Structuring Your Minimal API

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Establishing the problem

Investigating options for structuring minimal APIs

Options for Structuring Minimal APIs

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Using Methods Instead of Inline Handlers

Improves maintainability and testability

Program.cs

```
// existing code
dishesEndpoints.MapGet("", async Task<Ok<IEnumerable<DishDto>>> (
    DishesDbContext dishesDbContext,
    IMapper mapper,
    ClaimsPrincipal claimsPrincipal,
    string? name) =>
{
    Console.WriteLine($"User authenticated? {claimsPrincipal.Identity?.IsAuthenticated}");

    return TypedResults.Ok(mapper.Map<IEnumerable<DishDto>>(await dishesDbContext.Dishes
        .Where(d => name == null || d.Name.Contains(name)).ToListAsync()));
});
```

Using Methods Instead of Inline Handlers

Improves maintainability and testability

Program.cs

```
// improved code
dishesEndpoints.MapGet("", GetDishesAsync);

async Task<Ok<IEnumerable<DishDto>>> GetDishesAsync(
    DishesDbContext dishesDbContext,
    IMapper mapper,
    ClaimsPrincipal claimsPrincipal,
    string? name)
{
    Console.WriteLine($"User authenticated? claimsPrincipal.Identity?.IsAuthenticated}");

    return TypedResults.Ok(mapper.Map<IEnumerable<DishDto>>(await dishesDbContext.Dishes
        .Where(d => name == null || d.Name.Contains(name)).ToListAsync()));
}
```

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Separating Handler Methods Out in Classes

Cleans up Program.cs

DishesHandlers.cs

```
public static class DishesHandlers
{
    public static async Task<Ok<IEnumerable<DishDto>>> GetDishesAsync(
        DishesDbContext dishesDbContext,
        IMapper mapper,
        ClaimsPrincipal claimsPrincipal,
        string? name)
    {
        …
    }
}
```

Separating Handler Methods Out in Classes

Cleans up Program.cs

Program.cs

```
dishesEndpoints.MapGet("", DishesHandlers.GetDishesAsync);
```

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Extending IEndpointRouteBuilder

Improves code structure

EndpointRouteBuilderExtensions.cs

```
public static class EndpointRouteBuilderExtensions
{
    public static void RegisterDishesEndpoints(
        this IEndpointRouteBuilder app)
    {
        var dishesEndpoints = app.MapGroup("/dishes");

        dishesEndpoints.MapGet("",  async Task<Ok<IEnumerable<DishDto>>> GetDishesAsync(
             DishesDbContext dishesDbContext,
             IMapper mapper,
             ClaimsPrincipal claimsPrincipal,
             string? name)
             { … });
     }
}
```

Extending IEndpointRouteBuilder

Improves code structure

Program.cs

```
app.RegisterDishesEndpoints();

// other extension methods
app.RegisterIngredientsEndpoints();
```

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Combining 1, 2 and 3

Combine the Previous Approaches

Improves testability, maintainability, reusability and overall code structure

EndpointRouteBuilderExtensions.cs

```
public static class EndpointRouteBuilderExtensions
{
    public static void RegisterDishesEndpoints(this IEndpointRouteBuilder app)
    {
        var dishesEndpoints = app.MapGroup("/dishes");
        var dishWithGuidIdEndpoints = dishesEndpoints.MapGroup("/{dishId:guid}");

        dishesEndpoints.MapGet("", DishesHandlers.GetDishesAsync);
        dishWithGuidIdEndpoints.MapGet("", DishesHandlers.GetDishByIdAsync);
        dishesEndpoints.MapGet("/{dishName}", DishesHandlers.GetDishByDishNameAsync);
    }
}
```

Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Combining 1, 2 and 3

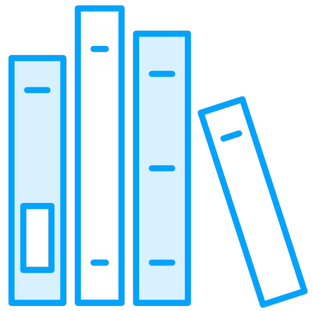
Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Combining 1, 2 and 3



3rd party libraries

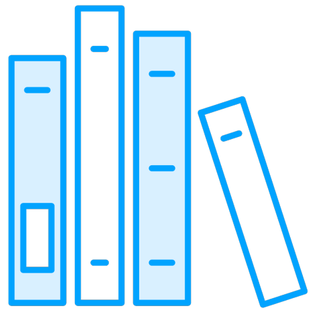
Options for Structuring Minimal APIs

Using methods instead of inline handlers

Separating handler methods out in classes

Extending `IEndpointRouteBuilder`

Combining 1, 2 and 3



3rd party libraries

Other approaches

Demo

Extending `IEndpointRouteBuilder` to structure your minimal API

Summary

Different options exist for structuring your minimal API code

- A good option is extending `IEndpointRouteBuilder`, combined with endpoint handlers as methods in separate classes

Up Next:

Handling Exceptions and Logging