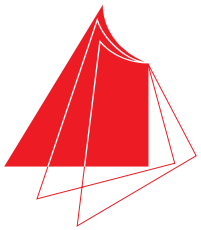


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Entwurf einer Architektur und eines Proof of Concept für ein echtzeitfähiges SCADA System mit Webfrontend

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Engineering

vorgelegt von

Florian Weber (44907)

12.11.2019

Erstprüfer: Prof. Dr.-Ing. Philipp Nenninger
Zweitprüfer: Prof. Dr. Stefan Ritter

Inhaltsverzeichnis

Abkürzungsverzeichnis

PoC Proof of Concept

SCADA Supervisory Control And Data Acquisition

HTTPS Hypertext Transfer Protocol Secure

WSS WebSocket Secure

OPC UA OPC Unified Architecture

HTML Hypertext Markup Language

JS JavaScript

CSS Cascading Style Sheets

SQL Structured Query Language

GUI Graphical User Interface

PLC Programmable Logic Controller

ERD Entity Relationship Diagram

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Durch mein Studium der Elektrotechnik mit Vertiefung in die Automatisierungstechnik konnte ich Eindrücke in die Vorgehensweise und Möglichkeiten der Graphical User Interface (GUI) Programmierung für Industrieanlagen gewinnen. Dabei fiel mir auf, dass der aktuelle Stand der Technik in der Automatisierungstechnik noch stark vom Stand in anderen softwaregeprägten Bereichen abweicht. So wird in der Automatisierungstechnik noch immer auf statisch geschriebene Benutzeroberflächen gesetzt, die kompilliert werden müssen und damit viele Einschränkungen mit sich bringen. Es ist zum Beispiel nicht möglich ein Steuerelement zur Laufzeit in abhängigkeit vorhandener Entitäten zu instanzieren. Meist schafft man sich Abhilfe indem man ein Element entweder ein- oder ausblendet. Ein weiteres Problem vorhandener Lösungen ist, dass diese meist plattformgebungen sind und nur lokal im Netz, mit entsprechender Software des Herstellers lauffähig sind. In der heutigen Informatik wird immer mehr auf grafische Benutzeroberflächen gesetzt, welche als Webapplikation in einem beliebigen Browser lauffähig sind.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist der Entwurf einer Architektur für ein echtzeitfähiges SCADA System mit Webfrontend. Durch ein Proof of Concept (PoC) wird herausgefunden ob die Architektur auch in die Praxis umsetzbar ist. Hierbei wird die Applikation streng getrennt in Frontend und Backend. Das Frontend wird als Webapplikation im PoC implementiert. Dabei werden folgende Anforderungen an die Architektur gestellt:

- Die Datenrate des Frontends soll bei vertretbarem Aufwand so klein wie möglich sein. Dies ermöglicht die Nutzung des Systems in einem Umfeld mit geringer verfügbarer Bandbreite zur Steuerungsebene.

- Die Architektur soll Steuerelemente unterstützen, die eine Eingabe durch den Nutzer zulassen, sowie Steuerelemente die eine Darstellung eines Prozesswerts ermöglichen.

- Die Webapplikation selbst soll so modular sein, dass man zur Laufzeit Steuerelemente hinzufügen und entfernen kann, ohne dass das System offline geht.
- Die Prozessdaten sollen nicht, wie aktuell bei vielen Webapplikationen üblich, durch Polling synchronisiert werden, sondern die Weboberfläche soll auf Datenänderungen des Prozesses asynchron in Echtzeit (bei statischem Routing im Netzwerk) reagieren. Dasselbe gilt für die Eingaben des Nutzers.
- Die Architektur soll ein herstellerunabhängige Schnittstelle zur Integration in ein vorhandenes System bereitstellen.
- Die Weboberfläche soll eine feste Auflösung haben und muss nicht auf Änderungen des Viewports reagieren. Ausnahmen bilden hierbei Darstellungen die dies, durch ihre einfache Gestalt erlauben.

Der Beweis der Realisierbarkeit soll durch eine Beispielimplementierung (PoC) erbracht werden. Dabei wird je ein Eingabeelement (z.B. Button), ein Ausgabeelement (z.B. Label), sowie ein Ein-/Ausgabeelement (zum Beispiel ein Textfeld) implementiert.

1.3 Gliederung

2 Theoretische Grundlagen

2.1 SCADA

2.2 Verteilte Systeme

2.2.1 Synchrones Datenmodell

2.2.2 Asynchrones Datenmodell

2.3 Security

2.3.1 Authentifizierung

2.3.2 Verschlüsselung

2.4 Safety

2.5 Websocket

2.6 Datenbanken

2.6.1 Relationale Datenbank

Unter einer Datenbank versteht man einen Server der Tabellen mit Datensätzen speichert und im Netzwerk zur Verfügung stellt. Eine Datenbank ermöglicht den schnellen Zugriff auf Datensätze und ermöglicht die Verknüpfung dieser Daten. Jeder Datensatz in einer Tabelle muss eindeutig durch einen Primärschlüssel (primaryKey) zu erreichen sein. Dieser Schlüssel kann aus einer oder mehreren Spalten einer Tabelle definiert werden. Außerdem bietet die Datenbank die Möglichkeit, weitere Schlüssel sogenannte *unique Keys*, zu definieren welche wieder einzigartig für jeden Datensatz innerhalb einer Tabelle sein müssen.
Die dritte Art von Key, ist der *foreignKey*.
Dieser Key wird nicht von jeder Datenbankengine unterstützt und ermöglicht die Beschreibung von Relationen, direkt in der Datenbank. So Kann man zum Beispiel eine Tabelle mit Risiken (*RiskRisks*) haben, sowie eine Tabelle mit Lebensphasen des Produkts (*RiskLifephases*). Jede Tabelle enthält als Primärschlüssel

eine Spalte des Typs *unsigned int* mit dem Namen *ID*. Außerdem enthalten sie Spalten, die Informationen zu den Lebensphasen und Risiken beschreiben. Eine dritte Tabelle (*RiskDocument*) repräsentiert die fertigen Dokumente. Sie einhält wieder einen ID Spalte als Primärschlüssel sowie eine ComponentID, RiskID sowie LifephaseID Spalte. Letztere sind sind vom selben Typ (*unsigned int*)wie die Primärschlüssel der Tabellen *Components* (im Releasemanagement), *RiskRisks* sowie *RiskLifephases*. Dies ermöglicht es, diese Spalten jeweils als foreignKey auf die jeweiligen Tabellen zu definieren und somit die Relation ausreichend zu beschreiben, sodass die Datenbank nun selbständig ihre Integrietät hält. Genau dieses Verhalten wünscht man sich von einer relationalen Datenbank. Wenn man zum Beispiel versuchen würde eine Lebensphase zu löschen, prüft die Datenbank zuerst, ob nicht eine andere Tabelle, die einen foreignKey auf die Lebensphasen Tabelle hält (zum Beispiel die Tabelle *RiskDocument*), einen Datensatz beinhaltet welcher eine Referenz auf die zu löschende Lebensphase darstellt. Wenn dies zutreffen würde, gäbe es 2 Möglichkeiten wie die Datenbank darauf reagieren könnte.

- Die Datenbank löscht die Lebensphase nicht und reagiert auf die Anfrage mit einer Fehlermeldung
- Die Datenbank löscht alle Datensätze die auf die zu löschende Lebensphase referenzieren

Welche dieser 2 Aktionen ausgeführt werden, kann man beim Erstellen des foreignKey definieren. Jedoch bleibt in beiden Fällen die Integrietät erhalten und es gibt danach keine Referenzen auf Datensätze die nicht mehr existieren.

2.6.2 SQL

SQL ist eine Abkürzung für Structured Query Language. Dabei handelt es sich um eine Sprache, welche das Erzeugen und Verwalten von Datenstrukturen ermöglicht. Außerdem ermöglicht sie das Abfragen, Einfügen, Verändern, sowie Verknüpfen von Datensätzen. Dabei unterscheidet sich SQL sehr von anderen Programmiersprachen. So besteht der Ansatz bei SQL eher darin, dass man eher definiert was man gerne als Erbegnis hätte, sich aber um die konkrete Implementierung der Operation, keinerlei Gedanken machen braucht. Um dies zu demonstrieren ist das folgende einfache Beispiel gegeben. Wie in Abbildung 2.1 zu sehen, besteht die Datenbank *Personal* aus zwei Tabellen. Die erste Tabelle (*Abteilung*) mit den Spalten *ID*, *Bezeichnung* sowie Standort. Die Spalte *ID* ist als primaryKey deklariert. Die zweite Tabelle (*Mitarbeiter*) mit den Spalten *Personalnummer*, *Vorname*, *Nachname*, *Telefonnummer*, *EmailAdresse*, *AbteilungID*. Nun möchte man gerne alle Telefonnummern, Namen und E-mail Adressen einer Abteilung haben, welche den Namen *HW-Entwicklung trägt*. Der Query dafür ist in Abbildung 2.2 abgebildet. Die Antwort des Servers ist in Abbildung 2.1

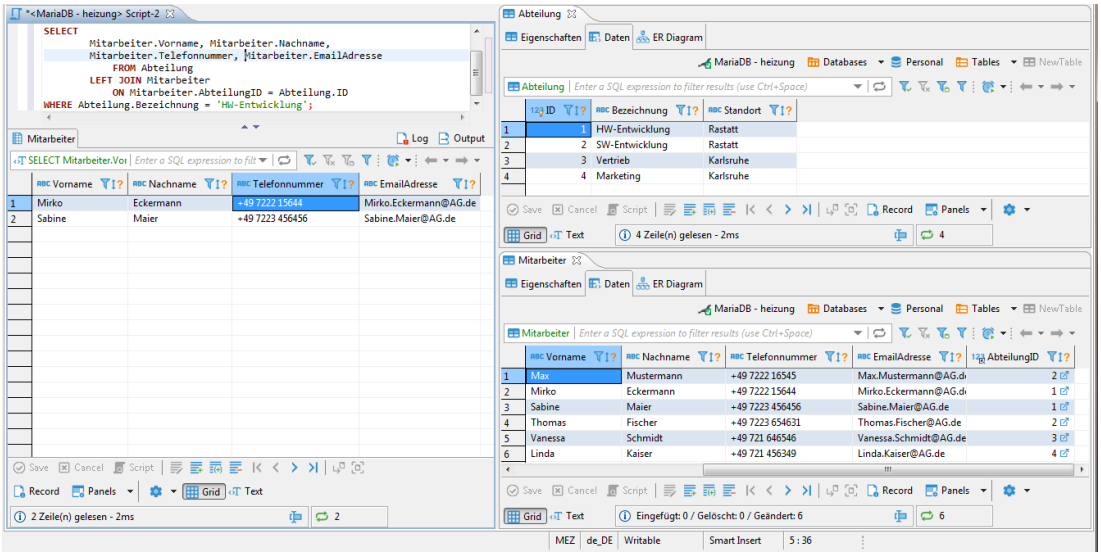


Abbildung 2.1: Beispiel Datenstruktur

```
SELECT
    Mitarbeiter.Vorname, Mitarbeiter.Nachname,
    Mitarbeiter.Telefonnummer, Mitarbeiter.EmailAdresse
      FROM Abteilung
    LEFT JOIN Mitarbeiter
      ON Mitarbeiter.AbteilungID = Abteilung.ID
WHERE Abteilung.Bezeichnung = 'HW-Entwicklung';
```

Abbildung 2.2: Beispiel sqlQuery - Select mit Join

unten links zu sehen. Rechts seht man die beiden Quelltabellen des Queries. Wahrscheinlich würde man diesen einfachen Datenbank Join, in C/C++, mit den Daten in Structures gespeichert, mittels verschachtelter For-Schleifen implementieren. Dass Problem bei dieser Implementierung ist, man muss durch jedes Element itterieren. Die Datenbank hat zur Lösung dieses Problems bessere Algorithmen hinterlegt (Stichwort Binärer Suchbaum). Außerdem hat man mit einem Sql-Server als Backend den Vorteil, dass die Applikation ohne weiteren Aufwand auch Multiuserended sein kann ohne wirklich mehr Code zu schreiben.

2.6.3 Stored Procedurs

Ein Sql-Server unterstützt nicht nur das Manipulieren von Daten durch eine Client-Anwendung, sondern auch das Speichern von Funktionen und Prozeduren,

welche in SQL geschrieben sind. Der wesentliche Unterschied zwischen einer Prozedur und einer Funktion besteht darin, dass eine Funktion einen Rückgabewert haben kann, eine Prozedur dagegen nicht. Das Fehlen eines Rückgabewerts einer Prozedur stellt aber, entgegen der allgemeinen Erwartung, kein Handicap dar. Prozeduren und Funktionen akzeptieren nähmlich auch sessionbezogene globale Variablen als *OUT* Argument. Desweiteren haben Prozeduren entgegen Funktionen die Möglichkeit, SQL-Queries zur Laufzeit zusammenzusetzen und auszuführen.

2.7 Echtzeitfähigkeit

2.8 OPCUA

3 Aktueller Stand der Technik

lsg vetablierter hersteller

3.1 Gui zeug auto

3.2 transport zeug

3.3 datenmanagement

4 Systementwurf

4.1 Use-Cases

4.1.1 Frontend

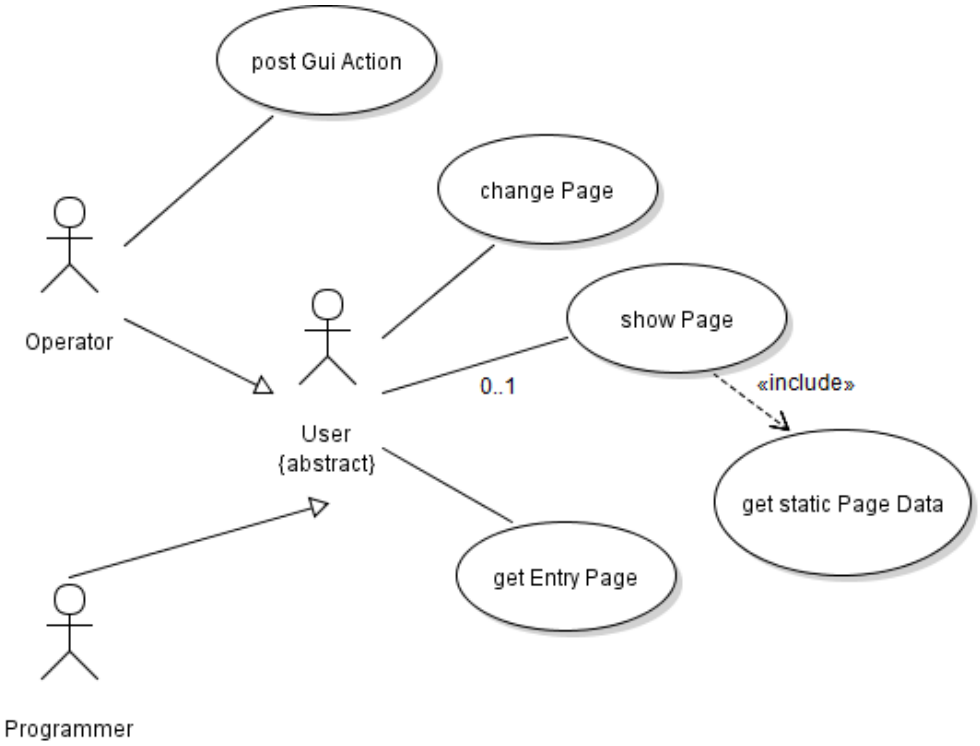


Abbildung 4.1: Use-Case Diagramm des Frontends

Aus der Zielsetzung der Arbeit ergibt sich das Use-Case Diagramm in Abbildung 4.2. Das Diagramm beschreibt abstrakt die Anwendungsfälle die das Supervisory Control And Data Acquisition (SCADA) System für den Anwender in der Rolle des Bedieners (Operator), sowie des Programmierers (Programmer) erfüllen muss.

4.1.2 Backend

Aus der Zielsetzung der Arbeit ergibt sich das Use-Case Diagramm für das Backend in Abbildung 4.2.

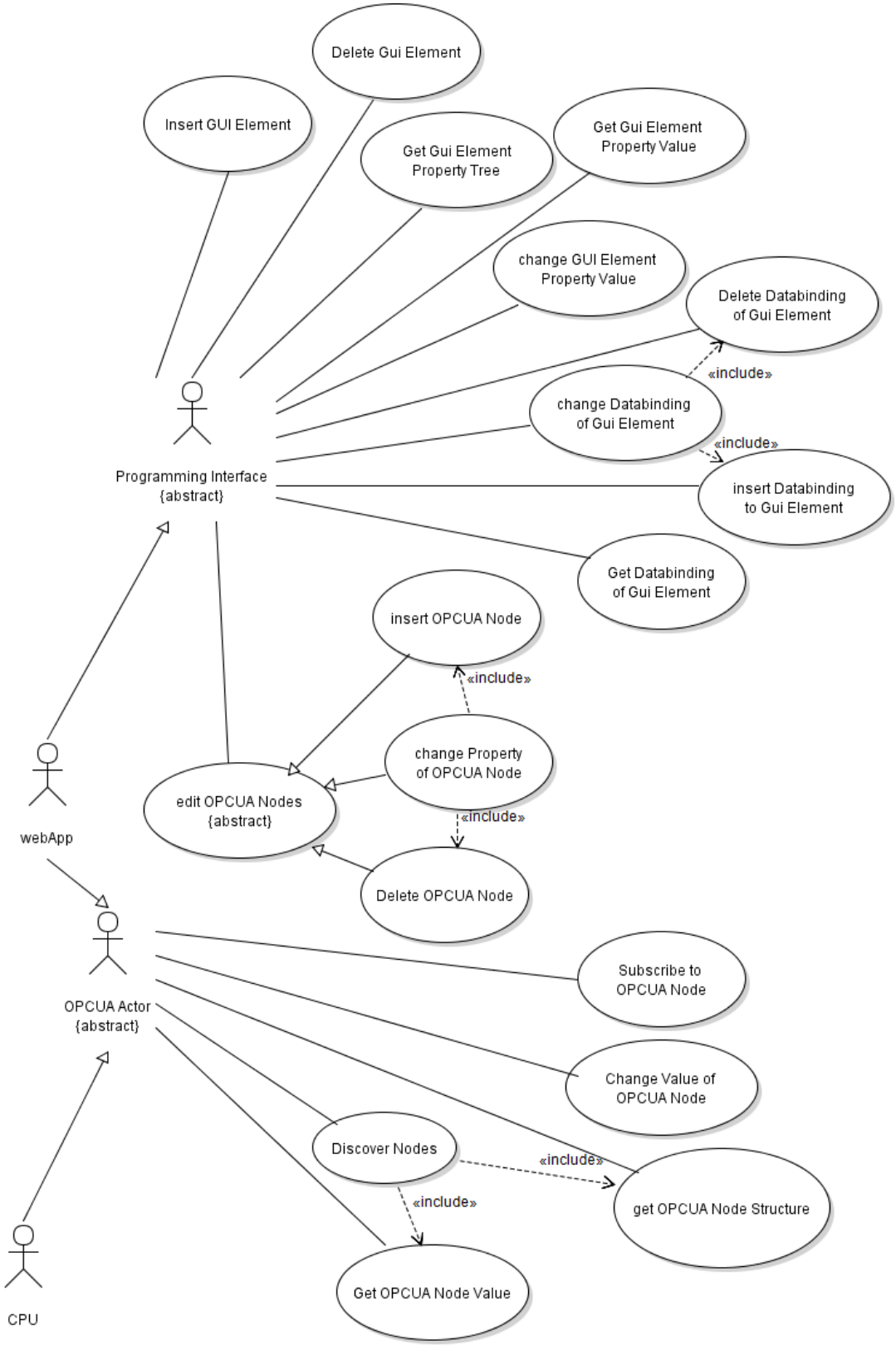


Abbildung 4.2

4.2 Architektur

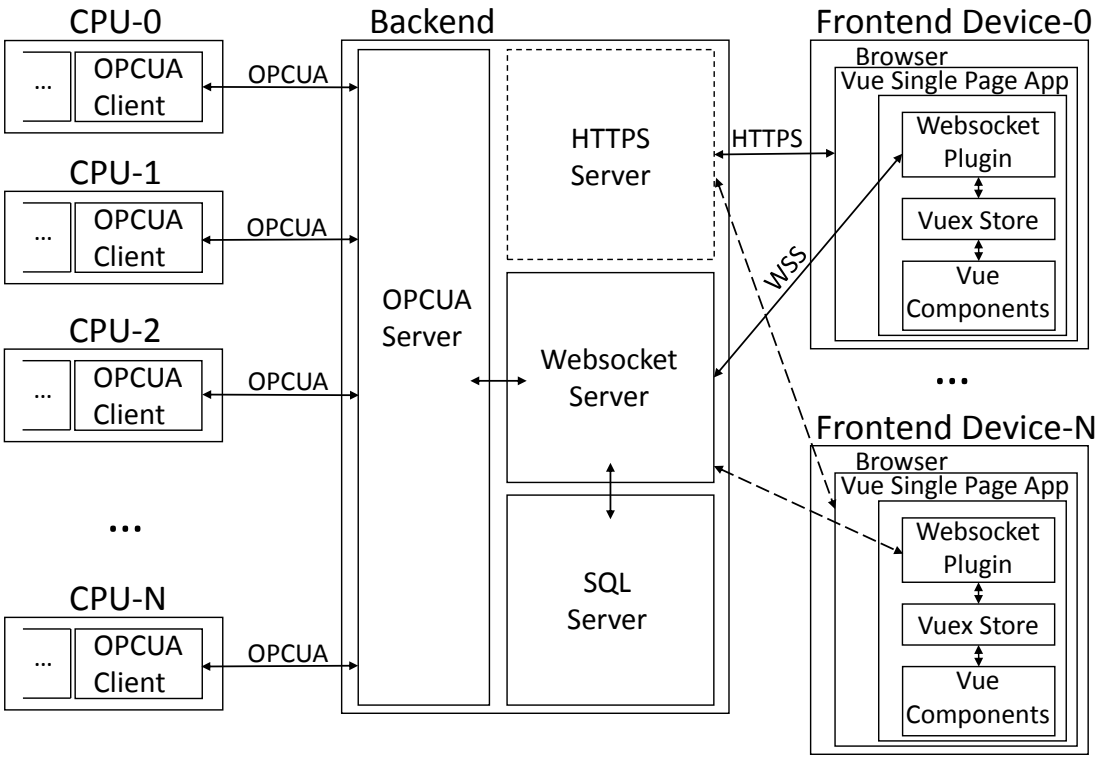


Abbildung 4.3: Kommunikationsmodell des SCADA Systems.
Die CPUs sind per OPC UA an das Scada System angebunden und haben die Rolle eines OPC UA Clients. Die Webapplikation im Browser kommuniziert über WSS sowie HTTPS verschlüsselt mit dem Backend.

Wie in Abbildung 4.3 dargestellt, lässt sich die Architektur des SCADA Systems in Programmable Logic Controllers (PLCs) Backend und Frontend unterteilen. Unter Frontend (Abschnitt 4.2.2)versteht man die (in diesem Fall) grafische Schnittstelle, die es dem Nutzer ermöglicht mit dem System zu interagieren. Das Backend (Abschnitt 4.2.1) fasst den Rest des Systems zusammen.

4.2.1 Backend

Das Backend besteht aus den folgenden vier Komponenten:

- Einem Hypertext Transfer Protocol Secure (HTTPS) Server
- Einem WebSocket Secure (WSS) Server
- Einem Structured Query Language (SQL) Server

- Einem OPC Unified Architecture (OPC UA) Server

Damit ergeben sich, wenn man das Backend von außen betrachtet, als Schnittstellen OPC UA, WSS und HTTPS. Über diese Schnittstellen stellt das Backend den PLCs sowie dem Frontend, Services zur Verfügung. Eine Sonderolle unter den Komponenten nimmt dabei der HTTPS Server ein, da er als einzigste keine Verbindung mit dem Rest des Backends hat. Dies ist deshalb möglich, da der HTTPS Server dazu da ist, die Hypertext Markup Language (HTML), die Cascading Style Sheets (CSS), sowie die JavaScript (JS) Dokumente einmalig beim Seitenaufruf an das Endgerät (Frontend Host Device) auszuliefern. Von diesem Zeitpunkt ab findet die Kommunikation zwischen Frontend und Backend ausschließlich über den WSS Server statt. Zwischen dem Websocket Server und dem Frontend können von da an Strings ausgetauscht werden. Der OPC UA Server hält die variablen Daten des Systems und bietet diese den angeschlossenen PLCs an. Die Parametrierung des Frontends sowie alle anderen Daten die persistiert werden müssen, werden in einem SQL Server in Form einer relationalen Datenbank gespeichert.

4.2.2 Frontend

Das Frontend ist eine HTML Webapplikation, ...

4.2.3 Protokoll zwischen Frontend und Backend

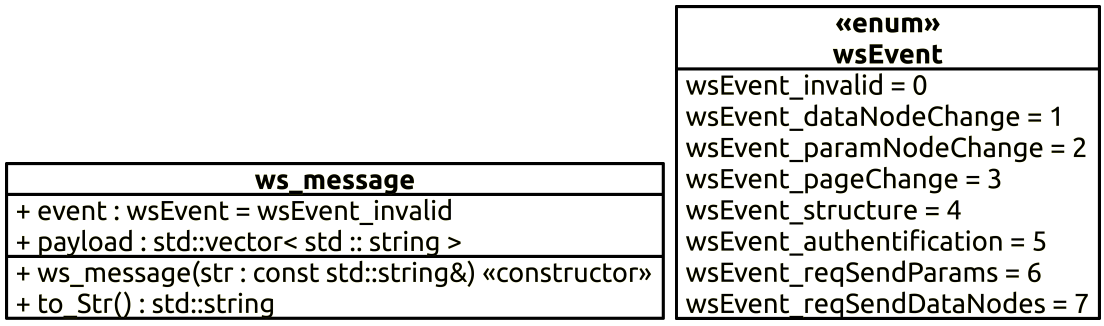
Websockets bieten die Möglichkeit asynchron Daten zwischen Frontend und Backend auszutauschen, bieten allerdings keinerlei Regeln für die Semantik dieser Daten. Desshalb ist ein weiteres Protokoll in der Applikationsebene erforderlich, welches die Semantik der ausgetauschten Daten definiert. An dieses Protokoll werden folgende Anforderungen gestellt:

- Das Protokoll soll so einfach wie möglich sein
- Das Protokoll soll alle Datentypen die das Backend definiert unterstützen
- Das Protokoll soll weitestgehend symetrisch sein (Die Nachrichten vom Server zum Client haben eine ähnliche Gestalt wie die vom Client zum Server)
- Das Protokoll soll möglichst leichtgewichtig sein
- Das Protokoll soll möglichst frei von Zuständen sein, sodass ein Vertauschen zweier Nachrichten keine Relevanz für die Applikation darstellt
- Das Protokoll soll eine Authentifizierung des Nutzers ermöglichen

Die meisten Anforderungen können erfüllt werden, allerdings stehen die letzten drei in Konkurenz zueinanader. Um eine Authentifizierung zu ermöglichen, hat man bei einer Verbindung die ab dem Aufbau dauerhaft besteht folgende Möglichkeiten:

- Die Zugangsdaten die die Session authentifizieren werden einmalig vom Client zum Server gesendet, die Session ist dann authentifiziert und hat ab diesem Punkt an entsprechend mehr Rechte.
- Die Zugangsdaten werden bei jeder ausgegangenen Nachricht mitgesendet und überprüft.

Die erste Variante ist möglichst leichtgewichtig da die Zugangsdaten nur einmal gesendet werden müssen. Allerdings ist sie nicht frei von Zuständen, da die Schnittstelle jenachdem ob eine Session bereits authentifiziert ist anderst auf eingehende Packete reagiert. Die zweite Variante reagiert immer identisch (deterministisch) auf das gleiche Packet, ist aber weniger performant, da das Verhältnis von Protokoll-Overhead zu Nutzdaten immer größer ist, als bei der ersten Variante. Aufgrund dieses Sachverhalts muss bei der Zustandsfreiheit eine Ausnahme gemacht werden. Das erarbeitete Protokoll tauscht Strings aus, dessen Struktur



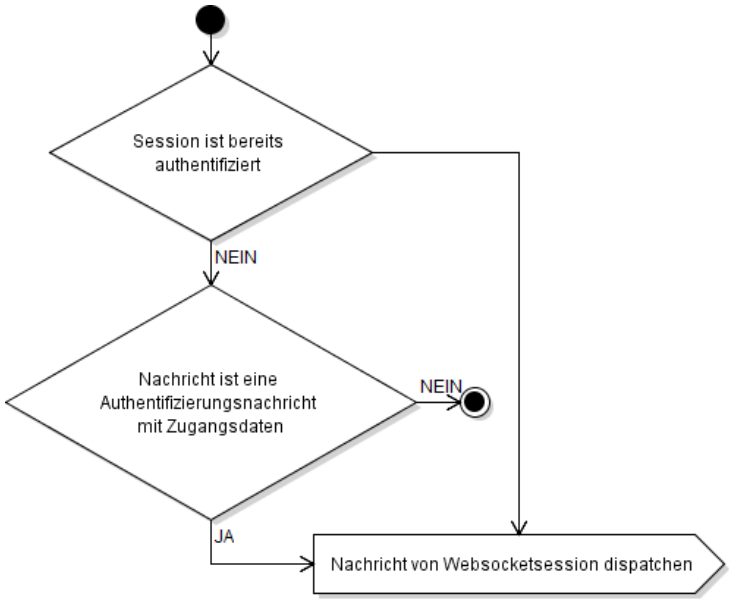
(a) Websocket Message Container mit *event* (Header) und *payload* (Nutzdaten)

(b) Websocket Event Definition als Aufzählung

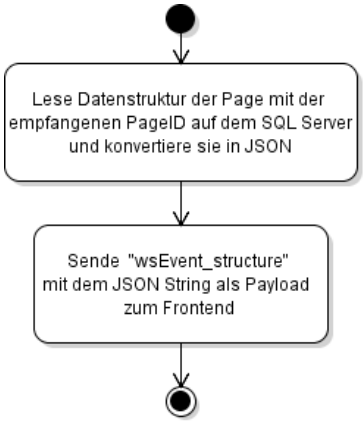
Abbildung 4.4: Websocket Message Container und Event

der Klasse in Abbildung 4.4a zugrunde liegt. Diese Klasse besteht aus Nutzdaten (*payload*), sowie einem Header (*event*). Nutzdaten sind nur optional vorhanden, so ist möglich nur ein Event zu verschicken. Der Header ist eine Aufzählung verschiedener Events. Diese Aufzählung (*wsEvent*) ist in Abbildung 4.4b dargestellt und gibt die einzelnen Typen von Nachrichten an. Ein solches Objekt kann seitens des Backends aus dem String in einer Websocket Nachricht konstruiert, sowie in einen solchen String übersetzt werden. Die Bildungsvorschrift ist dabei, alle Information welche das *ws_message* Objekt enthält (*event* und *payload*)als String zu kodieren und durch ein " ; " getrennt zu einem einzigen String zusammenzuführen. Dabei stellt das erste Feld in einem versendeten String immer das

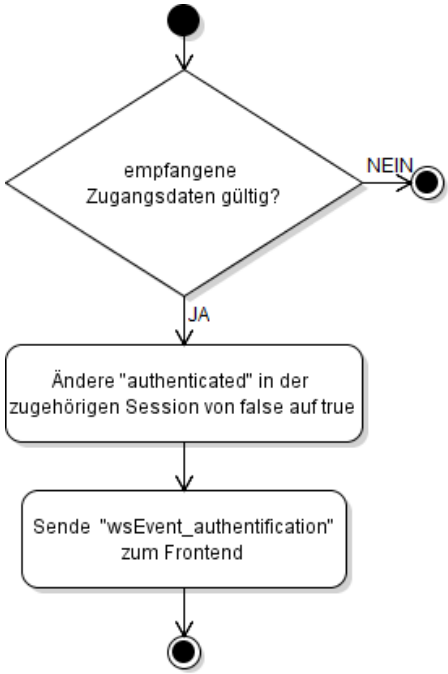
kodierte Event dar. Die restlichen Felder entsprechen dem Nutzdatenvektor der *ws_message*. Diese Bildungsvorschrift ist umkehrbar, sollange der Nutzdatenvektor kein String mit einem ";" enthält. Dieser Fall ist unbedingt zu vermeiden und muss vom Frontend und Backend beim kodieren des Strings abgefangen werden. Die umgekehrte Bildungsvorschrift implementiert die *ws_message* Klasse als Konstruktor. Das Protokoll unterstütz nun die in Abbildung 4.4b dargestellten Events. Das erste Event (*wsEvent_invalid*) ist der Standartwert des Events in jedem *ws_message* Objekt. Dies ist eine Absicherung um zu verhindern, dass eine Nachricht mit einem zufälligen Event versendet wird. Front- sowie Backend stellen einen Dispatcher zur Verfügung welcher, entsprechend des Events einer empfangen Nachricht, einen Handler aufruft. Der Dispacher des Backends unterscheidet sich vom Dispatcher im Frontend, da er die Authentifizierung des Nutzers realisiert. Das Verhalten ob und wie eine Nachricht vom Dispatcher im Backend dispatcht wird, ist durch das Aktivitätsdiagramm in Abbildung 4.5a definiert. Hier werden nur Nachrichten dispatcht wenn die Websocketsession bereits authentifiziert wurde oder die Nachricht eine Authentifizierungsnachricht ist. Ist dies der Fall werden die entsprechenden Handler zu den Events aufgerufen. Die Handler des Backends sind durch die verbleibende Diagramme in Abbildung 4.5, sowie durch die Diagramme in Abbildung 4.6 definiert.



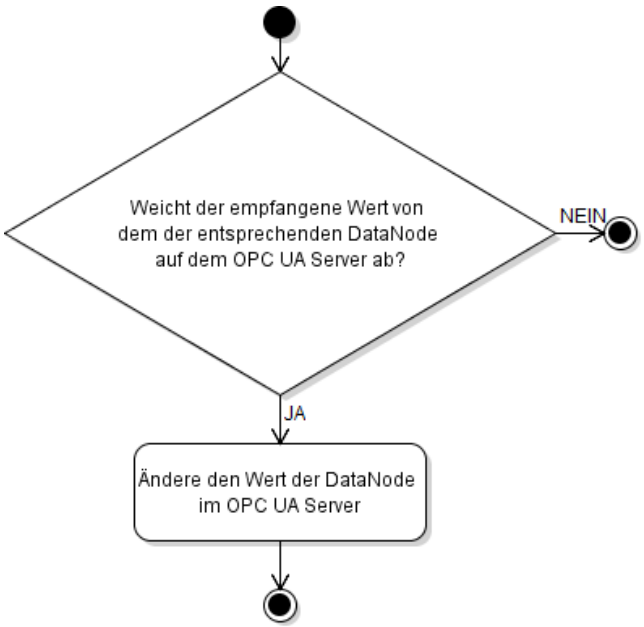
(a) WebsocketEvent empfangen



(b) WebsocketEvent: *Structure* empfangen

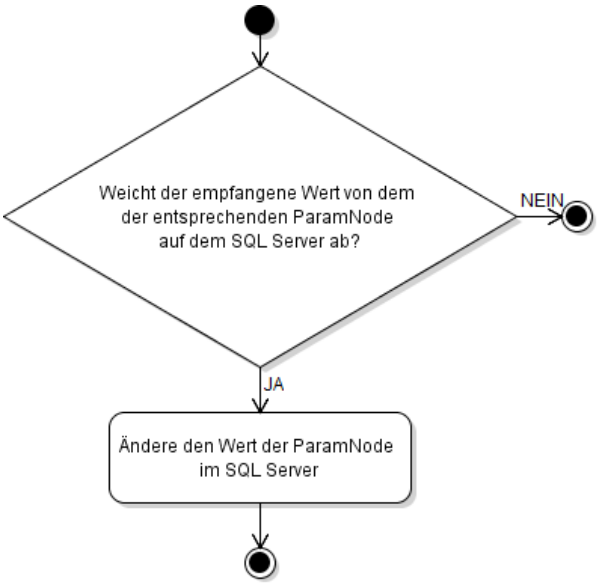


(c) WebsocketEvent: *Authentification* empfangen

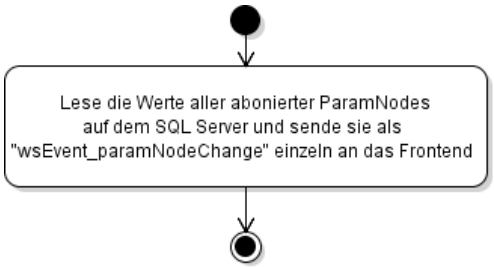


(d) WebsocketEvent: *DataNodeChange* empfangen

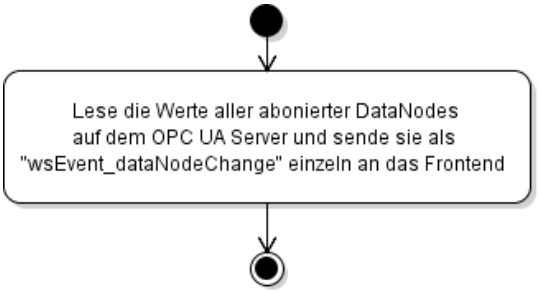
Abbildung 4.5: Aktivitätsdiagramme des Dispatchers im Backend I



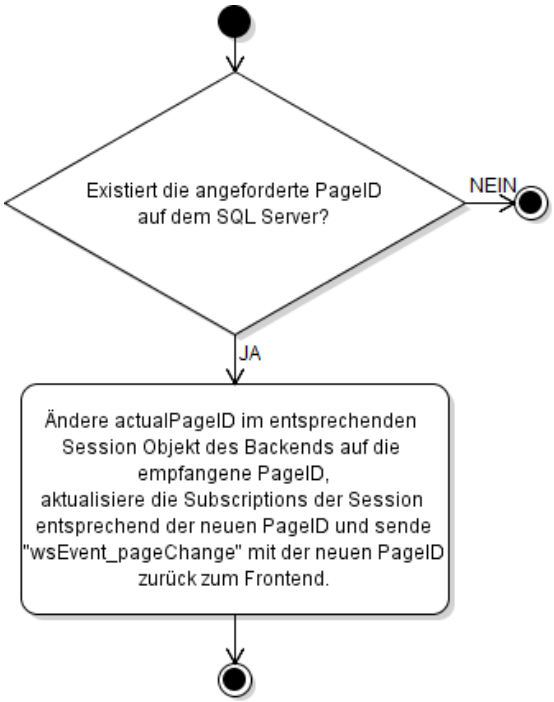
(a) WebsocketEvent: *ParamNodeChange* empfangen



(b) WebsocketEvent: *ReqSendParamNodes* empfangen

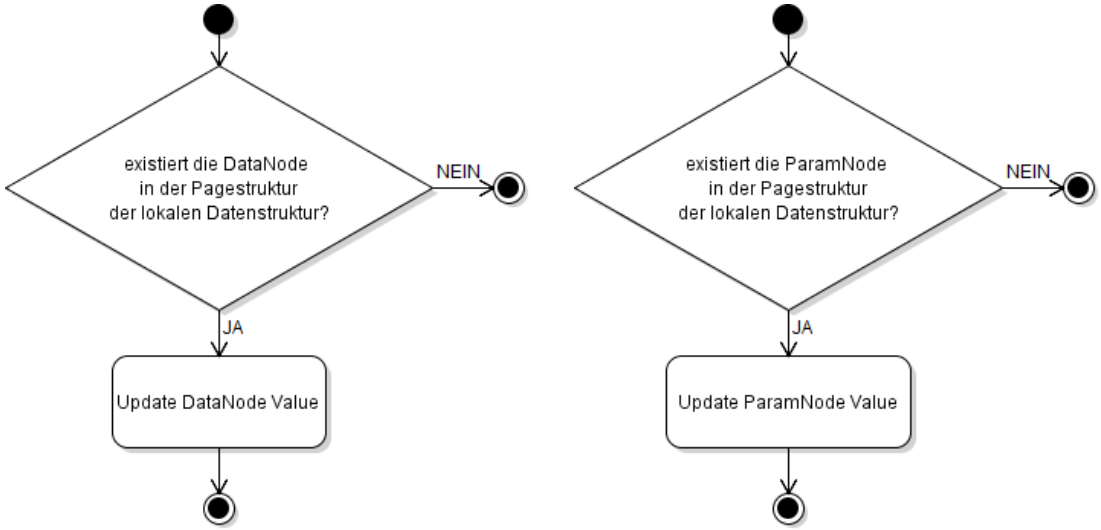


(c) WebsocketEvent: *ReqSendDataNodes* empfangen



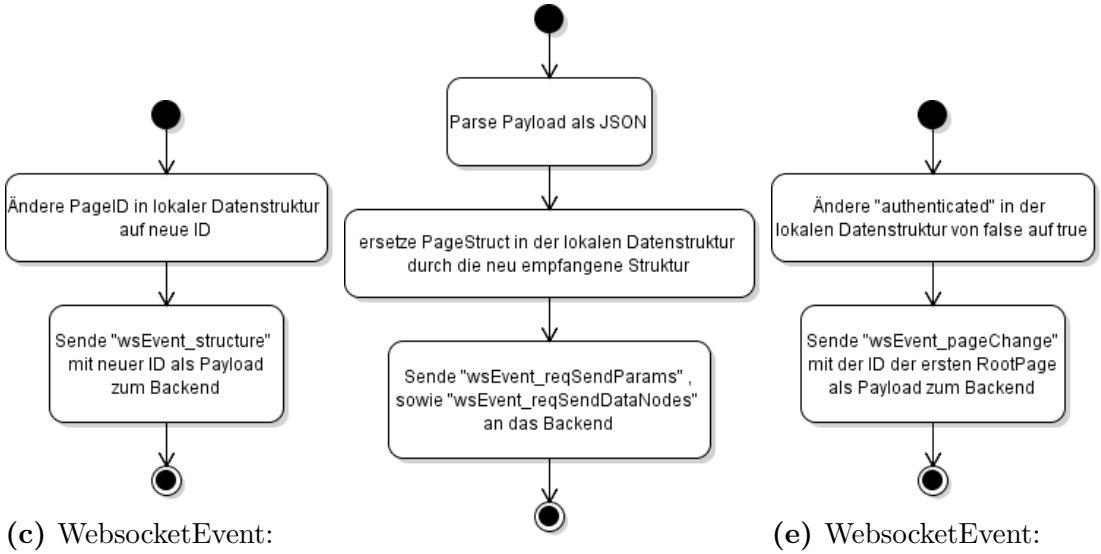
(d) WebsocketEvent: *PageChange* empfangen

Abbildung 4.6: Aktivitätsdiagramme des Dispatchers im Backend II



(a) WebsocketEvent: *DataNodeChange* empfangen

(b) WebsocketEvent: *ParamNodeChange* empfangen



(c) WebsocketEvent: *PageChange* empfangen

(d) WebsocketEvent: *Structure* empfangen

(e) WebsocketEvent: *Authentification* empfangen

Abbildung 4.7: Aktivitätsdiagramme des Dispatchers im Frontend

4.3 Datenstruktur

4.3.1 Backend

Das abgebildete Entity Relationship Diagram (ERD) in Abbildung 4.8, stellt die globale Datenstruktur der Datenbank auf dem SQL Server dar. Sie besteht aus einzelnen Tabellen deren Primärschlüssel ausgenommen einzelner Hilfstabellen immer eine ID ist. Zentrales Element sind dabei die GUI Elemente (Tabelle *GuiElements*). Jedes GUI Element ist einer Page (Tabelle *Pages*) zugeordnet. Eine Page kann wiederum einer anderen Page zugeordnet sein. Dies wird erreicht indem jeder Page Datensatz eine ParentID enthält, welche als Fremdschlüssel auf die übergeordnete Page zeigt. So entsteht eine Baumstruktur innerhalb der Entitäten. Ist der Wert dieses Fremdschlüssels *NULL* so hat die jeweilige Page keine übergeordnete Page und liegt damit im Wurzelverzeichnis.
Jedes GUI Element hat n Parameter und m Data Nodes (mit $n, m \in \mathbb{N}$). Diese werden in den Tabellen *GuiElementParams* sowie *GuiElementDataNodes* verwaltet. Jede DataNode, beziehungsweise jeder Parameter, hat einen Datentyp ein Namen, eine Beschreibung, sowie einen Wert. Bei den Parametern entspricht dies dem tatsächlichen aktuellen Wert, bei den DataNodes entspricht dies dem initialen Wert beim Starten des Backends. Die zuvor beschriebenen GUI Elemente besitzen außerdem einen Typ (verwaltet in der Tabelle *GuiElementTypes*). Jedem Typ sind nun n DataNodeTemplates (Tabelle *GuiElementDataNodeTemplates*) zugeordnet, aus denen die eigentlichen Datanodes beim Instanzieren erstellt werden. Da eine solche Vorlage für DataNodes auch für mehrere GUI Element Typen verwendet werden kann, ist die Verbindung eine n:m Bindung. Dies ist in einer Datenbank nur durch eine Hilfstabelle, welche die Primärschlüssel zweier Tabellen miteinander verknüpft, ereichbar. Analog zu den DataNodes ist der gleiche Mechanismus für die Parameter vorgesehen. Um ein sauberes Interface zu schaffen sind zum Instanzieren, sowie zum Löschen der GUI Elemente, Stored Procedures vorgesehen. Die komplette Datenstruktur des SQL Servers ist so abgelegt, dass er alle relevanten Relationen kennt und entsprechend verwaltet. So wird beispielsweise beim Löschen eines GUI Elements alle zugehörigen DataNodes mitgelöscht und beim Löschen einer kompletten Page werden alle der Page zugeordneten GUI Elemente auch gelöscht (*on delete cascade*, Abschnitt 2.6.1).

Der OPC UA Server hält alle DataNodes der GUI Elemente. Diese GUI Elemente werden entsprechen der Pages, denen sie zugeordnet sind, auf dem OPC UA Server abgelegt und sind den PLCs dort zugänglich. Ob eine DataNode von den PLCs nur gelesen oder auch beschrieben werden kann, wird durch das Flag *writePermission* auf dem SQL Server in der DataNodeTemplate Tabelle festgelegt. Beim Erstellen der DataNode auf dem OPC UA Server, wird dieses Flag ausgelesen und direkt in die Konfiguration des OPC UA Servers übernommen. Der

native Datentyp mit dem die DataNodes auf dem OPC UA Server abgelegt werden, wird den DataNodeTemplates des SQL Servers entnommen.
Auf dem SQL Server haben die DataNodes sowie Parameter immer den native SQL Datentyp *VARCHAR* um ein Speicherfeld zu erhalten, dass alle benutzten OPC UA Datentypen unterstützt. Beim Lesen beziehungsweise beim Schreiben wird als String gespeicherte wert auf dem SQL Server, entsprechend seines explizit abgespeicherten Typs, konvertiert. GuiElements und Pages haben als Datentyp auf dem OPC UA Server den Typ BaseObjectType. Auf einen Spezialisierung dieser Typdefinition für Pages und GUI Elements auf dem OPC UA Server wird absichtlich verzichtet, da sie nur zur Strukturierung der DataNodes genutzt werden.

4.3.2 Frontend

...

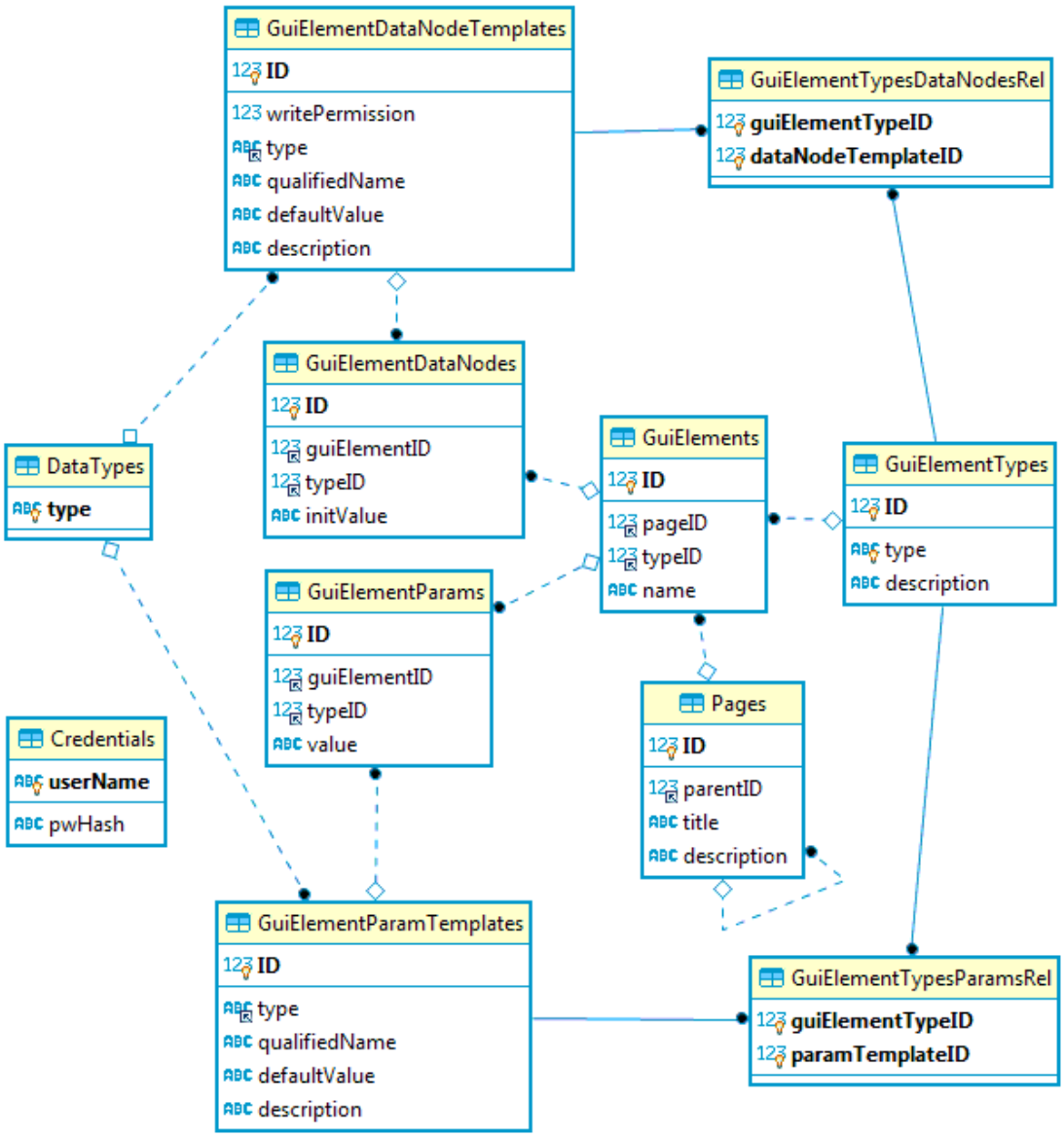


Abbildung 4.8: ERD des SCADA Systems.
Darstellung zeigt die einzelnen Tabellen der Datenbank auf dem SQL Server und deren Beziehungen zueinander.

5 Umsetzung des Proof of Concept

5.1 backend

5.1.1 HIER KLASSEN

5.2 Frontend

6 Zusammenfassung und Ausblick

1

[1] John Doe. „How do I get this to work?“ In: *TeX Monthly* 99 (1234), S. 1–10.

Literatur

Doe, John. „How do I get this to work?“ In: *TeX Monthly* 99 (1234), S. 1–10.