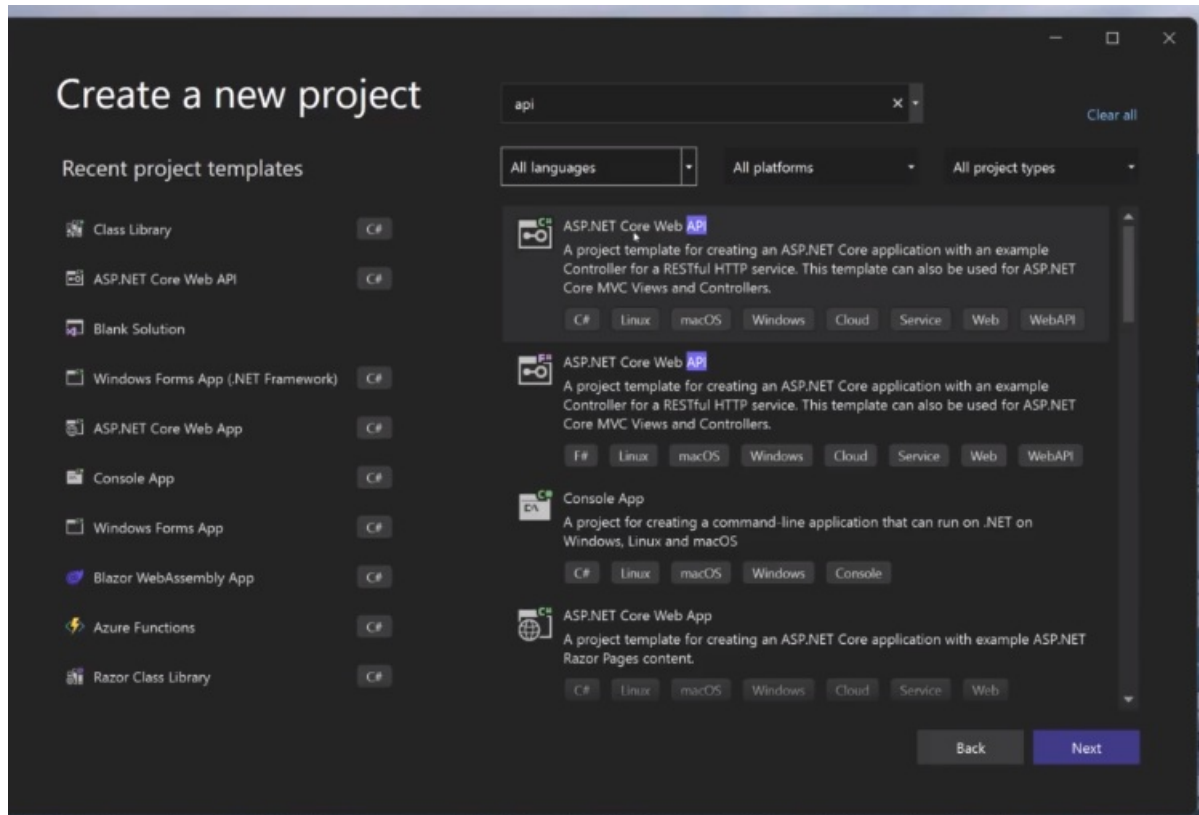


# Udemy course: Ultimate ASP.NET pt. 2

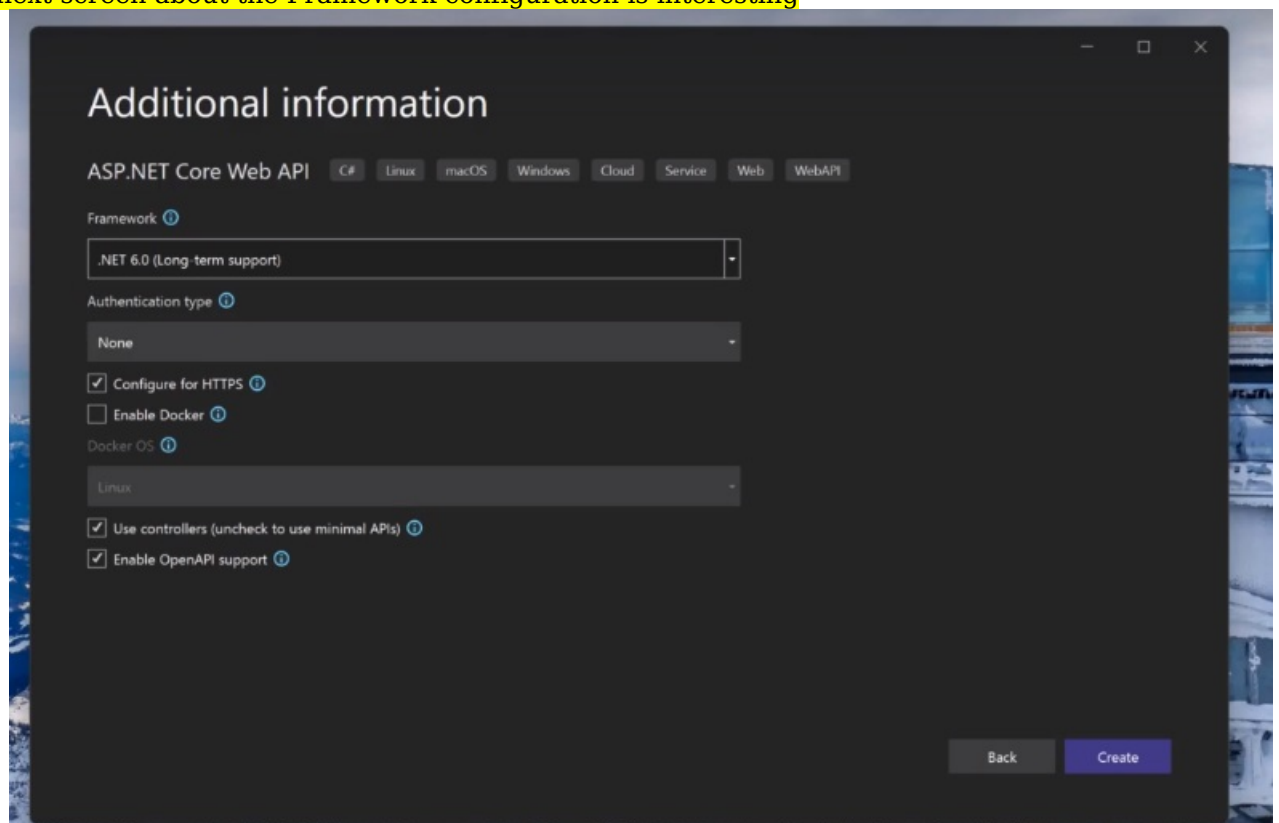
## Projekt Setup and Configuration

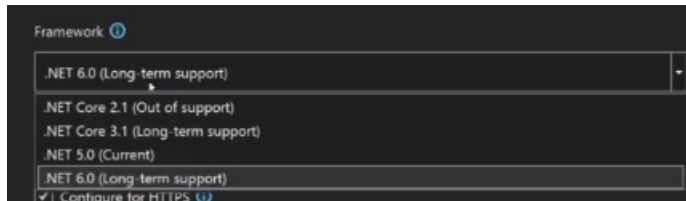
### Create a ASP.NET Core API Project in Visual Studio (in Windows!)

- in Visual Studio 2019 or 2021



- we select ASP.NET Core Web API
- name example: HotelListing.API
- Project name automatically sets the Solution name, but you can remove .ASP in Solution name
- the next screen about the Framework configuration is interesting

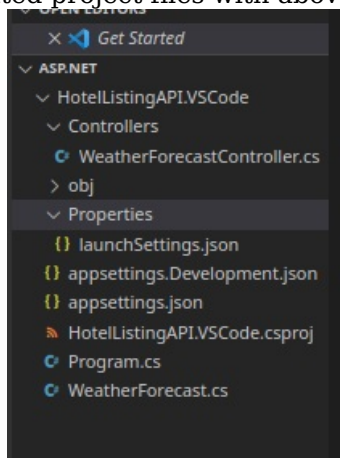




- course instructor is proceeding with .NET 6.0
  - but on .NET 5.0 it is mostly the same - only few differences the course instructor will speak about
- Authentication type
  - the one the course instructor wants to use is not available
  - so we choose None
  - Microsoft Identity platform
    - this would be with a identity server as your id
    - we won't be getting into that right now
  - Windows
    - in a corporate setting and you want to use Active Directory
    - or local Active Directory for your authentication
- Enable Docker
  - we will not enable that for now
- Use controller (uncheck to use minimal APIs)
  - only  $\geq$  .NET 6
- we enable OpenAPI support
  - gives access to Swagger documentation
  - easy out of the box way to document your API
- 

#### #7: Alternative: create ASP.NET Core Project in Visual Studio Code

- most or even all of the things done in the course can be done in Visual Studio Code as well
- install .NET 6 SDK on your environment
  - debian linux instructions: <https://docs.microsoft.com/de-de/dotnet/core/install/linux-debian#debian-10->
- entering in terminal
  - `$ dotnet --info`
  - should list information of the version
- to create a project enter following command
  - `dotnet new webapi -o HotelListingAPI.VSCode`
    - here 'webapi' is a templatename
    - -o for output
- main difference: Visual Studio vs Visual Studio Code development with ASP.NET
  - one has functionality in the UI
  - the other you need to use more the command line interface
- created project files with above command:



#### #8: Explore ASP.NET Core API Project and Explore Swagger UI

- all code and debugging we are carrying out on Visual Studio, can be replicated in Visual Studio Code
- about the files
  - Properties/launchSettings.json
    - usually not to be edited

- only very rarely
  - sometimes you would add new environment variables
  - usually not required to master this file
  - only modify it, when you know what you are doing!
- MVC
  - Model View Controller
  - Model of the data
  - View about what the user sees
  - Controller: pulls the strings between the model and view
    - \* gets request, processes it, sends a response
- code in controller:

```
[ApiController]

[Route("[controller]")]
```

- define how do we go to the controller name
  - this means just we use the name of the controller
  - for example when we are testing the API: this define how you get to that controller
- when you are calling the API you don't know anything about the code
- ```
[HttpGet(Name = "GetWeatherForecast")]
```
- when you send a request with the controller name - in the example WeatherForecast - /GetWeatherForecast
  - it is like calling that method
  - this method then returns the data
- this is a simple example
- another file: appsettings.json

```
{

  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore":
        "Warning"
    }
  },

  "AllowedHosts":
    "*"
}
```

- certain settings for development purposes
- .NET5 vs .NET6 differences
  - .NET6 more minimalistic mindset
  - difference e.g. in Program.cs file
  - takes away lot of defining of different namespaces
  - in .NET 6 a lot of constructs were introduced to reduce all of that
  - more on differences in the next video
  - all services to be configured are between the builder declaration in Program.cs and the Build() command

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

// Add services to the
container.
```

```

builder.Services.AddControllers();

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

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

```

- what happens here:
  - the builder is constructing all the services that need to be injected
  - before or by the time the app is run
  - all of those things need to be in place
  - so they can be accessed
  - that is what we call the AOC container
    - or inversion of Control Container
    - that is what needs us to do our dependency injection
    - better explanation on that later
- we are letting the app know
  - it needs to use controllers
  - it needs to use endpoints
  - API explorer
  - needs to use swagger engine
- then after the Build() command
  - configure the middleware
  - like request pipeline
  - we want to use Swagger in Development
  - we can use Authorization, MapController etc
  - finally we Run()
- before we run we can also introduce customized middleware
  - which we will be looking at also
  - and introduced that to the pipeline if we need to
- then we have our **model**: WeatherForecast.cs
  - it looks like the data should look like

```

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

    public int TemperatureC { get; set; }

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

    public string? Summary { get; set; }
}

```

- how to run it in Visual Studio Code?
  - error message "Scriptcs not found"

- Settings->Run code configuration
  - find "Run in terminal"; enable that
- via the "Open Settings" open the Json and add type "`code-runner.executorMap`" and then press enter
- below C# add "`cd $dir && dotnet run $fileName`" and save
- in the terminal it will indicate where the server is running e.g. at
  - <https://localhost:7213>
  - with this port you can see the swagger API documentation at
    - <https://localhost:7213/swagger/index.html>
  - you can click for method /WeatherForecast -> Try it out -> Execute
  - you will get a response

## #9 .NET 6 vs previous versions

- .NET5 support is over quite soon/ or already behind us (when the video was captured it was 5 months away)
- .NET6 will have longterm support
  - so better start new projects with .NET6!
- a major difference:
  - in .NET5 you have a Startup.cs file, and Program.cs
  - Program.cs looks like it is built with any version before .NET6
  - you have your Main function
  - the main function executes another function etc.
- another major difference:
  - you have builder.Services in .NET6 (in Program.cs) instead of just services in Startup.cs
  - that WebBuilder is inside Program.cs inside method "CreateHostBuilder" in .NET5
  - in the Configure method of Startup.cs are the pipeline objects etc as in Program.cs in .NET6
- .NET6 vs. .NET5 look different, but are basically very much the same!
- most of the things in the course can be done in both .NET6 and .NET5
  - where it is not possible or completely compatible, the course author will point it out
  - everything you are able to do in .NET5 you are able to do in .NET6

## #10: CORS configuration

- CORS:
  - Cross Origin Resource Sharing
  - so our API can be accessed by resource by clients that are not on the same server
  - e.g. you deployed it in your company or on the internet
  - and you want others to use your API to access information
- in Program.cs we are adding following line:
  -

```
builder.Services.AddCors(options => {
    options.AddPolicy("AllowAll", b => b.AllowAnyHeader().AllowAnyOrigin().AllowAnyMethod());
});
```

- "AllowAll" is just our tag name
- b: our actually security policies
- we actually don't have to set that in the application project; we could also change settings on our firewall or other security tools on the network!
  - however you can allow certain APIs, certain methods from specific services etc.
  - instead we are giving access to all the resources
- below we add the line:
  -

```
app.UseCors("AllowAll");
```

- we need to put the settings when
  - other systems want to access our API

## #11: Configure Serilog and Seq

- logging very important for API development
- we want to use an UI as an aggregator

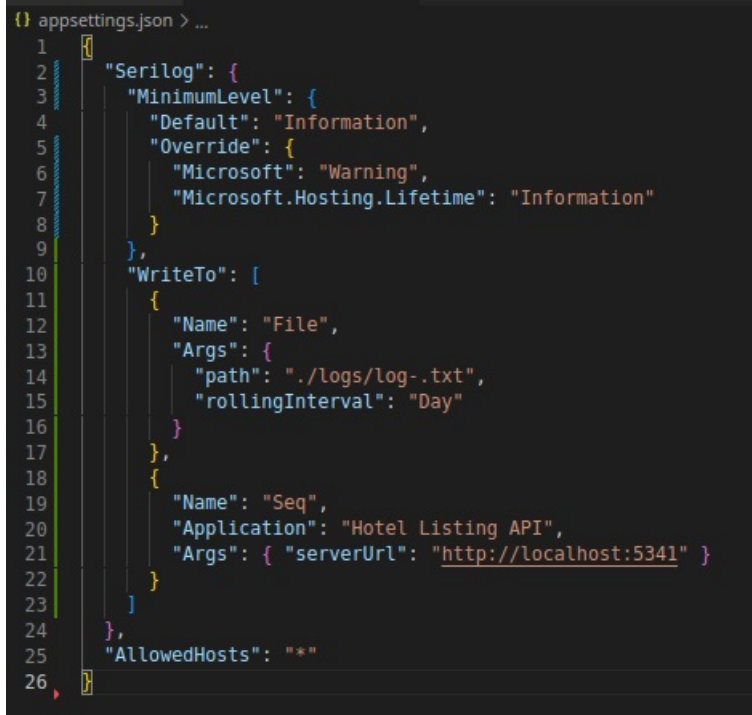
- we will use "sea"
  - a free aggregator for logging
- in Visual Studio: search for package Serilog.AspNetCore
- in Visual Studio Code:
  - \$ dotnet add package Serilog.AspNetCore
  - \$ dotnet add package Serilog.Expressions
  - and Serilog.Sinks.Seq
- it can also directly log to a database
  - package Serilog.Sinks.Console, Rolling File, Seq
- Serilog is quite powerful, you can explore
  - support for several different outputs
- Note: for installed packages
  - you get a new line in csproj file below package references
  -
- in our Program.cs
  - ctx: the context
  - ctx.Configuration: represents our configuration file that are in the project
  - we will have to do some configuration in our app settings json
  - we add the lines:

```
using Serilog; // at the top

// before builder.Build()...

builder.Host.UseSerilog((ctx, lc) => lc.WriteTo.Console().ReadFrom.Configuration(ctx.Configuration));
```

- lc: stands for logger configuration
- we just write it to our console, to see the output while running
- in appsettings.json
  - there is some code we don't need, which list Logging libraries which are builtin
    - we remove the existing "Logging" section
  - instead we add:
    - "Serilog" section like the following screenshot shows
    - for more information: read the documentation



```
{
  "Serilog": {
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    "WriteTo": [
      {
        "Name": "File",
        "Args": {
          "path": "./logs/log-.txt",
          "rollingInterval": "Day"
        }
      },
      {
        "Name": "Seq",
        "Application": "Hotel Listing API",
        "Args": { "serverUrl": "http://localhost:5341" }
      }
    ]
  },
  "AllowedHosts": "*"
}
```

- Seq: Machine data for humans
  - nice Web interface
  - allows you to sit through large logs
  - simple enough to get started with
- a free license is included
- you can download at: <https://datalust.co/download>
- to run with docker inside Linux:

```
docker run --name seq -d --restart unless-stopped -e ACCEPT_EULA=Y -p 5341:80 datalust/seq:latest
```

- to see the logging output you can go to
  - <http://localhost:5341>
  - in our example configuration
  - this is great when you have multiple applications
  - to have one place, to see it all
- after executing some requests via the swagger interface: <https://localhost:7213/swagger/index.html>
- the logging file should be filled
  - below logging
  - under Windows: you might not be able to open the file, while the application is running
  - on <http://localhost:5341/#/events> Seq should list the Events
    - in case of low free disk space, Events might be dropped - see possible warning messages inside Seq
    - there is a setting of minimum free disk space inside Seq

---

other Notes - not from Udemy course:

- run ASP.NET app with docker
  - <https://code.visualstudio.com/docs/containers/quickstart-aspnet-core>
- to build/run in Visual Studio Code without scripts:
  - how to run it in **Visual Studio Code?**
    - error message "Scripts not found"
    - Settings->Run code configuration
      - find "Run in terminal"; enable that
    - via the "Open Settings" open the Json and add type "`code-runner.executorMap`" and then press enter
    - **below C# add "cd \$dir && dotnet run \$fileName" and save**
- if there is a error like this:
  - ```
Only one compilation unit can have
top-level statements.
```
  - it means you have toplevel C-sharp code in more than one file!
- dotnet new gitignore
  - to create the gitignore file
- about the visual studio Code, tempCodeRunnerFile
  - turn it off in `File->Preference->Settings`
  - "code-runner.ignoreSelection": true