Helion

packt

Projektowanie aplikacji w .NET MAUI

Jak budować doskonałe interfejsy użytkownika dla aplikacji wieloplatformowych

ROGER YE

Tytuł oryginału: .NET MAUI Cross-Platform Application Development: Leverage a first-class cross-platform UI framework to build native apps on multiple platforms

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-289-0294-7

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/prapne
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/prapne.zip

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

CZĘŚĆ 2. Stosowanie .NET MAUI Blazor

Wprowadzenie do .NET MAUI

Rozdział 1

Od czasu wprowadzenia .NET 5 Microsoft stara się zunifikować różne implementacje .NET w jedną wersję. .NET Multi-platform App UI (czyli .NET MAUI) to efekt próby udostępnienia ujednoliconego wieloplatformowego frameworka do tworzenia interfejsów użytkownika. Z tej książki dowiesz się, jak używać .NET MAUI do tworzenia aplikacji wieloplatformowych.

W tym rozdziale znajdziesz następujące informacje:

- przegląd technologii wieloplatformowych,
- porównanie technologii wieloplatformowych (.NET, Javy i JavaScriptu),
- środowisko .NET i historia platformy Xamarin,
- funkcje .NET MAUI,
- aplikacje .NET MAUI Blazor,
- konfigurowanie środowiska programistycznego.

Jeśli dopiero zaczynasz tworzenie aplikacji .NET, ten rozdział pomoże Ci zrozumieć środowisko frameworka .NET. Dla programistów znających platformę Xamarin wiele zagadnień omawianych w tej książce może wydawać się znajomych. W tym rozdziale znajdziesz przegląd tematów opisywanych w tej pozycji.

Przegląd technologii wieloplatformowych

Przed przejściem do technologii wieloplatformowych warto najpierw przyjrzeć się środowisku rozwoju aplikacji, aby lepiej zrozumieć różne technologie wieloplatformowe.

.NET MAUI jest rozwijanym przez Microsoft wieloplatformowym frameworkiem do tworzenia aplikacji na urządzenia mobilne i stacjonarne z systemami Android, iOS, Windows i Tizen.

Na ogólnym poziomie rozwój oprogramowania można podzielić na dwie kategorie — programowanie systemów i programowanie aplikacji. Celem programowania aplikacji jest tworzenie oprogramowania, które bezpośrednio wykonuje usługi dla użytkownika. Z kolei celem programowania systemów jest rozwój oprogramowania i platform zapewniających usługi dla innego oprogramowania. W świecie .NET rozwój samej platformy .NET to przykład programowania systemów, natomiast tworzenie aplikacji opartych na platformie .NET to przykład programowania aplikacji.

Projekt lub architektura współczesnych systemów obejmuje część kliencką i część serwerową, które można też nazwać frontendem i backendem.

Oprogramowanie działające po stronie klienta można dodatkowo podzielić na dwie kategorie — aplikacje natywne i aplikacje internetowe.

Aplikacje natywne

Tworzenie aplikacji natywnych zwykle odbywa się z myślą o konkretnym systemie operacyjnym. Jeśli chodzi o aplikacje desktopowe, mogą to być aplikacje dla systemu Windows, macOS lub Linux. W obszarze aplikacji mobilnych można wyróżnić aplikacje dla systemów Android lub iOS.

W trakcie tworzenia aplikacji natywnej trzeba przygotować jej wersję dla każdej platformy (Windows, Linux, Android lub macOS czy iOS). Potrzebne są inne języki programowania, narzędzia i biblioteki, aby opracować każdą wersję z osobna.

Aplikacje internetowe

W ostatnich kilku dekadach dziedzina rozwoju aplikacji internetowych przeszła ewolucję obejmującą kilka generacji — od statycznych stron internetowych w przeglądarce Netscape po nowoczesne **aplikacje jednostronicowe** oparte na frameworkach javascriptowych (takich jak React lub Angular). W obszarze rozwoju aplikacji internetowych dominującą pozycję na rynku mają JavaScript i różne frameworki związane z tym językiem. W ekosystemie .NET odrobić dystans próbują twórcy technologii Blazor.

Usługi backendowe

Zarówno aplikacje natywne, jak i aplikacje internetowe zwykle potrzebują usług backendowych dających dostęp do logiki biznesowej lub bazy danych. Do rozwoju usług backendowych można używać wielu języków i frameworków, na przykład Java/Spring, .NET, Node.js, Ruby on Rails czy Python/Django. Aplikacje natywne i aplikacje internetowe zwykle mogą używać tej samej usługi backendowej. Do rozwoju usług backendowych najczęściej stosowane są Java i .NET.

Technologie wieloplatformowe

Technologie używane do rozwoju aplikacji internetowych i usług backendowych nie są zależne od platformy. Można z nich korzystać w ich pierwotnej postaci w różnych platformach. Gdy ktoś mówi o rozwoju aplikacji wieloplatformowych, zwykle ma na myśli tworzenie aplikacji natywnych. W tym kontekście technologie do rozwoju aplikacji wieloplatformowych mogą pomóc w obniżeniu kosztów i poprawie wydajności. Najczęściej używane technologie do rozwoju aplikacji wieloplatformowych to Flutter, .NET MAUI/Xamarin i React Native. W tabeli 1.1 znajdziesz przegląd dostępnych technologii wieloplatformowych i analogicznych rozwiązań Microsoftu. Przedstawiona tu lista nie jest kompletna. Chciałem jedynie dać wyobrażenie, jakiego rodzaju technologie są dostępne w każdej z wymienionych kategorii i jakich rozwiązań Microsoftu można użyć w zamian.

Tabela 1.1. Zestawienie języków i frameworków z rozwiązaniami Microsoftu

Kategoria	Technologie wieloplatformowe		Rozwiązania Microsoftu
	Język	Framework	
Aplikacje internetowe	JavaScript	React, Angular, Vue	Strony Blazor/Razor
Aplikacje natywne	JavaScript	Flutter	.NET MAUI/ Blazor/Xamarin
	Dart	Swing/Codename One	
	Java/Kotlin	Spring	
Usługi backendowe	Java	Spring	ASP.NET Core
	JavaScript	Node.js	
	Python	Django/Flask/Tornado	

Nie istnieje jedno najlepsze narzędzie wieloplatformowe lub jeden najlepszy framework tego rodzaju. Wybór zwykle zależy od wymagań biznesowych. Jednak z przedstawionej tabeli wynika, że ekosystem .NET udostępnia kompletny zestaw narzędzi. Zespół programistyczny rozwijający duży system zwykle musi obejmować osoby mające doświadczenie z różnymi językami programowania i frameworkami. Dzięki .NET złożoność zestawu języków programowania i frameworków można radykalnie zmniejszyć.

Porównanie .NET, Javy i JavaScriptu

Przedstawiłem przegląd narzędzi i frameworków używanych do tworzenia aplikacji internetowych, aplikacji natywnych i usług backendowych. W bardziej ogólnym ujęciu, czyli na poziomie ekosystemu .NET, ekosystem Javy lub JavaScriptu zapewnia prawie te same możliwości co .NET. Rozwiązania związane z Javą, JavaScriptem i .NET zapewniają narzędzia lub frameworki dla prawie wszystkich możliwych warstw. Ciekawe byłoby porównanie Javy, JavaScriptu i .NET na ogólnym poziomie.

Język Java jest rozwijany zgodnie z hasłem *napisz raz i uruchamiaj wszędzie*. Ten ekosystem jest oparty na języku programowania Java i **maszynie wirtualnej Javy** (ang. *Java Virtual Machine* — **JVM**). JVM to mechanizm uruchamiany w obsługiwanych platformach, który pomaga programistom wyeliminować zależność od platformy. Dzięki tej cesze Java stała się standardowym wyborem do tworzenia aplikacji i usług wieloplatformowych.

JavaScript to język opracowany dla przeglądarek internetowych, a jego możliwości są bardzo rozbudowane, co wynika z wymagań związanych z tworzeniem rozwiązań internetowych. Ograniczeniem JavaScriptu jest to, że jest to język skryptowy, dlatego nie udostępnia mechanizmów dostępnych w Javie lub C#. Jednak nie zmniejsza to liczby użytkowników i popularności JavaScriptu. W tabeli 1.2 znajdziesz porównanie trzech omawianych technologii.

W tabeli 1.2 widać, że zarówno .NET, jak i Java mają infrastrukturę odpowiednią do obsługi wielu języków. JavaScript ma ograniczenia, ponieważ jest językiem skryptowym, dlatego wymyślono TypeScript i CoffeeScript, aby wzbogacić jego możliwości.

Tabela 1.2. Porównanie Javy, JavaScriptu i .NET

Porównywany obszar	.NET	Java	JavaScript
Języki programowania	C#, F#, VB, C++, PHP, Ruby, Python i inne	Java, Kotlin, Clojure, Groovy, Scala i inne	JavaScript, TypeScript, CoffeeScript i inne
Środowisko uruchomieniowe	CLR	JVM	V8, SpiderMonkey, JavaScriptCore
Obsługiwane IDE	Microsoft Visual Studio, Rider, MonoDevelop i Visual Studio Code	Eclipse, IntelliJ Idea, Oracle NetBeans i Oracle JDeveloper	Visual Studio Code, Webstorm i Atom
Framework do tworzenia frontendów	ASP.NET Core Razor/Blazor	Możliwe tylko generowanie z poziomu serwera, na przykład za pomocą technologii JSP lub Thymeleaf	React, Angular, Vue
Aplikacje desktopowe	WinForms, Win UI, WPF, UWP i inne	Swing, JavaFX i inne	Electron, NW.js i inne
Aplikacje mobilne	.NET MAUI/Xamarin	Codename One	React Native, Cordova, Iconic i inne
Framework do tworzenia backendów	ASP.NET Core	Spring	Node.js

TypeScript został opracowany przez Microsoft, aby dodać do JavaScriptu nowoczesne, obiektowe mechanizmy języka. TypeScript jest na potrzeby wykonywania kompilowany do JavaScriptu, dlatego dobrze współdziała z istniejącymi bibliotekami tego języka.

Java jest oparta na maszynie JVM, natomiast .NET bazuje na środowisku **CLR** (ang. *Common Language Runtime*) i systemie typów **CTS** (ang. *Common Type System*). Ponieważ CTS i CLR są podstawą implementacji .NET, ten framework w naturalny sposób obsługuje wiele języków. Wszystkie one współdzielą bibliotekę klas **BCL** (ang. *Base Class Library*).

Choć istnieje wiele języków, które używają maszyny JVM jako warstwy abstrakcyjnej, aby zapewnić wieloplatformowość, współdziałanie między językami z rodziny Javy nie odbywa się na tym samym poziomie co między językami frameworka .NET. Wszystkie języki w .NET są zbudowane z użyciem jednej architektury i korzystają z tej samej biblioteki *BCL*, natomiast języki z rodziny Javy, na przykład Java, Kotlin lub Scala, są rozwijane niezależnie i służą do zupełnie innych celów.

To porównanie pomaga wybrać lub ocenić zestaw technologii pod kątem rozwoju aplikacji wieloplatformowych. Dla programistów korzystających z .NET MAUI te analizy mogą

pomóc lepiej zrozumieć wybrane narzędzia. Aby określić miejsce .NET MAUI w ekosystemie .NET, w następnym podrozdziale zapoznasz się z krótką historią .NET.

Przegląd środowiska .NET

Przed przejściem do szczegółowego omówienia .NET MAUI zapoznaj się ze środowiskiem .NET. Ten podrozdział jest istotny tylko dla osób dopiero poznających .NET. Jeśli jesteś programistą korzystającym już z tej technologii, możesz pominąć ten fragment.

Od czasu wprowadzenia platformy .NET przez Microsoft przeszła ona ewolucję od zamkniętego frameworka do tworzenia oprogramowania dla systemu Windows do narzędzia wieloplatformowego i otwartoźródłowego.

Na zestaw technologii .NET można patrzeć na wiele sposobów. Na podstawowym poziomie obejmuje on następujące komponenty:

- wspólną infrastrukturę (kompilator i zestaw narzędzi),
- biblioteki *BCL*,
- środowisko uruchomieniowe — **Windows Runtime** (**WinRT**) lub Mono.

.NET Framework

Historia .NET rozpoczyna się od .NET Framework. Jest to opracowany przez Microsoft zamknięty framework do tworzenia oprogramowania działający przede wszystkim w systemie Microsoft Windows. .NET Framework był początkowo przyszłościowym frameworkiem do budowania aplikacji, który miał pozwolić ustandaryzować oprogramowanie w ekosystemie Windowsa. Jest oparty na środowisku **CLI** (ang. *Common Language Infrastructure*) i języku C#. Choć podstawowym językiem programowania jest tu C#, .NET Framework został zaprojektowany jako niezależny od języka. Obsługiwane języki mogą korzystać z tego samego systemu typów CTS i środowiska uruchomieniowego CLR. Większość desktopowych aplikacji dla systemu Windows jest rozwijanych za pomocą .NET Framework, a sam .NET Framework jest udostępniany jako część systemu Windows.

Mono

Pierwsza próba przekształcenia .NET we framework otwartoźródłowy została podjęta przez firmę Ximian. Gdy CLI i C# zostały zatwierdzone przez organizację ECMA w 2001 roku i organizację ISO w 2003 roku, pojawiła się możliwość opracowania niezależnych implementacji.

W 2001 roku uruchomiono otwartoźródłowy projekt Mono, którego celem było zaimplementowanie .NET Framework w Linuksie.

Ponieważ wówczas .NET Framework był technologią zamkniętą, dla .NET Framework i Mono używane były różne kompilatory, biblioteki *BCL* i środowiska uruchomieniowe.

Z czasem Microsoft zaczął podążać w stronę oprogramowania otwartoźródłowego, a kod źródłowy .NET stał się otwarty. W projekcie Mono wykorzystano część kodu źródłowego i niektóre narzędzia z kodu bazowego .NET.

Jednocześnie także w projekcie Mono zachodziło wiele zmian. W czasie, gdy projekt Mono należał do firmy Xamarin, firma ta opracowała opartą na Mono platformę Xamarin, by umożliwić obsługę platformy .NET w systemach Android, iOS, **UWP** (ang. *Universal Windows Platform*) i macOS. W 2016 roku Microsoft przejął platformę Xamarin, a ta stała się wieloplatformowym narzędziem w ekosystemie .NET.

.NET Core

Przed przejęciem platformy Xamarin Microsoft rozpoczął już prace nad przekształceniem .NET we framework wieloplatformowy. Pierwszą próbą było udostępnienie .NET Core w 2016 roku. .NET Core to bezpłatny i otwartoźródłowy framework dostępny w systemach Windows, Linux i macOS. Można go używać do tworzenia nowoczesnych aplikacji internetowych, mikrousług, bibliotek i aplikacji konsolowych. Ponieważ aplikacje .NET Core mogą działać w Linuksie, możliwe jest budowanie mikrousług z wykorzystaniem kontenerów i infrastruktury chmurowej.

Po udostępnieniu .NET Core 3.x Microsoft zaczął pracę nad integracją i unifikacją technologii .NET na różnych platformach. Zunifikowana wersja miała zastąpić zarówno .NET Core, jak i .NET Framework. Aby uniknąć mylenia zunifikowanego frameworka z .NET Framework 4.x, nazwano go .NET 5. Od .NET 5. wspólna biblioteka *BCL* może być używana we wszystkich platformach. W .NET 5 nadal używane są dwa środowiska uruchomieniowe — **WinRT** (dla systemu Windows) i Mono (dla urządzeń mobilnych i systemu macOS).

W tej książce używana jest wersja .NET 6.

.NET Standard i przenośne biblioteki klas

Przed pojawieniem się .NET 5, gdy istniały .NET Framework, Mono i .NET Core, w różnych platformach używane były inne podzbiory bibliotek *BCL*. Aby móc współdzielić kod między różnymi środowiskami uruchomieniowymi lub platformami, stosowano **przenośne biblioteki klas** (ang. *Portable Class Libraries* — **PCL**). Gdy tworzysz przenośną bibliotekę klas, musisz wybrać kombinację platform, których obsługę chcesz zapewnić. To programiści decydują, w jakim stopniu kod będzie kompatybilny z różnymi platformami. Jeśli chcesz ponownie użyć jakiejś przenośnej biblioteki klas, musisz starannie przeanalizować listę platform, które mają być obsługiwane.

Choć przenośne biblioteki klas umożliwiają współużytkowanie kodu, nie rozwiązują w zgrabny sposób problemów z kompatybilnością. Aby poradzić sobie z tymi problemami, Microsoft wprowadził .NET Standard.

.NET Standard jest nie tyle odrębną wersją .NET, co specyfikacją zestawu API .NET, które mają być obsługiwane w większości implementacji .NET (.NET Framework, Mono, .NET Core, .NET 5, .NET 6 itd.).

Od .NET 5 dostępna jest zunifikowana biblioteka *BCL*, jednak .NET Standard nadal pozostanie jej częścią. Jeśli Twoja aplikacja ma obsługiwać tylko .NET 5 i nowsze wersje, .NET Standard nie ma dla Ciebie znaczenia. Jeżeli jednak chcesz zapewnić zgodność aplikacji ze starszymi wersjami .NET, najlepiej jest korzystać z .NET Standard. W tej książce użyję .NET Standard 2.0 do zbudowania modelu danych, ponieważ ta wersja jest zgodna z większością istniejących implementacji .NET i wszystkimi przyszłymi edycjami tego frameworka.

Microsoft nie będzie tworzyć nowych wersji .NET Standard, jednak .NET 5, .NET 6 i wszystkie przyszłe wersje .NET będą zgodne z .NET Standard 2.1 i wcześniejszymi edycjami. W tabeli 1.3 wymienione są platformy i wersje zgodne z .NET Standard 2.0. Rozwiązania wymienione na liście są też kompatybilne z modelami danych z tej książki.

Tabela 1.3. Implementacja kompatybilne z .NET Standard 2.0

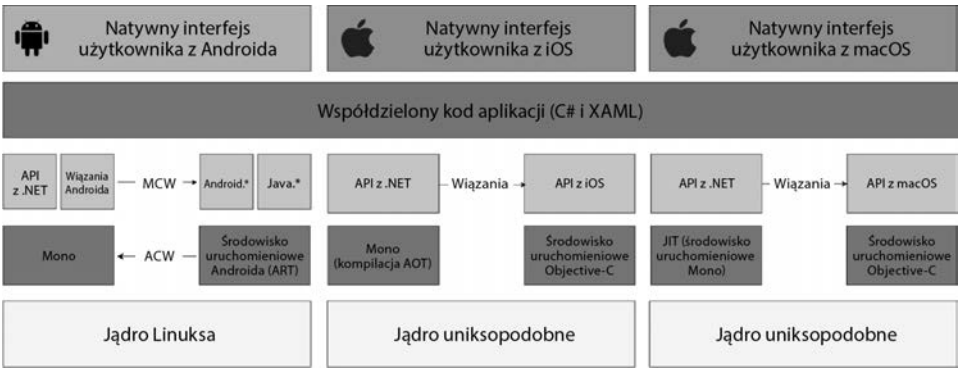
Implementacja .NET	Obsługiwane wersje
.NET i .NET Core	2.0, 2.1, 2.2, 3.0, 3.1, 5.0 i 6.0
.NET Framework 1	4.6.1 2, 4.6.2, 4.7, 4.7.1, 4.7.2 i 4.8
Mono	5.4 i 6.4
Xamarin.iOS	10.14 i 12.16
Xamarin.Mac	3.8 i 5.16
Xamarin.Android	8.0 i 10.0
UWP	10.0.16299, do ustalenia
Unity	2018.1

Otwartoźródłowy projekt `KPCLib` to biblioteka .NET Standard 2.0, które używam w rozwijanej aplikacji. W tabeli 1.3 widać, że biblioteki .NET Standard mogą być używane zarówno w aplikacjach opartych na Xamarinie, jak i aplikacjach .NET MAUI.

Używanie Xamarina do tworzenia aplikacji mobilnych

We wcześniejszym punkcie wspomniałem, że Xamarin był częścią projektu Mono i miał zapewnić obsługę .NET w systemach Android, iOS i macOS. Xamarin.Forms to wieloplatformowy framework do tworzenia interfejsów użytkownika rozwijany przez firmę Xamarin. .NET MAUI powstał w wyniku ewolucji Xamarin.Forms. Przed omówieniem .NET MAUI i Xamarin.Forms warto przyjrzeć się rysunkowi przedstawiającemu implementacje Xamarina z różnych platform (zobacz rysunek 1.1).

Na rysunku 1.1 pokazana jest ogólna architektura Xamarina. Za pomocą tego frameworka programiści mogą tworzyć natywne interfejsy użytkownika na wszystkich platformach i pisać w C# logikę biznesową, z której można korzystać w różnych platformach.



Rysunek 1.1. Implementacje Xamarina

W obsługiwanych platformach Xamarin udostępnia wiązania dla prawie całych pakietów SDK. Xamarin udostępnia też mechanizmy do bezpośredniego korzystania z bibliotek języków Objective-C, Java, C i C++, co daje możliwość używania dużej ilości zewnętrznego kodu. Możesz stosować istniejące biblioteki dla systemów Android, iOS i macOS napisane w językach Objective-C, Swift, Java lub C czy C++.

W tych platformach środowiskiem uruchomieniowym dla .NET jest Mono. Może ono działać w dwóch trybach — **JIT** (ang. *Just-in-Time*) i **AOT** (ang. *Ahead-of-Time*). Kompilacja w trybie JIT powoduje dynamiczne generowanie kodu w czasie jego wykonywania. W trybie AOT Mono wstępnie kompiluje cały kod, dzięki czemu można go używać w systemach operacyjnych, w których dynamiczne generowanie kodu jest niemożliwe.

Na rysunku 1.1 widać, że tryb JIT może być stosowany w systemach Android i macOS, natomiast tryb AOT jest używany w systemie iOS, gdzie dynamiczne generowanie kodu jest niedozwolone.

Istnieją dwa sposoby tworzenia aplikacji natywnych za pomocą Xamarina.

Możesz je pisać tak jak robią to programiści aplikacji dla systemów Android, iOS i macOS, czyli za pomocą natywnych API z poszczególnych platform. Różnica dotyczy tego, że używasz wtedy bibliotek .NET i języka C# zamiast bezpośrednio korzystać z języków i bibliotek specyficznych dla tych systemów. Zaletą tego podejścia jest to, że możesz posługiwać się jednym językiem i używać wielu tych samych komponentów za pomocą biblioteki *.NET BCL*, nawet jeśli pracujesz z różnymi platformami. Ponadto możesz wykorzystać możliwości platform, jak robią to programiści aplikacji natywnych.

Jeśli chcesz ponownie wykorzystać kod z warstwy interfejsu użytkownika, możesz zastosować Xamarin.Forms zamiast natywnych interfejsów użytkownika.

Xamarin.Forms

Xamarin.Android, Xamarin.iOS i Xamarin.Mac udostępniają środowisko .NET, które zapewnia prawie wszystkie możliwości oryginalnego SDK w poszczególnych platformach. Na przykład programista używający Xamarin.Android ma niemal te same możliwości co przy korzystaniu z oryginalnego SDK Androida. Aby dodatkowo ułatwić współdzielenie

kodu, opracowano otwartoźródłowy framework do tworzenia interfejsów użytkownika, Xamarin.Forms. Obejmuje on kolekcję wieloplatformowych komponentów interfejsu użytkownika. Taki interfejs można zaprojektować za pomocą języka znaczników XAML, co przypomina projektowanie windowsowych interfejsów użytkownika w WinUI lub w WPF.

Xamarin.Essentials

Ponieważ Xamarin udostępnia możliwości bazowych pakietów SDK poszczególnych platform, możesz używać mechanizmów urządzeń za pomocą API .NET. Jednak implementacje tego API są specyficzne dla platform. Na przykład gdy używasz usług lokalizacyjnych w systemach Android lub iOS, API .NET mogą być różne. Aby dodatkowo ułatwić współdzielenie kodu między platformami, można używać Xamarin.Essentials do dostępu do natywnych mechanizmów urządzeń. Jeśli korzystasz z Xamarin.Essentials zamiast z natywnych API, kod można ponownie wykorzystać w różnych platformach.

Oto niektóre mechanizmy dostępne za pomocą Xamarin.Essentials:

- informacje o urządzeniu,
- system plików,
- akcelerometr,
- wybieranie numeru,
- syntezator mowy,
- blokada ekranu.

Gdy używasz Xamarin.Forms razem z Xamarin.Essentials, dużą część implementacji, w tym logikę biznesową, projekt interfejsu użytkownika i niektóre mechanizmy specyficzne dla urządzeń, można wykorzystać we wszystkich platformach.

Porównanie projektowania interfejsów użytkownika na różnych platformach

Obecnie rozwój aplikacji na różnych platformach najczęściej odbywa się z wykorzystaniem wzorca projektowego **model-widok-kontroler** (ang. *Model-View-Controller* — **MVC**). Oddzielanie logiki biznesowej od projektu interfejsu użytkownika odbywa się w systemach Android, iOS/macOS i Windows za pomocą innych technik. Jednak choć we wszystkich platformach używane są różne języki programowania, do projektowania interfejsów użytkownika stosuje się język znaczników oparty na XML-u.

W systemach iOS i macOS programiści mogą używać narzędzia Interface Builder w środowisku XCode do generowania plików *.storyboard* lub *.xib*. I jeden, i drugi są plikami skryptowymi opartymi na XML-u i służącymi do przechowywania informacji o interfejsie użytkownika. Ten skrypt jest interpretowany w czasie wykonywania programu razem z kodem w językach Swift lub Objective-C, aby utworzyć interfejs użytkownika. W 2019 roku firma Apple ogłosiła powstanie nowego frameworka, SwiftUI. Za pomocą SwiftUI programiści mogą budować interfejsy użytkownika bezpośrednio za pomocą języka Swift w deklaratywny sposób.

W systemie Android programiści mogą korzystać z narzędzia **Layout Editor** w Android Studio, aby graficznie tworzyć interfejs użytkownika. Następnie można zapisać efekt w plikach z układem. Te pliki mają format XML i można je wczytywać w czasie wykonywania programu, aby utworzyć interfejs użytkownika.

W systemie Windows do projektowania interfejsów użytkownika służy języka **XAML** (ang. *Extensible Application Markup Language*). **XAML** jest językiem opartym na XML-u i używa się go do projektowania interfejsu użytkownika w systemie Windows. Dla aplikacji WPF lub UWP do projektowania interfejsu użytkownika można zastosować narzędzie XAML Designer. W .NET MAUI domyślny interfejs użytkownika aplikacji jest oparty na XAML-u. Można też stosować inny wzorzec, **MVU** (ang. *Model-View-Update*). W tym wzorcu interfejs użytkownika jest implementowany bezpośrednio w C#, bez używania XAML-a. Styl programowania z wykorzystaniem wzorca MVU przypomina korzystanie z frameworka SwiftUI.

Choć można używać SwiftUI w systemach firmy Apple lub wzorca MVU w .NET MAUI, to klasyczny sposób implementowania interfejsów użytkownika związany jest z językami znaczników opartymi na XML-u. Przyjrzyj się porównaniu z tabeli 1.4.

Tabela 1.4. Porównanie sposobów projektowania interfejsów użytkownika

Platforma	IDE	Edytor	Język	Rozszerzenie pliku
Windows	Visual Studio	XAML Designer	XAML/C#	*.xaml*
Android	Android Studio	Layout Editor	XML/Java/Kotlin	*.layout*
iOS/macOS	Xcode	Interface Builder	XML/Swift/ Objective C	*.storyboard* lub *.xib*
.NET MAUI/ Xamarin.Forms	Visual Studio	Brak	XAML/C#	*.xaml*
.NET MAUI Blazor			Razor/C#	*.razor*

W tabeli 1.4 przedstawione jest porównanie metod projektowania interfejsów użytkownika w różnych platformach.

W .NET MAUI i Xamarin.Forms do projektowania interfejsów użytkownika na wszystkie obsługiwane platformy używany jest dialekt XAML-a. W .NET MAUI można też używać innej techniki, a mianowicie Blazora, którego omówienie znajdziesz dalej w tym rozdziale.

W Xamarin.Forms interfejsy użytkownika są tworzone w XAML-u, a pliki code-behind w C#. W implementacji w każdej platformie używane są natywne kontrolki, dlatego wygląd i styl aplikacji opartych na Xamarin.Forms jest taki sam jak aplikacji natywnych.

Oto przykładowe mechanizmy zapewniane przez Xamarin.Forms:

- język XAML do tworzenia interfejsów użytkownika,
- wiązanie danych,

- gesty,
- efekty,
- style.

Choć dzięki Xamarin.Forms można współdzielić prawie cały kod interfejsu użytkownika, nadal trzeba osobno dla każdej platformy obsługiwać większość zasobów używanych przez aplikację. Tymi zasobami mogą być grafiki, czcionki czy napisy. W strukturze projektu w Xamarin.Forms występują wspólny projekt oparty na .NET Standard i kilka projektów specyficznych dla platformy. Większość prac programistycznych wykonywanych jest we wspólnym projekcie, jednak zasoby są obsługiwane osobno w projektach specyficznych dla platform.

Przejście na .NET MAUI

Wraz z unifikacją .NET Xamarin stał się częścią platformy .NET, a Xamarin.Forms został zintegrowany z .NET w formie .NET MAUI.

.NET MAUI jest podstawowym elementem .NET i odpowiada mu przestrzeń nazw `Microsoft.Maui`.

Przejście na .NET MAUI było dla Microsoftu okazją do przeprojektowania i przebudowania Xamarin.Forms od podstaw, co pozwoliło rozwiązać część niskopoziomowych problemów. .NET MAUI w porównaniu z Xamarin.Forms używa jednej struktury projektu, lepiej obsługuje mechanizm hot reloads, a także umożliwia stosowanie wzorca **MVU** oraz Blazora.

Na rysunku 1.2 widać, że dla wszystkich obsługiwanych systemów operacyjnych używana jest wspólna biblioteka *BCL*. Dla *BCL* dostępne są dwa środowiska uruchomieniowe, WinRT i Mono Runtime, zależne od platformy. Dla każdej platformy istnieje specjalna implementacja .NET, która zapewnia pełną obsługę tworzenia aplikacji natywnych.

Rysunek 1.2. Architektura .NET MAUI

Z tabeli 1.5 wynika, że w .NET MAUI wprowadzono wiele ulepszeń w porównaniu z Xamarin.Forms.

Tabela 1.5. Usprawnienia z .NET MAUI

	.NET MAUI	Xamarin.Forms
Struktura projektu	Jeden projekt	Wiele projektów
Zarządzanie zasobami	Jedna lokalizacja dla wszystkich platform	Osobno dla każdej platformy

Pełna integracja z .NET	Przestrzeń nazw `Microsoft.Maui`, oprócz Visual Studio można też stosować inne IDE. Działa w wierszu poleceń, umożliwia tworzenie, kompilowanie i uruchamianie rozwiązań w konsoli:
	`dotnet new maui`
	`dotnet build -t:Run -f net6.0-android`
	`dotnet build -t:Run -f net6.0-ios`
	`dotnet build -t:Run -f net6.0-maccatalyst`
		Przestrzeń nazw `Xamarin.Forms`, używane IDE to Visual Studio.

Usprawnienia projektowe	■ Konfiguracja za pomocą mechanizmu .NET Generic Host ■ Obsługa wstrzykiwania zależności	■ Konfiguracja rozproszona w różnych lokalizacjach
Wzorzec MVU	Implementowanie interfejsów użytkownika w nowoczesnym stylu	Brak
Blazor Hybrid	Obsługa za pomocą `BlazorWebView`	Brak

W .NET MAUI używana jest jedna struktura projektu, co upraszcza zarządzanie projektem. Można zarządzać zasobami, wstrzykiwaniem zależności i konfiguracjami w jednej lokalizacji, zamiast dla każdej platformy osobno.

.NET MAUI jest w pełsni zintegrowany z .NET, dlatego można tworzyć i kompilować projektu za pomocą wiersza poleceń pakietu .NET SDK. Daje to większy wybór jeśli chodzi o środowiska programistyczne.

W tabeli 1.5 widać, że .NET MAUI obsługuje konfigurowanie aplikacji za pomocą mechanizm .NET Generic Host, umożliwia korzystanie z różnych środowisk IDE, obsługuje wstrzykiwanie zależności, pozwala korzystać z narzędzi MVVM itd. Obsługuje też wzorzec MVU i interfejsy użytkownika aplikacji typu Blazor Hybrid. Teraz zapoznasz się z aplikacjami typu Blazor Hybrid.

Aplikacje Blazora w .NET MAUI

W tabeli 1.4, gdzie porównane są metody projektowania interfejsów użytkownika z różnych platform, wspomniałem, że w .NET MAUI dostępny jest też inny mechanizm projektowania interfejsów użytkownika — Blazor.

Framework Blazor został wprowadzony w ASP.NET Core 3.0 i służy do tworzenia za pomocą .NET interaktywnych internetowych interfejsów użytkownika działających po stronie klienta. Z .NET MAUI i Blazora można tworzyć aplikacje wieloplatformowe typu Blazor Hybrid. W aplikacjach tego rodzaju zaciera się różnica między aplikacjami natywnymi a aplikacjami internetowymi. Aplikacje typu Blazor Hybrid z .NET MAUI umożliwiają integrowanie komponentów Blazora z natywnymi mechanizmami platformy i kontrolkami interfejsu użytkownika. Komponenty Blazora uzyskują dzięki temu pełny dostęp do natywnych możliwości urządzenia.

Framework Blazor jest używany w .NET MAUI za pomocą komponentu BlazorWebView. Blazor w .NET MAUI umożliwia używanie natywnych i internetowych interfejsów użytkownika w jednej aplikacji, a nawet w jednym widoku. Dzięki temu w aplikacjach można korzystać z modelu komponentów z Blazora (komponentów Razor) opartego na technologiach HTML, CSS i Razor. We fragmentach aplikacji opartych na Blazorze można ponownie wykorzystać komponenty, układy i style stosowane w istniejącej standardowej aplikacji internetowej. Komponent `BlazorWebView` można łączyć z elementami natywnymi. Dodatkowo można korzystać z mechanizmów platformy i współdzielić stan z natywnymi odpowiednikami komponentów.

Wybór między XAML-em a Razorem w .NET MAUI

Do zaprojektowania interfejsu użytkownika w aplikacji w .NET MAUI można wybrać kilka rozwiązań:

- **XAML**. Interfejs użytkownika jest implementowany w XAML-u, podobnie jak w Xamarin.Forms. Można też zastosować wzorzec MVU, aby tworzyć elementy interfejsu użytkownika i określać ich styl bezpośrednio w C#. Niezależnie od tego, czy wybierzesz XAML, czy bezpośrednie programowanie w C#, bazowa implementacja będzie taka sama.
- **Blazor**. Interfejs użytkownika jest implementowany w formie stron Razora, co przypomina tworzenie aplikacji internetowych.
- **Aplikacja typu Blazor Hybrid**. W aplikacji używane są zarówno XAML, jak i strony Razora.

Od Ciebie zależy, jak zaprojektujesz aplikację. Możesz zdecydować się na jedną z przedstawionych możliwości lub połączyć XAML-a z Blazorem, by otrzymać optymalne rozwiązanie. Aby móc tworzyć aplikacje typu Blazor Hybrid, potrzebna jest możliwość bezpośredniego korzystania z większości istniejących bibliotek Blazora. Blazor dobrze współdziała z JavaScriptem, dlatego w trakcie prac możesz korzystać z ulubionych bibliotek javascriptowych.

Przygotowywanie środowiska programistycznego

Do tworzenia aplikacji w .NET MAUI możesz używać zarówno systemu Windows, jak i macOS, jednak niektóre platformy docelowe są dostępne w tylko jednym z nich. Do kompilacji wersji dla wszystkich docelowych platform będziesz potrzebować komputerów z oboma tymi systemami. W tej książce środowisko z systemem Windows służy do kompilowania i testowania wersji dla Androida i Windowsa. Wersje dla systemów iOS i macOS są kompilowane na komputerze Mac.

Aplikacje z .NET MAUI mogą być przeznaczone na następujące platformy:

- Android 5.0 (API 21) i nowsze,
- iOS 10 i nowsze,
- macOS 10.13 i nowsze, z obsługą technologii Mac Catalyst,
- Windows 11 i Windows 10 od wersji 1809, z **WinUI** 3 (ang. *Windows UI Library*).

Aplikacje Blazora z .NET MAUI używają kontrolek WebView specyficznych dla platform, dlatego obowiązują dla nich dodatkowe wymagania:

- Android 7.0 (API 24) i nowsze,
- iOS 14 i nowsze,
- macOS 11 i nowsze, z Mac Catalyst.

W .NET MAUI wersje aplikacji dla systemów Android, iOS, macOS i Windows można kompilować za pomocą środowiska Visual Studio na komputerze z systemem Windows. W takim środowisku potrzebny jest podłączony do sieci komputer Mac, aby możliwa była kompilacja wersji dla systemów iOS i macOS. W powiązanym komputerze Mac zainstalowane musi być środowisko Xcode, aby możliwe było debugowanie i testowanie aplikacji MAUI dla systemu iOS w środowisku programistycznym w systemie Windows.

Wersje opartych na .NET MAUI aplikacji dla systemów Android, iOS i macOS można też kompilować i testować w systemie macOS (zobacz tabelę 1.6).

Tabela 1.6. Środowiska programistyczne dla .NET MAUI

Docelowa platforma	Windows	macOS
Windows	Tak	Nie
Android	Tak	Tak
iOS	Tak (w powiązaniu z komputerem Mac)	Tak
macOS	Tylko kompilacja (w powiązaniu z komputerem Mac)	Tak

W tabeli 1.6 znajdziesz konfiguracje umożliwiające kompilację w systemach Windows i macOS.

Instalowanie .NET MAUI w systemie Windows

.NET MAUI można zainstalować w ramach środowiska Visual Studio 2022. Edycja Visual Studio Community jest bezpłatna i można ją pobrać z witryny Microsoftu (*https://visualstudio.microsoft.com/vs/community/*).

Po uruchomieniu instalatora środowiska Visual Studio pojawi się ekran podobny do tego z rysunku 1.3. Z listy opcji wybierz *.NET Multi-platform App UI development* i *.NET desktop development*. Należy też wybrać *ASP.NET and web development*, aby móc pisać w .NET MAUI aplikacje Blazora (omawiam je w Części 2. tej książki).

Rysunek 1.3. Instalowanie środowiska Visual Studio 2022

Po zakończeniu instalacji możesz sprawdzić jej efekt w wierszu poleceń za pomocą następującego polecenia `dotnet`:

```
C:\>dotnet workload list
Identyfikator zainstalowanego obciążenia      Wersja manifestu      Źródło
instalacji
----------------------------------------------------------------------------
--------
maui-windows                                  6.0.486/6.0.400    VS
17.3.32811.315
maui-maccatalyst                              6.0.486/6.0.400    VS
17.3.32811.315
maccatalyst                                   15.4.446-ci.-release-6-0-
4xx.446/6.0.400
                                                                 VS
17.3.32811.315
maui-ios                                      6.0.486/6.0.400    VS
17.3.32811.315
```

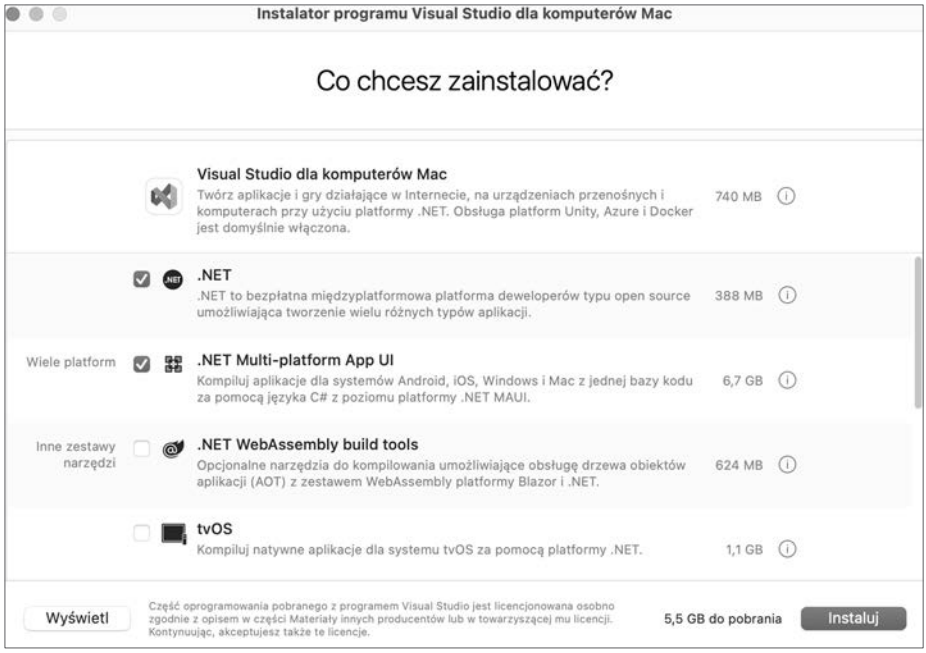
```
ios                                    15.4.446-ci.-release-6-0-
4xx.446/6.0.400
                                                              VS
17.3.32811.315
maui-android                           6.0.486/6.0.400        VS
17.3.32811.315
android                                32.0.448/6.0.400       VS
17.3.32811.315
Użyj polecenia "dotnet workload search", aby znaleźć dodatkowe obciążenia do
zainstalowania.
```

To umożliwia tworzenie, kompilowanie i uruchamianie aplikacji opartych na .NET MAUI w systemie Windows.

Instalowanie .NET MAUI w systemie macOS

Instalowanie wersji Visual Studio Community odbywa się podobnie jak w systemie Windows. Pakiet instalacyjny można pobrać za pomocą tego samego odsyłacza.

Po uruchomieniu instalatora środowiska Visual Studio zobaczysz ekran podobny do tego z rysunku 1.4.



Rysunek 1.4. Instalowanie środowiska Visual Studio for Mac 2022

Z listy opcji z rysunku 1.4 wybierz *.NET* i *.NET MAUI*.

Po zakończeniu instalacji możesz sprawdzić jej efekt w wierszu poleceń za pomocą następującej instrukcji `dotnet`:

```
% dotnet workload list
Identyfikator zainstalowanego obciążenia      Wersja manifestu      Źródło
instalacji
----------------------------------------------------------------------------
--------
wasm-tools                                    6.0.9/6.0.400         SDK 6.0.400
macos                                         12.3.453/6.0.400      SDK 6.0.400
maui-maccatalyst                              6.0.536/6.0.400       SDK 6.0.400
maui-ios                                      6.0.536/6.0.400       SDK 6.0.400
maui-android                                  6.0.536/6.0.400       SDK 6.0.400
ios                                           15.4.453/6.0.400      SDK 6.0.400
maccatalyst                                   15.4.453/6.0.400      SDK 6.0.400
maui                                          6.0.536/6.0.400       SDK 6.0.400
tvos                                          15.4.453/6.0.400      SDK 6.0.400
android                                       32.0.465/6.0.400      SDK 6.0.400
Użyj polecenia "dotnet workload search", aby znaleźć dodatkowe obciążenia do
zainstalowania.
```

To umożliwia tworzenie, kompilowanie i uruchamianie aplikacji opartych na .NET MAUI w systemie macOS.

Czego nauczysz się z tej książki?

Z tej książki nauczysz się tworzyć aplikacje wieloplatformowe za pomocą .NET MAUI. oto zagadnienia opisywane w tej pozycji:

- ekosystem .NET i .NET MAUI,
- projektowanie interfejsu użytkownika za pomocą XAML-a,
- wiązanie danych z wykorzystaniem wzorca projektowego MVVM,
- nawigacja w .NET MAUI z użyciem klasy `Shell` i projektowanie nawigacji z użyciem tras,
- wstrzykiwanie zależności,
- używanie Blazora do projektowania interfejsu użytkownika w aplikacjach typu Blazor Hybrid w .NET MAUI,
- używanie `xUnit.net` do pisania scenariuszy testów jednostkowych dla klas .NET i stosowanie `bUnit` do tworzenia scenariuszy testów dla stron Razora,
- publikowanie aplikacji w sklepach Google Play, Apple Store i Microsoft Store.

Aplikacja budowana w tej książce

W tej książce nauczysz się programowania w .NET MAUI na przykładzie tworzenia aplikacji do zarządzania hasłami. KeePass to otwartoźródłowy menedżer haseł używany przez wiele osób. Jest to jednak aplikacja zbudowana dla .NET Framework, dlatego działa tylko w systemie Windows. KeePass obejmuje bibliotekę *KeePassLib*, w której zaimplementowana jest większość logiki zarządzania zaszyfrowaną bazą danych. Dla tej bazy

stosowany jest popularny format używany do zarządzania hasłami. Na przykład środowisko IntelliJ IDEA używa bazy KeePass do przechowywania haseł używanych na różnych etapach prac programistycznych.

Aby utworzyć wieloplatformową wersję biblioteki *KeePassLib*, dostosowałem ją do .NET Standard 2.0 w ramach otwartoźródłowego projektu `KPCLib`. Jego kod źródłowy znajdziesz na stronie *https://github.com/passxyz/KPCLib*.

Bibliotekę *KPCLib* i utworzoną z użyciem Xamarin.Forms aplikację `PassXYZ.Vault` opublikowałem w sklepach z aplikacjami. W tej książce razem napiszemy aplikację `PassXYZ.Vault` od nowa z użyciem .NET MAUI. Możesz więc uczyć się programowania aplikacji wieloplatformowych na przykładzie stopniowego rozwoju tej aplikacji do momentu jej opublikowania w sklepach z aplikacjami.

Podsumowanie

W tym rozdziale zacząłem od omówienia technologii wieloplatformowych. Porównałem rozwiązania dostępne w .NET z innymi technologiami wieloplatformowymi. Następnie przyjrzałem się ekosystemowi .NET. Wyjaśniłem powiązania między .NET Framework, Mono i .NET Core. Dalej omówiłem Xamarin i .NET MAUI. Opisałem też różnice między Xamarin.Forms a .NET MAUI. Ważną cecha .NET MAUI jest to, że można używać Blazora razem z projektem interfejsu użytkownika w XAML-u. Następnie przedstawiłem .NET MAUI Blazor. Na koniec pokazałem, jak skonfigurować środowisko programistyczne używane w pozostałych rozdziałach.

W następnym rozdziale zobaczysz, jak od podstaw utworzyć aplikację opartą na .NET MAUI.

Dalsza lektura

- **.NET MAUI**. Więcej informacji na temat .NET MAUI znajdziesz w oficjalnej dokumentacji Microsoftu: *https://docs.microsoft.com/en-us/dotnet/maui/*.
- **KeePass**. Oficjalna witryna projektu KeePass to *https://keepass.info/*.

Oto .NET MAUI: aplikacja dla wielu systemów — jeden kod bazowy!

Spośród narzędzi do budowy aplikacji wieloplatformowych .NET MAUI wyróżnia się efektywnością i wysoką jakością tworzonego kodu. .NET MAUI powstał na podstawie Xamarin.Forms i służy do pisania natywnych aplikacji mobilnych i desktopowych w językach C# i XAML. Programiści cenią go za współużytkowanie zasobów, proste debugowanie i testowanie, a także za łatwą konfigurację.

Ten szczegółowy przewodnik pozwoli Ci na błyskawiczne zapoznanie się z .NET MAUI i sprawne rozpoczęcie pisania aplikacji za pomocą tej technologii. Zaprezentowano w nim filozofię działania .NET MAUI, jak również przebieg prac nad tworzeniem kompletnej aplikacji wieloplatformowej dla systemów: Android, iOS, macOS i Windows, na podstawie jednego wspólnego kodu bazowego. Podczas lektury zrozumiesz też cały cykl rozwoju oprogramowania, w tym zasady publikowania w sklepach z aplikacjami. Ciekawym elementem książki jest opis najnowszej technologii tworzenia frontendów — .NET MAUI Blazor.

Dzięki tej książce:

- odkryjesz najnowsze funkcje frameworka .NET
- nauczysz się pisać aplikacje wieloplatformowe za pomocą .NET MAUI
- zaczniesz stosować wzorzec MVVM, wiązać dane i wstrzykiwać zależności
- utworzysz aplikacje typu .NET MAUI Hybrid Blazor
- dowiesz się, jak wykonywać testy jednostkowe na kilka sposobów
- nauczysz się publikować aplikacje w różnych sklepach dla systemów mobilnych i desktopowych

Roger Ye jest menedżerem, od lat działającym w branży inżynierii oprogramowania. Tworzył systemy wbudowane dla takich firm jak Motorola, Emerson i Intersil, obecnie programuje aplikacje w EPAM Systems. Jest również autorem kilku książek o programowaniu.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-0294-7
9 788328 902947
Cena: 79,00 zł