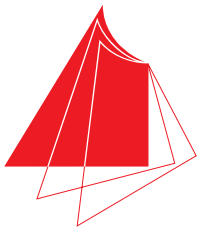


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

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

DBMS Datenbank Management System

crud Create, read, update and delete

TCP Transmission Control Protocol

CSMA/CD Carrier Sense Multiple Access/Collision Detection

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance

PoC Proof of Concept

SCADA Supervisory Control And Data Acquisition

HTTPS Hypertext Transfer Protocol Secure

WSS WebSocket Secure

OPC UA Open Platform Communications 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

CA Certification Authority

Abbildungsverzeichnis

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 verwendbar sind.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist der Entwurf einer Architektur für ein echtzeitfähiges Supervisory Control And Data Acquisition (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 in Frontend und Backend getrennt. 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 eine 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 (z.B. ein Textfeld) implementiert.

1.3 Gliederung

2 Theoretische Grundlagen

2.1 Security

2.1.1 Authentifizierung

2.1.2 Verschlüsselung

2.2 Websocket

2.3 Datenbanken

Der Zweck eines Datenbanksystems ist es Daten persistent zu speichern und zur verfügung zu stellen. Ein Datenbanksystem setzt sich aus einer Datenbasis, sowie einem
Datenbank Management System (DBMS) zusammen. Dabei stellt die Datenbasis den Speicher der Datenbank dar und das DBMS, das Programm durch das der Zugriff auf die Datenbasis geschieht. Das DBMS ist notwendig, um konkurierende Zugriffe auf die Datenbasis, von mehreren Nutzern zu Verwalten. [2] Meistens bietet das DBMS als Interface eine Netzwerkschnittstelle an (Server-Client Architektur), es gibt allerdings auch Datenbanken deren DBMS Teil der Applikation werden (z.B. SQLite).

2.3.1 Relationale Datenbank

Im Falle einer relationalen Datenbank werden Daten in Form von Relationen abgespeichert. Tabellen sind eine Darstellung von Relationen.
Jede Relation hat Attribute mit einem in der Relation einzigartigem Attributnamen. Die Domaine eines Attributs ist die Menge aller Werte die ein Attribut annehmen kann. Ist der Wert eines Attributs unbekannt oder noch nicht bestimmt, ist der Wert des Attributs „NULL".[3] Werden Daten nun in einer Datenbank gespeichert, wird ein Wertetupel in eine Relation eingefügt. Die Relationen sind nur das strukturelle theoretische Konzept, das sich hinter einer Relationenalen Datenbank verbirgt. Anfangs dieses Abschnitts wurde gesagt dass Relationenn in Form einer Tabelle darstellbar sind. Wenn den Relationen nun zur Ihrer Definition Wertetabellen zu Grunde liegen, ist es möglich zur Vereinfachung statt von Relationen, von Tabellen zu sprechen. Das DBMS stellt eine Schnittstelle bereit um

dem Nutzer die Daten zugänglich (Create, read, update and delete (crud) Operationen) zu machen. Die Schnittstelle unterstützt meist die Sprache Structured Query Language (SQL). Diese Sprache ist in Abschnitt 2.3.2 genauer beschrieben. Relationen können nicht nur als Tabelle ausgegeben werden, es ist auch möglich sie zu Verknüpfen. Wenn keine Regel angegeben wird wie zwei Relationen Verknüpft werden sollen, wird das informatische Kreuzprodukt gebildet. Das Bedeutet, dass jedes Wertetuple aus Relation A, mit jedem Wertetupel aus Relation B verknüft wird. Dadurch entsteht eine neue Realation. In einer Datenbank existieren immer Schlüssel (Keys) innerhalb einer Tabelle. Dabei gibt es drei Typen:

- Primary-Key (Primärschlüssel)
- Unique-Key
- Foreign-Key (Fremdschlüssel)

Der Primary-Key muss als einzigster Schlüssel zwingend vorhanden sein. Er bestimmt nach was das DBMS die Datensätze ablegt. Desshalb muss der Wert des Attributes das den Krimary-Key bildet innerhalb einer Tabelle einzigartig sein. Das heist es darf maximal ein Wertetupel innerhalb einer Relation existieren, mit dem selben Wert des Primärschlüssels. Der Primärschlüssel darf auch nicht „NULL" sein. Ein Unique-Key stellt ein zusätzlichen Schlüssel dar, über die jeder Eintrag in eine Tabell eindeutig identifizierbar ist. Wie auch der Primärschlüssel, darf dieser Schlüssel nicht „NULL" sein. Er ist in jeder Tabelle nur optional vorhanden und kann auch mehrfach vorkommen. Der Foreign-Key (Fremdschlüssel) ist ein Schlüssel der nur in einer relationalen DBMS existiert. Er ermöglicht es, einen Verweises auf eine andere Tabelle zu definieren. Zur Veranschaulichung sei folgendes Beispiel einer relationalen Datenstruktur, auf einem SQL Server gegeben. Wie in Abbildung 2.1 zu sehen, besteht die Datenbank *Personal* aus zwei Tabellen. Die erste Tabelle hat die Bezeichnung (*Abteilung*) mit den Spalten *ID*, *Bezeichnung* sowie Standort. Die Spalte *ID* ist als primaryKey deklariert. Die zweite Tabelle (*Mitarbeiter*) enthält die Spalten *Personalnummer*, *Vorname*, *Nachname*, *Telefonnummer*, *EmailAdresse*, *AbteilungID*. In dieser Tabelle ist die Personalnummer der Primärschlüssel. Um eine Verknüpfung der Spalte *AbteilungID* der Tabelle *Mitarbeiter* zu der Tabelle *Abteilung* herzustellen, wird ein Fremschlüssel definiert, der von der Spalte *AbteilungID* der Tabelle *Mitarbeiter* auf die Tabelle *Abteilung*, unter Benutzung deren Primärschlüssels, verweist. Die Benutzung eines solchen Frewmdschlüssels ist nicht zwingend notwendig um die Verknüfung nach dieser Regel zu ermöglichen, Es ermöglicht aber die selbstständige Erhaltung der Datenbank der referenziellen Integrietät. Dabei hat man bei der Erstellung des Fremdschlüssels die Möglichkeit zu entschieden, was beim Löschen, oder beim Ändern, eines referenzierten Datensatzes geschehen soll. Der Gängigste Umgang damit ist, dass man entweder das Löschen verbietet (*on delete restrict*,

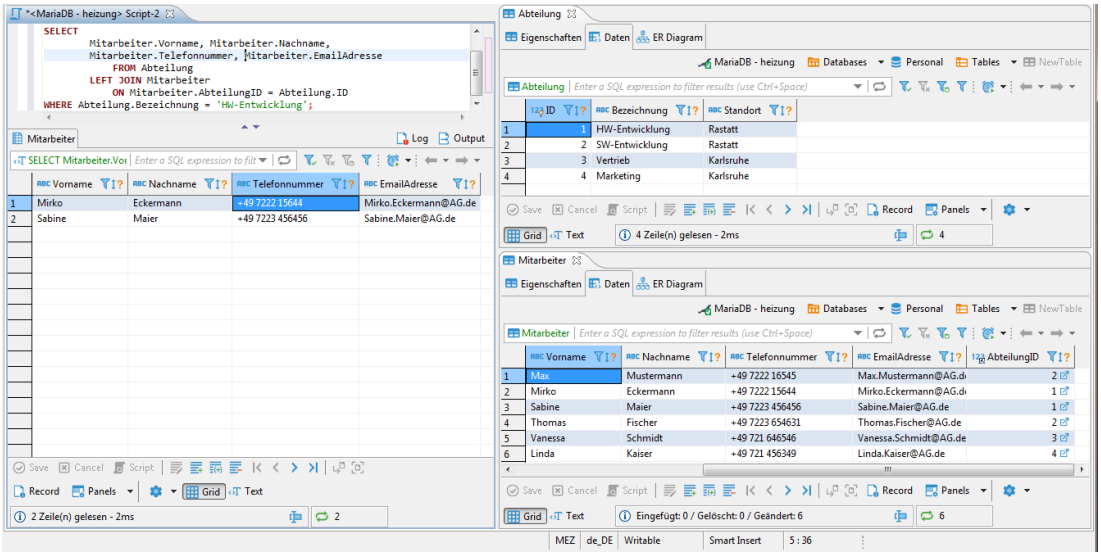


Abbildung 2.1: Beispiel Datenstruktur

oder das beim Löschen des referenzierten Datensatzes auf den referenzierenden Datensatz löscht (*on delete cascade*). Bei beiden Verfahren bleibt die referenzielle Integrietät erhalten, dass heist es gibt nach dem Löschen keine Referenzen die nicht aufgelöst werden können.

2.3.2 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 zu definieren, was man als Ergebnis möchte, sich aber um die konkrete Implementierung der Operation keinerlei Gedanken machen braucht. Um dies zu demonstrieren ist das folgende Beispiel, auf Basis der Beispieldatenbank in Abbildung 2.1, gegeben. Nun möchte man alle Telefonnummern, Namen und E-Mail Adressen einer Abteilung haben, welche den Namen *HW-Entwicklung* trägt. Dazu ist es notwendig die Datensätze der Mitarbeitertabelle mit den Datensätzen der Abteilungstabelle, entsprechend des Fremsschlüssels in der Mitarbeiter Tabelle, zu kombinieren und alle Datensätze der so entstanden Tabelle auszugeben, welche die Abteilungsbezeichnung *HW-Entwicklung* tragen. Die Query für diese Operation ist in Abbildung 2.2 abgebildet. Die Antwort des Servers ist in Abbildung 2.1 unten links zu sehen. Rechts seht man die Datensätze der beiden Quelltabellen des Queries. Wahrscheinlich würde man diesen einfachen Datenbank Join, in C/C++, mit den Daten in Structures gespeichert, mittels verschachtelter For-Schleifen imple-

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

Abbildung 2.2: Beispiel sqlQuery - Select mit Join

mentieren. 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, Hash-Maps...).

2.3.3 Stored Procedurs

Ein SQL Server unterstützt nicht nur das Manipulieren und Ausgeben von Daten mit SQL, sondern auch das Speichern von Funktionen und Prozeduren. Diese Prozeduren und Funktionen sind auch in SQL geschrieben. Sie ermöglichen dem Entwickler das Auslagern komplexer SQL Querys. 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, keine Einschränkung dar. Prozeduren und Funktionen akzeptieren auch sessionbezogene globale Variablen als *OUT* Argument. Desweiteren haben Prozeduren entgegen Funktionen die Möglichkeit, SQL-Queries zur Laufzeit zusammenzusetzen und auszuführen. Funktionenn können, in SQL Querys benutzt werden, Prozeduren nicht. Prozesuren werden durch ein *CALL* Befehl aufgerufen.

2.4 OPC UA

Open Platform Communications Unified Architecture (OPC UA) ist laut Der OPC UA Standart ist in der IEC 62541 definiert.

2.5 Echtzeit

Laut Peter Scholz ist Echtzeit wie folgt definiert:
„Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich

zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen." [1]
Das bedeutet, dass ein System nur dann echtzeitfähig ist wenn es die folgenden Bedingungen erfüllt:

- Jede Komponente des Systems die an dem Prozess beteiligt ist, muss selbst echtzeitfähig sein.
- Die Rechenzeit der verwendeten Algorithmen muss endlich sein.
- Die verwendenten Transportprotokolle um dem Prozess Daten zuzuführen bzw. Daten zu entnehmen müssen die Echtzeitfähigkeit unterstützen.

Wenn die obere Definion nun auf das Ethernet oder das Transmission Control Protocol (TCP) Protokoll angewand wird, fällt schnell auf, dass diese beide normalerweise nicht echtzeitfähig sind. Ethernet verhindert auf dem BUS (physikalischer Layer) keine Kollistionen sondern erkennt diese nur. Wenn eine Kollision erkannt wurde, wird das Senden auf den BUS abgebrochen und nach einer zufälligen Zeit erneut versucht. Dieses Verfahren nennt nennt man Carrier Sense Multiple Access/Collision Detection (CSMA/CD). Wer den Zugriff auf den Bus bekommt ist Zufall. Deshalb ist nicht garatiert, dass man innerhalb einer definierten Zeitspanne, die Daten übertragen kann. Man kann dieses Problem allerdings umgehen indem man im physikalischen Layer bereits dafür sorgt, dass keine Datenkollision auftreten kann. Dies ist zum Beispiel durch den Einsatz von Switches möglich. Diese trennen die einzelnen Sender voneinander und verhindern damit gezielt Kollisionen. Man muss anmerken man immernoch nicht vollständig Echtzeitfähig Daten übertragen kann, denn das Netzwerk muss auch auf die aufkommende Datenlast ausgelegt sein. Dadurch ist es möglich mit Ethernet in einer kontrollierten Umgebung Daten in Echtzeit zu übertragen. Ein weitverbreiteter Irrtum ist, dass Echtzeitfähigkeit bedeutet etwas müsse besonders schnell reagieren, es genügt dass die Zeit definiert und endlich ist.

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

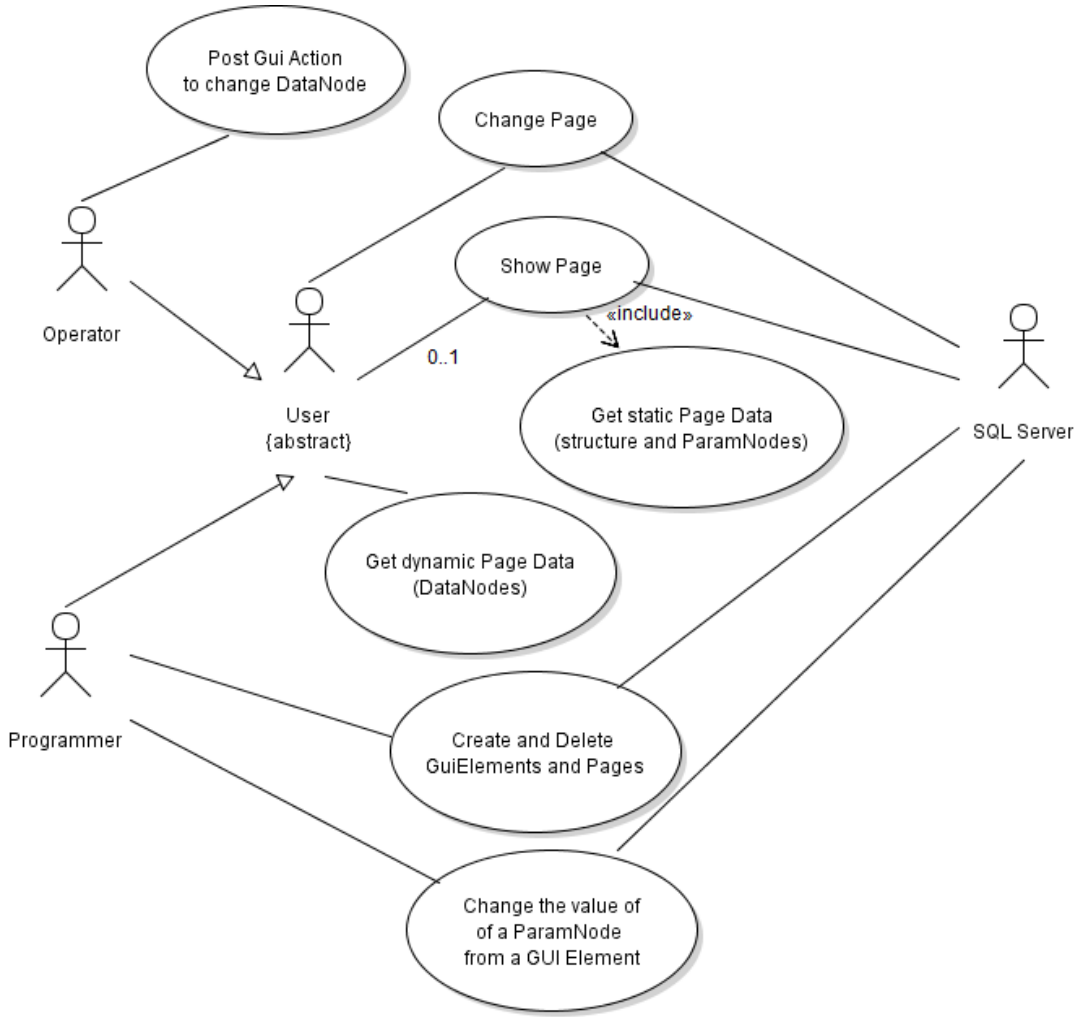


Abbildung 4.1: Anwendungsfalldiagramm des SCADA Systems I

Aus der Zielsetzung der Arbeit ergeben sich die Anwendungsfalldiagramme in Abbildungen 4.1 und 4.2. Das Diagramm in Abbildung 4.1 beschreibt abstrakt die Anwendungsfälle, die das SCADA System für den Anwender, in der Rolle des Bedieners (Operator), sowie des Programmierers (Programmer) erfüllen muss. Analog dazu beschreibt das Anwendungsfalldiagramm in Abbildung 4.2 die Anwendungsfälle die ein Programmable Logic Controller (PLC) an das SCADA System stellt. Es ist möglich dies in zwei getrennten Diagrammen abzuhandeln, da es keine Anwendungsfälle gibt, die einen Akteur aus beiden Diagrammen benötigt. Jedoch interagieren alle Akteure mit dem selben System.

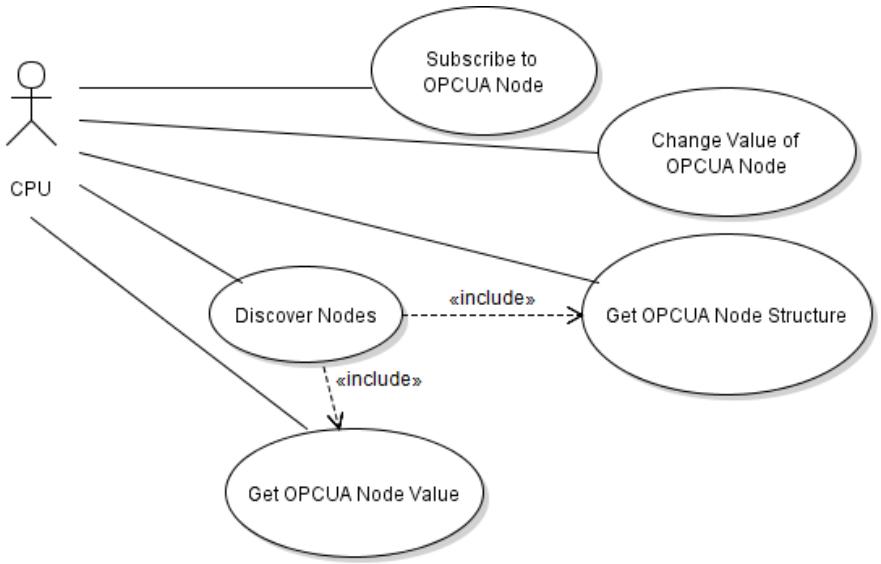


Abbildung 4.2: Use-Case Diagramm des SCADA Systems II

4.2 Architektur

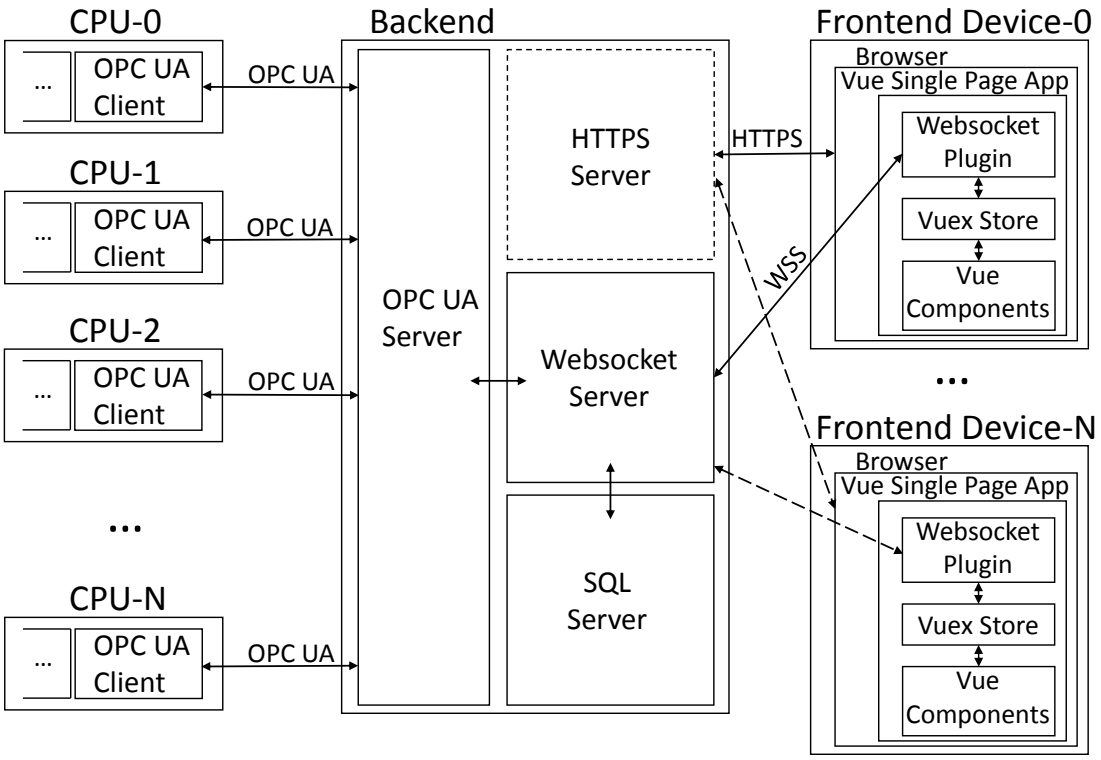


Abbildung 4.3: Kommunikationsmodell des SCADA Systems.
Die CPUs sind per OPC UA an das Scada System angebunden und haben die Rolle eines OPC UA Clients. Die Webapplikation im Browser kommuniziert über WSS sowie HTTPS verschlüsselt mit dem Backend.

Wie in Abbildung 4.3 dargestellt, lässt sich die Architektur des SCADA Systems in PLCs, Backend und Frontends 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 SQL Server

- Einem 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 Sonderrolle unter den Komponenten nimmt dabei der HTTPS Server ein, da er als einzige Komponente 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 an findet die Kommunikation zwischen Frontend und Backend ausschließlich über den WSS Server statt. Dabei exist immer genau eine Session für jede Frontendinstanz.Zwischen dem Websocket Server und dem Frontend können ab diesem Zeitpunkt 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 Webapplikation, mit Vue.js als Webframework.
Beim Laden der Seite wird eine WSS Verbindung zum Backend aufgebaut. Ohne Authentifizierung zeigt das Frontend eine Seite an, die es dem Nutzer ermöglicht sich durch individuelle Zugansdaten (Benutzername und Passwort) zu authentifizieren. Werden auf dieser Seite gültige Zugangsdaten eingegeben, ist die WSS Session authentifiziert und zeigt das Layout in Abbildung 4.4 an. Das Layout beinhaltet oben links einen Button um die Navigation der Seite ein und auszublenden. Ist die Navigation am linken Rand eingeblendet, so ermöglicht sie es zwischen den einzelnen Seiten der Visualisierung zu navigieren. Dazu zeigt die Navigation immer einen „Zurück" Button, um in die übergeordnete Seite zu wechseln, sowie Buttons um zu den untergeordneten Seiten zu wechseln. Existieren keine untergeordneten Seiten, oder keine übergeordnete Seite, so existieren auch die Buttons nicht. Schließlich wird im zentralen Fenster die aktuelle Page der Visualisierung angezeigt. Der komplette Seiteninhalt ist eine Repräsentation des Datenobjekts, das global in der Webapplikation vorliegt. Diese Datenstruktur ist eine Kopie der Daten aus dem Backend (Abschnitt 4.3.1), welche die aktuell angezeigte Seite betreffen. Sie ist in Abschnitt 4.3.2 genauer beschrieben und kann nur über die Websocket Schnittstelle verändert werden. So ist sichergestellt, dass die GUI Elemente immer die Daten des Backends darstellen und es keine Unterschiede gibt. Der Zustand ist also über alle Instanzen des Backends und Frontends immer konsistent. Wie Daten verändert werden können ist genauer in Abschnitt 4.2.3 beschrieben. Innerhalb der Webapplikation existiert eine Page

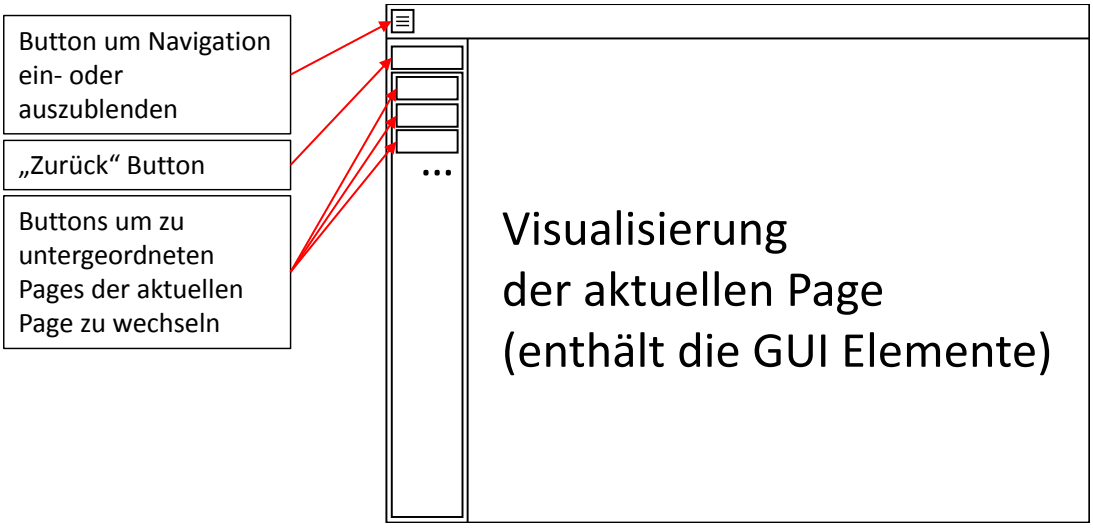


Abbildung 4.4: Layout des Frontends bei einer authentifizierten WSS Session

Komponente die, entsprechend der GUI Elemente in der globalen Datenstruktur, GUI Element Komponenten (*guiElement*) instanziert und auf der Page anzeigt. Diese GUI Elemente werden entsprechend ihres Typs (zum Beispiel *Button*) weiter in spezialisierte GUI Elemente (z.B. *guiElementButton*) unterteilt. Das Schema, wie innerhalb der Webapplikation auf Daten zugegriffen werden kann, bzw. Daten geändert werden können, ist in Abbildung 4.5 dargestellt. Die GUI Elemente haben nun Zugriff auf eine Funktion um Datenänderungen anzufordern, sowie auf den ihnen zugeordneten Teil der globalen Datenstruktur. Wie der Dis-

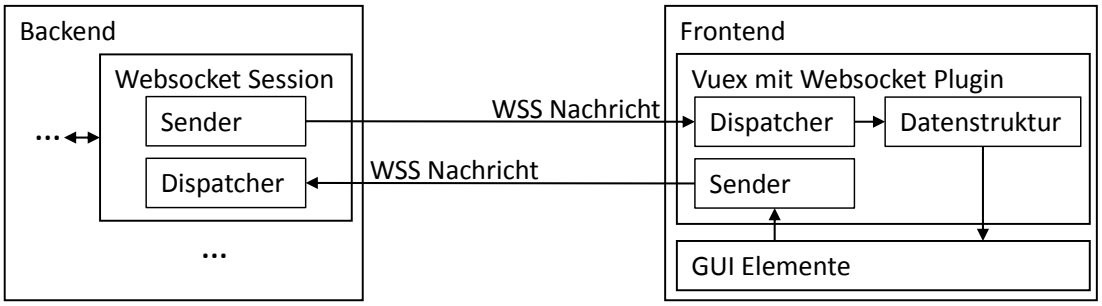


Abbildung 4.5: Schema des Zugriffs auf die Datenstruktur des Frontends

patcher die Struktur des Frontends ändert ist in Abschnitt 4.2.3 beschrieben. Die GUI Elemente können nur lesend auf die Datenstruktur zugreifen.

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. Deshalb 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.

... alle Datentypen unterstützen, die das Backend definiert.

... weitestgehend symmetrisch sein (Die Nachrichten vom Server zum Client haben eine ähnliche Gestalt wie die vom Client zum Server).

... möglichst leichtgewichtig sein.

... möglichst frei von Zuständen sein, sodass ein Vertauschen zweier Nachrichten keine Relevanz für die Applikation darstellt.

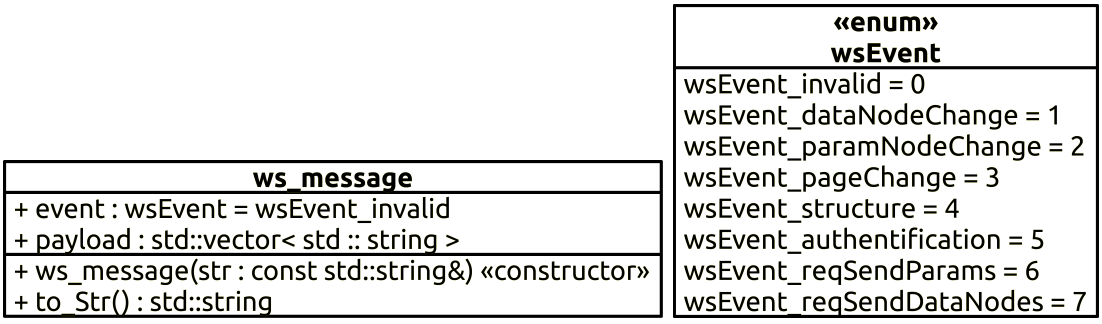
... eine Authentifizierung des Nutzers ermöglichen.

Die meisten Anforderungen können erfüllt werden, allerdings stehen die letzten drei in Konkurrenz zueinander. Um eine Authentifizierung zu ermöglichen gibt es, 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 Zeitpunkt mehr Rechte.

- Die Zugangsdaten werden bei jeder Nachricht mitgesendet und überprüft.

Die erste Variante ist möglichst leichtgewichtig, da die Zugangsdaten nur einmal gesendet werden. Allerdings ist sie nicht frei von Zuständen, da die Schnittstelle, je nachdem ob eine Session bereits authentifiziert ist, unterschiedlich auf eingehende Nachrichten reagiert. Die zweite Variante reagiert immer identisch (deterministisch) auf die gleiche Nachricht, ist aber weniger performant, da das Verhältnis von Protokoll-Overhead zu Nutzdaten immer größer ist als bei der ersten Variante. Aufgrund dieses Sachverhalts wird bei der Zustandsfreiheit eine Ausnahme gemacht werden.
Das erarbeitete Protokoll tauscht Strings aus, dessen Struktur der Klasse in Abbildung 4.6a 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 4.6b 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



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

(b) Websocket Event Definition als Aufzählung

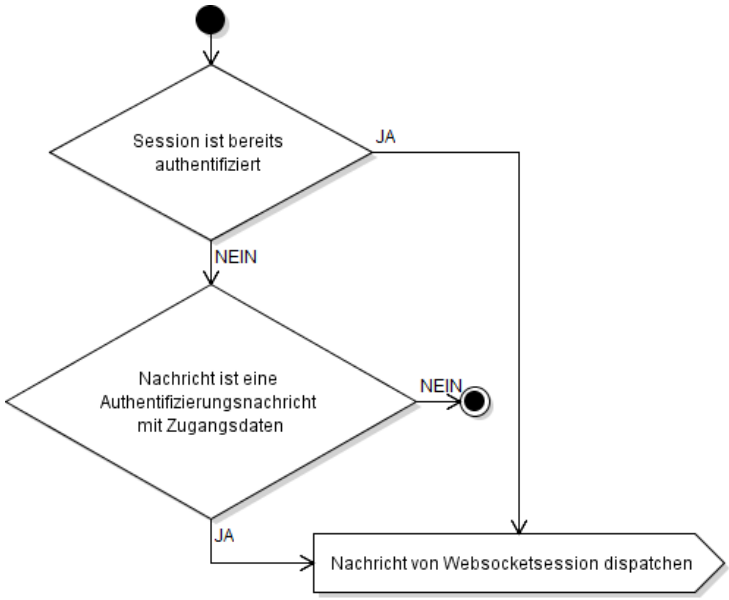
Abbildung 4.6: Websocket Message Container und Event

das *ws_message* Objekt enthält (*event* und *payload*), als String zu kodieren und durch ein Semikolon getrennt zu einem String zusammenzusetzen. 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, solange der Nutzdatenvektor kein String mit einem Semikolon 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 unterstützt nun die in Abbildung 4.6b dargestellten Events. Das erste Event (*wsEvent_invalid*) ist der Standartwert des Events in jedem *ws_message* Objekt. Auf eine Nachricht mit diesem Event wird beim Empfang von keiner Seite aus reagiert und die Nachricht verworfen. 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 empfangenen 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 4.7a 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 empfangenen Events aufgerufen.

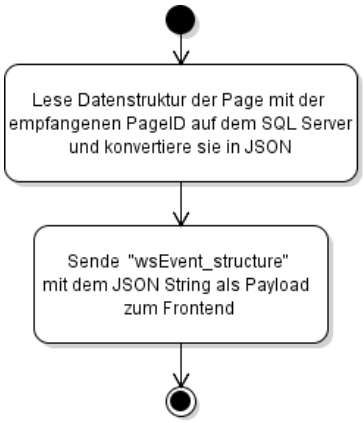
Die Handler des Backends sind durch die verbleibenden Diagramme in Abbildung 4.7, sowie durch die Diagramme in Abbildung 4.8 definiert. Die Handler des Frontends sind in Abbildung 4.9 dargestellt.

Das Frontend bedarf keines solchen Authentifizierungsmechanismus, da dies bereits im WSS Protokoll implementiert ist. Wie auch bei HTTPS geschieht das durch ein Serverzertifikat des Backends, dass durch eine externe Certification Authority (CA) beim Verbindungsaufbau verifiziert werden muss. Das Fro

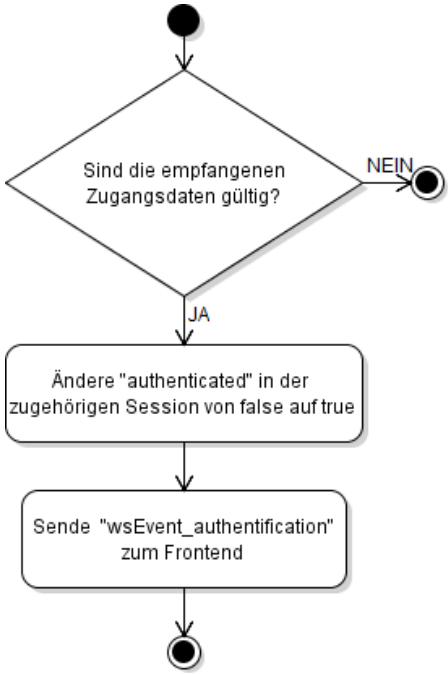
asdasd



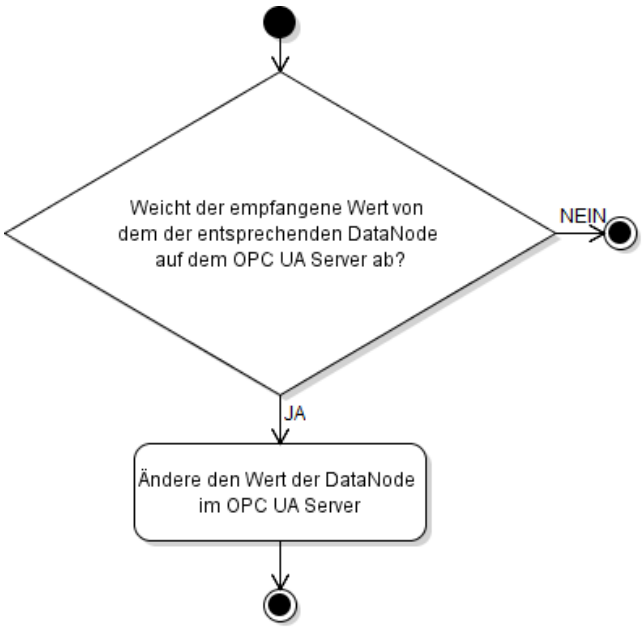
(a) WebsocketEvent empfangen



(b) WebsocketEvent: *Structure* empfangen

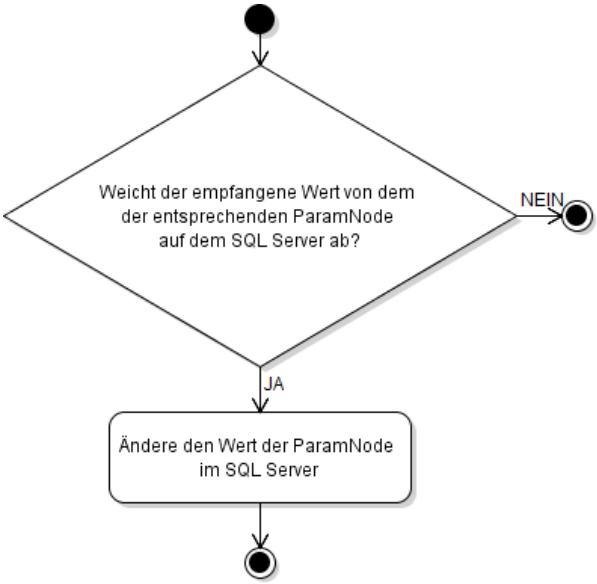


(c) WebsocketEvent: *Authentification* empfangen

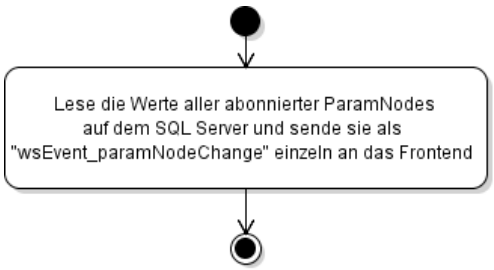


(d) WebsocketEvent: *DataNodeChange* empfangen

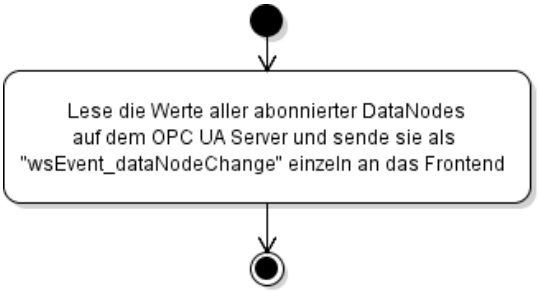
Abbildung 4.7: Aktivitätsdiagramme des Dispatchers im Backend I



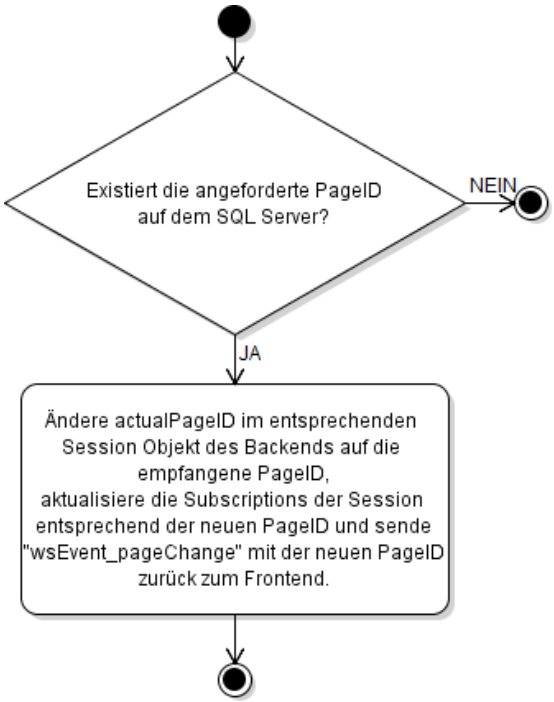
(a) WebsocketEvent: *ParamNodeChange* empfangen



(b) WebsocketEvent: *ReqSendParamNodes* empfangen

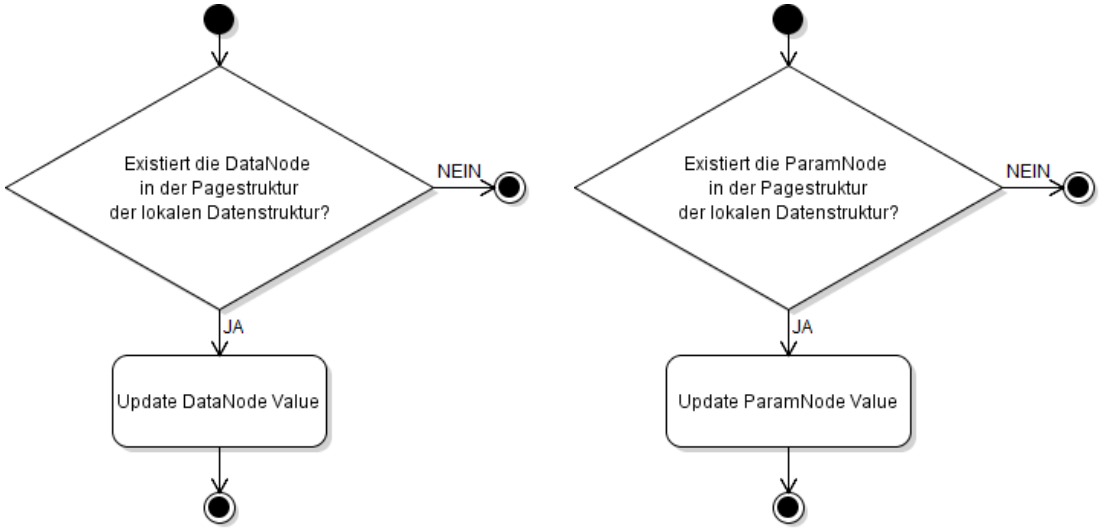


(c) WebsocketEvent: *ReqSendDataNodes* empfangen



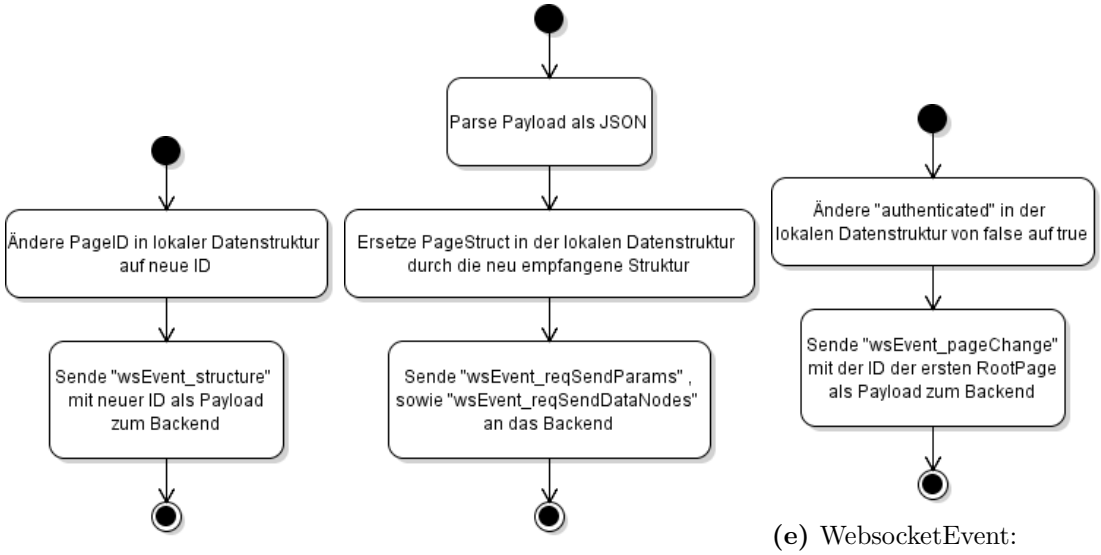
(d) WebsocketEvent: *PageChange* empfangen

Abbildung 4.8: 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 4.9: Aktivitätsdiagramme des Dispatchers im Frontend

4.3 Datenstruktur

4.3.1 Backend

Das abgebildete Entity Relationship Diagram (ERD) in Abbildung 4.10, 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, einen 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 dieser alle relevanten Relationen kennt und entsprechend verwaltet. So werden 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 2.3.1).

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 eine DataNode auf dem OPC UA Server abgelegt wird, wird den DataNodeTemplates des SQL Servers entnommen.
Auf dem SQL Server haben die DataNodes, sowie Parameter, immer den nativen SQL Datentyp *VARCHAR*, um ein Speicherfeld zu erhalten, das alle benutzten OPC UA Datentypen unterstützt. Beim Lesen, beziehungsweise beim Schreiben, wird der als String gespeicherte Wert auf dem SQL Server, entsprechend seines explizit abgespeicherten Datentyps, 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, auf dem OPC UA Server, genutzt werden.

4.3.2 Frontend

...

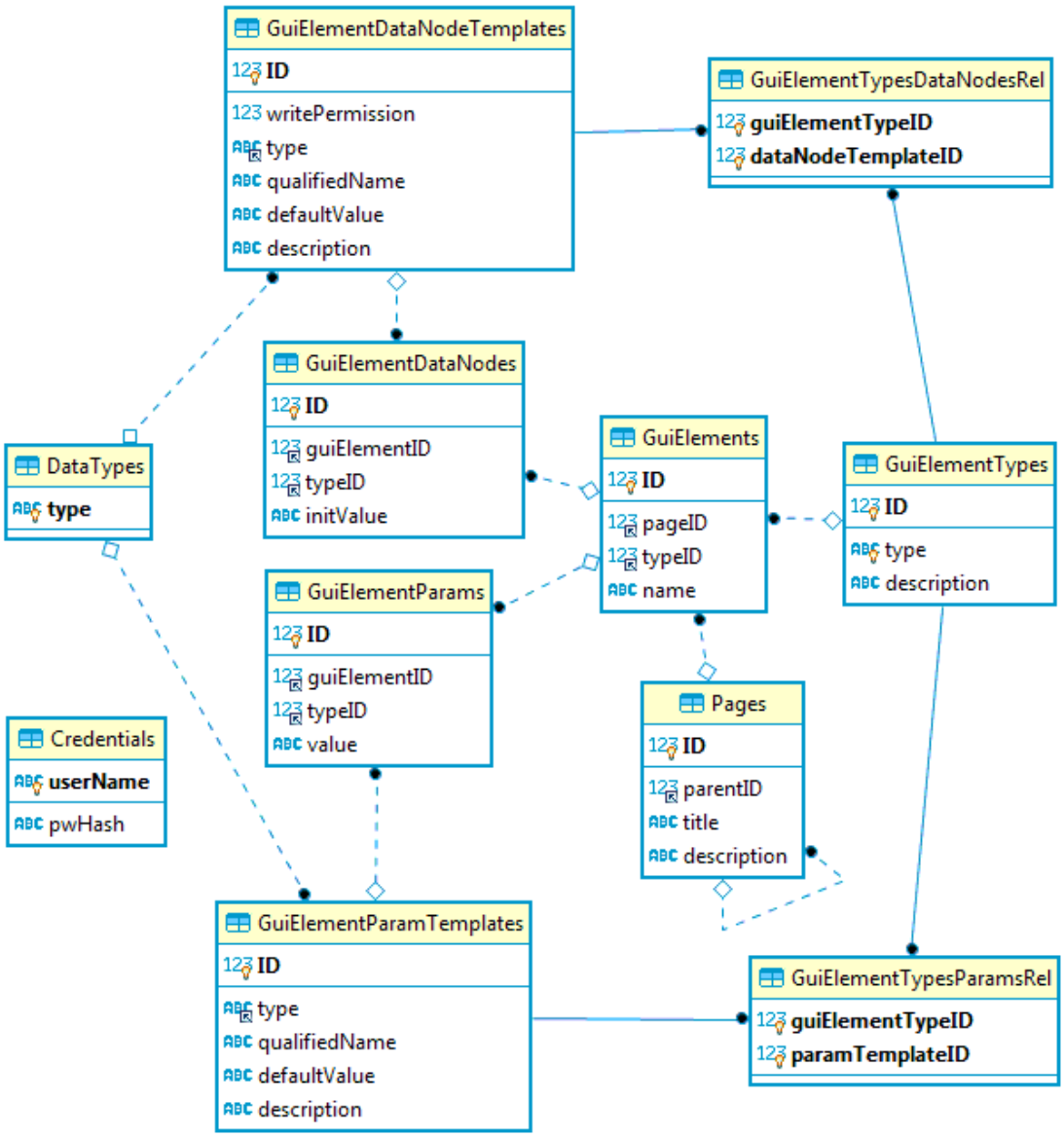


Abbildung 4.10: ERD des SCADA Systems.
Darstellung zeigt die einzelnen Tabellen der Datenbank auf dem SQL Server und deren Beziehungen zueinander.

5 Umsetzung des Proof of Concept

5.1 backend

5.1.1 HIER KLASSEN

5.2 Frontend

6 Zusammenfassung und Ausblick

Literaturverzeichnis

[1] Scholz, P. [2005]. *Echtzeit, Echtzeitsysteme, Echtzeitbetriebssysteme*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter 3, S. 39–73.
URL: *https://doi.org/10.1007/3-540-27522-3_3*

[2] Schubert, M. [2007]. *„Das wird teuer“ — der EDV-Spezialist tritt auf*, Teubner, Wiesbaden, chapter 3, S. 40–49.
URL: *https://doi.org/10.1007/978-3-8351-9108-2_3*

[3] Studer, T. [2016]. *Das Relationenmodell*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter 2, S. 9–21.
URL: *https://doi.org/10.1007/978-3-662-46571-4_2*