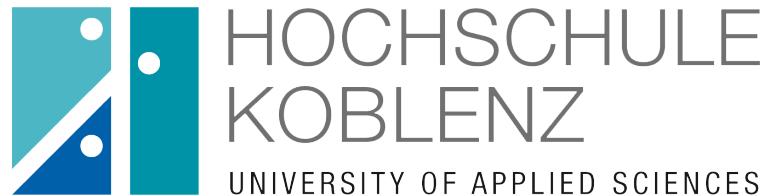


Hochschule Koblenz
Fachbereich Ingenieurwesen
Elektro- und Informationstechnik



Bachelorarbeit Bericht:
Entwicklung einer
Desktopanwendung zur
Datenerfassung und Analyse von
SPS-Variablen

Ayoub Saidi

543574

Betreuer: Prof. Dr. Markus Kampmann

Externer Betreuer Matthias Gros / Marc Reuter

Abgabedatum: 17.03.2025

Selbstständigkeitserklärung

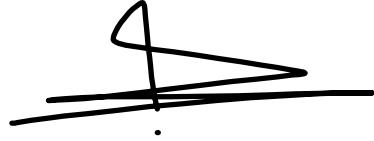
Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Diese Arbeit wurde in gleicher oder ähnlicher Fassung noch keiner anderen Prüfungsbehörde vorgelegt. Ich versichere, dass die elektronische Version der Arbeit mit der gedruckten Version der Arbeit inhaltlich übereinstimmt. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Mir ist bewusst, dass ein Nichteinhalten der Regeln guter wissenschaftlicher Praxis das Nichtbestehen der Prüfungsleistung zur Folge hat.

13.03.2025

Koblenz, den

Ayoub Saidi

A handwritten signature consisting of several black ink strokes forming a stylized, abstract shape.

Inhaltsverzeichnis

Selbstständigkeitserklärung	I
Inhaltsverzeichnis	II
Abbildungsverzeichnis	IV
Abkürzungsverzeichnis	V
1 Einleitung	1
1.1 Hintergrund der Arbeit	1
1.2 Problemstellung	1
1.3 Lösungsansatz	2
2 Grundlagen	3
2.1 Überblick über SPS-Systeme	3
2.2 OPC-UA als Kommunikationsstandard	6
2.3 Sicherheit und Authentifizierung in OPC-UA	8
3 Anforderungsanalyse und Systemarchitektur	9
3.1 Anforderungsanalyse	9
3.1.1 Funktionale Anforderungen	9
3.1.2 Nicht-funktionale Anforderungen	10
3.2 Systemarchitektur	12
4 Implementierung	14
4.1 Auswahl der verwendeten Technologien und Tools	14
4.2 OPC-UA-Verbindung	16
4.2.1 Architektur der OPC-UA-Verbindung	16
4.2.2 Verbindungsaufbau und Speicherung der Verbindung	17
4.2.3 Verbindungsaufbau mit Authentifizierung	17
4.2.4 Verwaltung und Speicherung der OPC-UA-Verbindungen	19
4.3 Datenbankverbindung	21
4.3.1 Auswahl der Datenbanktechnologien	21
4.3.2 Architektur der Datenbankverbindung	22
4.3.3 Verbindungsaufbau und Speicherung der Verbindung	23
4.4 Konfiguration der Excel-Trigger	25
4.4.1 Parameter der Excel-Trigger	25
4.4.2 Architektur und Implementierung der Excel-Trigger	27
4.5 Konfiguration der Datenbank-trigger	30
4.5.1 Parameter der Datenbank-Trigger	30
4.5.2 Architektur der Datenbank-Trigger	31
4.5.3 Implementierung der Datenbank-Schreibprozesse	32
4.6 Datenvizualisierung	36
4.6.1 Tabellenbasierte Visualisierung	36
4.6.2 Diagrammbasierte Visualisierung	37
4.7 Datenanalyse	40
4.7.1 Erklärung der Analyseverfahren	40
4.7.2 Architektur und Implementierung der Datenanalyse	45

4.8	Fehlerbehandlung und Logging	47
4.8.1	Logging-Mechanismus und Fehlererfassung	47
4.8.2	Anzeige und Verwaltung der Logs	48
5	Test und Validierung	49
5.1	Testkonzept und Methodik	49
5.2	Durchführung der Tests	50
6	Ergebnisse	54
7	Zusammenfassung	68
	Literatur	70

Abbildungsverzeichnis

1	Systemarchitektur der Anwendung	12
2	Komponentendiagramm zur Architektur der OPC-UA-Verbindung	16
3	Sequenzdiagramm der Verbindungsaufbau	17
4	ConnectAsync Methode	18
5	Klassendiagramm der OPC-UA-Verbindungsverwaltung	19
6	Klassendiagramm der Datenbankverbindung	22
7	Sequenzdiagramm der Verbindungsherstellung der Datenbankverbindungen	23
8	Klassendiagramm der Architektur der Excel-Trigger	27
9	Sequenzdiagramm für die Verarbeitung von Excel-Triggern	29
10	Sequenzdiagramm für die Verarbeitung von Datenbank-Triggern	31
11	Einfügen von Daten in MySQL	32
12	Überprüfung und Erstellung der BitBedeutung-Spalte in MySQL	32
13	Sicherstellen der Timestamp-Spalte in MySQL	33
14	Einfügen von Daten in PostgreSQL	34
15	Überprüfung und Erstellung der BitBedeutung-Spalte in PostgreSQL	34
16	Sicherstellen der Timestamp-Spalte in PostgreSQL	34
17	Sequenzdiagramm zur Tabellenvisualisierung	36
18	Klassendiagramm zur Diagrammbasierte Datenvisualisierung	38
19	Klassendiagramm der Datenanalyse	45
20	LoadDatabaseConnections() Methode	47
21	Hauptansicht der Anwendung	54
22	Fenster zur Konfiguration der SPS-Verbindung	55
23	Übersicht der aktiven SPS-Verbindungen	56
24	Durchsuchen der SPS-Variablenstruktur und Anzeige von Details	56
25	Echtzeit-Datenvisualisierung einer SPS-Variable	57
26	Konfigurationsfenster für eine MySQL-Datenbankverbindung	58
27	Übersicht der Datenbankverbindungen mit aktiven Verbindungen	59
28	Konfiguration der automatisierten Excel-Speicherung	60
29	Konfiguration der automatisierten Datenbankspeicherung	60
30	Verwaltung gespeicherter Trigger	61
31	Tabellenbasierte Anzeige der gespeicherten Daten	61
32	Konfigurationsfenster zur Erstellung eines Diagramms	62
33	Verwaltung gespeicherter Diagramme	63
34	Visualisierung gespeicherter Daten als Diagramm	63
35	Konfigurationsfenster für die Datenanalyse	64
36	Übersicht gespeicherter Analysen	65
37	Darstellung der Analyseergebnisse	65
38	Protokollansicht für System-Logs	66
39	Hilfefenster für detaillierte Anleitungen	66
40	Einstellungsfenster der Anwendung	67

Abkürzungsverzeichnis

AES Advanced Encryption Standard. 8

API Application Programming Interface. 14

CPU Central Processing Unit. 3

DBMS Database Management System (Datenbankverwaltungssystem). 15

GUI Grafische Benutzeroberfläche. 13

HTTPS Hypertext Transfer Protocol Secure. 7

HVAC Heating, Ventilation, Air Conditioning (Heizung, Lüftung, Klimatisierung). 5

IEC 61131-3 Internationale Norm für SPS-Programmiersprachen. 4

IoT Internet of things. 7

JSON JavaScript Object Notation. 17

OPC-UA Open Platform Communications - Unified Architecture. 3

SOA Serviceorientierte Architektur. 6

SPS speicherprogrammierbare Steuerung. 1

TCP/IP Transmission Control Protocol / Internet Protocol. 7

TIA Portal Totally Integrated Automation Portal. 5

TLS Transport Layer Security. 14

WPF Windows Presentation Foundation. 14

XAML Extensible Application Markup Language. 14

XLSX Microsoft Excel Open XML Spreadsheet Format. 15

1 Einleitung

1.1 Hintergrund der Arbeit

Moderne Fertigungsanlagen basieren heute auf speicherprogrammierbare Steuerung (SPS), die als zentrale Nervensysteme der Automatisierungstechnik gelten. Sie steuern nicht nur Maschinen und Anlagen präzise, sondern erfassen auch kontinuierlich Daten – vom einfachen Sensorwert bis hin zu komplexen Prozessparametern. Diese Informationen bilden das digitale Abbild der Produktion und bergen das Potenzial, Einblicke in Durchlaufzeiten, Maschinenauslastung oder Qualitätsabweichungen zu liefern.

Parallel dazu hat sich in den letzten Jahren das Schlagwort „Digitalisierung“ in der industriellen Praxis immer stärker etabliert. Mit dem Ziel, Abläufe zu optimieren und Kosten zu senken, setzen Unternehmen zunehmend auf Datenanalyse und -auswertung. So können Produktionsprozesse auf Basis realer Betriebsdaten laufend verbessert, potentielle Störungen frühzeitig erkannt und Reparaturen effizienter geplant werden. In der Theorie klingt dies verheißungsvoll, in der betrieblichen Realität zeigt sich allerdings oft eine Kluft zwischen dem theoretischen Potenzial und der praktischen Umsetzung. Zwar stehen viele Daten innerhalb der SPS zur Verfügung, doch der Zugang zu diesen Informationen ist häufig kompliziert, weil geeignete Schnittstellen oder standardisierte Lösungen fehlen. Dadurch bleiben Chancen zur Prozessoptimierung ungenutzt, und es entsteht ein blindes Vertrauen in altbewährte Abläufe, die womöglich nicht mehr optimal sind.

1.2 Problemstellung

Trotz umfangreicher Datenerfassung durch die SPS wird das volle Potenzial dieser Informationen in vielen Unternehmen nicht ausgeschöpft. In vielen Betrieben werden zwar kontinuierlich relevante Produktionsdaten in Form von Variablenwerten in der SPS erfasst, doch eine unmittelbare Weiterverarbeitung dieser Informationen ist häufig nicht möglich. Das liegt unter anderem daran, dass sich Steuerungen unterschiedlichster Hersteller nur schwer in bestehende IT-Systeme integrieren lassen. Zudem fehlen oft standardisierte Prozesse und Werkzeuge, um die Daten automatisiert auszulesen und für weiterführende Analysen aufzubereiten.

Die Folgen sind vielschichtig: Unentdeckte Ineffizienzen in der Produktion, verzögerte Reaktionen auf Störungen und ein eingeschränktes Potenzial für vorbeugende Wartung. Besonders kritisch wird dies im Kontext globaler Wettbewerbsdruck, der schnelle Anpassungen und maximale Ressourcennutzung erfordert. Zudem binden manuelle Datenexporte und individuelle Insellösungen personelle Kapazitäten und erhöhen die Fehleranfälligkeit. Es fehlt somit eine Brücke zwischen der operativen Steuerungsebene und der strategischen Datenanalyse – eine Lücke, die nicht nur technisch, sondern auch wirtschaftlich relevant ist.

Daraus ergibt sich ein dringender Bedarf nach einer integrierten Herangehensweise, die eine zuverlässige und effiziente Übertragung der SPS-Daten in externe Systeme sicherstellt. Nur so lassen sich aussagekräftige Analysen durchführen, auf deren Basis fundierte Entscheidungen getroffen und langfristige Verbesserungen im Produktionsablauf realisiert werden können.

1.3 Lösungsansatz

Im Rahmen dieser Bachelorarbeit bei der Firma Schiele Maschinenbau GmbH wird eine Desktopanwendung entwickelt, die eine Verbindung zu einer speicherprogrammierbaren Steuerung herstellt und Variablenwerte automatisch ausliest und in unterschiedlichen Formaten speichert. Damit können die erfassten Daten sowohl für eine laufende Überwachung als auch für vertiefende Analysen genutzt werden. Ein besonderer Schwerpunkt liegt auf der Anbindung an eine Datenbank . Somit gibt es die Möglichkeit, die Werte entweder in Tabellenkalkulationsprogramme wie Excel oder in Datenbanktabellen zu speichern . So erhalten Anwender einerseits Einblick in Echtzeitinformationen, während gleichzeitig langfristige Entwicklungen dokumentiert und Optimierungspotenziale aufgedeckt werden können. Wichtig ist zudem, dass die Lösung sich nahtlos in die bestehende Infrastruktur integrieren lässt und durch eine flexible Architektur künftigen Anforderungen gerecht wird.

Im weiteren Verlauf dieses Berichts werden zunächst in Kapitel 2 die relevanten Grundlagen zu SPS-Systemen und industriellen Kommunikationsstandards erläutert, um die technischen Voraussetzungen zu verdeutlichen. In Kapitel 3 folgen eine detaillierte Anforderungsanalyse sowie die Darstellung der geplanten Systemarchitektur. In Kapitel 4 werden anschließend die wesentlichen Schritte der Implementierung erläutert. Der darauf folgende Kapitel beschreibt den Test- und Validierungsprozess. Abschließend fasst Kapitel 6 die wichtigsten Ergebnisse zusammen.

2 Grundlagen

Bevor die praktische Umsetzung der Anwendung beginnen kann, ist es essenziell, ein solides Verständnis der zugrunde liegenden Technologien zu entwickeln. In diesem Kapitel werden daher die wichtigsten theoretischen Konzepte behandelt, die für die spätere Implementierung relevant sind.

Zunächst wird die SPS betrachtet, die in industriellen Automatisierungssystemen eine zentrale Rolle einnimmt. Danach wird der Kommunikationsstandard Open Platform Communications - Unified Architecture (OPC-UA) beschrieben, welcher den Datenaustausch zwischen SPS und anderen Systemen ermöglicht. Abschließend werden die Sicherheitsmechanismen und Authentifizierungsverfahren von OPC-UA erläutert.

2.1 Überblick über SPS-Systeme

Definition einer SPS

Eine SPS ist ein digitaler Steuerungsmechanismus, der in der Industrie weit verbreitet ist und zur Automatisierung von Maschinen und Prozessen eingesetzt wird [1]. Im Wesentlichen handelt es sich dabei um einen programmierbaren Computer, der so konzipiert ist, dass er Eingabesignale von Sensoren oder anderen Messgeräten empfängt, diese verarbeitet und entsprechend Ausgabesignale an Aktoren wie Motoren, Ventile oder Lichtsysteme sendet. Der große Vorteil einer SPS liegt darin, dass sie im Vergleich zu den traditionellen, fest verdrahteten Steuerungen sehr flexibel und anpassungsfähig ist.

Dank ihrer Fähigkeit, über ein programmiertes Steuerungsprogramm zu arbeiten, können SPS-Systeme problemlos auf unterschiedliche Anforderungen und Veränderungen im Produktionsprozess reagieren, ohne dass die gesamte Hardware neu verkabelt oder umgestellt werden muss. Das ermöglicht eine wesentlich schnellere Anpassung an neue Produktionsmethoden oder Maschinenkonfigurationen. Dies ist besonders wertvoll in der modernen Industrie, wo Prozesse häufig optimiert oder verändert werden müssen, um wettbewerbsfähig zu bleiben.

Aufbau und Komponenten einer SPS

Ein typisches SPS-System besteht aus mehreren Hauptkomponenten, die zusammenarbeiten, um eine effiziente Steuerung und Regelung von Prozessen zu gewährleisten [2]:

Zentraleinheit: Die Central Processing Unit (CPU) bildet das Herzstück der SPS. Sie verarbeitet die Eingangssignale, führt das Steuerungsprogramm aus und steuert die Ausgänge. Die CPU ist für die Berechnungen und logischen Operationen verantwortlich und sorgt dafür, dass die Steuerung in Echtzeit auf Veränderungen im Prozess reagieren kann.

Eingabemodule: Diese Module empfangen Signale von Sensoren, Tastern oder anderen Eingabegeräten und wandeln diese in digitale Signale um. Sie ermöglichen der SPS, Informationen über den Zustand von Maschinen, Anlagen oder Umgebungsbedingungen zu erhalten. Beispielsweise können Temperatursensoren, Druckschalter oder Näherungsschalter als Eingabegeräte dienen.

Ausgabemodule: Sie empfangen digitale Signale von der CPU und steuern damit Aktoren wie Motoren, Ventile oder Lampen. Die Ausgabemodule setzen die verarbeiteten Informationen der SPS in physische Aktionen um, die den Prozess steuern oder regeln. Beispielsweise können sie ein Relais ansteuern, das einen Motor startet, oder ein Ventil öffnen, um einen Flüssigkeitsstrom zu regulieren.

Kommunikationsmodule: Diese Module ermöglichen die Vernetzung der SPS mit anderen Systemen oder Geräten, beispielsweise über Feldbusse oder industrielle Netzwerke. Sie erweitern die Funktionalität der SPS, indem sie die Kommunikation mit übergeordneten Steuerungen, Leitsystemen oder anderen Maschinen ermöglichen. Dadurch kann die SPS in komplexe Automatisierungssysteme integriert werden, die eine zentrale Überwachung und Steuerung mehrerer Prozesse ermöglichen.

Programmiersprachen

Für die Programmierung von SPS-Systemen stehen verschiedene Programmiersprachen [3] zur Verfügung, die in der Internationale Norm für SPS-Programmiersprachen (IEC 61131-3) definiert sind:

Anweisungsliste(AWL): Die Anweisungsliste ist eine textbasierte Programmiersprache, die der Assemblersprache ähnelt. Sie ermöglicht eine direkte und effiziente Programmierung auf niedriger Ebene, was besonders bei ressourcenbeschränkten Systemen von Vorteil ist. Allerdings kann die Lesbarkeit und Wartbarkeit des Codes bei komplexeren Anwendungen erschwert sein.

Funktionsplan(FUP): Der Funktionsplan ist eine grafische Programmiersprache, die logische Funktionen in Form von Funktionsblöcken darstellt. Diese Blöcke sind durch Verbindungen miteinander verbunden, was eine klare und übersichtliche Darstellung der Logik ermöglicht. FUP eignet sich besonders für die Darstellung von logischen Verknüpfungen und wird häufig in der Prozessautomatisierung eingesetzt.

Kontaktplan(KOP): Der Kontaktplan ist eine weitere grafische Programmiersprache, die Schaltplänen ähnelt. Er stellt Kontakte und Relais in einer schematischen Form dar, was die Programmierung für Personen mit elektrotechnischem Hintergrund erleichtert. KOP ist besonders intuitiv und wird oft in der Maschinensteuerung verwendet.

Strukturierter Text(ST): Der strukturierte Text ist eine textbasierte Hochsprache, die der Programmiersprache Pascal ähnelt. Sie ermöglicht die Verwendung von komplexen Datenstrukturen, Schleifen und Bedingungen, was eine hohe Flexibilität bei der Programmierung bietet. ST eignet sich besonders für komplexe Algorithmen und wird häufig in der Prozesssteuerung eingesetzt.

Organisationsbausteine(OB): Organisationsbausteine sind spezielle Bausteine, die für zeit- und ereignisgesteuerte Aufgaben vorgesehen sind. Sie ermöglichen die Strukturierung des Programms in verschiedene Abschnitte, die zu bestimmten Zeiten oder bei bestimmten Ereignissen ausgeführt werden. OBs tragen zur Modularisierung und besseren Übersichtlichkeit des Programms bei.

Die Wahl der geeigneten Programmiersprache hängt von der Komplexität der zu steuernden Prozesse, den Anforderungen an die Lesbarkeit und Wartbarkeit des Codes sowie den individuellen Vorlieben und Erfahrungen des Programmierers ab. In der Praxis werden oft mehrere dieser Sprachen kombiniert, um die Vorteile jeder einzelnen optimal zu nutzen.

Anwendungsbereiche

SPS sind in der modernen Industrie unverzichtbar und finden in zahlreichen Bereichen Anwendung. Ihre Flexibilität und Anpassungsfähigkeit ermöglichen eine effiziente Steuerung und Überwachung verschiedener Prozesse.

In der Fertigung übernehmen SPS die Steuerung von Produktionslinien und Maschinen. Sie koordinieren Abläufe, überwachen Produktionsparameter und gewährleisten eine gleichbleibende Produktqualität. Beispielsweise steuern sie Roboterarme, Förderbänder und Werkzeugmaschinen, um eine effiziente Produktion zu ermöglichen.

In der Prozessindustrie regeln SPS chemische Prozesse, Wasseraufbereitung oder Energieversorgung. Sie überwachen kritische Parameter wie Temperatur, Druck und Durchfluss und reagieren in Echtzeit auf Veränderungen, um die Sicherheit und Effizienz der Prozesse zu gewährleisten.

SPS steuern in Gebäuden Systeme wie Heating, Ventilation, Air Conditioning (Heizung, Lüftung, Klimatisierung) (HVAC). Sie ermöglichen eine zentrale Überwachung und Steuerung, was zu Energieeinsparungen und erhöhtem Komfort führt. Durch die Integration verschiedener Systeme können beispielsweise Beleuchtung und Klimaanlage automatisch an die Anwesenheit von Personen angepasst werden.

Im Bereich der Verkehrstechnik übernehmen SPS die Steuerung von Ampelanlagen und Verkehrsleitsystemen. Sie optimieren den Verkehrsfluss, reduzieren Staus und erhöhen die Sicherheit im Straßenverkehr. Durch die Echtzeitverarbeitung von Verkehrsdaten können Ampelphasen dynamisch angepasst werden, um den Verkehrsfluss zu verbessern.

In der Energiebranche werden SPS zur Steuerung von Stromnetzen, Energieerzeugungsanlagen und zur Implementierung von Lastmanagement-Systemen verwendet. Sie tragen dazu bei, den Energieverbrauch zu optimieren und die Netzstabilität sicherzustellen. Durch die Überwachung und Steuerung von Energieflüssen können beispielsweise erneuerbare Energiequellen effizient in das Netz integriert werden.

SPS steuern in Kläranlagen Pumpen, Mischer und Chemikaliendosierungen. Sie gewährleisten eine effiziente Abwasserbehandlung und tragen so zum Umweltschutz bei. Durch die präzise Steuerung von Prozessen können beispielsweise Schadstoffkonzentrationen im Abwasser effektiv reduziert werden.

In modernen Automatisierungssystemen lassen sich SPS und übergeordnete Systeme über OPC-UA verbinden. Viele SPS, wie die von Siemens, bieten eine integrierte OPC-UA-Schnittstelle, Wie dies im Totally Integrated Automation Portal (TIA Portal) umgesetzt wird, wird später in kapitel 5.1 beschrieben.

2.2 OPC-UA als Kommunikationsstandard

Geschichte und Entwicklung

Die OPC-UA wurde entwickelt, um den zunehmenden Anforderungen der Industrie an eine zuverlässige, sichere und flexible Kommunikation gerecht zu werden. Ursprünglich in den späten 1990er Jahren auf der Grundlage der älteren OPC-Standards konzipiert, wurde OPC-UA als Reaktion auf die wachsende Komplexität und Vielfalt von Automatisierungssystemen ins Leben gerufen [4]. Der Standard sollte die Grenzen der älteren Technologien überwinden und eine Lösung bieten, die plattformunabhängig und skalierbar ist, um den verschiedenen Anforderungen moderner Industrien gerecht zu werden.

Die erste Version von OPC-UA wurde 2006 veröffentlicht und brachte bedeutende Verbesserungen mit sich. Im Gegensatz zu den vorherigen OPC-Standards, die stark an spezifische Betriebssysteme und Hardware gebunden waren, bot OPC-UA eine vollständig Serviceorientierte Architektur (SOA), die eine breitere Interoperabilität ermöglichte. Es ermöglichte eine nahtlose Kommunikation zwischen verschiedenen Geräten und Systemen, unabhängig von deren Hersteller oder Plattform. Diese Flexibilität wurde zu einem zentralen Merkmal von OPC-UA, das nicht nur in der Fertigungsbereich, sondern auch in anderen Bereichen wie Prozessautomatisierung, Gebäudeautomation und sogar der Energieversorgung erfolgreich eingesetzt wird.

Architektur

Die Architektur von OPC-UA folgt einem Client-Server-Modell [5], bei dem der Client Anfragen an den Server sendet, um Daten abzurufen. Der Server stellt dabei eine Vielzahl von Ressourcen wie Datenpunkte, Methoden und Ereignisse zur Verfügung, auf die der Client zugreifen kann.

OPC-UA basiert zudem auf einer SOA, die es ermöglicht, verschiedene Kommunikationsprotokolle miteinander zu verbinden und eine flexible Datenmodellierung zu realisieren. Diese Architektur ist darauf ausgelegt, eine reibungslose und effiziente Kommunikation zwischen unterschiedlichsten Geräten und Systemen zu ermöglichen – unabhängig von deren Betriebssystemen oder Hardware-Plattformen.

Die OPC-UA-Architektur ist in mehrere Schichten unterteilt, die jeweils eine spezifische Aufgabe in der Kommunikation und Datenverarbeitung übernehmen. Die wichtigste dieser Schichten sind:

Informationsmodellierung: Diese Schicht ist das Herzstück der Architektur. Hier werden Daten in einer strukturierten Form modelliert, sodass sie mit Kontext und Bedeutung versehen sind. Anstatt nur rohe Daten zu übertragen, ermöglicht OPC-UA eine komplexe Modellierung von Objekten, Variablen, Methoden und Ereignissen.

Serviceschicht: Diese Schicht sorgt dafür, dass alle Kommunikation und Interaktionen zwischen Client und Server auf einem standardisierten Satz von Diensten basieren. Diese Dienste umfassen unter anderem Funktionen zur Abfrage von Daten, zum Abonnieren von Ereignissen oder zum Starten von Methoden. Die Serviceschicht stellt sicher, dass alle Aktionen, die von einem Client angefordert werden, auf konsistente und standardisierte Weise ausgeführt werden.

Transportschicht: Diese Schicht ist dafür verantwortlich, die Daten sicher und effizient zwischen den beteiligten Systemen zu übertragen. Sie unterstützt eine Vielzahl von Protokollen wie Transmission Control Protocol / Internet Protocol (TCP/IP), Hypertext Transfer Protocol Secure (HTTPS) und WebSockets, um eine sichere und schnelle Kommunikation zu gewährleisten.

Vorteile von OPC-UA

Ein großer Vorteil von OPC-UA ist seine enorme Flexibilität und Skalierbarkeit. Dieser Standard lässt sich sowohl in kleinen Anwendungen als auch in großen, komplexen Netzwerken problemlos einsetzen. Das bedeutet, dass er nicht nur für einzelne Maschinen verwendet werden kann, sondern auch für die Vernetzung ganzer Produktionsanlagen oder sogar kompletten Unternehmensnetzwerken. Ein weiterer Pluspunkt von OPC-UA ist die Interoperabilität – also die Fähigkeit, dass Geräte und Systeme von unterschiedlichen Herstellern miteinander kommunizieren können.

Im Vergleich zu älteren Kommunikationsprotokollen wie Modbus oder Profibus bringt OPC-UA eine Reihe an Vorteilen mit sich. Während diese älteren Protokolle häufig auf spezifische Hardware oder Betriebssysteme angewiesen sind, ermöglicht OPC-UA eine plattformübergreifende Kommunikation. Das bedeutet, dass sowohl alte als auch neue Geräte problemlos miteinander verbunden werden können – was für die Zukunftssicherheit von Systemen enorm wichtig ist. Darüber hinaus unterstützt OPC-UA eine deutlich umfassendere Datenmodellierung und bietet starke Sicherheitsfunktionen wie Verschlüsselung und Authentifizierung, die bei älteren Protokollen oft nicht vorhanden sind [6].

OPC-UA ist also ein zukunftsweisender Kommunikationsstandard, der perfekt zu den Anforderungen der Industrie 4.0 und des Internet of things (IoT) passt. Er bietet eine zuverlässige und sichere Möglichkeit, dass verschiedene Systeme und Geräte problemlos miteinander kommunizieren können. Ein großer Vorteil von OPC-UA ist nicht nur die herstellerunabhängige Kommunikation, sondern auch die sichere Übertragung von Daten. Deshalb setzt OPC-UA auf umfassende Sicherheitsmechanismen und Authentifizierungsverfahren, die im nächsten Abschnitt näher betrachtet werden.

2.3 Sicherheit und Authentifizierung in OPC-UA

Ein wesentlicher Grund, warum OPC-UA so weit verbreitet ist, liegt in seinem ausgefeilten Sicherheitskonzept. Gerade in industriellen Umgebungen, wo sensible Daten über Produktionsprozesse, Energieversorgung und andere kritische Bereiche übertragen werden, ist es enorm wichtig, dass diese Kommunikation nicht von Unbefugten abgefangen oder manipuliert werden kann.

Nachfolgend eine Übersicht über die wichtigsten Sicherheitsfunktionen und Mechanismen von OPC-UA:

Authentifizierung

OPC-UA setzt auf X.509-Zertifikate [7], um sicherzustellen, dass sich sowohl der Client als auch der Server gegenseitig identifizieren können, bevor überhaupt eine Verbindung aufgebaut wird. Das bedeutet: Nur Geräte und Nutzer, die ein gültiges Zertifikat besitzen, können überhaupt Daten austauschen. Auf diese Weise wird von Anfang an ausgeschlossen, dass unberechtigte Teilnehmer ins Netzwerk eindringen.

Autorisierung

Selbst wenn ein Gerät oder Nutzer über ein Zertifikat verfügt und sich damit erfolgreich ausweisen kann, heißt das noch nicht, dass es automatisch alle Daten lesen oder ändern darf. Hier greift die sogenannte Autorisierung [8]. Mithilfe feingranularer Richtlinien lässt sich festlegen, welche Informationen ein bestimmter Benutzer oder Client überhaupt sehen oder bearbeiten darf.

Verschlüsselung

Damit niemand die ausgetauschten Informationen während der Übertragung „mit hören“ kann, verwendet OPC-UA moderne Verschlüsselungsverfahren zum Beispiel Advanced Encryption Standard (AES). Das heißt, die Daten werden so verschlüsselt, dass Außenstehende sie nicht entziffern können. Das ist besonders wichtig, wenn die Netzwerkverbindungen nicht vollständig abgesichert sind oder wenn Daten über das Internet verschickt werden.

Datenintegrität

Neben der Verschlüsselung sorgt eine digitale Signatur dafür, dass die übertragenen Daten unterwegs nicht unbemerkt verändert werden können. Sobald jemand versucht, die Daten zu manipulieren, wird diese Signatur ungültig, und der Empfänger erkennt sofort, dass etwas nicht stimmt.

Sicherheitsprotokolle

OPC-UA erlaubt unterschiedliche Sicherheitsmodi, etwa nur Signatur (wenn nur sichergestellt werden soll, dass Daten nicht verändert werden) oder SignAndEncrypt (wenn zusätzlich die Vertraulichkeit durch Verschlüsselung gewährleistet werden muss)[9]. Zusätzlich können Verbindungen über Protokolle wie HTTPS etabliert werden, was noch einmal eine Extra-Schicht an Sicherheit und Verschlüsselung hinzufügt.

3 Anforderungsanalyse und Systemarchitektur

3.1 Anforderungsanalyse

Bevor mit der eigentlichen Implementierung begonnen wird, ist es wichtig, die Anforderungen an das System klar zu definieren. In diesem Abschnitt werden zunächst die funktionalen Anforderungen formuliert, die beschreiben, welche konkreten Aufgaben das System erfüllen soll. Anschließend folgen die nicht-funktionalen Anforderungen, die vor allem Qualitäts- und Rahmenbedingungen betreffen, unter denen das System eingesetzt wird.

3.1.1 Funktionale Anforderungen

Verbindungsauflbau

Die Anwendung muss eine sichere und stabile Verbindung zu einer SPS über OPC-UA herstellen. Dabei soll sie Änderungen im Netzwerk automatisch erkennen und sich bei Verbindungsabbrüchen eigenständig neu verbinden können.

Durchsuchen und Auslesen von Variablen

Ein zentrales Feature ist das Browsing aller SPS-Variablen. Über eine übersichtliche Oberfläche soll der Benutzer gezielt nach bestimmten Werten suchen und diese in Echtzeit einsehen können.

Herstellung der Datenbankverbindung

Die Anwendung muss es dem Benutzer ermöglichen, eine Verbindung zu einer relationalen Datenbank herzustellen und diese Verbindung für die spätere Datenspeicherung zu nutzen. Über die Benutzeroberfläche der Anwendung kann der Benutzer die erforderlichen Verbindungsparameter angeben, wie die Datenbankadresse, Benutzernamen und Passwort.

Daten in Excel speichern

Auf Basis definierter Trigger soll die Anwendung Variablenwerte automatisch in Excel-Dateien schreiben können. Ob einzelne Zellen, ganze Zeilen oder ganze Datenblöcke – der Anwender entscheidet, wie und wohin die Werte exportiert werden. Wichtig ist dabei, dass vorhandene Einträge nicht versehentlich überschrieben werden, sondern neue Zeilen angefügt oder bestehende Blätter erweitert werden.

Daten in Datenbank speichern

Neben der Speicherung in Excel-Dateien soll die Anwendung auch die Möglichkeit bieten, Variablenwerte zusammen mit einem Zeitstempel in einer relationalen Datenbank abzulegen. Hierfür können spezielle Datenbanktrigger konfiguriert werden, die bestimmen, wann die Daten tatsächlich in die Tabelle geschrieben werden .

Anzeige der gespeicherten Daten

Die Anwendung muss es ermöglichen, die in der Datenbank gespeicherten Daten auf verschiedene Weisen anzuzeigen. Eine grundlegende Anzeige der Daten soll in tabellarischer Form erfolgen, damit der Anwender die erfassten Variablenwerte und Zeitstempel einfach einsehen kann. Darüber hinaus soll es die Möglichkeit geben, diese Daten in Diagrammen darzustellen, um Trends und Zusammenhänge besser zu visualisieren. Die Auswahl der anzuzeigenden Variablen sowie die Darstellung der Daten in Diagrammen soll intuitiv über die Benutzeroberfläche erfolgen, sodass der Anwender schnell auf die wichtigsten Informationen zugreifen und diese analysieren kann.

Analysefunktion

Die Anwendung soll grundlegende Analysen der erfassten Daten ermöglichen. Dazu gehört beispielsweise das Gegenüberstellen von zwei verschiedenen Variablen (z. B. Temperatur und Druck), das Erkennen von Trends oder das Auswerten von Grenzwertüberschreitungen. Die Analyseergebnisse sollen sowohl numerisch als auch visuell (z. B. in Diagrammen) dargestellt werden, damit der Anwender schnell einen Überblick über den Zustand des Systems bekommt.

Benutzerfreundliche Oberfläche

Eine intuitive Bedienoberfläche ist entscheidend. Sie soll eine klare Navigation bieten, damit Anwender Verbindungen einrichten, Variablen durchsuchen und Trigger konfigurieren können. Neben Echtzeit-Informationen wie dem Verbindungsstatus oder Fehlermeldungen sollten Hilfetexte und Tooltips verfügbar sein, die den Benutzer bei Bedarf unterstützen.

3.1.2 Nicht-funktionale Anforderungen

Sicherheit

Damit die Anwendung sicher in einer industriellen Umgebung betrieben werden kann, muss die Kommunikation zwischen OPC-UA-Client und SPS durch moderne Schutzmechanismen gesichert sein. Das bedeutet unter anderem, dass nur autorisierte Clients auf die SPS zugreifen dürfen und alle Daten während der Übertragung verschlüsselt werden.

Systemleistung

Gerade in Echtzeit-Anwendungen kommt es darauf an, dass das System auch bei einer großen Anzahl an Variablen oder komplexen Datenstrukturen schnell reagiert. Die Latenzzeiten beim Zugriff auf die SPS sollten möglichst gering sein. Das gilt nicht nur für die Kommunikation zwischen Client und SPS, sondern ebenso für die Speicherung und Abfrage großer Datenmengen in der Datenbank.

Zuverlässigkeit

Eine zuverlässige Funktionsweise ist entscheidend, gerade wenn es um industrielle Prozesse geht. Die Anwendung sollte daher möglichst robust sein und Fehler frühzeitig erkennen. Verbindungsprobleme zur SPS oder Fehler in der Datenverarbeitung müssen automatisch gehandhabt werden können, ohne dass der Anwender eingreifen muss. Damit sich Probleme im Nachhinein leichter analysieren lassen, sollten Fehlermeldungen klar und verständlich protokolliert.

Erweiterbarkeit

Die Software sollte so aufgebaut sein, dass neue Funktionen oder Schnittstellen ohne großen Aufwand hinzugefügt werden können. Das kann zum Beispiel bedeuten, dass sich zukünftig weitere SPS-Typen oder Kommunikationsprotokolle leicht integrieren lassen. Auch die Benutzeroberfläche sollte sich flexibel anpassen lassen, sodass zusätzliche Features eingebaut werden können, ohne die bestehende Struktur zu stören.

Benutzerfreundlichkeit

Eine intuitive und übersichtliche Benutzeroberfläche ist unerlässlich, damit auch Anwender ohne tiefgreifende Vorkenntnisse schnell verstehen, wie sie die Software nutzen können. Dazu gehören eine logische Menüführung, klare Beschriftungen und eine einheitliche Gestaltung. Farbliche Markierungen und kleine Hilfestellungen (z.B. Tooltips) helfen, sich in der Anwendung zurechtzufinden.

Skalierbarkeit

Die Anwendung sollte in der Lage sein, auch bei steigenden Anforderungen – etwa einer größeren Zahl an Variablen oder höheren Datenaufkommen in der Datenbank – flüssig zu laufen. Dafür braucht es eine Architektur, die sowohl in Bezug auf Rechenleistung als auch Speicherbedarf mitwachsen kann. Außerdem sollte die Lösung so ausgelegt sein, dass sie auch auf künftige Entwicklungen im OPC-UA-Bereich oder neue Anforderungen der Industrie 4.0 reagieren kann, ohne von Grund auf neu entwickelt werden zu müssen.

3.2 Systemarchitektur

In diesem Abschnitt wird die grundlegende Struktur des Gesamtsystems beschrieben. Ziel ist es, deutlich zu machen, wie die einzelnen Komponenten miteinander kommunizieren und wie diese Architektur die Anforderungen aus Kapitel 3.1 erfüllt. Das folgende Diagramm zeigt, wie die verschiedenen Blöcke zusammenarbeiten und die benötigten Funktionen bereitstellen.

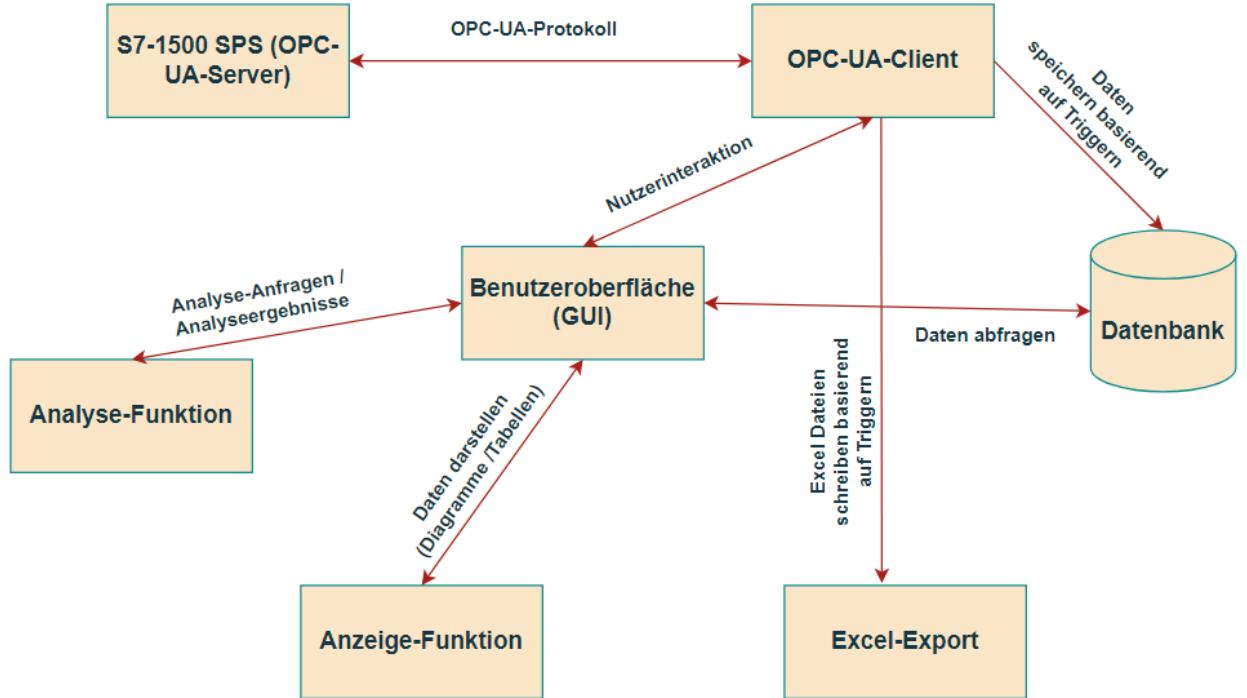


Abbildung 1: Systemarchitektur der Anwendung

S7-1500 SPS

Im Zentrum der Architektur steht die SIMATIC S7-1500 [10] von Siemens die in der Firma Schiele als SPS eingesetzt wird. Diese SPS fungiert als OPC-UA-Server und stellt Prozessdaten in Form von Variablen zur Verfügung, die von der Anwendung abgerufen werden. Sie ist die Hauptquelle für die zu verarbeitenden Daten. Die SPS implementiert Sicherheitsmechanismen wie Authentifizierung und Zugriffskontrolle, um sicherzustellen, dass nur autorisierte Clients auf die Daten zugreifen können. Darüber hinaus sorgt der OPC-UA-Server für eine stabile und sichere Kommunikation zwischen der SPS und der Client-Anwendung über das OPC-UA-Protokoll.

OPC-UA-Client

Die Desktop-Anwendung enthält einen OPC-UA-Client, der für die Kommunikation mit der SPS zuständig ist. Der Client stellt sicher, dass die Verbindung korrekt aufgebaut wird und verwaltet den Datentransfer zwischen den beiden Systemen. Er liest die Variablenwerte der SPS aus. Die Zuverlässigkeit und Stabilität des Systems werden durch den OPC-UA-Client gewährleistet, da er sicherstellt, dass bei

Verbindungsproblemen oder Netzwerkfehlern die Verbindung schnell wiederhergestellt wird.

Benutzeroberfläche

Die Grafische Benutzeroberfläche (GUI) fungiert als zentrale Steuerungseinheit für den Anwender. Hier können alle relevanten Aktionen wie das Durchsuchen der SPS-Variablen, das Konfigurieren von Triggern oder das Verwalten von Verbindungen vorgenommen werden. Die GUI gibt Echtzeitinformationen über den Verbindungsstatus, gespeicherte Daten und etwaige Fehler. Sie ermöglicht es dem Benutzer, Daten zu visualisieren und Konfigurationen vorzunehmen, die die Datenverarbeitung und -analyse betreffen. Besonders wichtig ist die Benutzerfreundlichkeit der Oberfläche, die sicherstellt, dass auch Nutzer ohne tiefgreifende technische Kenntnisse das System effizient bedienen können.

Datenbank

Die Datenbank dient als langfristiger Speicherort für die erfassten Variablenwerte. Alle Daten, die von der SPS über den OPC-UA-Client erfasst werden, werden mit einem Zeitstempel versehen und in der Datenbank gespeichert. Diese Speicherung erfolgt abhängig von den konfigurierten Triggerbedingungen. Die Datenbank sorgt dafür, dass große Datenmengen effizient gespeichert und schnell wieder abgerufen werden können. Sie unterstützt so die Datenpersistenz und ermöglicht eine schnelle Datenabfrage.

Analyse-Funktion

Die Analyse-Funktion ist dafür zuständig, die gespeicherte Daten aus der Datenbank zu verarbeiten und dem Benutzer grundlegende Analyse- und Auswertungsfunktionen zur Verfügung zu stellen. Diese Funktion ermöglicht es dem Benutzer, verschiedene Variablen miteinander zu vergleichen und Trends zu erkennen. Die Ergebnisse der Analyse werden dem Benutzer in Form von Texten und Diagrammen angezeigt, um eine schnelle Interpretation der Daten zu ermöglichen.

Anzeige-Funktion

Die Anzeige-Funktion ist direkt mit der GUI verbunden und stellt die visualisierte Darstellung der Daten bereit. Diese Funktion ist verantwortlich für die grafische Darstellung der erfassten Daten, sei es in Form von Diagrammen oder Tabellen. Der Benutzer kann hier die Daten nach verschiedenen Kriterien filtern und die Darstellung anpassen. Sie zeigt die Daten auf Basis historischer Werte an, was besonders für die Überwachung und Analyse von Prozessen wichtig ist.

Excel-Export

Der Excel-Export ermöglicht es dem Benutzer, die erfassten Daten in Excel-Dateien zu exportieren. Dieser Export erfolgt basierend auf den definierten Triggerbedingungen, was bedeutet, dass die SPS Daten bei bestimmten Ereignissen in Excel-Dateien geschrieben werden. Der Export ermöglicht eine weitergehende Bearbeitung der Daten oder deren Verwendung in externen Anwendungen.

Diese Systemarchitektur stellt sicher, dass die Anwendung robust, flexibel und benutzerfreundlich bleibt. Sie erfüllt die in Kapitel 3.1 definierten funktionalen und nicht-funktionalen Anforderungen und ermöglicht eine effiziente Datenerfassung, -verarbeitung und -analyse.

4 Implementierung

4.1 Auswahl der verwendeten Technologien und Tools

Die Entwicklung der Anwendung basiert auf einem durchdachten Einsatz moderner Tools und Technologien, die speziell für die Anforderungen industrieller Automatisierungslösungen ausgewählt wurden. Im Zentrum stehen dabei die Programmiersprache C sharp , das .NET Framework und Visual Studio als Entwicklungsumgebung.

C sharp

C sharp wurde aufgrund seiner Vielseitigkeit und Leistungsfähigkeit gewählt. Es ist eng mit dem .NET Framework integriert und eignet sich hervorragend für Anwendungen auf Microsoft-Systemen. Besonders vorteilhaft ist die Verfügbarkeit der OPC UA .NET Standard Library, die eine effiziente Integration des OPC-UA-Protokolls ermöglicht. Im Vergleich zu anderen Sprachen wie Python bietet C sharp eine bessere Performance und Unterstützung für Multithreading, was für industrielle Anwendungen wichtig ist.

.NET Framework

Das .NET Framework 4.7.1 bildet die technologische Grundlage der Anwendung. Es bietet eine umfangreiche Klassenbibliothek, die alles von Datei- und Netzwerkverwaltung bis hin zu Multithreading und Verschlüsselung abdeckt. Besonders relevant für dieses Projekt ist die Integration von Sicherheitsmechanismen wie Transport Layer Security (TLS) 1.2, die für die verschlüsselte Kommunikation zwischen der OPC-UA-Anwendung und die SPS erforderlich sind.

Windows Presentation Foundation

Für die grafische Benutzeroberfläche wurde Windows Presentation Foundation (WPF) verwendet. Mit WPF können moderne und interaktive Benutzeroberflächen erstellt werden, die eine klare Trennung zwischen Logik und Darstellung ermöglichen. WPF unterstützt die Datenbindung und garantiert, dass sich die Benutzeroberfläche an unterschiedliche Bildschirmgrößen und -auflösungen anpasst.

Visual Studio

Die Entwicklung erfolgte vollständig in Visual Studio, einer der führenden Entwicklungsumgebungen für C sharp und .NET-Projekte. Visual Studio bietet umfangreiche Werkzeuge wie einen visuellen Editor, eine Echtzeitvorschau und eine enge Integration mit Extensible Application Markup Language (XAML) für die Gestaltung von WPF-Anwendungen. Die integrierte Debugging-Umgebung ermöglicht eine schnelle Fehlerbehebung, und der NuGet-Paketmanager erleichtert die Installation und Verwaltung von Bibliotheken.

OPC-UA .NET Standard Library

Die Kommunikation zwischen der Anwendung und dem OPC-UA-Server der SIMATIC S7-1500 wird über die OPC-UA .NET Standard Library realisiert. Diese von der OPC Foundation entwickelte Bibliothek bietet eine umfassende Application Programming Interface (API) für das Herstellen sicherer Verbindungen, das Durchsuchen von OPC-UA-Adressräumen und das Lesen/Schreiben von Variablenwerten. Sie unterstützt auch Verschlüsselung und Zertifikats-basierte Authentifizierung, um die Sicherheit der Kommunikation zu gewährleisten.

EPPlus

Für den Export von Daten in Excel-Dateien wird die Bibliothek EPPlus verwendet. EPPlus ermöglicht das Schreiben und Bearbeiten von Excel-Dokumenten im Microsoft Excel Open XML Spreadsheet Format (XLSX) Format. Sie bietet eine einfache und effiziente Möglichkeit, Daten aus der Anwendung zu exportieren, ohne auf Microsoft Excel angewiesen zu sein. EPPlus bietet eine schnelle und performante Möglichkeit, Excel-Dokumente zu generieren und ist daher besonders nützlich für die Erstellung von Berichten oder die Speicherung von Daten zur späteren Analyse.

PdfSharpCore und iText 9.0.0

Für den PDF-Export kommen die Bibliotheken PdfSharpCore und iText 9.0.0 zum Einsatz. PdfSharpCore ermöglicht das Erstellen von PDF-Dokumenten und das Bearbeiten von bestehenden PDFs, während iText 9.0.0 eine umfangreiche Bibliothek ist, die zusätzliche Funktionen für die PDF-Verarbeitung bereitstellt, wie etwa das Hinzufügen von Text, Bildern und Tabellen. Beide Bibliotheken zusammen ermöglichen es der Anwendung, dynamisch PDF-Berichte zu erstellen, die für die Dokumentation und Archivierung von wichtigen Daten genutzt werden können.

LiveCharts und OxyPlot

Zur Visualisierung von Daten in Diagrammen werden die Bibliotheken LiveCharts und OxyPlot verwendet. LiveCharts bietet eine benutzerfreundliche Möglichkeit, interaktive Diagramme für Echtzeitdaten zu erstellen, während OxyPlot eine leistungsstarke, plattformübergreifende Bibliothek für die Erstellung von 2D-Diagrammen ist. Beide Tools bieten die Flexibilität, verschiedene Diagrammtypen wie Linien-, Balken- und Kreisdiagramme zu erstellen, die in der Anwendung zur Darstellung von Prozessdaten und Trends genutzt werden.

Datenbanken: MySQL, MSSQL, PostgreSQL

Die Anwendung ist mit drei gängigen relationalen Datenbankmanagementsystemen kompatibel: MySQL, MSSQL und PostgreSQL. Diese Flexibilität ermöglicht es, die Anwendung in unterschiedlichen Umgebungen einzusetzen, ohne an ein bestimmtes Database Management System (Datenbankverwaltungssystem) (DBMS) gebunden zu sein. Die Anwendung unterstützt Datenbankverbindungen zu allen drei Systemen und stellt sicher, dass Daten in einer stabilen und performanten Weise gespeichert und abgerufen werden können, unabhängig von der verwendeten Datenbanktechnologie.

OPC UA Simulation Server: Prosys und Unified Automation

Während der Entwicklungs- und Testphase kamen zwei OPC UA Simulation Server zum Einsatz: der Prosys OPC UA Simulation Server und der Unified Automation OPC UA Simulation Server. Beide Server simulieren die Kommunikation mit einem echten OPC-UA-Server und stellen eine Vielzahl von Testdatenpunkten zur Verfügung, die für das Testen und Verifizieren der Anwendung genutzt werden konnten. Diese Simulationen ermöglichen es, die Funktionalität der Anwendung vollständig zu überprüfen, ohne auf die physische SIMATIC S7-1500 SPS zugreifen zu müssen, wodurch der Testprozess erheblich vereinfacht und beschleunigt wurde.

4.2 OPC-UA-Verbindung

In diesem Abschnitt wird die Implementierung der OPC-UA-Verbindung beschrieben. Es wird erklärt, wie die Anwendung als OPC-UA-Client agiert und eine Verbindung zu einer Siemens S7-1500 SPS herstellt. Dabei werden die grundlegenden Schritte zur Konfiguration, Authentifizierung und Datenübertragung aufgezeigt. Außerdem wird darauf eingegangen, wie die Verbindung überwacht wird.

4.2.1 Architektur der OPC-UA-Verbindung

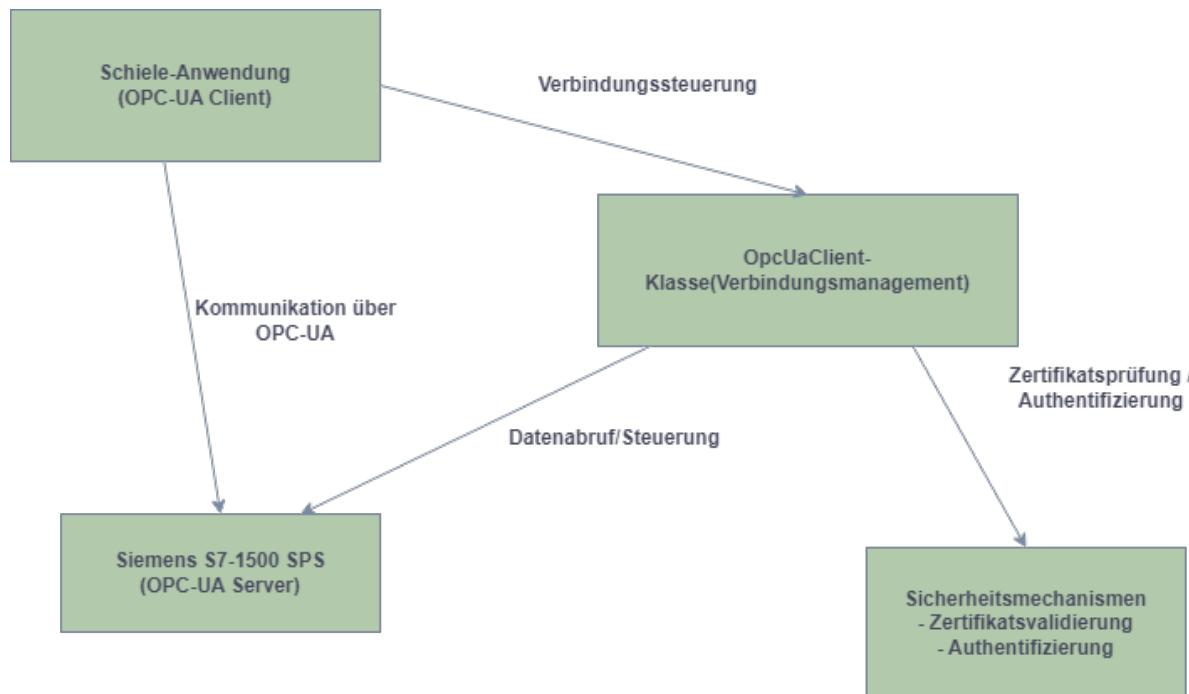


Abbildung 2: Komponentendiagramm zur Architektur der OPC-UA-Verbindung

Die OPC-UA-Verbindung in dieser Anwendung basiert auf einer Client-Server-Architektur. Die Siemens S7-1500 SPS fungiert als OPC-UA-Server und stellt Datenpunkte bereit, auf die die Anwendung zugreifen kann. Die Anwendung selbst arbeitet als OPC-UA-Client und ermöglicht das Abrufen und Schreiben von Daten sowie die Überwachung bestimmter Variablen.

Um eine stabile und sichere Verbindung zu gewährleisten, übernimmt die OpcUaClient-Klasse die Verwaltung der gesamten Kommunikation. Diese Klasse ist dafür verantwortlich, die Verbindung zum Server aufzubauen, sich gegebenenfalls zu authentifizieren und Daten in Echtzeit zu übertragen.

Zusätzlich kommen verschiedene Sicherheitsmechanismen zum Einsatz. Dazu gehören die Zertifikatsvalidierung sowie die Möglichkeit einer Benutzer-Authentifizierung, falls der OPC-UA-Server dies verlangt. Diese Mechanismen stellen sicher, dass nur autorisierte Clients auf die Steuerungssysteme zugreifen können.

4.2.2 Verbindungsauftbau und Speicherung der Verbindung

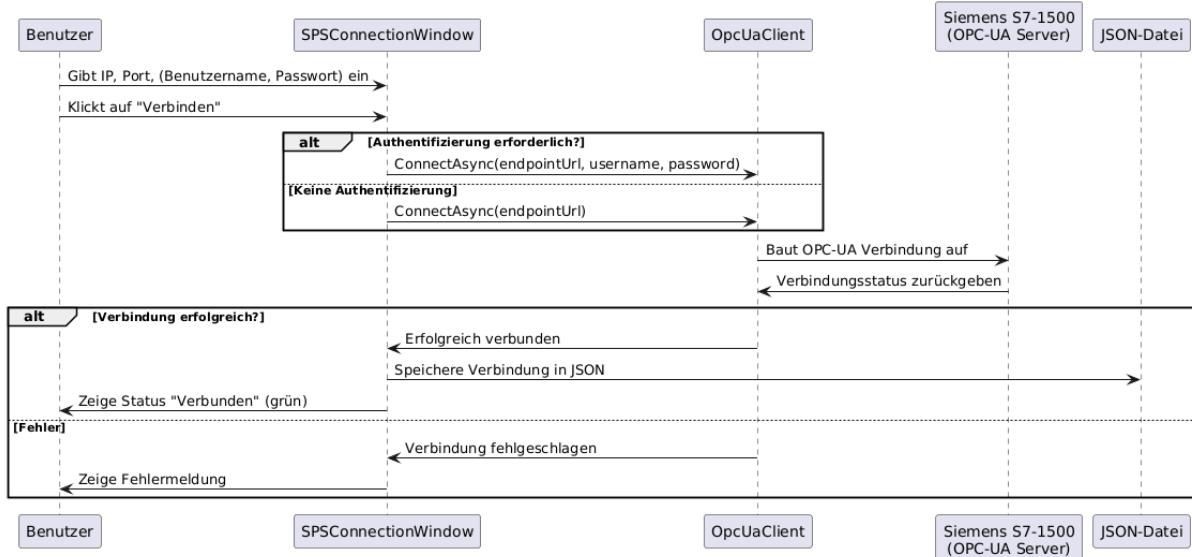


Abbildung 3: Sequenzdiagramm der Verbindungsauftbau

Das Sequenzdiagramm beschreibt den Ablauf der OPC-UA-Verbindungserstellung in der Anwendung. Der Benutzer gibt die Verbindungsparameter ein, darunter IP-Adresse, Port und optional Benutzername und Passwort, falls eine Authentifizierung erforderlich ist. Sobald der Benutzer auf „Verbinden“ klickt, überprüft das System, ob Anmeldedaten eingegeben wurden. Falls ja, wird die Verbindung mit Authentifizierung (ConnectAsync(endpointUrl, username, password)) hergestellt, andernfalls ohne (ConnectAsync(endpointUrl)).

Der OpcUaClient versucht dann, eine Verbindung zur Siemens S7-1500 SPS (OPC-UA Server) aufzubauen. Der Server gibt anschließend den Status der Verbindung zurück: Wenn die Verbindung erfolgreich ist, wird sie in einer JavaScript Object Notation (JSON)-Datei gespeichert. Dadurch kann sie später in der Benutzeroberfläche angezeigt werden. Zusätzlich wird dem Benutzer der Status „Verbunden“ (grün) angezeigt. Falls die Verbindung fehlschlägt, erhält der Benutzer eine Fehlermeldung, aber die fehlgeschlagene Verbindung wird nicht gespeichert.

4.2.3 Verbindungsauftbau mit Authentifizierung

In industriellen Umgebungen ist eine gesicherte OPC-UA-Verbindung mit Benutzeranmeldung oft erforderlich. Daher wird ConnectAsync(string endpointUrl, string username, string password) bevorzugt verwendet, um eine Session mit Authentifizierung aufzubauen. Diese Methode wird im SPSConnectionWindow verwendet, wenn der Benutzer eine Verbindung zu einer SPS herstellt.

```

public async Task<bool> ConnectAsync(string endpointUrl, string username, string password)
{
    try
    {
        // OPC UA Client Configuration with authentication
        config = new ApplicationConfiguration
        {
            ApplicationName = "OPCUAClient",
            ApplicationType = ApplicationType.Client,
            SecurityConfiguration = new SecurityConfiguration
            {
                AutoAcceptUntrustedCertificates = true,
                ApplicationCertificate = new CertificateIdentifier
                {
                    StoreType = "Directory",
                    StorePath = "pki/own",
                    SubjectName = "OPCUAClient"
                }
            },
            ClientConfiguration = new ClientConfiguration { DefaultSessionTimeout = 60000 }
        };

        // Validate the configuration
        await config.Validate(ApplicationType.Client);

        // Select the endpoint
        var selectedEndpoint = CoreClientUtils.SelectEndpoint(endpointUrl, false, 15000);
        var endpointConfiguration = EndpointConfiguration.Create(config);
        var endpoint = new ConfiguredEndpoint(null, selectedEndpoint, endpointConfiguration);

        // Use credentials for authentication
        UserIdentity userIdentity = new UserIdentity(username, password);

        // Create the session and attempt connection with credentials
        session = await Session.Create(config, endpoint, false, "OPC UA Client", 60000, userIdentity, null);

        return session != null && session.Connected;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Connection failed with credentials: {ex.Message}");
        return false;
    }
}

```

Abbildung 4: ConnectAsync Methode

Der erste Schritt in dieser Methode ist die Erstellung einer ApplicationConfiguration, die die grundlegenden Einstellungen für den OPC-UA-Client enthält. Dazu gehören unter anderem der Anwendungsname, die Sicherheitsrichtlinien und die Behandlung von Zertifikaten. In diesem Fall ist die Option AutoAcceptUntrustedCertificates aktiviert, um selbstsignierte Zertifikate automatisch zu akzeptieren, was den Verbindungsauflaufbau erleichtert.

Nachdem die Konfiguration validiert wurde, wird ein geeigneter OPC-UA-Endpunkt ermittelt. Dies geschieht mit der Methode SelectEndpoint, die automatisch den besten verfügbaren Endpunkt mit den passenden Sicherheitsparametern auswählt. Anschließend wird dieser Endpunkt weiter konfiguriert und für die Verbindung vorbereitet.

Der nächste zentrale Schritt ist die Erstellung einer Session. Diese Session repräsentiert die eigentliche Verbindung zwischen dem Client und dem OPC-UA-Server. Hierbei werden die Benutzeranmeldeinformationen (username und password) verwendet, um sich beim Server zu authentifizieren. Die Methode Session.Create übernimmt diesen Prozess und versucht, eine Verbindung mit den übergebenen Zugangsdaten herzustellen.

Sobald die Session aufgebaut wurde, überprüft die Methode, ob die Verbindung erfolgreich war. Falls die Session existiert und als „verbunden“ (session.Connected) markiert ist, wird true zurückgegeben. Andernfalls schlägt die Verbindung fehl, und die Methode gibt false zurück.

Falls während des Verbindungsauflaufbaus ein Fehler auftritt – sei es durch falsche Zugangsdaten, einen nicht erreichbaren Server oder ein Zertifikatsproblem – wird der Fehler durch den catch-Block abgefangen.

4.2.4 Verwaltung und Speicherung der OPC-UA-Verbindungen

Nachdem eine Verbindung erfolgreich hergestellt wurde, muss sie verwaltet werden, damit der Benutzer sie später wiederverwenden kann. Dies geschieht durch eine dynamische Speicherung der Verbindungen in einer JSON-Datei. Dadurch können Verbindungsdaten auch nach einem Neustart der Anwendung wiederhergestellt werden.

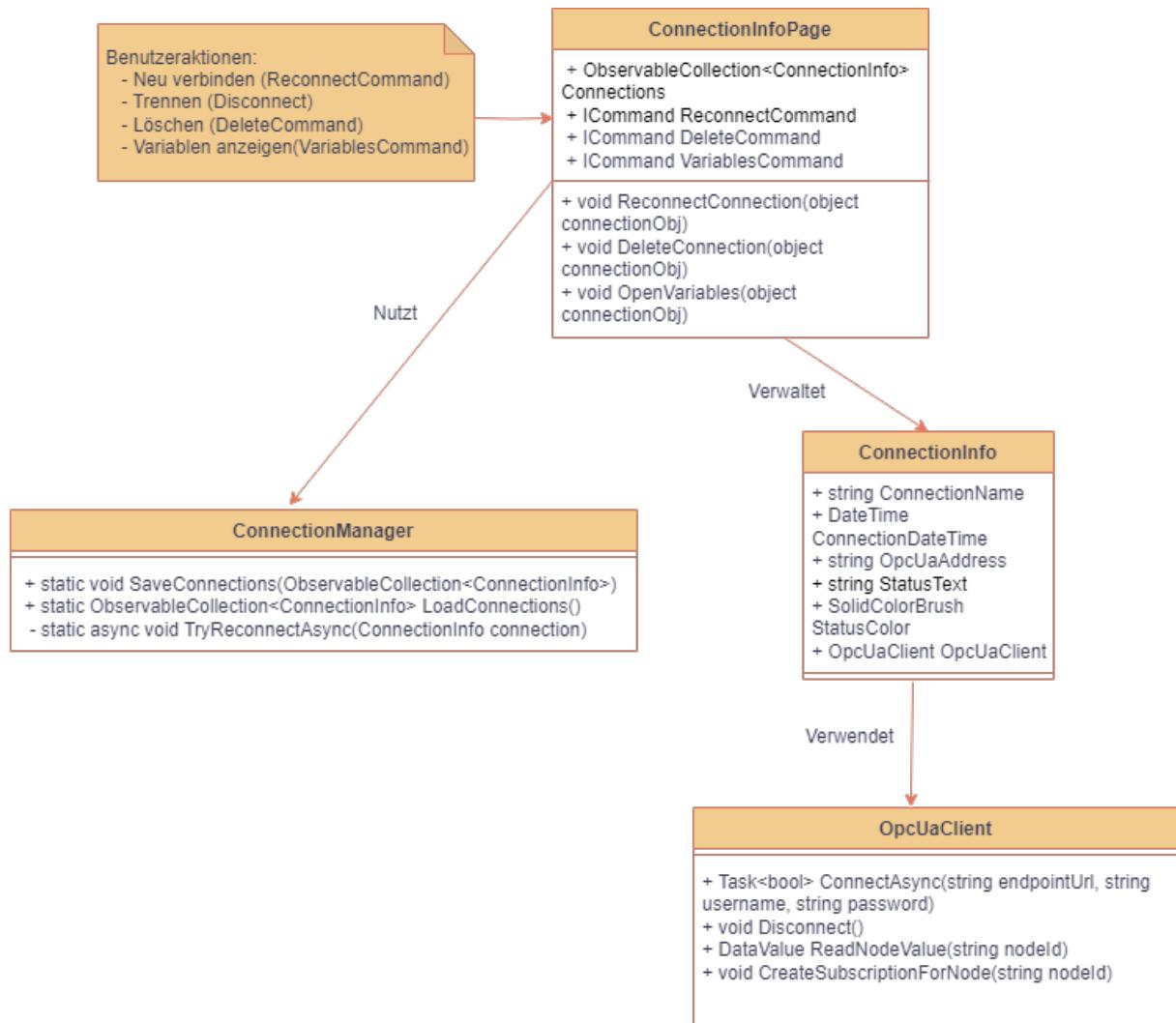


Abbildung 5: Klassendiagramm der OPC-UA-Verbindungsverwaltung

Das Klassendiagramm veranschaulicht die Struktur und Funktionsweise der Verbindungsverwaltung in der Anwendung. Die ConnectionInfoPage ist das zentrale Element der Benutzeroberfläche und dient zur Verwaltung der bestehenden OPC-UA-Verbindungen. Sie enthält eine Liste aller gespeicherten Verbindungen (Connections) und ermöglicht dem Benutzer, verschiedene Aktionen auszuführen. Dazu gehören das Neuverbinden einer bestehenden Verbindung, das Trennen einer aktiven Verbindung, das Löschen einer nicht mehr benötigten Verbindung sowie das Anzeigen von Variablen, die über die OPC-UA-Schnittstelle ausgelesen werden können. Diese vier Funktionen sind als ICommand-Befehle in der Klasse hinterlegt und direkt mit der Benutzeroberfläche verknüpft.

Jede Verbindung wird durch die Klasse ConnectionInfo repräsentiert, die wichtige Informationen wie den Namen, die Adresse, den aktuellen Status und den zugehörigen OpcUaClient speichert. Der Status einer Verbindung wird über StatusText und StatusColor visuell dargestellt, sodass der Benutzer auf einen Blick erkennen kann, ob eine Verbindung aktiv oder getrennt ist.

Die Verwaltung und Persistenz der Verbindungen übernimmt die Klasse ConnectionManager. Diese ist für das Speichern und Laden der Verbindungsdaten in einer JSON-Datei zuständig, sodass beim erneuten Start der Anwendung alle zuvor gespeicherten Verbindungen wiederhergestellt werden können. Zusätzlich enthält der ConnectionManager die Methode TryReconnectAsync(), die versucht, nach dem Laden der Daten automatisch eine Verbindung zu den gespeicherten OPC-UA-Servern herzustellen.

Die eigentliche Kommunikation mit dem OPC-UA-Server erfolgt über die OpcUaClient-Klasse. Sie enthält Methoden zur Verbindungsherstellung (ConnectAsync()), zur Trennung (Disconnect()) sowie zur Interaktion mit Variablen (ReadNodeValue() und CreateSubscriptionForNode()). Jede ConnectionInfo-Instanz nutzt einen OpcUaClient, um mit dem OPC-UA-Server zu interagieren.

4.3 Datenbankverbindung

In diesem Abschnitt wird die Implementierung der Datenbankverbindung beschrieben. Die Anwendung ermöglicht eine Verbindung zu PostgreSQL, MySQL und Microsoft SQL Server (MSSQL). Dabei werden die notwendigen Schritte zur Einrichtung, Authentifizierung und Verwaltung der Verbindungsdaten erklärt. Zusätzlich wird darauf eingegangen, wie die Verbindung gespeichert und abgesichert wird, um eine zuverlässige Kommunikation mit der Datenbank zu gewährleisten.

4.3.1 Auswahl der Datenbanktechnologien

Die Entscheidung fiel auf die Unterstützung von PostgreSQL, MySQL und Microsoft SQL Server (MSSQL), da diese relationalen Datenbanksysteme weit verbreitet sind und unterschiedliche Anforderungen abdecken.

PostgreSQL ist ein objektrelationales Datenbankmanagementsystem, das für seine Erweiterbarkeit und Einhaltung von SQL-Standards bekannt ist. Es bietet erweiterte Funktionen wie komplexe Abfragen, Fremdschlüssel, Trigger und Views. Zudem unterstützt PostgreSQL eine Vielzahl von Datentypen, einschließlich benutzerdefinierter Typen, Arrays und JSON. Diese Flexibilität macht es geeignet für Anwendungen, die komplexe Datenstrukturen und erweiterte Funktionalitäten erfordern.

MySQL ist ein relationales Datenbankmanagementsystem, das für seine Geschwindigkeit und Zuverlässigkeit geschätzt wird. Es ist besonders beliebt für Webanwendungen und wird häufig in Kombination mit PHP verwendet. MySQL bietet eine einfache Einrichtung und ist für seine hohe Leistung bei Leseoperationen bekannt. Es unterstützt grundlegende SQL-Funktionen und ist für Anwendungen geeignet, die schnelle Abfragen und einfache Transaktionen benötigen.

Microsoft SQL Server (MSSQL) ist ein relationales Datenbankmanagementsystem von Microsoft, das tief in die Windows-Umgebung integriert ist. Es bietet umfassende Verwaltungswerzeuge und unterstützt erweiterte Funktionen wie Stored Procedures, Trigger und Transaktionen. MSSQL ist bekannt für seine Skalierbarkeit und Sicherheit und eignet sich für Unternehmensanwendungen, die eine enge Integration mit anderen Microsoft-Produkten erfordern.

Die Unterstützung dieser drei Datenbanktypen ermöglicht es der Anwendung, flexibel auf unterschiedliche Benutzeranforderungen und bestehende Infrastrukturen einzugehen.

4.3.2 Architektur der Datenbankverbindung

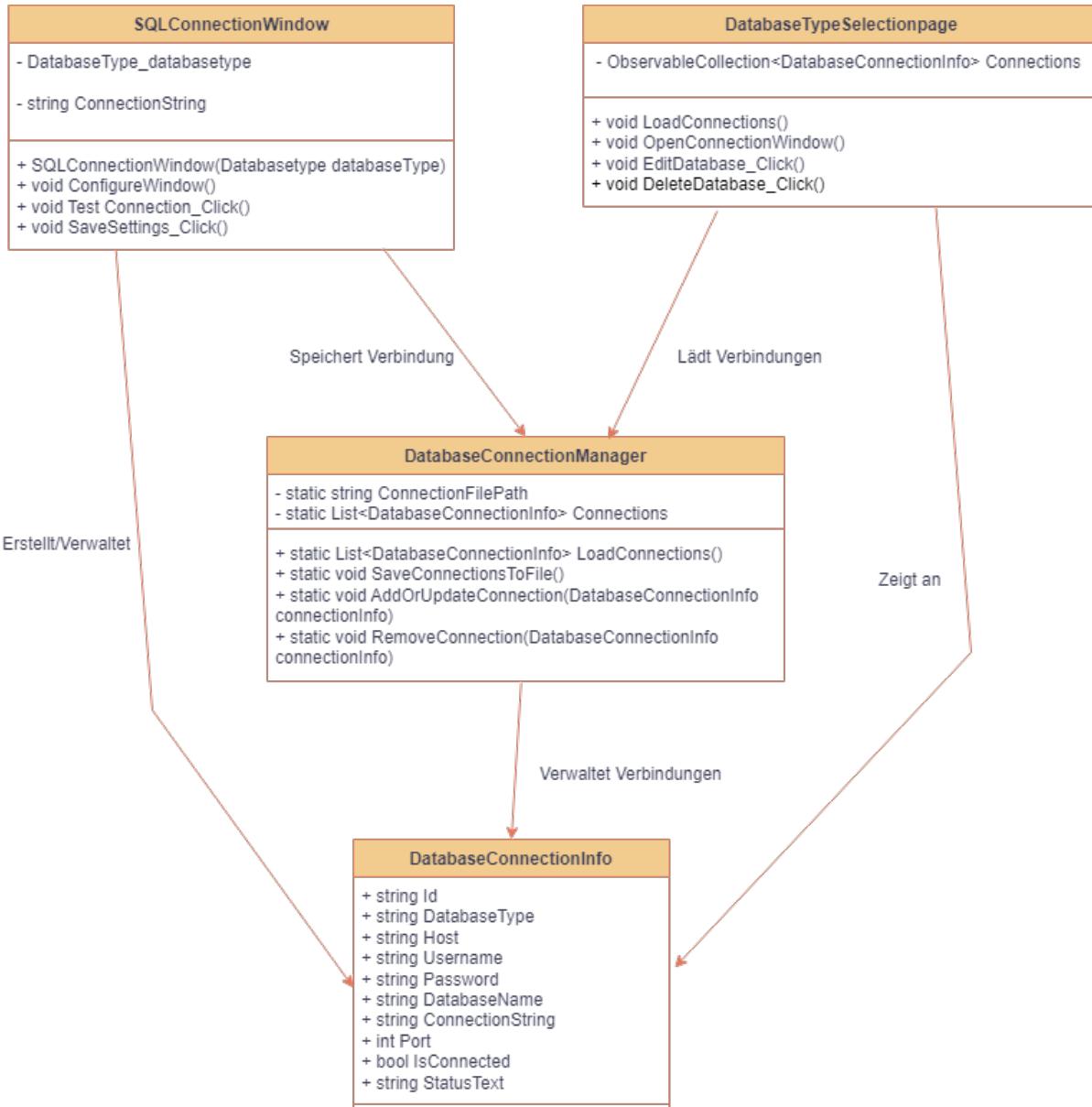


Abbildung 6: Klassendiagramm der Datenbankverbindung

Das Klassendiagramm zeigt die Architektur der Datenbankverbindung in der Anwendung und beschreibt die wichtigsten Klassen, ihre Attribute und Methoden sowie die Beziehungen zwischen ihnen.

Die Klasse **SQLConnectionWindow** stellt die Benutzeroberfläche bereit, in der der Nutzer die Verbindungsparameter für eine Datenbank eingeben kann. Hier werden der Host, der Port, der Benutzername und das Passwort für die Verbindung erfasst. Zudem bietet die Klasse Funktionen zur Konfiguration der Verbindung für verschiedene Datenbanktypen wie MSSQL, MySQL und PostgreSQL. Bevor eine Verbindung gespeichert wird, kann sie getestet werden, um sicherzustellen, dass die eingegebenen Daten korrekt sind und eine Verbindung erfolgreich hergestellt werden kann. Ist der Test erfolgreich, wird die Verbindung an den **DatabaseConnectionManager** übergeben.

Der DatabaseConnectionManager verwaltet alle gespeicherten Datenbankverbindungen und stellt sicher, dass diese bei Bedarf geladen oder aktualisiert werden können. Er speichert und lädt die Verbindungen aus einer JSON-Datei (connections.json), um sie auch nach dem Neustart der Anwendung verfügbar zu halten. Alle gespeicherten Verbindungen werden in einer Liste (List<DatabaseConnectionInfo>) verwaltet. Darüber hinaus bietet der DatabaseConnectionManager Methoden zum Hinzufügen, Aktualisieren und Löschen von Verbindungen. Beim Start der Anwendung sorgt er dafür, dass bereits gespeicherte Verbindungen automatisch wiederhergestellt werden.

Die Klasse DatabaseConnectionInfo dient als Datenmodell für eine gespeicherte Verbindung. Sie speichert alle wichtigen Informationen zur Datenbank, darunter Host, Port, Benutzername, Passwort und den Datenbanknamen. Zusätzlich enthält sie den ConnectionString, der für die Verbindung mit der jeweiligen Datenbank genutzt wird. Außerdem gibt es die Attribute IsConnected und StatusText, die anzeigen, ob eine Verbindung aktiv ist oder nicht. Jede gespeicherte Verbindung wird als Instanz dieser Klasse repräsentiert und vom DatabaseConnectionManager verwaltet.

Die DatabaseTypeSelectionPage stellt eine Benutzeroberfläche bereit, in der alle gespeicherten Datenbankverbindungen angezeigt werden. Sie lädt die Verbindungen aus dem DatabaseConnectionManager und listet sie in einer übersichtlichen Form auf. Dabei wird der aktuelle Verbindungsstatus (verbunden oder getrennt) angezeigt. Der Nutzer hat über diese Oberfläche die Möglichkeit, bestehende Verbindungen zu bearbeiten oder zu löschen.

4.3.3 Verbindaufbau und Speicherung der Verbindung

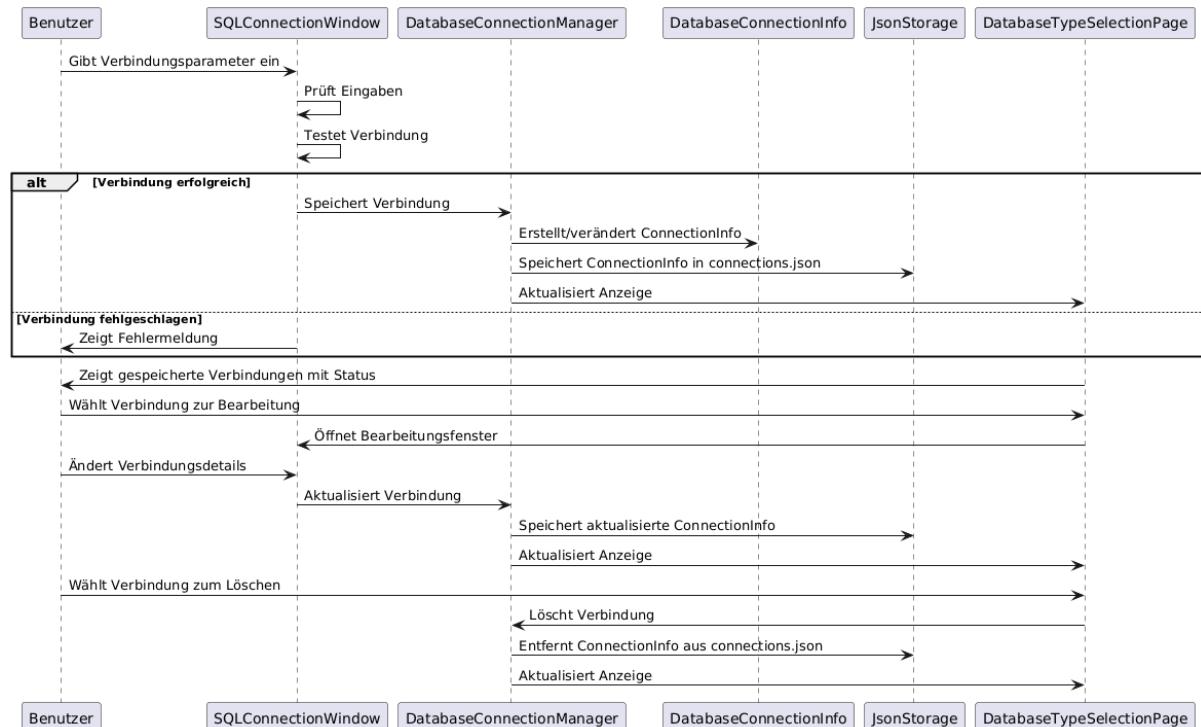


Abbildung 7: Sequenzdiagramm der Verbindungsverwaltung der Datenbankverbindungen

Das Sequenzdiagramm beschreibt den gesamten Prozess der Verbindungsherstellung, Speicherung, Bearbeitung und Löschung von Datenbankverbindungen in der Anwendung. Es zeigt, wie die einzelnen Komponenten zusammenarbeiten, um eine stabile und effiziente Verwaltung der Verbindungen zu gewährleisten.

Der Ablauf beginnt, wenn der Benutzer in der SqlConnectionWindow-Benutzeroberfläche die Verbindungsparameter eingibt. Dazu gehören der Host, Port, Benutzername und das Passwort der gewünschten Datenbank. Die Anwendung überprüft anschließend die Eingaben auf Vollständigkeit und testet die Verbindung zur ausgewählten Datenbank.

Falls die Verbindung erfolgreich hergestellt wird, wird sie an den DatabaseConnectionManager übergeben. Dieser erstellt oder aktualisiert ein DatabaseConnectionInfo-Objekt, das alle relevanten Verbindungsdetails enthält. Anschließend wird die Verbindung dauerhaft in der JSON-Datei (connections.json) gespeichert, sodass sie auch nach dem Neustart der Anwendung erhalten bleibt. Danach wird die DatabaseTypeSelectionPage aktualisiert, um die gespeicherte Verbindung mit ihrem aktuellen Status anzuzeigen.

Falls die Verbindung fehlschlägt, wird dem Benutzer eine entsprechende Fehlermeldung angezeigt, und es erfolgt keine Speicherung. In diesem Fall kann der Benutzer seine Eingaben korrigieren und die Verbindung erneut testen.

Neben der Speicherung ermöglicht die Anwendung auch die Bearbeitung und Löschung bestehender Verbindungen. Wenn der Benutzer eine gespeicherte Verbindung bearbeiten möchte, wählt er diese in der DatabaseTypeSelectionPage aus. Dadurch wird SqlConnectionWindow erneut geöffnet, wo die Verbindungsdetails geändert werden können. Nach der Bestätigung werden die aktualisierten Informationen in DatabaseConnectionManager übernommen und erneut in die JSON-Datei geschrieben.

Beim Löschen einer Verbindung wählt der Benutzer eine gespeicherte Verbindung in der DatabaseTypeSelectionPage aus und bestätigt das Löschen. Daraufhin entfernt DatabaseConnectionManager die Verbindung aus der Liste und löscht sie ebenfalls aus der JSON-Datei. Anschließend wird die Benutzeroberfläche aktualisiert, sodass die gelöschte Verbindung nicht mehr angezeigt wird.

Ein besonderer Vorteil dieses Systems besteht darin, dass beim Neustart der Anwendung alle gespeicherten Verbindungen aus DatabaseConnectionManager automatisch geladen und wiederhergestellt werden. Dadurch werden bestehende Datenbankverbindungen ohne weiteres Zutun des Benutzers erneut aufgebaut, sodass er nicht jedes Mal manuell eine Verbindung herstellen muss. Dies sorgt für eine komfortable und effiziente Nutzung der Anwendung.

4.4 Konfiguration der Excel-Trigger

In diesem Abschnitt wird beschrieben, wie die Excel-Trigger konfiguriert werden, um automatisch Werte aus der SPS zu erfassen und in eine Tabelle zu schreiben. Zunächst werden die Trigger eingerichtet, die festlegen, welche Variablen erfasst werden sollen. Anschließend sorgt das System dafür, dass diese Werte regelmäßig oder ereignisbasiert in das Excel-Dokument eingetragen werden.

4.4.1 Parameter der Excel-Trigger

In der Anwendung gibt es drei verschiedene Arten von Excel-Triggern, die genutzt werden, um Werte aus einer SPS in eine Excel-Datei zu schreiben. Jeder dieser Trigger dient einem bestimmten Zweck und ermöglicht eine flexible Datenaufzeichnung.

Die erste Art ist der SPS schreibt in eine einzelne Excel-Zelle-Trigger. Hier wird der Wert einer einzelnen SPS-Variablen in eine bestimmte Zelle innerhalb einer Excel-Datei geschrieben. Dieser Trigger eignet sich besonders gut für die Speicherung einzelner Werte, wie zum Beispiel eine aktuelle Temperatur oder den Status einer Maschine. Der Benutzer kann hierfür die gewünschte Variable sowie die exakte Zielzelle in der Excel-Datei festlegen.

Der zweite Typ ist der SPS schreibt in einen Excel-Datensatz-Trigger. Im Gegensatz zur ersten Variante werden hier mehrere Variablen in eine neue Zeile der Excel-Datei geschrieben. Dadurch entsteht eine fortlaufende Datenhistorie, die sich besonders für Prozessüberwachungen oder Analysen eignet. Die Werte werden immer in die nächste freie Zeile eingetragen, sodass keine alten Daten überschrieben werden. Neben der Auswahl der Variablen kann der Benutzer das gewünschte Datenformat definieren, beispielsweise ob die Werte als Dezimalzahl oder als String gespeichert werden sollen.

Der dritte Typ ist der SPS schreibt in einen Excel-Datenblock-Trigger. Bei dieser Methode werden mehrere Werte innerhalb eines bestimmten Zellbereichs in der Excel-Datei gespeichert. Die Speicherung beginnt an einer festgelegten Startadresse und verteilt sich über mehrere Zellen, wodurch sich diese Methode besonders für strukturierte Messdatenerfassungen eignet. Zusätzlich kann die Anzahl der Datenpunkte pro Block festgelegt werden, um eine genaue Kontrolle über die gespeicherten Werte zu haben.

Jeder dieser Trigger kann auf zwei verschiedene Arten ausgelöst werden: zeitbasiert oder ereignisbasiert.

Ein zeitbasierter Trigger speichert die SPS-Daten in regelmäßigen Abständen in die Excel-Tabelle. Der Benutzer kann dabei ein festes Intervall, beispielsweise alle 5 Sekunden oder jede Minute, einstellen. Zusätzlich kann ein Start- und Enddatum definiert werden, sodass der Trigger nur innerhalb eines bestimmten Zeitraums aktiv bleibt. Diese Methode eignet sich besonders für kontinuierliche Messungen oder zur langfristigen Überwachung von Produktionsprozessen.

Ein ereignisbasierter Trigger hingegen wird nur dann ausgelöst, wenn eine bestimmte Bedingung erfüllt ist. Hier gibt es vier unterschiedliche Ereignistypen, die für die Auslösung genutzt werden können.

Die erste Bedingung ist eine Datenänderung. Dabei wird der Trigger immer dann ausgelöst, wenn sich der Wert der überwachten Variable verändert. Diese Methode ist nützlich, wenn man nur dann einen Eintrag in Excel speichern möchte, wenn tatsächlich eine Veränderung der Daten auftritt, anstatt sie in festen Zeitintervallen zu protokollieren.

Die zweite Bedingung ist das Überschreiten eines Grenzwertes. Hier wird der Trigger aktiviert, wenn eine Variable einen bestimmten Wert über- oder unterschreitet. Um diese Bedingung zu definieren, muss ein Vergleichsoperator wie „größer als“ oder „kleiner als“ ausgewählt werden, zusammen mit dem entsprechenden Grenzwert. Ein Beispiel wäre die Speicherung eines Temperaturwertes nur dann, wenn dieser über 80°C steigt.

Der dritte Ereignistyp ist der Statuswechsel. Der Trigger wird in diesem Fall ausgelöst, wenn eine Variable von einem Zustand in einen anderen wechselt, beispielsweise wenn eine Maschine von „AUS“ auf „EIN“ schaltet oder umgekehrt. Der Benutzer kann festlegen, ob der Trigger beim Wechsel von „EIN zu AUS“ oder von „AUS zu EIN“ aktiviert werden soll.

Die vierte Möglichkeit ist die Bit-Überprüfung. Hierbei wird ein bestimmtes Bit innerhalb einer SPS-Variable überwacht, und der Trigger reagiert, wenn sich dessen Zustand ändert. Der Benutzer kann dabei eine Bit-Position zwischen 0 und 7 auswählen und definieren, unter welchen Bedingungen die Speicherung erfolgen soll. Es gibt drei Optionen: Der Trigger kann bei jeder Änderung des Bits ausgelöst werden, oder nur dann, wenn das Bit von 0 auf 1 oder von 1 auf 0 wechselt.

Für ereignisbasierte Trigger in den Datensatz- und Datenblock-Triggern gilt zusätzlich eine besondere Regel. Während der Benutzer mehrere SPS-Variablen auswählt, die in die Excel-Datei geschrieben werden sollen, muss für die eigentliche Bedingung eine spezifische Variable als Trigger-Variable festgelegt werden. Das bedeutet, dass alle Werte erst dann in Excel gespeichert werden, wenn die Bedingung für diese ausgewählte Variable erfüllt ist. Ein Beispiel hierfür wäre die Aufzeichnung von Temperatur- und Druckwerten, aber nur dann, wenn die Temperatur über 100°C steigt. In diesem Fall würde die Temperatur als Trigger-Variable fungieren, während der Druckwert einfach mitgespeichert wird.

4.4.2 Architektur und Implementierung der Excel-Trigger

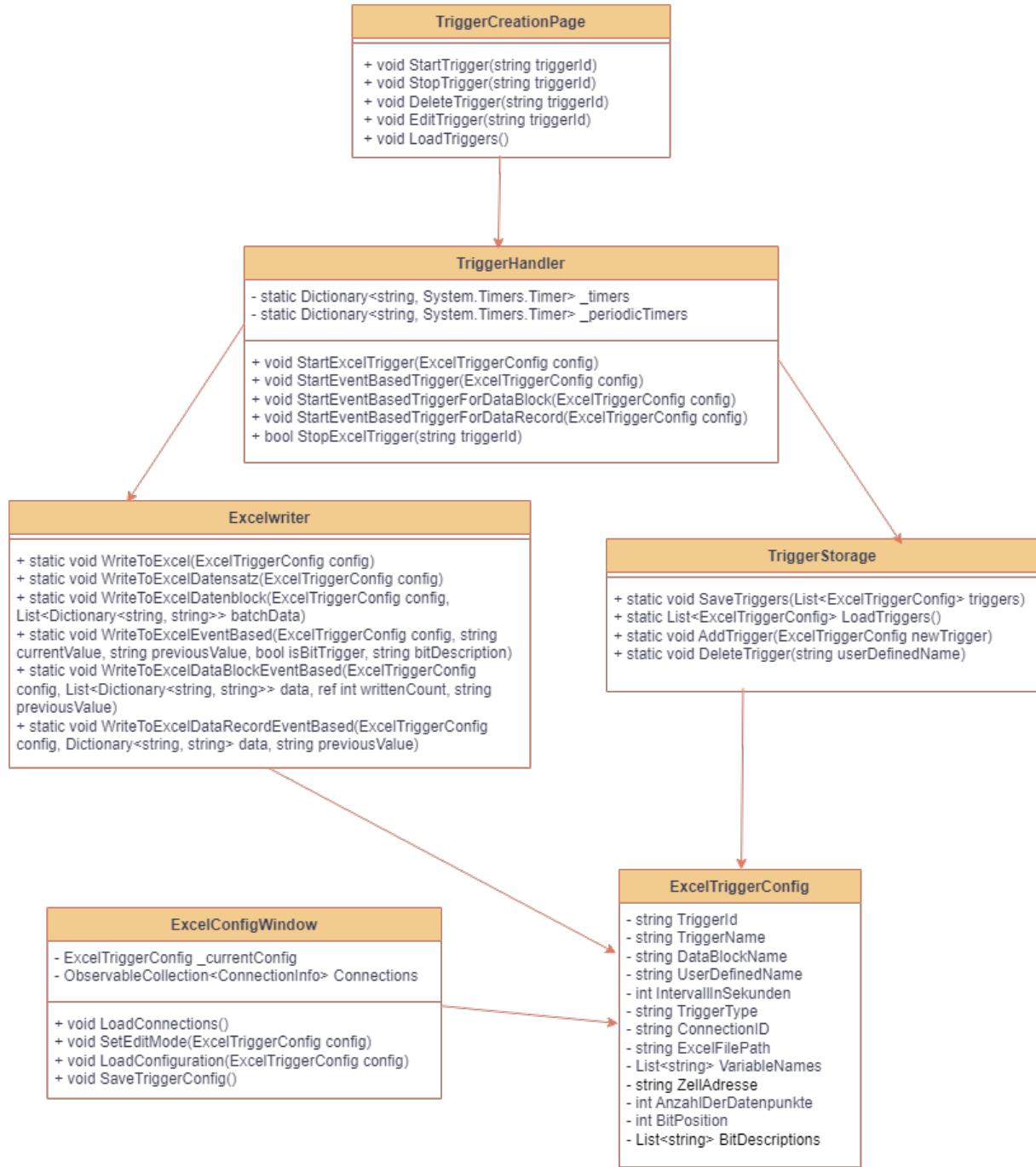


Abbildung 8: Klassendiagramm der Architektur der Excel-Trigger

Die ExcelTriggerConfig-Klasse bildet die zentrale Datenstruktur, die alle Informationen zu einem Trigger enthält. Hier werden Trigger-ID, Name, Datenquelle, Zielzelle oder Datenblock, das Intervall für zeitbasierte Trigger und die Bedingungen für ereignisbasierte Trigger gespeichert. Zusätzlich gibt es spezielle Attribute für Bit-Überwachung, bei denen eine Bit-Position und deren Beschreibung festgelegt werden können.

Die ExcelConfigWindow-Klasse stellt die Benutzeroberfläche für die Trigger-Konfiguration bereit. Hier kann ein Benutzer neue Trigger erstellen, bestehende Trigger bearbeiten und die Konfiguration speichern.

Die TriggerCreationPage verwaltet die Liste aller gespeicherten Trigger. Über diese Klasse kann der Benutzer Trigger starten, stoppen, bearbeiten oder löschen. Sie lädt gespeicherte Trigger aus der TriggerStorage-Klasse, aber startet sie nicht automatisch, sondern stellt nur die gespeicherte Konfiguration bereit.

Die TriggerHandler-Klasse ist für das Starten und Stoppen der Trigger verantwortlich. Sie verwaltet die Timer für zeitbasierte Trigger und die Überwachung für ereignisbasierte Trigger. Hier werden auch spezielle Methoden bereitgestellt, um Trigger für einzelne Werte, Datensätze oder ganze Datenblöcke zu starten.

Der ExcelWriter ist für das Schreiben von Daten in die Excel-Dateien zuständig. Er stellt Methoden bereit, um Werte in einzelne Zellen, neue Datensätze (Zeilen) oder ganze Datenblöcke zu schreiben. Zusätzlich gibt es spezielle Methoden für ereignisbasierte Trigger, bei denen nur dann Werte in die Excel-Datei geschrieben werden, wenn eine Bedingung erfüllt ist.

Die TriggerStorage-Klasse speichert alle Trigger-Konfigurationen in einer JSON-Datei. Sie ermöglicht das Hinzufügen, Aktualisieren, Laden und Löschen von Triggern. Allerdings startet sie die Trigger nach einem Neustart nicht automatisch – sie stellt lediglich die gespeicherten Konfigurationen zur Verfügung. Das bedeutet, dass der Benutzer nach einem Neustart der Anwendung seine Trigger manuell in der Oberfläche aktivieren muss, falls sie wieder ausgeführt werden sollen.

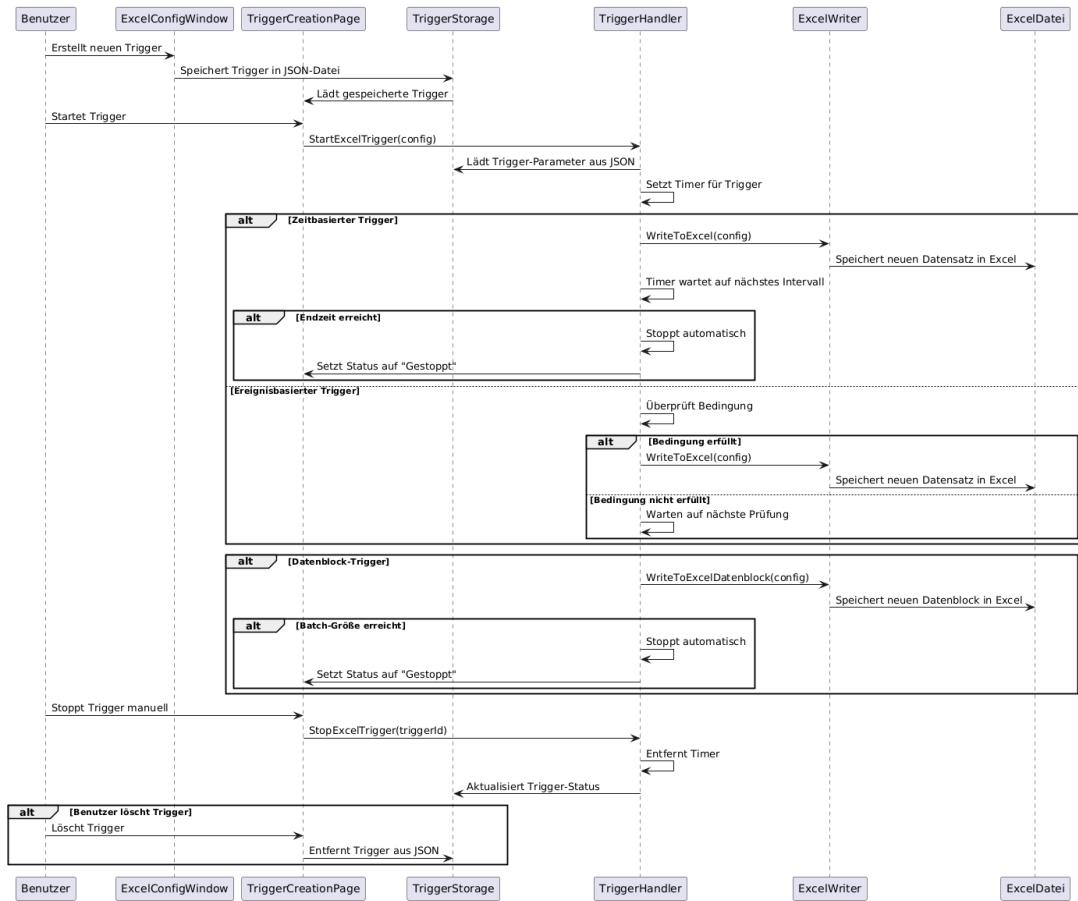


Abbildung 9: Sequenzdiagramm für die Verarbeitung von Excel-Triggern

Das Sequenzdiagramm zeigt den kompletten Ablauf eines Excel-Triggers, von der Erstellung über das Starten und Schreiben bis hin zum Stoppen oder Löschen.

Zunächst erstellt der Benutzer einen neuen Trigger über das ExcelConfigWindow. Nachdem alle Parameter eingegeben wurden, wird die Konfiguration in der TriggerStorage-Klasse gespeichert, damit sie später wieder geladen werden kann. Beim Start der Anwendung lädt die TriggerCreationPage alle gespeicherten Trigger.

Beim Start eines Triggers ruft die TriggerCreationPage den TriggerHandler auf. Dieser lädt die gespeicherten Parameter und entscheidet, ob es sich um einen zeitbasierten oder ereignisbasierten Trigger handelt.

Wenn der Trigger zeitbasiert ist, startet der TriggerHandler einen Timer. In regelmäßigen Intervallen werden dann Messwerte über den ExcelWriter in die Excel-Datei geschrieben. Der Trigger läuft weiter, bis die konfigurierte Endzeit erreicht ist. Sobald dieser Zeitpunkt erreicht ist, stoppt der Trigger automatisch und der Status wird in der Benutzeroberfläche als „Gestoppt“ angezeigt.

Bei einem ereignisbasierten Trigger überprüft der TriggerHandler kontinuierlich die festgelegte Bedingung. Falls zum Beispiel ein Grenzwert überschritten oder ein Bit-Status geändert wurde, wird der Wert durch den ExcelWriter in die Excel-Datei geschrieben. Falls die Bedingung nicht erfüllt ist, wartet der Trigger auf die nächste Überprüfung und wiederholt diesen Prozess, bis er manuell gestoppt wird.

Ein spezieller Fall ist der Datenblock-Trigger. Hier werden nicht einzelne Werte geschrieben, sondern mehrere Messpunkte gesammelt und als Batch in die Excel-Datei gespeichert. Sobald die konfigurierte Batch-Größe erreicht ist, wird der Trigger automatisch gestoppt.

4.5 Konfiguration der Datenbank-trigger

Die Datenbank-Trigger funktionieren ähnlich wie die Excel-Trigger: Sie erfassen und speichern Werte von SPS-Variablen basierend auf bestimmten Bedingungen oder Zeitintervallen. Allerdings gibt es einige wesentliche Vorteile, wenn Daten in einer Datenbank statt in einer Excel-Datei gespeichert werden.

Datenbanken sind leistungsfähiger bei der Verarbeitung großer Datenmengen und ermöglichen eine effiziente Abfrage und Analyse der gespeicherten Werte. Im Vergleich zu Excel-Tabellen bieten sie zudem mehr Struktur, bessere Mehrbenutzerfähigkeit und eine höhere Datensicherheit. Während Excel eher für kleinere Visualisierungen geeignet ist, sind relationale Datenbanken ideal, um historische SPS-Daten langfristig zu speichern und für weiterführende Analysen oder Automatisierungen zu nutzen.

4.5.1 Parameter der Datenbank-Trigger

Die Parameter der Datenbank-Trigger sind weitgehend mit denen der Excel-Trigger vergleichbar (siehe 4.4.1), mit nur einigen wesentlichen Unterschieden. Der wichtigste Unterschied besteht darin, dass der Benutzer zunächst eine bereits eingerichtete Datenbankverbindung auswählen muss. Diese Verbindung wurde zuvor in der Anwendung konfiguriert und kann entweder eine PostgreSQL-, MySQL- oder MSSQL-Datenbank sein.

Nachdem die Verbindung festgelegt wurde, wählt der Benutzer den entsprechenden Datenblock, in dem sich die gewünschten Variablen befinden. Anschließend kann er eine oder mehrere Variablen bestimmen, die regelmäßig oder basierend auf bestimmten Ereignissen in die Datenbank geschrieben werden sollen.

Ein weiterer wichtiger Schritt in der Konfiguration ist die Auswahl der Zieltabelle für die Speicherung der Variablenwerte. Hier hat der Benutzer zwei Möglichkeiten: Entweder kann er eine neue Tabelle direkt in der Anwendung erstellen, wobei er die Spaltennamen und die passenden Datentypen für jede Variable definiert, oder er kann eine bereits existierende Tabelle aus der Datenbank auswählen. Falls eine bestehende Tabelle verwendet wird, muss sichergestellt werden, dass die Spaltennamen und Datentypen exakt mit den in SPS konfigurierten Variablen übereinstimmen.

Ein entscheidender Punkt hierbei ist, dass die drei unterstützten Datenbanktypen unterschiedliche Datentyp-Definitionen verwenden. Beispielsweise gibt es für Boolesche Werte in den verschiedenen Datenbanken unterschiedliche Typbezeichnungen:

- MSSQL: Bit
- MySQL: BOOLEAN
- PostgreSQL: BOOLEAN

Falls ein falscher Datentyp zugewiesen wird, kann dies zu Schreibfehlern oder Inkompatibilitäten in der Datenbank führen. Deshalb ist es wichtig, die Datentypen mit den spezifischen Anforderungen der jeweiligen Datenbank abzugleichen.

Optional kann der Benutzer eine Batchgröße festlegen. Diese gibt an, wie viele Werte gesammelt werden sollen.

Die zeitbasierten und ereignisbasierten Trigger funktionieren identisch zu den Excel-Triggern. Der Benutzer kann entweder ein festes Zeitintervall für die Speicherung der Werte definieren oder Bedingungen festlegen, die den Trigger nur bei bestimmten Ereignissen aktivieren. Die genauere Beschreibung dieser Trigger-Typen ist in 4.4.1 zu finden.

4.5.2 Architektur der Datenbank-Trigger

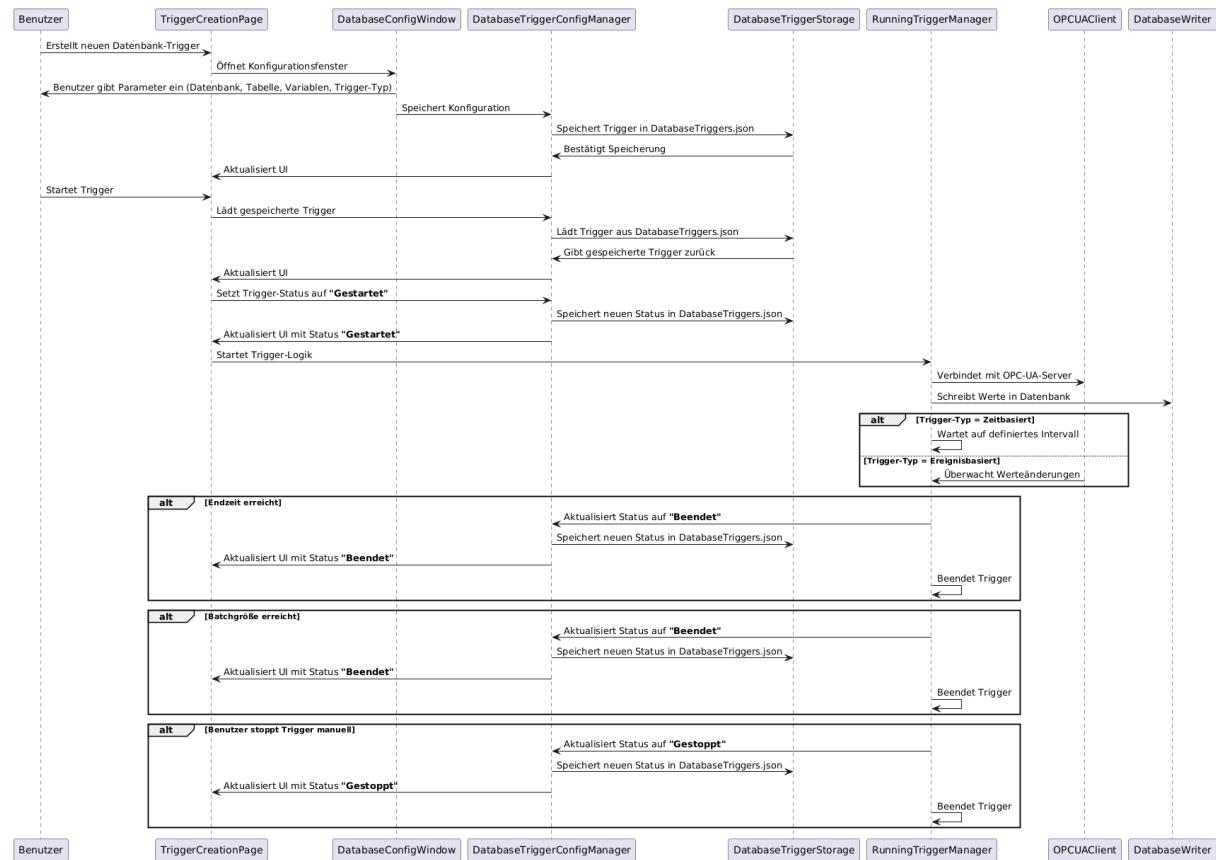


Abbildung 10: Sequenzdiagramm für die Verarbeitung von Datenbank-Triggern

Der Benutzer erstellt zunächst einen neuen Datenbank-Trigger in der TriggerCreationPage. Dabei öffnet sich das DatabaseConfigWindow, in dem der Benutzer alle erforderlichen Parameter eingibt. Dazu gehören unter anderem die Auswahl der Datenbank, die Tabelle, die zu speichernden Variablen und der gewünschte Trigger-Typ (zeit- oder ereignisbasiert). Sobald der Benutzer die Konfiguration speichert, wird diese an den DatabaseTriggerConfigManager übergeben, der sie wiederum in DatabaseTriggerStorage speichert, indem er die DatabaseTriggers.json aktualisiert. Anschließend erhält die TriggerCreationPage die aktualisierten Daten und zeigt sie in der Benutzeroberfläche an.

Wenn der Benutzer den Trigger startet, lädt die TriggerCreationPage die gespeicherten Trigger erneut aus DatabaseTriggerStorage, sodass der zuletzt konfigurierte Trigger verfügbar ist. Der DatabaseTriggerConfigManager setzt den Trigger-Status auf Gestartet und speichert diese Information in DatabaseTriggers.json, damit der Status auch nach einem Neustart der Anwendung erhalten bleibt. Danach wird die Benutzeroberfläche entsprechend aktualisiert, um anzusehen, dass der Trigger aktiv ist.

Beim Start des Triggers wird die Logik in RunningTriggerManager aufgerufen. Diese Klasse übernimmt die Verarbeitung des Triggers und sorgt für die Verbindung zum OPC-UA-Server über den OPCUAClient.

Es gibt mehrere Bedingungen, unter denen ein Trigger automatisch beendet wird. Falls eine Endzeit erreicht ist oder eine definierte Batchgröße an gespeicherten Daten überschritten wird, wird der Status auf "Beendet" gesetzt. Der DatabaseTriggerConfigManager speichert diesen Status ebenfalls in DatabaseTriggers.json, und die TriggerCreationPage aktualisiert die Benutzeroberfläche, um den neuen Status anzuzeigen.

Alternativ kann der Benutzer den Trigger auch manuell stoppen. In diesem Fall wird der Status des Triggers auf "Gestoppt" gesetzt, ebenfalls in DatabaseTriggers.json gespeichert und die Benutzeroberfläche wird aktualisiert. Nach dem Beenden oder Stoppen des Triggers beendet RunningTriggerManager den laufenden Prozess.

4.5.3 Implementierung der Datenbank-Schreibprozesse

```
private void WriteToDatabaseInternal(IDbConnection connection, string tableName, Dictionary<string, object> data)
{
    connection.Open();
    EnsureTimestampColumnExists(connection, tableName);
    if (data.ContainsKey("BitBedeutung"))
    {
        EnsureBitDescriptionColumnExists(connection, tableName);
    }

    // Add timestamp
    string timestamp = DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss.ffff");
    data["Timestamp"] = timestamp;

    string columns = string.Join(", ", data.Keys.Select(k => $"`{k}`"));
    string parameters = string.Join(", ", data.Keys.Select(k => $"@{k}"));
    string sql = $"INSERT INTO '{tableName}' ({columns}) VALUES ({parameters});";

    using (var command = connection.CreateCommand())
    {
        command.CommandText = sql;
        foreach (var kvp in data)
        {
            var parameter = command.CreateParameter();
            parameter.ParameterName = $"@{kvp.Key}";
            parameter.Value = kvp.Value ?? DBNull.Value;
            command.Parameters.Add(parameter);
        }
        command.ExecuteNonQuery();
    }
}
```

Abbildung 11: Einfügen von Daten in MySQL.

```
1 Verweis
private void EnsureBitDescriptionColumnExists(IDbConnection connection, string tableName)
{
    string checkColumnSql = $"SHOW COLUMNS FROM '{tableName}' LIKE 'BitBedeutung';";
    using (var command = connection.CreateCommand())
    {
        command.CommandText = checkColumnSql;
        var result = command.ExecuteScalar();

        if (result == null)
        {
            string alterTableSql = $"ALTER TABLE '{tableName}' ADD COLUMN 'BitBedeutung' VARCHAR(255) NULL;";
            using (var alterCommand = connection.CreateCommand())
            {
                alterCommand.CommandText = alterTableSql;
                alterCommand.ExecuteNonQuery();
            }
            Console.WriteLine("[DEBUG] Spalte 'BitBedeutung' erfolgreich hinzugefügt.");
        }
    }
}
```

Abbildung 12: Überprüfung und Erstellung der BitBedeutung-Spalte in MySQL.

```

1 Verweis
private void EnsureTimestampColumnExists(IDbConnection connection, string tableName)
{
    string checkColumnSql = $"SHOW COLUMNS FROM '{tableName}' LIKE 'Timestamp'";
    using (var command = connection.CreateCommand())
    {
        command.CommandText = checkColumnSql;
        var result = command.ExecuteScalar();

        if (result == null)
        {
            string alterTableSql = $"ALTER TABLE '{tableName}' ADD COLUMN 'Timestamp' VARCHAR(23) NOT NULL FIRST";
            using (var alterCommand = connection.CreateCommand())
            {
                alterCommand.CommandText = alterTableSql;
                alterCommand.ExecuteNonQuery();
            }
        }
    }
}

```

Abbildung 13: Sicherstellen der Timestamp-Spalte in MySQL.

MySQL-Writer-Klasse

Die ersten drei Codeabschnitte zeigen die Implementierung für das Schreiben in eine MySQL-Datenbank. Der Prozess beginnt in der Methode WriteToDatabaseInternal, die eine bestehende Datenbankverbindung, den Namen der Zieltabelle und die Daten als Dictionary<string, object> erhält. Bevor die Daten in die Tabelle geschrieben werden, wird sichergestellt, dass die Spalten Timestamp und BitBedeutung vorhanden sind. Dazu werden zwei separate Methoden aufgerufen, die diese Spalten bei Bedarf anlegen.

Die Methode EnsureTimestampColumnExists überprüft, ob die Spalte Timestamp bereits existiert. Dafür nutzt sie die SQL-Abfrage SHOW COLUMNS FROM tableName LIKE 'Timestamp';. Falls die Spalte fehlt, wird sie mit dem SQL-Befehl ALTER TABLE tableName ADD COLUMN 'Timestamp' VARCHAR(23) NOT NULL FIRST; hinzugefügt. Besonders auffällig ist hier das Schlüsselwort FIRST, das in MySQL erlaubt, eine neue Spalte direkt an den Anfang der Tabelle zu setzen.

Ähnlich funktioniert EnsureBitDescriptionColumnExists, allerdings für die Spalte BitBedeutung, die zur Speicherung von zusätzlichen Informationen für Bit-Variablen dient. Hier wird wieder SHOW COLUMNS verwendet, um zu prüfen, ob die Spalte existiert. Falls nicht, wird sie mit ALTER TABLE tableName ADD COLUMN 'BitBedeutung' VARCHAR(255) NULL; hinzugefügt. Diese Spalte ist optional (NULL erlaubt), weil nicht alle Trigger sie benötigen.

Sobald sichergestellt ist, dass die benötigten Spalten vorhanden sind, werden die eigentlichen Daten geschrieben. In WriteToDatabaseInternal wird dazu eine dynamische SQL-Insert-Abfrage generiert. Die Spaltennamen und die Werte werden durch string.Join zu einer SQL-kompatiblen Liste zusammengefügt. Danach wird mit einer foreach-Schleife jeder Wert als SQL-Parameter hinzugefügt, um SQL-Injections zu verhindern. Am Ende wird die SQL-Abfrage ausgeführt, und die Daten werden in die MySQL-Tabelle eingefügt.

```

1 Verweis
private void WriteToDatabaseInternal(IDbConnection connection, string tableName, Dictionary<string, object> data)
{
    connection.Open();
    EnsureTimestampColumnExists(connection, tableName);
    if (data.ContainsKey("BitBedeutung"))
    {
        EnsureBitDescriptionColumnExists(connection, tableName);
    }
    string timestamp = DateTime.Now.ToString("dd.MM.yyyy HH:mm:ss.ffff");
    data["Timestamp"] = timestamp;

    string columns = string.Join(", ", data.Keys.Select(k => $"\"{k}\""));
    string parameters = string.Join(", ", data.Keys.Select(k => $"@{k}"));
    string sql = $@"INSERT INTO \"[{tableName}]\" ({columns}) VALUES ({parameters});";

    using (var command = connection.CreateCommand())
    {
        command.CommandText = sql;

        foreach (var kvp in data)
        {
            var parameter = command.CreateParameter();
            parameter.ParameterName = $"@{kvp.Key}";
            parameter.Value = kvp.Value ?? DBNull.Value;
            command.Parameters.Add(parameter);
        }

        command.ExecuteNonQuery();
    }
}

```

Abbildung 14: Einfügen von Daten in PostgreSQL.

```

1 Verweis
private void EnsureBitDescriptionColumnExists(IDbConnection connection, string tableName)
{
    try
    {
        string checkColumnSql = $"SELECT 1 FROM information_schema.columns WHERE table_name = '{tableName}' AND column_name = 'BitBedeutung'";
        using (var command = connection.CreateCommand())
        {
            command.CommandText = checkColumnSql;
            var result = command.ExecuteScalar();

            if (result == null)
            {
                string alterTableSql = $"ALTER TABLE \"{tableName}\" ADD COLUMN \"BitBedeutung\" VARCHAR(255)";
                using (var alterCommand = connection.CreateCommand())
                {
                    alterCommand.CommandText = alterTableSql;
                    alterCommand.ExecuteNonQuery();
                }
                Console.WriteLine("[DEBUG] Spalte 'BitBedeutung' erfolgreich hinzugefügt.");
            }
        }
    }
}

```

Abbildung 15: Überprüfung und Erstellung der BitBedeutung-Spalte in PostgreSQL.

```

private void EnsureTimestampColumnExists(IDbConnection connection, string tableName)
{
    try
    {
        string checkColumnSql = $"SELECT 1 FROM information_schema.columns WHERE table_name = '{tableName}' AND column_name = 'Timestamp'";
        using (var command = connection.CreateCommand())
        {
            command.CommandText = checkColumnSql;
            var result = command.ExecuteScalar();

            if (result == null)
            {
                Console.WriteLine("[DEBUG] Spalte 'Timestamp' existiert nicht. Füge sie hinzu...");
                string alterTableSql = $"ALTER TABLE \"{tableName}\" ADD COLUMN \"Timestamp\" VARCHAR(23) NOT NULL";
                using (var alterCommand = connection.CreateCommand())
                {
                    alterCommand.CommandText = alterTableSql;
                    alterCommand.ExecuteNonQuery();
                }
                Console.WriteLine("[DEBUG] Spalte 'Timestamp' erfolgreich hinzugefügt.");
            }
            else
            {
                Console.WriteLine("[DEBUG] Spalte 'Timestamp' existiert bereits.");
            }
        }
    }
}

```

Abbildung 16: Sicherstellen der Timestamp-Spalte in PostgreSQL.

PostgreSQL-Writer-Klasse

Die letzten drei Codeabschnitte zeigen die gleiche Funktionalität, jedoch für PostgreSQL. Obwohl beide Datenbanken relationale SQL-Datenbanken sind, gibt es einige Unterschiede in der SQL-Syntax, die berücksichtigt werden müssen.

In WriteToDatabaseInternal bleibt der Ablauf weitgehend identisch. Die Methode stellt sicher, dass die Spalten Timestamp und BitBedeutung vorhanden sind, bevor die Daten geschrieben werden. Allerdings wird in PostgreSQL die Spaltenprüfung nicht mit SHOW COLUMNS durchgeführt, sondern mit einer SQL-Abfrage auf information schema.columns. Hierbei wird geprüft, ob eine Spalte mit dem Namen Timestamp oder BitBedeutung in der Tabelle existiert. Falls nicht, wird sie mit ALTER TABLE tableName ADD COLUMN "Timestamp"VARCHAR(23) NOT NULL; bzw. ALTER TABLE tableName ADD COLUMN "BitBedeutung"VARCHAR(255); hinzugefügt.

Ein wichtiger Unterschied zu MySQL ist, dass PostgreSQL das FIRST-Keyword nicht unterstützt, weshalb eine neue Spalte nicht direkt an den Anfang der Tabelle gesetzt werden kann. Das bedeutet, dass Timestamp immer ans Ende der Tabelle angefügt wird, was für einige Anwendungen eine Einschränkung sein kann.

Auch beim Schreiben der Daten bleibt die Logik weitgehend gleich. Die SQL-Abfrage wird dynamisch erstellt, und die Werte werden als SQL-Parameter hinzugefügt. Da PostgreSQL jedoch etwas strikter mit der Behandlung von Zeichenfolgen ist, müssen manche Tabellennamen und Spaltennamen in doppelte Anführungszeichen gesetzt werden, wenn sie Sonderzeichen oder Großbuchstaben enthalten.

Unterschiede zwischen MySQL- und PostgreSQL-Implementierung

Der Hauptunterschied zwischen den beiden Implementierungen liegt in der Art und Weise, wie geprüft wird, ob eine bestimmte Spalte bereits in der Tabelle existiert. In MySQL wird hierfür der Befehl SHOW COLUMNS FROM ... LIKE ... verwendet, während PostgreSQL stattdessen eine Abfrage auf information schema.columns nutzt, um zu prüfen, ob die Spalte vorhanden ist.

Ein weiterer wichtiger Unterschied zeigt sich bei der Reihenfolge von Spalten in der Tabelle. MySQL erlaubt es, mit ALTER TABLE ... ADD COLUMN ... FIRST; eine neue Spalte direkt an den Anfang einer Tabelle zu setzen. In PostgreSQL gibt es diese Option jedoch nicht – neue Spalten werden immer am Ende der Tabelle angehängt, was in manchen Anwendungen eine Einschränkung sein kann.

Trotz dieser technischen Unterschiede funktionieren beide Implementierungen grundsätzlich auf die gleiche Weise. Zunächst wird geprüft, ob die notwendigen Spalten Timestamp und BitBedeutung bereits existieren, und falls nicht, werden sie automatisch angelegt.

4.6 Datenvizualisierung

Die Datenvizualisierung spielt eine zentrale Rolle in dieser Anwendung, da sie den Benutzer dabei unterstützt, gespeicherte Daten effizient zu interpretieren. Um die gespeicherten Werte übersichtlich darzustellen, bieten wir zwei verschiedene Visualisierungsarten an. Erstens hat der Benutzer die Möglichkeit, die vollständigen gespeicherten Tabellen direkt aus der Datenbank einzusehen. Dies ermöglicht eine schnelle und detaillierte Überprüfung der gespeicherten Werte in ihrer Rohform.

Zweitens kann der Benutzer die gespeicherten Daten in Form eines Diagramms visualisieren. Dafür muss er zunächst bestimmte Parameter für das Diagramm konfigurieren, wie zum Beispiel die Auswahl der relevanten Spalten und den gewünschten Zeitraum.

4.6.1 Tabellenbasierte Visualisierung

In vielen Anwendungen, insbesondere wenn mit großen Mengen an gespeicherten Daten gearbeitet wird, ist die Tabellenvisualisierung eine der wichtigsten Methoden zur übersichtlichen Darstellung. Normalerweise könnten Benutzer externe Tools wie PgAdmin 4, MySQL Workbench oder SQL Server Management Studio verwenden, um gespeicherte Datenbanktabellen anzuzeigen. Allerdings sind diese Tools für viele Benutzer ohne technischen Hintergrund oft zu komplex. Deshalb wurde diese Funktion in die Anwendung integriert, sodass auch technisch weniger versierte Anwender die gespeicherten Daten direkt in der Anwendung einsehen können, ohne sich mit externen Datenbankverwaltungstools auseinandersetzen zu müssen.

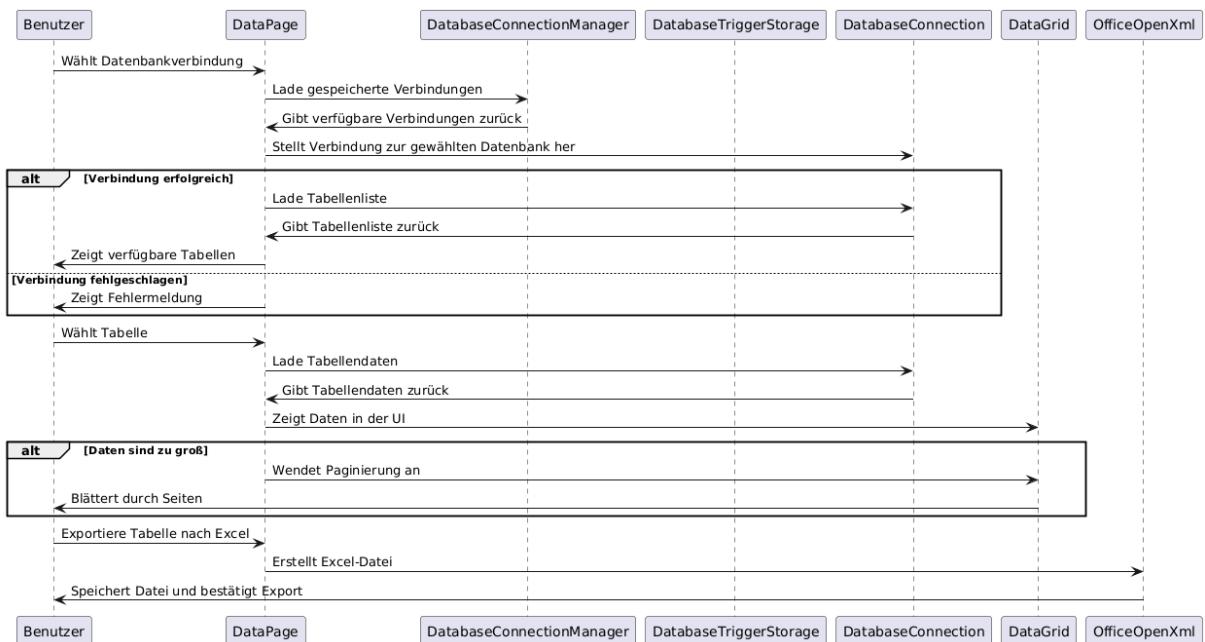


Abbildung 17: Sequenzdiagramm zur Tabellenvisualisierung

Das Sequenzdiagramm veranschaulicht den gesamten Prozess der Tabellenvisualisierung innerhalb der Anwendung.

Zunächst wählt der Benutzer eine Datenbankverbindung aus. Die Anwendung lädt daraufhin die gespeicherten Verbindungen aus dem DatabaseConnectionManager und zeigt diese in der Benutzeroberfläche an. Sobald der Benutzer eine Verbindung auswählt, stellt das System eine Verbindung zur gewählten Datenbank her. Falls die Verbindung erfolgreich hergestellt wird, ruft die Anwendung die vorhandenen Tabellen aus der Datenbank ab und zeigt sie in der UI an. Sollte die Verbindung fehlschlagen, wird dem Benutzer eine entsprechende Fehlermeldung angezeigt.

Nachdem eine Tabelle ausgewählt wurde, lädt das System die zugehörigen Tabellendaten aus der Datenbank und stellt sie in einem DataGridView dar. Falls die Tabelle eine große Datenmenge enthält, wird eine Paginierung angewendet. Diese sorgt dafür, dass die Daten in übersichtlichen Seitenabschnitten dargestellt werden, sodass der Benutzer bequem durch die Seiten blättern kann, anstatt eine unübersichtliche Menge an Daten auf einmal angezeigt zu bekommen.

Zusätzlich zur direkten Anzeige der Daten bietet die Anwendung auch eine Exportfunktion. Falls der Benutzer die Tabellendaten in Excel speichern möchte, kann er den Export starten. Die Anwendung nutzt dabei die Bibliothek EPPlus, um eine Excel-Datei zu erstellen und die Tabellendaten darin abzuspeichern. Nach dem erfolgreichen Speichern wird eine Bestätigung ausgegeben, sodass der Benutzer weiß, dass der Export erfolgreich abgeschlossen wurde.

4.6.2 Diagrammbasierte Visualisierung

Neben der tabellarischen Anzeige gespeicherter Daten ist die grafische Visualisierung ein essenzieller Bestandteil der Datenanalyse. Diagramme ermöglichen eine schnellere und intuitivere Interpretation großer Datenmengen, insbesondere wenn Trends oder Muster erkannt werden sollen.

Um diesen Aspekt in die Anwendung zu integrieren, wurde eine Funktion zur Erstellung und Anzeige von Diagrammen implementiert. Benutzer können verschiedene Parameter wie Datenbanktabelle, Zeitraum und Variablen auswählen, um ein individuelles Diagramm zu erzeugen. Anschließend können diese Diagramme nicht nur innerhalb der Anwendung betrachtet, sondern auch in verschiedene Formate wie JPEG, Excel oder PDF exportiert werden.

Die Funktionalität gliedert sich in drei Hauptbereiche:

Erstellung eines Diagramms: Hier werden alle notwendigen Parameter eingegeben und gespeichert.

Visualisierung eines Diagramms: Die gespeicherten Diagramme können geladen und angezeigt werden.

Export des Diagramms: Nutzer haben die Möglichkeit, Diagramme in verschiedenen Formaten zu speichern und weiterzuverarbeiten.

Im folgenden Abschnitt wird der technische Ablauf dieser Funktionen näher erläutert.

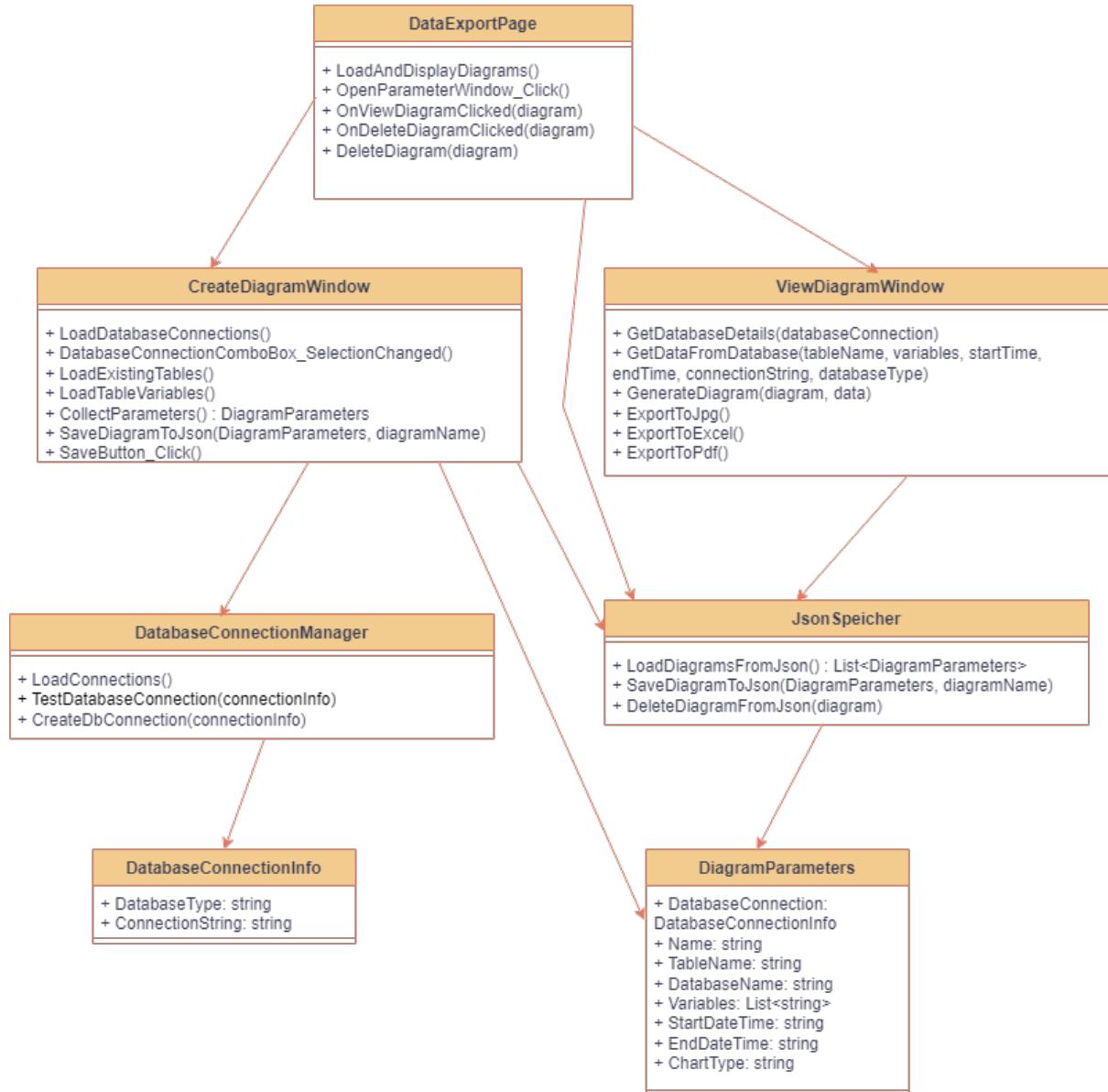


Abbildung 18: Klassendiagramm zur Diagrammbasierte Datenvisualisierung

Das Klassendiagramm stellt die Struktur der Implementierung zur Verwaltung, Erstellung, Anzeige und Exportierung von Diagrammen in der Anwendung dar. Es zeigt die Beziehungen zwischen den verschiedenen Klassen und wie sie miteinander interagieren. Die zentrale Klasse ist **DataExportPage**, die für die Verwaltung und Anzeige der gespeicherten Diagramme verantwortlich ist. Sie ruft gespeicherte Diagramme aus der JSON-Datei ab und ermöglicht dem Benutzer, neue Diagramme zu erstellen oder vorhandene zu löschen. Sie interagiert direkt mit der Klasse **CreateDiagramWindow**, wenn ein neues Diagramm erstellt wird. Sobald der Benutzer die Erstellung eines Diagramms initiiert, öffnet **DataExportPage** das **CreateDiagramWindow**, in dem der Benutzer eine Datenbankverbindung auswählt, eine Tabelle angibt, Variablen bestimmt und einen Zeitrahmen festlegt. Nachdem die Parameter gesetzt wurden, speichert **CreateDiagramWindow** die Diagrammdaten mit Hilfe der Klasse **JsonSpeicher**.

JsonSpeicher ist für das Laden und Speichern der Diagrammparameter in einer JSON-Datei zuständig. Es stellt Methoden zur Verfügung, um bestehende Diagramme zu laden, neue Diagramme in der Datei zu speichern und unerwünschte Diagramme zu löschen. Die gespeicherten Diagrammdaten werden in Objekten der Klasse DiagramParameters organisiert. Diese Klasse enthält Attribute wie den Namen des Diagramms, die gewählte Datenbank, die zugehörige Tabelle, die gewählten Variablen sowie den definierten Zeitraum für die Daten.

Die Klasse DatabaseConnectionManager spielt eine wesentliche Rolle, indem sie die vorhandenen Datenbankverbindungen lädt und verwaltet. Sie bietet Methoden zur Erstellung einer Datenbankverbindung und nutzt dazu die Klasse DatabaseConnectionInfo, die Informationen zur Datenbankverbindung speichert, wie beispielsweise den Datenbanktyp und den Verbindungsstring.

Sobald ein Benutzer ein gespeichertes Diagramm anzeigen möchte, wird es über die Klasse ViewDiagramWindow geladen. Diese Klasse ruft die entsprechenden Diagrammparameter aus JsonSpeicher ab, verbindet sich mit der Datenbank über den gespeicherten Verbindungsstring und ruft die für den gewählten Zeitraum relevanten Daten ab. Dafür wird die Methode GetDataFromDatabase verwendet, die auf Basis des Datenbanktyps eine SQL-Abfrage erstellt und die Daten in einer geeigneten Struktur zurückliefert.

Für die visuelle Darstellung des Diagramms nutzt ViewDiagramWindow die Bibliothek OxyPlot. Diese Bibliothek wird zur Erstellung verschiedener Diagrammtypen verwendet, beispielsweise Liniendiagramme. Die Klasse generiert ein Diagramm aus den abgerufenen Daten, fügt Achsenbeschriftungen und eine Legende hinzu und zeigt das Diagramm in der Benutzeroberfläche an. Sollte es Probleme beim Laden oder Darstellen der Daten geben, wird eine Fehlermeldung angezeigt.

Neben der Anzeige von Diagrammen bietet die Anwendung auch verschiedene Exportmöglichkeiten. In ViewDiagramWindow sind dafür drei Exportmethoden implementiert. Die Methode ExportToJpg verwendet die Bibliothek WPF RenderTargetBitmap, um das Diagramm als Bild zu speichern. ExportToExcel nutzt die Bibliothek EPPlus, um die Diagrammdaten in einer Excel-Datei abzuspeichern. Schließlich verwendet ExportToPdf die Bibliothek PdfSharpCore, um die Daten als PDF-Dokument zu exportieren. Der Benutzer kann dabei zwischen diesen Formaten wählen und das Diagramm auf seinem lokalen System speichern.

4.7 Datenanalyse

Die Datenanalyse spielt eine zentrale Rolle bei der Interpretation und Weiterverarbeitung der gespeicherten Daten. Sie ermöglicht es, Muster zu erkennen, statistische Auswertungen durchzuführen und automatisierte Prozesse basierend auf den gewonnenen Erkenntnissen zu steuern. In diesem Kapitel wird erläutert, wie die Analyse in der Anwendung umgesetzt wurde und welche Verfahren zur Datenverarbeitung genutzt werden.

4.7.1 Erklärung der Analyseverfahren

In diesem Abschnitt werden die vier verwendeten Analyseverfahren detailliert beschrieben.

Durchschnittsanalyse

Die Durchschnittsanalyse ist eine grundlegende Methode der deskriptiven Statistik, die dazu dient, den zentralen Tendenzwert einer Variablen über einen bestimmten Zeitraum zu bestimmen. Sie liefert eine schnelle Einschätzung über die Werteverteilung, indem der Mittelwert sowie weitere Kennwerte wie das Minimum, das Maximum und die Standardabweichung berechnet werden. Der Mittelwert, auch als arithmetisches Mittel bezeichnet, gibt an, welcher Wert im betrachteten Zeitraum typisch ist. Das Minimum und das Maximum zeigen die Extremwerte der Datenreihe, während die Standardabweichung die Schwankung der Werte um den Mittelwert beschreibt. Eine hohe Standardabweichung deutet darauf hin, dass die Werte stark variieren, während eine niedrige Standardabweichung auf eine relativ stabile Datenlage hinweist.

In der Analyse wird für jede Variable ein kurzer Bericht erstellt, der den berechneten Durchschnittswert sowie den Zeitraum der Analyse enthält. Zum Beispiel kann die Analyse für eine Temperaturvariable ergeben, dass der Durchschnittswert innerhalb eines bestimmten Zeitraums bei 23,45 Grad Celsius liegt. Neben dem Durchschnittswert werden auch die minimalen und maximalen Werte dieser Variablen ermittelt, um die gesamte Spannweite der Messwerte abzubilden. Zusätzlich wird die Standardabweichung berechnet, um zu bewerten, wie stark die Werte um den Durchschnitt streuen.

Die Zusammenfassung der Analyse gibt eine kompakte Übersicht über die berechneten Kennwerte und hebt besondere Auffälligkeiten hervor. Falls eine Variable eine hohe Schwankung aufweist, beispielsweise wenn die Standardabweichung mehr als 20 Prozent des Durchschnittswerts beträgt, wird eine entsprechende Warnung ausgegeben. Ebenso wird geprüft, ob der maximale Wert erheblich über dem Durchschnitt liegt – in diesem Fall wird eine zusätzliche Warnung generiert, die darauf hinweist, dass ein ungewöhnlich hoher Spitzenwert festgestellt wurde. Dadurch ermöglicht die Zusammenfassung eine schnelle Interpretation der Ergebnisse, insbesondere im Hinblick auf potenzielle Unregelmäßigkeiten oder Anomalien in den Daten.

Mathematisch wird der Mittelwert einer Variable als Summe aller Werte dividiert durch die Anzahl der Werte berechnet. Die Formel hierfür lautet:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

wobei x_i die einzelnen Werte der Variable sind und n die Anzahl der Werte darstellt. Die Standardabweichung, welche die Variabilität der Daten beschreibt, wird mit folgender Formel berechnet:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Mediananalyse

Die Mediananalyse ist eine andere Methode der deskriptiven Statistik, die dazu dient, den mittleren Wert einer Datenverteilung zu bestimmen, ohne dass dieser durch extreme Ausreißer beeinflusst wird. Während der arithmetische Mittelwert stark von extrem hohen oder niedrigen Werten beeinflusst werden kann, stellt der Median den Wert dar, der eine geordnete Datenmenge in zwei gleich große Hälften teilt. Er ist besonders nützlich, wenn die Daten asymmetrisch verteilt sind oder starke Schwankungen aufweisen.

In der Analyse wird für jede Variable der Medianwert berechnet und mit weiteren statistischen Kennwerten wie dem Minimum, dem Maximum, der Spannweite und dem Durchschnittswert verglichen. Die Spannweite, definiert als die Differenz zwischen dem größten und dem kleinsten Wert, zeigt, wie stark die Werte auseinanderliegen. Der Vergleich mit dem Durchschnittswert ermöglicht es zudem, Einschätzungen über die Datenverteilung zu treffen: Liegt der Median deutlich unter dem Durchschnitt, deutet dies darauf hin, dass einige hohe Werte den Mittelwert nach oben ziehen. Umgekehrt kann ein Median, der über dem Durchschnitt liegt, darauf hindeuten, dass einige niedrige Werte das arithmetische Mittel nach unten beeinflussen.

Während der Analyse wird für jede Variable ein kurzer Bericht erstellt, der den berechneten Medianwert und den entsprechenden Zeitraum dokumentiert. Beispielsweise könnte die Analyse für eine Temperaturvariable ergeben, dass der Medianwert innerhalb eines bestimmten Zeitraums bei 22,5 Grad Celsius liegt, während die Spannweite zwischen 15,0 Grad und 30,0 Grad variiert. Solche Informationen sind besonders hilfreich, wenn man wissen möchte, welche Werte in einem Datensatz typisch sind und ob es starke Schwankungen oder Ausreißer gibt.

Die Zusammenfassung der Analyse bietet eine kompakte Übersicht über die berechneten Kennwerte und hebt mögliche Auffälligkeiten hervor. Eine wichtige Erkenntnis aus der Mediananalyse kann sein, dass ein hoher Unterschied zwischen Median und Mittelwert auf eine asymmetrische Verteilung oder extreme Ausreißer hindeutet. Falls die Spannweite besonders groß ist, wird eine Warnung ausgegeben, da dies auf starke Schwankungen innerhalb der betrachteten Variable hinweist.

Mathematisch wird der Median einer geordneten Datenmenge x_1, x_2, \dots, x_n wie folgt berechnet:

Falls die Anzahl der Werte n ungerade ist:

$$\tilde{x} = x_{\frac{n+1}{2}}$$

In diesem Fall ist der Median einfach der mittlere Wert der sortierten Liste.

Falls die Anzahl der Werte n gerade ist:

$$\tilde{x} = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$$

Hier wird der Median als der Durchschnitt der beiden mittleren Werte der sortierten Daten berechnet.

Zusätzlich wird die Spannweite (R) wie folgt berechnet:

$$R = x_{\max} - x_{\min}$$

korrelationanalyse

Die Korrelationsanalyse ist eine statistische Methode, die den linearen Zusammenhang zwischen zwei Variablen untersucht. Sie zeigt, ob eine Variable steigt oder fällt, wenn sich die andere verändert. Dabei kann eine positive, negative oder keine Korrelation zwischen den Variablen bestehen. Eine starke positive Korrelation bedeutet, dass ein Anstieg der einen Variable mit einem Anstieg der anderen einhergeht, während eine starke negative Korrelation darauf hinweist, dass eine Variable sinkt, wenn die andere steigt.

In dieser Analyse wird der Pearson-Korrelationskoeffizient verwendet, um die Stärke und Richtung der Beziehung zwischen zwei Variablen zu bestimmen.

Der Korrelationskoeffizient r liegt immer im Bereich von -1 bis $+1$:

$r = +1$ Perfekte positive Korrelation – beide Variablen steigen oder fallen gemeinsam.

$r = -1$ Perfekte negative Korrelation – eine Variable steigt, während die andere fällt.

$r = 0$ Keine Korrelation – es gibt keinen linearen Zusammenhang zwischen den Variablen.

$0.3 < r < 0.7$ Mäßige positive Korrelation.

$-0.7 < r < -0.3$ Mäßige negative Korrelation.

$-0.3 < r < 0.3$ Schwache oder keine Korrelation.

Die Berechnung des Pearson-Korrelationskoeffizienten basiert auf folgender mathematischer Formel:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Dabei sind:

x_i und y_i die einzelnen Werte der beiden Variablen.

\bar{x} und \bar{y} die Mittelwerte der jeweiligen Variablen.

Die Summe im Zähler beschreibt die Kovarianz der Variablen.

Der Nenner stellt die Standardabweichungen beider Variablen dar.

Während der Analyse werden die zwei ausgewählten Variablenpaare betrachtet. Für jede Kombination wird der Korrelationskoeffizient berechnet und in einem Bericht zusammengefasst. Falls eine Variable nicht genügend Datenpunkte hat, um eine Korrelation zu berechnen, wird eine Warnung ausgegeben.

Die Zusammenfassung der Korrelationsanalyse gruppiert die Ergebnisse basierend auf der Stärke der Korrelation:

Starke positive Korrelationen: Variablenpaare, die eng miteinander zusammenhängen, werden hier aufgelistet.

Mäßige positive Korrelationen: Diese Beziehungen sind weniger stark, könnten aber dennoch auf eine gewisse Verbindung zwischen den Variablen hinweisen.

Schwache oder keine Korrelation: Falls der Korrelationskoeffizient nahe 0 liegt, wird das entsprechende Variablenpaar hier aufgenommen.

Mäßige negative Korrelationen: Diese Werte zeigen einen umgekehrten, aber nicht extrem starken Zusammenhang.

Starke negative Korrelationen: Hier werden Variablenpaare mit einem stark entgegengesetzten Verlauf aufgelistet.

Wenn eine starke positive oder negative Korrelation festgestellt wird, kann dies ein Hinweis darauf sein, dass eine Variable potenziell zur Vorhersage der anderen genutzt werden könnte. Falls jedoch keine signifikanten Korrelationen ermittelt wurden, deutet dies darauf hin, dass die untersuchten Variablen möglicherweise unabhängig voneinander sind.

Trend-Analyse mit gleitendem Durchschnitt

Die Trendanalyse ist eine Methode der Zeitreihenanalyse, die darauf abzielt, langfristige Veränderungen innerhalb einer Datenreihe zu identifizieren. Ein besonders häufig verwendetes Verfahren zur Glättung von Schwankungen ist der gleitende Durchschnitt (Moving Average), der zur Berechnung eines Trends eingesetzt wird. Diese Methode hilft dabei, kurzfristige Schwankungen zu reduzieren und ermöglicht eine bessere Erkennung von übergeordneten Trends in den Daten.

Bei der Trendanalyse wird zunächst für jede Variable eine Zeitreihe von Werten betrachtet. Anschließend wird ein gleitender Durchschnitt berechnet, indem ein Mittelwert über eine bestimmte Anzahl von vorhergehenden Werten (das sogenannte Fenster oder Window Size) bestimmt wird. Je größer das Fenster, desto stärker werden kurzfristige Schwankungen herausgefiltert.

In der Analyse wird für jede Variable geprüft, ob der Trend über den analysierten Zeitraum steigend, fallend oder stabil ist. Dazu werden die Durchschnittswerte am Anfang und am Ende der Zeitreihe verglichen. Ist der Wert am Ende höher als am Anfang, spricht man von einem steigenden Trend; ist er niedriger, liegt ein fallender Trend vor. Falls sich die Werte kaum verändern, wird die Variable als stabil klassifiziert.

Die Analyse liefert für jede Variable folgende Kennwerte:

Start-Durchschnitt: Durchschnittswert am Beginn des Zeitraums.

End-Durchschnitt: Durchschnittswert am Ende des Zeitraums.

Trendwert: Die Differenz zwischen End-Durchschnitt und Start-Durchschnitt.

Die mathematische Berechnung der Trendanalyse basiert auf dem gleitenden Durchschnitt (Moving Average), welcher folgendermaßen definiert ist:

$$MA_t = \frac{1}{n} \sum_{i=t-n+1}^t x_i$$

Dabei gilt:

MA_t ist der gleitende Durchschnitt zum Zeitpunkt t .

x_i sind die Werte der betrachteten Variable.

n ist die Fenstergröße, über die der Durchschnitt berechnet wird.

Der Trendwert wird durch die Differenz des letzten und ersten gleitenden Durchschnittswerts berechnet:

$$T = MA_{\text{Ende}} - MA_{\text{Start}}$$

wobei:

MA_{Start} der erste berechnete gleitende Durchschnitt ist.

MA_{Ende} der letzte berechnete gleitende Durchschnitt ist.

T gibt an, ob der Trend positiv (steigend), negativ (fallend) oder stabil ist.

Die Klassifizierung erfolgt nach folgenden Kriterien:

$$\begin{cases} T > 0, & \text{positiver Trend} \\ T < 0, & \text{negativer Trend} \\ T \approx 0, & \text{stabiler Trend} \end{cases}$$

4.7.2 Architektur und Implementierung der Datenanalyse

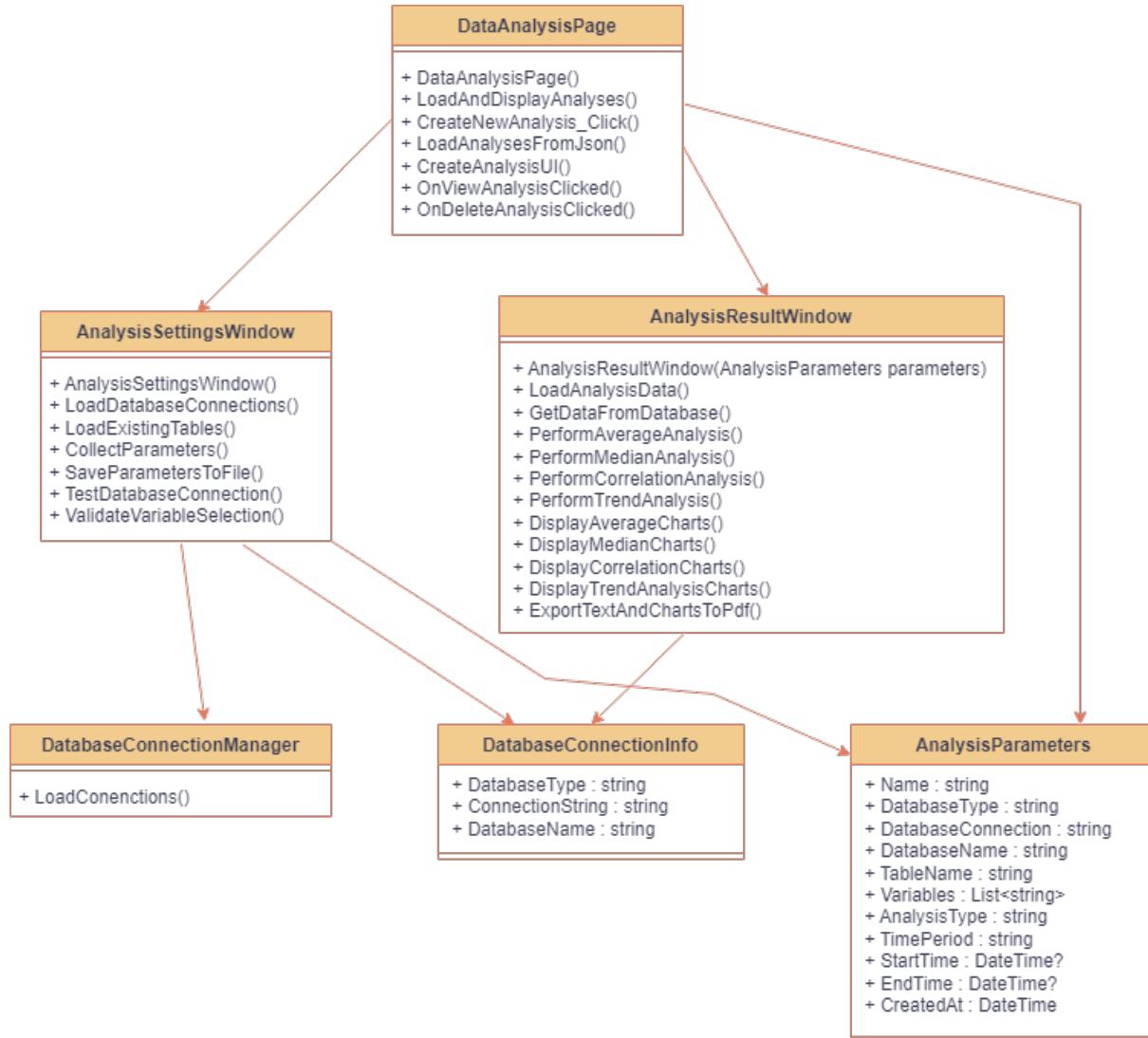


Abbildung 19: Klassendiagramm der Datenanalyse

Das Klassendiagramm stellt die wichtigsten Klassen des Datenanalyse-Systems dar und zeigt ihre Beziehungen zueinander. Die zentrale Klasse ist **AnalysisResultWindow**, die als Hauptfenster für die Anzeige der Analyseergebnisse dient. Diese Klasse nutzt verschiedene Methoden zur Durchführung der Analysen, darunter **PerformAverageAnalysis**, **PerformMedianAnalysis**, **PerformTrendAnalysis** und **PerformCorrelationAnalysis**. Jede dieser Methoden ist für eine bestimmte Art der Datenanalyse verantwortlich.

Die Klasse **DataAnalysisPage** ist das Hauptsteuerungselement für die Verwaltung und Anzeige von gespeicherten Analysen. Sie ist mit der Klasse **AnalysisSettingsWindow** verbunden, die für die Konfiguration neuer Analysen genutzt wird. Über das **AnalysisSettingsWindow** kann der Nutzer eine Datenbankverbindung auswählen, Tabellen und Variablen bestimmen und eine bestimmte Analyseart festlegen.

Die **AnalysisParameters**-Klasse speichert alle relevanten Parameter einer Analyse, darunter die ausgewählten Variablen, die Art der Analyse, die Datenbankverbindung sowie den analysierten Zeitraum. Sie wird von verschiedenen Komponenten genutzt, um die benötigten Daten aus der Datenbank zu laden und die Analyse durchzuführen.

Die Klasse DatabaseConnectionInfo verwaltet die Informationen zur Datenbankverbindung. Dies geschieht in Zusammenarbeit mit DatabaseConnectionManager, der eine Liste der gespeicherten Datenbankverbindungen bereitstellt und das Laden bestehender Verbindungen aus der Konfiguration ermöglicht.

Die AnalysisResultWindow-Klasse verwendet GetDataFromDatabase, um die Daten aus der Datenbank zu extrahieren. Diese Methode erstellt die notwendigen SQL-Abfragen basierend auf dem gewählten Zeitraum und den ausgewählten Variablen. Sobald die Daten geladen sind, werden sie von den jeweiligen Analyse-Methoden verarbeitet und in Diagrammen und Berichten dargestellt.

Aufbau der Diagramme in der Analyse

Für die Darstellung der Analyseergebnisse werden Diagramme verwendet, die mit der LiveCharts-Bibliothek (LiveCharts.Wpf) erstellt werden. Es gibt verschiedene Arten von Diagrammen, die je nach Analyseart verwendet werden:

Balkendiagramme für den Durchschnitt (ColumnSeries): Diese Diagramme zeigen den Durchschnitt einer Variablen über einen bestimmten Zeitraum. Die Balkenhöhe entspricht dem berechneten Mittelwert der Variablen.

Streudiagramme für Korrelationen (ScatterSeries): Diese Diagramme stellen die Beziehung zwischen zwei Variablen dar. Jeder Punkt auf dem Diagramm entspricht einem Wertepaar aus den beiden Variablen. Zusätzlich wird eine Trendlinie hinzugefügt, die die Richtung und Stärke der Korrelation visuell verdeutlicht.

Liniendiagramme für Trends (LineSeries): Die Trendanalyse wird durch ein Liniendiagramm dargestellt, das die Werte einer Variablen über die Zeit zeigt. Zusätzlich wird eine gleitende Durchschnittslinie berechnet und über das Diagramm gelegt, um langfristige Trends zu erkennen.

Histogramme für die Median-Analyse: Diese Diagramme zeigen den Median einer Variablen zusammen mit ihrer Spannweite (Minimum und Maximum). Ein Balken repräsentiert den Median, während eine Linie die gesamte Spannweite der Werte markiert.

Darstellung des Analyse-Textes

Der detaillierte Analysebericht wird neben den Diagrammen in einem formatierten Textfeld des Fensters angezeigt. Dieser Bericht startet mit einer kurzen Einleitung, in der die wichtigsten Ergebnisse der Analyse zusammengefasst werden. Anschließend folgen die berechneten Werte jeder einzelnen Variablen sowie relevante Hinweise oder Warnungen, falls besondere Auffälligkeiten in den Daten entdeckt wurden. Abschließend erfolgt eine Bewertung, die mögliche Interpretationen der Resultate anbietet. Um entscheidende Informationen optisch hervorzuheben, werden Symbole in verschiedenen Farben dargestellt.

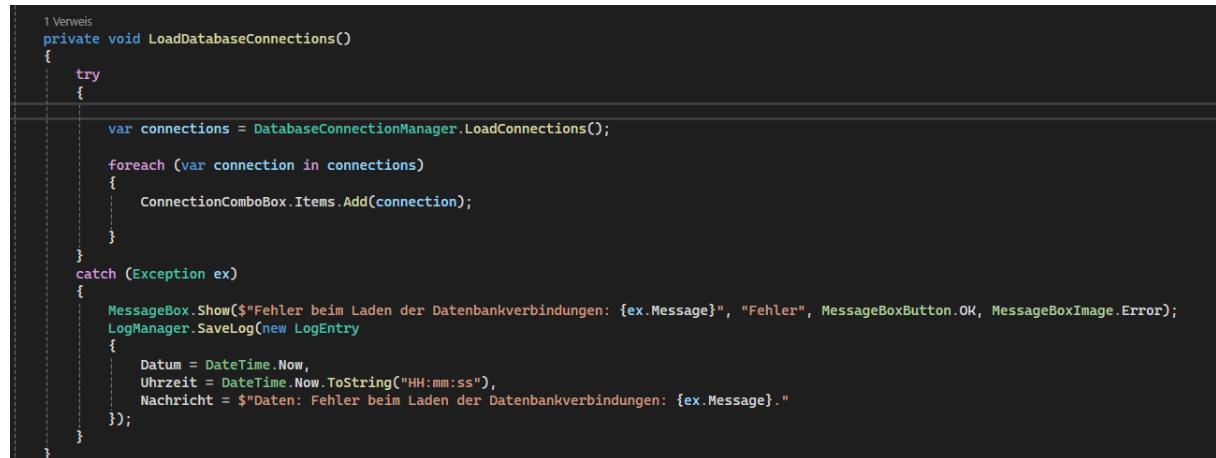
4.8 Fehlerbehandlung und Logging

In einer Softwareanwendung ist eine zuverlässige Fehlerbehandlung und Logging-Mechanismus unerlässlich, um Probleme zu identifizieren, zu diagnostizieren und nachzuvollziehen. Das Logging hilft nicht nur bei der Fehlersuche, sondern speichert auch wichtige Systemereignisse, wie erfolgreiche Aktionen oder Benutzerinteraktionen, um die Anwendungsstabilität zu überwachen.

4.8.1 Logging-Mechanismus und Fehlererfassung

Der Logging-Mechanismus in unserer Anwendung basiert auf zwei Hauptkomponenten: der Klasse LogEntry und der Klasse LogManager. Die LogEntry-Klasse dient als Datenstruktur für jeden einzelnen Log-Eintrag und speichert relevante Informationen wie das Datum, die Uhrzeit und die eigentliche Nachricht des Ereignisses. Dies ermöglicht eine strukturierte und nachvollziehbare Erfassung von Fehlern oder anderen wichtigen Ereignissen.

Die LogManager-Klasse übernimmt die Verwaltung dieser Log-Einträge. Sie bietet Methoden zum Speichern (SaveLog) und Laden (LoadLogs) der Logs. Beim Speichern wird ein neuer Eintrag der bestehenden Log-Datei hinzugefügt, die im JSON-Format vorliegt. Die Ladefunktion ermöglicht das Auslesen der gespeicherten Logs, um diese in der Benutzeroberfläche anzuzeigen. Durch diesen Mechanismus können sowohl Fehler als auch erfolgreiche Vorgänge protokolliert werden, was eine wichtige Grundlage für die Systemüberwachung bildet.



```
1 Verweis
private void LoadDatabaseConnections()
{
    try
    {
        var connections = DatabaseConnectionManager.LoadConnections();

        foreach (var connection in connections)
        {
            ConnectionComboBox.Items.Add(connection);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Fehler beim Laden der Datenbankverbindungen: {ex.Message}", "Fehler", MessageBoxButtons.OK, MessageBoxIcon.Error);
        LogManager.SaveLog(new LogEntry
        {
            Datum = DateTime.Now,
            Uhrzeit = DateTime.Now.ToString("HH:mm:ss"),
            Nachricht = $"Daten: Fehler beim Laden der Datenbankverbindungen: {ex.Message}."
        });
    }
}
```

Abbildung 20: LoadDatabaseConnections() Methode

Im gezeigten Codeabschnitt ist die Methode LoadDatabaseConnections() zu sehen, die für das Laden von Datenbankverbindungen verantwortlich ist. Der Prozess beginnt mit einem try-Block, in dem die Methode DatabaseConnectionManager.LoadConnections() aufgerufen wird, um die verfügbaren Verbindungen abzurufen. Diese Verbindungen werden dann in einer foreach-Schleife durchlaufen und der ConnectionComboBox hinzugefügt, sodass der Benutzer die Datenbankverbindungen in einer Auswahlbox sieht.

Falls jedoch während dieses Prozesses ein Fehler auftritt, wird der catch-Block aktiv. In diesem wird zunächst eine MessageBox angezeigt, die dem Benutzer eine Fehlermeldung mit der Ursache (ex.Message) liefert. Zusätzlich wird ein neuer Log-Eintrag mit dem aktuellen Datum, der Uhrzeit und der Fehlerbeschreibung erstellt und mit LogManager.SaveLog() gespeichert.

Dieser Mechanismus stellt sicher, dass Fehlermeldungen nicht nur kurzfristig sichtbar sind, sondern auch langfristig protokolliert werden. Dadurch kann ein Entwickler oder Administrator zu einem späteren Zeitpunkt die Logs auswerten und genau nachvollziehen, wann und warum ein Fehler aufgetreten ist.

4.8.2 Anzeige und Verwaltung der Logs

Die Anzeige und Verwaltung der Logs in der Anwendung läuft über die LogsPage, wo alle gespeicherten Log-Einträge in einer übersichtlichen Liste dargestellt werden. Jeder Log-Eintrag besteht aus dem Datum, der Uhrzeit und der eigentlichen Nachricht. Um die Lesbarkeit zu verbessern, werden Fehler-Logs optisch hervorgehoben – sie bekommen eine dunklere Hintergrundfarbe, einen roten Rahmen und ein Warnsymbol, sodass sie sofort auffallen. Normale Meldungen und Erfolgsmeldungen bleiben hingegen in der Standardfarbe.

Damit die Nutzer nicht endlos scrollen müssen, gibt es eine Paginierung, die die Logs seitenweise lädt. So bleibt die Anwendung schnell und übersichtlich, auch wenn viele Logs gespeichert sind. Außerdem können Nutzer gezielt nach bestimmten Begriffen suchen oder die Einträge nach Datum sortieren – entweder von neu nach alt oder umgekehrt. Das erleichtert es, schnell die relevanten Logs zu finden.

Ein weiteres wichtiges Feature ist der Export als PDF. Nutzer können einzelne oder mehrere Logs mit Checkboxen markieren und dann per Knopfdruck als PDF speichern. Dafür nutzt die Anwendung die PdfSharpCore-Bibliothek, die eine saubere Formatierung ermöglicht. In der PDF gibt es eine klare Tabellenstruktur mit Spalten für Datum, Uhrzeit und Nachricht. Fehlerhafte Logs werden zusätzlich in roter Schrift dargestellt, um sie auch im Dokument sofort erkennbar zu machen.

5 Test und Validierung

5.1 Testkonzept und Methodik

Die Tests für unsere Anwendung wurden in zwei Phasen durchgeführt. Dabei wurde zunächst jede Funktion einzeln getestet, bevor die gesamte Anwendung mit einer echten SPS überprüft wurde. Ziel war es, sicherzustellen, dass die OPC-UA-Kommunikation stabil funktioniert, die Datenbank korrekt arbeitet und die Visualisierung fehlerfrei läuft.

Erste Phase: Tests mit Simulationsservern

Während der Implementierung wurde jede Funktion direkt nach ihrer Fertigstellung getestet. Dafür wurden zwei verschiedene OPC-UA-Simulationsserver genutzt: der Prosys Simulation Server und der Unified Automation C++ OPC-UA Server (UACPPServer). Der Prosys Server bot die Möglichkeit, eigene OPC-UA-Variablen mit verschiedenen Datentypen anzulegen. Dies war besonders hilfreich, um zu testen, wie die Anwendung mit unterschiedlichen Datentypen umgeht – zum Beispiel Integer, Float oder Strings. Dadurch konnte die Datenverarbeitung flexibel überprüft werden.

Der UACPPServer von Unified Automation funktionierte etwas anders, da die Variablen dort vordefiniert waren und nicht angepasst werden konnten. Trotzdem war dieser Server hilfreich, weil er sich mehr wie eine echte SPS verhielt. So konnte sichergestellt werden, dass die Anwendung auch mit festen Variablen zuverlässig funktioniert.

In dieser Phase wurde vor allem geprüft, ob die Verbindung zu den Servern stabil ist, die Daten korrekt ausgelesen und verarbeitet werden und wie sich die Anwendung in verschiedenen Szenarien verhält.

Zweite Phase: Test mit der echten SPS

In dieser Phase war der Test mit einer realen SPS unter echten Bedingungen. Damit die SPS als OPC-UA-Server genutzt werden konnte, musste zunächst der OPC-UA-Server im TIA Portal aktiviert und konfiguriert werden[11]. Dies erfolgt über die Gerätekonfiguration, wo OPC-UA in den Eigenschaften der SPS aktiviert wird. Anschließend müssen die Endpunkte, Sicherheitsrichtlinien und Benutzerrechte definiert werden, damit die Anwendung Zugriff auf die gewünschten Variablen erhält. Schließlich werden die entsprechenden Datenbausteine als OPC-UA-Variablen freigegeben, sodass sie vom Client ausgelesen werden können. Erst nach dieser Vorbereitung konnte geprüft werden, ob die Anwendung sich mit der physischen Steuerung verbinden lässt und die Datenübertragung genauso funktioniert wie in der Simulation.

Während der Tests wurde untersucht, ob die Anwendung sich korrekt mit der SPS verbindet und die ausgelesenen Daten konsistent verarbeitet. Zudem wurde beobachtet, wie das System auf unerwartete Situationen reagiert, beispielsweise auf Verbindungsunterbrechungen .

5.2 Durchführung der Tests

Um sicherzustellen, dass die entwickelte Anwendung stabil funktioniert und in unterschiedlichen Umgebungen zuverlässig arbeitet, wurden die Tests in zwei Phasen durchgeführt. Jede Phase hatte ihren eigenen Fokus und wurde unter realistischen Bedingungen getestet.

Die folgende Dokumentation beschreibt die einzelnen Testfälle in jeder Phase detailliert und zeigt, ob die Tests erfolgreich waren oder ob Anpassungen notwendig wurden.

Phase 1: Tests mit Simulationsservern

Testfall 1: Verbindungsaufbau zum Simulationsserver

Erwartetes Ergebnis: Verbindung zum Simulationsserver herstellen, sowohl mit als auch ohne Benutzername und Passwort.

Ergebnis: Verbindung erfolgreich hergestellt.

Anmerkung: Anfangs trat der Fehler 'BadCertificateInvalid' auf. Dies wurde behoben, indem die Zertifikatsprüfung deaktiviert bzw. das korrekte Zertifikat importiert wurde.

Testfall 2: Abruf von OPC-UA-Variablen

Erwartetes Ergebnis: Verschiedene Variablen abrufen und Werte auslesen.

Ergebnis: Server-Variablen wurden erfolgreich ausgelesen und in Echtzeit aktualisiert.

Anmerkung: Keine Fehler aufgetreten.

Testfall 3: Herstellung der Datenbankverbindung

Erwartetes Ergebnis: Verbindung zu verschiedenen Datenbanken (MSSQL, MySQL, PostgreSQL) herstellen.

Ergebnis: Verbindung erfolgreich hergestellt.

Anmerkung: Keine Fehler aufgetreten.

Testfall 4: Speicherung von OPC-UA-Variablen in Excel

Erwartetes Ergebnis: OPC-UA-Variablen in eine Excel-Tabelle speichern, basierend auf verschiedenen Trigger-Typen (zeit- oder ereignisbasiert).

Ergebnis: Variablen wurden in der Excel-Tabelle gespeichert.

Anmerkung: Manche Variablen im Simulationsserver hatten keine passenden Datentypen für bestimmte Trigger-Typen, daher konnten einige Trigger nicht getestet werden.

Testfall 5: Speicherung von OPC-UA-Variablen in der Datenbank

Erwartetes Ergebnis: OPC-UA-Variablen in verschiedenen Datenbanken speichern, basierend auf verschiedenen Trigger-Typen (zeit- oder ereignisbasiert).

Ergebnis: Variablen wurden in der Datenbank gespeichert.

Anmerkung: Ähnlich wie bei den Excel-Triggern konnten einige Trigger nicht getestet werden, da nicht alle Simulationsserver-Variablen kompatible Datentypen hatten.

Testfall 6: Anzeige gespeicherter Daten in Tabellenform

Erwartetes Ergebnis: Anzeige der gespeicherten Daten in einer Datenbanktabelle.

Ergebnis: Datenbanktabellen mit gespeicherten Werten wurden erfolgreich angezeigt.

Anmerkung: Eine Paginierung wurde implementiert, um die UI bei einer großen Datenmenge nicht zu blockieren.

Testfall 7: Darstellung gespeicherter Daten in Diagrammform

Erwartetes Ergebnis: Diagramme konfigurieren und anzeigen.

Ergebnis: Diagramme wurden erfolgreich angezeigt und konnten in verschiedene Formate exportiert werden.

Anmerkung: Fehler : Bei großen Datenmengen kam es zu Ladeverzögerungen, da alle Daten auf einmal geladen wurden.

→ Lösung: Implementierung einer Datenaggregation, um die Anzahl der zu ladenden Datenpunkte zu reduzieren.

Testfall 8: Durchführung der Datenanalyse

Erwartetes Ergebnis: Analyse-Einstellungen konfigurieren, Analyse-Ergebnisse anzeigen und exportieren.

Ergebnis: Analyse-Ergebnisse wurden für verschiedene Analysearten erfolgreich berechnet und konnten als PDF Datei exportiert werden..

Anmerkung: Manche Analyse-Berechnungen dauerten bei großen Datenmengen zu lange und führten zu einer UI-Blockade.

→ Lösung: Implementierung eines asynchronen Analyseprozesses, um die UI nicht zu blockieren.

Phase 2: Tests mit der echten SPS

Testfall 1: Verbindungsaufbau zur SPS

Erwartetes Ergebnis: Verbindung zur Siemens S7-1500 SPS erfolgreich herstellen.

Ergebnis: Verbindung konnte erfolgreich hergestellt werden.

Anmerkung: Fehler: Die S7-1500 verfügt über einen zusätzlichen Sicherheitsmechanismus, der in den Simulationsservern nicht vorhanden war. Dies führte dazu, dass die Verbindung zunächst blockiert wurde.

→ Lösung: Anpassung der ConnectAsync Methode um die Verbindung erfolgreich herzustellen.

Testfall 2: Browsing der SPS-Variablen

Erwartetes Ergebnis: Alle relevanten Variablen der SPS sollten in der UI korrekt angezeigt werden.

Ergebnis: Variablen wurden erfolgreich gebrowszt und angezeigt.

Anmerkung:

Fehler 1: Bei Simulationsservern lagen die Variablen direkt unter dem Root-Ordner, während sie bei der S7-1500 unter namespaceindex 3 in spezifischen Datenblöcken gespeichert waren.

→ Lösung: Implementierung eines erweiterten Browsing-Mechanismus, der die Struktur der SPS berücksichtigt und nur relevante Datenblöcke anzeigt.

Fehler 2: Viele Variablen waren in User Defined Types (UDT) als ByteArray gespeichert, was das direkte Browsen erschwerte.

→ Lösung: Zusätzlich zur bestehenden Browsing-Funktion wurde ein Mechanismus implementiert, um UDT-Variablen separat zu browsen und die darin enthaltenen Untervariablen aufzulösen.

Testfall 3: Herstellung der Datenbankverbindung

Erwartetes Ergebnis: Verbindung zu verschiedenen Datenbanken (MSSQL, MySQL, PostgreSQL) herstellen.

Ergebnis: Verbindung erfolgreich hergestellt.

Anmerkung: Keine Fehler aufgetreten.

Testfall 4: Speicherung von SPS Variablen in Excel

Erwartetes Ergebnis: SPS Variablen sollten basierend auf Trigger-Typen in einer Excel-Tabelle gespeichert werden.

Ergebnis: Variablen wurden in der Excel-Tabelle gespeichert.

Anmerkung: Fehler: In der vorherigen Implementierung wurden die Variablen direkt unter dem Root-Ordner gebrowszt. Da SPS-Variablen jedoch in Datenblöcken liegen, wurden einige Variablen nicht korrekt angezeigt.

→ Lösung: Anpassung der Browsing-Funktionalität, sodass zunächst die Datenblöcke angezeigt werden und anschließend die darin enthaltenen Variablen.

Testfall 5: Speicherung von SPS Variablen in der Datenbank

Erwartetes Ergebnis: SPS Variablen sollten basierend auf Trigger-Typen in der Datenbank gespeichert werden.

Ergebnis: Werte wurden erfolgreich in der Datenbank gespeichert.

Anmerkung:

Fehler 1: Ähnlich wie bei den Excel-Triggern konnten anfangs nicht alle relevanten Variablen korrekt gebrowst werden, da sie sich innerhalb von Datenblöcken (DBs) befanden und nicht direkt unter dem Root-Ordner.

Fehler 2: Beim gleichzeitigen Starten vieler Trigger trat der Fehler 'BadTooManySessions' auf, da für jede gestartete Speicherung eine neue OPC-UA-Session aufgebaut wurde.

→ Lösung: Implementierung eines Session-Management-Systems, das prüft, ob bereits eine offene Session existiert, bevor eine neue aufgebaut wird.

Testfall 6: Anzeige gespeicherter Daten in Tabellenform

Erwartetes Ergebnis: Anzeige der gespeicherten Daten in einer Datenbanktabelle.

Ergebnis: Datenbanktabellen mit gespeicherten Werten wurden erfolgreich angezeigt.

Anmerkung: Keine Fehler aufgetreten.

Testfall 7: Darstellung gespeicherter Daten in Diagrammform

Erwartetes Ergebnis: Diagramme konfigurieren und anzeigen.

Ergebnis: Diagramme wurden erfolgreich angezeigt und konnten in verschiedene Formate exportiert werden.

Anmerkung: Fehler: Wenn eine große Anzahl von Daten ausgewählt wurde, dauerte das Rendern des Diagramms sehr lange.

→ Ursache: Die verwendeten Open-Source-Bibliotheken für die Diagrammdarstellung waren nicht optimiert für große Datenmengen.

Testfall 8: Durchführung der Datenanalyse

Erwartetes Ergebnis: Analyse-Einstellungen konfigurieren, Analyse-Ergebnisse anzeigen und exportieren.

Ergebnis: Analyse-Ergebnisse wurden für verschiedene Analysearten erfolgreich berechnet und konnten als PDF Datei exportiert werden.

Anmerkung: Keine Fehler aufgetreten.

6 Ergebnisse

Die durchgeführten Tests haben gezeigt, dass die entwickelte Anwendung in den meisten Fällen stabil funktioniert und die gewünschten Funktionen korrekt umgesetzt wurden. Während der Tests mit den Simulationsservern konnte erfolgreich nachgewiesen werden, dass die Anwendung sich zuverlässig verbindet, Variablen ausliest und die Daten in unterschiedlichen Formaten speichert. Auch die Visualisierung der Werte in Tabellen und Diagrammen verlief weitgehend fehlerfrei, wobei in einigen Fällen Optimierungen erforderlich waren, um die Performance zu verbessern.

Beim Test mit der echten SPS stellte sich heraus, dass einige Anpassungen notwendig waren, da sich die reale Steuerung in bestimmten Aspekten von den Simulationsservern unterschied. Besonders auffällig war, dass die Variablen in Datenblöcken mit spezifischen Namespace-Indizes gespeichert wurden, was eine Erweiterung der Browsing-Funktion erforderlich machte. Zudem mussten einige Sicherheitsmechanismen der SPS berücksichtigt werden, um eine stabile Verbindung herzustellen. Trotz dieser Herausforderungen konnten die meisten Probleme durch gezielte Anpassungen im Code gelöst werden, sodass die Anwendung letztendlich auch unter realen Bedingungen zuverlässig arbeitete.

Im Folgenden sind einige Screenshots der Benutzeroberfläche dargestellt, die die Testergebnisse veranschaulichen.

Homepage der Anwendung

Die Abbildung zeigt die Hauptansicht der Anwendung. Von hier aus kann der Nutzer auf verschiedene Funktionen zugreifen, darunter OPC-UA-Verbindungen, Datenbankverbindungen, Trigger-Management und Datenvisualisierung. Die Navigation erfolgt entweder über die Seitenleiste mit intuitiven Symbolen oder direkt über die interaktiven Funktionsfelder auf der Startseite, die durch Anklicken geöffnet werden können.



Abbildung 21: Hauptansicht der Anwendung

SPS-Verbindungenfenster

Dieses Fenster ermöglicht die Konfiguration der SPS-Verbindung. Der Nutzer kann die IP-Adresse, den Port sowie Benutzername und Passwort eingeben, falls eine Sicherheitsanmeldung erforderlich ist. Ein Klick auf Verbinden stellt die Verbindung zur SPS her.

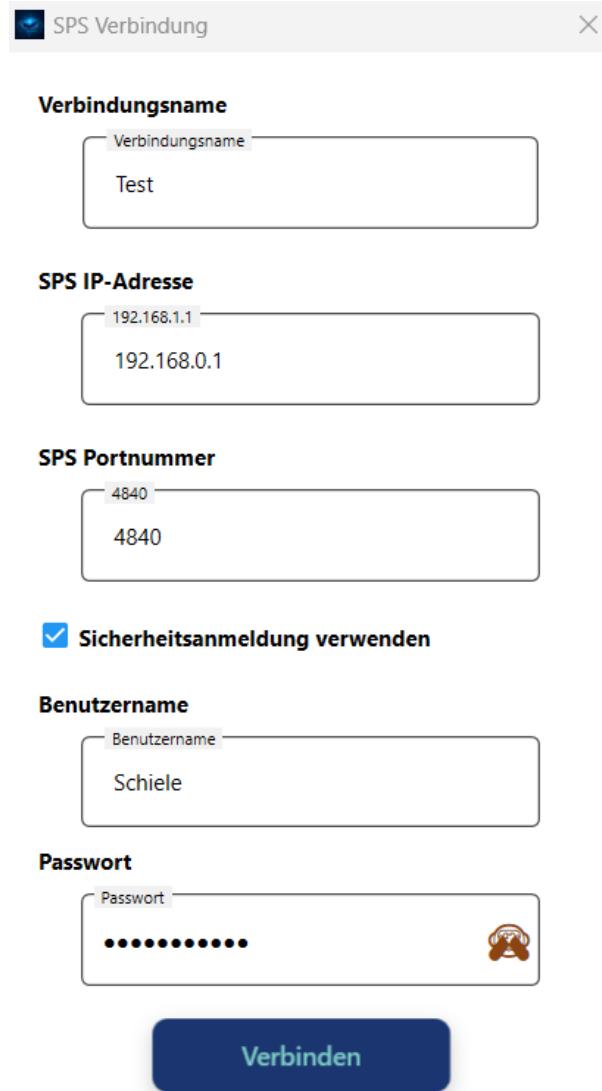


Abbildung 22: Fenster zur Konfiguration der SPS-Verbindung

SPS Verbindungen

Nach der erfolgreichen Herstellung der Verbindung wird die aktive SPS-Verbindung in einer Übersicht dargestellt. Hier sind der Verbindungsname, das Erstellungsdatum, die OPC-UA-Adresse sowie der Verbindungsstatus ersichtlich. Ein grüner Statusindikator signalisiert, dass die Verbindung aktiv ist. Zudem stehen Funktionen zur Verfügung, um die Variablen der SPS einzusehen, die Verbindung zu trennen, Einstellungen zu ändern oder die Verbindung zu löschen.

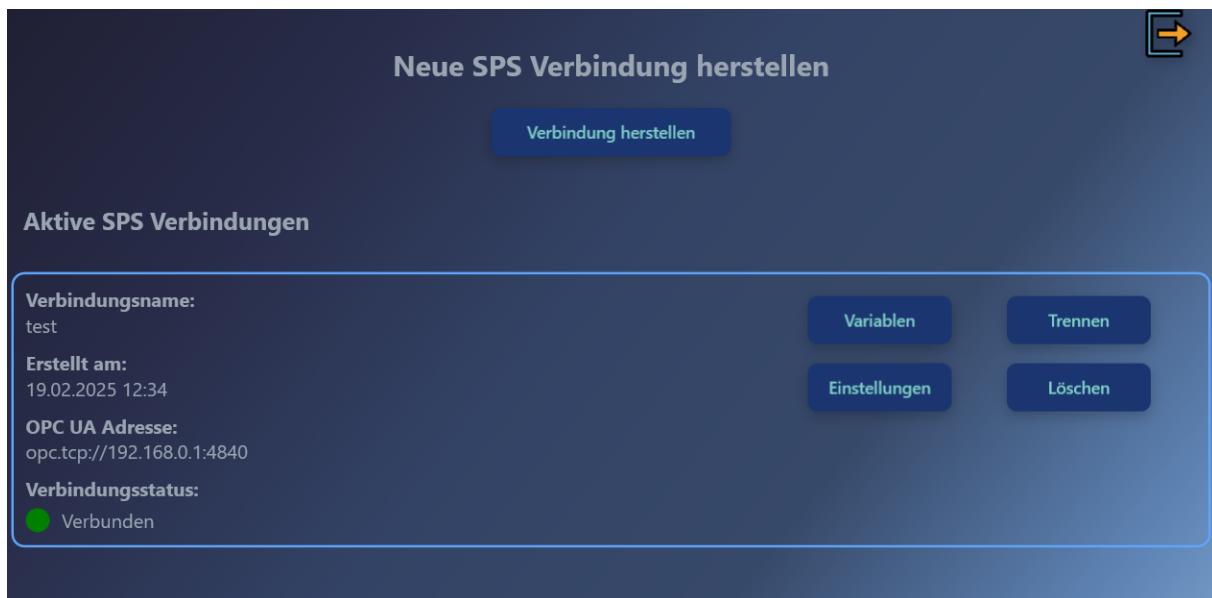


Abbildung 23: Übersicht der aktiven SPS-Verbindungen

Variablen Browsen

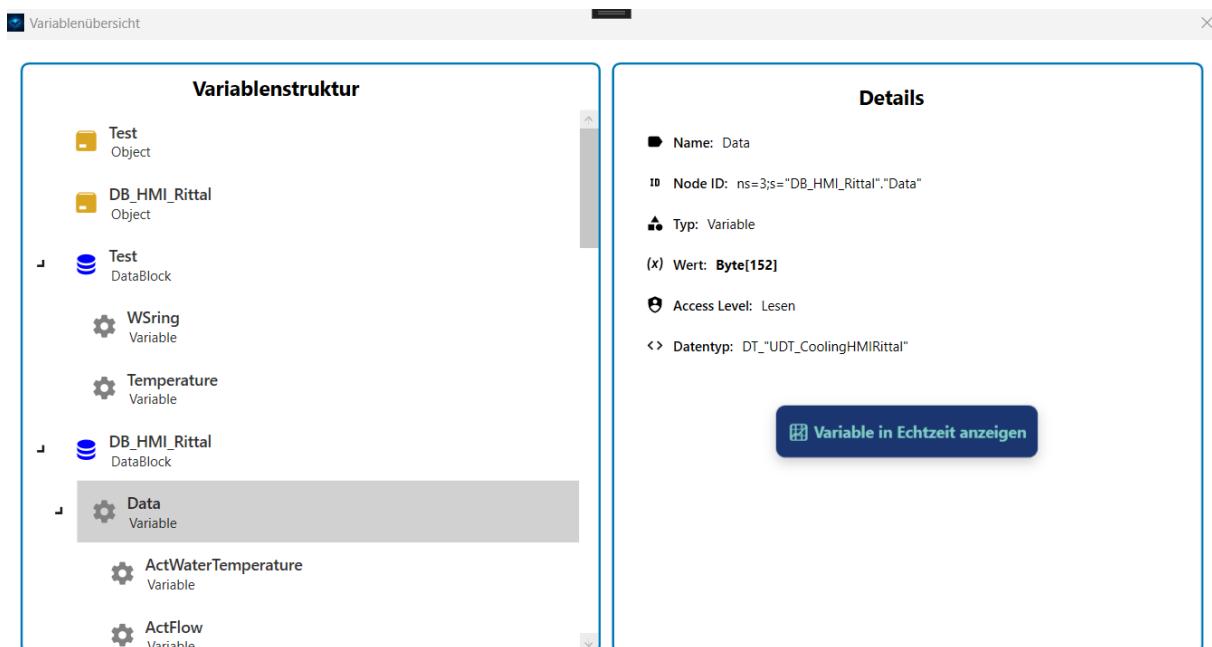


Abbildung 24: Durchsuchen der SPS-Variablenstruktur und Anzeige von Details

Echtzeitanzeige der Variablen

Das Diagramm stellt die Echtzeit-Werte einer SPS-Variablen dar. Nach Auswahl einer Variable im Variablen-Browser kann der Benutzer deren Werte in einem Live-Diagramm verfolgen. Dies ermöglicht eine kontinuierliche Überwachung der Prozesswerte, um Veränderungen sofort zu erkennen.

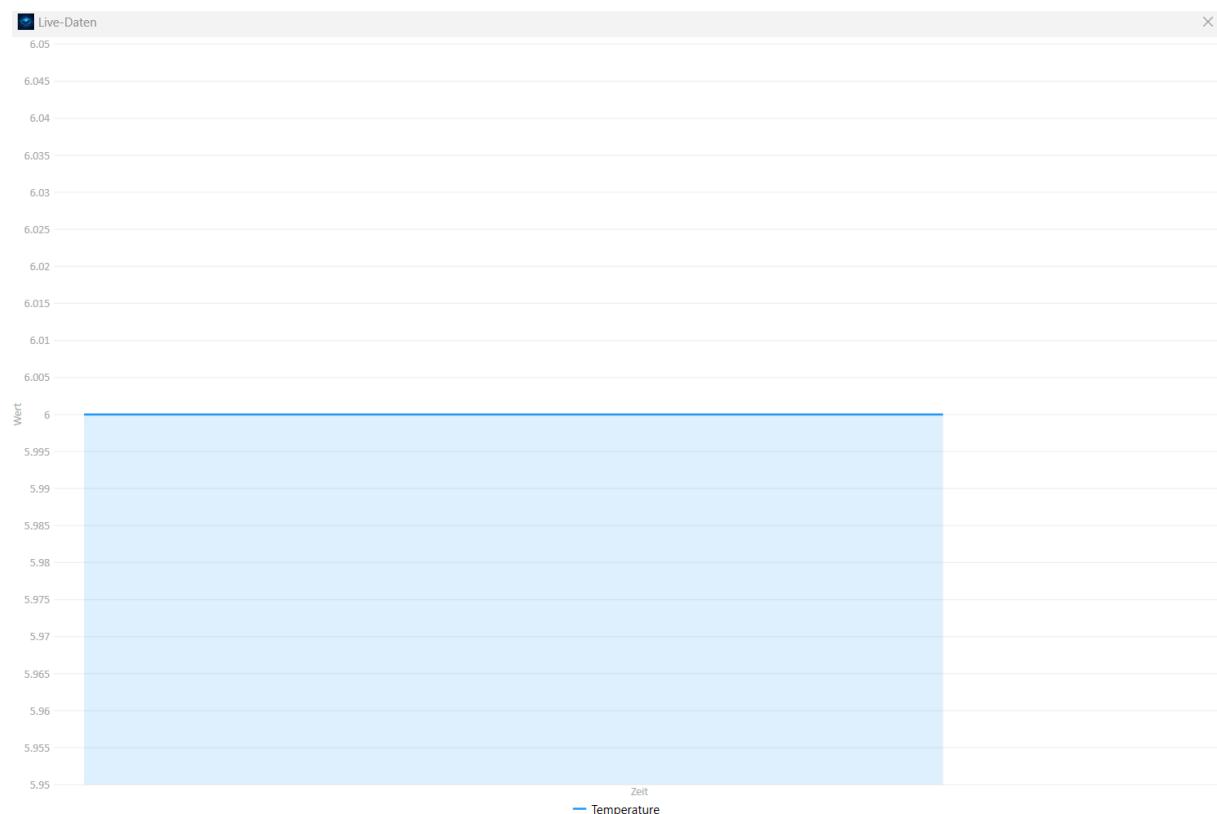


Abbildung 25: Echtzeit-Datenvizualisierung einer SPS-Variablen

Datenbank-Verbindungenfenster

Für die Einrichtung einer MySQL-Datenbank stehen Eingabefelder für Host/IP, Port, Benutzername, Passwort und Datenbankname zur Verfügung. Vor dem Speichern muss die Verbindung getestet werden, um sicherzustellen, dass die Zugangsdaten korrekt sind.

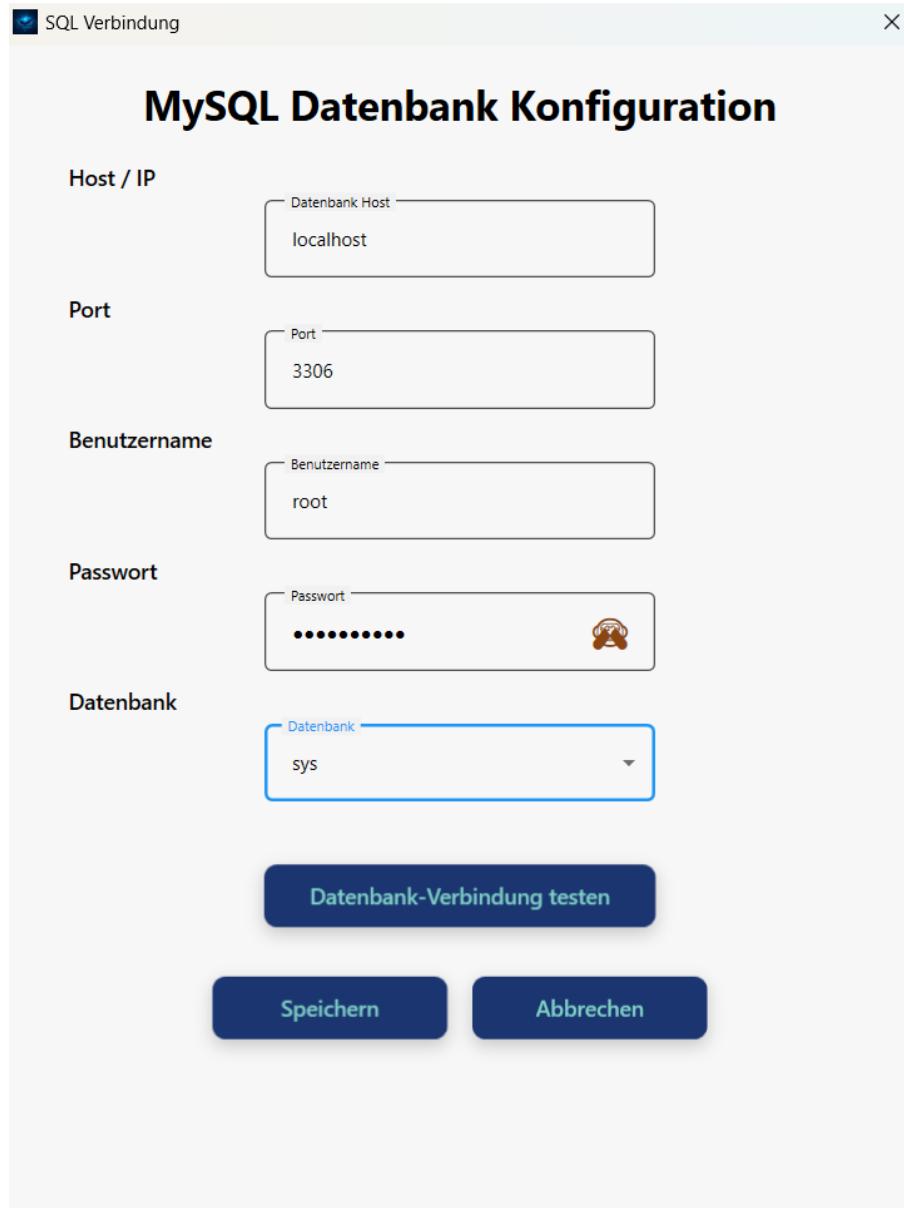


Abbildung 26: Konfigurationsfenster für eine MySQL-Datenbankverbindung

Übersicht der Datenbankverbindungen

Die Abbildung zeigt die Verwaltung und Konfiguration von Datenbankverbindungen in der Anwendung. Oben können neue Verbindungen zu MSSQL, MySQL und PostgreSQL erstellt werden. Bereits verbundene Datenbanken werden darunter angezeigt, inklusive Datenbanktyp, Name, Erstellungszeitpunkt und Verbindungsstatus. Über die Schaltflächen können die Einstellungen einer bestehenden Verbindung angepasst oder die Verbindung gelöscht werden.



Abbildung 27: Übersicht der Datenbankverbindungen mit aktiven Verbindungen

Excel-trigger Konfigurationsfenster

Die Abbildung zeigt die Excel-Konfiguration in der Anwendung, mit der SPS-Variablen automatisiert in eine Excel-Datei geschrieben werden können. Der Nutzer wählt den Schreibmodus, die Datenbausteine und die Variablen aus. Zudem kann das Speicherintervall festgelegt und ein Trigger-Typ gewählt werden.

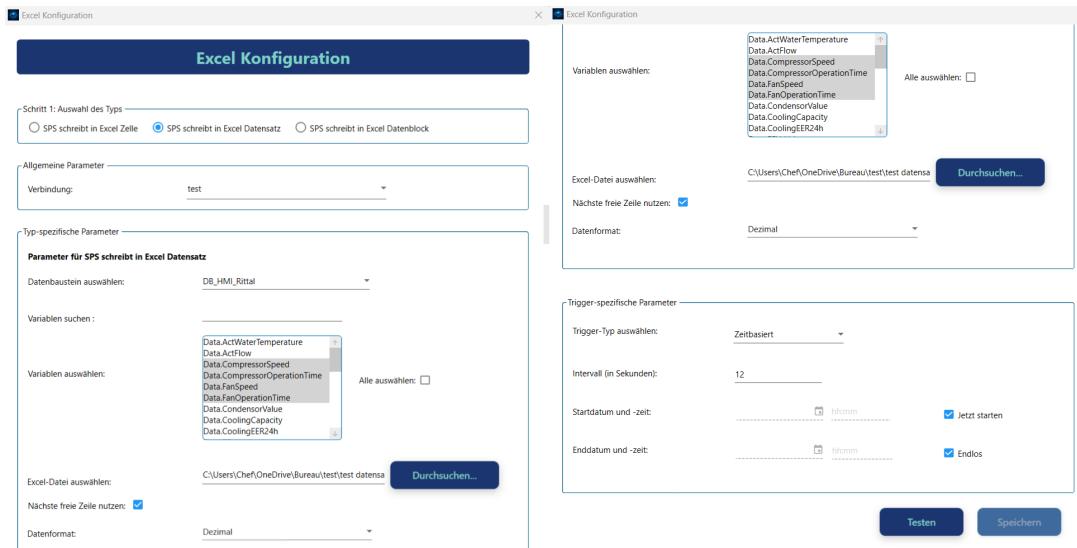


Abbildung 28: Konfiguration der automatisierten Excel-Speicherung

Datenbank-trigger Konfigurationsfenster

Die Abbildung zeigt die Datenbank-Konfiguration in der Anwendung, mit der SPS-Variablen automatisch in einer Datenbank gespeichert werden können. Der Nutzer wählt die Datenbankverbindung, die OPC-UA-Verbindung und die zu speichernden Variablen aus. Zusätzlich kann eine neue Tabelle erstellt oder eine bestehende Tabelle ausgewählt werden. Die Speicherung kann entweder zeitbasiert oder ereignisgesteuert erfolgen.

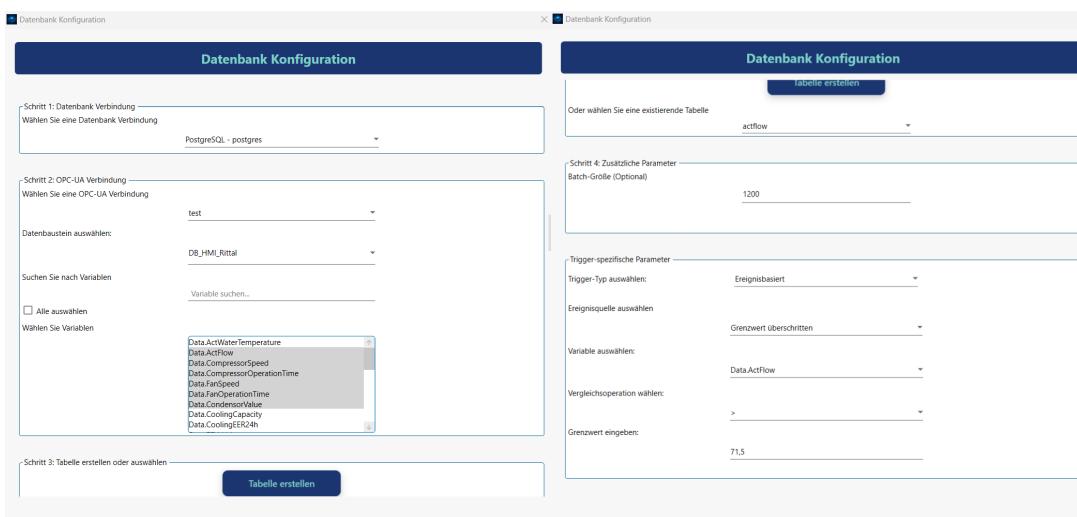


Abbildung 29: Konfiguration der automatisierten Datenbankspeicherung

Trigger Management

Nach der Konfiguration und Speicherung eines Triggers wird dieser in der Trigger-Verwaltung angezeigt. Hier können Nutzer alle gespeicherten Trigger einsehen und verwalten. Über die Schaltflächen können die Trigger gestartet, gestoppt, bearbeitet oder gelöscht werden.

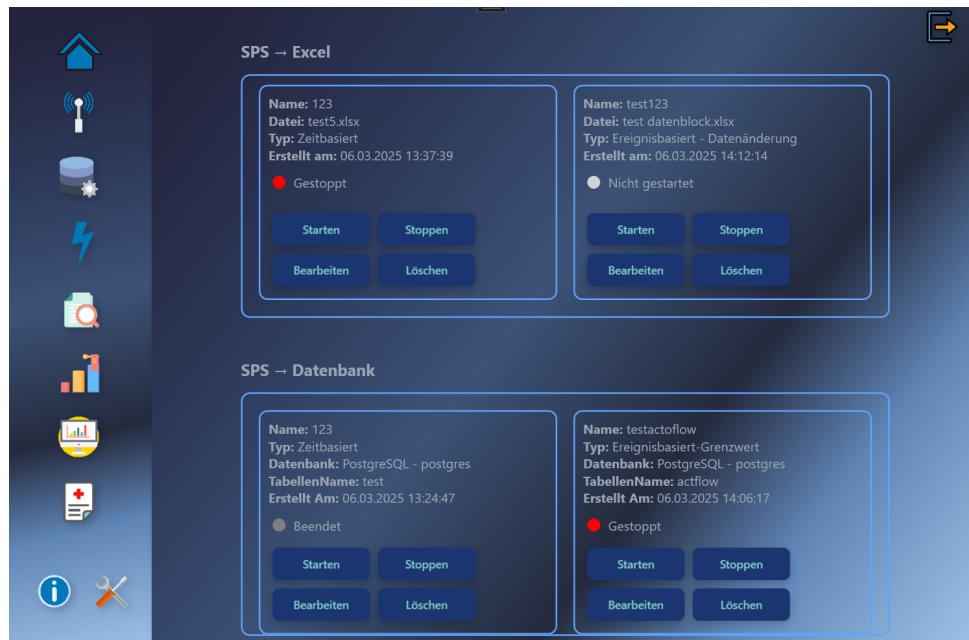


Abbildung 30: Verwaltung gespeicherter Trigger

Tabellenbasierte Datenanzeige

The screenshot shows the 'Daten' (Data) section of a software interface. On the left is a vertical toolbar with icons for upload, download, database, trigger, search, chart, dashboard, and help. The main area has input fields for 'Verbindung wählen:' (Connection select: PostgreSQL - postgres) and 'Tabelle wählen:' (Table select: testtemperature), along with buttons for 'Daten anzeigen' (Display data) and 'Als Excel exportieren' (Export to Excel). Below these is a table with data, and at the bottom are navigation buttons for the first and last pages.

Timestamp	Temperature	
20.02.2025 10:28:25.146	6	
20.02.2025 10:28:26.105	6	
20.02.2025 10:28:27.097	6	
20.02.2025 10:28:28.121	6	
20.02.2025 10:28:29.107	6	
20.02.2025 10:28:30.097	6	
20.02.2025 10:28:31.105	6	
20.02.2025 10:28:32.096	6	
20.02.2025 10:28:33.105	6	
20.02.2025 10:28:34.104	6	
20.02.2025 10:28:35.105	6	
20.02.2025 10:28:36.105	6	
20.02.2025 10:28:37.126	6	
20.02.2025 10:28:38.097	6	

Abbildung 31: Tabellenbasierte Anzeige der gespeicherten Daten

Diagramm Konfigurationsfenster

Für die Erstellung eines neuen Diagramms können die Datenbankverbindung, die gewünschte Tabelle und spezifische Variablen ausgewählt werden. Zudem lässt sich der Zeitbereich definieren und der Diagrammtyp festlegen, um eine maßgeschneiderte Datenvisualisierung zu ermöglichen.

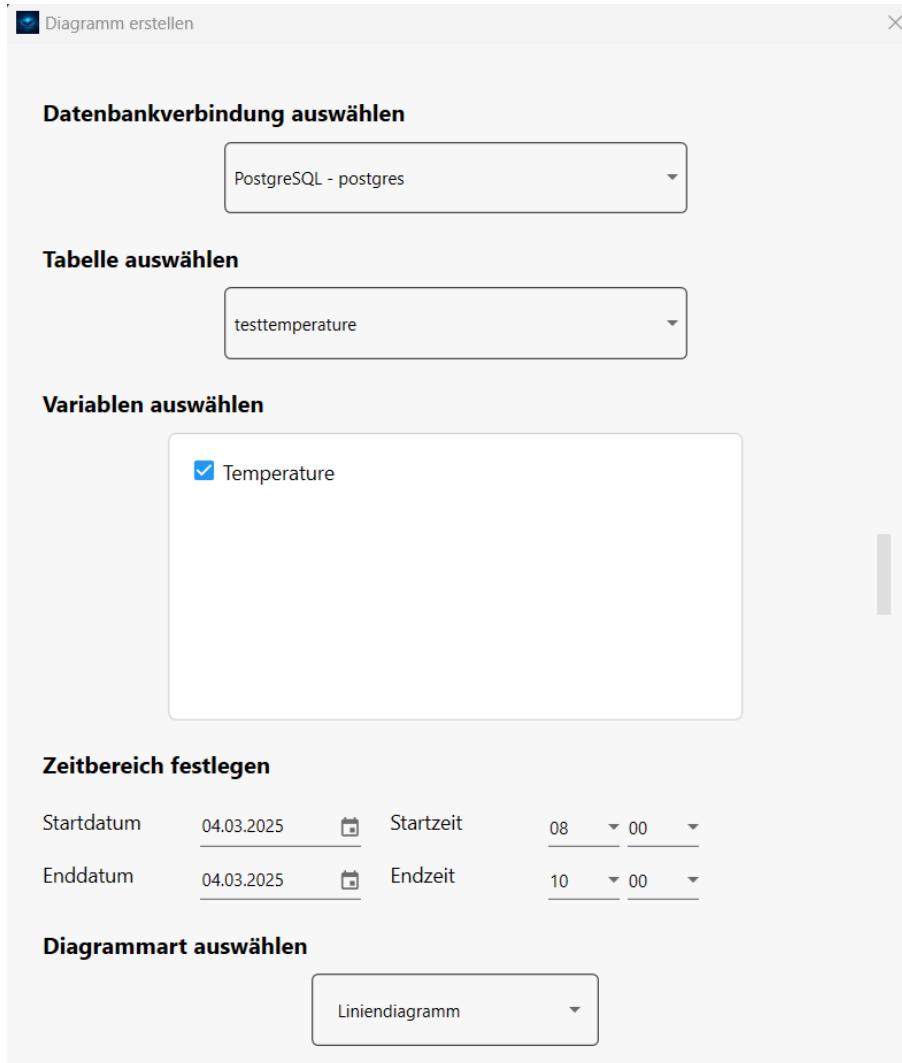


Abbildung 32: Konfigurationsfenster zur Erstellung eines Diagramms

Daten Export Manager

Gespeicherte Diagramme können im Daten Export Manager verwaltet werden. Hier werden vorhandene Diagramme mit Datenbankname, Tabellenname, Erstellungsdatum und Diagrammtyp angezeigt. Über die Optionen kann ein Diagramm betrachtet oder gelöscht werden, und neue Diagramme können erstellt werden.

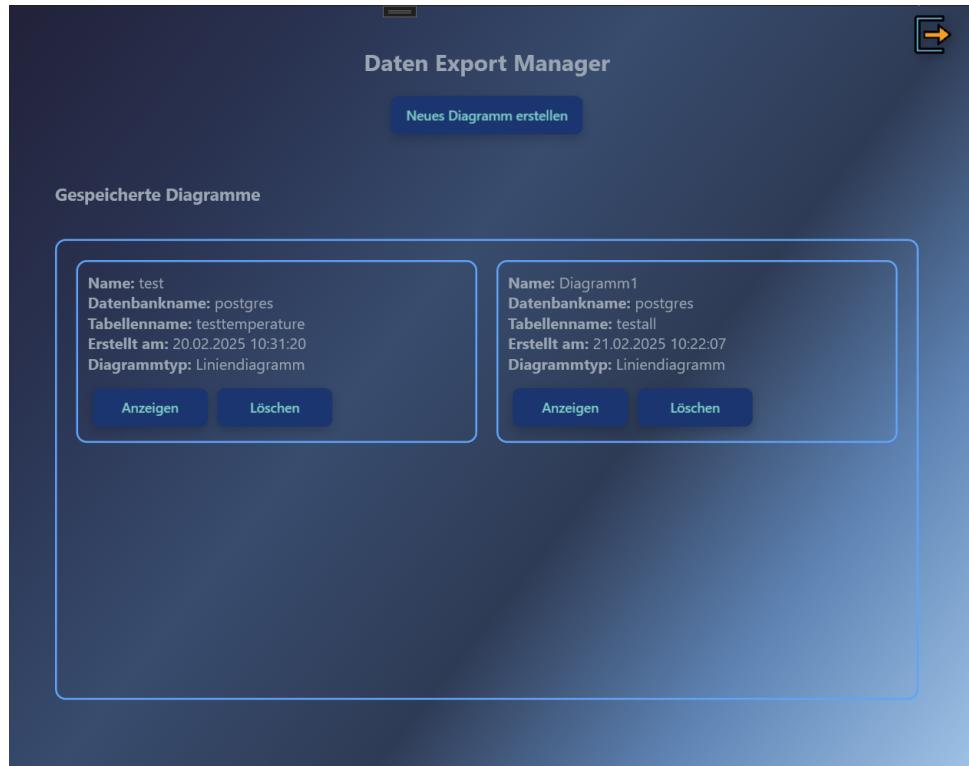


Abbildung 33: Verwaltung gespeicherter Diagramme

Diagramm Anzeigefenster

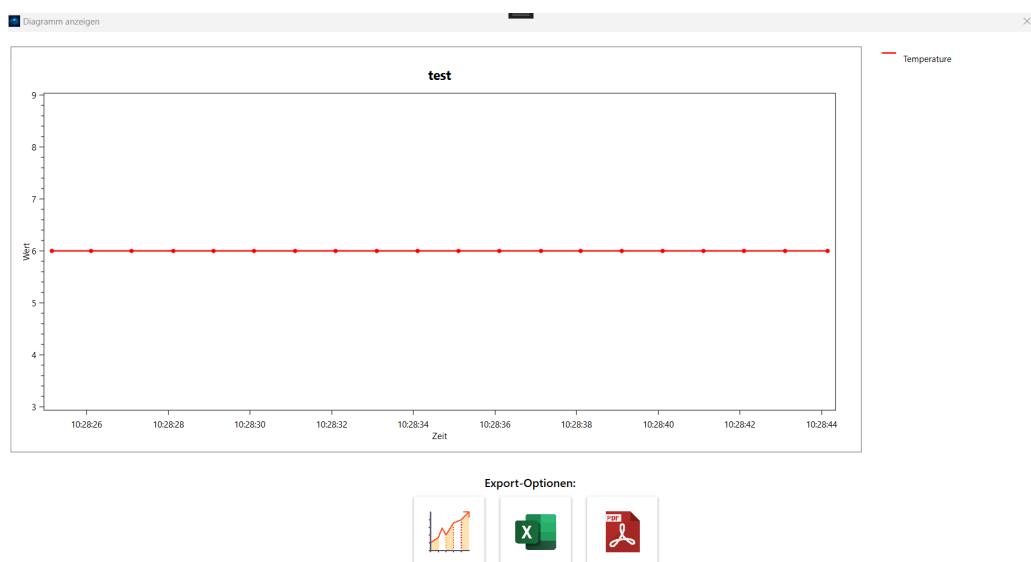


Abbildung 34: Visualisierung gespeicherter Daten als Diagramm

Analyse Konfigurationsfenster

Für die Datenanalyse können verschiedene Datenbankverbindungen, Tabellen und Variablen ausgewählt werden. Zudem lassen sich Analysearten wie Median oder Durchschnitt bestimmen und ein Zeitbereich festlegen, um gezielte Berechnungen durchzuführen.

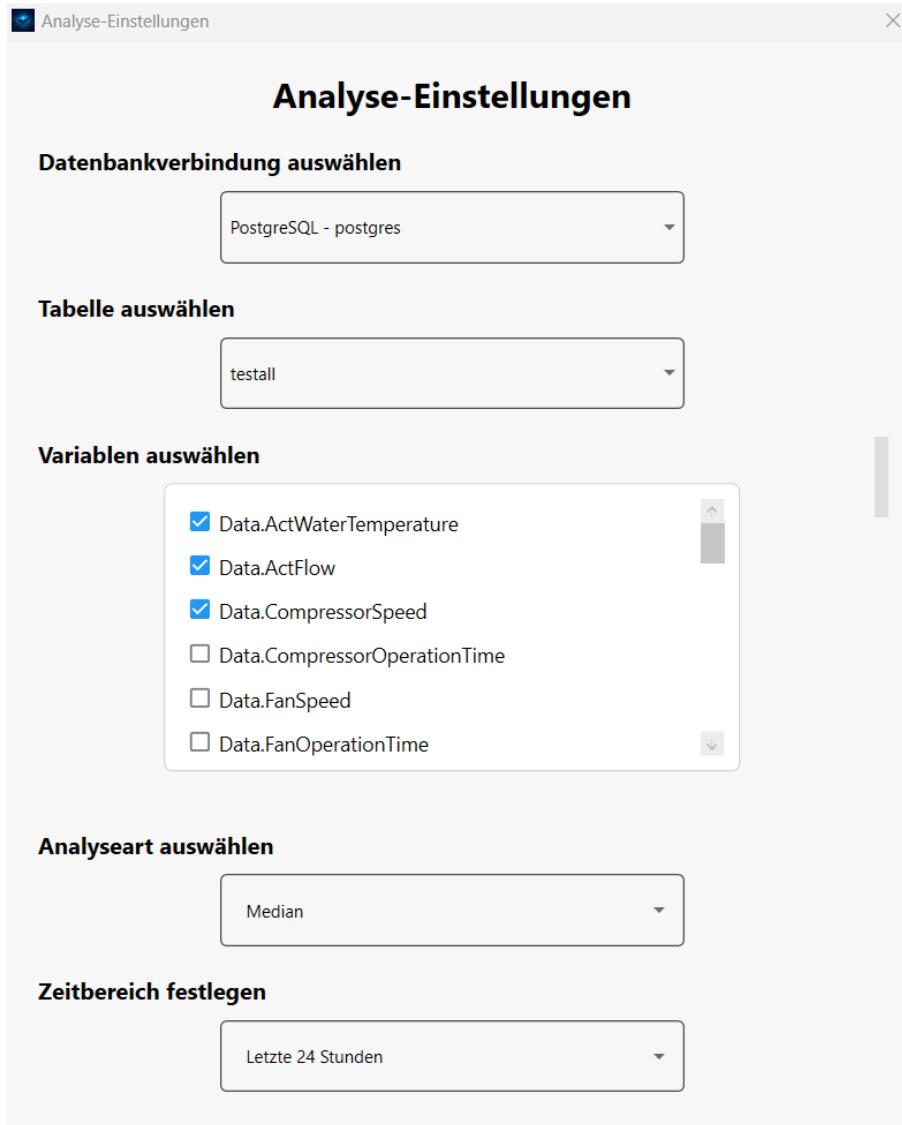


Abbildung 35: Konfigurationsfenster für die Datenanalyse

Daten Analyse Manager

Die gespeicherten Analysen werden in einer Übersicht angezeigt und können später erneut betrachtet oder gelöscht werden. Jede Analyse enthält Angaben zur Datenbank, Tabelle, Erstellungszeitpunkt und Analyseart.

The screenshot shows a dark-themed application window titled "Daten Analyse". At the top right is a blue "Neue Analyse erstellen" button with a white arrow icon. Below it is a dropdown menu labeled "Analyseart auswählen:" with "Alle" selected. The main area is titled "Gespeicherte Analysen" and lists four entries:

- Name:** test
Datenbankname: postgres
Tabellenname: testtemperature
Erstellt am: 20.02.2025 10:31:56
Analyseart: Durchschnitt
Buttons: Anzeigen, Löschen
- Name:** test1
Datenbankname: postgres
Tabellenname: testtemperature
Erstellt am: 20.02.2025 10:32:20
Analyseart: Trend-Analyse
Buttons: Anzeigen, Löschen
- Name:** test2
Datenbankname: postgres
Tabellenname: testtemperature
Erstellt am: 20.02.2025 10:32:53
Analyseart: Median
Buttons: Anzeigen, Löschen
- Name:** testhh
Datenbankname: postgres
Tabellenname: testall
Erstellt am: 20.02.2025 14:23:21
Analyseart: Korrelation
Buttons: Anzeigen, Löschen

Abbildung 36: Übersicht gespeicherter Analysen

Analyseergebnisse



Abbildung 37: Darstellung der Analyseergebnisse

Protokolle

Die Protokollseite bietet eine Übersicht über Systemereignisse, Fehler und Aktionen innerhalb der Anwendung. Einträge lassen sich nach Datum sortieren und gezielt durchsuchen.

The screenshot shows the 'Protokolle' (Protocols) page. On the left is a vertical sidebar with icons for Home, Network, Databases, Power, Data, Reports, Help, and Tools. The main area is titled 'Protokolle' and contains a search bar with 'Sortieren:' dropdown set to 'Neuste zuerst' (Newest first), a search input field 'Suchbegriffe eingeben...', and a 'Suchen' button. Below is a table with columns 'Datum' (Date), 'Uhrzeit' (Time), and 'Nachricht' (Message). The table lists five log entries:

Datum	Uhrzeit	Nachricht
04.03.2025	12:06:06	Datenbank :SQL Fehler beim Abrufen der Datenbanken: Fehler bei der...
04.03.2025	12:06:06	Datenbank :Fehler :Verbindungsfehler bei der Verbindung zu MSSQL ...
04.03.2025	12:05:33	OPC-UA: Verbindung fehlgeschlagen: 12
04.03.2025	12:02:18	DatenExport: Das Diagramm'1' wurde erfolgreich erstellt.
04.03.2025	11:46:18	DatenExport: Das Diagramm'1' wurde erfolgreich erstellt.

At the bottom are buttons for 'Ausgewählte Logs exportieren' (Export selected logs) and 'Logs löschen' (Delete logs), along with a page navigation indicator '1 / 1'.

Abbildung 38: Protokollansicht für System-Logs.

Hilfefenster

Das Hilfefenster bietet eine strukturierte Übersicht über die verschiedenen Funktionen der Anwendung. Durch Anklicken einer der Funktionsbereiche wird eine detaillierte Erklärung angezeigt, die dem Nutzer hilft, die jeweilige Funktion besser zu verstehen.

The screenshot shows the 'Hilfe' (Help) window with a close button 'X'. It contains six cards with icons and labels:

- OPC-UA Verbindungen (OPC-UA Connections)
- Datenbank Verbindungen (Database Connections)
- Trigger Management (Trigger Management)
- Tabellenbasierte Anzeige (Table-based Display)
- Diagrammbasierte Anzeige (Diagram-based Display)
- Datenanalyse (Data Analysis)

Abbildung 39: Hilfefenster für detaillierte Anleitungen.

Einstellungsfenster

Im Einstellungsfenster können grundlegende Konfigurationen der Anwendung vorgenommen werden. Dazu gehören die Sprachauswahl, die Option zum automatischen Start beim Hochfahren von Windows, sowie eine Update-Prüfung zur Überprüfung auf neue Versionen. Zudem enthält das Fenster einen Über-uns-Bereich mit Informationen zur verwendeten Bibliotheken und Lizzenzen.

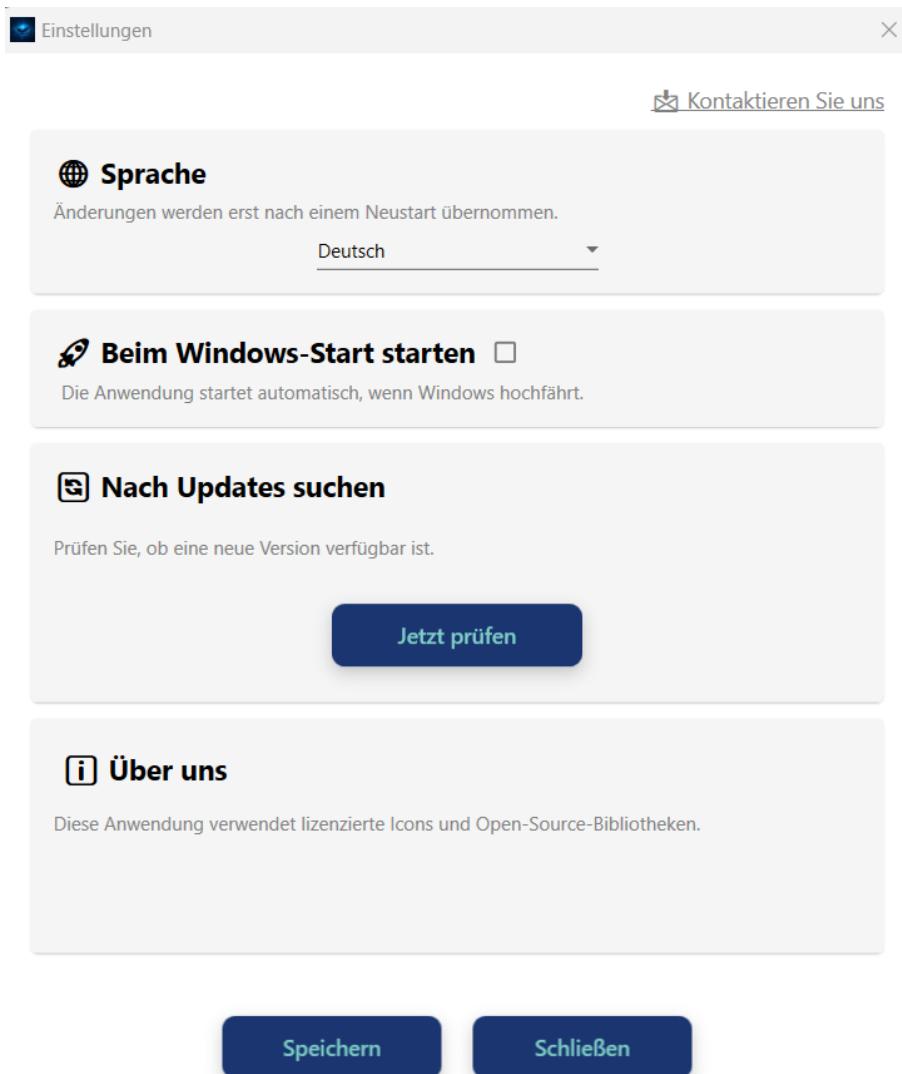


Abbildung 40: Einstellungsfenster der Anwendung

7 Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde die Software UACortex für die Firma Schiele Maschinenbau GmbH entwickelt. UACortex ist eine Desktopanwendung zur Erfassung, Speicherung, Analyse und Visualisierung von SPS-Daten. Der Name setzt sich aus OPC UA, einem etablierten Kommunikationsstandard in der industriellen Automatisierung, und Cortex, der äußeren Schicht des Gehirns, zusammen – ein Symbol für die intelligente Verarbeitung von Steuerungsdaten. Ziel der Software war es, eine zuverlässige und flexible Datenkommunikation zwischen SPS und Datenbanksystemen zu ermöglichen, um Prozesswerte effizient zu erfassen, zu speichern und auszuwerten.

Die Anwendung wurde auf Grundlage klar definierter funktionaler Anforderungen entwickelt. Sie stellt eine sichere OPC-UA-Verbindung zur SPS her und erkennt Netzwerkänderungen automatisch. Über die Benutzeroberfläche können Anwender SPS-Variablen durchsuchen und in Echtzeit auslesen. Zur Datenspeicherung wurde eine flexible Datenbankanbindung integriert, die eine Speicherung von Variablen in relationale Datenbanken wie MSSQL, MySQL oder PostgreSQL ermöglicht. Dabei können die Werte über zeit- oder ereignisgesteuerte Trigger automatisch gespeichert werden. Darüber hinaus kann die Anwendung Variablenwerte direkt in Excel-Dateien schreiben, ebenfalls basierend auf Triggern. Der Nutzer hat dabei die Wahl, ob die Werte in einzelne Zellen, ganze Zeilen oder Datenblöcke geschrieben werden.

Für eine übersichtliche Darstellung der gespeicherten Daten bietet UACortex sowohl eine tabellarische Ansicht als auch eine Visualisierung in Diagrammen, um Entwicklungen und Zusammenhänge besser zu erkennen. Zusätzlich wurde eine Analysefunktion integriert, die es ermöglicht, verschiedene Variablen miteinander zu vergleichen, Grenzwertüberschreitungen zu analysieren und langfristige Trends nachzuvollziehen. Die gesamte Benutzeroberfläche wurde bewusst funktional gehalten, um eine klare Navigation und eine intuitive Nutzung zu ermöglichen. Echtzeit-Statusanzeigen, Tooltips und eine integrierte Hilfefunktion unterstützen den Anwender bei der Bedienung.

Die durchgeführten Tests haben gezeigt, dass die Software ihre Kernfunktionen zuverlässig umsetzt und stabil in verschiedenen Umgebungen läuft. In der ersten Testphase mit Simulationsservern konnte bestätigt werden, dass die Verbindung zu OPC-UA-Servern problemlos aufgebaut wurde, Variablen korrekt ausgelesen und Daten sowohl in Datenbanken als auch in Excel-Dateien gespeichert wurden. Auch die Visualisierung der Daten in Tabellen und Diagrammen funktionierte, wobei Optimierungen notwendig waren, um die Performance bei großen Datenmengen zu verbessern.

Beim Test mit einer realen SPS traten zusätzliche Herausforderungen auf. Die Browsing-Funktion für SPS-Variablen musste angepasst werden, da die reale SPS eine andere Struktur als die Simulationsserver verwendete. Auch die Sicherheitsmechanismen der SPS erforderten Anpassungen, um eine stabile Verbindung zu gewährleisten. Trotz dieser Hürden konnte die Software nach den Optimierungen auch in der realen Umgebung erfolgreich betrieben werden.

Ein weiteres wichtiges Ergebnis war die Optimierung der Ladeprozesse und Datenanalyse. Während der Tests stellte sich heraus, dass die Verarbeitung großer Datenmengen zu Verzögerungen in der Benutzeroberfläche führen konnte. Durch die Implementierung asynchroner Berechnungen, eine verbesserte Datenaggregation und ein optimiertes Session-Management für OPC-UA-Verbindungen konnten diese Probleme behoben werden.

Beim Design der Benutzeroberfläche lag der Fokus bewusst auf der stabilen Funktionalität, nicht auf einem ausgefeilten visuellen Design. Dennoch wurde Wert auf eine einfache Bedienung, klare Strukturen und eine intuitive Navigation gelegt. Durch integrierte Protokoll- und Hilfefunktionen wird sichergestellt, dass Nutzer Probleme schnell identifizieren und lösen können.

Obwohl UACortex die gestellten Anforderungen erfüllt, gibt es mehrere sinnvolle Erweiterungen, die die Anwendung noch leistungsfähiger machen könnten. Eine Möglichkeit wäre die Erweiterung der Analysefunktionen, indem zusätzliche statistische Methoden und KI-gestützte Algorithmen integriert werden. Dadurch ließen sich nicht nur Trends besser erkennen, sondern auch Vorhersagen auf Basis historischer Daten treffen. Besonders durch maschinelles Lernen könnten Anomalien frühzeitig erkannt und entsprechende Maßnahmen eingeleitet werden.

Ein anderer wichtiger Schritt wäre die Echtzeit-Datenverarbeitung. Die Einführung einer Live-Analyse würde es ermöglichen, kritische Zustände sofort zu erkennen und darauf zu reagieren, anstatt nur mit historischen Daten zu arbeiten. Dies könnte insbesondere für Predictive Maintenance oder die Überwachung von Produktionsprozessen ein großer Vorteil sein.

Auch die Benutzeroberfläche könnte weiter verbessert werden. Während die aktuelle GUI funktional und auf die wichtigsten Aufgaben optimiert ist, könnte eine modernere und flexiblere Gestaltung die Benutzerfreundlichkeit noch weiter steigern. Dazu gehören eine bessere Anpassung an verschiedene Bildschirmgrößen, eine dynamischere Darstellung von Daten sowie eine intuitivere Bedienung durch interaktive Elemente.

Schließlich wäre eine webbasierte Version von UACortex eine sinnvolle Erweiterung. Eine browserbasierte Oberfläche würde es ermöglichen, die Anwendung plattformunabhängig und ortsunabhängig zu nutzen. Dies wäre besonders für Unternehmen mit mehreren Standorten oder für Fernüberwachungsanwendungen von Vorteil.

Diese Erweiterungen könnten UACortex noch leistungsfähiger und vielseitiger machen, sodass die Anwendung auch für anspruchsvollere industrielle Anforderungen und moderne datengetriebenen Prozesse optimal eingesetzt werden kann.

Literatur

- [1] Katharina Juschkat. Was ist eine sps?definition, grundlagen und funktion. <https://www.elekrotechnik.vogel.de/was-ist-eine-sps-definition-grundlagen-und-funktion-a-ac7e836b2194b3cac340ceddf92838d8/>, August 2024. Zugegriffen: 29. Januar 2025.
- [2] Simpleclub. Sps (speicherprogrammierbare steuerung). <https://simpleclub.com/lessons/mechatronikerin-sps-speicherprogrammierbare-steuerung/>. Zugegriffen: 14. November 2024.
- [3] International Electrotechnical Commission. *IEC 61131-3: Programmable Controllers – Part 3: Programming Languages*. IEC, Geneva, Switzerland, 2013. Third Edition.
- [4] Wikipedia. Opc unified architecture. https://en.wikipedia.org/wiki/OPC_Unified_Architecture?utm_, August 2024. Zugegriffen: 28. Januar 2025.
- [5] Anthony King Ho. Understanding the opc unified architecture (opc ua) protocol. <https://control.com/technical-articles/understanding-the-opc-ua-protocol/>, April 2023. Zugegriffen: 29. Januar 2025.
- [6] OPC-Router. Was ist opc ua? die wichtigsten begriffe im Überblick. <https://www.opc-router.de/was-ist-opc-ua/>. Zugegriffen: 29. Januar 2025.
- [7] OPC-Foundation. Security. <https://opcfoundation.org/security/>, Januar 2025. Zugegriffen: 30. Januar 2025.
- [8] OPC-Foundation. Opc ua security architecture. <https://reference.opcfoundation.org/Core/Part2/v104/docs/4.5>. Zugegriffen: 30. Januar 2025.
- [9] OPC-Foundation. Opc ua connection protocol. <https://reference.opcfoundation.org/Core/Part6/v104/docs/7>. Zugegriffen: 30. Januar 2025.
- [10] Siemens AG. Simatic s7-1500 getting started. Online verfügbar unter https://cache.industry.siemens.com/dl/files/272/71704272/att_92234/v1/s71500_getting_started_de-DE_de-DE.pdf, 2021. Zugegriffen: 30. Januar 2025.
- [11] Siemens AG. Opc ua .net client für den simatic s7-1500 opc ua server. Online verfügbar unter https://cache.industry.siemens.com/dl/files/901/109737901/att_1116715/v2/109737901_OPCT_UA_Client_S7-1500_DOKU_V1_5_de.pdf, 2022. Zugegriffen: 25. Februar 2025.