3. Your First Application

Visual Studio

CodeLab 1.1

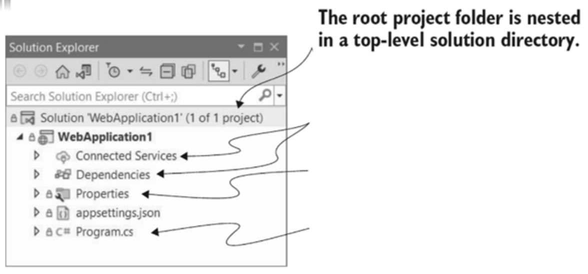
- For most examples in the course:
  - Use Visual Studio Community 2022
  - Use ASP.NET Core 8 with .NET 8
  - Use Entity Framework 6

Basic Steps of Application Development

CodeLab 1.2

- Create
  - Create the base application from a **template** to get started
- Restore
  - Restore all the packages and dependencies to the local project folder using **NuGet**.
- Build
  - Compile the application, and generate all the necessary artifacts
- Run

Project Layout （一）解决方案和项目



The root project folder is nested in a top-level solution directory.

Solution, Project

- In .NET, a project is a unit of deployment, which will be compiled into a .dll file or an executable.
- Each separate app is a separate project.
- A solution can contain multiple projects.

The .csproj Project File 项目文件（一）

- Visual Studio doesn't show the project file explicitly.
- The project file is the most important file.
- This file describes how to build your project and lists any additional NuGet packages that it requires.

The .csproj Project File 项目文件（二）

The SDK attribute specifies the type of project you're building

The TargetFramework is the framework you'll run on

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <RootNamespace>CH01_HelloWorld</RootNamespace>
  </PropertyGroup>
</Project>
```

Enables the C# 8, 10 feature

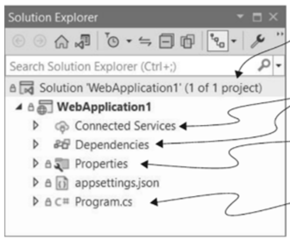
The .csproj Project File项目文件（三）

■**NuGet** is the library package manager for .NET, where libraries are packaged in NuGet.

■Below add a NuGet reference to the project.

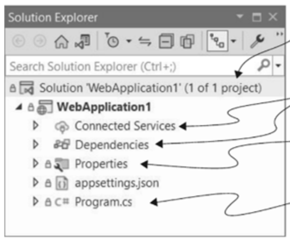
```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="NewtonSoft.Json" Version="13.0.1" />
  </ItemGroup>
</Project>
```

Project Layout （二）特殊节点



The Connected Services and Dependencies nodes do not exist on disk.

Project Layout （三）应用入口代码



Program.cs controls the startup and configuration of your application at runtime.

Program.cs File —— Startup Code

■All ASP.NET Core applications start life as a .NET Console application.

■In .NET, it means a Program class written with top-level statements instead of inside a Main function.

```
using System;
namespace MyApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Main function

```
Console.WriteLine("Hello World!");
```

Top-level statements

Program.cs File —— An Example

Creates a WebApplicationBuilder using the CreateBuilder method

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

Builds and returns an instance of WebApplication from the WebApplicationBuilder

```
var app = builder.Build();
```

Defines an endpoint for your application, which returns Hello World! when the path "/" is called

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

Runs the WebApplication to start listening for requests and generating responses

```
app.Run();
```

Program.cs File —— Builder Pattern

- ■**Builder pattern**: use a builder object to configure a complex object.
- ■Use **WebApplication** object to define how the app handlers and responds to requests.

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddHttpLogging(…);
builder.Logging.AddFilter(…);
var app = builder.Build();
if (app.Environment.IsDevelopment())
{    app.UseHttpLogging();     }
app.MapGet("/", () => "Hello World!");
app.Run();
```

30

Program.cs File —— Modular Design

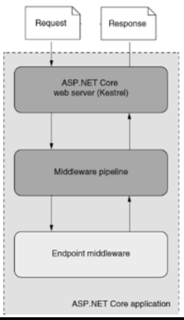
- ■ASP.NET Core uses small modular components for each distinct feature.
- ■These modular components are exposed as one or more services.
- ■Services must be registered by the Services property of the builder object.

```
builder.Services.AddHttpLogging(opts => opts.LoggingFields
 = HttpLoggingFields.RequestProperties);
```

31

Handling Requests with Middleware and Endpoints （一）

- ■The WebApplication instance defines how your application handles and responds to requests, using two building blocks:
- ■**Middleware** are small components execute in sequence when the application receives an HTTP request.
- ■**Endpoints** define how the response should be generated for a specific request to a URL.



32

Handling Requests with Middleware and Endpoints （二）

- ■Middleware is typically added to WebApplication by calling **Use*** extension methods.
- ■The **routing middleware** is added automatically to the start of the **pipeline**, before any of the additional middleware added.

```
if (app.Environment.IsDevelopment())
{
    app.UseHttpLogging();
}
```

33

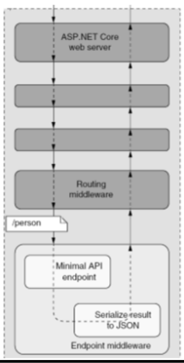
Handling Requests with Middleware and Endpoints （三）

- ■Use Map* methods to define endpoints.
- ■The routing and endpoint middleware work in tandem, using the set of endpoints defined for your application.

```
app.MapGet("/", () => "Hello World!");
app.MapGet("/author", () => new Author("Lee", "Lee"));
```

34

Summary



The ASP. NET Core web server (Kestrel) receives the HTTP request and passes it to the middleware.

The request path /person is routed to the minimal API endpoint, so the request passes through the middleware pipeline unmodified.

The minimal API endpoint handles the request by returning a Person instance.

The JSON response passes back through each middleware to the ASP. NET Core web server.

The minimal API infrastructure serializes the Person object to JSON in the endpoint middleware.

35

Basics of ASP.NET Core

Defining Middleware（一）

- ■What is middleware?
  - ■In ASP.NET Core, middleware is a C# class that can handle an HTTP request or response.
  - ■Middleware is the fundamental source of behavior in your application.

Defining Middleware（二）

- ■What can middleware do?
  - ■Handle an incoming HTTP request by generating a response;
  - ■Process an incoming HTTP request, modify it, and pass it on to another piece of middleware;
  - ■Process an outgoing HTTP response, modify it, and pass it on to another piece of middleware or to the web server.

1. Middleware Pipeline

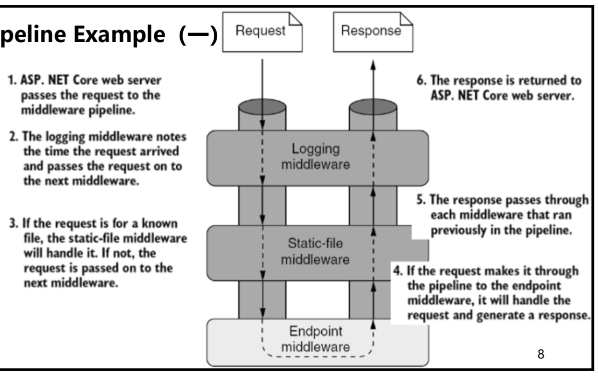
Examples of Middleware

- ■Logging middleware
  - ■note when a request arrived and then pass it on to another piece of middleware.
- ■Static-file middleware
  - ■spot an incoming request for an image, load the image from disk, and send it back to the user without passing it on.
- ■Endpoint middleware
  - ■generates all your HTML and JSON responses.

Defining Pipeline

- Middleware is chained together, with the output of one acting as the input to the next to form a pipeline.
- You can think of each piece of middleware as being like a section of pipe; when you connect all the sections, a request flows through one piece and into the next.

Pipeline Example（一）



1. ASP. NET Core web server passes the request to the middleware pipeline.
2. The logging middleware notes the time the request arrived and passes the request on to the next middleware.
3. If the request is for a known file, the static-file middleware will handle it. If not, the request is passed on to the next middleware.
4. If the request makes it through the pipeline to the endpoint middleware, it will handle the request and generate a response.
5. The response passes through each middleware that ran previously in the pipeline.
6. The response is returned to ASP. NET Core web server.