JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 33 (2024) 251–257
Received: 28 May 2024
Accepted: 3 September 2024

Comparative analysis of performance between .NET platform versions
Analiza porównawcza wydajności pomiędzy wersjami platformy .NET

Grzegorz Grzegorczyk*, Małgorzata Plechawska-Wójcik

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

Abstract
The article analyzes the performance of traditional API controllers compared to MinimalAPI across various .NET platform versions, from 3.1 to 9 preview 3. The primary objective was to determine which technology offers superior performance in handling simple HTTP requests. For the analysis, a single application was developed in two variants: one using traditional API controllers and the other using MinimalAPI. This study provides significant insights into the efficiency of both approaches for handling HTTP requests on different .NET versions, contributing to a better understanding of their performance.

Keywords: .NET; MVC; Minimal API; HTTP; REST; performance

Streszczenie
Artykuł analizuje wydajność tradycyjnych kontrolerów API w porównaniu do MinimalAPI w różnych wersjach platformy .NET, od 3.1 do 9 preview 3. Głównym celem było określenie, która technologia oferuje lepszą wydajność w obsłudze prostych zapytań HTTP. W celu przeprowadzenia analizy opracowano jedną aplikację w dwóch wariantach: jeden korzystający z tradycyjnych kontrolerów API, a drugi z MinimalAPI. Badanie dostarcza istotnych informacji na temat efektywności obu podejść w kontekście obsługi zapytań HTTP na różnych wersjach .NET, przyczyniając się do lepszego zrozumienia ich wydajności.

Słowa kluczowe: .NET; MVC; Minimal API; HTTP; REST; performance

*Corresponding author
Email address: **grzegorz.grzegorczyk@pollub.edu.pl** (G. Grzegorczyk)

1. Wstęp

Początkowo, wraz z wprowadzeniem ASP.NET MVC, kontrolery stały się kluczowym elementem obsługi zapytań HTTP. Wraz z pojawieniem się ASP.NET Core, nastąpił znaczący krok naprzód. Nowy szkielet dostarczył bardziej elastyczną i wydajną architekturę, umożliwiającą tworzenie lżejszych i efektywnych kontrolerów API. Wprowadzenie Minimal API było kolejnym kamieniem milowym, oferując bardziej zwięzłe podejście do tworzenia interfejsów API bez zbędnego nakładu kodu.

Na wydajność aplikacji może mieć wpływ używany system operacyjny. Linux przewyższa Windows w operacjach na plikach tekstowych i CRUD, podczas gdy Windows jest bardziej efektywny w operacjach na plikach graficznych [1]. Kolejną warstwą mającą wpływ na wydajność może mieć wybrany ORM [2].

Badania wskazują, że aplikacje .NET mają lepszy czas ładowania strony, ale PHP ma wyższą prędkość transferu żądań i odpowiedzi [3]. Wyniki sugerują, że wybór technologii zależy od specyficznych potrzeb projektu: .NET jest lepszy dla szybszego ładowania stron, a PHP dla szybszego przetwarzania żądań. Natomiast [4] pokazuje, że Java EE ma krótsze czasy odpowiedzi i mniejsze zużycie pamięci niż .NET, co sugeruje, że Java EE jest lepsza dla systemów wymagających wysokiej wydajności i niskiego zużycia zasobów.

Zastosowanie technologii .NET i architektury MVC poprawia wydajność, skalowalność i łatwość zarządzania systemem kursów multimedialnych na żądanie [5].

Podczas projektowania API wykazano, że wytyczne takie jak używanie bezstanowości i formatu JSON do przesyłania danych są powszechnie akceptowane, podczas gdy stosowanie HATEOAS jest często odrzucane z powodu praktycznych trudności. Projektanci napotykają na wyzwania związane z implementacją autoryzacji i uwierzytelniania [6], co może prowadzić do utrat wydajności aplikacji.

Badania [7] wykazały, że HTTP/2.0 znacząco poprawia wydajność aplikacji dzięki multipleksowaniu i kompresji nagłówków, co jest szczególnie korzystne dla platformy .NET. Wyniki [8] wskazują, że HTTP/3, dzięki zastosowaniu protokołu QUIC, oferuje lepszą wydajność niż HTTP/2 w warunkach wysokiej latencji, co może sugerować przewagę nowszych wersji .NET w odpowiednich scenariuszach sieciowych. Badania [9] podkreślają, że HTTP z utrzymywaniem połączeń (persistent TCP) znacząco poprawia wydajność w porównaniu do standardowego TCP, zwłaszcza w sieciach o dużej przepustowości. To sugeruje, że nowoczesne wersje .NET powinny lepiej integrować te zaawansowane funkcje transportowe. Z kolei że optymalizacja serwerów cache HTTP poprzez eliminację zbędnych żądań za pomocą kodów odpowiedzi HTTP 304 i 404 znacząco poprawia wydajność i zmniejsza obciążenie serwerów źródłowych. Nowoczesne wersje platformy .NET, integrujące

zaawansowane mechanizmy zarządzania zasobami i optymalizacje sieciowe, mogą oferować lepszą wydajność w aplikacjach streamingowych na żywo. Implementacja tych optymalizacji może prowadzić do efektywniejszego wykorzystania zasobów i lepszej skalowalności systemów opartych na .NET [10]. Optymalizacja zarządzania ruchem HTTP i sesjami może poprawić wydajność aplikacji [11].

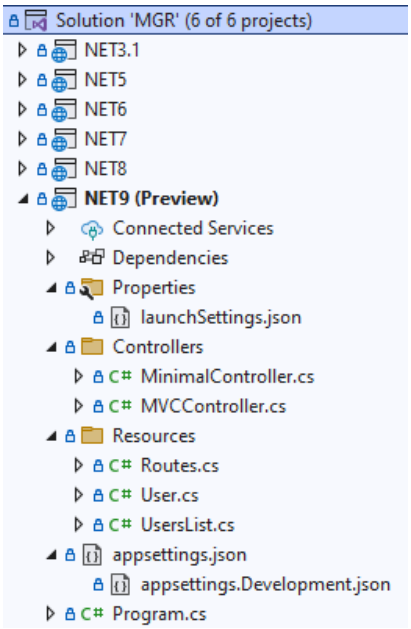
Celem badań jest zbadanie wydajności platformy .NET w wersjach od 3.1 do 9 preview 3 w liczbie obsłużonych żądań HTTP.

W tym celu pod uwagę wzięto kryterium liczby obsłużonych zapytań HTTP przez kontrolery MVC i Minimal API w z góry określonym czasie.

W artykule postawiono hipotezę badawczą, że technologia Minimal API jest wydajniejsza niż tradycyjne kontrolery MVC dla prostych zapytań HTTP. Zostanie to udowodnione za pomocą wyżej wymienionego kryterium.

2. Opis testowanej aplikacji

Struktura testowanej aplikacji serwerowej, która składa się z sześciu mniejszych aplikacji dla każdej wersji platformy .NET, została przedstawiona na Rysunku 1.



Rysunek 1: Struktura projektu.

Projekty .NET Core 3.1 i .NET 5 zawierają tylko implementację zapytań HTTP takich jak GET, POST, PUT, DELETE w technologii kontrolerów MVC a pozostałe (tj. .NET 6, 7, 8, 9 preview 3) dodatkowo implementację tych samych żądań w technologii Minimal API.

2.1. Obiekt User

Klasa *User* implementuje obiekt, na którym będą prowadzone scenariusze badania. Kod został przedstawiony na Listingu 1.

Listing 1: Definicja obiektu User

```
namespace NET9.Resources
{
    18 references
    public class User
    {
        18 references
        public Guid UserId { get; set; }
        14 references
        public string? FirstName { get; set; }
        14 references
        public string? LastName { get; set; }
        14 references
        public string? Email { get; set; }
        14 references
        public string? PasswordHash { get; set; }
        14 references
        public string? PhoneNumber { get; set; }
        14 references
        public string? StreetAddress { get; set; }
        14 references
        public string? ApartmentNumber { get; set; }
        14 references
        public string? City { get; set; }
        14 references
        public string? PostalCode { get; set; }
        14 references
        public string? Country { get; set; }
        14 references
        public DateTime DateOfBirth { get; set; }
        14 references
        public bool Gender { get; set; }
        14 references
        public DateTime CreatedAt { get; set; }
        14 references
        public DateTime? UpdatedAt { get; set; }
    }
}
```

2.2. Metoda GET

Metoda GET ma za zadanie pobranie listy użytkowników. Na Listingu 2 została przedstawiona implementacja w MVC a na Listingu 3 w Minimal API.

Listing 2: Implementacja żądania GET w MVC

```
[HttpGet]
[Route("get")]
1 reference
public ActionResult<List<User>> Get()
{
    var usersList = new UsersList();
    return usersList.Items;
}
```

Listing 3: Implementacja żądania GET w Minimal API

```
app.MapGet(Routes.Minimal + "get", () =>
{
    var usersList = new UsersList();
    return Results.Ok(usersList.Items);
});
```

2.3. Metoda POST

Metoda POST ma za zadanie utworzenie nowego obiektu użytkownika w aplikacji. Na Listingu 4 przedstawiono implementację w MVC a na Listingu 5 w Minimal API.

Listing 4: Implementacja żądania POST w MVC

```
[HttpPost]
[Route("post")]
0 references
public ActionResult<User> Post(User user)
{
    var usersList = new UsersList();
    user.UserId = Guid.NewGuid();
    usersList.Items.Add(user);
    return CreatedAtAction(nameof(Get), new { id = user.UserId }, user);
}
```

Listing 5: Implementacja żądania POST w Minimal API

```
app.MapPost(Routes.Minimal + "post", (User user) =>
{
    var usersList = new UsersList();
    user.UserId = Guid.NewGuid();
    usersList.Items.Add(user);
    return Results.Created($"{user.UserId}", user);
});
```

2.4. Metoda PUT

Metoda PUT ma za zadanie zmodyfikowanie jednej z wcześniej zdefiniowanych wartości. Na Listingu 6 jest zaprezentowana implementacja w MVC a na Listingu 7 w Minimal API.

Listing 6: Implementacja żądania PUT w MVC

```
[HttpPut]
[Route("put/{id}")]
0 references
public IActionResult Put(Guid id, User user)
{
    var usersList = new UsersList();
    var existingUser = usersList.Items.FirstOrDefault(x => x.UserId == id);
    if (existingUser == null) return NotFound();

    existingUser.FirstName = user.FirstName;
    existingUser.LastName = user.LastName;
    existingUser.Email = user.Email;
    existingUser.PasswordHash = user.PasswordHash;
    existingUser.PhoneNumber = user.PhoneNumber;
    existingUser.StreetAddress = user.StreetAddress;
    existingUser.ApartmentNumber = user.ApartmentNumber;
    existingUser.City = user.City;
    existingUser.PostalCode = user.PostalCode;
    existingUser.Country = user.Country;
    existingUser.DateOfBirth = user.DateOfBirth;
    existingUser.Gender = user.Gender;
    existingUser.CreatedAt = user.CreatedAt;
    existingUser.UpdatedAt = user.UpdatedAt;

    return NoContent();
}
```

Listing 7: Implementacja żądania PUT w Minimal API

```
app.MapPut(Routes.Minimal + "put/{id}", (Guid id, User updateUser) =>
{
    var usersList = new UsersList();
    var existingUser = usersList.Items.FirstOrDefault(x => x.UserId == id);
    if (existingUser == null) return Results.NotFound();

    existingUser.FirstName = updateUser.FirstName;
    existingUser.LastName = updateUser.LastName;
    existingUser.Email = updateUser.Email;
    existingUser.PasswordHash = updateUser.PasswordHash;
    existingUser.PhoneNumber = updateUser.PhoneNumber;
    existingUser.StreetAddress = updateUser.StreetAddress;
    existingUser.ApartmentNumber = updateUser.ApartmentNumber;
    existingUser.City = updateUser.City;
    existingUser.PostalCode = updateUser.PostalCode;
    existingUser.Country = updateUser.Country;
    existingUser.DateOfBirth = updateUser.DateOfBirth;
    existingUser.Gender = updateUser.Gender;
    existingUser.CreatedAt = updateUser.CreatedAt;
    existingUser.UpdatedAt = updateUser.UpdatedAt;

    return Results.NoContent();
});
```

2.5. Metoda DELETE

Metoda DELETE ma za zadanie usunięcie jednej z istniejących wartości. Na Listingu 8 przedstawiono implementację w MVC a na Listingu 9 w Minimal API.

Listing 8: Implementacja żądania DELETE w MVC

```
[HttpDelete]
[Route("delete/{id}")]
0 references
public IActionResult Delete(Guid id)
{
    var usersList = new UsersList();
    var user = usersList.Items.FirstOrDefault(x => x.UserId == id);
    if (user == null) return NotFound();

    usersList.Items.Remove(user);
    return NoContent();
}
```

Listing 9: Implementacja żądania DELETE w Minimal API

```
app.MapDelete(Routes.Minimal + "delete/{id}", (Guid id) =>
{
    var usersList = new UsersList();
    var user = usersList.Items.FirstOrDefault(x => x.UserId == id);
    if (user == null) return Results.NotFound();

    usersList.Items.Remove(user);
    return Results.NoContent();
});
```

3. Metody i przebieg badań

Badania zostały przeprowadzone według poniżej opisanego scenariusza badań a także wykonanie na poniżej opisanym środowisku testowym i środowisku uruchomieniowym.

3.1. Scenariusz badań

Scenariusz polega na zmierzeniu maksymalnej liczby zapytań, jakie system może obsłużyć w określonym czasie przy ustalonej liczbie użytkowników. To pozwoli zrozumieć, jak aplikacja radzi sobie pod ciągłym, intensywnym obciążeniem. Scenariusz ten jest szczególnie przydatny do oceny skalowalności i wydajności aplikacji.

Badanie zostało przeprowadzone na wersji wydaniowej (*ang. Release*) co zapewniło optymalizację kodu pod kątem wydajności oraz umożliwiając dokładną ocenę zachowania aplikacji w warunkach zbliżonych do rzeczywistych środowisk produkcyjnych. Zakres operacji, który został użyty do scenariusza badawczego jest następujący:

- GET – pobranie danych kolekcji,
- POST – utworzenie obiektu,
- PUT – edycja istniejącego obiektu
- DELETE – usunięcie istniejącego obiektu.

Do wykonania badania została użyta aplikacja Apache JMeter w wersji 5.6.3. Parametry scenariusza grupy wątków (*ang. Thread Group*) są następujące:

- Liczba wątków (użytkowników) (*ang. Number of Threads (users)*) – 100,
- Okres narastania (*ang. Ramp-up period*) – 10 sekund,
- Ten sam użytkownik przy każdej iteracji (*ang. Same user on each iteration*) – wyłączone,
- Czas trwania (*ang. Duration*) – 30 sekund.

3.2. Środowisko testowe

W celu realizacji scenariusza badania zastosowano komputer o następujących parametrach technicznych wymienionych w Tabeli 1.

Tabela 1: Komponenty stanowiska testowego

Komponent	Parametr
RAM	Pamięć: 16 GB Rodzaj: DDR4 Szybkość: 3600 MHZ
CPU	Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz Szybkość podstawowa: 2,90 GHz Gniazda: 1 Rdzenie: 6

Dysk	SPCC M.2 PCIe SSD Pojemność: 239 GB
Płyta główna	MPG Z490 GAMING PLUS
Visual Studio	Microsoft Visual Studio Community 2022 (64-bitowy) - Preview Wersja 17.10.0 Preview 7.0
System operacyjny	Windows 11 Home 23H2, Kompilacja 22631.3593, 64-bitowy

W Tabeli 2 podano parametry środowiska uruchomieniowego i użytych wersji bibliotek do przeprowadzenia badania.

Tabela 2: Wersje Runtime dla każdej wersji platformy .NET

Wersja .NET	Wersja Runtime
.NET Core 3.1 (LTS)	Microsoft.AspNetCore.App 3.1.32 Microsoft.NETCore.App 3.1.32 Microsoft.WindowsDesktop.App 3.1.32
.NET 5 (STS)	Microsoft.AspNetCore.App 5.0.17 Microsoft.NETCore.App 5.0.17 Microsoft.WindowsDesktop.App 5.0.17
.NET 6 (LTS)	Microsoft.AspNetCore.App 6.0.30 Microsoft.NETCore.App 6.0.30 Microsoft.WindowsDesktop.App 6.0.30
.NET 7 (STS)	Microsoft.AspNetCore.App 7.0.19 Microsoft.NETCore.App 7.0.19 Microsoft.WindowsDesktop.App 7.0.19
.NET 8 (LTS)	Microsoft.AspNetCore.App 8.0.5 Microsoft.NETCore.App 8.0.5 Microsoft.WindowsDesktop.App 8.0.5
.NET 9 (STS) (Preview w wersji 3)	Microsoft.AspNetCore.App 9.0.0-preview.3.24172.13 Microsoft.NETCore.App 9.0.0-preview.3.24172.9 Microsoft.WindowsDesktop.App 9.0.0-preview.3.24175.3

4. Wyniki badań

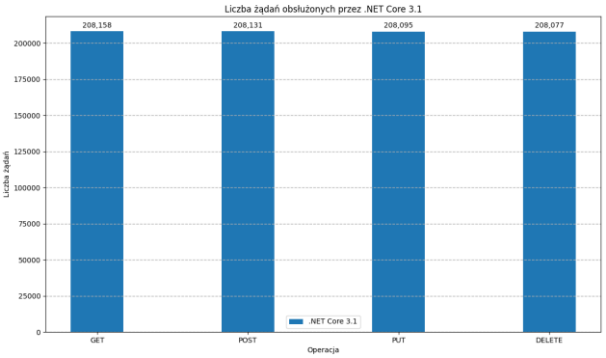
Po przeprowadzeniu badań wyniki zostały przedstawione w Tabelach zamieszczonych poniżej.

4.1. .NET Core 3.1

Dla wersji .NET Core 3.1 (Tabela 3), MVC obsłużyło łącznie 832 461 żądań, przy czym każde żądanie GET, POST, PUT i DELETE miało zbliżone wyniki, oscylujące wokół 208 tysięcy. Jest to najniższy wynik spośród wszystkich analizowanych wersji, co wskazuje na ograniczenia w skalowalności i wydajności MVC w tej wersji platformy (Rysunek 2).

Tabela 3: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	208 158
MVC:POST	208 131
MVC:PUT	208 095
MVC:DELETE	208 077
SUMA	832 461



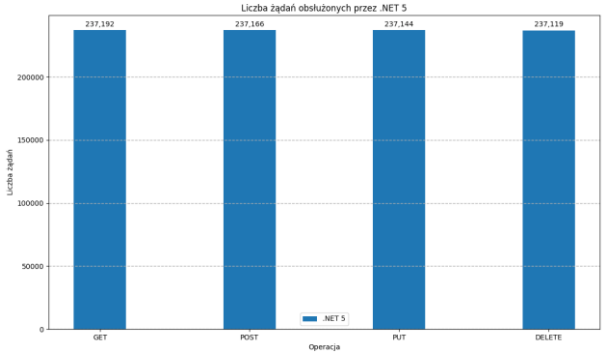
Rysunek 2: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET Core 3.1.

4.2. .NET 5

W wersji .NET 5 (Tabela 4), MVC obsłużyło łącznie 948 621 żądań, co stanowi znaczny wzrost w porównaniu do .NET Core 3.1. Wyniki dla poszczególnych typów żądań (GET, POST, PUT, DELETE) wzrosły do około 237 tysięcy. Ta poprawa wskazuje na znaczną optymalizację wprowadzaną w .NET 5, która zwiększa efektywność obsługi żądań przez kontrolery MVC (Rysunek 3).

Tabela 4: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	237 192
MVC:POST	237 166
MVC:PUT	237 144
MVC:DELETE	237 119
SUMA	948 621



Rysunek 3: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET5.

4.3. .NET 6

W .NET 6 (Tabela 5 i 6), Minimal API zostało wprowadzone jako nowa, bardziej wydajna alternatywa dla MVC. Minimal API obsłużyło 1 052 562 żądania, znacznie przewyższając wynik MVC, które obsłużyło 908 986 żądań. Każde z żądań GET, POST, PUT i DELETE dla Minimal API wyniosło około 263 tysięcy, podczas gdy dla MVC wyniki oscylowały wokół 227 tysięcy. Wyniki te sugerują, że Minimal API oferuje lepszą skalowalność

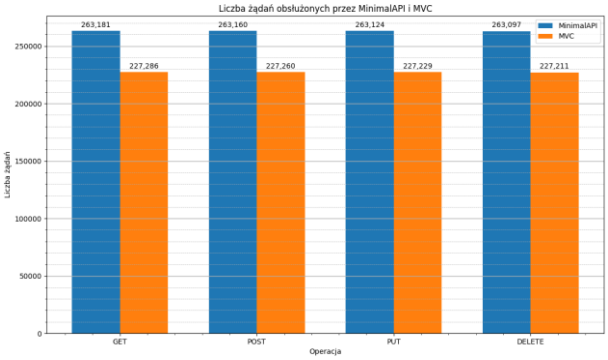
i wydajność w porównaniu do MVC w tej wersji platformy (Rysunek 4).

Tabela 5: Wyniki liczby zapytań w Minimal API

Rodzaj zapytania HTTP	Liczba żądań
Minimal API:GET	263 181
Minimal API:POST	263160
Minimal API:PUT	263 124
Minimal API:DELETE	263 097
SUMA	1 052 562

Tabela 6: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	227 286
MVC:POST	227 260
MVC:PUT	227 229
MVC:DELETE	227 211
SUMA	908 986



Rysunek 4: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET6.

4.4. .NET 7

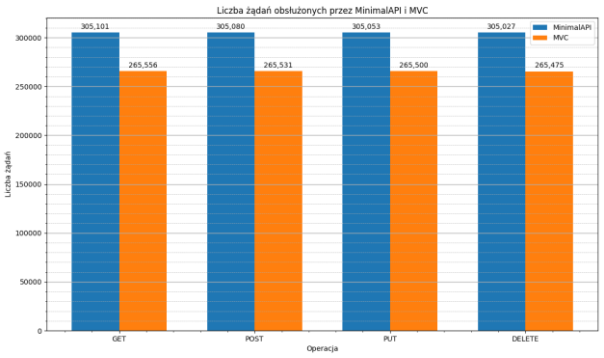
W .NET 7 (Tabela 7 i 8), różnice pomiędzy Minimal API a MVC stają się jeszcze bardziej widoczne. Minimal API obsłużyło 1 220 261 żądań, podczas gdy MVC obsłużyło 1 062 062 żądania. Liczba obsłużonych żądań GET przez Minimal API wyniosła 305 101, a dla MVC 265 556. Te wyniki podkreślają dalszą optymalizację Minimal API, które staje się coraz bardziej efektywne w porównaniu do MVC (Rysunek 5).

Tabela 7: Wyniki liczby zapytań w Minimal API

Rodzaj zapytania HTTP	Liczba żądań
Minimal API:GET	305 101
Minimal API:POST	305 080
Minimal API:PUT	305 053
Minimal API:DELETE	305 027
SUMA	1 220 261

Tabela 8: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	265 556
MVC:POST	265 531
MVC:PUT	265 500
MVC:DELETE	265 475
SUMA	1 062 062



Rysunek 5: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET7.

4.5. .NET 8

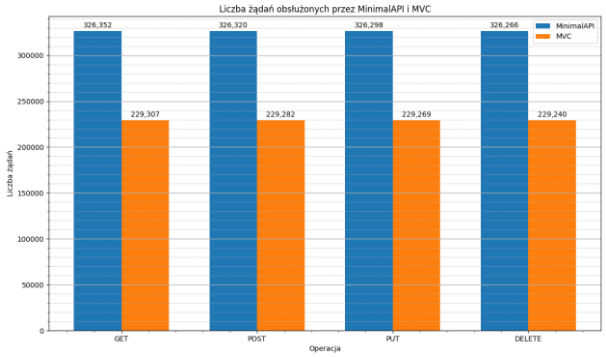
W .NET 8 (Tabela 9 i 7.10), Minimal API wykazało jeszcze wyższą wydajność, obsługując 1 305 236 żądań, natomiast MVC obsłużyło 917 098 żądań. Liczby te wskazują na dalsze usprawnienia w Minimal API, które są bardziej widoczne w zwiększonej liczbie obsłużonych żądań GET (326 352 dla Minimal API vs 229 307 dla MVC). Taka różnica pokazuje, że Minimal API jest bardziej zoptymalizowane i lepiej radzi sobie z dużym obciążeniem (Rysunek 6).

Tabela 9: Wyniki liczby zapytań w Minimal API

Rodzaj zapytania HTTP	Liczba żądań
Minimal API:GET	326 352
Minimal API:POST	326 320
Minimal API:PUT	326 298
Minimal API:DELETE	326 266
SUMA	1 305 236

Tabela 10: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	229 307
MVC:POST	229 282
MVC:PUT	229 269
MVC:DELETE	229 240
SUMA	917 098



Rysunek 6: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET8.

4.6. .NET 9 Preview 3

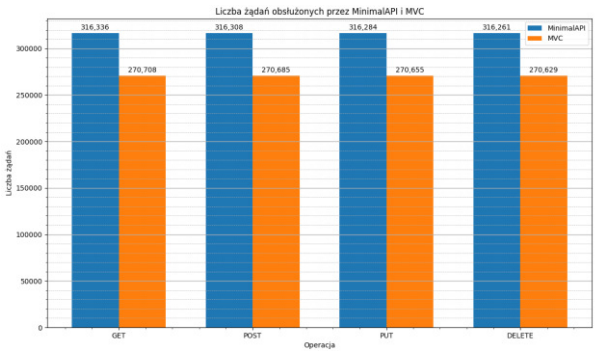
Najbardziej aktualne wyniki dla .NET 9 Preview 3 (Tabela 7.11 i 7.12) pokazują, że Minimal API obsłużyło 1 265 189 żądań, a MVC obsłużyło 1 082 677 żądania. Liczba obsłużonych żądań GET dla Minimal API wyniosła 316 336, natomiast dla MVC 270 708. Te dane potwierdzają kontynuację trendu, w którym Minimal API przewyższa MVC pod względem wydajności i skalowalności (Rysunek 7).

Tabela 11: Wyniki liczby zapytań w Minimal API

Rodzaj zapytania HTTP	Liczba żądań
Minimal API:GET	316 336
Minimal API:POST	316 308
Minimal API:PUT	316 284
Minimal API:DELETE	316 261
SUMA	1 265 189

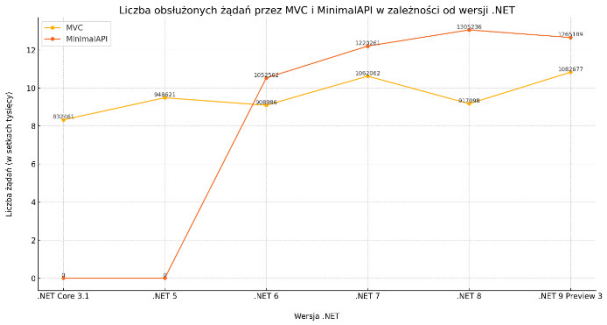
Tabela 12: Wyniki liczby zapytań w kontrolerach MVC

Rodzaj zapytania HTTP	Liczba żądań
MVC:GET	270 708
MVC:POST	270 685
MVC:PUT	270 655
MVC:DELETE	270 629
SUMA	1 082 677



Rysunek 7: Liczba żądań obsłużona przez Minimal API i MVC dla wersji .NET9 (Preview 3).

Analiza wyników pokazuje, że Minimal API konsekwentnie przewyższa MVC pod względem liczby obsłużonych żądań w każdym z testowanych środowisk .NET. Od .NET Core 3.1 do .NET 9 Preview 3, Minimal API wykazuje lepszą skalowalność i efektywność, szczególnie widoczną w nowszych wersjach platformy. Minimal API wprowadzone w .NET 6 szybko okazało się bardziej wydajne niż MVC, a kolejne wersje .NET tylko umacniają tę przewagę. Optymalizacje w architekturze Minimal API oraz redukcja narzutów związanych z przetwarzaniem zapytań pozwalają na obsługę większej liczby równoczesnych żądań, co czyni je bardziej odpowiednim wyborem dla aplikacji wymagających wysokiej przepustowości i niskich czasów odpowiedzi (Rysunek 8).



Rysunek 8: Liczba żądań obsłużona przez Minimal API i MVC dla różnych wersji .NET.

5. Wnioski

Celem niniejszej analizy było porównanie wydajności technologii Minimal API i MVC w kontekście obsługi prostych zapytań HTTP. Badanie miało na celu określenie, która z tych technologii jest bardziej wydajna do obsługi dużej liczby zapytań w określonym czasie. W związku z tym postawiono pytania dotyczące liczby obsłużonych zapytań przez każdą z technologii oraz ich szybkości w przetwarzaniu prostych zapytań HTTP

Scenariusz badań miał na celu ocenę zdolności obu technologii do obsługi dużej liczby zapytań HTTP. Badanie koncentrowało się na liczbie zapytań, które każda technologia może obsłużyć w określonym czasie, w warunkach intensywnego ruchu sieciowego. Analiza wykazała, że Minimal API obsługuje więcej zapytań niż MVC we wszystkich testowanych wersjach .NET. Różnice są znaczące, co sugeruje, że Minimal API lepiej radzi sobie z intensywnym ruchem sieciowym, oferując większą przepustowość i wydajność. Zdolność Minimal API do obsługi większej liczby zapytań jest szczególnie widoczna w nowszych wersjach .NET.

Hipoteza artykułu, która zakłada, że celem analizy jest wskazanie, która technologia jest wydajniejsza do obsługi prostych zapytań HTTP, została potwierdzona. Wyniki jednoznacznie wskazują, że Minimal API przewyższa MVC pod względem wydajności w zakresie obsługi prostych zapytań HTTP w kontekście liczby obsłużonych zapytań.

Literatura

[1] M. Plechawska-Wójcik, E. Piątkowska, K. Wąsik, The use of .NET Core in web applications development, Journal of Computer Sciences Institute 7 (2018) 142-149, https://doi.org/10.35784/jcsi.663.

[2] D. Zmaranda, L-L. Pop-Fele, C. Gyorödi, R. Gyorödi, G. Pecherle, Performance comparison of CRUD methods using .NET Object relational mappers: A case study, International Journal of Advanced Computer Science and Applications 11 (2020) 55-65, https://dx.doi.org/10.14569/IJACSA.2020.0110107.

[3] M. Jailia, A. Kumar, M. Agarwal, I. Sinha, Behavior of MVC (Model View Controller) based Web Application developed in PHP and .NET framework, In International Conference on ICT in Business Industry & Government (ICTBIG) (2016) 1-5, https://doi.org/10.1109/ICTBIG.2016.7892651

[4] H. Osama, K. Nedal, Performance testing for web-based application architectures (.NET vs. Java EE), In First International Conference on Networked Digital Technologies (NDT) (2009) 218-224, https://doi.org/10.1109/NDT.2009.5272178

[5] L. Zhongwei, W. Shu-Guang, Design of .Net Courseware on Demand System, In Eighth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA) (2016) 206-209, https://doi.org/10.1109/ICMTMA.2016.59

[6] M. Coblenz, W. Guo, K. Voozhian, J. Foster, A Qualitative Study of REST API Design and Specification Practices, In IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (2023) 148-156, https://doi.org/10.1109/VL-HCC57772.2023.00025.

[7] I. Grigorik, Making the web faster with HTTP 2.0, Communications of the ACM 56 (2013) 42-49, https://doi.org/10.1145/2534706.2534721.

[8] M. Trevisan, D. Giordano, I. Drago, A. Khatouni, Measuring HTTP/3: Adoption and performance, In 19th Mediterranean Communication and Computer Networking Conference (MedComNet) (2021) 1-8, https://doi.org/10.1109/MedComNet52149.2021.9501274

[9] J. Heidemann, K. Obraczka, J. Touch, Modeling the performance of HTTP over several transport protocols, IEEE/ACM Transactions on Networking 5 (1997) 616-630, https://doi.org/10.1109/90.649564.

[10] W. Xie, Y. Li, Ch. Lu, R. Shen, Optimizing The Resource-updating Period Behavior of HTTP Cache Servers for Better Scalability of Live HTTP Streaming Systems, In IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB) (2012) 1-6, https://doi.org/10.1109/BMSB.2012.6264230

[11] A. Popescu, Y. Bhole, Measurement and analysis of HTTP traffic, Journal of Network and Systems Management 13 (2005) 357-371, https://doi.org/10.1109/NDT.2009.5272178