ASP.NET CONTINUED

AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

HENRIK BITSCH KIRK
ASSOCIATE PROFESSOR

AGENDA

- Routing
- HTTPClient
- Minimal API
- Swagger / Open API

ROUTING

When building APIs use attribute routing.

- Setup with app.MapControllers()

- This maps controllers and routes with default settings
  - E.g. for WeatherForecastController -> `{controller=WeatherForecast}/{action=Index}/{id?}`
- Customize routing with [Route]
  - On Controller e.g. `[Route("api/[controller]")]`
  - Or `[Route("api/weather")]`
- On Endpoints
  - `[HttpPost("delete")]`
  - Or `[Route("delete")]`
  - These will be appended on the route

PARAMETERS

We have already seen some

```
[HttpGet("{name}", Name = "GetWeatherForecast")]
public ActionResult<WeatherForecast> GetSingle(string name) {
```

- The above one will add name to URL
- Could also be added as URL paramter (without {name})

```
[HttpGet("withName", Name = "GetWeatherForecast")]
public ActionResult<WeatherForecast> GetSingle(string name) {
```

- This is called with ?name=yourGivenName

EXERCISES

Start working on **1-3**

Then continue on the **Bonus** exercise

HTTPCLIENT

This class is used to access an HTTP endpoint from an application.

- Can be create with or without BaseAdddress

```
private HttpClient httpClient = new HttpClient();
```

Or

```
private HttpClient httpClient2 = new HttpClient() {
   BaseAddress = new Uri("https://jsonplaceholder.typicode.com")
};
```

WHAT IS HTTPCLIENT

- General purpose class for accessing HTTP endpoints

- Handles Json automatically, but also methods for
  - Byte[], Stream, String
  - Note: Json methods are all extensions methods

- Methods for all HTTP verbs (GET, POST, PUT, PATCH, DELETE)

HTTP VERBS

- GET

  -
    ```
    await httpClient.GetStringAsync(
                                 "https://jsonplaceholder.typicode.com/posts");
    ```

  -
    ```
    await httpClient2.GetStringAsync("posts");
    ```
- POST

  -
    ```
    stringContent jsonContent = new(JsonSerializer.Serialize(
           new { userId = 77, id = 1, title = "write code sample",
                      completed = false }
    ), Encoding.UTF8, "application/json")
    await httpClient2.PostAsync("todos", jsonContent)
    ```
- PUT

  -
    ```
    await httpClient2.PutAsync("todos/1", jsonContent)
    ```

WORKING WITH JSON

- All these methods are from the extension method namespace
  - `using System.Net.Http.Json`
- Reading
  - `await httpClient2.GetFromJsonAsync<List<Todo>>("todos")`
- Writing

  -
    ```
    await httpClient2.PostAsJsonAsync("todos",
               new {UserId: 9, Id: 99, Title: "Show extensions", Completed:
    false});
    ```

EXERCISES

Continue on 4-7

Then continue on the **Bonus** exercise

MINIMAL API

ASP.NET WebAPI is a framework with lots and lots of features

- Security, routing, middlewhere, etc etc etc.
- *SW4BED – for all the details on database and server applications*

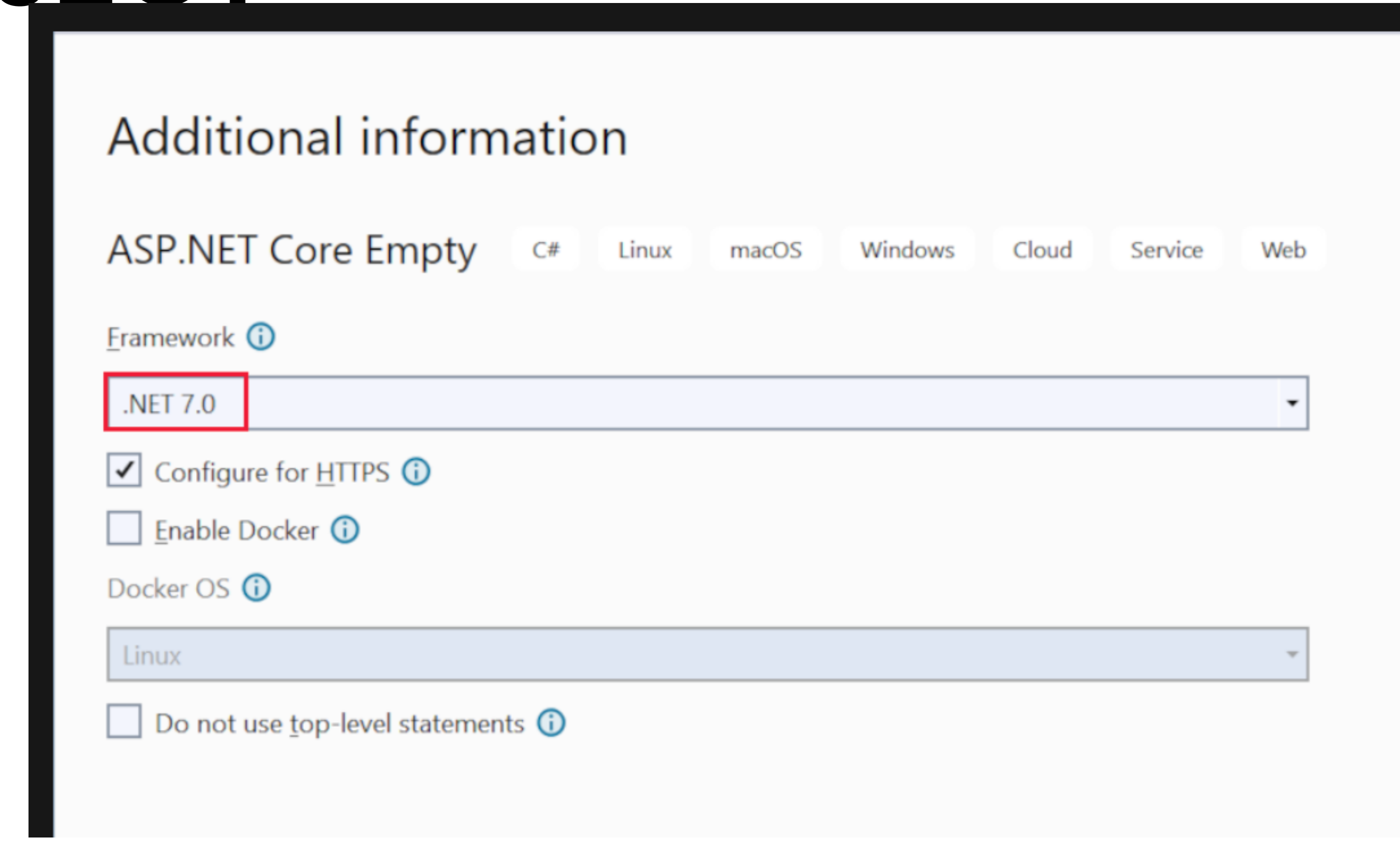
Sometimes you need something a little simpler

- Minimal ASP.NET WebAPI is .NETs solution to this

CREATING THE PROJECT

Create a project as normal but uncheck 'Do not use top-level statements'.

This should create a project with only a Program.cs file.



Additional information

ASP.NET Core Empty

Framework: .NET 7.0

[✓] Configure for HTTPS

[ ] Enable Docker

Docker OS: Linux

[ ] Do not use top-level statements

HELLO WORLD

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

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

app.Run();
```

ADDING AN DATABASE HERE

For an InMemory database (one that don't persists across restarts)

1. Add NuGET package 'Microsoft.EntityFrameworkCore.InMemory'

2. Create a Context file (way to communicate with database)

3. Add this to your builder in Program.cs
```
builder.Services.AddDbContext<Db>(opt =>
                             opt.UseInMemoryDatabase("Data"));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
```
4. Use data in your endpoints
```
app.MapGet("/items", async (Db db) => await db.Todos.ToListAsync());
```

CONTEXT FILE

```
// DB.cs

using Microsoft.EntityFrameworkCore;

class Db : DbContext
{
    public TodoDb(DbContextOptions<Db> options)
        : base(options) { }

    public DbSet<Data> Data => Set<Data>();
}
```

HTTP VERBS IN MINIMAL

- GET
  app.MapGet(…)

- POST
  app.MapPost(…)

- PUT
  app.MapPut(…)

- DELETE
  app.MapDelete(…)

- Note routes needs to be unique

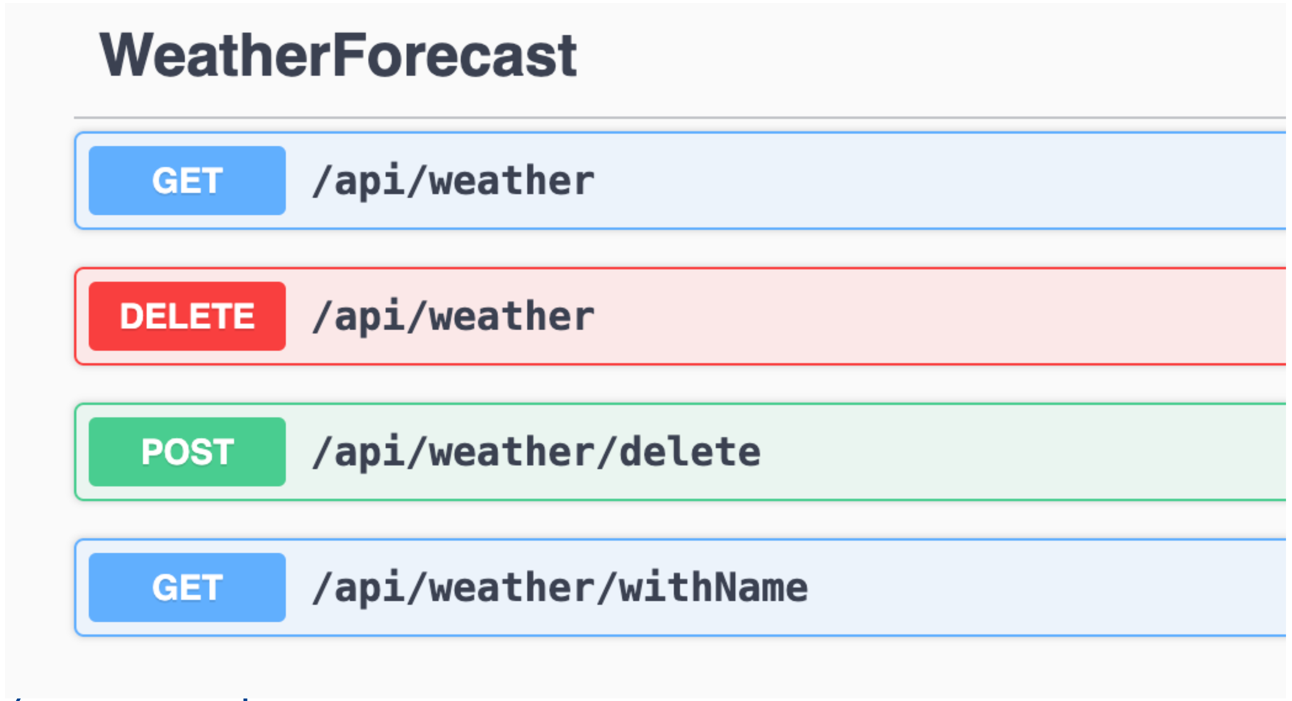
SWAGGER / OPEN API

- Swagger can also be added in a Minimal API

```
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

SWAGGER UI

- Builds on top of Open API
- Specification off an HTTP Web API.
  - All open endpoints
- Described in a Json fil
  - http://localhost:5207/swagger/v1/swagger.json
  - Contains endpoints, input, output, and return codes


WeatherForecast

GET /api/weather

DELETE /api/weather

POST /api/weather/delete

GET /api/weather/withName

EXERCISES

Continue on 8-

Then continue on the **Bonus** exercise

AARHUS
UNIVERSITY