Native Desktop Applications with .NET 8

Build cross-platform apps using .NET MAUI, Blazor Hybrid, and Native UI

Sai Kumar Koona

www.bpbonline.com

First Edition 2025

ISBN: 978-93-55519-313

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.

Dedicated to

My beloved parents:
Vijay Kumar Koona *and* ***Narmada***
My wife ***Sowmya***
and
My son ***Jatin Koona***

About the Author

Sai Kumar Koona completed his master's program in computer applications at Osmania University in Hyderabad, India. He began his career with C++ and C# since he had a strong passion for programming, which led him to write the book "In-depth C++ Programming Language". He now has over 15 years of experience working with Microsoft technology and has created numerous tools and reusable components that have aided in the faster and smoother development of products.

As a developer and tech lead, he was involved in developing several software applications. Currently, he works as a Technical Architect. He has created several intricate apps using .NET, Docker, Kubernetes, Microservices, Azure, SQL Server, and other open-source tools to make them more scalable, performant, and maintainable.

Sai has been designated an MVP (Most Valuable Programmer) in the C# language and .NET area by C# Corner for his multiple informative blogs/articles that share his wealth of knowledge and experience. He likes to keep up with technology developments and is enthusiastic about sharing his knowledge and expertise with user groups and inspiring the next generation of developers.

About the Reviewers

- For more than 10 years, **Imad Sayah** has been designing and delivering software solutions for financial institutions as a seasoned Technical Lead and Release Manager. His expertise spans Domain-Driven Design (DDD), .NET, Docker, Kubernetes, and Azure DevOps, providing a robust technical foundation that drives innovative and effective solutions.

  Driven by a commitment to pushing the boundaries of what's possible, Imad Sayah excels in creating scalable and efficient systems that meet the complex demands of the financial sector. His deep understanding of DDD ensures that software architecture aligns seamlessly with business needs, facilitating impactful and strategic solutions.

  As a dedicated advocate for technological advancement and team empowerment, Imad Sayah is committed to fostering an environment where creativity and technical excellence thrive. By leveraging cutting-edge tools and methodologies, Imad Sayah drives projects' success and inspires and empowers his team to achieve outstanding results.

- **Yassin Lokhat** started his programming journey in 2008 at the age of 15, discovering the vast world of C programming. His dedication and talent quickly became evident, and in 2010, he was qualified to represent Madagascar at the International Olympiad in Informatics (IOI). Driven by a passion for technology, Yassin pursued higher education at the Polytechnic High Institute of Madagascar. Throughout his five years of rigorous study, he learned and used C# and desktop development, graduating with honors in Electronics, Computer Systems, and Artificial Intelligence. With his distinguished diploma in hand, Yassin ventured to Mauritius to further his professional career. During his five-year journey there, he honed his skills and expertise, becoming a senior software engineer. After these productive years, he returned to Madagascar, taking on the leadership of the Malagasy team within his project. Currently, Yassin serves as the Senior Software Engineer - Technical Lead for the Astek Group, where he continues to drive innovation and excellence in the field of development. His journey is a testament to his unwavering commitment to growth and his passion for technology.

Acknowledgement

I want to express my deepest gratitude to my family and friends for their unwavering support and encouragement throughout this book's writing, especially my wife Sowmya, and my son Jatin.

I am also grateful to my brother Santosh K for being such an incredible mentor to me. Your insights and advice have helped me navigate challenges with clarity and confidence, and I feel fortunate to have you in my corner.

I'd like to thank BPB Publications for their guidance and expertise in bringing this book to fruition. It was a long journey of revising this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

Finally, I would like to thank all the readers who have taken an interest in my book and for their support in making it a reality. Your encouragement has been invaluable.

Preface

The goal of this book is to give readers a thorough understanding of the .NET ecosystem and the evolution of desktop frameworks since the platform's inception in the early 2000s.

Throughout the book, you will learn about various native-desktop application frameworks available in the .NET ecosystem like WinForms, Windows presentation framework (WPF), Multi-platform App UI (.NET MAUI), Windows App SDK, and Blazor.

This book also serves as a thorough guide for senior engineers and application architects who need a solid foundation to understand numerous architectural principles. It offers a list of best practices for all programmers to follow while developing logic.

With this book, you will gain the knowledge and skills to become a proficient developer in the field of native-desktop application development using C# and .NET.

Chapter 1: Introduction to .NET 8 - In this chapter, we will cover the evolution of .NET Technology from its inception and the issues with the existing .NET Framework, which led to the introduction of .NET Core. Later, we'll look at the obstacles that .NET Core faced and how they were overcome. Next, the versioning strategy of the .NET platform will be discussed. After that, there will be a brief introduction to the Command Line Interface (CLI). Then, there will be an overview of the many supported Desktop-based UI Frameworks.

Chapter 2: Exploring .NET 8's Features - In this chapter, we will cover the most recent .NET version is .NET 8, and since it is a Long-Term Support (LTS) version, support will be available for the next three years or beyond. Thus, this chapter's main goal is to give readers a thorough grasp of the most important features that came with the .NET 8 release.

Chapter 3: Working with Command Line Interface - In this chapter, we will cover the explanation of Command Line Interface (CLI). The most popular commands for creating, building, publishing, and testing any installed.NET application template on the workstation will then be discussed, along with knowing where to find it and how to utilize it.

Chapter 4: Working with Windows Forms - In this chapter, we will cover how the .NET Framework was first published in 2002 when WinForms was the sole platform available for developing desktop applications. This chapter will provide a detailed introduction to the framework and walk through how to develop a form using its unique visual form designer.

It will also cover the fundamental setups needed to support multi-screen environments. After that, we'll go over the basic and widely used controls found in Windows Forms applications. Lastly, to have a deeper grasp of the platform, we will construct a basic application and explore its deployment possibilities.

Chapter 5: Working with Windows Presentation Foundation - In this chapter, we will cover how the **Windows Presentation Foundation** (**WPF**) is another GUI framework from Microsoft that comes along with .NET Framework 3.0, and we'll look at how it differs from WinForms. We will then go over an overview of XAML, or Extended Application Markup Language, which is a markup language used to design the WPF application's user interface. Next, we will learn how the binding framework functions. Afterward, we will discuss the main controls that are frequently utilized in WPF applications. To have a better knowledge of the subject, we will conclude with a simple application creation using the WPF framework.

Chapter 6: Working with Multi-platform App UI - In this chapter, we will cover Multi-platform App UI (MAUI), the most modern cross-platform framework for developing desktop and mobile applications in C# and XAML. This chapter will walk you through the steps necessary to understand how the framework works. Next, the project structure for the MAUI application will be discussed, along with the newly added controls. We'll go into more detail later about the widely used Model–View–ViewModel (MVVM) paradigm and how it improves the MAUI application. To further grasp the new and popular platform, we'll conclude with a basic application creation using the MVVM architectural pattern and an MAUI application.

Chapter 7: Working with Windows App SDK - In this chapter, we will cover how the Windows App SDK is a set of components and APIs that aid in the development of applications running on Windows 10 and later operating systems. Next, a thorough explanation of WinUI's evolution will also be explored. Later, we'll look at the project structure and functionality of the WinUI 3.0 application. Using this development kit, we will design a basic application and learn how to package it so that it can run on any version of Windows 10 or later.

Chapter 8: Working with Blazor - In this chapter, we will cover that Microsoft has released a new Web framework called Blazor for developing web-based apps. We shall gain more knowledge about this framework and its types of hosting. We'll talk about the Razor components later, where they'll be utilized to build the application's user interface. This is essentially a web framework, but we can also use Blazor Hybrid apps—which we will go into detail about—to combine this application with WinForms and MAUI applications. Afterward, we'll build a simple application with this framework to see how Blazor Hybrid

apps function, particularly with MAUI, which eliminates the requirement for specialized XAML knowledge if you come from the web to create desktop or mobile apps.

Chapter 9: Application Architecture - In this chapter, we will cover the various architectural concepts, including monolithic and microservices, as well as their benefits and drawbacks. Next, using simple use cases, we will learn about some software design patterns and SOLID principles. Afterwards, will delve deeply into the new gRPC platform, which is mostly used for service-to-service communication, particularly in Microservices design, and then will learn about the Docker tool. This chapter is dedicated to senior engineers or application architects.

Chapter 10: Best Practices - This chapter covers the best practices for building code using entity frameworks, asynchronous programming, and so on. Every time a developer writes code, they should adhere to some industry best practices.

Code Bundle and Coloured Images

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

https://rebrand.ly/juwv0iq

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Native-Desktop-Applications-with-.NET-8.
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.

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**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction to .NET 8

Introduction

This chapter aims to introduce the overview of the evolution of .NET Technology since it launched in the early 2000s and understand the problems with the existing .NET Framework, which led to the development of a new platform called .NET Core from scratch. The next part of this chapter is to understand the challenges faced by .NET Core, which leads to thinking of building a new ecosystem to achieve a unified development platform, meaning how it brings a bunch of separate .NET-based technologies together under one umbrella.

Next, we will introduce the **command-line interface** (**CLI**), a powerful tool packed with .NET Core. As this chapter is to give some introduction, we will walk through the basic commands.

If you are new to .NET-based desktop applications or an experienced developer, this introduction section on *Development with native desktop application* will provide you the complete details on the evaluation of desktop frameworks since the launch of .NET Framework and understanding how these applications give more advantage when compared with web-based applications.

Later, we will discuss various supported desktop-based UI Frameworks from WinForms, WPF, UWP, and WinUI to .NET MAUI and their features.

Structure

In this chapter, we will cover the following topics:

- Understanding the .NET ecosystem
- Understanding one .NET vision – A unified development platform
- Support tracks and release cycles
- Supported operating systems
- Command-line interface
- Development with native desktop applications
- Universal windows platform
- Windows UI library
- .NET Multi-platform App UI (.NET MAUI)

Objectives

By the end of this chapter, the reader will understand the evolution of .NET Technology, .NET Core, a unified development platform, and desktop technologies on .NET Framework and .NET (core).

Understanding the .NET ecosystem

Microsoft started developing the .NET Framework in the late 1990s, and the first version of the .NET Framework was released on 13th February 2002. Before the .NET Framework, many technologies and frameworks were used to develop Windows software applications. For example, **Component Object Model** (**COM**), **Distributed COM** (**DCOM**), and COM+ are the technologies used to communicate between the components, performing transactions, and messaging services. Every new version of Windows extends the existing API by introducing new features or functions. So, there will always be an issue with backward compatibility when using software on different versions of Windows.

Microsoft identified that it needs to upgrade the development tools and languages whenever a new feature is added. It is becoming more and more complex down the line to maintain. So, Microsoft decided to build a new platform from scratch to provide a simple and sophisticated set of languages, environments, and tools by which developers can write neat code to develop software that supports backward compatibility.

Now all the existing software will still work, and .NET was designed to communicate with these legacy programs as on Windows, communication between software programs will happen using COM. Considering this, .NET can have a wrapper around the existing COM components to communicate with .NET components.

.NET Framework is a software development framework for building and running applications on Windows.

Many changes and features are added to every version of the .NET Framework; the latest version is .NET Framework 4.8.1 while writing this chapter. This framework is a Windows-only development platform for desktop and web application development.

Exploring .NET platforms

.NET platform is used for the development of desktop and web applications. Similarly, the Xamarin platform is used to develop the mobile-based application, which uses the .NET framework's cross-platform mono implementations. The Xamarin framework is an alternative platform for building applications targeting mobile OS like Windows, Android, and iOS.

These two .NET platforms, .NET Framework and Xamarin Framework, are separate. Also, these two platforms have components with almost similar API implementations. Each platform has its own set of libraries; some of them contain base classes, and others are specific to supported app models, as shown in *Figure 1.1*:

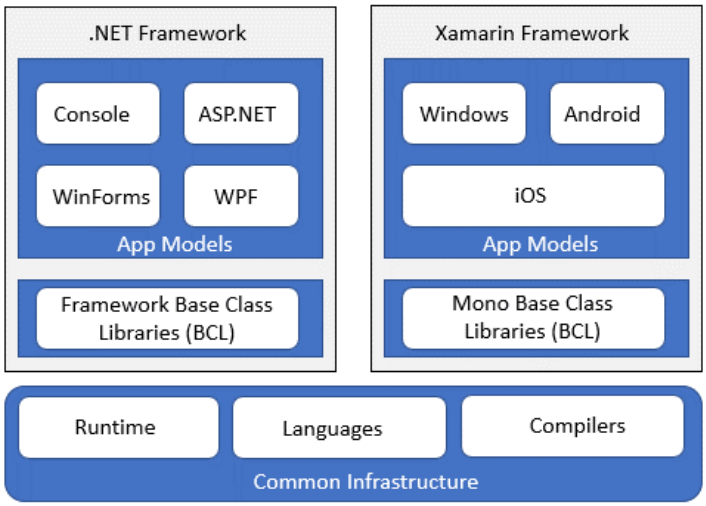

***Figure 1.1**: Comparing .NET Framework and Xamarin with layers*

The above figure shows a stack of three layers on each platform, and the top layer contains libraries specific to their supporting app models. For example, the .NET Framework supports developing various app models such as console, desktop, web, and **Windows Presentation Foundation** (**WPF**). On the other end, the Xamarin Framework contains libraries that can be used to develop mobile applications for Windows, Android, and iOS. However, we cannot create a console-based application using the Xamarin platform, and similarly, we cannot develop an Android-based application using the .NET Framework.

The next layer on the stack is a **Base Class Library** (**BCL**) for the .NET Framework, and the Xamarin platform contains a BCL with mono implementations. These BCLs contain fundamental .NET types such as *int*, *string*, streams, accessing files, and other APIs. Even

though these BCLs contain similar APIs, fundamentally, both are different; that is, we cannot guarantee the given type, method, or property will be contained on both platforms.

The bottom layer consists of compilers, languages, and **common language runtime** (**CLR**); without this tooling, we cannot build or deploy our applications on any .NET-based platforms. Here we compared only a couple of .NET-based platforms. Still, we have many other frameworks, such as **Universal Windows Platform** (**UWP**), Unity for gaming applications, IoT, etc., where we see the same BCLs specific to their platforms having many similar APIs.

Till now, everything looks good as each platform has a separate set of app model development life cycles. But the real problem is the BCLs of each platform, as they are not fundamentally interoperable, that is, we cannot share or use the libraries built on one platform and want to use them on another .NET-based platform. To do this, the developer should know the platform APIs and change the code slightly with specific API methods or properties to support the targeting platform. Another problem Microsoft identified is that the present software development has rapidly changed by developing small, lightweight, and cloud-support web applications such that they can be deployed in a cross-platform environment. But the current .NET Framework is tightly coupled with Windows and does not suit modern cloud-native web application developments.

Microsoft wanted to start building a new framework from scratch that supported all these scenarios and started a project with the internal name *Project K* during development. Later it was officially named *.NET Core*.

Understanding .NET Core

.NET Core is a cross-platform, high-performance, and open-source framework for building modern, cloud-based applications. This platform has its own cross-platform BCL libraries called **CoreFX** as it contains types for file systems, collections, JSON, XML, and many more, and cross-platform runtime called *CoreCLR* includes the garbage collector, JIT compiler, and other low-level classes which are used to execute .NET core applications.

This platform is bundled with its own CLI tooling, with which we can create and manage the .NET Core-based applications without any rich IDE support. It also has rich ASP.NET Core and EF Core libraries by which we can build cloud-based native Web applications with high performance. This highly modular platform can deploy side-by-side with other .NET Core version installations. Another feature of this platform is that it is self-contained. During deployment, we can opt to have complete .NET Core libraries distributed along with the application. With this feature, the target system does not need .NET installed in advance, and the application will work independently.

With the introduction of the .NET Core platform, we overcame one of the abovementioned problems: Developing modern, cloud-based applications supporting deployment on cross-platform environments. But another problem still exists: sharing the libraries across different platforms in the .NET ecosystem. .NET Core is another new platform introduced

along with other existing platforms, as discussed in previous sections, with its own set of BCLs and runtime. Microsoft introduced .NET Standard to solve this library sharing across the platforms (*Figure 1.2)*:

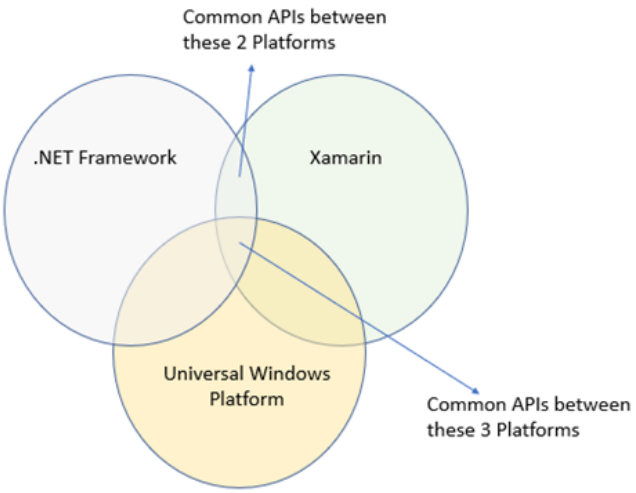

***Figure 1.2**: Overlapping common APIs between .NET platforms*

The .NET Standard, the name itself, suggests that it ensures a standard set of APIs is available on every .NET platform, as shown in *Figure 1.2*. The developer does not need to know the other platform APIs, and the logic built using the .NET Standard can be re-used on other platforms. For example, write a logic to send an email and build a library using the .NET Standard without any changes. We can use the same library with .NET Framework, .NET Core, Xamarin, and Silverlight platform.

Still, we are left with the fundamental issue of managing multiple platforms. Each platform has its source code, and Microsoft needs to manage it separately, even though most are fundamentally identical in most areas. Also, Microsoft is quickly introducing new features to the .NET Core platform and new C# language versions to make more productive development with high performance. But unfortunately, none of the other platforms are taking advantage of these without doing additional work. Microsoft wants to handle this by introducing a *One .NET*.

Understanding One .NET vision: A unified development platform

To develop a desktop application, you need to use the .NET Framework. Similarly, suppose you want to develop a Windows, Android, and iOS mobile application. In that case, you need to use Xamarin Framework, and if you want to develop cross-platform web applications, you need to use .NET Core. Here, each app model is tightly bound to its BCL platform.