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: Mateusz Barnacki
Kierunek studiów: Informatyka Stosowana
Opiekun pracy: 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

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 przejrzyście 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 niezmienne 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, 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*.

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 implementacje 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 wykonać poniższą komendę.

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

Listing 2 Zbudowanie oraz uruchomienie aplikacji

Przedstawiona komenda umożliwia wygenerowanie obrazów Dockera. Następnie na podstawie obrazów tworzone są kontenery zawierające poszczególne komponenty aplikacji. Opcja *-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.

```
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.

```
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*.

```
curl -d
'{"username": "Admin",
  "email": "admin@agh.edu.pl",
  "description": "Admin user",
  "password": "admin",
  "roles": ["ADMIN"],
  "projects": []}'
-H "Content-Type: application/json"
-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 przejrzenia 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-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 1 Fragment wyniku działania polecenia docker compose ps dla serwisu do akwizycji danych

Autor pracy zdecydował się usunąć ze Zdjecia 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 narzędzie 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
```

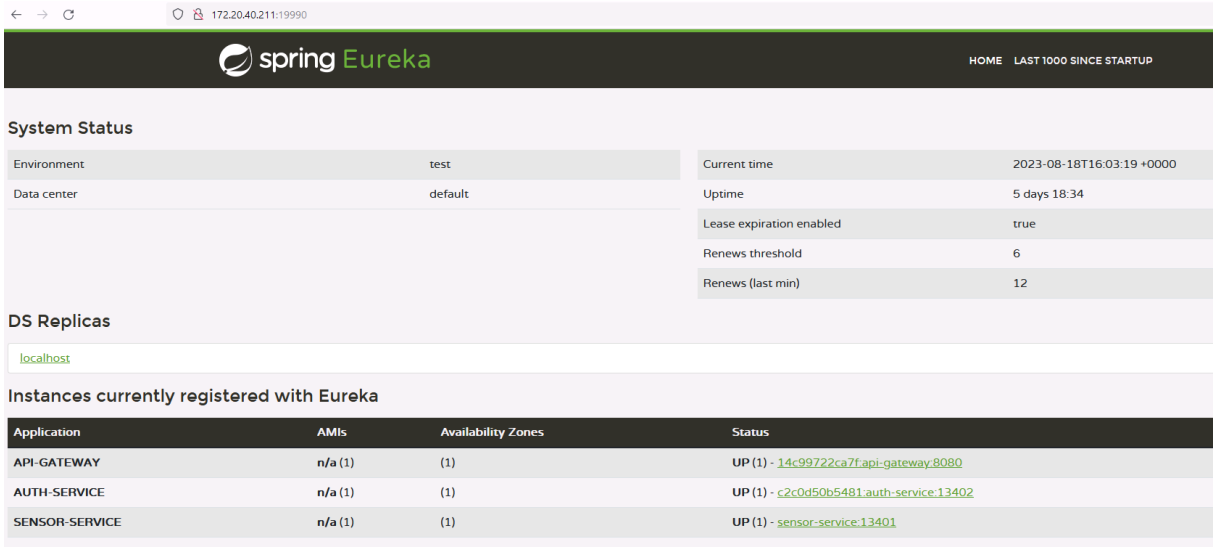
Listing 8 Uruchomienie konsoli Mongo Shell w kontekście bazy danych weather

7.6. Service Registry

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 [1], 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 rozprowadzają 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 zapytania 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.



Zdjęcie 2 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.

7.7. 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 [2]. Paradygmat programowania reaktywnego pozwala programistą na budowanie nieblokujących, asynchronicznych aplikacji [3], 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.

Uprawnienie *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

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). 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 [4], 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 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. Wartym 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 użytkownia, 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.

7.9. 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 ciążącym 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.

8. Podsumowanie

Bibliografia

[1] Chris Richardson, *Service Registry pattern*, https://microservices.io/patterns/service-registry.html, dostęp w dniu 24.08.2023 r.

[2] 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.

[3] spring by Vmware Tanzu, *Spring | Reactive*, https://spring.io/reactive, dostęp w dniu 23.08.2023 r.

[4] auth0 by Okta, *JSON Web Tokens Introduction* https://jwt.io/introduction, dostęp w dniu 21.08.2023 r.