Akademia Górniczo-Hutnicza im Stanisława Staszica w Krakowie
Wydział Inżynierii Metali i Informatyki Przemysłowej

Projekt

Aplikacja do zarządzania domową siecią IoT

Autor: Waldemar Świder
Kierunek studiów: Informatyka Techniczna

Kraków, 2022

Spis treści

1. Informacje wstępne

W dzisiejszym dynamicznym świecie technologicznym, Internet Rzeczy (IoT) odgrywa znaczącą rolę w naszym życiu codziennym. Koncepcja IoT polega na połączeniu różnych urządzeń z siecią internetową, umożliwiając im komunikację, wymianę danych oraz zdalne sterowanie nimi. Domowa sieć IoT skupia się na wykorzystaniu tych zaawansowanych technologii w kontekście zarządzania urządzeniami w naszym domu.

Zastosowanie domowej sieci IoT przynosi szereg korzyści dla użytkowników. Inteligentne urządzenia, takie jak oświetlenie, termostaty, zamki do drzwi, kamery monitoringu, głośniki czy nawet urządzenia kuchenne, mogą być połączone i sterowane za pomocą aplikacji na smartfonie lub innym urządzeniu. Dzięki temu możliwe jest zdalne zarządzanie nimi, tworzenie harmonogramów działania, monitorowanie zużycia energii oraz integracja z innymi systemami w domu, np. systemem alarmowym. Aplikacje zarządzania domową siecią IoT pełnią kluczową rolę w umożliwianiu użytkownikom pełnej kontroli nad swoimi inteligentnymi urządzeniami. Dają one możliwość wygodnego zarządzania i monitorowania wielu urządzeń jednocześnie, nawet gdy użytkownik jest poza domem. To pozwala na optymalne wykorzystanie zasobów, oszczędność energii oraz zapewnienie większego bezpieczeństwa.

Kiedy projektujemy aplikację zarządzania domową siecią IoT, istotne jest uwzględnienie różnorodności urządzeń i protokołów komunikacyjnych wykorzystywanych w domowej sieci. Różne urządzenia mogą korzystać z różnych technologii, takich jak Wi-Fi, Bluetooth, Zigbee czy Z-Wave, dlatego aplikacja powinna być zdolna do obsługi tych różnic i zapewnić spójne zarządzanie nimi. Ważnym aspektem projektu jest również zabezpieczenie aplikacji i urządzeń przed potencjalnymi zagrożeniami. Bezpieczeństwo jest niezwykle istotne, ponieważ domowa sieć IoT może być narażona na ataki hakerskie, wycieki danych czy naruszenia prywatności. Projektowanie aplikacji powinno uwzględniać protokoły komunikacyjne, uwierzytelnianie, szyfrowanie danych i inne mechanizmy, które zapewnią bezpieczne korzystanie z sieci IoT.

W kolejnych rozdziałach projektu, szczegółowo zostaną omówione kwestie związane z analizą wymagań, projektowaniem interfejsu użytkownika, implementacją aplikacji, testowaniem oraz wdrażaniem. Priorytetem będzie stworzenie aplikacji, która będzie łatwa w obsłudze, intuicyjna i spełniająca oczekiwania użytkowników dotyczące zarządzania domową siecią IoT.

2. Cel projektu

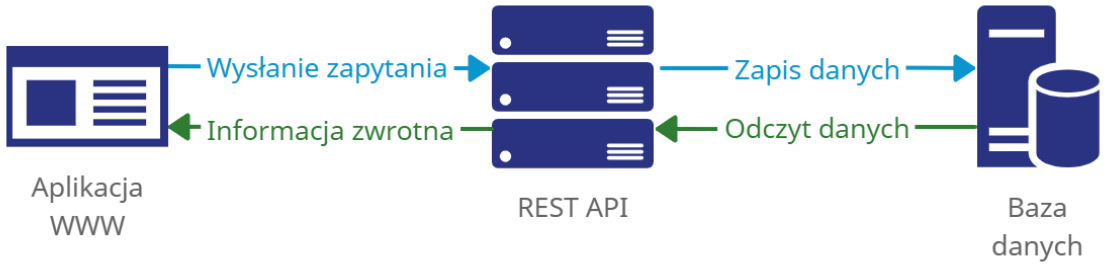
Celem tego projektu jest opracowanie aplikacji zarządzania domową siecią IoT, która umożliwi użytkownikom przegląd i zarządzanie strukturą sieci IoT. Aplikacja będzie oferować intuicyjny interfejs użytkownika, umożliwiający łatwe zarządzanie i przegląd ewentualnych problemów z siecią.

Główne funkcje, które mają zostać zaimplementowane w aplikacji, obejmują:

- Wyświetlanie listy podłączonych urządzeń IoT
- Podgląd ewentualnych problemów z siecią
- Przegląd ostatnich pomiarów czujników
- Modyfikacja struktury sieci
- Dodawanie nowych modeli urządzeń do sieci

3. Implementacja rozwiązania

Na najwyższym poziomie ogólności, proponowane rozwiązanie problemu postawionego w poprzednim punkcie zaprezentować można w następujący sposób:



Powyższy schemat jest uproszczonym planem projektu; który składa się z trzech połączonych komponentów, współpracujących ze sobą. Są to:

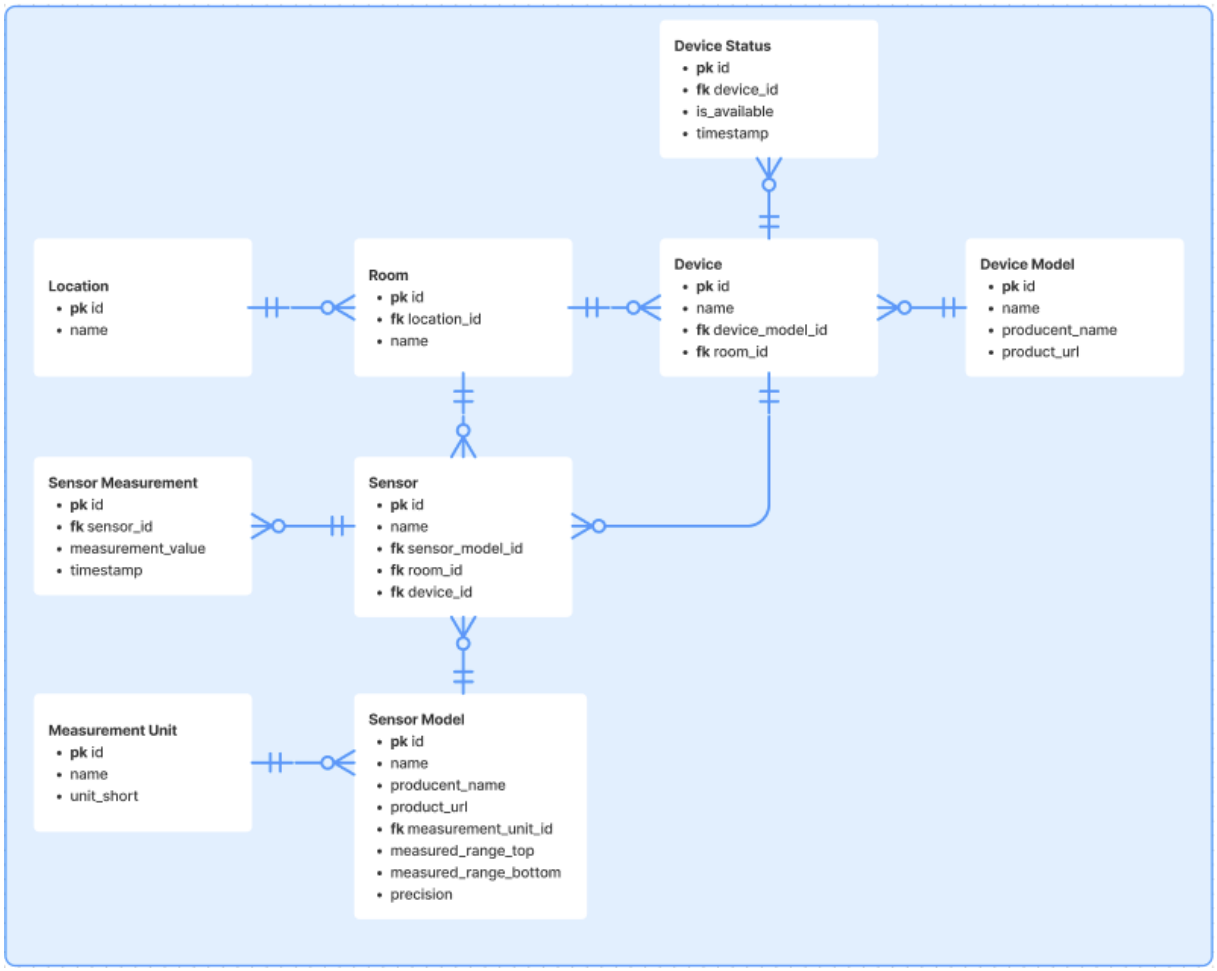
- **Aplikacja WWW** – aplikacja webowa pełniąca funkcję interfejsu użytkownika. W przejrzysty sposób umożliwia osobom korzystającym z portalu dostęp do wszystkich przewidzianych funkcjonalności. Będzie przetwarzać akcje klienta w zapytania kierowane do systemu.

- **REST API** – jest to warstwa pośrednicząca między aplikacją kliencką a bazą danych serwisu. Będzie przetwarzać zapytania aplikacji webowej w odpowiednie modyfikacje bazy, jak również odczytywać dane z bazy i przekazywać je klientowi. Dodatkowym atutem zastosowania REST API jako pośrednika jest eliminacja ryzyka związanego z bezpośrednim umieszczaniem usług w aplikacji webowej.

- **Baza danych** – dokumentowa baza przechowująca niezbędne do funkcjonowania serwisu dane. Obsługiwana i modyfikowana przez usługi REST.

3.1 Baza danych

Baza danych przechowująca dane użytkowników korzystających z serwisu ma zostać zaimplementowana według poniższego diagramu związków encji.



Poniżej zamieszczam opis kolekcji jakie zamierzam zaimplementować w systemie wraz z krótkim wyjaśnieniem do każdego z wykorzystanych pól:

1. Kolekcja "device_statuses":
    a. Pole "_id": identyfikator statusu urządzenia (typ: ObjectId)
    b. Pole "id": identyfikator urządzenia (typ: str)
    c. Pole "metadata": metadane urządzenia (typ: object)
    d. Pole "is_available": flaga informująca o dostępności urządzenia (typ: bool)
    e. Pole "timestamp": znacznik czasu (typ: datetime)
2. Kolekcja "sensor_measurements":
    a. Pole "_id": identyfikator pomiaru czujnika (typ: ObjectId)
    b. Pole "id": identyfikator czujnika (typ: str)
    c. Pole "metadata": metadane czujnika (typ: object)
    d. Pole "measurement_value": wartość pomiaru (typ: float)
    e. Pole "timestamp": znacznik czasu (typ: datetime)
3. Kolekcja "sensor_models":
    a. Pole "_id": identyfikator modelu czujnika (typ: ObjectId)
    b. Pole "id": identyfikator modelu czujnika (typ: str)
    c. Pole "name": nazwa modelu czujnika (typ: str)
    d. Pole "producent_name": nazwa producenta (typ: str)

    e. Pole "product_url": adres URL produktu (typ: str)
    f. Pole "measured_phenomenom": mierzone zjawisko (typ: str)
    g. Pole "measurement_unit_short": jednostka miary (typ: str)
    h. Pole "measured_range_top": górny zakres pomiarowy (typ: float)
    i. Pole "measured_range_bottom": dolny zakres pomiarowy (typ: float)
    j. Pole "precision": precyzja (typ: float)
4. Kolekcja "sensors":
    a. Pole "_id": identyfikator czujnika (typ: ObjectId)
    b. Pole "id": identyfikator czujnika (typ: str)
    c. Pole "name": nazwa czujnika (typ: str)
    d. Pole "device_id": identyfikator urządzenia, do którego jest przypisany czujnik (typ: str lub null)
    e. Pole "room_id": identyfikator pomieszczenia, w którym znajduje się czujnik (typ: str lub null)
    f. Pole "sensor_model_id": identyfikator modelu czujnika (typ: str)
    g. Pole "sensor_model": szczegóły modelu czujnika (typ: dict lub null)
5. Kolekcja "device_models":
    a. Pole "_id": identyfikator modelu urządzenia (typ: ObjectId)
    b. Pole "id": identyfikator modelu urządzenia (typ: str)
    c. Pole "name": nazwa modelu urządzenia (typ: str)
    d. Pole "producent_name": nazwa producenta (typ: str)
    e. Pole "product_url": adres URL produktu (typ: str)
6. Kolekcja "devices":
    a. Pole "_id": identyfikator urządzenia (typ: ObjectId)
    b. Pole "id": identyfikator urządzenia (typ: str)
    c. Pole "name": nazwa urządzenia (typ: str)
    d. Pole "device_model_id": identyfikator modelu urządzenia (typ: str)
    e. Pole "device_model": szczegóły modelu urządzenia (typ: dict lub null)
    f. Pole "room_id": identyfikator pomieszczenia, w którym znajduje się urządzenie (typ: str)
    g. Pole "sensors": lista czujników przypisanych do urządzenia (typ: list[dict] lub null)
7. Kolekcja "rooms":
    a. Pole "_id": identyfikator pomieszczenia (typ: ObjectId)
    b. Pole "id": identyfikator pomieszczenia (typ: str)
    c. Pole "name": nazwa pomieszczenia (typ: str)
    d. Pole "devices": lista urządzeń znajdujących się w pomieszczeniu (typ: list[dict] lub null)
    e. Pole "sensors": lista czujników znajdujących się w pomieszczeniu (typ: list[dict] lub null)
8. Kolekcja "locations":
    a. Pole "_id": identyfikator lokalizacji (typ: ObjectId)

b. Pole "id": identyfikator lokalizacji (typ: str)
c. Pole "name": nazwa lokalizacji (typ: str)
d. Pole "rooms": lista pomieszczeń w lokalizacji (typ: list[dict] lub null)

3.2 Serwer

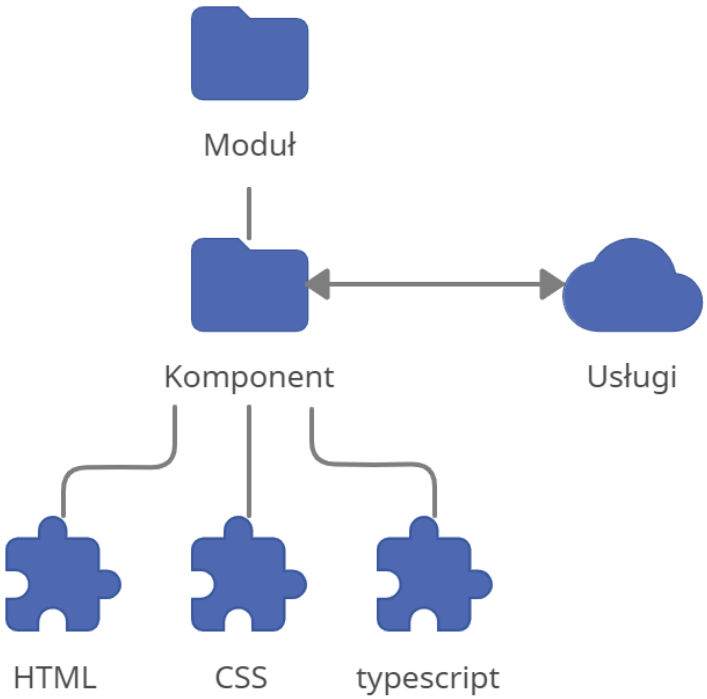
Serwer do komunikacji z bazą danych wykorzystuje bibliotekę PyMongo, zaś do wystawienia RESTowego API wykorzystuje bibliotekę FastAPI. Poniżej zamieszczam opis ogólny dla każdego enpointu.

1. Endpoint / (metoda GET):
    - Zwraca odpowiedź "Hello: World".
    - Służy jako endpoint sprawdzający poprawność działania serwera.
2. Endpoint /locations (metoda GET):
    - Zwraca listę wszystkich lokalizacji w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_locations().
3. Endpoint /devices (metoda GET):
    - Zwraca listę wszystkich urządzeń w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_devices().
4. Endpoint /sensors (metoda GET):
    - Zwraca listę wszystkich czujników w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_sensors().
5. Endpoint /device-set-room (metoda POST):
    - Aktualizuje przynależność urządzenia do pomieszczenia.
    - Otrzymuje argumenty deviceId i newRoomId.
    - Wykorzystuje funkcję mongo_api_setters.update_device_room().
6. Endpoint /sensor-set-assignment (metoda POST):
    - Aktualizuje przynależność czujnika do pomieszczenia i urządzenia.
    - Otrzymuje argumenty sensorId, newRoomId i newDeviceId.
    - Wykorzystuje funkcję mongo_api_setters.update_sensor_assignments().
7. Endpoint /device-statuses (metoda GET):
    - Zwraca listę wszystkich statusów urządzeń w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_device_statuses().
8. Endpoint /sensor-measurements (metoda GET):
    - Zwraca listę wszystkich pomiarów czujników w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_sensor_measurements().
9. Endpoint /location-set-name (metoda POST):
    - Aktualizuje nazwę lokalizacji.
    - Otrzymuje argumenty id i newName.
    - Wykorzystuje funkcję mongo_api_setters.update_location_name().
10. Endpoint /room-set-name (metoda POST):
    - Aktualizuje nazwę pomieszczenia.
    - Otrzymuje argumenty id i newName.
    - Wykorzystuje funkcję mongo_api_setters.update_room_name().
11. Endpoint /device-set-name (metoda POST):
    - Aktualizuje nazwę urządzenia.
    - Otrzymuje argumenty id i newName.
    - Wykorzystuje funkcję mongo_api_setters.update_device_name().
12. Endpoint /sensor-set-name (metoda POST):
    - Aktualizuje nazwę czujnika.
    - Otrzymuje argumenty id i newName.
    - Wykorzystuje funkcję mongo_api_setters.update_sensor_name().
13. Endpoint /location-create (metoda PUT):
    - Tworzy nową lokalizację.
    - Otrzymuje argument name.
    - Wykorzystuje funkcję mongo_api_setters.create_location().

14. Endpoint /room-create (metoda PUT):
    - Tworzy nowe pomieszczenie.
    - Otrzymuje argumenty name i locationId.
    - Wykorzystuje funkcję mongo_api_setters.create_room().
15. Endpoint /device-create (metoda PUT):
    - Tworzy nowe urządzenie.
    - Otrzymuje argumenty name, device_model_id i room_id.
    - Wykorzystuje funkcję mongo_api_setters.create_device().
16. Endpoint /sensor-create (metoda PUT):
    - Tworzy nowy czujnik.
    - Otrzymuje argumenty name, sensor_model_id, room_id i device_id.
    - Wykorzystuje funkcję mongo_api_setters.create_sensor().
17. Endpoint /device-model-create (metoda PUT):
    - Tworzy nowy model urządzenia.
    - Otrzymuje argumenty name, producent_name i product_url.
    - Wykorzystuje funkcję mongo_api_setters.create_device_model().
18. Endpoint /sensor-model-create (metoda PUT):
    - Tworzy nowy model czujnika.
    - Otrzymuje argumenty name, producent_name, product_url, measured_phenomenom, measurement_unit_short, measured_range_top, measured_range_bottom i precision.
    - Wykorzystuje funkcję mongo_api_setters.create_sensor_model().
19. Endpoint /device-models (metoda GET):
    - Zwraca listę wszystkich modeli urządzeń w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_device_models().
20. Endpoint /sensor-models (metoda GET):
    - Zwraca listę wszystkich modeli czujników w bazie danych.
    - Wykorzystuje funkcję mongo_api_getters.get_all_sensor_models().
21. Endpoint /delete-location/{id} (metoda DELETE):
    - Usuwa lokalizację o podanym identyfikatorze.
    - Otrzymuje identyfikator lokalizacji jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_location().
22. Endpoint /delete-room/{id} (metoda DELETE):
    - Usuwa pomieszczenie o podanym identyfikatorze.
    - Otrzymuje identyfikator pomieszczenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_room().
23. Endpoint /delete-device/{id} (metoda DELETE):
    - Usuwa urządzenie o podanym identyfikatorze.
    - Otrzymuje identyfikator urządzenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_device().
24. Endpoint /delete-sensor/{id} (metoda DELETE):
    - Usuwa czujnik o podanym identyfikatorze.
    - Otrzymuje identyfikator czujnika jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_sensor().
25. Endpoint /delete-device-model/{id} (metoda DELETE):
    - Usuwa model urządzenia o podanym identyfikatorze.
    - Otrzymuje identyfikator modelu urządzenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_device_model().
26. Endpoint /delete-sensor-model/{id} (metoda DELETE):
    - Usuwa model czujnika o podanym identyfikatorze.
    - Otrzymuje identyfikator modelu czujnika jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_setters.delete_sensor_model().

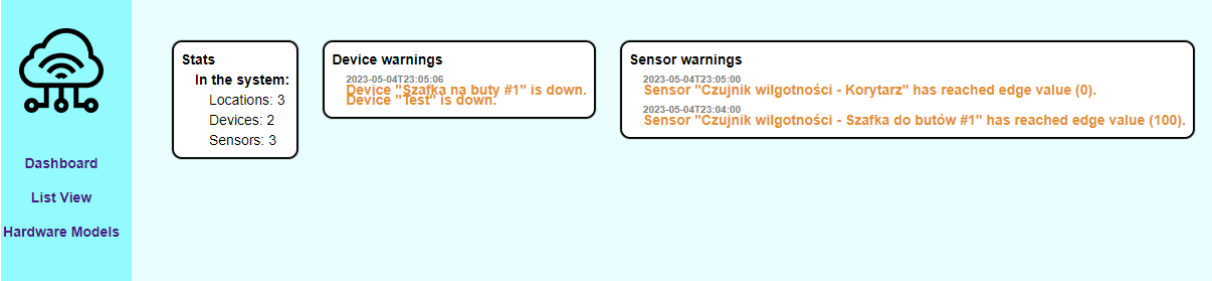
27. Endpoint /change-device-model (metoda POST):
    - Zmienia model urządzenia dla podanego urządzenia.
    - Otrzymuje argumenty id i model_id.
    - Wykorzystuje funkcję mongo_api_setters.change_model_of_device().
28. -sensor-model (metoda POST):
    - Endpoint /change Zmienia model czujnika dla podanego czujnika.
    - Otrzymuje argumenty id i model_id.
    - Wykorzystuje funkcję mongo_api_setters.change_model_of_sensor().
29. Endpoint /edit-device-model (metoda POST):
    - Edytuje model urządzenia o podanym identyfikatorze.
    - Otrzymuje argumenty id, name, producent_name i product_url.
    - Wykorzystuje funkcję mongo_api_setters.edit_device_model().
30. Endpoint /edit-sensor-model (metoda POST):
    - Edytuje model czujnika o podanym identyfikatorze.
    - Otrzymuje argumenty id, name, producent_name, product_url, measured_phenomenom, measurement_unit_short, measured_range_top, measured_range_bottom i precision.
    - Wykorzystuje funkcję mongo_api_setters.edit_sensor_model().
31. Endpoint /room-sensors (metoda GET):
    - Zwraca listę czujników znajdujących się w danym pomieszczeniu.
    - Otrzymuje identyfikator pomieszczenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_getters.get_room_sensors().
32. Endpoint /room-devices (metoda GET):
    - Zwraca listę urządzeń znajdujących się w danym pomieszczeniu.
    - Otrzymuje identyfikator pomieszczenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_getters.get_room_devices().
33. Endpoint /device-sensors (metoda GET):
    - Zwraca listę czujników przypisanych do danego urządzenia.
    - Otrzymuje identyfikator urządzenia jako parametr ścieżki.
    - Wykorzystuje funkcję mongo_api_getters.get_device_sensors().

3.3 Aplikacja webowa



Struktura aplikacji webowej składać będzie się z komponentów umieszczonych w modułach. Każdy komponent będzie składać się z dedykowanego pliku HTML, CSS oraz TypeScript. Ponadto, każdy z komponentów będzie mieć dostęp do usług działających wewnątrz aplikacji. Powyższy diagram opisuje ogólny obraz planowanej struktury. Do jej implementacji wykorzystany został framework Angular w wersji 14.

Widok panelu kontrolnego:

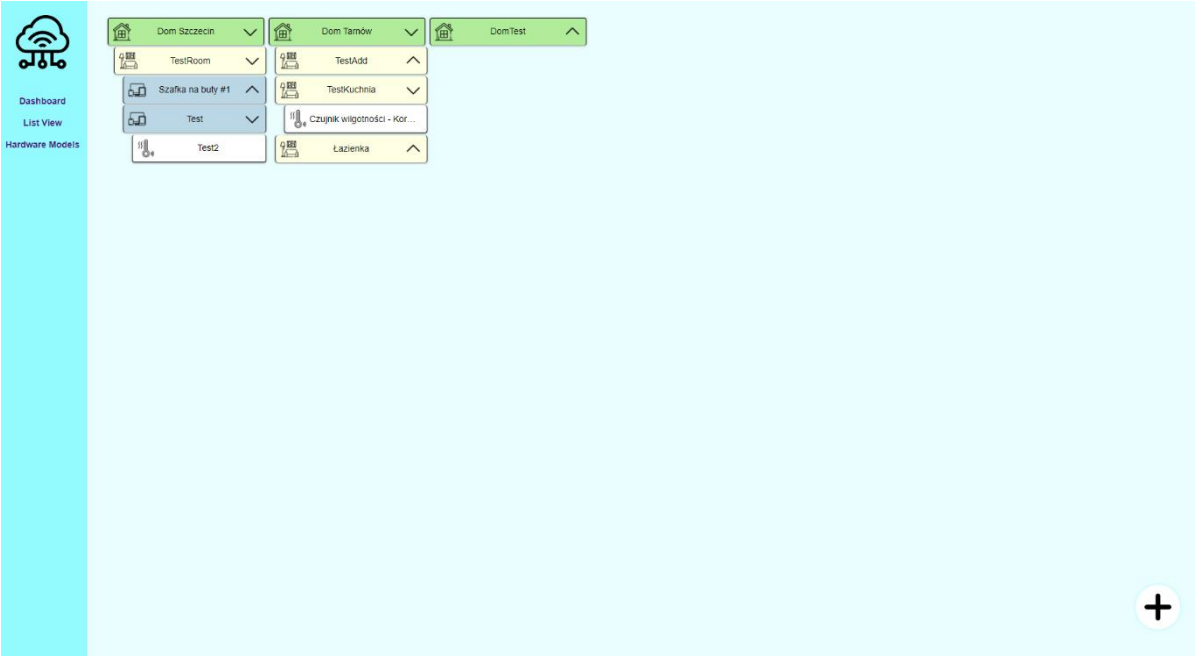


Zadania tego widoku:

1. Monitorowanie stanu systemu: Widok "Stats" wyświetla informacje na temat liczby lokalizacji, urządzeń i czujników w systemie. Może to być przydatne do monitorowania ogólnego stanu i rozmiaru sieci IoT. Użytkownik może na przykład szybko sprawdzić, ile lokalizacji jest w systemie, ile urządzeń jest zarejestrowanych i ile czujników jest aktywnych.
2. Wykrywanie niedostępnych urządzeń: Blok "Device warnings" wykorzystuje dyrektywę *ngFor do iteracji przez tablicę devicesUnavailable i wyświetla ostrzeżenia o niedostępnych urządzeniach. To umożliwia użytkownikowi natychmiastowe zidentyfikowanie urządzeń, które nie działają poprawnie lub są niedostępne w sieci IoT.
3. Wykrywanie czujników osiągających wartość graniczną: Blok "Sensor warnings" wykorzystuje dyrektywę *ngFor do iteracji przez tablicę sensorsWithEdgeValue i wyświetla ostrzeżenia o czujnikach, które osiągnęły wartość graniczną. Może to pomóc użytkownikowi w identyfikacji czujników, które wymagają uwagi lub interwencji, na przykład jeśli czujnik temperatury osiągnął niebezpiecznie wysoką wartość.

Ten widok dostarcza użytkownikowi szybkiego przeglądu stanu systemu IoT oraz informacji o niedostępnych urządzeniach i czujnikach z wartościami granicznymi. Umożliwia to skuteczne monitorowanie i zarządzanie domową siecią IoT.

Widok listy hierarchii:



Krótka lista funkcji tego widoku:

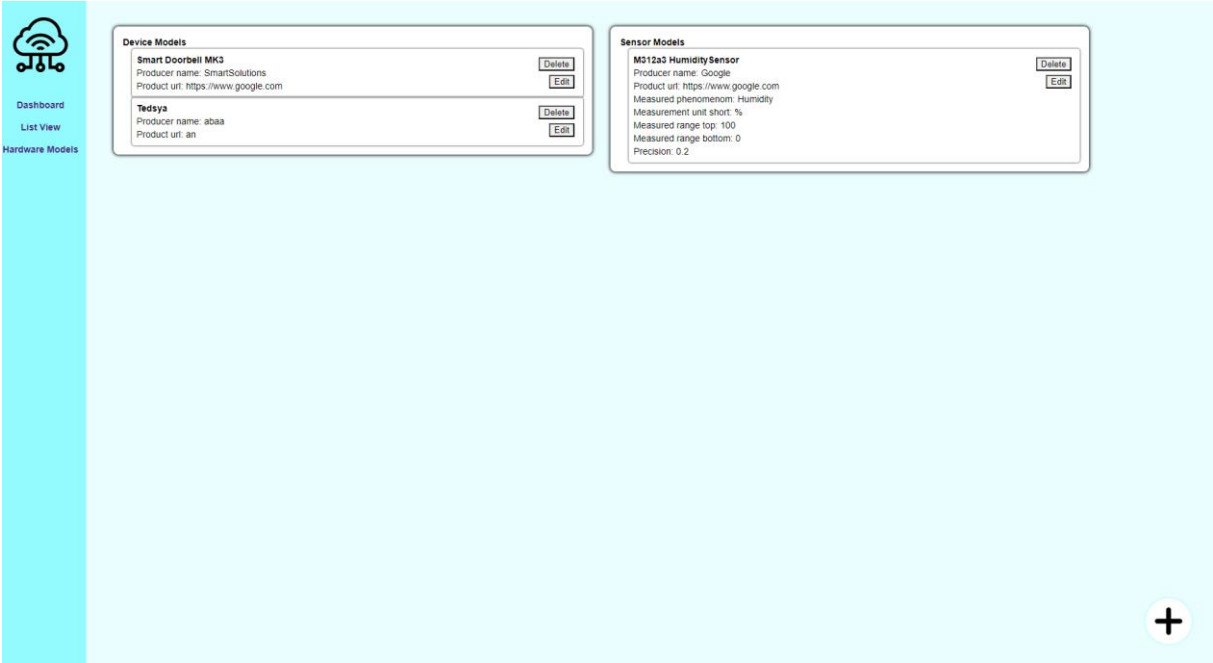
1. Wyświetlanie struktury systemu IoT: Użytkownik może zobaczyć hierarchiczną strukturę systemu IoT, zawierającą lokalizacje, pokoje, urządzenia i czujniki. To umożliwia łatwe zrozumienie organizacji sieci i relacji między elementami.
2. Rozwijanie i zwijanie kontenerów: Użytkownik może rozwinąć lub zwinąć poszczególne kontenery, takie jak lokalizacje, pokoje i urządzenia. To pozwala na skupienie się na konkretnych częściach struktury i ukrycie niepotrzebnych szczegółów.
3. Wyświetlanie szczegółów kontenerów: Po kliknięciu na kontener użytkownik może zobaczyć szczegóły dotyczące nazwy, modelu, statusu i pomiarów związanych z tym kontenerem. To umożliwia szybkie sprawdzenie informacji na temat konkretnego urządzenia lub czujnika.
4. Edycja nazw, modeli i usuwanie kontenerów: Użytkownik ma możliwość edycji nazw kontenerów, takich jak lokalizacje, pokoje, urządzenia i czujniki. Może również zmieniać przypisane modele urządzeń i czujników. Dodatkowo, użytkownik może usuwać niepotrzebne kontenery ze struktury systemu IoT.
5. Dodawanie nowych kontenerów: Użytkownik może dodać nowe kontenery do struktury systemu IoT, takie jak lokalizacje, pokoje, urządzenia i czujniki. Może wybrać typ kontenera, podać jego nazwę oraz wybrać odpowiedni model urządzenia lub czujnika.
6. Przenoszenie kontenerów za pomocą funkcji "drag and drop": Użytkownik ma możliwość przenoszenia urządzeń i czujników między różnymi kontenerami, takimi jak pokoje i urządzenia. Może łatwo zmieniać przynależność i rozmieszczenie elementów w strukturze systemu IoT.
7. Aktualizacja danych o stanie urządzeń i pomiarach czujników: Widok automatycznie odświeża dane o statusie urządzeń oraz pomiarach czujników, zapewniając użytkownikowi aktualne informacje o działaniu i parametrach w systemie IoT.
8. Widoki dialogowe dla szczegółów kontenerów: Użytkownik może interaktywnie edytować nazwy, modele i usuwać kontenery za pomocą dialogów szczegółów. To

zapewnia intuicyjne sposoby modyfikacji i zarządzania elementami w systemie IoT.

9. Możliwość dodawania własnych modeli urządzeń i czujników: Użytkownik może definiować własne modele urządzeń i czujników, które następnie mogą być przypisywane do konkretnych kontenerów. To umożliwia

Widok ten umożliwia użytkownikowi wizualizację i zarządzanie strukturą kontenerów w systemie IoT. Wykorzystuje hierarchiczną listę, która obejmuje lokalizacje, pokoje, urządzenia i czujniki. Każdy kontener jest reprezentowany przez ikonę oraz nazwę, a za pomocą strzałki rozwijającej/zwijającej można wyświetlić lub ukryć szczegóły podkontenerów. Kliknięcie na kontener otwiera jego szczegóły, gdzie użytkownik ma możliwość edycji nazwy oraz innych właściwości. Dialogi detalów są dostępne dla różnych typów kontenerów, takich jak lokalizacje, urządzenia i czujniki, umożliwiając użytkownikowi dostosowanie ich parametrów. Dodatkowo, istnieje przycisk dodawania, który umożliwia użytkownikowi dodawanie nowych kontenerów do struktury systemu IoT. Po kliknięciu przycisku otwiera się dialog, w którym można wybrać odpowiedni typ kontenera i podać jego szczegóły. Ten interfejs użytkownika zapewnia prosty sposób zarządzania strukturą kontenerów, wyświetlanie i edycję ich właściwości, a także dodawanie nowych kontenerów do systemu IoT. Ułatwia to użytkownikowi kontrolę nad organizacją i konfiguracją swojej domowej sieci IoT.

Widok panelu modeli:



Ten widok służy do zarządzania modelami urządzeń i modelami czujników w systemie IoT. Obejmuje listę modeli urządzeń i modeli czujników, które można edytować, usuwać i dodawać nowe modele. Oto lista funkcji tego widoku:

1. Wyświetlanie listy modeli urządzeń: Widok wyświetla sekcję "Device Models", która zawiera listę modeli urządzeń. Dla każdego modelu urządzenia wyświetlane są informacje, takie jak nazwa producenta i URL produktu. Użytkownik może zobaczyć wszystkie dostępne modele urządzeń w systemie.

2. Edycja modeli urządzeń: Dla każdego modelu urządzenia istnieje przycisk "Edit", który po kliknięciu otwiera panel edycji. Użytkownik może edytować nazwę, nazwę producenta i URL produktu modelu urządzenia. Po dokonaniu zmian można zapisać zmienione dane, klikając przycisk "Save".

3. Usuwanie modeli urządzeń: Dla każdego modelu urządzenia istnieje przycisk "Delete", który po kliknięciu usuwa model urządzenia z systemu. Użytkownik może usunąć niepotrzebne modele urządzeń, które nie są już używane w systemie.

4. Wyświetlanie listy modeli czujników: Widok wyświetla sekcję "Sensor Models", która zawiera listę modeli czujników. Dla każdego modelu czujnika wyświetlane są informacje, takie jak nazwa producenta, URL produktu, mierzone zjawisko, jednostka pomiarowa, zakres pomiarowy i precyzja. Użytkownik może zobaczyć wszystkie dostępne modele czujników w systemie.

5. Edycja modeli czujników: Dla każdego modelu czujnika istnieje przycisk "Edit", który po kliknięciu otwiera panel edycji. Użytkownik może edytować nazwę, nazwę producenta, URL produktu, mierzone zjawisko, jednostkę pomiarową, zakres pomiarowy i precyzję modelu czujnika. Po dokonaniu zmian można zapisać zmienione dane, klikając przycisk "Save".

6. Usuwanie modeli czujników: Dla każdego modelu czujnika istnieje przycisk "Delete", który po kliknięciu usuwa model czujnika z systemu. Użytkownik może usunąć niepotrzebne modele czujników, które nie są już używane w systemie.

7. Dodawanie nowych modeli: Istnieje przycisk "Add", który po kliknięciu otwiera dialog dodawania nowych modeli urządzeń i czujników. Użytkownik może wprowadzić nazwę, nazwę producenta, URL produktu, mierzone zjawisko, jednostkę pomiarową, zakres pomiarowy i precyzję dla nowego modelu. Po dodaniu nowego modelu, zostanie wyświetlony na liście dostępnych modeli.

8. Obsługa dialogów: Widok wykorzystuje dialogi, które umożliwiają interaktywną edycję i dodawanie modeli urządzeń i czujników. Pojawiają się one po kliknięciu przycisków "Edit" i "Add". Dialogi zawierają formularze do wprowadzania danych i przyciski do zapisywania zmian i anulowania operacji.

9. Powiązanie danych: Widok korzysta z dwustronnego powiązania danych za pomocą dyrektywy [(ngModel)], dzięki czemu wprowadzone przez użytkownika zmiany są automatycznie odzwierciedlane w modelach danych. To zapewnia płynną interakcję i aktualizację danych w czasie rzeczywistym.

10. Widok zawiera również przycisk "Add" na górze, który otwiera dialog dodawania nowych modeli urządzeń i czujników.

4. Podsumowanie

W ramach tego projektu została opracowana aplikacja do zarządzania domową siecią IoT. Aplikacja umożliwia użytkownikom przegląd i zarządzanie strukturą sieci IoT, dostarczając intuicyjny interfejs użytkownika. Celem projektu było stworzenie aplikacji, która będzie łatwa w obsłudze, intuicyjna i spełniająca oczekiwania użytkowników dotyczące zarządzania domową siecią IoT. Główne funkcje, które zostały zaimplementowane w aplikacji, to:

1. Wyświetlanie listy podłączonych urządzeń IoT: Aplikacja umożliwia użytkownikom przegląd wszystkich podłączonych urządzeń IoT w sieci. Użytkownicy mogą zobaczyć informacje o urządzeniach, takie jak nazwa, model i status.
2. Podgląd ewentualnych problemów z siecią: Aplikacja monitoruje stan urządzeń w sieci IoT i informuje użytkowników o ewentualnych problemach, takich jak niedostępność urządzeń.
3. Przegląd ostatnich pomiarów czujników: Aplikacja umożliwia użytkownikom sprawdzenie ostatnich pomiarów dokonywanych przez czujniki w sieci IoT. Użytkownicy mogą zobaczyć wartości pomiarów i monitorować zmiany w czasie.
4. Modyfikacja struktury sieci: Aplikacja umożliwia użytkownikom zarządzanie strukturą sieci IoT, taką jak lokalizacje, pokoje, urządzenia i czujniki. Użytkownicy mogą dodawać, edytować i usuwać kontenery oraz przenosić urządzenia i czujniki między nimi.
5. Dodawanie nowych modeli urządzeń do sieci: Aplikacja umożliwia użytkownikom dodawanie nowych modeli urządzeń do sieci IoT. Użytkownicy mogą określić nazwę, producenta i adres URL produktu dla nowego modelu.

Projekt składa się z trzech głównych komponentów: aplikacji webowej, serwera i bazy danych. Aplikacja webowa została zaimplementowana przy użyciu frameworku Angular w wersji 14. Serwer wykorzystuje bibliotekę PyMongo do komunikacji z bazą danych, a REST API zostało wystawione za pomocą biblioteki FastAPI.

Baza danych jest dokumentową bazą danych, która przechowuje informacje o lokalizacjach, pokojach, urządzeniach, czujnikach i modelach urządzeń i czujników. Dla każdej kolekcji zostały określone pola, takie jak identyfikatory, nazwy, metadane i inne informacje związane z danym obiektem. Aplikacja webowa zaś oferuje użytkownikom trzy główne widoki: panel kontrolny, listę hierarchii i panel modeli. Panel kontrolny umożliwia użytkownikom szybki podgląd stanu sieci IoT, wyświetlając informacje takie jak liczba podłączonych urządzeń, status sieci, ostatnie pomiary czujników itp. Użytkownicy mogą również zobaczyć powiadomienia o ewentualnych problemach w sieci.Lista hierarchii prezentuje strukturę sieci IoT w formie drzewa, gdzie użytkownicy mogą zobaczyć lokalizacje, pokoje, urządzenia i czujniki. Kliknięcie na poszczególne elementy umożliwia użytkownikom przeglądanie szczegółowych informacji oraz dokonywanie modyfikacji.Panel modeli pozwala użytkownikom zarządzać dostępnymi modelami urządzeń i czujników w sieci IoT. Użytkownicy mogą przeglądać, dodawać, edytować i usuwać modele, określając ich nazwę,

producenta i adres URL. Dodanie nowego modelu umożliwia późniejsze przypisanie go do konkretnych urządzeń i czujników. Wszystkie zmiany dokonywane w aplikacji są automatycznie synchronizowane z serwerem i bazą danych, co umożliwia współdzielenie danych między różnymi użytkownikami i urządzeniami. Założenia projektowe obejmowały obsługę różnych modeli urządzeń i czujników, możliwość rozszerzania sieci o nowe modele, monitorowanie stanu urządzeń, wyświetlanie pomiarów czujników oraz intuicyjne zarządzanie strukturą sieci.

Opracowana aplikacja stanowi kompleksowe rozwiązanie do zarządzania domową siecią IoT, które może znacznie ułatwić użytkownikom korzystanie z technologii IoT i zapewnić im pełną kontrolę nad swoimi urządzeniami i czujnikami.