JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 30 (2024) 26–30
Received: 25 September 2023
Accepted: 11 October 2023

Comparative analysis of NodeJs frameworks

Analiza porównawcza szkieletów programistycznych środowiska uruchomieniowego NodeJs

Bartłomiej Zima*, Marcin Barszcz

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

Abstract
The aim of the article is to compare two popular NodeJs frameworks. The analysis was performed for the ExpressJs and NestJs frameworks. Two proprietary applications supporting CRUD operations, implemented in these technologies, containing the same functionalities were used for the research. The comparison includes application performance, code metrics, documentation quality and completeness, and community support. The analysis showed that ExpressJs is minimalistic and suitable for less complex applications, while NestJs provides a standardized framework that allows the creation and development of large and complex projects.

Keywords: ExpressJs; NestJs; NodeJs; NodeJs frameworks

Streszczenie
Celem artykułu jest porównanie dwóch popularnych frameworków środowiska NodeJs. Analiza została przeprowadzona dla frameworków ExpressJs i NestJs. Do badań użyto dwóch autorskich aplikacji wspierających operacje typu CRUD, zaimplementowanych w tych technologiach, posiadających identyczne funkcjonalności. Porównanie obejmuje wydajność aplikacji, metryki kodu, jakość i kompletność dokumentacji oraz wsparcie społeczności. Analiza wykazała, że ExpressJs jest minimalistyczny i nadaje się do mniej złożonych aplikacji, podczas gdy NestJs zapewnia ustandaryzowaną strukturę pozwalającą na tworzenie i rozwijanie obszernych i skomplikowanych projektów.

Słowa kluczowe: ExpressJs; NestJs; NodeJs; szkielety programistyczne NodeJs

*Corresponding author
Email address: bartlomiej.zima@pollub-edu.pl (B. Zima)

1. Wstęp

Postęp w tworzeniu aplikacji internetowych, począwszy od momentu, gdy statyczne strony stworzone wyłącznie w języku HTML zaczęły rozwijać się w kierunku większej interaktywności z użytkownikiem, jest efektem wprowadzenia na rynek języka JavaScript,

Język ten powstał w 1995 roku i funkcjonował jako język skryptowy, działający po stronie klienta, dla przeglądarek internetowych [1], pozwalający na tworzenie interaktywnych stron. Elementami interaktywności były przykładowo walidowanie formularzy lub manipulowanie elementami HTML obecnymi na stronie. Z biegiem czasu wykonywanie wyłącznie synchronicznych operacji na stronach stało się kosztowne i znacznie spowalniające wydajność. W 2009 roku frustracja na rynku wywołana niezdolnością przetworzenia równoczesnych żądań liczonych w tysiącach osiągnęła takie rozmiary, że podjęto decyzję o znalezieniu rozwiązania [1].

Wówczas powstał NodeJs, czyli środowisko uruchomieniowe umożliwiające wykorzystywanie języka Javascript i Typescript (rozszerzenie JavaScript z wprowadzonymi typami danych) poza przeglądarką internetową. Technologia ta znacząco zwiększyła wydajność serwera wykorzystując pętlę zdarzeń (ang. Event loop) i operacje asynchroniczne wejścia i wyjścia, które służą do obsługi wielu zdarzeń jednocześnie [1].

Podobnie jak większość obecnie używanych technologii tego typu, NodeJs również doczekał się własnych szkieletów programistycznych (ang. Frameworks), pozwalających na składne i zorganizowane operowanie nim po stronie serwera. Opisywane frameworki mają następujące cechy i zastosowanie:

- ExpressJs – framework, powstały w 2010 roku w którym używany jest język Javascript. Oferuje zestaw funkcjonalności typu routing i oprogramowanie pośrednie Jest to framework elastyczny, nienarzucający ścisłych konwencji programistycznych, co daje dużą swobodę programistom. Jest podstawą dla wielu innych frameworków, włącznie z drugim poddanym analizie w niniejszym artykule – NestJs.
- NestJs – szkielet programistyczny, opracowany w 2017 roku który używa języka Typescript. W przeciwieństwie do języka Javascript, język ten pozwala na wykrycie błędów wynikających z typów danych już podczas kompilacji, a nie dopiero w trakcie działania programu. Ponadto NesJs posiada ściśle zdefiniowaną strukturę, inspirowaną technologią AngularJs [2]. Zawiera ona wbudowane wstrzykiwanie zależności (ang. Dependency injection), które pozwala na łatwiejsze zarządzanie zależnościami między modułami aplikacji. Sprawia to, że kod staje się łatwiejszy w utrzymaniu, testowaniu itp.

Celem artykułu jest przeprowadzenie analizy porównawczej tych dwóch frameworków, ze szczególnym

uwzględnieniem ich wydajności, jakości kodu, kompletności dokumentacji oraz wsparcia społeczności.

2. Metodyka i obiekt badań

W celu przeprowadzenia analizy porównawczej rozważanych w artykule frameworków opracowano dwie zaimplementowane od nowa aplikacje posiadające takie same funkcjonalności. Zwrócono szczególną uwagę na minimalizację wykorzystania bibliotek zewnętrznych, co podnosi miarodajność porównania. Podstawowymi elementami aplikacji były operacje typu CRUD. Do realizacji autoryzacji wykorzystano bibliotekę obsługującą JsonWebToken oraz bazę danych PostgreSQL.

Podczas implementacji aplikacji wykorzystano najpopularniejsze rozwiązania dostępne w obu frameworkach. Aby dokładniej zbadać wydajność aplikacji, ograniczono ilość kodu do minimum koniecznego do prawidłowego przetworzenia żądań. W przypadku autoryzacji wykorzystano nagłówek autoryzacyjny http.

Analizę porównawczą przeprowadzono na podstawie szeregu istotnych kryteriów, które stanowią kluczowe elementy oceny szkieletów programistycznych w środowisku uruchomieniowym NodeJs. Przeprowadzone badania koncentrowały się na następujących aspektach:

- Metryki kodu aplikacji – oceniono ilość linii kodu oraz plików niezbędnych do wykonania określonej operacji, zgodnie z potencjalną strukturą architektoniczną. Analiza tej metryki pozwoliła na ocenę stopnia skomplikowania kodu w poszczególnych szkieletach programistycznych.
- Czas i wydajność przetwarzania żądań http – przeprowadzono badania mające na celu zbadanie czasu odpowiedzi serwera dla aplikacji w różnych warunkach obciążenia, zarówno przy niewielkim, jak i znacznym obciążeniu. Analiza ta pozwoliła na ocenę wydajności poszczególnych szkieletów w kontekście obsługi żądań http. W celu oceny wydajności przetwarzania żądań http zastosowano narzędzie Apache JMeter, umożliwiające symulację różnych stopni obciążenia aplikacji.
- Jakość i kompletność dokumentacji – dokonano analizy oficjalnych dokumentacji frameworków pod kątem ich organizacji, przejrzystości i szczegółowości. Dokumentacja stanowi kluczowe źródło wiedzy podczas implementacji aplikacji, dlatego jej jakość ma istotny wpływ na proces programowania.
- Popularność i dostępność porad wśród społeczności – badania w zakresie popularności technologii zostały zrealizowane na podstawie statystyk z popularnych portali technologicznych. Popularność i dostępność porad wśród społeczności związane są z rozwojem Internetu i technologii używanych do tworzenia aplikacji. W wyniku tego rozwoju powstały społeczności skupiające entuzjastów, gdzie ludzie dzielą się wiedzą, pomagają sobie wzajemnie oraz dyskutują na temat napotkanych problemów technicznych, starając się je rozwiązać.

Wnikliwa analiza tych kryteriów pozwoliła na pełniejsze zrozumienie i ocenę szkieletów programistycznych w kontekście tworzenia aplikacji w środowisku NodeJs.

W niniejszych badaniach wykorzystano szereg kluczowych narzędzi i technologii, które odegrały istotną rolę w przeprowadzeniu analizy i testowania. Poniżej przedstawiono szczegółowe informacje na temat użytych wersji oprogramowania oraz parametrów sprzętu badawczego.

Wykorzystane wersje technologii:

- NodeJs: Wersja 18.12.1 – popularne środowisko uruchomieniowe oparte na silniku V8 JavaScript, stanowiło fundament aplikacji, umożliwiając dynamiczną i wydajną obsługę żądań.
- NestJs: Wersja 9.3.0 – to framework dla NodeJs, zapewnił skalowalność oraz strukturalność projektu, ułatwiając rozwój aplikacji poprzez zastosowanie wzorca architektonicznego.
- ExpressJs: Wersja 4.8.12 – to framework dla NodeJs, który posłużył do szybkiego tworzenia interfejsu API oraz zarządzania ścieżkami żądań.
- JMeter: Wersja 5.4.1 – to narzędzie do testowania wydajności aplikacji, było kluczowe w analizie obciążenia systemu oraz ocenie jego responsywności.
- PostgreSQL: Wersja 13.5 – to zaawansowany system zarządzania bazami danych, posłużył jako główna baza danych do przechowywania i zarządzania danymi aplikacji.

Parametry sprzętu badawczego:

- Rodzaj platformy: Komputer stacjonarny – został wybrany jako platforma badawcza, zapewniająca stabilność i wydajność podczas testowania aplikacji.
- Procesor: 13th Gen Intel(R) Core(TM) i7-13700K – wykorzystany procesor 13. generacji firmy Intel, charakteryzujący się wysoką mocą obliczeniową, był kluczowym elementem umożliwiającym szybkie przetwarzanie danych i zapytań.
- Pamięć RAM: 32GB – duża ilość pamięci RAM pozwoliła na płynne działanie aplikacji oraz manipulację dużymi ilościami danych podczas testów.
- System operacyjny: Windows 10 – wykorzystany system operacyjny Windows 10, popularny i szeroko używany, zapewnił stabilne środowisko pracy do przeprowadzenia eksperymentów i testów.

3. Dyskusja i analiza wyników

W poniższym rozdziale dokonano analizy zagadnień podjętych w niniejszym badaniu. Przedstawiono konteksty badawcze, w jakich omawiane szkielety programistyczne operują w środowisku uruchomieniowym NodeJs. Przeanalizowano również rezultaty przeprowadzonych badań w kontekście określonych kryteriów, dając pełny wgląd w porównawczą ocenę tychże szkieletów. Dokładnie przeanalizowano wpływ różnych czynników na wydajność i skalowalność i złożoność implementacji, a także zidentyfikowano mocne i słabe strony każdego szkieletu. Zawarto także omówienie

kwestii związanych z praktycznym wykorzystaniem szkieletów w projekcie aplikacji, uwzględniając aspekty takie jak łatwość użycia, społeczność wsparcia i dostępność dokumentacji. Analiza ta stanowi kluczowy etap, umożliwiający pełne zrozumienie i wykorzystanie wyników badań w kontekście praktycznego zastosowania szkieletów programistycznych w projektach oparty na NodeJs.

3.1 Metryki kodu aplikacji

NestJs jest szkieletem opartym na architekturze modułowej, co oznacza że aplikacja zostaje podzielona na niezależne moduły, z których każdy zawiera zbiór powiązanych komponentów, jak na przykład kontrolery, repozytoria lub modele. Taka struktura architektoniczna zapewnia skalowalność, ułatwia utrzymanie kodu i uniezależnia fragmenty systemu od siebie [3]. Z kolei ExpressJs nie narzuca żadnej konkretnej struktury aplikacji, dając deweloperowi swobodę projektowania architektury zgodnie z indywidualnymi preferencjami i potrzebami. Jednakże, w przypadku wykorzystania tego frameworka w większym projekcie, zaleca się zastosowanie jednego z wzorców architektonicznych, w celu zapewnienia łatwiejszej skalowalności [3].

Porównując kod aplikacji odpowiedzialny za obsługę żądania, w którym wymagane jest połączenie z bazą danych, w praktyce zrealizowano go na podstawie funkcji pobierającej wszystkie rekordy z tabeli w bazie danych.

W aplikacji ExpressJs, do pobrania danych z bazy PostgreSQL w praktyce wystarczy otworzyć połączenie do niej, podając odpowiednią konfigurację w pliku przetwarzającym żądanie http, wykonać zapytanie SQL i następnie zwrócić otrzymane dane użytkownikowi. Natomiast w NestJs potrzebne jest kilka dodatkowych kroków. Ze względu na silne typowanie tego frameworka, oprócz samej biblioteki do obsługi PostgreSQL, trzeba zainstalować bibliotekę umożliwiającą mapowanie obiektów na struktury bazodanowe (ang. Object-Relational-Mapping) [4]. Należy także zdefiniować konkretne encje w kodzie aplikacji, oznaczone adnotacjami @Entity(). Pola encji, jeśli nie stosujemy dodatkowej walidacji danych, należy oznaczyć adnotacjami @Column() i @PrimaryGeneratedColumn(). Oznaczają one odpowiednio zwykłe kolumny w bazie danych oraz kolumnę będącą kluczem głównym tabeli. Po nawiązaniu połączenia z bazą danych, repozytorium, operujące na danych zdefiniowanej encji, zostaje wstrzyknięte do serwisu. Ten serwis z kolei zostaje wstrzyknięty do odpowiedniego kontrolera, który obsługuje żądania HTTP. Kontroler użyje konkretnych funkcji serwisu, aby pobrać dane i następnie zwrócić je użytkownikowi.

Tabela 1 pokazuje liczbową różnicę wymaganych do implementacji jednej kompletnej operacji zasobów pomiędzy frameworkami. Aby zrealizować taką implementację w przypadku NestJs potrzebna jest względnie dużo większa ilość plików i linii kodu.

Tabela 1: Porównanie wymaganej ilości zasobów dla operacji GET na bazie danych

	Express	NestJs
Liczba plików	1	5
Łączna liczba linii kodu	16	53

3.2 Czas i wydajność przetwarzania żądań http

W celu zbadania średnich czasów przetwarzania niezautoryzowanych żądań poszczególnych metod http w symulacji jednego użytkownika, przeprowadzono 100 prób. Wyniki przedstawiono w Tabeli 2.

Tabela 2: Porównanie uśrednionych czasów odpowiedzi serwera dla niezautoryzowanych żądań typu CRUD

	GET	POST	PUT	DELETE
ExpressJs	37 ms	5 ms	14 ms	9 ms
NestJs	40 ms	11 ms	8 ms	40 ms

Porównując ExpressJs z NestJs, można zauważyć, że ExpressJs jest w stanie szybciej zwrócić odpowiedź w przypadku metod GET, POST, i DELETE, jednakże, jeżeli chodzi o metodę PUT, różnica w milisekundach między ExpressJs, a NestJs jest względnie największa. Tabela 3 przedstawia wyniki podobnych operacji, w tym przypadku żądania są autoryzowane.

Tabela 3: Porównanie uśrednionych czasów odpowiedzi serwera dla zautoryzowanych żądań typu CRUD

	GET	POST	PUT	DELETE
ExpressJs	24 ms	8 ms	15 ms	11 ms
NestJs	40 ms	14 ms	11 ms	44 ms

Żądania wymagające autoryzacji posiadają w większości dłuższe czasy odpowiedzi niż poprzednie, ale różnice pozostały relatywnie proporcjonalne. Można z tego wywnioskować, że w przypadku pojedynczych żądań, konieczność przetworzenia nagłówka autoryzacyjnego z JsonWebToken, nie wpływa znacznie na różnice czasu oczekiwania na odpowiedź serwera między frameworkami.

Przeprowadzono również szczegółowe testy wydajnościowe aplikacji, z podziałem na zautoryzowane i niezautoryzowane żądania http. Zasymulowano jednoczesne wysyłanie żądań łącznie dla 120 000 użytkowników w próbach po 40 000 każda. Uśrednione czasy odpowiedzi serwera, jak również procent żądań zakończonych niepowodzeniem zostały zawarte w Tabelach 4 i 5.

Tabela 4: Porównanie wyników testów obciążeniowych dla niezautoryzowanych metod typu CRUD

	ExpressJs		**NestJs**	
Metoda	Średni czas	% błędu	Średni czas	% błędu
GET	934 ms	22.5	1167 ms	28.75
POST	1056 ms	21.5	1211 ms	58.76
PUT	655 ms	65.40	327 ms	80.12
DELETE	610 ms	45.67	487 ms	97.37

Tabela 5: Porównanie wyników testów obciążeniowych dla zautoryzowanych metod typu CRUD

	ExpressJs		**NestJs**	
Metoda	Średni czas	% błędu	Średni czas	%błędu
GET	844 ms	30.48	221 ms	85.27
POST	19 ms	87.03	278 ms	86.6
PUT	838 ms	40.85	170 ms	87.22
DELETE	725 ms	42.45	318 ms	98.15

Tabele 4 i 5 wyraźnie pokazują, że ExpressJs jest bardziej odporny na obciążenia. Mimo wielu przypadków wyższego czasu oczekiwania na odpowiedź niż NestJs, potrafił procentowo przetworzyć znacznie więcej żądań od niego.

Co do NestJs można powiedzieć, że w przypadkach wymagających autoryzacji, framework jest wyjątkowo nieprzystosowany do dużego obciążenia.

Należy jednak pamiętać, że różnice w czasie odpowiedzi mierzonej w milisekundach, nawet jeśli wydają się duże między szkieletami programistycznymi, są praktycznie nieodczuwalne dla użytkownika końcowego.

3.3 Jakość i kompletność dokumentacji

W kontekście tworzenia aplikacji internetowych, poprawnie opracowana dokumentacja używanej technologii jest kluczowym elementem pomagającym zrozumieć używane funkcje i poprawnie je zaimplementować. Oba frameworki posiadają obszerne dokumentacje, które stanowią podstawowe źródło informacji i przewodników w procesie tworzenia systemu.

W przypadku NestJs dokumentacja jest bardzo szczegółowa i zorganizowana w sposób łatwy do nawigacji [4]. Została podzielona na sekcje tematyczne i zawiera przykłady użycia, co pomaga deweloperowi zrozumieć wykorzystanie danych funkcji w praktyce. Obejmuje szeroki zakres tematów, takich jak routing, autoryzacja i walidacja danych. Dokumentacja ta jest regularnie aktualizowana wraz z wprowadzaniem nowych wersji tego frameworka.

Dokumentacja ExpressJs również jest obszerna [5] i opisuje podstawowe koncepcje i funkcje frameworka. Jest łatwa do zrozumienia i zawiera przykłady kodu oraz opisuje różne techniki i wzorce, które można zastosować przy implementacji. Niemniej jednak dokumentacja ta jest mniej szczegółowa niż dokumentacja NestJs, więc znalezienie konkretnych informacji może zająć programiście więcej czasu.

Warto zaznaczyć, że oba szkielety programistyczne posiadają dokumentację dobrej jakości. NestJs wyróżnia się jako szkielet oferujący bardziej szczegółowe i kompleksowe informacje, podczas gdy ExpressJs dostarcza dokumentację, która jest łatwa do zrozumienia i zapewnia podstawowe informacje o frameworku. Co w przypadku minimalistycznego frameworka, można uznać za wystarczające. Dodatkowo w dokumentacji ExpresssJs można znaleźć wyjaśnienia pojęć którymi operuje, niekiedy również z graficznym zobrazowaniem i wyjaśnieniem procesów.

3.4 Popularność i dostępność porad wśród społeczności

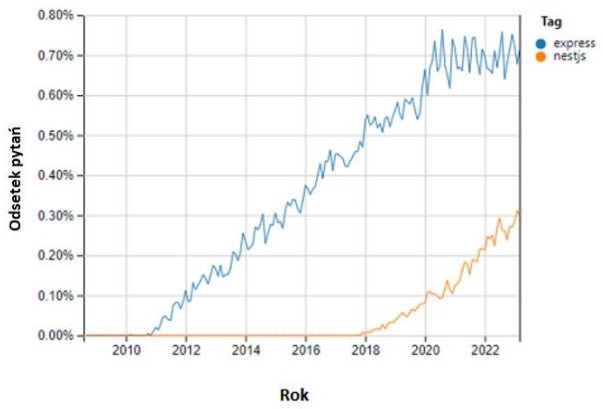
Do oceny popularności szkieletów programistycznych wykorzystano dane liczbowe zebrane ze społeczności programistycznej na platformie StackOverflow [6]. Jest to platforma działająca jak forum dyskusyjne, na której programiści zadają pytania dotyczące problemów technicznych, dzielą się wiedzą z innymi oraz uczestniczą w społeczności programistycznej. StackOverflow wykorzystuje tagi do kategoryzacji pytań, umożliwiając również wyszukiwanie konkretnych tagów i uzyskiwanie informacji o liczbie związanych z nimi wątków. Badania zostały przeprowadzone dnia 03.05.2023

- NestJs – tag [nestjs] posiada 11034 wątków, z czego 10 powstało w dniu przeprowadzenia badań [7].
- ExpressJs – tag [express] posiada 92969 wątków, przy czym 23 powstały w dniu przeprowadzania badań [8].

Ponadto istotnym wskaźnikiem jest liczba repozytoriów na platformie Github, która pełni funkcję bazy milionów projektów aplikacji internetowych. Platforma ta umożliwia również kategoryzację repozytoriów według technologii:

- NestJs – liczba repozytoriów oznaczonych tą technologią wynosiła 62 903 [9] w dniu przeprowadzenia badań
- ExpressJs – liczba repozytoriów oznaczonych tą technologią wynosiła 557 907 [10] w dniu przeprowadzenia badań

Rysunek 1 przedstawia wykres odsetka zadawanych pytań na StackOverflow dotyczących analizowanych frameworków. Obserwacje pokazują, że dla ExpressJs od roku 2010 do 2020 odsetek ten wyraźnie wzrastał. Jednak po roku 2020 zaczęły występować oscylacje między tendencją wzrostową a spadkową. Dla NestJs natomiast, od momentu jego wprowadzenia, trend ten niepodważalnie ukazuje wzrost.



Rysunek 1: Wykres procentowy zadawanych pytań na przestrzeni lat na platformie StackOverflow.

4. Wnioski

Po przeprowadzonej analizie porównawczej dwóch szkieletów programistycznych, można stwierdzić, że ExpressJs wyróżnia prostota i minimalistyczne podejście, co czyni go dobrym wyborem dla aplikacji o niskim stopniu skomplikowania. NestJs natomiast cechuje się bardziej strukturalnym podejściem opartym na architekturze modułowej, co dobrze sprawdzi się w bardziej złożonych projektach, zapewniając im lepszą skalowalność i łatwiejsze utrzymanie. Pomimo faktu, że wymaga on większej ilości zasobów takich jak czas programistów czy pamięć na dysku, do stworzenia jednej funkcjonalności niż ExpressJs, zapewnia im zorganizowaną formę i czyni kod bardziej zrozumiałym.

NestJs wyróżnia bogata i szczegółowa dokumentacja. Oferuje dużo informacji na temat wszystkich aspektów frameworka, zawiera przykłady kodu ułatwiające zrozumienie i implementację. ExpressJs również posiada czytelną dokumentację, jednakże nie tak szczegółową jak Nest.

Analizując liczbę wątków na dwóch popularnych platformach StackOverflow i Github, można stwierdzić że ExpressJs jest technologią bardziej popularną niż NestJs. Może to wynikać z faktu że został opracowany około 7 lat wcześniej [4]. Wykres przedstawiony na rysunku 1 pokazuje jednak, że NestJs zyskuje coraz większą popularność.

W przypadku dużego obciążenia aplikacji w jednym momencie, ExpressJs wykazuje się większą niezawodnością. Pomimo, że NestJs w wielu przypadkach był w stanie przetworzyć żądania znacznie szybciej, to wysoki procent żądań zakończonych niepowodzeniem sprawia, że w systemach narażonych na duże obciążenie nie jest on optymalnym wyborem.

Pod względem symulacji żądań pojedynczego użytkownika, frameworki nie różnią się znacząco. Jedynie w przypadku metody DELETE wyniki wykazują względnie największą różnicę. Warto zaznaczyć jednak, że w praktycznym użytkowaniu aplikacji te różnice są na tyle marginalne, że użytkownik końcowy nie odczuje ich znacząco.

Literatura

[1] R. Kempf, History of JavaScript, 2021 https://www.azion.com/en/blog/history-of-javascript/ , [06.05.2023]

[2] R. Benita, Clean Node.js Architecture – With NestJs and Typescript, https://betterprogramming.pub/clean-node-js-architecture-with-nestjs-and-typescript-34b9398d790f , [06.05.2023]

[3] S. Pasquali, Node.js Projektowanie, wdrażanie i utrzymywanie aplikacji, Helion, 2017

[4] Oficjalna dokumentacja technologii NestJs, https://docs.nestjs.com , [06.05.2023]

[5] Oficjalna dokumentacja technologii ExpressJs, https://expressjs.com , [06.05.2023]

[6] Oficjalna strona platformy StackOverflow: https://stackoverflow.com/ , [06.05.2023]

[7] Pytania otagowane słowem kluczowym „nestjs” na platformie StackOverflow, https://stackoverflow.com/questions/tagged/nestjs , [03.05.2023]

[8] Pytania otagowane słowem kluczowym „express” na platformie StackOverflow, https://stackoverflow.com/questions/tagged/express , [03.05.2023]

[9] Repozytoria NestJs na platformie Github, https://github.com/search?q=nestjs&type=repositories , [03.05.2023]

[10] Repozytoria ExpressJs na platformie Github, https://github.com/search?q=express&type=repositories&p=2 , [03.05.2023]