



Technologia i rozwiązania

Xamarin

Tworzenie aplikacji cross-platform

Receptury



George Taskos



Tytuł oryginału: Xamarin Cross-Platform Development Cookbook

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-3537-0

Copyright © Packt Publishing 2016

First published in the English language under the title 'Xamarin Cross-Platform Development Cookbook - (9781785880537)'

Polish edition copyright © 2017 by Helion SA. All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/xamari.zip>

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

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	9
O korektorze merytorycznym	11
Wstęp	13
Rozdział 1. Jeden pierścień, by rządzić wszystkimi	19
Wprowadzenie	19
Tworzenie uniwersalnego rozwiązania	20
Tworzenie uniwersalnego ekranu logowania	29
Korzystanie ze wspólnych funkcjonalności systemów	35
Uwierzytelnianie użytkowników za pomocą serwisów Facebook i Google	43
Rozdział 2. Raz deklaruje, wszędzie wizualizuje	53
Wprowadzenie	53
Tworzenie uniwersalnej aplikacji z interfejsem zakładkowym	54
Kodowanie bloków funkcjonalnych interfejsu użytkownika i wyzwalaczy	61
Umieszczanie w pliku XAML wartości właściwych dla danego systemu	70
Stosowanie własnych mechanizmów do zmiany wyglądu kontrolki	75
Rozdział 3. Natywne kontrolki urządzeń i ich funkcjonowanie	81
Wprowadzenie	81
Wyświetlanie natywnych stron za pomocą wizualizatorów	82
Obsługa gestów na różnych urządzeniach	91
Wykonywanie zdjęć w aplikacji za pomocą natywnej kontrolki aparatu fotograficznego	95
Rozdział 4. Różne pojazdy, ten sam silnik	109
Wprowadzenie	109
Sposoby tworzenia uniwersalnego kodu dla różnych systemów	110
Korzystanie z lokalizatora zależności	118

Korzystanie z zewnętrznego kontenera wstrzykiwanych zależności	122
Wzorzec projektowy MVVM aplikacji	127
Korzystanie z komunikatora zdarzeń	136
Globalizowanie aplikacji	138
Rozdział 5. Hej, gdzie są moje dane?	147
Wprowadzenie	147
Kodowanie uniwersalnego dostępu do bazy danych SQLite	148
Wykonywanie operacji CRUD na bazie SQLite	155
Korzystanie z internetowych usług REST	161
Korzystanie z natywnych bibliotek REST i wydajne wysyłanie zapytań przez sieć	168
Rozdział 6. Jeden za wszystkich, wszyscy za jednego	177
Wprowadzenie	177
Tworzenie uniwersalnych wtyczek	178
Robienie zdjęć i nagrywanie filmów	185
Odczytywanie danych GPS	190
Wyświetlanie i wysyłanie lokalnych powiadomień	195
Rozdział 7. Wiązanie danych	201
Wprowadzenie	201
Wiązanie danych w kodzie C#	202
Wiązanie danych w kodzie XAML	204
Dwukierunkowe wiązanie danych	206
Korzystanie z konwerterów wartości	211
Rozdział 8. Lista do wglądu	217
Wprowadzenie	217
Wyświetlanie kolekcji danych i zaznaczanie wiersza listy	218
Tworzenie, usuwanie i odświeżanie elementów listy	222
Dostosowywanie szablonu wiersza	228
Grupowanie elementów i tworzenie listy nawigacyjnej	233
Rozdział 9. Gesty i animacje	239
Wprowadzenie	239
Definiowanie detektorów gestów w języku XAML	240
Obsługa gestów za pomocą natywnych wizualizatorów	242
Definiowanie uniwersalnych animacji	249
Rozdział 10. Koniecznie przetestuj aplikację	255
Wprowadzenie	255
Definiowanie testów jednostkowych	256
Definiowanie testów akceptacyjnych za pomocą platformy Xamarin.UITest	262
Testowanie interfejsu użytkownika za pomocą terminala Xamarin.UITest REPL	269
Przesyłanie definicji testów do usługi Xamarin Test Cloud i ich uruchamianie	279

Rozdział 11. Trzy, dwa, jeden — start i kontrola	291
Wprowadzenie	291
Korzystanie z usługi Xamarin Insights	292
Publikowanie aplikacji dla systemu iOS	302
Publikowanie aplikacji dla systemu Android	307
Publikowanie aplikacji dla Windows Phone	316
Skorowidz	321

Jeden pierścień, by rządzić wszystkimi

W tym rozdziale opisane są następujące procedury:

- tworzenie uniwersalnego rozwiązania
- tworzenie uniwersalnego ekranu logowania
- korzystanie ze wspólnych funkcjonalności systemów
- uwierzytelnianie użytkowników za pomocą serwisów Facebook i Google

Wprowadzenie

Xamarin.Forms jest platformą dającą możliwość tworzenia nie tylko współdzielonych modeli, algorytmów i procedur dostępu do danych, jak w tradycyjnych rozwiązaniach opartych na platformie Xamarin, ale również uniwersalnych interfejsów użytkownika, wspólnych dla urządzeń z systemami iOS, Android i Windows Phone. Dzięki Xamarin.Forms można szybko i łatwo tworzyć efektywne aplikacje i narzędzia do przetwarzania danych.

Realizacja tych celów jest możliwa, ponieważ platforma Xamarin wykorzystuje supernowoczesny język C#, siłę bibliotek **.NET Base Class Libraries (BCL)** obsługujących natywne interfejsy API oraz dwa doskonale środowiska programistyczne: Xamarin Studio i Microsoft Visual Studio. Pamiętaj jednak, że do tworzenia i uruchamiania aplikacji dla iOS za pomocą systemu Xamarin Build Host niezbędny jest komputer Mac podłączony do sieci i działający jako serwer.

Ta książka zawiera praktyczne przepisy, rozwiązania i instrukcje, opisujące krok po kroku tworzenie profesjonalnych, uniwersalnych aplikacji. Dowiesz się z niej, jak według najlepszych

wzorców i praktyk budować interfejsy użytkownika wspólne dla różnych urządzeń, dostosowywać warstwy i widoki oraz wstrzykiwać implementacje charakterystyczne dla poszczególnych typów urządzeń.

W tym rozdziale szczegółowo opisuję, jak tworzy się uniwersalne rozwiązanie, definiuje ekran logowania, zapisuje dane na różnych urządzeniach i wykorzystuje komponent `Xamarin.Auth` umożliwiający logowanie się do aplikacji za pomocą serwisów Facebook i Google. Świetnie! To jest właśnie to, co jest potrzebne do tworzenia aplikacji z prawdziwego zdarzenia.

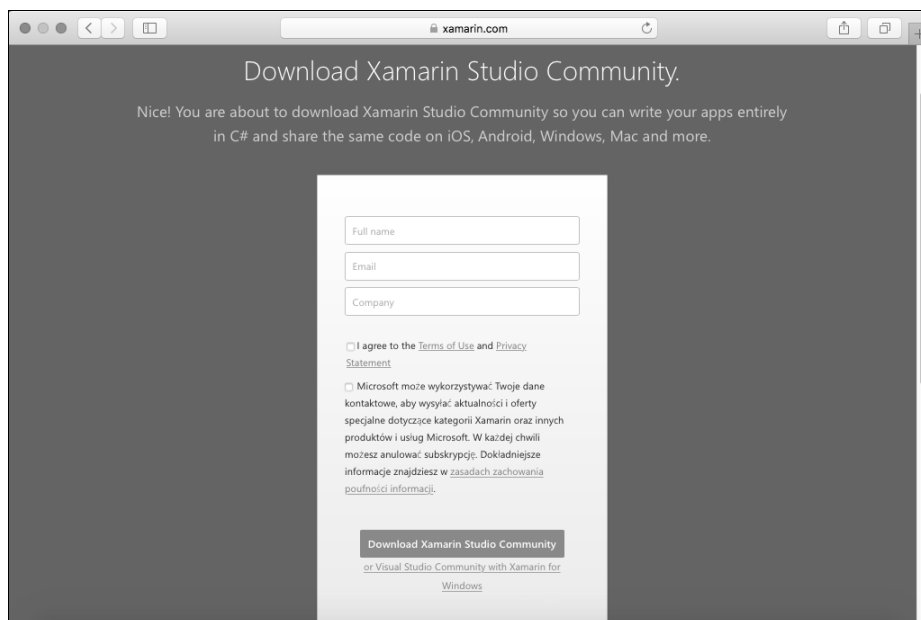
Tworzenie uniwersalnego rozwiązania

Pierwsze kroki z platformą `Xamarin.Forms` są bardzo proste. Program instalacyjny wszystko konfiguruje, środowisko IDE tworzy projekt i działająca aplikacja jest gotowa w ciągu kilku minut! Zaczynamy!

Przygotuj się

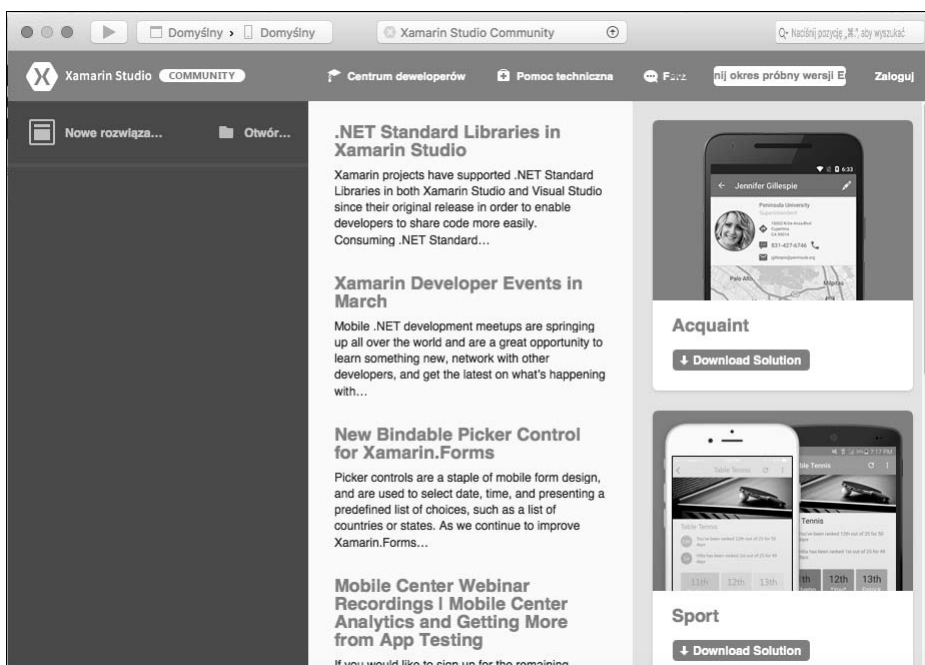
Zanim zaczniesz tworzyć uniwersalną aplikację, musisz przygotować niezbędne narzędzia. Wystarczy do tego celu użyć tylko jednego programu instalacyjnego, pobranego ze strony `Xamarin`. Wykonaj następujące kroki:

1. Otwórz stronę <http://xamarin.com/download>.



2. Wprowadź dane niezbędne do rejestracji w systemie.
3. Kliknij odnośnik *Download Xamarin Studio Community* (pobierz środowisko Xamarin Studio Community).
4. Po pobraniu pliku instalacyjnego uruchom go i postępuj według pojawiających się wskazówek. W trakcie instalacji program pobierze i zainstaluje wszystkie potrzebne komponenty.

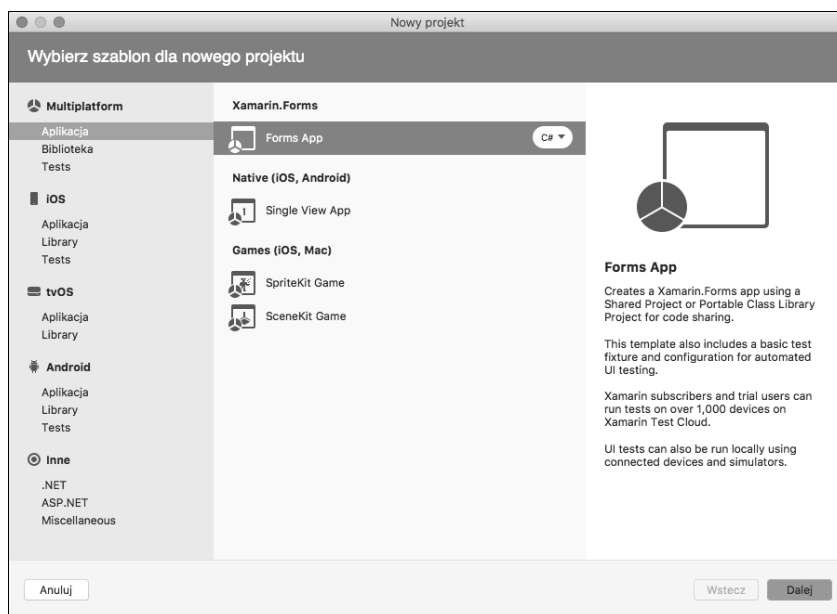
Po zakończonej instalacji będzie dostępne środowisko IDE Xamarin Studio, przeznaczone do tworzenia uniwersalnych aplikacji:



Jak to zrobić?

Tworzenie rozwiązania opartego na platformie Xamarin.Forms, wykorzystującego szablony dostępne w środowiskach Xamarin Studio i Visual Studio, jest proste. W zależności od środowiska dostępne są trzy (Xamarin Studio dla Windows) lub cztery (Xamarin Studio dla iOS i Visual Studio) wzorcowe projekty. Oczywiście tworzone rozwiązanie możesz otworzyć w dowolnym środowisku i w trakcie programowania dodawać do niego kolejne projekty. W tej części rozdziału utworzysz rozwiązanie oparte na pustym szablonie, dostępnym w środowisku Xamarin Studio dla iOS, a następnie w środowisku Visual Studio dodasz do niego projekt dla systemu Windows Phone.

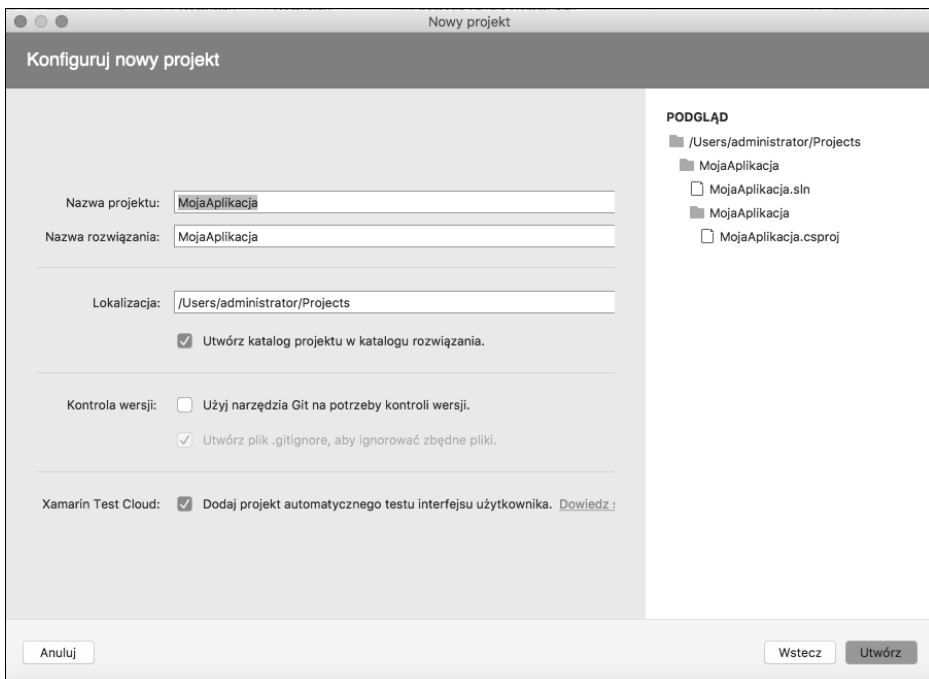
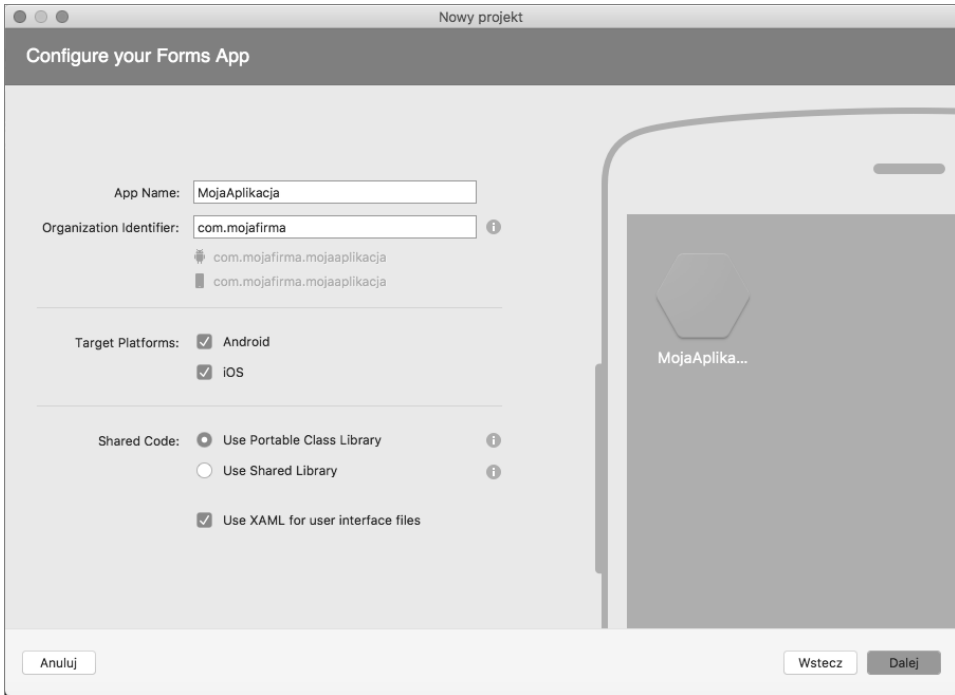
Uruchom środowisko Xamarin Studio i kliknij polecenie menu *Plik/Nowe rozwiązanie*. W oknie, które się pojawi, w sekcji *Multiplatform* kliknij opcję *Aplikacja*. W środkowej części okna pojawi się kilka szablonów aplikacji, jak zrzucie:



Kliknij opcję *Forms App*, a następnie przycisk *Dalej*. Pojawi się okno z polem *App Name* (nazwa aplikacji) do wpisania nazwy tworzonej aplikacji oraz polem *Organization Identifier* (identyfikator organizacji) do wprowadzenia nazwy pakietu (*package*) w systemie Android lub paczki (*bundle*) w systemie iOS. Docelowy system możesz wybrać za pomocą opcji *Target Platforms* (docelowe systemy).

Następna opcja, *Shareed Code* (współdzielony kod), jest wykorzystywana w przypadku zastosowania w aplikacji biblioteki przenośnych klas (*Portable Class Library*) lub biblioteki współdzielonej (*Shared Library*). Wybranie opcji *Shared Library* powoduje utworzenie projektu wykorzystującego dyrektywy kompilacji warunkowej, w którym biblioteki są umieszczane w głównym pliku binarnym. Natomiast opcja *Portable Class Library* umożliwia wybór docelowego profilu aplikacji i rozpowszechnianie jej w serwisach NuGet lub Xamarin Component Store. W rozwiązaniach opisanych w tej książce stosowana jest tylko ta druga opcja wraz ze związanymi z nią wzorcami i praktykami.

Wybierz opcje jak na drugim zrzucie na następnej stronie i kliknij przycisk *Dalej*. W następnym oknie w polu *Nazwa projektu* wpisz tekst, który będzie wykorzystywany do tworzenia nazw projektów w formacie *[NazwaProjektu].[System]* dla poszczególnych systemów. W polu *Nazwa rozwiązania* wpisz nazwę rozwiązania, jak na kolejnym zrzucie. W polu *Lokalizacja* możesz wskazać katalog, w którym zostanie zapisane rozwiązanie, a za pomocą opcji *Kontrola wersji* wybrać możliwość korzystania z systemu kontroli wersji Git.



Zaznacz opcję *Xamarin Test Cloud* i kliknij przycisk *Utwórz*. Zwróć uwagę, że nowe rozwiązanie składa się z czterech projektów. W rozdziale 10., „Koniecznie przetestuj aplikację”, dowiesz się, jak tworzyć testy rozwiązania. Ostatni projekt jest dodawany tylko wtedy, gdy rozwiązanie jest tworzone w środowisku Xamarin Studio.

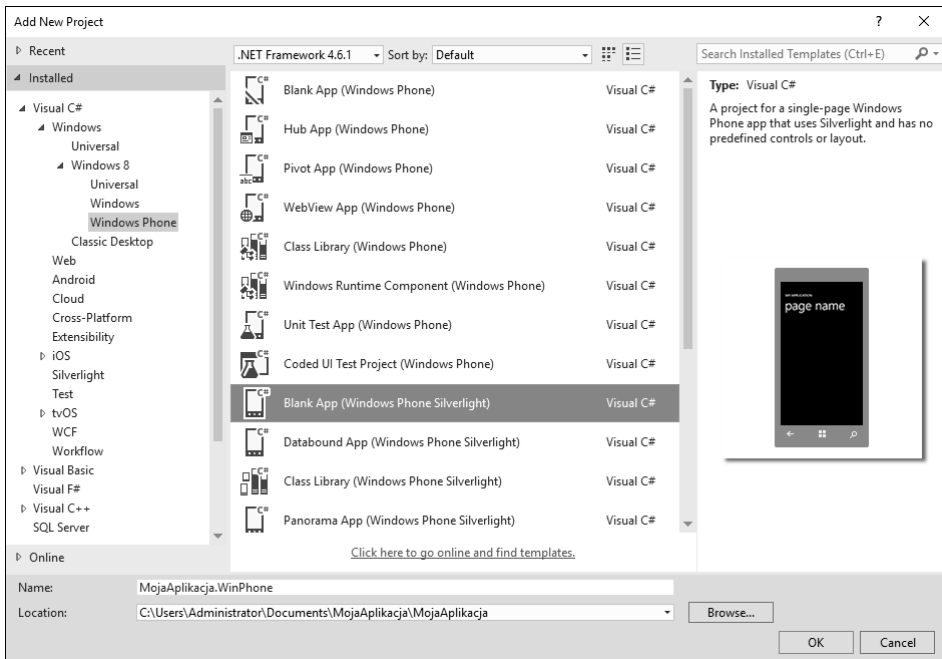


Teraz w górnej części okna wybierz symulator telefonu iPhone i kliknij przycisk odtwarzania. Uruchomi się symulator, a w nim Twoja aplikacja z komunikatem *Welcome to Xamarin Forms!* (Witaj w Xamarin Forms!). Gratulacje! Podobną operację możesz wykonać dla systemu Android, klikając prawym przyciskiem myszy projekt *MojaAplikacja.droid* i wybierając opcję *Ustaw jako projekt startowy*. Do uruchamiania projektów dla systemu Android możesz wykorzystać symulator Google Xamarin Android Player, dostępny na stronie <https://xamarin.com/android-player>, który moim zdaniem jest bardziej efektywny i szybszy. Kliknij przycisk odtwarzania i zobacz ten sam komunikat w swoim ulubionym symulatorze.

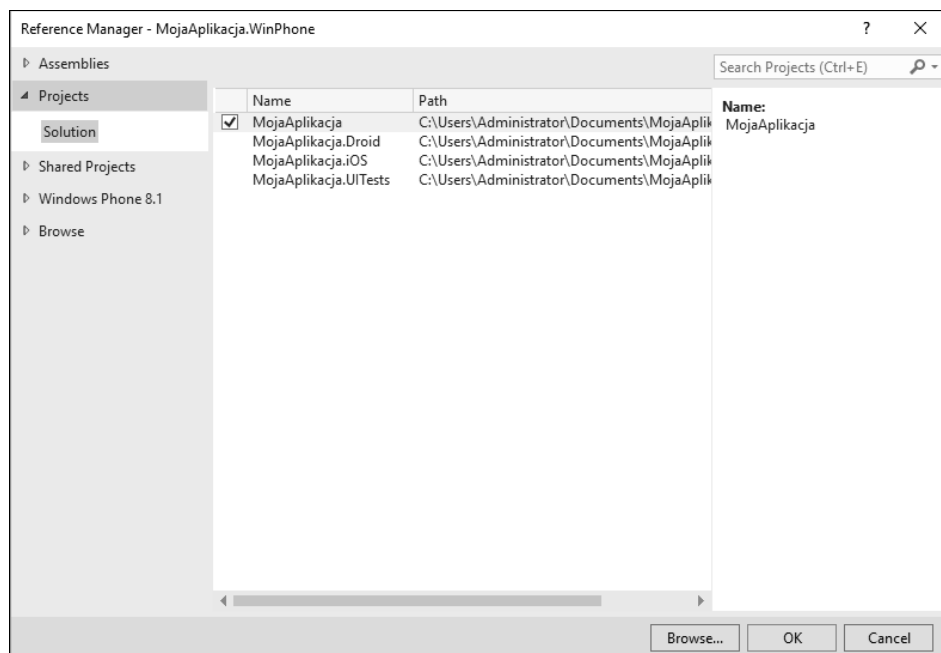
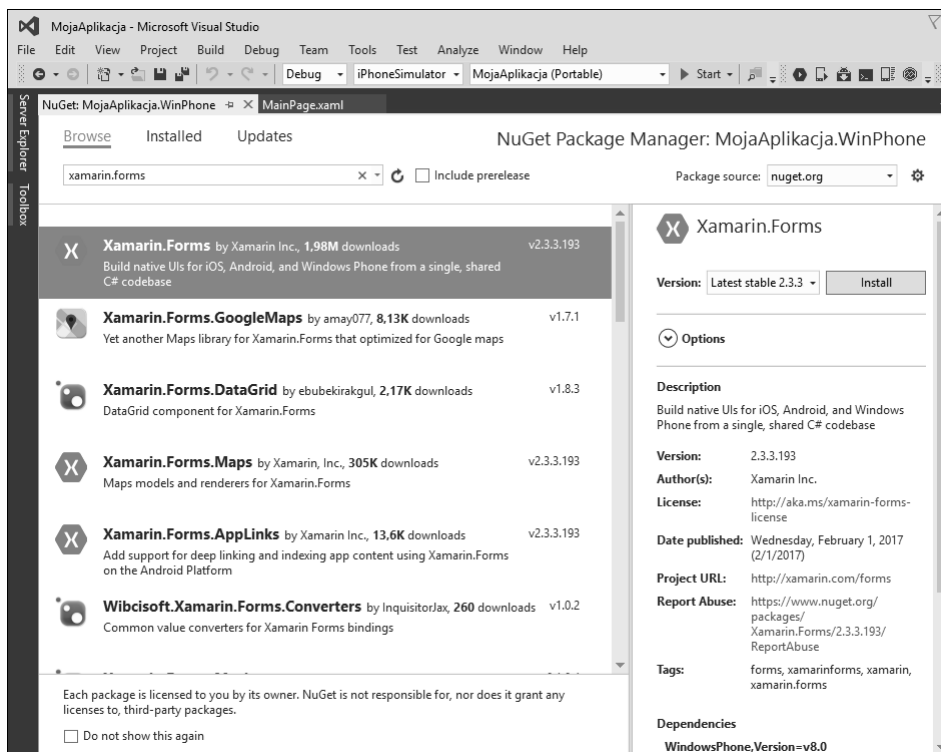
Świetnie! Wystarczyło kilka razy kliknąć, aby otrzymać aplikację dla systemów iOS i Android! Zanim dokładniej poznasz strukturę projektu, przejdź do środowiska Visual Studio i dodaj do rozwiązania projekt Windows Phone. Wykonaj w tym celu następujące czynności:

1. Otwórz środowisko Visual Studio, kliknij opcję menu *File/Open/Project/Solution* (plik/otwórz/projekt/rozwiązanie) i wybierz plik rozwiązania, w tym przypadku *MojaAplikacja.sln*.
2. W panelu *Solution Explorer* (eksplorator rozwiązania) kliknij prawym przyciskiem myszy nazwę rozwiązania i z podręcznego menu wybierz *Add/New Project* (dodaj/nowy projekt).
3. W panelu po lewej stronie kliknij sekcję *Visual C#/Windows/Windows 8/Windows Phone*, w środkowej części kliknij pozycję *Blank App (Windows Phone Silverlight)*, a następnie w polu *Name* (nazwa) wpisz nazwę projektu. Aby zachować przyjętą konwencję nazw, wpisz **MojaAplikacja.WinPhone** i kliknij przycisk *OK* (zobacz zrzut na następnej stronie).
4. W następnym oknie, które się pojawi, wybierz opcję *Windows Phone 8.0* lub *Windows Phone 8.1*.

Voilà! Do rozwiązania dodałeś projekt dla kolejnego urządzenia. Jednak brakuje w nim kilku elementów, warunkujących korzystanie z platformy Xamarin.Forms. Musisz jeszcze dodać niezbędne pakiety i biblioteki oraz dokonać niewielkich modyfikacji w kodzie startowym aplikacji.

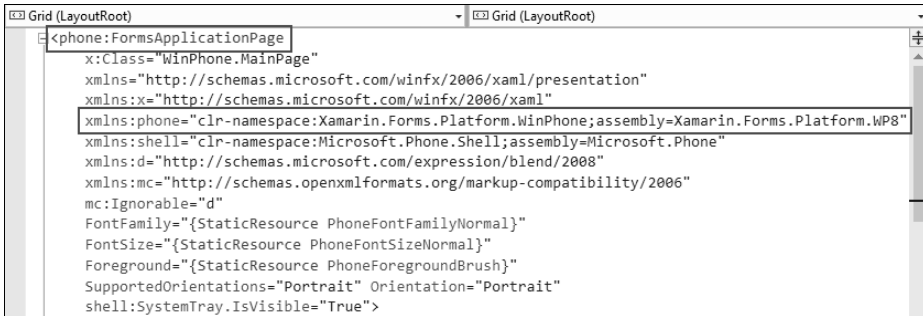


5. Kliknij prawym przyciskiem myszy nazwę utworzonego przed chwilą projektu *MojaAplikacja.WinPhone* i wybierz polecenie *Manage NuGet Packages* (zarządzaj pakietami NuGet).
6. Kliknij odnośnik *Browse* (przeglądaj), w polu wyszukiwania wpisz **xamarin.forms**. Na liście znalezionych pakietów kliknij *Xamarin.Forms*, a następnie przycisk *Install* (zainstaluj), aby dodać pakiet do projektu (zobacz pierwszy zrzut na następnej stronie).
7. Do projektu musisz jeszcze dodać odwołanie do biblioteki PCL. W tym celu kliknij prawym przyciskiem myszy sekcję *MojaAplikacja.WinPhone/References* i wybierz polecenie *Add Reference* (dodaj odwołanie). W panelu po lewej stronie kliknij sekcję *Projects/Solution* (projekty/rozwiązania). W środkowym panelu zaznacz pole wyboru w wierszu *MojaAplikacja* (zobacz drugi zrzut na następnej stronie).
Analogicznie kliknij sekcję *Assemblies/Extensions* (moduły/rozszerzenia) i zaznacz moduł *System.Windows.Interactivity*. Na koniec kliknij *OK*.
8. Prawie gotowe. Teraz musisz jeszcze wprowadzić zmiany w pliku XAML głównej strony, aby przekształcić projekt w aplikację opartą na *Xamarin.Forms*. Kliknij dwukrotnie plik *MojaAplikacja.WinPhone/MainPage.xaml* i zmień znacznik `phone:PhoneApplicationPage` na `phone:FormsApplicationPage`, jak na następnym zrzucie.



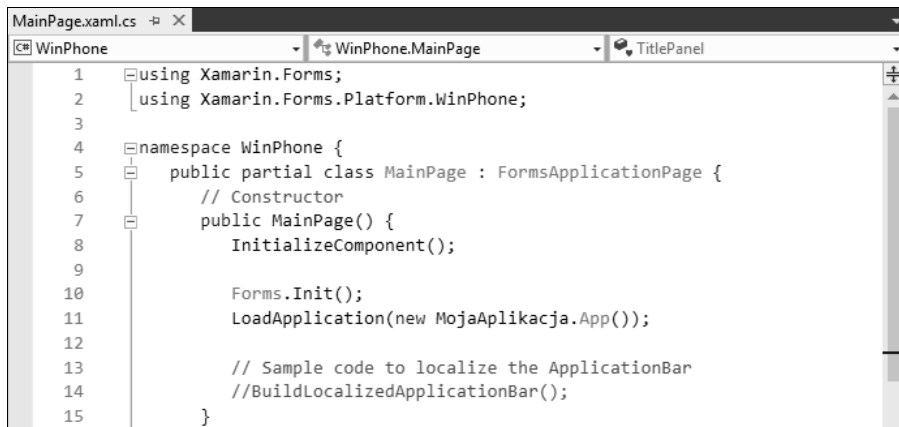
9. Oprócz tego musisz jeszcze zmienić przestrzeń nazw (nie jest to jednak konieczne, jeżeli korzystasz z jakiegoś pomyslowego narzędzia, na przykład ReSharper, które wykona tę operację za Ciebie). Odszukaj wiersz rozpoczynający się od `xmlns:phone` i zastąp go poniższym kodem. Upewnij się, że listing po zmianach wygląda jak na zrzucie.

```
xmlns:phone="clr-namespace:Xamarin.Forms.Platform.WinPhone;
↳assembly=Xamarin.Forms.Platform.WP8"
```



```
Grid (LayoutRoot)
  <phone:FormsApplicationPage
    x:Class="WinPhone.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Xamarin.Forms.Platform.WinPhone;assembly=Xamarin.Forms.Platform.WP8"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">
```

10. Teraz otwórz plik *MainPage.xaml.cs* i zmień w nim odwołanie do klasy rodzicielskiej `PhoneApplicationPage` na `FormsApplicationPage`. Oprócz tego usuń wszystkie istniejące instrukcje `using` i wpisz dwie inne, jak na następnym zrzucie. Na koniec dopisz kod pokazany w wierszach 10 i 11:



```
MainPage.xaml.cs
WinPhone
  WinPhone.MainPage
  TitlePanel
  1  using Xamarin.Forms;
  2  using Xamarin.Forms.Platform.WinPhone;
  3
  4  namespace WinPhone {
  5  public partial class MainPage : FormsApplicationPage {
  6  // Constructor
  7  public MainPage() {
  8  InitializeComponent();
  9
  10 Forms.Init();
  11 LoadApplication(new MojaAplikacja.App());
  12
  13 // Sample code to localize the ApplicationBar
  14 //BuildLocalizedApplicationBar();
  15 }
```

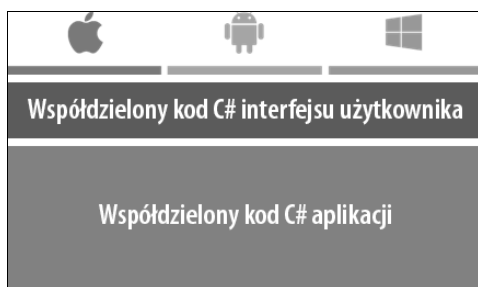
11. Kliknij prawym przyciskiem myszy projekt *MojaAplikacja.WinPhone*, wybierz opcję *Set as Start Up Project* (ustaw jako projekt startowy), a następnie naciśnij klawisz *F5* lub kliknij opcję menu *Debug/Start Debugging* (debugowanie/rozpocznij debugowanie).

Gratulacje! Utworzyłeś rozwiązanie złożone z trzech projektów, przeznaczonych dla poszczególnych systemów (iOS, Android i Windows Phone), oraz z biblioteki PCL. Pamiętaj, że projekt dla Windows Phone jest aktywny tylko w środowisku Visual Studio.

Jak to działa?

Utworzyłeś więc trzy klasyczne natywne aplikacje, jak również współdzielony kod. Świetnie!

Przyjrzyj się temu oto schematowi architektury platformy Xamarin.Forms, odzwierciedlającemu utworzoną właśnie aplikację:



Na najwyższym poziomie architektury widnieją trzy projekty charakterystyczne dla poszczególnych systemów. Poniżej znajduje się poziom współdzielonego kodu C# interfejsu użytkownika, a pod nim warstwa współdzielonego kodu C# aplikacji.

Używając platformy Xamarin.Forms, interfejs użytkownika wykorzystujący współdzielone kontrolki, wyświetlany przez natywne mechanizmy, w poszczególnych systemach definiuje się tylko raz. W każdym systemie zawarty jest natywny mechanizm wyświetlający każdą współdzieloną kontrolkę. Gdy umieścisz w interfejsie etykietę (Label), wtedy mechanizm dostępny w iOS przekształca ją w kontrolkę UILabel, w Androidzie w TextView, a w Windows Phone w TextBlock.

Rozwiń główny projekt *MojaAplikacja (Portable)*, a następnie *References*. Na liście znajdziesz między innymi dwie biblioteki: `Xamarin.Forms.Core` i `Xamarin.Forms.Xaml`. Pierwsza zawiera kod głównego algorytmu platformy Xamarin.Forms, a druga służy do przetwarzania kodu XAML, szczegółowo opisanego w rozdziale 2., „Raz deklaruj, wszędzie wizualizuj”.

Teraz przejrzyj projekty dla poszczególnych systemów. Najpierw zajmij się Androidem. Rozwiń sekcję *MojaAplikacja.Droid/References*. Znajdziesz w niej między innymi wspomniane wcześniej biblioteki `Xamarin.Forms.Core` i `Xamarin.Forms.Xaml` oraz dodatkowo `Xamarin.Forms.Platform.Android`. Ta ostatnia jest biblioteką charakterystyczną dla systemu Android. Zdefiniowane są w niej wszystkie mechanizmy wyświetlające treść na ekranie. Jej zadaniem jest kojarzenie ogólnych klas interfejsu użytkownika z klasami typowymi dla Androida.

Ta sama zasada obowiązuje w projekcie na iOS. Charakterystyczna dla niego biblioteka nosi nazwę `Xamarin.Forms.Platform.iOS`.

W każdym projekcie w liście bibliotek znajduje się również odwołanie do głównego projektu definiującego bibliotekę PCL, bo bez głównego kodu aplikacja nie będzie działać, prawda?

Wszystko wygląda świetnie, ale w jaki sposób projekty dla poszczególnych systemów są powiązane z platformą Xamarin.Forms? Aby się tego dowiedzieć, musisz zajrzeć do kodu startowego aplikacji, właściwego dla każdego systemu.

Zacznij od projektu dla Windows Phone. Zmieniłeś w nim kod w pliku *MainPage.xaml.cs*. Interfejs użytkownika nie jest teraz oparty na klasie *PhoneApplicationPage*, lecz na *FormsApplicationPage* z przestrzeni *Xamarin.Forms.Platform.WinPhone*. W kodzie ważne jest wywołanie konstruktora tej klasy, *Forms.Init()*, a także metody *LoadApplication()*. Argumentem tej metody jest nowa instancja klasy *Xamarin.Forms.Application*, zdefiniowanej w bibliotece PCL. Otwórz teraz plik *MojaAplikacja (Portable)/App.xaml/App.xaml.cs*. Zawiera on kod startowy *Xamarin.Forms*. Konstruktor klasy zawiera trochę kodu inicjalizującego interfejs użytkownika. Z następnej części rozdziału dowiesz się więcej na temat tego kodu, jak również go zmienisz.

W projekcie dla iOS w pliku *AppDelegate.cs* wywoływana jest analogiczna do opisanej wyżej metoda *FinishedLaunching()*. Zwróć jednak uwagę, że klasa *AppDelegate* dziedziczy cechy po klasie *Xamarin.Forms.Platform.iOS.FormsApplicationDelegate*.

Na koniec przyjrzyj się projektowi dla Androida. W pliku *MainActivity.cs* zdefiniowana jest klasa aktywności *MainActivity*, dziedzicząca cechy po klasie *Xamarin.Forms.Platform.Android.FormsApplicationActivity*. Główna metoda tej klasy, *OnCreate()*, inicjalizuje pewne charakterystyczne zmienne oraz klasę *Xamarin.Forms*. Zwróć uwagę, że metoda *Forms.Init()* w tym systemie jest wywoływana inaczej, to jest z dwoma argumentami: obiektami typu *MainActivity* i *Bundle*.

Zobacz też

- <https://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/views>

Tworzenie uniwersalnego ekranu logowania

W niemal każdej aplikacji jest ekran logowania. W tym przykładzie poznasz jedną z najbardziej podstawowych stron platformy *Xamarin.Forms*. Utworzysz ekran logowania, w którym użytkownik będzie wpisywał swój login i hasło, a następnie klikał przycisk, aby uzyskać dostęp do aplikacji.

Możesz utworzyć nowe lub kontynuować pracę nad aktualnym rozwiązaniem.

Jak to zrobić?

W środowisku Xamarin Studio rozwiń główny projekt *MojaAplikacja* i wykonaj następujące kroki:

1. Otwórz plik *App.xaml/App.xaml.cs* i zastąp kod konstruktora `App()` poniższym kodem:

```
var userNameEntry = new Entry {
    Placeholder = "login"
};

var passwordEntry = new Entry {
    Placeholder = "hasło",
    IsPassword = true
};

var loginButton = new Button {
    Text = "Zaloguj"
};

loginButton.Clicked += (sender, e) => {
    Debug.WriteLine(string.Format("Login: {0} - Hasło: {1}",
        userNameEntry.Text, passwordEntry.Text));
};

MainPage = new ContentPage {
    Content = new StackLayout {
        VerticalOptions = LayoutOptions.Center,
        Children = {
            userNameEntry,
            passwordEntry,
            loginButton
        }
    }
};
```

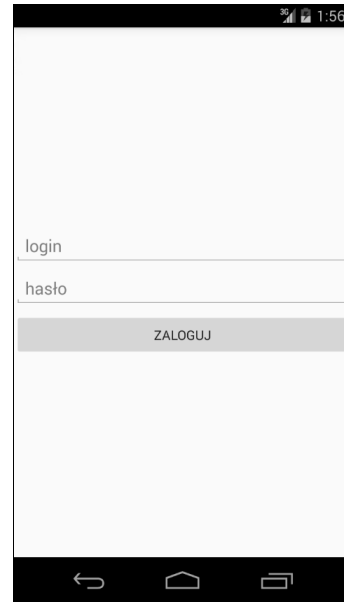
2. Uruchom aplikację w każdym systemie. Na ekranie powinny pojawić się proste pola testowe i przycisk.
3. W odpowiednich polach wpisz login i hasło.
4. Kliknij przycisk *Zaloguj* i obserwuj komunikaty wyświetlane przez aplikację, zdefiniowane w procedurze obsługi zdarzenia `Clicked`.

Ekran powinien wyglądać jak na zrzutach.

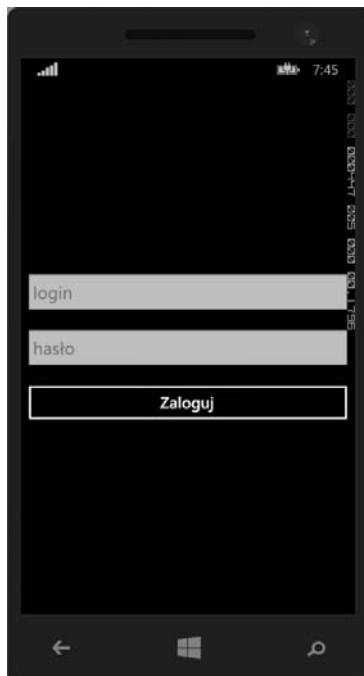
W systemie iOS:



W systemie Android:



W systemie Windows Phone:



Wszystko wygląda świetnie: elementy w domyślnym układzie umieszczone są w środkowej części ekranu i zajmują całą jego szerokość. Oczywiście każdy widok i kontener ma właściwości umożliwiające dowolne rozmieszczanie kontrolki na ekranie i określanie odstępów pomiędzy nimi. Zmień teraz nieco układ. Przy okazji uporządkujesz kod. W tym celu wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy główny projekt *MojaAplikacja*, następnie wybierz polecenie *Dodaj/Nowy folder* i utwórz nowy folder o nazwie *Custom Pages* (własne strony).
2. Kliknij prawym przyciskiem myszy nowo utworzony folder i wybierz polecenie *Dodaj/Nowy plik*. W oknie, które się pojawi, zaznacz opcję *Pusta klasa*, w polu *Nazwa* wpisz **MainPage** (strona główna) i kliknij przycisk *Nowy*.
3. W definicji nowej klasy wpisz następujący kod:

```
public class MainPage: ContentPage {
    Entry userNameEntry;
    Entry passwordEntry;
    Button loginButton;
    StackLayout stackLayout;

    public MainPage() {
        userNameEntry = new Entry {
            Placeholder = "login"
        };

        passwordEntry = new Entry {
            Placeholder = "hasło",
            IsPassword = true
        };

        loginButton = new Button {
            Text = "Zaloguj"
        };

        loginButton.Clicked += LoginButton_Clicked;
        this.Padding = new Thickness(20);
        stackLayout = new StackLayout {
            VerticalOptions = LayoutOptions.FillAndExpand,
            HorizontalOptions = LayoutOptions.FillAndExpand,
            Orientation = StackOrientation.Vertical,
            Spacing = 10,
            Children = {
                userNameEntry,
                passwordEntry,
                loginButton
            }
        };

        this.Content = stackLayout;
    }
}
```

```

void LoginButton_Clicked(object sender, EventArgs e) {
    Debug.WriteLine(string.Format("Login: {0} - Hasło: {1}",
        userNameEntry.Text, passwordEntry.Text));
}
}

```

4. Na początku pliku wpisz następujący wiersz:

```
using Xamarin.Forms;
```

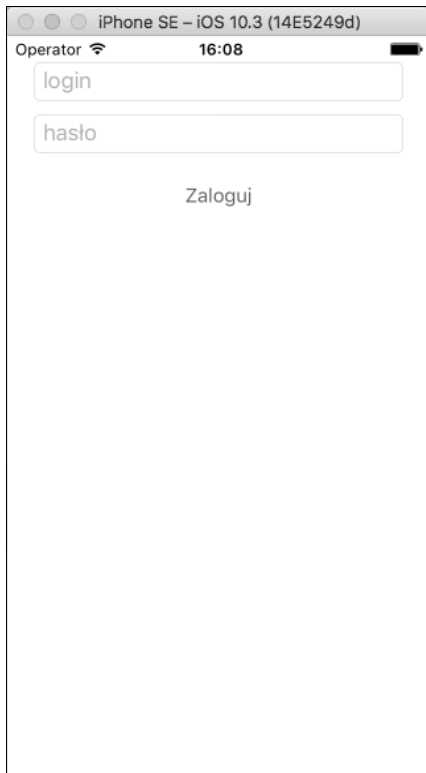
5. W pliku *App.xaml/App.xaml.cs* usuń wpisany wcześniej kod i wpisz w jego miejscu następujący wiersz:

```
MainPage = new MainPage();
```

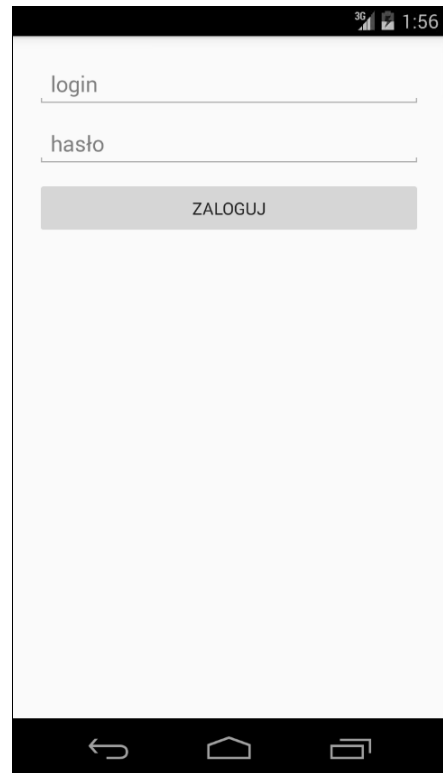
6. Uruchom aplikację dla każdego systemu i sprawdź efekty wprowadzonych zmian.

Ekran powinien wyglądać jak na zrzutach.

W systemie iOS:



W systemie Android:



W systemie Windows Phone:



Jak to działa?

Plik *App.xaml.cs* zawiera główny kod startowy aplikacji. Definiowana w nim klasa ma właściwość *MainPage*, której przypisywany jest obiekt tworzący stronę startową aplikacji.

Właściwość *Content* (zawartość) klasy *MainPage* jest przypisywany obiekt typu *StackLayout* (układ pionowy). W jego właściwości *Children* (dzieci) umieszczane są trzy widoki: *userNameEntry* (pole do wpisania loginu), *passwordEntry* (pole do wpisania hasła) i *loginButton* (przycisk logowania). Dodatkowo właściwościom *VerticalOptions* (opcje układu pionowego), *HorizontalOptions* (opcje układu poziomego), *Orientation* (orientacja) i *Spacing* (odstęp) przypisywane są odpowiednie wartości.

Właściwość *Clicked* (kliknięty) obiektu *loginButton* została przypisana procedura obsługi zdarzenia wywołanego kliknięciem przycisku logowania. W tej chwili procedura ta nie wykonuje żadnych operacji oprócz wyświetlania komunikatów w panelu *Dane wyjściowe aplikacji*.

W utworzonym interfejsie użytkownika wykorzystany jest adaptacyjny układ elementów, dostępny w platformie Xamarin.Forms. Nie trzeba określać położenia poszczególnych kontroltek, wystarczy zdefiniować kontenery, umieścić w nich elementy potomne i zdefiniować reguły ich rozmieszczenia na ekranie.

Aby nadać ekranowi logowania ostateczny wygląd, zostały zmienione właściwości `VerticalOptions` i `HorizontalOptions` kontenera `StackLayout`, definiujące rozmieszczenie kontroltek na ekranie, oraz właściwość `Spacing`, określająca odstępy pomiędzy kontrolkami i marginesy wokół nich.

W platformie Xamarin.Forms nie istnieje prosty sposób określania wysokości i szerokości elementów. Definiując widok, określa się wymagania dotyczące ich wymiarów bez gwarancji, że zostaną one spełnione. Właściwości `Width` (szerokość) i `Height` (wysokość) są tylko do odczytu.

Co dalej?

Tworząc uniwersalne aplikacje, na przykład oparte na platformie Xamarin.Forms, nie zapominaj, że użytkownicy korzystają z wielu różnych urządzeń o różnych wielkościach ekranów, a w poszczególnych systemach stosowane są inne jednostki wymiarów elementów. W iOS są to unity, a w Androidzie i Windows Phone tak zwane piksele niezależne od urządzenia (dpi).

Jak sobie z tym radzi Xamarin.Forms? Jeżeli na przykład właściwość `Spacing` zostanie przypisana wartość 10, wówczas zostanie ona przełożona na 10 unitów w iOS i 10 dpi w Androidzie i Windows Phone.

Korzystanie ze wspólnych funkcjonalności systemów

Największym wyzwaniem towarzyszącym tworzeniu uniwersalnych aplikacji jest uwzględnianie różnic pomiędzy interfejsami API typowymi dla poszczególnych systemów. Wiele funkcjonalności jest wspólnych, ale korzystanie z nich nie zawsze jest proste. Na przykład wyświetlenie okienka z komunikatem lub otwarcie strony o zadanym adresie URL jest możliwe we wszystkich systemach, ale interfejsy API umożliwiające wykonanie tych operacji są w poszczególnych urządzeniach zupełnie inne. Ponadto biblioteka PCL nie oferuje dostępu do wszystkich funkcjonalności urządzeń.

Standardowo Xamarin.Forms obsługuje niektóre wspólne dla wszystkich systemów funkcjonalności. W rozdziale 4., „Różne pojazdy, ten sam silnik”, opisana jest architektura biblioteki oraz sposoby tworzenia własnych klas, ich implementacji i wykorzystania w wykonywanym kodzie.

Jak to zrobić?

W tej części poznasz następujące interfejsy API platformy Xamarin.Forms:

- `Device.OpenUri` do otwierania adresów URL w przeglądarce danego systemu;
- `Device.StartTimer` do wykonywania zadań w określonym momencie;
- `Device.BeginInvokeOnMainThread` do uruchamiania kodu, zazwyczaj w tle systemu, w celu zmiany zawartości ekranu;
- `Page.DisplayAlert` do wyświetlania prostych okien z komunikatami;
- `Xamarin.Forms.Maps` do wyświetlania map (ta funkcjonalność jest zawarta w uniwersalnym pakiecie NuGet).

Aby dowiedzieć się, jak korzystać z powyższych interfejsów, utwórz nową aplikację. W tym celu wykonaj następujące kroki:

1. Utwórz od podstaw w opisany wcześniej sposób nowy projekt oparty na domyślnym szablonie i nadaj mu nazwę `CommonPlatform`.
2. Utwórz nowy folder, a w nim nową klasę o nazwie `MainPage`.
3. W definicji przestrzeni wpisz następujący kod:

```
public class MainPage: ContentPage {
    private Button openUriButton;
    private Button startTimerButton;
    private Button marshalUIThreadButton;
    private Button displayAlertButton;
    private Button displayActionSheetButton;
    private Button openMapButton;
    private StackLayout stackLayout;

    public MainPage() {
        openUriButton = new Button {
            Text = "Otwórz Xamarin Evolve"
        };
        startTimerButton = new Button {
            Text = "Uruchom stoper"
        };
        marshalUIThreadButton = new Button {
            Text = "Uruchom w głównym wątku"
        };
        displayAlertButton = new Button {
            Text = "Wyświetl komunikat"
        };
        displayActionSheetButton = new Button {
            Text = "Wyświetl listę akcji"
        };
        openMapButton = new Button {
            Text = "Otwórz mapę"
        };
    }
}
```



```

openUriButton.Clicked += OpenUriButton_Clicked;
startTimerButton.Clicked += StartTimerButton_Clicked;
marshalUiThreadButton.Clicked += MarshalUiThreadButton_Clicked;
displayAlertButton.Clicked += DisplayAlertButton_Clicked;
displayActionSheetButton.Clicked += DisplayActionSheetButton_Clicked;
openMapButton.Clicked += OpenMapButton_Clicked;

stackLayout = new StackLayout {
    Orientation = StackOrientation.Vertical,
    Spacing = 10,
    Padding = new Thickness(10),
    VerticalOptions = LayoutOptions.FillAndExpand,
    HorizontalOptions = LayoutOptions.FillAndExpand,
    Children = {
        openUriButton,
        startTimerButton,
        marshalUiThreadButton,
        displayAlertButton,
        displayActionSheetButton,
        openMapButton
    }
};
Content = stackLayout;
}

void OpenMapButton_Clicked(object sender, EventArgs e) {
}

async void DisplayActionSheetButton_Clicked(object sender, EventArgs e) {
    string action = await DisplayActionSheet("Prosta lista akcji", "Anuluj", "Usuń",
        new string[] {
            "Akcja 1",
            "Akcja 2",
            "Akcja 3",
        });
    Debug.WriteLine("Kliknięte {0}", action);
}

async void DisplayAlertButton_Clicked(object sender, EventArgs e) {
    bool result = await DisplayAlert("Prosty komunikat", "Super!", "OK", "Anuluj");
    Debug.WriteLine("Kliknięty przycisk: {0}", result ? "OK" : "Anuluj");
}

void MarshalUiThreadButton_Clicked(object sender, EventArgs e) {
    Task.Run(async() => {
        for (int i = 0; i < 3; i++) {
            await Task.Delay(1000);
            Device.BeginInvokeOnMainThread(() => {
                marshalUiThreadButton.Text = string.Format("Wywołany {0}", i);
            });
        }
    });
}
}

```

```

void StartTimerButton_Clicked(object sender, EventArgs e) {
    Device.StartTimer(new TimeSpan(0, 0, 1), () => {
        Debug.WriteLine("Wywołany delegat stopera");
        return true; //false, jeżeli stoper ma być zatrzymany
    });
}

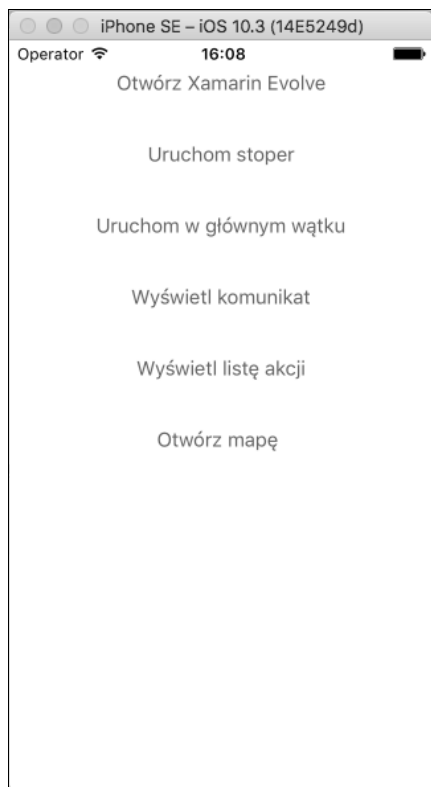
void OpenUriButton_Clicked(object sender, EventArgs e) {
    Device.OpenUri(new Uri("http://xamarin.com/evolve"));
}
}

```

4. Na razie nie zwracaj uwagi na pustą metodę `OpenMapButton_Clicked()`. Aby móc z niej korzystać, za chwilę zmienisz nieco konfigurację rozwiązania.
5. Uruchom aplikację i sprawdź funkcjonowanie wszystkich przycisków z wyjątkiem *Otwórz mapę*, z którym na razie nie jest związana żadna funkcjonalność.

Ekran powinien wyglądać jak na zrzutach.

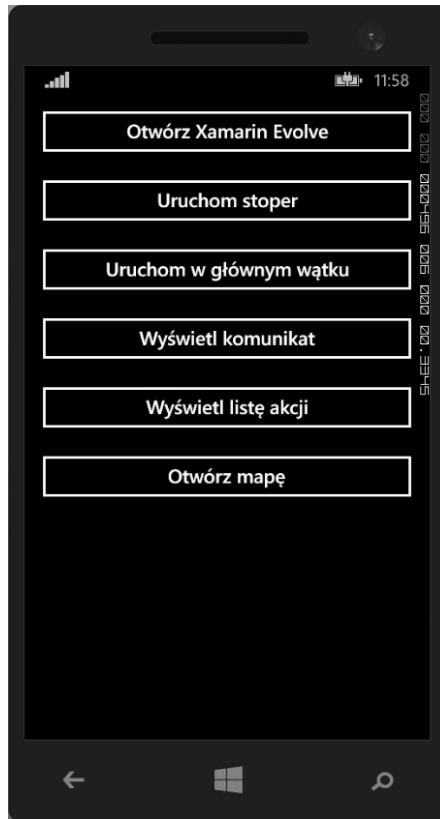
W systemie iOS:



W systemie Android:



W systemie Windows Phone:



Aby móc wyświetlać mapy w aplikacji, musisz w konfiguracji rozwiązania wprowadzić kilka zmian, odpowiednich dla poszczególnych urządzeń. Wykonaj poniższe kroki:

1. Do każdego projektu dodaj pakiet NuGet o nazwie `Xamarin.Forms.Maps`.
2. W projekcie dla Windows Phone w pliku `MainPage.xaml/MainPage.xaml.cs` wpisz pod wywołaniem metody `Forms.Init()` poniższy kod. Analogiczną zmianę wprowadź w projekcie dla Androida w pliku `MainActivity.cs` i w projekcie dla iOS w pliku `AppDelegate.cs`.

```
Xamarin.FormsMaps.Init();
```

3. W projekcie dla Androida zmień powyższy kod, wpisując wymagane argumenty metody:

```
Xamarin.FormsMaps.Init(this, bundle);
```

4. Jeżeli tworzysz aplikację dla systemu iOS 7, nie musisz wprowadzać żadnych dodatkowych zmian. Natomiast w przypadku systemu iOS 8 musisz zmienić plik *info.plist*. Otwórz ten plik, kliknij zakładkę *Źródło* i dodaj do tabeli następujące wiersze:

NSLocationAlwaysUsageDescription	Ciąg	Zawsze musimy znać Twoje położenie!
NSLocationWhenInUseUsageDescription	Ciąg	Musimy znać Twoje położenie, gdy aplikacja jest aktywna!

5. Aby wyświetlać mapy w Androidzie, musisz uzyskać klucz Google Maps API v2. W tym celu otwórz stronę <https://developers.google.com/maps/documentation/android>, kliknij przycisk *Get a key* (uzyskaj klucz) i postępuj według pojawiających się wskazówek.
6. Otwórz plik *Properties/AndroidManifest.xml*, wyświetl jego źródło i w sekcji `<application>` wpisz następujący wiersz:


```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="TWÓJ KLUCZ" />
```
7. Kliknij prawym przyciskiem myszy projekt dla Androida i wybierz polecenie *Opcje*. W oknie, które się pojawi, kliknij po lewej stronie pozycję *Kompilacja/Android Application* i w panelu *Wymagane uprawnienia* zaznacz następujące opcje:
 - Internet
 - AccessNetworkState
 - AccessCoarseLocation
 - AccessFineLocation
 - AccessLocationExtraCommands
 - AccessMockLocation
 - AccessWifiState
8. W projekcie dla Windows Phone otwórz plik *Properties/WMAppmanifest.xml*, kliknij zakładkę *Capabilities* (możliwości) i zaznacz następujące opcje:
 - ID_CAP_MAP
 - ID_CAP_LOCATION
9. Dodaj do rozwiązania nową stronę. W tym celu kliknij prawym przyciskiem folder *Custom Pages*, wybierz polecenie *Dodaj/Nowy plik* i utwórz pustą klasę o nazwie *MapPage*.
10. Zmień klasę *MapPage*, definiując ją jako pochodną klasy *ContentPage*.
11. W definicji klasy wpisz następujący kod:

```
private Map map;
private StackLayout stackLayout;
public MapPage() {
```

```

MapSpan span = MapSpan.FromCenterAndRadius(
    new Position(50.289192, 18.659656),
    Distance.FromMiles(0.4));

map = new Map(span) {
    VerticalOptions = LayoutOptions.FillAndExpand
};

stackLayout = new StackLayout {
    Spacing = 0,
    Children = {
        map
    }
};
Content = stackLayout;
}

```

12. W pliku *MainPage.cs* w definicji metody `OpenMapButton_Clicked()` wpisz następujący kod:

```
Navigation.PushModalAsync(new MapPage());
```

13. Uruchom aplikację, kliknij przycisk *Otwórz mapę* i podziwiaj swoją pracę!

Możesz zmieniać typ mapy, jak również umieszczać na niej znaczniki. Dostępne są następujące rodzaje znaczników:

- Generic (ogólny)
- Place (miejsce)
- SavedPin (zapisany znacznik)
- SearchResult (wynik wyszukiwania)

Aby umieścić znacznik na mapie, wpisz w konstruktorze klasy `MapPage` następujący kod:

```

Position position = new Position(50.289192, 18.659656);
Pin pin = new Pin {
    Type = PinType.Place,
    Position = position,
    Label = "Helion",
    Address = "Helion"
};
map.Pins.Add(pin);

```

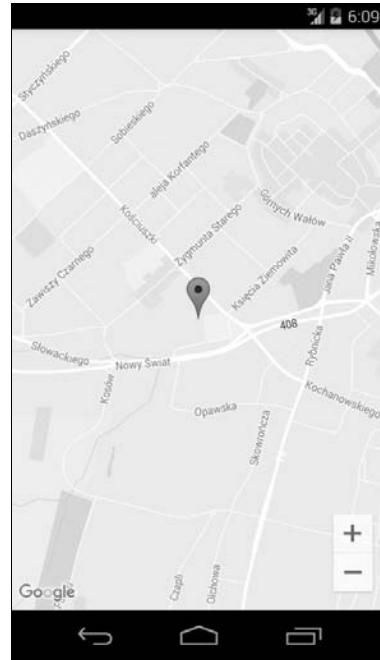
Bardzo proste! Twórcy platformy Xamarin wykonali dobrą robotę, definiując klasy umożliwiające wykonywanie podstawowych operacji na mapach.

Ekran powinien wyglądać jak na zrzutach.

W systemie iOS:



W systemie Android:



W systemie Windows Phone:



Jak to działa?

Platforma Xamarin zawiera klasy, które można stosować we współdzielonej bibliotece PCL, jak również w projektach dla poszczególnych systemów. Mechanizm Service Locator, będący częścią usługi DependencyService (którą zajmiemy się w rozdziale 4., „Różne pojazdy, ten sam silnik”), rozwiązuje zależności w trakcie działania aplikacji, dzięki czemu szybko można tworzyć aplikacje wykorzystujące najczęściej stosowane funkcjonalności.

Metoda `DisplayAlert()` wstrzykuje w trakcie działania aplikacji kod klasy `UIAlertController` w iOS, kod klasy `AlertDialog` w Androidzie oraz kod klasy `MessageBox` w Windows Phone. Metoda `Device.StartTimer()` ukrywa szczegóły odpowiednich klas, charakterystycznych dla poszczególnych systemów.

Ta sama zasada jest stosowana we wszystkich opisanych wcześniej metodach, a także w metodach pakietu `Xamarin.Forms.Maps`, jednak w ostatnim przypadku wymagane jest wprowadzenie pewnych zmian w konfiguracjach projektów.

Platformy Xamarin, a szczególnie `Xamarin.Forms`, powstały po to, aby można było efektywniej tworzyć aplikacje. Temu celowi służą wtyczki Xamarin i Community. Zanim zaczniesz kodować, sprawdź ich zawartość, aby nie wyważać otwartych drzwi!

Zobacz też

- Rozdział 4., „Różne pojazdy, ten sam silnik”
- <https://developer.xamarin.com/guides/xamarin-forms/user-interface/controls/layouts>
- <https://components.xamarin.com>

Uwierzytelnianie użytkowników za pomocą serwisów Facebook i Google

Jedną z najczęściej wykorzystywanych w aplikacjach funkcjonalności jest uwierzytelnianie użytkownika za pomocą jego ulubionego serwisu. Ostatnim tematem tego rozdziału jest implementacja tej funkcjonalności z użyciem wtyczki `Xamarin.Auth`. Wtyczka ta umożliwia połączenie się aplikacji i uwierzytelnienie użytkownika poprzez jeden z dwóch popularnych serwisów — Facebook lub Google — dzięki czemu użytkownik nie musi rejestrować się w aplikacji. Ponadto programista oszczędza sobie dodatkowej pracy związanej z implementacją procedury logowania na urządzeniu i serwerze.

Jak to zrobić?

1. Utwórz w środowisku Xamarin Studio lub Visual Studio nowe uniwersalne rozwiązanie o nazwie `AuthenticateProviders`. Jeżeli nie pamiętasz, jak to się robi, zajrzyj do części „Tworzenie uniwersalnego rozwiązania” na początku tego rozdziału.
2. Do projektów dla systemów iOS i Android (projektem dla Windows Phone na razie się nie zajmujemy) dodaj komponent uwierzytelniający. W tym celu kliknij prawym przyciskiem myszy folder `AuthenticateProviders.iOS/Składniki` i wybierz polecenie *Edytuj składniki*. W oknie, które się pojawi, wyszukaj komponent `Xamarin.Auth` i dodaj go do projektu. Tę samą operację wykonaj w przypadku projektu dla Androida.
3. Aby można było uwierzytelniać użytkowników za pomocą serwisów Facebook i Google, musisz w sekcjach deweloperskich obu serwisów uzyskać własne identyfikatory klienckie. W tym celu otwórz strony <https://developers.facebook.com> oraz <https://console.developers.google.com> i postępuj według znajdujących się na nich wskazówek.
4. Proces uwierzytelniania użytkownika w obu serwisach i w obu systemach wygląda niemal tak samo: tworzony jest nowy projekt i uzyskiwane są klucze. Więcej informacji na ten temat znajdziesz w opisach poszczególnych systemów. Najważniejszą kwestią jest zdefiniowanie takiego samego przekierowującego adresu URL, aby aplikacja po pomyślnym uwierzytelnieniu użytkownika otwierała odpowiednią stronę.
5. Kliknij prawym przyciskiem myszy główny projekt `AuthenticateProviders`, wybierz polecenie *Dodaj/Nowy folder* i utwórz folder o nazwie `Models`.
6. W nowo utworzonym folderze utwórz dwie klasy pomocnicze o nazwach `OAuthSettings` i `ProviderManager`.
7. W pliku `OAuthSettings.cs` wpisz następujący kod:

```
public class OAuthSettings {
    public string ClientId {
        get;
        set;
    }
    public string ClientSecret {
        get;
        set;
    }
    public string AuthorizeUrl {
        get;
        set;
    }
    public string RedirectUrl {
        get;
        set;
    }
    public string AccessTokenUrl {
        get;
        set;
    }
}
```



```

    }
    public List<string> Scopes {
        get;
        set;
    }
    public string ScopesString {
        get {
            return Scopes.Aggregate((current, next) =>
                string.Format("{0}+{1}", current, next));
        }
    }
    public OAuthSettings() {
        Scopes = new List<string>();
    }
}

```

8. W pliku *ProviderManager.cs* wpisz następujący kod:

```

public enum Provider {
    Unknown = 0,
    Facebook,
    Google
}

public static class ProviderManager {
    private static OAuthSettings FacebookOAuthSettings {
        get {
            return new OAuthSettings {
                ClientId = "TWÓJ_ID_KLIENCKI",
                ClientSecret = "DWAJEDY_HASŁO_KLIENCKIE",
                AuthorizeUrl = "https://m.facebook.com/dialog/oauth/",
                RedirectUrl = "http://www.facebook.com/connect/login_success.html",
                AccessTokenUrl = "https://graph.facebook.com/v2.3/oauth/access_token",
                Scopes = new List<string> {
                    ""
                }
            };
        }
    }
    private static OAuthSettings GoogleOAuthSettings {
        get {
            return new OAuthSettings {
                ClientId = "TWÓJ_ID_KLIENCKI",
                ClientSecret = "DWAJEDY_HASŁO_KLIENCKIE",
                AuthorizeUrl = "https://accounts.google.com/o/oauth2/auth",
                RedirectUrl = "https://www.googleapis.com/plus/v1/people/me",
                AccessTokenUrl = "https://accounts.google.com/o/oauth2/token",
                Scopes = new List<string> {
                    "openid"
                }
            };
        }
    }
}

```

```

public static OAuthSettings GetProviderOAuthSettings(Provider provider) {
    switch (provider) {
        case Provider.Facebook:
            {
                return FacebookOAuthSettings;
            }
        case Provider.Google:
            {
                return GoogleOAuthSettings;
            }
    }
    return new OAuthSettings();
}
}

```

9. Kliknij prawym przyciskiem myszy projekt *AuthenticateProviders*, wybierz polecenie *Dodaj/Nowy folder* i utwórz folder o nazwie *Custom Pages*.
10. Kliknij prawym przyciskiem myszy nowo utworzony folder i utwórz w nim dwie klasy o nazwach *LoginPage* i *ProvidersAuthPage*. W pliku *LoginPage.cs* wpisz następujący kod:

```

public class LoginPage : ContentPage {
    public OAuthSettings ProviderOAuthSettings { get; set; }
    public LoginPage(Provider provider) {
        ProviderOAuthSettings = ProviderManager.GetProviderOAuthSettings(provider);
    }
}

```

11. W pliku *ProvidersAuthPage.cs* wpisz następujący kod:

```

public class ProvidersAuthPage : ContentPage {
    StackLayout stackLayout;
    Button facebookButton;
    Button googleButton;
    public ProvidersAuthPage() {
        facebookButton = new Button {
            Text = "Facebook"
        };
        facebookButton.Clicked += async(sender, e) =>
            await Navigation.PushModalAsync(new LoginPage(Provider.Facebook));
        googleButton = new Button {
            Text = "Google"
        };
        googleButton.Clicked += async(sender, e) =>
            await Navigation.PushModalAsync(new LoginPage(Provider.Google));
        stackLayout = new StackLayout {
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center,
            Orientation = StackOrientation.Vertical,
            Spacing = 10,
            Children = {
                facebookButton,

```

```

        googleButton
    }
};
this.Content = stackLayout;
}
}

```

12. W projekcie *AuthenticateProviders* w pliku *App.xaml/App.xaml* wpisz w konstruktorze klasy następujący kod:

```
MainPage = new ProvidersAuthPage();
```

Teraz musisz zakodować proces uwierzytelniania użytkownika. W punkcie 3. dodałeś do dwóch projektów komponent *Xamarin.Auth*, który w zależności od systemu wykorzystuje się inaczej. Aby w różnych systemach był wykonywany inny kod, zastosujesz klasę *PageRenderer* i atrybut przestrzeni, dzięki którym wykorzystana zostanie usługa *DependencyService* oferowana przez platformę *Xamarin.Forms*. Więcej informacji na ten temat znajdziesz w rozdziale 2., „Raz deklaruji, wszędzie wizualizuję”.

W projekcie dla systemu Android wykonaj następujące operacje:

1. Utwórz nowy folder o nazwie *Platform Specific*, a w nim nową klasę *LoginPageRenderer*, dziedziczącą właściwości klasy *PageRenderer*. W definicji klasy wpisz następujący kod:

```

LoginPage page;
bool loginInProgress;

protected override void OnElementChanged(ElementChangedEventArgs<Page> e) {
    base.OnElementChanged(e);
    if (e.OldElement != null || Element == null)
        return;
    page = e.NewElement as LoginPage;
    if (page == null || loginInProgress)
        return;
    loginInProgress = true;
    try {
        // Twój identyfikator kliencki OAuth2
        OAuth2Authenticator auth = new OAuth2Authenticator(
            page.ProviderOAuthSettings.ClientId,
            // Twoje hasło klienckie OAuth2
            page.ProviderOAuthSettings.ClientSecret,
            // Zakresy
            page.ProviderOAuthSettings.ScopesString,
            // Zakresy rozdzielone znakiem "+"
            new Uri(page.ProviderOAuthSettings.AuthorizeUrl),
            // Przekierowujący adres URL
            new Uri(page.ProviderOAuthSettings.RedirectUrl),
            new Uri(page.ProviderOAuthSettings.AccessTokenUrl)
        );
        auth.AllowCancel = true;
        auth.Completed += async(sender, args) => {
            // Wykonywane operacje
            await page.Navigation.PopAsync();
        }
    }
}

```

```

        loginInProgress = false;
    };
    auth.Error += (sender, args) => {
        Console.WriteLine("Błąd uwierzytelnienia: {0}", args.Exception);
    };
    var activity = Xamarin.Forms.Forms.Context as Activity;
    activity.StartActivity(auth.GetUI(Xamarin.Forms.Forms.Context));
} catch (Exception ex) {
    Console.WriteLine(ex);
}
}
}

```

2. Wpisz poniższe instrukcje i atrybuty dekoracyjne przestrzeni nazw, aby kod poprawnie się skompilował:

```

using XamFormsAuthenticateProviders;
using XamFormsAuthenticateProviders.Droid;
using Xamarin.Forms.Platform.Android;
using Xamarin.Auth;
[assembly: ExportRenderer(typeof(LoginPage), typeof(LoginPageRenderer))]
namespace AuthenticateProviders.Droid

```

W projekcie dla systemu iOS wykonaj następujące operacje:

1. Jak poprzednio, utwórz nowy folder *Platform Specific*, a w nim klasę *LoginPageRenderer* pochodną od klasy *PageRenderer*.
2. W definicji klasy wpisz następujący kod, odpowiedni dla iOS:

```

LoginPage page;
bool loginInProgress;

protected override void OnElementChanged(VisualElementChangedEventArgs e) {
    base.OnElementChanged(e);
    if (e.OldElement != null || Element == null)
        return;
    page = e.NewElement as LoginPage;
}

public override async void ViewDidAppear(bool animated) {
    base.ViewDidAppear(animated);
    if (page == null || loginInProgress)
        return;
    loginInProgress = true;
    try {
        // Twój identyfikator kliencki OAuth2
        OAuth2Authenticator auth = new OAuth2Authenticator(
            page.ProviderOAuthSettings.ClientId,
            // Twoje hasło klienckie OAuth2
            page.ProviderOAuthSettings.ClientSecret,
            // Zakresy
            page.ProviderOAuthSettings.ScopesString,
            // Zakresy rozdzielone znakiem "+"
            new Uri(page.ProviderOAuthSettings.AuthorizeUrl),

```

```

//Przekierowujący adres URL
new Uri(page.ProviderOAuthSettings.RedirectUrl),
new Uri(page.ProviderOAuthSettings.AccessTokenUrl)
);
auth.AllowCancel = true;
auth.Completed += async(sender, args) => {
    //Wykonywane operacje
    await DismissViewControllerAsync(true);
    await page.Navigation.PopModalAsync();
    loginInProgress = false;
};
auth.Error += (sender, args) => {
    Console.WriteLine("Błąd uwierzytelnienia: {0}", args.Exception);
};
await PresentViewControllerAsync(auth.GetUI(), true);
} catch (Exception ex) {
    Console.WriteLine(ex);
}
}
}

```

3. Wpisz jeszcze poniższe instrukcje i atrybut ExportRenderer przestrzeni nazw:

```

using System;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using Xamarin.Auth;
using AuthenticateProviders;
using AuthenticateProviders.iOS;

[assembly: ExportRenderer(typeof(LoginPage), typeof(LoginPageRenderer))]
namespace AuthenticateProviders.iOS

```

A co z pominiętym Windows Phone? Na pewno zauważyłeś, że w serwisie Xamarin Component Store nie ma komponentu uwierzytelniającego przeznaczonego dla tego systemu. Nie przejmuj się tym jednak, istnieją inne możliwości. Jeden z magicznych sposobów polega na wykorzystaniu klasy `PageRenderer`, zastosowanej w dwóch poprzednich projektach. W projekcie dla systemu Windows Phone utwórz w opisany wyżej sposób klasę i wpisz własny kod uwierzytelniający. Możesz również wykonać następujące operacje i wypróbować eksperymentalny pakiet NuGet `Xamarin.Auth`:

1. Kliknij prawym przyciskiem myszy projekt dla systemu Windows Phone i dodaj do niego pakiet NuGet `Xamarin.Auth`.
2. Utwórz folder *Platform Specific*, a w nim klasę `LoginPageRenderer` pochodną od klasy `PageRenderer`.
3. W definicji klasy zaimplementuj własny algorytm uwierzytelniający lub wykorzystaj pakiet `Xamarin.Auth`.

Aby utworzyć gotową aplikację, skorzystaj z dołączonego do książki kodu.

Poczyniłeś duże postępy. Oczywiście w tym rozwiązaniu został wykorzystany zaledwie drobny fragment możliwości tokenów uwierzytelniających, ale ten temat zostawiam Tobie. Zazwyczaj proces uwierzytelnienia polega na uzyskaniu tokenu, zarejestrowaniu jego daty wygaśnięcia i zapisaniu go w pamięci.

Jak to działa?

Poznałeś zatem potęgę platformy Xamarin.Forms, umożliwiającą tworzenie uniwersalnych aplikacji wykorzystujących współdzielony kod, który można dostosowywać odpowiednio do potrzeb. Dzięki klasie `PageRenderer` można wyświetlać widoki i strony w sposób odpowiedni dla różnych systemów operacyjnych.

Pierwszą rzeczą, jaką zrobiłeś po utworzeniu rozwiązania, było zalogowanie się do konsoli deweloperskiej serwisów Facebook i Google w celu utworzenia nowego projektu, uaktywnienia mechanizmu uwierzytelniającego OAuth2 oraz uzyskania kluczy i przekierowujących adresów URL stron otwieranych po pomyślnym zalogowaniu użytkownika. Mechanizmy OAuth i OAuth2 służą do sprawdzania, czy logujący się użytkownik jest zarejestrowany w bazie serwisu uwierzytelniającego. Dzięki temu użytkownik w celu zalogowania się do aplikacji może używać tych samych poświadczeń, które stosuje do logowania się do serwisu.

Kolejną operacją było utworzenie dwóch klas pomocniczych. Pierwsza z nich, `OAuthSettings`, zawiera wszystkie ustawienia wymagane przez serwis uwierzytelniający do otwarcia dostępu do aplikacji. Druga, statyczna klasa `ProviderManager`, zawiera statyczną metodę, której argumentem jest identyfikator serwisu uwierzytelniającego. Metoda ta zwraca instancję klasy `OAuthSettings` odpowiednią dla wskazanego serwisu.

Uniwersalny interfejs użytkownika składa się z dwóch stron. Pierwsza to strona główna, `ProvidersAuthPage`, która zawiera dwa przyciski. Po kliknięciu dowolnego z tych przycisków wywoływana jest metoda obsługi zdarzenia `Clicked` i następuje przejście do strony `LoginPage`, uwierzytelniającej użytkownika za pomocą wybranego serwisu.

W konstruktorze klasy `LoginPage` argument `provider` jest wykorzystywany do uzyskania odpowiedniej instancji klasy `OAuthSettings` i zapisania jej w publicznej właściwości. To wszystko. Cała reszta jest wykonywana przez kod przystosowany do właściwego systemu operacyjnego.

Kody klasy `LoginPageRenderer` są dostosowane do poszczególnych systemów i wykorzystują funkcjonalności platformy Xamarin.Forms. W iOS jest to klasa `UIViewController`, w Androidzie klasa bardzo podobna do klasy `Activity`, a w Windows Phone klasa `PhoneApplicationPage`. W platformie dostępnych jest kilka innych klas do wyświetlania różnego typu widoków.

Przestrzeń nazw, w której zdefiniowałeś powyższą klasę, opatrzyłeś atrybutem `ExportRenderer`, dzięki któremu platforma Xamarin.Forms podczas działania aplikacji ładuje do pamięci odpowiedni kod.

W klasie `LoginPageRenderer` nadpisywana jest metoda `OnElementChanged()` i uzyskiwana strona platformy `Xamarin.Forms`. W iOS wykorzystywana jest w tym celu metoda `ViewDidAppear()`. Jednak w Androidzie nie istnieje analogiczna metoda, dlatego w metodzie `OnElementChanged()` wpisałeś własny kod uwierzytelniający. Kod ten sprawdza, czy proces uwierzytelniania użytkownika cały czas trwa, gdyby zamknął on stronę logowania i otworzył ją ponownie.

Wtyczka `Xamarin.Auth` zawiera klasę `OAuth2Authenticator`, której instancja jest tworzona z wykorzystaniem informacji zawartych w instancji klasy `OAuthSettings`, zapisanej we właściwości `ProviderOAuthSettings`. W ten sposób wykorzystywane są najlepsze cechy obu klas.

W dalszej części kodu rejestrowane są trzy metody wywoływane po pomyślnym uwierzytelnieniu użytkownika, po przerwaniu procesu uwierzytelnienia oraz po pojawieniu się błędów. Za pomocą metody `Auth.GetUI()` uzyskiwany jest obiekt właściwy danemu systemowi operacyjnemu, wykorzystywany do wyświetlenia widoku uwierzytelniającego. Na koniec, w kodzie obsługi zdarzenia `Completed`, zamykany jest widok uwierzytelniający.

Zobacz też

- Rozdział 2., „Raz deklaruje, wszędzie wizualizuj”
- Rozdział 4., „Różne pojazdy, ten sam silnik”
- <https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/custom-renderer>
- <https://components.xamarin.com/view/xamarin.auth>

Skorowidz

A

- adres URL, 58
- animacje, 249
- AOP, aspect-oriented programming, 109
- aparatur fotograficzny, 95
- aplikacje
 - lokalne, 147
 - mieszane, 147
 - natywne, 28
 - sieciowe, 147
- atrybut PrimaryKey, 157

B

- baza danych SQLite, 147, 148
- BCL, Base Class Libraries, 19
- biblioteka
 - Calabash, 267
 - CLLocationManager, 195
 - Flurl, 175
 - Fusillade, 175
 - LocationManager, 195
 - NSURLSession, 174
 - OkHttp, 174
 - PCL, 43, 54, 152
 - Picasso, 175
 - Plugin.XamFormsCrossPlugin.Abstractions, 182
 - Plugin.XamFormsCrossPlugin.WindowsPhone8, 182
 - Refit, 175

- Xamarin.Forms.Core, 28
- Xamarin.Forms.Platform.iOS, 28
- Xamarin.Forms.Xaml, 28
- bloki funkcjonalne, 61, 67

C

- CI, Continuous Integration, 288
- CRUD, create, read, update, delete, 148

D

- definiowanie
 - detektorów gestów, 240
 - interfejsu użytkownika, 160
 - testów, 279
 - akceptacyjnych, 262
 - jednostkowych, 256
 - uniwersalnych animacji, 249
- detektor gestów, 240
 - TapGestureRecognizer, 242
- dostęp do bazy danych, 148
- dostosowywanie szablonu wiersza, 228
- dwukierunkowe powiązanie danych, 210

E

- efekty specjalne, 239
- ekran logowania, 29
- element typu Character, 226

F

Facebook, 43
 filmy, 185
 format JSON, 161
 funkcjonalności systemów, 35

G

gesty, 91, 240
 globalizowanie aplikacji, 138, 144
 Google, 43
 GPS, 190
 grupowanie elementów, 233

I

informacje o użytkownikach, 301
 instrukcja public, 156
 interfejs

- API, 36
- IDataService, 125
- IMarkupExtension, 142
- INotifyCollectionChanged, 226
- INotifyPropertyChanged, 207, 210, 215, 226
- IPlatform, 119
- IRepository, 160
- ISQLiteConnection, 151
- ISQLitePlatform, 154
- IValueConverter, 215
- IxamFormsCrossPlugin, 183
- IXamFormsCrossPlugin, 180
- użytkownika, 50
- zakładkowy, 54

 IoC, Inverse of Control, 122

J

język XAML, 240
 JIT, Just in Time, 315

K

klasa

- Android.Views.GestureDetector, 248
- AndroidAppConfigurator, 269
- AppResources, 144
- Assert, 261

AVCaptureDevice, 107
 AVCaptureDeviceInput, 107
 Behavior<T>, 69
 BoxView, 91
 CameraCaptureTask, 105, 107
 Character, 218, 226
 ContactDetailPage, 161
 ContactDetailsViewModel, 134
 ContactViewModel, 131
 ContentPage, 40
 Context, 296
 CustomBoxView, 94
 CustomBoxViewRenderer, 92
 DataService, 134, 167, 174
 DateTimeToStringConverter, 213
 EmailValidatorBehavior, 68
 EntryRenderer, 76
 GeolocatorImplementation, 192, 195
 GestureImage, 248
 GestureImageDroidRenderer, 244
 GestureImageListener, 248
 GestureImagePhoneRenderer, 247
 GroupingObservableCollection<K, T>, 237
 ImageCell, 231
 Insights, 301
 iOSAppConfigurator, 269
 ListView, 226
 Localize, 140
 LocalNotificationsImplementation, 199
 LoginPageRenderer, 47, 50
 MainActivity, 125
 MainPage, 34
 MainPageRenderer, 83
 MessageBox, 43
 MessagingCenter, 138
 NameService, 111
 NativeMessageHandler, 168
 ObservableCollection, 237
 OnPlatform<T>, 74
 PageRenderer, 47, 81, 90, 98
 PhoneApplicationPage, 27, 29
 Rectangle, 252
 Repository, 160
 SettingsTouch, 125
 SimpleOnGestureListener, 92, 243
 SQLiteAsyncConnection, 154
 SQLiteConnectionDroid, 151
 StoreCameraMediaOptions, 190

TabbedPage, 55, 60
 Task, 227
 TextureView.ISurfaceTextureListener, 106
 TouchRunner, 258
 TranslateExtension, 145
 TriggerAction<T>, 64
 UIAlertController, 43
 UIViewController, 50
 View, 79
 ViewModelLocator, 131
 VisualElement, 90
 WindowsPhoneControl, 86, 90
 Xamarin.Forms.Application, 29
 Xamarin.Forms.Platform.iOS.FormsApplicationDelegate, 29
 Xamarin.FormsCrossPluginImplementation, 180
 kod uniwersalny, 110
 kodowanie bloków funkcjonalnych, 61
 kolekcja, 217
 komórka

- EntryCell, 228
- ImageCell, 228
- SwitchCell, 228
- TextCell, 228

 kompilacja

- JIT, 315
- warunkowa, 117, 266, 272

 komunikator zdarzeń, 136
 kontener

- NinjectContainer, 125
- XLabs.IoC, 124

 konto Xamarin, 294
 kontrolka, 75

- aparatu fotograficznego, 95
- CGContext, 91
- CustomBoxView, 91
- CustomEntry, 76
- Entry, 75
- Image, 232
- ImageCell, 231
- Label, 202
- ListView, 157, 217, 231
- ViewCell, 231
- ViewGroup, 91

 kontrolki urządzeń, 81
 konwerter wartości, 211
 kucz podstawowy, 157

L

lista, 217

- nawigacyjna, 233
- odświeżanie elementów, 222
- tworzenie, 222
- usuwanie elementów, 222

 lokalizator zależności, 118
 lokalne powiadomienia, 195, 197

M

mapa, 39
 mapowanie obiektowo-relacyjne, 148
 mechanizm uwierzytelniający OAuth2, 50
 metoda

- app.Repl(), 273
- AuthorizeCameraUseAsync(), 102
- BeforeEachTest(), 278
- ConfigureAwait(), 227
- Convert(), 216
- ConvertBack(), 216
- Device.BeginInvokeOnMainThread(), 227
- DisplayAlert(), 43
- DummyTest(), 277
- FadeTo(), 252
- FinishedLaunching(), 132, 193, 267
- Forms.Init(), 186, 192
- GetGreeting(), 112
- global
 - Xamarin.Forms.Init(), 199
 - HttpClient.GetAsync(), 174
 - ILocalNotifications.Cancel(), 198
 - ILocalNotifications.Show(), 198
 - IMedia.takePhotoAsync(), 189
 - InsertItemAsync(), 175
 - InsertOrdersAsync(), 172
 - IsPickPhotoSupported(), 190
 - LayoutTo(), 252
 - OnAppearing(), 72
 - OnAttachedTo(), 69
 - OnCreate(), 132, 192, 266
 - OnElementChanged(), 51, 89, 97
 - OnLayout(), 98
 - OnTakePhoto(), 97
 - OnToolbarClick(), 226, 227
 - OpenMapButton_Clicked(), 38
 - Page.OnButtonPressed(), 90

metoda
 PrepareAndStartCamera(), 98
 PropertyChangedEventHandler(), 206
 Raise(), 210
 RegisterViews(), 131
 RotateTo(), 252
 ScaleTo(), 252
 SetFieldAndRaise<T>, 210
 Should_Reverse_Word(), 258
 UpdateAsync(), 167

modyfikator

async, 227
 await, 227
 static, 211

MVVM, Model-View-ViewModel, 127

N

nagrywanie filmów, 185

narzędzie

ProGuard, 310
 Windows App Certification Kit, 320

natywna kontrolka aparatu fotograficznego, 95

natywne

kontrolki urządzeń, 81
 wizualizatory, 242

O

obiekt typu

Bundle, 29
 Contact, 128
 MainActivity, 29
 StackLayout, 34

obsługa gestów, 91, 242

odbiornik GPS, 190

odczytywanie danych GPS, 190

odwracanie sterowania, 122

Okunevic Stan, 11

operacje CRUD, 147, 148, 155

optymalizowanie kodu, 319

ORM, Object-Relational Mapping, 148

P

pakiet

Microsoft.Net.Http, 162, 166, 168
 modernhttpclient, 168
 Newtonsoft.Json, 162, 168

NuGet, 39
 Punchclock, 168
 SQLite.Net.Async-PCL, 148
 SQLite.Net-PCL, 148, 154
 Xam.Plugin.Media, 185, 189
 Xam.Plugins.Notifier, 196
 Xamarin Insights, 296
 Xamarin.Forms, 186
 Xamarin.Forms.Maps, 43
 Xamarin.Insights.Signed, 295
 Xamarin.UITest, 264

panel Test Cloud, 285

PCL, Portable Class Library, 109

platforma

NUnit, 261
 NUnitLite, 261
 Xamarin.Forms, 20, 308
 Xamarin.UITest, 262, 268

plik

App.cs, 70
 App.xaml.cs, 30, 34
 AppDelegate.cs, 29, 39, 193, 196
 AppResources.pl.resx, 139
 Basketball.xaml, 58
 ContactDetailPage.xaml, 157
 ContactDetailsPage.xaml.cs, 158
 ContactListPage.xaml.cs, 134
 ContactViewModel.cs, 129
 CrossXamFormsCrossPlugin.cs, 184
 DataService.cs, 128, 169
 DetailsPage.xaml.cs, 282
 FormsTabPage.xaml, 55, 56
 GestureImageListener.cs, 243
 GroupingObservableCollection.cs, 235
 InAppCameraPage.cs, 102
 InAppCameraPageRenderer.cs, 96
 Info.plist, 141, 304
 LoginPage.xaml, 68
 MainActivity.cs, 125
 MainPage.cs, 41
 MainPage.xaml, 169, 229
 MainPage.xaml.cs, 27, 124, 137, 142, 164, 172, 214
 MainPageRenderer.cs, 85
 NameService.cs, 115
 OAuthSettings.cs, 44
 OrderDetailsPage.xaml, 163
 Person.cs, 207
 PhotoPage.xaml, 58

PreviewImageUserControl.xaml, 104
 Properties/WMAppmanifest.xml, 40
 ProviderManager.cs, 45
 ReverseServiceTests.cs, 257
 SQLiteConnectionTouch.cs, 151
 Tests.cs, 283
 WMAppManifest.xml, 186
 XamFormsCrossPluginImplementation.cs,
 184
 XamFormsInsights.cs, 296, 298
 pliki
 .dex, 315
 dSYM, 300
 XAML, 70
 plugin for Xamarin templates, 179
 polecenie
 app, 274
 copy, 277
 Manage NuGet Packages, 162, 168
 tree, 274
 powiadomienia, 195
 program MSTest, 261
 programowanie aspektowe, 109
 projekt
 typu Blank Xaml App, 70
 UnitTestCore, 257
 współdzielony, 109
 XamFormsDependencyInjection.Droid, 124
 XamFormsReplTest.UITests, 282
 XamFormsTestCloud.Droid, 283
 XamFormsUnitTesting, 257
 przesyłanie definicji testów, 279
 publikowanie aplikacji
 dla systemu Android, 307
 dla systemu iOS, 302
 dla Windows Phone, 316

R

REST, Representational State Transfer, 161
 robienie zdjęć, 185

S

selektor this, 279
 serwer
 Xamarin Test Cloud Server, 267
 GitHub, 190

strumień danych, 217
 systemy CI, 288
 szablon
 Android Layout, 83
 Cross Platform App, 110, 119, 148, 161, 191,
 316
 Forms Blank Content Page Xaml, 163, 202,
 219
 Interface, 119, 149, 155
 Resources File, 139
 Text File, 113
 Unit Test App, 258
 wiersza, 228
 Windows Phone User Control, 86

Ś

śledzenie
 użytkowników, 301
 zdarzeń, 301
 środowisko
 Visual Studio, 24
 Xamarin Studio, 21

T

Taskos George, 9
 TDD, test-driven development, 256
 technika Bait and Switch, 178
 terminal Xamarin.UITest REPL, 269
 testowanie, 255
 interfejsu użytkownika, 269
 testy
 akceptacyjne, 262, 269
 jednostkowe, 256
 tworzenie
 animacji, 249
 listy, 222
 listy nawigacyjnej, 233
 testów akceptacyjnych, 269
 uniwersalnego
 ekranu logowania, 29
 kodu, 109, 110
 rozwiązania, 20
 uniwersalnej aplikacji, 54
 uniwersalnych wtyczek, 178

U

układ

- AbsoluteLayout, 253
- StackLayout, 232, 252

uniwersalna aplikacja, 54

uniwersalne animacje, 249

uniwersalny

- ekran logowania, 29
- interfejs użytkownika, 50, 177

usługa

- DependencyService, 114, 115, 120, 121
- REST, 147, 161, 166
- Xamarin Insights, 292, 299
- Xamarin Test Cloud, 256, 279

ustawienia językowe, 142

usuwanie elementów, 226

uwierzytelnianie użytkowników, 43

V

Visual Studio, 24

W

weryfikowanie poprawności danych, 67

wiązanie danych, 201

- dwukierunkowe, 206
- w kodzie C#, 202
- w kodzie XAML, 204

widok

- loginButton, 34
- passwordEntry, 34
- userNameEntry, 34

wizualizator, 82

właściwość

- antecedent.Result, 174
- BackgroundColor, 68
- BindableProperty, 82
- BindingContext, 206, 221
- binding, 204
- Characters, 222, 228, 234
- Clicked, 34
- Command, 242
- CommandParameter, 242
- Content, 34
- ContentDescription, 268
- Culture, 144

DateOrdered, 215

DateTime, 211

DisableCollectionTypes, 301

DisableDataTransmission, 301

Footer, 232

GestureRecognizers, 242, 249

GroupDisplayBinding, 237

GroupShortNameBinding, 237

Header, 232

HeaderTemplate, 233

Height, 35

HorizontalOptions, 34

IList<ContactViewModel>, 134

Implementation.Value, 183

IsDestructive, 226

IsFocused, 62

IsGroupingEnabled, 237

IsPullToRefreshEnabled, 227

IsRefreshing, 227

ItemsSource, 231

ItemTappedEventArgs.Item, 221

Label.Text, 233

NavigationProperty.Child, 204

NumberOfTapsRequired, 242

ObjectId, 167

OrderNumber, 166

Orientation, 34

RowHeight, 232

SelectedItem, 221

Spacing, 34

StyleId, 268

Text, 231

UIViewController, 258

VerticalOptions, 34

Width, 35

WPF, Windows Presentation Foundation, 53

wstrzykiwanie zależności, 109, 122, 154

wtyczka

obsługująca gesty, 242

uniwersalna, 178

Xam.Plugins.Notifier, 198

Xamarin, 185

wygląd kontrolki, 75

wyjątek

DivideByZeroException, 299

NotImplementedException, 183

NotSupportedException, 137

wykonywanie zdjęć, 95

wysyłanie
 lokalnych powiadomień, 195
 zapytań, 168

wyświetlanie
 danych, 221
 kolekcji danych, 218
 lokalnych powiadomień, 195
 mapy, 39
 natywnych stron, 82

wyzwalacz, 61
 DataTrigger, 61, 63, 68
 EventTrigger, 61, 69
 MultiTrigger, 61, 65, 68
 PropertyTrigger, 61, 68

wzorzec projektowy MVVM, 127

X

Xamarin Android Player, 24
 Xamarin Insights, 292
 Xamarin Studio, 54
 Xamarin Studio Community, 21
 Xamarin.Forms, 19

Z

zaznaczanie wiersza listy, 218

zdarzenie
 ButtonPressed, 82
 ItemTapped, 221
 ListView.ItemTapped, 221
 ManipulationCompleted, 247
 OnAddContactClick, 265
 OnDelete, 223
 OnDetailsClick, 282
 OnImageTapped, 241
 OnLogInClick, 272
 OnPinchUpdated, 241
 OnRefreshing, 223
 OnSaveClick, 265
 OnTakePhotoButtonClicked, 189
 OnToolbarClick, 223
 PropertyChanged, 89, 136, 210
 TextChanged, 69
 TouchUpInside, 100
 Xamarin.Forms.Forms.ViewInitialized, 268

zdjęcia, 95, 185
 zmiana wyglądu kontrolki, 75

znacznik
 BoxView.WidthRequest, 75
 Button, 58, 63
 ContentPage, 187
 ContentPage.BackgroundColor, 71
 ContentPage.Resources, 62
 DataTemplate, 226, 237
 DataTrigger, 69
 GroupHeaderTemplate, 237
 Image, 58
 ImageCell, 231
 Label, 70
 MenuItem, 225
 NavigationPage, 60
 OnPlatform, 74
 ResourceDictionary, 66
 TabbedPage.Children., 60
 TextCell, 226
 TextCell.ContextActions, 226

zrzut ekranu, 287

Ż

źródło danych, 204

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

Xamarin

Tworzenie aplikacji cross-platform

Receptury

Platforma Xamarin.Forms jest wszechstronnym narzędziem dla nowoczesnych programistów. Umożliwia budowanie aplikacji w języku C#, włączając w to interfejsy graficzne na urządzenia macOS, Android i Windows Phone. Ułatwia budowanie niestandardowych widoków, układów i kontrolki. Xamarin pozwala również na tworzenie własnych uniwersalnych wtyczek i udostępnianie ich w systemie NuGet. Obecnie Xamarin.Forms zdobywa coraz większą popularność. W wielu dużych firmach stanowi standard budowy oprogramowania na urządzenia mobilne.

W tej książce opisano zasady programowania aspektowego przy tworzeniu architektury aplikacji, która działa efektywnie na każdej platformie i korzysta z wbudowanego lokalizatora usług. Przedstawiono dobre praktyki tworzenia i dostosowywania kontrolki Xamarin.Forms ListView, grupowania elementów, list szybkiego dostępu i niestandardowych komórek. Opisano również procedury testowania interfejsu użytkownika, zarówno lokalnie, jak i za pomocą Xamarin Test Cloud. Czytelnik dowie się również, w jaki sposób monitorować aplikację za pomocą usługi Xamarin Insights, a także jak przygotować aplikację do udostępnienia i umieścić ją w sklepie internetowym.

Xamarin.Forms — twórz aplikacje idealne dla urządzeń mobilnych!



W książce między innymi:

- rozpoczęcie pracy na platformie Xamarin.Forms
- tworzenie interfejsu użytkownika i wyświetlanie widoków
- budowa kodu wielokrotnego użytku
- tworzenie animacji i obsługa gestów użytkownika
- testowanie aplikacji, w tym za pomocą platformy Calabash i terminala REPL

George Taskos programuje od dzieciństwa. W 2005 roku zajął się profesjonalnym tworzeniem aplikacji. Tworzył wieloserwerowe aplikacje oparte na różnych technologiach, m.in. Windows Forms, WPF, ASP.NET MVC, SOAP i REST. Od kilku lat rozwija aplikacje na systemy OS i Android, wykorzystując technologię Xamarin Cross-Platform Mobile. W 2009 roku Taskos uzyskał tytuły Microsoft Certified Solutions Developer i Xamarin Certified Mobile Developer. Mieszka w Nowym Jorku. W wolnym czasie angażuje się we wspieranie rozwoju nowych firm.

PACKT open source
PUBLISHING community experience distilled

Helion

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
● <http://helion.pl/promocje>
Książki najchętniej czytane:
● <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
● <http://helion.pl/nowosci>

sięgnij po **WIĘCEJ**



ISBN 978-83-283-3537-0

ISBN 978-83-283-3537-0



9 788328 335370

cena: 59,00 zł