ASP.NET Core - True Ultimate Guide

Section 30: Minimal API - Notes

Minimal APIs in ASP.NET Core

Minimal APIs in ASP.NET Core allow you to create HTTP APIs with minimal code and configuration. They simplify the process of building APIs by reducing the boilerplate code typically required with traditional controller-based approaches.

1. Minimal API Overview

Minimal APIs provide a streamlined way to define routes and handle HTTP requests directly in the Program.cs file, eliminating the need for controllers and action methods. This approach is especially useful for small, microservices, or applications where you want to reduce overhead.

2. Defining Minimal APIs

Basic Example:

Here's how to set up a basic minimal API in Program.cs:

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();


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


app.Run();
```

- **MapGet**: Maps an HTTP GET request to a handler. In this example, a request to /hello returns the string "Hello, world!".

3. Route Parameters

You can define route parameters to capture values from the URL and use them in your handlers.

Example:

```
app.MapGet("/greet/{name}", (string name) => $"Hello, {name}!");
```

- **{name}**: Captures the route parameter from the URL. If you access /greet/Alice, the handler will return "Hello, Alice!".

4. MapGroups

MapGroups is used to group related routes together. This can be helpful for organizing routes that share a common prefix.

Example:

```
app.MapGroup("/api")
  .MapGet("/products", () => new[] { "Product1", "Product2" })
  .MapGet("/orders", () => new[] { "Order1", "Order2" });
```

- **MapGroup**: Groups routes under the /api prefix. Routes are then mapped to endpoints such as /api/products and /api/orders.

5. IResult

IResult represents the result of an HTTP request and can be used to return various types of responses.

Example:

```
app.MapGet("/status", () =>
{
   return Results.Ok(new { Status = "Running" }); // Return 200 OK with JSON response
});
```

- **Results.Ok**: Creates a result that indicates a successful HTTP response with a status code of 200 and includes a JSON payload.

Other common IResult methods include:

- Results.NotFound() for a 404 Not Found response.
- Results.BadRequest() for a 400 Bad Request response.
- Results.Created() for a 201 Created response.

6. Endpoint Filters

Endpoint filters allow you to run custom logic before or after the request handler is executed. They can be used for tasks like validation, logging, or authentication.

Example of a Simple Endpoint Filter:

```
public class LoggingFilter : IEndpointFilter
{
   public Task<object?> InvokeAsync(EndpointFilterInvocationContext context,
EndpointFilterInvocationDelegate next)
   {
      Console.WriteLine("Handling request...");
      return next(context);
   }
}

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/data", () => "Some data")
  .AddEndpointFilter<LoggingFilter>();

app.Run();
```

- **IEndpointFilter**: Interface used to create filters that can be added to endpoints.

7. IEndpointFilter Interface

The IEndpointFilter interface is used to create custom filters that can be applied to endpoints. This allows you to inject logic into the request processing pipeline.

Example of Custom Filter Implementation:

```
public class CustomFilter : IEndpointFilter
{
   public Task<object?> InvokeAsync(EndpointFilterInvocationContext context,
EndpointFilterInvocationDelegate next)
   {
      // Custom logic before request handling
      Console.WriteLine("Custom filter logic before handler.");

      // Call the next filter or endpoint
      var result = next(context);

      // Custom logic after request handling
      Console.WriteLine("Custom filter logic after handler.");

      return result;
   }
}
```

Detailed Explanation of Code

- **Minimal APIs**: Provide a way to define routes and request handlers directly in Program.cs, simplifying the API development process.
- **Route Parameters**: Capture values from URLs and use them in request handlers.
- **MapGroups**: Organize related routes under a common prefix.
- **IResult**: Represents HTTP responses and can be used to return various types of responses (OK, NotFound, etc.).
- **Endpoint Filters**: Allow custom logic to be executed before or after request handlers.
- **IEndpointFilter**: Interface for creating custom endpoint filters.

Key Points to Remember

1. **Minimal APIs**: Simplify API development with direct route and handler definitions in Program.cs.
2. **Route Parameters**: Use {param} syntax to capture values from URLs.
3. **MapGroups**: Group related routes under a common prefix for better organization.
4. **IResult**: Return various HTTP responses using built-in methods like Results.Ok, Results.NotFound, etc.
5. **Endpoint Filters**: Implement custom logic in request handling using IEndpointFilter.