Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Eine IoT Anwendung als verteiltes System

im Rahmen des Moduls **Verteilte Systeme**
von **Prof. Dr. Christin Schmidt**

Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang: Angewandte Informatik

Eingereicht von:

Corvin Zloch [587624]
Arsene Rayane Wanda Tchatat [585008]
Emma Schüler [589448]
Daniel Marks [589544]

Abstract

Eine Beispielhafte Anwendung aus dem Bereich des Internet der Dinge / Internet of Things (IoT) ist eine Pflanzen Monitoring App für Werte aus der physischen Welt, beispielsweise Temperatur oder Bodenfeuchtigkeit. Diese Werte können in einer derartigen Anwendung ausgelesen, digitalisiert verarbeitet und Nutzern angezeigt werden. Dieses Paper analysiert die Beispielanwendung unter der Perspektive von Verteilten Systemen und bietet eine detaillierte Grundlage zur Nachimplementierung.

Inhaltsverzeichnis

1.1 Eigenschaften vom IoT

Das Internet of Things, oder IoT, ist das Produkt vieler verschiedener Technologien und technologischen Fähigkeiten, von einzelnen Geräten und Objekten, die zusammenarbeiten, um die verschiedenen Anwendungen mit dem alltäglichen Leben verschmelzen zu lassen. Zu diesen Fähigkeiten zählen die Kommunikation und Kooperation zwischen Geräten, die Ansprechbarkeit, durch welche es möglich ist, Objekte innerhalb eines Systems zu kontrollieren, als auch die Identifizierbarkeit, also die Tatsache, dass jedes Objekt des Systems eindeutig erkennbar ist. Hier können zum Beispiel NFC-Chips genutzt werden. Durch Mediatoren wie Smartphones können diese dann eingelesen und bestimmten Informationen zugeordnet werden. Ist ein Objekt mit Sensoren ausgestattet, so kann es entsprechende Daten sammeln und teilen. Falls möglich, kann auf diese direkt reagiert werden. Auch die Kontrolle von technischen Geräten und Prozessen in der realen Welt über das Internet können damit möglich gemacht werden. “Smarte“-Objekte können durch einen eigenen Prozessor und Speicherkapazität direkt Sensordaten verarbeiten und interpretieren. Ebenso können sie ihren Standort durch Technologien wie GPS und Mobilfunknetze herausfinden. Interagieren Menschen mit dem Gerät, direkt oder indirekt, werden User Interfaces wie zum Beispiel Sensoren, die Handbewegungen interpretieren, oder Stimmerkennung benötigt. [17]

1.2 Technologische Herausforderung

Wie auch in den meisten anderen Domänen der IT gibt es beim IoT besondere technologische Herausforderungen. Dazu zählt, dass IoT Systeme sowohl lokal als auch auf großer Ebene funktionieren müssen, die Skalierbarkeit. Auch sollten alltägliche genutzte Objekte nicht als komplizierte Geräte wahrgenommen werden. Gerade sporadisch genutzt sollten sich schnell an ihre Situation anpassen und effizient Netzwerke untereinander aufbauen. Das Einhalten von einheitlichen Standards ist ebenso schwer, durch die große Diversität der physischen Welt und den daraus folgenden Einschränkungen. Da dadurch eine Software-Infrastruktur, die auf Servern im Hintergrund läuft, benötigt wird, steigt die Softwarekomplexität ebenso bei größeren Anwendungen. Gerade bei Sensoren Netzwerken werden eben solche Serverkapazitäten durch eine hohe Datenlast an ihr Limit getrieben. Auch Fehlertoleranz spielt in einer komplizierten Welt eine große Rolle. Hohe Komplexität birgt immer große Gefahr für Fehler Anfälligkeit, gerade bei Systemen, in denen sich die Umstände konstant verändern. Zusätzlich kommt es zu Risikos bei der Datensicherheit, durch bestehende Risiken des Internets und zusätzlich dem Risiko von eingeschleusten Geräten im System. [17]

1.3 Einordnung

Das Internet of Things ordnet sich bei den pervasiven Systemen ein, wobei es die Charakteristika aller drei Untertypen von Ubiquitous Computing, Mobilen Systeme und Sensoren Netzwerken teilt. So sind die Geräte oder Objekte im Internet of Things, meist Autonom, verteilt, unauffällig in der Interaktion mit dem Nutzer, sich Bewusst in welchem Kontext sie genutzt werden und intelligent, im Sinne, dass sie viele verschiedene Aktionen verarbeiten und ausführen können, was sie dem Ubiquitous Computing zuteilen würde. Gleichzeitig sind diese jedoch ebenso häufig, in sich oft ändernden Umständen in der Nutzung, als auch extrem abhängig von vielen Sensoren. Die Kategorien haben große Überschneidungen, was nicht per se ungewollt ist. Ihr Sinn ist es, eher verschiedene Aspekte von pervasiven Systemen hervorzuheben. Pervasive Systeme, im Vergleich zu anderen verteilten Systemen, haben das Ziel, sich ihrer Umgebung anzupassen und mit ihr zu verschmelzen. Auch die Grenzen zum Nutzer sind weitaus mehr verschwommen, als bei anderen Systemen, die zum Beispiel eine Tastatur und Maus benötigten, um mit ihnen zu interagieren. Es gibt meist kein einziges, eindeutiges User Interface, sondern eine Ansammlung verschiedener Geräte, die Informationen aus ihrer Umgebung aufnehmen und Informationen zur Verfügung stellen. Da pervasive Systeme Eigenschaften von dezentralen, als auch verteilten Systemen besitzen, kommt es auch hier zur weiteren Verwischung von Grenzen zwischen verschiedenen Konzepten. [18]

2. Relevante Technologien

2.1 Mikrocontroller und Sensoren als Komponenten in verteilten Systemen

Mikrocontroller:

Ein Mikrocontroller, wie der ESP32, ESP8266 oder Arduino, ist ein kompakter, stromsparender Computer. Er besteht aus einem Mikroprozessor, Speicher und Schnittstellen zum Anschließen der Sensoren. Für mehr Details, siehe 3.3 Hardware. Mikrocontroller in verteilten Systemen übernehmen die Aufgaben der Datenerfassung, Vorverarbeitung und Kommunikation. Sie fungieren als steuernde Instanz, die Sensordaten verarbeitet und beispielsweise das Senden von Daten an einen zentralen Server, Broker oder andere Geräte.

Ein wesentlicher Vorteil moderner Mikrocontroller ist ihre Integration mit Kommunikationsmodulen wie WLAN, welches wir auch in unserem Beispiel nutzen, Bluetooth (Low Energie) oder Zigbee, wodurch sie drahtlos mit anderen Knoten im Netzwerk interagieren können. Diese Fähigkeit ist wichtig für die Skalierbarkeit und Flexibilität verteilter Systeme.

Sensoren:

Sensoren sind Geräte, die Temperatur, Feuchtigkeit, aber auch Lichtintensität und Bewegung messen können und diese in elektrische Signale umwandeln. Sie stellen die primäre Datenquelle in unserem verteilten System dar und ermöglichen eine kontinuierliche Überwachung der Pflanzendaten. Sensoren können analog (z. B. Bodenfeuchtigkeitssensoren) oder digital (z. B. DHT22) ausgelegt sein und liefern Daten, die über analoge oder digitale Schnittstellen an Mikrocontroller weitergegeben werden.

Rolle in verteilten Systemen

In einem verteilten System agieren Mikrocontroller und Sensoren als dezentrale Einheiten. Die Erfassung und Verarbeitung von Daten erfolgt lokal, danach werden

Daten über Protokolle wie MQTT an zentrale Komponenten, wie Datenbanken via Broker, weitergeleitet. Dies ermöglicht Echtzeitüberwachung und in manchen Fällen auch Echtzeitsteuerung.

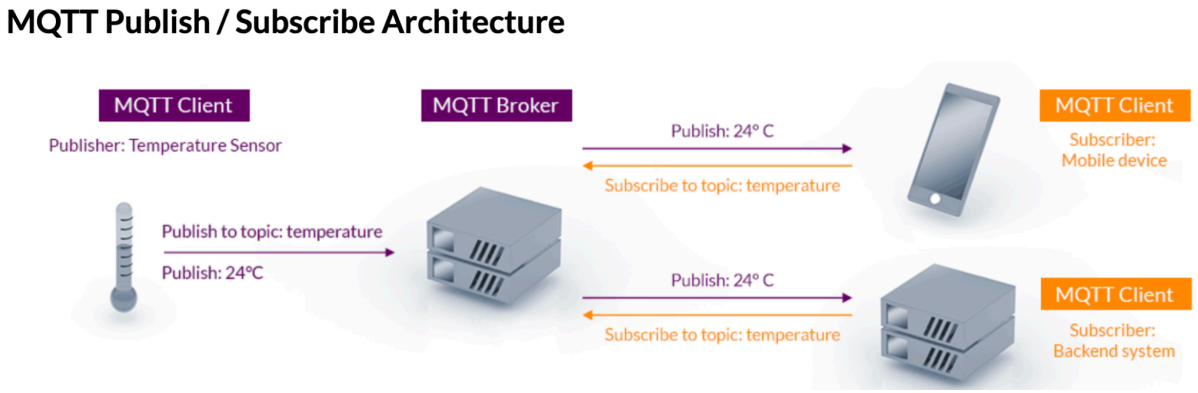
Herausforderungen

Zu den Herausforderungen gehören Energieeffizienz, insbesondere bei batteriebetriebenen Systemen, die Kalibrierung der Sensoren und die Robustheit der Kommunikation. Dennoch sind Mikrocontroller und Sensoren unverzichtbare Komponenten, um verteilte Systeme effizient und skalierbar zu gestalten.

Die Entscheidung, die Mikrokontroller und damit die Sensoren batteriebetrieben oder mit einem Kabel mit Strom zu versorgen, ist Anwendungsabhängig und hat Vor- und Nachteile.

2.2 MQTT

(Message Queuing Telemetry Transport) ist ein Kommunikationsprotokoll mit besonderen Features, das speziell für IoT-Anwendungen in verteilten Systemen entwickelt wurde. Es basiert auf einem **Publish-Subscribe-Modell**, bei dem ein **Broker** als Vermittler zwischen **Publishers** (Mikrocontroller) und **Subscribers** (Datenbanken/ Android App) agiert. Im Pflanzenmonitoring überträgt MQTT Sensordaten wie Temperatur und Feuchtigkeit an den Broker, von wo sie in Datenbanken gespeichert oder an Apps weitergeleitet werden. Es ist mit MQTT möglich, gezielt **bestimmte Topics zu abonnieren,** um Datenmengen zu minimieren und nur relevante Informationen zu empfangen. [15]



https://mqtt.org/assets/img/mqtt-publish-subscribe.png

2.3 Datenbanken

Eine Datenbank ist ein Sortiment von Daten, die in einer logischen Weise gestapelt sind und von einem selbst erstellten Datenbankverwaltungssystem (Database Management System, DBMS) betrieben werden. [16] Sie spielt eine zentrale Rolle in einem verteilten System, da sie die Speicherung, Verwaltung und Zugriff auf Daten ermöglicht. Es bestehen viele Datenbankmodelle (relationales Datenbankmodell, Schlüssel-Wert-Modell, hierarchische Datenbanken…), die meistens unterschiedliche Eigenschaften präsentieren und deswegen auch in verschiedenen Systemen eingesetzt werden sollen. In unserem Fall haben wir im Pflanzen Monitoring ein Zeitreihen Datenbankmodell angewendet, was wir Ihnen im nächsten Abschnitt präsentieren werden

3 Praxisbeispiel eines verteilten Systems

3.1 Beschreibung des Praxisbeispiels

In diesem Praxisbeispiel wird ein einfaches verteiltes System aufgebaut, in welchem ein ESP8266 dazu genutzt wird, um verschiedene Sensorwerte auszulesen und an eine Datenbank zu senden. Diese Werte werden dann von einer Java-Anwendung ausgelesen und können verarbeitet werden.

3.2 Datenbank

Es wird eine InfluxDB Cloud benutzt, da diese eine Skalierung für mehrere Sensoren im späteren Verlauf, eine genaue Zeit der Eintragung, sowie eine automatische Löschung nach vorgegebener Zeit ermöglicht. Die Cloud-Variante ermöglicht einen einfachen Start, da man keinen Server aufsetzen muss, um diesem Beispiel zu folgen [11].

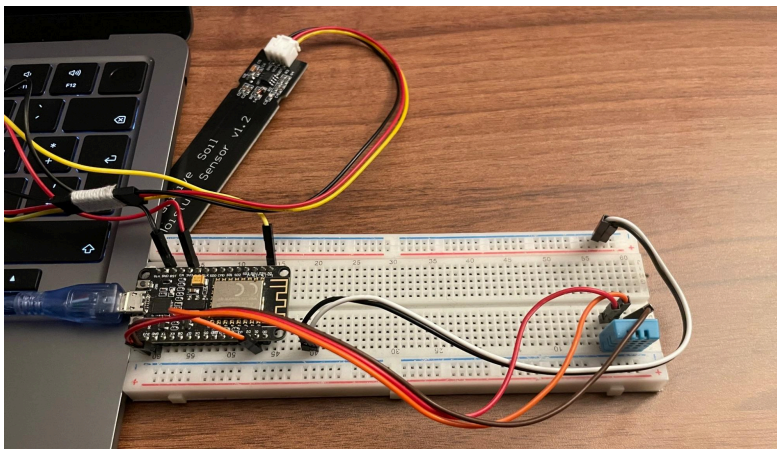
Zu Beginn muss ein Bucket erstellt werden, in welchem man die Measurements festlegen kann, welche als eigenständige Datenbanken agieren. Die Measurements sind hierbei die einzelnen Sensorwerte, welche in den Datenbanken gespeichert werden. In jedem Measurement wird als Field dabei von uns „value“ festgelegt. In diesem werden die jeweiligen Werte dann mit ihrer genauen Zeit eingetragen.

Screenshot aus der InfluxDB Website, gezeigt werden die Values für das Measurement ”temperature”

3.3 Hardware

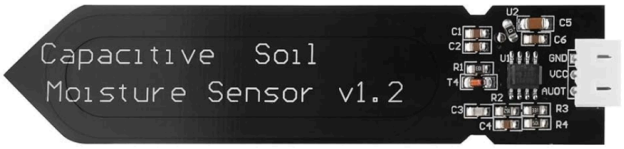
Wir benutzen in unserem Praxisbeispiel einen ESP8266. Vorteile von diesem Mikrocontroller sind das integrierte WLAN-Modul, welches wir zum Senden unserer Daten an die Datenbank verwenden. [1]

Angesteckt wird der ESP an ein Breadboard, an welchem wir die Sensoren anschließen. In einem Breadboard befinden sich mehrere Kontaktreihen, an denen wir die Sensoren mit dem ESP verbinden können.

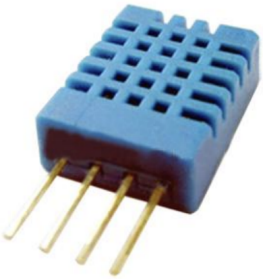


ESP8266 auf einem Breadboard zusammen mit den Sensoren verkabelt, eigenes Foto

Als Sensoren kommt bei uns ein DHT11 Sensor zum Einsatz, welcher Temperatur und Luftfeuchtigkeit messen kann [2]. Dieser wird an den Strom sowie an einen der digitalen Pins (D1) des ESP angeschlossen.

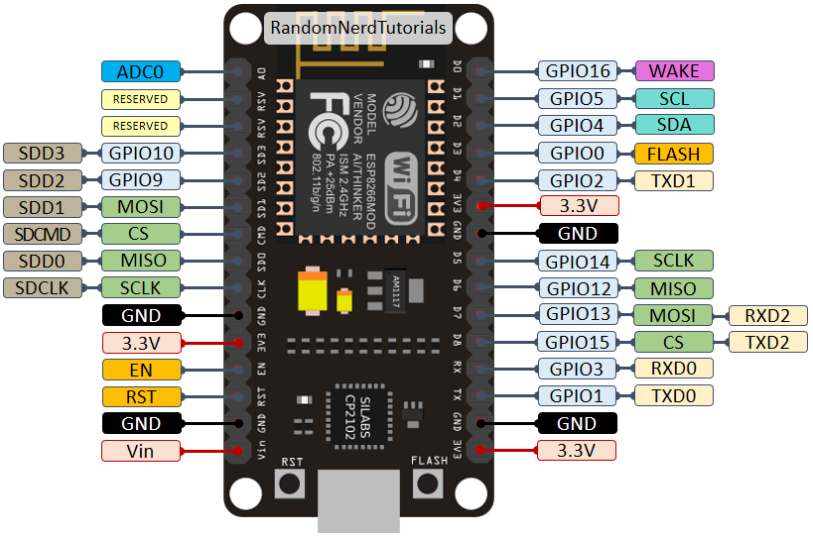


APKLVSR Bodenfeuchtigkeitssensor:
https://www.amazon.de/APKLVSR-Bodenfeuchtesensor-Kapazitive-Hygrometer-Feuchtigkeitssensor/dp/B0CQNF7S7L



DHT11: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Zudem wird ein Bodenfeuchtigkeitssensor der Firma APKLVSR benutzt, welcher die Feuchtigkeit des Bodens misst. Dieser wird auch an den Strom des ESP, sowie an den analogen Pin (A0) des ESP angeschlossen, da dieser den Wert direkt ohne weitere Verarbeitung durchgibt [3]. Welcher Pin an welchen Steckplatz kommt, muss in den jeweiligen PinOut-Dokumenten der Sensoren nachgelesen werden.



Pin Layout des ESP8266: https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

Strom bekommt dieses System durch eine Akku-Powerbank, welche durch ein Micro-USB Kabel den ESP mit Strom versorgt.

3.4 Einrichtung des ESP8266

Zur Einrichtung des ESP und späteren Hochladen des Codes nutzen wir die Arduino IDE[4]. In dieser müssen wir erstmal sicherstellen, dass die Software auf den ESP8266 eingestellt ist. [5]. In einem Mikrocontroller-Code gibt es zwei wichtige Methoden. Einmal die Methode „Setup“ in welchem wir den ESP für die jeweilige Funktionalität einrichten
Und eine Methode „Loop", in welchem der Code später geloopt wird und, selbst beim Aus- und Ein-schalten, der ESP die Werte senden kann[6][7].

Wir richten also im ersten Schritt mithilfe des Setups den ESP ein. Zuerst binden wir die Bibliotheken ein, welche auch in der Arduino IDE heruntergeladen werden müssen und definieren die Variablen.

Benutzte Bibliotheken:
- DHT11 Sensor[8]
- InfluxDb[9]
- ESP8266 WiFi [12]

Dazu werden die Variablen wie WLAN-Name und -Passwort, Steckplätze der Kabel und InfluxDB Daten sowie die Definition der Data Points für die Datenbank initialisiert.

In der Setup Methode des ESP werden die Sensoren gestartet. Danach wird eine Verbindung mit dem WLAN hergestellt, sowie mit der InfluxDB.

Als nächstes definieren wir die Loop-Methode, welche kontinuierlich wiederholt wird, solange der ESP mit Strom versorgt ist.
Unsere Variablen für „Humidity“, „Temperature“ und „Moisture“ werden mit den neuesten Sensorwerten beschrieben und danach in der Konsole ausgegeben, um diese später zu überprüfen.

```
Waiting 10 seconds
Writing temperature: temperature value=19i
Writing humidity: humidity value=50i
Writing soil moisture: soil_moisture value=612i
Waiting 10 seconds
Writing temperature: temperature value=19i
Writing humidity: humidity value=50i
Writing soil moisture: soil_moisture value=606i
```

Eigener Screenshot der Konsole der Arduino IDE

Gesendet werden diese dann an die InfluxDB, und deren zugehörige Data Points.
Zum Schluss wartet der ESP eine bestimmte Zeit, bis der Vorgang wiederholt wird und neue Daten an die Datenbank gesendet werden.

3.5 Datenabfrage per Java

Wollen wir nun die Sensorwerte unseres EPS auslesen, so können wir dies mit einer einfachen Java Anwendung tun.

Wir erstellen ein neues Java Projekt, und fügen als erstes den „influxdb-client-java“ als Dependency in unsere Pom.xml Datei ein [10].
Wir erstellen nun eine Java-Klasse „InfluxDBReader.java“. Dort werden zu Beginn die jeweiligen InfluxDB Werte für „url,token,org und bucket“ für die Abfrage hinterlegt, diese werden für eine Verbindung der InfluxDB benutzt.

Als nächstes können wir mithilfe einer fluxQuery die jeweiligen Daten abfragen:

```
String fluxQuery = String.format(
        "from(bucket:\"%s\") " +
                "|> range(start: -1h) " + // Daten aus der letzten Stunde
                "|> filter(fn: (r) => r._measurement == \"humidity\" or " +
                "r._measurement == \"temperature\" or r._measurement == \"soil_moisture\") " +
                "|> sort(columns: [\"_time\"], desc: true) " + // Neueste zuerst
                "|> limit(n: 1)",                              // Nur den letzten Wert
        bucket);
```

Eigener Screenshot der JetBrains IntelliJ IDE, unserer Abfrage.

Wir stellen dabei die Range auf „-1h“, dadurch bekommen wir nur die Werte der letzten Stunde und ignorieren ältere Einträge.
Im Filter haben wir eine „or“ Abfrage für unsere einzelnen Measurements, welche wir extrahieren möchten.
Zum Schluss definieren wir mit „desc:true“ und „limit(n:1)“ noch, dass wir nur den neuesten Wert aus der Datenbank nehmen möchten.

In einem Thread wird jetzt dieser Query für jede Tabelle, die wir zum Speichern der Messwerte benutzen, abgefragt und in der Konsole untereinander ausgegeben.

Dieser Thread läuft in unserem Beispiel jede 10 Sekunden durch.
In unserem Beispiel ist noch wichtig zu erwähnen, dass die Bodenfeuchtigkeit als analoge Zahlenwerte ausgegeben wird. Diese Werte erstrecken sich von ungefähr 685 (0% Feuchtigkeit) bis zu 499 (100% Feuchtigkeit). Sollte mit diesen Werten eine benutzerfreundliche Ausgabe erfolgen, so kann man diese in eine prozentuale Einteilung zwischen diesen beiden Werten umrechnen. Wir verzichten in diesem Beispiel darauf, damit alle Werte, die ausgegeben werden, auch den Werten der Sensoren entsprechen.

```
--- Neuste Messwerte ---
Time: 2025-01-15T20:22:09.930109047Z, Measurement: humidity, Field: value, Value: 50
Time: 2025-01-15T20:22:11.847131096Z, Measurement: soil_moisture, Field: value, Value: 612
Time: 2025-01-15T20:22:23.826860178Z, Measurement: temperature, Field: value, Value: 19
```

Eigener Screenshot der JetBrains IntelliJ IDE, Konsole gibt gelesene Werte aus.

Wir haben nun ein simples Verteiltes System aufgebaut, in welchem der ESP durch Sensoren Werte aufnimmt und an eine Datenbank sendet. Diese Werte werden jetzt von unserem Java Programm abgefragt und können auch für größere Projekte wie zum Beispiel einer App genutzt werden.
Den Code zum genauerem Nachlesen findet man hier: https://github.com/muriius/ESP8266-Sensoren-in-Java-auslesen

Ausblick:

In einem Paper veröffentlicht von IEEE [13] wird bestätigt wie IoT - Anwendungen von der Definition bedingt Verteilte Systeme sind, indem IoT- Anwendungen als eine Sammlung weitgehend unabhängiger, miteinander verbundener Computerelemente, die einige physische Ressourcen in einer Weise überwachen oder steuern, die den Nutzern des Systems als Betrieb einer einzigen Anlage erscheint, die einen bestimmten Geschäftsprozess realisiert. Woraus geschlussfolgert wird, dass Industrielle IoT-Systeme perfekt den klassischen Definitionen für verteilte Systeme entsprechen. Das Paper erläutert weiter, wie sich Industrielle IoT Anwendungen von traditionellen Verteilten Systemen unterscheiden, was allerdings nicht auf vergleichbar komplexe Anwendungen wie in unserem Fall zutrifft.

Ferner wird diskutiert, welchen Einfluss IoT von anderen Technologien wie Deep Learning, Digital Currencies, Augmented Reality, Virtual Reality und weiteren haben wird. Hierbei geht es um das Erkennen und Interpretieren von Mustern / Patterns in den von Sensoren ausgelesenen Daten, um Entscheidungen zu treffen, die die physikalische Welt beeinflussen. Oder um Digital Currencies, die eingesetzt werden, um mit Micropayments zwischen den einzelnen Knoten einen IoT-verteilten System Transaktionen durchzuführen. Neue Interfaces für Internet of Things Applications könnten mit Hilfe von AR und VR gestaltet und genutzt werden. [13]

Quellenverzeichnis

1: Ivan Grokhotkov, “ESP8266 Arduino Core 3.1.2-21-ga348833 documentation“, 2017
Url:https://arduino-esp8266.readthedocs.io/en/latest/libraries.html#wifi-esp8266wifi-library

2: Az-Delivery, “DHT11 Temperatursensor Datenblatt”
Url:https://cdn.shopify.com/s/files/1/1509/1638/files/DHT11_Temperatursensor_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1608564724

3: Amazon.com, “APKLVSR 5 Stück Bodenfeuchtesensor,Kapazitive Analoger Hygrometer Feuchtigkeitssensor ohne Korrosion,Soil Moisture Sensor für Arduino”
Url: https://www.amazon.de/dp/B0CQNF7S7L?ref=ppx_yo2ov_dt_b_fed_asin_title

4: docs.arduino.cc, „Arduino IDE“
Url: https://docs.arduino.cc/software/ide/

5: Rui Santos, RandomNerdTutorials.com : ”Installing ESP8266 Board in Arduino IDE (Windows, Mac OS X, Linux)”, 2019
Url: https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/

6: docs.arduino.cc, “setup()”, 2024
Url: https://docs.arduino.cc/language-reference/en/structure/sketch/setup/

7: docs.arduino.cc, “loop()”, 2024
Url: https://docs.arduino.cc/language-reference/de/struktur/sketch/loop/

8: docs.arduino.cc, Adafruit “DHT sensor library”, 2023
Url: https://docs.arduino.cc/libraries/dht-sensor-library/

9: Github.com, tobiasschuerg “InfluxDB-Client-for-Arduino”, 2024
Url: https://github.com/tobiasschuerg/InfluxDB-Client-for-Arduino

10: Github.com, influxdata “influxdb-client-java”, 2025
Url: https://github.com/influxdata/influxdb-client-java

11: docs.influxdata.com, “Get started with InfluxDB Cloud”
Url: https://docs.influxdata.com/influxdb/cloud/get-started/

12: Github.com, esp8266 “ESP8266WiFiMulti.h”, 2024
Url:https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFiMulti.h

13: K. Iwanicki, "A Distributed Systems Perspective on Industrial IoT," *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2018, pp. 1164-1170, doi: 10.1109/ICDCS.2018.00116.keywords: {Sensors;Scalability;Protocols;Internet of Things;Interoperability;Middleware;internet of things;iot;industrial internet of things;industrial iot;iiot;distributed systems;interoperability;scalability;dependability;vision;challenge},

15: MQTT.org, “MQTT: The Standard for IoT Messaging ", 2025
URL: https://mqtt.org/

16: Edwin Schicker, “Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL”, 2017, pp. 3
URL:
https://www.google.de/books/edition/Datenbanken_und_SQL/NCfXDQAAQBAJ?hl=fr&gbpv=1&dq=Was+ist+eine+Datenbank&pg=PA19&printsec=frontcover

17: Mattern, F., Floerkemeier, C. (2010). From the Internet of Computers to the Internet of Things. In: Sachs, K., Petrov, I., Guerrero, P. (eds) From Active Data Management to Event-Based Systems and More. Lecture Notes in Computer Science, vol 6462. Springer, Berlin, Heidelberg.
https://doi.org/10.1007/978-3-642-17226-7_15

18: M. van Steen and A.S. Tanenbaum, Distributed Systems, 4th ed., distributed-systems.net, 2023.