Fachhochschule
Dortmund

Bachelorarbeit

Oleksii Baida
Matrikelnummer 7210384

Sicherheits- & Steuerungssytem für das Haus

Bericht

24. Januar 2025

Inhaltsverzeichnis

1 Einleitung

In einer Welt, die zunehmend von vernetzten Geräten und dem Internet der Dinge geprägt ist, wird die Entwicklung effizienter und benutzerfreundlicher Systeme zur Steuerung und Überwachung von Gebäuden immer relevanter. Im Jahr 2024 wurde in Deutschland die Anzahl von über 19 Millionen Haushalten, die ein oder mehrere smarte Geräte besaßen, verzeichnet. Es wird prognostiziert, dass sich diese Zahl innerhalb der nächsten drei Jahre verdoppeln wird [4].

Moderne Steuerungs- und Sicherheitssysteme tragen zur Effizienzsteigerung und Ressourcenschonung bei. Laut Günther Ohland, Vorstandsmitglied des Branchenverbands SSmarthome Initiative Deutschland", ermöglichen diese Systeme eine Reduktion des Heizenergieverbrauchs um 20 bis 30 Prozent [5]. Die Kosten für die smarte Technik rechnen sich in der Regel nach zwei Jahren. Die Systeme übernehmen ein Teil der täglichen Aufgaben, wie das Ein- und Ausschalten des Lichts, die Reglung der Raumtemperatur oder das Aufräumen des Hauses etc. Der Aufgabenbereich der Systeme ist dabei nur nach den Bedürfnissen der Benutzerinnen und Benutzer abgegrenzt.

Im Rahmen meiner Bachelorarbeit wird ein System zur Steuerung und Überwachung des Hauses entwickelt. Das Ziel dieser Arbeit ist die Erstellung einer Schnittstelle, die die Interaktion des Benutzers mit den Geräten in seinem Haushalt ermöglicht und den Benutzer über gefährliche Vorgänge in seinem Haus informiert.

Im Rahmen der Entwicklung dieses Systems wurden die aktuellen Technologien zur Erstellung eines Webinterfaces und zur Kommunikation zwischen den Geräten eingesetzt. **TODO Kurz erklären was in Kapitel 2,3,4,5 ... erklärt wird**

2 Grundlagen & Theorie

In diesem Abschnitt erfolgt die detaillierte Darstellung der technischen Informationen zu den verwendeten Komponenten und Technologien.

2.1 Hardware

2.1.1 Arduino

2.1.2 ESP8266

2.1.3 Raspberry Pi

2.1.4 ESP32

2.1.5 M5Stick

2.1.6 BME680

2.1.7 VCNL

2.2 Kommunikationsprotokolle

2.2.1 HTTP

2.2.2 MQTT

2.2.3 UART

2.2.4 I2C

2.3 Software

2.3.1 PlatformIO

2.3.2 Uvicorn

2.3.3 HTML & TailwindCSS

2.3.4 Javascript

2.3.5 WebSocket

2.3.6 Linux-Pakete für Raspberry Pi

2.3.7 Python

2.3.8 Asyncio

2.3.9 FastAPI

2.3.10 SQLAlchemy

3 Konzeption das Systems

Im Rahmen dieses Projektes wurde ein System entwickelt, welches die Funktionalitäten eines Kontroll- und Verwaltungssystems mit denen eines IoT-Systems vereint. Die Integration von Sensordaten und Benutzerinteraktionen stellt einen wesentlichen Aspekt des Systems dar. Die Realisierung erfolgt durch die Kombination verschiedener Technologien und Teilsysteme, darunter eine Webanwendung auf FastAPI, eine MQTT-Kommunikationsschicht und verschiedene Aktoren und Sensoren, die auf Arduino- oder ESP-Module basieren.

3.1 Komponenten des Systems

Das entwickelte System basiert auf einer modularen Architektur, die mehrere Komponenten integriert. Jede dieser Komponenten erfüllt eine spezifische Rolle im Gesamtsystem:

- **Raspberry Pi:** Der zentrale Server, der das lokale WLAN-Netzwerk bereitstellt, den MQTT-Broker hostet und die Webanwendung ausführt.
- **Arduino:** Ausgestattet mit mehreren Sensoren, die Gefahren wie Feuer und Gas erkennen und den Zugang zum Haus sichern. Entsprechende Meldungen werden an den Server gesendet.
- **M5Stick:** Angeschlossen an die Temperatur- und Lichtsensoren und sendet die aufgezeichneten Daten auf den Server.
- **ESP32:** TTTOOOOODDDDOOOO
- **Webserver:** Eine auf FastAPI basierende RESTful API, die Benutzern den Zugriff auf das System und die Steuerung von Geräten ermöglicht.
- **Datenbank:** Eine lokale Datenbank zur Speicherung von Benutzerinformationen und Gerätekonfigurationen.
- **Web-Anwendung:** Eine browserbasierte Benutzeroberfläche, die den Benutzern eine intuitive Steuerung und Visualisierung der Daten ermöglicht.

3.2 Architektur und Datenfluss

Der Raspberry Pi dient als zentraler Server des Systems. Der Minicomputer stellt ein WLAN-Netzwerk zur Verfügung und hostet den Webserver mit der Datenbank sowie den MQTT-Broker. Alle Benutzerinteraktionen, Sensordaten, Datenflüsse und Datenverarbeitungen finden auf dem Server statt. Somit ist das System stark zentralisiert. Das System ist somit stark zentralisiert und arbeitet nur lokal. Das bedeutet, dass sich alle Benutzer in einem lokalen Netzwerk mit dem Raspberry Pi befinden müssen.

Die Geräte verbinden sich mit dem WLAN, das vom Raspberry Pi zur Verfügung gestellt wird. Die Sensoren senden ihre Daten an den MQTT-Broker, der auf dem Raspberry Pi läuft. Die Aktoren abonnieren die Command-Topics mit der entsprechenden "Geräte-ID".

Im Kern des Systems befindet sich ein Webserver, der auf dem Raspberry Pi ausgeführt wird. Dieser Webserver stellt eine RESTful-API zur Verfügung. Für den Zugriff zu der API wurde eine Webseite aufgebaut. Die API bietet folgende Funktionen an:

- Authentifizierung und Autorisierung der Benutzer
- Verwaltung der Gerätekonfigurationen und Benutzerprofile

- Bereitstellung von Endpunkten zur Abfrage und Steuerung von IoT-Geräten
- Bidirektionale Kommunikation mit den IoT-Geräten

Die Benutzerdaten und Konfigurationen werden in der lokalen Datenbank auf dem Server gespeichert.

4 Implementierung und praktische Umsetzung

4.1 Einrichtung der Hardwarekomponenten

4.1.1 Arduino

Der Arduino mit den angeschlossenen Sensoren stellt das Sichterheitskomponente des Systems dar. Dieses Teilsystem erkennt die Gefahren von Gas und Feuer und alarmiert den Benutzer. Außerdem stellt der Arduino der gesicherte Zugang zu dem Haus durch die Eingabe einer PIN. Die Anschließung der Sensoren und Aktoren an Arduino ist in der PA1 [1] beschrieben.

Der Arduino kann nicht direkt mit einem WLAN verbunden werden, da er kein WLAN-Modul besitzt. Für diesen Zweck habe ich ein ESP8266-Modul verwendet. Der wird mit dem Arduino durch eine UART-Schnittstelle angeschlossenen und mit dem WLAN verbunden. Die Kommunikation zwischen Arduino und ESP8266 erfolgt über die serielle Schnittstelle. Der Arduino gibt die entsprechenden Kommanden durch Serial an ESP8266 weiter und der ESP8266 führt die Befehle aus. Die detaillierte Beschreibung zur Verbindung von Arduino und ESP8266 ist in dem Bericht zu meiner Projektarbeit 2 [2] Kapitel 4.1 zu finden.

Der Arduino sendet und empfängt die MQTT-Nachrichten via ESP8266. Ein Mal pro Sekunde sendet der Arduino eine MQTT-Nachricht in das Topic ßtatus/¡ID¿/ïn dem die Bereitschaft der angeschlossenen Sensoren geteilt wird. Bei der Erkennung einer Gefahr wird sofort der Alarm ausgelöst und eine MQTT-Nachricht in das Topic älarm/¡ID¿"mit dem entsprechenden Text gesendet.

4.1.2 Raspberry Pi

4.1.3 Setup der ESP-Module

In diesem Projekt werden drei verschiedene ESP-Module verwendet: ESP8266, ESP32 und der auf dem ESP32 basierende M5Stick. Jedes Modul erfüllt spezifische Aufgaben, aber alle Module müssen sowohl mit dem WLAN als auch mit dem MQTT-Broker verbunden sein. Daher ist die Implementierung der Funktion `setup()` für alle drei Module ähnlich aufgebaut.

Beim Start des ESP-Moduls wird die Funktion `setup()` (Listing 4) aufgerufen. Zunächst wird es versucht, die WLAN-Zugangsdaten aus dem EEPROM auszulesen (Zeile 5-10). Das EEPROM wird mit einer Größe von 128 Byte initialisiert. Dies ist notwendig, bevor Daten aus dem Speicher gelesen werden können. Es werden zwei Felder vom Typ `char` wurden mit den in den eckigen Klammern angegebenen Längen erstellt (Zeile 6-7), um die aus dem EEPROM gelesenen Daten zu speichern. Beide Felder sind zunächst mit Nullen gefüllt. Der Name des WLAN-Netzes (SSID) wird ab Adresse 0 des EEPROM ausgelesen und mit der Funktion `EEPROM.get(0, eeprom_ssid)` im Char-Feld gespeichert. Das Passwort wird ab Adresse 32 aus dem EEPROM ausgelesen.

Anschließend prft die Boolean-Funktion `is_valid_string()` (Listing 5), ob der übergebene String-Parameter eine Länge größer als 0 und kleiner als `max_length`, sowie eine korrekte Terminierung hat. Zusätzlich gibt die Funktion den Wert `False` zurück, wenn der String ein Standard-EEPROM-Zeichen mit dem Wert `0xFF` enthält.

Sind die gültigen Werte im EEPROM gespeichert, versucht das Modul, sich mit diesen Zugangsdaten mit dem WLAN zu verbinden (Zeile 14). Die Funktion `connect_wifi` (Listing 6) schaltet das WLAN-Modul in den Modus SStationüm, was die Verbindung mit anderen WLAN-Netzwerken erlaubt. Danach wird es versucht mit dem WLAN mit übergebenen Zugangsdaten zu verbinden. In der Variable `wifi_repeat` ist die Anzahl der Versuche zur Erstellung der Verbindung gespeichert. Jede Sekunde wird den Status der Verbindung überprüft. Wenn die Verbindung erfolgreich erstellt wurde, gibt die Funktion `True` aus.

Nach erfolgreicher Verbindung mit dem WLAN, wird die Funktion `connect_mqtt` (Listing 7) aufgerufen. Zunächst wird die Funktion `set_topics` aufgerufen, die die Topics zum Senden und Empfangen der Nachrichten erstellt. Dabei werden die Topics mit `DEVICE_ID` für jedes Gerät individuell formuliert. Der `mqttClient` ist ein Objekt der Klasse `PubSubClient`. Die Funktion `setServer()` erhält als Parameter die IP-Adresse des MQTT-Brokers sowie den Port, auf dem der Broker lauscht. Diese Parameter sind als Konstanten definiert. Die Callback-Funktion wird beim Empfang einer MQTT-Nachricht ausgeführt und ist für jedes Modul unterschiedlich aufgebaut. Anschließend verbindet sich das Modul mit dem MQTT-Broker und abonniert das `SUBSCRIBE_TOPIC`.

Falls die aus dem EEPROM ausgelesenen Daten ungültig sind oder keine erfolgreiche WLAN-Verbindung aufgebaut werden kann, wird die Funktion setup_ap() (Listing 10) aufgerufen. Diese Funktion versetzt das WLAN-Modul in den Modus „Access Point“ und stellt einen Webserver bereit. Die Konfiguration des Access Points erfolgt über die Befehle WiFi.softAP() und WiFi.softAPConfig(), wobei die Parameter des Access Points in Listing 9 definiert sind.

Der Webserver wird durch das Objekt `server` der Klasse `AsyncWebServer` zur Verfügung gestellt. Der Webserver wird mit dem Befehl `server.begin()` gestartet. Die HTTP-Route wird innerhalb der Funktion server.on() festgelegt. Beim Aufruf der Root-Route (/") wird eine HTTP-GET-Anfrage bearbeitet, die mit einer HTML-Seite beantwortet wird. Diese Seite, deren Inhalt in der Variablen `html_page` (Listing 9) gespeichert ist, enthält ein Formular zur Eingabe der WLAN-Zugangsdaten.

Wenn der Benutzer auf die Taste „Submit“ drückt, wird eine HTTP-POST-Anfrage an den Server gesendet. Die Zugangsdaten werden aus dem request-Objekt ausgelesen und auf ihre maximale Länge überprüft. Wenn die Daten gültig sind, werden sie im EEPROM gespeichert. Der EEPROM wird initialisiert, die neuen Zugangsdaten werden gespeichert, und die Änderungen mit `EEPROM.commit()` übernommen. Anschließend wird das ESP-Modul mit `ESP.restart()` neu gestartet, um die neuen WLAN-Daten zu verwenden.

Diese Setup-Konfiguration wird bei allen ESP-Modulen verwendet. Je nach Modul kann diese Funktion erweitert sein. Beispielsweise wird bei Modulen mit Display angezeigt, ob die Verbindung aufgebaut wurde.

4.1.4 ESP8266

Der ESP8266 wird für die Verbindung des Arduinos mit dem WLAN verwendet. Zu seinem Aufgaben gehört das Senden und Empfangen der MQTT-Nachrichten in entsprechenden Topics. Wie in meinem Bericht zur Projektarbeit 2 [2] Kapitel 4.1 beschrieben, ist der ESP8266 über eine UART-Schnittstelle mit dem Arduino verbunden.

Beim Start des Moduls wird die Funktion `setup()` (Listing 4) ausgeführt. Zusätzlich läuchtet der blaue LED während der Einrichtung des Moduls. Der Diode läuchtet, wenn es keine Spannung auf dem GPIO 1 fällt.

```
digitalWrite(1, LOW);   // Diode einschalten
digitalWrite(1, HIGH);  // Diode ausschalten
```

Nach der erfolgreichen Einrichtung und Verbindung mit dem WLAN sowie MQTT-Broker, schaltet der Diode aus. Bei jedem Pogrammdurchlauf wird die Funktion `readSerialData` (Listing 11) aufgerufen. Stehen die Daten in der seriellen Schnittstelle zur Verfügung, werden diese zunächst als String ausgelesen und gespeichert. Der String wird auf ihre maximale Länge geprüft. Anschließend muss der String in ein char-Feld konvertiert werden. Um das Topic und den Text der Nachrichten zuzuordnen, wird nach der Position von „:“ gesucht. Alle Zeichen rechts von „:“ gehören zum Text der Nachricht und werden in der Variablen „`message`“ gespeichert. Aus den Zeichen links von „:“ wird das Topic gebildet. Arduino überträgt nur die

Namen der Haupttopics. Das ESP8266 erweitert das von Arduino übergebene Topic mit der ID des Gerätes und formuliert das `PUBLISH_TOPIC` (Zeile 22-32). Wenn das Topic aus irgendeinem Grund nicht korrekt formuliert werden kann, wird die Nachricht im Haupttopic gesendet. Damit ist sichergestellt, dass die Nachricht versendet wird. Sobald das ESP8266 die Daten von dem Arduino empfängt, wird die MQTT-Nachricht an den MQTT-Broker mit dem Befehl „`mqttClient.publish(topic, message)`“ versandt.

Die Verarbeitung der empfangenen Nachricht erfolgt in der Funktion `callback` (Listing ??). Der ESP8266 abonniert nur das Topic, das in der Variablen `SUBSCRIBE_TOPIC` definiert ist. Beim Empfang der Nachricht aus dem Topic wird geprüft, ob die Nachricht aus dem dem Gerät zugewiesenen Topic stammt. Wenn dies der Fall ist, wird der Text der Nachricht über Serial an Arduino gesendet.

4.1.5 M5Stick

Der M5Stick wird zur Überwachung der Licht- und Luftbedingungen eingesetzt. Die Idee ist, dass der Sensor die aktuelle Wetterbedingungen überwacht und diese an dem Server sendet. Für diesen Zweck soll der Sensor an dem Fenster draußen positioniert werden.

Am Modul sind die Sensoren BME680 und VCNL4040 angebunden. Der BME680 erfasst Umweltdaten: Temperatur, Luftfeuchtigkeit und Gaswiderstand. Der VCNL4040 dient als Lichtsensor und misst Lichtbedingungen: Umgebungshelligkeit (Ambient Light), weißes Licht sowie Annäherung (Proximity). Diese Daten werden an dem Server via MQTT-Nachrichten gesendet.

Für die Verwendung der beiden Sensoren im Programmcode müssen die entsprechenden Bibliotheken importiert werden und die Klassenobjekten für die Sensoren erstellt werden (Listing 1).

```
1 #include <Adafruit_BME680.h>
2 #include <Adafruit_VCNL4040.h>
3 Adafruit_BME680 bme_sensor;
4 Adafruit_VCNL4040 vcnl4040 = Adafruit_VCNL4040();
```

Listing 1: M5Stick: Sensoren

Beim Start des Moduls wird die Funktion `setup()` aufgerufen. Zusätzlich zu der im Listing 4 duchgeführte Einrichtungen müssen noch die beiden Sensoren eingerichted und im Betriebszustand gebracht werden. Dacher werden im Setup die Funktionen `vcnl_setup()` und `bme_setup()` für die Einrichtung der Sensoren aufgerufen.

Listing 2: M5Stick: vcnl_setup

4.1.6 ESP32

4.2 Softwareentwicklung

In diesem Abschnitt wird die Aufbau und Entwicklung der Software für das System beschrieben. Für die Entwicklung wurden Programmiersprachen Python und JavaScript mit mehreren Bibliotheken sowie die Markierungssprache HTML und CSS für die Visualisierung der Webseite verwendet.

Das Programm läuft auf dem Raspberry Pi. Der Programmcode ist in dem Projektordner unter `bachelor/code/spa` zu finden. Das Programm ist modular aufgebaut, wobei jedes Paket (Package) in einem eigenen Unterordner organisiert ist und eine spezifische Aufgabe erfüllt. Die Struktur des Programms sieht wie folgt aus:

```
bachelor/code/spa/
|
|--- app/
|   |--- __pycache__/
|   |--- config/
|   |--- db/
|   |--- mqtt/
|   |--- webserver/
|   |--- __init__.py
|   |--- __main__.py
|
|--- requirements.txt
|--- tailwind.config.js
```

- **app/** — Haputmodul des Programms. Enthält die Startdatei `__main__.py` sowie eine Initialisierungsdatei `__init__.py`, um das gesamte Modul als Package erkennbar zu machen.
- **confif/** — Enthält die Einstellungen für die Datenbankverbindungen, MQTT-Client, Logger und den Webserver.
- **db/** — Verwaltet der Datenbank. Enthält die Funktionen für die Verbindung zur Datenbank, die Erstellung von Tabellen und die Interaktion mit der Datenbank.
- **mqtt/** — Enthält die Logik für die Kommunikation über das MQTT-Protokoll. Enthält Code für den Empfang, die Verarbeitung und das Senden von MQTT-Nachrichten.
- **webserver/** — Stellt die API-Endpunkte und eine Webseite zur Verfügung. Enthält Klassen zur Verwaltung der Benutzerinteraktionen und zur Bereitstellung der Webseite.

Zuerst müssen die benötigten Module auf dem Server installiert werden. Die Module sind in der Datei `requirements.txt` aufgelistet. Mit dem Befehl werden die Module heruntergeladen und installiert:

```
pip install -r requirements.txt
```

4.2.1 Datenbank

Struktur der Datenbank

Die Datenbank für dieses Projekt wurde mit SQLAlchemy implementiert. Ziel war es, eine relationale Datenbank zu erstellen, die modular und flexibel aufgebaut ist, sowie die Möglichkeit bietet die Datenbankoperationen ohne direkten SQL-Abfragen durchzufüchren.

Die Funktionen zur Verwaltung der Datenbank sind in einem Paket zusammengefasst. Dieses Paket bündelt alle relevanten Komponenten, wie die Konfiguration der Datenbankverbindung, das Session-Management sowie die Definition der Datenbankmodelle.Durch diese modulare Struktur kann das Datenbankpaket in anderen Klassen oder Paketen problemlos importiert werden, um eine zentrale und einheitliche Interaktion mit der Datenbank zu ermöglichen.

Die Struktur des Packets sieht wie folgt aus:

```
bachelor/code/spa/db/
|
|--- __init__.py
```

```
|--- base.py
|--- database.db
|--- models.py
|--- queries.py
```

Die Datei `database.db` stellt die Datenbank dar. Es enthält alle Tabellen und Daten, die eingetragen wurden. Die Abbildung 1 zeigt die in der Datenbank gelegte Tabellen und deren Verknüpfungen:

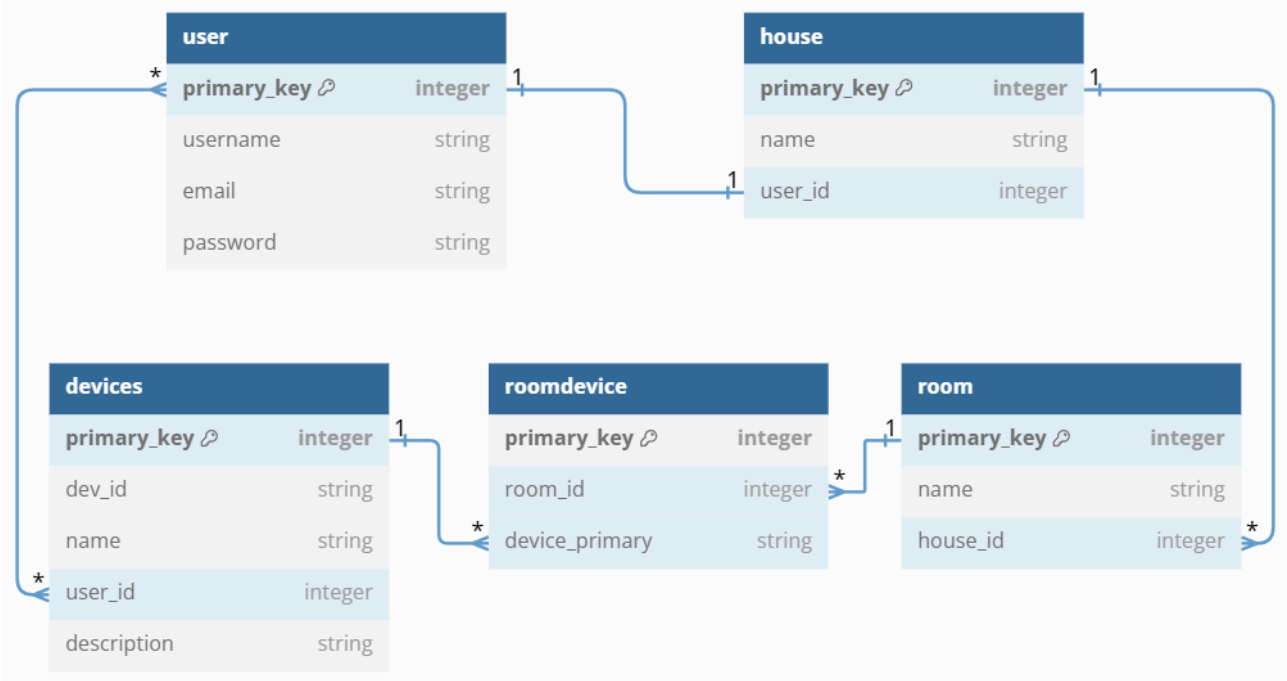

Abbildung 1: Datenbank

- **user** — Enthält die Benutzerdaten.
  - ***primary_key:*** Ein Primärschlüssel vom Typ Integer, der automatisch inkrementiert wird.
  - ***username:*** Der Benutzername, der für jeden Benutzer eindeutig ist.
  - ***email:*** Die E-Mail-Adresse des Benutzers, ebenfalls einzigartig.
  - ***password:*** Das Passwort des Benutzers, das verschlüsselt gespeichert wird.

  Ein Benutzer kann mehrere Häuser und Geräte haben.
- **house** — Speichert die vom Benutzer erstellten Häuser.
  - ***primary_key:*** Ein Primärschlüssel vom Typ Integer, der automatisch inkrementiert wird.
  - ***name:*** Der vom Benutzer erstellte Name des Hauses.
  - ***user_id:*** Ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle user verweist, um anzugeben, welchem Benutzer das Haus gehört.

  Ein Haus gehört zu einem bestimmten Benutzer. Ein Haus kann mehrere Räume enthalten.

- **room** — Speichert die vom Benutzer erstellten Räume.
  - ***primary_key:*** Ein Primärschlüssel vom Typ Integer, der automatisch inkrementiert wird.
  - ***name:*** Der vom Benutzer erstellte Name des Raums.
  - ***house_id:*** Ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle house verweist.

  Ein Raum gehört immer zu einem bestimmten Haus. Ein Raum kann mehrere Geräte enthalten.

- **devices** — Speichert die vom Benutzer angemeldeten Geräte.
  - ***primary_key:*** Ein Primärschlüssel vom Typ Integer, der automatisch inkrementiert wird.
  - ***dev_id:*** Die eindeutige, vordefinierte ID des Geräts.
  - ***name:*** Der vom Benutzer erstellte Name des Geräts.
  - ***user_id:*** Ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle user verweist und angibt, welchem Benutzer das Gerät gehört.
  - ***description:*** Eine optionale Beschreibung des Geräts.

  Ein Gerät gehört genau einem Benutzer und einem Raum. Ein anderes Gerät mit derselben Geräte-ID kann jedoch von einem anderen Benutzer angemeldet und einem anderen Raum zugeordnet werden.

- **roomdevice** — Speichert die Zuordnung der Geräte zu den Räumen.
  - ***primary_key:*** Ein Primärschlüssel vom Typ Integer, der automatisch inkrementiert wird.
  - ***room_id:*** Ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle room verweist.
  - ***device_primary:*** Ein Fremdschlüssel, der auf den Primärschlüssel der Tabelle device verweist.

Aufbau mit SQLAlchemy

Jetzt müssen diese Tabellen und deren Verknüpfungen in Form der Modellen in SQLAlchemy definiert werden. Die Models sind in der Datei `models.py` (Listing 13) definiert.

Zunächst müssen die benötigte Module importiert werden. Das Modul `bcrypt` wird für die Entschlüssertung und Verifizierung der Passwörter verwendet. Die Module aus der Bibliothek `sqlalchemy` enthalten die Funktion für die Interaktionen mit der Datenbank. Das Modul `.base` ist die Datei `base.py` (Lisitng 3). Da wird nur ein `declarative_base` importiert, und in der Variable `Base` initialisiert.

```
1  from sqlalchemy.orm import declarative_base
2  Base = declarative_base()
```

Listing 3: db: base.py

Die Tabellenmodelle werden in Klassen definiert, die von der `Base` erben. Dies ist die Standardeinstellung für SQLAlchemy. Die Variable `__tablename__` legt der Name der Tabelle fest. Die Spalten sind als Objekte der Klasse `Column` mit gegebenen Parametern definiert. Die Klasse

`UserModel` enthält eine Funktion für die Validierung der Passwörter (Zeile 17). Die Funktion bekommt ein Password-String als Parameter und vergleicht dieser mit dem im Modell gespeicherten Passwort.

In SQLAlchemy, wenn ein `ChlidModel` Modell einen Fremdschlüssel des `ParentModel` Modells beinhaltet, muss eine Beziehung mit dem `ChlidModel` Modell in der `ParentModel` Klasse definiert werden. Dies wird verwendet, um Datenbankkonflikte beim Löschen oder Aktualisieren von Einträgen im Modell `ChlidModel` zu vermeiden.

Die Klasse `HouseModel` enthält neben der Definition der Spalten eine Beziehung zwischen `HouseModel` und `RoomModel` im Attribut `rooms`. Die Beziehung ist so definiert, dass alle zugehörigen Räume gelöscht werden, wenn das Haus gelöscht wird. Außerdem ist in der Klassenvariablen `__table_args__` mit dem Befehl `UniqueConstraint('name', 'user_id')` (Zeile 27) definiert, dass die Kombination der Spalten `name` und `user_id` eindeutig sein muss. Ein Benutzer kann also nicht 2 Häuser mit dem gleichen Namen anlegen.

Die Klassen `RoomModel` und `DeviceModel` haben eine Beziehung zur Klasse `RoomDeviceModel`, und in der Klasse `RoomDeviceModel` werden die Beziehungen zu den beiden Klassen definiert. Damit wird sichergestellt, dass beim Löschen eines Raumes oder eines Gerätes auch die entsprechende Verknüpfung gelöscht wird. Dabei ist zu beachten, dass beim Löschen eines Raumes das zum Raum gehörende Gerät nicht gelöscht wird und umgekehrt.

Interaktionen mit der Datenbank

In der Datei `queries.py` (Listing 14) sind die benötigten Funktionen für die Interaktion mit der Datenbank definiert. Diese Funktionen ermöglichen das Einfügen, Löschen, Abrufen und Aktualisieren der Daten in der Datenbank.

Jede Funktion bekommt ein `AsyncSession` als erster Parameter. Über diese Session werden die Operationen in der Datenbank duchgeführt. Das erlaubt auch asynchroner Zugriff zur Datenbank. Das Erzeugen der Sessions wird im nächsten Paragraph beschrieben.

Als Beispiel werde ich die Struktur der Funktion `add_user()` näher erläutern. Diese Funktion fügt einen neuen Benutzer in die Datenbank ein. Die Funktion erhält als Parameter `AsyncSession` und die Benutzerdaten, d.h. Benutzername, Email und Passwort. Zuerst wird das Passwort verschlüsselt. Anschließend wird ein Objekt `new_user` der Klasse `UserModel` mit den übergebenen Parametern erzeugt. Dieses Objekt wird mit dem Befehl `db_session.add(new_user)` in die Datenbank eingefügt. Nach dem Einfügen muss der Befehl `db_session.commit()` aufgerufen werden, um die Änderungen in der Datenbank zu speichern. Anschließend wird das `new_user` aktualisiert und der Primärschlüssel zurückgegeben.

Beim Arbeiten mit der Datenbank müssen mögliche Fehler behandelt werden. Deshalb enthält jede Funktion einen `Try-Catch-Block`.

Ein `IntegrityError` wird ausgelöst, wenn eine Datenbankoperation gegen eine Integritätsbedingung der Datenbank verletzt. Er wird bei Verletzung einer Fremdschlüsselbeziehung, der Unique Constraints oder beim Einfügen von Daten des falschen Typs erzeugt.

Ein `SQLAlchemyError` umfasst alle Fehler, die von SQLAlchemy ausgelöst werden, wenn die Operation nicht erfolgreich war. Dieser Fehler wird bei Syntaxfehlern in der SQL-Abfrage oder bei Fehlern in der Datenbankverbindung ausgelöst.

Alle anderen möglichen Fehler, die keine `SQLAlchemy`-Fehler sind, werden im Block `Exception` gesammelt.

Bei jedem Fehler wird der Fehler geloggt und die Session zurückgesetzt, damit keine unvollständigen Daten in der Datenbank gespeichert werden. Anschließend wird eine `HTTPException` mit Status und Beschreibung des Fehlers zurückgegeben.

Initialisierung der Datenbank

Beim Import des Moduls `db` wird die Klasse `__init__` (Listing **??**) aufgerufen. In dieser Klasse wird zunächst der Logger initialisiert. Anschließend wird eine asynchrone Engine `async_engine` für die Verbindung zur Datenbank erzeugt. Mit dem Parameter `async_engine` wird das Objekt `async_session` der Klasse `Sessionmaker` erzeugt. Die Funktion `create_tables()` initialisiert alle Tabellen in den Klassenmodellen, die von der Klasse `Base` erben. Die Funktion `get_session()` gibt eine asynchrone Session aus der `async_session` zurück. Mit dieser Session wird dann auf die Datenbank zugegriffen.

4.2.2 Webserver

4.2.3 Frontend

4.2.4 Integration der Komponenten

MQtt client, erhalten Daten von Broker etc

5 Ergebnisse und Diskussion

6 Quellen

Literatur

[1] O. Baida, *Anbindung der Sensoren und Aktoren an den Arduino zur Realisierung eines Sicherheitssystems*, Projektarbeit 1, 2024.

[2] O. Baida, Projektarbeit 2 *Sicherheitssystem fur das Haus basierend auf Arduino, ESP8266 & Raspberry Pi* https://github.com/oleksiibaida/PA2.git

[3]

[4] Statista, *„Smart Home - Anzahl der Haushalte in Deutschland 2028“*, Zugegriffen: 13. Januar 2025. [Online]. Verfügbar unter https://de.statista.com/prognosen/885611/anzahl-der-smart-home-haushalte-in-deutschland

[5] J. Breithut, *„Strom und Heizung: Wann ein Smart Home wirklich beim Energiesparen hilft“*, Der Spiegel, 17. Juli 2022. Zugegriffen: 13. Januar 2025. [Online]. Verfügbar unter: https://www.spiegel.de/netzwelt/gadgets/strom-und-heizung-wann-ein-smart-home-wirklich-beim-energiesparen-hilft-a-ffb4b710-

Links zur verwendeten Hardware:

[6] Arduino.cc, *Arduino UNO*, https://docs.arduino.cc/hardware/uno-rev3/

[7] Raspberry Pi Foundation, *Raspberry Pi 1 B+*, https://www.raspberrypi.com/products/raspberry-pi-1-model-b-plus/

[8] Espressif, *ESP8266*, https://www.espressif.com/, https://www.electronicwings.com/sensors-modules/esp8266-wifi-module

Links zur verwendeten Software:

[9] Dr Andy Stanford-Clark, Arlen Nipper, *Message Queuing Telemetry Transport*, https://mqtt.org/

[10] Guido van Rossum, Python Software Foundation, *Python*, https://www.python.org/

[11] Telegram FZ-LLC, *Telegram Messenger*, https://github.com//telegramdesktop/tdesktop

Linux-Packete:

[12] Jouni Malinen, *hostapd*, https://w1.fi/hostapd/, Zugriff am: 19. September 2024.

[13] Simon Kelley, *dnsmasq*, https://dnsmasq.org/doc.html, Zugriff am: 20. September 2024.

[14] Eclipse Foundation, *Eclipse Mosquitto*, https://mosquitto.org/

ESP- und Arduino-Bibliotheken

[15] Knolleary, *PubSubClient*, https://pubsubclient.knolleary.net/, Zugriff am: 21. Oktober 2024.

[16] ESPWIFI.h, https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

[17] EEPROM.h, https://docs.arduino.cc/learn/built-in-libraries/eeprom/

[18] Keypad.h https://docs.arduino.cc/libraries/keypad/

[19] R. Scholz, *Syncloop*, Persönliche Mitteilungen

Python-Bibliotheken

[20] Pierre Fersing, Roger Light *paho-mqtt*, https://pypi.org/project/paho-mqtt/, Zugriff am: 21. Oktober 2024.

[21] Open Source, *python-telegram-bot*, https://docs.python-telegram-bot.org/en/v21.6/

[22] Python Software Foundation, *json*, https://docs.python.org/3/library/json.html

[23] Python Software Foundation, *threading*, https://docs.python.org/3/library/threading.html

[24] Python Software Foundation, *queue*, https://docs.python.org/3/library/queue.html

[25] Gerhard Häring, *sqlite3*, https://docs.python.org/3/library/sqlite3.html

[26] Lawrence Hudson, *pyzbar*, https://github.com/NaturalHistoryMuseum/pyzbar/

[27] Intel, *OpenCV*, https://github.com/opencv/opencv-python

[28] Aio-Libs, *aiohttp*, https://github.com/aio-libs/aiohttp

Abbildungsverzeichnis

Tabellenverzeichnis

7 Programmcode

```
void setup()
{
  Serial.begin(9600);
  // GET WiFi Daten aus EEPROM
  EEPROM.begin(128);
  char eeprom_ssid[MAX_SSID_LENGTH] = {0};
  char eeprom_password[MAX_PASSWORD_LENGTH] = {0};
  EEPROM.get(0, eeprom_ssid);
  EEPROM.get(32, eeprom_password);
  EEPROM.end();
  // Daten gefunden
  if (is_valid_string(eeprom_ssid, MAX_SSID_LENGTH) &&
     is_valid_string(eeprom_password, MAX_PASSWORD_LENGTH))
  {
    if (connect_wifi(eeprom_ssid, eeprom_password))
    {
      connect_mqtt();
      return;
    }
  }
  // keine WLAN-DAten gefunden
  setup_ap();
}
```

Listing 4: ESP: setup

```
bool is_valid_string(char *data, int max_length)
{
  if (strlen(data) == 0 or strlen(data) > max_length)
    return false;
  for (int i = 0; i < max_length; i++)
  {
    if (data[i] == '\0')
      return true; // End of valid String
    if (data[i] == 0xFF) // Default EEPROM
      return false;
  }
  return false;
}
```

Listing 5: ESP: is_valid_string

```
bool connect_wifi(char *ssid, char *password)
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  for (uint8_t i = 0; i < wifi_repeat; i++)
  {
    if (WiFi.status() == WL_CONNECTED)
    {
      Serial.println(WiFi.localIP());
```

```
        return true;
      }
      delay(1000);
    }
    return false;
  }
```

Listing 6: ESP: connect_wifi

```
  void mqtt_connect()
  {
    set_topics();
    mqttClient.setServer(MQTT_BROKER_ADRRESS, MQTT_PORT);
    mqttClient.setCallback(callback);
    for (int i = 0; i < wifi_repeat; i++)
    {
      if (mqttClient.connect(DEVICE_ID))
        mqttClient.subscribe(SUBSCRIBE_TOPIC);
        return;
      else
        delay(1000);
    }
  }
```

Listing 7: ESP: connect_mqtt

```
  void set_topics()
  {
    SUBSCRIBE_TOPIC = (char *)malloc(strlen(TOPIC_COMMAND) + strlen(
       DEVICE_ID) + 2);
    PUBLISH_TOPIC = (char *)malloc(strlen("data") + strlen(DEVICE_ID)
       + 2);

    strcpy(SUBSCRIBE_TOPIC, TOPIC_COMMAND);
    strcat(SUBSCRIBE_TOPIC, "/");
    strcat(SUBSCRIBE_TOPIC, DEVICE_ID);

    strcpy(PUBLISH_TOPIC, "data");
    strcat(PUBLISH_TOPIC, "/");
    strcat(PUBLISH_TOPIC, DEVICE_ID);
  }
```

Listing 8: ESP: set_topics

```
  const char AP_SSID[] = "ESP8266";
  const char AP_PASSWORD[] = "setupesp";
  IPAddress ap_ip(10, 0, 0, 1);
  IPAddress ap_gateway(10, 0, 0, 1);
  IPAddress ap_subnet(255, 255, 255, 0);
  AsyncWebServer server(80);
  const String html_page = R"rawliteral(
    <html>
    <body>
      <h2>Wi-Fi Configuration</h2>
      <form action="/save" method="POST">
```

```
        SSID:<br>
        <input type="text" name="ssid" required><br>
        Password:<br>
        <input type="password" name="password" required><br><br>
        <input type="submit" value="Submit">
      </form>
    </body>
    </html>
  )rawliteral";
```

Listing 9: ESP8266: ap_konfiguration

```
void setup_ap()
{
  WiFi.mode(WIFI_AP);
  WiFi.softAP(AP_SSID, AP_PASSWORD);
  WiFi.softAPConfig(ap_ip, ap_gateway, ap_subnet);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(200, "text/html", html_page); });

  server.on("/save", HTTP_POST, [](AsyncWebServerRequest *request)
            {
              // get input data
              String ssid = request->getParam("ssid", true)->value()
                 ;
              String password = request->getParam("password", true)
                 ->value();
              if (ssid.length() > MAX_SSID_LENGTH - 1 || password.
                 length() > MAX_PASSWORD_LENGTH - 1)
              {
                return;
              }
              // String in char[]
              char new_ssid[MAX_SSID_LENGTH] = {0};
              strncpy(new_ssid, ssid.c_str(), MAX_SSID_LENGTH - 1);
              char new_password[MAX_PASSWORD_LENGTH] = {0};
              strncpy(new_password, password.c_str(),
                 MAX_PASSWORD_LENGTH - 1);

              // in EEPROM speichern
              EEPROM.begin(128);
              EEPROM.put(0, new_ssid);
              EEPROM.put(32, new_password);
              EEPROM.commit();
              EEPROM.end();

              request->send(200, "text/html", "WiFi saved. Rebooting
                 ...");
              ESP.restart(); });

  server.begin();
}
```

Listing 10: ESP: setup_ap

```
void readSerialData()
{
  if (Serial.available() > 0)
  {
    String readString = "";
    // Lese Daten aus Serial als String ab
    readString = Serial.readStringUntil('\n');
    if (sizeof(readString) > buss_serial)
      return;
    // String in char-Feld konvertieren
    char readSerialChar[readString.length() + 1];
    readString.toCharArray(readSerialChar, readString.length() + 1);
    // Suche Position von ':'
    char *delim_pos = strchr(readSerialChar, ':');
    if (delim_pos != NULL)
    {
      size_t topic_length = delim_pos - readSerialChar;
      char topic[topic_length + 1];
      strncpy(topic, readSerialChar, topic_length);
      topic[topic_length] = '\0';
      char *message = delim_pos + 1;
      PUBLISH_TOPIC = (char *)malloc(strlen(CLIENT_ID) + strlen(
          topic) + 2);
      if (PUBLISH_TOPIC == NULL)
      {
        PUBLISH_TOPIC = topic;
      }
      else
      {
        strcpy(PUBLISH_TOPIC, topic);
        strcat(PUBLISH_TOPIC, "/");
        strcat(PUBLISH_TOPIC, CLIENT_ID);
      }
      // MQTT-Nachricht senden
      mqttClient.publish(PUBLISH_TOPIC, message);
    }
    else // kein : gefunden
      return;
  }
}
```

Listing 11: ESP8266: readSerialData

```
void callback(char *topic, byte *payload, unsigned int length)
{
  String id = String(topic).substring(String(topic).indexOf('/') +
      1);
  if (id == CLIENT_ID)
  {
    String text = "";
    for (int i = 0; i < length; i++)
    {
      text += (char)payload[i];
```

```
      }
      text.trim();
      // Print in Arduino
      Serial.println(text);
    }
  }
```

Listing 12: ESP8266: callback

```
import bcrypt
from sqlalchemy import Column, Integer, String, ForeignKey,
   PrimaryKeyConstraint
from sqlalchemy.orm import relationship
from sqlalchemy.schema import UniqueConstraint
from .base import Base
from app.config import Config
__logger = Config.logger_init()


class UserModel(Base):
    __tablename__ = 'user'
    primary_key = Column(Integer, primary_key=True, autoincrement=
       True, nullable=False)
    username = Column(String(50), unique=True, nullable=False)
    email = Column(String(100), unique=True, nullable=False)
    password = Column(String(200), nullable=False)

    def verify_password(self, password:str):
        return bcrypt.checkpw(password.encode('utf-8'), self.
           password)

class HouseModel(Base):
    __tablename__ = 'house'
    primary_key = Column(Integer, primary_key=True, autoincrement=
       True, nullable=False)
    name = Column(String(50), nullable=False)
    user_id = Column(Integer, ForeignKey("user.primary_key",
       ondelete='CASCADE'), nullable=False)
    rooms = relationship("RoomModel", backref="house", cascade="all,
        delete-orphan", lazy='selectin')

    __table_args__ = (UniqueConstraint('name', 'user_id'),)

class RoomModel(Base):
    __tablename__ = 'room'
    primary_key = Column(Integer, primary_key=True, autoincrement=
       True, nullable=False)
    name = Column(String(50), nullable=False)
    house_id = Column(Integer, ForeignKey('house.primary_key',
       ondelete='CASCADE'), nullable=False)

    devices = relationship("RoomDeviceModel", back_populates="room",
        cascade="all, delete-orphan", lazy='selectin')

```

```
37      __table_args__ = (UniqueConstraint('name', 'house_id'),)
38
39  class DeviceModel(Base):
40      __tablename__ = 'device'
41      primary_key = Column(Integer, primary_key=True, autoincrement=
            True, nullable=False)
42      dev_id = Column(String(10), nullable=False)
43      name = Column(String(50), nullable=False)
44      user_id = Column(Integer, ForeignKey('user.primary_key',
            ondelete='CASCADE'), nullable=False, unique=False)
45      description = Column(String(250), nullable=True)
46      dev_rooms = relationship("RoomDeviceModel", back_populates="
            device", cascade="all, delete-orphan", lazy='selectin')
47
48      __table_args__ = (UniqueConstraint('user_id', 'dev_id'),)
49
50  class RoomDeviceModel(Base):
51      __tablename__ = 'room_device'
52      primary_key = Column(Integer, primary_key=True, autoincrement=
            True, nullable=False)
53      room_id = Column(Integer, ForeignKey('room.primary_key',
            ondelete='CASCADE'), nullable=False)
54      device_primary = Column(Integer, ForeignKey('device.primary_key'
            , ondelete='CASCADE'), nullable=False)
55
56      room = relationship("RoomModel", back_populates="devices")
57      device = relationship("DeviceModel", back_populates="dev_rooms")
58
59      __table_args__ = (UniqueConstraint('room_id', 'device_primary')
            ,)
```

Listing 13: db: models.py

```
1   import bcrypt
2   from app.config import Config
3   from sqlalchemy import select, insert, update, delete, func
4   from sqlalchemy.exc import IntegrityError, NoResultFound,
        SQLAlchemyError
5   from sqlalchemy.ext.asyncio import AsyncSession
6   from sqlalchemy.orm import joinedload
7   from .models import UserModel, HouseModel, RoomModel, DeviceModel,
        RoomDeviceModel
8   from fastapi import Depends, HTTPException
9   _logger = Config.logger_init()
10
11  async def add_user(db_session: AsyncSession, username: str, email:
        str, password: str):
12      """
13      Adds new user to Database. SIGNUP.
14      :param db_session: SQLAlchemy AsyncSession
15      :param username, email, password: user data
16      :return: True -> user added; IntegityError -> username or email
            already in DB; HTTPExcepion
17      """
```

```
18        if username is None or email is None or password is None:
19            _logger.error("All user data must be provided")
20            return False
21        try:
22            password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.
                  gensalt())
23            new_user = UserModel(username=username, email=email,
                  password=password)
24            _logger.debug(f'ADD U_NAME {new_user.username} START')
25            db_session.add(new_user)
26            await db_session.commit()
27            await db_session.refresh(new_user)
28            _logger.debug(f'ADD U_NAME {new_user.username} COMPLETED')
29            return new_user.primary_key
30        except IntegrityError as e:
31            _logger.error(f"IntegrityError: {e}")
32            await db_session.rollback()
33            raise HTTPException(status_code=422, detail="ALREADY EXISTS"
                  )
34        except SQLAlchemyError as e:
35            _logger.error(f"SQLAlchemyError: {e}")
36            await db_session.rollback()
37            raise HTTPException(status_code=500, detail="DATABASE ERROR"
                  )
38        except Exception as e:
39            _logger.error(f"Exception: {e}")
40            await db_session.rollback()
41            raise HTTPException(status_code=500, detail="UNEXPECTED
                  DATABASE ERROR")
42
43    async def get_user_data(db_session: AsyncSession, user_primary: int
          = None, username: str = None, email: str = None):
44        if not user_primary and not username and not email:
45            _logger.error(msg="EMPTY SET")
46            raise ValueError()
47        query = select(UserModel)
48        if user_primary:
49            query = query.where(UserModel.primary_key == user_primary)
50        if username:
51            query = query.where(UserModel.username == username)
52        if email:
53            query = query.where(UserModel.email == email)
54        try:
55            res = await db_session.execute(query)
56            return res.scalars().first()
57        except IntegrityError as e:
58            _logger.error(f"IntegrityError: {e}")
59            await db_session.rollback()
60            raise HTTPException(status_code=422, detail="NOT FOUND")
61        except SQLAlchemyError as e:
62            _logger.error(f"SQLAlchemyError: {e}")
63            await db_session.rollback()
64            raise HTTPException(status_code=500, detail="DATABASE ERROR"
```

```
            )
    except Exception as e:
        _logger.error(f"Exception: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")

async def add_new_house(db_session: AsyncSession, user_primary: int,
    house_name: str):
    try:
        new_house = HouseModel(user_id=user_primary, name=house_name
            )
        db_session.add(new_house)
        await db_session.commit()
        await db_session.refresh(new_house)
        _logger.info(f"U_ID: {user_primary} ADD HOUSE_ID {new_house.
            primary_key}")
        return new_house
    except IntegrityError as e:
        _logger.error(f"IntegrityError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=422, detail="ALREADY EXISTS"
            )
    except SQLAlchemyError as e:
        _logger.error(f"SQLAlchemyError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
        _logger.error(f"Exception: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")

async def delete_house(db_session: AsyncSession, house_id):
    try:
        house = await db_session.get(HouseModel, house_id)
        await db_session.delete(house)
        await db_session.commit()
        return True
    except IntegrityError as e:
        _logger.error(f"IntegrityError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=422, detail="NOT FOUND")
    except SQLAlchemyError as e:
        _logger.error(f"SQLAlchemyError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
        _logger.error(f"Exception: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="UNEXPECTED
```

```
            DATABASE ERROR")

async def get_house(db_session: AsyncSession, house_primary: int):
    try:
        return await db_session.get(HouseModel, house_primary)
    except NoResultFound:
        _logger.error(f'HOUSE_ID {house_primary} HOUSE NOT FOUND')
        raise HTTPException(status_code=404, detail='NOT FOUND')
    except IntegrityError as e:
        _logger.error(f"IntegrityError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=422, detail="NOT FOUND")
    except SQLAlchemyError as e:
        _logger.error(f"SQLAlchemyError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
        _logger.error(f"Exception: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")

async def get_houses_on_user(db_session: AsyncSession, user_primary:
    int):
    try:
        stmt = (
            select(HouseModel)
            .options(
                joinedload(HouseModel.rooms)
                .joinedload(RoomModel.devices)
                .joinedload(RoomDeviceModel.device)
                .joinedload(DeviceModel.dev_rooms)
            )
            .filter(HouseModel.user_id == user_primary)
        )
        # print(str(stmt))
        houses = await db_session.execute(stmt)
        houses = houses.scalars().unique().all()
        return houses
    except NoResultFound:
        _logger.error(f'U_ID {user_primary} HOUSE NOT FOUND')
        raise HTTPException(status_code=404, detail='NOT FOUND')
    except IntegrityError as e:
        _logger.error(f"IntegrityError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=422, detail="NOT FOUND")
    except SQLAlchemyError as e:
        _logger.error(f"SQLAlchemyError: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
```

```
157             _logger.error(f"Exception: {e}")
158             await db_session.rollback()
159             raise HTTPException(status_code=500, detail="UNEXPECTED
                    DATABASE ERROR")
160
161     async def get_house_by_room(db_session: AsyncSession, room_id: int):
162         try:
163             stmt = select(HouseModel).join(RoomModel, RoomModel.house_id
                        == HouseModel.primary_key).where(RoomModel.primary_key
                    == room_id)
164             res = await db_session.execute(stmt)
165             return res.scalar_one_or_none()
166         except NoResultFound as e:
167             _logger.error(f"NoResultFound:{e}")
168             raise HTTPException(404, 'NOT FOUND')
169         except IntegrityError as e:
170             _logger.error(f"IntegrityError: {e}")
171             await db_session.rollback()
172             raise HTTPException(status_code=422, detail="NOT FOUND")
173         except SQLAlchemyError as e:
174             _logger.error(f"SQLAlchemyError: {e}")
175             await db_session.rollback()
176             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                    )
177         except Exception as e:
178             _logger.error(f"Exception: {e}")
179             await db_session.rollback()
180             raise HTTPException(status_code=500, detail="UNEXPECTED
                    DATABASE ERROR")
181
182     async def verify_house_owner(db_session: AsyncSession, user_primary:
            int, house_id: int):
183         try:
184             house = await db_session.get(HouseModel, house_id)
185             if house:
186                 return house.user_id == user_primary
187         except NoResultFound as e:
188             _logger.error(f"NoResultFound: {e}")
189             raise HTTPException(404, 'NOT FOUND')
190         except IntegrityError as e:
191             _logger.error(f"IntegrityError: {e}")
192             await db_session.rollback()
193             raise HTTPException(status_code=422, detail="NOT FOUND")
194         except SQLAlchemyError as e:
195             _logger.error(f"SQLAlchemyError: {e}")
196             await db_session.rollback()
197             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                    )
198         except Exception as e:
199             _logger.error(f"Exception: {e}")
200             await db_session.rollback()
201             raise HTTPException(status_code=500, detail="UNEXPECTED
                    DATABASE ERROR")
```

```
202
203 async def add_new_room(db_session: AsyncSession, house_id: int,
        room_name: str):
204     try:
205         new_room = RoomModel(name = room_name, house_id = house_id)
206         db_session.add(new_room)
207         await db_session.commit()
208         _logger.info(f"ADD ROOM_NAME {room_name} HOUSE_ID {house_id}
                ")
209         return True
210     except NoResultFound as e:
211         _logger.error(f"NoResultFound:{e}")
212         raise HTTPException(404, 'NOT FOUND')
213     except IntegrityError as e:
214         _logger.error(f"IntegrityError: {e}")
215         await db_session.rollback()
216         raise HTTPException(status_code=422, detail="ALREADY EXISTS"
                )
217     except SQLAlchemyError as e:
218         _logger.error(f"SQLAlchemyError: {e}")
219         await db_session.rollback()
220         raise HTTPException(status_code=500, detail="DATABASE ERROR"
                )
221     except Exception as e:
222         _logger.error(f"Exception: {e}")
223         await db_session.rollback()
224         raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")
225
226 async def delete_room(db_session: AsyncSession, room_id: int):
227     try:
228         # get RoomModel from DB
229         del_room = await db_session.execute(
230             select(RoomModel)
231             .where(RoomModel.primary_key == room_id)
232         )
233         # fetch result
234         del_room = del_room.scalars().one()
235
236         await db_session.delete(del_room)
237         await db_session.commit()
238         return True
239     except NoResultFound as e:
240         _logger.error(f"NoResultFound:{e}")
241         raise HTTPException(404, 'NOT FOUND')
242     except IntegrityError as e:
243         _logger.error(f"IntegrityError: {e}")
244         await db_session.rollback()
245         raise HTTPException(status_code=422, detail="NOT FOUND")
246     except SQLAlchemyError as e:
247         _logger.error(f"SQLAlchemyError: {e}")
248         await db_session.rollback()
249         raise HTTPException(status_code=500, detail="DATABASE ERROR"
```

```
            )
    except Exception as e:
        _logger.error(f"An unexpected error: {e}")
        await db_session.rollback()
        raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")

async def add_new_device(db_session: AsyncSession, user_id: int,
    device_data):
    try:
        if device_data.dev_id is None or device_data.name is None or
            user_id is None:
            return False
        new_dev = DeviceModel(
            dev_id = device_data.dev_id,
            name = device_data.name,
            user_id = user_id,
            description = device_data.description
        )
        db_session.add(new_dev)
        await db_session.commit()
        await db_session.refresh(new_dev)
        return new_dev
    except NoResultFound as e:
        _logger.error(f"NoResultFound:{e}")
        raise HTTPException(404, 'NOT FOUND')
    except IntegrityError as e:
        await db_session.rollback()
        _logger.error(f"IntegrityError: {e}")
        raise HTTPException(status_code=422, detail="ALREADY EXISTS"
            )
    except SQLAlchemyError as e:
        await db_session.rollback()
        _logger.error(f"SQLAlchemyError: {e}")
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
        await db_session.rollback()
        _logger.error(f"An unexpected error: {e}")
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )

async def get_device(db_session:AsyncSession, user_id: int,
    primary_key: int = None, dev_id: str = None, name: str = None):
    """
    Get Device data
    :param db_session: AsyncSession to access DB
    :param user_id: Owner of the Sensor
    :param id: Optional; device.primary_key in DB
    :param dev_id: Optional; device.dev_id (Factory ID)
    :param name: OPtional Device Name
    """

```

```
295         if not primary_key and not dev_id and not name:
296             return None
297         stmt = select(DeviceModel)
298         if primary_key:
299             stmt = stmt.where(DeviceModel.primary_key == primary_key,
                    DeviceModel.user_id == user_id)
300         if dev_id:
301             stmt = stmt.where(DeviceModel.dev_id == dev_id, DeviceModel.
                    user_id == user_id)
302         if name:
303             stmt = stmt.where(DeviceModel.name == name, DeviceModel.
                    user_id == user_id)
304         try:
305             res = await db_session.execute(stmt)
306             return res.scalar_one_or_none()
307         except NoResultFound:
308             return None
309         except IntegrityError as e:
310             await db_session.rollback()
311             _logger.error(f"IntegrityError: {e.orig}")
312             raise HTTPException(status_code=422, detail="NOT FOUND")
313         except SQLAlchemyError as e:
314             await db_session.rollback()
315             _logger.error(f"SQLAlchemyError: {e}")
316             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                    )
317         except Exception as e:
318             await db_session.rollback()
319             _logger.error(f"An unexpected error: {e}")
320             raise HTTPException(status_code=500, detail="UNEXPECTED
                    DATABASE ERROR")
321
322     async def get_devices_on_user(db_session: AsyncSession, user_primary
            : int):
323         try:
324             stmt = (
325                 select(DeviceModel)
326                 .options(
327                     joinedload(DeviceModel.dev_rooms)
328                     .joinedload(RoomDeviceModel.room)
329                     .joinedload(RoomModel.devices)
330                 )
331                 .filter(DeviceModel.user_id == user_primary)
332             )
333
334             devices = await db_session.execute(stmt)
335             return devices.scalars().unique().all()
336         except NoResultFound:
337             _logger.error(f'U_ID {user_primary} HOUSE NOT FOUND')
338             raise HTTPException(status_code=404, detail='NOT FOUND')
339         except IntegrityError as e:
340             _logger.error(f"IntegrityError: {e}")
341             await db_session.rollback()
```

```
342             raise HTTPException(status_code=422, detail="NOT FOUND")
343         except SQLAlchemyError as e:
344             _logger.error(f"SQLAlchemyError: {e}")
345             await db_session.rollback()
346             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                )
347         except Exception as e:
348             _logger.error(f"Exception: {e}")
349             await db_session.rollback()
350             raise HTTPException(status_code=500, detail="UNEXPECTED
                DATABASE ERROR")
351
352     async def update_device(db_session: AsyncSession, user_id:int,
            new_device_data):
353         try:
354             # TODO also change room
355             if new_device_data.primary is None:
356                 _logger.error("DEVICE_PRIMARY IS NOT PROVIDED")
357                 raise HTTPException(status_code=400, detail="
                    DEVICE_PRIMARY IS NOT PROVIDED")
358             if new_device_data.name is None and new_device_data.
                description is None:
359                 _logger.error("NEW DATA IS NOT PROVIDED")
360                 raise HTTPException(status_code=400, detail="NEW DATA IS
                     NOT PROVIDED")
361             device = await db_session.get(DeviceModel, new_device_data.
                primary)
362             if device:
363                 if device.user_id != user_id:
364                     _logger.critical(f"UNAUTHORIZED ACCESS U_ID {user_id
                        } ON DEVICE {device.primary_key}")
365                     raise HTTPException(status_code=401, detail="User is
                         not owner of this device")
366                 if new_device_data.name:
367                     device.name = new_device_data.name
368                 if new_device_data.description:
369                     device.description = new_device_data.description
370                 await db_session.commit()
371             return device
372         except NoResultFound:
373             _logger.error(f'{new_device_data.primary} DEVICE NOT FOUND')
374             raise HTTPException(status_code=404, detail='NOT FOUND')
375         except IntegrityError as e:
376             _logger.error(f"IntegrityError: {e}")
377             await db_session.rollback()
378             raise HTTPException(status_code=422, detail="NOT FOUND")
379         except SQLAlchemyError as e:
380             _logger.error(f"SQLAlchemyError: {e}")
381             await db_session.rollback()
382             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                )
383         except HTTPException as e:
384             raise e
```

```
385         except Exception as e:
386             _logger.error(f"Exception: {e}")
387             await db_session.rollback()
388             raise HTTPException(status_code=500, detail="UNEXPECTED
                DATABASE ERROR")
389
390     async def delete_device(db_session: AsyncSession, device_primary_key
        : int, device_id: str = None):
391         try:
392             if not device_id and not device_primary_key:
393                 _logger.error("EMPTY SET")
394                 raise HTTPException(status_code=400, detail="NO DEVICE
                    DATA PROVIDED")
395             # del_device = DeviceModel(primary_key = device_primary_key,
                 dev_id = device_id)
396             del_device = await db_session.get(DeviceModel,
                device_primary_key)
397             if del_device:
398                 await db_session.delete(del_device)
399                 await db_session.commit()
400                 return True
401             else:
402                 raise HTTPException(404, 'NOT FOUND')
403         except NoResultFound as e:
404             _logger.error(f"NoResultFound:{e}")
405             raise HTTPException(404, 'NOT FOUND')
406         except IntegrityError as e:
407             _logger.error(f"IntegrityError: {e}")
408             await db_session.rollback()
409             raise HTTPException(status_code=422, detail="NOT FOUND")
410         except SQLAlchemyError as e:
411             _logger.error(f"SQLAlchemyError: {e}")
412             await db_session.rollback()
413             raise HTTPException(status_code=500, detail="DATABASE ERROR"
                )
414         except Exception as e:
415             _logger.error(f"An unexpected error: {e}")
416             await db_session.rollback()
417             raise HTTPException(status_code=500, detail="UNEXPECTED
                DATABASE ERROR")
418
419     async def add_room_device(db_session: AsyncSession, device_primary,
        room_id):
420         try:
421             rd = RoomDeviceModel(room_id = room_id, device_primary =
                device_primary)
422             db_session.add(rd)
423             await db_session.commit()
424             _logger.info(f'DEV_ID {device_primary} ADDED TO ROOM_ID {
                room_id}')
425             return True
426         except IntegrityError as e:
427             await db_session.rollback()
```

```
            _logger.error(f"IntegrityError: {e.orig}")
            raise HTTPException(status_code=422, detail="NOT FOUND")
        except SQLAlchemyError as e:
            await db_session.rollback()
            _logger.error(f"SQLAlchemyError: {e}")
            raise HTTPException(status_code=500, detail="DATABASE ERROR"
                )
        except Exception as e:
            await db_session.rollback()
            _logger.error(f"An unexpected error: {e}")
            raise HTTPException(status_code=500, detail="UNEXPECTED
                DATABASE ERROR")

async def delete_room_device(db_session: AsyncSession, room_id: int,
    device_primary_key: int):
    try:
        # get room_device from DB
        del_rd = await db_session.execute(
            select(RoomDeviceModel).where(
                RoomDeviceModel.room_id==room_id,
                RoomDeviceModel.device_primary==device_primary_key
            )
        )
        del_rd = del_rd.scalars().first()
        if del_rd:
            await db_session.delete(del_rd)
            await db_session.commit()
            return True
        raise HTTPException(status_code=422, detail="NOT FOUND")
    except IntegrityError as e:
        await db_session.rollback()
        _logger.error(f"IntegrityError: {e.orig}")
        raise HTTPException(status_code=422, detail="NOT FOUND")
    except SQLAlchemyError as e:
        await db_session.rollback()
        _logger.error(f"SQLAlchemyError: {e}")
        raise HTTPException(status_code=500, detail="DATABASE ERROR"
            )
    except Exception as e:
        await db_session.rollback()
        _logger.error(f"An unexpected error: {e}")
        raise HTTPException(status_code=500, detail="UNEXPECTED
            DATABASE ERROR")
```

Listing 14: db: queries.py

```
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from sqlalchemy.orm import sessionmaker
from app.config import Config
from fastapi import Depends
from . import models
from .base import Base

logger = Config.logger_init()
```

```
logger.info("START DB")
__all__=["Base", "models"]

async_engine = create_async_engine(Config.SQLALCHEMY_DATABASE_URL)

async_session = sessionmaker(
    async_engine, expire_on_commit=False, class_=AsyncSession
)

async def create_tables():
    async with async_engine.begin() as connection:
        await connection.run_sync(Base.metadata.create_all)
        await connection.commit()
        logger.debug("TABLES CREATED")

async def get_session():
    async with async_session() as session:
        yield session
```

Listing 15: db: __init__.py