

Steven F. Daniel

# Xamarin

Tworzenie  
interfejsów  
użytkownika

Helion 

Packt 

Tytuł oryginału: Mastering Xamarin UI Development

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-3948-4

Copyright © Packt Publishing 2017. First published in the English language under the title 'Mastering Xamarin UI Development - (9781786462008)'

Polish edition copyright © 2018 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](mailto: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/xamati.zip>

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

Printed in Poland.

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

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

# Spis treści

|                                                                  |           |
|------------------------------------------------------------------|-----------|
| <b>O autorze</b>                                                 | <b>11</b> |
| <b>O recenzentach</b>                                            | <b>13</b> |
| <b>Wstęp</b>                                                     | <b>15</b> |
| <b>Rozdział 1. Tworzenie macierzystej aplikacji TrackMyWalks</b> | <b>21</b> |
| <b>Tworzenie rozwiązania TrackMyWalks</b>                        | <b>22</b> |
| Aktualizowanie pakietów rozwiązania TrackMyWalks                 | 27        |
| <b>Tworzenie modelu TrackMyWalks</b>                             | <b>29</b> |
| <b>Tworzenie strony głównej prezentacji szlaków</b>              | <b>31</b> |
| <b>Tworzenie strony treści nowego wpisu</b>                      | <b>34</b> |
| <b>Strona opisu szlaku</b>                                       | <b>37</b> |
| Dodawanie pakietu NuGet Xamarin.Forms.Maps                       | 39        |
| <b>Tworzenie strony treści DistanceTravelledPage</b>             | <b>41</b> |
| <b>Tworzenie strony ekranu startowego</b>                        | <b>45</b> |
| Modyfikacja klasy App Xamarin.Forms                              | 47        |
| <b>Różnice między Xamarin Studio i Visual Studio</b>             | <b>48</b> |
| <b>Uruchamianie aplikacji TrackMyWalks w symulatorze</b>         | <b>49</b> |
| <b>Podsumowanie</b>                                              | <b>51</b> |
| <b>Rozdział 2. MVVM i wiązanie danych</b>                        | <b>53</b> |
| <b>Wzorzec architekuralny MVVM</b>                               | <b>54</b> |
| <b>Implementowanie modeli widoków MVVM w aplikacji</b>           | <b>55</b> |
| <b>Tworzenie klasy WalkBaseViewModel</b>                         | <b>56</b> |
| <b>Implementowanie klasy WalksPageViewModel</b>                  | <b>59</b> |
| Przystosowywanie strony głównej szlaków do modelu MVVM           | 61        |

|                                                                                             |            |
|---------------------------------------------------------------------------------------------|------------|
| <b>Implementowanie modelu widoku strony wprowadzania nowego wpisu</b>                       | <b>63</b>  |
| Przystosowywanie strony WalksEntryPage do modelu MVVM                                       | 67         |
| <b>Implementowanie modelu widoku strony szlaku</b>                                          | <b>70</b>  |
| Przystosowywanie strony WalksTrailPage do modelu MVVM                                       | 71         |
| <b>Implementowanie modelu DistanceTravelledViewModel</b>                                    | <b>73</b>  |
| Przystosowywanie strony DistanceTravelledPage do modelu MVVM                                | 76         |
| <b>Podsumowanie</b>                                                                         | <b>80</b>  |
| <b>Rozdział 3. Nawigacja po modelu MVVM według Xamarin.Forms</b>                            | <b>81</b>  |
| <b>Interfejs API nawigacji Xamarin.Forms</b>                                                | <b>82</b>  |
| <b>Różnice między nawigacją i modelami widoków</b>                                          | <b>83</b>  |
| <b>Implementowanie usługi nawigacji</b>                                                     | <b>84</b>  |
| Tworzenie interfejsu usługi nawigacji                                                       | 85         |
| Tworzenie usługi nawigacji pośród modeli widoków                                            | 88         |
| Przystosowywanie modelu WalkBaseViewModel do korzystania z usługi nawigacji                 | 92         |
| Modyfikowanie modelu widoku strony głównej szlaków i usługi nawigacji                       | 94         |
| Dostosowywanie strony głównej szlaków do korzystania<br>ze zmienionego modelu widoku        | 97         |
| Modyfikowanie modelu widoku strony dodawania wpisów i usługi nawigacji                      | 100        |
| Dostosowywanie strony WalksEntryPage do zmienionego modelu widoku                           | 104        |
| Modyfikowanie modelu widoku strony szlaku i usługi nawigacji                                | 107        |
| Przystosowywanie strony WalksTrailPage do korzystania<br>ze zmienionego modelu widoku       | 109        |
| Modyfikowanie modelu widoku przebytego dystansu i usługi nawigacji                          | 111        |
| Przystosowywanie strony DistanceTravelledPage<br>do korzystania z odnowionego modelu widoku | 115        |
| Dostosowywanie klasy Xamarin.Forms.App do korzystania z usługi nawigacji                    | 118        |
| <b>Podsumowanie</b>                                                                         | <b>120</b> |
| <b>Rozdział 4. Funkcje dotyczące lokalizacji</b>                                            | <b>121</b> |
| <b>Tworzenie i używanie usług specyficznych dla konkretnej platformy</b>                    | <b>122</b> |
| Tworzenie interfejsu usługi lokalizacji dla aplikacji TrackMyWalks                          | 122        |
| Klasa usługi lokalizacji dla platformy Android                                              | 124        |
| Klasa usługi lokalizacji dla platformy iOS                                                  | 130        |
| Włączanie aktualizacji w tle i sprawdzanie aktualnej pozycji użytkownika                    | 135        |
| Dostosowywanie klasy WalkEntryViewModel do współpracy z usługą lokalizacji                  | 138        |
| Dostosowywanie klasy DistanceTravelledViewModel do usługi lokalizacyjnej                    | 143        |
| Rejestrowanie modeli widoków na stronie SplashPage                                          | 147        |
| Integrowanie klasy MainActivity z Xamarin.Forms.Maps                                        | 149        |
| Dodawanie do klasy Xamarin.Forms.App funkcji specyficznych dla platform                     | 150        |
| <b>Podsumowanie</b>                                                                         | <b>153</b> |

|                                                                                 |            |
|---------------------------------------------------------------------------------|------------|
| <b>Rozdział 5. Regulacja parametrów interfejsu użytkownika</b>                  | <b>155</b> |
| <b>Tworzenie klasy DataTemplate dla aplikacji TrackMyWalks</b>                  | <b>156</b> |
| Wykorzystywanie szablonu danych na stronie głównej                              | 160        |
| <b>Tworzenie elementu wyboru opcji dla platformy iOS</b>                        | <b>162</b> |
| <b>Tworzenie renderera dla własnej kontrolki dla platformy iOS</b>              | <b>164</b> |
| Stosowanie własnego elementu wyboru opcji na stronie WalksEntryPage             | 168        |
| <b>Tworzenie efektów dla systemu iOS za pomocą API Effects</b>                  | <b>170</b> |
| <b>Tworzenie efektów dla platformy Android za pomocą interfejsu API Effects</b> | <b>174</b> |
| <b>Implementowanie konwerterów wartości w aplikacji TrackMyWalks</b>            | <b>177</b> |
| Wykorzystywanie konwertera w klasie WalkBaseViewModel                           | 180        |
| Wykorzystywanie konwertera logicznego w modelu WalksPageViewModel               | 181        |
| Dostosowywanie strony głównej do zmienionego modelu widoku                      | 183        |
| Stosowanie zmienionego modelu widoku na stronie WalksTrailPage                  | 186        |
| Wykorzystywanie zmienionego modelu widoku na stronie DistanceTravelledPage      | 188        |
| Wykorzystywanie efektów platformy w klasie WalkCellDataTemplate                 | 189        |
| <b>Podsumowanie</b>                                                             | <b>191</b> |
| <br>                                                                            |            |
| <b>Rozdział 6. Szablony Razor</b>                                               | <b>193</b> |
| <b>Podstawy biblioteki Razor</b>                                                | <b>194</b> |
| <b>Tworzenie i implementowanie szablonów Razor w Xamarin Studio</b>             | <b>194</b> |
| Dodawanie pakietu SQLite.Net do rozwiązania BookLibrary                         | 197        |
| <b>Tworzenie i implementowanie opakowania bazy danych książek</b>               | <b>201</b> |
| <b>Tworzenie i implementowanie opakowania bazy danych aplikacji BookLibrary</b> | <b>205</b> |
| <b>Tworzenie i implementowanie strony głównej listy książek</b>                 | <b>207</b> |
| <b>Tworzenie i implementowanie szablonu BookLibraryAdd</b>                      | <b>209</b> |
| <b>Tworzenie i implementowanie szablonu BookLibraryEdit</b>                     | <b>210</b> |
| <b>Tworzenie i implementowanie klasy WebViewController</b>                      | <b>212</b> |
| Rozbudowa arkusza stylów biblioteki książek                                     | 218        |
| <b>Podsumowanie</b>                                                             | <b>220</b> |
| <br>                                                                            |            |
| <b>Rozdział 7. API Data Access i usługi aplikacji Microsoft Azure</b>           | <b>221</b> |
| <b>Rejestracja aplikacji TrackMyWalks w Microsoft Azure</b>                     | <b>222</b> |
| Dodawanie pakietu NuGet Json.Net do aplikacji TrackMyWalks                      | 227        |
| Dodawanie pakietu NuGet HttpClient do aplikacji TrackMyWalks                    | 229        |
| Wykorzystywanie biblioteki Json.Net w modelu WalkEntries                        | 230        |
| <b>Tworzenie klasy usługi sieciowej HTTP dla aplikacji TrackMyWalks</b>         | <b>231</b> |
| <b>Tworzenie API DataService dla aplikacji TrackMyWalks</b>                     | <b>234</b> |
| <b>Tworzenie klasy API DataService dla aplikacji TrackMyWalks</b>               | <b>235</b> |
| Zmiany w klasie WalkBaseViewModel, aby móc korzystać z API DataService          | 238        |
| Dostosowywanie klasy WalkEntryViewModel do interfejsu DataService               | 241        |
| Dostosowywanie klasy WalksPageViewModel do API DataService                      | 243        |
| Dostosowywanie strony WalksPage do nowego modelu widoku                         | 245        |
| Dostosowywanie klasy elementu wyboru opcji do platformy iOS                     | 248        |
| Dostosowywanie strony WalksEntryPage do nowego elementu wyboru opcji            | 250        |
| <b>Podsumowanie</b>                                                             | <b>253</b> |

|                                                                                                        |            |
|--------------------------------------------------------------------------------------------------------|------------|
| <b>Rozdział 8. Uspołecznianie aplikacji, czyli jak korzystać z API Facebooka</b>                       | <b>255</b> |
| <b>Rejestracja aplikacji TrackMyWalks na Facebooku</b>                                                 | <b>256</b> |
| Dodawanie pakietu Xamarin.Auth do aplikacji TrackMyWalks                                               | 259        |
| Dodawanie biblioteki SDK Facebooka do aplikacji TrackMyWalks                                           | 261        |
| <b>Tworzenie modelu użytkownika Facebooka w aplikacji TrackMyWalks</b>                                 | <b>264</b> |
| Tworzenie klasy FacebookCredentials w aplikacji TrackMyWalks                                           | 266        |
| Logowanie do aplikacji TrackMyWalks za pomocą Facebooka                                                | 269        |
| Tworzenie klasy logowania na Facebooku w aplikacji TrackMyWalks (iOS)                                  | 270        |
| <b>Dostosowywanie interfejsu NavigationService do aplikacji TrackMyWalks</b>                           | <b>273</b> |
| Dostosowywanie klasy NavigationService do aplikacji TrackMyWalks                                       | 274        |
| <b>Dostosowywanie strony WalksPage do mechanizmu logowania Facebooka</b>                               | <b>277</b> |
| Uzdatnianie modelu widoku WalksPage do współpracy z FacebookApiUser                                    | 278        |
| Dostosowywanie klasy DistanceTravelledPage do aplikacji TrackMyWalks                                   | 279        |
| Obsługa logowania na Facebooku przez klasę Xamarin.Forms App                                           | 282        |
| Funkcjonalność Facebooka w aplikacji TrackMyWalks                                                      | 283        |
| <b>Podsumowanie</b>                                                                                    | <b>287</b> |
| <b>Rozdział 9. Testowanie jednostkowe aplikacji Xamarin.Forms przy użyciu środowisk NUnit i UITest</b> | <b>289</b> |
| <b>Tworzenie folderu rozwiązania testów jednostkowych w Xamarin Studio</b>                             | <b>290</b> |
| <b>Tworzenie projektu testów jednostkowych w Xamarin Studio</b>                                        | <b>291</b> |
| Dodawanie pakietu NuGet Moq do projektu testów jednostkowych                                           | 293        |
| Dodawanie projektu TrackMyWalks do TrackMyWalks.UnitTests                                              | 295        |
| Implementowanie klasy testowej NUnit WalksTrailViewModel                                               | 296        |
| Implementowanie klasy testowej NUnit WalkEntryViewModel                                                | 298        |
| Wykonywanie testów TrackMyWalks.UnitTests w Xamarin Studio                                             | 302        |
| <b>Tworzenie projektu testowania interfejsu użytkownika w Xamarin Studio</b>                           | <b>305</b> |
| <b>Najczęściej używane metody UITest</b>                                                               | <b>307</b> |
| Przygotowywanie i inicjalizowanie aplikacji TrackMyWalks do testów UITest                              | 308        |
| Implementowanie CreateNewWalkEntry przy użyciu UITest.Framework                                        | 310        |
| <b>Dodawanie agenta Xamarin Test Cloud do projektu iOS</b>                                             | <b>314</b> |
| Dostosowywanie klasy AppDelegate do współpracy z Xamarin Test Cloud Agent                              | 315        |
| <b>Testowanie aplikacji TrackMyWalks za pomocą testów UITest w Xamarin Studio</b>                      | <b>318</b> |
| <b>Podsumowanie</b>                                                                                    | <b>320</b> |
| <b>Rozdział 10. Pakowanie i wdrażanie aplikacji Xamarin.Forms</b>                                      | <b>321</b> |
| <b>Tworzenie i przygotowywanie zespołu programistów iOS</b>                                            | <b>322</b> |
| <b>Tworzenie certyfikatu dla aplikacji iOS TrackMyWalks</b>                                            | <b>326</b> |
| Uzyskiwanie certyfikatu programistycznego iOS od firmy Apple                                           | 328        |
| Tworzenie identyfikatora aplikacji TrackMyWalks (iOS)                                                  | 331        |
| <b>Tworzenie profilu dostarczania aplikacji TrackMyWalks</b>                                           | <b>333</b> |
| <b>Przygotowywanie aplikacji TrackMyWalks (iOS) do wystania do weryfikacji</b>                         | <b>339</b> |
| Wysyłanie aplikacji TrackMyWalks (iOS) do iTunes Connect przy użyciu Xamarin Studio                    | 344        |
| <b>Podsumowanie</b>                                                                                    | <b>351</b> |
| <b>Skorowidz</b>                                                                                       | <b>353</b> |

# Tworzenie macierzystej aplikacji TrackMyWalks

Wraz z pojawieniem się platformy **Xamarin** kilka lat temu programiści cieszą się możliwością tworzenia macierzystych aplikacji mobilnych przeznaczonych dla innych platform niż Microsoftu oraz możliwością wyboru języka **C#** lub **F#**, dzięki której mogą przygotowywać produkty zarówno dla systemu iOS, jak i Android.

Z książki tej nauczysz się najlepszych zasad tworzenia wieloplatformowych aplikacji mobilnych i poznasz wzorce projektowe na platformie **Xamarin.Forms** służącej do tworzenia wieloplatformowych interfejsów użytkownika, które mogą być wykorzystywane w systemach Android, iOS oraz Windows Phone.

Ponieważ każdą z tych aplikacji można napisać za pomocą jednego języka programowania, dobrym rozwiązaniem wydaje się utworzenie pojedynczej bazy kodu, przy użyciu której można następnie kompilować i budować osobne aplikacje dla każdej z wymienionych platform.

Na początku tego rozdziału dowiesz się, jak przygotować z wykorzystaniem **Xamarin.Forms** podstawową strukturę aplikacji, która będzie stanowiła podstawę pracy w następnych rozdziałach. Bazując na niej, będziemy rozbudowywać naszą aplikację, stosując w niej coraz to nowsze rozwiązania. W tym rozdziale pokazuję, jak utworzyć wstępną, niezależną od platformy, macierzystą aplikację **Xamarin.Forms** oraz jak dodawać nowe i aktualizować istniejące pakiety w swoim rozwiązaniu.

Nauczysz się tworzyć modele aplikacji w postaci klas **C#** oraz tworzyć strony z treścią, które będą stanowiły interfejs użytkownika. Na końcu tego rozdziału opisuję różnice w podejściu do tworzenia aplikacji przy użyciu środowisk **Xamarin Studio** i **Microsoft Visual Studio**.

W tym rozdziale:

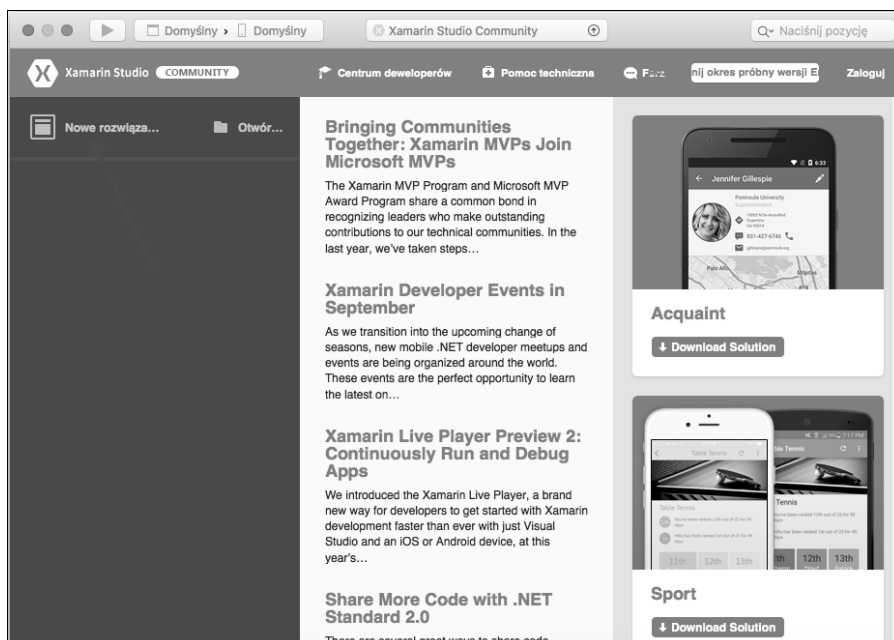
- Tworzenie mobilnej aplikacji Xamarin.Forms o nazwie *TrackMyWalks*.
- Sposób aktualizacji pakietów rozwiązania *TrackMyWalks* za pomocą menedżera pakietów *NuGet*.
- Tworzenie modelu danych *TrackMyWalks*.
- Metoda tworzenia stron treści (*ContentPage*) w rozwiązaniu *TrackMyWalks*.
- Różnice między środowiskami Xamarin Studio i Visual Studio.

# Tworzenie rozwiązania TrackMyWalks

W tym podrozdziale pokażę Ci, jak utworzyć pierwsze rozwiązanie Xamarin.Forms. Zaczniemy od przygotowania podstawowej struktury naszej aplikacji, po czym dodamy niezbędne modele jednostek i zaprojektujemy pliki interfejsu użytkownika.

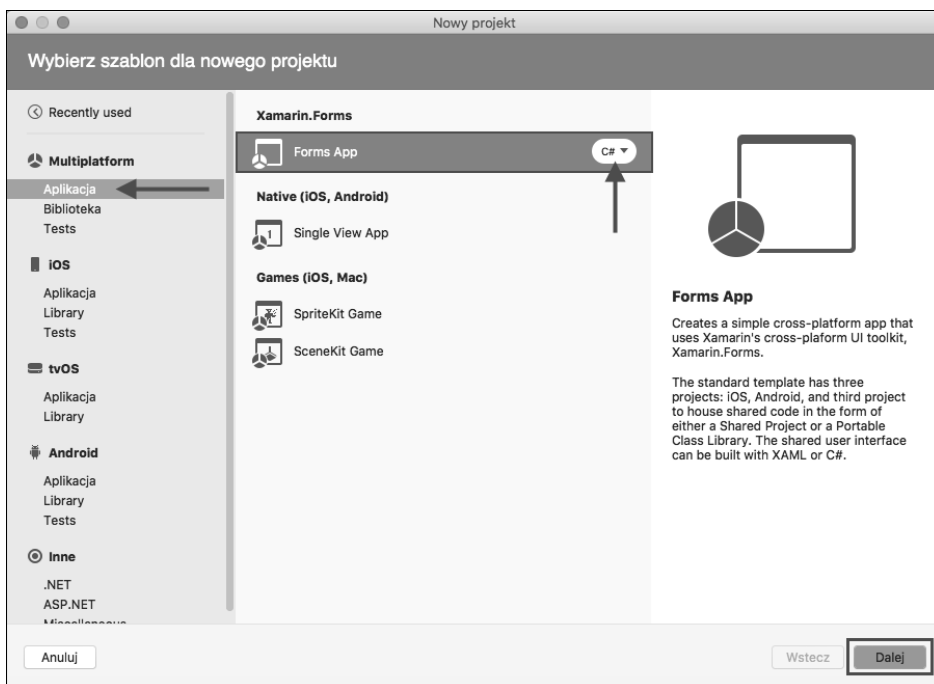
Pierwszą naszą czynnością będzie utworzenie projektu o nazwie *TrackMyWalks*. W Xamarin Studio projekty tworzy się bardzo łatwo. Wystarczy wykonać następujące czynności:

1. Uruchom aplikację Xamarin Studio. Na ekranie pojawi się widoczne poniżej okno:



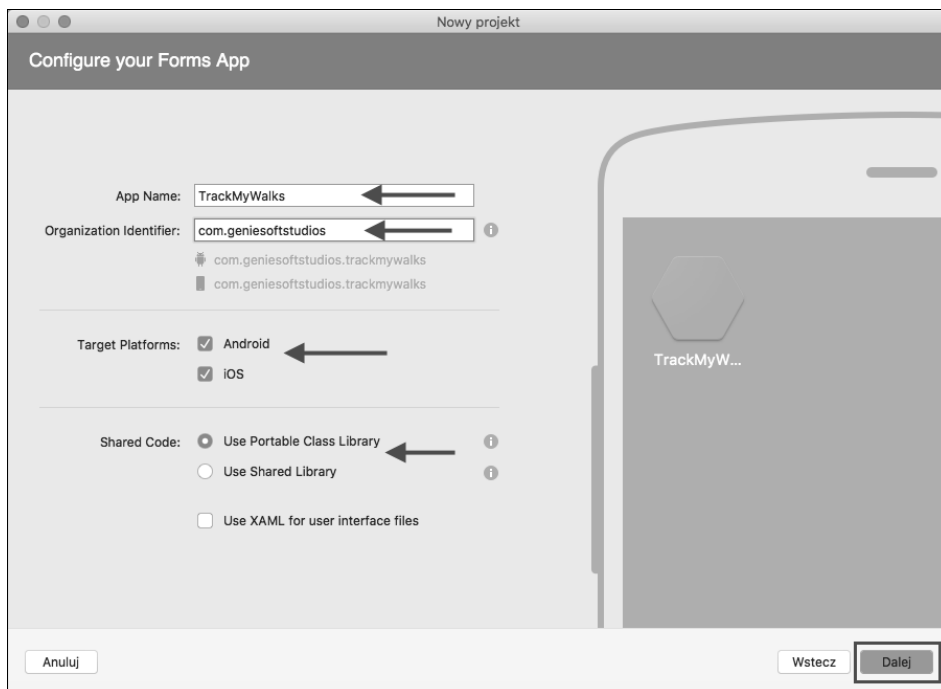


2. Kliknij przycisk *Nowe rozwiązanie* albo naciśnij kombinację klawiszy *Shift + Command + N*.
3. W sekcji *Multiplatform/Aplikacja* kliknij opcję *Forms App* (aplikacja Forms).  
Z listy rozwijanej widocznej po prawej stronie wybierz język programowania *C#*.



4. W polu *App Name* (nazwa aplikacji) wpisz nazwę swojej aplikacji: **TrackMyWalks**.
5. Następnie podaj nazwę w polu *Organization Identifier* (identyfikator organizacji).
6. W kolejnym kroku w polach *Target Platforms* (platformy docelowe) zaznacz pola wyboru *Android* i *iOS*, jeśli nie są jeszcze zaznaczone.
7. Następnie sprawdź, czy w sekcji *Shared Code* (wspólny kod) zaznaczona jest opcja *Use Portable Class Library* (użyj biblioteki klas przenośnych), i jeśli nie jest, to ją zaznacz, jak widać na poniższym zrzucie ekranu.

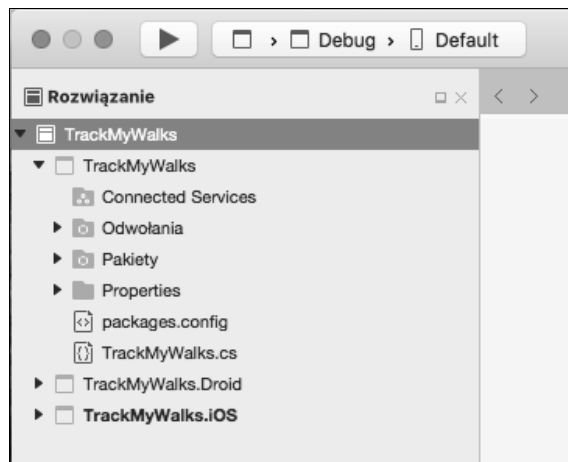
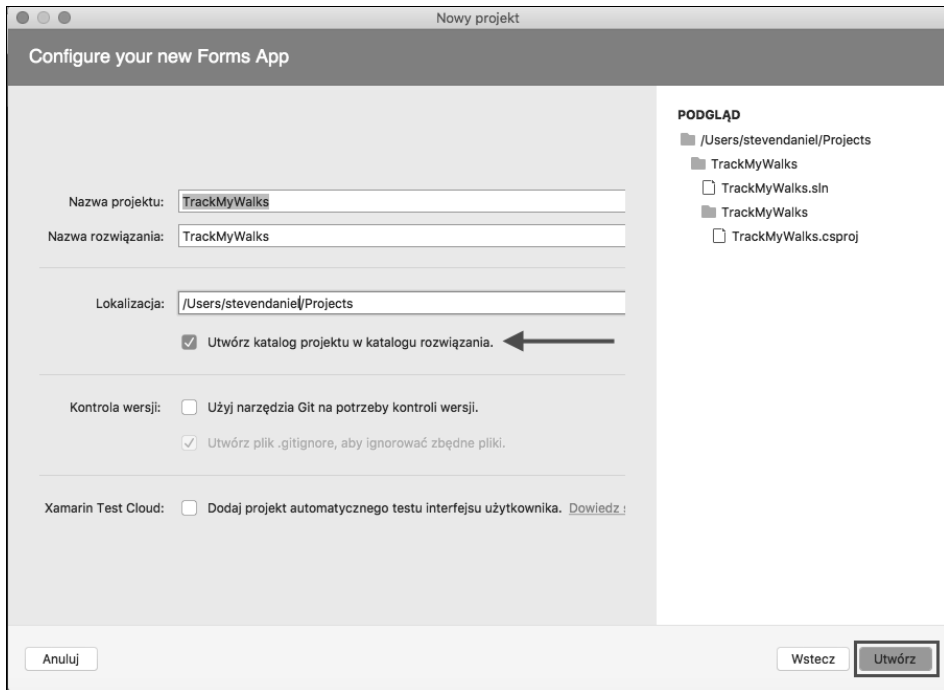
Różnica między Portable Class Library a Shared Library polega głównie na tym, że pierwsza opcja umożliwia napisanie kodu tylko raz i wykorzystywanie go na różnych platformach, takich jak serwisy internetowe czy systemy Android i iOS. Z kolei Shared Library umożliwia skopiowanie wszystkich plików należących do projektu biblioteki do wszystkich projektów znajdujących się w rozwiązaniu podczas kompilacji dla różnych platform, które będą z tej biblioteki korzystać.



Nazwa wpisana w polu *Organization Identifier* nie może się powtarzać. W Xamarin Studio zaleca się stosowanie nazw odwziewierciedlających odwróconą kolejność elementów adresu domeny, np. *com.nazwaDomeny.nazwaAplikacji*.

8. Wyłącz opcję *Use XAML for user interface files* (twórz pliki interfejsu użytkownika w języku XAML).
9. Kliknij przycisk *Dalej*, aby przejść do następnego etapu (zobacz pierwszy rysunek na następnej stronie).
10. Następnie zaznacz opcję *Utwórz katalog projektu w katalogu rozwiązania*.
11. Potem kliknij przycisk *Utwórz*, aby zapisać projekt w wybranym folderze.

Po utworzeniu projektu zostanie wyświetlone okno środowiska programistycznego Xamarin z wykazem kilku domyślnych plików szablonowych (patrz drugi zrzut ekranu na następnej stronie).



Jak widać na zrzucie ekranu, rozwiązanie *TrackMyWalks* zostało podzielone na trzy główne obszary. W poniższej tabeli znajduje się zwięzły opis każdego z nich.

| Nazwa obszaru      | Opis                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TrackMyWalks       | <p>To jest projekt PCL (ang. <i>Portable Class Library</i>), który będzie odgrywał rolę głównej warstwy architekuralnej rozwiązania <i>TrackMyWalks</i>.</p> <p>Projekt ten zawiera całą logikę biznesową, obiekty danych, <i>Xamarin.FormsPages</i>, widoki i inny kod niespecyficzny dla żadnej platformy.</p> <p>Kod źródłowy napisany w tym projekcie może być wykorzystywany w wielu różnych projektach specyficznych dla konkretnych platform.</p> |
| TrackMyWalks.Droid | <p>To jest projekt dla systemu Android, a więc zawiera cały kod źródłowy i wszystkie zasoby potrzebne do skompilowania i wdrożenia aplikacji dla systemu Android należącej do rozwiązania.</p> <p>Domyślnie projekt ten odwołuje się do biblioteki PCL <i>TrackMyWalks</i>.</p>                                                                                                                                                                          |
| TrackMyWalks.iOS   | <p>To jest projekt dla systemu iOS, a więc zawiera cały kod źródłowy i wszystkie zasoby potrzebne do skompilowania i wdrożenia aplikacji dla systemu iOS należącej do rozwiązania.</p> <p>Domyślnie projekt ten odwołuje się do biblioteki PCL <i>TrackMyWalks</i>.</p>                                                                                                                                                                                  |

Zwróć uwagę, że nasze rozwiązanie zawiera plik o nazwie *TrackMyWalks.cs*, który stanowi składnik biblioteki PCL *TrackMyWalks*. W pliku tym znajduje się klasa o nazwie *App*, która dziedziczy z hierarchii klas *Xamarin.Forms.Application*, co widać w poniższym kodzie:

```
//
// TrackMyWalks.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Xamarin.Forms;

namespace TrackMyWalks
{
    public class App : Application
    {
        public App()
        {
            // sprawdzenie systemu docelowego
            if (Device.OS == TargetPlatform.Android)
            {
                MainPage = new SplashPage();
            }
            else
            {
                // strona główna aplikacji
                var navPage = new NavigationPage(new TrackMyWalks.WalksPage()
                {
                    Title = "Track My Walks"
                });
                MainPage = navPage;
            }
        }
    }
}
```

```

protected override void OnStart()
{
    // operacje wykonywane podczas uruchamiania aplikacji
}

protected override void OnSleep()
{
    // operacje wykonywane, gdy aplikacja zostaje uśpiona
}

protected override void OnResume()
{
    // operacje wykonywane w chwili wznowienia działania aplikacji
}
}
}

```

Konstruktor App przypisuje własności MainPage nowy egzemplarz obiektu ContentPage, który na razie tylko wyświetla pewien napis utworzony domyślnie przez kreator projektu. W dalszej części tego rozdziału pokazuję, jak zbudować początkowe widoki stron interfejsu użytkownika, oraz demonstruję sposoby modyfikacji własności MainPage klasy App zawartej w pliku *TrackMyWalks.cs*.

## Aktualizowanie pakietów rozwiązania TrackMyWalks

W tej sekcji pokazuję, jak można zaktualizować pakiety Xamarin.Forms należące do naszego rozwiązania *TrackMyWalks*. Z pewnością udało Ci się już zauważyć, że każdy projekt w naszym rozwiązaniu zawiera folder *Packages*.

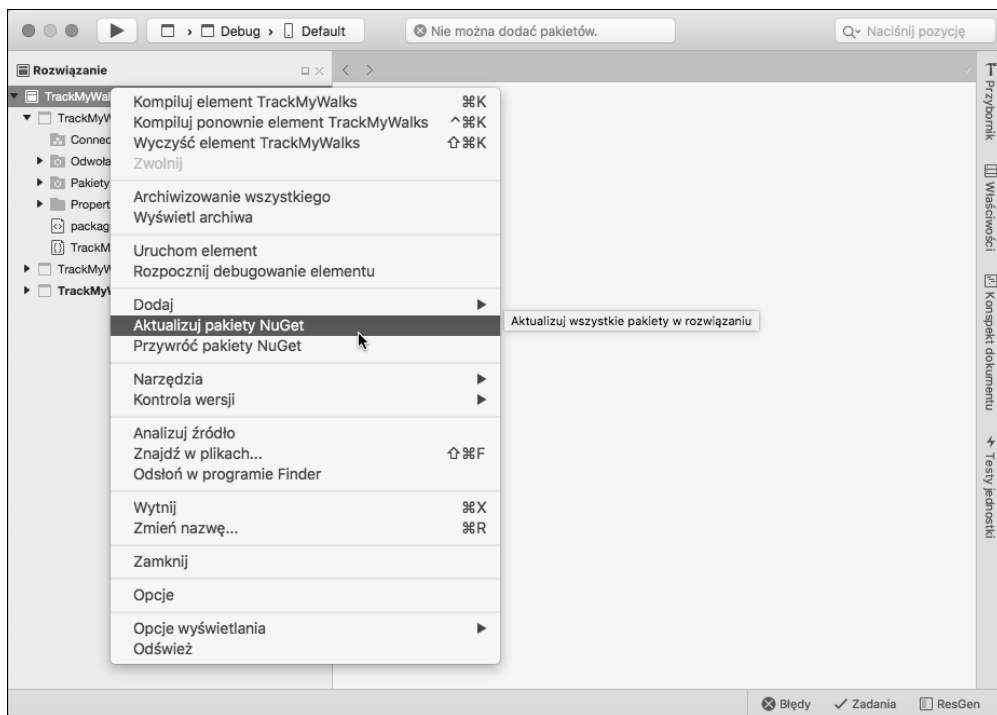
Pakiet Xamarin.Forms to w istocie pakiet *NuGet*, który został automatycznie dodany do naszego rozwiązania, gdy stwierdziliśmy, że chcemy utworzyć szablon projektu Xamarin.FormsApp.

Czasami Xamarin powiadamia, że jakiś pakiet jest przestarzały i trzeba go zaktualizować do najnowszej wersji.

Pakiet *NuGet* to w istocie menedżer pakietów platformy Microsoft Development Platform zawierający narzędzia klienckie, za pomocą których można tworzyć i eksploatować pakiety .NET.

Zobaczymy, jak zaktualizować pakiety *NuGet* w naszym rozwiązaniu *TrackMyWalks*, aby mieć pewność, że posługujemy się najnowszymi pakietami Xamarin.Forms. Wykonujemy następujące czynności:

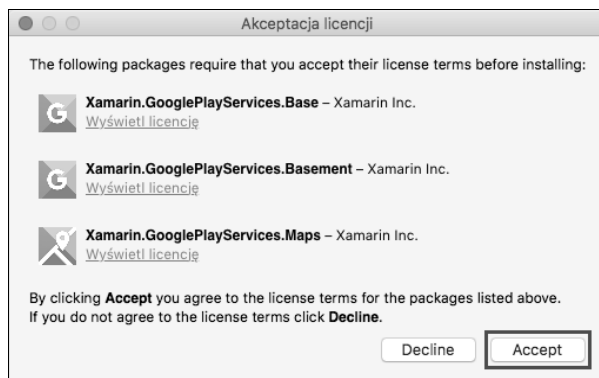
1. Kliknij prawym przyciskiem myszy rozwiązanie *TrackMyWalks* i z menu wybierz polecenie *Aktualizuj pakiety NuGet*, jak pokazano na zrzucie ekranu na następnej stronie.



Gdy włączysz tę opcję, Xamarin Studio zaktualizuje wszystkie pakiety znajdujące się w rozwiązaniu *TrackMyWalks* w projektach dotyczących wszystkich platform i wyświetli wskaźnik postępu o wyglądzie podobnym do pokazanego poniżej:



Podczas instalowania aktualizacji niektórych pakietów dotyczących konkretnych platform będzie konieczne zaakceptowanie licencji, co widać na poniższym rysunku:



2. Kliknij przycisk *Accept* (akceptuj), aby zaakceptować licencje pakietów wymienionych w oknie dialogowym i je zainstalować (zobacz poprzedni rysunek).

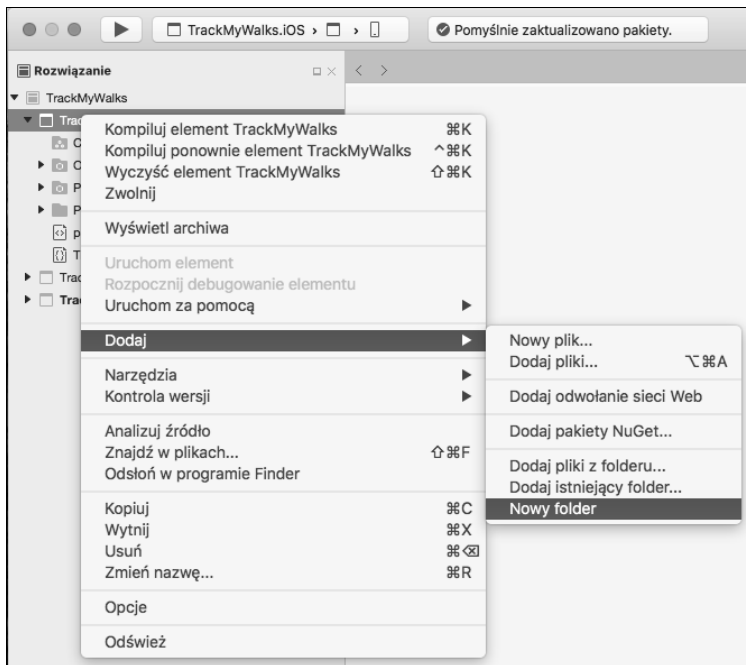
Po zaktualizowaniu pakietów *Xamarin.Forms* w swoim rozwiązaniu możesz przejść do tworzenia plików interfejsu użytkownika w rozwiązaniu *TrackMyWalks*.

## Tworzenie modelu TrackMyWalks

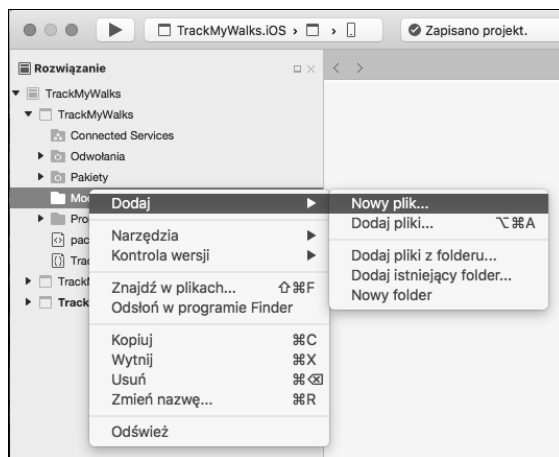
W tym podrozdziale pokażę Ci, jak utworzyć model *TrackMyWalks*, który będzie reprezentował wpisy na temat przebytych przez nas szlaków. Potem dowiesz się, jak przy użyciu tego modelu przygotować i zainicjalizować kilka wpisów dla naszej głównej strony *WalksPage* przy użyciu kontrolki *ListView*, tak aby w każdym wierszu *Listview* można było wyświetlić wpis.

Poniżej przedstawiam listę czynności do wykonania:

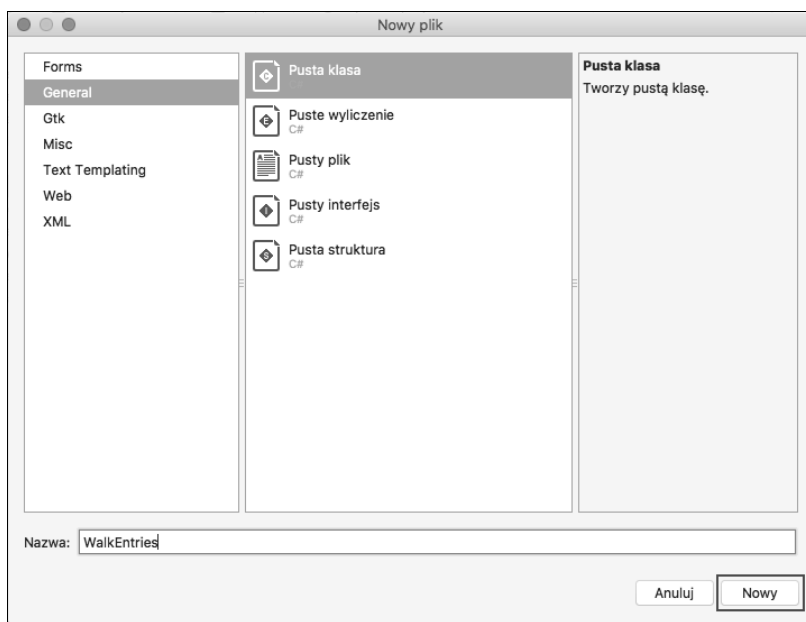
1. W projekcie *Portable Class Library TrackMyWalks* utwórz folder o nazwie *Models*, jak pokazano na poniższym zrzucie ekranu:



2. Następnie w folderze *Models* utwórz nowy plik klasy zgodnie z tym, co widać na poniższym rysunku:



3. W sekcji *General* (ogólne) wybierz opcję *Pusta klasa* i w polu nazwy klasy wpisz nazwę **WalkEntries**:



4. Następnie kliknij przycisk *Nowy*, aby pozwolić kreatorowi na utworzenie nowego pliku (zobacz powyższy rysunek).

Gratulacje! Właśnie został utworzony Twój pierwszy folder i plik klasy C# w rozwiązaniu. Teraz możesz przejść do dodawania deskryptorów własności, które będą definiować nasz model.



5. Otwórz plik *WalkEntries.cs*, znajdź w nim konstruktor klasy `WalkEntries` i wpisz kod zaznaczony pogrubieniem na listingu:

```
//
// WalkEntries.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016.
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//

namespace TrackMyWalks.Models
{
    public class WalkEntries
    {
        public string Title { get; set; }
        public string Notes { get; set; }
        public double Longitude { get; set; }
        public double Latitude { get; set; }
        public double Kilometers { get; set; }
        public string Difficulty { get; set; }
        public double Distance { get; set; }
        public string imageUrl { get; set; }
    }
}
```

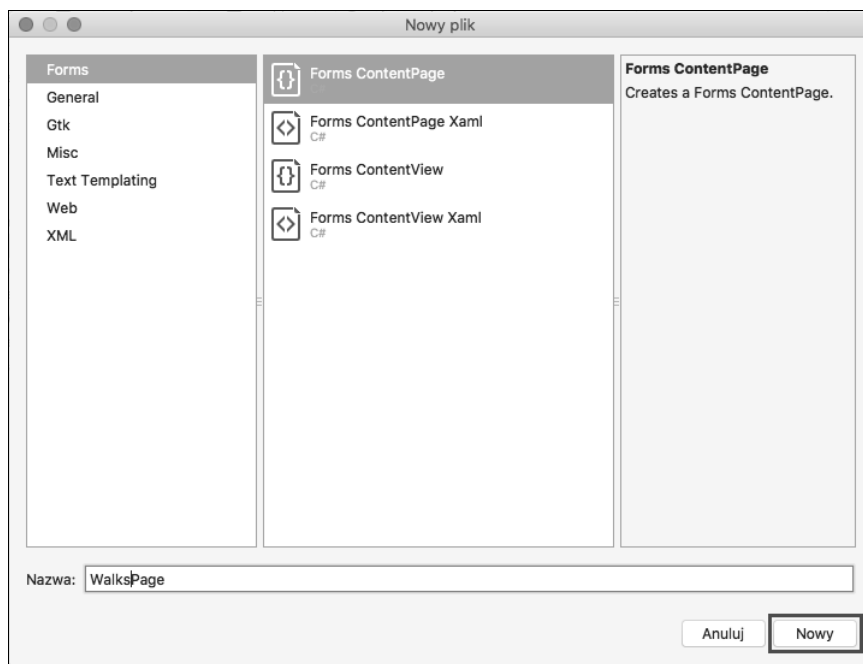
W kodzie tym zdefiniowaliśmy model, który będzie nam służył do reprezentowania wpisów dotyczących naszych szlaków. W następnym podrozdziale przy użyciu tego modelu zainicjalizujemy kilka wpisów dla głównej strony `WalksPage` za pomocą kontrolki `ListView`, a następnie przy użyciu obiektu `DataTemplate` opiszemy sposób prezentacji danych tego modelu w każdym wierszu kontrolki `ListView`.

## Tworzenie strony głównej prezentacji szlaków

Jak napisałem wcześniej, `WalksPage` będzie nam służyć jako główny punkt wejściowy do naszej aplikacji. Przy użyciu modelu `WalkEntries` przekazemy trochę statycznych danych na temat szlaków i wyświetlimy je w kontrolce `ListView` za pomocą obiektu `DataTemplate`. Oto lista czynności, które należy wykonać:

1. W rozwiązaniu Portable Class Library *TrackMyWalks* utwórz folder o nazwie *Pages* w taki sam sposób jak poprzednio.
2. Na ekranie *Nowy plik* w lewym okienku otwórz sekcję *Forms* (formularze).
3. W prawym okienku zaznacz opcję *Forms ContentPage*.
4. W polu *Nazwa* nowo tworzonej klasy wpisz **WalksPage**.

5. Na zakończenie kliknij przycisk *Nowy* (zobacz poniższy rysunek).



6. Następnie otwórz plik *WalksPage.cs* w edytorze kodu i wpisz zaznaczone pogrubieniem fragmenty poniższego kodu.

```
//
// WalksPage.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using System.Collections.Generic;
using Xamarin.Forms;
using TrackMyWalks.Models;

namespace TrackMyWalks
{
    public class WalksPage : ContentPage
    {
        public WalksPage()
        {
            var newWalkItem = new ToolbarItem
            {
                Text = "Dodaj szlak"
            };
            newWalkItem.Clicked += (sender, e) =>

```

```

{
    Navigation.PushAsync(new WalkEntryPage());
};

ToolbarItems.Add(new WalkItem);

var walkItems = new List<WalkEntries>
{
    new WalkEntries {
        Title = "10-milowy szlak wzdłuż strumienia, Margaret River",
        Notes = "10-milowy szlak wzdłuż strumienia zaczyna się
↳w Rotary Park w pobliżu Old Kate, czyli starej lokomotywy
↳stojącej w północnej części Margaret River.",
        Latitude = -33.9727604,
        Longitude = 115.0861599,
        Kilometers = 7.5,
        Distance = 0,
        Difficulty = "Średni",
        ImageUrl = "http://trails.wa.com.au/media/cache/media/images/
↳trails/_mid/FullSizeRender1_600_480_c1.jpg"
    },
    new WalkEntries {
        Title = "Szlak Ancient Empire, Dolina Gigantów",
        Notes = "Ancient Empire to 450-metrowy szlak pośród
↳gigantycznych drzew, wśród których znajdują się popularne
↳sękate olbrzymy zwane Grandma Tingle.",
        Latitude = -34.9749188,
        Longitude = 117.3560796,
        Kilometers = 450,
        Distance = 0,
        Difficulty = "Wysoki",
        ImageUrl = "http://trails.wa.com.au/media/cache/media/images/
↳trails/_mid/Ancient_Empire_534_480_c1.jpg"
    },
};

var itemTemplate = new DataTemplate(typeof(ImageCell));
itemTemplate.SetBinding(TextCell.TextProperty, "Title");
itemTemplate.SetBinding(TextCell.DetailProperty, "Notes");
itemTemplate.SetBinding(ImageCell.ImageSourceProperty, "ImageUrl");

var walksList = new ListView
{
    HasUnevenRows = true,
    ItemTemplate = itemTemplate,
    ItemsSource = walkItems,
    SeparatorColor = Color.FromHex("#ddd"),
};

```

```

// konfiguracja procedury obsługi zdarzeń
walksList.ItemTapped += (object sender, ItemTappedEventArgs e) =>
{
    var item = (WalkEntries)e.Item;
    if (item == null) return;
    Navigation.PushAsync(new WalkTrailPage(item));
    item = null;
};

Content = walksList;
}
}
}

```

Na początku zadeklarowaliśmy zmienną `newWalkItem` typu klasy `ToolBarItem`. Za pomocą tej zmiennej wiążemy nowy przycisk *Dodaj szlak* z głównym paskiem narzędzi podstawowej kolekcji `ContentPage.ToolbarItems`, aby umożliwić użytkownikom aplikacji dodawanie informacji o nowych szlakach.

Następnie tworzymy zdarzenie dla obiektu `newWalkItem`, wykorzystując zdarzenie `Clicked` klasy `ToolBarItem`, za pomocą którego będzie można przejść do nowej strony `WalksEntryPage`.

Kolejną naszą czynnością jest zadeklarowanie nowej zmiennej o nazwie `walkItems`, która jest kolekcją elementów listy do przechowywania wszystkich naszych wpisów w modelu. Za pomocą klasy `DataTemplate` opisujemy żądany sposób prezentowania danych modelu w każdym z wierszy zadeklarowanych w `ListView`.

Na koniec definiujemy procedurę obsługi zdarzeń dla naszego widoku `ListView`, która pozwoli nam przejść do strony `WalksTrailPage` w celu wyświetlenia informacji o wybranym elemencie.

## Tworzenie strony treści nowego wpisu

W tym podrozdziale zbudujemy interfejs użytkownika na naszej stronie `WalkEntryPage`, służącej do dodawania informacji o nowym szlaku do aplikacji. Jej wywołanie nastąpi, gdy użytkownik naciśnie przycisk *Dodaj szlak* na stronie głównej.

Stronę do wprowadzania nowych danych można przedstawić na wiele sposobów. W naszej aplikacji skorzystamy z widoku `Tableview`, ale równie dobrze mogliśmy posłużyć się widokiem `StackLayout` i zaprezentować informacje w postaci serii obiektów `Label` i `EntryCell`.

Aby utworzyć nową stronę `WalkEntryPage`, wykonaj następujące czynności:

1. Utwórz nową stronę `ContentPage` o nazwie `WalkEntryPage` w sposób opisany w poprzednim podrozdziale.

2. Otwórz w edytorze kodu plik *WalkEntryPage.cs* i wpisz fragmenty kodu zaznaczone poniżej pogrubieniem:

```
//
// WalkEntryPage.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Xamarin.Forms;
using TrackMyWalks.Models;
using System.Collections.Generic;

namespace TrackMyWalks
{
    public class WalkEntryPage : ContentPage
    {
        public WalkEntryPage()
        {
            // tytuł strony
            Title = "Nowy wpis";

            // pola do wypełnienia
            var walkTitle = new EntryCell
            {
                Label = "Tytuł:",
                Placeholder = "Nazwa szlaku"
            };

            var walkNotes = new EntryCell
            {
                Label = "Uwagi:",
                Placeholder = "Opis"
            };

            var walkLatitude = new EntryCell
            {
                Label = "Szerokość geograficzna:",
                Placeholder = "Szerokość",
                Keyboard = Keyboard.Numeric
            };

            var walkLongitude = new EntryCell
            {
                Label = "Długość geograficzna:",
                Placeholder = "Długość",
                Keyboard = Keyboard.Numeric
            };

            var walkKilometers = new EntryCell
            {
```

```

        Label = "Liczba kilometrów:",
        Placeholder = "Liczba kilometrów",
        Keyboard = Keyboard.Numeric
    };

    var walkDifficulty = new EntryCell
    {
        Label = "Poziom trudności:",
        Placeholder = "Poziom trudności szlaku"
    };

    var walkImageUrl = new EntryCell
    {
        Label = "URL obrazu:",
        Placeholder = "URL obrazu"
    };

    // definicja widoku TableView
    Content = new TableView
    {
        Intent = TableIntent.Form,
        Root = new TableRoot
        {
            {
                new TableSection()
                {
                    walkTitle,
                    walkNotes,
                    walkLatitude,
                    walkLongitude,
                    walkKilometers,
                    walkDifficulty,
                    walkImageUrl
                }
            }
        }
    };

    var saveWalkItem = new ToolbarItem
    {
        Text = "Zapisz"
    };
    saveWalkItem.Clicked += (sender, e) =>
    {
        Navigation.PopToRootAsync(true);
    };

    ToolbarItems.Add(saveWalkItem);
}
}
}

```

Najpierw zadeklarowaliśmy etykiety `EntryCell` dla naszego interfejsu użytkownika, za pomocą którego będziemy przyjmować wprowadzane informacje — *Tytuł*, *Uwagi*, *Szerokość geograficzna*, *Długość geograficzna*, *Liczba kilometrów*, *Poziom trudności* oraz *URL obrazu*. Potem pokażę Ci, jak modyfikować wygląd i styl komórek `EntryCell` przez utworzenie specyficznego dla danej platformy elementu do wybierania ustawień dotyczących poziomu trudności i liczby kilometrów.

Następnie zdefiniowaliśmy widok `TableView` i dodaliśmy każde z naszych pól `EntryCell` do własności `TableSection` kontrolki `TableView`. Każda własność `TableSection` zdefiniowana w `TableView` zawiera nagłówek i jedną lub więcej komórek `ViewCell`, które w naszym przypadku są polami `EntryCell`.

Na koniec zadeklarowaliśmy i dodaliśmy do kolekcji `ContentPageToolbarItems` obiekt `ToolbarItem` o nazwie `saveWalkItem`, a następnie utworzyliśmy zdarzenie, które w reakcji na kliknięcie powoduje zapisanie wprowadzonych informacji dotyczących szlaku na stronie głównej. Oczywiście później jeszcze wielokrotnie będziemy zmieniać kod strony `WalkEntryPage`, która w reakcji na naciśnięcie przycisku *Zapisz* będzie wysyłała informacje do serwera za pośrednictwem REST-owego interfejsu API, a następnie będzie odświeżała stronę główną *TrackMyWalks*.

## Strona opisu szlaku

W tym podrozdziale rozpoczniemy budowę interfejsu użytkownika dla naszej strony `WalksTrailPage`. Strona ta będzie wywoływana kliknięciem wpisu w widoku `ListView` na głównej stronie treści *TrackMyWalks* i będzie prezentowała informacje dotyczące wybranego szlaku.

1. Utwórz stronę `ContentPage` o nazwie `WalkTrailPage` w sposób opisany w jednym z wcześniejszych podrozdziałów.
2. Otwórz w edytorze kodu plik `WalkTrailPage.cs` i wpisz fragmenty kodu zaznaczone poniżej pogrubieniem:

```
//
// WalkTrailPage.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016.
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Xamarin.Forms;
using TrackMyWalks.Models;

namespace TrackMyWalks
{
    public class WalkTrailPage : ContentPage
    {
        public WalkTrailPage(WalkEntries walkItem)
        {
            Title = "Szlak";
        }
    }
}
```

```

var beginTrailWalk = new Button
{
    BackgroundColor = Color.FromHex("#008080"),
    TextColor = Color.White,
    Text = "Rozpocznij ten szlak"
};

// definicja procedury obsługi zdarzeń
beginTrailWalk.Clicked += (sender, e) =>
{
    if (walkItem == null) return;
    Navigation.PushAsync(new DistanceTravelledPage(walkItem));
    Navigation.RemovePage(this);
    walkItem = null;
};

var walkTrailImage = new Image()
{
    Aspect = Aspect.AspectFill,
    Source = walkItem.ImageUrl
};

var trailNameLabel = new Label()
{
    FontSize = 28,
    FontAttributes = FontAttributes.Bold,
    TextColor = Color.Black,
    Text = walkItem.Title
};

var trailKilometersLabel = new Label()
{
    FontAttributes = FontAttributes.Bold,
    FontSize = 12,
    TextColor = Color.Black,
    Text = $"Długość: { walkItem.Kilometers } km"
};

var trailDifficultyLabel = new Label()
{
    FontAttributes = FontAttributes.Bold,
    FontSize = 12,
    TextColor = Color.Black,
    Text = $"Poziom trudności: { walkItem.Difficulty } "
};

var trailFullDescription = new Label()
{
    FontSize = 11,
    TextColor = Color.Black,
    Text = $"{ walkItem.Notes }",
    HorizontalOptions = LayoutOptions.FillAndExpand
};

```



```

this.Content = new ScrollView
{
    Padding = 10,
    Content = new StackLayout
    {
        Orientation = StackOrientation.Vertical,
        HorizontalOptions = LayoutOptions.FillAndExpand,
        Children =
        {
            walkTrailImage,
            trailNameLabel,
            trailKilometersLabel,
            trailDifficultyLabel,
            trailFullDescription,
            beginTrailWalk
        }
    }
};
}
}
}

```

Najpierw importujemy klasę `TrackMyWalks.Models`, przy użyciu której będziemy pobierać informacje przekazywane ze strony `WalksPage`.

Następnie deklarujemy zmienną `beginTrailWalk` dziedziczącą po klasie `Button`. Potem definiujemy zdarzenie `Clicked` klasy `Button`, które pozwoli nam przejść do strony treści `DistanceTravelledPage` w celu wyświetlenia informacji o naszym szlaku po usunięciu strony treści szlaku z hierarchii `NavigationPage`.

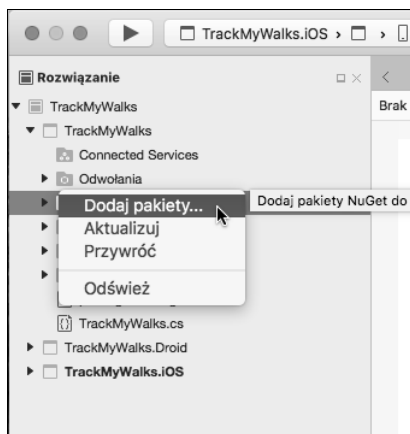
W następnym kroku deklarujemy zmienną obrazu `walkTrailImage` i ustawiamy jego własność źródła `Source` na obraz wybranego elementu `walkItem` z `ListView`. W dalszej kolejności deklarujemy i inicjalizujemy kilka obiektów etykiet, które będą zawierać informacje `walkItem` przekazane z kontrolki `ListView` strony treści `WalksPage` i które zostaną wyświetlone.

Potem definiujemy kontrolkę `ScrollView`, która jest częścią bazowej klasy `Xamarin.Forms.Core` i dodajemy wszystkie nasze pola `Image` i `Label` formularza do kontrolki `StackLayout`. Kontrolka `ScrollView` to fantastyczny element umożliwiający przewijanie treści strony `ContentPage`, jeśli ta nie mieści się na pojedynczym ekranie urządzenia.

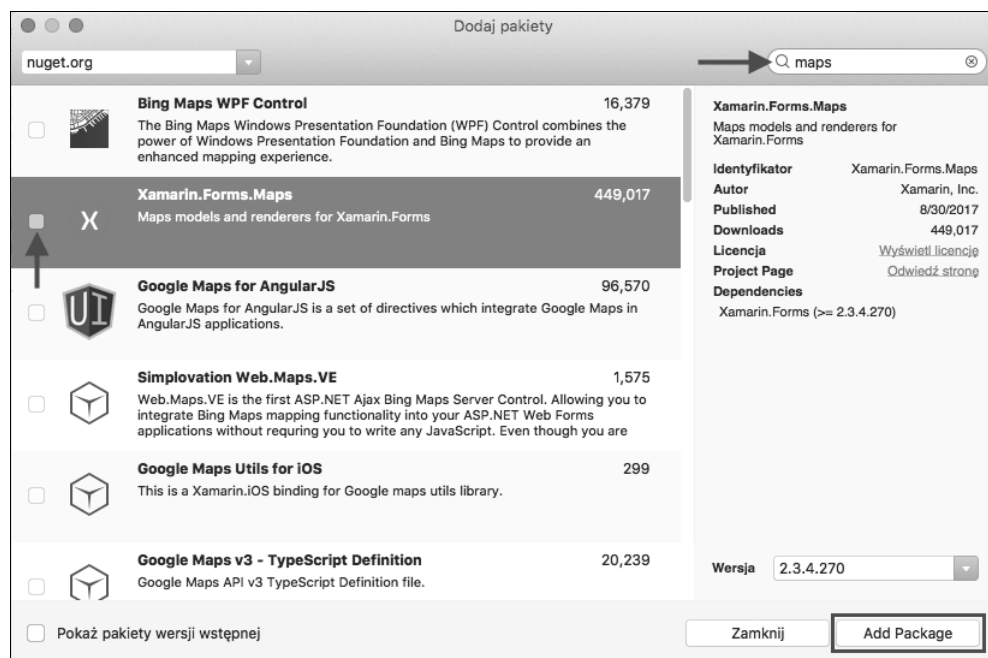
## Dodawanie pakietu NuGet Xamarin.Forms.Maps

W tej sekcji dodamy pakiet NuGet `Xamarin.Forms.Maps` do naszego projektu głównego i projektów specyficznych dla platform `iOS` i `Android`. Jest on potrzebny do tego, abyśmy mogli korzystać z kontrolki `Xamarin.FormsMap` na stronie treści `DistanceTravelled`, którą utworzymy w następnym podrozdziale.

1. Kliknij prawym przyciskiem myszy rozwiązanie *TrackMyWalks* i w menu kliknij opcję *Dodaj pakiety*, co zilustrowano na poniższym zrzucie ekranu:



2. Zostanie wyświetlone okno dialogowe *Dodaj pakiety*. W polu wyszukiwania wpisz słowo **maps**, a następnie na liście, która się pojawi, kliknij opcję *Xamarin.Forms.Maps*:



3. Na koniec kliknij przycisk *Add Package* (dodaj pakiet), aby dodać pakiet NuGet *Xamarin.Forms.Maps* do rdzenia rozwiązania *TrackMyWalks*.

4. Powtórz te same czynności, aby dodać pakiet NuGet `Xamarin.Forms.Maps` do projektów iOS i Android znajdujących się w rozwiązaniu *TrackMyWalks*.

Mając pakiet NuGet `Xamarin.Forms` z kontrolką `Map`, możemy zacząć jej używać na stronie treści `DistanceTravelled`, którą opisuję w następnym podrozdziale.

## Tworzenie strony treści `DistanceTravelledPage`

W tym podrozdziale rozpoczynamy prace nad interfejsem użytkownika strony `DistanceTravelledPage`. Strona ta będzie wywoływana, gdy użytkownik kliknie przycisk *Rozpocznij ten szlak* na stronie treści `WalksTrailPage`, która będzie służyła do prezentowania informacji na temat wybranego szlaku, jak również wbijania szpilki na kontrolce `Xamarin.Forms.Maps` oraz obliczania przebytego dystansu i czasu podróży:

1. Utwórz nową stronę `ContentPage` o nazwie `DistanceTravelledPage` w sposób opisany w poprzednim podrozdziale.
2. Otwórz plik *DistanceTravelledPage.cs* w edytorze kodu i wpisz pogrubione poniżej fragmenty kodu:

```
//
// DistanceTravelledPage.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Xamarin.Forms;
using Xamarin.Forms.Maps;
using TrackMyWalks.Models;

namespace TrackMyWalks
{
    public class DistanceTravelledPage : ContentPage
    {
```

3. Następnie do konstruktora `DistanceTravelledPage` dodaj parametr `walkItem` określający wybrany szlak, co zaznaczono poniżej pogrubieniem:

```
public DistanceTravelledPage(WalkEntries walkItem)
{
    Title = "Przebyty dystans";
```

4. Następnie zadeklaruj zmienną `trailMap`, która będzie wskazywała egzemplarz kontrolki `Xamarin.Forms.Maps` do utworzenia szpilki na mapie. Posługując się współrzędnymi geograficznymi, wpisz w pliku poniższe pogrubione fragmenty kodu:

```
// utworzenie egzemplarza obiektu mapy
var trailMap = new Map();
// wbicie szpilki w mapę dla wybranego szlaku
trailMap.Pins.Add(new Pin
{
    Type = PinType.Place,
    Label = walkItem.Title,
    Position = new Position(walkItem.Latitude, walkItem.Longitude)
});
// wyśrodkowanie mapy na początku szlaku
trailMap.MoveToRegion(MapSpan.FromCenterAndRadius(new
    Position(walkItem.Latitude, walkItem.Longitude),
    Distance.FromKilometers(1.0)));
```

5. Następnie zadeklaruj kilka obiektów `Label` do prezentowania informacji z obiektu `walkItem` przekazanego ze strony treści `WalkTrailPage`. Wpisz poniższe pogrubione fragmenty kodu:

```
var trailNameLabel = new Label()
{
    FontSize = 18,
    FontAttributes = FontAttributes.Bold,
    TextColor = Color.Black,
    Text = walkItem.Title
};
var trailDistanceTravelledLabel = new Label()
{
    FontAttributes = FontAttributes.Bold,
    FontSize = 20,
    TextColor = Color.Black,
    Text = "Przebyty dystans:",
    HorizontalTextAlignment = TextAlignment.Center
};
var totalDistanceTaken = new Label()
{
    FontAttributes = FontAttributes.Bold,
    FontSize = 20,
    TextColor = Color.Black,
    Text = $"{walkItem.Distance} km",
    HorizontalTextAlignment = TextAlignment.Center
};
var totalTimeTakenLabel = new Label()
{
    FontAttributes = FontAttributes.Bold,
    FontSize = 20,
    TextColor = Color.Black,
    Text = "Czas:",
    HorizontalTextAlignment = TextAlignment.Center
};
var totalTimeTaken = new Label()
{
    FontAttributes = FontAttributes.Bold,
```

```

        FontSize = 20,
        TextColor = Color.Black,
        Text = "0 h 0 m 0 s",
        HorizontalTextAlignment = TextAlignment.Center
    };

```

6. Następnie zadeklaruj zmienną `walksHomeButton`, która dziedziczy po klasie `Button`, i przejdź do definicji procedury obsługi kliknięć, która w przypadku wykrycia kliknięcia będzie przenosić użytkownika do głównej strony treści `WalksPage`. Wpisz poniższe fragmenty kodu:

```

var walksHomeButton = new Button
{
    BackgroundColor = Color.FromHex("#008080"),
    TextColor = Color.White,
    Text = "Zakończ ten szlak"
};
// definicja procedury obsługi zdarzeń
walksHomeButton.Clicked += (sender, e) =>
{
    if (walkItem == null) return;
    Navigation.PopToRootAsync(true);
    walkItem = null;
};

```

7. Następnie zdefiniuj kontrolkę `ScrollView` wchodzącą w skład klasy bazowej `Xamarin.Forms.Core` oraz dodaj wszystkie pola formularza `Button` i `Label` do kontrolki `StackLayout`. Wpisz poniższy fragment kodu:

```

this.Content = new ScrollView
{
    Padding = 10,
    Content = new StackLayout
    {
        Orientation = StackOrientation.Vertical,
        HorizontalOptions = LayoutOptions.FillAndExpand,
        Children = {
            trailMap,
            trailNameLabel,
            trailDistanceTravelledLabel,
            totalDistanceTaken,
            totalTimeTakenLabel,
            totalTimeTaken,
            walksHomeButton
        }
    }
};
}
}
}
}

```

Pracę zaczęliśmy od zaimportowania klasy `TrackMyWalks.Models`, ponieważ za jej pomocą będziemy pobierać informacje przekazywane ze strony `WalksPage`. Pakiet `NuGet Xamarin.Forms.Maps` jest wieloplatformową biblioteką, za pomocą której programista może wyświetlać i oznaczać informacje na mapie. My użyjemy tej kontrolki do wyświetlenia szpilki na mapie wraz z dodatkowymi informacjami dotyczącymi danego szlaku.

Następnie deklarujemy zmienną `trailMap` wskazującą egzemplarz kontrolki `Xamarin.Forms.Maps` i tworzymy znacznik w postaci szpilki na mapie zawierającej dane dotyczące wybranego szlaku, po czym centrujemy go na mapie, aby ukazać cały obszar objęty przez szlak na podstawie współrzędnych szerokości i długości geograficznej. Potem deklarujemy i inicjalizujemy kilka obiektów `Label` zawierających informacje `walkItem`, które zostały przekazane ze strony treści `WalkTrailPage`, po czym deklarujemy zmienną `walksHomeButton`, która dziedziczy po klasie `Button`, i konfigurujemy zdarzenie `Clicked` dla klasy `Button`, przy użyciu którego będziemy wracać do `TrackMyWalks`.

Na koniec definiujemy kontrolkę `ScrollView`, która wchodzi w skład klasy bazowej `Xamarin.Forms.Core`, i dodajemy wszystkie pola formularza `Button` i `Label` do kontrolki `StackLayout`.

W następnej kolejności musimy zainicjalizować pakiet `NuGet Xamarin.Forms.Maps` w klasach startowych dotyczących każdej z naszych konkretnych platform (np. `AppDelegate.cs` w przypadku systemu iOS i `MainActivity.cs` w przypadku systemu Android). Oto opis, jak to zrobić krok po kroku.

1. Otwórz plik `AppDelegate.cs` w edytorze kodu i dodaj do niego fragment kodu zaznaczony pogrubieniem:

```
//
// AppDelegate.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Foundation;
using UIKit;

namespace TrackMyWalks.iOS
{
    [Register("AppDelegate")]
    public partial class AppDelegate :
    global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
    {
        public override bool FinishedLaunching(UIApplication app,
        ↪NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init();
            Xamarin.Forms.Maps.Init();
            LoadApplication(new App());
            return base.FinishedLaunching(app, options);
        }
    }
}
```

Najpierw zaznaczyliśmy, że nasza klasa `AppDelegate` ma korzystać z biblioteki `Xamarin.Forms.Maps`, dodając wywołanie metody `Xamarin.Forms.Maps.Init`, która inicjalizuje platformę `Xamarin.Forms`, dzięki czemu nasze rozwiązanie *TrackMyWalks* może wykorzystywać mapy. Gdyby zabrakło tego wywołania w tym miejscu, na stronie treści `DistanceTravelledPage` nie pojawiałaby się mapa i strona ta nie działałaby zgodnie z oczekiwaniami.

Więcej informacji na temat biblioteki `Xamarin.Forms.Maps` i różnych rodzajów dostępnych klas można znaleźć w dokumentacji Xamarin pod adresem <https://developer.xamarin.com/api/namespace/Xamarin.Forms.Maps/>.

## Tworzenie strony ekranu startowego

W tym podrozdziale rozpoczniemy budowę interfejsu użytkownika dla naszej strony startowej, która będzie wykorzystywana tylko na platformie Android, ponieważ w systemie iOS jest dostępna odpowiednia metoda. Na razie nasza strona startowa (ang. *splash page*) będzie zawierała jedynie domyślną ikonę Xamarin, ale później zmienimy ją na coś bardziej odpowiedniego dla naszej aplikacji.

1. Utwórz nową stronę `ContentPage` o nazwie `SplashPage` w sposób opisany w poprzednim podrozdziale.
2. Otwórz plik `SplashPage.cs` w edytorze kodu i wpisz fragment kodu zaznaczony pogrubieniem:

```
//
// SplashPage.cs
// TrackMyWalks
//
// Autor: Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using System;
using System.Threading.Tasks;
using Xamarin.Forms;
```

```
namespace TrackMyWalks
{
    public class SplashPage : ContentPage
    {
```

Następnie znajdź konstruktor `SplashPage` i wpisz poniższy pogrubiony kod:

```
public SplashPage()
{
    AbsolutePath splashLayout = new AbsoluteLayout
    {
        HeightRequest = 600
    };
    var image = new Image()
}
```

```

        Source = ImageSource.FromFile("icon.png"),
        Aspect = Aspect.AspectFill,
    };

    AbsoluteLayout.SetLayoutFlags(image, AbsoluteLayoutFlags.All);
    AbsoluteLayout.SetLayoutBounds(image, new Rectangle(0f, 0f,
        1f, 1f));

    splashLayout.Children.Add(image);

    Content = new StackLayout()
    {
        Children = { splashLayout }
    };
}

```

3. W dalszej kolejności znajdź metodę `OnAppearing` i wpisz kod:

```

protected override async void OnAppearing()
{
    base.OnAppearing();

    // kilkusekundowe opóźnienie wyłączenia ekranu startowego
    await Task.Delay(3000);

    // utworzenie egzemplarza NavigationPage i przypisanie go do MainPage
    var navPage = new NavigationPage(new WalksPage())
    {
        Title = "Track My Walks"
    });
    Application.Current.MainPage = navPage;
}
}

```

Najpierw importujemy klasę `System.Threading.Tasks`, za pomocą której będziemy mogli zdefiniować krótkie opóźnienie pozwalające użytkownikom przez chwilę oglądać ekran startowy zgodnie z definicją w metodzie klasowej `OnAppearing`.

Następnie deklarujemy zmienną `splashLayout` typu `AbsoluteLayout`, przy użyciu której proporcjonalnie ustawiamy położenie i rozmiar każdego elementu potomnego w naszym widoku, a potem ustawiamy własność `HeightRequest`.

Później deklarujemy zmienną `image` dziedziczącą po klasie `Image`, własności `Source` przypisujemy obraz, którego chcielibyśmy użyć, a własność `Aspect` ustawiamy tak, aby obraz ten wypełniał cały widok.

Na zakończenie przypisujemy wartości własnościom `LayoutFlags` i `LayoutBounds` w klasie `AbsoluteLayout` w taki sposób, aby obraz zmieniał rozmiar wraz z widokiem. Następnie dodajemy `splashLayout` do strony treści za pomocą kontrolki `StackLayout`. Przesłaniamy też metodę



OnAppearing cyklu życia strony Xamarin.Forms.Page, która to metoda jest wywoływana bezpośrednio przed pojawieniem się strony na ekranie, i definiujemy opóźnienie długości trzech sekund przed utworzeniem nowego egzemplarza strony NavigationPage, która wyznaczy naszą stronę WalksPage jako główną stronę treści.

## Modyfikacja klasy App Xamarin.Forms

W tej sekcji zainicjalizujemy klasę App z biblioteki Xamarin.Forms, aby wywoływała naszą stronę SplashPage i ustawiała stronę główną aplikacji, jeśli wykryje urządzenie z systemem operacyjnym Android.

Oto lista czynności do wykonania:

1. Otwórz plik *TrackMyWalks.cs* znajdujący się w grupie *TrackMyWalks* w edytorze kodu.
2. Znajdź metodę App i wpisz poniższy pogrubiony fragment kodu, jak pokazano na listingu:

```
//
// TrackMyWalks.cs
// TrackMyWalks
//
// Autor Steven F. Daniel, 04.08.2016
// Copyright © 2016 GENIESOFT STUDIOS. All rights reserved.
//
using Xamarin.Forms;

namespace TrackMyWalks
{
    public class App : Application
    {
        public App()
        {
            // sprawdzenie platformy
            if (Device.OS == TargetPlatform.Android)
            {
                MainPage = new SplashPage();
            }
            else
            {
                // strona główna aplikacji
                var navPage = new NavigationPage(new TrackMyWalks.WalksPage())
                {
                    Title = "Track My Walks"
                };
                MainPage = navPage;
            }
        }
    }
}
```

W kodzie tym najpierw za pomocą klasy `Device.OS` sprawdziliśmy, na czym działa nasza aplikacja `Xamarin.Forms`, następnie przy użyciu klasy `TargetPlatform` sprawdziliśmy, czy działa ona w systemie operacyjnym Google Android. Jeśli tak, to właściwości `MainPage` konstruktora `App` przypisujemy nowy egzemplarz strony `ContentPage`, który będzie wykorzystywał stronę `SplashPage` jako stronę główną. Jeśli wynik testu jest negatywny, tworzymy nowy egzemplarz klasy `NavigationPage`, który ustawiamy na naszą stronę `WalksPage`, po czym ustawiamy go jako główną stronę treści.

## Różnice między Xamarin Studio i Visual Studio

Jeżeli ktoś chce tworzyć aplikacje mobilne, to obecnie ma do wyboru dwa środowiska programistyczne: Xamarin Studio i Microsoft Visual Studio. Podkreślę, że choć wszystkie zrzuty ekranu i przykłady kodu pokazywane w tej książce zostały utworzone w Xamarin Studio na komputerze Apple Macintosh, to kod powinien bez problemu dać się skompilować także w systemie Windows w środowisku Microsoft Visual Studio 2015.

Zanim jednak na poważnie zagłębisz się w tajniki tworzenia aplikacji mobilnych, musisz wiedzieć, że te dwa środowiska dzielą istotne różnice. Jeśli korzystasz z Xamarin Studio w systemie Windows, to przy tworzeniu nowego rozwiązania `Xamarin.Forms` zawsze automatycznie otrzymasz rozwiązanie Android.

Jeśli chcesz integrować i tworzyć aplikacje dla systemu Windows Phone, musisz posługiwać się środowiskiem Microsoft Visual Studio w systemie Windows. Aby tworzyć aplikacje dla systemu iOS, musisz przygotować Maca do roli hosta kompilacji Xamarin, włączając *zdalne logowanie* w preferencjach systemowych i wybierając Maca jako host kompilacji w środowisku Microsoft Visual Studio na komputerze z systemem Windows.

Więcej informacji na temat przygotowania Maca do roli hosta kompilacji Xamarin znajduje się w dokumentacji środowiska Xamarin na stronie [https://developer.xamarin.com/guides/ios/getting\\_started/installation/windows/connecting-to-mac/](https://developer.xamarin.com/guides/ios/getting_started/installation/windows/connecting-to-mac/).

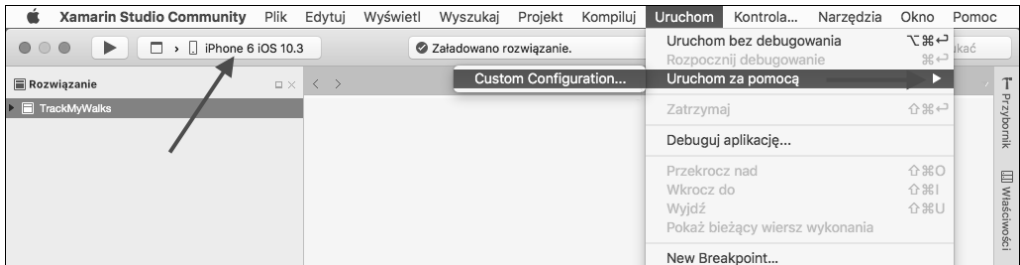
Znając podstawowe różnice między Xamarin Studio i Microsoft Visual Studio, możemy przejść do kompilacji, budowy i uruchomienia naszej aplikacji `TrackMyWalks` w symulatorze systemu iOS.

# Uruchamianie aplikacji TrackMyWalks w symulatorze

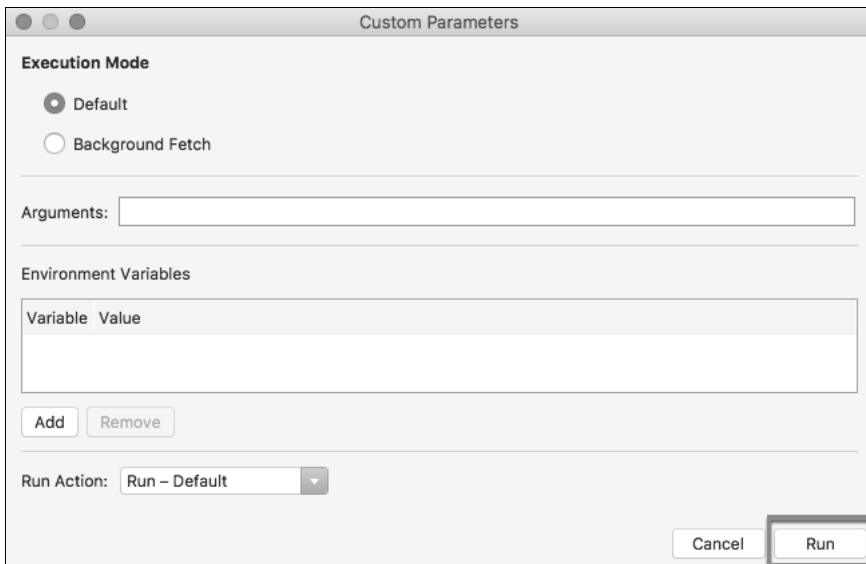
W tym podrozdziale dowiesz się, jak skompilować i uruchomić aplikację *TrackMyWalks*. Możesz ją uruchomić w prawdziwym urządzeniu lub wybrać jeden z kilku dostępnych symulatorów urządzeń z systemem iOS.

Numer wersji symulatora zależy od wersji pakietu SDK systemu iOS zainstalowanego w komputerze. Wykonaj następujące czynności:

1. Aby uruchomić aplikację, wybierz urządzenie z listy dostępnych symulatorów systemu iOS i kliknij opcję menu *Uruchom*.

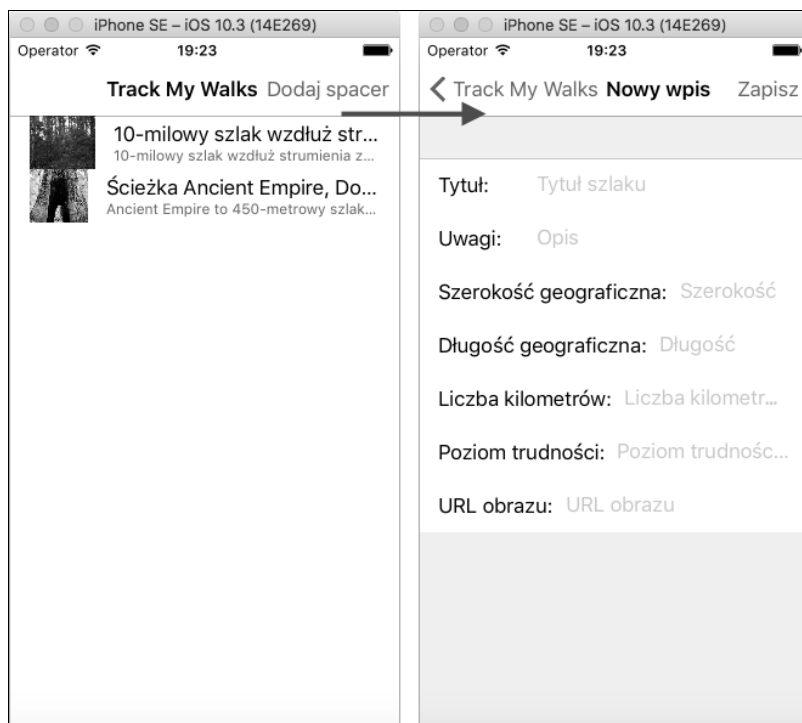


2. Następnie otwórz podmenu *Uruchom za pomocą*, kliknij opcję *Custom Configuration...* (konfiguracja użytkownika), po czym kliknij przycisk *Run* (uruchom) w oknie dialogowym *Custom Parameters* (parametry użytkownika).



3. Ewentualnie możesz skompilować i uruchomić aplikację *TrackMyWalks* naciśnięciem kombinacji klawiszy *Command+Return*.

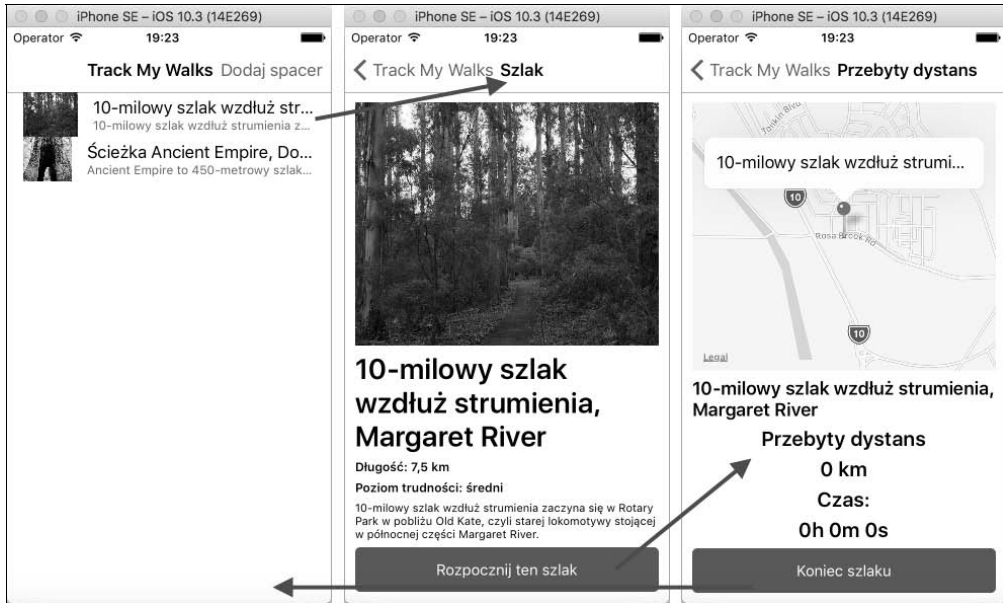
Po zakończeniu kompilacji pojawi się okno symulatora systemu iOS z uruchomioną aplikacją *TrackMyWalks*, co przedstawiono na poniższym zrzucie ekranu:



Jak widać na powyższym rysunku, obecnie aplikacja wyświetla listę statycznych wpisów na temat szlaków, które to wpisy są prezentowane w widoku `ListView`. Gdy użytkownik kliknie przycisk *Dodaj szlak*, zostanie wyświetlona strona treści `WalkEntry`, na której można wprowadzić dane dotyczące kolejnego szlaku.

Obecnie strona ta jeszcze nie zapisuje wprowadzonych informacji, ale w następnych rozdziałach dodamy odpowiednie mechanizmy. Kliknięcie przycisku *Zapisz* będzie powodowało powrót do głównej strony aplikacji *TrackMyWalks* (zobacz rysunek na następnej stronie).

Na zrzucie ekranu przedstawiono sposób nawigacji między stronami, gdy użytkownik wybierze z listy szlak. Na ostatnim ekranie pokazany jest przebyty dystans wraz ze szpilką wskazującą miejsce szlaku na mapie.



Gratulacje! Właśnie ukończyłeś prace nad podstawową strukturą aplikacji *TrackMyWalks* i interfejsem użytkownika wszystkich stron treści, które będą wykorzystywane w tej aplikacji. W następnych rozdziałach zastosujemy w naszej aplikacji lepsze wzorce projektowe, zdefiniujemy ładniejsze elementy interfejsu użytkownika oraz dodamy mechanizm synchronizacji danych na bieżąco oparty na REST-owych interfejsach API usług sieciowych.

## Podsumowanie

W tym rozdziale pokazałem Ci, jak utworzyć wieloplatformową aplikację *Xamarin.Forms* dla systemów iOS i Android. Potem opisałem kilka stron treści ze statycznymi danymi.

W następnym rozdziale przyjrzymy się, jak za pomocą domyślnych interfejsów API nawigacji *Xamarin.Forms* można przechodzić między stronami z treścią (które w znacznym stopniu omówimy w rozdziale 3.) z zastosowaniem w nich elastyczniejszej usługi nawigacyjnej, łatwiej poddającej się konfiguracji. Na koniec omówiłem najważniejsze różnice między środowiskami programistycznymi *Xamarin Studio* i *Microsoft Visual Studio* oraz pokazałem, jak uruchomić aplikację *TrackMyWalks* w symulatorze.

W kolejnym rozdziale poznasz wzorec architekuralny MVVM (ang. *Model-View-ViewModel* — „model – widok – model widoku”) i dowiesz się, jak go zaimplementować w swojej aplikacji, oraz nauczysz się dodawać nowe modele widoków do swojego rozwiązania *Xamarin*.



# Skorowidz

## A

Android, 26, 45, 48, 53, 151, 278, 291, 307, 319  
kontrolka, 169, 174  
typ pola, 310  
aplikacja  
debugowanie, 309  
ekran, 307  
identyfikator, 331  
UDID, 339  
interakcja, 307  
kompilowanie, 49, 50  
mobilna, 223  
profil dostarczania, 333  
publicznie dostepna na Facebooku, 259  
sciezka do pakietu, 309  
uruchamianie, 49, 50  
uruchamianie w symulatorze, 309  
Apple App Store, 321, 322, 328, 333, 344  
Member Center, 336  
opcja cenowa, 343  
Azure, 222, 236, 243  
komunikacja, 238  
konto, 222

## B

baza danych  
relacyjna, 201  
wykluczania wzajemne, 202  
biblioteka, *Patrz też:* pakiet  
.NET Xamarin.Auth, 259  
Moq, 293, 297, 298, 299  
NUnit.Framework, 302  
Portable Class Library, 23, 29

Shared Library, 23  
SQLite, 203  
sqlite-net, 199  
System.Net.Http.HttpClient, 229, 232  
Xamarin.Forms.Maps, 45  
Bundle ID, 342

## D

dane  
baza, *Patrz:* baza danych  
szablon, *Patrz:* szablon danych  
typ, *Patrz:* typ  
dyrektywa @model, 208, 211

## E

efekty, 172, 174, 186  
ekran  
aplikacji, 307  
startowy, 45, 47, *Patrz też:* strona startowa  
rzzut, 309  
element potomny, 46  
etykieta, 55  
EntryCell, 37

## F

Facebook, 256, 257, 310  
uwierzytelnianie, 257, 259, 261, 269, 311  
OAuth 2.0, 259  
token, 265, 266, 267, 272  
format JSON, 227  
funkcja SSO, 283

## G

geolokalizacja, *Patrz:* lokalizacja

## I

instrukcja using, 161

interfejs, 87

API, 82, 156, 169, 170, 174, 222, 227

Facebooka, 259

Graph, 267, 268

Effects Xamarin.Forms, 169, 170, 174

ILocationListener, 127

INotifyCollectionChanged, 55

INotifyPropertyChanged, 55, 56

INotifyPropertyChangedInterface, 58

lokalizacji, 122

NavigationService, 273

usługi nawigacji, 84

tworzenie, 85, 87

użytkownika, 34, 53, 54, 156,

*Patrz też:* widok

ekran startowy, *Patrz:* ekran startowy,

strona startowa

testowanie, 305, 316

tworzenie, 37, 41

wskaźnik aktywności, 183

Xamarin.Forms.INavigation, 82, 83

iOS, 45, 48, 53, 151, 164, 195, 278, 279, 283,

291, 307, 315, 319

certyfi­kat programistyczny, 326, 328

algorytm, 326

długość klucza, 326

kontrolka, 169, 170

typ pola, 310

wersja, 137

iTunes Connect, 321, 323

powiadomienia, 324

pozycja tworzenie, 340

rola, 323

użytkownik, 323

## J

język

C#, 21, 314

F#, 21

XAML, 53

## K

klasa

AbsoluteLayout, 46

App, 26, 47

AppDelegate, 315

AppInitializer, 308, 309, 310

AppQuery, 308

Assert, 300, 301

BooleanConverter, 178

Button, 39, 43

CLLocationManager, 131, 133

ConfigureApp, 307, 308, 309

DataService, 235, 238, 241

DataTemplate, 155, 156

DataTemplates, 160

odwołanie, 161

Device, 151, 152

EntryCellEditText, 310

FacebookApiAuthToken, 265, 272, 281

FacebookApiUser, 256, 264, 279

FacebookAuthToken, 267

FacebookCredentials, 266, 281

HttpClient, 233, 234

HttpClientHandler, 233, 234

HttpMessageHandler, 233

HttpMethod, 238

HttpRequestMessage, 233

IApp, 307, 308

Image, 46

IValueConverter, 178, 179

Java.Lang.Object, 127

JsonProperty, 231

ListView, 161

LocationEventArgs, 127

MainActivity, 149, 218

Mock, 297, 299

modeli widoków, 56, 58

NavigationService, 273, 274

Newtonsoft.Json, 231

NSAppTransportSecurity, 285

NUnit.Framework.Test, 302

OAuth2Authenticator, 257, 271

OAuth2Request, 269

opakowująca, 199, 205

PageViewModel, 243

PlatformEffect, 171, 172, 173, 174, 176

PlatformEffects, 183, 186, 189



SQLite, 218  
 SQLiteConnection, 203  
 System.IO, 218  
 System.Threading.Tasks, 46  
 TargetPlatform, 151  
 Task, 88  
 testowa, 296  
     TestFixture, 297, 298, 299, 301  
 ToolbarItem, 34  
 UIAlertView, 131  
 UIPickerView, 168  
 UIPickerViewModel, 165  
 UITextField, 310  
 usługi nawigacji, 84, 88  
 ViewRenderer, 271, 273  
 WebViewController, 208, 217  
 WebViewController., 218  
 Xamarin.Auth, 269  
 Xamarin.Forms.App, 118, 150, 282  
 Xamarin.Forms.Application, 26  
 Xamarin.Forms.Core, 43, 44  
 konsola klienta REST, 227  
 kontrolka, 156  
     EntryCell, 162, 163  
     formularza, 55  
     ListView, 31, 162, 184, 245, 252  
     progressLabel, 184  
     ScrollView, 39, 43, 44  
     StackLayout, 44  
     TableView, 37  
     UIPickerView, 168, 169  
     Xamarin.Forms.Map, 39  
 konwerter  
     BooleanConverter, 177  
     wartości, 153, 177  
         logicznych, 178, 180

## L

logika biznesowa, 26, 291, 305  
 lokalizacja, 143  
     Android, 124  
     iOS, 130  
     monitorowanie w tle, 134, 136  
     uwierzelnianie, 132, 133, 134

## M

menedżer pakietów, 198  
 metoda  
     AllowsBackgroundLocationUpdates, 137  
     AppBundle, 309  
     AreEqual, 300, 301  
     asynchroniczna, 88  
     Class, 308  
     ClearText, 307  
     Convert, 178, 179  
     ConvertBack, 178, 179  
     Css, 308  
     Debug, 309  
     DeleteItem, 204  
     Detached, 176  
     DeviceIdentifier, 309  
     didAuthorizationChange, 133  
     DisplayActionSheet, 247, 248  
     Effect.Resolve, 186  
     EnableLocalScreenshots, 309  
     EnterText, 307  
     ExecuteSaveCommand, 142  
     FacebookApiUser, 266  
     FinishedLaunching., 315  
     GenerateString, 213, 214  
     GetItem, 203  
     GetItems, 203, 213  
     GetMyLocation, 131  
     GetProfileInformation, 269  
     GetResponseAsync, 268  
     HttpMethod GET, 232  
     HttpMethod.Delete, 237, 238  
     HttpMethod.Get, 237, 238  
     HttpMethod.Post, 237, 238  
     InsertPageBefore, 83  
     JsonConvert, 233  
     LoadHtmlString, 213, 214  
     Marked, 308, 313  
     NavigateToViewModel, 87  
     NetworkProvider, 129  
     OnAppearing, 46  
         przesłanianie, 47  
     OnAttached, 171, 176  
     OnDetached, 171, 173, 175  
     OnLocationChanged, 127, 128  
     OnPropertyChanged, 58  
     OnProviderDisabled, 127, 128

metoda

- OnProviderEnabled, 127, 128
- OnStatusChanged, 127, 128
- PopAsync, 83
- PopModalAsync, 83
- PopToRootAsync, 83
- PostTaskAsync, 267
- PushAsync, 83
- PushModalAsync, 83, 279
- Query, 307, 308
- RemovePage, 83
- Repl, 307, 309
- RequestAlwaysAuthorization, 137
- SaveItem, 204
- Screenshot, 307
- SendRequestAsync, 237
- SetDatabaseConnection, 206
- StartApp, 308, 309
- Tap, 307, 313
- TargetPlatform, 151
- testowa, 302
- UITest, 307
- WaitForElement, 307, 308
- zwrotna, 284

Microsoft Azure, *Patrz też*: Azure

Microsoft Development Platform, 27

Microsoft Visual Studio, 48

model, 53, 54, 55

- danych, 53
  - FacebookApiUser, 264
  - TrackMyWalks, 29, 31
- widoku, 53, 54, 55, 61, 94, 97, 100, 107, 111, 138, 180, 186, 188, 243, 245, 274
  - implementowanie, 55, 59, 60, 63, 70
  - testowanie, 295
  - tworzenie, 73

MVVM, 53, 54, 56

**N**

nawigacja, 88

- hierarchiczna, 82
- implementowanie, 84
- modalna, 82
- przy użyciu bezpośrednich odwołań, 83
- Xamarin.Forms, 82

nazwa odwrócona kolejność elementów adresu domeny, 24, 331

**O**

obiekt

- AppResult, 307
- ContentPage, 27
- Context, 128
- control, 174, 176
- danych, 26
- DataTemplate, 31
- EntryCell, 34
- EventHandler, 128, 131
- Label, 34
- pickerView, 165
- ToolBarItem, 37
- wiązalny, 70

okienko dialogowe, 133

**P**

pakiet, *Patrz też*: biblioteka

- aktualizowanie, 27
- NuGet, 27, 198
  - HttpClient, 229
  - Json.Net, 227, 228
  - Moq, 293, 297, 298, 299
  - SQLite.Net, 197, 198
  - Xamarin Test Cloud Agent, 314, 315, 319
  - Xamarin.Auth, 259, 261, 283
  - Xamarin.Forms.Maps, 39, 41, 44, 45
- SDK, 49
  - Facebooka, 259, 261, 263, 283
  - Xamarin.Forms, 27

pamięci zwalnianie, 142

parametr

- ściśle typizowany, 88
- TParameter, 87

pasek

- narzędzi, 167
- NavigationBar, 148
- nawigacji, 277

plik

- .cs, 26, 31
- .cshtml, 196
- AppDelegate.cs, 217
- AppInitializer.cs, 305
- info.plist, 342
- MainActivity.cs, 217
- style.css, 197, 218

szablonowy domyślny, 24  
 Test.cs, 305  
 WebViewController.cs, 199, 200

projekt  
 obszar, 25  
 PCL, 26  
 TrackMyWalks, 22, 26  
 tworzenie, 22

protokół  
 App Transport Security, 285  
 HTTP, 221, 238

przycisk, 55, 281

## R

Razor, 194, 199, 206, 208, 212  
 ASP.NET, 194  
 tworzenie, 194, 207, 209, 210, 215  
 wywołanie, 208

renderer, 156  
 tworzenie, 162, 164  
 Xamarin.Forms.ViewCell, 160

renderowanie, 155

## S

sieć komórkowa, 129

silnik szablonu Razor, *Patrz:* Razor

słownik, 84

splash page, *Patrz:* strona startowa

stos NavigationStack, 276

strona  
 Client OAuth Settings, 257  
 ContentPage, 34, 37  
 tworzenie, 31  
 główna szablon danych, 160  
 przystosowywanie do MVVM, 61, 67, 71, 76,  
 97, 104, 109, 115  
 startowa, 45, 147

strongly typed, *Patrz:* parametr ściśle typizowany

symulator, 49

system  
 Android, *Patrz:* Android  
 iOS, *Patrz:* iOS  
 Windows Phone, *Patrz:* Windows Phone

szablon danych, 156  
 na stronie głównej, 160  
 Razor, *Patrz:* Razor

## Ś

środowisko  
 Calabash, 316  
 NUnit, 290, 291, 314  
 NUnit.Test, 301  
 testowe, 290, 291  
 UITest, 290, 305, 307  
 Xamarin Studio, 290, 291, 305, 314, 344,  
 349, 351  
 Xcode, 339, 340

## T

test  
 akceptacyjny, 316  
 jednostkowy, 291  
 tworzenie, 296  
 uruchamianie, 302, 303, 304  
 warunek testowy, 302  
 NUnit, 290, 291, 296, 298  
 UITest, 290, 305, 318  
 wstrzymanie, 307, 309  
 Xamarin.UITest, 314

TrackMyWalks model, 29, 31

typ  
 AbsoluteLayout, 46  
 EntryCellEditText, 310  
 IDictionary, 236  
 List, 208  
 PropertyChangedEventHandler, 58  
 StackLayout, 185  
 UITextField, 310  
 UIWebView, 214

## U

usługa  
 IaaS, 222  
 lokalizacji, *Patrz:* lokalizacja  
 PaaS, 222  
 SaaS, 222  
 sieciowa chmurowa, 221  
 Xamarin.Forms.DependencyService, 127

**W**

wiązanie, 190  
    BindingMode.OneWay, 69, 70  
    BindingMode.OneWayToSource, 70  
    BindingMode.TwoWay, 70  
widok, 26, 53, 54, 55, 274  
    StackLayout, 34  
    TableView, 34, 37, 162  
    UIPickerView, 165  
wiersz poleceń, 227  
Wi-Fi, 129  
Windows, 151, 177  
Windows Phone, 48, 53, 151, 307  
wzorzec model – widok – model widoku,  
    *Patrz:* MVVM

**X**

Xamarin Studio, 48  
Xamarin.Forms, 22, 27

**Z**

zdarzenie  
    Clicked, 39, 109, 117  
    CollectionChanged, 55  
    Completed, 272  
    PropertyChanged, 55  
zespół programistów iOS, 321, 322

**Ż**

żądanie HTTP, 227

# 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>

## Oto Xamarin: funkcjonalność, elastyczność, niezależność od platformy!

Platforma Xamarin jest wszechstronnym, nowoczesnym narzędziem do budowania aplikacji działających w wielu systemach. Pozwala na pisanie kodu w języku C# i ułatwia tworzenie niestandardowych widoków, układów i kontrolek. Interfejsy graficzne tworzone w Xamarin urzekają swoją estetyką. Platformy Xamarin i Xamarin.Forms zdobywają coraz większą popularność i właściwie stają się standardową technologią pisania oprogramowania na urządzenia mobilne.

Dzięki tej książce dowiesz się, jak zaimplementować struktury i układy interfejsu użytkownika, a także jak tworzyć własne elementy i pisać służące do ich obsługi skrypty C#. Poznasz architekturę MVVM i nauczysz się implementować ją w swoich aplikacjach. Zrozumiesz trudniejsze zagadnienia, takie jak włączanie do aplikacji funkcji specyficznych dla danej platformy mobilnej, współpraca z usługami Microsoft Azure App, korzystanie z pakietu SDK Facebooka oraz obsługa zewnętrznych bibliotek, takich jak Razor. Dowiesz się też, jak zaimplementować kluczowe techniki wiązania danych i efektów wizualnych w interfejsach użytkownika za pomocą własnych rendererów i interfejsu API PlatformEffects.

### Najważniejsze zagadnienia:

- model aplikacji w postaci klas C#
- wzorzec MVVM i implementacja architektury zgodnej z tym wzorcem
- funkcje zależne od lokalizacji
- komponenty współpracujące z bazą SQLite
- testy aplikacji za pomocą narzędzi NUnit i UTest

**Steven F. Daniel** — jest prezesem australijskiej firmy programistycznej GENIESOFT STUDIOS. Od niemal 20 lat tworzy oprogramowanie na komputery PC i urządzenia mobilne. Jest znany z tego, że chętnie dzieli się swoim bogatym doświadczeniem z programistami, a środowisko Xamarin to jedno z jego ulubionych narzędzi. Daniel należy do SQLSIG (SQL Server Special Interest Group), Melbourne CocoaHeads i Java Community.

|                                                                                                                             |                                                                                                                                                                                       |                                                |                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|-------------------------------------------------------------------------------------|
|                                           | <i>Sprawdź nasze szkolenia</i>                                                                                                                                                        | <b>KOD KORZYŚCI</b><br><i>Śledź po więcej!</i> |  |
|  <a href="http://helion.pl">helion.pl</a> | <br>AKADEMIA IT & BUSINESS<br><a href="http://WWW.SZKOLENIA.HELION.PL">WWW.SZKOLENIA.HELION.PL</a> | ISBN 978-83-283-3948-4                         |  |
|  <b>0 801 339900</b>                      |                                                                                                                                                                                       | 9 788328 339484                                |                                                                                     |
|  <b>0 601 339900</b>                      |                                                                                                                                                                                       |                                                |                                                                                     |
| <b>INFORMATYKA W NAJLEPSZYM WYDANIU</b>                                                                                     |                                                                                                                                                                                       | Cena: 67,00 zł                                 |                                                                                     |

**Packt**