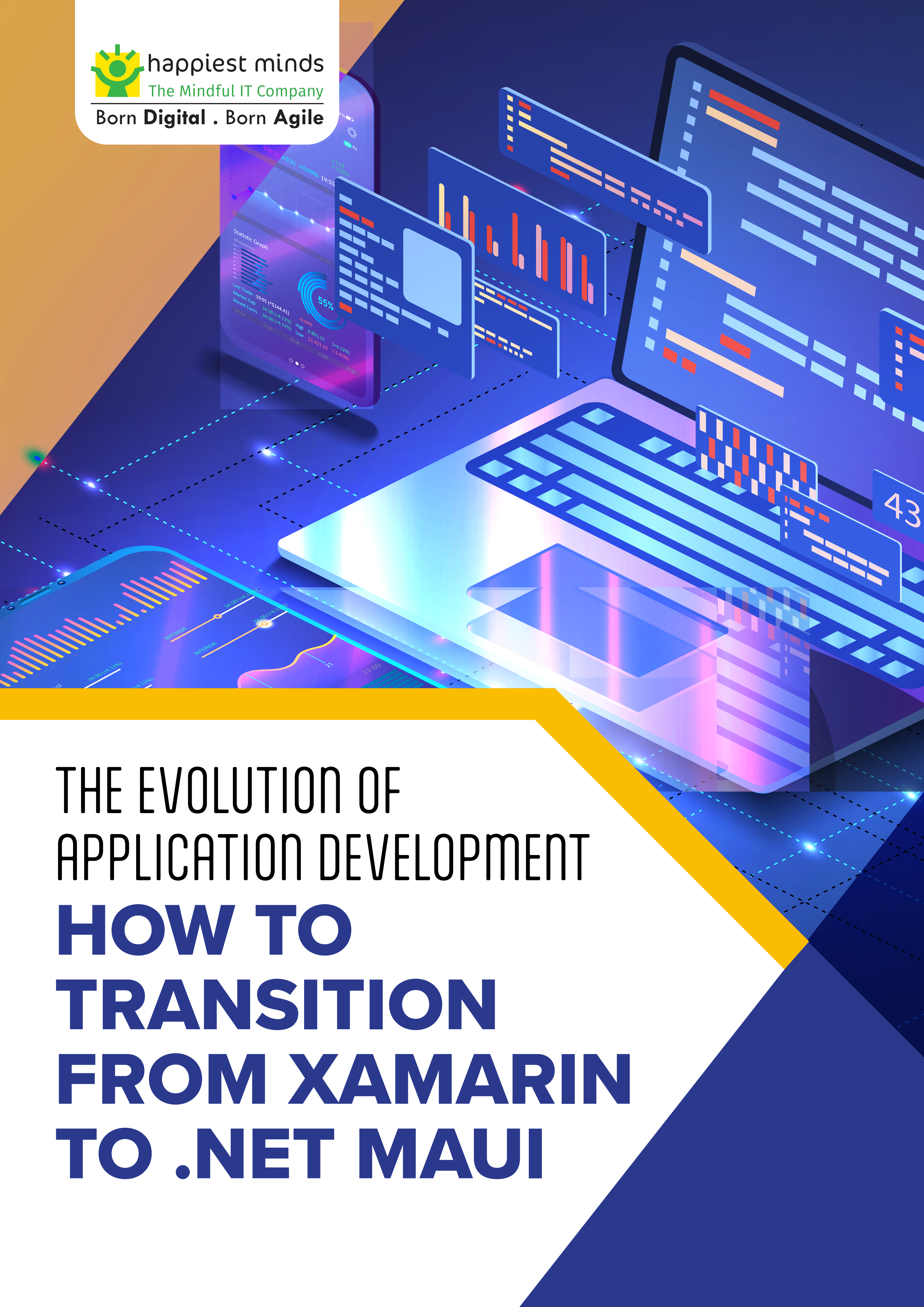
happiest minds
The Mindful IT Company
Born **Digital** . Born **Agile**



THE EVOLUTION OF
APPLICATION DEVELOPMENT

HOW TO TRANSITION FROM XAMARIN TO .NET MAUI

TABLE OF CONTENT

1 INTRODUCTION

Microsoft's strategic shift from Xamarin to .NET Multi-platform App UI (.NET MAUI) signifies a pivotal transformation in the mobile and desktop application development landscape. This evolution transcends a mere change in technology; it embodies Microsoft's foresight in unifying and refining the framework for crafting cross-platform applications. With .NET MAUI, developers are ushered into a new era where the efficiency of development and the performance of applications are paramount, ensuring a seamless and robust experience across all platforms.

1.1 Understanding Microsoft's Decision to Retire Xamarin

The journey from Xamarin to .NET MAUI is rooted in a vision that seeks to harmonize the development process by integrating all platform-specific needs into a singular, comprehensive framework. Xamarin, once a beacon for cross-platform development using C#, has paved the way for this transition, setting the stage for a more advanced framework. Microsoft support for Xamarin will end on May 1, 2024, for all Xamarin SDKs, including Xamarin.Forms. The move towards .NET MAUI reflects a strategic pivot aimed at enhancing developer productivity and application performance, ensuring that the future of app development aligns with the rapidly advancing technological environment and user expectations for dynamic, responsive applications.

1.2 The Catalysts for Change: WHY .NET MAUI?

Microsoft's embrace of .NET MAUI is driven by several compelling factors that promise to reshape the development landscape:

Simplified Project and Development Experience:
.NET MAUI simplifies the project structure and unifies the development experience, making it easier to manage code and reduce complexity across different platforms. This change improves workflow efficiency and makes it easier to bring ideas to life.

Modern, Adaptive UI Toolkit:
It introduces a modern UI toolkit that offers adaptive layouts and controls, ensuring that applications provide a consistent and high-quality user experience on any device, automatically adjusting to different screen sizes and orientations.

Enhanced Performance and Flexibility:
Building on the improvements from .NET 6, .NET MAUI boosts application performance, offering faster startup times and smoother interactions. It also addresses previous performance issues, ensuring applications are responsive and fluid across all devices.

Comprehensive Ecosystem and Advanced Feature Integration:
Developers have access to the extensive .NET ecosystem, allowing for the integration of advanced features such as AI, machine learning, and cloud services. This capability enables the creation of complex, feature-rich applications that are both innovative and cross-platform.

1.3 Implications for Developers: NAVIGATING THE TRANSITION

The shift to .NET MAUI is a monumental step for developers, offering a blend of challenges and opportunities:

Skillset Evolution

For developers versed in Xamarin, the transition to .NET MAUI is an opportunity to evolve their skillset, leveraging the familiar .NET principles and C# language in a more advanced framework. This evolution is supported by Microsoft's commitment to providing comprehensive resources, ensuring developers can seamlessly adapt to and excel in the new environment.

Resource Availability

Microsoft's dedication to facilitating this transition is evident in the wealth of resources made available to developers. From detailed documentation and hands-on tutorials to active community forums, developers are well-supported as they navigate the nuances of .NET MAUI, ensuring they can leverage the full potential of the framework.

Future-Proofing Projects

Embracing .NET MAUI is not merely about keeping pace with technological advancements; it's about setting a foundation for the future. This framework ensures that applications remain at the forefront of performance, usability, and relevance, securing a competitive edge in the ever-evolving digital landscape.

1.4 What's Next?

As we transition from Xamarin to the broader horizons of .NET MAUI, it's clear that this evolution signifies more than just a technological update; it's a significant advancement in cross-platform development practices. .NET MAUI brings the promise of unified codebases for mobile and desktop platforms, improved performance, and an expansive feature set, setting new benchmarks for application development.

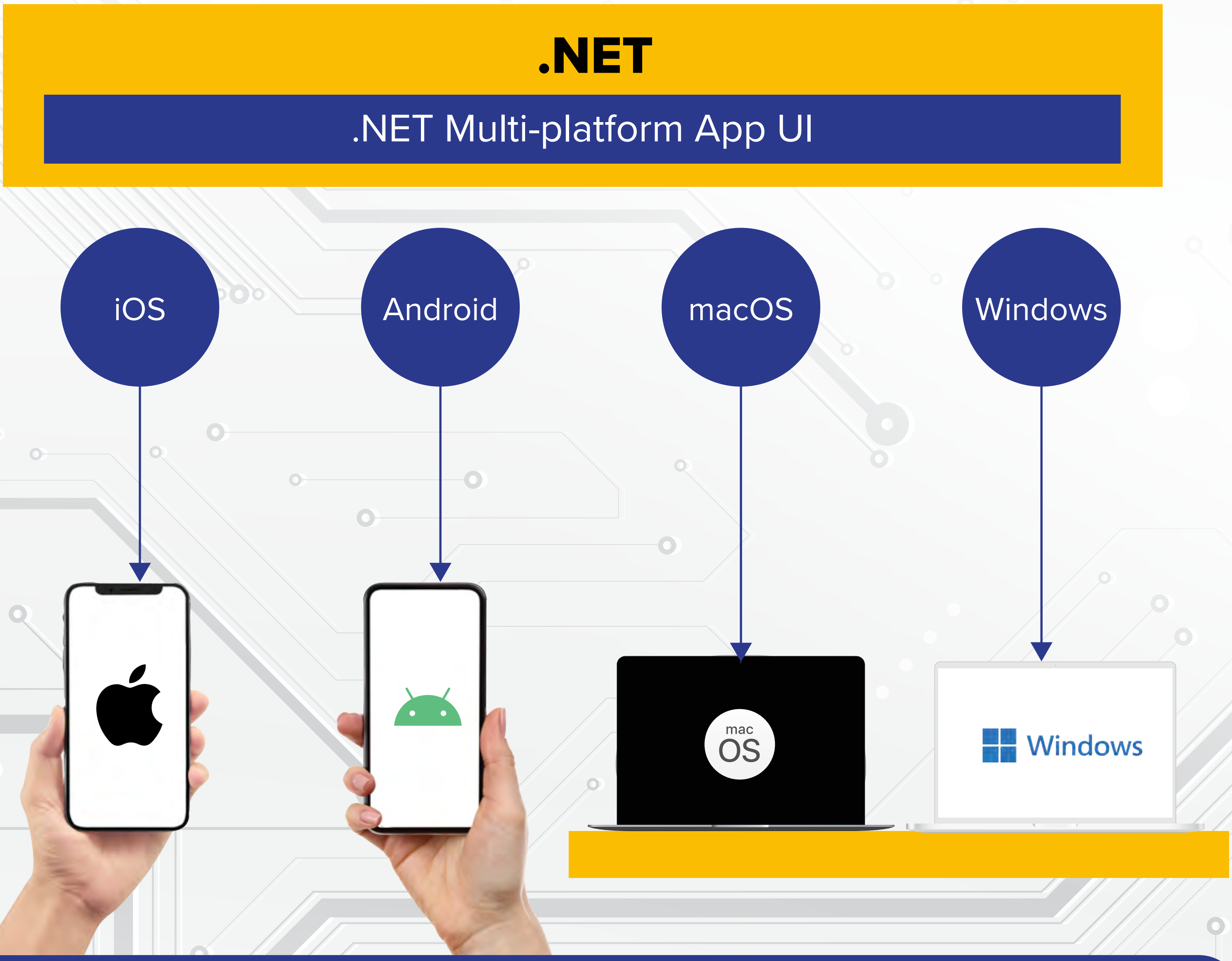
In our forthcoming discussion, we will closely examine .NET MAUI's architecture and capabilities, particularly highlighting its role in enhancing Xamarin's development paradigms. This exploration is crucial for understanding the migration process from Xamarin—both Forms and Native—ensuring developers are well-prepared to utilize .NET MAUI effectively in their projects.

Our focus will sharpen migration strategies and critical considerations, equipping developers with the insights needed to fully leverage .NET MAUI's potential to upgrade their mobile applications.

.NET Multi-platform App UI (.NET MAUI)

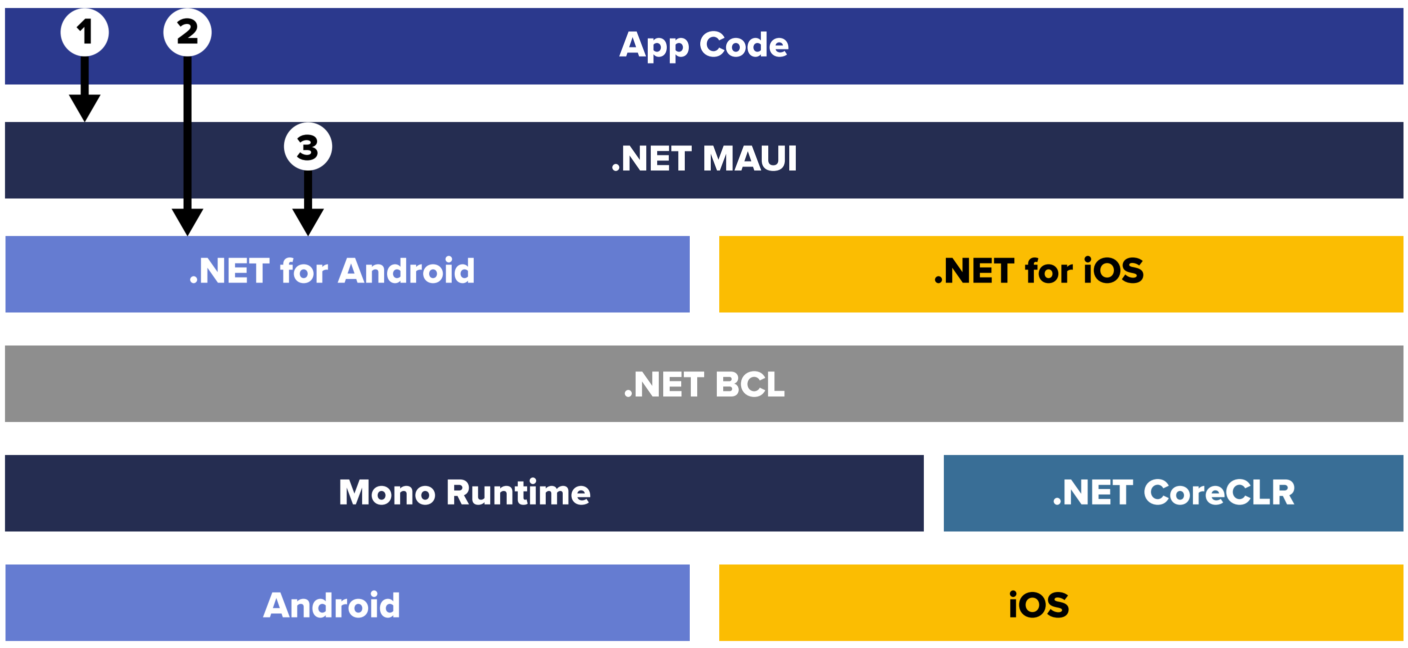
Microsoft's Latest Cross-Platform Framework

2.1 Exploring the Essence of .NET MAUI



.NET Multi-platform App UI (.NET MAUI) is Microsoft's cross-platform open-source application development framework that allows you to create native mobile and desktop applications using C# and XAML. It is designed to unify and enhance the development experience for building native mobile and desktop apps that can run on Android, iOS, macOS, and Windows from a single shared code base. While .NET MAUI directly succeeds Xamarin.Forms, it also addresses the broader Xamarin ecosystem, including Xamarin.Native (Xamarin.Android and Xamarin.iOS). Developers who have utilized Xamarin.Native for platform-specific functionalities can find solace in .NET MAUI's architecture, which incorporates these native capabilities seamlessly into its framework, ensuring that all Xamarin developers, regardless of their focus, benefit from the transition.

2.2 Understanding How .NET MAUI Operates



The diagram above provides a high-level overview of the .NET MAUI app architecture. At its foundation, .NET MAUI capitalizes on the advancements of .NET 6, integrating with specialized frameworks for each platform while sharing a common .NET Base Class Library. This shared library abstracts the complexities of each platform, supported by Mono for Android, iOS, and macOS and by .NET CoreCLR for Windows, thus standardizing the development environment across ecosystems.

The innovation of .NET MAUI lies in its ability to streamline the traditionally fragmented process of UI development. Where once separate codebases were the norm for different platforms, .NET MAUI introduces a cohesive framework, enabling the design of versatile user interfaces with minimal redundancy.

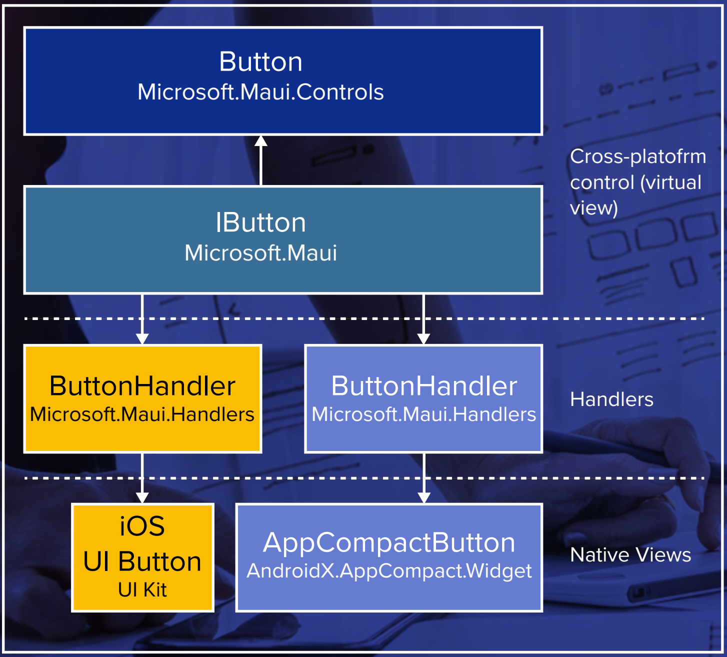
Developers engage primarily with the .NET MAUI API, which acts as the conduit to native platform functionalities. This direct interaction ensures that applications can fully utilize the unique features of each platform, offering the flexibility to incorporate platform-specific APIs into the development process when necessary.

Compiling with Versatility:

On Android, the transition from C# code to native assembly is managed through just-in-time (JIT) compilation, optimizing launch times.	For iOS, ahead-of-time (AOT) compilation translates C# into native ARM assembly code, ensuring efficient performance.	macOS applications leverage Mac Catalyst to extend iOS apps to the desktop environment, integrating additional APIs for a broader application scope.	Windows applications utilize the WinUI 3 library, focusing on creating rich desktop experiences with native UI capabilities.

NET MAUI's design philosophy balances the ease of a unified development approach with the depth needed for accessing platform-specific features. This framework empowers developers to craft applications that are not only versatile across platforms but also optimized for performance, embodying the true spirit of modern, cross-platform development.

2.3 UI Handler Architecture



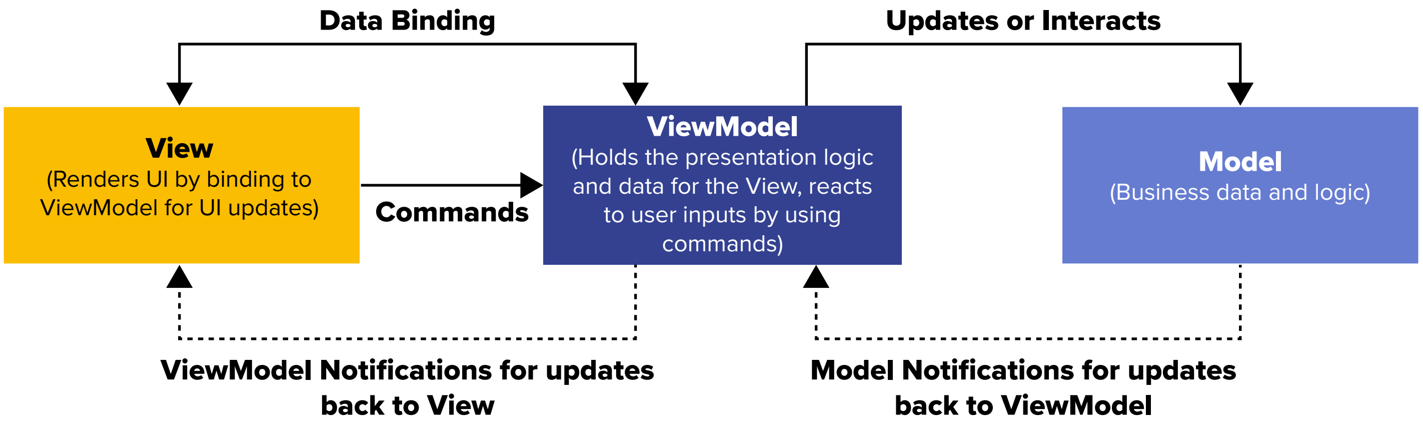
At the heart of .NET MAUI's versatility is its UI Handler architecture, which abstracts the platform-specific implementations of UI components into a set of common interfaces and handlers. This innovative approach allows for more direct communication between the shared code and the native platform controls, leading to better performance and more control over the rendering and behavior of UI elements.

Customization and Extensibility:
Developers can easily customize and extend existing controls or create new ones, thanks to the flexibility offered by the UI Handler architecture.

Performance:
Reduces the overhead typically associated with cross-platform UI rendering, ensuring applications run smoothly across all target platforms.

2.4 Supported Design Patterns

2.4.1. Model-View-ViewModel (MVVM)


Data Binding
Updates or Interacts
View
(Renders UI by binding to ViewModel for UI updates)
Commands
ViewModel
(Holds the presentation logic and data for the View, reacts to user inputs by using commands)
Model
(Business data and logic)
ViewModel Notifications for updates back to View
Model Notifications for updates back to ViewModel

MVVM remains a cornerstone for developers working within the .NET ecosystem, and .NET MAUI enhances its support for this pattern. MVVM facilitates a clear separation of concerns between the presentation logic and the business logic of an application. .NET MAUI’s data binding capabilities, combined with the powerful XAML for UI definition, allow developers to create responsive and dynamic interfaces while maintaining clean, testable, and maintainable code.

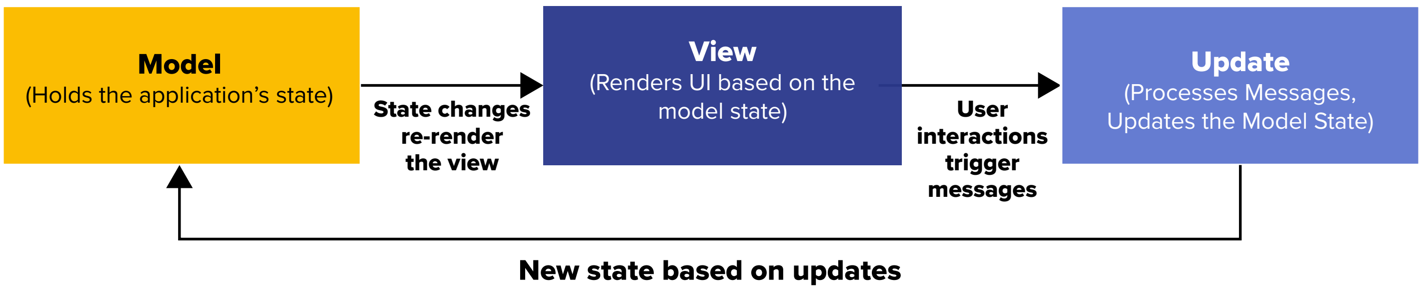
Data Binding

Allows the View to automatically update and display data from the ViewModel, ensuring the UI reflects any changes to the underlying data without manual intervention. This mechanism supports a seamless synchronization between the View and the ViewModel, facilitating a dynamic and responsive user experience.

Commands

Enable the View to send user actions (like clicks and selections) to the ViewModel, which processes these actions without directly manipulating the Model. Commands simplify event handling by encapsulating action logic, significantly reducing the need for boilerplate code and further decoupling the UI from the business logic.

2.4.2 Model-View-Update (MVU)


Model
(Holds the application's state)
State changes re-render the view
View
(Renders UI based on the model state)
User interactions trigger messages
Update
(Processes Messages, Updates the Model State)
New state based on updates

The MVU pattern is gaining traction for its simplicity and the way it revolutionizes state management in applications. .NET MAUI introduces first-class support for MVU through the Community Toolkit, empowering developers to build applications where the UI is a function of the state. This pattern encourages a unidirectional data flow, making state management predictable and debugging simpler.

State Management:

Centralizes the state of the app, ensuring UI consistency and simplifying state updates.

Component Reusability:

Enhances the ability to reuse components across different parts of the application, promoting code reusability and consistency.

2.5 Embracing Modern Development Practices

.NET MAUI revolutionizes the cross-platform development landscape by unifying the development experience and championing modern practices. Its support for design patterns like MVVM and MVU allows developers to tailor their approach to fit project requirements, enhancing code quality in terms of organization, maintainability, and testability. Additionally, the introduction of the UI Handler architecture showcases .NET MAUI's dedication to modernization, paving the way for applications that are both high-performing and customizable.

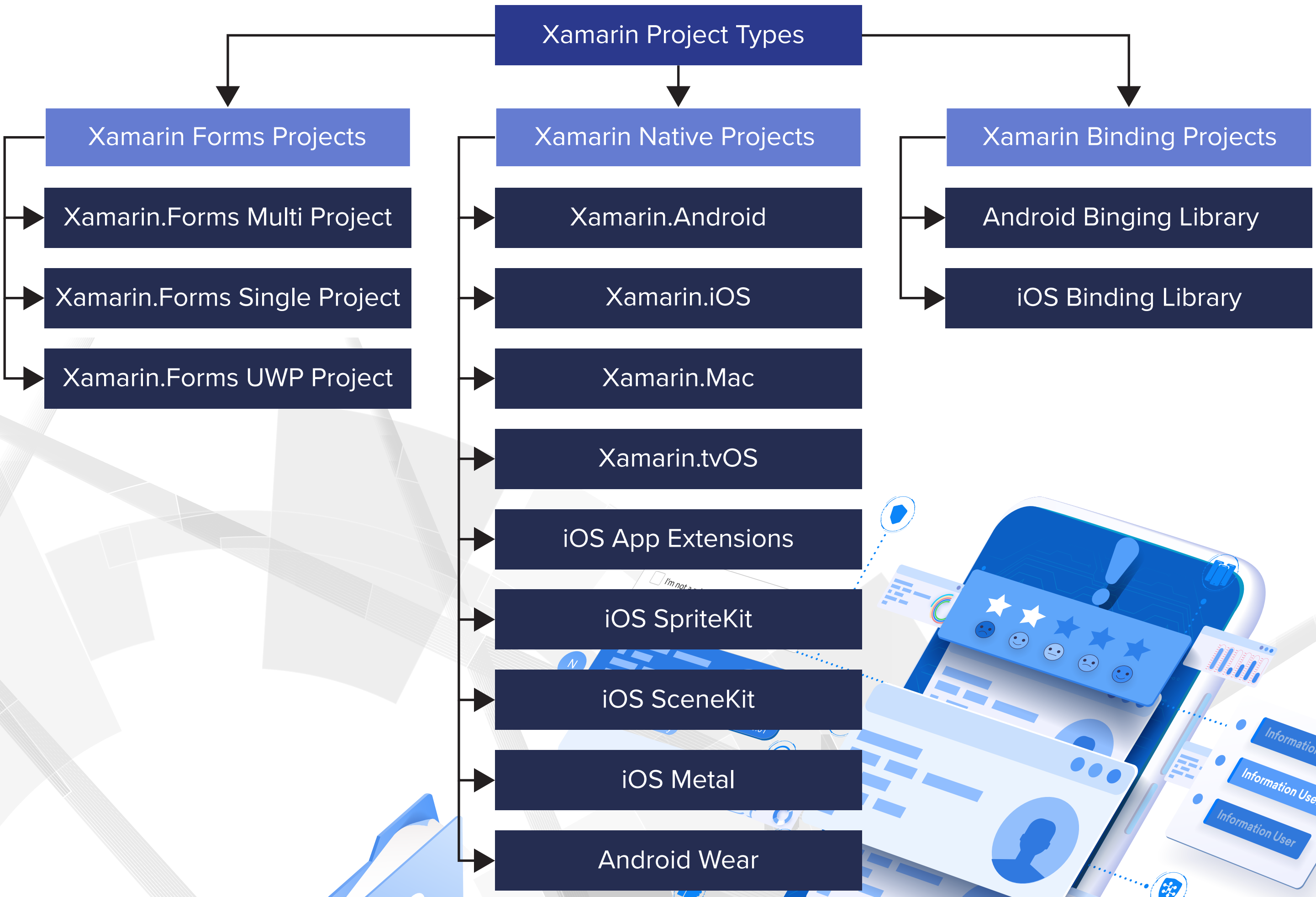
This overview lays a foundational understanding of .NET MAUI's core features, emphasizing its adaptability and the significant performance benefits it offers. As we consider the transition from Xamarin to .NET MAUI, it's crucial to grasp these design patterns and architectural advancements. They underscore .NET MAUI's role in shaping the future of cross-platform application development.

Next, we will delve into the migration process from Xamarin to .NET MAUI. This section aims to guide developers through leveraging .NET MAUI's robust capabilities to achieve a seamless upgrade, marking a step towards adopting a more flexible, powerful, and forward-looking framework.

3 Guide for Migrating Your Apps to .NET MAUI

3.1 Introduction to Migrating with .NET MAUI

Embarking on the journey to .NET MAUI represents a pivotal shift towards streamlined and innovative app development across multiple platforms. This guide aims to illuminate the path for seamlessly transitioning your applications, leveraging .NET MAUI's robust capabilities for a modern, unified development experience. The Xamarin project types that can be considered for upgrading to .NET MAUI are:

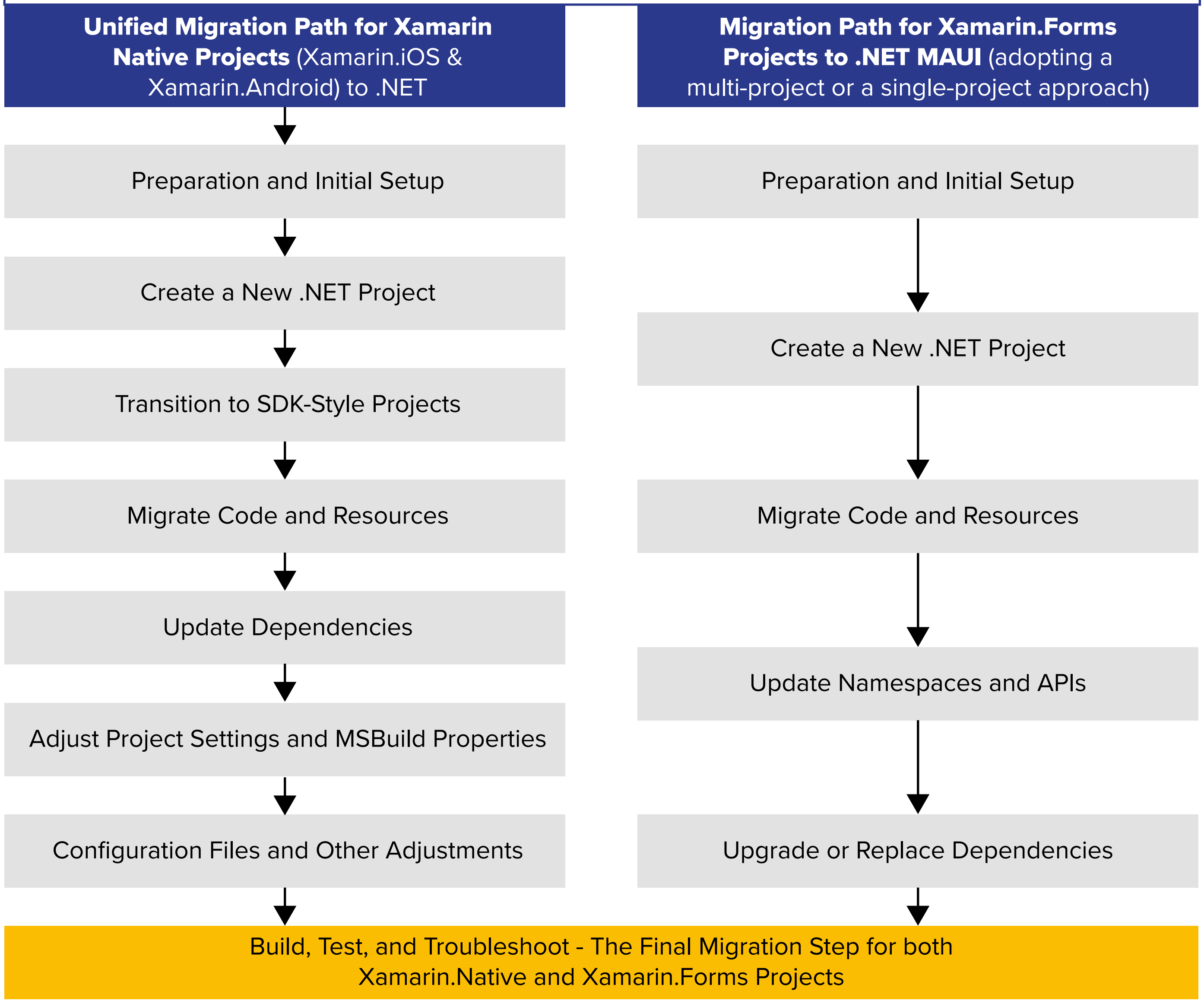


3.2 Migration Prerequisites

Following is the pre-migration checklist/prerequisites, which must be considered before the transition from Xamarin to .NET MAUI application.

Prerequisites	
Android version	5.0 (API 21) or higher
IOS version	10 or higher
Visual Studio version	2022 v17.6 or later
.NET version	.NET 8 or later
Xamarin.Forms	Xamarin.Forms 5.0 or higher

In the upcoming discussion, we will primarily focus on,


Unified Migration Path for Xamarin Native Projects (Xamarin.iOS & Xamarin.Android) to .NET
Preparation and Initial Setup
Create a New .NET Project
Transition to SDK-Style Projects
Migrate Code and Resources
Update Dependencies
Adjust Project Settings and MSBuild Properties
Configuration Files and Other Adjustments
Migration Path for Xamarin.Forms Projects to .NET MAUI (adopting a multi-project or a single-project approach)
Preparation and Initial Setup
Create a New .NET Project
Migrate Code and Resources
Update Namespaces and APIs
Upgrade or Replace Dependencies
Build, Test, and Troubleshoot - The Final Migration Step for both Xamarin.Native and Xamarin.Forms Projects

A Note on .NET Upgrade Assistant:

It's important to note that while the .NET Upgrade Assistant significantly streamlines this process, it may not cover the entire upgrade path, necessitating manual interventions for a complete migration. Each approach, from automated assistance to detailed manual steps, is designed to guide developers through the nuanced journey of transitioning to .NET MAUI, ensuring a thorough understanding of the process and its intricacies.

3.3 Unified Migration Path for Xamarin Native Projects (Xamarin.iOS & Xamarin.Android) to .NET

This segment will cover the consolidated common steps for migrating both Xamarin.iOS and Xamarin.Android projects to the .NET ecosystem:

1 Preparation and Initial Setup

Check Microsoft's official .NET compatibility and Xamarin migration guides for any known issues or considerations.

Create a branch of your existing project in your version control system or make a full copy of the project directory for safety.

2 Create a New .NET Project

In Visual Studio, create a new .NET project, selecting a template that closely matches your app's platform (iOS or Android) and architecture. Ensure it targets .NET 8 (net8.0-ios or net8.0-android) and name it to match your existing Xamarin project for ease of transition.

3 Transition to SDK-Style Projects

Confirm your new project uses an SDK-style .csproj file, which simplifies project management and is automatically configured for new .NET projects.

4 Migrate Code and Resources

Manually transfer your source code, .xaml files, images, fonts, storyboards (iOS), XML layouts (Android), and other resources to the new project, maintaining a similar structure for ease of management.

5 Update Dependencies

Use Visual Studio's NuGet package manager to find and update each dependency to its .NET or .NET Standard version. For packages without direct .NET equivalents, look for alternatives or pre-release versions targeting .NET.

6 Adjust Project Settings and MSBuild Properties

Set up the necessary platform-specific configurations in your new .csproj file, including runtime identifiers, MSBuild properties adaptation, and code signing (iOS).

7 Configuration Files and Other Adjustments

iOS: Migrate key-value pairs from your Xamarin.iOS Info.plist to the new project's Info.plist. Copy app entitlements and ensure they're correctly associated with project configurations.

Android: Adjust configuration settings from AndroidManifest.xml to the .NET project file where applicable. Address runtime behavior differences and adjust linker settings to optimize app size and performance.

3.4 Migration Path for Xamarin.Forms Projects to .NET MAUI

(adopting a multi-project or a single-project approach)

This segment will cover the process of migrating Xamarin.Forms project to .NET MAUI, considering both the multi-project and single-project approaches:

1 Preparation and Initial Setup (Update Xamarin.Forms App)

Ensure your app is on Xamarin.Forms version 5.0 for optimal compatibility.

Upgrade all dependencies to their latest versions to ensure .NET 8 compatibility.

Thoroughly test your app to verify that updates do not introduce regressions.

2 Create a New .NET MAUI Project

In Visual Studio, initiate a new .NET MAUI project. Use the same name as your Xamarin.Forms app to maintain consistency.

For a multi-project setup, ensure the project is configured for multi-targeting to cover all desired platforms (Android, iOS, macOS, and Windows).

For a single-project setup, the .NET MAUI single-project approach naturally targets multiple platforms under one project.

3 Migration of Code and Resources

Move shared logic, XAML files, and resources from your Xamarin.Forms app to the .NET MAUI project, keeping the folder structure intact.

Migrate custom renderers to .NET MAUI handlers or use built-in handlers as replacements.

Adjust platform-specific code by relocating it to the respective “Platforms” directories within the .NET MAUI project for a multi-project setup. In a single-project setup, this involves integrating platform-specific code within conditional compilation blocks as necessary.

4 Update Namespaces and APIs

Replace Xamarin.Forms namespaces with Microsoft.Maui and Microsoft.Maui.Controls in your migrated code.

Address any API changes between Xamarin.Forms and .NET MAUI, consulting the official documentation for guidance on replacements and new patterns.

5 Upgrade or Replace Dependencies

Confirm the compatibility of all external libraries or NuGet packages with .NET 8. Replace or update incompatible dependencies.

3.5 Build, Test, and Troubleshoot

The Final Migration Step for both Xamarin.Native and Xamarin.Forms Projects

This final segment emphasizes the importance of validating functionality, optimizing performance, and resolving any emerging issues to guarantee a successful transition to .NET MAUI:

Build the .NET MAUI project to identify compile-time errors. Use Visual Studio's error messages to guide necessary code adjustments.

Extensively test the app on all target platforms to ensure functionality, performance, and appearance meet expectations.

Use debugging tools to identify and fix runtime issues.

Utilize community forums, .NET MAUI documentation, and GitHub for troubleshooting.

3.6 Key Considerations for Migration

API and Namespace Changes

Both migration paths require attention to .NET API changes and updates in namespaces. Minor adjustments may be needed to ensure compatibility with the .NET ecosystem, whether moving to .NET MAUI or .NET for Xamarin Native projects.

.NET Standard and Library Compatibility

Libraries targeting .NET Standard are generally compatible with .NET 8, simplifying the migration process for both Xamarin.Forms and Xamarin Native projects. This eases the update of dependencies and integration of external libraries into the new .NET project.

Preview Features and Recompilation

Using preview versions of libraries or recompiling them against .NET 8 target frameworks may be necessary for both migrations, especially for libraries that haven't been updated to support .NET 8 officially.

Performance Optimization

While more emphasized in the transition to .NET MAUI due to its specific performance features, optimizing application performance is a universal consideration. This includes leveraging new .NET performance improvements and best practices.

UI and UX Enhancements (Primarily for .NET MAUI)

Migrating to .NET MAUI offers an opportunity to enhance the app's UI and UX through advanced components and layout systems. While this is specific to .NET MAUI, the underlying principle of using migration as an opportunity to refine the app applies broadly.

Lifecycle Management (Primarily for .NET MAUI)

Understanding changes in app lifecycle management is crucial for .NET MAUI migration. For Xamarin Native to .NET migrations, understanding .NET's lifecycle events and handling is also beneficial, even though the context may differ.

Optimize Your Code to Take Advantage of .NET 8 Features and Improvements

As you migrate, it's an opportune time to revise and enhance your codebase to leverage the new features and performance improvements available in .NET 8, ensuring your application benefits from the latest advancements in the .NET ecosystem.

3.7 Additional Resources

Microsoft Documentation
Essential for all developers, offering comprehensive guides, API references, and best practices for .NET development and migration.

.NET Upgrade Assistant
Useful for automating aspects of the migration process. Its applicability can vary, but it's a valuable tool for simplifying migrations to .NET MAUI and potentially for some Xamarin Native to .NET migrations.

Community Support
Engaging with the broader .NET, Xamarin, and .NET MAUI communities through forums, GitHub, and social media is invaluable for accessing shared knowledge, troubleshooting tips, and migration advice.

A Note on Community and Support

The .NET MAUI ecosystem is supported by a vibrant community and robust backing from Microsoft, creating a dynamic environment for both novice and seasoned developers to learn, create, and collaborate on cross-platform applications. Here's an organized overview of the crucial resources and support channels, guiding you from learning materials through community engagement, and ultimately, to official support avenues.

Learning Resources

.NET MAUI on Microsoft Learn:

Dive into a comprehensive array of tutorials, detailed guides, and reference materials on Microsoft's official learning platform, perfect for developers of all skill levels. Access these resources at **Microsoft Learn - .NET MAUI**.

Dev Blogs:

Keep up with the latest .NET MAUI insights, tips, and news directly from the developers at Microsoft Developer Blogs. Explore at **DevBlogs - Microsoft**.

Community Engagement

.NET MAUI Community Toolkit (GitHub):

This hub features a collection of invaluable extensions and controls created by the community, fostering the sharing of tools that enhance .NET MAUI development.
Visit
GitHub - CommunityToolkit/Maui.

.NET MAUI Community Stand-ups (YouTube):

Stay informed with the latest developments and interact with the .NET MAUI team during their live-streamed sessions on the **.NET Foundation YouTube channel.**

Forums and Social Media:

Engage with the .NET MAUI community on platforms like Stack Overflow (using the ".net-maui" tag), Reddit (r/dotnetMAUI subreddit), and Twitter (follow hashtags like #dotnetMAUI) for a rich array of discussions, advice, and insights.

Official Support

.NET MAUI Documentation:

The definitive source for all .NET MAUI technical documentation, offering exhaustive guides and how-tos. Visit
Microsoft Docs - .NET MAUI.

GitHub Issues:

For bug reports and feature requests, the .NET MAUI GitHub issues page is the designated platform. Report and track issues at
GitHub - dotnet/maui/issues.

Microsoft Support for .NET MAUI:

Gain access to official support for .NET MAUI through Microsoft's support policy, ensuring reliable assistance for your development needs. Learn more at **Microsoft Support Policy for .NET MAUI** and the comprehensive **.NET Support Policy.**

5

MAUI's Future Outlook

Advancing Cross-Platform Excellence

.NET MAUI's integration of Xamarin.Forms with the .NET ecosystem, offering a unified codebase for Android, iOS, Windows, macOS, and potentially web platforms, sets a strong foundation for its future in cross-platform development. Leveraged by the extensive .NET libraries and Visual Studio, it promises streamlined development, high native performance, and robust Microsoft support. Coupled with a vibrant developer community and alignment with technological trends, including web development potential via Blazor, .NET MAUI is positioned to lead in creating modern, versatile applications. Its comprehensive platform coverage and focus on performance and innovation make it a pivotal tool for developers navigating the evolving digital landscape.

Criteria	Description
Development Language	C# and XAML, offering a robust, type-safe language with rich development features, leveraging the extensive .NET ecosystem.
Platform Coverage	Comprehensive, including iOS, Android, Windows, and Mac, with plans to expand to the Web via Blazor integration, offering a truly unified approach.
Native Performance	High, as it compiles to native code, ensuring optimal performance and access to native APIs and features across all platforms.
UI Consistency & Flexibility	High, as it directly uses native UI controls, ensuring consistency and flexibility in design across platforms.
Development Ecosystem	Rich, as it leverages the entire .NET ecosystem, including NuGet packages, and Visual Studio, providing a comprehensive toolset for developers.
Community & Support	Strong, as it is backed by Microsoft, ensuring long-term support, stability, and a thriving community for resources and collaboration.
Future-Proof	High, with Microsoft's commitment, suggests continuous improvements and additions, with a focus on integrating modern web technologies like Blazor for web development.

MAUI + Blazor Why it Can Be a Game Changer

The integration of .NET MAUI with Blazor marks a significant advancement in cross-platform development, offering a unified framework for building applications across iOS, Android, Windows, macOS, and the web. This powerful combination leverages the strength of .NET MAUI and the flexibility of web technologies through Blazor, simplifying the development process and expanding the potential for application projects.

Key Benefits

Unified Development

Utilize C# across all platforms, streamlining the development process and enabling greater innovation without the overhead of multiple languages.

Seamless Transition for Web Developers

Blazor's integration means web development skills directly translate to building native applications, lowering the barrier to entry and enhancing productivity.

Efficiency in Code Reuse

The capability to share business logic and UI components across platforms accelerates development and simplifies updates, contributing to a more efficient workflow.

Comprehensive Tooling

Visual Studio enhances developer efficiency with robust features for building, debugging, and testing, encapsulating all necessary tools in one environment.

Performance and Reliability

.NET MAUI's native performance, bolstered by Blazor and WebAssembly, ensures applications are fast and responsive across all platforms.

Community and Microsoft Support

Backed by Microsoft and an active community, .NET MAUI and Blazor benefit from continuous improvements and a rich pool of resources.

Business and Branding Advantage

This combination facilitates the creation of modern, consistent user experiences, strengthening brand identity and user satisfaction across various platforms.

Embracing .NET MAUI with Blazor doesn't just represent a step forward in technology but a comprehensive shift towards more accessible, efficient, and versatile application development. This fusion positions developers and businesses to effectively address today's digital demands while being well-prepared for future advancements.

7 Conclusion

In conclusion, transitioning from Xamarin to .NET MAUI represents a significant leap forward in cross-platform development, unifying and enhancing the framework for building applications across mobile, desktop, and beyond. This shift not only preserves the value of existing Xamarin investments but also smoothly integrates Xamarin Forms and Native projects into the .NET ecosystem's future.

The migration to .NET MAUI, while necessitating adaptation to its architectural changes—such as moving from custom renderers to handlers—offers considerable advantages. Developers can expect faster application startups, improved performance, and broader platform compatibility, establishing .NET MAUI as a cornerstone for future projects.

.NET MAUI provides developers with powerful tools and capabilities to create highly performant, functional, and maintainable applications. With the backing of the .NET community and the assistance of the .NET Upgrade Assistant, transitioning to .NET MAUI is an insightful step to future-proof your applications and fully leverage the .NET platform's capabilities.

References 8

https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-8.0
https://learn.microsoft.com/en-us/dotnet/maui/migration/?view=net-maui-8.0
https://learn.microsoft.com/en-us/dotnet/architecture/maui/

About the Author

Hariprasad C.R. Rao is a Senior Engineering Manager at Happiest Minds Technologies, with around 18 years of experience specializing in digital solutions strategies, architecture evaluation, and technology assessment for mobile platforms. His expertise spans designing, architecting, and developing robust applications and games for macOS, Android, and iOS platforms, leveraging both native and cross-platform development tools, with a firm grasp on the application development lifecycle.

Hari has a track record of leading projects that deliver scalable, resilient mobile applications across diverse industries such as Education, Media, IoT, Healthcare, Industrial, and Automotive. At Happiest Minds, for nearly 8 years, he has excelled in transforming functional requirements into comprehensive mobile solutions, guiding technical teams, and ensuring high-quality deliverables that enhance competitive positions. His commitment to quality and innovation is evident in his hands-on approach to product management, development, and the enforcement of sound development practices.