1
Introducing ASP.NET Core 9 Concepts

Before starting to develop solutions using the ASP.NET Core 9 platform, we must learn the fundamentals and how to prepare the environment, as well as become accustomed to the main concepts that will be used during the development process.

In this chapter, we'll learn about how to prepare our development environment, understand the differences between .NET and .NET Framework, and see what's new in ASP.NET Core 9.

We'll cover the following topics in this chapter:

- Why ASP.NET Core 9?
- Comparing .NET and .NET Framework
- Preparing our development environment
- What's new in ASP.NET Core 9?

My purpose is to introduce you to the main concepts of ASP.NET Core 9 and how to use this powerful platform to deliver web-based applications. I will explain the fundamentals of the platform, giving you context about the difference between .NET and .NET Framework and how to prepare your own Windows, Mac, or Linux environment, which you are going to work on until the end of this topic. Furthermore, we'll learn the most important improvements on this platform that have seen an increase in new features over the years. Let's start to learn the basics of ASP.NET Core 9.

Technical requirements

To take advantage of all the knowledge that will be shared, it is important that you have access to a computer, with administrative privileges, and also access to the internet.

Why ASP.NET Core 9?

ASP.NET core is a platform that has existed since 2016 and has been constantly improved, allowing the development of high-performance, modern, and cloud-ready solutions.

A few years ago, it was only possible to develop solutions on the .NET platform using the Windows operating system. However, with the great market demand and the rapid evolution of technologies, Microsoft began a one-way process of restructuring and redesigning the platform, adopting the open source model, which gave the developer community the opportunity to adopt a robust development model, independent of the operating system.

The last major version of ASP.NET to run exclusively on the Windows operating system was 4.x, and after the redesign, the platform was renamed *ASP.NET Core*, which is currently in version 9 STS(Standard Term Support).

Recently, ASP.NET Core 9 has become an extremely rich platform, enabling the delivery of solutions for different types of purposes and, what's more, with the full support and focus of the open source community.

Using ASP.NET Core 9 provides us with a rich tool with the following benefits:

- The ability to develop web UI solutions
- The ability to develop web APIs
- Interoperability of operating systems
- Cloud-readiness
- High performance
- Integrations with modern client-side frameworks
- The use of best practices and design standards

It is a complete platform that unifies everything necessary for the development of rich solutions using the best practices, technologies, and other aspects, which you will learn about throughout the chapters.

Performance improvements

ASP.NET Core 9 features several important performance improvements over ASP.NET Core 7, making it the best-performing version to date. Some important improvement points are as follows:

- **Faster execution and startup**: ASP.NET Core 9 is faster than the previous version. Every version have been receiving many contributions from the technical community. The et runtime, responsible for the application execution, has some improvements like loop optimizations, many other code generation and Garbage Collector improvements.
- **Minimal API Performance**: According to benchmarks, the new version of the Minimal API is 15% faster than the previous version and consumes 93% less memory.
- **Native AOT (Ahead-of-Time) compilation**: Expanded support for native AOT in ASP.NET Core 9 allows applications to be compiled to native code, reducing disk footprint, improving startup times, and decreasing memory consumption, which is ideal for cloud-native environments.
- **ML.NET** important improvements enabling integrations with machine-learning models with the version ML.NET 4.0, with additional tokenizer support, necessary for modern IA models.
- **.NET Aspire** The .Net Aspire was introduced on the previous version .Net 9 adding an improved cloud-ready stack for building observable, production ready, distributed applications in ASP.NET Core 9. Many concerns related to the cloud-native approach during the development phase was abstracted with .NET Aspire, combining a couple of Nuget Packages and projects templates.
- **.NET MAUI**: The .Net MAUI (Multi-platform Application UI) which provides an unified way to develop applications for platforms like Web and Mobile, including IOS and Android. .NET MAUI has quality improvements like test coverage, end to end scenario test, and bug fixes. Now, as part of the project templates of ASP.NET Core 9 there is a hybrid project including MAUI integrated with Blazor. With this project software engineers are able to delivery applications not only for web, but also for mobile and windows.
- **Entity Framework Core**: Entity Framework Core is one of the most powerful feature of .Net, providing a way to abstract the communication between applications and database, using an approach called ORM(Object Relational Model). In the new version some more features and improvements was added like a Azure Cosmos DB for NoSQL provider and capabilities to work with AOT.
- **ASP.NET Core**: The entire ASP.NET Core platform has many improvements on Blazor, SignalR, minimal APIs, authentication and authorization and better support to the OpenAPI.
- **Swagger:** One of the most famous libraries for generating API documentation, Swagger, from the **Nuget** package **Swashbuckle.AspNetCore** will no longer be part of the default API template of ASP.NET Core 9. This is due to the fact of reducing the dependence on this library in .NET projects and improving support for Open API, a language-agnostic and platform-neutral of web based APIs.

For more details, check out the following link: `https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-9/`.

However, before starting to prepare our environment and use the ASP.NET Core 9, in the next section, let's understand how the platform has been evolving with collaboration between Microsoft and the open source community, by comparing the .NET platform and the .NET Framework.

Comparing .NET and .NET Framework

Both .NET and .NET Framework (usually called *Full Framework*) have similarities – that is, they are platforms that allow us to deliver great solutions.

In general, the platform is a framework with a set of features that allows us to develop different types of applications. In February 2002, .NET Framework brought a new development model using a centralized platform, allowing us to develop applications for Windows and the web through technologies such as Windows Forms, ASP.NET Web Forms, ASP.NET MVC, as well as console applications, among some other extensions.

Since the first versions, it was already possible to develop applications in different languages such as C#, Visual Basic, and any other languages that implemented the .NET Framework specifications. However, it depended on the Windows operating system and its system APIs.

The evolution and redesign of the .NET Core platform has brought many benefits to the Microsoft ecosystem, while maintaining the main idea of being a unified platform for the development of robust solutions. It is possible to develop in languages such as C#, F#, or even Visual Basic.

The structuring and redesign meant that the .NET core platform, now called just .NET, was developed in a modularized manner and independent of the operating system, with the support of the open source community.

Today, the entire .NET platform ecosystem is maintained by the .NET Foundation (`https://dotnetfoundation.org/`), a non-profit, independent organization that supports the entire open source platform ecosystem. With that, new possibilities were created for the .NET community, including the reduction in the lead time for delivery of new versions of the framework with new features and bug fixes.

The new releases of the .NET platform are made available annually, every November, in STS(**Standard Term Support)** versions, launched in even years and receiving support for 18 months, or LTS(**Long Term Support)**, launched in odd years and receiving support for three years. There are also monthly patch updates, which speed up the agile correction of problems and vulnerabilities, maintaining compatibility between each patch and eliminating the greater risk of updates.

Understanding the platform update process brings great benefits to development times, as updates can cause non-conformities in applications, generating several problems.

Microsoft provides a complete roadmap of features that are added to the platform, as well as the implementation of improvements and, most importantly, fixes of bugs and vulnerabilities. Keep the roadmap link as a favorite in your browser: `https://github.com/MoienTajik/AspNetCore-Developer-Roadmap`.

Now that we understand some of the platform's basics, let's prepare our development environment on different operating systems.

Preparing our development environment

The .NET platform has a set of tools available to offer the best experience to developers, regardless of the operating system used.

ASP.NET Core 9 applications can be developed and run on Windows, Linux, and macOS operating systems.

Code snippets will be presented throughout the discussion in order to demonstrate the concepts of ASP. NET Core 9 through practical examples. All supporting material for the topic can be found in the GitHub repository, the link to which can be found in the *Technical requirements* section.

In this section, we will configure our environment on the three operating systems and create our first ASP.NET Core 9 project, but first, let's see what things we will require to get started.

The development tool

We can develop ASP.NET Core 9 applications using any text editor and then compile the developed code using the SDK(**Software Development Kit)**, which will be discussed in the next section.

Microsoft offers two code-editing tools, Visual Studio and Visual Studio Code.

Visual Studio Code is a rich, extensible, and lightweight code editor that makes it possible to develop any type of application. It is a free tool, has several extensions, is widely used by the community, and can run on any operating system.

Conversely, Visual Studio is a more robust version of the IDE, having several visual functionalities that support development, in addition to several tools such as application profiling and a rich debug tool. Visual Studio only runs on the Windows operating system and a license must be purchased. However, Microsoft offers a version of Visual Studio called Community that is free and, despite some limitations, offers an excellent development experience. For the remainder of this discussion, we use Visual Studio as the main code editor, because it is extensible and, most importantly, free.

> **Visual Studio for Mac**
>
> Visual Studio for Mac will not be continued by Microsoft, and its support will end on August 31, 2024.

SDK and runtime

When proceeding with the installation of the .NET platform on your machine, some questions may arise regarding the SDK and the runtime.

The SDK allows us to develop and run ASP.NET Core 9 applications, while the runtime only has the necessary dependencies to run the applications.

In general, we always choose to use the SDK on development machines; however, when hosting environments, only the runtime is required. We'll discuss hosting applications in more detail in *Chapter 10*.

CLI (Command-Line Interface)

Along with .NET and/or Visual Studio Code, a **CLI** will also be installed that will be used extensively.

The CLI is nothing more than software executed through the command line, allowing to you to execute tasks for different purposes, such as the following command:

```
dotnet new webapp --name hello-world
```

The CLI command here is called `dotnet`. This command has some parameters, responsible for determining the type of task that will be executed.

In a nutshell, the previous command creates a new project (`new`) of type `webapp` with the name (`--name`) `hello-world`.

CLI tools bring great flexibility, avoid UI dependencies, are extensible, and allow us to use automation strategies through scripts.

Throughout the discussion, we will use some CLI commands to support solution development and learning.

In the next sections, we will look at installing the ASP.NET Core 9 SDKs for all three operating systems.

Windows installation

Windows offers some options for installing the SDK for the .NET platform:

- Installation together with Visual Studio.
- Using the Windows package manager, **Winget**. The SDK can be installed by running the following command:

  ```
  winget install -e --id Microsoft.DotNet.SDK.9
  ```

- Via PowerShell.

However, we will proceed with the installation through Visual Studio, but if you prefer to use other installation options, check out this link: `https://learn.microsoft.com/en-us/dotnet/core/install/windows`.

Installation through Visual Studio is very simple and involves just a few steps:

1. Go to `https://visualstudio.microsoft.com/`.
2. Download Visual Studio for Windows and save it. After downloading it, run the following file: `VisualStudioSetup.exe`.
3. After installation, locate the `VisualStudioSetup.exe` file, run it, and then click on **Continue**.

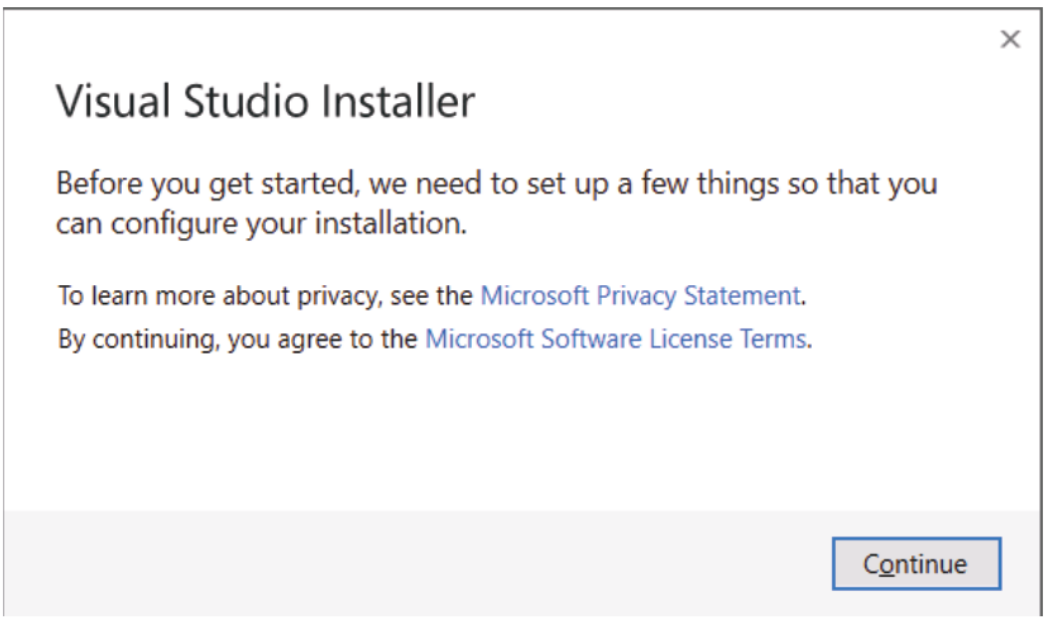

Figure 1.1 – The Visual Studio Installer message

4. On the **Workloads** tab, select the **ASP.NET and web development** option:

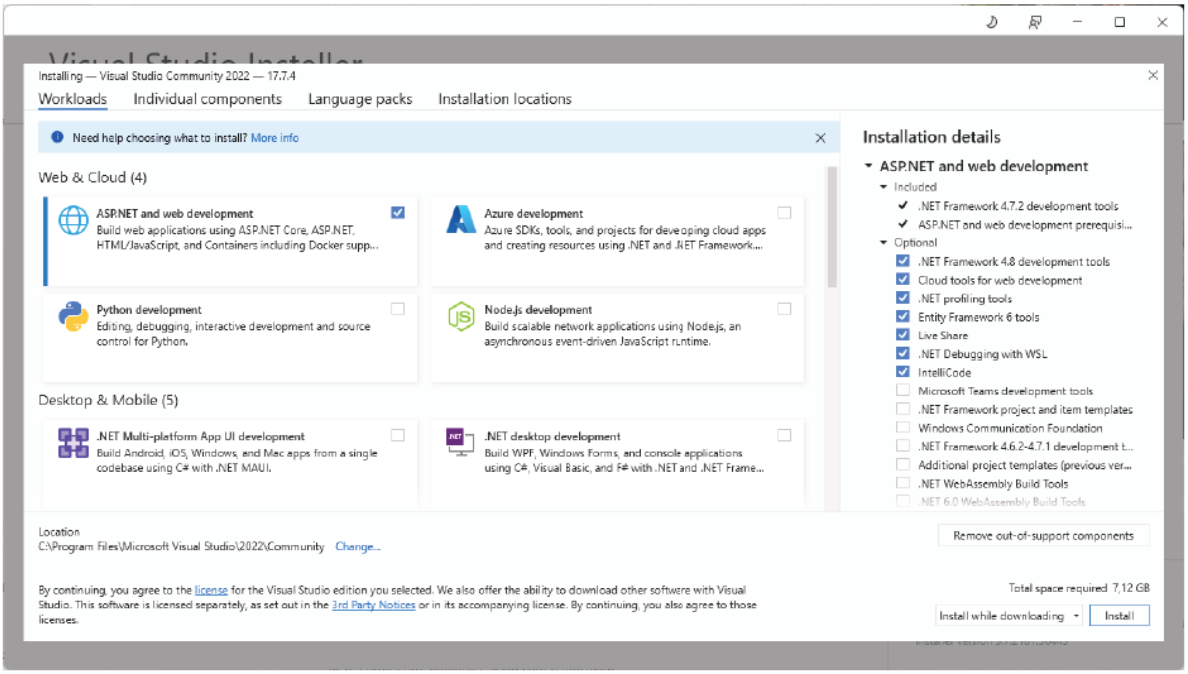

Figure 1.2 – Visual Studio installation options

5. Then, click on the install button and proceed with the installation.

macOS installation

macOS offers an executable that allows you to follow a simple installation process:

1. Download the .NET platform from `https://dotnet.microsoft.com/download/dotnet`.
2. Select the version best suited to your processor (ARM64 or x64)
3. After downloading, run the installer and complete the steps to complete the installation.

> **Supported versions**
>
> Microsoft does not support .NET in versions before 6. From version 6 (LTS) onwards, current Apple processors are supported.

For more details on installing on macOS, refer to this link: `https://learn.microsoft.com/en-us/dotnet/core/install/macos`.

Linux installation

.NET supports several versions of Linux. For the next steps, we will focus on the Ubuntu 22.04 version, and we will run the process based on the script provided by Microsoft:

1. Access the command terminal, and create a folder called `dotnet-install` in your home directory.
2. Download the `sh` script, provided by Microsoft with the following command:

```
wget https://dot.net/v1/dotnet-install.sh -O dotnet-install.sh
```

3. You will need to add permission to the script before running it with the command:

```
chmod +x ./dotnet-install.sh
```

4. Now, run the installation command:

```
./dotnet-install.sh --version latest
```

5. This command will install the latest SDK version.

> **Dependencies**
> The .NET platform depends on some libraries that are specific to each version of Linux. For more details, see the following link: `https://learn.microsoft.com/en-us/dotnet/core/install/linux-ubuntu#dependencies`.

Next, let's install Visual Studio Code.

Visual Studio Code

Visual Studio Code is a great editor, and its installation is very simple.

Just go to `https://code.visualstudio.com/download`, download the version specific to your operating system, run the installer, and complete the steps.

Code command

When installed, Visual Studio Code also installs its CLI. On Windows and Linux systems, the CLI is usually automatically added to the system *PATH environment variable*. On macOS, additional configuration will be required. To do this, follow these steps:

1. Press the *CMD* + *Shift* + *P* keys.
2. Type in the following command:

```
Install 'code' command in PATH
```

3. Press *Enter*, and the CLI will be added to the *Path environment variable*.

The time has come to validate the functioning of the environment, and to do so, we will create our first project.

Testing the development environment

After installing the editor, code, and SDK, it's time to create our first application and ensure that the environment is functional for the rest.

In this step, we will use the Terminal or Bash, the dotnet CLI, and also Visual Studio Code as the IDE. We will create a simple web application using the command line.

To do so, open a command terminal or bash and follow these instructions:

1. In your preferred directory, create a folder called `Projects`:

```
mkdir Projects
```

2. Access the created folder with the following command:

```
cd projects
```

3. Now, run the following command:

```
dotnet new
```

The `dotnet new` command needs some instructions for us to proceed with creating a project. When running it, the following instructions are displayed:

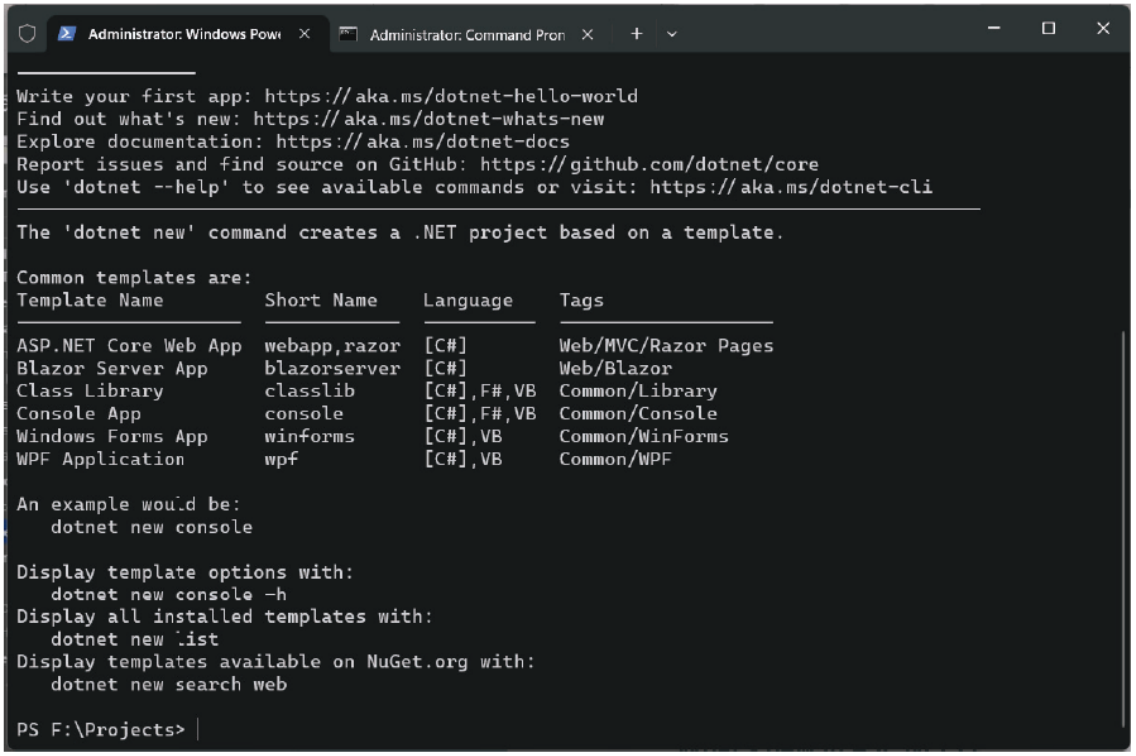

Figure 1.3 – Executing the dotnet new command

4. As in Visual Studio, the .NET CLI has some templates that are used when creating projects. You will be able to see the templates installed on your machine by typing the highlighted command:

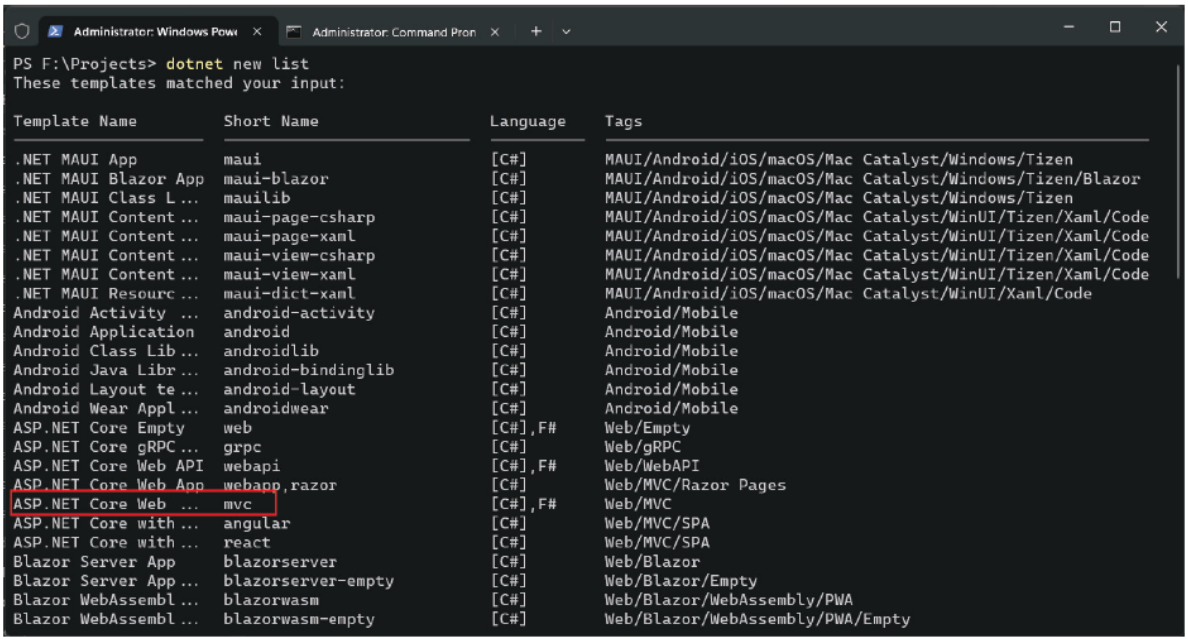

Figure 1.4 – The available dotnet templates

5. We will continue with the creation of a web application in the MVC template. To do this, run the following command:

```
dotnet new mvc --name my-first-app
```

The previous command is basically a template for the type of project that will be created. In this case, we will use the MVC model, which is a project that uses the **Model-View-Controller** (**MVC**) architectural pattern. Don't worry about these details at this point. We will learn more about the .NET CLI tool, project models, and the MVC model throughout.

6. Now, access the directory of the created application with the following command:

```
cd my-first-app
```

7. Next, let's open Visual Studio Code to make some changes to the application's source code with the following command:

```
code .
```

8. The previous command will open a new instance of Visual Studio Code in the previously created application directory.

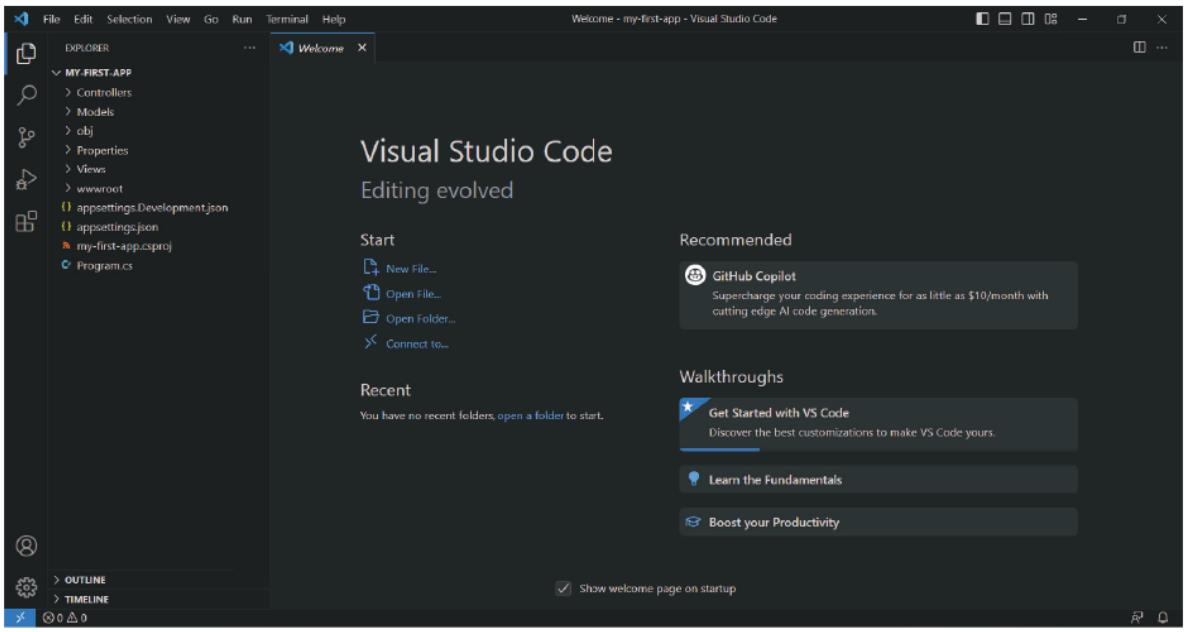

Figure 1.5 – My first app, opened on Visual Studio Code

9. In Visual Studio Code, locate the `index.cshtml` file in **Views** | the `Home` folder.

10. Replace line 6 of the `index.cshtml` file with the following code, and then select **File |ZSave**, or just press *Ctrl + S*:

```
<h1 class="display-4">
  Welcome to the ASP.NET Core 9!
</h1>
```

11. Return to the terminal or bash and run the following command:

```
dotnet run
```

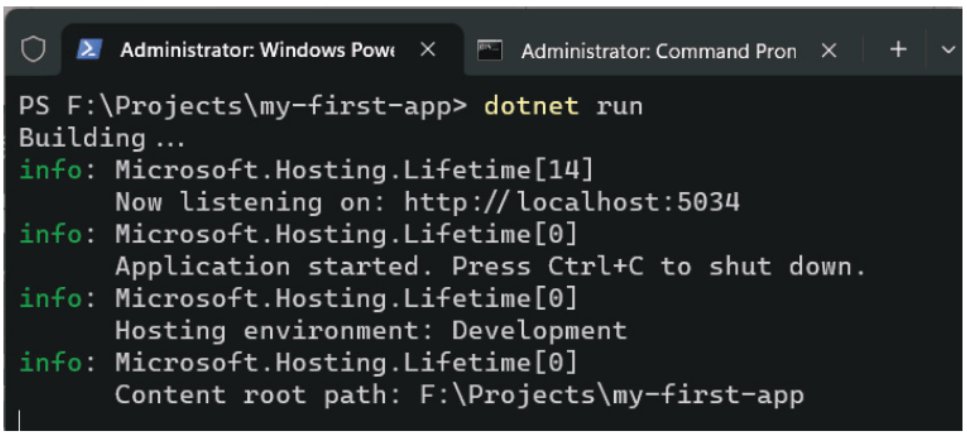

Figure 1.6 – Running the dotnet run command

If you can see a message like the one shown in *Figure 1.6*, then your environment is working correctly.

12. Now, access the browser and type the address presented, as in the `http://localhost:5034` example.

Note the address on your terminal, as the application execution port number may be different.

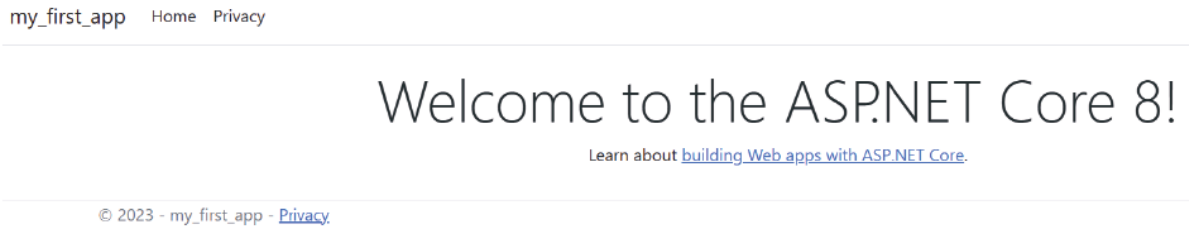

Figure 1.7 – The application running

If all the previous steps were executed successfully, it means that your code editor was correctly configured, as well as the .NET SDK.

If there is any problem, review the installation steps according to your operating system.

With our environment configured, we are ready to continue learning the platform and implementing the examples set.

What is new in ASP.NET Core 9?

As it is open source and independent of any operating system, the .NET platform has received several improvements in recent years, further improving the experience of developers and bringing several new features constantly requested by the community and, in addition, several improvements in performance and bug fixes.

Some improvements are as follows:

- **Native AOT**: This option was introduced in version 7.0 of the framework to create a self-contained application, in a single file, adding support for the x64 processors and ARM64 architectures of macOS and greatly reducing the size of the file, reaching up to 50% reduction in Linux environments.
- **Improvements in serialization libraries**: Applications constantly interact with data in JSON and the **System.Text.Json** API, native to .NET, has been constantly revised and improved, avoiding dependence on third-party libraries and considerably improving its performance and support.
- **Performance**: There has been a performance improvement of approximately 15%. You can check this link for more performance improvement details: `https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0?view=aspnetcore-8.0`.

These and many other features can be consulted directly from the platform roadmap at the following link: `https://github.com/dotnet/aspnetcore/issues/44984`.

The .NET roadmap is very well maintained and updated by the community, together with Microsoft and the .NET Foundation. It is exactly this great support from the community, as well as the other companies and organizations involved, that has turned .NET into a powerful development platform for different purposes. By following the other need of the market, it has considerably improved the options for the development of web-based solutions, using different approaches to meet different needs.

Summary

In this chapter, we learned about the .NET platform and ASP.NET Core 9, available in the main operating systems. We learned the differences between the .NET and .NET Framework, in addition to learning about the process of updating the framework versions and the difference between the STS and LTS versions. We also configured our development environment on different operating systems and developed an ASP.NET MVC application, validating the entire working environment .