



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**  
**Wydział Fizyki i Informatyki Stosowanej**

KATEDRA INFORMATYKI STOSOWANEJ I FIZYKI KOMPUTEROWEJ

## Praca dyplomowa

*System do akwizycji danych z rozproszonych systemów  
pomiarowych*  
*System for data acquisition from distributed measurement  
systems*

Autor:  
Kierunek studiów:  
Opiekun pracy:

Mateusz Barnacki  
Informatyka Stosowana  
dr inż. Antoni Dydejczyk

Kraków, 2023r.

## Spis treści

1. Wstęp
2. Przegląd wzorców architektury aplikacji
  - 2.1. Architektura monolityczna
  - 2.2. Architektura mikrousługowa
3. Architektura aplikacji
  - 3.1. Wymagania biznesowe
  - 3.2. Architektura systemu
  - 3.3. Relacyjna baza danych
  - 3.4. Dokumentowa baza danych
4. Obsługa systemu
  - 4.1. Struktura projektu
  - 4.2. Budowanie projektu
  - 4.3. Wdrożenie aplikacji na serwer wydziału
  - 4.4. Pierwsze uruchomienie systemu
  - 4.5. Wdrażanie zmian w systemie
  - 4.6. Monitorowanie działania systemu
  - 4.7. Połączenie z bazą MongoDB
5. Dokumentacja komponentów
  - 5.1. Service Registry
  - 5.2. API Gateway
  - 5.3. Serwis autoryzacji
  - 5.4. Serwis czujników
  - 5.5. Klient aplikacji
6. Podsumowanie

## **1. Wstęp**

## **2. Przegląd wzorców architektury aplikacji**

### **2.1. Architektura monolityczna**

### **2.2. Architektura mikrousługowa**

### 3. Architektura aplikacji

#### 3.1. Wymagania biznesowe

Kluczowym etapem cyklu życia oprogramowania jest faza zebrania wymagań biznesowych. Wymagania biznesowe pozwalają na płynne przejście z fazy koncepcji do fazy projektowania systemu. Zebranie wymagań odbyło się w trakcie spotkania organizacyjnego w pierwszej połowie października. W oparciu o zasady metodyki zwinnej dokonano konwersji wymagań biznesowych na historyjki użytkownika. Historyjki użytkownika są opisem funkcjonalności systemu widzianych z perspektywy użytkownika końcowego. Głównym celem jest podkreślenie, jaka wartość zostanie dostarczona użytkownikowi za pośrednictwem danej funkcjonalności [1]. Poniżej umieszczono stworzone historyjki.

*Jako użytkownik chciałbym, żeby...*

- *osoba z określoną rolą mogła stworzyć projekt badawczy za pomocą interfejsu użytkownika, aby móc rozwijać zasięg prac naukowych.*
- *osoba z określoną rolą mogła stworzyć nowy projekt badawczy na podstawie istniejącego projektu, aby móc szybciej rozpocząć nowy projekt badawczy.*
- *osoba z określoną rolą mogła czytać informacje na temat projektu badawczego, aby móc poszerzać wiedzę nowych członków zespołów badawczych.*
- *osoba z określoną rolą mogła eksportować projekty badawcze, aby móc pracować w warunkach braku dostępu do Internetu.*
- *można było zapisywać dane przesyłane przez czujniki, aby móc gromadzić dane badawcze.*
- *można było weryfikować pomiary za pomocą API Key, aby mieć pewność, że dane nie są sfałszowane.*
- *następowała walidacja poprawności otrzymywanych danych pomiarowych, aby mieć gwarancję składowania rzetelnych danych.*
- *można było agregować dane, aby mieć szersze spektrum gromadzonych informacji.*
- *można było odczytywać określoną ilość danych, aby móc zweryfikować poprawność działania systemu gromadzenia danych.*

Na podstawie zebranych informacji można zauważyć potencjalne problemy, które należy rozwiązać w trakcie projektowania systemu. Wydział Fizyki i Informatyki Stosowanej posiada różne rodzaje czujników pomiarowych. Wobec tego interfejs aplikacji odpowiedzialny za rejestrowanie pomiarów przesyłanych przez urządzenia musi umożliwić zapis danych sformatowanych w różny sposób. Ponadto dane odebrane przez serwis muszą zostać zapisane w bazie danych, która obsługuje możliwość zapisu danych nie posiadających określonego schematu. Kolejnym aspektem, który należy uwzględnić w procesie projektowania systemu, jest przeprowadzenie walidacji danych. Należy zaprojektować system w taki sposób, aby narzut czasowy spowodowany koniecznością zdobycia informacji na temat parametrów podlegających weryfikacji był jak najmniejszy. Ostatnim zagadnieniem jest zapewnienie zdolności dostosowania systemu do odbierania dużej liczby pomiarów w przyszłości.

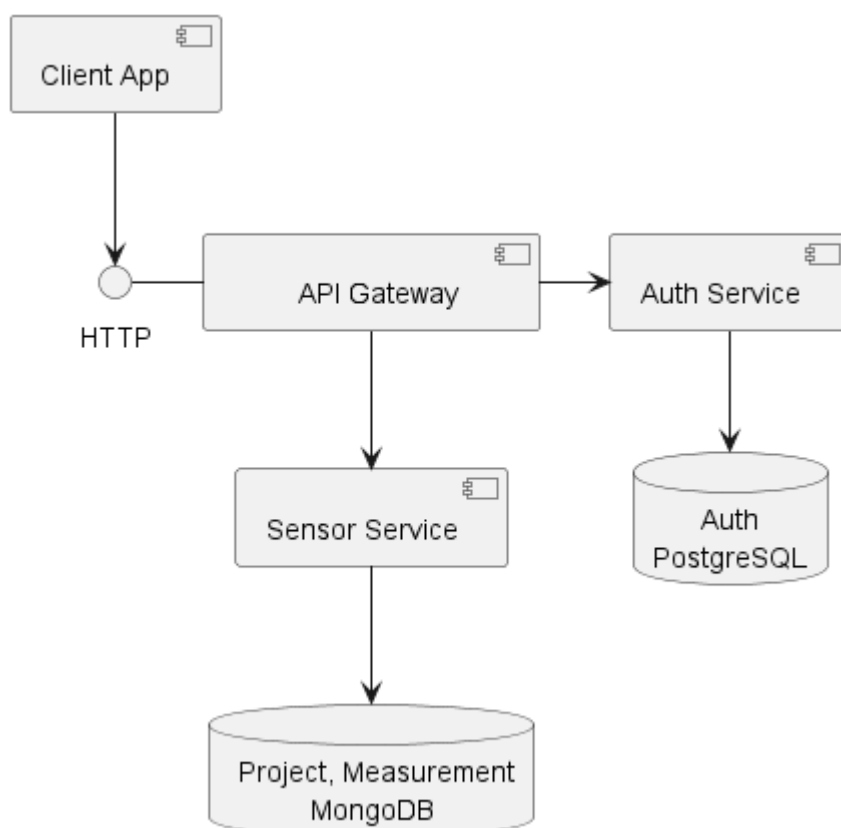
Dokładna analiza powyższych zagadnień umożliwi zaprojektowanie architektury rozwiązania, dobór odpowiednich technologii oraz dostosowanie metodyki pracy.

### 3.2. Architektura systemu

Faza projektowania systemu rozpoczęła się od przeglądu zebranych wymagań funkcjonalnych oraz utworzonych na ich bazie historyjek użytkownika. W celu dostosowania architektury systemu do wskazanych potrzeb dokonano ewaluacji istniejących rozwiązań. Na podstawie analizy dostępnych materiałów oraz konsultacji z opiekunem pracy dyplomowej została podjęta decyzja o implementacji w systemie architektury mikrousługowej.

Architektura mikrousługowa zapewnia elastyczność w doborze technologii i baz danych dla poszczególnych komponentów aplikacji. Wskazana cecha jest kluczowa w kontekście dopasowania odpowiedniej technologii do wymagań funkcjonalnych. Ponadto umożliwia łatwiejsze rozbudowywanie systemu w przyszłości, ponieważ zezwala programiście na wykorzystanie preferowanych narzędzi. Ważnym aspektem jest również możliwość wydzielenia funkcjonalności, których nie można szybko rozwijać w technologii preferowanej przez dewelopera oprogramowania.

Cechą charakterystyczną mikrousług jest zapewnienie łatwej skalowalności aplikacji. Każdy serwis składający się na całość systemu jest traktowany jako niezależny komponent. Wobec tego w sytuacji dużego obciążenia sieciowego dla funkcjonalności dodania pomiaru do systemu można stworzyć drugą instancję aplikacji i rozłożyć ruch sieciowy równomiernie na wszystkie dostępne instancje. W wyniku tego działania serwis odpowiedzialny za funkcjonalności autoryzacji użytkownika nie poniesie żadnych konsekwencji, ponieważ nie ma nic wspólnego z funkcjonalnością dodania pomiaru. Za dostosowanie działania aplikacji do rejestrowanych warunków obciążenia sieciowego odpowiadają osoby specjalizujące się w zagadnieniach DevOps.



**Zdjęcie 1** Diagram komponentów systemu do akwizycji danych

Jednym z wyzwań związanych z zastosowaniem architektury mikrousługowej jest zagadnienie podziału systemu będącego monolitem na mniejsze komponenty. Błędny podział aplikacji może doprowadzić do powstania silnych zależności pomiędzy poszczególnymi komponentami. Współzależność komponentów może się objawić jako konieczność stałego przesyłania żądań HTTP pomiędzy dwoma serwisami lub procesowanie tych samych danych przez różne serwisy. Obecnie trwają prace nad odkryciem uniwersalnego sposobu pozwalającego na prawidłowe wydzielanie komponentów aplikacji. Jedną z koncepcji jest podejście o nazwie *Domain-Driven Design*. Jego głównym celem jest zaprogramowanie domeny, która będzie respektowała wewnętrzne procesy i zasady [2]. W praktyce oznacza to taki sposób definiowania komponentów, aby wiernie odzwierciedlały rzeczywistość. Proces odkrywania domeny może być realizowany za pomocą podejścia o nazwie *Event Storming*.

Wydzielenie komponentów w ramach systemu do akwizycji danych nastąpiło w sposób arbitralny. Pierwszym powodem zastosowania wskazanego podejścia był brak możliwości skonfrontowania własnych przemyśleń z innymi członkami zespołu, ponieważ projekt był realizowany samodzielnie. Drugim argumentem przemawiającym za zastosowaniem podziału arbitralnego było zaoszczędzenie czasu koniecznego na przeprowadzanie sesji *Event Storming*.

Diagram komponentów aplikacji został zaprezentowany na zdjęciu 2. Dokładny opis funkcjonalności oferowanych przez poszczególne komponenty aplikacji został umieszczony w rozdziale 5. Każdy element zaprezentowany na diagramie może być rozwijany i uruchamiany niezależnie od pozostałych komponentów. Odpowiednikiem elementu na diagramie w działającym systemie jest kontener Docker. Wzorzec jednego serwisu przypadającego na jeden kontener pozwala odseparować informacje na temat potrzebnych technologii i zależności. Ponadto Docker umożliwia ograniczenie zasobów obliczeniowych konsumowanych przez kontener. Narzędzia przeznaczone do orkiestracji kontenerów pozwalają w łatwy sposób zarządzać dostępną ilością instancji serwisów.

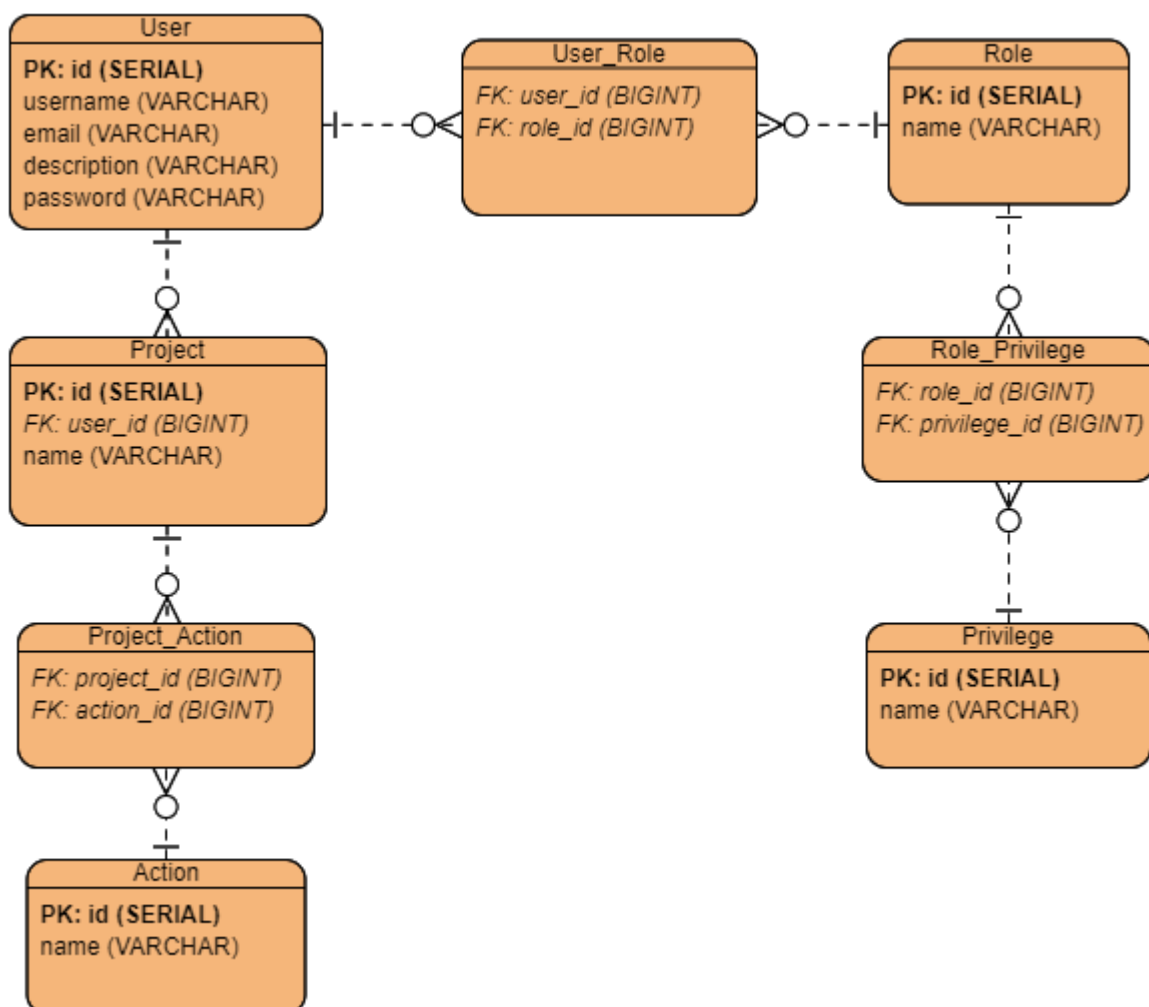
Głównym powodem rozdzielenia serwisu czujników i serwisu autoryzacji jest zastosowanie różnych technologii w obydwu serwisach. Serwis czujników został napisany za pomocą języka JavaScript. Wybór technologii został wymuszony koniecznością spełnienia wymagania dotyczącego zapisu pomiarów przesyłanych przez różne typy czujników w ramach jednego kontrolera. Kontrolery zaprojektowane w języku Java wymagają przesyłania danych o odgórnie określonym formacie. Dane nie posiadające jednoznacznego formatu byłyby bardzo ciężkie do obsługi po stronie serwera aplikacji. Język JavaScript jest językiem skryptowym, wobec czego nie wymusza operacji na określonych typach danych. Ta cecha języka pozwala na obsługę danych bez względu na ich strukturę. Serwis autoryzacji został zaimplementowany w języku Java. Jest to argumentowane wcześniejszym wykorzystaniem przez autora niniejszej pracy technologii Java przy implementacji zagadnień związanych z zabezpieczaniem aplikacji. Rozdzielenie wskazanych serwisów umożliwiło wykorzystanie różnych typów baz danych dla poszczególnych komponentów.

Utworzenie komponentu API Gateway jest spowodowane wybraniem architektury mikrousługowej. API Gateway odgrywa kluczową rolę w komunikacji pomiędzy klientami i serwerem aplikacji. Komponent można określić jako bramkę oddzielającą sieć wewnętrzną służącą do komunikacji między serwisami od sieci zewnętrznej. Bramka pozwala ukryć wewnętrzną strukturę systemu przed światem zewnętrznym. Wszystkie funkcjonalności realizowane przez bramkę zostały opisane w rozdziale 5.2.

Aplikacja klienta przedstawiona na diagramie jest aplikacją internetową dostępną z poziomu przeglądarki internetowej. Komunikuje się z API Gateway za pośrednictwem interfejsu HTTP i pozwala zrealizować większość operacji udostępnianych przez serwisy tworzące system do akwizycji danych.

### 3.3. Relacyjna baza danych

W trakcie procesu projektowania architektury systemu przeanalizowano wiele możliwości składowania danych w serwisie autoryzacji. Jednym z głównych argumentów przemawiających za wykorzystaniem relacyjnej bazy danych było dostrzeżenie relacji pomiędzy użytkownikami, rolami i uprawnieniami. Podobne rozwiązanie mogłoby zostać zaprojektowane przy użyciu dokumentowej bazy danych. W dokumentowej bazie danych można zastąpić relacje przez mechanizm zagnieżdżania dokumentów. Konsekwencją zastosowania tego rozwiązania byłaby redundancja danych opisujących role i uprawnienia. Relacyjna baza danych dzięki zastosowaniu tabel asocjacyjnych pozwala tworzyć wiele powiązań pomiędzy rekordami występującymi w różnych tabelach. Wobec tego nie jest koniecznym utworzenie dwóch niezależnych rekordów opisujących tę samą rolę. Drugim kluczowym aspektem argumentującym wybór bazy relacyjnej jest dostępność mechanizmu transakcyjności. Transakcyjność powinna spełniać warunki ACID. ACID to skrót od angielskich słów atomicity (atomowość), consistency (spójność), isolation (izolacja), durability (trwałość). Wymienione cechy definiują działanie mechanizmu transakcyjności. W kontekście systemu do akwizycji danych transakcje są wykorzystywane podczas tworzenia kont użytkowników wraz z przypisanymi do użytkowników projektami badawczymi.



Zdjęcie 2 Diagram ERD bazy auth



Kolejnym krokiem w procesie projektowania rozwiązania był wybór systemu zarządzania relacyjnymi bazami danych. W wyniku analizy dostępnych narzędzi oraz konsultacji z opiekunem pracy podjęto decyzję o zastosowaniu bazy danych PostgreSQL. Wskazany system zarządzania relacyjnymi bazami danych jest narzędziem darmowym oraz posiada dobrze utrzymaną dokumentację. Dodatkowym atutem jest wykorzystanie wybranej bazy danych w ramach zajęć akademickich, wobec czego obsługa systemu w przyszłości nie będzie stanowiła problemu dla żadnej osoby związanej z wydziałem.

Na zdjęciu 2. zaprezentowano diagram ERD bazy gromadzącej dane dla systemu autoryzacji. Najważniejszą encją na zaprezentowanym diagramie ERD jest tabela User, ponieważ zawiera wszystkie kluczowe informacje na temat użytkowników aplikacji. Hasło składowane w bazie danych jest haszowane w trakcie procesowania przez serwer aplikacji. Pozostałe dane nie zostały uznane za dane wrażliwe i mogą zostać odczytane przez administratora bazy danych. Tabela Role służy do przechowywania ról użytkowników. Każda z ról posiada przypisane uprawnienia. Uprawnienia są składowane w tabeli Privilege. Tabela Project przechowuje nazwy projektów badawczych przypisanych do danego użytkownika. Każdy z użytkowników może być zaangażowany w wiele projektów, wobec tego między encjami User i Project występuje relacja jeden do wielu. Dzięki zastosowaniu tego mechanizmu każdy projekt jest traktowany jako niezależny rekord w tabeli i umożliwia przypisanie akcji. Tabela Action służy do przechowywania nazw akcji, których widoczność jest zależna od projektu badawczego. Informacje na temat widoczności akcji w zależności od projektu są gromadzone w tabeli Project\_Action. Każdy użytkownik może mieć przypisane określone role zdefiniowane w systemie. Przypisywanie ról określonym użytkownikom odbywa się w tabeli User\_Role. Relacje pomiędzy rolami i uprawnieniami są gromadzone w tabeli Role\_Privilege.

Podjęto decyzję, że dostęp do danych z poziomu serwera aplikacji odbywa się za pośrednictwem frameworka Hibernate. Hibernate oferuje możliwość odwzorowania encji z bazy danych na obiekty po stronie serwera aplikacji. Wadą wykorzystania tego narzędzia jest dodanie kolejnej warstwy w komunikacji pomiędzy aplikacją a bazą danych, czego skutkiem jest dłuższy czas wykonywania zapytań bazodanowych. Konfiguracja frameworka znajduje się w pliku *application.yml*. Jedną z oferowanych opcji jest możliwość utworzenia schematu bazy danych na podstawie klas napisanych w języku Java. W ramach pracy zdecydowano się zastąpić ten mechanizm poprzez napisanie skryptów w języku SQL, które tworzą bazę danych, schemat, tabele oraz sekwencje. Ponadto dla tabel Role, Privilege i Action utworzono zapytania dodające dane predefiniowane po stronie serwera aplikacji. Skrypty zostały umieszczone w folderze *auth-service/src/main/resources/db-scripts*. W celu przyspieszenia procesu tworzenia połączono wszystkie instrukcje do jednego skryptu o nazwie *init\_db.sql*. Utworzenie schematu bazy danych musi nastąpić po pierwszym uruchomieniu aplikacji na środowisku produkcyjnym. Instrukcja inicjalizacji bazy danych znajduje się w rozdziale 5.4.

Operacje na danych po stronie serwera aplikacji są wykonywane za pomocą frameworka Spring Data JPA. Dostęp do operacji bazodanowych odbywa się za pośrednictwem interfejsów nazywanych repozytoriami. Dodatkową funkcjonalnością oferowaną przez framework jest tworzenie zapytań za pomocą specjalnie dostosowanych nazw metod. Dzięki temu proces rozwijania aplikacji jest znacznie szybszy, ponieważ programista nie musi samodzielnie pisać zapytań bazodanowych. Każda operacja wykonana po stronie serwera jest traktowana jako pojedyncza transakcja. W przypadku wystąpienia błędu użytkownik zostaje poinformowany za pomocą *unchecked exception*. Ten rodzaj wyjątku nie zatrzymuje działania serwera aplikacji i pozwala przeanalizować stos wywołania metody, który pełni rolę logu działania aplikacji w danej chwili czasowej.

### 3.4. Dokumentowa baza danych

Jednym z problemów podczas projektowania architektury systemu było znalezienie optymalnego rozwiązania dla zagadnienia składowania danych przesyłanych przez czujniki pomiarowe oraz danych reprezentujące projekty badawcze. Wymagania zebrane na wczesnym etapie projektowania aplikacji sugerowały dużą różnorodność gromadzonych danych. Ponadto należało uwzględnić niezależność każdego projektu badawczego. Aplikacja stworzona na zasadach architektury monolitycznej wymagałaby użycia bazy relacyjnej, ponieważ w ramach systemu można użyć tylko jednego rodzaju bazy danych. Składowanie danych w bazie relacyjnej doprowadziłoby do zarządzania dużą ilością tabel niepowiązanych między sobą relacjami. Ponadto proces dodania danych nie wymaga wykorzystania mechanizmu transakcji, ponieważ użytkownikowi systemu nie zależy na natychmiastowym dostępie do nowych rekordów. Duża ilość danych dostarczonych za pośrednictwem czujników może wymusić skalowanie bazy danych, co w przypadku baz relacyjnych jest trudnym procesem.

W wyniku konsultacji przeprowadzonych z opiekunem pracy dyplomowej została podjęta decyzja o przeanalizowaniu rozwiązań oferowanych przez bazy NoSQL. Celem wynalezienia baz NoSQL było ułatwienie procesu horyzontalnego skalowania bazy. Horyzontalne skalowanie bazy danych polega na dodaniu kolejnego serwera, a następnie partycjonowaniu obecnego zbioru danych oraz obciążenia pomiędzy wszystkie dostępne instancje. Proces analizy dostępnych rozwiązań został zakończony wyborem dokumentowej bazy danych MongoDB. W dokumentowych bazach danych obiekty JSON są traktowane jako dokumenty. JSON gromadzony w bazie danych nie musi posiadać określonego schematu budowy. Wobec tego narzut czasowy spowodowany koniecznością wykorzystania frameworka odpowiedzialnego za proces mapowania obiektowo relacyjnego jest zniwelowany.

MongoDB jest jednym z najpopularniejszych narzędzi przeznaczonych do tworzenia i zarządzania dokumentowymi bazami danych. Baza danych jest kompatybilna z szeroką gamą języków programowania. Jednym z wariantów dostępu do baz danych jest aplikacja *MongoDB Compass*, która pozwala monitorować stan bazy za pośrednictwem graficznego interfejsu użytkownika. Alternatywną opcją dostępu do danych jest wykorzystanie narzędzia *MongoDB Shell*, które pozwala wykonywać operacje bazodanowe przy pomocy wiersza poleceń. Największym atutem bazy MongoDB jest możliwość opcjonalnego włączenia walidacji schematu dodawanych danych. Wykorzystanie wskazanej opcji pozwala weryfikować poprawność struktury dokumentu przy próbie dodania nowego zasobu do bazy danych.

Baza MongoDB jest wykorzystana do gromadzenia danych przetwarzanych przez serwis czujników. Obiekty składowane w bazie danych można podzielić na dwie grupy. Pierwszą grupę stanowią dokumenty reprezentujące projekty badawcze. Projekty badawcze są tworzone na podstawie formularzy dostępnych po stronie klienta aplikacji. Niektóre z pól występujących w obiekcie JSON są wymagane ze względu na proces dalszego przetwarzania danych. Ponadto w części pól spodziewane jest wystąpienie określonej wartości enumerowanej. Wszystkie dokumenty reprezentujące projekty badawcze są gromadzone w jednej kolekcji danych. Każda kolekcja może posiadać własny zestaw reguł walidacyjnych. Weryfikacja schematu dokumentu odbywa się przy użyciu mechanizmu o nazwie JSON Schema. Zaletami wykorzystania mechanizmu JSON Schema są zdefiniowanie istniejącego formatu danych oraz zapewnienie przetwarzania odpowiedniej jakości danych obsługiwanych przez aplikację [3]. Przykładowa struktura dokumentu reprezentującego projekt badawczy została zaprezentowana na zdjęciu 3.

```

{
  "_id": "64dc9bc82e32b6ade9ace014",
  "name": "Temperature Test Project",
  "acronym": "temp-proj",
  "description": "This project tests whether application can save data from sensor instances or not.",
  "timeMode": "PERMANENTLY",
  "spatialMode": "STATIONARY",
  "measurementMode": "SINGLE",
  "sensors": [
    {
      "deviceId": "temperature-sens-1",
      "latitude": 31.21,
      "longitude": 12.345,
      "altitude": 573,
      "measurementSchema": {
        "measurements": [
          {
            "name": "T",
            "description": "Temperature",
            "unit": "Degree",
            "range": {
              "min": -50,
              "max": 100
            },
            "validate": true,
            "errorValue": 999,
            "aggregate": true,
            "aggregationInterval": "DAY",
            "maxBreak": 300
          }
        ]
      }
    }
  ]
}

```

**Zdjęcie 3** Obiekt JSON reprezentujący strukturę projektu badawczego

Drugą grupę danych stanowią pomiary. Każdy rodzaj pomiaru charakteryzuje się własnym formatem danych. MongoDB umożliwia składowanie dowolnie sformatowanych dokumentów w ramach jednej kolekcji danych. Ponadto istnieje możliwość dynamicznego utworzenia kolekcji w trakcie działania serwera aplikacji.

Serwis czujników łączy się z bazą danych za pomocą natywnego klienta NodeJS dla bazy MongoDB. Wobec tego wszystkie operacje po stronie serwera aplikacji są wykonywane za pomocą takich samych instrukcji jak wewnątrz bazy danych. Inicjalizacja kolekcji projektów badawczych w bazie następuje po pierwszym odwołaniu się do kolekcji, a nie w momencie uruchomienia programu. Kod źródłowy aplikacji zawiera instrukcje utworzenia indeksów dla pól *name* oraz *akronim*. Ponadto dla wskazanych pól dodano wymóg przechowywania *unikalnych wartości* w każdym dokumencie. W ramach tej samej sekwencji operacji dodawana jest JSON Schema walidująca poprawność budowy projektu badawczego. JSON Schema musi zostać dodana przed pierwszym odwołaniem się do kolekcji. W przeciwnym wypadku nie można weryfikować schematu projektów badawczych dodawanych do bazy danych. Zamiana JSON Schema dla istniejącej kolekcji może się odbyć tylko z poziomu bazy danych. Jest to operacja niezalecana, ponieważ może prowadzić do rozbieżności w składowanych danych. Opisaną operację zostają wykonane po pierwszym udanym zalogowaniu do aplikacji, ponieważ pierwszym odwołaniem do bazy danych jest instrukcja pobrania nazw projektów na widoku prezentującym listę projektów badawczych.

Kolekcje przeznaczone do składowania pomiarów są tworzone przy pierwszej operacji dodania zasobu do bazy danych. Nazwa kolekcji to akronim zapisany w projekcie badawczym, w ramach którego gromadzone są pomiary. Podczas operacji dodania obiektu do bazy danych nie następuje walidacja schematu pomiaru w bazie. Odpowiedzialność za weryfikację poprawności procesowanych danych spoczywa na serwisie czujników, który odrzuca pomiary nie mieszczące się w danym zakresie lub zawierające określoną wartość błędu.

## 4. Obsługa systemu

### 4.1. Struktura projektu

Kod źródłowy systemu jest przechowywany w ramach jednego repozytorium. Wobec tego dostęp do wszystkich komponentów aplikacji odbywa się z poziomu tego samego katalogu nadrzędnego. Ze względu na wykorzystanie systemu kontroli wersji Git za pośrednictwem platformy GitHub nazwa katalogu nadrzędnego jest odpowiednikiem nazwy zdalnego repozytorium. W momencie pisania niniejszej pracy katalog nadrzędny projektu nazywa się *master-thesis*. W katalogu *master-thesis* znajduje się folder *arch-docs*, który zawiera diagramy prezentujące architekturę aplikacji oraz dokumentację systemu. Podczas uruchomienia systemu zostaje utworzony katalog *docker-volumes*, który odpowiada za przechowywanie kopii danych składowanych w bazach danych. Folder *gradle* zawiera skrypt uruchomieniowy aplikacji Gradle Wrapper używany podczas budowania projektu. Istotną rolę podczas budowania projektu odgrywają również pliki *build.gradle*, *settings.gradle*, *docker-compose.yml*. Proces budowania projektu został opisany w następnym rozdziale. Pozostałe katalogi zawierają kod źródłowy poszczególnych komponentów systemu.

Komponent API Gateway jest budowany na podstawie kodu źródłowego dostępnego w katalogu o nazwie *api-gateway*. W podkatalogu *src/main/java* znajdują się klasy zawierające konfigurację oraz logikę działania aplikacji. Pakiet *configuration* zawiera pliki odpowiedzialne za dostosowanie pracy frameworka Spring Security oraz utworzenie własnych zmiennych środowiskowych. Celem pakietu *exception* jest utworzenie dedykowanej hierarchii wyjątków. W pakiecie *filter* znajduje się logika biznesowa filtra, który służy do weryfikacji uprawnień w żetonie JWT. Pakiet *gateway* zawiera konfigurację ścieżek obsługiwanych przez aplikację API Gateway. W pakiecie *jwt* został umieszczony kod odpowiedzialny za weryfikację poprawności oraz konwertowanie żetonów do postaci obiektów. Podkatalog *src/main/resources* służy do przechowywania pary kluczy RSA wykorzystanych przy kodowaniu i dekodowaniu żetonów, zmiennych środowiskowych oraz własnego banneru startowego aplikacji. W podkatalogu *src/test/java* umieszczono testy jednostkowe kodu do walidacji żetonów oraz test integracyjny sprawdzający działanie konfiguracji bramki.

Serwis autoryzacji jest tworzony na podstawie kodu źródłowego znajdującego się w katalogu *auth-service*. Aplikacja została zaimplementowana w formie architektury warstwowej. Logika odpowiedzialna za tworzenie żetonów JWT na podstawie przekazanego loginu i hasła znajduje się w pakiecie *src/main/java/agh/wfiis/weather/auth*. Pakiet *src/main/java/agh/wfiis/weather/principal* zawiera kod odpowiedzialny za tworzenie kont użytkowników aplikacji oraz zarządzanie uprawnieniami użytkowników. W pakiecie *src/main/java/agh/wfiis/weather/exception* znajduje się implementacja hierarchii wyjątków. Konfiguracje konieczne do prawidłowego działania aplikacji są dostępne w pakiecie *src/main/java/agh/wfiis/weather/config*. Podkatalog *src/main/resources* zawiera parę kluczy RSA wykorzystanych przy kodowaniu i dekodowaniu żetonów, skrypt SQL potrzebny do inicjalizacji relacyjnej bazy danych, zmienne środowiskowe oraz własny baner startowy aplikacji. W podkatalogu *src/test/java* umieszczono testy jednostkowe i testy integracyjne logiki aplikacji. Na szczególną uwagę zasługuje klasa *UserRepositoryTest*, która wykorzystuje bibliotekę *Testcontainers* do stworzenia testowej bazy danych w kontenerze. Warunkiem koniecznym przejścia testu jest dostęp do środowiska Docker.

Do zbudowania aplikacji klienta jest potrzebny kod znajdujący się w katalogu o nazwie *client*. W folderze *public* znajduje się szablon strony internetowej. Folder *src/api* zawiera stałe takie jak adres URL serwera aplikacji, adresy URL widoków aplikacji, zmienne globalne. Logo

AGH zostało umieszczone w katalogu *src/assets*. Katalog *src/components* zawiera kod źródłowy komponentów.

Serwis czujników jest budowany na podstawie kodu źródłowego dostępnego w katalogu *sensor-service*. Logika działania aplikacji znajduje się w folderze *src*. Aplikacja została stworzona na podstawie architektury warstwowej podobnie do serwisu autoryzacji. W katalogu *src/logger* została umieszczona implementacja mechanizmu logowania zdarzeń. Folder *src/middleware* zawiera metody pomocnicze wywoływane przed rozpoczęciem procesowania plików. W folderze *resource* znajduje się JSON Schema, na podstawie której następuje walidacja dokumentów reprezentujących projekty badawcze. Folder *\_\_test\_\_* zawiera testy integracyjne kontrolerów udostępnionych przez serwer aplikacji.

Komponent odpowiedzialny za działanie mechanizmu Service Discovery jest tworzony na podstawie kodu źródłowego znajdującego się w katalogu *service-registry*. Komponent nie zawiera implementacji logiki biznesowej, ponieważ do uruchomienia mechanizmu wystarczy dodanie adnotacji `@EnableEurekaServer` nad główną klasą aplikacji. Konfiguracja działania mechanizmu została umieszczona w folderze *src/main/resources*.

## 4.2. Budowanie projektu

System został zaprojektowany w taki sposób, aby każdy komponent mógł zostać zbudowany niezależnie od działania pozostałych elementów aplikacji. Podstawowymi kryteriami uwzględnionymi podczas doboru narzędzi do budowania poszczególnych aplikacji były kompatybilność narzędzia z daną technologią oraz popularność rozwiązania wśród programistów.

Aplikacje napisane przy użyciu technologii Java są budowane za pomocą narzędzia o nazwie Gradle. Gradle służy do budowania niemal każdego rodzaju aplikacji. Dzięki bogatej gamie rozszerzeń umożliwia wygenerowanie plików wykonywalnych dla większości dostępnych języków programowania [4]. Zaletą tego programu jest przejrzysta składnia oparta na językach Groovy lub Kotlin. Ponadto Gradle cechuje się lepszą wydajnością od innego popularnego rozwiązania o nazwie Maven szczególnie pod kątem czyszczenia oraz budowania projektów [4].

Podczas inicjalizacji projektu za pomocą dowolnego narzędzia przygotowany projekt zawiera katalog o nazwie *gradle*. W katalogu *gradle* znajduje się skrypt uruchomieniowy programu Gradle Wrapper. Rolą tego programu jest ustandaryzowanie wersji Gradle w całym projekcie. Dzięki temu użytkownik zostaje uchroniony przed konsekwencjami wykorzystania różnych wersji narzędzia budującego w zależności od komponentu aplikacji. Najważniejszymi elementami w procesie budowania aplikacji są pliki *settings.gradle* oraz *build.gradle*. Plik *settings.gradle* zawiera nazwę głównego komponentu budowanej aplikacji. Ponadto umożliwia dodanie komponentów zależnych, które zostaną utworzone w trakcie jednej sesji budowania. Plik *build.gradle* zawiera wszystkie metadane budowanego projektu. Pozwala na ustawienie repozytoriów, z których zostają pobierane zależności potrzebne do budowy projektu. Ponadto udostępnia funkcjonalności dodawanie własnych skryptów nazywanych zadaniami (ang. *tasks*), które są wykonywane w trakcie procesu budowy.

Gradle został wykorzystany zarówno do budowania pojedynczej aplikacji, jak również do zbudowania kilku komponentów jednocześnie. Serwisem, którego proces budowania jest odseparowany od pozostałych elementów jest API Gateway. Powodem izolacji tego komponentu jest test integracyjny, który wymaga połączenia z wcześniej zbudowanym i uruchomionym serwisem czujników. Z kolei serwis autoryzacji oraz komponent Service Discovery mogą być zbudowane razem za pomocą jednej komendy lub niezależnie od siebie,

ponieważ testy wykonywane podczas procesu ich budowy nie wymagają połączenia z zewnętrznymi serwisami.

Serwis czujników oraz klient aplikacji są budowane za pomocą narzędzie o nazwie npm (Node Package Manager). Npm funkcjonuje analogicznie do narzędzia Gradle, ale jest kompatybilny tylko z frameworkami opartymi na języku JavaScript [5]. Każdy z wymienionych komponentów jest budowany indywidualnie. Menedżer pakietów samoczynnie zaciąga wymagane zależności. Ponadto posiada możliwość dodania własnych skryptów, które mogą usprawnić często wykonywane procesy np. testowanie aplikacji. Metadane aplikacji, zależności oraz ustawienia skryptów znajdują się w pliku *package.json*.

### 4.3. Wdrożenie aplikacji na serwer wydziału

Jednym z problemów jakie należało rozwiązać w trakcie projektowania aplikacji było znalezienie sposobu na względnie szybkie zbudowanie i uruchomienie systemu na dowolnym środowisku. Istotnymi elementami przy wyborze odpowiedniego narzędzia było uwzględnienie popularności wykorzystania danej technologii przez środowisko programistów oraz przejrzyste napisana dokumentacja. Ponadto ze względów formalnych narzędzie musi być darmowe.

Uwzględniając wyżej wymienione kryteria autor pracy podjął decyzję o wykorzystaniu technologii konteneryzacji przy użyciu narzędzia Docker. Każdy kontener zawiera w sobie wszystkie potrzebne zależności do zbudowania oraz uruchomienia danego komponentu aplikacji. Kontenery są tworzone na podstawie obrazów, które w terminologii Dockera oznaczają niezmiennie szablony definiujące reguły budowania kontenerów. Obrazy są definiowane za pomocą instrukcji zawartych w specjalnych plikach o nazwie Dockerfile [6].

W przypadku zastosowania architektury mikrousługowej uruchomienie programu wymaga stworzenia większej ilości kontenerów. Narzędziem dedykowanym do zarządzania rozbudowaną infrastrukturą aplikacji jest Docker Compose, który umożliwia zbudowanie i uruchomienie programu przy użyciu jednej komendy. Reguły budowania projektu są umieszczone w pliku o nazwie *docker-compose.yml*. W ramach budowania wielokontenerowej aplikacji można ustalić m.in. kolejność budowania oraz uruchamiania kontenerów, lokalizacje plików Dockerfile na podstawie których budowane są kontenery, lokalizację pliku zawierającego kopię zapasową dla baz danych.

Ważnym aspektem związanym z wykorzystaniem technologii konteneryzacji jest oddzielenie kontenera od środowiska zewnętrznego. Aplikacja działa w taki sam sposób na dowolnym systemie operacyjnym lub maszynie wirtualnej. Wobec tego można przetestować program na lokalnej maszynie nie wpływając na działanie systemu na środowisku produkcyjnym. W przypadku naprawy błędu wystarczy zreprodukować dane wejściowe, które spowodowały błąd i wprowadzić konieczne poprawki.

W celu uruchomienia aplikacji wymagana jest instalacja na docelowym urządzeniu systemu kontroli wersji Git oraz wyżej opisanego narzędzia Docker. System kontroli wersji Git służy do stworzenia lokalnej kopii zdalnego repozytorium. Jeżeli aplikacja jest uruchamiana na systemie operacyjnym Windows należy zainstalować dodatek o nazwie Git Bash. Uzasadnienie wykorzystania konsoli Git Bash nastąpi w kolejnej sekcji. Narzędzie Docker umożliwia automatyczne zbudowania oraz uruchomienia aplikacji.

W momencie pisania niniejszej pracy kod źródłowy programu jest umieszczony na platformie GitHub. W celu zabezpieczenia wrażliwych informacji dotyczących sposobu budowania API-Key oraz JWT repozytorium zostało oznaczone jako prywatne. Wydział udostępnił maszynę wirtualną wykorzystującą system operacyjny Rocky Linux. Maszyna znajduje się pod adresem *172.20.40.211*. W celu uruchomienia systemu zainstalowałem wszystkie potrzebne aplikacje wymienione powyżej. Działanie aplikacji można sprawdzić tylko i wyłącznie będąc zalogowanym do sieci wydziału. Strona logowania do systemu znajduje się pod adresem *http://172.20.40.211:3000*.

#### 4.4. Pierwsze uruchomienie systemu

Poniższa instrukcja przedstawia wykorzystanie komend w dowolnym wierszu poleceń. Część niżej wymienionych operacji może zostać zastąpiona poprzez wykorzystanie narzędzi posiadających graficzny interfejs użytkownika takich jak IntelliJ IDEA lub Docker Desktop.

Pierwszym krokiem wymaganym do uruchomienia systemu na dowolnej maszynie jest utworzenie lokalnej kopii repozytorium za pomocą komendy *git clone*. Aplikacja do wygenerowania JWT wykorzystuje parę asymetrycznych kluczy służących do kodowania oraz dekodowania żetonów. W związku z tym do prawidłowego działania mechanizmu autoryzacji wymagane jest utworzenie klucza publicznego oraz klucza prywatnego. W tym celu trzeba przenieść się do lokalizacji *master-thesis/auth-service/src/main/resources*. Następnie administrator musi utworzyć nowy katalog o nazwie *certs*. W katalogu *certs* należy wykonać poniższą sekwencję komend.

```
1 openssl genrsa -out keypair.pem 2048
2 openssl rsa -in keypair.pem -pubout -out public.pem
3 openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in keypair.pem -out private.pem
```

##### Listing 1 Utworzenie klucza publicznego oraz klucza prywatnego

Biblioteka OpenSSL jest dostępna na systemach operacyjnych MacOS oraz Linux. W przypadku systemu operacyjnego Windows rekomendowane jest zainstalowanie konsoli Git Bash, która jest dodatkiem przy instalacji systemu kontroli wersji Git i zawiera wbudowaną wersję biblioteki OpenSSL. Pierwsza linia przedstawionego powyżej skryptu generuje klucz prywatny za pomocą algorytmu RSA o rozmiarze 2048 bitów, a następnie zapisuje go do pliku *keypair.pem*. Druga instrukcja procesuje wygenerowany klucz prywatny i generuje na jego podstawie klucz publiczny zapisując go do pliku *public.pem*. Ostatnia komenda tworzy klucz prywatny na podstawie danych z pliku *keypair.pem* w formacie PKCS8 i zapisuje go do pliku *private.pem*. Opcja *-nocrypt* została użyta ze względu na implementację obsługi JWT we frameworku Spring Boot Security. Po utworzeniu plików *public.pem* oraz *private.pem* można usunąć plik *keypair.pem* oraz skopiować cały katalog do folderu *master-thesis/api-gateway/src/main/resources*.

Kolejnym krokiem jest wykorzystanie narzędzia Docker Compose. W tym celu należy przejść do folderu *master-thesis/* i utworzyć kontenery za pomocą poniższej komendy.

```
1 docker compose up -d
```

##### Listing 2 Zbudowanie oraz uruchomienie aplikacji

Kontenery zostają utworzone zgodnie z instrukcją zawartą w pliku *docker-compose.yml*. Parametr *-d* umożliwia wyłączenie śledzenia logów aplikacji po zakończeniu procesu budowania kontenerów.

Aplikacja po pierwszym uruchomieniu zawiera puste bazy danych. Dane logowania dla utworzonych użytkowników znajdują się w pliku *docker-compose.yml*. Utworzenie schematu bazy danych wymaga wykorzystania narzędzia konsolowego *psql*. Poniższa komenda umożliwi uruchomienie wiersza poleceń PostgreSQL.

```
1 | docker exec -it master-thesis-postgres-1 psql -U postgres
```

#### **Listing 3** Uruchomienie wiersza poleceń dla użytkownika *postgres*

Z perspektywy konsoli PostgreSQL wprowadzanie zapytań odbywa się w kontekście użytkownika *postgres*. Baza danych musi nazywać się *auth*. Kolejnym elementem jest przejście do konsoli PostgreSQL w kontekście nowoutworzonej bazy danych.

```
1 | docker exec -it master-thesis-postgres-1 psql -U postgres -W auth
```

#### **Listing 4** Uruchomienie wiersza poleceń dla bazy danych *auth*

Do utworzenia schematu bazy relacyjnej należy wykorzystać skrypt, który zawiera polecenia tworzące tabele oraz polecenia wstawiające podstawowe encje. Skrypt znajduje się w pliku *master-thesis/auth-service/src/main/resources/db-scripts/init\_db.sql*.

Ostatnim elementem koniecznym do rozpoczęcia korzystania z aplikacji jest utworzenie pierwszego konta użytkownika o uprawnieniach administratora. Użytkownik ma przypisaną rolę administratora, ponieważ jest to jedyna rola, która daje uprawnienie do stworzenia nowego konta z poziomu klienta aplikacji. Serwis służący do autoryzacji podczas tworzenia nowego konta koduje hasło za pomocą funkcji haszującej o nazwie *bcrypt*. W konsekwencji nie można dodać nowego użytkownika z poziomu bazy danych, ponieważ podczas operacji logowania niezakodowane hasło znajdujące się w bazie będzie się różniło od hasła przetworzonego przez serwer aplikacji. Jedynym sposobem dodania nowego utworzenia z poziomu wiersza poleceń jest wykorzystanie biblioteki *cURL*.

```
1 | curl -d
2 | '{ "username": "Admin",
3 |   "email": "admin@agh.edu.pl",
4 |   "description": "Admin user",
5 |   "password": "admin",
6 |   "roles": ["ADMIN"],
7 |   "projects": [] }'
8 | -H "Content-Type: application/json"
9 | -X POST http://localhost:8080/users
```

#### **Listing 5** Utworzenie konta użytkownika przy użyciu biblioteki *cURL*

Powyższe polecenie wysyła żądanie HTTP typu POST. Do wysłanego żądania dołączony jest obiekt JSON (JavaScript Object Notation). W ciele obiektu JSON znajdują się dane potrzebne do stworzenia nowego konta. Alternatywnym sposobem wysłania powyższego



zapytania jest wykorzystanie narzędzia o nazwie Postman. Postman udostępnia graficzny interfejs użytkownika, który znacząco ułatwia zmiany parametrów wykonywanych zapytań.

Po utworzeniu konta użytkownika można zalogować się do systemu działającego pod adresem `http://172.20.40.211:3000`.

#### 4.5. Wdrażanie zmian w systemie

Jednym z elementów cyklu życia oprogramowania jest utrzymanie aplikacji. Przytoczony termin może być rozumiany jako naprawa błędów lub wdrożenie nowych funkcjonalności. Moment wprowadzania nowej wersji kodu źródłowego na serwerze wydziału musi wiązać się z zatrzymaniem dotychczas działającej aplikacji. W przypadku komponentu działającego w obrębie kontenera operacja wymaga przeprocesowania następującej sekwencji komend.

```
1 git pull origin main
2 docker stop auth-service
3 docker rm auth-service
4 docker rmi auth-service
5 docker compose up -d
```

**Listing 6** Aktualizacja aplikacji działającej w kontenerze

Powyższy skrypt stanowi przykład aktualizacji serwisu o nazwie auth-service. Pierwsza z wymienionych komend pobiera do lokalnego repozytorium zmiany wprowadzone w kodzie źródłowym. Kolejna instrukcja zatrzymuje działanie kontenera. Trzecie polecenie usuwa starą wersję kontenera. Komenda w czwartej linii usuwa stary obraz na podstawie którego zbudowany był wcześniejszy kontener. Ostatnia instrukcja sprawdza status działania kontenerów, które są wymienione w pliku `docker-compose.yml`. Jeżeli któryś z kontenerów nie prezentuje statusu „Running”, Docker tworzy na nowo obraz i na podstawie tego obrazu buduje kontener, a następnie uruchamia aplikację. Analogiczny zestaw komend może być wykorzystany dla dowolnego komponentu tworzącego aplikację. W celu sprawdzenia, czy nowa wersja aplikacji uruchomiła się w sposób prawidłowy można wykorzystać komendę `docker logs`.

#### 4.6. Monitorowanie działania systemu

Podczas działania aplikacji mogą wystąpić błędy lub nieoczekiwane zachowania. W takich sytuacjach zadaniem administrator systemu jest sprawdzenie przyczyny błędu w logach aplikacji. Każdy serwis generuje własne logi niezależnie od działania pozostałych serwisów. Architektura monolityczna charakteryzuje się działaniem wszystkich funkcjonalności w ramach jednolitego serwera aplikacyjnego. W związku z tym archiwizuje każde zdarzenie konieczne do wykonania żądanej operacji niezależnie od złożoności operacji wykonywanej przez użytkownika. Przy zastosowaniu architektury mikrousługowej odpowiedzialność za realizację złożonych operacji może zostać podzielona na kilka niezależnych serwisów. Zatem w sytuacji wystąpienia błędu administrator jest zmuszony do przejrzenia logów zdarzeń w każdym komponentcie uczestniczącym w realizacji żądania.

Kolejnym nieoczywistym problemem podczas debugowania aplikacji opartej na architekturze mikrousługowej jest rozstrzygnięcie skąd pochodzą dane prezentowane w poszczególnych komponentach wyświetlanych po stronie klienta aplikacji. W opisywanym systemie do akwizycji danych z rozproszonych systemów pomiarowych funkcjonalność zmiany uprawnień użytkownika przez administratora systemu jest realizowana na widoku panelu administratora. Jedno z okien dialogowych zawiera listę nazw projektów badawczych. Użytkownik aplikacji może pomyśleć, że skoro akcja dotyczy zmiany uprawnień innego użytkownika to wszystkie dane powinny pochodzić z serwisu odpowiedzialnego za autoryzację oraz zarządzanie uprawnieniami. W rzeczywistości nazwy projektów badawczych pochodzą z bazy danych serwisu do zarządzania czujnikami pomiarowymi. Wobec tego w przypadku wystąpienia problemu z nazwami projektów przyczyn błędu należy szukać w serwisie do zarządzania czujnikami. Wskazany przykład pokazuje, że kluczową rolę w działaniu całego systemu odgrywa starannie zaimplementowane logowanie oraz archiwizowanie zdarzeń w aplikacji.

Wśród najczęściej spotykanych przyczyn problemów w aplikacjach internetowych można wskazać zatrzymanie działania serwisu biorącego udział w operacji lub błąd w implementacji funkcjonalności. W celu weryfikacji działania kontenerów tworzących aplikację należy wylistować wszystkie aktualnie działające kontenery.

```
1 | docker compose ps
```

**Listing 7** Komenda wyświetlająca listę działających kontenerów

Przedstawiona komenda musi zostać wykonana z poziomu katalogu, w którym znajduje się plik *docker-comopose.yml*.

NAME	IMAGE	CREATED	STATUS
api-gateway	api-gateway:latest	47 hours ago	Up 47 hours
auth-service	auth-service:latest	47 hours ago	Up 47 hours
client	client:latest	47 hours ago	Up 47 hours
master-thesis-mongodb-1	mongo:6.0.5	7 days ago	Up 7 days
master-thesis-postgres-1	postgres:15.3	7 days ago	Up 7 days
sensor-service	sensor-service:latest	43 hours ago	Up 43 hours
service-registry	service-registry:latest	5 days ago	Up 5 days

**Zdjęcie 4** Fragment wyniku działania polecenia *docker compose ps* dla serwisu do akwizycji danych

Na Zdjęciu 1 usunięto kolumny **COMMAND**, **SERVICE** oraz **PORTS**. Wynik komendy widoczny na zdjęciu prezentuje kolejno od lewej nazwę kontenera, wersję obrazu na podstawie której zbudowano kontener, czas utworzenia kontenera oraz aktualny status pracy kontenera. Kolumna **COMMAND**, która nie została przedstawiona na zdjęciu zawiera informację o komendzie wykorzystanej do uruchomienia aplikacji w kontenerze. Kolumna **SERVICE** określa nazwę aplikacji w grupie kontenerów zarządzanych przez Docker Compose. Kolumna **PORT** przedstawia numery portów na jakich działają skonteneryzowane aplikacje oraz mapowania tych portów na porty wystawione w sieci zewnętrznej. Status działania poszczególnych aplikacji w formie graficznej można sprawdzić również za pomocą narzędzia Docker Desktop.

W celu odczytania logów z poszczególnych serwisów można wykorzystać komendę *docker logs*. Polecenie wyświetla zarchiwizowane informacje dotyczące działania aplikacji dla wskazanego serwisu. Framework Spring Boot umieszcza w logach informacje na temat błędów w postaci stosu wywołania metody w danej chwili czasowej wraz z opisem błędu. NodeJS nie

zapewnia sposobu logowania zdarzeń w aplikacji. Jednym z etapów implementacji serwisu było dodanie wpisów prezentujących informacje na temat statusu wykonywania danej operacji.

Alternatywnym sposobem analizy działania całego systemu jest wykorzystanie mechanizmu Service Discovery. Mechanizm udostępnia panel administratora w formie graficznego interfejsu użytkownika dostępnego z poziomu przeglądarki internetowej. Jedną z funkcjonalności oferowanych przez wyżej wymienioną usługę jest prezentowanie aktualnie działających serwisów wraz ze statusem ich działania.

#### 4.7. Połączenie z bazą MongoDB

Baza danych MongoDB jest wykorzystywana do gromadzenia danych dotyczących projektów badawczych oraz pomiarów przesyłanych przez czujniki lub przekazywanych za pośrednictwem klienta aplikacji w formie plików CSV. Stworzenie bazy danych oraz kolekcji odbywa się przy pierwszym odwołaniu do zasobu z poziomu aplikacji sensor-service. W momencie pierwszego połączenia z kolekcją dodawana jest JSON Schema używana do sprawdzenia poprawności przekazywanych obiektów oraz tworzone są indeksy przyspieszające proces wyszukiwania danych. W przyszłości w ramach rozbudowy aplikacji może wystąpić potrzeba modyfikacji pierwotnych ustawień. MongoDB udostępnia narzędzie konsolowe o nazwie Mongo Shell, które umożliwia wykonanie operacji na bazie danych z poziomu wiersza poleceń. Poniższa komenda umożliwi uruchomienie wspomnianego narzędzi dla bazy MongoDB umieszczonej w kontenerze.

```
1 docker exec -it \  
2 master-thesis-mongodb-1 \  
3 mongosh mongodb://admin:admin@localhost:27017/weather?authSource=admin
```

**Listing 8** *Uruchomienie konsoli Mongo Shell w kontekście bazy danych weather*

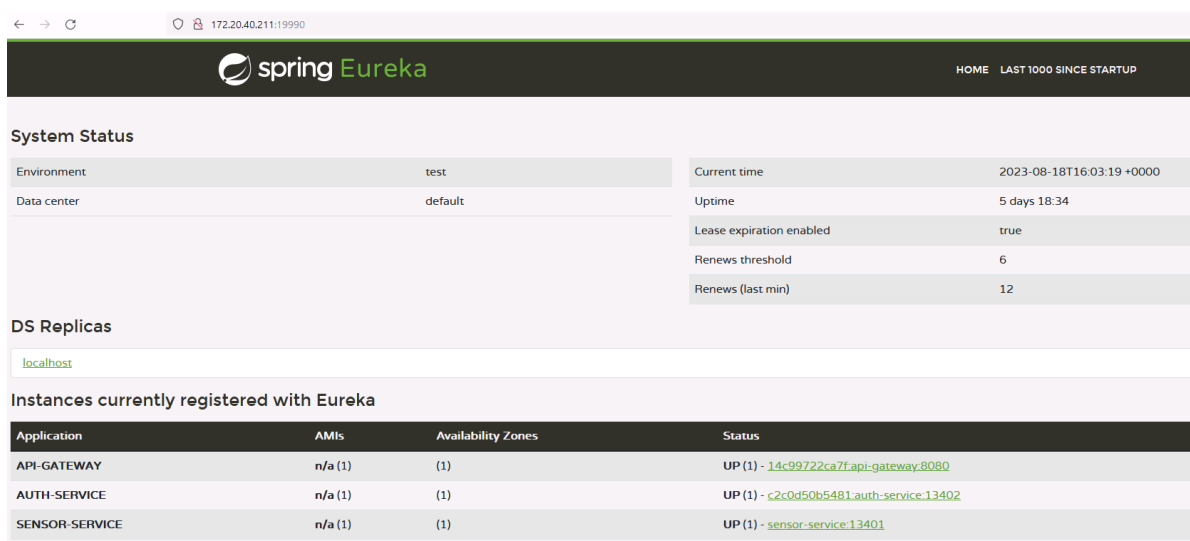
## 5. Dokumentacja komponentów

### 5.1. Service Registry

Service Registry jest wzorcem wykorzystywanym podczas budowania systemów bazujących na architekturze mikrousługowej. Jego zadaniem jest monitorowanie aktualnego stanu poszczególnych komponentów tworzących aplikację. Usługa może zostać określona jako „baza danych” dostępnych serwisów [7], ponieważ gromadzi informacje na temat działających serwisów wraz z numerami portów zajmowanych przez poszczególne instancje. Ponadto Service Registry umożliwia sprawdzenie aktualnej kondycji danej instancji poprzez bezpośrednie połączenie z węzłem końcowym odpowiedzialnym za zwracanie informacji na temat stanu aplikacji. Wykorzystanie tego wzorca wymaga zapewnienia ciągłej dostępności serwisu będącego serwerem usługi, ponieważ inne komponenty takie jak API Gateway lub Routery rozprawdają żądania po wewnętrznych serwisach bazując na informacjach zgromadzonych w Service Registry.

Service Registry zaimplementowany w ramach serwisu do akwizycji danych opiera się na rozwiązaniu oferowanym przez framework Spring Cloud Netflix Eureka. Zadaniem serwisu jest uruchomienie usługi i oczekiwanie na połączenie przez klientów aplikacji. Jeżeli zewnętrzna aplikacja zostanie zarejestrowana w usłudze Spring Eureka, serwer wysyła co pewien czas zapytanie do klienta usługi w celu sprawdzenia jej statusu.

Wykorzystanie wzorca Service Registry może przynieść wymierne korzyści dla użytkowników aplikacji. Framework Spring Cloud Netflix Eureka Client umożliwia zastosowanie wzorca o nazwie client-side discovery. Jeżeli wszystkie możliwe lokalizacje instancji serwisów są zarejestrowane w jednym miejscu, aplikacja kliencka może odwołać się do dowolnej instancji pożądanego serwisu poprzez podanie jedynie nazwy serwisu. Klient aplikacji wysyłający żądanie sprawdzi w usłudze Service Registry dostępne porty i za pomocą algorytmu realizującego load balancing będzie rozdzielał żądania równomiernie pomiędzy wszystkie dostępne instancje. Dzięki temu żądanie zostanie obsłużone szybciej. Wadą tego podejścia jest konieczność zaimplementowania logiki obsługującej połączenie klienta usługi Service Registry w każdym z serwisów wewnętrznych. Warto również odnotować, że skutkiem zastosowania tego wzorca jest uzależnienie każdej aplikacji klienckiej od działania usługi Spring Eureka.



The screenshot shows the Spring Eureka web interface. At the top, there's a navigation bar with the Spring Eureka logo and links for HOME and LAST 1000 SINCE STARTUP. Below this, the 'System Status' section is divided into two columns. The left column shows 'Environment: test' and 'Data center: default'. The right column shows 'Current time: 2023-08-18T16:03:19+0000', 'Uptime: 5 days 18:34', 'Lease expiration enabled: true', 'Renews threshold: 6', and 'Renews (last min): 12'. Below the system status, there's a section for 'DS Replicas' showing 'localhost'. At the bottom, the 'Instances currently registered with Eureka' section contains a table with columns: Application, AMIs, Availability Zones, and Status.

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 14c99722ca7f:api-gateway:8080
AUTH-SERVICE	n/a (1)	(1)	UP (1) - c2c0d50b5481:auth-service:13402
SENSOR-SERVICE	n/a (1)	(1)	UP (1) - sensor-service:13401

Zdjęcie 5 Graficzny interfejs użytkownika usługi Service Registry

W ramach panelu administratora oferowanego przez usługę Spring Eureka można odczytać takie informacje jak nazwy poszczególnych serwisów, status działania danego serwisu oraz ilość instancji wraz z numerami portów na których uruchomione są poszczególne instancje aplikacji. Każda z nazw instancji serwisu jest linkiem pozwalającym na przekierowanie administratora systemu do strony przedstawiającej aktualny stan działania serwisu. Ponadto interfejs użytkownika zawiera informacje o ilości dostępnej pamięci, stopniu wykorzystania pamięci, czasie działania aplikacji, adresie IP oraz statusie działania serwera Eureka. Aplikacja dokonuje również archiwizacji historii połączeń aplikacji klienckich z usługą Spring Eureka, co może ułatwić znalezienie przyczyn zatrzymania działania poszczególnych serwisów.

## 5.2. API Gateway

API Gateway to wzorzec wykorzystywany przy budowie systemów bazujących na architekturze mikrousługowej, którego rolą jest oddzielenie wewnętrznej struktury systemu od świata zewnętrznego. Bramka wykorzystywana przez system do akwizycji danych bazuje na rozwiązaniu oferowanym przez framework Spring Cloud Gateway. Warto odnotować, że w celu zapewnienia obsługi obciążenia sieciowego skutkującego powielaniem instancji serwisów, API Gateway został zaprojektowany przez specjalistów jako narzędzie reaktywne [8]. Paradygmat programowania reaktywnego pozwala programistą na budowanie nieblokujących, asynchronicznych aplikacji [9], które mają zdolność obsługi przeciążenia systemu spowodowanego dużą liczbą wykonanych żądań. Podstawową funkcją realizowaną przez komponent jest przekierowanie żądania do odpowiedniego serwisu na podstawie zarejestrowanych ścieżek. Każda ścieżka jest definiowana na podstawie adresu URI. Dopasowanie żądania z sieci zewnętrznej do ścieżki zarejestrowanej w API Gateway odbywa się na podstawie predykatów walidujących adres URI, typ metody HTTP, załączone nagłówki oraz parametry żądania. Dodatkową funkcjonalnością oferowaną przez Spring Cloud Gateway jest możliwość dodania własnych filtrów. Filtry mogą zostać wykonane przed przekierowaniem żądania do odpowiedniego serwisu oraz po otrzymaniu odpowiedzi od wewnętrznego serwisu.

Pierwszym krokiem obsługi zapytania przez bramkę jest próba dopasowania zapytania do zarejestrowanych ścieżek. Następnie zapytanie jest przekazywane do łańcucha filtrów odpowiedzialnego za sprawdzenie, czy żądanie spełnia oczekiwane warunki. W ostatnim kroku filtry procesują odpowiedź wysłaną przez wewnętrzny serwis.

Bramka zaimplementowana w ramach systemu do akwizycji danych poza przekierowaniem żądania do odpowiedniego serwisu dokonuje sprawdzenia uprawnień użytkownika. Uprawnienia umieszczone są w żetonie JWT. Żeton jest załączone do żądania w nagłówku o nazwie Authorization. W procesie przekształcenia JWT z typu tekstowego na obiekt następuje sprawdzenie poprawności przekazanego żetonu. Jeżeli użytkownik wysyłający żądanie nie posiada wymaganego uprawnienia lub przekazany żeton nie jest wiarygodny aplikacja zwraca odpowiedź ze statusem Unauthorized. Funkcjonalności, które nie wymagają posiadania żadnych uprawnień to: listowanie nazw projektów badawczych zapisanych w systemie, autoryzacja, utworzenie konta użytkownika, dodawanie pomiaru przez czujnik oraz wszystkie operacje związane z czytaniem pomiarów. Poniższa lista zawiera podział funkcjonalności pod względem wymaganego uprawnienia.

Upewnienie *READ\_PROJECT*:

- pobranie projektu na podstawie nazwy projektu
- pobranie projektu na podstawie akronimu przypisanego do projektu
- pobranie informacji o akcjach dostępnych dla użytkownika przypisanego do projektu

Uprawnienie *CREATE\_PROJECT*:

- dodanie zasobu reprezentującego projekt badawczy

Uprawnienie *DELETE\_PROJECT*:

- usunięcie zasobu na podstawie nazwy projektu badawczego

Uprawnienie *ADD\_MEASUREMENT*:

- dodanie pomiarów zawartych w pliku

Uprawnienie *UPDATE\_PRIVILEGES*:

- pobranie listy wszystkich użytkowników zapisanych w systemie
- aktualizacja ról i projektów przypisanych do użytkownika
- aktualizacja widoczności akcji dla projektu, do którego użytkownik jest przypisany
- usunięcie projektu z listy projektów przypisanych do użytkownika

### 5.3. Serwis autoryzacji

Celem aplikacji służącej do autoryzacji jest zapewnienie możliwości bezpiecznego logowania do systemu. Ponadto zaimplementowany serwis umożliwia zarządzanie rolami i uprawnieniami użytkowników w zakresie przynależności do projektów badawczych. Do stworzenia aplikacji wykorzystano język programowania Java w wersji 17 oraz framework Spring Boot w wersji 3.0.3. Java jest jedną z najpopularniejszych technologii wykorzystywanych do tworzenia aplikacji biznesowych. Framework Spring Boot znacząco przyspiesza pisanie kodu, ponieważ zwalnia programistę z konieczności konfiguracji zewnętrznych serwerów aplikacyjnych. Ponadto przy zastosowaniu dodatkowego frameworka o nazwie Hibernate osoba odpowiedzialna za rozwój aplikacji nie musi konfigurować połączenia z bazą danych i może skupić się na rozwijaniu obiektów odzwierciedlających encje bazodanowe. Dane przechowywane są w bazie PostgreSQL. Główną motywacją wybrania bazy relacyjnej jest uwzględnienie występowania relacji pomiędzy rolą posiadaną przez użytkownika oraz uprawnieniami przypisanymi do danej roli. Aplikacja wykorzystuje wbudowany serwer aplikacyjny Tomcat i działa na porcie 13402.

Funkcjonalność logowania do systemu jest oparta na technologii JWT (JSON Web Tokens). Żetony JWT są bezpiecznym sposobem przesyłania danych pomiędzy niezależnymi komponentami. Zaimplementowany system umożliwia utworzenie, walidację oraz weryfikację poprawności budowy żetonu. JWT wygenerowane przez serwis zawierają podpis cyfrowy bazujący na kluczu publicznym i kluczu prywatnym. Para kluczy zostaje utworzona za pomocą algorytmu RSA przed pierwszym uruchomieniem systemu. Prawidłowy żeton jest zbudowany z trzech części: nagłówka, treści i sygnatury. W części nagłówka zawarte są informacje na temat typu obiektu oraz algorytmu wykorzystanego do utworzenia podpisu cyfrowego. Treść żetonu zawiera żądania [10], które są informacjami na temat użytkownika oraz dodatkowymi danymi. JWT utworzony przez serwis autoryzacji w części przechowującej treść zawiera informację na temat podmiotu emitującego żeton, datę wygenerowania żetonu, datę wygaśnięcia ważności żetonu, nazwę użytkownika żądającego utworzenia żetonu oraz uprawnienia. Zakres uprawnień tworzony jest na podstawie ról przypisanych do konta danego użytkownika. Użytkownik posiadający rolę *Administratora* posiada wszystkie dostępne uprawnienia. Osoba posiadająca rolę *Twórca Projektów* może czytać informacje na temat projektów oraz tworzyć nowe instancje projektów. Rola *Badacz* uprawnia do czytania informacji na temat projektów, czytania ostatnich pomiarów oraz do dodawania nowych pomiarów przesyłanych za pośrednictwem

klienta aplikacji w formie plików CSV. Zarówno nagłówek jak i treść żetonu są kodowane za pomocą szyfru Base64. Do utworzenia sygnatury JWT wykorzystywany jest zaszyfrowany nagłówek, znak kropki, zaszyfrowana treść oraz podpis cyfrowy. Następnie całość jest kodowana za pomocą algorytmu przekazanego w nagłówku żetonu.

Po sukcesywnym logowaniu użytkownik otrzymuje własny żeton dostępu. JWT zostaje zapisane w lokalnym oknie przeglądarki. Żeton jest dołączony do każdego żądania wysyłanego do serwera aplikacji w nagłówku *Authorization* z przedrostkiem Bearer. Jeżeli użytkownik aplikacji podejmuje próbę wykonania akcji, do której nie posiada wymaganych uprawnień, API-Gateway odpowiadający za filtrowanie zapytań zwraca odpowiedź o statusie *Unauthorized*. W momencie wygaśnięcia uprawnień następuje wylogowanie i przekierowanie użytkownika do strony logowania.

Niemniej ważną funkcją serwisu autoryzacji jest zarządzanie użytkownikami oraz uprawnieniami. Serwis umożliwia utworzenie konta użytkownika, przypisanie użytkownikom ról *Badacz* lub *Twórca Projektów*, wylistowanie kont, przypisanie do projektów badawczych, zarządzanie projektami badawczymi oraz rolami użytkownika. Wyżej wymienione funkcjonalności są wykorzystywane na widoku panelu administratora. Warty odnotowania faktem jest brak możliwości utworzenia użytkownika z przypisaną rolą Administratora. Jedynym sposobem stworzenia konta o najwyższych uprawnieniach jest wysłanie żądania bezpośrednio na węzeł końcowy. W tym celu należy użyć narzędzi zewnętrznych takich jak Postman lub cURL dostępny z poziomu wiersza poleceń.

Serwis udostępnia węzły końcowe, które umożliwiają komunikację serwera aplikacji ze światem zewnętrznym. Poniżej została przedstawiona lista udostępnionych węzłów końcowych wraz z krótkim opisem funkcjonalności.

## **Klasa AuthenticationController**

### Metody POST

*/authentication/auth* – tworzy żeton JWT dla użytkownika na podstawie loginu i hasła przekazanego za pośrednictwem AuthenticationDto. Jeżeli użytkownik z podanym loginem nie istnieje metoda wyrzuca wyjątek UsernameNotFoundException i zwraca status Unauthorized.

## **Klasa UserController**

### Metody GET

*/users/all* – zwraca listę zawierającą nazwę, przypisane role oraz przypisane projekty każdego użytkownika zarejestrowanego w systemie.

*/users/{username}/{project}* – zwraca kolekcję zawierającą nazwę projektu oraz przypisane akcje. Jeżeli użytkownik o podanej nazwie nie istnieje metoda wyrzuca wyjątek UsernameNotFoundException i zwraca status Unauthorized. Jeżeli użytkownik o podanej nazwie ma przypisaną rolę Administratora, metoda zwraca pustą kolekcję.

### Metody POST

*/users* – tworzy nowe konto użytkownika wykorzystując dane przekazane w obiekcie UserDto. Jeżeli istnieje konto zawierające podaną nazwę użytkownika, metoda wyrzuca wyjątek UserAlreadyExistsException i zwraca status Bad Request.

### Metody PATCH

*/users* – aktualizuje projekty i role przypisane do użytkownika o nazwie przekazanej w obiekcie UserInfoDto. Jeżeli nie istnieje konto zawierające podaną nazwę użytkownika, metoda wyrzuca wyjątek UsernameNotFoundException i zwraca status Unauthorized.

## Klasa ProjectController

### Metody DELETE

*/user-projects/{name}* – usuwa wszystkie projekty o przekazanej nazwie.

### Metody PATCH

*/user-projects* – aktualizuje akcje przypisane do projektów o identyfikatorach przekazanych w obiekcie ProjectActionsDto. Jeżeli identyfikator jednego z projektów nie istnieje, metoda wyrzuca wyjątek ProjectNotFoundException i zwraca status Not Found.

## 5.4. Serwis czujników

Serwis czujników umożliwia wykonanie operacji powiązanych z działaniem czujników pomiarowych. Aplikacja została zaimplementowana przy użyciu języka JavaScript oraz frameworka NodeJS w wersji 18.15.0. Język JavaScript ma szerokie zastosowanie w rozwoju aplikacji webowych. Umożliwia tworzenie nowych funkcjonalności zarówno po stronie klienta aplikacji, jak również serwera. NodeJS jest jednym z najczęściej wykorzystywanych frameworków do implementacji serwerów aplikacji. Stanowi idealne narzędzie do budowania systemów realizujących procesowanie dużych ilości danych. Głównym argumentem przemawiającym za wyborem NodeJS jest wykorzystanie przez framework mechanizmu pętli zdarzeń, która pozwala wykonywać dużą ilość krótkotrwałych operacji w ramach pracy jednego wątku aplikacji. Dane przetwarzane przez serwis są gromadzone w bazie danych MongoDB. Wybór bazy dokumentowej został podyktowany koniecznością procesowania danych o niejednorodnym schemacie. W momencie pierwszego uruchomienia aplikacji zostaje zdefiniowana jedynie kolekcja służąca do przechowywania informacji o projektach badawczych. Pozostałe kolekcje odpowiedzialne za gromadzenie pomiarów z urządzeń badawczych są tworzone dynamicznie w trakcie działania programu. Aplikacja działa na porcie 13401.

Agregacja danych z rozproszonych systemów pomiarowych opiera się na funkcjonalności zarządzania czujnikami w obrębie projektów badawczych. Serwis udostępnia możliwość utworzenia nowego projektu badawczego, wylistowania nazw wszystkich istniejących projektów badawczych, wypisania informacji na temat pojedynczego projektu badawczego oraz usunięcia projektu. Wyszukanie konkretnego projektu badawczego może odbyć się na podstawie nazwy projektu lub przypisanego akronimu. Przykłady prawidłowych URI (Uniform Resource Identifier) służących do wyszukiwania zostały przedstawione w części rozdziału opisującej węzły końcowe udostępnione przez serwis. Usunięcie projektu odbywa się na podstawie przekazanej nazwy. W ramach klienta aplikacji dodano możliwość klonowania projektów. Klonowanie oznacza utworzenie nowego projektu na podstawie danych zawartych w innym projekcie badawczym. Czujnik może monitorować nawet kilkadziesiąt parametrów. Głównym celem tej funkcjonalności jest przyspieszenie procesu tworzenia nowych projektów, które wykorzystywałyby takie same lub zbliżone modele urządzeń pomiarowych. Proces klonowania odbywa się poprzez pobranie projektu badawczego na podstawie jego nazwy z bazy danych. Następnie po dostosowaniu struktury nowego projektu użytkownik **musi** zmienić nazwę projektu oraz akronim. Nazwa projektu oraz akronim są parametrami unikalnymi w bazie danych. W związku z tym przy próbie utworzenia projektu zawierającego istniejącą w bazie nazwę lub akronim użytkownik zostanie poinformowany o błędzie tworzenia projektu. Funkcjonalność klonowania jest dostępna tylko dla użytkowników posiadających rolę *Twórca Projektów*.



Podstawowym zadaniem serwisu jest umożliwienie zapisu pomiarów zarejestrowanych przez urządzenia pomiarowe do bazy danych. Przesłanie danych może odbyć się za pośrednictwem protokołu HTTP lub poprzez załączenie pliku.

Pierwszy z wymienionych sposobów wymaga ustawienia w urządzeniu pomiarowym adresu URL serwera aplikacji. Do żądania musi zostać dołączony nagłówek X-API-Key zawierający w polu wartości prawidłowo zbudowany klucz. Klucz API jest jednym z najprostszych sposobów weryfikacji aplikacji lub urządzenia próbującego dokonać połączenia z innym serwisem. Jedynym wymogiem ciężącym na obydwu stronach połączenia jest uzgodnienie sposobu budowy klucza. Wadą tej metody jest brak standaryzacji. W związku z tym klucz może być przesyłany w formie nagłówka żądania, jako parametr żądania lub w ciele żądania. Serwis czujników oczekuje klucza zbudowanego w następujący sposób: nazwa projektu, kropka, akronim, kropka, data w formie ISO, znak spacji, godzina w UTC+0, dwukropek, minuty, kropka, długość nazwy projektu. Utworzony tekst jest kodowany za pomocą algorytmu MD5.

Proces dodania pomiaru przesłanego za pomocą protokołu HTTP wymaga wykonania ściśle określonej sekwencji czynności. W pierwszym kroku dokonywane jest sprawdzenie, czy projekt dla którego należy dokonać zapisu pomiaru nie jest projektem typu OFFLINE. Projekty OFFLINE umożliwiają dodanie danych tylko z pliku, zatem w przypadku próby dodania pomiaru zarejestrowanego przez czujnik zostaje wyrzucony wyjątek. Kolejnym etapem jest weryfikacja poprawności klucza API. W przypadku odnotowania nieprawidłowej budowy klucza API pomiar nie zostanie zapisany i użytkownik zostanie poinformowany o błędzie ze statusem Unauthorized. Jeżeli klucz API został pomyślnie zweryfikowany, następuje faza walidacji przesłanych wartości. Z dokumentu reprezentującego projekt badawczy pobierana jest lista parametrów, które należy walidować. W drugim kroku następuje sprawdzenie, czy wartość parametru mieści się w oczekiwanym zakresie oraz czy przesłana wartość nie jest równa oczekiwanej wartości błędu. Po przeprowadzeniu weryfikacji pomiaru następuje sprawdzenie, czy zostały wykryte błędy. Jeżeli na tym etapie serwis odnotuje błąd, pomiar nie zostanie zapisany i użytkownik zostanie poinformowany o błędzie ze statusem Internal Server Error. W przypadku braku błędów walidacyjnych pomiar zostanie zapisany w bazie danych. Sukcesywnie zakończony proces dodania pomiaru powinien zwrócić status Created.

Pomiary dodane w formie pliku nie wymagają walidacji klucza API. Plik z danymi można przesłać za pośrednictwem klienta aplikacji lub narzędzia służącego do wysyłania żądań HTTP. Serwis umożliwia tylko i wyłącznie dodanie pomiarów zapisanych w formie pliku CSV (ang. comma-separated values, wartości rozdzielone przecinkiem). Autor pracy zaleca, aby plik z danymi nie zawierał wiersza z nazwami kolumn, ani żadnych innych wierszy nie będących pomiarami. W pierwszym kroku aplikacja przetwarza wiersz z pliku na strukturę dokumentu możliwego do zapisu w bazie danych. Następnie odbywa się walidacja poprawności danych, która przebiega w taki sam sposób jak dla pomiarów przesłanych za pomocą protokołu HTTP. Jeżeli zostanie wykryty błąd w wierszu to dany wiersz nie zostaje zapisany. Wykrycie błędu w pojedynczym wierszu nie wpływa na zapisanie pozostałych danych zawartych w pliku. Na końcu procesu w logach aplikacji zostają zarchiwizowane błędy wykryte podczas walidacji poszczególnych wierszy. Wszystkie prawidłowe wiersze zostają zapisane w bazie danych. Proces zapisania plików o dużych rozmiarach jest czasochłonny. Główną przyczyną tego zjawiska jest konieczność synchronicznego przetworzenia wierszy w pliku na strukturę bazodanową oraz zapis dużych ilości danych w bazie MongoDB. Wiersze muszą zostać przetworzone synchronicznie, aby można było odfiltrować rekordy zawierające błędne dane. Sukcesywnie zakończony proces dodawania pomiarów powinien zwrócić status Created.

Serwis udostępnia węzły końcowe, które umożliwiają komunikację serwera aplikacji ze światem zewnętrznym. Poniżej została przedstawiona lista udostępnionych węzłów końcowych wraz z krótkim opisem funkcjonalności.

### **Klasa ProjectController**

#### Metody GET

*/projects/names* – zwraca listę zawierającą nazwy wszystkich projektów znajdujących się w systemie.

*/projects?name={value}* – zwraca dane zasobu reprezentującego projekt na podstawie przekazanej nazwy.

*/projects?acronym={value}* – zwraca dane zasobu reprezentującego projekt na podstawie przekazanego akronimu.

#### Metody POST

*/projects* – tworzy nowy zasób zawierający dane projektu na podstawie danych przekazanych w ciele zapytania.

#### Metody DELETE

*/projects/{name}* – usuwa zasób reprezentujący projekt na podstawie przekazanej nazwy.

### **Klasa MeasurementController**

#### Metody GET

*/measurements/{acronym}* – zwraca listę zawierającą wszystkie pomiary zapisane w ramach programu badawczego o przekazanym akronimie.

*/measurements/{acronym}/latest* – zwraca listę zawierającą trzy ostatnio zapisane pomiary w ramach programu badawczego o przekazanym akronimie.

#### Metody POST

*/measurements/{acronym}/{deviceId}* – tworzy zasób reprezentujący pomiar. Pomiary są gromadzone w ramach projektu badawczego o przekazanym akronimie. Walidacja poprawności pomiaru odbywa się dla czujnika posiadającego przekazany identyfikator czujnika.

*/measurements/upload/{acronym}/{deviceId}* – tworzy zasoby reprezentujące pomiary na podstawie przekazanego pliku CSV. Pomiary są gromadzone w ramach projektu badawczego o przekazanym akronimie. Walidacja poprawności pomiaru odbywa się dla czujnika posiadającego przekazany identyfikator czujnika.

## **5.5. Klient aplikacji**

W celu zapewnienia użytkownikom przyjemnych wrażeń podczas korzystania z systemu do akwizycji danych zaprojektowano graficzny interfejs użytkownika umożliwiający wykonanie przeważającej części dostępnych operacji w oknie przeglądarki internetowej. Program został napisany przy użyciu języka JavaScript oraz frameworka ReactJS. ReactJS umożliwia tworzenie jednostronicowych aplikacji internetowych (ang. single page application). Cechą charakterystyczną tego typu aplikacji jest brak konieczności odświeżenia strony podczas wykonywania operacji przez użytkownika. Powodem wyboru ReactJS było wykorzystanie tego narzędzia przy rozwoju innych projektów realizowanych w ramach studiów. Aplikacja działa na porcie 3000.

**Zdjęcie 6** Formularz logowania do systemu

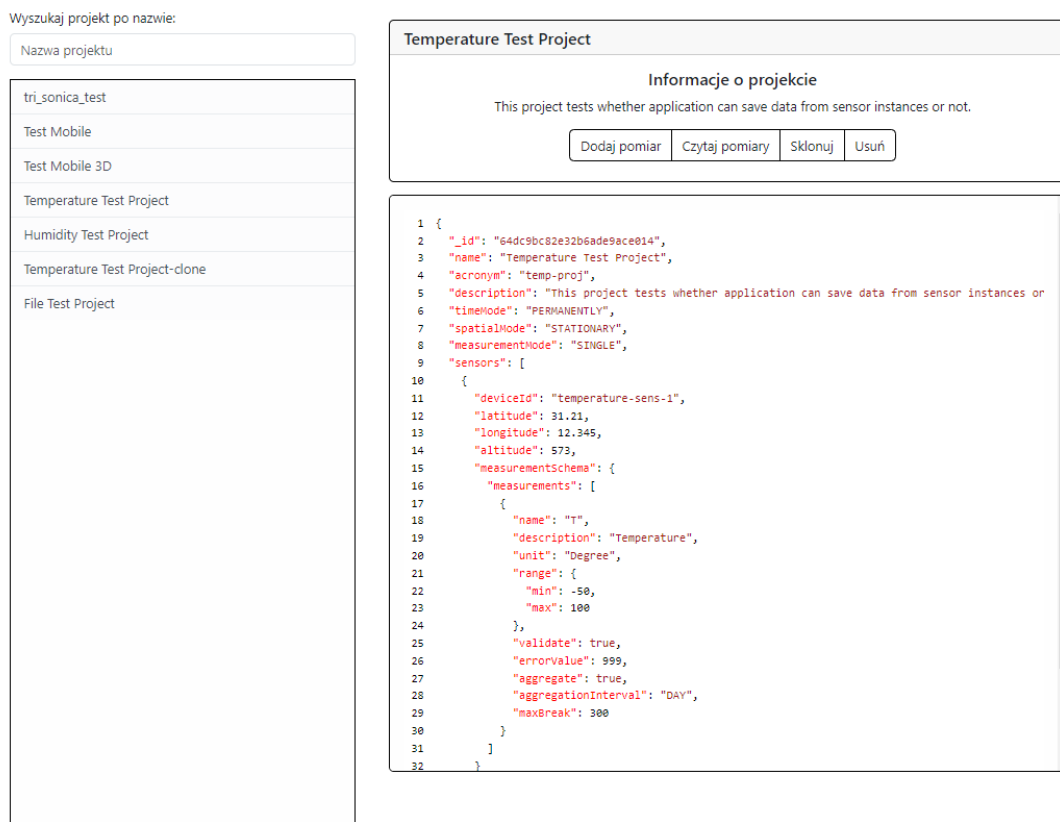
Zaprojektowany interfejs użytkownika oferuje cztery widoki oraz pięć okien modalnych. Dostępność widoków aplikacji jest zależna od roli posiadanej przez użytkownika. Po udanej próbie logowania do systemu w lokalnej pamięci okna przeglądarki zostaje zapisany żeton JWT, nazwa użytkownika oraz przypisane role. Jeżeli podczas logowania użytkownik wprowadzi nieprawidłowy login lub nieprawidłowe hasło, aplikacja poinformuje użytkownika za pomocą alertu o nieoczekiwanym błędzie. Analogiczny alert pojawi się, jeżeli przy procesowaniu autoryzacji po stronie serwera wystąpi błąd jak np. niedostępny API Gateway lub nieprawidłowa polityka CORS.

**Zdjęcie 7** Alert wyświetlany przez aplikację po wystąpieniu błędu

Po zalogowaniu do aplikacji użytkownik zostaje przekierowany do widoku listy projektów. W górnej części okna przeglądarki znajduje się pasek nawigacji. Widok utworzenia nowego projektu o nazwie *Dodaj projekt* jest widoczny tylko dla użytkowników posiadających rolę *Twórca Projektów*. Panel administratora umożliwiający zarządzanie użytkownikami aplikacji jest dostępny pod nazwą *Użytkownicy* tylko dla osoby o przypisanej roli *Administratora*. Opcje *Lista projektów* oraz *Wyloguj się* są widoczne dla wszystkich niezależnie od przypisanych ról. Użytkownik może korzystać tylko z widoków dostępnych w pasku nawigacji. Jeżeli nastąpi próba nieuprawnionego wywołania widoku, użytkownik zostanie natychmiastowo wylogowany z aplikacji i przekierowany do strony logowania.

**Zdjęcie 8** Pasek nawigacji

Widok listy projektów to strona umożliwiająca wykonanie operacji związanej z zarządzaniem projektami badawczymi. Lewą kolumnę stanowi lista zawierająca nazwy projektów badawczych. Lista zawiera **wszystkie** nazwy projektów badawczych utworzonych przez użytkowników i jest widoczna **niezależnie** od posiadanych ról i uprawnień. Pole tekstowe umieszczone nad listą służy do filtrowania wartości prezentowanych w komponencie na podstawie wpisanej frazy. Prawa kolumna składa się z karty informacji o projekcie oraz karty prezentującej strukturę projektu w formacie JSON. Karta informacji o projekcie znajduje się w górnej części widoku. W ciele karty pod adnotacją *Informacje o projekcie* można odczytać opis projektu, lub w przypadku braku opisu wyświetlona zostaje adnotacja *Akcje*.



All rights reserved © 2023 Akademia Górniczo-Hutnicza

**Zdjęcie 4** Widok listy projektów

Dostępność przycisków w karcie zależy od posiadanych przez użytkownika ról oraz uprawnień. Widok przedstawiony na Zdjęciu 6. ukazuje perspektywę widoczną przez użytkownika o przypisanej roli Administratora. Przycisk *Usuń* jest widoczny tylko dla użytkowników z przypisaną rolą Administratora. Przycisk *Sklonuj* jest widoczny dla posiadaczy roli Administratora lub Twórcy Projektów. Przyciski *Dodaj pomiar* oraz *Czytaj pomiary* wymagają posiadania roli Administratora lub Badacza. W przypadku roli Badacz udostępniono możliwość sterowania widocznością tych przycisków w zależności od projektu badawczego. Sterowanie dostępnością akcji odbywa się na widoku panelu administratora. Wobec tego osoba posiadająca jedynie rolę Badacz może w karcie informacji o projekcie widzieć obydwa przyciski, jeden z przycisków lub nie będzie miała dostępu do żadnej akcji. Naciśnięcie któregośkolwiek z przycisków skutkuje wyświetleniem okna modalnego umożliwiającego wykonanie pożądanej akcji. Użytkownik bez przypisanej roli zobaczy na widoku jedynie listę z nazwami projektów. Po wybraniu nazwy projektu z listy nie pojawi się

karta prezentująca strukturę projektu zaś karta informacji nie będzie zawierała ani opisu projektu ani przycisków.

Okno modalne przeznaczone dla akcji dodania pomiaru umożliwia przesłanie plików CSV. W pierwszym kroku należy wybrać czujnik, na podstawie którego ma się dokonać walidacja przesłanych pomiarów. Następnie należy upuścić plik w obrębie podpisanego pola lub po kliknięciu na pole można wyszukać plik w eksploratorze plików. Można upuścić i wyszukiwać tylko pliki typu CSV. Próba dodania pliku innego typu zakończy się niepowodzeniem bez poinformowania użytkownika. Po udanym dodaniu pliku do wysłania pojawia się ikona wraz z nazwą dodanego pliku. W przypadku chęci zmiany pliku można anulować operację lub usunąć wybrany plik poprzez kliknięcie w znak krzyżyka po prawej stronie pliku.


**Dodaj pomiar**

---


Identyfikator czujnika:

1

▼



Upuść plik lub wyszukaj na dysku

 TriSonica\_20210908.csv

×

---

Anuluj

Zapisz pomiar

**Zdjęcie 5** Okno modalne dodania pomiaru

Akcja czytania pomiarów wykonana bezpośrednio po dodaniu pomiarów nie pokaże żadnych rezultatów. Informacje prezentowane we wszystkich oknach modalnych są pobierane z serwera w momencie wyboru projektu z listy. Za każdym razem niezależnie od sposobu dostarczenia danych sugeruje się użytkownikowi ponowne wybranie projektu z list. Jeżeli po ponownym wyborze nazwy projektu badawczego karta wyników w oknie modalnym pozostanie pusta, należy sprawdzić logi aplikacji w celu weryfikacji czy podczas próby dodania pomiaru nie występuje błąd.

Karta rezultatów widoczna w oknie prezentuje maksymalnie trzy ostatnie pomiary zarejestrowane w bazie danych. Jeżeli baza danych zawiera mniej niż trzy pomiary, w karcie zostaną wypisane wszystkie pomiary zgromadzone w bazie. W przypadku dużej ilości parametrów zawartych w pomiarze użytkownik ma możliwość przewijania karty. Brak pomiarów wyświetlonych w karcie oznacza, że w ramach danego projektu badawczego nie zostały zarejestrowane żadne pomiary. Zamknięcie okna modalnego odbywa się poprzez naciśnięcie czerwonego przycisku na dole okna.

## Czytaj ostatnie pomiary

```
{
  "_id": "64e8e131d1cb309bea4bd562",
  "Data i czas": "2021-08-09T08:18:52.000Z",
  "Lat": 50.045934,
  "Lon": 19.935948,
  "Alt": 207.3,
  "Wiek danej": 970,
  "Ilosc satelitow": 8,
  "HDOP": 108,
  "S": 0.4,
  "S2D": 0.4,
  "D": 71,
  "DV": -9,
  "U": -0.38,
  "V": -0.13,
  "W": -0.06,
  "T": 19.5,
  "CS": 343.2,
  "RHtemp": 1.1,
  "RH": 1,
  "H": 32,
  "DP": 25.
```

Zamknij

### Zdjęcie 6 Okno modalne czytania pomiarów

Akcja usunięcia projektu badawczego otwiera okno modalne, którego celem jest zatwierdzenie operacji przez użytkownika. Zaimplementowany mechanizm ma na celu uniknięcie sytuacji, w której użytkownik przez pomyłkę usunie nieprawidłowy projekt. Operacja usunięcia projektu jest **nieodwracalna**. Użytkownik może zatwierdzić usunięcie projektu poprzez naciśnięcie przycisku *Tak* lub anulować operację poprzez naciśnięcie przycisku *Nie*.

## Usuń projekt

Czy na pewno chcesz usunąć projekt?

Nie

Tak

### Zdjęcie 7 Okno modalne operacji usunięcia projektu badawczego

Okno modalne przeznaczone dla akcji klonowania projektu udostępnia formularz analogiczny do formularza prezentowanego na widoku *Dodania projektu*. Dokładny opis pól formularza zostanie przedstawiony przy opisie wspomnianego widoku. Formularz w oknie modalnym zawiera pola uzupełnione informacjami zawartymi w projekcie, który podlega powieleniu. Użytkownik jest zobowiązany do zmiany nazwy projektu oraz akronimu. Wspomniane wartości muszą być unikalne w bazie danych. Sugestia zmiany nazwy projektu następuje poprzez dodanie sufiksu „-clone” do aktualnej nazwy projektu. W przypadku wystąpienia nieprawidłowości użytkownik zostaje poinformowany o błędach za

pośrednictwem alertu wyświetlonego w górnej części okna. Anulowanie operacji za pomocą przycisku *Anuluj* nie zapisuje zmian wprowadzonych przez użytkownika w formularzu. Zapisanie zmian w bazie danych odbywa się poprzez naciśnięcie przycisku *Zapisz*. Jeżeli formularz nie zawiera błędów, operacja zostanie zakończona zamknięciem okna modalnego.

### Klonuj projekt

Etap 1. Uzupełnij informacje o projekcie

Nazwa projektu:

Temperature Test Project-clone

Akronim:

możliwe jest wykorzystanie liter, cyfr oraz myślnika

maksymalnie 10 znaków

słowa mogą być rozdzielone za pomocą myślnika

nie można używać znaków regionalnych

cyfra nie może być pierwszym znakiem

temp-proj

Opis projektu:

This project tests whether application can save data from sensor instances or not.

Tryb pomiaru:

trwały

Rodzaj pomiaru:

stacjonarny

Typ pomiaru:

pojedynczy

Etap 2. Uzupełnij informacje o czujnikach

Czujnik #1

×

ID urządzenia pomiarowego:

temperature-sens-1

Położenie czujnika:

Długość geograficzna:

12,345

Szerokość geograficzna:

31,21

Wysokość n.p.m.:

573

Etap 3. Uzupełnij informacje o parametrach pomiarowych tego czujnika

Parametr #1

×

Nazwa pomiaru:

T

Jednostka pomiaru:

Degree

Opis pomiaru:

Temperature

Agregacja:

tak

Jednostka częstości pomiaru:

dzień

Maksymalna przerwa w pomiarach:

300

Walidacja:

tak

Minimalna wartość:

-50

Maksymalna wartość:

100

Wartość błędu:

999

Dodaj parametr

+

Anuluj

Zapisz

**Zdjęcie 8** Okno modalne służące do klonowania projektu

Operacja dodania nowego projektu badawczego została odseparowana od pozostałych akcji związanych z zarządzaniem projektami. Na widoku listy projektów wszystkie dostępne akcje są zależne od wybranego projektu badawczego. Proces stworzenia projektu jest operacją niezależną i ograniczoną wymogiem posiadania roli *Twórca projektów*. Ponadto autor pracy uznał, że wydzielenie formularza zwiększy jego czytelność. Formularz stworzenia projektu badawczego składa się z trzech części.

Etap 1. Uzupełnij informacje o projekcie

Nazwa projektu:

Nazwa projektu

Akronim:

- możliwe jest wykorzystanie liter, cyfr oraz myślnika
- maksymalnie 10 znaków
- słowa mogą być rozdzielone za pomocą myślnika
- nie można używać znaków regionalnych
- cyfra nie może być pierwszym znakiem

Nazwa projektu w serwisie www

Opis projektu:

Opis projektu

Tryb pomiaru: trwały

Rodzaj pomiaru: stacjonarny

Typ pomiaru: pojedynczy

### Zdjęcie 9 Formularz dodania projektu – część pierwsza

Etap pierwszy wymaga uzupełnienia ogólnych informacji o projekcie. Informacje wprowadzone w tej części służą do zdefiniowania projektu badawczego oraz dostosowania pozostałych fragmentów formularza do wybranych trybów przeprowadzania pomiarów. Nazwa projektu może zawierać tekst złożony z dowolnych znaków i jest elementem obowiązkowym. Akronim to skrótowe oznaczenie projektu wykorzystywane w adresach URL. Adresy URL wykorzystujące wartość przekazanego akronimu służą do dodawania i odczytywania pomiarów dla danego projektu badawczego z bazy danych. Obecność tego parametru w adresie URL wymusza spełnienie wymagań określonych w standardach. Ograniczenia nałożone na użytkownika zostały przedstawione w sekcji znajdującej się nad obowiązkowym polem tekstowym. Opis projektu jest polem opcjonalnym i umożliwia użytkownikowi przekazanie dodatkowych informacji o projekcie badawczym. Informacje wprowadzone w tym polu zostaną wyświetlone w karcie informacji o projekcie dostępnej na widoku listy projektów. Tryb pomiaru służy do określenia formy dostarczania pomiarów w ramach projektu badawczego. Opcja „offline” umożliwia dodawanie pomiarów **tylko i wyłącznie** za pośrednictwem plików CSV. Pozostałe opcje pozwalają na przekazanie pomiarów zarówno w formie plików jak również za pośrednictwem interfejsu HTTP. Rodzaj pomiaru służy do zdefiniowania trybu mobilności czujników w ramach projektu badawczego. Opcja „stacjonarny” określa czujnik zlokalizowany w jednym punkcie przez cały okres gromadzenia pomiarów, wobec czego użytkownik może wprowadzić parametry położenia czujnika w drugiej części formularza. Wybór opcji „mobilny 2D” lub „mobilny 3D” ukrywa możliwość wprowadzenia informacji o położeniu czujnika, ponieważ aktualne położenie czujnika jest przekazywane w formie parametrów pomiaru badawczego. Typ pomiaru pozwala zdefiniować ilość czujników wykorzystanych w projekcie. Opcja „pojedynczy” oznacza, że w ramach projektu badawczego wykorzystany jest jeden czujnik. Wobec tego w formularzu zostaje ukryta opcja dodania czujnika. Opcja „sieć” umożliwia dodanie nieograniczonej liczby czujników.



Etap 2. Uzupełnij informacje o czujnikach

Czujnik #1

ID urządzenia pomiarowego:

Identyfikator urządzenia pomiarowego

Położenie czujnika:

Długość geograficzna:

Długość geograficzna

Szerokość geograficzna:

Szerokość geograficzna

Wysokość n.p.m.:

Wysokość

Etap 3. Uzupełnij informacje o parametrach pomiarowych tego czujnika

Parametr #1

Nazwa pomiaru:

Nazwa pomiaru

Jednostka pomiaru:

Jednostka pomiaru

Opis pomiaru:

Opis pomiaru

Agregacja:

tak

Jednostka częstości pomiaru:

minuta

Maksymalna przerwa w pomiarach:

sekundy

Walidacja:

tak

Minimalna wartość:

Minimalna wartość

Maksymalna wartość:

Maksymalna wartość

Wartość błędu:

Wartość błędu

Dodaj parametr

**Zdjęcie 10** Formularz dodania projektu – część druga

Etap drugi wymaga uzupełnienia informacji o czujniku pomiarowym. Identyfikator urządzenia pomiarowego to obowiązkowe pole tekstowe, którego celem jest umożliwienie przekazania wartości pozwalającej na rozróżnienie czujnika względem pozostałych urządzeń pomiarowych wykorzystanych w ramach projektu badawczego. Sekcja o nazwie *Położenie czujnika* jest widoczna w zależności od opcji wybranej w polu *Rodzaj pomiaru*. Wszystkie pola w ramach sekcji są opcjonalne. Długość geograficzna to pole liczbowe, które akceptuje wartości z przedziału  $[-180^{\circ}, 180^{\circ}]$ . Szerokość geograficzna to pole liczbowe, które akceptuje wartości z przedziału  $[-90^{\circ}, 90^{\circ}]$ . Wysokość jest polem liczbowym, które nie ma żadnych ograniczeń.

Etap trzeci wymaga uzupełnienia informacji o pomiarach rejestrowanych przez czujnik. Nazwa pomiaru jest obowiązkowym polem tekstowym, które definiuje nazwę parametru obecną w obiekcie JSON. Jednostka pomiaru jest wymaganym polem tekstowym. Wartość przekazana w tym polu nie jest brana pod uwagę w procesie walidacji pomiaru. Opis pomiaru jest polem opcjonalnym, które może służyć do przekazania pełnej nazwy pomiaru. Agregacja jest powiązana z polami *Jednostka częstości pomiaru* oraz *Maksymalna przerwa w pomiarach*. Jednostka częstości pomiaru określa na podstawie jakiego przedziału czasowego należy agregować gromadzone wartości. Maksymalna przerwa w pomiarach pozwala zdefiniować najdłuższą dopuszczalną przerwę w przesłaniu dwóch następujących po sobie pomiarów. Walidacja jest powiązana z polami *Minimalna wartość* oraz *Maksymalna wartość*. Walidacja parametru działa na podstawie przedziału wartości podanego w wymienionych polach. Wartość liczbową przekazaną w polu *Maksymalna wartość* musi być większa niż wartość liczbową w polu *Minimalna wartość*. Wartość błędu to pole liczbowe, które umożliwia zdefiniowanie kodu błędu wysyłanego przez czujnik w momencie wadliwego działania urządzenia pomiarowego.

Operacja zapisu projektu badawczego do bazy danych odbywa się poprzez naciśnięcie przycisku *Zapisz projekt*. W pierwszym kroku następuje walidacja pól formularza. Wykrycie błędu skutkuje wyświetleniem alertu w górnej części ekranu. Alert zawiera wszystkie odnotowane błędy. Należy pamiętać, że każdy projekt badawczy musi posiadać **unikalną nazwę oraz akronim**. Jeżeli wartość przekazana w tych polach występuje już w bazie danych,

użytkownik również zostanie poinformowany o błędzie. Udany proces dodania projektu zostaje zakończony przekierowaniem użytkownika do widoku listy projektów.

Część operacji możliwych do realizacji w ramach systemu do akwizycji danych zostało obwarowanych wymogiem posiadania roli Administratora. Zdecydowana większość z nich została wydzielona do dedykowanego widoku panelu administratora. Administrator systemu jest odpowiedzialny za tworzenie kont oraz zarządzanie rolami i uprawnieniami użytkowników systemu. Widok został podzielony na dwie części.

**Dodaj nowego użytkownika**

**Login:**

  
**Email:**

**Informacje o użytkowniku:**

**Uczestnictwo w projektach:**

tri_sonica_test	<input type="checkbox"/>
Test Mobile	<input type="checkbox"/>
Test Mobile 3D	<input type="checkbox"/>
Temperature Test Project	<input type="checkbox"/>

**Hasło:**

**Powtórz hasło:**

**Dodaj użytkownika**

**Zdjęcie 11** Formularz dodania użytkownika

W prawej części okna przeglądarki został umieszczony formularz dodania nowego użytkownika. Formularz umożliwia utworzenie nowego konta w systemie. Login jest równoznaczny z nazwą użytkownika i jest polem obowiązkowym. Email będzie w przyszłości wykorzystany do wysyłania notyfikacji m.in. po utworzeniu przez użytkownika nowego projektu i jest polem obowiązkowym. Informacje o użytkowniku to pole opcjonalne, które służy do przekazania dodatkowych informacji na temat użytkownika systemu. Uczestnictwo w projektach służy do przypisania projektów badawczych, w które dany użytkownik jest zaangażowany. Lista projektów zawiera wszystkie utworzone projekty badawcze dostępne w bazie danych. W celu ułatwienia przeszukiwania listy dodano pole tekstowe umożliwiające filtrowanie projektów po nazwie. Jeżeli użytkownik jest zaangażowany w projekt, należy zaznaczyć *checkbox* w wierszu zawierającym nazwę projektu badawczego. Ostatnim etapem wypełnienia formularza jest wprowadzenie hasła do konta. Pole *Powtórz hasło* jest oznaczone na czerwono dopóki tekst nie będzie zgodny z wartością w polu *Hasło*. Zapis danych w bazie odbywa się po naciśnięciu przycisku *Dodaj użytkownika*. Jeżeli operacja zakończy się

sukcesem, widok zostanie odświeżony i nazwa nowoutworzonego użytkownika będzie widoczna na liście użytkowników.

#### Zarządzaj uprawnieniami użytkowników

Admin [Administrator]

test

User

**mateusz barnacki**

Role:

Badacz

Twórca projektów

Widoczność akcji dla roli Badacz:

#	Nazwa projektu	Dodanie pomiaru	Czytanie pomiaru
1	File Test Project		x
2	tri_sonica_test	x	x
3	Test Mobile 3D	x	
4	Temperature Test Project	x	x

Edytuj uprawnienia

Edytuj akcje

**Zdjęcie 12** Panel zarządzania rolami i uprawnieniami

W lewej kolumnie widoku umieszczono listę zawierającą spis wszystkich użytkowników systemu. Każdy element listy jest obiektem rozwijalnym. Po rozwinięciu obiektu można zobaczyć aktualnie przypisane role użytkownika oraz tabelę zawierającą spis projektów, w które użytkownik jest zaangażowany. Po naciśnięciu przycisku *Edytuj uprawnienia* zostaje wyświetlone okno modalne, w którym można aktualizować role użytkownika oraz listę projektów, w które użytkownik jest zaangażowany. Zatwierdzenie wprowadzonych zmian skutkuje odświeżeniem listy użytkowników. Operacja w każdym momencie może zostać anulowana. Przycisk *Edytuj akcje* jest widoczny tylko dla użytkownika, który ma przypisaną rolę *Badacz*. Po naciśnięciu przycisku zostaje wyświetlone okno modalne, które zawiera tabelę *Widoczność akcji dla roli Badacz* w wersji modyfikowalnej. Po zatwierdzeniu zmian wprowadzonych w tabeli następuje odświeżenie widoku. Opisana operacja może zostać anulowana. Użytkownik, który ma przypisaną rolę *Administrator* jest oznaczony na liście specjalnym przyrostkiem *[Administrator]*. Rozwinięcie karty tego użytkownika nie daje możliwości edycji żadnych uprawnień, ponieważ z definicji administrator systemu może realizować wszystkie operacje dostępne w ramach systemu do akwizycji danych.

## **6. Podsumowanie**

## Bibliografia

- [1] Atlassian, *User Stories / Examples and Template*, <https://www.atlassian.com/agile/project-management/user-stories>, dostęp w dniu 31.08.2023 r.
- [2] Martin Fowler, *DomainDrivenDesign*, <https://martinfowler.com/bliki/DomainDrivenDesign.html>, dostęp w dniu 31.08.2023 r.
- [3] JSON Schema Community, *JSON Schema*, <https://json-schema.org/>, dostęp w dniu 30.08.2023 r.
- [4] Gradle Inc., *Gradle Build Tool*, <https://gradle.org/>, dostęp w dniu 29.08.2023 r.
- [5] npm Inc., *npm*, <https://www.npmjs.com/>, dostęp w dniu 29.08.2023 r.
- [6] Docker Inc., *Docker Docs*, <https://docs.docker.com/>, dostęp w dniu 18.08.2023 r.
- [7] Chris Richardson, *Service Registry pattern*, <https://microservices.io/patterns/service-registry.html>, dostęp w dniu 24.08.2023 r.
- [8] spring by VMware Tanzu, *Spring Cloud Gateway*, <https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/#gateway-starter>, dostęp w dniu 23.08.2023 r.
- [9] spring by VMware Tanzu, *Spring / Reactive*, <https://spring.io/reactive>, dostęp w dniu 23.08.2023 r.
- [10] auth0 by Okta, *JSON Web Tokens Introduction* <https://jwt.io/introduction>, dostęp w dniu 21.08.2023 r.