

# Beginning gRPC with ASP.NET Core 6

Build Applications using ASP.NET  
Core Razor Pages, Angular,  
and Best Practices in .NET 6

---

Anthony Giretti

apress®

# **Beginning gRPC with ASP.NET Core 6**

**Build Applications using ASP.NET  
Core Razor Pages, Angular, and Best  
Practices in .NET 6**

**Anthony Giretti**

Apress®

# ***Beginning gRPC with ASP.NET Core 6: Build Applications using ASP.NET Core Razor Pages, Angular, and Best Practices in .NET 6***

Anthony Giretti  
La Salle, QC, Canada

ISBN-13 (pbk): 978-1-4842-8007-2  
<https://doi.org/10.1007/978-1-4842-8008-9>

ISBN-13 (electronic): 978-1-4842-8008-9

Copyright © 2022 by Anthony Giretti

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Joan Murray  
Development Editor: Laura Berendson  
Coordinating Editor: Jill Balzano  
Copyeditor: Bill McManus

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, email [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <https://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page at <https://github.com/Apress/beg-grpc-w-asp.net-core-6>.

Printed on acid-free paper

# Table of Contents

**About the Author ..... ix**

**About the Technical Reviewer ..... xi**

**Acknowledgments ..... xiii**

**Introduction .....xv**

**Part I: Getting Started with .NET 6..... 1**

**Chapter 1: Welcome to Modern .NET ..... 3**

    A Brief History of .NET..... 3

        .NET Framework ..... 4

        .NET Core..... 5

        .NET Standard..... 6

    Modern .NET: A Unified Platform ..... 7

        Mono and CoreCLR ..... 8

    .NET Schedule and What It Means ..... 9

    How to Explore .NET 6..... 9

        .NET 5 and 6 Improvements ..... 10

        Get Started with .NET 6 ..... 11

    Recap of C# 9 and Introduction to C# 10 ..... 17

        Recap of C# 9 ..... 17

        Introduction to C# 10 ..... 29

    Summary..... 31

**Chapter 2: Introducing ASP.NET Core 6..... 33**

    ASP.NET Core Fundamentals..... 34

    ASP.NET Core Web API..... 42

    ASP.NET Core MVC ..... 53

TABLE OF CONTENTS

ASP.NET Core Razor Pages..... 59

ASP.NET Core Blazor ..... 64

ASP.NET Core SignalR ..... 72

ASP.NET Core gRPC ..... 76

ASP.NET Core Minimal APIs ..... 77

Summary..... 81

**Part II: gRPC Fundamentals ..... 83**

**Chapter 3: Understanding the gRPC Specification ..... 85**

Introduction to Remote Procedure Calls ..... 85

gRPC Concepts..... 87

    Protocol Buffers..... 87

    gRPC Channel ..... 88

    Types of gRPC Services ..... 91

    Trailers..... 93

    gRPC Status..... 94

    Deadline and Cancellation ..... 95

    gRPC Requests and Responses over HTTP/2 ..... 95

Introduction to the HTTP/2 Protocol ..... 97

    Multiplexing..... 98

    Compression and Binary Data Transport ..... 99

    Flow Control ..... 99

    Server Push ..... 99

Benefits, Drawbacks, and Use Cases..... 99

    Benefits ..... 100

    Drawbacks..... 100

    Use Cases ..... 101

Summary..... 102

**Chapter 4: Protobufs..... 103**

    About Protocol Buffers ..... 103

    Individual Declarations..... 104

Services Declaration .....	108
Messages Declaration.....	111
Scalar Type Values.....	113
Collections .....	113
Enumerations .....	119
Nested Types .....	122
Import Types .....	122
Any, Value, Struct, Wrappers, Dates, and Times (Well-Known Types).....	123
Bytes.....	137
One of .....	141
Empty Messages .....	147
Comments .....	149
Summary.....	151
<b>Part III: gRPC and ASP.NET Core .....</b>	<b>153</b>
<b>Chapter 5: Creating an ASP.NET Core gRPC Application .....</b>	<b>155</b>
Create an ASP.NET Core gRPC Application .....	155
Create and Compile Protobuf Files.....	160
Write, Configure, and Expose gRPC Services .....	165
Test Using gRPCurl and gRPCui Tools .....	180
gRPCurl.....	180
gRPCui .....	189
TLS Certificates .....	195
Manage Errors, Handle Responses, and Perform Logging.....	196
Perform Message Validation .....	214
Support of ASP.NET Core gRPC on Microsoft Azure.....	219
Summary.....	221
<b>Chapter 6: API Versioning.....</b>	<b>223</b>
Version gRPC Services.....	223
Expose the Versions of Your Protobuf with ASP.NET Core Minimal APIs.....	232
Summary.....	237

TABLE OF CONTENTS

**Chapter 7: Create a gRPC Client ..... 239**

    Create a Console Application ..... 240

    Compile Protobuf Files and Generate gRPC Clients ..... 244

    Consume gRPC Services with .NET 6..... 252

    Optimize Performance ..... 268

        Take Advantage of Compression ..... 268

        Define a Limit to Message Size ..... 272

        Keep HTTP/2 Connections Open ..... 273

        Increase HTTP/2 Maximum Connections..... 277

    Get Message Validation Errors from the Server ..... 278

    Summary..... 281

**Chapter 8: From WCF to gRPC ..... 283**

    Differences and Similarities Between WCF and gRPC ..... 283

    What and What Not to Migrate from WCF to gRPC..... 286

    Summary..... 298

**Chapter 9: Import and Display Data with ASP.NET Core Razor Pages, Hosted Services, and gRPC ..... 299**

    Scenario Explanation ..... 300

    Create and Layer the ASP.NET Core gRPC Application ..... 301

    Set Up a SQL Server Database and Use Entity Framework Core to Access Data..... 310

        Set Up a SQL Server Database ..... 310

        Using Entity Framework Core to Access Data ..... 311

    Write the Business Logic and Expose the Country gRPC Microservice ..... 330

        Write the Business Logic into the CountryService.BLL Layer ..... 330

        Write the Country gRPC Service ..... 332

    Create and Layer the ASP.NET Core Razor Application..... 341

        Create the Application Skeleton ..... 342

        Define Contracts and Domain Objects..... 343

        Implement the Data Access Layer with the gRPC Client ..... 348

Implement the Business Logic Layer.....	353
Configure the ASP.NET Core Razor Pages Application .....	358
Upload a Data File with a Form, Display and Manage Data on Razor Pages .....	367
Summary.....	385
<b>Part IV: gRPC-web and ASP.NET Core .....</b>	<b>387</b>
<b>Chapter 10: The gRPC-web Specification .....</b>	<b>389</b>
History and Specification of gRPC-web .....	389
History of gRPC-web .....	389
The gRPC-web Specification .....	391
The gRPC-web JavaScript Libraries.....	392
gRPC-web vs. REST APIs.....	393
Summary.....	394
<b>Chapter 11: Create a gRPC-web service from a gRPC-service with             ASP.NET Core .....</b>	<b>395</b>
Working with gRPC-web and the .NET Ecosystem.....	396
gRPC-web and ASP.NET Core 6.....	396
gRPC-web and All .NET Clients.....	399
gRPC-web and ASP.NET Core 3+ Clients .....	402
Reworking the CountryService gRPC service for Browser Apps.....	404
Support of ASP.NET Core gRPC-web on Microsoft Azure .....	416
Summary.....	417
<b>Chapter 12: Import and Display Data with Angular 12 and gRPC-web .....</b>	<b>419</b>
Introduction to SPAs.....	419
Generate TypeScript Stubs with Protoc.....	421
Download the Correct Version of Protoc and Protobuf Well-Known Types .....	422
Download the ts-protoc-gen Plug-in .....	426
Download Improbable's gRPC-web Library and Google Protobufs Library.....	426
Executing the Protoc Command .....	426

TABLE OF CONTENTS

Write Data Access with Improbable’s gRPC-web Client..... 430

Upload a Data File and Display Data with TypeScript, a Web Worker, and gRPC-web..... 440

Manage Data with TypeScript and gRPC-web..... 450

Summary..... 456

**Part V: Security..... 457**

**Chapter 13: Secure Your Application with OpenId Connect..... 459**

Introduction to OpenId Connect ..... 459

Configure ASP.NET Core ..... 462

Use gRPCurl and gRPCui with a JWT ..... 469

    gRPCurl..... 469

    gRPCui ..... 471

Use a C# Client with a JWT ..... 473

Use a gRPC-web Client with a JWT ..... 476

Get User Identity Server Side..... 478

Summary..... 478

**Index..... 481**

# About the Author

**Anthony Giretti** is a senior lead software developer at OneOcean in Montreal, Canada. He is a technical leader and four-time Microsoft MVP award recipient. Anthony specializes in web technologies (17 years' experience) and .NET. His expertise in technology and IT, and a heartfelt desire to share his knowledge, motivates him to dive into and embrace any web project, complex or otherwise, in order to help developers achieve their project goals. He invites challenges such as performance constraints, high availability, and optimization with open arms. He is a certified MCSD who is passionate about his craft and always game for learning new technologies.

# About the Technical Reviewer



**Fiodar Sazanavets** is an experienced full-stack lead software engineer who mainly works with the Microsoft software development stack. The main areas of his expertise include ASP.NET (Framework and Core), SQL Server, Azure, Docker, Internet of Things (IoT), microservices architecture, and various front-end technologies.

Fiodar has built his software engineering experience while working in a variety of industries, including water engineering, financial, retail, railway, and defense. He has played a leading role in various projects and, as well as building software, his duties have included performing architectural and design tasks. He has also performed a variety of technical duties on clients' sites, such as in-house software development and deployment of both software and IoT hardware.

Fiodar is passionate about teaching other people programming skills. He has published a number of programming courses on various online platforms. Fiodar regularly writes about software development on his personal website, <https://scientificprogrammer.net>. He has also published a number of articles on other websites.

# Acknowledgments

The completion of this book could not have been possible without the participation and assistance of many people and I would like to express my special thanks to them.

First, thanks to Camille Viot, my boss, for accommodating me so that I could overcome this immense challenge.

Next, I would like to thank my friend Dave Brock (Madison, Wisconsin) for both his moral but technical support; he was a great help when I felt overwhelmed by the magnitude of the task. I also thank him for reviewing my chapters one by one—many thanks for his contribution! Thanks also to Damien Vande Kerckhove for his technical support, which allowed me to adjust the shot when I was not going in the right direction. He was also an essential asset for ensuring this book was able to see the light of day.

I also thank all my family for their unwavering support. Finally, I would like to thank a special member of my family that I unfortunately lost recently; he was there every night next to me when I was writing my lines. Thank you, Ulysse, you helped me so much and kept me company.



# Introduction

Take a new technological turn with gRPC and ASP.NET Core while discovering .NET 6, the latest release of the Microsoft .NET platform, and C# 10.

gRPC has become more and more famous because of its performance compared to JSON/XML APIs. In this book, you'll discover how to develop ASP.NET Core APIs with the gRPC specification, and gRPC will no longer be mysterious to you.

After you discover how gRPC works, you'll learn how to use it to build high-performance web applications with the best development standards. You'll use gRPC with various ASP.NET Core 6 project types such as Razor Pages and minimal APIs. You'll also discover gRPC-web and the great mix it does with Angular 12.

For Windows Communication Foundation (WCF) developers, you will learn how to migrate from WCF to gRPC by comparing the similarities and differences between the two frameworks.

We'll also explore using gRPC and gRPC-web with OpenId Connect authentication and authorization to secure your applications.

Let's go!

# **PART I**

## **Getting Started with .NET 6**

# CHAPTER 1

# Welcome to Modern .NET

.NET is 20 years old, having been introduced in 2002 with the release of the .NET Framework, .NET 1. Since then, it has evolved with the needs of the computing industry to become even faster, lightweight, and cross-platform. As I write this book, we are at a crossroads, if you will, of the original .NET Framework and the newer .NET Core framework coming together under one new .NET. Microsoft has recently released .NET 5 and .NET 6 in November 2021, and with it, you can build powerful web applications with ASP.NET Core 6.

For those of you who are already .NET developers, feel free to skip this chapter. For the rest of you, this chapter is designed to give you just enough history and background to provide some foundation for your learning moving forward. We'll cover the following topics:

- A brief history of .NET
- Modern .NET, a unified platform
- .NET schedule and what it means
- How to explore .NET 6
- Recap of C# 9 and introduction to C# 10

## A Brief History of .NET

A .NET application is developed for and runs in one or more implementations of .NET. Implementations include the *.NET Framework*, *.NET Core*, *Mono*, .NET 5 and now *.NET 6*. There is an API specification common to several implementations of .NET, called *.NET Standard*. This section introduces these concepts.

# .NET Framework

Since Microsoft’s release of .NET 1, there have been nine releases of the .NET Framework, with seven of them released with a new version of Visual Studio. Two of these releases, .NET Framework 2.0 and .NET Framework 4.0, have upgraded the *Common Language Runtime (CLR)*, which runs .NET applications. When the CLR version is the same, new versions of the .NET Framework replace older versions. .NET Framework 4.8 is the latest version of the .NET Framework. Table 1-1 shows .NET Framework releases from .NET 1 to .NET 4.8.

**Table 1-1.** All .NET Framework Versions Released

Version	Release Date	Visual Studio Version
1.0 (major version)	2/13/2002	VS.NET
1.1 (minor version)	4/24/2003	VS.NET 2003
2.0 (major version)	11/7/2005	VS 2005
3.0 (major version)	11/6/2006	VS 2005
3.5 (major version)	11/19/2007	VS 2008
4.0 (major version)	4/12/2010	VS 2010
4.5 (major version)	8/15/2012	VS 2012
4.5.1 (minor version)	10/17/2013	VS 2013
4.5.2 (minor version)	5/5/2014	VS 2015
4.6 (major version)	7/20/2015	VS 2015
4.6.1 (minor version)	11/30/2015	VS 2015
4.6.2 (minor version)	8/2/2016	VS 2017
4.7 (major version)	4/5/2017	VS 2017
4.7.1 (minor version)	10/17/2017	VS 2017
4.7.2 (minor version)	4/30/2018	VS 2017
4.8 (major version)	4/18/2019	VS 2019

The .NET Framework was designed to develop Windows-only applications, as Windows is heavily reliant on the .NET Framework. Its successor, .NET Core, changed that by becoming open source software and providing cross-platform support.

## .NET Core

In June 2016, Microsoft announced the .NET Core project, an open source, cross-platform successor with compatibility for Windows, macOS, and Linux. Since then, Microsoft has released two significant versions, .NET Core 2.0 and .NET Core 3.0, both of which have minor releases associated with them. .NET Core 3.1 is the latest version of .NET Core and will be supported until December 2022. Table 1-2 shows the .NET Core releases since 2016.

**Table 1-2.** *All .NET Core Versions Released*

Version	Release Date	Visual Studio Version
.NET Core 1.0 (major version)	6/27/2016	VS 2015
.NET Core 1.1 (minor version)	11/16/2016	VS 2017
.NET Core 2 (major version)	8/14/2017	VS 2017
.NET Core 2.1 (minor version)	5/30/2018	VS 2017
.NET Core 2.2 (minor version)	12/4/2018	VS 2019
.NET Core 3.0 (major version)	9/23/2019	VS 2019
.NET Core 3.1 (minor version)	12/3/2019	VS 2019

In addition to .NET Core and .NET Framework, Microsoft also maintains the *Mono* project, an open source implementation of Microsoft's .NET Framework. Launched in 2004 to allow developers to create cross-platform applications easily, it's based on the *European Computer Manufacturers Association (ECMA)* standards for C# and the CLR.

**Note** ECMA is a European nonprofit organization responsible for defining IT standards, both for hardware and software (programming languages), ECMAScript being the most famous standard developed by this organization. ECMA is also known for having developed the *Near Field Communication (NFC)* standard.

When it comes to API surface area, .NET Core 3 is not as robust as .NET Framework 4.8, a mature platform with a 15-year head start. However, Microsoft has added about 50,000 .NET APIs to the .NET Core platform to date. To continue closing this gap, Microsoft has built on the efforts made with .NET Core and taken the best of Mono to create a unique platform that you can use for all your .NET programs: .NET 5 and so on with .NET 6.

Microsoft has named this new version simply *.NET 5* (and then .NET 6) so as not to confuse developers, because it's *not* the successor to .NET Framework 4.8.

## .NET Standard

In 2011, Microsoft released the *Portable Class Libraries (PCL)*, which are binaries that are compatible with many frameworks. PCLs were a significant improvement because they were supported by several runtimes such as Mono, *Universal Windows Platform (UWP)*, and .NET. In the meantime, it was hard to find information on what APIs were available or not. To help with this confusion, .NET Standard was born.

.NET Standard is a bunch of APIs implemented by the *Base Class Library (BCL)*. It's a specification of .NET APIs that proposes a unified set of contracts that you can compile in your compatible projects. These contracts are implemented in several .NET implementations. Various .NET implementations target specific versions of .NET Standard. Table 1-3 shows the minimum implementation versions that support each .NET Standard version.

**Table 1-3.** All .Net Standard Versions Supported by .NET Implementations

.NET Standard Versions	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	3.1
.NET Framework	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1	N/A
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
UWP	10.0	10.0	10.0	10.0	10.0	10.0.x	10.0.x	10.0.x	N/A
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	N/A

.NET 6 implements .NET Standard 2.1 (and earlier), which is not deprecated, but .NET 6 (unified across platforms) is the new Microsoft implementation of .NET to share code between .NET projects.

## Modern .NET: A Unified Platform

Released in November 2020, .NET 5 is the next major evolution of .NET after .NET Core. The later has been followed by .NET 6 released in November 2022. You can now create various applications with the same runtime, allowing uniformity in the execution behaviors of your .NET applications, all with a homogeneous development experience (single code base). Therefore, code in your applications and your project files will look similar regardless of the type of your project. To make all this possible, .NET 5 & NET 6 combines the best of .NET Core and Mono. Figure 1-1 shows the unified ecosystem of .NET 5. In November 2021, .NET 6 has came and offers everything that has been brought by .NET 5 plus huge features like *Multi-platform App UI (MAUI)* and *ahead-of-time (AOT)* compilation.

### .NET – A unified platform



**Figure 1-1.** .NET 5/6 unified ecosystem (source: Microsoft)

---

**Note** .NET 5 was released after this diagram was released. Since then, Microsoft pushed the launch of Xamarin in the .NET unified platform to .NET 6.

---

## Mono and CoreCLR

We'll discuss two different development experiences with .NET: .NET with Mono and .NET with CoreCLR.

### Differences and Commonalities

Mono is the cross-platform implementation of .NET. It started as an open source alternative to the .NET Framework and made the transition to targeting mobile devices like iOS and Android much easier. Mono is the runtime used to run Xamarin. Mono allows developers to run .NET applications [cross-platform](#) (even older game consoles such as PlayStation 3 and Xbox 360) and provides powerful development tools for Linux. *Core Common Language Runtime (CoreCLR)* is the runtime used as part of .NET Core.

.NET Core and Mono have a lot of similarities but also many differences. As a developer, you have the capability to select the desired development experience you want while making the switch from one to the other as straightforward as possible.

### JIT

Since the beginning of .NET, .NET was based on a *just-in-time (JIT)* compiler to translate *Intermediate Language (IL)* code into optimized code. Microsoft built an efficient, high-performance runtime that made programming easy and efficient.

The default experience for most .NET 6 applications will use the JIT-based CoreCLR runtime, but there are exceptions: Xamarin and Blazor WebAssembly. Microsoft delivers *AOT* compilation for both projects in .NET 6.

---

**Note** AOT support has been planned for .NET 5 but finally postponed to .NET 6.

---

### AOT

The Mono compiler is an AOT compiler that allows you to compile native code that can be executed everywhere. The Blazor project uses Mono AOT compilation since .NET 6. However, AOT compilation is required for Xamarin (Android/iOS) and gaming consoles (Unity). AOT compilation is mostly intended for applications that need a quick start and a small footprint.

## The Best of Both Worlds

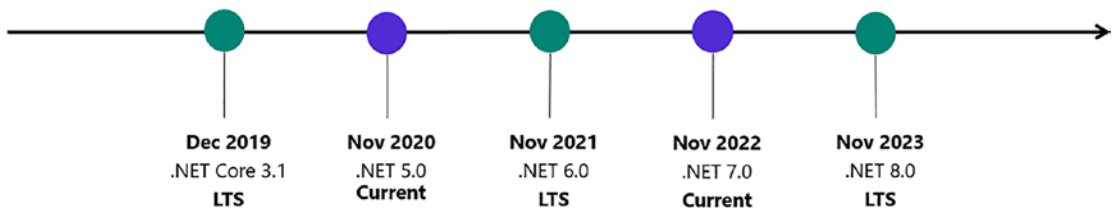
Microsoft will invest effort in improving throughput and reliability in CoreCLR while working further to improve bootability and memory consumption with the Mono AOT compiler.

Since the effort is not identical in these aspects, this doesn't mean that the investment in others will be different. For example, the diagnostic capabilities must be the same on .NET 6 for all kinds of diagnostics.

Finally, all .NET 6 applications will also build with the *.NET command-line interface* (*.NET CLI*), providing developers the same command-line tools.

## .NET Schedule and What It Means

.NET 6 will be *supported in the long term (LTS release)*, unlike .NET 5, which is why many companies have waited for .NET 6 instead of jumping on .NET 5. Only even-numbered versions will be supported in the long term. Finally, Microsoft plans to release no (or few) minor versions, and instead intends to release a major version of .NET once a year. Figure 1-2 shows Microsoft's .NET release cadence.



**Figure 1-2.** .NET release cadence (source: Microsoft)

To stay informed about upcoming releases, support information, and .NET release schedules, visit the Microsoft page “*.NET and .NET Core Support Policy*”: <https://dotnet.microsoft.com/platform/support/policy/dotnet-core>.

## How to Explore .NET 6

While this chapter aims to introduce you to .NET 6, this book will not cover this framework in detail. The primary focus of this book is to help you learn how to begin using gRPC and ASP.NET Core 6. However, I will list some notable improvements made

since .NET Core 3.1 and explain why they are so good, then show you how to install .NET 6 so you can take full advantage. Because .NET 5 is a lightweight version of .NET 6, I will recap what .NET 5 introduced so that you understand why .NET 6 and its improvements make it a modern .NET platform.

## .NET 5 and 6 Improvements

Before .NET 5, Microsoft was responsible for maintaining about 100 repositories between ASP.NET Core, .NET Core, and Entity Framework Core—making things quite difficult. Microsoft has significantly simplified this by offering three consolidated repositories; if you want to find out more about .NET 6 (or even contribute to the open source projects), you can visit the following repositories, which will make it easier for you to understand what’s going on and what you can do if you want to contribute:

- The runtime, which combines the previous repositories dotnet/corefx, dotnet/coreclr, and dotnet/core-setup (<https://github.com/dotnet/runtime>)
- ASP.NET Core, which combines several repositories from the ASP.NET organization (<https://github.com/dotnet/aspnetcore>)
- The .NET SDK, which combines the previous repositories dotnet/sdk and dotnet/cli (<https://github.com/dotnet/sdk>)

In terms of performance, .NET 5 has several huge improvements, which makes .NET 6 (and 5) significantly faster:

- Much more efficient machine code generated by the JIT compiler. While I can’t list them all here, the following is the GitHub repository if you want to know more: <https://github.com/dotnet/runtime>
- Many improvements to the garbage collector (GC).
- Improved HTTP/1.1 and HTTP/2 performance.
- Improved performance of extensions on strings (two to five times faster).
- Performance improvement for ARM64-type processors.
- Reduction in the size of container images such as Docker.

The list of other improvements is too extensive to include here. However, you can check out the interesting links on Microsoft's blog detailing their announcements as the previews were released: <https://devblogs.microsoft.com/dotnet/>.

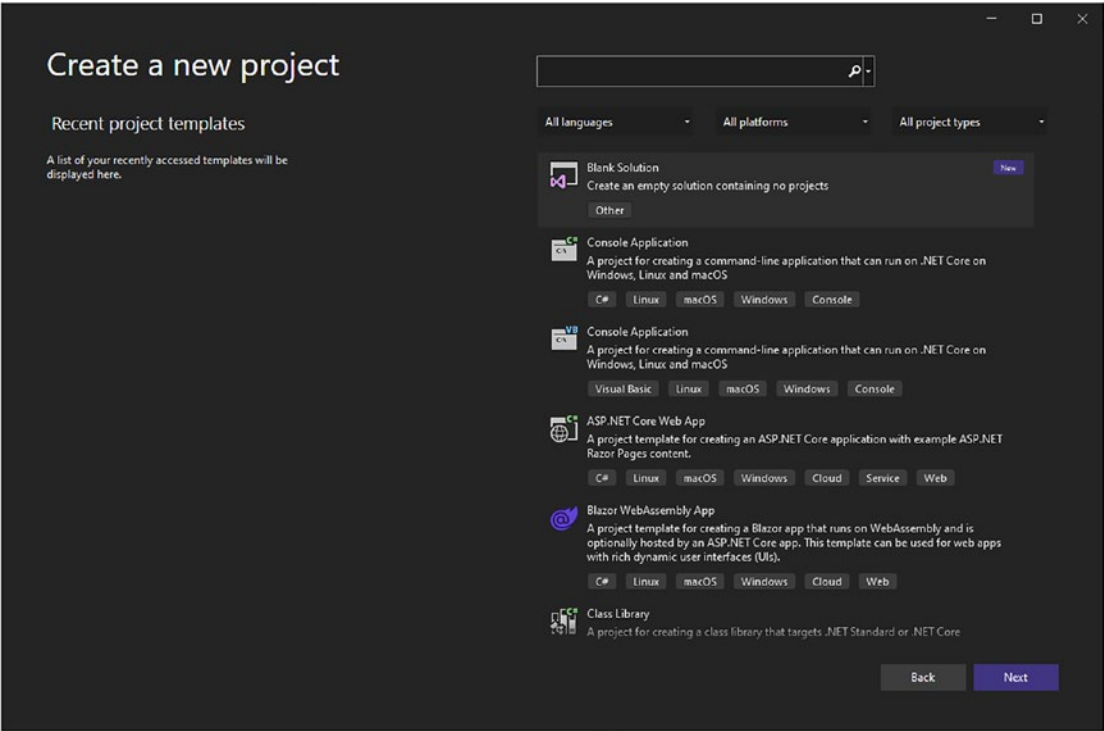
Regarding .NET 6 specifically, here is what it offers:

- Support of HTTP/3, which offers development opportunities in the web world
- Unification of Xamarin through MAUI, which provides a unified .NET experience across many devices
- AOT compilation in MAUI and Blazor, which makes applications faster because the code is not compiled at the first application execution (which can cause slowness)
- Hot reload, which allows you to modify your code without restarting your app, making the development experience faster

## Get Started with .NET 6

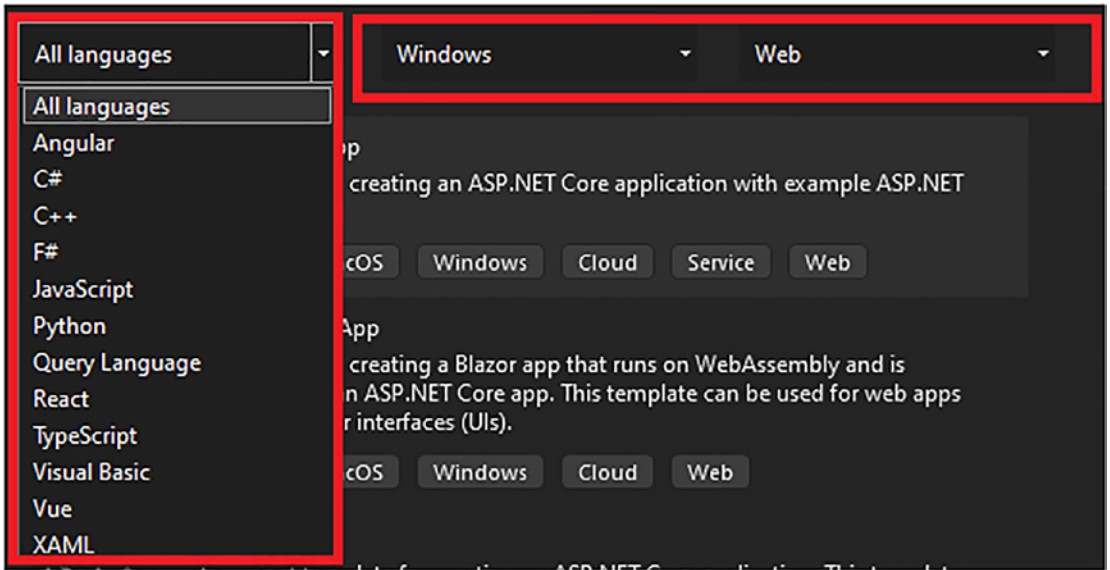
Now let's take a quick tour of .NET. Before we get started, you will want to set up your environment. If you haven't already done so, go ahead and download Visual Studio 2022 from here: <https://visualstudio.microsoft.com/vs/>. The latter included all what you need to get started, even the .NET 6 SDK.

Now that you have your environment set up, let's begin by looking at the templates you can use in Visual Studio 2022. Figure 1-3 shows the main Visual Studio project creation window with all available project types.



**Figure 1-3.** Visual Studio 2022 main project creation window

Visual Studio 2022 introduced a great context menu to choose the language, the project type, and the platform you want to use for your new project, as shown in Figure 1-4.

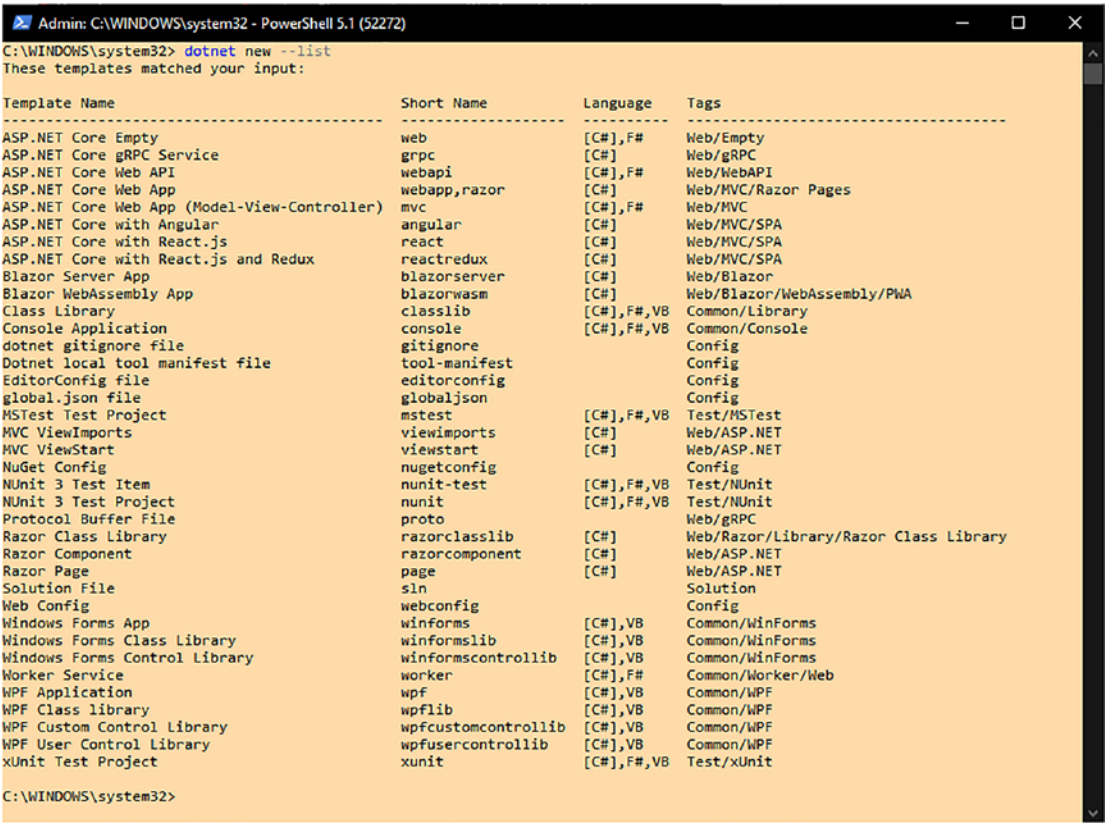


**Figure 1-4.** Visual Studio's 2022 context menu

If you prefer, you can also use the .NET CLI to get the same information by opening a terminal window and entering the following:

```
dotnet new --list
```

The output of this command is shown in Figure 1-5.

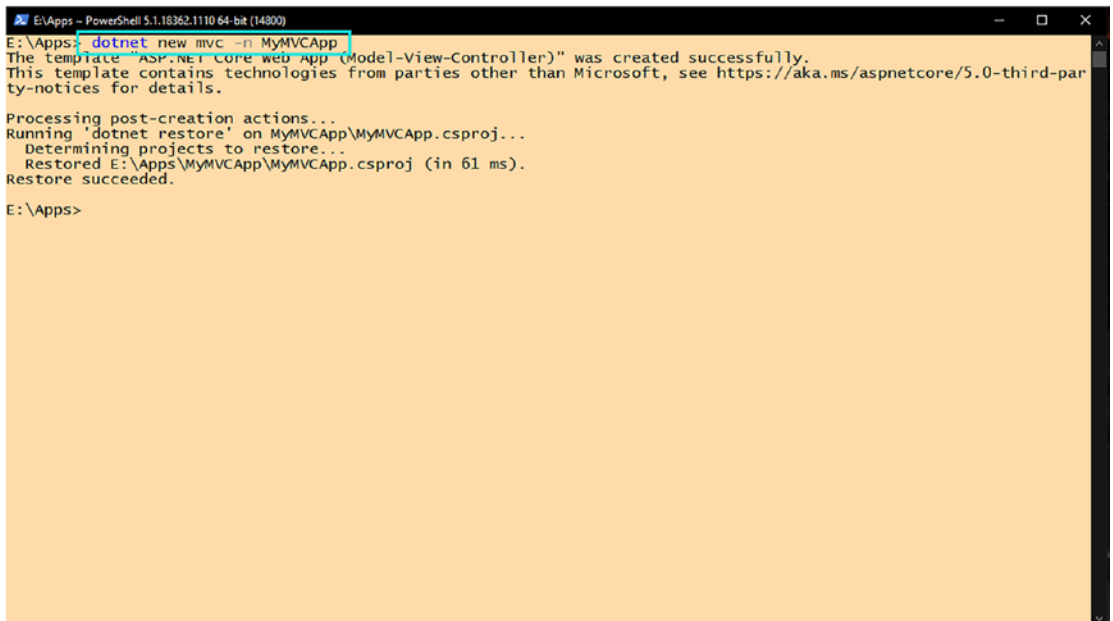


**Figure 1-5.** All available project types and languages from the command line

Personally, I like both ways to create a project. Both are simple. Let’s now create a new project named **MyMVCApp**, where -o allows to specify the project name (and its folder name), which uses an ASP.NET MVC template with the command as seen here:

```
dotnet new mvc -o MyMVCApp
```

The command output is shown in Figure 1-6.

A screenshot of a PowerShell terminal window titled "E:\Apps - PowerShell 5.1.18362.1110 64-bit (14800)". The terminal shows the command `dotnet new mvc -n MyMVCAApp` being executed. The output indicates that the "ASP.NET Core web App (Model-View-Controller)" template was created successfully and that the project dependencies were restored. The prompt `E:\Apps>` is visible at the bottom.

```
E:\Apps - PowerShell 5.1.18362.1110 64-bit (14800)
E:\Apps> dotnet new mvc -n MyMVCAApp
The template "ASP.NET Core web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/5.0-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on MyMVCAApp\MyMVCAApp.csproj...
  Determining projects to restore...
    Restored E:\Apps\MyMVCAApp\MyMVCAApp.csproj (in 61 ms).
Restore succeeded.

E:\Apps>
```

**Figure 1-6.** The output generated after creating a new project with the .NET CLI

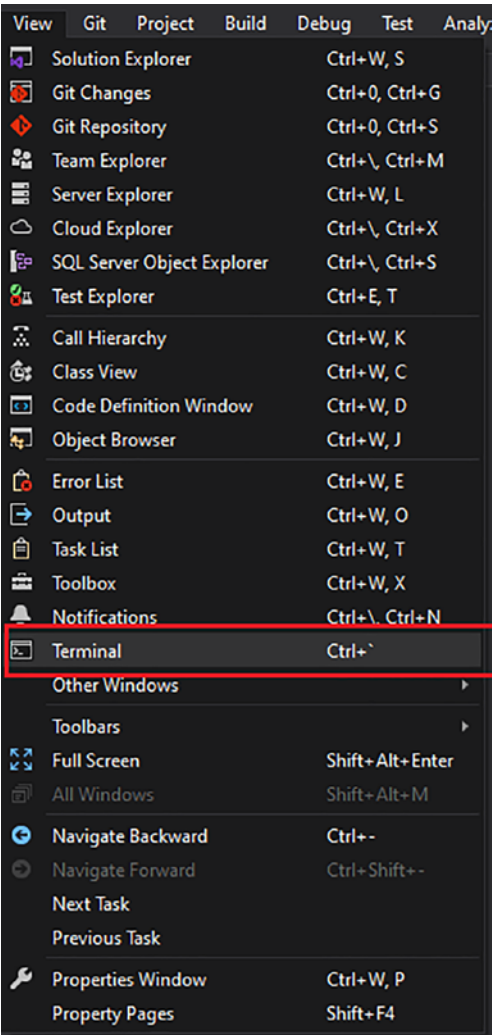
To confirm your project is set up to use .NET 6, you can build it by running the following command:

```
dotnet build
```

Or, if you want to build and run your project, you can use the following command:

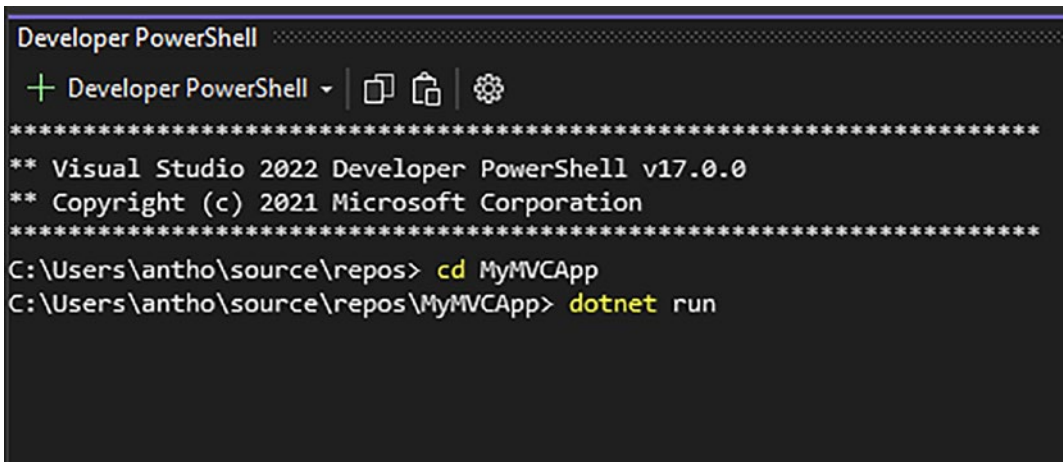
```
dotnet run
```

You can integrate the new Windows Terminal with Visual Studio. To enable it in Visual Studio 2022, click the View menu and choose Terminal, as shown in Figure 1-7.



**Figure 1-7.** Enabling the new Windows Terminal in Visual Studio 2022

Once completed, the terminal window appears in the bottom panel. Then, you’ll be able to run any command you want—such as PowerShell, Git, and CLI commands, as shown in Figure 1-8.



```
Developer PowerShell
+ Developer PowerShell | [Icons]
*****
** Visual Studio 2022 Developer PowerShell v17.0.0
** Copyright (c) 2021 Microsoft Corporation
*****
C:\Users\antho\source\repos> cd MyMVCAApp
C:\Users\antho\source\repos\MyMVCAApp> dotnet run
```

**Figure 1-8.** *Running a command in the terminal window*

It’s is my favorite feature because I don’t need to open a new window on my computer. I don’t know about you, but I find it a bit annoying to deal with multiple windows. On Visual Studio, no problem! The Terminal is integrated into the existing menu, positioned by default at the bottom of the menu, and you can easily drag it and drop it elsewhere!

## Recap of C# 9 and Introduction to C# 10

We can’t discuss the C# language without mentioning the latest updates with C# 9 and C# 10. Although this isn’t a C# programming book, I’ll help you discover the new features because most of my examples use C# 9 and C# 10 features. Going into detail about each version of C# is beyond the scope of this book, so I strongly recommend that you visit this web page that describes all the C# versions and their main features: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>.

## Recap of C# 9

Here are the most important improvements introduced in C# 9:

- Init-only properties
- Records
- Improved pattern matching

- Improved target typing
- Covariant returns
- Static anonymous functions

## Init-Only Properties

C# 9 introduced an `init` accessor, a variant of the `set` accessor. This accessor allows properties to be assigned once during object initialization. If you apply this accessor to all the properties of your object, it makes the object immutable. If you try to reassign a property initialized with this accessor, the compiler will warn you of an error. Listing 1-1 shows an example of a `Product` class with its immutable `CategoryId` property; this code could be created in any C# project.

### **Listing 1-1.** Product Class with `CategoryId` Property and Its `init` Accessor

```
using System;
namespace CSharp9Demo.Models
{
    public class Product
    {
        public string Name { get; set; }
        public int CategoryId { get; init; }
    }
}
```

## Records

C# 9 added a new `record` keyword. A record makes it possible to create an immutable reference type object (either with the `init` accessor or a primary constructor) and give it a value type object for comparison. Listing 1-2 shows an immutable record with `init`-only properties, and Listing 1-3 shows an immutable record with a primary constructor.