**Samouczek: tworzenie minimalnego interfejsu API przy użyciu platformy ASP.NET Core**

**W tym artykule**

1. [Omówienie](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#overview)
2. [Wymagania wstępne](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#prerequisites)
3. [Tworzenie projektu interfejsu API](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#create-an-api-project)
4. [Dodawanie pakietów NuGet](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#add-nuget-packages)

Autor [: Rick Anderson](https://twitter.com/RickAndMSFT) i [Tom Dykstra](https://github.com/tdykstra)

Minimalne interfejsy API są tworzone w celu tworzenia interfejsów API HTTP z minimalnymi zależnościami. Są one idealne dla mikrousług i aplikacji, które chcą uwzględniać tylko minimalne pliki, funkcje i zależności w ASP.NET Core.

W tym samouczku przedstawiono podstawy tworzenia minimalnego interfejsu API przy użyciu platformy ASP.NET Core. Innym podejściem do tworzenia interfejsów API w programie ASP.NET Core jest użycie kontrolerów. Aby uzyskać pomoc dotyczącą wybierania między minimalnymi interfejsami API i interfejsami API opartymi na kontrolerach, zobacz [Omówienie](https://learn.microsoft.com/pl-pl/aspnet/core/fundamentals/apis?view=aspnetcore-8.0) interfejsów API. Aby zapoznać się z samouczkiem dotyczącym tworzenia projektu interfejsu API na [podstawie kontrolerów](https://learn.microsoft.com/pl-pl/aspnet/core/web-api/?view=aspnetcore-8.0) zawierających więcej funkcji, zobacz [Tworzenie internetowego interfejsu API](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0).

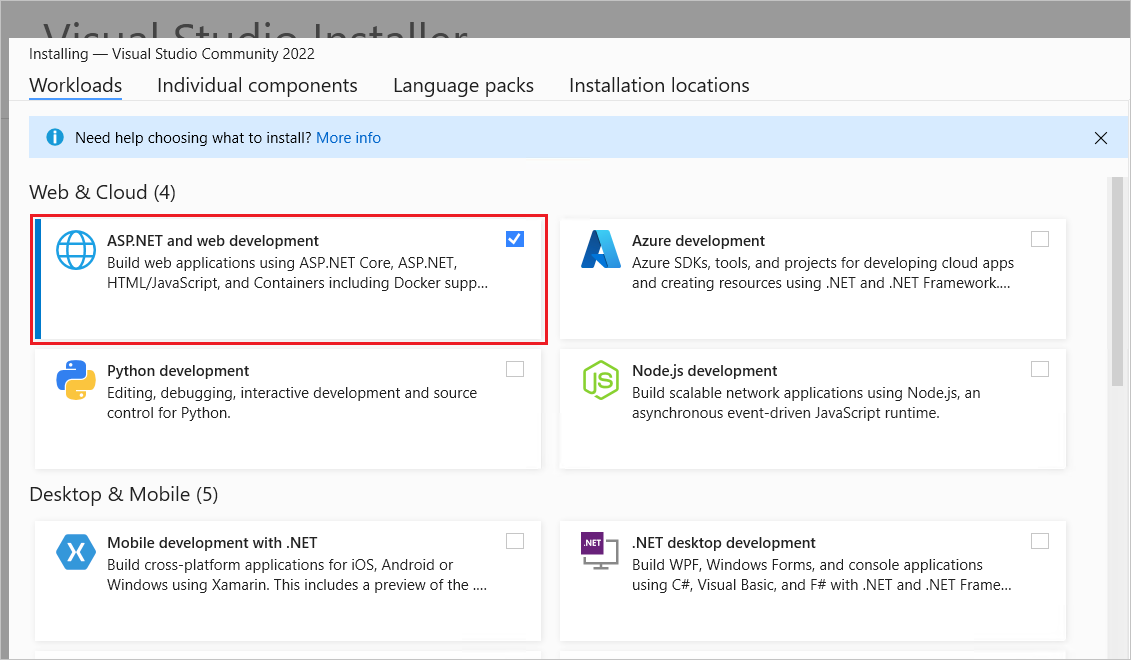
**Omówienie**

Ten samouczek tworzy następujący interfejs API:

| **Interfejs API** | **opis** | **Treść żądania** | **Treść odpowiedzi** |
| --- | --- | --- | --- |
| GET /todoitems | Pobieranie wszystkich elementów do wykonania | Brak | Tablica elementów do wykonania |
| GET /todoitems/complete | Pobieranie ukończonych elementów do wykonania | Brak | Tablica elementów do wykonania |
| GET /todoitems/{id} | Pobieranie elementu według identyfikatora | Brak | Element do wykonania |
| POST /todoitems | Dodawanie nowego elementu | Element do wykonania | Element do wykonania |
| PUT /todoitems/{id} | Aktualizowanie istniejącego elementu | Element do wykonania | Brak |
| DELETE /todoitems/{id} | Usuwanie elementu | Brak | Brak |

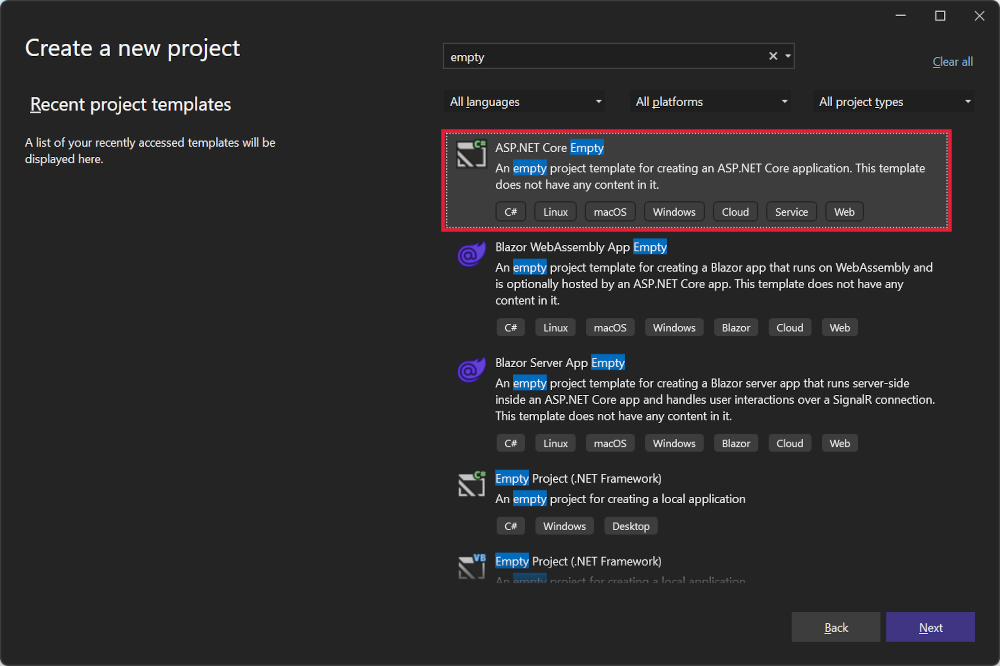
**Wymagania wstępne**

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio-code)
* [Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_1_visual-studio-mac)
* [Program Visual Studio 2022](https://visualstudio.microsoft.com/downloads/) z pakietem roboczym **tworzenia aplikacji ASP.NET i aplikacji internetowych**.

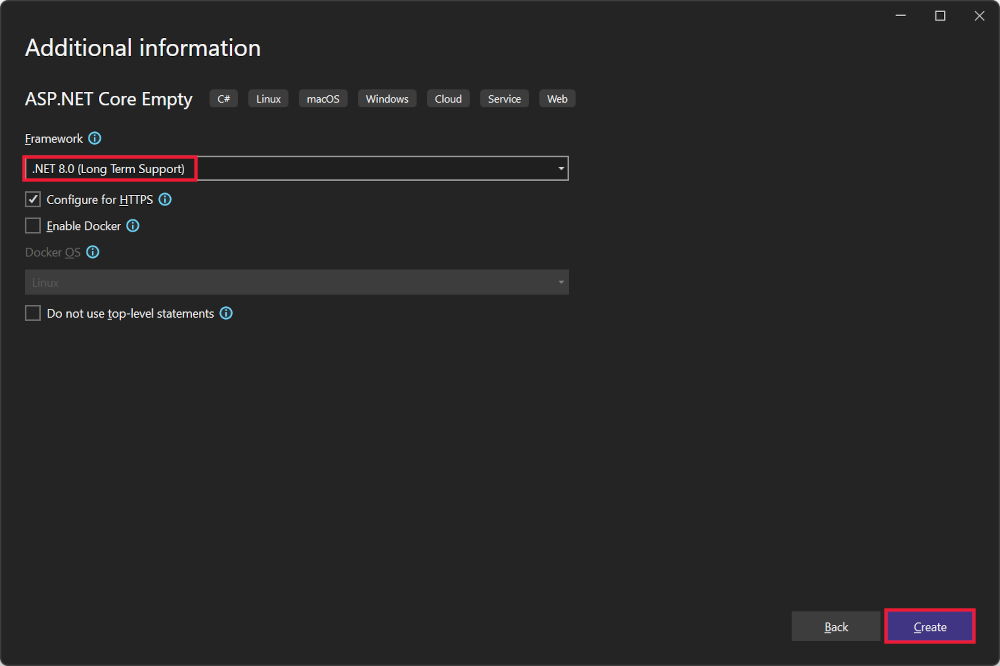


**Tworzenie projektu interfejsu API**

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio-code)
* [Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_2_visual-studio-mac)
* Uruchom program Visual Studio 2022 i wybierz pozycję **Utwórz nowy projekt**.
* **W oknie dialogowym Tworzenie nowego projektu**:
  + Wprowadź ciąg Empty w **polu wyszukiwania Wyszukaj szablony** .
  + **Wybierz szablon ASP.NET Core Empty** i wybierz przycisk **Dalej**.



* Nadaj projektowi *nazwę TodoApi* i wybierz pozycję **Dalej**.
* **W oknie dialogowym Dodatkowe informacje**:
  + Wybierz pozycję **.NET 8.0 (obsługa długoterminowa)**
  + Usuń zaznaczenie pola **Wyboru Nie używaj instrukcji najwyższego poziomu**
  + Wybierz pozycję **Utwórz**



**Analizowanie kodu**

Plik Program.cs zawiera następujący kod:

C#

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();

Powyższy kod ma następujące działanie:

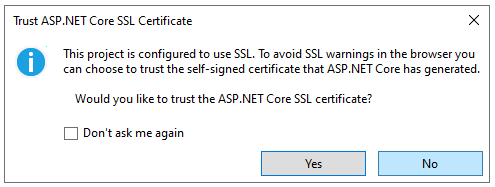
* Tworzy obiekt [WebApplicationBuilder](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.builder.webapplicationbuilder) i [WebApplication](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.builder.webapplication) ze wstępnie skonfigurowanymi wartościami domyślnymi.
* Tworzy punkt końcowy / HTTP GET, który zwraca wartość Hello World!:

**Uruchom aplikację**

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_3_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_3_visual-studio-code)
* [Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_3_visual-studio-mac)

Naciśnij klawisze Ctrl+F5, aby uruchomić bez debugera.

Program Visual Studio wyświetla następujące okno dialogowe:



Wybierz pozycję **Tak** , jeśli ufasz certyfikatowi SSL usług IIS Express.

Zostanie wyświetlone następujące okno dialogowe:



Wybierz pozycję **Tak**, jeśli wyrażasz zgodę na zaufanie certyfikatowi programistycznemu.

Aby uzyskać informacje na temat zaufania przeglądarce Firefox, zobacz [Błąd](https://learn.microsoft.com/pl-pl/aspnet/core/security/enforcing-ssl?view=aspnetcore-8.0#trust-ff) certyfikatu przeglądarki Firefox SEC\_ERROR\_INADEQUATE\_KEY\_USAGE.

Program Visual Studio uruchamia [Kestrel serwer](https://learn.microsoft.com/pl-pl/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-8.0) internetowy i otwiera okno przeglądarki.

Hello World! jest wyświetlany w przeglądarce. Plik Program.cs zawiera minimalną, ale kompletną aplikację.

Zamknij okno przeglądarki.

**Dodawanie pakietów NuGet**

Pakiety NuGet należy dodać do obsługi bazy danych i diagnostyki używanej w tym samouczku.

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_4_visual-studio)
* [Visual Studio Code](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_4_visual-studio-code)
* [Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_4_visual-studio-mac)
* **W menu Narzędzia** wybierz pozycję **NuGet Menedżer pakietów > Zarządzaj pakietami NuGet dla rozwiązania**.
* Wybierz kartę **Przeglądaj**.
* Wprowadź ciąg **Microsoft.EntityFrameworkCore.InMemory** w polu wyszukiwania, a następnie wybierz pozycję Microsoft.EntityFrameworkCore.InMemory.
* **Zaznacz pole wyboru Projekt** w okienku po prawej stronie, a następnie wybierz pozycję **Zainstaluj**.
* Postępuj zgodnie z poprzednimi instrukcjami, aby dodać Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore pakiet.

**Klasy kontekstu modelu i bazy danych**

* W folderze projektu utwórz plik o nazwie Todo.cs z następującym kodem:

C#

public class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

}

Powyższy kod tworzy model dla tej aplikacji. *Model* to klasa reprezentująca dane, którymi zarządza aplikacja.

* Utwórz plik o nazwie z TodoDb.cs następującym kodem:

C#

using Microsoft.EntityFrameworkCore;

class TodoDb : DbContext

{

public TodoDb(DbContextOptions<TodoDb> options)

: base(options) { }

public DbSet<Todo> Todos => Set<Todo>();

}

Powyższy kod definiuje *kontekst* bazy danych, który jest główną klasą, która koordynuje [funkcje programu Entity Framework](https://learn.microsoft.com/pl-pl/ef/core/) dla modelu danych. Ta klasa pochodzi z [Microsoft.EntityFrameworkCore.DbContext](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.entityframeworkcore.dbcontext) klasy .

**Dodawanie kodu interfejsu API**

* Zastąp zawartość pliku Program.cs następującym kodem:

C#

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

app.MapGet("/todoitems", async (TodoDb db) =>

await db.Todos.ToListAsync());

app.MapGet("/todoitems/complete", async (TodoDb db) =>

await db.Todos.Where(t => t.IsComplete).ToListAsync());

app.MapGet("/todoitems/{id}", async (int id, TodoDb db) =>

await db.Todos.FindAsync(id)

is Todo todo

? Results.Ok(todo)

: Results.NotFound());

app.MapPost("/todoitems", async (Todo todo, TodoDb db) =>

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

return Results.Created($"/todoitems/{todo.Id}", todo);

});

app.MapPut("/todoitems/{id}", async (int id, Todo inputTodo, TodoDb db) =>

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return Results.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return Results.NoContent();

});

app.MapDelete("/todoitems/{id}", async (int id, TodoDb db) =>

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return Results.NoContent();

}

return Results.NotFound();

});

app.Run();

Poniższy wyróżniony kod dodaje kontekst bazy danych do kontenera [wstrzykiwania zależności (DI)](https://learn.microsoft.com/pl-pl/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0) i umożliwia wyświetlanie wyjątków związanych z bazą danych:

C#

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

Kontener DI zapewnia dostęp do kontekstu bazy danych i innych usług.

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_5_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_5_visual-studio-code+visual-studio-mac)

W tym samouczku używane są [pliki](https://learn.microsoft.com/pl-pl/aspnet/core/test/http-files?view=aspnetcore-8.0#use-endpoints-explorer) Endpoints Explorer i .http do testowania interfejsu API.

**Testowanie publikowania danych**

Poniższy kod w pliku Program.cs tworzy punkt końcowy /todoitems HTTP POST, który dodaje dane do bazy danych w pamięci:

C#

app.MapPost("/todoitems", async (Todo todo, TodoDb db) =>

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

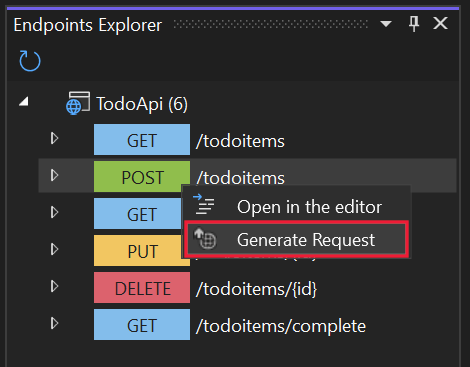
return Results.Created($"/todoitems/{todo.Id}", todo);

});

Uruchom aplikację. Przeglądarka wyświetla błąd 404, ponieważ nie ma już punktu końcowego / .

Punkt końcowy POST będzie używany do dodawania danych do aplikacji.

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_6_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_6_visual-studio-code+visual-studio-mac)
* Wybierz pozycję **Wyświetl**>**inne eksploratora punktów końcowych systemu Windows.>**
* Kliknij prawym przyciskiem myszy **punkt końcowy POST** i wybierz polecenie **Generuj żądanie**.



Nowy plik jest tworzony w folderze projektu o nazwie TodoApi.http, z zawartością podobną do następującego przykładu:

 @TodoApi\_HostAddress = https://localhost:7031

Post {{TodoApi\_HostAddress}}/todoitems

###

* Pierwszy wiersz tworzy zmienną używaną dla wszystkich punktów końcowych.
* Następny wiersz definiuje żądanie POST.
* Potrójny hasztag (###) wiersz jest ogranicznikiem żądania: co następuje po nim dla innego żądania.

 Żądanie POST wymaga nagłówków i treści. Aby zdefiniować te części żądania, dodaj następujące wiersze bezpośrednio po wierszu żądania POST:

Content-Type: application/json

{

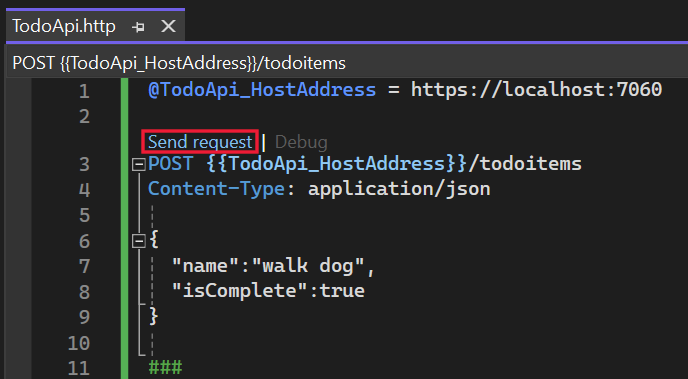
"name":"walk dog",

"isComplete":true

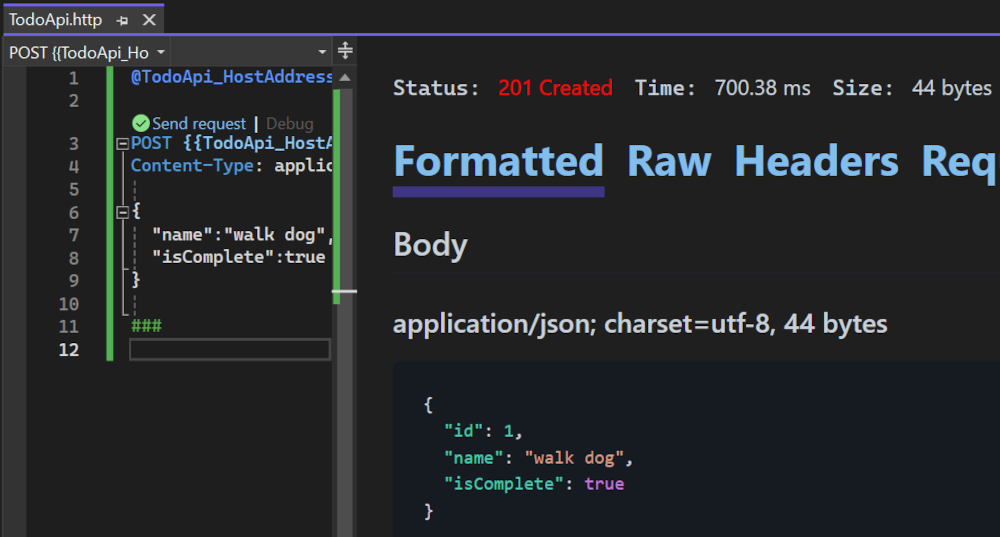
}

Powyższy kod dodaje nagłówek Content-Type i JStreść żądania ON. Plik TodoApi.http powinien teraz wyglądać podobnie do poniższego przykładu, ale z numerem portu:

* @TodoApi\_HostAddress = https://localhost:7057
* Post {{TodoApi\_HostAddress}}/todoitems
* Content-Type: application/json
* {
* "name":"walk dog",
* "isComplete":true
* }
* ###
* Uruchom aplikację.
* **Wybierz link Wyślij żądanie** powyżej POST wiersza żądania.



Żądanie POST jest wysyłane do aplikacji, a odpowiedź jest wyświetlana w okienku **Odpowiedź** .



**Sprawdzanie punktów końcowych GET**

Przykładowa aplikacja implementuje kilka punktów końcowych GET, wywołując metodę MapGet:

| **Interfejs API** | **opis** | **Treść żądania** | **Treść odpowiedzi** |
| --- | --- | --- | --- |
| GET /todoitems | Pobieranie wszystkich elementów do wykonania | Brak | Tablica elementów do wykonania |
| GET /todoitems/complete | Pobieranie wszystkich ukończonych elementów do wykonania | Brak | Tablica elementów do wykonania |
| GET /todoitems/{id} | Pobieranie elementu według identyfikatora | Brak | Element do wykonania |

C#

app.MapGet("/todoitems", async (TodoDb db) =>

await db.Todos.ToListAsync());

app.MapGet("/todoitems/complete", async (TodoDb db) =>

await db.Todos.Where(t => t.IsComplete).ToListAsync());

app.MapGet("/todoitems/{id}", async (int id, TodoDb db) =>

await db.Todos.FindAsync(id)

is Todo todo

? Results.Ok(todo)

: Results.NotFound());

**Testowanie punktów końcowych GET**

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_7_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_7_visual-studio-code+visual-studio-mac)

Przetestuj GET aplikację, wywołując punkty końcowe z przeglądarki lub przy użyciu **Eksploratora** punktów końcowych. Poniższe kroki dotyczą **Eksploratora** punktów końcowych.

* W **Eksploratorze** punktów końcowych kliknij prawym przyciskiem myszy pierwszy **punkt końcowy GET** i wybierz polecenie **Generuj żądanie**.

Do pliku zostanie dodana następująca TodoApi.http zawartość:

 Get {{TodoApi\_HostAddress}}/todoitems

###

 **Wybierz link Wyślij żądanie** powyżej nowego GET wiersza żądania.

Żądanie GET jest wysyłane do aplikacji, a odpowiedź jest wyświetlana w okienku **Odpowiedź** .

 Treść odpowiedzi jest podobna do następującej:JS

JSON

 [

{

"id": 1,

"name": "walk dog",

"isComplete": true

}

]

 W **Eksploratorze** punktów końcowych kliknij prawym przyciskiem**/todoitems/{id} myszy punkt końcowy GET** i wybierz pozycję **Generuj żądanie**. Do pliku zostanie dodana następująca TodoApi.http zawartość:

 GET {{TodoApi\_HostAddress}}/todoitems/{id}

###

 Zamień {id} na 1.

 **Wybierz link Wyślij żądanie** powyżej nowego wiersza żądania GET.

Żądanie GET jest wysyłane do aplikacji, a odpowiedź jest wyświetlana w okienku **Odpowiedź** .

 Treść odpowiedzi jest podobna do następującej:JS

JSON

* {
* "id": 1,
* "name": "walk dog",
* "isComplete": true
* }

Ta aplikacja używa bazy danych w pamięci. Jeśli aplikacja zostanie ponownie uruchomiona, żądanie GET nie zwraca żadnych danych. Jeśli żadne dane nie są zwracane, [prześlij dane POST](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#post) do aplikacji i spróbuj ponownie wysłać żądanie GET.

**Wartości zwracane**

ASP.NET Core automatycznie serializuje obiekt wł[JS](https://www.json.org). i zapisuje wartość JSON w treści komunikatu odpowiedzi. Kod odpowiedzi dla tego typu zwracanego to [200 OK](https://developer.mozilla.org/docs/Web/HTTP/Status/200), zakładając, że nie ma żadnych nieobsługiwane wyjątki. Nieobsługiwane wyjątki są tłumaczone na błędy 5xx.

Typy zwracane mogą reprezentować szeroki zakres kodów stanu HTTP. Na przykład GET /todoitems/{id} może zwrócić dwie różne wartości stanu:

* Jeśli żaden element nie pasuje do żądanego identyfikatora, metoda zwraca [kod błędu stanu](https://developer.mozilla.org/docs/Web/HTTP/Status/404)[NotFound](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.mvc.controllerbase.notfound) 404.
* W przeciwnym razie metoda zwraca wartość 200 z treścią JSodpowiedzi ON. Zwracanie item wyników w odpowiedzi HTTP 200.

**Badanie punktu końcowego PUT**

Przykładowa aplikacja implementuje pojedynczy punkt końcowy PUT przy użyciu polecenia MapPut:

C#

app.MapPut("/todoitems/{id}", async (int id, Todo inputTodo, TodoDb db) =>

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return Results.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return Results.NoContent();

});

Ta metoda jest podobna do metody , z tą różnicą MapPost , że używa protokołu HTTP PUT. Pomyślna odpowiedź zwraca [wartość 204 (brak zawartości).](https://www.rfc-editor.org/rfc/rfc9110#status.204) Zgodnie ze specyfikacją PROTOKOŁU HTTP żądanie PUT wymaga od klienta wysłania całej zaktualizowanej jednostki, a nie tylko zmian. Aby obsługiwać aktualizacje częściowe, użyj poprawki [HTTP PATCH](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.mvc.httppatchattribute).

**Testowanie punktu końcowego PUT**

W tym przykładzie użyto bazy danych w pamięci, która musi zostać zainicjowana przy każdym uruchomieniu aplikacji. Przed wykonaniem wywołania PUT musi istnieć element w bazie danych. Wywołaj metodę GET, aby upewnić się, że istnieje element w bazie danych przed wykonaniem wywołania PUT.

Zaktualizuj element to-do, który ma Id = 1 wartość , i ustaw jego nazwę na "feed fish".

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_8_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_8_visual-studio-code+visual-studio-mac)
* W **Eksploratorze** punktów końcowych kliknij prawym przyciskiem **myszy punkt końcowy PUT** i wybierz pozycję **Generuj żądanie**.

Do pliku zostanie dodana następująca TodoApi.http zawartość:

 Put {{TodoApi\_HostAddress}}/todoitems/{id}

###

 W wierszu żądania PUT zastąp ciąg {id} ciągiem 1.

 Dodaj następujące wiersze bezpośrednio po wierszu żądania PUT:

* Content-Type: application/json
* {
* "name": "feed fish",
* "isComplete": false
* }
* Powyższy kod dodaje nagłówek Content-Type i JStreść żądania ON.
* **Wybierz link Wyślij żądanie** powyżej nowego wiersza żądania PUT.

Żądanie PUT jest wysyłane do aplikacji, a odpowiedź jest wyświetlana w okienku **Odpowiedź** . Treść odpowiedzi jest pusta, a kod stanu to 204.

**Sprawdzanie i testowanie punktu końcowego DELETE**

Przykładowa aplikacja implementuje pojedynczy punkt końcowy DELETE przy użyciu polecenia MapDelete:

C#

app.MapDelete("/todoitems/{id}", async (int id, TodoDb db) =>

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return Results.NoContent();

}

return Results.NotFound();

});

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_9_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_9_visual-studio-code+visual-studio-mac)
* W **Eksploratorze** punktów końcowych kliknij prawym przyciskiem **myszy punkt końcowy DELETE** i wybierz pozycję **Generuj żądanie**.

Żądanie DELETE jest dodawane do TodoApi.httpelementu .

* Zastąp element {id} w wierszu żądania DELETE ciągiem 1. Żądanie DELETE powinno wyglądać podobnie do następującego przykładu:
* DELETE {{TodoApi\_HostAddress}}/todoitems/1
* ###
* **Wybierz link Wyślij żądanie** dla żądania DELETE.

Żądanie DELETE jest wysyłane do aplikacji, a odpowiedź jest wyświetlana w okienku **Odpowiedź** . Treść odpowiedzi jest pusta, a kod stanu to 204.

**Korzystanie z interfejsu API grupy map**

Przykładowy kod aplikacji powtarza prefiks adresu URL za każdym razem, gdy konfiguruje todoitems punkt końcowy. Interfejsy API często mają grupy punktów końcowych z typowym prefiksem adresu URL, a [MapGroup](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.builder.endpointroutebuilderextensions.mapgroup) metoda jest dostępna w celu ułatwienia organizowania takich grup. Zmniejsza powtarzalny kod i umożliwia dostosowywanie całych grup punktów końcowych za pomocą jednego wywołania metod, takich jak [RequireAuthorization](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.builder.authorizationendpointconventionbuilderextensions.requireauthorization) i [WithMetadata](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.builder.routingendpointconventionbuilderextensions.withmetadata).

Zastąp zawartość Program.cs następującym kodem:

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_10_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_10_visual-studio-code+visual-studio-mac)

C#

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

var todoItems = app.MapGroup("/todoitems");

todoItems.MapGet("/", async (TodoDb db) =>

await db.Todos.ToListAsync());

todoItems.MapGet("/complete", async (TodoDb db) =>

await db.Todos.Where(t => t.IsComplete).ToListAsync());

todoItems.MapGet("/{id}", async (int id, TodoDb db) =>

await db.Todos.FindAsync(id)

is Todo todo

? Results.Ok(todo)

: Results.NotFound());

todoItems.MapPost("/", async (Todo todo, TodoDb db) =>

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

return Results.Created($"/todoitems/{todo.Id}", todo);

});

todoItems.MapPut("/{id}", async (int id, Todo inputTodo, TodoDb db) =>

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return Results.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return Results.NoContent();

});

todoItems.MapDelete("/{id}", async (int id, TodoDb db) =>

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return Results.NoContent();

}

return Results.NotFound();

});

app.Run();

Powyższy kod ma następujące zmiany:

* Dodaje var todoItems = app.MapGroup("/todoitems"); element do skonfigurowania grupy przy użyciu prefiksu /todoitemsadresu URL .
* Zmienia wszystkie app.Map<HttpVerb> metody na todoItems.Map<HttpVerb>.
* Usuwa prefiks /todoitems adresu URL z Map<HttpVerb> wywołań metody.

Przetestuj punkty końcowe, aby sprawdzić, czy działają one tak samo.

**Korzystanie z interfejsu API TypedResults**

Zwracanie [TypedResults](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.http.typedresults) , a nie [Results](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.http.results) ma kilku zalet, w tym możliwości testowania i automatycznego zwracania metadanych typu odpowiedzi dla interfejsu OpenAPI w celu opisania punktu końcowego. Aby uzyskać więcej informacji, zobacz [TypedResults vs Results (TypdResults a wyniki](https://learn.microsoft.com/pl-pl/aspnet/core/fundamentals/minimal-apis/responses#typedresults-vs-results)).

Metody Map<HttpVerb> mogą wywoływać metody obsługi tras zamiast używać lambd. Aby zobaczyć przykład, zaktualizuj *Program.cs* przy użyciu następującego kodu:

* [Program Visual Studio](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_11_visual-studio)
* Program [Visual Studio Code/Visual Studio dla komputerów Mac](https://learn.microsoft.com/pl-pl/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio#tabpanel_11_visual-studio-code+visual-studio-mac)

C#

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

var todoItems = app.MapGroup("/todoitems");

todoItems.MapGet("/", GetAllTodos);

todoItems.MapGet("/complete", GetCompleteTodos);

todoItems.MapGet("/{id}", GetTodo);

todoItems.MapPost("/", CreateTodo);

todoItems.MapPut("/{id}", UpdateTodo);

todoItems.MapDelete("/{id}", DeleteTodo);

app.Run();

static async Task<IResult> GetAllTodos(TodoDb db)

{

return TypedResults.Ok(await db.Todos.ToArrayAsync());

}

static async Task<IResult> GetCompleteTodos(TodoDb db)

{

return TypedResults.Ok(await db.Todos.Where(t => t.IsComplete).ToListAsync());

}

static async Task<IResult> GetTodo(int id, TodoDb db)

{

return await db.Todos.FindAsync(id)

is Todo todo

? TypedResults.Ok(todo)

: TypedResults.NotFound();

}

static async Task<IResult> CreateTodo(Todo todo, TodoDb db)

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

return TypedResults.Created($"/todoitems/{todo.Id}", todo);

}

static async Task<IResult> UpdateTodo(int id, Todo inputTodo, TodoDb db)

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return TypedResults.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return TypedResults.NoContent();

}

static async Task<IResult> DeleteTodo(int id, TodoDb db)

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return TypedResults.NoContent();

}

return TypedResults.NotFound();

}

Kod Map<HttpVerb> wywołuje teraz metody zamiast lambd:

C#

var todoItems = app.MapGroup("/todoitems");

todoItems.MapGet("/", GetAllTodos);

todoItems.MapGet("/complete", GetCompleteTodos);

todoItems.MapGet("/{id}", GetTodo);

todoItems.MapPost("/", CreateTodo);

todoItems.MapPut("/{id}", UpdateTodo);

todoItems.MapDelete("/{id}", DeleteTodo);

Te metody zwracają obiekty, które implementują [IResult](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.http.iresult) obiekty i są definiowane przez [TypedResults](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.http.typedresults)element :

C#

static async Task<IResult> GetAllTodos(TodoDb db)

{

return TypedResults.Ok(await db.Todos.ToArrayAsync());

}

static async Task<IResult> GetCompleteTodos(TodoDb db)

{

return TypedResults.Ok(await db.Todos.Where(t => t.IsComplete).ToListAsync());

}

static async Task<IResult> GetTodo(int id, TodoDb db)

{

return await db.Todos.FindAsync(id)

is Todo todo

? TypedResults.Ok(todo)

: TypedResults.NotFound();

}

static async Task<IResult> CreateTodo(Todo todo, TodoDb db)

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

return TypedResults.Created($"/todoitems/{todo.Id}", todo);

}

static async Task<IResult> UpdateTodo(int id, Todo inputTodo, TodoDb db)

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return TypedResults.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return TypedResults.NoContent();

}

static async Task<IResult> DeleteTodo(int id, TodoDb db)

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return TypedResults.NoContent();

}

return TypedResults.NotFound();

}

Testy jednostkowe mogą wywoływać te metody i testować, czy zwracają prawidłowy typ. Jeśli na przykład metoda to GetAllTodos:

C#

static async Task<IResult> GetAllTodos(TodoDb db)

{

return TypedResults.Ok(await db.Todos.ToArrayAsync());

}

Kod testu jednostkowego może sprawdzić, czy obiekt typu [Ok<Todo[]>](https://learn.microsoft.com/pl-pl/dotnet/api/microsoft.aspnetcore.http.httpresults.ok-1.value#microsoft-aspnetcore-http-httpresults-ok-1-value) jest zwracany z metody obsługi. Na przykład:

C#

public async Task GetAllTodos\_ReturnsOkOfTodosResult()

{

// Arrange

var db = CreateDbContext();

// Act

var result = await TodosApi.GetAllTodos(db);

// Assert: Check for the correct returned type

Assert.IsType<Ok<Todo[]>>(result);

}