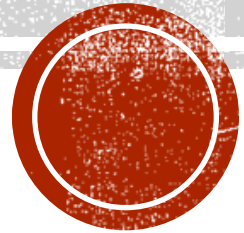
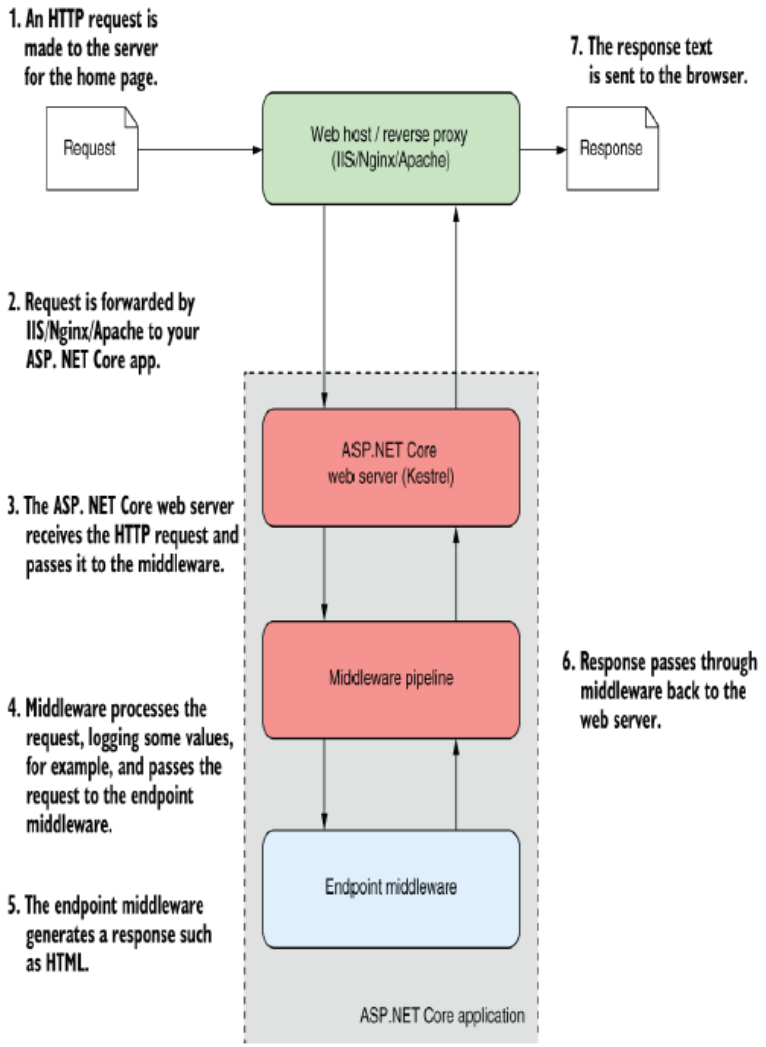
YOUR FIRST APPLICATION



Kamal Beydoun
Lebanese University – Faculty of Sciences I
Kamal.beydoun@ul.edu.lb

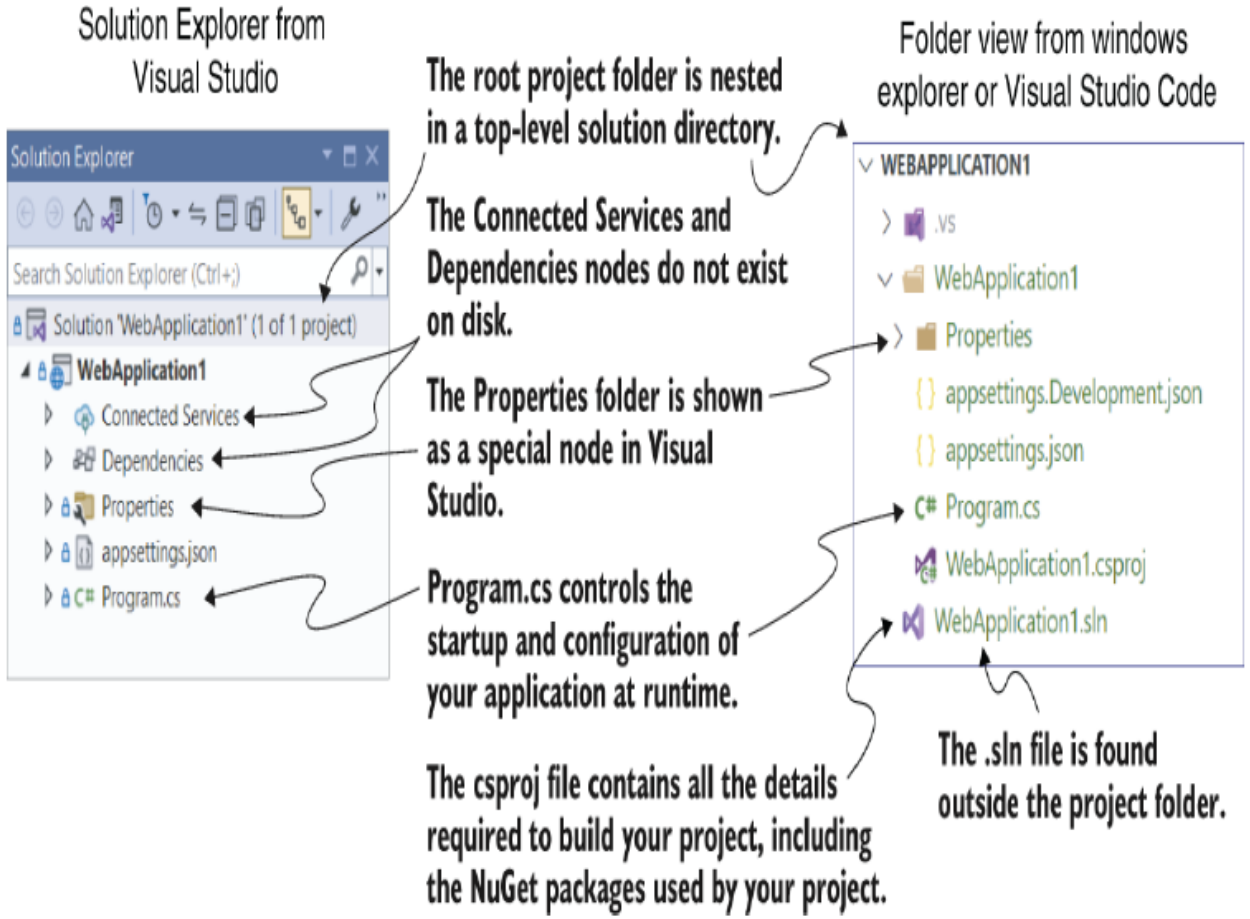
A BRIEF OVERVIEW OF AN ASP.NET CORE APPLICATION



CREATE NEW WEB APPLICATION

- Visaul Studio

UNDERSTANDING THE PROJECT LAYOUT



.CSPROJ PROJECT FILE: DECLARING YOUR DEPENDENCIES

- **Purpose of .csproj file:**
  - Project file for .NET applications.
- **Key Details in .csproj:**
  - Information necessary for .NET tooling to build the project.
- **Project Type Definition:**
  - Specifies the type of project (web app, console app, or library).
- **Targeted Platforms:**
  - Defines the platforms the project targets (e.g., .NET Core 3.1, .NET 7).
- **Dependency Management:**
  - Lists the NuGet packages the project depends on.

PROGRAM.CS FILE: DEFINING YOUR APPLICATION

- **ASP.NET Core Application Initialization:**
  - All ASP.NET Core applications **begin** as a .NET Console application.
- **Transition in .NET 6:**
  - In .NET 6, ASP.NET Core applications typically start as programs with top-level statements.
- **Top-Level Statements:**
  - Startup code is directly written in a file instead of inside a static void Main function.
  - You can write the body of this method directly in the file, and the compiler generates the Main method for you.

PROGRAM.CS FILE: DEFINING YOUR APPLICATION

```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);   ❶
WebApplication app = builder.Build();                                 ❷

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

app.Run();                                                            ❹
```

❶ Creates a WebApplicationBuilder using the CreateBuilder method

❷ Builds and returns an instance of WebApplication from the WebApplicationBuilder

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

❹ Runs the WebApplication to start listening for requests and generating responses

ADDING FUNCTIONALITY TO YOUR APPLICATION

```
using Microsoft.AspNetCore.HttpLogging;

WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

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

builder.Logging.AddFilter(                                       ❷
    "Microsoft.AspNetCore.HttpLogging", LogLevel.Information);   ❷

WebApplication app = builder.Build();

if (app.Environment.IsDevelopment())                             ❸
{
    app.UseHttpLogging();                                        ❹
}

app.MapGet("/", () => "Hello World!");
app.MapGet("/person", () => new Person("Andrew", "Lock"));       ❺

app.Run();

public record Person(string FirstName, string LastName);        ❻
```

❶ You can customize features by adding or customizing the services of the application.
❷ Ensures that logs added by the HTTP logging middleware are visible in the log
output
❸ You can add middleware conditionally, depending on the runtime environment.
❹ The HTTP logging middleware logs each request to your application in the log
output.
❺ Creates a new endpoint that returns the C# object serialized as JSON
❻ Creates a record type

ADDING FUNCTIONALITY TO YOUR APPLICATION

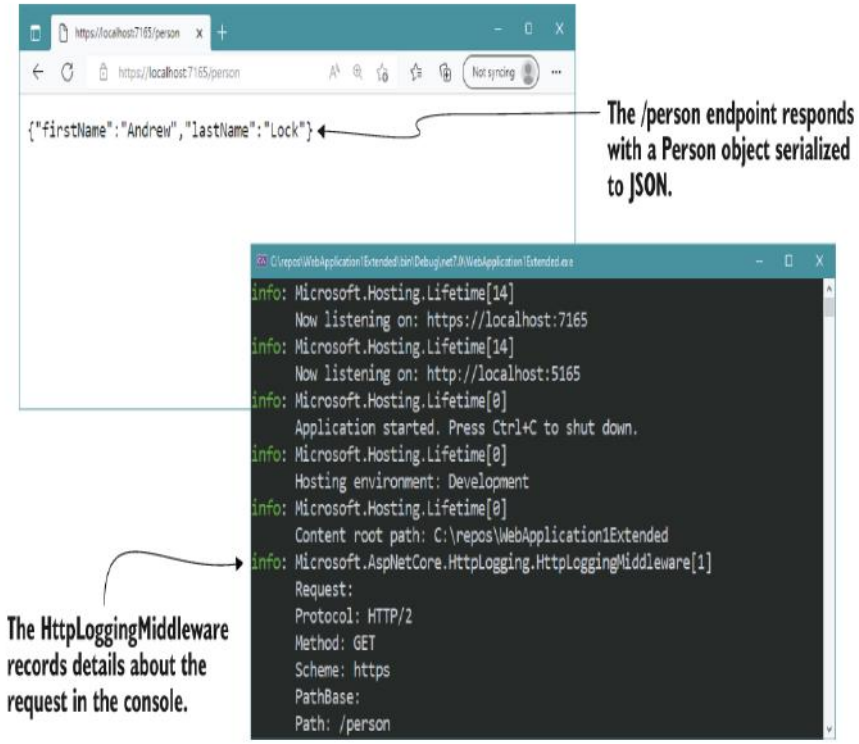


Figure 3.8 Calling the `/person` endpoint returns a JSON-serialized version of the `Person` record instance. Details about each request are logged to the console by the `HttpLoggingMiddleware`.

ADDING AND CONFIGURING SERVICES

- **ASP.NET Core Design Approach:**
  - Utilizes small modular components for **distinct** features.
  - Enables independent evolution of features with **loose coupling**.
- **Modular Components in Applications:**
  - Exposed as one or more services within the application.
  - These services are used by the application.

DEFINING HOW REQUESTS ARE HANDLED WITH MIDDLEWARE AND ENDPOINTS

WebApplication Instance Abilities:

- **Add Middleware to Pipeline:**
  - Middleware, **small components** executed sequentially on HTTP requests.
    - logging, user identification, static file serving, error handling.
  - Added using Use* extension methods.
  - Sequence of Use* calls in the builder determines execution order in the final pipeline.
- **Map Endpoints:**
  - Generate response for requests by endpoint mapping.
  - Defines application responses to specific requests.
- **Run the Application:**
  - Execute application by calling Run() method.

DEFINING HOW REQUESTS ARE HANDLED WITH MIDDLEWARE AND ENDPOINTS

- WebApplication automatically adds more middleware, including two of the most important and substantial pieces of middleware in the pipeline:
  - the **routing** middleware
  - the **endpoint** middleware.

```
app.MapGet("/", () => "Hello World!");
app.MapGet("/person", () => new Person("Andrew", "Lock"));
```