Choose between controller-based APIs and minimal APIs

Article • 04/11/2023

ASP.NET Core supports two approaches to creating APIs: a controller-based approach and minimal APIs. *Controllers* in an API project are classes that derive from ControllerBase. *Minimal APIs* define endpoints with logical handlers in lambdas or methods. This article points out differences between the two approaches.

The design of minimal APIs hides the host class by default and focuses on configuration and extensibility via extension methods that take functions as lambda expressions. Controllers are classes that can take dependencies via constructor injection or property injection, and generally follow object-oriented patterns. Minimal APIs support dependency injection through other approaches such as accessing the service provider.

Here's sample code for an API based on controllers:

C#

```csharp

namespace APIWithControllers;

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddControllers();
        var app = builder.Build();

        app.UseHttpsRedirection();

        app.MapControllers();

        app.Run();
    }
}
```

C#

```csharp
using Microsoft.AspNetCore.Mvc;

namespace APIWithControllers.Controllers;
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
    };

    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
        _logger = logger;
    }

    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
    {
        return Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = Summaries[Random.Shared.Next(Summaries.Length)]
        })
        .ToArray();
    }
}
```

The following code provides the same functionality in a minimal API project. Notice that the minimal API approach involves including the related code in lambda expressions.

C#

```
namespace MinimalAPI;

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        var app = builder.Build();

        app.UseHttpsRedirection();

        var summaries = new[]
        {
            "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
        };

        app.MapGet("/weatherforecast", (HttpContext httpContext) =>
        {
            var forecast = Enumerable.Range(1, 5).Select(index =>
                new WeatherForecast
                {
                    Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
                    TemperatureC = Random.Shared.Next(-20, 55),
                    Summary = summaries[Random.Shared.Next(summaries.Length)]
                })
                .ToArray();
            return forecast;
        });

        app.Run();
    }
}
```

Both API projects refer to the following class:

C#

```
namespace APIWithControllers;

public class WeatherForecast
{
    public DateOnly Date { get; set; }

    public int TemperatureC { get; set; }

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

    public string? Summary { get; set; }
}
```

Minimal APIs have many of the same capabilities as controller-based APIs. They support the configuration and customization needed to scale to multiple APIs, handle complex routes, apply authorization rules, and control the content of API responses. There are a few capabilities available with controller-based APIs that are not yet supported or implemented by minimal APIs. These include:

- No built-in support for model binding (IModelBinderProvider, IModelBinder). Support can be added with a custom binding shim.
- No built-in support for validation (IModelValidator).
- No support for application parts or the application model. There's no way to apply or build your own conventions.
- No built-in view rendering support. We recommend using Razor Pages for rendering views.
- No support for JsonPatch
- No support for OData

See also

- Create web APIs with ASP.NET Core.
- Tutorial: Create a controller-based web API with ASP.NET Core
- Minimal APIs overview
- Tutorial: Create a minimal API with ASP.NET Core