



**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
3. Elementy charakterystyczne dla architektury mikrousługowej
4. Zapewnienie jakości w projekcie IT
5. Architektura aplikacji
6. Technologie wykorzystane w projekcie
7. Dokumentacja komponentów
8. Podsumowanie

## **1. Wstęp**

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

### **3. Elementy charakterystyczne dla architektury mikrousługowej**

#### **4. Zapewnienie jakości w projekcie IT**

## **5. Architektura aplikacji**

## **6. Technologie wykorzystane w projekcie**



## 7. Dokumentacja komponentów

### 7.1. 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.

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, lokalizację 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>.

## 7.2. 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 w 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 wykonać poniższą komendę.

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

*Listing 2 Zbudowanie oraz uruchomienie aplikacji*

Przedstawiona komenda umożliwi wygenerowanie obrazów Dockera. Następnie na podstawie obrazów tworzone są kontenery zawierające poszczególne komponenty aplikacji. Opcja `-d` umożliwi 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>.

### 7.3. 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 i 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*.

### 7.4. Obsługa 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 przejrzania logów zdarzeń w każdym komponencie 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 staranne zaimplementowanie logowania oraz archiwizowania 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 wykorzystać poniższą komendę.

```
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-compose.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 1 Fragment wyniku działania polecenia docker compose ps dla serwisu do akwizycji danych*

Autor pracy zdecydował się usunąć ze Zdjęcia 1 kolumny COMMAND, SERVICE oraz PORTS. Widoczne na zdjęciu kolumny oznaczają 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 wywołań metod 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.



## 7.5. Połączenie z bazą MongoDB

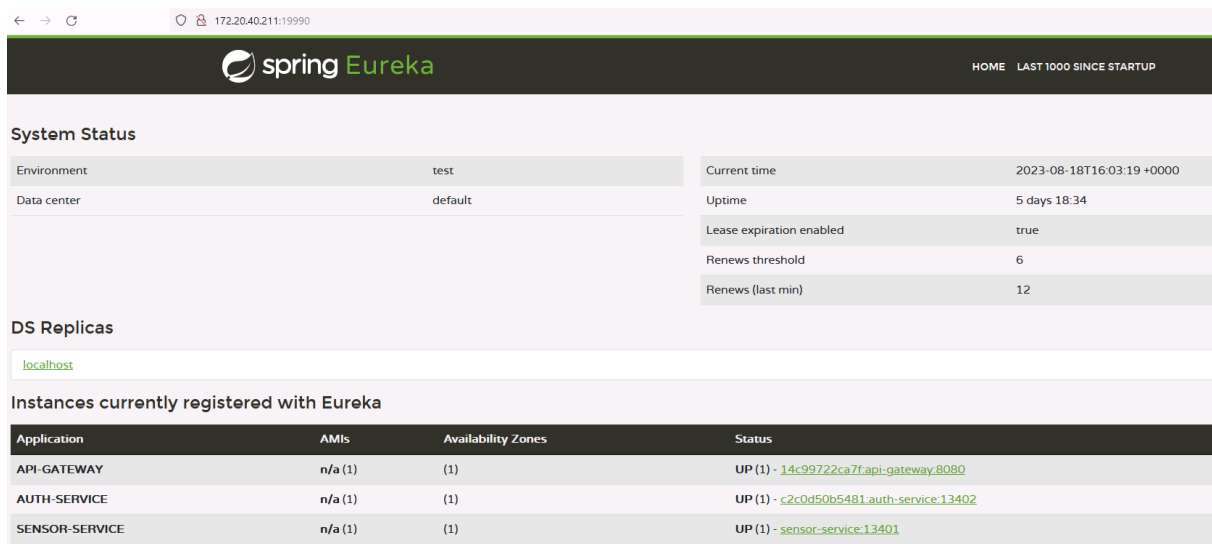
Baza danych MongoDB jest wykorzystywana do gromadzenia danych dotyczących projektów badawczych oraz pomiarów przesyłanych przez czujniki lub przekazywanych za pośrednictwem klienta aplikacji w formie plików CSV. Stworzenie bazy danych oraz kolekcji odbywa się przy pierwszym odwołaniu do zasobu z poziomu 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 umożliwia uruchomienie wspomnianego narzędzia 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

## 7.6. Service Registry

W ramach systemu została zaimplementowana usługa Service Discovery. Sercem usługi jest aplikacja o nazwie service-registry. Aplikacja zawiera w sobie serwer o nazwie Eureka. Główną funkcją serwera jest ustanowienie komunikacji z instancjami pozostałych aplikacji, które należy określić terminem klientów usługi.



The screenshot displays the Spring Eureka web interface. At the top, there's a navigation bar with the 'spring Eureka' logo and a 'HOME' link. Below this, the 'System Status' section provides details about the environment (test), data center (default), current time (2023-08-18T16:03:19 +0000), uptime (5 days 18:34), lease expiration (enabled), renewals threshold (6), and renewals (last min) (12). The 'DS Replicas' section shows 'localhost'. The 'Instances currently registered with Eureka' section lists three applications: API-GATEWAY, AUTH-SERVICE, and SENSOR-SERVICE, each with its status (UP) and instance details.

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 14c99722ca7fapi-gateway:8080
AUTH-SERVICE	n/a (1)	(1)	UP (1) - c2c0d50b5481auth-service:13402
SENSOR-SERVICE	n/a (1)	(1)	UP (1) - sensor-service:13401

Zdjęcie 2 Graficzny interfejs użytkownika usługi Service Discovery

Na Zdjęciu 2 widać wygenerowany panel administratora systemu. W ramach tego panelu 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 składających się na system. Ponadto interfejs użytkownika zawiera informacje o ilości dostępnej pamięci, stopniu wykorzystania pamięci, czasie działania aplikacji, adresie IP oraz statusie działania serwera Eureka. Dodatkowym atutem

wykorzystania usługi Service Discovery jest możliwość zastosowania funkcjonalności o nazwie *load balancing*. Load balancing służy do równomiernego rozłożenia obciążenia sieciowego pomiędzy wszystkie instancje danego serwisu.

## 7.7. API Gateway

## 7.8. 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).

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 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*.

#### Metody DELETE

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

## 7.9. Serwis czujników

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.

## 8. Podsumowanie

## **Bibliografia**