JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 27 (2023) 112–120
Received: 17 October 2022
Accepted: 26 October 2022

Analysis of the Spring Boot and Spring Cloud in developing Java cloud applications

Analiza zastosowania Spring Boot i Spring Cloud w tworzeniu aplikacji chmurowych w Javie

Mateusz Kozak*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The paper is of a review character. It analyzes the possibilities of using the Spring Boot programming framework with the Spring Cloud extension to create cloud applications in Java. The article presents the concepts that have to be dealt with when implementing the application in cloud environments and the technologies that were used in the applications, along with the justification of the choice. Scalability tests were carried out on two applications - non-scalable with the use of Spring Boot and scalable with the use of Spring Boot and Spring Cloud. The research carried out in the article showed how to implement a scalable application using Spring Boot and Spring Cloud.

Keywords: analysis; Spring Boot; Spring Cloud; cloud applications

Streszczenie
Artykuł ma charakter przeglądowy. Przeanalizowano w nim możliwości zastosowania szkieletu programistycznego Spring Boot z rozszerzeniem Spring Cloud do tworzenia aplikacji chmurowych w Javie. W artykule przedstawiono pojęcia z jakimi trzeba się mierzyć wdrażając aplikację w środowiskach chmurowych oraz technologie jakie zastosowano w aplikacjach wraz z uzasadnieniem wyboru. Przeprowadzono badania pod kątem skalowalności na dwóch aplikacjach – nieskalowalnej z wykorzystaniem Spring Boot i skalowalnej z wykorzystaniem Spring Boot i Spring Cloud. Badania przeprowadzone w artykule wskazały sposób realizacji skalowalnej aplikacji z zastosowaniem Spring Boot i Spring Cloud.

Słowa kluczowe: analiza; Spring Boot; Spring Cloud; aplikacje chmurowe

*Corresponding author
Email address: **mateusz.kozak3@pollub.edu.pl** (M. Kozak)

1. Wstęp

Popularność mikroserwisów w ostatnich latach rośnie, mogą one rozwiązać wiele aktualnych wyzwań w sektorze IT, takich jak zwiększenie wydajności, skalowalność aplikacji i przeprowadzanie testów. W dzisiejszych czasach wiele firm wdrażając swoje aplikacji zdecydowało się na wykorzystanie architektury mikrousług. Najbardziej rozpoznawalne firmy to Coca-Cola, Spotify, Netflix i Amazon [1]. Powód dlaczego architektura mikrousług jest używana to przede wszystkim skalowalność, elastyczność, zdolność adaptacji (wykorzystywanie różnych technologii) oraz możliwości wdrażania bez ingerowania w pozostałe aplikacje [2].

Na rynku jest wiele technologii ułatwiających pracę z mikroserwisami. Najpopularniejsze z nich to Docker, Kubernetes oraz REST [3]. Java jest jednym z najpopularniejszych języków wykorzystywanych w mikroserwisach [4] ze względu na czytelność kodu oraz składnię adnotacji (ang. annotation syntax). Java posiada wiele narzędzi ułatwiających pracę z mikroserwisami, w tym Spring Boot [5].

Wdrażanie aplikacji opartych na mikroserwisach w chmurze udostępnia aplikacje na pewne problemy, takie jak skalowalność, wykrywanie usług czy zewnętrzna konfiguracja. Spring Cloud zapewnia zestaw narzędzi do szybkiego tworzenia aplikacji w chmurze oferując rozwiązania problemów występujących w środowiskach rozproszonych [6].

Celem artykułu jest analiza możliwości zastosowania Spring Boot i Spring Cloud w tworzeniu aplikacji chmurowych w Javie. W pracy omówiono pojęcia, występujące w środowiskach chmurowych. Przedstawiono zarys szablonowej aplikacji, którą można wdrożyć do platformy chmurowej, takiej jak Kubernetes. Przeprowadzono badania pod kątem skalowalności na dwóch aplikacjach, które demonstrują możliwości oferowane przez narzędzia Spring Boot i Spring Cloud.

2. Wprowadzenie

2.1. Mikroserwisy

Mikroserwis [7] (inaczej architektura mikrousług) umożliwia szybkie i niezawodne rozwijanie złożonych aplikacji. Wraz z rozwojem nieskalowalnych aplikacji trudniej jest wdrażać kolejne programy. Mikroserwisy oddzielają poszczególne fragmenty aplikacji. Pozwalają się skupić na konkretnym podzbiorze całej aplikacji, można wdrażać nowe funkcjonalności niezależnie od już istniejących. Takie rozwiązanie pozwala również na używanie różnych technologii.

Do najbardziej popularnych technologii tworzenia aplikacji z wykorzystaniem architektury mikrousług należą języki programowania Node.js, Java i Python [3], a także narzędzia API Fortress, Postman i RabbitMQ [4].

2.2. Spring Boot

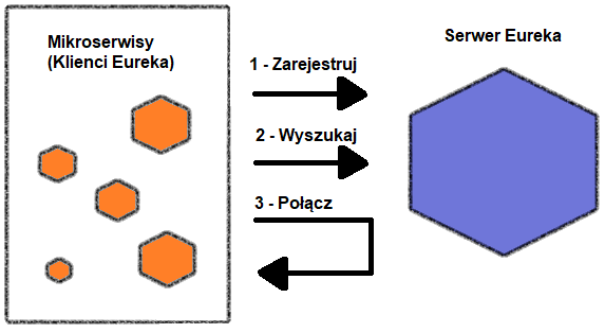
Spring Boot [8] jest rozszerzeniem szkieletu programistycznego Spring. Pozwala na szybkie opracowanie aplikacji i jest najpopularniejszym szkieletem języka Java. Dostarcza klasy i interfejsy potrzebne do opracowania aplikacji, które zapewniają bezpieczeństwo, integrację, łączenie z wieloma bazami, itp.

2.3. Spring Cloud

Budując serwisy w Spring wykorzystywane jest rozszerzenie Spring Cloud [9], które dostarcza wiele narzędzi i pomaga w pracy przy tworzeniu złożonych aplikacji w środowiskach systemów rozproszonych.

Service Discovery

Service Discovery [10] (w przypadku Spring Cloud jest to serwer Eureka) to proces automatycznego wykrywania urządzeń i usług w sieci. Mikrousługi są klientami serwera Eureka. Jest scentralizowanym miejscem, w którym wszyscy klienci rejestrują się na serwerze. Kiedy nastąpi rejestracja klienta, serwer zna dokładne informacje gdzie jest uruchomiona usługa, tj. host i port. Gdy mikroserwisy chcą komunikować się z innym serwisem muszą wyszukać na serwerze te informacje i w ten sposób mogą się ze sobą łączyć. Serwer kojarzy wszystkie aplikacje klienckie działające na każdym porcie i adresie IP (Rys. 1).



Rysunek 1: Rejestracja oraz komunikacja serwisów z serwerem Eureka [11].

Load Balancer

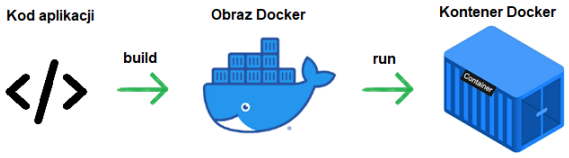
Klienci wysyłają żądania poprzez publiczny Internet. Load Balancer [12] służy jako główny punkt wejścia. Zwykle podczas tworzenia aplikacji klient nie komunikuje się bezpośrednio z usługami, ponieważ te serwisy powinny znajdować się w sieci prywatnej. Load Balancer jest odpowiedzialny za wykonanie przekierowania (ang. routing) do odpowiedniej mikrousługi w oparciu o ścieżkę.

Load Balancer jest konieczny podczas wdrażania w środowisku produkcyjnym. Klient nie powinien uzyskiwać dostępu do jednej instancji za pośrednictwem adresu IP. Im większy ruch jest w aplikacji tym trudniej sobie z tym poradzić.

2.4. Docker

Docker [13] jest platformą do budowania, uruchamiania i dostarczania (ang. shipping) aplikacji. Deweloperzy oprogramowania mogą z łatwością budować i opracowywać aplikacje uruchamiane w kontenerach. Kontener jest spakowaną instancją aplikacji. Lokalny rozwój (ang. local development) aplikacji jest taki sam w dowolnym stosie technologicznym (ang. environment). Docker jest wykorzystywany również w obiegu ciągłej integracji i ciągłego dostarczania (ang. continuous integration and continuous delivery/deployment - CI/CD) [14], tj. dostarczaniu aplikacji poprzez wprowadzanie automatyzacji na etapach opracowywania aplikacji.

Przy tworzeniu oprogramowania, kod aplikacji może być napisany w dowolnym języku z wykorzystaniem dowolnej technologii. Aplikacja jest budowana jako obrazy Docker (ang. Docker Images), z których może zostać uruchomiony kontener. Kontener jest na swój sposób szablonem do uruchamiania aplikacji. Z utworzonego obrazu można uruchomić wiele kontenerów, korzystających z różnych silników (Rys. 2).



Rysunek 2: Budowanie obrazu Docker oraz uruchomienie kontenera Docker [13].

2.5. Kubernetes

Kubernetes [15] jest otwarto-źródłowym narzędziem, napisanym w języku Go [16], pochodzącym od narzędzi Borg [17] i Omega [18]. Kuberneter jest zarządcą aplikacji, jego głównym zadaniem jest wdrażanie i zarządzanie aplikacjami (kontenerami).

3. Przegląd literatury

Większość istniejących pozycji opisuje wzorce, podejście oraz architekturę rozwiązań chmurowych. Badania dotyczą procesu tworzenia aplikacji. Istnieje niewiele badań i testów aplikacji chmurowych z wykorzystaniem architektury mikroserwisów.

W artykule [19] przedstawiono zastosowanie wstępnego zestawu wymagań, które mogą być przydatne przy wyborze wzorca architektury w celu wsparcia badań nad powtarzalnymi mikrousługami. Autor wskazuje na brak powtarzalnych badań dotyczących projektowania, rozwoju i oceny aplikacji mikrousług.

Autor książki [20] opisuje jak działa Spring Boot, zapoznaje z technologią Spring – przedstawia wzorce natywne dla chmury, programowanie reaktywne i aplikacje.

Autor książki [21] przedstawia koncepcje, architektury i scenariusze mikrousług pod względem technologicznym oraz pokazuje, jak je wdrażać za pomocą tech-

nologii, takich jak Docker, Java, Spring Boot i Spring Cloud.

Wydanie drugie pozycji [22] opisuje proces tworzenia aplikacji opartych na mikrousługach przy użyciu Java i Spring. Omawia proces tworzenia podstawowych usług, monitorowania aplikacji, opisuje sposoby refaktoryzacji za pomocą narzędzi Spring i wprowadza Spring Cloud Gateway do zarządzania API. Opisuje proces wdrażania aplikacje z użyciem technologii Spring Cloud z AWS i Kubernetes.

Autor książki [23] przedstawia jak za pomocą Spring Boot tworzyć aplikacje korporacyjne, internetowe i mikrousługowe oparte na Javie. W treści znajduje się między innymi to jak Spring Boot zwiększa produktywność dewelopera i jak działa autokonfiguracja Spring.

Badania w artykule [24] oceniają czy Spring promuje dobre praktyki definiowania architektury. Analiza wyników wykazała, że zdecydowana (70%) większość projektów łączy wszystkie funkcje definicji architektury Spring. Autorzy artykułu wskazali na niewystarczające dokumentacje dobrych praktyk oraz podsumowali zalecenia dotyczące pomocy programistom.

W artykule [25] przedstawiono podejście do opracowywania aplikacji z wykorzystaniem środowiska do wdrażania mikroserwisów skalowalnych. Zaletą korzystania z tego podejścia jest to, że systemy mogą być ciągle rozwijane.

Autorzy książki [26] opisują proces wdrażania systemów opartych na architekturze mikrousług. Wprowadzają w obszar projektowania mikrousług pozwalając zrozumieć jak rozwiązywać problemy napotkane podczas programowania.

Celem badań w artykule [27] jest opracowanie aplikacji internetowej zachowując architekturę mikroserwisów Spring Boot. Artykuł przedstawia proponowane rozwiązanie jako zaletę, ponieważ można dodać więcej mikrousług bez wpływu na inne.

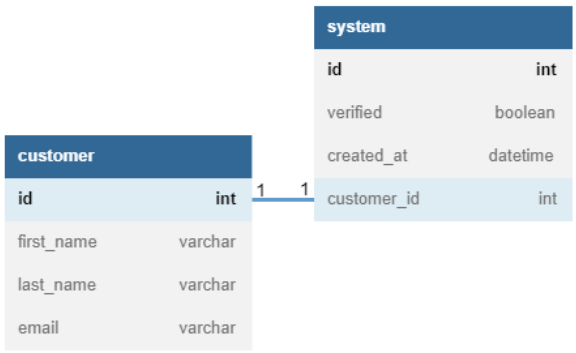
W niniejszej pracy podjęto tematykę tworzenia aplikacji chmurowych w Javie z wykorzystaniem narzędzi dostarczanych przez Spring Boot i Spring Cloud. Jak wskazują autorzy artykułu [24] sam Spring nie zawiera wielu dobrze opracowanych praktyk oraz dokumentacji. W pracy podjęto się zadania poszerzenia wiedzy na temat Spring Boot i Spring Cloud w tworzeniu aplikacji chmurowych, aby ułatwić zrozumienie związanych z tym pojęć i możliwości.

4. Aplikacje testowe

Do celów badawczych zostały opracowane dwie aplikacje. Z założenia są to proste aplikacje typu REST [28], które udostępniają dwa punkty końcowe (ang. endpoints). Pierwszy serwis *customer* udostępnia punkt końcowy *api/v1/customers*, gdzie można zarejestrować użytkownika poprzez wysłanie żądania metodą POST z danymi użytkownika. Drugi serwis *system* udostępnia punkt końcowy *api/v1/system/{customerId}*, który obsługuje żądanie przesłane metodą GET służący do zasymulowania weryfikacji użytkownika w systemie na podstawie podanego identyfikatora klienta. Na potrzeby badań serwis *system* zwraca pomyślną weryfikację użytkownika.

Aplikacje zostały wykorzystane w celu zbadania wydajności (liczby żądań na sekundę) oraz możliwości skalowalności aplikacji z wykorzystaniem narzędzi dostarczanych przez Spring Boot i Spring Cloud.

Diagram prezentuje strukturę bazy danych, wykorzystaną w aplikacjach i zawiera podstawowe informacje do zasymulowania rejestracji oraz weryfikacji użytkownika.



Rysunek 3: Diagram ERD.

W projekcie aplikacji testowych wykorzystano:

- Maven do zarządzania projektem;
- środowisko IntelliJ IDEA 2020.3 Ultimate Edition,
- SDK Java 14;
- bazę danych PostgreSQL wraz z graficznym interfejsem PGAdmin.

Spring wykorzystuje zewnętrzną konfigurację do definiowania właściwości aplikacji. Istnieje kilka sposobów konfiguracji, takich jak plik właściwości (application.properties), plik YAML (application.yml), zmienne środowiskowe lub argumenty wiersza poleceń. W projekcie zastosowano plik YAML ze względu na czytelną strukturę [29].

W aplikacjach zastosowano narzędzie Docker, w celu uruchomienia poszczególnych modułów jako oddzielne kontenery Docker. Cała konfiguracja usług znajduje się w jednym pliku o nazwie *docker-compose.yml*. Uruchomienie komendy *docker compose up -d* uruchomi aplikację jako izolowane środowisko, czyli kontenery [30].

Przygotowane zostały dwie aplikacje testowe:

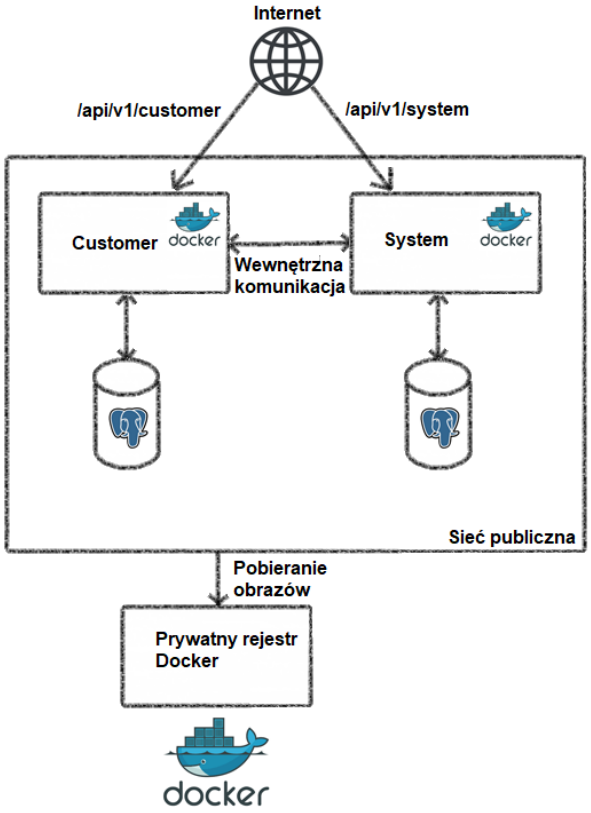
- aplikacja nieskalowana w Spring Boot;
- aplikacja skalowalna w Spring Boot i Spring Cloud.

Aplikacja nieskalowalna udostępnia jedynie dwa punkty końcowe służące do zasymulowania rejestracji użytkownika oraz weryfikacji w systemie. Zapisuje w bazie dane użytkownika oraz datę i rezultat weryfikacji. Aplikacja skalowalna pozwala na tworzenie wielu instancji serwisów oraz przygotowuje do wprowadzenia w środowisko chmurowe.

4.1. Aplikacja nieskalowalna

Pierwsza aplikacja jest podstawową aplikacją bazującą na architekturze mikroserwisów. Składa się z tylko jednej instancji każdego mikroserwisu. W projekcie znajduje się moduł nadrzędny oraz dwa moduły pod-

rzędne, czyli mikroserwisy. Schemat aplikacji nieskalowalnej przedstawiono na Rysunku 4.



Rysunek 4: Schemat aplikacji nieskalowalnej [31].

Aplikacja nieskalowalnej posiada podstawową funkcjonalność wymaganą do przeprowadzenia testów obciążeniowych. Udostępnia punkt końcowy, który umożliwia rejestrację użytkownika. Serwis *customer* komunikuje się z drugim serwisem *system* w celu weryfikacji użytkownika. Dane użytkownika oraz weryfikacji są zapisywane w bazie danych.

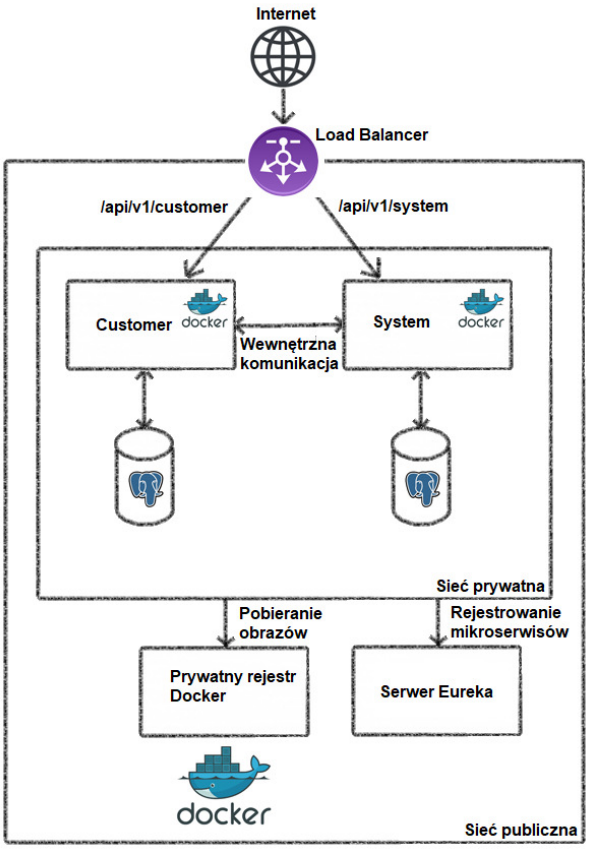
4.2. Aplikacja skalowalna

Druga, skalowalna aplikacja (Rys. 5) jest rozbudowaniem pierwszej, nieskalowalnej aplikacji. Zaimplementowano tutaj dodatkowo serwer Eureka, Spring Cloud OpenFeign oraz Spring Cloud Gateway.

Pakiet Spring Cloud Netflix dostarcza swój serwer Eureka. Panel serwera dostarcza informacje, takie jak zarejestrowani klienci Eureka, ich adres URL wraz z portem, a także liczba uruchomionych instancji [32].

Spring Cloud OpenFeign to deklaratywny klient REST dla aplikacji Spring Boot. Jest bardzo przydatnym narzędziem do tworzenia klientów usług internetowych za pomocą składni adnotacji. Zaletą OpenFeign jest prostota, do wywołania usługi wystarczy definicja interfejsu danego modułu [33].

Spring Cloud dostarcza swój własny Load Balancer – Spring Cloud Gateway. Umożliwia on opracowywanie mechanizmu przekierowań w aplikacjach opartych na architekturze mikrousług. Spring Cloud Gateway może działać zarówno jako Load Balancer jak i interfejs API [34].



Rysunek 5: Schemat aplikacji skalowalnej [31], [35].

Aplikacja skalowalna umożliwia przeprowadzenie testów obciążeniowych z wykorzystaniem wielu instancji serwisów. Posiada funkcjonalność opisaną w rozdziale 4.1. Konieczne jest, aby każda instancja była uruchomiona na innym, niezajętym porcie.

5. Metoda badań

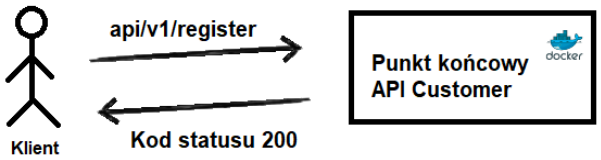
Aby pokazać możliwości skalowalności, dzięki narzędziom dostarczanym przez pakiet Spring Cloud, zostały przeprowadzone testy obciążeniowe na opracowanych aplikacjach. Każdy z eksperymentów został przeprowadzony w lokalnym środowisku. Skalowalność została zmierzona jako możliwość zwiększenia wydajności poprzez zwiększenie liczby instancji danego mikroserwisu.

W badaniach wzięto pod uwagę następujące czynniki:

- architekturę aplikacji (nieskalowalnej i skalowalna);
- liczbę instancji (1 dla aplikacji nieskalowalnej, od 1 do 5 dla poszczególnych serwisów aplikacji skalowalnej);
- liczbę zapytań (100, 1000, 10000).

5.1. Obiekt badań

Do celów badawczych zostały opracowane dwie aplikacje, które opisano w rozdziale 4. Aplikacje udostępniają punkt końcowy REST, rejestrujący nowego użytkownika w bazie danych (Rys. 6).



Rysunek 6: Schemat komunikacji użytkownika z serwisem customer [36].

Listing 1 przedstawia testowe dane rejestracji nowego użytkownika.

Listing 1: Parametry do rejestracji nowego użytkownika za pomocą serwisu customer

```
{
  "firstName": "firstName1",
  "lastName": "lastName1",
  "email": "email1@mail.com"
}
```

5.2. Środowisko badawcze

Eksperyment został przeprowadzony na lokalnym komputerze o następującej konfiguracji:

- procesor Intel Core i5-7300HQ CPU @ 2.50GHz;
- zainstalowana pamięć RAM 8,00 GB;
- typ systemu 64-bitowy system operacyjny, procesor x64;
- system operacyjny Windows 10 Home Edition.

5.3. Realizacja badań

Jako narzędzia badawcze zostały użyte programy popularny Apache JMeter oraz nowoczesny i konfigurowalny K6 [37].

Apache JMeter [38] to otwarto-źródłowy program Java, z prostym i intuicyjnym interfejsem graficznym (GUI). JMeter może przeprowadzać testy obciążenia i wydajności dla wielu różnych typów serwerów. Program jest elastyczny, pozwala na łatwą konfigurację parametrów testowych, takich jak liczba żądań i wątków.

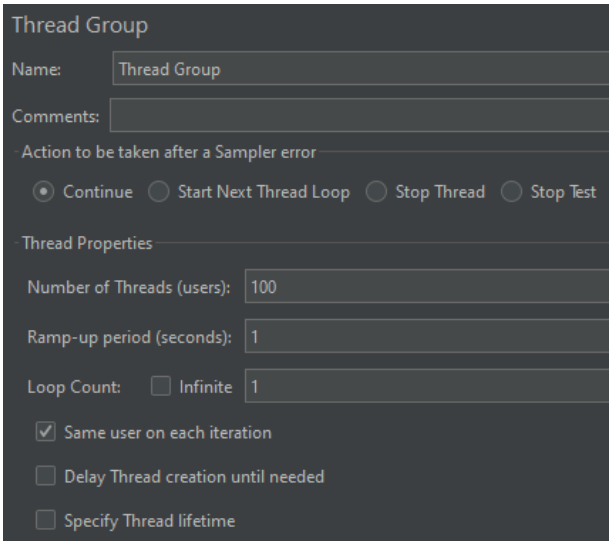
Na Rysunku 7 została przedstawiona konfiguracja programu, wykonująca konkretną liczbę zapytań (pole Number of Threads).

W oknie HTTP Request należy wprowadzić dane do zapytania REST, takie jak adres URL (pole Path), parametry żądania (pole Body Data) (Rys. 8).

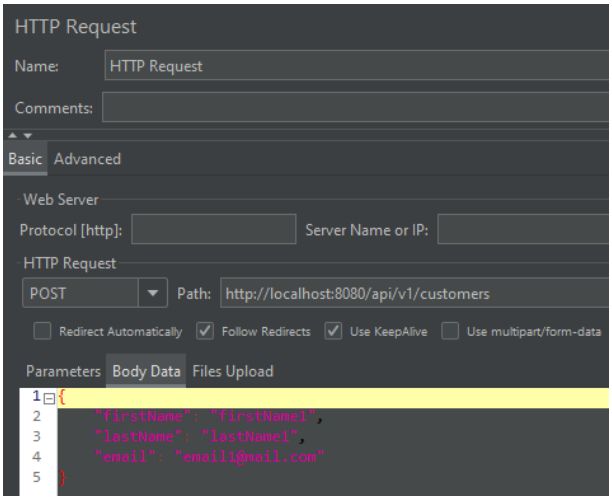
Po uruchomieniu zapytania w zakładkach podsumowania widoczne są różne parametry, takie jak czas wykonania danej liczby zapytań (Rys. 9).

K6 [39] jest również otwarto-źródłowym narzędziem testowania obciążenia (ang. load testing tool) do testowania wydajności API, mikroserwisów oraz stron internetowych. Program nie posiada GUI, może być używany jako wtyczka do środowiska Visual Studio Code. Wykorzystuje język JavaScript do tworzenia skryptów testów i posiada wiele możliwości konfiguracji.

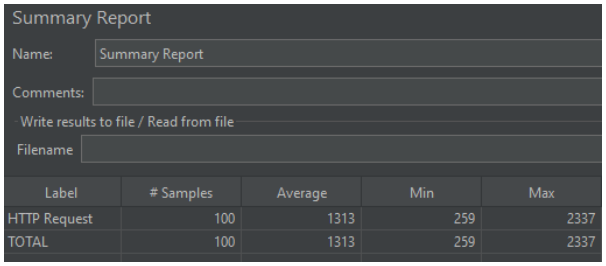
Na Listingu 2 przedstawiono skrypt JavaScript do przeprowadzenia testów obciążeniowych. Zmienna *options* odpowiada za konfigurację opcji testów, takich jak liczba wirtualnych maszyn – liczby zapytań (zmienna *vus*). Główna funkcja uruchamia testy i należy w niej zdefiniować metodę żądania, adres URL oraz parametry.



Rysunek 7: Konfiguracja grupy wątków programu JMeter.



Rysunek 8: Konfiguracja żądania HTTP w JMeter.



Rysunek 9: Przykładowy wynik uruchomienia testów w JMeter.

Listing 2: Skrypt JavaScript do przeprowadzenia testów obciążeniowych programem K6

```
// zaimportowanie modułu http do wykonywania zapytań API
import http from 'k6/http';
// konfiguracja testów
export const options = {
  // liczba maszyn wirtualnych (symyulujące użytkowników wykonujących zapytania)
  vus: 100,
  // czas trwania testów
  duration: '1s'
```

```
};
// główna funkcja uruchamiająca testy
export default function () {
  // parametry zapytania
  const payload = JSON.stringify({
    "firstName": "firstName1",
    "lastName": "lastName1",
    "email": "email1@mail.com"
  });
  // wykonanie testów
 http.post('http://localhost:8080/api/v1/customers'
, payload, {
    headers: {
      'Content-Type': 'application/json',
    },
  });
}
```

W trakcie badań skrypt uruchamiano w następujący sposób:

k6 run nazwa_pliku.js.

W konsoli wyświetlony zostanie wynik przeprowadzonych testów (Rys. 10).

```
data_received..................: 13 kB 5.4 kB/s
data_sent......................: 38 kB 17 kB/s
http_req_blocked...............: avg=18.92ms  min=0s       med=22ms     max=68.64ms
http_req_connecting............: avg=17.75ms  min=0s       med=21ms     max=66.83ms
http_req_duration..............: avg=973.63ms min=167.03ms med=1.12s    max=2.14s
  { expected_response:true }...: avg=973.63ms min=167.03ms med=1.12s    max=2.14s
http_req_failed................: 0.00% ✓ 0         ✗ 171
http_req_receiving.............: avg=193.83µs min=0s       med=0s       max=9.04ms
http_req_sending...............: avg=3.28ms   min=0s       med=999.6µs  max=24ms
http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s
http_req_waiting...............: avg=970.15ms min=167.03ms med=1.12s    max=2.13s
http_reqs......................: 171   73.989316/s
iteration_duration.............: avg=993.04ms min=167.56ms med=1.12s    max=2.17s
iterations.....................: 171   73.989316/s
vus............................: 35    min=35      max=100
vus_max........................: 100   min=100     max=100
```

Rysunek 10 Przykładowy rezultat przeprowadzonego testu programem K6.

6. Wyniki badań

Wyniki przeprowadzonych testów zostały przedstawione w Tabelach 1-6 oraz na Rysunkach 11-16.

Wydajność aplikacji badano przy różnej licznie zapytań: 100, 1000 i 10000.

W tabelach i na wykresach zastosowano oznaczenia:

- AN – aplikacja nieskalowalna;
- AS – aplikacja skalowalna;
- AxB: A – liczba instancji serwisu *system*, B – liczba instancji serwisu *customer*.

Tabela 1: Wydajność aplikacji przy liczbie 100 zapytań

Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 1x2	AS 1x3	AS 1x4	AS 1x5
JMeter	42.7	75.5	60.1	44.1	94.5	79.1
K6	302.9	124.9	143.0	239.0	194.0	137.6

Tabela 2: Wydajność aplikacji przy liczbie 1000 zapytań

Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 1x2	AS 1x3	AS 1x4	AS 1x5
JMeter	100.6	91.4	99.4	106.8	137.4	137.4
K6	193.2	181.9	187.8	254.5	160.2	189.2

Tabela 3: Wydajność aplikacji przy liczbie 10000 zapytań

Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 1x2	AS 1x3	AS 1x4	AS 1x5
JMeter	84.5	83.6	158.5	266.2	201.3	153.3
K6	132.7	104.8	100.3	110.6	70.0	119.1

Tabela 4: Wydajność aplikacji przy liczbie 100 zapytań

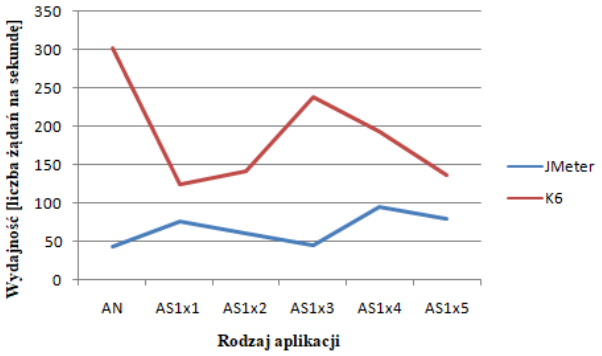
Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 2x2	AS 3x3	AS 4x4	AS 5x5
JMeter	42.7	75.5	30.0	98.7	72.0	98.8
K6	302.9	124.9	38.5	215.5	97.6	154.0

Tabela 5: Wydajność aplikacji przy liczbie 1000 zapytań

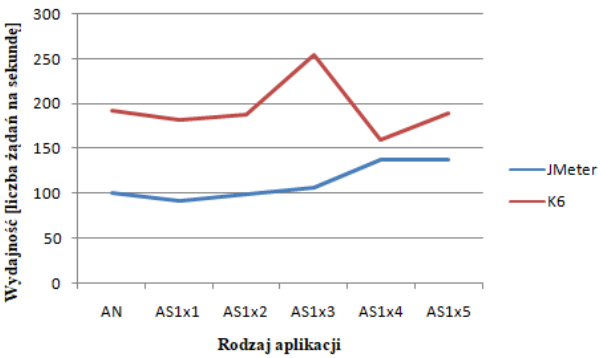
Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 2x2	AS 3x3	AS 4x4	AS 5x5
JMeter	100.6	91.4	100.2	136.6	147.2	115.8
K6	193.2	181.9	197.0	100.2	157.4	102.0

Tabela 6: Wydajność aplikacji przy liczbie 10000 zapytań

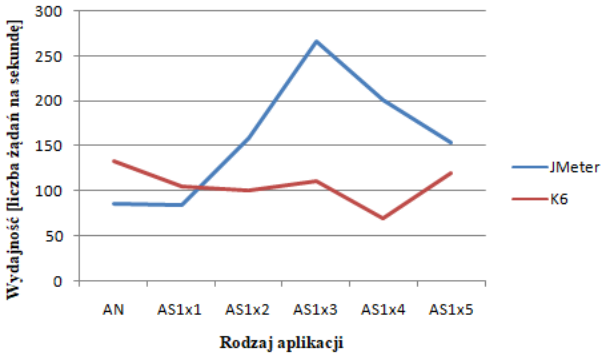
Aplikacja	Wydajność (liczba żądań na sekundę)					
	AN	AS 1x1	AS 2x2	AS 3x3	AS 4x4	AS 5x5
JMeter	84.5	83.6	101.9	146.8	95.0	84.0
K6	132.7	104.8	147.2	84.7	114.2	82.0



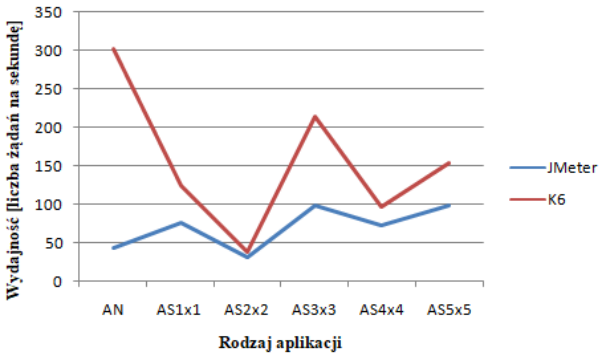
Rysunek 11: Wydajność aplikacji przy liczbie 100 zapytań.



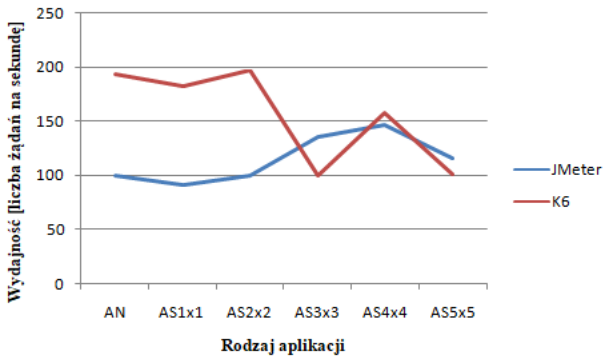
Rysunek 12: Wydajność aplikacji przy liczbie 1000 zapytań.



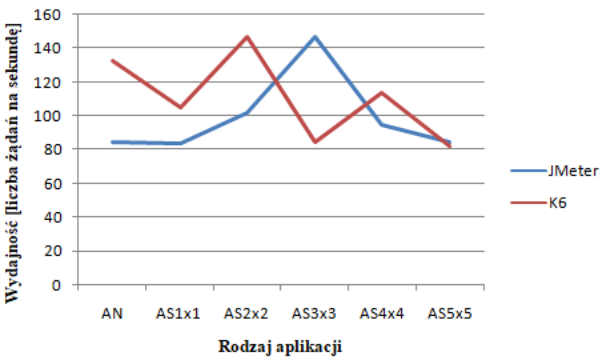
Rysunek 13: Wydajność aplikacji przy liczbie 10000 zapytań.



Rysunek 14: Wydajność aplikacji przy liczbie 100 zapytań.



Rysunek 15: Wydajność aplikacji przy liczbie 1000 zapytań.



Rysunek 16: Wydajność aplikacji przy liczbie 10000 zapytań.

7. Analiza wyników badań

Otrzymane wyniki pokazały, że Spring Cloud pozwala na skalowanie aplikacji, za pomocą narzędzi, takich jak Spring Cloud OpenFeign oraz Spring Cloud Gateway.

Warto zaznaczyć, że w pracy opracowane aplikacje były uruchomione w lokalnym środowisku, w którym nie zastosowano skalowania poziomowego (zwiększenia zasobów, tj. pamięci RAM lub CPU) lub pionowego (większej liczbie maszyn). Badania w lokalnym środowisku nie wykazały znaczącej poprawy wydajności ze względu na rodzaj skalowania (tj. zwiększanie liczby instancji poszczególnych mikroserwisów) oraz ograniczenia konfiguracji maszyny, tj. zwiększenia zasobów. Badania miały na celu jedynie pokazać efekty działających aplikacji z użyciem narzędzi dostarczanych przez Spring Cloud do tworzenia aplikacji, które mogą działać w chmurze.

Wybór narzędzi do przeprowadzenia testów wydajnościowych miał znaczenie. Zarówno narzędzie JMeter jak i K6 były dosyć łatwe w konfiguracji. K6 pozwolił na znacząco szybsze przeprowadzenie testów. Jest narzędziem lepiej zoptymalizowanym pod kątem wykonywania testów obciążeniowych. K6 pozwala również na szerszą konfigurację testów. JMeter posiada bardziej przyjazny i intuicyjny interfejs. K6 wymaga znajomości języka JavaScript oraz posługiwania się konsolą CLI.

Duże różnice w wynikach przy liczbie 100 (Tabele 1, 4 i Rysunki 11, 14) i 10000 (Tabele 3, 6 i Rysunki 13, 16) zapytań wynikały z zastosowanych aplikacji do przeprowadzania testów. K6 jest bardziej nowoczesnym narzędziem w porównaniu do JMeter, jednym z najlepszych dostępnych do tego typu testów [45].

Kolejnym powodem tak dużych różnic w wynikach była zbyt mała (100) lub zbyt duża liczba zapytań (1000). Przy liczbie 100 zapytań aplikacje szybko przetwarzały żądania, a przy liczbie 10000 zapytań platforma testowa (lokalny komputer) była zbyt obciążona.

Na różnice wpływ miało również lokalne środowisko, w którym nie zastosowano skalowania poziomowego lub pionowego. Rodzaj skalowania ma znaczenie w przypadku tego typu testów. W przypadku skalowania pionowego dodawane jest więcej zasobów (np. pamięci lub CPU), a w przypadku skalowania poziomowego ruch sieciowy jest rozprowadzany po większej liczbie maszyn.

Najbardziej optymalna okazała się liczba 1000 (Tabele 2, 5 i Rysunki 12, 15) zapytań. Wydajność aplikacji nieskalowanej i skalowalnej utrzymywała się na podobnym poziomie, ponieważ skalowanie aplikacji odbywało się poprzez zwiększanie liczby instancji poszczególnych serwisów. Nie zastosowano w badaniach skalowania poziomowego (zwiększenia zasobów, tj. pamięci RAM lub CPU) lub pionowego (większej liczbie maszyn) ze względu na ograniczenia spowodowane lokalnym środowiskiem.

Uruchomienie aplikacji w środowisku chmurowym, takim jak Kubernetes pozwala na skalowanie pionowe lub poziome. Skalowanie poziome oznacza to, że zwiększone obciążenie aplikacji zbyt dużych ruchem sieciowym zostanie rozprowadzone po większej liczbie

maszyn. Skalowanie pionowe polega na zwiększeniu zasobów (np. pamięci lub CPU) maszyn już uruchomionych. Pozwala to na przeprowadzenie nowych testów wydajnościowych w platformie chmurowej Kubernetes.

8. Wnioski i przyszłe kierunki badań

Narzędzia Spring Boot i rozszerzenie Spring Cloud zdecydowanie pomagają programistom w tworzeniu aplikacji bazujących na architekturze mikroserwisów w środowiskach chmurowych. W ostatnich latach można znaleźć więcej pozycji w literaturze, w których przedstawiona zostaje architektura mikroserwisów z użyciem Spring Boot, Spring Cloud. Również wzrasta liczba badań skupiających się na wydajności i skalowalności takich aplikacji.

W pracy przeanalizowano głównie możliwości jakie oferuje Spring Boot i Spring Cloud w tworzeniu aplikacji chmurowych. Opracowane aplikacje pokazały jakie korzyści daje wykorzystanie odpowiednich narzędzi takich jak Spring Cloud OpenFeign czy Spring Cloud Gateway. Aplikacje opracowane w pracy pozwoliły na przeprowadzenie testów obciążeniowych. Aplikacja nieskalowana skupia się jedynie na stworzeniu własnych mikroserwisów z użyciem Spring Boot. Aplikacja skalowalna z wykorzystaniem narzędzi dostarczanych przez Spring Boot i Spring Cloud pozwala na uruchomienie wielu instancji mikroserwisów. Lokalne środowisko nie daje możliwości pokazania prawdziwego skalowania aplikacji w środowisku produkcyjnym.

Następnym krokiem w badaniach w świecie mikroserwisów powinno być wdrożenie aplikacji przedstawionej w rozdziale 4 do środowiska chmurowego. Istnieje wiele platform chmurowych pozwalających na uruchomienie aplikacji z wykorzystaniem Spring Cloud, takich jak Kubernetes.

Artykuł miał charakter przeglądowy, z ukierunkowaniem na wskazanie możliwości oferowanych przez Spring do tworzenia aplikacji chmurowych. Badania przeprowadzone w pracy pokazały jedynie możliwości skalowania. Wdrożenie aplikacji do środowiska chmurowego Kubernetes pozwoliłoby na skalowanie pionowe lub poziome, co otwiera to nowe możliwości do przeprowadzenia kolejnych badań.

Literatura

[1] Strona internetowa z informacjami na temat branż korzystających z architektury mikrousług, https://codeandpepper.com/companies-using-microservices, [1.08.2022].

[2] Strona internetowa z informacjami na temat powodów, dla których branże zwracają się ku mikroserwisom, https://annexbyte.com/blog/industries-turning-to-microservices, [1.08.2022].

[3] Strona internetowa z informacjami na temat technologii i języków do wyboru w budowaniu architektury mikroserwisów, https://www.mindinventory.com/blog/technologies-for-mircroservices-architecture, [1.08.2022].

[4] Strona internetowa z informacjami na temat popularności technologii i języków do wyboru w budowaniu architektury mikroserwisów, https://www.thirdrocktechkno.com/blog/top-languages-for-microservices-architecture, [1.08.2022].

[5] Strona internetowa z informacjami na temat narzędzi do budowania mikroserwisów, https://dzone.com/articles/30top-tools-for-building-microservices-on-all-leve, [1.08.2022].

[6] Strona internetowa z informacjami na temat Spring Cloud, https://spring.io/projects/spring-cloud, [1.08.2022].

[7] Strona internetowa z informacjami na temat mikroserwisów, https://aws.amazon.com/microservices, [1.08.2022].

[8] Strona internetowa z informacjami na temat Spring Boot, https://spring.io/projects/spring-boot, [1.08.2022].

[9] Strona internetowa z informacjami na temat Spring Cloud, https://spring.io/projects/spring-cloud, [1.08.2022].

[10] Strona internetowa z informacjami na temat Service Discovery, https://avinetworks.com/glossary/service-discovery, [1.08.2022].

[11] Strona internetowa z informacjami na temat mechanizmu serwera Eureka, https://medium.com/@ijayakantha/microservices-service-registration-and-discovery-with-netflix-eureka-9a2aa729da96, [1.08.2022].

[12] Strona internetowa z informacjami na temat Load Balancer, https://www.ovhcloud.com/pl/public-cloud/what-load-balancing, [1.08.2022].

[13] Strona internetowa z informacjami na temat Docker, https://docs.docker.com/get-started/overview, [1.08.2022].

[14] Strona internetowa z informacjami na temat mechanizmu CI/CD, https://docs.gitlab.com/ee/ci/introduction, [1.08.2022].

[15] Strona internetowa z informacjami na temat Kubernetes, https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-is-kubernetes, [1.08.2022].

[16] Strona internetowa z informacjami na temat Kubernetes, https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-is-kubernetes, [1.08.2022].

[17] Strona internetowa z informacjami na temat języku programowania Borg, https://memory-alpha.fandom.com/wiki/Borg_language, [1.08.2022].

[18] Strona internetowa z informacjami na temat języku programowania Omega, https://hackage.haskell.org/package/omega, [1.08.2022].

[19] C. M. Aderaldo, N. C. Mendonça, C. Pahl, P. Jamshidi, Benchmark Requirements for Microservices Architecture Research, 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for

Architecture-Based Software Engineering (ECASE) (2017) 8-13.

[20] D. Rajput, Mastering Spring Boot 2.0: Build modern, cloud-native, and distributed systems using Spring Boot, Packt Publishing Ltd, 2018.

[21] E. Wolff, Microservices: flexible software architecture, Addison-Wesley Professional, 2016.

[22] J. Carnell, I. H. Sánchez, Spring microservices in action, Simon and Schuster, 2021.

[23] K. S. P. Reddy, Beginning Spring Boot 2: Applications and microservices with the Spring framework, Apress, 2017.

[24] Q. Perez, A. Le Borgne, C. Urtado, S. Vauttier, An Empirical Study about Software Architecture Configuration Practices with the Java Spring Framework, SEKE: Software Engineering and Knowledge Engineering (2019) 465-468.

[25] V. Saquicela, G. Campoverde, J. Avila, M. E. Fajardo, Building microservices for scalability and availability: Step by step, from beginning to end, International Conference on Software Process Improvement, Springer, Cham (2020) 169-184.

[26] S. Sharma, Mastering Microservices with Java - Third Edition: Build enterprise microservices with Spring Boot 2.0, Spring Cloud, and Angular, Packt Publishing Ltd, 2019.

[27] S. Hatma, J. D. Puji, T. Aris, Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot, Procedia Computer Science (2017) 124, 736-743.

[28] Strona internetowa z informacjami na temat REST API, https://www.redhat.com/en/topics/api/what-is-a-rest-api, [1.08.2022].

[29] Strona internetowa z informacjami na temat konfiguracji zewnętrznej Spring Boot, https://www.baeldung.com/spring-boot-yaml-vs-properties, [1.08.2022].

[30] Strona internetowa z informacjami na temat uruchamiania aplikacji Spring Boot w kontenerach Docker, https://www.baeldung.com/dockerizing-spring-boot-application, [1.08.2022].

[31] Strona internetowa z informacjami na temat tworzenia aplikacji monolitycznych oraz opartych o architekturę mikrousług, https://www.dineshonjava.com/microservices-with-spring-boot, [1.08.2022].

[32] Strona internetowa z informacjami na temat serwera Eureka w aplikacji Spring Boot, https://www.tutorialspoint.com/spring_boot/spring_boot_eureka_server.htm, [1.08.2022].

[33] Strona internetowa z informacjami na temat Spring Cloud OpenFeign, https://www.baeldung.com/spring-cloud-openfeign, [1.08.2022].

[34] Strona internetowa z informacjami na temat Load Balancer i interfejsu API, https://www.techtarget.com/searchitoperations/answer/Whats-the-role-of-an-application-load-balancer-vs-API-gateway, [1.08.2022].

[35] Strona internetowa z informacjami na tworzenia aplikacji opartych o architekturę mikrousług z wykorzystaniem narzędzi Spring Cloud, https://spring.io/microservices, [1.08.2022].

[36] Strona internetowa z informacjami na temat mechanizmu REST API, https://www.geeksforgeeks.org/rest-api-introduction, [1.08.2022].

[37] Strona internetowa z informacjami na temat narzędzi do przeprowadzania testów obciążeniowych, https://pflb.us/blog/best-load-testing-tools, [1.08.2022].

[38] Strona internetowa z informacjami na temat narzędzia JMeter, https://www.simplilearn.com/tutorials/jmeter-tutorial/jmeter-performance-testing, [1.08.2022].

[39] Strona internetowa z informacjami na temat narzędzia K6, https://www.geeksforgeeks.org/performance-testing-with-k6, [1.08.2022].