**<https://docs.microsoft.com/es-es/aspnet/core/tutorials/min-web-api?view=aspnetcore-6.0&tabs=visual-studio-code>**

**Tutorial: Creación de una API web mínima con ASP.NET Core**

* Artículo
* 28/02/2022
* Tiempo de lectura: 15 minutos

Las API mínimas están diseñadas para crear API HTTP con dependencias mínimas. Son ideales para microservicios y aplicaciones que desean incluir solo los archivos, las características y las dependencias mínimas en ASP.NET Core.

En este tutorial se enseñan los conceptos básicos de la compilación de una API web mínima con ASP.NET Core.

Para ver un tutorial sobre cómo crear un proyecto de API web basado en [controladores](https://docs.microsoft.com/es-es/aspnet/core/web-api/?view=aspnetcore-6.0) que contienen más características, vea [Creación de una API web](https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0).

**Información general**

En este tutorial se crea la siguiente API:

| **INFORMACIÓN GENERAL** | | | |
| --- | --- | --- | --- |
| **API** | **Descripción** | **Cuerpo de la solicitud** | **Response body** |
| GET / | Prueba del explorador, "Hola mundo" | None | "¡Hola mundo!" |
| GET /todoitems | Obtener todas las tareas pendientes | None | Matriz de tareas pendientes |
| GET /todoitems/complete | Obtener tareas pendientes completadas | None | Matriz de tareas pendientes |
| GET /todoitems/{id} | Obtener un elemento por identificador | None | Tarea pendiente |
| POST /todoitems | Incorporación de un nuevo elemento | Tarea pendiente | Tarea pendiente |
| PUT /todoitems/{id} | Actualizar un elemento existente | Tarea pendiente | None |
| DELETE /todoitems/{id} | Eliminación de un elemento | None | None |

* Abra el [terminal integrado](https://code.visualstudio.com/docs/editor/integrated-terminal).
* Cambie los directorios (cd) a la carpeta que va a contener la carpeta del proyecto.
* Ejecute los comandos siguientes:

CLI de .NETCopiar

dotnet new webapi -minimal -o TodoApi

cd TodoApi

code -r ../TodoApi

* Cuando un cuadro de diálogo le pregunte si desea confiar en los autores, seleccione **Sí**.
* Cuando en un cuadro de diálogo se le pregunte si quiere agregar al proyecto los recursos necesarios, seleccione **Sí**.

El comando anterior crea un proyecto de API web mínima y lo abre en Visual Studio Code.

**Examen del código Program.cs**:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

var summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

app.MapGet("/weatherforecast", () =>

{

var forecast = Enumerable.Range(1, 5).Select(index =>

new WeatherForecast

(

DateTime.Now.AddDays(index),

Random.Shared.Next(-20, 55),

summaries[Random.Shared.Next(summaries.Length)]

))

.ToArray();

return forecast;

})

.WithName("GetWeatherForecast");

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)

{

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

}

La plantilla de proyecto crea una API WeatherForecast compatible con WeatherForecast. Swagger se usa para generar una documentación útil y páginas de ayuda para API web.

El código resaltado siguiente agrega compatibilidad con Swagger:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

**Ejecutar la aplicación**

Para confiar en el certificado de desarrollo de HTTPS, ejecute el comando siguiente:

dotnet dev-certs https --trust

El comando anterior no funciona en Linux. Vea la documentación de su distribución de Linux para confiar en un certificado. El comando anterior muestra el siguiente cuadro de diálogo, siempre que el certificado no fuera de confianza previamente:



* Si acepta confiar en el certificado de desarrollo, seleccione **Sí**.

Para obtener más información, vea [Confiar en el certificado de desarrollo de ASP.NET Core HTTPS](https://docs.microsoft.com/es-es/aspnet/core/security/enforcing-ssl?view=aspnetcore-6.0#trust-the-aspnet-core-https-development-certificate-on-windows-and-macos).

Para obtener información sobre cómo confiar en el explorador Firefox, consulte [Error de certificado SEC\_ERROR\_INADEQUATE\_KEY\_USAGE de Firefox](https://docs.microsoft.com/es-es/aspnet/core/security/enforcing-ssl?view=aspnetcore-6.0#trust-ff).

Presione Ctrl+F5 para ejecutar la aplicación. Se abre una ventana del explorador. Anexe /swagger a la dirección URL en el explorador; por ejemplo, https://localhost:7122/swagger.

Se abre la página de Swagger /swagger/index.html. Seleccione **GET > Try it out> Execute**. La página muestra lo siguiente:

* Comando de [Curl](https://curl.haxx.se/) para probar la API WeatherForecast
* Dirección URL para probar la API WeatherForecast
* Código de respuesta, cuerpo y encabezados
* Cuadro de lista desplegable con los tipos de medios y el esquema y valor de ejemplo

Copie y pegue la **URL de solicitud** en el explorador: . Se devuelve un JSON similar al siguiente:

JSONCopiar

[

{

"date": "2021-10-19T14:12:50.3079024-10:00",

"temperatureC": 13,

"summary": "Bracing",

"temperatureF": 55

},

{

"date": "2021-10-20T14:12:50.3080559-10:00",

"temperatureC": -8,

"summary": "Bracing",

"temperatureF": 18

},

{

"date": "2021-10-21T14:12:50.3080601-10:00",

"temperatureC": 12,

"summary": "Hot",

"temperatureF": 53

},

{

"date": "2021-10-22T14:12:50.3080603-10:00",

"temperatureC": 10,

"summary": "Sweltering",

"temperatureF": 49

},

{

"date": "2021-10-23T14:12:50.3080604-10:00",

"temperatureC": 36,

"summary": "Warm",

"temperatureF": 96

}

]

**Actualización del código generado**

Este tutorial se centra en la creación de una API web, por lo que eliminaremos el código de Swagger y el código WeatherForecast. Reemplace el contenido del archivo *Program.cs* con lo siguiente:

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

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

app.Run();

El código resaltado siguiente crea una clase [WebApplicationBuilder](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.builder.webapplicationbuilder) y una clase [WebApplication](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.builder.webapplication) con parámetros predeterminados configurados previamente:

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

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

app.Run();

El código siguiente crea un punto de conexión HTTP GET / que devuelve Hello World!:

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

app.Run(); ejecuta la aplicación.

Quite las dos líneas "launchUrl": "swagger", del archivo "launchUrl": "swagger",. Cuando no se especifica launchUrl, el explorador web solicita el punto de conexión /.

Ejecutar la aplicación. Se muestra Hello World!. El archivo *Program.cs* actualizado contiene una aplicación mínima pero completa.

**Adición de paquetes NuGet**

Se deben agregar paquetes NuGet para admitir la base de datos y los diagnósticos usados en este tutorial.

dotnet add package Microsoft.EntityFrameworkCore.InMemory --prerelease

dotnet add package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore --prerelease

Reemplace ***Program.cs*** por el código siguiente:

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("/", () => "Hello World!");

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.Ok(todo);

}

return Results.NotFound();

});

app.Run();

**// MODELO TODO**

class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

}

Un *modelo* es una clase que representa los datos que la aplicación administra. El modelo para esta aplicación es la clase Todo.

**// CONTEXT**

class TodoDb : DbContext

{

public TodoDb(DbContextOptions<TodoDb> options)

: base(options) { }

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

}

El *contexto de base de datos* es la clase principal que coordina la funcionalidad de [Entity Framework](https://docs.microsoft.com/es-es/ef/core/) para un modelo de datos. Esta clase se crea derivándola de la clase [Microsoft.EntityFrameworkCore.DbContext](https://docs.microsoft.com/es-es/dotnet/api/microsoft.entityframeworkcore.dbcontext).

El código resaltado siguiente agrega el contexto de base de datos al contenedor de [inserción de dependencias (ID)](https://docs.microsoft.com/es-es/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0) y permite mostrar excepciones relacionadas con la base de datos:

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

El contenedor de ID proporciona acceso al contexto de la base de datos y a otros servicios.

El código siguiente crea un punto de conexión HTTP POST /todoitems para agregar datos a la base de datos en memoria:

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

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

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

});

**Instalación de Postman para probar la aplicación**

En este tutorial se usa Postman para probar la API.

* Instale [Postman](https://www.getpostman.com/downloads/).
* Inicie la aplicación web.
* Inicie Postman.
* Deshabilite **Comprobación del certificado SSL**.
  + En **ArchivoConfiguración** (pestaña **General**), deshabilite **Comprobación del certificado SSL**.

**Advertencia**

Vuelva a habilitar la comprobación del certificado SSL tras probar el controlador.

**Prueba de la publicación de datos**

Las instrucciones siguientes publican datos en la aplicación:

* Cree una nueva solicitud.
* Establezca el método HTTP en POST.
* Establezca el URI en https://localhost:<port>/todoitems. Por ejemplo: https://localhost:5001/todoitems
* Seleccione la pestaña **Cuerpo**.
* Seleccione **Sin formato**.
* Establezca el tipo en **JSON**.
* En el cuerpo de la solicitud, introduzca JSON para una tarea pendiente:

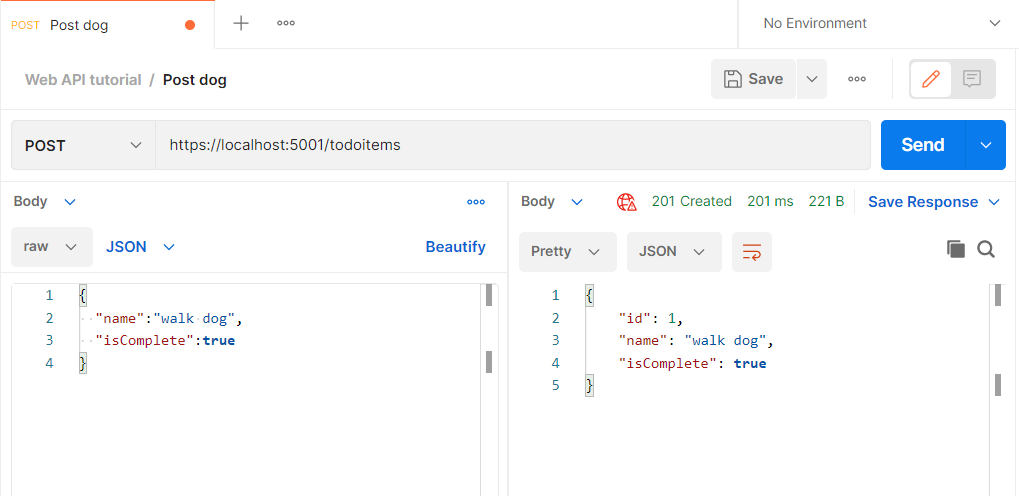
JSONCopiar

{

"name":"walk dog",

"isComplete":true

}

Seleccione **Enviar**. 

**Examen de los puntos de conexión GET**

La aplicación de ejemplo implementa varios puntos de conexión GET mediante llamadas a MapGet:

| **EXAMEN DE LOS PUNTOS DE CONEXIÓN GET** | | | |
| --- | --- | --- | --- |
| **API** | **Descripción** | **Cuerpo de la solicitud** | **Response body** |
| GET / | Prueba del explorador, "Hola mundo" | None | Hello World! |
| GET /todoitems | Obtener todas las tareas pendientes | None | Matriz de tareas pendientes |
| GET /todoitems/{id} | Obtener un elemento por identificador | None | Tarea pendiente |

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

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());

**Prueba de los puntos de conexión GET**

Llame a los dos puntos de conexión desde un explorador o Postman para probar la aplicación. Por ejemplo:

* GET https://localhost:5001/todoitems
* GET https://localhost:5001/todoitems/1

La llamada a GET /todoitems genera una respuesta similar a la siguiente:

[

{

"id": 1,

"name": "Item1",

"isComplete": false

}

]

**Prueba de los puntos de conexión GET con Postman**

* Cree una nueva solicitud.
* Establezca el método HTTP en **GET**.
* Establezca el URI de solicitud en https://localhost:<port>/todoitems. Por ejemplo: https://localhost:5001/todoitems.
* Seleccione **Enviar**.

Esta aplicación utiliza una base de datos en memoria. Si se reinicia la aplicación, la solicitud GET no devuelve ningún dato. Si no se devuelve ningún dato, primero [PUBLIQUE](https://docs.microsoft.com/es-es/aspnet/core/tutorials/min-web-api?view=aspnetcore-6.0&tabs=visual-studio-code#post) los datos en la aplicación.

**Valores devueltos**

ASP.NET Core serializa automáticamente el objeto a [JSON](https://www.json.org/) y escribe el JSON en el cuerpo del mensaje de respuesta. El código de respuesta de este tipo de valor devuelto es [200 OK](https://developer.mozilla.org/docs/Web/HTTP/Status/200), suponiendo que no haya ninguna excepción no controlada. Las excepciones no controladas se convierten en errores 5xx.

Los tipos de valores devueltos pueden representar una gama amplia de códigos de estado HTTP. Por ejemplo, GET /todoitems/{id} puede devolver dos valores de estado diferentes:

* Si no hay ningún elemento que coincida con el identificador solicitado, el método devolverá un código de error de [estado 404](https://developer.mozilla.org/docs/Web/HTTP/Status/404).
* En caso contrario, el método devuelve 200 con un cuerpo de respuesta JSON. Devolver item genera una respuesta HTTP 200.

**Examen del punto de conexión PUT**

La aplicación de ejemplo implementa un único punto de conexión PUT mediante MapPut:

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();

});

Este método es similar al método MapPost, salvo que usa HTTP PUT. Una respuesta correcta devuelve [204 (Sin contenido)](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html). Según la especificación HTTP, una solicitud PUT requiere que el cliente envíe toda la entidad actualizada, no solo los cambios. Para admitir actualizaciones parciales, use [HTTP PATCH](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.httppatchattribute).

**Prueba del punto de conexión PUT**

En este ejemplo se usa una base de datos en memoria que se debe inicializar cada vez que se inicia la aplicación. Debe haber un elemento en la base de datos antes de que realice una llamada PUT. Llame a GET para asegurarse de que hay un elemento en la base de datos antes de realizar una llamada PUT.

Actualice el elemento to-do que tiene el Id = 1 y establezca su nombre en "feed fish":

JSONCopiar

{

"Id": 1,

"name": "feed fish",

"isComplete": false

}

**Examen del punto de conexión DELETE**

La aplicación de ejemplo implementa un único punto de conexión DELETE mediante MapDelete:

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.Ok(todo);

}

return Results.NotFound();

});

Use Postman para eliminar una tarea pendiente:

* Establezca el método en DELETE.
* Establezca el URI del objeto que quiera eliminar (por ejemplo, https://localhost:5001/todoitems/1).
* Seleccione **Enviar**.

**Prevención del exceso de publicación**

Actualmente, la aplicación de ejemplo expone todo el objeto Todo. Las aplicaciones de producción suelen limitar los datos que se escriben y se devuelven mediante un subconjunto del modelo. Hay varias razones para ello y la seguridad es una de las principales. El subconjunto de un modelo se suele conocer como un objeto de transferencia de datos (DTO), modelo de entrada o modelo de vista. En este artículo, se usa **DTO**.

Se puede usar un DTO para:

* Evitar el exceso de publicación.
* Ocultar las propiedades que los clientes no deben ver.
* Omitir algunas propiedades para reducir el tamaño de la carga.
* Acoplar los gráficos de objetos que contienen objetos anidados. Los gráficos de objetos acoplados pueden ser más cómodos para los clientes.

Para mostrar el enfoque del DTO, actualice la clase Todo a fin de que incluya un campo secreto:

public class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public string? Secret { get; set; }

}

El campo secreto debe ocultarse en esta aplicación, pero una aplicación administrativa podría decidir exponerlo.

Compruebe que puede publicar y obtener el campo secreto.

Cree un modelo de DTO:

public class TodoItemDTO

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public TodoItemDTO() { }

public TodoItemDTO(Todo todoItem) =>

(Id, Name, IsComplete) = (todoItem.Id, todoItem.Name, todoItem.IsComplete);

}

Actualice el código para usar TodoItemDTO:

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

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

var app = builder.Build();

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

await db.Todos.Select(x => new TodoItemDTO(x)).ToListAsync());

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

await db.Todos.FindAsync(id)

is Todo todo

? Results.Ok(new TodoItemDTO(todo))

: Results.NotFound());

app.MapPost("/todoitems", async (TodoItemDTO todoItemDTO, TodoDb db) =>

{

var todoItem = new Todo

{

IsComplete = todoItemDTO.IsComplete,

Name = todoItemDTO.Name

};

db.Todos.Add(todoItem);

await db.SaveChangesAsync();

return Results.Created($"/todoitems/{todoItem.Id}", new TodoItemDTO(todoItem));

});

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

{

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

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

todo.Name = todoItemDTO.Name;

todo.IsComplete = todoItemDTO.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.Ok(new TodoItemDTO(todo));

}

return Results.NotFound();

});

app.Run();

public class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public string? Secret { get; set; }

}

public class TodoItemDTO

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public TodoItemDTO() { }

public TodoItemDTO(Todo todoItem) =>

(Id, Name, IsComplete) = (todoItem.Id, todoItem.Name, todoItem.IsComplete);

}

class TodoDb : DbContext

{

public TodoDb(DbContextOptions<TodoDb> options)

: base(options) { }

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

}

Compruebe que no puede publicar ni obtener el campo secreto.

**Diferencias entre las API mínimas y las API con controladores**

* No se admiten filtros: por ejemplo, no se admiten [IAsyncAuthorizationFilter](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncauthorizationfilter), [IAsyncActionFilter](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncactionfilter), [IAsyncExceptionFilter](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncexceptionfilter), [IAsyncResultFilter](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncresultfilter) ni [IAsyncResourceFilter](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.filters.iasyncresourcefilter).
* No se admite el enlace de modelos, es decir, [IModelBinderProvider](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.imodelbinderprovider) y [IModelBinder](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.imodelbinder). Se puede agregar compatibilidad con una corrección de compatibilidad de enlace personalizada.
  + No se admite el enlace desde formularios. Esto incluye el enlace [IFormFile](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.http.iformfile). Tenemos previsto agregar compatibilidad con IFormFile en el futuro.
* No hay compatibilidad integrada con la validación, es decir, [IModelValidator](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.mvc.modelbinding.validation.imodelvalidator).
* No se admiten [elementos de aplicación](https://docs.microsoft.com/es-es/aspnet/core/mvc/advanced/app-parts?view=aspnetcore-6.0) ni el [modelo de aplicación](https://docs.microsoft.com/es-es/aspnet/core/mvc/controllers/application-model?view=aspnetcore-6.0). No hay ninguna manera de aplicar o crear sus propias convenciones.
* No se admite la representación de vistas integrada. Se recomienda usar [Razor Pages](https://docs.microsoft.com/es-es/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-6.0) para representar vistas.
* No se admite [JsonPatch](https://www.nuget.org/packages/Microsoft.AspNetCore.JsonPatch/).
* No se admite [OData](https://www.nuget.org/packages/Microsoft.AspNetCore.OData/).
* No se admite [ApiVersioning](https://www.nuget.org/packages/Microsoft.AspNetCore.Mvc.Versioning/). Vea [esta incidencia](https://github.com/dotnet/aspnet-api-versioning/issues/751) para obtener más información.

**Uso de JsonOptions**

El código siguiente usa [JsonOptions](https://docs.microsoft.com/es-es/dotnet/api/microsoft.aspnetcore.http.json.jsonoptions):

using Microsoft.AspNetCore.Http.Json;

var builder = WebApplication.CreateBuilder(args);

// Configure JSON options

builder.Services.Configure<JsonOptions>(options =>

{

options.SerializerOptions.IncludeFields = true;

});

var app = builder.Build();

app.MapGet("/", () => new Todo { Name = "Walk dog", IsComplete = false });

app.Run();

class Todo

{

// These are public fields instead of properties.

public string? Name;

public bool IsComplete;

}

El código siguiente usa [JsonSerializerOptions](https://docs.microsoft.com/es-es/dotnet/api/system.text.json.jsonserializeroptions):

using System.Text.Json;

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

var options = new JsonSerializerOptions(JsonSerializerDefaults.Web);

app.MapGet("/", () => Results.Json(new Todo {

Name = "Walk dog", IsComplete = false }, options));

app.Run();

class Todo

{

public string? Name { get; set; }

public bool IsComplete { get; set; }

}

El código anterior usa [valores predeterminados web](https://docs.microsoft.com/es-es/dotnet/standard/serialization/system-text-json-configure-options#web-defaults-for-jsonserializeroptions), que convierte los nombres de propiedad en una combinación de mayúsculas y minúsculas.

**Prueba de la API mínima**

Para obtener un ejemplo de prueba de una aplicación de API mínima, consulte [este ejemplo de GitHub](https://github.com/davidfowl/CommunityStandUpMinimalAPI/blob/main/TodoApi.Tests/TodoTests.cs).

**Recursos adicionales**

* [Introducción a las API mínimas](https://docs.microsoft.com/es-es/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0)