JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 27 (2023) 104–111
Received: 17 October 2022
Accepted: 20 November 2022

Comparative Analysis of the Net 6 and NestJS Programming Frameworks in Terms of their Suitability for User Authentication and Authorization

Analiza porównawcza szkieletów programistycznych Net 6 i NestJS pod kątem ich przydatności do autentykacji i autoryzacji użytkowników

Przemysław Szymon Rodzik*

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

Abstract
The aim of the article was to perform a comparative analysis of the Net 6 and NestJS programming framework in terms of their suitability for user authentication and authorization. The functionalities and programming libraries offered by the researched technologies were reviewed. Applications were created in the tested skeletons. Application performance and load tests were carried out. The obtained test results showed that the application written in NestJS offered a shorter time to service the request and was able to handle a larger number of users compared to the application using Net 6. Net 6 offered a greater number of functionalities in the field of authentication and authorization, their implementation required less work from the developer compared to the NestJS backbone.

Keywords: comparative analysis; authentication; NestJS; Net

Streszczenie
Celem artykułu było przeprowadzenie analizy porównawczej szkieletu programistycznego Net 6 i NestJS pod kątem ich przydatności w autentykacji i autoryzacji użytkowników. Dokonano przeglądu funkcjonalności oraz bibliotek programistycznych oferowanych przez badane technologie. Utworzono aplikacje w badanych szkieletach. Przeprowadzono testy wydajnościowe oraz obciążeniowe aplikacji. Otrzymane wyniki testów pokazały, iż aplikacja napisana w NestJS oferowała krótszy czas obsłużenia żądania oraz była w stanie obsłużyć większą liczbę użytkowników w porównaniu do aplikacji wykorzystującej Net 6. Net 6 oferował większą liczbę funkcjonalności z dziedziny autentykacji i autoryzacji, ich implementacja wymagała mniej pracy od programisty w porównaniu do szkieletu NestJS.

Słowa kluczowe: analiza porównawcza; autentykacja; NestJS; Net

*Corresponding author
Email address: **przemyslaw.rodzik@pollub.edu.pl** (P. S. Rodzik)

1. Wstęp

Kluczowym elementem wielu systemów jest proces autentykacji i autoryzacji użytkowników korzystających z tego systemu. Proces autentykacji to sposób, w którym użytkownik dostarcza aplikacji dowodów, iż jest on w rzeczywistości osobą za którą się podaje poprzez przedstawienie ważnego poświadczenia [1]. Najczęściej za poświadczenie uważa się login i hasło. W przypadku dostarczenia poprawnych danych przez użytkownika jego tożsamość zostaje potwierdzona, a proces uwierzytelnienia kończy się powodzeniem. Autoryzacja jest to proces nadawania uprawnień do wykonywania określonych akcji w systemie określonym podmiotom [2]. Najczęściej do akcji tych zalicza się wykonywanie operacji na chronionych zasobach. Proces autoryzacji wymaga, by dany podmiot został uprzednio uwierzytelniony.

W artykule dokonano analizy porównawczej szkieletu programistycznego Net 6 [3] i NestJS [4] pod kątem ich przydatności w autentykacji i autoryzacji użytkowników. Analiza ta miała na celu dostarczyć programiście przydatnych informacji dotyczących zagadnień związanych z autentykacją i autoryzacją, występujących w omawianych szkieletach programistycznych, które mogłyby pomóc mu w podjęciu decyzji dotyczącej wyboru najlepszego szkieletu programistycznego implementującego badane zagadnienie. W tym celu porównano biblioteki programistyczne oraz funkcjonalności z dziedziny autentykacji i autoryzacji oferowane przez badane szkielety programistyczne. Utworzono aplikacje w każdym z badanych szkieletów oferujące identyczne funkcjonalności umożliwiające autentykację i nadanie uprawnień użytkownikowi. Wykonano testy wydajnościowe, w których mierzono czas wykonania zapytań w zależności od zastosowanych algorytmów do procesu uwierzytelniania i autoryzacji. Sprawdzono jak duży ruch sieciowy jest w stanie obsłużyć dana technologia.

1.1. Przegląd literatury

Powstało wiele pozycji poruszających temat procesu autentykacji i autoryzacji użytkowników. W artykule „Evaluation of password hashing schemes in open source web platforms” [5] autorzy zbadali ponad 10 szkieletów programistycznych pod kątem algorytmów, które są w nich domyślnie zaimplementowane w procesie uwierzytelniania. Wyniki pokazały, iż w większości technologii użyte algorytmy są przestarzałe, nie gwarantują bezpieczeństwa. Dodatkowo sporządzili listę wytycznych, zaproponowali alternatywne algorytmy w celu

zwiększenia poziomu bezpieczeństwa. W pracy „Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms" [6] utworzono aplikacje, w której przeprowadzono testy mierzące czas obsłużenia żądań logowania dla algorytmu Bcrypt [7], Scrypt [8] i PBKDF2 [9] w zależności od użytych parametrów wejściowych. Udowodniono, iż najszybszym rozwiązaniem jest algorytm PBKDF2. Niestety w badaniach nie uwzględnili jaki wpływ ma liczba zapytań wykonywanych jednocześnie na czas obsłużenia żądania. Branimir Pervan, Josip Knezovic i Katja Pericin [10] dokonali testów wydajnościowych opracowanego przez nich systemu rozproszonego do obliczania wartości funkcji skrótu algorytmu Bcrypt. Pokazali iż wydajność ich systemu, jak i jego efektywność energetyczna nie odbiega znacząco od porównywanych systemów. Aleksander Dikanski i Roland Steinegger [11] przeanalizowali szkielet Spring Security pod kątem jego wsparcia dla autentykacji i autoryzacji użytkowników. Ponadto dostarczyli czytelnikom wzorców do implementacji procesu uwierzytelnienia. Autorzy „Systematic Review of Authentication and Authorization Advancements for the Internet of Things" [12] przeanalizowali 1622 artykuły z dziedziny Internetu Rzeczy [13] w celu oceny aktualnego poziomu jakości implementacji rozwiązań z dziedziny autentykacji i autoryzacji. Wyniki ich badań pokazały, iż w większości systemów, moduły realizujące proces autentykacji, autoryzacji użytkowników działają w architekturze rozproszonej. Ponadto metoda autoryzacji użytkowników oparta na rolach [14] (ang. Role-based access control) traci na popularności na rzecz metody opierającej się na atrybutach [15] (ang. Attribute-based access control).

1.2. Cel badań

Celem badań było porównanie szkieletów programistycznych Net w wersji 6.0.101 i NestJS w wersji 8.0.0 pod kątem ich przydatności do autentykacji i autoryzacji użytkowników.

Hipotezy, których weryfikacji podjęto się w niniejszym artykule brzmiały następująco:

1. Technologia Net 6 oferuje większą liczbę funkcjonalności z zakresu autentykacji i autoryzacji niż technologia NestJS.
2. Średni czas obsłużenia żądania wysłanego do aplikacji napisanej przy wykorzystaniu szkieletu programistycznego NestJS jest krótszy niż w przypadku aplikacji korzystającej z technologii Net 6.
3. Aplikacja napisana w technologii NestJS w ciągu 10 sekund jest w stanie obsłużyć większą liczbę żądań wysyłanych przez użytkowników niż serwis napisany przy użyciu szkieletu programistycznego Net 6.

W celu weryfikacji powyższych hipotez utworzono dwie aplikacje prezentujące te same funkcjonalności, składające się z trzech mikro-usług. Każda z nich została napisana przy wykorzystaniu najnowszych wersji tychże technologii, tj. NestJS w wersji 8.0.0 i Net w wersji 6.0.101. Dokonano przeglądu funkcjonalności oraz rozwiązań z zakresu autentykacji i autoryzacji dostępnych w badanych szkieletach. Wykonano testy wydajnościowe oraz obciążeniowe badanych aplikacji w zależności od algorytmów użytych w procesie autentykacji. Otrzymane wyniki umożliwiły sformułowanie wniosków, weryfikację sformułowanych hipotez.

2. Metody badań

W celu porównania badanych technologii pod kątem ich przydatności do autentykacji i autoryzacji użytkowników:

- dokonano przeglądu dostępnych funkcjonalności oraz bibliotek programistycznych w badanych technologiach z zakresu uwierzytelniania i autoryzacji,
- utworzono aplikację testową w każdym z badanych szkieletów,
- przeprowadzono testy wydajnościowe mające na celu zebrać dane na temat czasu żądania wysłanego przez klienta. Testy te miały za zadanie porównać wydajność aplikacji w zależności od algorytmów użytych w procesie uwierzytelniania i autoryzacji,
- wykonano testy sprawdzające ile maksymalnie żądań wysłanych przez użytkowników jest w stanie obsłużyć serwis w zależności od sprzętu, na którym został uruchomiony.

2.1. Środowisko testowe

Testy przeprowadzono na laptopie marki Lenovo Legion V. Tabela 1 przedstawia parametry maszyny.

Tabela 1: Parametry komputera testowego

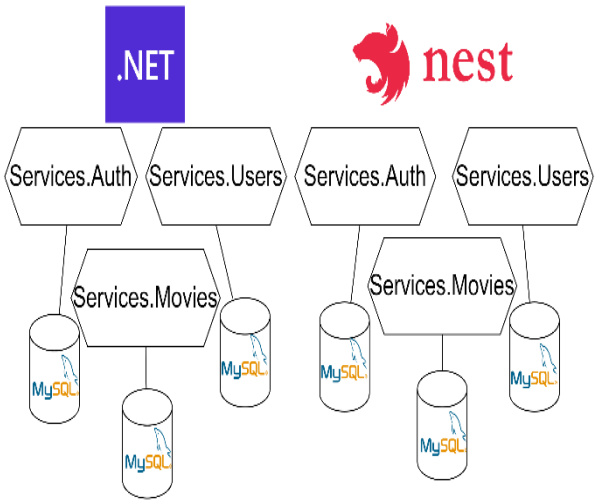
Parametr	Opis
Jednostka obliczeniowa	AMD Ryzen 7 4800H, 8 rdzeni, 16 wątków, 4,2Ghz
Karta graficzna	Nvidia Geforce RTX 2060 6GB
Pamięć RAM	32GB
Dysk	Crucial CT1000MX500SSD1
System operacyjny	Windows 10 Professional 21H1 19043.1586

W przypadku testów obciążeniowych wykorzystano dodatkowo maszynę wirtualną Vmware Workstation Player 16.0 [16] w celu zasymulowania działania aplikacji na mniej wydajnym sprzęcie. Zadbano by laptop był podłączony do sieci elektrycznej podczas przeprowadzania testów. Wszystkie inne procesy oprócz procesów systemu operacyjnego, które nie musiały pozostać uruchomione, zostały wyłączone.

2.2. Architektura aplikacji testowych

Aplikacje testowe zostały napisane przy wykorzystaniu architektury implementującej wzorzec architektury zorientowanej na usługi [17].

Na system składały się trzy mikro-usługi: uwierzytelnienia, użytkowników i filmów. Każda z nich posiadała własną bazę danych. Komunikowały się między sobą przy wykorzystaniu protokołu zdalnego wywoływania procedur (ang. Remote Procedure Call) firmy Google [18]. Wszystkie z nich wystawiały serwer http w celu udostępnienia swoich funkcjonalności użytkownikom. Rysunek 1 ilustruje schemat opisanej architektury dwóch systemów.



Rysunek 1: Architektura aplikacji testowych.

2.3. Implementacja aplikacji testowych

Aplikacje napisano wykorzystując szkielety programistyczne:

- Net w wersji 6.0.101,
- NestJS w wersji 8.0.0.

Zaprojektowano je w taki sposób by programista mógł zintegrować dane funkcjonalności do swojego systemu.

Szczególny nacisk postawiono na implementację serwisu uwierzytelniania. Tabela 2 przedstawia żądania możliwe do wysłania przez użytkownika do serwisu uwierzytelniania. Szczegółowo opisano tylko żądanie logowania, ponieważ tylko ono było wykorzystane w testach wydajnościowych.

Tabela 2: Żądania http możliwe do wysłania do serwisu uwierzytelnienia

Nazwa żądania	Ścieżka (względna)	Typ metody
Register	/register	POST
Login	/login	POST
Refresh	/refresh	POST
Logout	/logout	POST

Zadaniem żądania o nazwie „Login” było wygenerowanie tokenu JWT (ang. Json Web Token) [19], który umożliwiał użytkownikowi czasowy dostęp do serwisu. Klucz dostępu był generowany po uprzednim sprawdzeniu czy dostarczone dane logowania są prawidłowe. Po otrzymaniu danych logowania serwis uwierzytelniania w pierwszej kolejności wywoływał zdalnie procedurę, która zwracała dane o kliencie z serwisu użytkowników. Jeśli serwis uwierzytelniania nie otrzymał w odpowiedzi informacji o użytkowniku, zwracał kod błędu o numerze 401. W innym przypadku, serwis ten porównywał skrót hasła otrzymany w odpowiedzi od serwisu użytkowników ze skrótem obliczonym z hasła wprowadzonego przez klienta. Jeśli skróty się zgadzały, mikrousługa generowała token dostępu i token odświeżenia [20]. Następnie zwracała do użytkownika wiadomość zawierającą opisane powyżej tokeny.

Zadaniem serwisu użytkowników było zarządzanie znajdującymi się w bazie danych klientami aplikacji. Serwis ten odpowiadał na żądania wysyłane ze strony serwisu uwierzytelniania.

Serwis o nazwie „Services.Movies” został utworzony w celu wykonania testów odnoszących się do autoryzacji użytkownika w zależności od rodzaju użytego tokenu do walidacji. Implementacja serwisu filmów ograniczała się do udostępnienia punktu dostępu umożliwiającego pobranie informacji opisujących dany film. Skupiono się by dostęp do podanych treści był możliwy jedynie po pomyślnej walidacji tokenu JWT i w sytuacji, gdy użytkownik posiadał wymagane uprawnienie.

2.4. Przegląd funkcjonalności oraz bibliotek programistycznych do autentykacji i autoryzacji

Dokonano przeglądu bibliotek oferujących rozwiązania w zakresie autentykacji i autoryzacji, dostępnych w badanych szkieletach programistycznych. Sprawdzano dostępność i jakość bibliotek implementujących następujące funkcjonalności:

- generowanie i walidacja tokenów JWT (ang. Json Web Token),
- tworzenie skrótów kryptograficznych,
- tworzenie losowych ciągów bezpiecznych kryptograficznie,
- autoryzacje użytkowników dzięki rozpoznaniu ich twarzy bądź gestów,
- generowanie jednorazowych haseł.

Dla każdej funkcjonalności wybrano najlepszą bibliotekę. Wyboru najlepszej z nich dokonano biorąc pod uwagę parametry:

- średnią liczbę pobrań z oficjalnych repozytoriów pakietów omawianych szkieletów,
- liczbę nierozwiązanych błędów zgłaszanych przez użytkowników,
- siłę oferowanych algorytmów,
- obszerność dokumentacji,
- siłę domyślnej konfiguracji,
- stopień aktywności ze strony twórców oprogramowania, tj. liczbę przeprowadzonych modyfikacji biblioteki w okresie dwóch lat.

Porównano funkcjonalności z dziedziny autentykacji i autoryzacji oferowane przez badane technologie. Podczas analizy zwrócono szczególną uwagę czy implementacja danej funkcjonalności była możliwa bez konieczności dołączenia bibliotek firm trzecich oraz, w której z technologii implementacja danej funkcjonalności wymagała mniej pracy od programisty.

2.5. Testy wydajnościowe

Napisane aplikacje poddano testom wydajnościowym. Do tego celu wykorzystano narzędzie k6 [21]. Oprogramowanie to pozwala na pisanie skryptów w języku JavaScript opisujących przebieg testu, które są następnie interpretowane i wykonywane przez program. Dodatkowo w czasie trwania testu zapisywano informacje na temat zużycia procesora i pamięci RAM przy użyciu programu dołączonego z systemem Windows o nazwie Performance Monitor [22].

Dla każdej z aplikacji przeprowadzone zostały testy logowania w zależności od rodzaju algorytmu użytego do obliczenia funkcji skrótu [23] z dostarczonego hasła użytkownika. Każdy typ testu trwał 9 minut. Zbierano

informację o czasie obsłużenia pojedynczego żądania (czas liczony od chwili wysłania żądania do otrzymania odpowiedzi zwrotnej od serwisu). Badania były przeprowadzane dla 5, 10 i 15 użytkowników. Pojęcia określane jako „liczba wirtualnych użytkowników", bądź „liczba użytkowników", które mogą pojawić się w tekście, oznaczają liczbę jednocześnie wykonywanych zapytań do aplikacji do momentu zakończenia testu.

Kod źródłowy realizujący scenariusz logowania do sytemu ukazany na listingu 1 różnił się jedynie adresem bazowym żądań http w zależności od testowanej aplikacji. Skrypt był wykonywany w nieskończoność przez każdego z wirtualnych użytkowników do chwili zakończenia testu. Czas oczekiwania na wysłanie kolejnego żądania przez danego użytkownika wynosił 500ms. Zadbano by każdy z wirtualnych użytkowników odzwierciedlał innego użytkownika z bazy danych.

Listing 1: Kod źródłowy skryptu realizujący scenariusz zalogowania użytkownika do systemu

```
let body = JSON.stringify({
  username: `user${__VU}`,
  password: "VqhvQXxR",
});
const params = {
  headers: {
    "Content-Type": "application/json",
  },
  tags: {
    name: `login${__VU}`,
  },
};
const responseLogin = http.post(URL_LOGIN, body, params);
check(responseLogin, {
  "logged successfully": (r) =>
    r.status === 200 ? r.json().hasOwnProperty("accessToken") : false,
});
sleep(SLEEP_DURATION_IN_SEC);
}
```

Tabela 3 przedstawia algorytmy poddane testom wraz z bibliotekami użytymi do ich zaimplementowania. Biblioteki wybrano kierując się kryteriami wyboru przedstawionymi w rozdziale 2.4.

Tabela 3: Algorytmy użyte w testach logowania

Algorytm	Biblioteka dla NestJS (wersja)	Biblioteka dla Net 6 (wersja)
Bcrypt	bcrypt (5.0.1)	BCrypt.Net-Next (4.03)
Argon2id	argon2 (0.28.5)	Konscious.Security.Cryptography.Argon2 (1.21)

Testy logowania z algorytmem Bcrypt wykonano zmieniając odpowiednio wartość parametru „work factor" na 10, 11, 12, 13 i 14. Zmienna ta opisuje liczbę iteracji algorytmu mieszającego, które są wykonywane dla każdego hasła.

Algorytm Argon2id [24] posiada 3 główne parametry, których wartości można zmieniać, są to:

- memory cost,
- iterations,
- parallelism.

Testy przeprowadzano zmieniając wartości parametru „memory cost" na 16MB, 32MB, 64MB, 128MB, 256MB. Zmienna ta informuje nas o ilości pamięci RAM jaką algorytm może wykorzystać w celu obliczenia wartości skrótu. Pozostałe parametry ustawiono na minimalne wartości rekomendowane przez fundację OWASP [25].

Zadaniem testów obciążeniowych było sprawdzenie ile maksymalnie zapytań wysyłanych przez użytkowników jest w stanie obsłużyć serwis w zależności od sprzętu, na którym pracuje. Za maksymalną liczbę zapytań przyjęto wartość, którą zwiększenie spowoduje wystąpienie błędów w aplikacji, odrzucanie przez nią żądań. Test był przeprowadzony w ten sposób, iż w ciągu 10 sekund określona liczba użytkowników miała za zadanie zalogować się do systemu. Stopniowo zwiększano liczbę użytkowników do momentu kiedy serwis zaczął generować błędy, odrzucać żądania. Testy przeprowadzano na maszynie wirtualnej. Z puli 16 procesorów oraz 32GB pamięci RAM oferowanych przez maszynę hosta, zmieniano liczbę dostępnych procesorów (4, 6, 8), ilość pamięci RAM (8GB, 16GB) oraz algorytmy użyte w procesie autentykacji (Bcrypt, Argon2id). Oba algorytmy skonfigurowano przy wykorzystaniu minimalnych, rekomendowanych przez fundację OWASP parametrów. Dla algorytmu Bcrypt parametr „work factor" został ustawiony na wartość 10. W przypadku Argon2id parametry: parallelism, memory, time ustawiono odpowiednio na wartości: 1, 16MB, 2.

3. Wyniki badań

Przegląd rozwiązań do autentykacji i autoryzacji pokazał, iż więcej bibliotek ma do zaoferowania technologia Net 6 niż NestJS. Siła, bezpieczeństwo algorytmów oferowanych przez obydwie technologie była wystarczająca, lecz w przypadku Net 6 biblioteki były częściej aktualizowane. Rozwiązania napisane dla technologii NestJS posiadały większą liczbę błędów zgłaszanych przez użytkowników niż dla technologii Net 6. W przypadku obu technologii nie znaleziono biblioteki, która dostarczałaby implementacji wszystkich rekomendowanych przez fundacje OWASP algorytmów do tworzenia skrótów kryptograficznych. W celu zaimplementowania do swojego systemu algorytmu Bcrypt, Scrypt bądź Argon2 programista był zmuszony do dołączenia dodatkowych bibliotek. Wybierając bibliotekę należy brać pod uwagę siłę domyślnej konfiguracji oferowanych przez nią algorytmów. Domyślna konfiguracja algorytmów dostępnych w badanych bibliotekach dla Bcrypt, Scrypt, Argon2 oraz HMAC była tożsama z tą rekomendowaną przez fundację OWASP. Tabela 4 przedstawia listę rekomendowanych bibliotek możliwych do wykorzystania w przypadku konieczności zaimplementowania danej funkcjonalności.

Analizując funkcjonalności oferowane przez technologie stwierdzono, iż szkielet Net 6 posiada ich więcej oraz oferowana przez nie domyślna implementacja jest bardziej rozbudowana i dostarcza wielu metod, które przyspieszają pracę programisty. Tabela 5 przedstawia podsumowanie dostępnych funkcjonalności z zakresu

autentykacji i autoryzacji w każdym z omawianych szkieletów programistycznych.

Tabela 4: Zalecane biblioteki do wykorzystania w celu implementacji wybranych funkcjonalności

Opis funkcjonalności	Nazwa biblioteki (wersja)	
	Net 6	NestJS
Generowanie i walidacja tokenów JWT	jose-jwt (4.0.0)	jose (4.8.1)
Tworzenie skrótów kryptograficznych	Konscious.Security.Cryptography.Argon2 (1.21), BCrypt.Net-Next (4.03), System.Security.Cryptography.Algorithms (4.01), Scrypt.NET (1.3.0)	argon2 (0.28.5), bcrypt (5.0.1), hash.js (1.0.1), node-scrypt (6.0.3)
Tworzenie losowych ciągów bezpiecznych kryptograficznie	System.Security.Cryptography.RandomCryptoGenerator (2.16)	Crypto-random-string (5.0.0)
Autoryzacja – rozpoznawanie twarzy	FaceRecognitionDotNet (1.3.07)	node-facenet (0.10.3)
Generowanie jednorazowych haseł	Otp.NET (1.2.2)	Otplib (12.0.1)

Tabela 5: Spis funkcjonalności z zakresu autentykacji i autoryzacji dostępnych w technologii Net 6 i NestJS

Funkcjonalność	NestJS	Net 6
Autentykacja przy użyciu hasła	✓	✓
Autentykacja przy użyciu tokenu JWT	✓	✓
Autentykacja przy użyciu zewnętrznych serwisów	✓	✓
Polityka haseł	×	✓
Obliczanie wartości funkcji skrótu	×	✓
Blokowanie użytkowników	×	✓
Dwuetapowe logowanie	×	✓
Tworzenie sesji	✓	✓
Potwierdzenie adresu e-mail	×	✓
Autoryzacja oparta na rolach	✓	✓
Autoryzacja oparta na oświadczeniach	✓	✓
Ukrywanie elementów interfejsu graficznego	×	✓
Tworzenie tabel przechowujących informacje o użytkownikach, ich rolach, oświadczeniach	×	✓
Zwracanie informacji o przypisanej roli, oświadczeniach zalogowanego użytkownika	×	✓

Tabele 6 i 7 przedstawiają informacje na temat maksymalnej liczby użytkowników, wykonującej żądanie logowania, możliwej do obsłużenia przez aplikacje napisaną przy użyciu szkieletu Net 6 oraz NestJS w zależności od parametrów maszyny testowej odpowiednio przy autentykacji wykorzystującej algorytm Bcrypt oraz Argon2id. Po analizie wyników zaprezentowanych w tabelach 6 i 7 stwierdzono, iż aplikacja napisana w technologii NestJS obsłuży większą liczbę użytkowników niż aplikacja wykorzystująca szkielet Net 6. Zauważono, iż dla szkieletu Net autentykacja z wykorzystaniem algorytmu Bcrypt pozwala na obsłużenie większej liczby żądań niż z wykorzystaniem algorytmu Argon2id. Z kolei dla technologii NestJS wykorzystanie algorytmu Argon2id umożliwia obsłużenie większej liczby użytkowników.

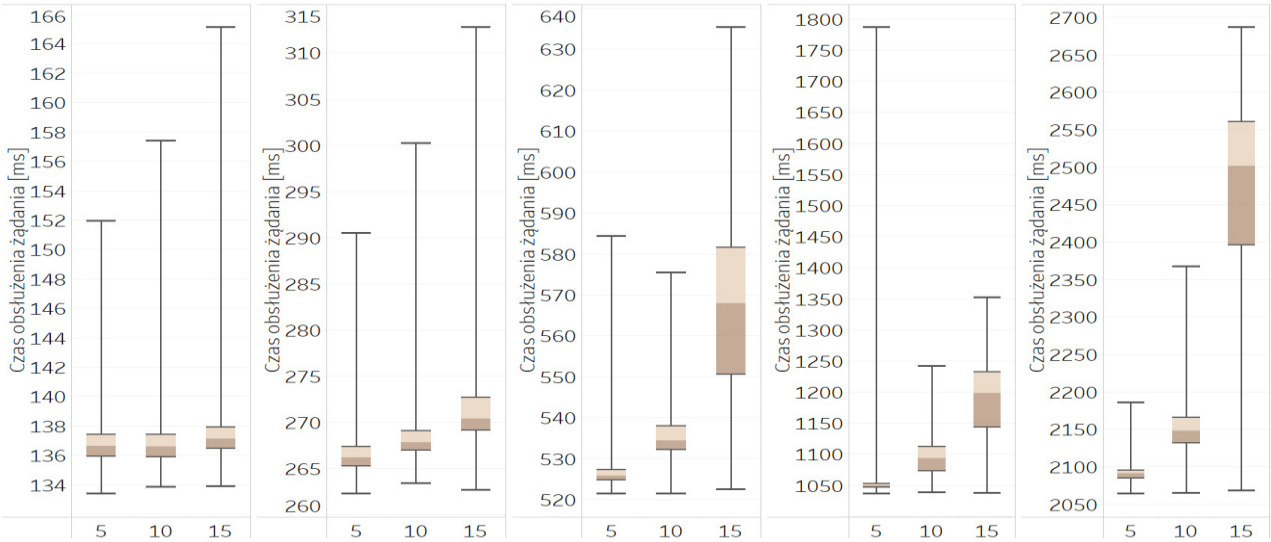
Tabela 6: Maksymalna liczba użytkowników wykonująca żądanie logowania możliwa do obsłużenia w danej technologii przy wykorzystaniu algorytmu Bcrypt do procesu autentykacji w zależności od ustawionych parametrów sprzętowych

Maksymalna liczba użytkowników		
Technologia / Parametry	Net 6	NestJS
4 procesory, 8GB RAM	1050	1300
4 procesory, 16GB RAM	1080	1320
6 procesorów, 8GB RAM	1600	1900
6 procesorów, 16GB RAM	1650	1920
8 procesorów, 8GB RAM	2020	2300
8 procesorów, 16GB RAM	2000	2260

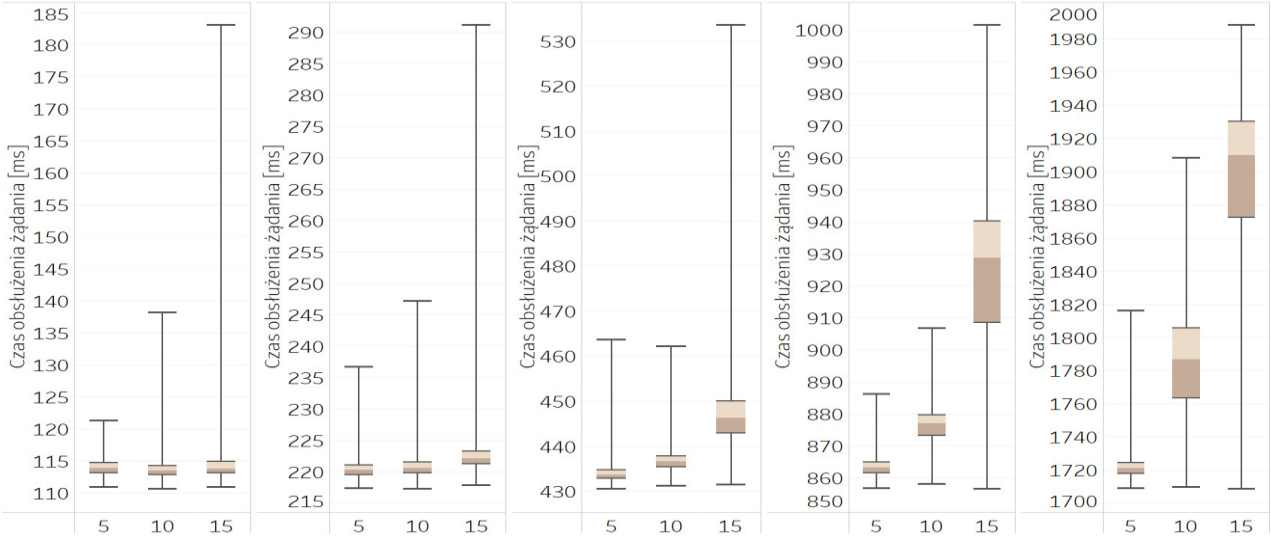
Tabela 7: Maksymalna liczba użytkowników wykonująca żądanie logowania możliwa do obsłużenia w danej technologii przy wykorzystaniu algorytmu Argon2id do procesu autentykacji w zależności od ustawionych parametrów sprzętowych

Maksymalna liczba użytkowników		
Technologia / Parametry	Net 6	NestJS
4 procesory, 8GB RAM	800	2200
4 procesory, 16GB RAM	830	2180
6 procesorów, 8GB RAM	1040	2970
6 procesorów, 16GB RAM	1000	3000
8 procesorów, 8GB RAM	1200	3180
8 procesorów, 16GB RAM	1220	3190

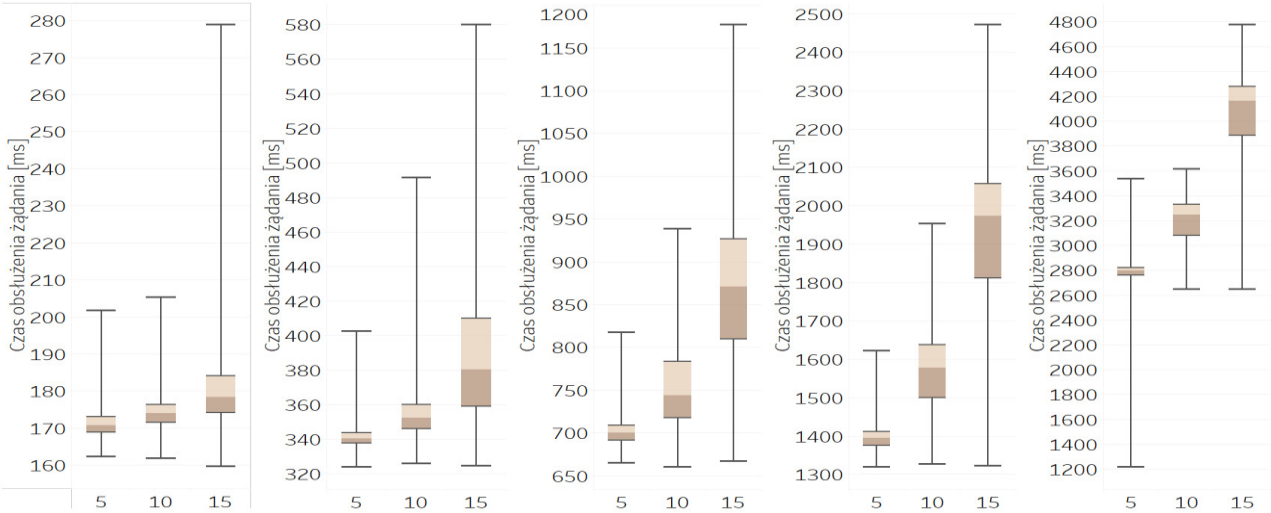
Analizując czasy logowania przedstawione na rysunkach 2-5 zauważono, iż niezależnie od użytego szkieletu programistycznego wraz ze wzrostem wartości współczynnika "work factor" w algorytmie Bcrypt jak i wraz ze zwiększeniem wartości parametru „memory” w algorytmie Argon2id czas obsłużenia żądania zwiększał się ok. dwukrotnie.



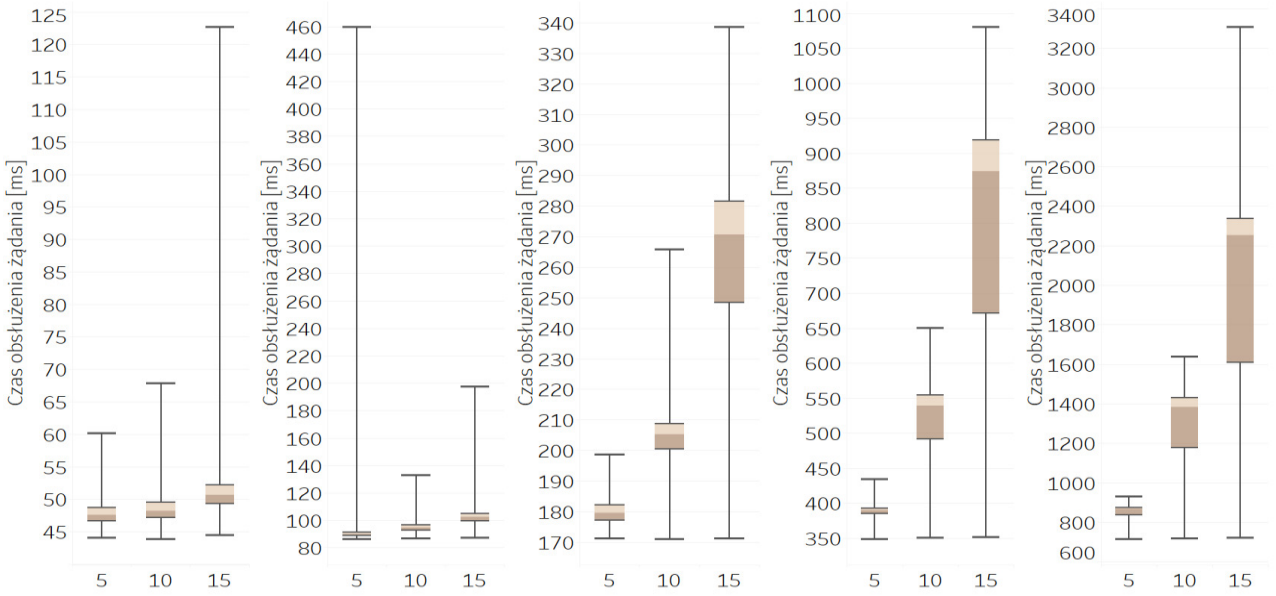
Rysunek 2: Wykresy czasów obsłużenia żądań logowania z aplikacji napisanej przy użyciu technologii Net 6 wykorzystującej algorytm Bcrypt w zależności od parametru „work factor" odpowiednio od lewej dla wartości równej 10, 11, 12, 13 i 14 dla 5, 10 i 15 użytkowników jednocześnie wykonujących żądania.



Rysunek 3: Wykresy czasów obsłużenia żądań logowania z aplikacji napisanej przy użyciu technologii NestJS wykorzystującej algorytm Bcrypt w zależności od parametru „work factor" odpowiednio od lewej dla wartości równej 10, 11, 12, 13 i 14 dla 5, 10 i 15 użytkowników jednocześnie wykonujących żądania.



Rysunek 4: Wykresy czasów obsłużenia żądań logowania z aplikacji napisanej przy użyciu technologii Net 6 wykorzystującej algorytm Argon2id w zależności od parametru „memory cost" wyrażonego w Megabajtach odpowiednio od lewej dla wartości równej 16, 32, 64, 128 i 256 dla 5, 10 i 15 użytkowników jednocześnie wykonujących żądania.



Rysunek 5: Wykresy czasów obsłużenia żądań logowania z aplikacji napisanej przy użyciu technologii NestJS wykorzystującej algorytm Argon2id w zależności od parametru „memory cost" wyrażonego w Megabajtach odpowiednio od lewej dla wartości równej 16, 32, 64, 128 i 256 dla 5, 10 i 15 użytkowników jednocześnie wykonujących żądania.

4. Wnioski

Dokonując przeglądu funkcjonalności oferowanych przez badane technologie stwierdzono, iż Net 6 ma ich do zaoferowania więcej. Domyślna implementacja była bardziej rozbudowana i dostarczała większej liczby metod przyspieszających pracę programisty. Implementacja wielu metod dostępnych w technologii NestJS była niewystarczająca, od programisty wymagało się samodzielnego zaimplementowania takich funkcjonalności jak generowanie skrótu kryptograficznego z hasła, bądź wprowadzenie polityki haseł. Do funkcjonalności, których zabrakło w NestJS, a były one dostępne w Net należało między innymi ukrywanie elementów interfejsu graficznego, tworzenie struktury bazy danych przechowujących informacje o użytkownikach i przyznanych im rolach. Na bazie uzyskanych wyników należało potwierdzić hipotezę badawczą mówiącą, iż technologia Net 6 oferuje większą liczbę funkcjonalności z zakresu autentykacji i autoryzacji użytkowników niż technologia NestJS. Analiza bibliotek programistycznych pokazała, iż w każdym z omawianych szkieletów było wiele rozwiązań, które cieszyły się popularnością wśród programistów, były na bieżąco aktualizowane, siła dostarczanych przez nie algorytmów jak i ich domyślna konfiguracja była wystarczająca by zapewnić wysoki poziom bezpieczeństwa w systemie. W każdym z badanych szkieletów znaleziono biblioteki implementujące algorytmy uznawane za bezpieczne i rekomendowane przez fundacje OWASP. Na bazie przeprowadzonych testów wydajnościowych stwierdzono, iż czas obsłużenia pojedynczego żądania był krótszy dla aplikacji napisanej w technologii NestJS niż w aplikacji wykorzystującej szkielet Net 6 niezależnie od algorytmów wykorzystanych do autentykacji i autoryzacji. Zauważono, iż niezależnie od użytego szkieletu programistycznego wraz ze wzrostem współczynnika "work factor" dla algorytmu Bcrypt jak i wraz ze zwiększeniem wartości parametru „memory" w algorytmie Argon2id czas obsłużenia żądania zwiększał się ok. dwukrotnie. Wraz ze wzrostem liczby użytkowników biorących udział w testach dane przyjmowały bardziej różniące się wartości i wydłużał się czas potrzebny na zakończenie pojedynczego żądania. Wyniki testów wydajnościowych pozwoliły potwierdzić hipotezę badawczą stanowiącą, iż średni czas obsłużenia żądania wysłanego do aplikacji napisanej przy wykorzystaniu szkieletu programistycznego NestJS jest krótszy niż w przypadku aplikacji wykorzystującej technologie Net 6. Ostatni z przeprowadzonych testów pokazał, iż serwis napisany w technologii NestJS był w stanie obsłużyć większą liczbę użytkowników niż ten sam serwis wykorzystujący szkielet Net 6. Zauważono, iż aplikacja napisana w Net 6, w której do procesu autentykacji wykorzystano algorytm Bcrypt, była w stanie obsłużyć większą liczbę użytkowników w porównaniu do konfiguracji z algorytmem Argon2id. Z kolei dla technologii NestJS wykorzystanie algorytmu Argon2id umożliwiło obsłużenie większej liczby użytkowników niż w przypadku konfiguracji z algorytmem Bcrypt. Należało potwierdzić hipotezę badawczą mówiącą, iż aplikacja napisana w technologii NestJS w ciągu 10 sekund jest w stanie obsłużyć większą liczbę żądań wysyłanych przez użytkowników niż serwis napisany przy użyciu szkieletu programistycznego Net 6.

Podsumowując, wyniki badań pokazały, iż nie można jednoznacznie stwierdzić, który ze szkieletów programistycznych jest bardziej przydatny do autoryzacji i autentykacji użytkowników. Biorąc pod uwagę szybkość obsłużenia pojedynczego żądania jak i maksymalną liczbę użytkowników, którą serwis może obsłużyć, godnym polecenia szkieletem będzie NestJS. Z kolei technologia Net 6 jest lepszym wyborem jeśli zależy nam na dużej liczbie oferowanych funkcjonalności oraz bibliotek programistycznych z dziedziny autentykacji i autoryzacji.

Literatura

[1] S. Tumin, S. Encheva, A Closer Look at Authentication and Authorization Mechanisms for Web-based Applications, Proceedings of the 5th WSEAS Congress on Applied Computing Conference, and Proceedings of the 1st International Conference on Biologically Inspired Computation (2012) 100-105.

[2] J. Lopez, R. Oppliger, G. Pernul, Authentication and authorization infrastructures (AAIs): a comparative survey, Computers & Security 23(7) (2004) 578-590, https://doi.org/10.1016/j.cose.2004.06.013.

[3] M. J. Price, C# 10 and .NET 6 - Modern Cross-Platform Development, Packt Publishing, 2021.

[4] G. Magolan, et.al., Nest.js: A Progressive Node.js Framework, Packt Publishing, 2022.

[5] C. Ntantogian, et.al., Evaluation of password hashing schemes in open source web platforms, Computers & Security 84 (2019) 206-224, https://doi.org/10.1016/j.cose.2019.03.011.

[6] L. Ertaul, et.al., Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms, Proceedings of the International Conference on Wireless Networks (ICWN) (2016) 66-72.

[7] N. Provos, D. Mazières, A Future-Adaptable Password Scheme, FREENIX Track: 1999 USENIX Annual Technical Conference Proceedings (1999) 81-92.

[8] C. Percival, S. Josefsson, The scrypt Password-Based Key Derivation Function, RFC 7914 (2016) 1-16, https://doi.org/10.17487/RFC7914.

[9] Ed. K. Moriarty, B. Kaliski, A. Rusch, PKCS #5: Password-Based Cryptography Specification Version 2.1, RFC 8018 (2017) 1-40, https://doi.org/10.17487/RFC8018.

[10] B. Pervan, J. Knezovic, K. Pericin, Distributed Password Hash Computation on Commodity Heterogeneous Programmable Platforms, 13th USENIX Workshop on Offensive Technologies (WOOT 19) (2019) 1-8.

[11] A. Dikanski, R. Steinegger, Identification and implementation of authentication and authorization patterns in the spring security framework, SECURWARE 2012 - 6th International Conference on Emerging Security Information, Systems and Technologies (2012) 14-20.

[12] M. Trnka, et.al., Systematic Review of Authentication and Authorization Advancements for the Internet of Things, Sensors 22(4) (2022) 1361-1385, https://doi.org/10.3390/s22041361.

[13] Internet rzeczy, definicja, https://en.wikipedia.org/wiki/Internet_of_things, [03.05.2022].

[14] Wikipedia - Role-based access control, https://en.wikipedia.org/wiki/Role-based_access_control, [22.09.2022].

[15] Wikipedia - Attribute-based access control, https://en.wikipedia.org/wiki/Attributebased_access_control, [19.09.2022].

[16] Oficjalna strona producenta maszyny wirtualnej Vmware, https://www.vmware.com/pl.html, [09.10.2022].

[17] S. Newman, Building Microservices, 2nd Edition, O'Reilly Media, 2021.

[18] K. Indrasiri, D. Kuruppu, gRPC: Up and Running, O'Reilly Media, 2020.

[19] Json Web Token, definicja, https://datatracker.ietf.org/doc/html/rfc7519, [03.05.2022].

[20] Token odświeżenia, definicja, https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/, [15.08.2022].

[21] Narzędzie do tworzenia testów wydajnościowych k6, https://k6.io, [03.05.2022].

[22] Informacje o programie Performance Monitor, https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc749154(v=ws.10), [03.05.2022].

[23] Definicja funkcji skrótu, https://en.wikipedia.org/wiki/Hash_function, [05.05.2022].

[24] A. Biryukov, et.al., Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications, 2016 IEEE European Symposium on Security and Privacy (EuroS&P) (2016) 292-302, https://www.doi.org/10.1109/EuroSP.2016.31.

[25] Główna strona internetowa Open Web Application Security Project, https://owasp.org, [22.06.2022].