitdaten erstellt werden. Jede Metrik kann nach der Übergabe von Echtzeitdaten ausgeführt werden. Nach dem Erhalt von neuen Echtzeitdaten müssen alle konfigurierten Metriken ausgewertet werden, dies kann mit einer einfachen Schleife erreicht werden. Wenn eine Metrik den Wert „True“ zurückgibt, ist die Metrik erfüllt und muss im späteren Programmablauf in einer Alarmierung resultieren.

Das System benötigt für die Auswertung der Alarmierung eine Funktion, welche die Metrik auf dessen Erfüllung überprüft. Da die Information über die zu alarmierenden Metriken reicht, wird nur eine Liste an AlarmIDs benötigt, für welche die Metrik zugetroffen hat. Da die Methoden der Klassen keine Abhängigkeiten zum Monitoring-System aufweisen, kann es als eigenständige C# *Dynamic Link Library (DLL)* entwickelt werden, welche in beliebigen Monitoring-System eingesetzt werden kann. Diese wird in den folgenden Abschnitten als „MetricMonitor.dll“ bezeichnet.

Konfiguration eines Alarms

Damit der Nutzer komplexe Bedingungen einfach konfigurieren kann, wird ein Dialog benötigt, welcher ansprechend aussieht, um einer visuellen Überforderung entgegenzuwirken. Bei der Konfiguration werden zwei grundlegende Informationen definiert. Die Bedingung der Alarmierung, und die Ausführung der Alarmierung. Diese zwei Bereiche können durch den Einsatz von sogenannten „Stepper Dialogen“ visuell getrennt werden. Die Mockups 6.6, 6.7 und 6.8 aus Kapitel 6.1.4 stellen die einzelnen Seiten des Stepper-Dialoges dar.

Das Layout dieses Dialogs kann durch den Einsatz einer „component library“ umgesetzt werden. MB PALAMAX Pico nutzt hierfür die UI Bibliothek „MUI“. Deswegen werden die neuen Dialoge mithilfe der von „MUI“ bereitgestellten Komponenten umgesetzt [36]. Damit der Nutzer Alarme unterscheiden kann, besitzt der erste Stepper-Bereich ein Textfeld, mit dem der Name des Alarms definiert werden kann.

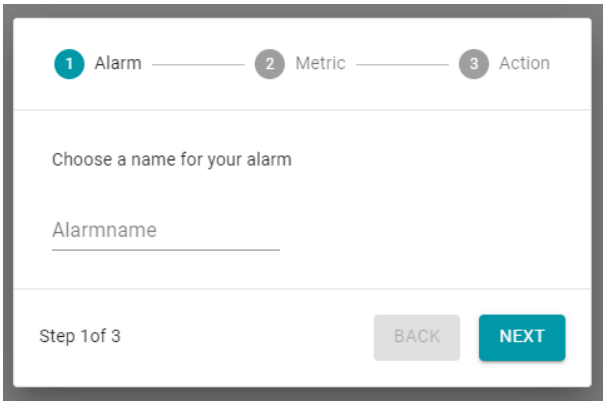


Abbildung 6.14: Erster Abschnitt im Dialog

Im zweiten Bereich kann der Nutzer die Bedingung konfigurieren. Wenn eine neue Bedingung hinzugefügt werden soll, kann dies über den „ADD“-Button erzeugt werden. Damit fügen sich eine neue Bedingungszeile und ein Operator, der die letzten beiden Bedingungen verknüpft, an.

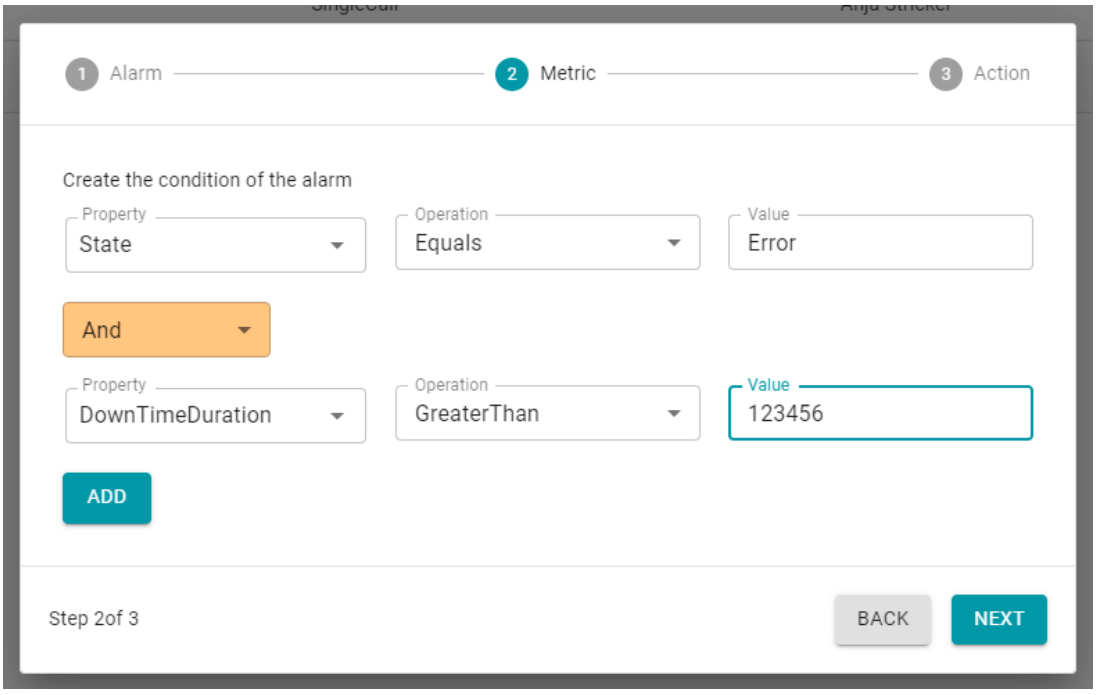


Abbildung 6.15: Alarm Konfiguration

Die Verknüpfungs-Operatoren zwischen Bedingungen werden farblich hervorgehoben, um die Übersichtlichkeit der Liste zu bewahren. Wird der vertikale Bereich des Stepper-Dialogs voll, erscheint ein Scrollbalken, um die Leserlichkeit beizubehalten. Damit behält der Dialog stets die gleiche Höhe.

Der dritte und letzte Abschnitt im Dialog bietet die Wahl der Aktion. Für die Wahl eines Aktions-Typen wird eine Dropdown-Liste positioniert. Die Aktion beinhaltet immer einen oder mehrere Empfänger, daher wird die Option zum Hinzufügen von bereits definierten Empfängern bereitgestellt. Um einen neuen Empfänger hinzufügen zu können, muss der Nutzer Dialog aus der Nutzer-Seite geöffnet werden, welcher in Kapitel 6.2.1 vorgestellt wurde.

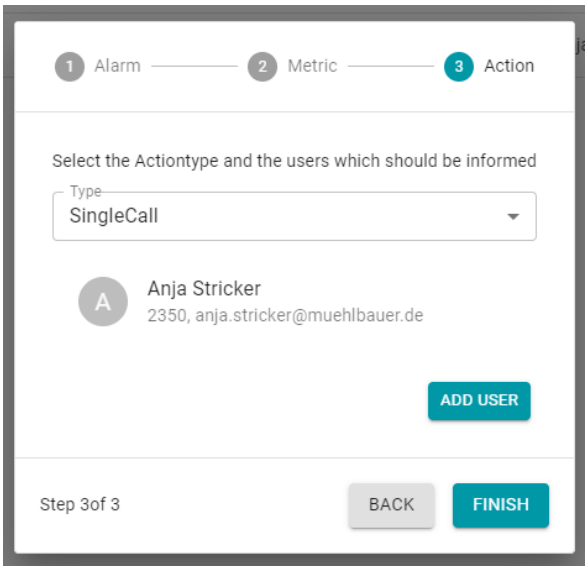


Abbildung 6.16: Alarm Konfiguration

Wenn alle Bereiche ausgefüllt sind, kann der „FINISH“-Button betätigt werden, um den konfigurierten Alarm zu sichern. Dabei werden die Daten über die REST-Schnittstelle an das Backend weitergegeben. Um den Alarm in der Datenbank sichern zu können, muss die Metrik in ein anderes Format umgewandelt werden. Hierfür eignet sich die Speicherung als JSON im Textformat. Um das Metrik-Objekt umzuwandeln, kann die Methode `JsonConvert.SerializeObject` genutzt werden. Die Umsetzung ist im Listing 6.6 beschrieben.

```
1 var converted = new AlertConfiguration()
2 {
3     Name = value.Name,
4     MetricCondition = JsonConvert.SerializeObject(value.MetricCondition),
5     ActionType = value.Action.ActionType,
6     Users = new List<User>()
7 };
```

Listing 6.6: Vorbereiten des Alert-Objekts zur Sicherung in der Datenbank in der POST Methode des AlertsControllers

Da die User bereits in der Datenbank existieren, muss die User-Liste mit den passenden Datenbankeinträgen gefüllt werden. Danach kann die konvertierte Alarmkonfiguration in der Datenbank abgespeichert werden. Dies ist in den Zeilen [1-8] im Listing 6.7 zu sehen. Zum Schluss wird die Metrik dem MetricMonitor hinzugefügt. Nachdem der Alarm angelegt wurde, wird dieser direkt in der Auswertung der Echtzeitdaten berücksichtigt und ist somit sofort einsatzbereit.

```
1  foreach (var item in value.Action.UserIds)
2  {
3      var user = _picoContext.Users.Single(x => x.Id == item);
4      converted.Users.Add(user);
5  }
6
7  _picoContext.Alerts.Add(converted);
8  await _picoContext.SaveChangesAsync();
9
10 // save in metric monitor
11 var condition = value.MetricCondition;
12 _metricMonitor.AddMetric(condition, value.Name, converted.Id);
13
14 return Ok(converted.Id);
```

Listing 6.7: Sichern der Alarmkonfiguration und Anfügen der Metrik im Metrikmonitor - POST Methode im AlertsController

Logik zum Erstellen einer Metrik

Da die MetricMonitor.dll nur eine bestimmte Objektstruktur entgegennehmen kann, muss im Frontend eine erweiterte Logik zum Erstellen von Metriken eingebaut werden. Um Binary-Trees in TypeScript erzeugen zu können, ist die Anwendung des Binary-Tree-Algorithmus notwendig.

Um die Alarmbedingungen regelkonform aufbauen zu können, müssen die Grundsätze der Booleschen Algebra in der Informatik befolgt werden. Viele Programmiersprachen wie beispielsweise Java [40, S.22] oder C++ [41, S.93] benutzen die gleiche Operatorrangfolge. Die nachfolgende Liste zeigt die Operatorrangfolge für den AND und OR-Operator absteigend sortiert. Durch das Setzen von Klammern `()` kann die Reihenfolge der Auswertung verändert werden.

1. Logischer AND-Operator `&&`
2. Logischer OR-Operator `||`

Da zur Konfiguration nur die bedingten logischen Operatoren AND und OR zur Verfügung stehen, muss bei der Auswertung nur auf die höhere Rangfolge der AND-Operatoren geachtet werden, welche von links nach rechts aufgelöst werden [42]. Wenn keine AND-Operatoren mehr existieren, werden die OR-Operatoren gleichermaßen aufgelöst.

In dem nachfolgenden Berechnungsbeispiel wird der Ausdruck unter Anwendung der logischen Operatorrangfolge aufgelöst, wobei A=4, B=5 und C=3 ist.

1. `B <= 10 && C == 2 || A >= 3 && C != 2`
2. `(B <= 10 && C == 2) || A >= 3 && C != 2`
3. `(True && C == 2) || A >= 3 && C != 2`
4. `(True && False) || A >= 3 && C != 2`
5. `False || (A >= 3 && C != 2)`
6. `False || (True && C != 2)`
7. `False || (True && True)`
8. `False || True`
9. `True`

Da die Konfiguration aus einer Liste von Bedingungen und Operatoren besteht, muss die Metrik-Struktur nachträglich erstellt werden. Der Algorithmus muss rückwärts erfolgen, damit der Baum von außen nach innen gebaut werden kann. Folglich werden erst alle OR-Operatoren von rechts nach links abgearbeitet. Für jede Verknüpfung wird der verbliebene Ausdruck aufgeteilt, um die Kindknoten zu erstellen. Dies geschieht so lange, bis keine OR-Operatoren mehr übrig sind. Danach werden alle AND-Operatoren gleichermaßen abgearbeitet. Die Logik wird erneut am selben Ausdruck erläutert.

`B <= 10 && C == 2 || A >= 3 && C != 2`

Es gibt lediglich einen OR-Operator. Der erste Knoten wird zu einem OR-Operator und besitzt als linken Knotenkind den Ausdruck `B <= 10 && C == 2` und als rechten Knotenkind den Ausdruck `A >= 3 && C != 2`

`B <= 10 && C == 2`

Der erste Knoten wird zu einem AND-Operator und besitzt als linken Knotenkind den Ausdruck `B <= 10` und als rechten Knotenkind den Ausdruck `C == 2`. Da es keine weiteren logischen Operatoren gibt ist diese Seite des BinaryTrees vollständig ausformuliert.

`A >= 3 && C != 2`

Der erste Knoten wird zu einem AND-Operator und besitzt als linken Knotenkind den Ausdruck `A >= 3` und als rechten Knotenkind den Ausdruck `C != 2`. Auch hier gibt es keine weiteren logischen Operatoren.

Die zuvor beschriebene Logik ist im nachfolgenden Code-Beispiel 6.8 umgesetzt. Die Methode „generateMetric“ wird so lange rekursiv aufgerufen, bis keine Operatoren mehr im „operations“ Array übrig sind. Die Operationen werden als Zahlen dargestellt,

wobei ein OR-Operator die Zahl 1 ist und ein AND-Operator die Zahl 0 ist. Die Zahlen definieren die Rangfolge des Operators.

```
generateMetric(conditions:MetricBinaryCondition[], operations:number[])
: MetricBooleanCondition
{
    const deepMetric = {} as MetricBooleanCondition;
    if (operations.length > 0) { // checking if any operations exist
      let index = operations.lastIndexOf(1);
      if (index >= 0) { // this checks for 'or' statements
        deepMetric.linkedOperation = 1;
      } else if (operations.lastIndexOf(0) >= 0) {
        index = operations.lastIndexOf(0);
        deepMetric.linkedOperation = 0;
      }

      const splitConditionsLeft = conditions.slice(0, index + 1);
      deepMetric.left = splitConditionsLeft.length == 1 ? splitConditionsLeft[0] :
        this.generateMetric(splitConditionsLeft, operations.slice(0, index));

      const splitConditionsRight = conditions.slice(index + 1, conditions.length);
      deepMetric.right = splitConditionsRight.length == 1 ? splitConditionsRight[0] :
        this.generateMetric(splitConditionsRight,
                operations.slice(index + 1, operations.length));
      return deepMetric;
    }
    return deepMetric;
}
```

Listing 6.8: Metrik Generierung im Frontend

Durch Anwenden des oben beschriebenen Algorithmus auf den Beispiel-Ausdruck `B <= 10 && C == 2 || A >= 3 && C != 2` ergibt sich der Binärbaum in Abbildung 6.17.

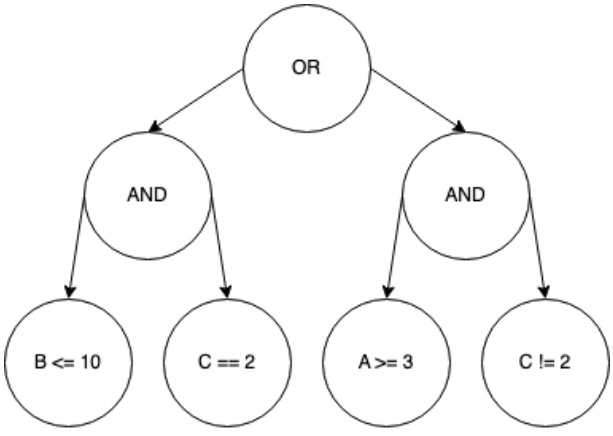


Abbildung 6.17: Binärbaum zum Ausdruck B <= 10 && C == 2 || A >= 3 && C != 2

Befüllen der Parameterliste

Damit der Dialog in Abbildung 6.15 alle möglichen Parameter in den jeweiligen Dropdown-Listen anzeigen kann, wird ein spezieller API-Request benötigt. Dieser soll alle aktuell möglichen Parameter in einer Liste zurückgeben. Anstatt eine konstante Liste an Parametern zurückzugeben, eignet es sich der dynamische Aufbau dieser Liste eher, wenn man das System modular halten will. Somit wäre die Parameterliste immer aktuell und müsste nicht per Hand abgeändert werden. Das Code-Beispiel 6.10 zeigt die Umsetzung der REST API, welche die Parameter in einer Liste zurückgibt.

```
1 public async Task<ActionResult<IEnumerable<PropertyData>>> GetParameters()
2 {
3     var l = new List<PropertyData>();
4     l.AddRange(GetAllProperties(typeof(FrontendModelsLiveData), ""));
5     l = l.OrderBy(x => x.Name).DistinctBy(x => x.Name).ToList();
6     return l;
7 }
```

Listing 6.9: REST API "GetParameters"welche alle möglichen Parameter zurückgibt

In MB PALAMAX® Pico können alle errechneten Werte aus einem Klassen-Objekt ausgelesen werden. Extrahiert man alle Properties aus dieser Klasse, erhält man alle möglichen Parameter. Da sich diese jedoch in unterschiedlicher Objekttiefe befinden, muss zusätzlich der Pfad der Property zurückgegeben werden. Um Bedingungen nur mit den gleichen Typen zu erlauben, sollte neben dem Namen und dem Pfad des Parameters auch der Typ übergeben werden. Die genaue Umsetzung dieser Logik ist in dem Code-Beispiel B.1 aus dem Anhang zu entnehmen.

Job Status

Da der Job Status des Alarmsystems über einen Websocket bereitgestellt wird (siehe Kapitel 5.3.1) muss im Monitoring-System auf diesen zugegriffen werden, um die Informationen zu erhalten. Die erhaltenen Informationen müssen danach noch ansprechend formatiert werden, damit der Nutzer diese überwachen kann. Dazu eignet sich eine eigene Übersichtsseite, welche die Elemente in einer Liste anzeigt. Wenn der Nutzer die Seite nicht mehr betrachten will, wird die Websocket Verbindung wieder geschlossen, um den Netzwerkverkehr auf ein Minimum zu begrenzen.

AlarmId	Name	State	Type	Start timestamp	Last edit timestamp
9	running	Running	SingleCall	21.10.2022 14:00	21.10.2022 14:01
9	running	Queued	SingleCall	21.10.2022 14:02	21.10.2022 14:02

Abbildung 6.18: Alarm Job Übersicht

6.2.3 Datenbank

Mit EF-Core wird die bestehende Datenbank um die neuen Tabellen 6.2 erweitert. Mit EF-Core ist es möglich, Zwischentabellen automatisch zu generieren. Man muss daher nur zwei Tabellen mit ihren Attributen angeben. Die Erstellung der Zwischentabelle geschieht durch den Einsatz des Schlüsselworts „HasMany“, die ist in dem folgenden Code Beispiel ersichtlich.

```
1  private void CreateAlertModel(ModelBuilder modelBuilder)
2  {
3      modelBuilder.Entity<Alert>(entity =>
4      {
5          entity.Property(x => x.Id).ValueGeneratedOnAdd();
6          entity.HasKey(x => x.Id);
7          entity.Property(x => x.Name);
8          entity.Property(x => x.MetricCondition);
9          entity.Property(x => x.ActionType);
10         entity.HasMany(x => x.Users);
11     });
12 }
13
14 private void CreateUserModel(ModelBuilder modelBuilder)
15 {
16     modelBuilder.Entity<User>(entity =>
17     {
18         entity.Property(x => x.Id).ValueGeneratedOnAdd();
19         entity.HasKey(x => x.Id);
20         entity.Property(x => x.Name);
21         entity.Property(x => x.Mail);
22         entity.Property(x => x.Phone);
23     });
24 }
```

Listing 6.10: Datenbank-Modellierung mit EF-Core

6.2.4 Software Architektur

Die Software basiert auf ASP.NET Core und wurde in C# entwickelt. Als Datenbank wird PostgreSQL benutzt. Es wurde als monolithische Anwendung entwickelt. Eine monolithische Anwendung besitzt eine einzelne Code-Basis und wird in einem einzigen Programm ausgeführt [43]. Dabei besteht die Architektur einer monolithischen Anwendung aus drei Layern [44]:

- User Interface Layer
- Business Logic Layer
- Persistence- / Data-Access Layer

Auch der Code für die Weboberfläche befindet sich im selben Projekt. Das hat den Vorteil, dass beim Bereitstellen der Applikation nur Webserver erstellt werden muss.

6.3 Zusammenfassung

Das Alarmsystem wurde in diesem Kapitel in das Monitoring-System MB PALAMAX® Pico implementiert. Um die neuen Funktionalitäten nutzen zu können, wurde die bestehende Weboberfläche um neue Bedienelemente erweitert. Um das Problem generischer Serialisierung und Deserialisierung von Metriken zu lösen, wurde ein eigener Algorithmus, basierend auf „Expression Trees“, entwickelt. Die Umsetzung dieser Logik wurde anhand von Code-Ausschnitten und Beispielen erläutert. Nach Fertigstellung der Implementierung kann nun im nachfolgenden Kapitel eine Evaluation durchgeführt werden.

Kapitel 7

Evaluation

In diesem Abschnitt wird überprüft, ob das entwickelte Alarmsystem die erforderlichen Anforderungen erfüllt und ob diese erfolgreich in ein beliebiges Monitoring-System implementiert werden können. Abschließend wird ein Fazit gezogen.

7.1 Prüfen der Anforderungen

Aus dem Anforderungskontext im Kapitel 1.3 haben sich vier große Themenbereiche kristallisiert. Das Alarmsystem soll die Herausforderung von reaktiver Überwachung lösen, welches in dieser Ausarbeitung durch Einsatz von Alarmklassifizierungen umgesetzt wurde. Es soll in Hochsicherheitsbereichen genutzt werden können, um muss daher sichere Kommunikationswege zur Alarmierung nutzen. Zudem soll die Reaktionszeit auf gravierende Ereignisse erhöht werden, um die negativen Auswirkungen zu vermindern. Außerdem sollte die Anwendung so entwickelt werden, damit es von mehreren unterschiedlichen Monitoring-Systemen genutzt werden kann. Diese Themenbereiche werden nachfolgend im Detail behandelt.

Klassifizierung von Alarmierungen

Um Alarmierungen in unterschiedlichen Kategorien unterscheiden zu können, ist eine Klassifizierung erforderlich. Um eine minimale Klassifizierung zu ermöglichen, ist die Nutzung von mindestens zwei unterschiedlichen Klassen nötig. Nach der Evaluation von unterschiedlichen Kommunikationsmethoden in Kapitel 4.1,konnten alle Methoden einer entsprechenden Klassifizierung zugeordnet werden. Da die Zeit für die Ausarbeitung dieses Projekts begrenzt ist, musste eine Wahl über die Methoden getroffen werden, welche schlussendlich in das Alarmsystem implementiert werden sollen.

Hierfür wurde jeweils eine Methode für weniger schwerwiegende und sehr schwerwiegende Ereignisse gefunden, welche die essenzielle Grundlage einer effektiven Alarmierung darstellt. Die gewählte Methode für sehr schwerwiegende Ereignisse ist

die Telefonie, und für weniger schwerwiegende würde die Alarmierung über E-Mail gewählt.

Angepasst auf Hochsicherheitsbereiche

Damit Daten sicher im internen Netzwerk verbleiben, ist die Nutzung von externen Diensten nicht gestattet. Die Umsetzung des Alarmsystems sollte Methoden wählen welche die nötigen Funktionalitäten ohne externen Zugang bewältigen kann. Hierfür wurde in der Evaluation im Kapitel 4.1.2 eine Überprüfung durchgeführt, welche die On-premise Nutzung der ausgewählten Methoden analysiert.

Die Ergebnisse der Überprüfung wurden in der Umsetzung des Alarmsystems besitzt berücksichtigt. Damit könnte ein Alarmsystem entwickelt werden, welche keine Anbindung nach außen benötigt und nur lokale Ressourcen benutzt, um eine effektive Alarmierung zu gewährleisten. Für die Alarmierung über die Telefonie können eigene Telefonanlagen und für das Versenden von E-Mails kann ein interner *SMTP*-Server genutzt werden. Mit beiden Varianten können die Informationen über ein sicheres Übertragungsmedium versendet werden.

Erhöhung der Reaktionszeit

Das Ziel reaktiver Überwachung ist die Verkürzung der Reaktionszeit zu schwerwiegenden Ereignissen. Um diesem Ziel zu entsprechen, wurde nach der Ausführung einer Nutzwertanalyse im Kapitel 4.1.2 der Kommunikationsweg über die Telefonie gewählt. Die Vermeidung des Cry Wolf Effekts durch den Nutzen von klassifizierten Alarmierungen trägt dem Ziel der reaktiven Überwachung zusätzlich bei. Eine Klassifizierung bringt den Vorteil, dass der Nutzer bereits bei der Alarmierung weiß, welcher Schweregrad das Ereignis besitzt.

Die Informationen zu Ereignissen, welche keine hohe Priorität zur schnellen Reaktion aufweisen, werden deswegen über E-Mail versendet. Der Nutzer kann sich deswegen darauf verlassen, dass dieser über die Telefonie nur besonders wichtige Alarmierungen zu Ereignisse erhält. Damit kann der primäre Fokus auf die besonders schwerwiegenden Ereignisse gelenkt werden und erhöht somit zusätzlich die Reaktionszeit.

Nutzbar von beliebigen Monitoring-Systemen

Das Alarmsystem ist ein eigenständiges System, welches von allen *REST* fähigen Systemen gesteuert werden kann. Somit kann es von jeglichen Monitoring-Systemen zur Alarmierung genutzt werden, solange sich diese im selben internen Netzwerk befinden.

Da die konkrete Auswertung von Bedingungen in den jeweiligen Monitoring-Systemen selbst ausgeführt wird, ist das Alarmsystem frei von Abhängigkeiten und kann in jeglichem Alarmierungskontext benutzt werden. Im Falle von der Firma Mühlbauer GmbH & Co. KG kann das Alarmsystem sowohl in der industriellen Produktionsüberwachung, als auch in der Systemüberwachung einer IT-Abteilung eingesetzt werden.

7.2 Nutzerbericht

Das System wird bereits für interne Tests an Mühlbauer Maschinen eingesetzt. Dies ist besonders hilfreich, um frühzeitig fehlerhafte Werte zu erkennen. Auch für die Entwickler von MB PALAMAX® und MB PALAMAX® Pico ist das Alarmsystem eine Bereicherung. Der Zeitraum für Tests zur Datenverarbeitung beträgt meist mehrere Stunden, um genügend Daten während einer Produktion zu sammeln. Durch den Zeitaufwand ist es nicht möglich das System dauerhaft zu überwachen. Wenn während eines Tests ein Maschinenfehler auftaucht, kann dieser erst später wahrgenommen werden, wenn der Test nicht über die gesamte Dauer überwacht wird. Die Testdaten werden ungültig und der Test muss erneut gestartet werden. Der Einsatz des Alarmsystems verkürzt die Dauer dieser Tests. Die Ereignisse werden schneller wahrgenommen und die Tests können schneller abgeschlossen werden.

7.3 Fazit

Das Ergebnis der Anforderungsprüfung zeigt, dass alle gestellten Anforderungen erfolgreich umgesetzt wurden. Somit kann das Alarmsystem zur reaktiven Alarmierung in Hochsicherheitsbereichen eingesetzt werden. Bereits durch den Einsatz in internen Tests konnte der vorteilhafte Nutzen des Alarmsystems bestätigt werden. Die Umsetzung entspricht den Erwartungen des Auftragstellers gänzlich.

Kapitel 8

Epilog

Der letzte Teil dieser Arbeit gibt einen abschließenden Ausblick zum Projekt und über weitere Schritte die zukünftig angegangen werden können. Außerdem wird hier eine umfassende Zusammenfassung über die Ergebnisse der Arbeit beschrieben.

8.1 Ausblick

Nach erfolgreicher Implementierung des Alarmsystems in MB PALAMAX® Pico ist nun auch die Implementierung in weitere Monitoring-Systeme in der Firma Mühlbauer GmbH & Co. KG geplant. Durch die modulare Entwicklung des Alarmsystems kann es in Zukunft durch beliebige weitere Kommunikationsschnittstellen erweitert werden. Viele Kommunikationswege die im Forschungsergebnis im Kapitel 2.2.4 vorgestellt wurden, bieten Potenzial an welches die Reaktionszeit auf Alarmierungen weiter steigern kann.

Beispielsweise ist die Entwicklung einer eigenständigen mobilen Applikation durch die Vielzahl an Vorteilen vielversprechend. Wie bereits im Kapitel 2.2.4 beschrieben, wäre die App dazu in der Lage visuell und auditive Signale an den Nutzer zu senden. Alarmierungen könnten vom Nutzer bestätigt oder weitergeleitet werden. Auch die Übersicht zu den aktiven und vergangenen Alarmierungen könnten bequem vom Nutzer eingesehen werden während er mobil unterwegs ist und keinen Zugang auf ein Monitoring-System hat. Generell können die Informationen detailreicher versendet und angezeigt werden.

Um die Reaktionszeit über die Alarmierungsmethode der Telefonie zu erhöhen, ist die Nutzung von *DTMF* Tönen empfohlen. Damit wäre es möglich, unterschiedliche Antwortmöglichkeiten zu empfangen und im Alarmsystem zu verarbeiten. Beispielsweise könnte ein Empfänger den Erhalt der Alarmierung bestätigen, jedoch angeben, dass dieser aktuell verhindert ist. Die Nutzung von *DTMF* Tönen hängt von der Wahl des *SIP* Clients und den dazugehörigen Bibliotheken ab. Diese müssen mit der bestehenden Telefonanlage des Nutzers kompatibel sein. Daher ist es notwendig, die Wahl der Bibliotheken auf die Kundenspezifikationen abzustimmen.

Damit der Nutzer mehr Kontrolle über die laufenden Jobs des Alarmsystems hat, ist die Implementierung neuer Funktionalitäten zum Pausieren, Priorisieren und Beenden von Jobs denkbar. Dies könnte durch die Erweiterung der Job Controllers ermöglicht werden. Diese Funktionalitäten wären auch in einer mobilen Applikation hilfreich, da die Bediener nach einer erfolgreichen Kontrolle der Maschine den Job-Status ändern könnten. Damit könnten Eskalationslogiken schneller beendet werden, und die Nutzer in den jeweiligen Nutzergruppen wären vor unnötigen Alarmierungen geschützt.

Um die Effizienz des Alarmsystems zu erhöhen, können zusätzliche Methoden zur Auswertung einer Alarmbedingung implementiert werden. Dies kann beispielsweise durch die Konfiguration einer Verzögerungszeit erreicht werden [45, S. 48]. Ereignisse, welche eine Alarmierung auslösen würden, jedoch bereits in kurzer Zeit behoben werden, benötigen keine zusätzliche Alarmierung. Maschinen, die ein Verbrauchsmaterial zum Produzieren benötigen, könnten beispielsweise einen Fehler auslösen, wenn kein Verbrauchsmaterial zur Verfügung steht. Dieser Fall tritt beispielsweise kurz nach einem Produktionsstart ein, weswegen sich der Maschinenbediener noch in unmittelbarer Nähe zur Maschine befindet. Der Maschinenbediener könnte den Fehler sofort beheben und eine Alarmierung durch die Verzögerungszeit verhindern.

Die Kunden von Mühlbauer GmbH & Co. KG haben zudem den Wunsch geäußert, eine automatische Benachrichtigung über bestimmte Systemzustände zu verschiedenen Tageszeiten zu ermöglichen. Damit wäre es möglich, eine Historie über vergangene Ereignisse einsehen zu können, ohne dafür die Weboberfläche des Monitoring-Systems öffnen zu müssen. Durch das Quartz Package, welches aktuell im Alarmsystem implementiert ist, kann diese Funktionalität umgesetzt werden. Durch die Konfiguration einer Aktion zu einer bestimmten Tageszeit können somit die erforderlichen Daten kumuliert gesendet werden.

8.2 Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde erfolgreich ein eigenständiges Alarmsystem konzipiert und umgesetzt, welches dazu in der Lage ist Alarmierungen für mehrere unterschiedliche Monitoring-Systeme zu verarbeiten. Hierbei wurde auch eine Eskalationslogik implementiert, welche nicht empfangene oder abgewiesene Alarmierungen erneut einreihen kann und Alarmierungen für mehrere Nutzergruppen ermöglicht.

Zudem wurde in dieser Arbeit die Implementierung des Alarmsystems in ein bestehendes Monitoring-System beispielhaft erläutert. Dabei wurden neben der Anbindung an die *REST*-Schnittstellen und den Websocket auch neue Ansichtsseiten in der Weboberfläche entwickelt. Diese ermöglichen dem Nutzer die vollständige Nutzung aller zur Verfügung stehenden Funktionalitäten des Alarmsystems.

Die Ergebnisse der Forschung haben gezeigt, welche Alarmierungsmethoden möglich sind, wie diese effizient genutzt werden können und welche sich für entsprechende Schweregrade eignen. Nach Evaluierung der Ergebnisse wurden für das Projekt zwei Kommunikationswege gewählt welche für eine reaktive und effiziente Alarmierung und für die gegebenen Anforderungen geeignet sind. Der gewählte Kommunikations-

weg mit der höchsten Reaktionspriorität ist die Telefonie über *SIP*. Der *SIP* Client ist so implementiert worden, dass dieser mit beliebigen anderen Bibliotheken ausgetauscht werden kann, um den Kundenwünschen im Bereich der Telefonie zu entsprechen. Alarmierungen mit geringer Reaktionspriorität werden über E-Mail versendet.

Sowohl das Alarmsystem als auch die Implementierung in das bestehende Monitoring-System MB PALAMAX® Pico konnten innerhalb der Bachelorarbeit fertiggestellt werden und wurden bereits auf ersten Produktivsystemen getestet. Die Ergebnisse sind zufriedenstellend und bereichern sowohl die Kunden als auch die Mitarbeiter der Firma Mühlbauer GmbH & Co. KG.

Literaturverzeichnis

[1] Kerner, A., Stück, R., Weiß, F. P. (2011). Der Unfall in Tschernobyl 1986. Gesellschaft für Anlagen-und Reaktorsicherheit (GRS) mbH, Garching bei München, Köln, Sonderdruck in atw, Ausgabe, 2, 2011.

[2] U. Rosenthal, A. Boin, und L. K. Comfort, MANAGING CRISES: Threats, Dilemmas, Opportunities. Charles C Thomas Publisher, 2001.

[3] I. Izadi, S. L. Shah, D. S. Shook, und T. Chen, „An Introduction to Alarm Analysis and Design“, IFAC Proceedings Volumes, Bd. 42, Nr. 8, S. 645–650, Jän. 2009, doi: 10.3182/20090630-4-ES-2003.00107.

[4] S. Ligus, Effective Monitoring and Alerting: For Web Operations. O'Reilly Media, Inc., 2012.

[5] Aesop: Die 46. Fabel, vom Hirtenbuben und dem Wolf. In: Projekt Gutenberg-DE. Nach Heinrich Steinhöwels »Erneuertem Esopus«, bearbeitet von Victor Zobel, 1919. Zugegriffen 18. November 2022. [Online]. Verfügbar unter: https://www.projekt-gutenberg.org/aesop/fabeln1/chap046.html

[6] The explosion and fires at the Texaco Refinery, Milford Haven, 24 July 1994: A report of the investigation by the Health and Safety Executive into the explosion and fires on the Pembroke Cracking Company Plant at the Texaco Refinery, Milford Haven on 24 July 1994. HSE Books 1997, ISBN 0 7176 1413 1

[7] "Zenduty". Zugegriffen 8. November 2022. [Online]. Verfügbar unter: https://www.zenduty.com/product/

[8] "Micromedia International". Zugegriffen 8. November 2022. [Online]. Verfügbar unter: https://alert.micromedia-int.com/de

[9] Fisher, C. (2018) Cloud versus On-Premise Computing. American Journal of Industrial and Business Management, 8, 1991-2006. doi: 10.4236/ajibm.2018.89133.

[10] „User Notifications“, Apple Developer Documentation. Zugegriffen 11. Dezember 2022. [Online]. https://developer.apple.com/documentation/usernotifications

[11] „Firebase Cloud Messaging | Send notifications across platforms at no-cost“, Firebase. Zugegriffen 11. Dezember 2022. [Online]. https://firebase.google.com/products/cloud-messaging

[12] Managen mit Methode: Instrumente für individuelle Lösungen. Springer-Verlag, 2013.

[13] A. B. Johnston, SIP: Understanding the Session Initiation Protocol, Fourth Edition. Artech House, 2015.

[14] „drachtio - the open source SIP application server framework“. Zugegriffen 25. November 2022 [Online]. Verfügbar unter: https://drachtio.org/

[15] „JsSIP - the Javascript SIP library“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://jssip.net/

[16] „PJSIP - Open Source SIP, Media, and NAT Traversal Library“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://www.pjsip.org/

[17] „KaplanSoft - SipCLI“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://www.kaplansoft.com/sipcli/

[18] „A SIP SDK for software developers“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://voip-sip-sdk.com/

[19] „openSIPS“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://opensips.org/

[20] „The Next-generation SIP Server: Routr“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://routr.io/

[21] „OverSIP - the SIP framework you dreamed about“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://oversip.versatica.com/

[22] „drachtio Server“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://drachtio.org/docs/drachtio-server

[23] A. Lombardi, WebSocket: Lightweight Client-Server Communications. O'Reilly Media, Inc., 2015.

[24] V. Sarcar, Simple and Efficient Programming with C: Skills to Build Applications with Visual Studio and .NET. Berkeley, CA: Apress, 2021. doi: 10.1007/978-1-4842-7322-7.

[25] "Awesome .Net Core". Zugegriffen 9. November 2022. [Online]. Verfügbar unter: https://github.com/thangchung/awesome-dotnet-core

[26] L. Lowrey, „FluentEmail - All in one email sender for .NET and .NET Core“. Zugegriffen: 25. November 2022. [Online]. Verfügbar unter: https://github.com/lukencode/FluentEmail

[27] „MailKit“. 25. November 2022. Zugegriffen: 25. November 2022. [Online]. Verfügbar unter: https://github.com/jstedfast/MailKit

[28] „MimeKit“. 25. November 2022. Zugegriffen: 25. November 2022. [Online]. Verfügbar unter: https://github.com/jstedfast/MimeKit

[29] „FluentScheduler“. FluentScheduler, 25. November 2022. Zugegriffen: 25. November 2022. [Online]. Verfügbar unter: https://github.com/fluentscheduler/FluentScheduler

[30] „Quartz Enterprise Scheduler .NET“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://github.com/quartznet/quartznet.

[31] „Hangfire – Background jobs and workers for .NET and .NET Core“. Zugegriffen 25. November 2022. [Online]. Verfügbar unter: https://www.hangfire.io.

[32] L. Sperry, „leosperry/Chroniton“. 9. November 2022. Zugegriffen: 25. November 2022. [Online]. Verfügbar unter: https://github.com/leosperry/Chroniton

[33] “Entity Framework Core”. Zugegriffen 9. November 2022. [Online]. Verfügbar unter: https://learn.microsoft.com/de-de/ef/core/

[34] “Architectural principles” Zugegriffen 10.November. [Online]. Verfügbar unter: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles

[35] B. Preim und R. Dachselt, Interaktive Systeme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-05402-0.

[36] “Material UI”. Zugegriffen 19. November 2022. [Online]. Verfügbar unter: https://mui.com/

[37] „DataGrid API - MUI X“. Zugegriffen 19. November 2022. [Online]. https://mui.com/x/api/data-grid/data-grid/

[38] „React Dialog component - Material UI“. Zugegriffen 19. November 2022. [Online]. https://mui.com/material-ui/react-dialog/

[39] „Expression Trees (C)“. Zugegriffen 28. November 2022. [Online]. Verfügbar unter: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/expression-trees/

[40] D. Abts, „Operatoren“, in Grundkurs JAVA, Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 19–28. doi: 10.1007/978-3-658-30494-2_3.

[41] M. Aupperle, „Operationen mit fundamentalen Typen“, in Die Kunst der Programmierung mit C++, Wiesbaden: Vieweg+Teubner Verlag, 2002, S. 75–97. doi: 10.1007/978-3-663-07766-4_5.

[42] “Boolean logical operators”. Zugegriffen 23. November 2022. [Online]. Verfügbar unter: https://learn.microsoft.com/en-US/dotnet/csharp/language-reference/operators/boolean-logical-operators

[43] „Common web application architectures“. Zugegriffen 19. November 2022. [Online]. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[44] S. Whitesell, R. Richardson, und M. D. Groves, Pro Microservices in .NET 6: With Examples Using ASP.NET Core 6, MassTransit, and Kubernetes. Berkeley, CA: Apress, 2022. doi: 10.1007/978-1-4842-7833-8.

[45] J. Welch, „An Evidence-Based Approach to Reduce Nuisance Alarms and Alarm Fatigue“, Biomedical Instrumentation Technology, Bd. 45, Nr. s1, S. 46–52, März 2011, doi: 10.2345/0899-8205-45.s1.46.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Bilder

A.1 E-Mail Template

Mithilfe des E-Mail-Templates in MB PALAMAX® Pico kann beim Versenden die nachfolgende E-Mail erzeugt werden.

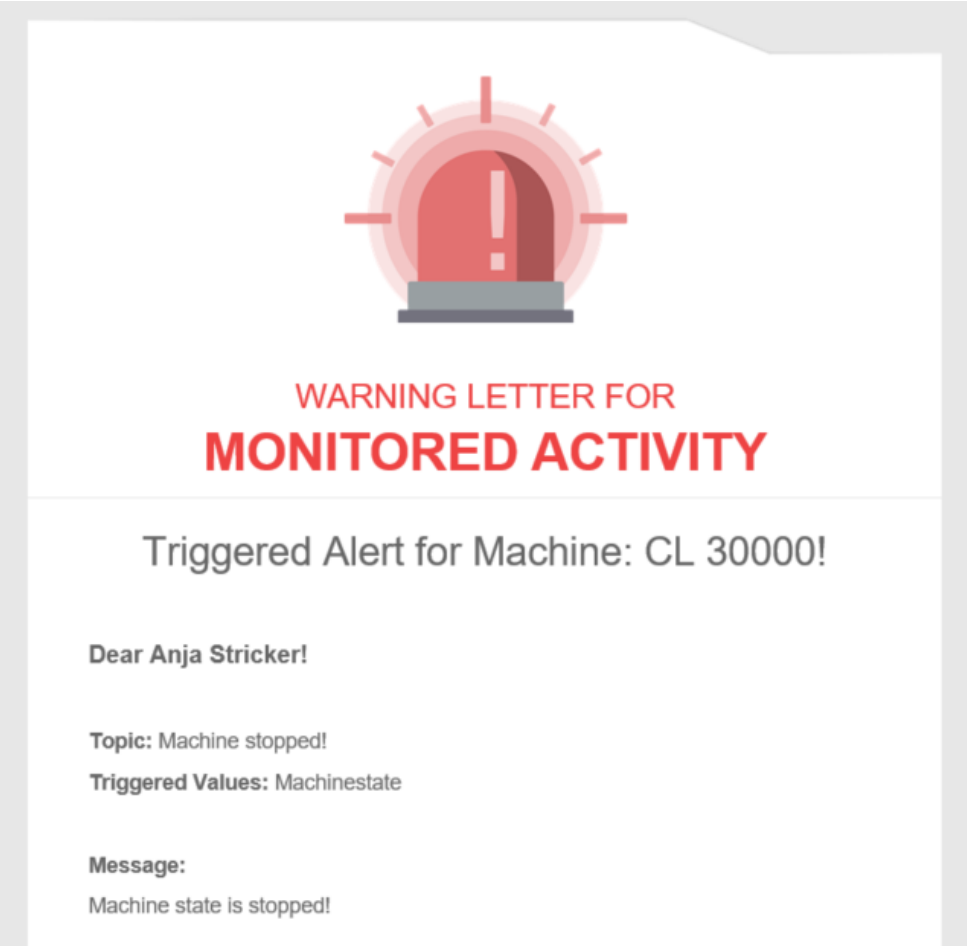


Abbildung A.1: Ausschnitt des E-Mail-Templates für MB PALAMAX® Pico

A.2 Live-Daten Seite

Auf dem folgenden Bild ist ein Ausschnitt der Live-Daten-Seite zu sehen. Diese Ansicht wurde für Vorführungszwecke mit Testdaten befüllt.

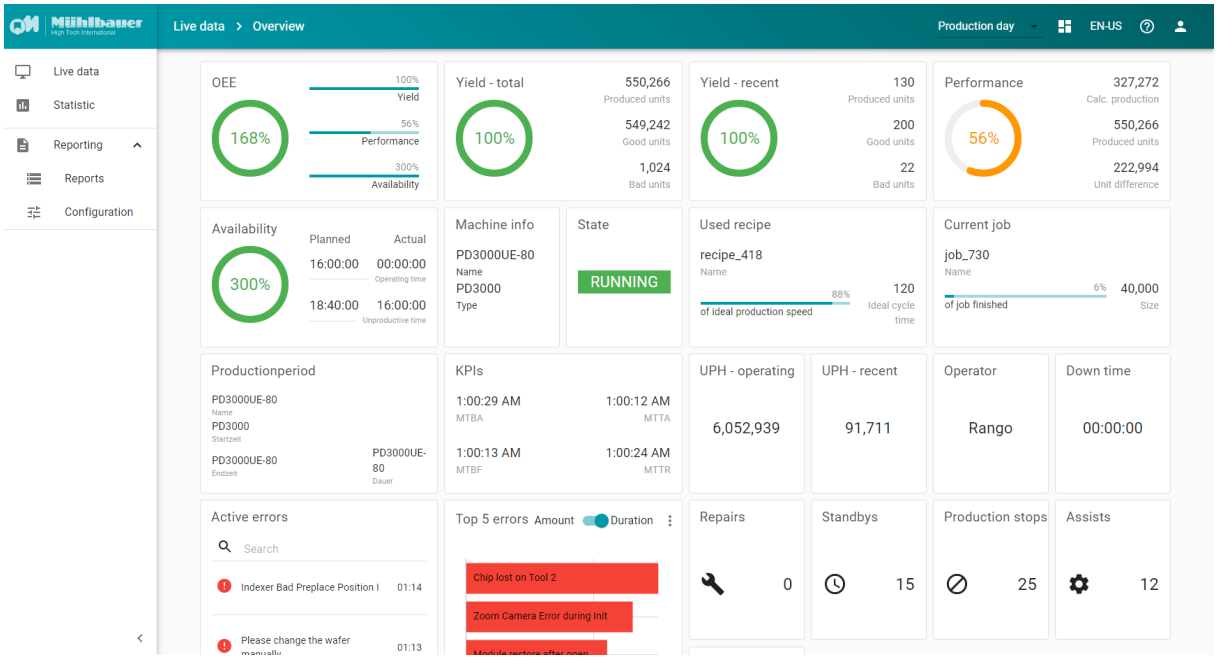


Abbildung A.2: MB PALAMAX® Pico Live-Daten Seite

A.3 Statistik Seite

Auf dem folgenden Bild ist ein Ausschnitt der Statistik-Seite zu sehen. Diese Ansicht wurde für Vorführungszwecke mit Testdaten befüllt.

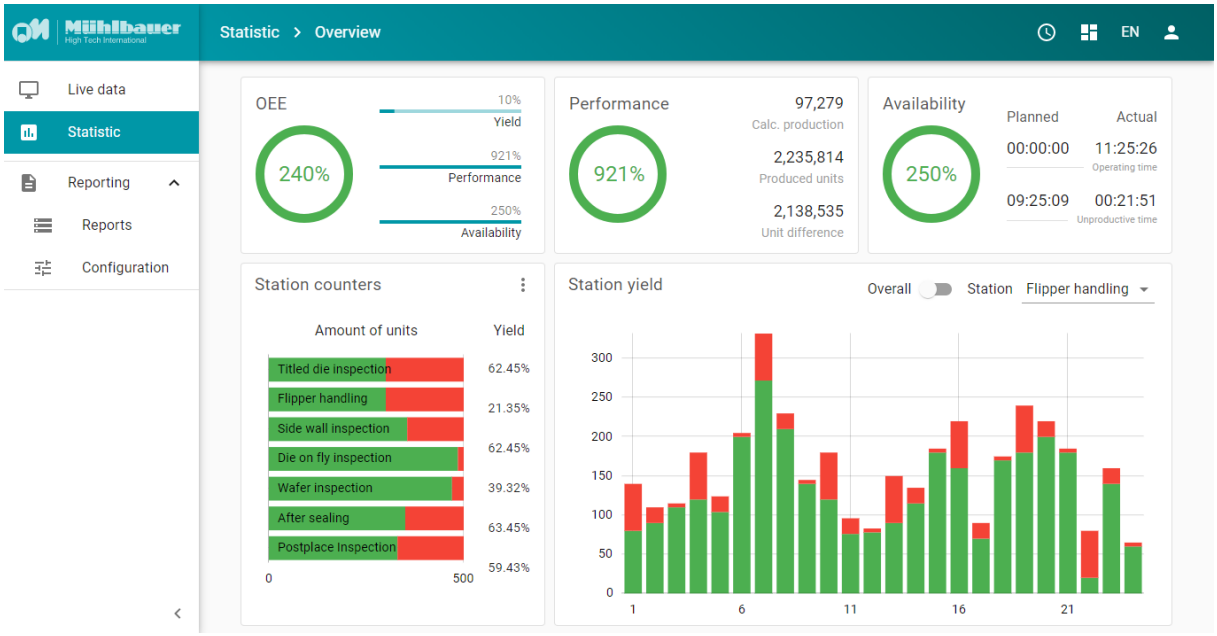


Abbildung A.3: MB PALAMAX® Pico Statistik Seite

Anhang B

Codeauszüge

B.1 AlertsController

```
List<PropertyData> GetAllProperties(Type type, string path)
   {
       var newPath = path == "" ? path : path + ".";
       var t = new List<PropertyData>();
       var props = type.GetProperties();
       foreach (var item in props)
       {
           var typeItem = item.PropertyType;
           if (typeItem.GetTypeInfo().IsClass &&
               !(typeof(string) == typeItem.GetTypeInfo()) &&
               !(typeof(int[]) == typeItem.GetTypeInfo()))
               t.AddRange(GetAllProperties(typeItem, newPath + item.Name));
           else
           {
               var convertedType = MetricMonitor.Models.Type.Boolean;
               if (typeItem == typeof(int[])) continue;
               if (typeItem == typeof(string))
                   convertedType = MetricMonitor.Models.Type.String;
               else if (typeItem == typeof(double) || typeItem == typeof(long) ||
                   typeItem == typeof(int) || typeItem.IsEnum)
                   convertedType = MetricMonitor.Models.Type.Number;

               t.Add(new() {
                   Name = item.Name,
                   Path = newPath + item.Name,
                   Type = convertedType
               });
           }
       }
       return t;
   }
```

Listing B.1: Extrahieren aller Properties aus dem übergebenen Typen

B.2 JobHandler

```
public Task JobToBeExecuted(IJobExecutionContext context,
CancellationToken cancellationToken = default)
{
    lock (jobDatabaseLock)
    {
        _jobsService.UpdateJobState(context.JobDetail.JobDataMap
        .GetLongValue("jobId"), Models.JobState.Running);
    }

    return Task.CompletedTask;
}
```

Listing B.2: Funktion, die vor der Ausführung eines Jobs ausgeführt wird

```
public void ReScheduleJob(IJobExecutionContext context) {
    var jobname = context.JobDetail.JobDataMap.Get("alarmName").ToString()
        + DateTime.Now.Millisecond;
    IJobDetail job = JobBuilder.Create<CallingJob>()
    .WithIdentity(jobname).UsingJobData("jobId", jobId)
    .UsingJobData("number", context.JobDetail.JobDataMap.Get("number").ToString())
    .UsingJobData("message", context.JobDetail.JobDataMap.Get("message").ToString())
    .Build();
    AddJob(job);
    Console.WriteLine("Ringing timeout for job: {0} in group {1}. Details : {2}",
       context.JobDetail.Key.Name, context.JobDetail.Key.Group, jobException.Message);
}
```

Listing B.3: Hilfsmethode, um einen Job erneut in die Warteschlange einzureihen

```
 public void GetNextJob(IJobExecutionContext context){
    var nextJob = QueuedJobs.FirstOrDefault();
    if (nextJob != null)
    {
        QueuedJobs.Remove(nextJob); // removes job from queue and schedules it
        var jobname = context.JobDetail.JobDataMap.Get("alarmName").ToString()
            + DateTime.Now.Millisecond;
        ITrigger trigger = TriggerBuilder.Create()
            .WithIdentity(jobname).StartNow().Build();
        context.Scheduler.ScheduleJob(nextJob, trigger);
    }
    else
    {
        CurrentJob = null; // clearing current calling job
    }
}
```

Listing B.4: Hilfsmethode, um den nächsten Job aus der Warteschlange auszuführen

```
1  public Task JobWasExecuted(IJobExecutionContext context,
2  JobExecutionException? jobException, CancellationToken cancellationToken = default)
3  {
4      var jobId = context.JobDetail.JobDataMap.GetLongValue("jobId");
5
6      // only run second job if first job was executed successfully
7      if (jobException == null)
8      {
9          lock (jobDatabaseLock)
10         {
11             _jobsService.UpdateJobState(jobId, Models.JobState.Finished);
12         }
13         // this logic deletes every call with the same escallationcallid
14         var groupedAlarm = QueuedJobs.FindAll(x =>
15             x.JobDataMap.GetLongValue("jobId") == jobId);
16         foreach (var item in groupedAlarm)
17             QueuedJobs.Remove(item);
18     }
19     else
20     {
21         if (jobException.Message == "Ringing timeout")
22         {
23             lock (jobDatabaseLock)
24             {
25                 _jobsService.UpdateJobState(jobId, Models.JobState.Queued);
26             }
27
28             ReScheduleJob(context.JobDetail);
29         }
30         else
31         {
32             lock (jobDatabaseLock)
33             {
34                 _jobsService.UpdateJobState(jobId, Models.JobState.Error);
35             }
36         }
37     }
38
39     lock (jobqueueLock)
40     {
41         GetNextJob(context);
42     }
43     return Task.CompletedTask;
44 }
```

Listing B.5: Funktion, die nach der Ausführung eines Jobs ausgeführt wird