Technical Report : OPCUA-Netzwerk

Johannes Horst, Manuel Zimmermann, Patrick Sabau, Saniye Ogul, Stefan Ries und Tobias Schotter

I. EINLEITUNG

Ziel dieses Projektes soll die Entwicklung eines Sensor-Netzwerks für den Heimbereich auf Basis von OPC-UA sein. Verschiedene Sensorknoten sollen mit entsprechender Sensorik und Aktorik ausgestattet sein, um bestimmte Daten des eigenen Zuhauses zu sammeln. Eine grafische Oberfläche ermöglicht dem Nutzer das Abrufen der Daten und außerdem Steuerungsfunktionalitäten, um z.B. eine Heizung oder eine Lüftungsanlage ein- bzw. auszuschalten. Hierdurch ergeben sich große Energiesparpotentiale, da man so nur lüftet bzw. heizt, wenn die Luftqualität/Temperatur dies erfordert.

OPC-UA steht hierbei für „Open Platform Communication – Unified Architecture“ und ist eine aktuelle Technologie aus dem Industrie-4.0-Umfeld. OPC-UA soll die plattformunabhängige Machine-to-Machine Kommunikation ermöglichen. Dies wird durch ein zu modellierendes Informationsmodell möglich, welches den realen Sachverhalt abbildet und von einem zentralen Server verwaltet wird. Clients können sich mit dem Server verbinden und dort relevante Informationen ablegen bzw. diese abrufen oder durch Publish/Subscribe auch vom Server Daten erhalten. Die Informationen werden auf dem Server hierbei in einer Baumstruktur verwaltet. Durch die entsprechende Modellierung der Knoten des Baums erhalten diese z.B. durch die Festlegung eigener Datentypen eine semantische Bedeutung.

II. SENSOREN UND KOMMUNIKATION

Sensoren ...

III. BACKEND + DATENBANK

Das Backend wurde mit dem Framework **FastAPI** in Python umgesetzt und ist die einzige Schnittstelle zur Datenbank **MongoDB**.

A. Datenbank

Um eine MongoDB lokal (für die Entwicklung) zu starten:

- Herunterladen von MongoDB über die offizielle Website oder einen Package Manager (z.B. ABT)
- Ausführung der in der ReadMe-Datei hinterlegten Konsolenbefehle

Die Datenbank wird dabei in der Standardkonfiguration genutzt, somit ist es nicht erforderlich Nutzer oder anderes anzulegen. Lediglich sollte die Datenbank über den Standardport 27017 erreichbar sein.

MongoDB enthält zu Beginn einige Tabellen die zur Konfiguration und zur internen Konsistenz genutzt werden. Die eigentlichen Daten werden dabei in einer neuen Datenbank (der Name dieser kann konfiguriert werden) gespeichert. Eine solche Datenbank kann mehrere Collectionen beinhalten, das Äquivalent zu Tabellen in einer SQL Datenbank. Diese Collectionen beinhalten i.d.R ähnliche Documente im Format BSON (Binary JSON).

B. Datenformat

Es werden Datenformate genutzt, die jeweils in Collections gespeichert werden : **Sensoren**.

Listing 1 zeigt ein Sensor Objekt, wie es in der Datenbank gespeichert sein könnte.

```
{
    "_id": ObjectID(abc123def456),
    "sensornode": "SensorNode_1",
    "sensorname": "BME280",
    "sensortyp" : "AirPressure",
    "unit"      : "hPa,
    "value"     : 1040.6999999999991,
    "timestamp" : "2022-12-06T15:15:28.112Z"
}
```

Listing 1. Sensor Objekt

Das Feld **'_id'** ist ein Primärschlüssel, welcher einzigartig ist und automatisch von MongoDB vergeben wird. Mit diesem können Objekte eindeutig referenziert werden. Über **'sensornode'** wird die Bezeichnung der einzelnen Nodes gespeichert. Das Feld **'sensorname'** speichert den technische Namen eines Sensors als String und das Feld **'sensortyp'** die tatsächliche Sensorbezeichnung. Durch Kombination dieser drei Werte können die Sensoren eindeutig zugeordnet werden, sowie im Frontend passend der per Name angezeigt werden.

Die Datenbankobjekte werden im Code als Pydantic Dataclasses hinterlegt, was das Parsen dieser Objekte (z.B. als JSON Payload) erleichtert. Dabei werden die Klassen mit dem Decorator **@dataclass** verziert und erhalten **TypeHints** mit entsprechden Datentypen

```
@dataclass
class Sensor_value_dto():
    sensornode: str
    sensorname: str
    sensortyp: str
    value: Union[float,bool]
    timestamp: datetime.datetime
    unit: str = ""
```

Listing 2. Sensor Dataclass

```
@dataclass
class actuator_list_dto:
    actuator_node  : str
    actuator_act   : str
    actuator_value: Union[str,float,bool,None]
    actuator_dtype: str
```

Listing 3. Actuator Dataclass

C. Backend

Mittels dem Python Framework **FastAPI** werden diverse Endpoints bereitgestellt, die das Erstellen und Ausgeben der Datenobjekte ermöglichen.

Die Sensoren können über folgende Routen ausgegeben werden:

- **GET /sensornames**: Gibt eine List aller zur verfügung stehenden Sensoren zurück.
- **GET /sensorvalues/current**: Gibt den aktuellen Wert jedes Sensors zurück.
- **GET /sensorvalues**: Gibt eine List von Sensoren zurück, welche sich nach ihren Attributen filtern lassen.

Die Aktoren können über folgende Routen ausgelesen und gesteuert werden:

- **GET /actuatornames**: Gibt eine List aller zur verfügung stehenden Aktorenbezeichnungen zurück.
- **GET /actuators**: Gibt eine Liste von JSON-Objekten zurück mit entsprechenden Informationen einzelner Aktoren.
- **GET /actuators/filter**: Gibt ebenso eine Liste von JSON-Objekten zurück, welche per Aktorbezeichnung gefiltert wurde.
- **PUT /actuators**: Ermöglicht per Übergabe von Aktorenknoten und Bezeichnung den Wert des Aktors zu ändern.

Die Endpunkte nutzen dabei ein automatisiertes Umwandeln zu entsprechenden Dataclasses. Dies sorgt dafür, dass Pflichtfelder übergeben werden und nicht benötigte Elemente verworfen werden. Über einen PyMongo Client werden diese Dataclasses also entweder von Route zur Datenbank weitergeleitet, oder aus der Datenbank zur Route. Die Verbindung erfolgt dabei über eine **MONGO_URI**, welche Hostnamen und Port beinhaltet. Dieser String sieht wie folgt aus: *mongodb://localhost:27017/*.
Als tatsächlicher Webserver wird **uvicorn** genutzt, welcher im **__main__.py** gestartet wird. Dies ermöglicht es, das Backend mittels des Befehlt *python -m backend* zu starten.

D. Tests

In einem separaten Testscript **test_backend.py** werden die Routen getestet. Hierbei wird jedoch eine Testtabelle in der Datenbank genutzt, um nicht mit anderen Daten zu interferieren. In einer Setup-Methode werden dabei die Referenzen auf die Tabelle ausgetauscht. Der Server wird ebenfalls durch einen **TestClient** ersetzt, welcher in dem FastAPI Framework integriert ist.

!!! TODO: Beschreibung der Test selbst !!!

Die Tests können mittels des Befehls *python -m pytest -s* gestartet werden.

IV. FRONTEND

Das Frontend wurde mithilfe der Frameworks Angular entwickelt.

A. Architektur

B. Entwickelte Komponenten

V. AUSBLICK