Mark J. Price

C# 7.1 i .NET Core 2.0

dla programistów aplikacji wieloplatformowych

Helion

Packt›

Tytuł oryginału: C# 7.1 and .NET Core 2.0 - Modern Cross-Platform Development

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-4450-1

HELION SA
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/c71nc2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/c71nc2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Cześć, C#! Witaj, .NET Core

W tym rozdziale zajmiemy się przygotowaniem środowiska programistycznego i przedstawimy podobieństwa i różnice pomiędzy .NET Core, .NET Framework, .NET Standard i .NET Native. Spróbujemy też użyć najróżniejszych narzędzi, żeby przygotować bardzo prostą aplikację z wykorzystaniem C# i .NET Core.

Większość ludzi uczy się złożonych rzeczy przez powtarzanie widzianych już akcji, a nie przez czytanie wyczerpujących opisów teoretycznych. Dlatego nie będę tu objaśniał każdego użytego słowa kluczowego i kroku. Chodzi o to, żeby nauczyć Cię pisania kodu, budowania aplikacji i uruchamiania jej. Na razie nie musisz znać każdego szczegółu ani wiedzieć dokładnie, jak to wszystko działa.

Przywołując słowa Samuela Johnsona, autora słownika języka angielskiego z 1755 r., najpewniej popełniłem tu „kilka błędów oraz sporych absurdów, od których nie jest wolna żadna praca tej wielkości". Biorę na siebie pełną odpowiedzialność za te błędy, mając nadzieję, że docenisz moje starania podczas pisania książki o platformie .NET Core i związanych z nią narzędziach już w czasie jej narodzin w latach 2016 i 2017.

W rozdziale zostaną omówione następujące zagadnienia:

- konfigurowanie środowiska programistycznego,
- poznawanie .NET,
- pisanie i kompilowanie kodu za pomocą narzędzi wiersza poleceń z .NET Core,
- pisanie i kompilowanie kodu za pomocą Visual Studio 2017,
- pisanie i kompilowanie kodu za pomocą Visual Studio Core,
- pisanie i kompilowanie kodu za pomocą Visual Studio for Mac,
- obsługa kodu źródłowego z wykorzystaniem platformy GitHub.

Konfigurowanie środowiska programistycznego

Zanim zaczniemy programować, trzeba wybrać **zintegrowane środowisko programistyczne** (ang. *Integrated Development Environment* — IDE), którego częścią jest edytor kodu dla języka C#. Microsoft udostępnia małą rodzinę takich środowisk:

- Visual Studio 2017,
- Visual Studio for Mac,
- Visual Studio Code.

Najbardziej rozbudowanym wariantem, oferującym najwięcej funkcji, jest **Microsoft Visual Studio 2017**, ale działa on jedynie w systemach operacyjnych Windows.

Najnowocześniejsze i najlżejsze IDE, jakie mamy do wyboru, i jedyne, które w rodzinie Microsoftu jest wieloplatformowe, to **Microsoft Visual Studio Code**. Działa ono w wielu różnych systemach operacyjnych, takich jak Windows, macOS, oraz w wielu wariantach Linuksa, np. **Red Hat Enterprise Linux** (**RHEL**) albo Ubuntu.

> W podjęciu decyzji, czy Visual Studio Code jest dla Ciebie właściwym wyborem, z pewnością pomoże obejrzenie wideo *Beginner's Guide to VS Code: Up and Running in Ten Minutes*: *https://channel9.msdn.com/Blogs/raw-tech/Beginners-Guide-to-VS-Code*.

IDE najlepiej dostosowane do rozwijania programów dla urządzeń mobilnych to **Visual Studio for Mac**. Aby tworzyć oprogramowanie dla systemów iOS (dla iPhone'a i iPada), tvOS, macOS i watchOS, trzeba mieć dostępny system macOS i środowisko Xcode. Co prawda można też skorzystać z Visual Studio 2017 z rozszerzeniem Xamarin do **napisania** wieloplatformowej aplikacji mobilnej, ale do jej **skompilowania** niezbędne będą macOS i Xcode.

W poniższej tabeli można sprawdzić, które IDE i który system operacyjny będziemy wykorzystywali w poszczególnych rozdziałach tej książki.

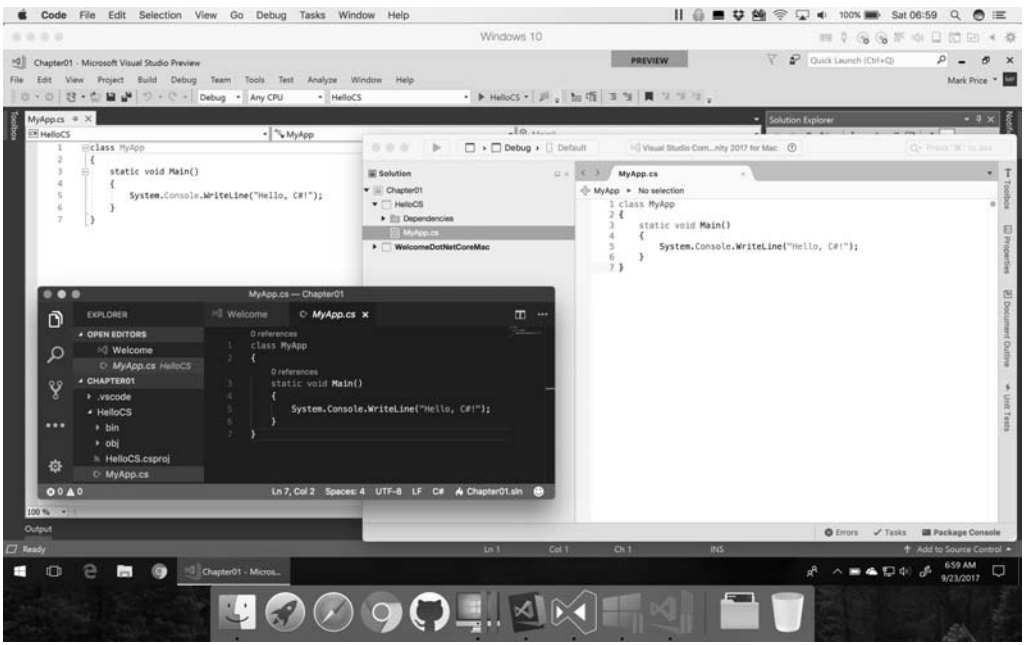
Rozdziały	IDE	Systemy operacyjne
od 1. do 16.	Visual Studio 2017	Windows 7 SP1 lub nowszy
od 1. do 16.	Visual Studio Code	Windows, macOS, Linux
od 1. do 16.	Visual Studio for Mac	macOS
17.	Visual Studio 2017	Windows 10
18.	Visual Studio for Mac	macOS

Dobra praktyka

Jeżeli masz taką możliwość, to zachęcam do wykonania wszystkich ćwiczeń w środowiskach Visual Studio 2017 dla Windowsa, Visual Studio Code w systemach macOS, Linux i Windows oraz Visual Studio for Mac. Dzięki temu zyskasz doświadczenie w pracy z językiem C# i platformą .NET Core w różnych systemach operacyjnych i z wykorzystaniem różnych narzędzi.

Pisząc trzecie wydanie tej książki, używałem oprogramowania wymienionego poniżej, przedstawionego na poniższym rysunku:

- Visual Studio 2017 w systemie Windows 10 w maszynie wirtualnej,
- Visual Studio for Mac w systemie macOS,
- Visual Studio Code w systemie macOS,
- Visual Studio Code w systemie RHEL (niewidoczne na rysunku).



Używanie innych IDE dla języka C#

Istnieją też inne IDE dla języka C#, np. **Mono Develop** albo **JetBrains Rider**. Możesz zainstalować dowolne z nich, korzystając z poniższych adresów URL:

- Środowisko MonoDevelop znajdziesz pod adresem *http://www.monodevelop.com/*.
- JetBrains Rider dostępne jest pod adresem *https://www.jetbrains.com/rider/*.

Instalowanie na wielu platformach

Wybór IDE i systemu operacyjnego na potrzeby tworzenia oprogramowania nie ogranicza nam możliwości instalowania go na innych platformach. .NET Core 2.0 pozwala zainstalować oprogramowanie na następujących platformach:

- Windows 7 SP1 i nowsze,
- Windows Server 2008 R2 SP1 i nowsze,
- Windows IoT 10 i nowsze,
- macOS Sierra (wersja 10.12) i nowsze,
- RHEL 7.3 i nowsze,
- Ubuntu 14.04 i nowsze,
- Fedora 25 i nowsze,
- Debian 8.7 i nowsze,
- openSUSE 42.2 i nowsze,
- Tizen 4 i nowsze.

> Systemy linuksowe są bardzo popularne na platformach hostujących usługi serwerowe ze względu na ich lekkość i możliwość tańszego skalowania w porównaniu do takich platform jak Windows lub macOS.

W następnym podrozdziale zainstalujemy Visual Studio 2017 w systemie Windows. Jeżeli wolisz używać Visual Studio Code, to możesz od razu przejść do podrozdziału „Instalowanie Microsoft Visual Studio Code" dla systemu Windows, macOS lub Linux. Jeżeli chcesz korzystać z Microsoft Visual Studio for Mac, to przejdź do podrozdziału „Instalowanie Visual Studio for Mac".

Instalowanie Microsoft Visual Studio 2017

Do wykonania zadań z większości rozdziałów tej książki wystarczy system Windows 7 SP1, ale znacznie lepszym rozwiązaniem jest zastosowanie systemu Windows 10 Fall Creators Update.

W październiku 2014 r. Microsoft udostępnił za darmo doskonale wyposażoną wersję Visual Studio, która otrzymała nazwę **Community Edition**.

Korzystając z poniższego adresu, możesz pobrać i zainstalować **Microsoft Visual Studio 2017 w wersji 15.4 lub nowszej**:

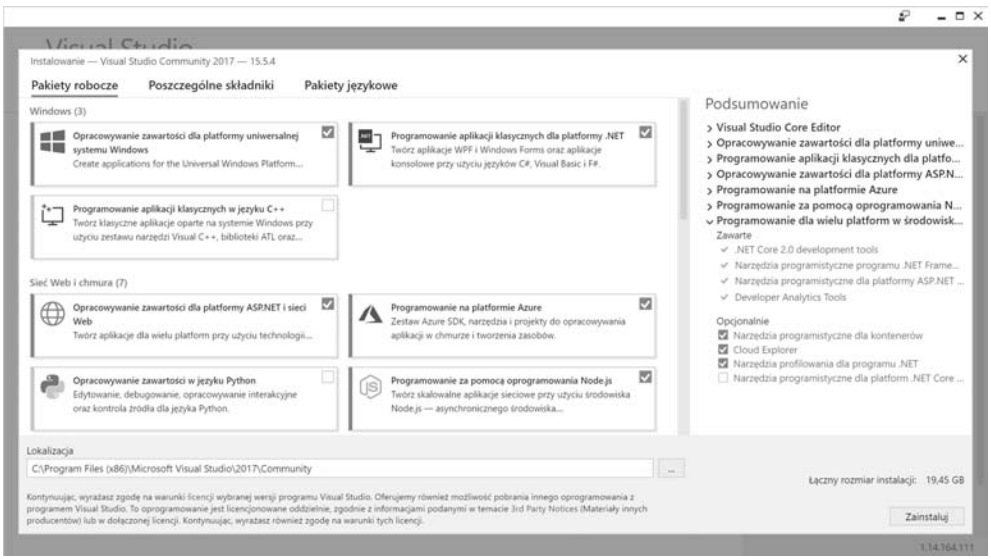
https://www.visualstudio.com/downloads/.

> Żeby móc skorzystać z platformy .NET Core dla UWP, musisz zainstalować Visual Studio 2017 w wersji 15.4 albo nowszej. Wersja 15.3 lub nowsza wymagana jest do pracy z platformą .NET Core 2.0. Starsze wersje Visual Studio 2017 pozwalają jedynie na obsługę .NET Core 1.0 i 1.1.

Wybieranie pakietów

Na karcie *Pakiety robocze* zaznacz poniższe elementy, które są też częściowo widoczne na poniższym rysunku:

- *Opracowanie zawartości dla platformy uniwersalnej systemu Windows*,
- *Programowanie aplikacji klasycznych dla platformy .NET*,
- *Opracowywanie zawartości dla platformy ASP.NET i sieci Web*,
- *Programowanie na platformie Azure*,
- *Programowanie za pomocą oprogramowania Node.js*,
- *Programowanie dla wielu platform w środowisku .NET Core*.



Wybieranie dodatkowych komponentów

Na karcie *Individual components* zaznacz jeszcze poniższe komponenty:

- *Class Designer*,
- *GitHub extension for Visual Studio*,
- *PowerShell tools*.

Kliknij przycisk *Install* i czekaj, aż instalator pobierze wybrane pakiety i je zainstaluje. Po zakończeniu instalacji kliknij przycisk *Launch*.

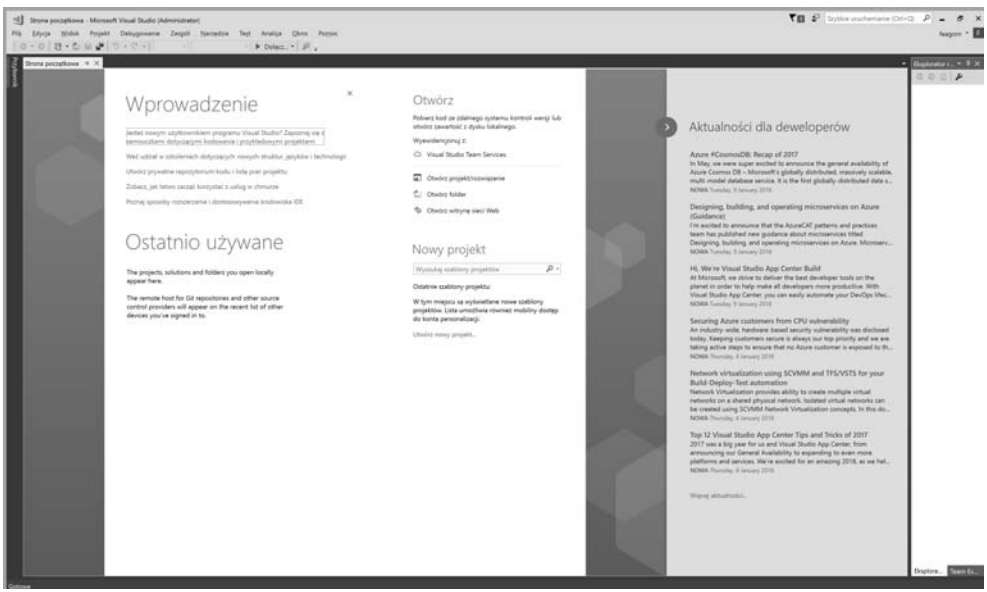
> Czekając na zainstalowanie Visual Studio 2017, możesz zajrzeć do podrozdziału „Poznawanie .NET" w dalszej części tego rozdziału.

Przy pierwszym uruchomieniu Visual Studio 2017 pojawi się prośba o zalogowanie. Jeżeli masz konto Microsoftu, to użyj go do zalogowania się. Jeżeli nie masz jeszcze konta, to możesz utworzyć nowe, korzystając z poniższego adresu:

http://signup.live.com/.

Uruchamiając pierwszy raz Visual Studio 2017, zobaczysz ekran zachęcający do skonfigurowania środowiska. W sekcji *Ustawienia* zaznacz opcję *Visual C#*. Jeżeli chodzi o motyw kolorów, to sam wybieram niebieski, ale tutaj każdy może zdecydować według własnego uznania.

Później zobaczysz interfejs użytkownika Microsoft Visual Studio ze *Stroną początkową* otwartą w środkowej części. Podobnie jak w większości aplikacji dla systemu Windows, Visual Studio ma własny pasek menu, pasek narzędzi z najważniejszymi poleceniami oraz pasek stanu na dole. Po prawej znajduje się *Eksplorator rozwiązań*, który zawiera listę otwartych projektów:



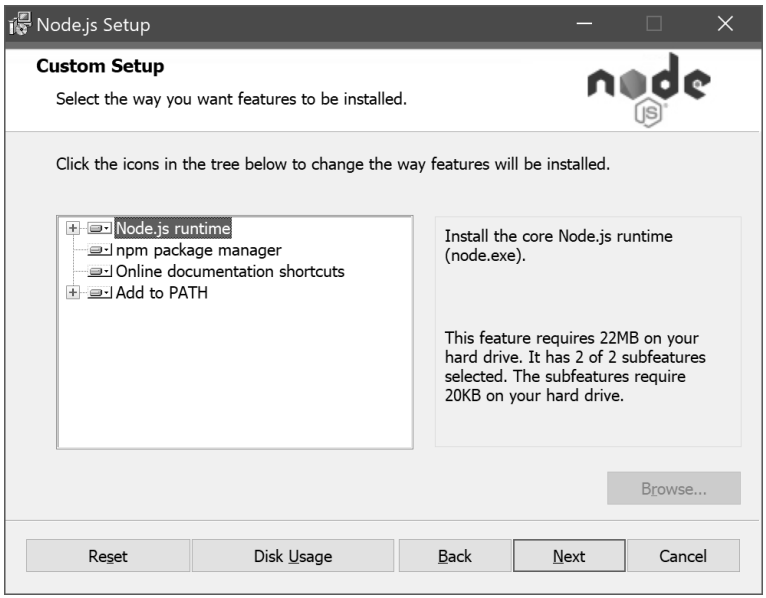
Jeśli chcesz mieć w przyszłości szybki dostęp do Visual Studio, możesz kliknąć prawym przyciskiem myszy jego ikonę na pasku zadań systemu Windows i wybrać opcję *Przypnij do paska zadań*.

Do prac z rozdziałów 14., 15. i 16. konieczne będzie zainstalowanie pakietów Node.js oraz NPM.

Instalator pakietu Node.js dla systemu Windows możesz pobrać z poniższego adresu:

https://nodejs.org/en/download/.

Po uruchomieniu instalatora zastosuj ustawienia jak na poniższym rysunku:



Instalowanie Microsoft Visual Studio Code

Pomiędzy czerwcem 2015 r. a wrześniem 2017 r. Microsoft niemal co miesiąc wydawał nową wersję **Visual Studio Code**. Pakiet ten rozwija się bardzo szybko i jego popularność chyba trochę zaskoczyła Microsoft. Nawet jeżeli planujesz korzystać z Visual Studio 2017 lub Visual Studio for Mac jako głównego narzędzia do pracy, to i tak zalecam zapoznanie się również z Visual Studio Code i narzędziami wiersza poleceń .NET Core.

Visual Studio Code możesz pobrać z poniższego adresu:

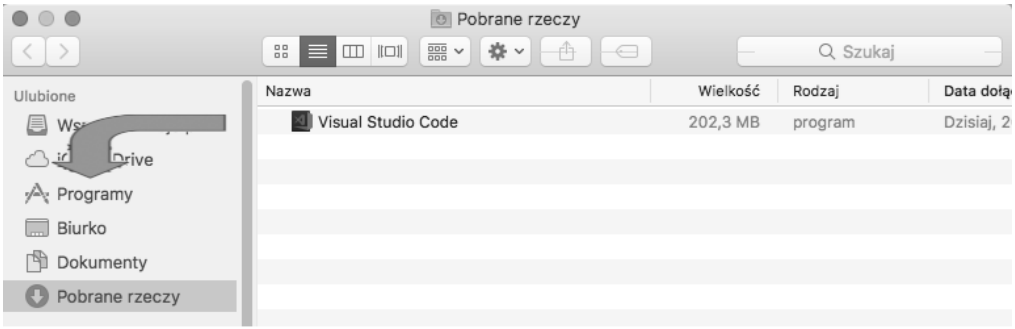
https://code.visualstudio.com/.

> O planach rozwoju Visual Studio Code w 2018 r. możesz poczytać pod tym adresem: *https://github.com/Microsoft/vscode/wiki/Roadmap*.

Instalowanie Microsoft Visual Studio Code for macOS

W książce przedstawię przykłady i zrzuty ekranów pochodzących z Visual Studio Code w wersji dla macOS. Kroki niezbędne do wykonania tych samych operacji w systemach Windows i Linux są bardzo podobne, dlatego nie będę powtarzał tu instrukcji dla każdego z tych systemów z osobna.

Po pobraniu pakietu Visual Studio for macOS musisz przeciągnąć go i upuścić na folder *Programy*, tak jak pokazano na poniższym rysunku:



Teraz musisz jeszcze zainstalować .NET Code SDK for macOS. Dokładne instrukcje instalacji, wraz z filmem instruktażowym, są dostępne pod poniższym adresem, a w tej książce zawarłem jeszcze proste kroki wykonania tej instalacji:

https://www.microsoft.com/net/core#macos.

W pierwszym kroku należy zainstalować Homebrew (o ile nie jest już zainstalowane).

Następnie musisz uruchomić aplikację *Terminal*, a po pojawieniu się wiersza poleceń wpisać w nim:

```
/usr/bin/ruby -e "$(curl -fsSL
↪https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Aplikacja *Terminala* poprosi o wciśnięcie klawisza *Enter*, a następnie zapyta o hasło użytkownika.

> Jeżeli używasz .NET Core w wersji 1.0 lub 1.1, to na tym etapie skorzystaj z Homebrew do zainstalowania pakietu OpenSSL, który jest wymagany przez wcześniejsze wersje .NET Core for macOS.

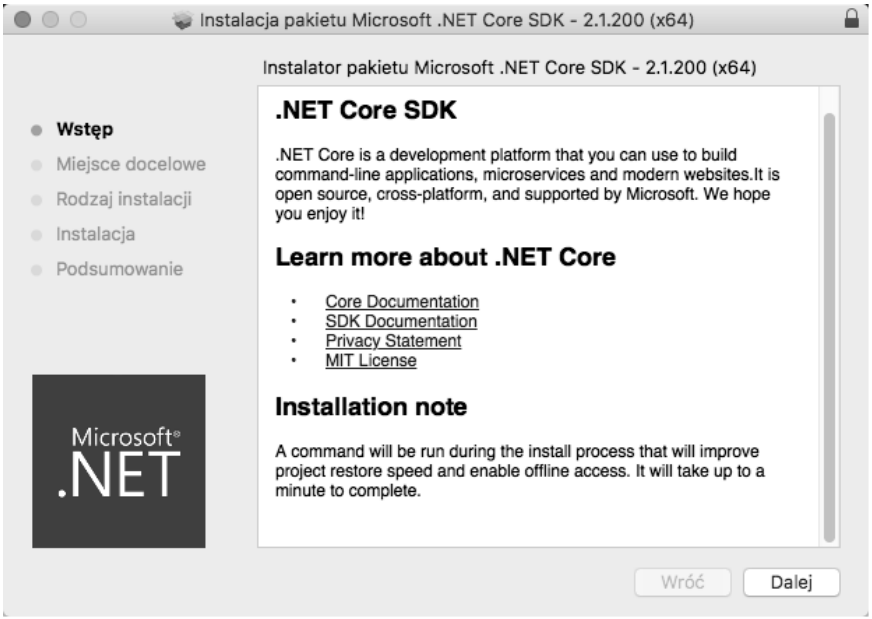
Instalowanie .NET Core SDK for macOS

W kolejnym kroku musisz pobrać instalator pakietu .NET Core SDK w wersji dla macOS (x64). Skorzystaj z poniższego adresu:

https://www.microsoft.com/net/download/core.

Uruchom instalator pakietu *dotnet-sdk-2.0.0-sdk-osx-x64.pkg*, tak jak pokazałem na rysunku na następnej stronie.

Kliknij przycisk *Dalej*, zaakceptuj umowę licencyjną, a potem kliknij przycisk *Install*. Po zakończeniu instalacji pozostaje już tylko kliknąć przycisk *Close*.



Instalowanie menedżera pakietów Node dla macOS

Do prac prowadzonych w rozdziałach 14., 15. i 16. konieczne będzie zainstalowanie pakietów Node.js oraz NPM.

W *Terminalu* wprowadź polecenia instalujące pakiety Node.js i NPM, a następnie skontroluj ich wersje. W czasie gdy pisałem tę książkę, Node.js był dostępny w wersji 8.4, a NPM w wersji 5.3, co potwierdza pierwszy rysunek na następnej stronie.

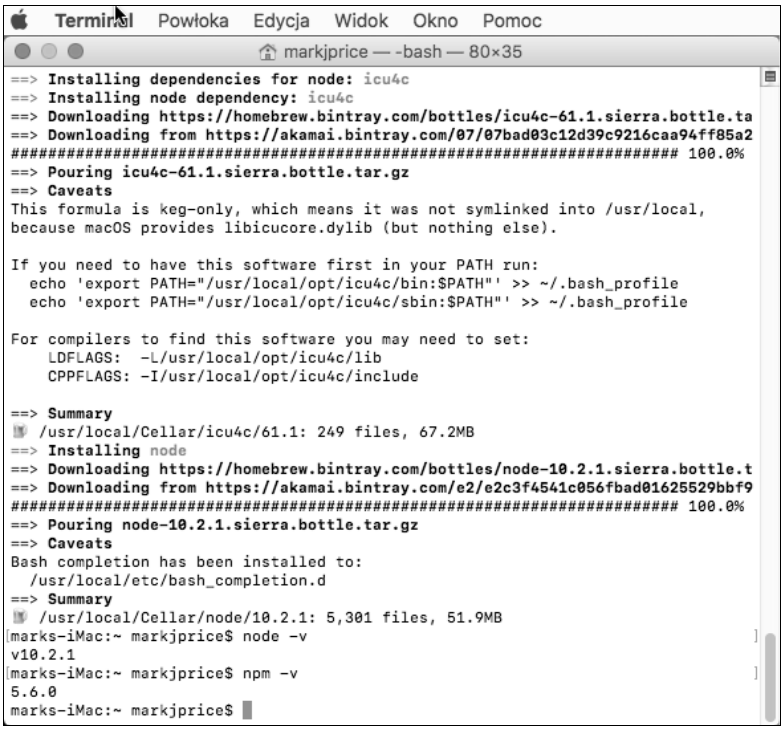
```
brew install node
node -v
npm -v
```

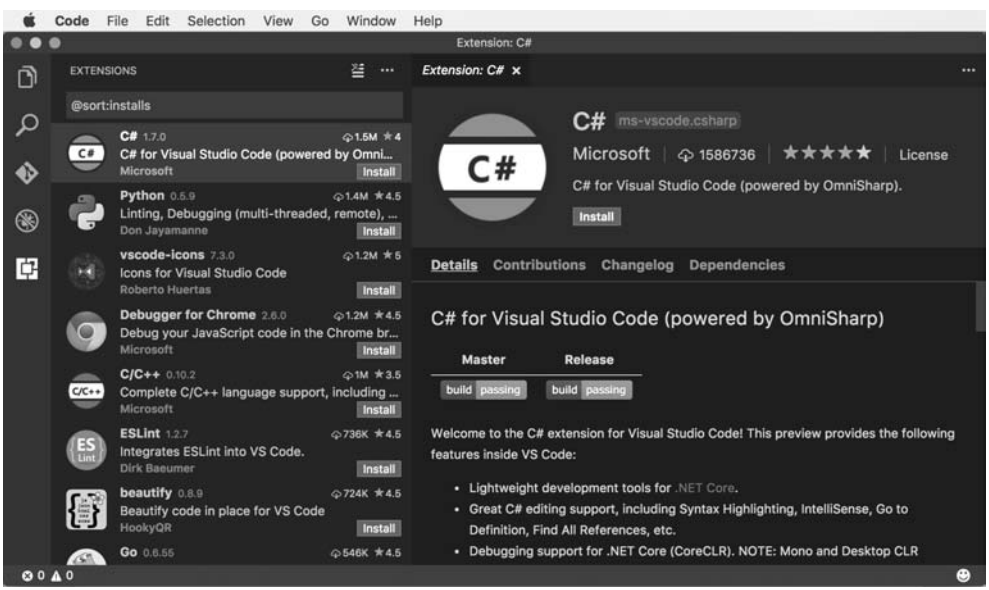
Instalowanie rozszerzenia C# for Visual Studio Code

Rozszerzenie *C# for Visual Studio Code* nie jest elementem wymaganym, ale umożliwia skorzystanie z funkcji IntelliSense w czasie pisania kodu, co jest wyjątkowo użyteczne.

Uruchom Visual Studio Code, a następnie kliknij ikonę *Extensions* albo wybierz z menu pozycję *View/Extensions*, albo naciśnij klawisze *Cmd+Shift+X*.

C# jest bardzo popularnym rozszerzeniem, dlatego powinno znaleźć się na samym szczycie listy, tak jak na drugim rysunku na następnej stronie.





Kliknij tu przycisk *Install*, a następnie przycisk *Reload*, żeby ponownie załadować zawartość okna i aktywować rozszerzenie.

Instalowanie Visual Studio for Mac

W listopadzie 2016 r. Microsoft udostępnił wstępną wersję środowiska Visual Studio for Mac. Początkowo pozwalało ono na tworzenie wyłącznie aplikacji mobilnych Xamarin, ponieważ była to specjalna wersja produktu Xamarin Studio. W ostatecznym wydaniu, które pojawiło się w maju 2017 r., dodana została możliwość tworzenia bibliotek klas .NET Standard 2.0, aplikacji ASP.NET Core oraz serwisów i aplikacji konsolowych. Oznacza to, że można go użyć do wykonania (prawie) wszystkich ćwiczeń z tej książki.

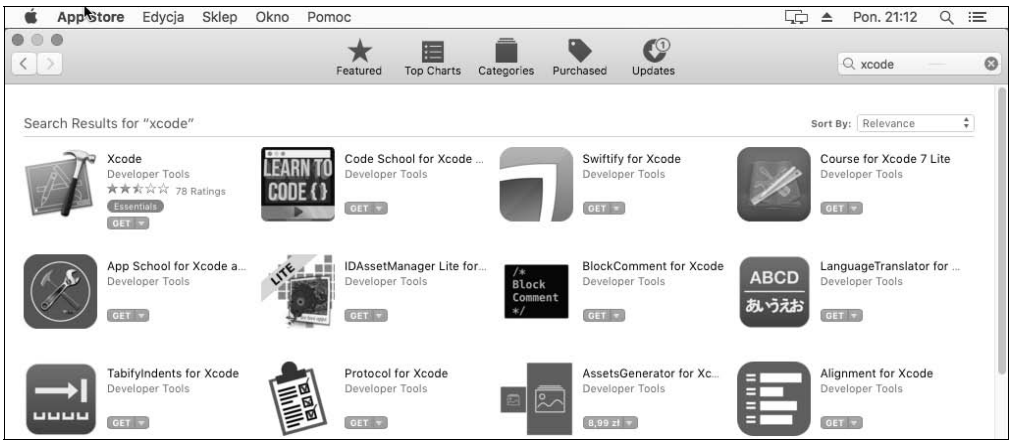
Co prawda Visual Studio 2017 dla Windowsa można wykorzystać do tworzenia aplikacji mobilnych dla systemów iOS i Android, ale do skompilowania wersji dla iOS potrzeba narzędzi Xcode działających w systemie macOS. Moim zdaniem można swobodnie skorzystać ze środowiska Visual Studio for Mac do tworzenia aplikacji mobilnych.

Instalowanie Xcode

Jeżeli jeszcze nie masz zainstalowanego Xcode na swoim komputerze mac, to skorzystaj z App Store, żeby je zainstalować.

Z menu *Apple* wybierz pozycję *App Store…*

W oknie *App Store* wpisz w polu wyszukiwania `xcode`, a jednym z pierwszych wyników będzie aplikacja *Xcode*, tak jak pokazano na poniższym rysunku:



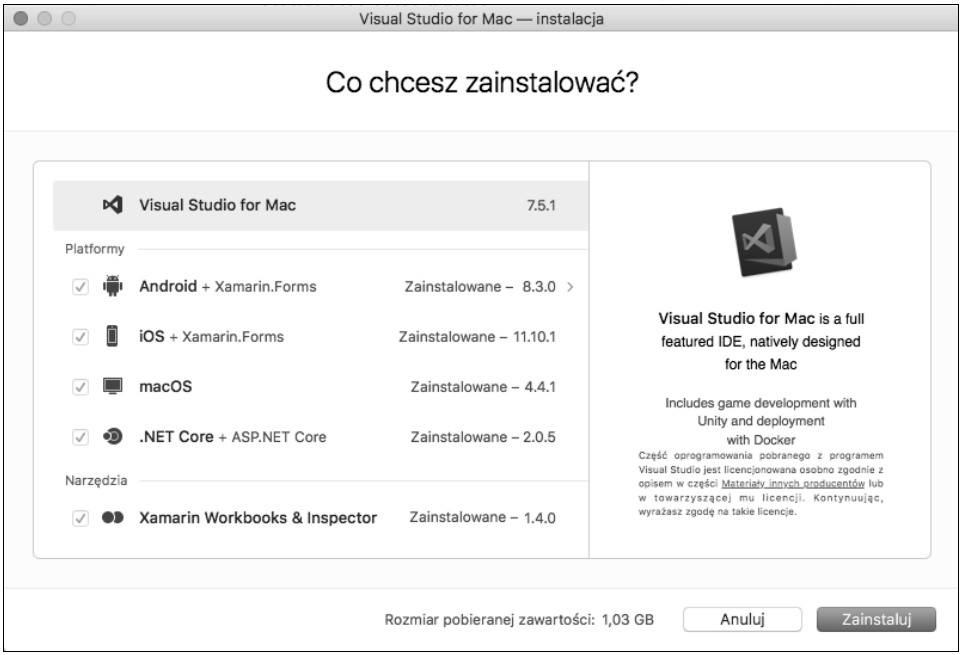
Kliknij przycisk *Get* i poczekaj na zakończenie instalacji.

Pobieranie i instalowanie Visual Studio for Mac

Visual Studio for Mac możesz pobrać z poniższego adresu URL:

https://www.visualstudio.com/vs/visual-studio-mac/.

Po uruchomieniu instalatora *Visual Studio for Mac* trzeba zaakceptować warunki licencji i informacje o zachowaniu prywatności, a następnie wybrać komponenty do zainstalowania, tak jak pokazano na poniższym rysunku. Na zakończenie kliknij jeszcze przycisk *Kontynuuj*.



W kolejnym oknie kliknij przycisk *Kontynuuj*, a w następnym — *Zainstaluj*.

Pozostaje jeszcze zaakceptować warunki licencji poszczególnych instalowanych komponentów, np. SDK systemu Android, kliknąć przycisk *Kontynuuj* i poczekać na zakończenie instalacji Visual Studio for Mac.

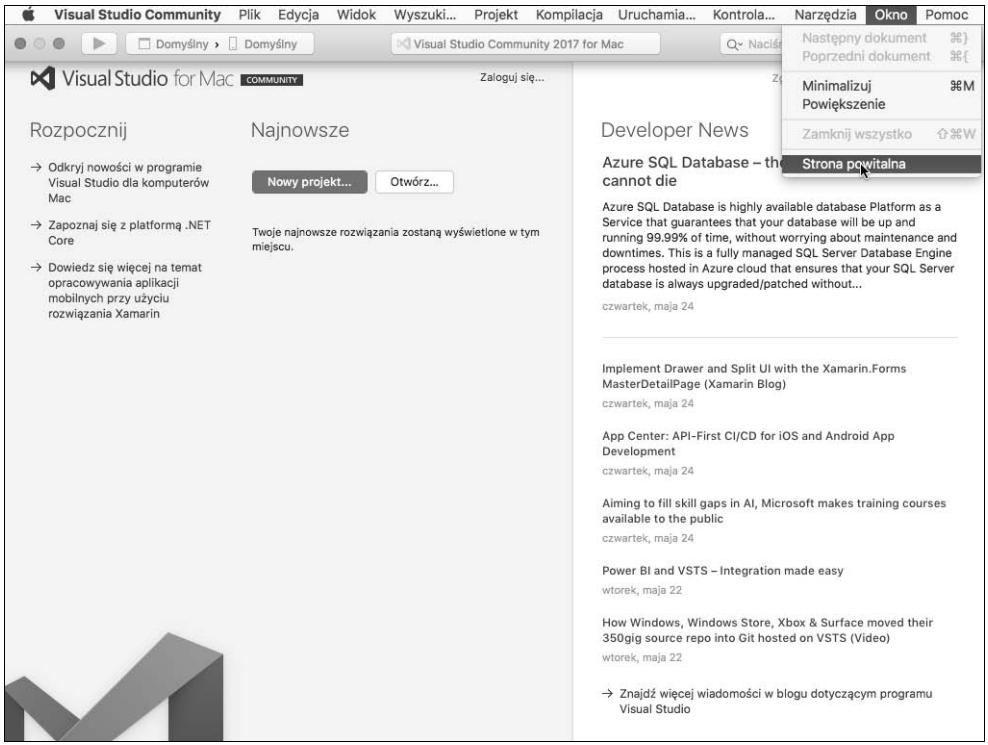
Po uruchomieniu Visual Studio for Mac zobaczysz stronę *Strona powitalna*, taką jak na rysunku na następnej stronie.

Jeżeli pojawi się prośba o aktualizację, zaakceptuj ją.

Skoro masz już zainstalowane i skonfigurowane środowisko programistyczne, możemy przystąpić do poznawania środowiska .NET i rozpocząć pisanie kodu.

Poznawanie .NET

.NET Framework, .NET Core, .NET Standard i .NET Native są powiązanymi ze sobą platformami programistycznymi, których programiści mogą używać do tworzenia aplikacji i usług.



Poznawanie .NET Framework

.NET Framework jest platformą programistyczną, w skład której wchodzi środowisko wykonawcze **CLR** (ang. *Common Language Runtime*) zajmujące się wykonywaniem kodu programów i udostępniające im bogatą bibliotekę klas.

Microsoft zaprojektował .NET Framework w taki sposób, żeby zachować możliwość pracy wieloplatformowej, ale jednocześnie całość swoich prac prowadził tak, żeby ta platforma najlepiej działała w systemie Windows.

To oznacza, że .NET Framework jest platformą wyłącznie dla Windowsa i jako taka jest powoli porzucana.

Poznawanie projektów Mono i Xamarin

Powstała też niezależna implementacja platformy .NET, o której możesz przeczytać pod poniższym adresem:

http://www.mono-project.com/.

Mono jest implementacją wieloplatformową, ale pozostaje daleko w tyle za oficjalną implementacją platformy .NET Framework. Znalazła swoją niszę, stając się bazą dla platformy mobilnej **Xamarin**.

Microsoft wykupił platformę Xamarin w 2016 r. i teraz rozpowszechnia to, co było bardzo drogim pakietem, jako bezpłatny dodatek do Visual Studio 2017. Istniejące wcześniej **Xamarin Studio** Microsoft przemianował na **Visual Studio for Mac** i dodał do niego możliwość tworzenia serwisów ASP.NET Core dla sieci WWW. Pierwotnie platforma Xamarin specjalizowała się w tworzeniu aplikacji dla urządzeń mobilnych oraz budowaniu usług współdziałających z tymi aplikacjami.

Poznawanie .NET Core

Żyjemy w świecie realizującym ideę wieloplatformowości. Nowoczesne metody tworzenia oprogramowania dla urządzeń mobilnych i dla chmury sprawiły, że Windows stracił na znaczeniu jako system operacyjny. W związku z tym Microsoft intensywnie pracuje nad tym, żeby poluzować związki łączące platformę .NET z systemami Windows.

Podczas ponownego projektowania środowiska .NET w sposób rzeczywiście wieloplatformowy Microsoft skorzystał z tej okazji, żeby przebudować platformę, usuwając z niej te ważne elementy, które dzisiaj nie są już uznawane za **podstawowe** (ang. *core*).

Nowy produkt otrzymał nazwę **.NET Core**, a jego podstawami stały się wieloplatformowa implementacja środowiska CLR o nazwie **CoreCLR** oraz odchudzona biblioteka klas, która otrzymała nazwę **CoreFX**.

Scott Hunter — odpowiedzialny za platformę .NET menedżer programu Microsoft Partner Director — powiedział: „Czterdzieści procent naszych klientów .NET Core to programiści wdrażający się dopiero w platformę. Tego właśnie tutaj chcemy. Chcemy pozyskać nowych ludzi".

W poniższej tabeli podaję, kiedy zostały udostępnione poszczególne wersje .NET Core oraz plan Microsoftu opisujący przyszłe wydania platformy:

Wersja	Data wydania
.NET Core RC1	listopad 2015
.NET Core 1.0	czerwiec 2016
.NET Core 1.1	listopad 2016
.NET Core 1.0.4 i .NET Core 1.1.1	marzec 2017
.NET Core 2.0	sierpień 2017
.NET Core for UWP jako część aktualizacji Windows 10 Fall Creators Update	październik 2017
.NET Core 2.1	pierwszy kwartał 2018

Jeżeli musisz pracować z .NET Core 1.0 i .NET Core 1.1, to proponuję przeczytać ogłoszenie związane z .NET Core 1.1, ponieważ informacje zawarte pod poniższym adresem URL są przydatne dla każdego programisty używającego .NET Core:

https://blogs.msdn.microsoft.com/dotnet/2016/11/16/announcing-net-core-1-1/.

.NET Core jest znacznie mniejsza od aktualnej wersji .NET Framework, ponieważ zostało z niej usuniętych wiele różnych elementów.

Na przykład biblioteki **Windows Forms** oraz **Windows Presentation Foundation** (**WPF**) można wykorzystać do tworzenia aplikacji z graficznym interfejsem użytkownika (GUI), ale obie są ściśle związane z systemem Windows i dlatego nie znalazły miejsca w .NET Core. Najnowsza technologia do tworzenia aplikacji dla systemu Windows to **Universal Windows Platform** (**UWP**), zbudowana na specjalnej wersji .NET Core. Więcej informacji na jej temat znajdziesz w rozdziale 17. „Tworzenie aplikacji dla Windowsa przy użyciu XAML i Fluent Design".

Biblioteki **ASP.NET Web Forms** i **Windows Communication Foundation** (**WCF**) to stare technologie tworzenia aplikacji WWW i serwisów, których dzisiaj używa już coraz mniej programistów, dlatego obie zostały usunięte z .NET Core. Programiści znacznie chętniej używają bibliotek ASP.NET MVC i ASP.NET Web API, dlatego obie zostały przebudowane i połączone w jeden produkt działający w ramach .NET Core pod nazwą **ASP.NET Core**. Więcej informacji na temat biblioteki **ASP.NET Core MVC** znajdziesz w rozdziale 15., „Tworzenie aplikacji WWW przy użyciu ASP.NET Core MVC". Rozdział 14., „Tworzenie witryn WWW przy użyciu ASP.NET Core Razor Pages", został poświęcony technologii **ASP.NET Core Razor Pages**. Z kolei technologie **ASP.NET Core Web API** i **Single Page Applications** (**SPAs**) będę opisywać w rozdziale 16., „Tworzenie usług i aplikacji WWW przy użyciu ASP.NET Core".

Entity Framework 6 (**EF**) to technologia mapowania relacji obiektów, współpracująca z relacyjnymi bazami danych, takimi jak Oracle albo Microsoft SQL Server. Przez lata nazbierała wiele obciążeń, dlatego jej nowa, wieloplatformowa wersja została odchudzona i otrzymała nazwę **Entity Framework Core**. Więcej informacji na ten temat znajdziesz w rozdziale 11., „Praca z bazami danych przy użyciu Entity Framework Core".

Po usunięciu sporych elementów .NET Framework w celu utworzenia .NET Core Microsoft podzielił całość na komponenty i stworzył z nich pakiety NuGet. Powstały zatem małe wycinki z funkcjami, które można wykorzystywać niezależnie od siebie.

Ostatecznym celem Microsoftu nie jest zmniejszenie .NET Core względem .NET Framework. Jego celem jest podzielenie .NET Core na komponenty, które pozwalałyby na obsługę nowoczesnych technologii i tworzenie mniejszej liczby zależności. Dzięki temu instalowanie aplikacji wymagać będzie skopiowania tylko tych pakietów, z których faktycznie korzystają programiści.

Poznawanie .NET Standard

Obecnie platforma .NET została podzielona na trzy osobne platformy, a każda z nich jest kontrolowana przez Microsoft:

- .NET Framework,
- .NET Core,
- Xamarin.

Każdy z tych wariantów ma swoje mocne i słabe strony, ponieważ były one projektowane dla różnych zastosowań. W związku z tym każdy programista ma problem, ponieważ musi się uczyć zasad działania trzech platform i poznawać związane z każdą z nich ograniczenia.

W związku z tym Microsoft zdefiniował specyfikację .NET Standard 2.0, która jest zbiorem różnych API, jakie musi implementować każda platforma .NET. Nie da się nigdzie zainstalować.NET Standard 2.0, podobnie jak nie da się zainstalować HTML5. Aby skorzystać z HTML5, musisz zainstalować przeglądarkę, która będzie implementowała specyfikację HTML5. Podobnie aby użyć .NET Standard 2.0, musisz zainstalować platformę .NET, która będzie implementowała specyfikację .NET Standard 2.0.

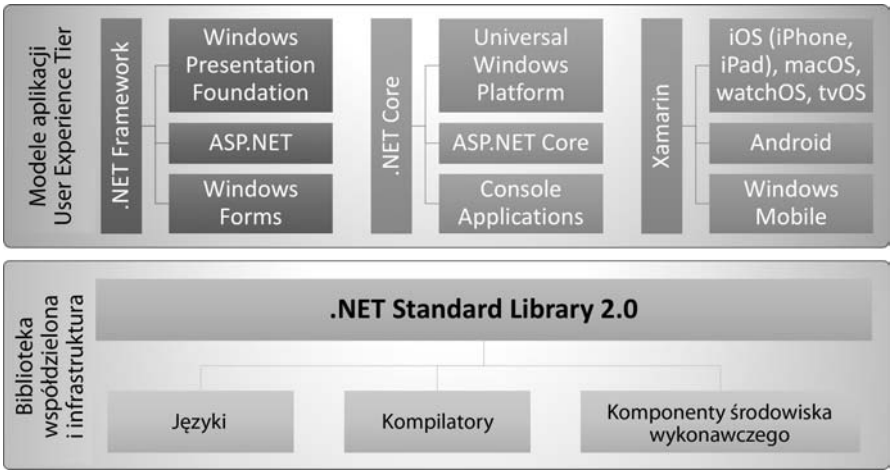
Specyfikacja .NET Standard 2.0 implementowana jest przez najnowsze wersje .NET Framework, .NET Core i Xamarin. Dzięki tej specyfikacji programiści znacznie łatwiej mogą przenosić kod pomiędzy poszczególnymi wariantami platformy .NET.

W przypadku .NET Core 2.0 spowodowało to dodanie wielu brakujących API, które programiści mogą wykorzystać do przenoszenia kodu przygotowanego dla .NET Framework na wieloplatformowy wariant .NET Core. Niestety, część z tych API została co prawda **zaimplementowana**, ale i tak rzuca wyjątki informujące programistę, że nie powinien z nich korzystać! Zazwyczaj wynika to z różnic w systemie operacyjnym, na którym uruchamiana jest .NET Core. W rozdziale 2., „Mówimy w C#”, dowiesz się, jak obsługiwać takie wyjątki.

Na poniższym rysunku widać, w jaki sposób wszystkie trzy warianty platformy .NET (czasami nazywane też modelami aplikacji) współdzielą ze sobą specyfikację .NET Standard 2.0 oraz całą infrastrukturę — zobacz rysunek na następnej stronie.

W pierwszym wydaniu tej książki koncentrowałem się na .NET Core, ale używałem też .NET Framework w przypadku, gdy ważne lub przydatne funkcje nie zostały jeszcze zaimplementowane w .NET Core. W większości przypadków używałem Visual Studio 2015, a Visual Studio Code prezentowałem jedynie pobieżnie.

W drugim wydaniu niemal całkowicie usunąłem wszystkie przykłady kodu dla .NET Framework.



Trzecie wydanie jest dopełnieniem całości. Większość książki została napisana od nowa, dzięki czemu całość kodu jest przystosowana do .NET Core i może być pisana w Visual Studio 2017, Visual Studio for Mac albo Visual Studio Code, i to niezależnie od systemu operacyjnego. Jedynymi wyjątkami są dwa ostatnie rozdziały. W rozdziale 17., „Tworzenie aplikacji dla Windowsa przy użyciu XAML i Fluent Design", skorzystamy z .NET Core for UWP, co wymusi zastosowanie Visual Studio 2017 działającego w systemie Windows 10. Z kolei w rozdziale 18., „Tworzenie aplikacji mobilnych przy użyciu XAML i Xamarin.Forms", zamiast .NET Core użyjemy wariantu Xamarin.

Poznawanie .NET Native

Inną inicjatywą rozwijania platformy .NET jest .NET Native. Pozwala ona na skompilowanie kodu języka C# bezpośrednio do postaci instrukcji procesora, a nie do instrukcji języka pośredniego (ang. *Intermediate Language* — IL), które dopiero w czasie wykonywania przekładane są przez środowisko wykonawcze CLR na kod procesora.

.NET Native poprawia czasy wykonania kodu i zmniejsza zapotrzebowanie na pamięć aplikacji. Pozwala obsłużyć następujące technologie:

- aplikacje UWP dla Windows 10, Windows 10 Mobile, Xbox One, HoloLens oraz urządzeń **internetu rzeczy**, takich jak Raspberry Pi;
- aplikacje WWW po stronie serwera przy użyciu ASP.NET Core;
- aplikacje konsolowe używane w wierszu poleceń.

Porównanie technologii .NET

Poniższa tabela podsumowuje informacje o poszczególnych technologiach .NET:

Technologia	Zbiór funkcji	Kompilowane do	Systemy operacyjne
.NET Framework	nowoczesne i tradycyjne	kod IL	tylko Windows
Xamarin	wyłącznie mobilne	kod IL	iOS, Android, Windows Mobile
.NET Core	wyłącznie nowoczesne	kod IL	Windows, macOS, Linux
.NET Native	wyłącznie nowoczesne	kod procesora	Windows, macOS, Linux

Pisanie i kompilowanie kodu przy użyciu narzędzi wiersza poleceń z .NET Core

Po zainstalowaniu Visual Studio 2017, Visual Studio for Mac albo .NET Core SDK dostępne jest narzędzie wiersza poleceń o nazwie `dotnet` oraz środowisko wykonawcze .NET Core.

Zanim jednak skorzystamy z takich narzędzi jak `dotnet`, musimy napisać choć trochę kodu!

Pisanie kodu za pomocą prostego edytora tekstu

Jeżeli używasz systemu Windows, uruchom Notatnik.

Jeżeli używasz systemu macOS, uruchom TextEdit. W menu wybierz pozycję *TextEdit/Preferencje*, usuń zaznaczenie opcji *Cudzysłowy inteligentne*, a następnie zamknij okno dialogowe. W menu wybierz pozycję *Format/Zamień na tekst zwykły*.

Oczywiście zawsze możesz skorzystać ze swojego ulubionego edytora tekstu.

Wpisz poniższy kod:

```
class MyApp { static void Main() {
  System.Console.WriteLine("Witaj, C#!"); } }
```

> Język C# rozróżnia wielkości liter, a to oznacza, że wielkie i małe litery musisz wpisać dokładnie tak, jak podałem w poprzednim przykładowym kodzie. Z drugiej strony, języka C# nie interesują znaki białe, a to oznacza, że nie ma znaczenia, czy użyjesz spacji, tabulacji czy znaków końca wiersza, żeby wprowadzić do kodu strukturę.

Całość kodu możesz wpisać do jednego wiersza albo rozdzielić go na kilka wierszy, każdemu z nich nadając odpowiednie wcięcie. Na przykład poniższy kod tak samo będzie można skompilować i da te same wyniki co poprzedni:

```
class
      MojaAplikacja       {
```

```
 static                    void
Main          (){System.        Console.
    WriteLine(        "Witaj, C#!");         }           }
```

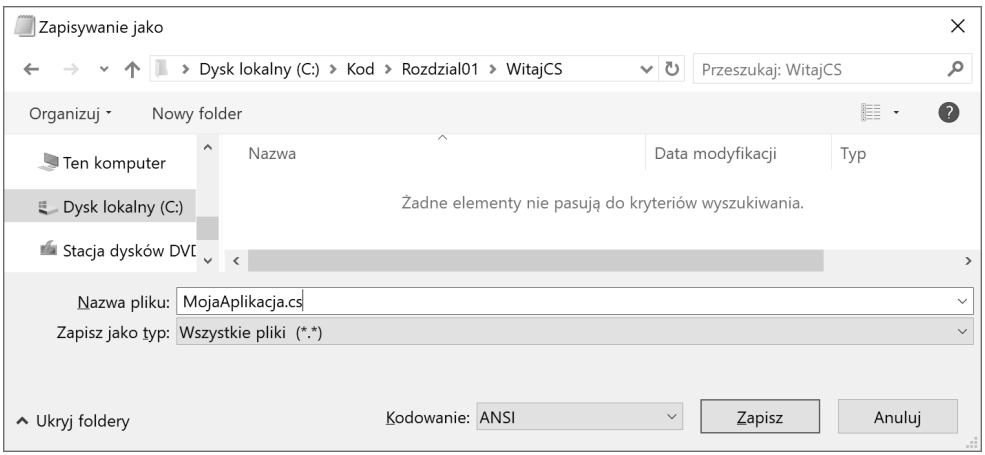
Oczywiście najlepiej jest tak pisać kod, żeby każdy inny programista (również Ty) mógł nawet po miesiącach lub latach bez problemu go odczytać!

Notatnik Windows

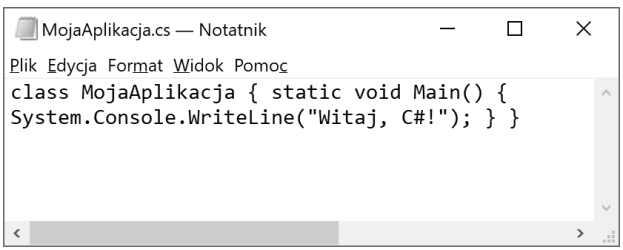
W Notatniku wybierz z menu pozycję *Plik/Zapisz jako…*

W oknie dialogowym *Zapisz jako* przejdź na dysk *C:* (albo dowolny inny, na którym chcesz zapisywać swoje projekty), kliknij przycisk *Nowy folder* i nadaj utworzonemu folderowi nazwę *Kod*. Otwórz utworzony właśnie folder i ponownie kliknij przycisk *Nowy folder*, a utworzonemu folderowi nadaj nazwę *Rozdzial01*. Otwórz ten folder, jeszcze raz kliknij przycisk *Nowy folder* i nazwij utworzony folder *WitajCS*. Na zakończenie otwórz folder *WitajCS*.

Z listy rozwijanej *Zapisz jako typ* wybierz opcję *Wszystkie pliki*, aby uniknąć dodania rozszerzenia *.txt* do tworzonego pliku. Wprowadź nazwę pliku *MojaAplikacja.cs*, tak jak pokazałem na poniższym rysunku:



Kod zapisany w Notatniku powinien wyglądać tak jak na poniższym rysunku:



Edytor TextEdit w systemie macOS

W edytorze TextEdit wybierz z menu *Plik/Zachowaj* albo naciśnij klawisze *Cmd+S*.

W oknie dialogowym *Zachowaj* przejdź do folderu użytkownika (mój nazywa się *markprice*) albo dowolnego innego, w którym chcesz zapisywać swoje projekty. Kliknij przycisk *Nowy katalog* i nadaj utworzonemu folderowi nazwę *Kod*. Otwórz utworzony właśnie folder i ponownie kliknij przycisk *Nowy katalog*, a utworzonemu folderowi nadaj nazwę *Rozdzial01*. Otwórz ten folder, jeszcze raz kliknij przycisk *Nowy katalog* i nazwij utworzony folder *WitajCS*. Na zakończenie otwórz folder *WitajCS*.

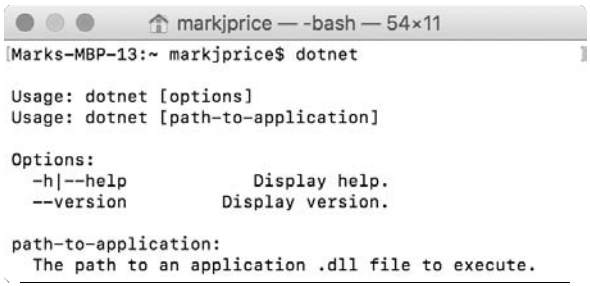
W polu *Kodowanie tekstu zwykłego* wybierz z listy rozwijanej pozycję *Unicode (UTF-8)* i usuń zaznaczenie z opcji *W przypadku braku rozszerzenia, użyj ".txt"*, aby uniknąć dodania rozszerzenia *.txt* do tworzonego pliku. Wpisz nazwę pliku *MyApp.cs* i kliknij przycisk *Zachowaj*.

Tworzenie i kompilowanie aplikacji za pomocą narzędzi wiersza poleceń .NET Core

Jeżeli używasz systemu Windows, uruchom *Wiersz poleceń*.

Jeżeli używasz systemu macOS, uruchom *Terminal*.

W otwartym oknie wpisz polecenie `dotnet` i przyjrzyj się tekstom wypisanym na ekranie, pokazanym na poniższym rysunku:



Tekst wyświetlany przez polecenie `dotnet` będzie taki sam w systemach Windows, macOS i Linux.

Tworzenie aplikacji konsolowej z wiersza poleceń

W wierszu poleceń wpisz poniższe polecenia, aby wykonać następujące operacje:

- Przejść do folderu z projektem.
- Utworzyć w tym folderze nową aplikację konsolową.

- Wypisać pliki, jakie utworzyło polecenie `dotnet`.

Jeżeli używasz systemu Windows, to w *Wierszu poleceń* wpisz następujące polecenia:

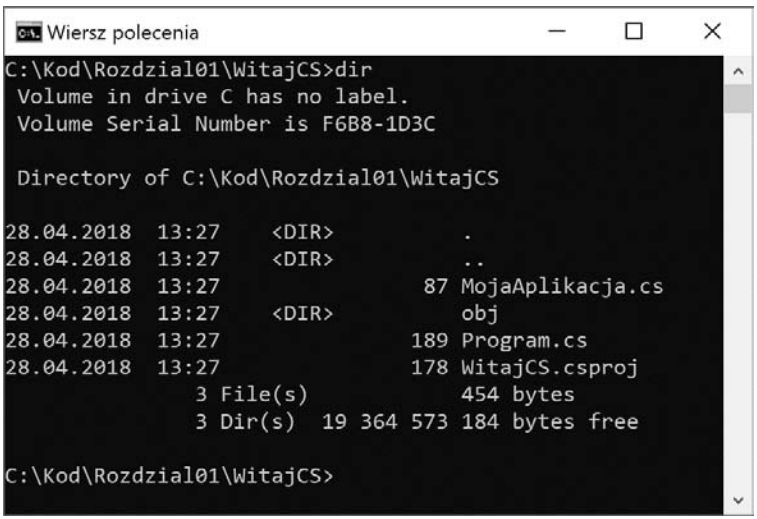
```
cd C:\Kod\Rozdzial01\WitajCS
dotnet new console
dir
```

Jeżeli używasz systemu macOS, to w *Terminalu* wprowadź następujące polecenia:

```
cd Kod/Rozdzial01/WitajCS
dotnet new console
ls
```

Polecenie `dotnet` powinno utworzyć dwa nowe pliki, takie jak widoczne na rysunku ze zrzutem ekranu z systemu Windows:

- *Program.cs* — kod źródłowy prostej aplikacji konsolowej.
- *WitajCS.csproj* — plik projektu zawierający listę zależności różnych ustawień samego projektu.



W tym przykładzie musimy usunąć plik o nazwie *Program.cs*, ponieważ już wcześniej utworzyliśmy własny plik o nazwie *MojaAplikacja.cs*.

Jeżeli używasz systemu Windows, to w *Wierszu poleceń* wpisz następujące polecenie:

```
del Program.cs
```

Jeżeli używasz systemu macOS, to w *Terminalu* wpisz następujące polecenie:

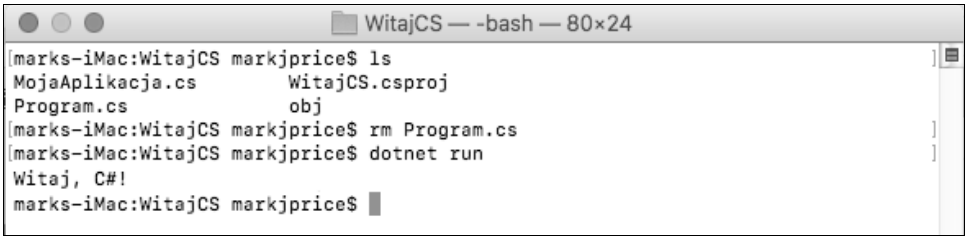
```
rm Program.cs
```

We wszystkich kolejnych przykładach zamiast tworzyć własne pliki z kodem, będziemy korzystać z pliku *Program.cs* tworzonego przez polecenie `dotnet`.

Odtwarzanie pakietów, kompilowanie kodu i uruchamianie aplikacji

W wierszu poleceń wpisz polecenie `dotnet run`.

Po kilku sekundach wszystkie pakiety wymagane przez nasz kod zostaną pobrane, kod źródłowy zostanie skompilowany, a sama aplikacja zostanie uruchomiona. Wszystko to można zobaczyć na poniższym rysunku:



Nasz kod źródłowy, czyli plik *MojaAplikacja.cs*, został skompilowany do pliku o nazwie *WitajCS.dll*, który umieszczany jest w podkatalogu *bin/Debug/netcoreapp2.0*. (Jeżeli masz ochotę, możesz przejrzeć zawartość struktury katalogów. Ja tutaj poczekam, aż skończysz).

Na razie taki plik może zostać uruchomiony wyłącznie przez polecenie `dotnet run`. W rozdziale 7., „Poznawanie typów .NET Standard", dowiesz się więcej na temat przygotowywania skompilowanego kodu do wykonania w dowolnym systemie operacyjnym współpracującym z .NET Core.

Naprawianie błędów kompilacji

Jeżeli kompilator wyświetli jakiekolwiek błędy, to dokładnie przeczytaj ich treść, a następnie popraw je w edytorze tekstu. Zapisz wprowadzone zmiany i spróbuj ponownie.

W wierszu poleceń możesz skorzystać ze strzałek w górę i w dół, żeby przywoływać poprzednio wprowadzone polecenia.

Często występującymi błędami jest wpisanie litery o niewłaściwej wielkości, pominięcie średnika na końcu instrukcji albo niedopasowanie do siebie nawiasów klamrowych. Na przykład jeżeli zdarzy Ci się wpisać małą literę `m` w nazwie metody `Main()`, to zobaczysz poniższy komunikat o błędzie:

```
error CS5001: Program nie zawiera statycznej metody "Main" odpowiedniej jako punkt wejścia
```

Poznawanie języka pośredniego

Kompilator języka C# (nazywany **Roslyn**) używany przez polecenie `dotnet` zajmuje się przekształcaniem kodu źródłowego C# na kod języka pośredniego (IL), który jest zapisywany w pliku o rozszerzeniu *.dll* lub *.exe*.

Polecenia języka IL przypominają instrukcje języka asemblera, ale są wykonywane przez maszynę wirtualną .NET Core, nazywaną **CoreCLR**.

W czasie pracy CoreCLR ładuje kod IL z pliku wykonywalnego, a następnie kompiluje go do postaci instrukcji procesora, na którym działa. Dopiero te instrukcje przekazywane są procesorowi do wykonania.

Zaletą takiej dwuetapowej kompilacji jest to, że Microsoft może przygotować maszyny wirtualne CLR nie tylko dla systemów Windows, ale i dla Linuksa oraz systemu macOS. Ten sam kod IL może działać wszędzie, ponieważ dopiero drugi etap kompilacji generuje kod właściwy dla danego systemu operacyjnego i procesora.

Niezależnie od tego, w jakim języku napisany jest kod źródłowy (czy będzie to C#, czy też F#), wszystkie aplikacje działające w środowisku .NET mają zapisane w plikach wykonywalnych instrukcje języka IL. Microsoft i inni producenci udostępniają deasemblery, które odczytują pliki wykonywalne i podają zawarty w nich kod IL.

> Tak naprawdę nie wszystkie aplikacje .NET używają kodu IL! Niektóre z nich używają kompilatora .NET Native, który generuje bezpośrednio kod procesora, a nie kod IL. Poprawia to wydajność aplikacji i zmniejsza zapotrzebowanie na pamięć, ale za cenę przenośności kodu.

Pisanie i kompilowanie kodu za pomocą Visual Studio 2017

Przygotujemy teraz podobną aplikację, wykorzystując Visual Studio 2017. Nawet jeżeli wolisz używać Visual Studio for Mac albo Visual Studio Code, to i tak zachęcam do przejrzenia tego podrozdziału i zapoznania się z rysunkami, ponieważ oba warianty środowiska programistycznego mają bardzo podobne, choć nie aż tak rozbudowane funkcje.

Od ponad dekady nauczam ludzi stosowania Visual Studio przy pracy i nadal zaskakuje mnie, że programiści nie umieją wykorzystać potencjału tego narzędzia.

Na kolejnych stronach nauczę Cię wprowadzania kodu wiersz po wierszu. Może się to wydawać niepotrzebne, ale z pewnością warto się dowiedzieć, jaką pomoc i jakie informacje udostępnia

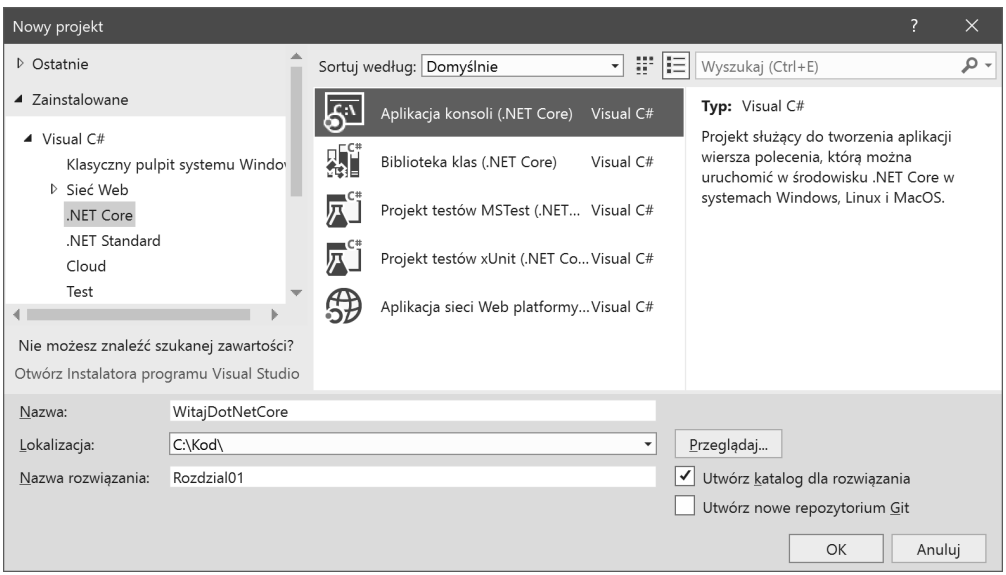
nam Visual Studio w czasie, gdy wprowadzamy kod. Jeżeli chcesz się nauczyć szybko i bezbłędnie wprowadzać swój kod, to bardzo wiele zyskasz, pozwalając Visual Studio na samodzielne napisanie większości kodu.

Pisanie kodu w Visual Studio 2017

Uruchom Visual Studio 2017.

Wybierz z menu pozycję *Plik/Nowy/Projekt…* albo naciśnij klawisze *Ctrl+Shift+N*.

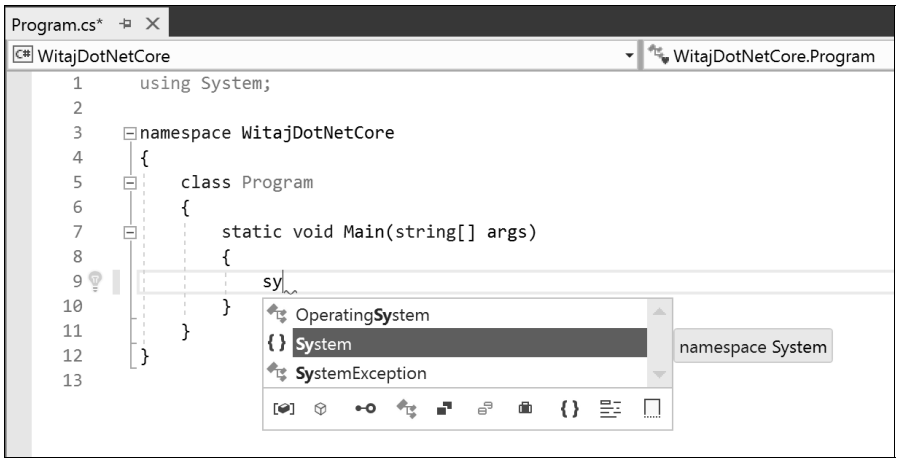
Ze znajdującej się po lewej stronie listy *Zainstalowane* rozwiń pozycję *Visual C#* i wybierz opcję *.NET Core*. Z listy na środku wybierz opcję *Aplikacja konsoli (.NET Core)*. Wprowadź nazwę projektu — `WitajDotNetCore`, jako lokalizację wybierz folder *C:\Kod*, a jako nazwę rozwiązania wpisz `Rozdzial01`. Na zakończenie kliknij przycisk *OK* albo naciśnij klawisz *Enter*, tak jak na poniższym rysunku:



W edytorze kodu usuń znajdującą się w wierszu 9. instrukcję o treści: `Console.WriteLine("Witaj, świecie!");`.

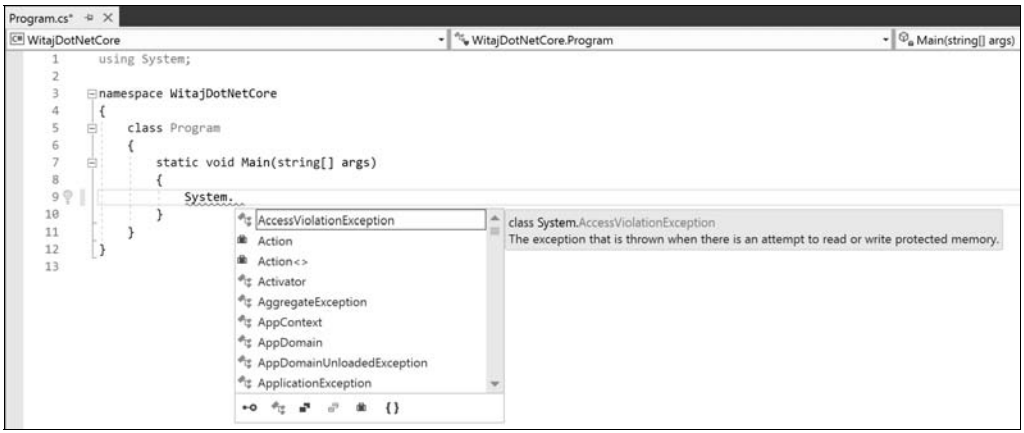
W metodzie `Main` wpisz litery `sy`, tak jak na rysunku na następnej stronie i zwróć uwagę na to, że pojawi się okienko usługi IntelliSense:

Usługa IntelliSense wyświetla filtrowaną listę **słów kluczowych**, **przestrzeni nazw** oraz **typów**, które zawierają litery `sy`, i wyróżnia na niej pozycję zaczynającą się od tych liter. W tym przypadku jest to przestrzeń nazw, której poszukujemy — `System`.



Wpisz kropkę.

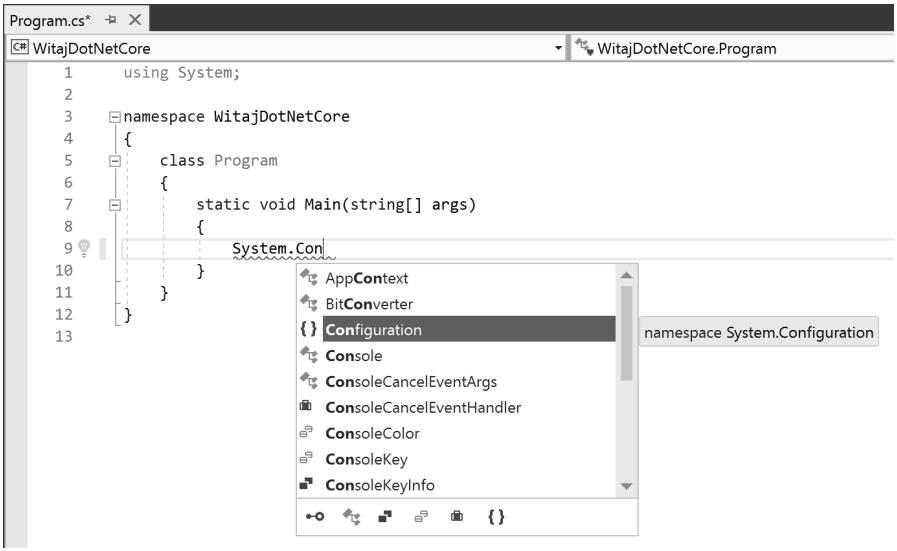
IntelliSense automatycznie uzupełni w kodzie słowo `System`, wprowadzi kropkę i wyświetli listę typów, takich jak `AggregateException` lub `Action`, które są dostępne w przestrzeni nazw `System`, tak jak na poniższym rysunku:

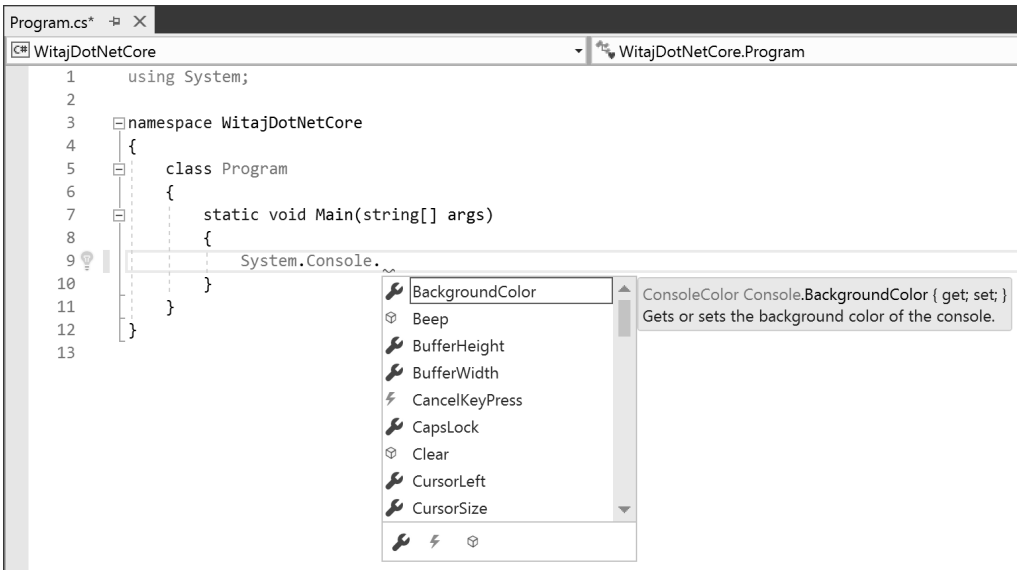


Wpisz litery `con`, a IntelliSense wyświetli listę pasujących typów i przestrzeni nazw, tak jak na pierwszym rysunku na następnej stronie.

Poszukujemy tu słowa `Console`. Wciśnij na klawiaturze strzałkę w dół, żeby je wybrać. Po zaznaczeniu na liście słowa `Console` wpisz kropkę.

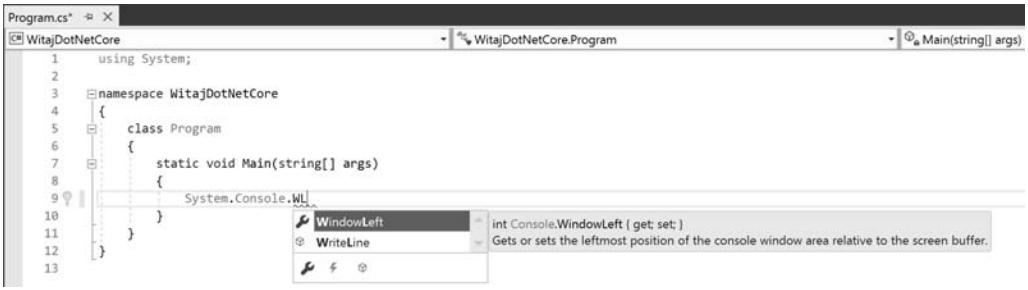
IntelliSense wyświetli teraz listę elementów klasy `Console`, taką jak na drugim rysunku na następnej stronie.





Elementami klasy są **właściwości** (atrybuty obiektu, takie jak `BackgroundColor`), **metody** (akcje, jakie może wykonać obiekt, takie jak `Beep`), **zdarzenia** i inne.

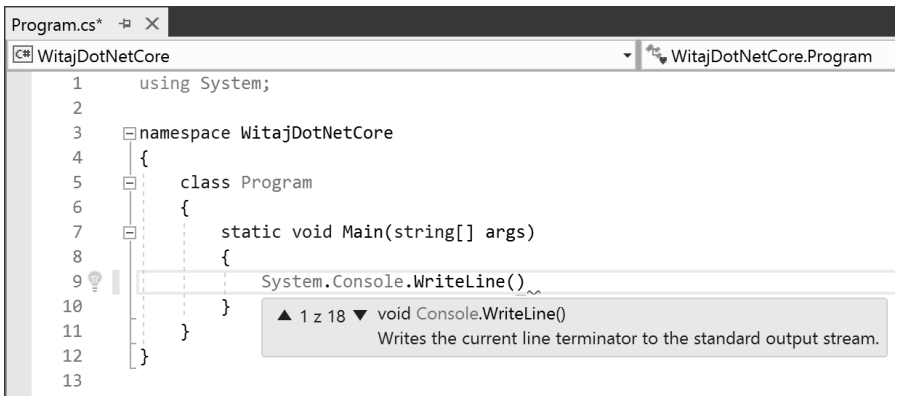
Wpisz litery `wl`. IntelliSense wyświetli teraz pasujące elementy klasy, czyli takie, w których nazwie słowa zaczynają się od podanych liter. W tym przypadku będą to `WindowsLeft` i `WriteLine`, co można zobaczyć na rysunku na następnej stronie.



Użyj strzałki w dół, żeby zaznaczyć na liście pozycję `WriteLine`, i wprowadź nawias otwierający (().

IntelliSense samodzielnie wpisze nazwę metody `WriteLine` i doda za nią nawiasy okrągłe.

Pojawi się też pole wskazówki informujące, że metoda `WriteLine` ma 18 różnych wariantów, takie jak na poniższym rysunku:



Wprowadź znak cudzysłowu ("). IntelliSense wpisze do programu dwa takie znaki i umieści kursor pomiędzy nimi.

Wpisz tekst `Witaj, .NET Core!`, tak jak na rysunku na następnej stronie.

Czerwone podkreślenie na końcu wiersza oznacza błąd, ponieważ w języku C# każda instrukcja musi zostać zakończona znakiem średnika. Przenieś kursor na koniec wiersza i wpisz znak średnika, żeby naprawić ten błąd.

Kompilowanie kodu za pomocą Visual Studio 2017

Wybierz z menu pozycję *Debugowanie/Uruchom bez debugowania* albo naciśnij klawisze *Ctrl+F5*.

```
Program.cs*
WitajDotNetCore                                  WitajDotNetCore.Program
    1   using System;
    2
    3   namespace WitajDotNetCore
    4   {
    5       class Program
    6       {
    7           static void Main(string[] args)
    8           {
    9               System.Console.WriteLine("Witaj, .NET Core!")
   10          }
   11      }
   12  }
   13
```

Na pasku statusu Visual Studio pojawi się komunikat `Kompilacja rozpoczęta...`, a chwilę później — `Kompilacja zakończona sukcesem`. Zaraz potem uruchomiona zostanie aplikacja i otworzone okno konsoli, takie jak na poniższym rysunku:

Aby oszczędzać miejsce w tej książce i poprawić czytelność, nie będę już zamieszczał zrzutów ekranu aplikacji konsolowych, tak jak zrobiłem to powyżej. Zamiast tego będę prezentował wyniki ich pracy w ten sposób:

```
Witaj, .NET Core!
```

Poprawianie pomyłek z listy błędów

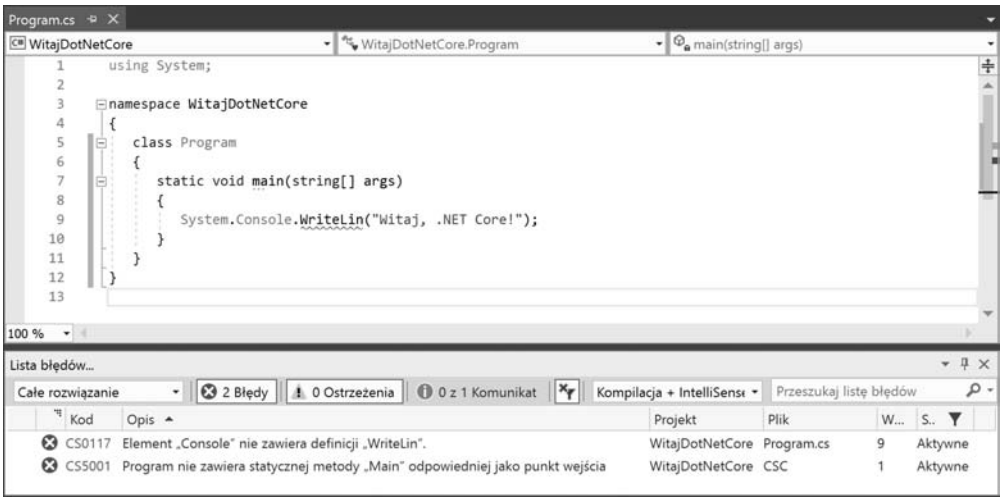
Wprowadźmy teraz do kodu dwa błędy:

- Zmień wielką literę `M` w nazwie funkcji `Main` na małą literę `m`.
- Usuń literę `e` z końca nazwy funkcji `WriteLine`.

Wybierz z menu pozycję *Debugowanie/Uruchom bez debugowania* albo naciśnij klawisze *Ctrl+F5*.

Po kilku sekundach na pasku stanu zostanie wyświetlony komunikat `Kompilacja nie powiodła się...` i pojawi się okienko z informacją o błędzie, w którym należy kliknąć przycisk *Nie*.

Pojawi się wtedy panel *Lista błędów*, taki jak na poniższym rysunku:



Listę można filtrować za pomocą przycisków w górnej części panelu *Lista błędów*, tak żeby pojawiały się na niej lub znikały z niej *Błędy*, *Ostrzeżenia* oraz *Komunikaty*.

Jeżeli przy komunikacie o błędzie pojawia się nazwa pliku i numer wiersza, np. *Plik: Program.cs* i *Wiersz: 9*, to możesz kliknąć dwukrotnie ten błąd, aby od razu przejść do odpowiedniego miejsca w programie.

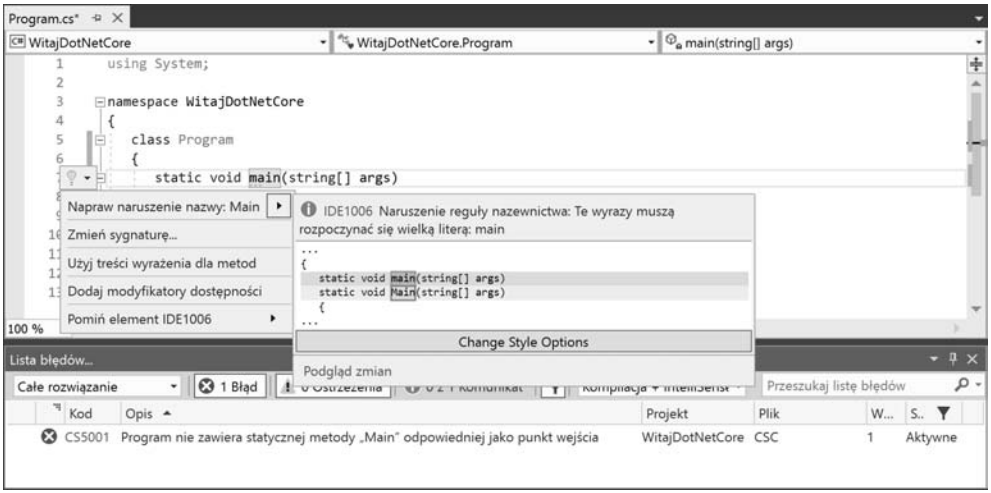
Jeżeli jest to bardziej ogólny błąd, taki jak brakująca metoda `Main`, to kompilator nie jest w stanie podać konkretnego numeru wiersza. W swoim programie możesz przecież chcieć mieć metodę `main`, która będzie całkowicie niezależna od metody `Main`. Pamiętaj, że język C# rozróżnia wielkości liter, więc takie nazwy metod są dozwolone.

Visual Studio może jednak analizować tworzony kod i podawać różne propozycje, prezentowane jako trzy szare kropki w miejscu wykrycia potencjalnych problemów. Po kliknięciu tak wyróżnionej instrukcji pojawi się ikona z żarówką, a po kliknięciu tej ikony wyświetlona zostanie lista propozycji. Mogą to być wskazówki mówiące, że nazwy metod powinny się zaczynać od wielkiej litery, takie jak na rysunku na następnej stronie.

Popraw oba błędy tak jak na powyższym rysunku i ponownie uruchom aplikację, upewniając się, że działa ona prawidłowo. Zauważ, że *Lista błędów* jest teraz pusta.

Dodawanie istniejących projektów do Visual Studio 2017

Wcześniej w tym rozdziale utworzyliśmy już jeden projekt, używając do tego narzędzia `dotnet`. Teraz mamy otwarte rozwiązanie w Visual Studio 2017, do którego możemy dodać utworzony wcześniej projekt.



Wybierz z menu pozycję *Plik/Dodaj/Istniejący projekt…*, w otwartym oknie dialogowym przejdź do folderu *C:\Kod\Rozdzial01\WitajCS*, a w nim wybierz plik *WitajCS.csproj*.

Aby uruchomić ten projekt, musisz w *Eksploratorze rozwiązań* kliknąć prawym przyciskiem myszy pozycję *Rozwiązanie „Rozdzial01" (2 projekty)* i z menu kontekstowego wybrać pozycję *Właściwości* albo nacisnąć klawisze *Alt+Enter*.

Dla opcji *Projekt startowy* wybierz wartość *Bieżące zaznaczenie* i kliknij przycisk *OK*.

W *Eksploratorze rozwiązań* kliknij dowolny plik poza projektem *WitajCS*, a następnie naciśnij klawisze *Ctrl+F5* albo wybierz z menu pozycję *Debugowanie/Uruchom bez debugowania*.

Automatyczne formatowanie kodu

Kod daje się znacznie lepiej czytać i jest bardziej zrozumiały, jeżeli zastosujesz w nim odpowiednie wcięcia i odstępy.

Jeżeli kod daje się skompilować, to Visual Studio 2017 zacznie go automatycznie formatować, wprowadzając jednolite wcięcia i odstępy.

W *Eksploratorze rozwiązań* kliknij dwukrotnie plik *MojaAplikacja.cs*, tak jak na pierwszym rysunku na następnej stronie.

Z menu wybierz pozycję *Kompilowanie/Kompiluj WitajCS* albo naciśnij klawisze *Shift+F6* i poczekaj na zakończenie kompilacji. Następnie wybierz z menu pozycję *Edycja/Zaawansowane/Formatuj dokument* albo naciśnij klawisze *Ctrl+E, D*. Twój kod zostanie automatycznie sformatowany, tak jak na drugim rysunku na następnej stronie.

Eksperymentowanie z interaktywnym C#

Co prawda Visual Studio od zawsze dysponowało oknem *Bezpośrednie* z obsługą funkcji **REPL** (ang. *Read-Eval-Print-Loop* — pętla czytaj-wykonaj-wypisz), ale w Visual Studio 2017 dodano bardziej rozbudowane okno *C# Interactive*, w którym działa IntelliSense oraz kolorowanie składni.

W Visual Studio 2017 wybierz z menu pozycję *Widok/Inne okna/C# Interactive*.

Napiszemy teraz krótki interaktywny kod pobierający stronę *About* z witryny WWW Microsoftu.

> To jest tylko przykład. Nie musisz jeszcze w szczegółach rozumieć tego kodu.

W oknie *C# Interactive* będziemy wprowadzać instrukcje wykonujące następujące operacje:

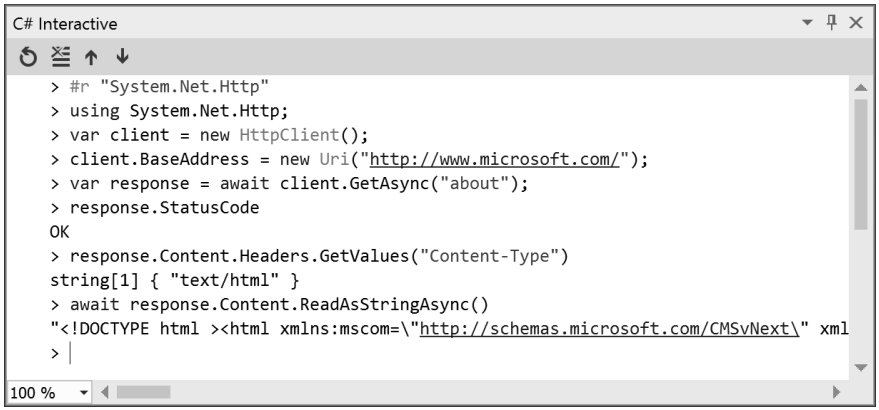
- Odwołanie do zestawu `System.Net.Http`.
- Zaimportowanie przestrzeni nazw `System.Net.Http`.
- Zadeklarowanie i utworzenie zmiennej klienta HTTP.
- Ustalenie podstawowego adresu strony WWW Microsoftu.
- Asynchroniczne oczekiwanie na odpowiedź na żądanie typu GET zawierające kod strony *About*.
- Odczytanie kodu stanu przesłanego przez serwer WWW.

- Odczytanie zawartości nagłówka `content-type`.
- Odczytanie zawartości strony HTML jako ciągu znaków.

Wpisz poniższe polecenia w wierszu ze znakiem > i naciśnij klawisz *Enter*:

```
> #r "System.Net.Http"
> using System.Net.Http;
> var client = new HttpClient();
> client.BaseAddress = new Uri("http://www.microsoft.com/");
> var response = await client.GetAsync("about");
> response.StatusCode
OK
> response.Content.Headers.GetValues("Content-Type")
string[1] { "text/html" }
> await response.Content.ReadAsStringAsync()
"<!DOCTYPE html ><html
xmlns:mscom="http://schemas.microsoft.com/CMSvNext"
xmlns:md="http://schemas.microsoft.com/mscom-data" lang="en"
xmlns="http://www.w3.org/1999/xhtml"><head><meta http-equiv="X-UA-Compatible"
content="IE=edge" /><meta charset="utf-8" /><meta name="viewport"
↪content="width=device-width, initial-scale=1.0"/><link rel="shortcut icon"
↪href="//www.microsoft.com/favicon.ico?v2" /><script type="text/javascript"
↪src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.2.min.js">rn // Third party
↪scripts and code linked to or referenced from this website are licensed to you by
↪the parties that own such code, not by Microsoft. See ASP.NET Ajax CDN Terms of
↪Use - http://www.asp.net/ajaxlibrary/CDN.ashx.rn </script><script type="
↪text/javascript" language="javascript">/*<![CDATA[*/if($(document).bind
↪("mobileinit",function(){$.mobile.autoInitializePage=!1}),navigator.userAgent.
↪match(/IEMobile\/10\.0/)){varmsViewportStyle=document.createElement("style ...
```

Na poniższym rysunku widać, jak powinno wyglądać Visual Studio 2017 po wprowadzeniu powyższych poleceń w oknie *C# Interactive*:



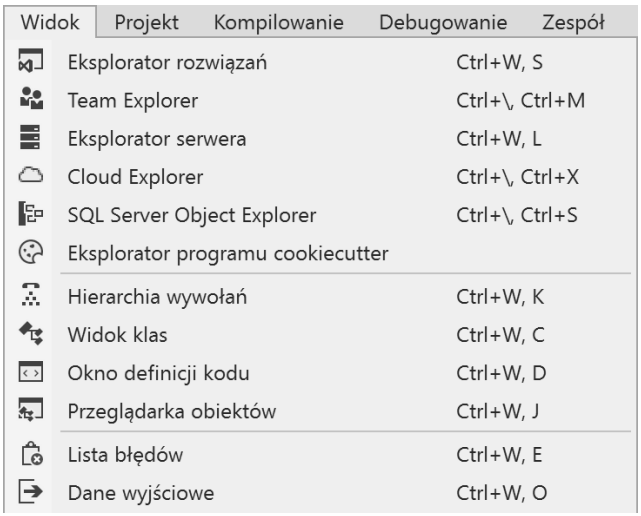
Roslyn to nazwa kompilatora języka C#. Roslyn w wersji 1.0 obsługiwał język C# 6, w wersji 2.0 związany był z językiem C# 7, natomiast wersja 2.3 i wyższe to język C# 7.1.

Inne przydatne okna

W Visual Studio 2017 znajdziesz też wiele innych przydatnych okien, np. takie:

- *Eksplorator rozwiązań* pozwala na zarządzanie projektami i plikami.
- *Team Explorer* umożliwia pracę z narzędziami do zarządzania kodem źródłowym.
- *Eksplorator serwera* daje możliwość obsługiwania połączeń z bazami danych i zarządzania zasobami w usłudze Microsoft Azure.

Jeżeli nie widzisz potrzebnego Ci okna, to otwórz menu *Widok* i wybierz je z listy menu. Spróbuj zapamiętać przypisany mu skrót klawiszowy. Przykłady takich skrótów zobaczysz na poniższym rysunku:



Jeżeli skróty klawiszowe w Twoim systemie są inne niż przedstawione na powyższym rysunku, to znaczy, że w czasie instalowania Visual Studio został wybrany inny zestaw ustawień. Wszystkie skróty klawiszowe możesz zmienić tak, żeby były zgodne z prezentowanymi w tej książce. Wystarczy wybrać z menu pozycję *Narzędzia/Import i eksport ustawień…*, w wyświetlonym oknie dialogowym zaznaczyć opcję *Resetuj wszystkie ustawienia*, a w następnym kroku wybrać zbiór ustawień *Visual C#*.

Pisanie i kompilowanie kodu w Visual Studio Code

Instrukcje i zrzuty ekranów z tego podrozdziału pochodzą z systemu macOS, ale te same operacje można wykonywać w Visual Studio Code działającym w systemach Windows i Linux. Główne różnice będą polegały na wykonywaniu działań w wierszu poleceń systemu operacyjnego, np. usuwaniu plików. W takich przypadkach zarówno polecenia, jak i ścieżki z całą pewnością będą się różniły. Z kolei polecenie `dotnet` działa tak samo w wierszu poleceń wszystkich systemów operacyjnych.

Pisanie kodu w Visual Studio Code

Uruchom Visual Studio Code.

W menu wybierz pozycję *File/Open…* albo naciśnij klawisze *Cmd+O*.

W oknie dialogowym otwórz folder *Kod*, następnie wybierz folder *Rozdzial01* i kliknij przycisk *New Folder*, wpisz nazwę folderu — `WitajDotNetCore` i kliknij przycisk *Create*. Wybierz folder *WitajDotNetCore* i kliknij przycisk *Open* albo naciśnij klawisz *Enter*.

W Visual Studio Code wybierz z menu pozycję *View/Integrated Terminal* albo naciśnij klawisze *Ctrl+`*.

W oknie *Terminal* wpisz następujące polecenie:

```
dotnet new console
```

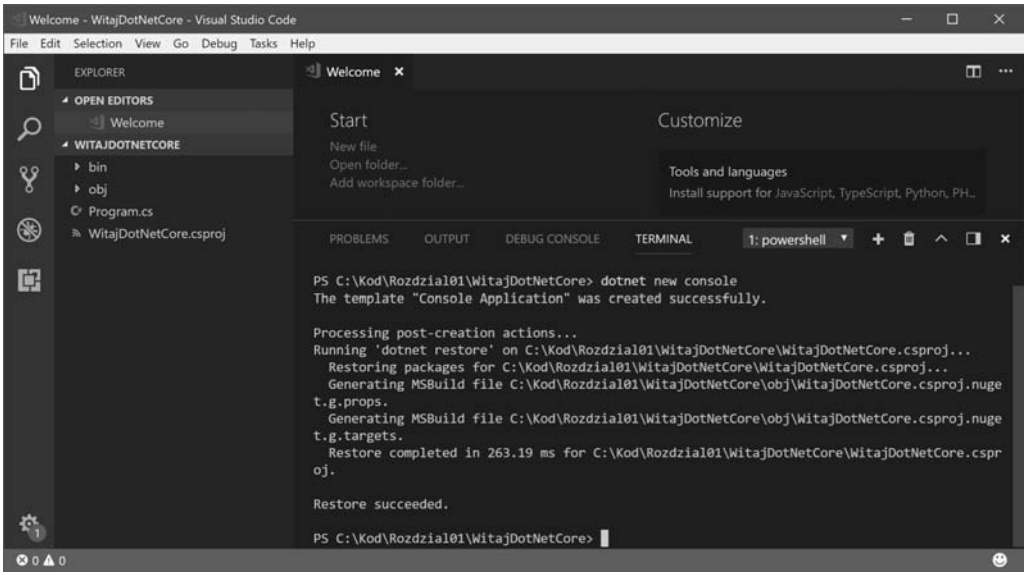
Zobaczysz, że polecenie `dotnet` tworzy w aktualnym folderze nowy projekt aplikacji konsoli, a w oknie *Explorer* pojawia się lista utworzonych plików. Całość wygląda tak jak na pierwszym rysunku na następnej stronie.

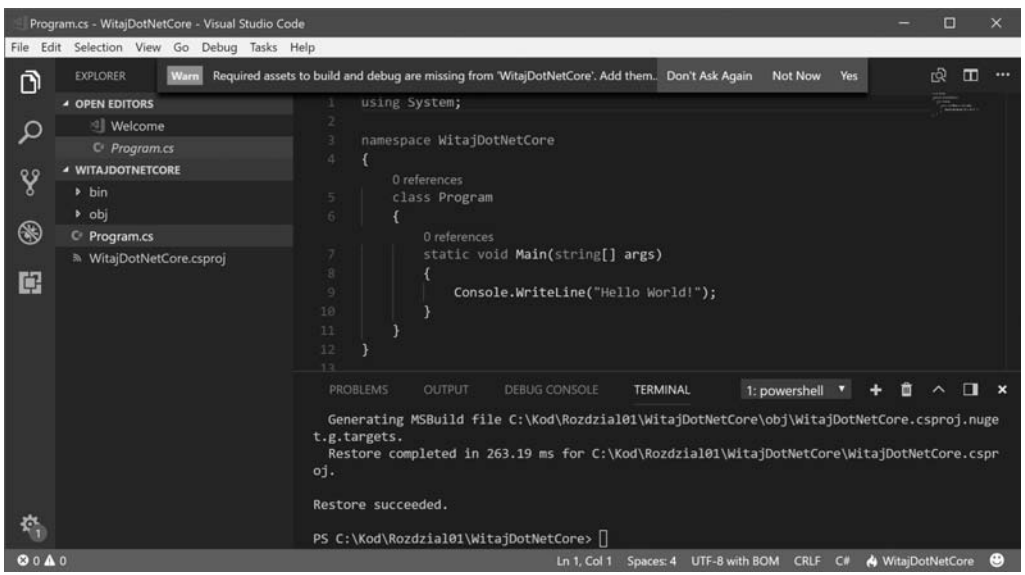
W oknie *EXPLORER* kliknij plik *Program.cs*, aby otworzyć go w oknie edytora.

Jeżeli zobaczysz komunikat mówiący o brakujących elementach, taki jak na drugim rysunku na następnej stronie, to kliknij przycisk *Yes*.

Zmień tekst, który ma zostać wypisany w konsoli, na `Witaj, .NET Core!`

W menu wybierz pozycję *File/Auto Save*. Dzięki temu nie trzeba będzie pamiętać o zapisywaniu kodu programu przed uruchomieniem kompilacji.





Kompilowanie kodu w Visual Studio Code

Wybierz z menu pozycję *View/Integrated Terminal* albo naciśnij klawisze *Ctrl+`*, a następnie wprowadź poniższe polecenie:

```
dotnet run
```

Tekst, który pojawi się w oknie *TERMINAL*, będzie przedstawiał wyniki działania uruchomionej aplikacji.

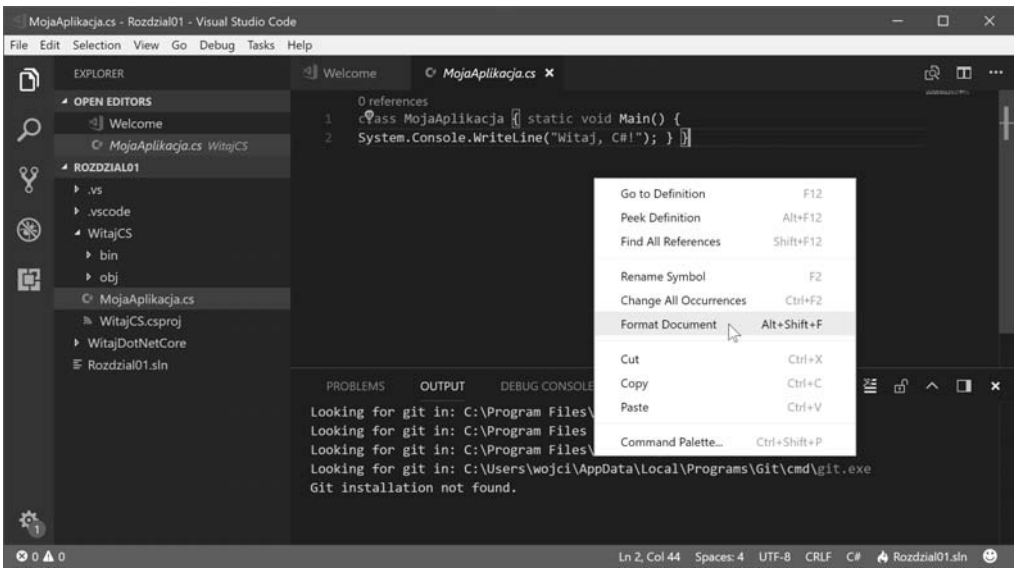
Automatyczne formatowanie kodu

W Visual Studio Code wybierz z menu pozycję *File/Open*, a następnie otwórz folder *Rozdzial01*.

W oknie *Explorer* rozwiń projekt *WitajCS* i zaznacz plik *MyApp.cs*.

Kliknij przycisk *Yes*, jeżeli pojawi się komunikat o konieczności dołączenia wymaganych elementów.

W oknie Visual Studio Code kliknij prawym przyciskiem myszy okno z kodem i wybierz z menu kontekstowego pozycję *Format Document* albo naciśnij klawisze *Alt+Shift+F*, tak jak na poniższym rysunku:



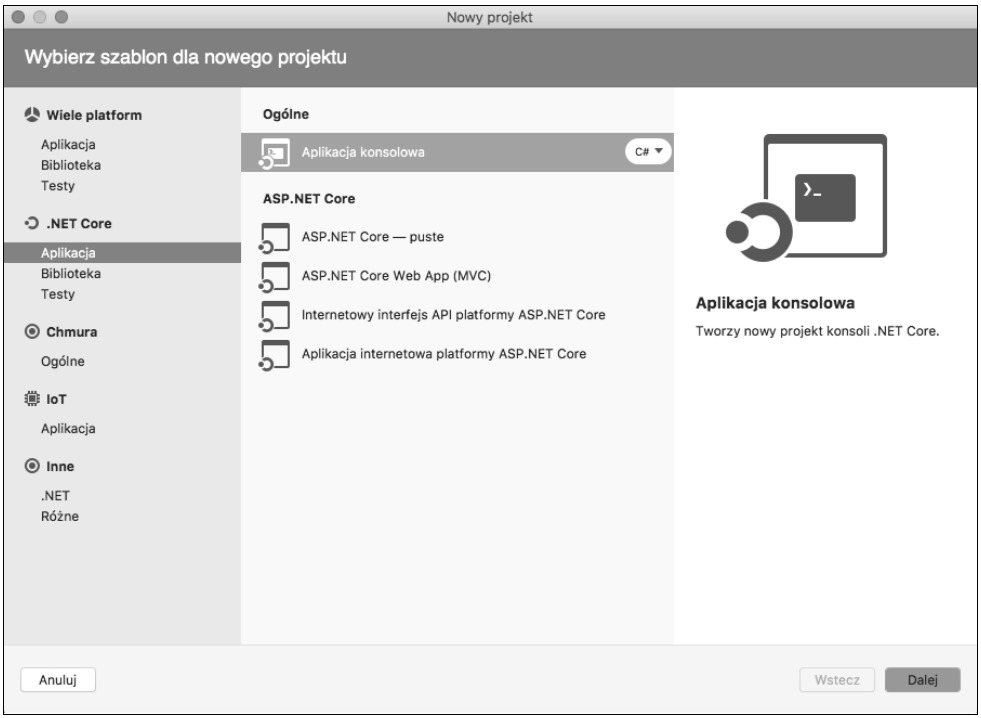
Visual Studio Code jest szybko rozwijającą się alternatywą dla Visual Studio 2017 dla Windowsa.

Pisanie i kompilowanie kodu za pomocą Visual Studio for Mac

Uruchom Visual Studio for Mac i wybierz z menu pozycję *Plik/Nowy projekt*.

Z listy po lewej stronie w sekcji *.NET Core* wybierz pozycję *Aplikacja*.

Z listy szablonów projektów znajdującej się na środku okna wybierz pozycję *Aplikacja konsolowa*, a następnie kliknij przycisk *Dalej*, tak jak na poniższym rysunku:



W kroku *Konfiguruj nowy element Aplikacja konsolowa* z listy rozwijanej *Framework* wybierz opcję *.NET Core 2.0* i kliknij przycisk *Dalej*.

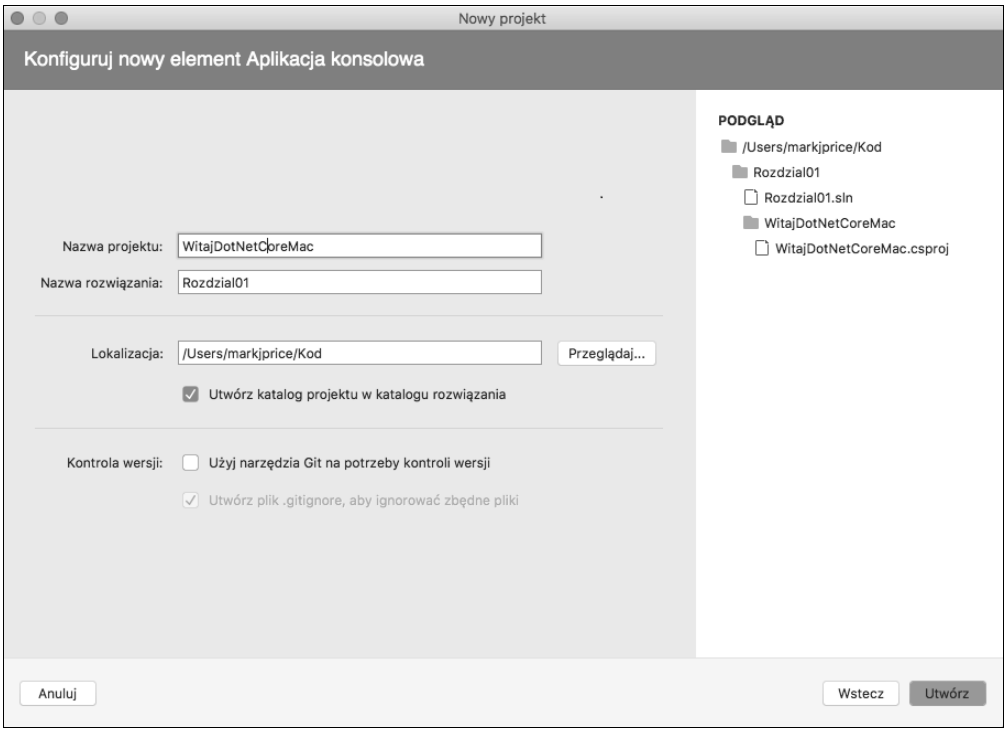
W następnym kroku *Konfiguruj nowy element Aplikacja konsolowa* wprowadź nazwę projektu — `WitajDotNetCoreMac`, jako *Nazwa rozwiązania* wpisz `Rozdzial01`, a w polu *Lokalizacja* podaj folder *Kod*, tak jak na pierwszym rysunku na następnej stronie. Na koniec kliknij przycisk *Utwórz*.

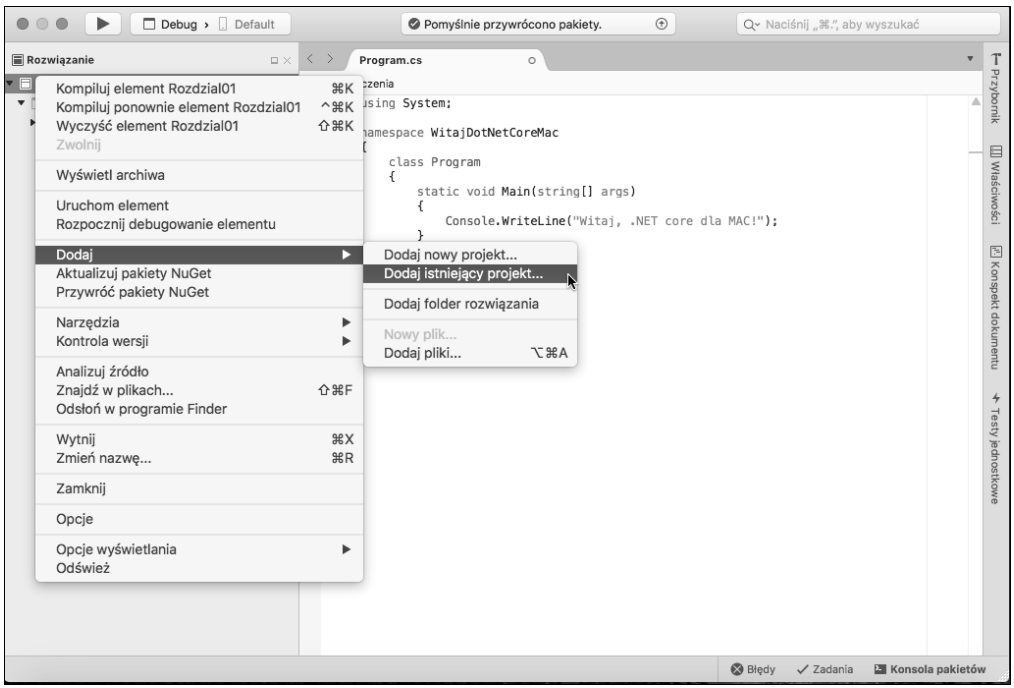
Zmień tekst, który ma zostać wypisany w konsoli, np. na `Witaj, .NET Core dla Mac!`

W Visual Studio for Mac wybierz z menu pozycję *Uruchamianie/Uruchom bez debugowania* albo naciśnij klawisze *Cmd+Option+Enter*.

Tekst, który pojawi się w oknie *Terminal*, będzie wynikiem działania naszej aplikacji.

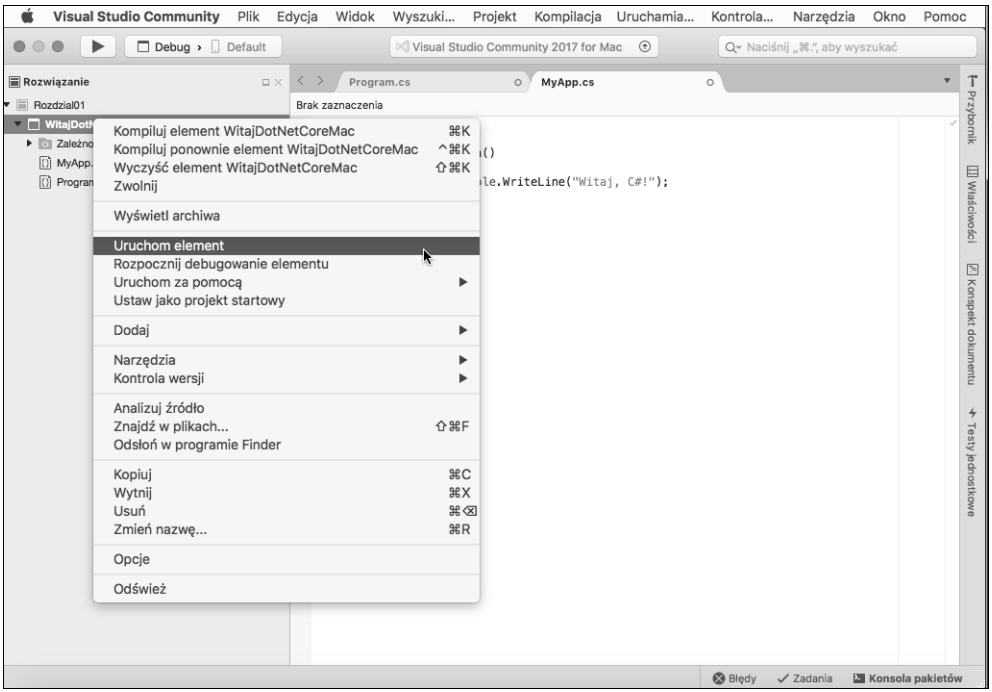
W panelu *Rozwiązanie* kliknij prawym przyciskiem myszy pozycję *Rozdział01*, a następnie wybierz z menu kontekstowego pozycję *Dodaj/Dodaj istniejący projekt…*, tak jak na drugim rysunku na następnej stronie.





W folderze *WitajCS* wybierz plik *WitajCS.cs.proj*.

W panelu *Rozwiązanie* kliknij prawym przyciskiem myszy projekt *WitajCS*, a z menu kontekstowego wybierz pozycję *Uruchom element*, tak jak na poniższym rysunku:



Następne kroki

Teraz już wiesz, jak tworzyć i kompilować proste aplikacje środowiska .NET Core w systemach Windows i macOS (praca w Linuksie jest równie prosta).

Będziesz w stanie wykonać prawie wszystkie zadania z kolejnych rozdziałów tej książki, korzystając z Visual Studio 2017 w systemie Windows, z Visual Studio for Mac albo z Visual Studio Code w systemach Windows, macOS lub Linux.

Zarządzanie kodem źródłowym przy użyciu platformy GitHub

Git to powszechnie stosowany system zarządzania kodem źródłowym. **GitHub** to firma, witryna WWW oraz aplikacja, która ułatwia korzystanie z systemu Git.

Używałem platformy GitHub do przechowywania rozwiązań wszystkich praktycznych ćwiczeń, jakie zamieszczam na końcu każdego z rozdziałów. Możesz je przejrzeć i pobrać pod poniższym adresem:

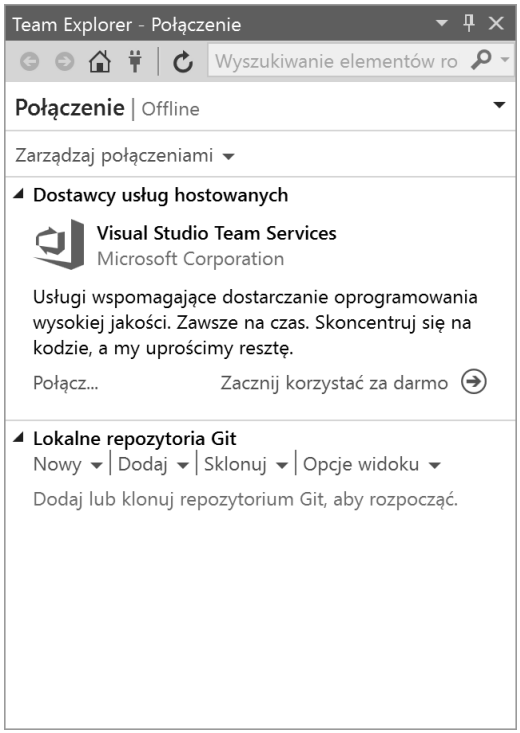
https://github.com/markjprice/cs7dotnetcore2.

Używanie systemu Git w Visual Studio 2017

Visual Studio 2017 ma wbudowane mechanizmy obsługi systemu Git i platformy GitHub, jak również własnego systemu Microsoftu o nazwie **Visual Studio Team Services** (**VSTS**).

Używanie okna Team Explorer

W Visual Studio 2017 wybierz z menu pozycję *Widok/Team Explorer*, żeby wyświetlić okno przedstawione na poniższym rysunku:



Oczywiście zachęcam Cię do utworzenia własnego konta w sieciowym systemie zarządzania kodem źródłowym, choć możesz też pobrać zawartość mojego repozytorium GitHub bez zakładania konta.

Klonowanie repozytorium GitHub

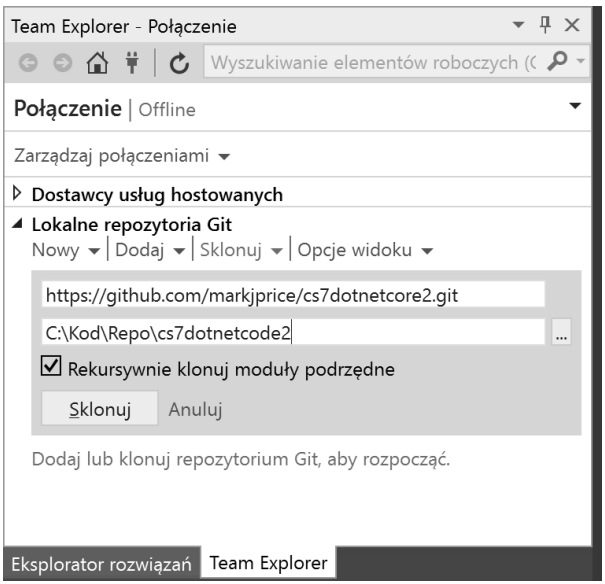
W oknie *Team Explorer* rozwiń sekcję *Lokalne repozytoria Git*, a następnie kliknij menu *Sklonuj* i wprowadź poniższy adres URL, wskazujący na repozytorium Git do sklonowania:

https://github.com/markjprice/cs7dotnetcore2.git.

Wprowadź też ścieżkę do sklonowanego repozytorium:

C:\Kod\Repo\cs7dotnetcore2.

Jeżeli wszystko wygląda tak jak na poniższym rysunku, to kliknij przycisk *Sklonuj*.



Poczekaj, aż repozytorium zostanie sklonowane.

Teraz masz na swoim dysku lokalną kopię rozwiązań wszystkich praktycznych zadań, jakie zamieściłem w tej książce.

Zarządzanie repozytorium GitHub

Kliknij dwukrotnie repozytorium *cs7dotnetcore2*, żeby otworzyć widok szczegółowy.

Możesz kliknąć poszczególne opcje w sekcji *Projekt*, żeby przejrzeć listę *Przychodzących zatwierdzeń* albo listę problemów lub inne informacje o repozytorium.

Możesz też kliknąć dwukrotnie dowolną pozycję w sekcji *Rozwiązania*, żeby otworzyć ją w *Eksploratorze rozwiązań*.

Używanie systemu Git w Visual Studio Code

Visual Studio Code może używać systemu Git, ale wykorzystuje przy tym zainstalowany w systemie operacyjnym pakiet Git. Musisz zatem sam zainstalować pakiet Git 2.0 albo nowszy. Pod poniższym adresem możesz pobrać niezbędny instalator:

https://git-scm.com/download.

> Jeżeli wolisz korzystać z programów z interfejsem graficznym, to pod poniższym adresem możesz pobrać program GitHub Desktop:
>
> *https://desktop.github.com*.

Konfigurowanie systemu Git z wiersza poleceń

Uruchom *Terminal* i wprowadź poniższe polecenie, żeby sprawdzić aktualną konfigurację:

```
git config --list
```

W wypisanych w odpowiedzi danych powinny się znaleźć Twoja nazwa użytkownika oraz adres e-mail, ponieważ będą one używane przy każdym wykonywanym zatwierdzeniu:

```
...pozostałe opcje konfiguracji...
user.name=Mark J. Price
user.email=markjprice@gmail.com
```

Jeżeli Twoja nazwa użytkownika i adres e-mail nie zostały jeszcze zapisane, możesz to zrobić, wprowadzając poniższe polecenia. Pamiętaj o podaniu w tych instrukcjach własnych danych, nie moich:

```
git config --global user.name "Mark J. Price"
git config --global user.email markjprice@gmail.com
```

Poszczególne opcje konfiguracji możesz skontrolować za pomocą takich instrukcji jak poniższa:

```
git config user.name
```

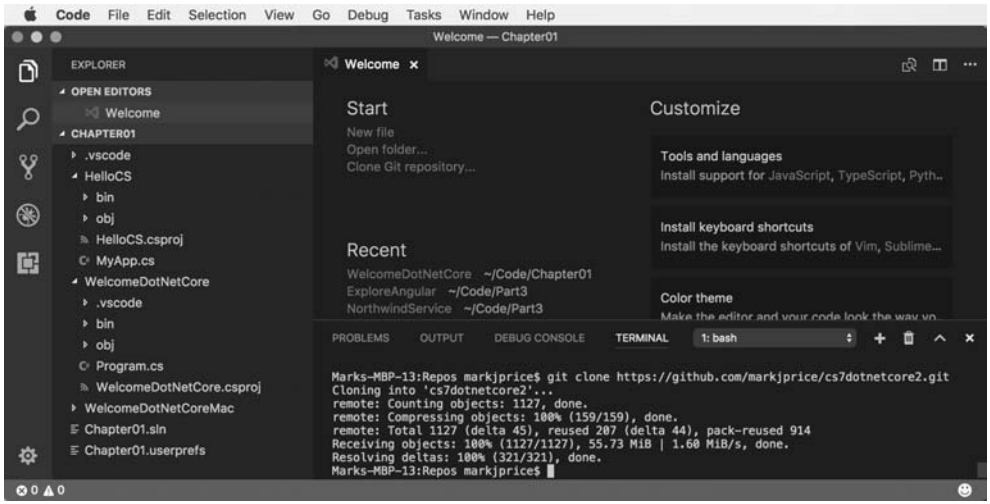
Zarządzanie systemem Git w Visual Studio Code

Uruchom Visual Studio Code i otwórz folder *Kod*.

Wybierz z menu pozycję *View/Integrated Terminal* albo naciśnij klawisze *Ctrl+`*, a następnie wprowadź poniższe polecenia:

```
mkdir Repo
cd Repo
git clone https://github.com/markjprice/cs7dotnetcore2.git
```

Klonowanie wszystkich rozwiązań dla poszczególnych rozdziałów na lokalny dysk twardy może zająć kilka minut, a całość będzie wyglądać tak jak na rysunku na następnej stronie.



Więcej informacji na temat systemów kontroli wersji kodu źródłowego w Visual Studio Code znajdziesz pod poniższym adresem:

https://code.visualstudio.com/Docs/editor/versioncontrol.

Praktyka i ćwiczenia

Sprawdź swoją wiedzę i wiadomości, odpowiadając na kilka prostych pytań. Zyskaj trochę doświadczenia w zakresie tematów omawianych w tym rozdziale.

Ćwiczenie 1.1 — sprawdź swoją wiedzę

Odpowiedz na poniższe pytania:

1. Dlaczego programista może używać różnych języków (np. C# i F#) podczas pisania aplikacji działających w .NET Core?
2. Co należy wpisać w wierszu poleceń, żeby skompilować i uruchomić kod źródłowy w C#?
3. Jakie skróty klawiszowe zdefiniowane są w zbiorze Visual C# dla operacji zapisywania, kompilowania i uruchamiania aplikacji bez podłączania debugera?
4. Jaki skrót klawiszowy w Visual Studio Code otwiera okno *Integrated Terminal*?
5. Czy Visual Studio 2017 jest lepsze od Visual Studio Code?
6. Czy .NET Core jest lepsza od .NET Framework?
7. Czym różni się .NET Native od .NET Core?
8. Czym jest .NET Standard i dlaczego jest to tak ważne?

9. Jaka jest różnica między systemem Git a platformą GitHub?
10. Jak się nazywa metoda startowa w aplikacjach konsoli środowiska .NET i jak należy ją deklarować?

Ćwiczenie 1.2 — ćwicz C# gdzie się da

Do ćwiczenia języka C# nie potrzebujesz Visual Studio 2017, Visual Studio for Mac ani Visual Studio Code.

Wystarczy, że odwiedzisz jedną z poniższych stron WWW i zaczniesz pisać kod.

- **.NET Fiddle**: *https://dotnetfiddle.net/*.
- **Cloud9**: *https://c9.io/web/sign-up/free*.

Ćwiczenie 1.3 — dalsza lektura

Skorzystaj z poniższych adresów, żeby dowiedzieć się więcej o zagadnieniach poruszanych w tym rozdziale.

- **Witaj, .NET Core**: *http://dotnet.github.io*.
- **Narzędzia wiersza poleceń .NET Core**: *https://github.com/dotnet/cli*.
- **.NET Core, CoreCLR**: *https://github.com/dotnet/coreclr/*.
- **Plany rozwoju .NET Core**: *https://github.com/dotnet/core/blob/master/roadmap.md*.
- **Często zadawane pytania dotyczące .NET Standard**: *https://github.com/dotnet/standard/blob/master/docs/faq.md*.
- **Dokumentacja Visual Studio**: *https://docs.microsoft.com/en-us/visualstudio/*.
- **Blog Visual Studio**: *https://blogs.msdn.microsoft.com/visualstudio/*.
- **Git i Team Services**: *https://www.visualstudio.com/en-us/docs/git/overview*.
- **Najprostszy sposób na podłączenie repozytoriów GitHub do Visual Studio**: *https://visualstudio.github.com/*.

Podsumowanie

W tym rozdziale przygotowaliśmy środowisko programistyczne, skorzystaliśmy z wiersza poleceń systemu Windows oraz terminala systemu macOS, żeby uruchomić aplikację konsoli. Użyliśmy również Visual Studio 2017, Visual Studio for Mac oraz Visual Studio Code, żeby przygotować podobne aplikacje. Omówiliśmy też różnice pomiędzy .NET Framework, .NET Core, .NET Standard i .NET Native.

W następnym rozdziale nauczymy się mówić w języku C#.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ż


PROGRAM
PARTNERSKI
GRUPY 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
GRUPA
Helion

Wieloplatformowa aplikacja — profesjonalny deweloper!

Powszechnie uważa się, że C# jest świetnym językiem ogólnego zastosowania, który nadaje się do tworzenia każdego rodzaju oprogramowania — od serwisów internetowych, przez aplikacje biznesowe, aż po gry. Oprogramowanie takie działa na komputerach biurkowych, serwerach, urządzeniach mobilnych, a także na specjalizowanych systemach do gier. Z kolei .NET Core pozwala na tworzenie rozwiązań serwerowych w chmurze oraz na pracę z urządzeniami wirtualnej lub rozszerzonej rzeczywistości. To wszystko sprawia, że technologie C# i .NET Core umożliwiają tworzenie nowoczesnych systemów, które będą wydajnie działać na wielu różnych platformach.

Ta książka jest znakomitym, spójnym i bardzo praktycznym podręcznikiem do nauki języka C#. Pokazuje również najlepsze praktyki stosowane na platformie .NET Core. Z publikacji skorzysta każdy, kto pragnie zdobyć dobre przygotowanie do pracy z językiem i platformą. Znalazły się tu przystępnie wyłożone podstawy języka C#, a także sposoby debugowania kodu i zasady programowania obiektowego. Nie zabrakło informacji o najnowszych elementach języka C# 7.1, takich jak domyślne literały, krotki, dopasowywanie wzorców, zmienne typu out i inne. Dokładnie opisano biblioteki klas .NET Standard 2.0. Zaprezentowano najważniejsze rodzaje aplikacji, takie jak witryny i aplikacje internetowe, serwisy sieciowe, aplikacje UWP oraz aplikacje mobilne.

W książce między innymi:

- Solidne podstawy C# 7.1 i .NET Core 2.0
- Profesjonalne aplikacje w ASP.NET Core 2.0 i wielozadaniowość
- Stosowanie Entity Framework Core oraz LINQ do pracy nad danymi
- Korzystanie ze środowiska UWP oraz z Xamarin.Forms
- Szyfrowanie, strumienie i serializacja
- Planowane funkcje języka C# w wersji 8

Mark J. Price specjalizuje się w programowaniu w języku C#. Ma certyfikat MCSD oraz Episerver. Na początku XXI wieku pisał oficjalne kursy programowania dla Microsoftu, włączając w to kursy języka C#. Poza doświadczeniem w rozwijaniu aplikacji i pisaniu poprawnego kodu ma duże umiejętności edukacyjne: potrafi efektywnie przekazać wiedzę zarówno profesjonalistom z wieloletnim doświadczeniem, jak i 16-latkom stawiającym pierwsze kroki w pisaniu kodu. Obecnie tworzy i prowadzi kursy dotyczące usług Digital Experience Cloud firmy Episerver.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
WWW.SZKOLENIA.HELION.PL
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-4450-1
9 788328 344501
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 99,00 zł
Packt>