Wydanie VI

C# 10 i .NET 6

dla programistów aplikacji wieloplatformowych

Twórz aplikacje, witryny WWW oraz serwisy sieciowe za pomocą ASP.NET Core 6, Blazor i EF Core 6 w Visual Studio 2022 i Visual Studio Code

Mark J. Price

Helion

Packt

Tytuł oryginału: C# 10 and .NET 6 - Modern Cross-Platform Development Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code, 6th Edition

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-9074-4

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/c10ne6
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/c10ne6.zip

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

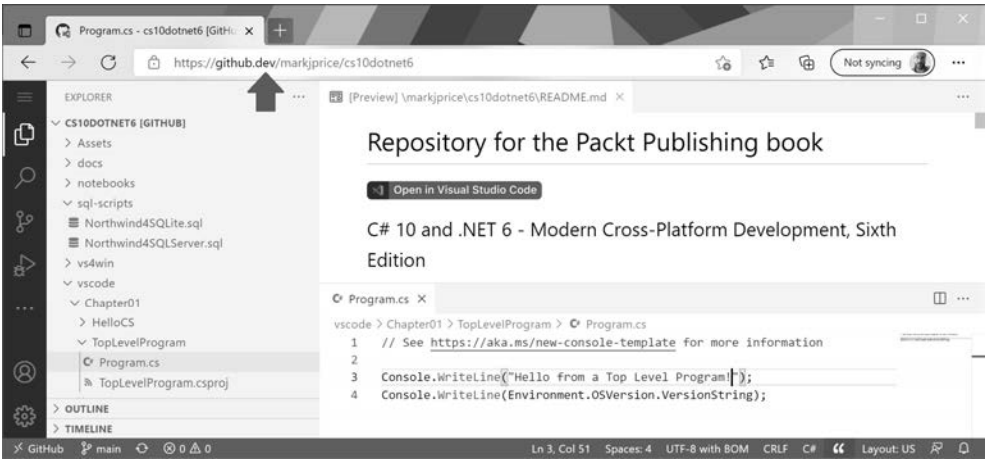
ROZDZIAŁ 1

Cześć, C#! Witaj, .NET!

W pierwszym rozdziale zajmiemy się przygotowaniem środowiska programistycznego i przedstawimy podobieństwa i różnice pomiędzy nowoczesnymi środowiskami .NET, takimi jak .NET Core, .NET Framework, Mono, Xamarin i .NET Standard. Przygotujemy tu najprostszą z możliwych aplikację napisaną w języku C# 10 i działającą w środowisku .NET 6. Będziemy przy tym używać różnych edytorów kodu i poznamy kilka dobrych miejsc do szukania pomocy.

Całość kodu źródłowego prezentowanego w tej książce, w tym wszystkie projekty aplikacji, znajdziesz pod adresem *https://github.com/markjprice/cs10dotnet6*. Z kolei spolonizowaną wersję kodu możesz pobrać pod adresem *https://ftp.helion.pl/przyklady/c10ne6.zip*.

Jeżeli w podanym adresie serwisu GitHub zmienisz domenę *.com* na *.dev*, to umieścisz repozytorium kodu w edytorze Visual Studio Code for Web (zobacz rysunek 1.1).



Rysunek 1.1. Edytor Visual Studio Code for Web z repozytorium GitHuba dla tej książki

To doskonałe rozwiązanie wspomagające pracę z tradycyjnym edytorem kodu nad zadaniami z tej książki. Umożliwia łatwe porównanie kodu z obu rozwiązań i w razie potrzeby skopiowanie wybranych części.

W tekście książki będę używał pojęcia **nowoczesne .NET**, które oznacza .NET 6 i jego poprzednika .NET 5, wywodzące się z platformy .NET Core. Z kolei terminu **stare .NET** będę używał zbiorczo, odnosząc się do platform .NET Framework, Mono, Xamarin i .NET Standard. Nowoczesne .NET jest środowiskiem łączącym te starsze platformy i standardy.

Po pierwszym rozdziale książkę można podzielić na trzy części: pierwszą — poświęconą gramatyce i słownictwu języka C#, drugą — opisującą typy danych w .NET używane do tworzenia aplikacji i trzecią — przytaczającą przykłady typowych aplikacji wieloplatformowych, które można tworzyć w języku C# i na platformie .NET.

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.

Nie musisz od razu znać wszystkich zawiłych szczegółów. Będzie to coś, co przyjdzie z czasem, gdy zaczniesz tworzyć własne aplikacje i wykraczać poza to, czego może Cię nauczyć jakakolwiek książka.

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 szybko rozwijających się technologiach, takich jak C# i .NET, oraz aplikacjach, które możesz za ich pomocą zbudować.

W tym rozdziale zostaną omówione następujące zagadnienia:

- konfigurowanie środowiska programistycznego;
- poznawanie .NET;
- tworzenie aplikacji konsoli za pomocą Visual Studio 2022;
- tworzenie aplikacji konsoli za pomocą Visual Studio Code;
- używanie interaktywnych notatników .NET do sprawdzania działania kodu;
- przeglądanie folderów i plików wchodzących w skład projektu;
- obsługa kodu źródłowego z wykorzystaniem platformy GitHub;
- szukanie pomocy.

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 2022 dla Windows,
- Visual Studio 2022 dla Mac,
- Visual Studio Code dla Windowsa, Maca i Linuksa,
- GitHub Codespaces.

Inne firmy również oferują własne edytory kodu C#, czego przykładem może być JetBrains Rider.

Wybieranie narzędzia i typu aplikacji właściwych do nauki

Czy istnieje najlepsze narzędzie i najlepszy rodzaj aplikacji do nauki języka C# i platformy .NET?

Podczas nauki najlepszym narzędziem jest takie, które wspomaga przy pisaniu kodu i konfigurowaniu, a jednocześnie nie ukrywa mechanizmów języka. Środowiska z graficznym interfejsem użytkownika są bardzo łatwe w obsłudze, ale czy wiesz, co właściwie robią one w tle? W tym przypadku prostszy edytor kodu, który pozostawia Ci większą kontrolę, a jednocześnie pomaga przy pisaniu kodu będzie najlepszym narzędziem do nauki.

Po tym wstępie można twierdzić, że najlepszym narzędziem jest to, z którym już wcześniej pracowaliśmy, albo to, którego będzie używał Twój zespół w codziennej pracy. Z tego powodu kolejne zadania zawarte w niniejszej książce możesz wykonywać w swoim własnym edytorze kodu C# (lub środowisku IDE). Do wyboru masz Visual Studio Code, Visual Studio dla Windows, Visual Studio dla Mac, a nawet JetBrains Rider.

W trzecim wydaniu tej książki w każdym zadaniu podawałem dokładne instrukcje zarówno dla Visual Studio dla Windows, jak i dla Visual Studio Code. Niestety sprawiało to, że tekst stawał się zagmatwany. W tym wydaniu podaję dokładne instrukcje wykonania działań w obu edytorach kodu C#, ale jedynie w projektach realizowanych w rozdziale 1. W kolejnych rozdziałach będę podawał tylko nazwy projektów oraz ogólne instrukcje do wykonania w wybranym przez Ciebie edytorze kodu.

Najlepszym rodzajem aplikacji przy uczeniu się mechanizmów języka C# oraz wielu bibliotek .NET jest aplikacja, która nie rozprasza dodatkowym, niepotrzebnym kodem. Na przykład nie ma potrzeby tworzenia całej aplikacji graficznej dla systemu Windows tylko po to, żeby nauczyć się posługiwać instrukcją `switch`.

Z tego powodu uważam, że najlepszą metodą uczenia się języka C# i platformy .NET jest tworzenie aplikacji konsoli, co będziemy robić w rozdziałach od 1. do 12. W kolejnych rozdziałach, od 13. do 19., będziemy tworzyć witryny WWW, serwisy sieciowe oraz aplikacje graficzne i mobilne.

Zalety i wady stosowania interaktywnych notatników .NET

Kolejną zaletą Visual Studio Code jest możliwość skorzystania z rozszerzenia interaktywnych notatników .NET (.NET Interactive Notebooks). To rozszerzenie daje łatwy i bezpieczny dostęp do miejsca, w którym można pisać i testować działanie wycinków kodu. Umożliwia ono tworzenie pliku notatnika łączącego w sobie komórki tekstu z formatowaniem Markdown i z kodem C# lub innych języków, takich jak PowerShell, F# lub SQL (do pracy z bazami danych).

Niestety interaktywne notatniki .NET mają też kilka ograniczeń:

- Nie mogą pobierać danych od użytkownika, na przykład nie pozwalają na użycie metod `ReadLine` lub `ReadKey`.
- Nie można przekazywać do nich żadnych argumentów.
- Nie pozwalają definiować własnych przestrzeni nazw.
- Nie udostępniają żadnych narzędzi do debugowania kodu (choć takie narzędzia mają się pojawić w przyszłości).

Programowanie na wielu platformach przy użyciu Visual Studio Code

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.

Visual Studio Code to dobry wybór przy tworzeniu nowoczesnego programowania międzyplatformowego, ponieważ ma obszerny i wciąż rosnący zestaw rozszerzeń obsługujących wiele języków oprócz C#.

Jednocześnie jest lekkim i wieloplatformowym narzędziem, dzięki czemu można je instalować na wszystkich platformach, na których będą wdrażane Twoje aplikacje, co usprawnia naprawianie błędów lub wykonywanie innych działań. Visual Studio Code sprawia, że programista może korzystać z wieloplatformowego edytora kodu, aby za jego pomocą tworzyć wieloplatformowe aplikacje.

Visual Studio Code ma wiele funkcji wspomagających tworzenie aplikacji dla sieci WWW, ale aktualnie nie ma tak dobrze rozwiniętych funkcji ułatwiających budowanie aplikacji mobilnych i dla komputerów stacjonarnych.

Visual Studio Code może działać również na procesorach ARM, dzięki czemu można z niego korzystać na komputerach Apple Silicon albo Raspberry Pi.

Visual Studio Code jest aktualnie najpopularniejszym środowiskiem IDE. W ankiecie przeprowadzonej przez serwis StackOverflow w 2021 roku do jego używania przyznało się ponad 70% programistów.

Tworzenie systemów chmurowych w GitHub Codespaces

GitHub Codespaces jest doskonale skonfigurowanym środowiskiem programistycznym bazującym na Visual Studio Code, które można przekształcić w środowisko uruchamiane w chmurze z dostępem poprzez dowolną przeglądarkę. Obsługuje ono repozytoria Gita i rozszerzenia, a dodatkowo udostępnia interfejs wiersza poleceń, umożliwiający edytowanie, uruchamianie i testowanie kodu z dowolnego urządzenia.

Tworzenie aplikacji z wykorzystaniem Visual Studio dla komputerów Mac

Microsoft Visual Studio 2022 dla Mac pozwala tworzyć większość rodzajów aplikacji, w tym aplikacje konsoli, witryny WWW, serwisy sieciowe, aplikacje dla komputerów stacjonarnych i dla urządzeń mobilnych.

Aby tworzyć oprogramowanie dla systemów operacyjnych firmy Apple, takich jak iOS (dla iPhone'a i iPada), trzeba mieć dostępny system macOS i środowisko Xcode.

Tworzenie aplikacji za pomocą Visual Studio

Microsoft Visual Studio 2022 dla Windows umożliwia tworzenie większości rodzajów aplikacji, w tym aplikacje konsoli, witryny WWW, serwisy sieciowe, aplikacje dla komputerów stacjonarnych i dla urządzeń mobilnych. Co prawda można używać Visual Studio 2022 w połączeniu z rozszerzeniami Xamarin, aby tworzyć wieloplatformowe aplikacje mobilne, ale do ich skompilowania potrzebny jest system macOS i środowisko Xcode.

Visual Studio działa tylko w systemie Windows w wersji 7 SP1 lub nowszej. Dodatkowo, trzeba je uruchomić w systemie Windows 10 lub Windows 11, aby tworzyć aplikacje **Universal Windows Platform** (**UWP**) instalowane za pomocą Microsoft Store i uruchamiane w wydzielonej piaskownicy.

Jakich narzędzi używałem?

Do pisania i testowania kodu prezentowanego w tej książce wykorzystywałem następujący sprzęt:

- laptop HP Spectre (z procesorem Intel),
- komputer Apple Silicon Mini (z procesorem M1),
- komputer Raspberry Pi 400 (z procesorem ARM v8).

Podczas pracy używałem też następującego oprogramowania:

- Visual Studio Code działające w systemie:
  - macOS na komputerze Apple Silicon Mini (z procesorem M1),
  - Windows 10 na laptopie HP Spectre (z procesorem Intel),
  - Ubuntu 64 na komputerze Raspberry Pi 400.
- Visual Studio 2022 dla Windows działające w systemie:
  - Windows 10 na laptopie HP Spectre (z procesorem Intel).
- Visual Studio 2022 dla Mac działające w systemie:
  - macOS na komputerze Apple Silicon Mini (z procesorem M1).

Mam nadzieję, że również masz dostęp do różnego sprzętu i oprogramowania, ponieważ poznawanie różnic między platformami pozwoli poznać wyzwania związane z tworzeniem oprogramowania. Pamiętaj jednak, że dowolna z wymienionych wyżej kombinacji całkowicie wystarcza do zgłębiania podstaw języka C# i platformy .NET oraz metod budowania praktycznych aplikacji i witryn WWW.

Więcej informacji

Dodatkowe informacje na temat pisania kodu w języku C# na komputerze Raspberry Pi 400 z systemem Ubuntu Desktop 64 znajdziesz w napisanym przeze mnie artykule dostępnym pod adresem *https://github.com/markjprice/cs9dotnet5-extras/blob/main/raspberry-pi-ubuntu64/README.md*.

Instalowanie na wielu platformach

Wybór IDE i systemu operacyjnego na potrzeby tworzenia oprogramowania nie ogranicza nam możliwości instalowania go na innych platformach.
.NET 6 pozwala zainstalować oprogramowanie na następujących platformach:

- **Windows:** 7 SP1 i nowsze, Windows 10 w wersji 1607 lub nowszy, w tym Windows 11. Windows Server 2012 R2 SP1 i nowsze, Nano Server w wersji 1809 lub nowszej.
- **Mac:** macOS Mojave (wersja 10.14) i nowsza.
- **Linux:** Alpine Linux 3.13 lub nowszy. CentOS 7 lub nowszy. Debian 10 lub nowszy. Fedora 32 lub nowszy. openSUSE 15 lub nowszy. RHEL 7 i nowszy. SUSE Enterprise Linux 12 SP2 lub nowszy. Ubuntu 16.04, 18.04, 20.04 lub nowszy.
- **Android:** API 21 lub nowszy.
- **iOS:** 10 lub nowszy.

Wprowadzenie obsługi systemów Windows ARM64 w platformie .NET 5 sprawiło, że możemy tworzyć i instalować oprogramowanie na urządzeniach Windows ARM, takich jak Microsoft Surface Pro X. Okazuje się jednak, że praca na komputerze Apple Mac M1 z wykorzystaniem systemu Parallels i maszyny wirtualnej Windows 10 ARM jest dwa razy szybsza!

Pobieranie i instalowanie Visual Studio 2022 dla Windows

Wielu zawodowych programistów pracujących z technologiami Microsoftu w swojej codziennej pracy używa Visual Studio 2022 dla Windows. Nawet jeżeli do wykonywania zadań z tej książki używasz Visual Studio Code, to i tak zalecam Ci zapoznanie się z Visual Studio 2022.

Jeżeli nie masz komputera z systemem Windows, to możesz pominąć ten punkt i przejść do następnego, w którym zajmiemy się pobieraniem i instalowaniem Visual Studio Code w systemach macOS i Linux.

W październiku 2014 roku Microsoft udostępnił bezpłatną edycję Visual Studio dla Windows, której używać mogą studenci, osoby rozwijające projekty o otwartych źródłach oraz osoby prywatne. Ten wariant edytora nazywa się Community Edition. Do pracy z tą książką możesz wykorzystać dowolny wariant Visual Studio. Jeżeli nie masz go jeszcze zainstalowanego w systemie, to możesz zrobić to teraz.

1. Z adresu *https://visualstudio.microsoft.com/downloads/* pobierz instalator Visual Studio 2022 w wersji 17 lub nowszej.
2. Uruchom instalator.
3. W karcie *Workloads* zaznacz następujące opcje:
   - *ASP.NET and web development*
   - *Azure development*
   - *.NET desktop development*
   - *Desktop development with C++*
   - *Universal Windows Platform development*
   - *Mobile development with .NET*
4. Na karcie *Individual components*, w sekcji *Code tools*, zaznacz następujące opcje:
   - *Class Designer*
   - *Git for Windows*
   - *PreEmptive Protection - Dotfuscator*
5. Jeżeli chcesz używać spolszczonej wersji Visual Studio, na karcie *Language packs* wybierz opcję *Polish*.
6. Kliknij przycisk *Install* i czekaj, aż instalator pobierze wybrane komponenty i je zainstaluje.
7. Po zakończeniu instalacji kliknij przycisk *Launch*.
8. Przy pierwszym uruchomieniu Visual Studio program poprosi Cię o zalogowanie się. Jeżeli masz już konto Microsoftu, możesz użyć go w tym miejscu. Jeżeli nie masz jeszcze konta, możesz je utworzyć pod adresem *https://signup.live.com*.
9. Przy pierwszym uruchomieniu Visual Studio program poprosi Cię o skonfigurowanie swojego środowiska. W sekcji *Development Settings*

wybierz pozycję *Visual C#*, natomiast z listy schematów kolorystyki możesz wybrać najbardziej odpowiadający Ci schemat. Ja lubię używać schematu *Blue*.

10. Jeżeli chcesz dostosować skróty klawiszowe, to wybierz z menu pozycję *Tools/Options…*, a w otwartym oknie wybierz sekcję *Keyboard*.
11. Jeżeli chcesz zmienić język Visual Studio na polski, to wybierz z menu pozycję *Tools/Options…*, w otwartym oknie wybierz sekcję *Environment/International Settings*, a z listy *Language* wybierz pozycję *Polski*. Zmiana języka zostanie wprowadzona po ponownym uruchomieniu Visual Studio.

Skróty klawiszowe w Microsoft Visual Studio dla Windows

W tej książce staram się unikać podawania skrótów klawiszowych, ponieważ można je bardzo łatwo zmienić. Jeżeli takie skróty są jednakowe w wielu różnych edytorach kodu, to będę je prezentował w tekście książki. Jeżeli chcesz przejrzeć i zmienić skróty klawiszowe w swoim edytorze, możesz to zrobić zgodnie z instrukcjami dostępnymi pod adresem *https://docs.microsoft.com/pl-pl/visualstudio/ide/identifying-and-customizing-keyboard-shortcuts-in-visual-studio*.

Pobieranie i instalowanie Microsoft Visual Studio Code

Edytor Visual Studio Code bardzo się rozwinął w ciągu kilku ostatnich lat, co przełożyło się na ciągle rosnącą popularność. Dla odważnych, którzy chcą cały czas testować najnowsze rozwiązania, dostępne są wydania z linii Insider, zawierające codzienne kompilacje nowych wersji edytora.

Nawet jeżeli planujesz pracować wyłącznie z Visual Studio 2022 dla Windows, polecam zainstalować Visual Studio Code i spróbować wykonać z jego pomocą zadania z tego rozdziału. Potem możesz zdecydować, czy chcesz dalej używać tego edytora, czy w pozostałej części książki wolisz pracować z Visual Studio 2022.

Jesteśmy już gotowi do pobrania i zainstalowania Visual Studio Code, rozszerzenia dla języka C#, .NET Interactive Notebooks oraz pakietu .NET SDK.

1. Pobierz i zainstaluj Visual Studio Code w wydaniu stabilnym albo z kanału Insiders, korzystając z adresu *https://code.visualstudio.com*.

> **Więcej informacji**
>
> Jeżeli potrzebujesz pomocy przy instalowaniu Visual Studio Code, możesz przeczytać oficjalny poradnik instalowania dostępny pod adresem *https://code.visualstudio.com/docs/setup/setup-overview*.

2. Pobierz i zainstaluj pakiet .NET SDK w wersji 3.1, 5.0 i 6.0, korzystając z adresu *https://www.microsoft.com/net/download*.

Więcej informacji

Aby nauczyć się w pełni panować nad pakietami .NET SDK, musimy zainstalować kilka wersji. .NET Core 3.1, .NET 5.0 i .NET 6.0 to najaktualniejsze wersje SDK. Można je bezpiecznie instalować w systemie obok siebie. W dalszej części tej książki nauczysz się wybierać wersję SDK potrzebną przy tworzeniu swojego projektu.

3. Aby zainstalować rozszerzenie dla języka C#, musisz najpierw uruchomić aplikację Visual Studio Code.
4. W Visual Studio Code kliknij ikonę *Extensions* albo wybierz z menu pozycję *View/Extensions*.
5. Rozszerzenie *C#* jest jednym z najpopularniejszych, dlatego powinno być widoczne na samym szczycie listy. Możesz też wprowadzić tekst `C#` w okienku wyszukiwania.
6. Kliknij przycisk *Install* i poczekaj na pobranie i zainstalowanie niezbędnych pakietów.
7. W polu wyszukiwania wpisz tekst `.NET Interactive`, aby znaleźć rozszerzenie *.NET Interactive Notebooks*.
8. Kliknij przycisk *Install* i poczekaj na pobranie i zainstalowanie rozszerzenia.

Instalowanie innych rozszerzeń

W kolejnych rozdziałach tej książki będziemy korzystać z większej liczby rozszerzeń. Jeżeli chcesz zainstalować je już teraz, to możesz posiłkować się listą zamieszczoną w poniższej tabeli.

Rozszerzenie	Opis
C# for Visual Studio Code (firmy OmniSharp) `ms-dotnettools.csharp`	Obsługa edytowania kodu C#, kolorowanie składni, IntelliSense, Idź do definicji, Znajdź wszystkie odwołania, obsługa debugowania w .NET Core (CoreCLR) oraz obsługa plików *csproj* w systemach Windows, macOS i Linux.
.NET Interactive Notebooks `ms-dotnettools.dotnet-interactive-vscode`	Dodaje obsługę funkcji .NET Interactive do notatnika Visual Studio Code. Ma własną zależność od rozszerzenia Jupyter (`ms-toolsai.jupyter`).
MSBuild project tools `tinytoy.msbuild-project-tools`	Udostępnia funkcję IntelliSense w plikach projektów MSBuild, w tym automatyczne uzupełnianie elementów `<PackageReference>`.
REST Client `humao.rest-client`	Wysyłanie żądań HTTP i przeglądanie odpowiedzi bezpośrednio w Visual Studio Code.
ILSpy .NET Decompiler `icsharpcode.ilspy-vscode`	Dekompilacja zestawów MSIL. Obsługuje nowoczesne .NET, .NET Framework, .NET Core oraz .NET Standard.

Rozszerzenie	Opis
Azure Functions dla Visual Studio Code `ms-azuretools.vscode-azurefunctions`	Tworzenie, debugowanie, zarządzanie i instalowanie aplikacji bezserwerowych bezpośrednio z VS Code. Ma zależności od rozszerzeń Azure Account (`ms-vscode.azure-account`) i Azure Resources (`ms-azuretools.vscode-azureresourcegroups`).
GitHub Repositories `github.remotehub`	Przeglądanie, wyszukiwanie, edytowanie i tworzenie commitów w dowolnym zdalnym repozytorium GitHuba bezpośrednio z Visual Studio Code.
SQL Server (mssql) dla Visual Studio Code `ms-mssql.mssql`	Bogaty zbiór funkcji przeznaczonych do pracy z bazami danych MS SQL Server, Azure SQL Database i SQL Data Warehouse.
Protobuf 3 support dla Visual Studio Code `zxh404.vscode-proto3`	Kolorowanie i sprawdzanie poprawności składni, wycinki kodu, uzupełnianie i formatowanie kodu, dopasowywanie nawiasów i komentowanie wierszy i bloków kodu.

Rozpoznawanie wersji programu Microsoft Visual Studio Code

Co miesiąc (no prawie) firma Microsoft wydaje nową wersję programu Visual Studio Code, a jeszcze częściej pojawiają się wersje naprawiające błędy. I tak:

- Wersja 1.59 to wydanie z sierpnia 2021 r.
- Wersja 1.39.1 to wydanie poprawkowe z sierpnia 2021 r.

Wersja użyta w tej książce to 1.59, jednak wersja Visual Studio Code jest tu mniej ważna niż wersja rozszerzenia **C# for Visual Studio Code**, które zainstalowaliśmy wcześniej.

Chociaż rozszerzenie dla języka C# nie jest wymagane, zapewnia ono funkcję IntelliSense, a także funkcje w nawigacji po kodzie oraz funkcję debugowania, więc jest to coś, co z pewnością warto zainstalować.

Skróty klawiszowe w Visual Studio Code

W tej książce będę unikał prezentowania skrótów klawiszowych używanych do wykonywania takich zadań jak tworzenie nowego pliku, ponieważ są one zwykle inne w poszczególnych systemach operacyjnych. Będę jednak podawał skróty klawiszowe w sytuacjach wymagających wielokrotnego naciśnięcia danego klawisza, na przykład podczas debugowania. Ten rodzaj skrótów klawiszowych jest też raczej niezmienny, niezależnie od systemu operacyjnego.

Jeżeli chcesz zmienić skróty klawiszowe używane w Twoim Visual Studio Code, możesz to zrobić zgodnie z instrukcjami dostępnymi pod adresem *https://code.visualstudio.com/docs/get-started/keybindings*.

Dobrze jest pobrać sobie plik PDF z listą skrótów klawiszowych właściwych dla Twojego systemu operacyjnego:

- **Windows:** *https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf*.
- **macOS:** *https://code.visualstudio.com/shortcuts/keyboard-shortcuts-macos.pdf*.
- **Linux:** *https://code.visualstudio.com/shortcuts/keyboard-shortcuts-linux.pdf*.

Poznawanie .NET

.NET 6, .NET Core, .NET Framework i Xamarin są powiązanymi ze sobą platformami programistycznymi, których programiści mogą używać do tworzenia aplikacji i usług. W tym podrozdziale spróbuję przedstawić pokrótce każdy z tych wariantów platformy .NET.

Poznawanie .NET Framework

.NET Framework jest platformą programistyczną, w skład której wchodzi środowisko wykonawcze **CLR** (ang. *Common Language Runtime*) zajmujące się wykonywaniem kodu programów, a także biblioteka klas **BCL** (ang. *Base Class Library*), która udostępnia aplikacjom szeroki zakres różnorodnych 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.

Od wersji 4.5.2 .NET Framework stała się oficjalnie składnikiem systemu Windows. Platforma jest zainstalowana na miliardach komputerów, zatem nie powinno się w niej wprowadzać wielkich zmian. Problemy mogą powodować nawet poprawki błędów, dlatego całość nie jest zbyt często aktualizowana.

Od wersji .NET Framework 4.0 wszystkie aplikacje dla komputerów napisane na bazie .NET Framework korzystają z tej samej wersji środowiska CLR oraz z bibliotek zgromadzonych w globalnym zbiorze **GAC** (ang. *Global Assembly Cache*). Może to powodować różnorakie problemy, jeżeli dana aplikacja wymaga do działania konkretnej wersji biblioteki.

> **Dobra praktyka**
>
> Praktycznie rzecz biorąc, .NET Framework jest platformą schodzącą, przeznaczoną wyłącznie dla systemów Windows. Nie powinno się już tworzyć korzystających z niej aplikacji.

Poznawanie projektów Mono, Xamarin i Unity

Powstała też niezależna implementacja platformy .NET, o nazwie **Mono**. Jest ona implementacją wieloplatformową, ale pozostaje daleko w tyle za oficjalną implementacją platformy .NET Framework.

Projekt Mono znalazł swoją niszę, stając się bazą dla platformy mobilnej **Xamarin** oraz różnych wieloplatformowych systemów do programowania gier, takich jak **Unity**.

Microsoft wykupił platformę Xamarin w 2016 r. i teraz rozpowszechnia to, co było bardzo drogim pakietem, jako bezpłatny dodatek do Visual Studio. Istniejące wcześniej Xamarin Studio, które pozwalało na tworzenie wyłącznie aplikacji mobilnych, Microsoft przemianował na Visual Studio for Mac i dodał do niego możliwość tworzenia innych rodzajów aplikacji. W wersji Visual Studio for Mac 2022 zastąpiono dotychczasowy edytor, pochodzący z Xamarin Studio, tym znanym z Visual Studio dla systemów Windows, aby ujednolicić sposób pracy i wydajność wszystkich narzędzi. Visual Studio 2022 dla Mac został też napisany od nowa, tak aby był w pełni aplikacją dla systemu macOS ze zwiększoną stabilnością i możliwością wykorzystania technologii wspomagających wbudowanych w ten system operacyjny.

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.

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".

.NET Core rozwija się bardzo szybko, a dzięki temu, że może być instalowana razem z aplikacją, zyskujemy pewność, że tak często wprowadzane zmiany nie będą miały wpływu na pozostałe aplikacje napisane w .NET Core działające na tym samym komputerze. Usprawnienia, jakie Microsoft wprowadza do .NET Core, nie mogą niestety zostać przeniesione do .NET Framework.

Droga do jednej platformy .NET

Na konferencji Microsoft Build w maju 2020 roku zespół rozwijający platformę .NET ogłosił, że plany unifikacji platformy .NET zostaną opóźnione. Poinformowano, że .NET 5.0 pojawi się 10 listopada 2020 roku i będzie unifikacją wszystkich platform .NET z wyjątkiem platformy mobilnej. Dopiero w listopadzie 2021 roku platforma mobilna również stanie się częścią zunifikowanej platformy .NET 6.0.

.NET Core zostało przemianowane na .NET. Dodatkowo numeracja kolejnych wersji pomija numer 4, aby uniknąć mylenia tej platformy z .NET Framework 4.x. Microsoft zamierza co roku wydawać kolejne wersje .NET w każdym listopadzie, podobnie jak Apple wydaje kolejne wersje systemu iOS, udostępniając je zawsze we wrześniu.

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

Wersja	Data wydania	Wydanie	Opublikowano
.NET Core RC1	listopad 2015	Pierwsze	marzec 2016
.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	Drugie	marzec 2017
.NET Core 2.0	sierpień 2017		
.NET Core for UWP jako część aktualizacji Windows 10 Fall Creators Update	październik 2017	Trzecie	listopad 2017
.NET Core 2.1	maj 2018		
.NET Core 2.2	grudzień 2018		
.NET Core 3.0 (Current)	wrzesień 2019	Czwarte	październik 2019
.NET Core 3.1 (LTS)	listopad 2019		
.NET 5.0 (Current)	listopad 2020	Piąte	listopad 2020
.NET 6.0 (LTS)	listopad 2021	Szóste	listopad 2021
.NET 7.0 (Current)	listopad 2022	Siódme	listopad 2022
.NET 8.0 (LTS)	listopad 2023	Ósme	listopad 2023

Częścią .NET Core 3.1 była też technologia Blazor Server przeznaczona do tworzenia komponentów dla sieci WWW. Firma Microsoft chciała też dołączyć do tego wydania technologię Blazor WebAssembly, ale została ona opóźniona. Ostatecznie znalazła się w opcjonalnym dodatku dla .NET Core 3.1. Wspominam o tym tutaj, ponieważ ten dodatek otrzymał numer wersji 3.2, aby odróżnić go od wersji LTS, jaką było wydanie .NET Core 3.1.

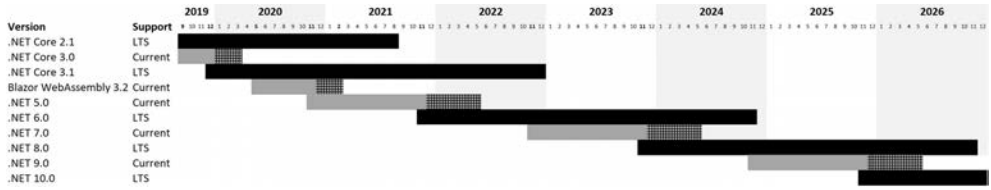
Plany obsługi platformy .NET

Wersje platformy .NET oznaczane są jako **Current** albo **LTS** (ang. *Long-Term Support* — długoterminowa obsługa). Oba oznaczenia wyjaśniam poniżej:

- **LTS.** Te wydania uznaje się za stabilne i nie będą one wymagały wielu poprawek w czasie swojego życia. Warto je stosować w aplikacjach, które raczej nie będą często aktualizowane. Wydania LTS mają otrzymywać aktualizacje do trzech lat od oficjalnego wydania albo do roku po pojawieniu się następnego wydania LTS, zależnie, co będzie dłuższym terminem.
- **Current.** Te wydania udostępniają funkcje, które mogą zostać zmienione w wyniku reakcji użytkowników. Świetnie nadają się do stosowania przy aktywnie rozwijanych aplikacjach, gdyż umożliwiają dostęp do najnowszych usprawnień środowiska. Każde z pomniejszych wydań jest aktualizowane jeszcze przez sześć miesięcy po pojawieniu się nowszego wydania albo do 18 miesięcy od jej udostępnienia.

Oba rodzaje wydań w czasie swojego życia otrzymują wszystkie poprawki związane z bezpieczeństwem i stabilnością systemu. Obsługa tych wydań wymaga jednak, żeby na bieżąco instalować wszystkie pojawiające się poprawki. Na przykład, jeżeli system działa z wersją 1.0, a pojawiła się wersja 1.0.1, to konieczne jest jak najszybsze zainstalowanie wersji 1.0.1.

Aby móc lepiej wybrać między wydaniem Current a LTS, dobrze jest przyjrzeć się diagramowi z rysunku 1.2. W tym diagramie czarne paski reprezentują 3-letnie wydania LTS, natomiast szare paski opisują wydania Current, w których ciemniejsza część oznacza okres 6 miesięcy aktualizacji po wydaniu nowszej wersji.



Rysunek 1.2. Diagram aktualizacji różnych wersji .NET

Na przykład jeżeli projekt powstał przy użyciu .NET Core 3.0 w grudniu 2019 roku, czyli w czasie gdy Microsoft wydał już .NET Core 3.1, to projekt musi zostać zaktualizowany do .NET Core 3.1 najpóźniej w marcu 2020 roku. (Przed pojawieniem się .NET 5.0 okres obsługi wydań Current wynosił tylko 3 miesiące.)

Jeżeli potrzebujesz dłuższej obsługi wydań, to dzisiaj możesz wybrać .NET 6.0 i korzystać z niego aż do pojawienia się .NET 8.0, nawet jeśli w międzyczasie Microsoft udostępni wydanie .NET 7.0. Po prostu .NET 7.0 będzie wydaniem z linii Current, co oznacza, że jego obsługa zostanie zakończona przed zakończeniem okresu obsługi wersji .NET 6.0. Pamiętaj jednak, że nawet w przypadku wersji LTS musisz aktualizować środowisko wersjami poprawkowymi, takimi jak 6.0.1.

Wszystkie wersje .NET Core i nowoczesnego .NET osiągnęły już koniec swojego okresu obsługi, z wyjątkiem następujących:

- .NET 5.0 — okres obsługi zakończy się w maju 2022 roku.
- .NET Core 3.1 — okres obsługi zakończy się w grudniu 2022 roku.
- .NET 6.0 — okres obsługi zakończy się w listopadzie 2024 roku.

Rozpoznawanie wersji .NET Runtime i .NET SDK

Numeracja wersji środowiska wykonawczego .NET Runtime jest zgodna z semantycznym schematem numerów wersji, a zatem zmiana głównego numeru oznacza poważną zmianę w środowisku, zmiana pobocznego numeru oznacza dodanie nowych funkcji, a zmiana numeru poprawkowego oznacza usunięcie błędów.

Numeracja wersji .NET SDK nie zachowuje semantycznego schematu numerów wersji. Główny i podoczny numer są dopasowane do wersji .NET Runtime, natomiast numer poprawkowy nadawany jest zgodnie ze schematem oznaczającym główne i mniejsze wersje SDK.

Przykłady tej numeracji możesz zobaczyć w poniższej tabeli:

Zmiana	.NET Runtime	.NET SDK
Wydanie początkowe	6.0.0	6.0.100
Poprawki w SDK	6.0.0	6.0.101
Poprawki w Runtime i SDK	6.0.1	6.0.102
Nowa funkcja w SDK	6.0.1	6.0.200

Usuwanie starszych wersji .NET

Aktualizacje .NET Runtime są zawsze zgodne z głównymi wersjami, takimi jak 6.x, natomiast aktualizacje wydań .NET SDK zachowują możliwość obsługi aplikacji kompilowanych dla poprzednich wersji tego samego środowiska wykonawczego. Pozwala to zatem usunąć starsze wersje.

Aby sprawdzić, jakie wersje SDK i Runtime są zainstalowane w Twoim systemie, możesz posłużyć się następującymi poleceniami:

- `dotnet --list-sdks`
- `dotnet --list-runtimes`

W systemie Windows możesz skorzystać z sekcji *Aplikacje i funkcje* w oknie ustawień systemu, aby usunąć niepotrzebne wersje .NET SDK. W systemie macOS lub Windows możesz też użyć narzędzia `dotnet-core-uninstall`, które jednak nie jest instalowane domyślnie.

Na przykład podczas pisania tej książki co miesiąc korzystałem z następującego polecenia:

```
dotnet-core-uninstall remove --all-previews-but-latest --sdk
```

Co się zmienia w nowoczesnym .NET?

Nowoczesne .NET jest zdecydowanie lepiej podzielone na moduły w porównaniu do starego .NET Framework, który jest rozwiązaniem praktycznie monolitycznym. Jest projektem o otwartych źródłach i Microsoft ciągle publikuje decyzje na temat wprowadzanych zmian i usprawnień. Firma wkłada wiele wysiłków w poprawianie i usprawnianie nowoczesnego .NET.

Nowoczesne .NET jest znacznie mniejsze od aktualnej wersji .NET Framework, ponieważ zostało z niej usuniętych wiele starszych elementów oraz tych związanych z konkretną platformą. Na przykład biblioteki **Windows Forms** i **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 dla systemów macOS i Linux.

Tworzenie oprogramowania dla Windows

Jedną z nowych funkcji w .NET jest obsługa starych aplikacji Windows Forms oraz WPF, możliwa dzięki dodatkowi Windows Desktop Pack, dołączanemu do wydań .NET dla systemów Windows. Właśnie z tego powodu wydanie to jest większe od wydań dla systemów macOS i Linux. Jeżeli to konieczne, można wprowadzić drobne zmiany do starych aplikacji dla systemów Windows i skompilować je ponownie z .NET 6. Pozwala to nadal korzystać ze starszych funkcji platformy, ciesząc się jednocześnie usprawnieniami w wydajności systemu.

Tworzenie oprogramowania dla sieci WWW

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, ASP.NET Web API, SignalR i gRPC, dlatego zostały one przebudowane i połączone w jeden produkt działający w ramach .NET pod nazwą ASP.NET Core. Więcej informacji na temat tych technologii znajdziesz w rozdziale 14., „Tworzenie witryn WWW przy użyciu ASP.NET Core Razor Pages", rozdziale 15., „Tworzenie aplikacji WWW przy użyciu ASP.NET Core MVC", w rozdziale 16., „Tworzenie i używanie serwisów sieciowych", oraz w bonusowym rozdziale 18.

Więcej informacji

Niektórzy programiści używający .NET Framework są rozgoryczeni faktem, że technologie ASP.NET Web Forms, WCF i Windows Workflow (WF) nie zostały wprowadzone do nowoczesnego .NET i postulują do firmy Microsoft o zmianę tej decyzji. Istnieją też projekty o otwartych źródłach, które próbują przenieść WCF i WF do nowoczesnego .NET. Pod tym adresem dowiesz się więcej na ich temat: *https://devblogs.microsoft.com/dotnet/supporting-the-community-with-wf-and-wcf-oss-projects/*. Istnieje też otwartoźródłowy projekt emulujący komponenty Web Forms w technologii Blazor. Jest on dostępny pod adresem *https://github.com/FritzAndFriends/BlazorWebFormsComponents*.

Praca z bazami danych

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, uzupełniona o obsługę nierelacyjnych baz danych, takich jak Microsoft Azure Cosmos DB, i otrzymała nazwę Entity Framework Core. Więcej informacji na ten temat znajdziesz w rozdziale 10., „Praca z bazami danych przy użyciu Entity Framework Core".

Jeżeli masz istniejącą aplikację, która używa starszych wersji Entity Framework, to pamiętaj, że .NET Core 3.0 i nowsze wersje nadal pozwalają na używanie Entity Framework w wersji 6.3.

Motywy kolorystyczne w nowoczesnym .NET

Firma Microsoft przygotowała w technologii Blazor stronę WWW, która prezentuje najważniejsze motywy kolorystyczne nowoczesnego .NET: *https://themes.net/*.

Poznawanie .NET Standard

W 2019 r. platforma .NET pozostaje podzielona na trzy osobne platformy, a każda z nich jest kontrolowana przez Microsoft:

- **.NET Core** przeznaczona dla nowych, wieloplatformowych aplikacji.
- **.NET Framework** dla starszych aplikacji.
- **Xamarin** dla aplikacji mobilnych.

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, która jest zbiorem różnych API, jakie musi implementować każda platforma .NET, wykazując tym samym poziom zgodności ze standardem. Na przykład dana platforma może oferować jedynie podstawową zgodność, ograniczając się do implementowania .NET Standard 1.4.

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 i nowszych 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.

Trzeba tu zaznaczyć, że .NET Standard jest tylko definicją standardu. Nie da się nigdzie zainstalować .NET Standard, podobnie jak nie da się zainstalować HTML5. Aby skorzystać z HTML5, musisz zainstalować przeglądarkę, która będzie implementowała specyfikację HTML5.

Podobnie, aby użyć .NET Standard, musisz zainstalować platformę .NET, która będzie implementowała specyfikację .NET Standard. Najnowsza specyfikacja .NET Standard 2.1 została zaimplementowana jedynie w .NET Core 3.0, Mono i Xamarin. Niektóre funkcje języka C# 8.0 wymagają implementacji .NET Standard 2.1, a ta specyfikacja nie została zaimplementowana w .NET Framework 4.8. W związku z tym musimy traktować platformę .NET Framework jako element przestarzały.

Wraz z pojawieniem się wersji .NET 6 (w listopadzie 2021 roku) znaczenie i potrzeba istnienia .NET Standard bardzo się zmniejszyły, ponieważ teraz istnieje już tylko jedna platforma .NET obejmująca również rozwiązania mobilne. W skład .NET 6 wchodzą jedna biblioteka klas (BCL) oraz dwa środowiska uruchomieniowe (CLR). Środowisko CoreCLR zostało zoptymalizowane do pracy na serwerach i komputerach stacjonarnych, czyli do uruchamiania witryn WWW i aplikacji dla komputerów, natomiast środowisko Mono zostało zoptymalizowane do pracy w przeglądarkach i na urządzeniach mobilnych o ograniczonych zasobach.

To wszystko nie sprawia, że możemy zapomnieć o istnieniu aplikacji i witryn działających na platformie .NET Framework, a zatem i one będą musiały być aktualizowane. To właśnie dlatego tak ważna jest umiejętność tworzenia bibliotek klas .NET Standard 2.0, które będą zgodne wstecznie ze starszymi platformami .NET.

Platformy .NET i narzędzia używane w tym wydaniu

W pierwszym wydaniu tej książki, które powstało w marcu 2016 r., koncentrowałem się na .NET Core, ale używałem też .NET Framework w przypadku, gdy ważne lub przydatne funkcje nie zostały jeszcze zaimplementowane w .NET Core, ponieważ dostępne było jedynie wydanie .NET Core 1.0. 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, tak żeby czytelnicy mogli się skoncentrować na przykładach związanych z .NET Core, czyli środowiskiem rzeczywiście wieloplatformowym.

Trzecie wydanie jest dopełnieniem całości. Większość książki została napisana od nowa, dzięki czemu całość kodu jest przystosowana do działania w czystym .NET Core. Podawanie dokładnych instrukcji dla wielu różnych narzędzi wprowadzało do książki niepotrzebną złożoność.

W czwartym wydaniu kontynuuję ten trend, prezentując przykładowe kody wyłącznie w Visual Studio Code. Jedynymi wyjątkami są dwa ostatnie rozdziały. W rozdziale 20., skorzystamy z Visual Studio działającego w systemie Windows 10. Z kolei w rozdziale 21., będziemy używać Visual Studio for Mac.

W szóstym wydaniu bonusowy rozdział 20. został zaktualizowany, aby zaprezentować sposób tworzenia mobilnych i stacjonarnych aplikacji za pomocą Visual Studio 2022 oraz biblioteki **.NET MAUI** (**Multi-platform App UI**).

Do czasu pojawienia się kolejnego wydania książki, opisującego już .NET 7, Visual Studio Code uzyska rozszerzenie umożliwiające mu obsługę .NET MAUI. Oznacza to, że czytelnicy będą mogli wykonać wszystkie zadania z tej książki wyłącznie za pomocą Visual Studio Code.

Poznawanie języka IL

Kompilator języka C# (nazywa się **Roslyn**) używane przez program `dotnet` (działający w wierszu poleceń) zmienia kod źródłowy języka C# w kod języka **IL** (**intermediate language** — język pośredni), a następnie zapisuje go w pliku zestawu (są to pliki DLL lub EXE). Polecenia języka IL przypominają instrukcje języka assembler, które jednak wykonywane są w maszynie wirtualnej .NET o nazwie 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, czego przykładem może być rozszerzenie dekompilujące ILSpy.

Porównanie technologii .NET

Poniższa tabela podsumowuje informacje o poszczególnych technologiach .NET:

Technologia	Opis	Systemy operacyjne
Nowoczesne .NET	Zbiór nowoczesnych funkcji, pełna obsługa języka C# 8, 9 i 10, pozwala na przenoszenie istniejących lub tworzenie nowych aplikacji i usług dla sieci WWW i systemów Windows	Windows, macOS, Linux, Android, iOS
.NET Framework	Zbiór starszych funkcji, ograniczona obsługa języka C# 8.0, konserwacja istniejących aplikacji	Tylko Windows
Xamarin	Aplikacje mobilne i dla komputerów	iOS, Android, macOS

Tworzenie aplikacji konsoli za pomocą Visual Studio 2022

Zadaniem tego podrozdziału jest zaprezentowanie procedury tworzenia aplikacji konsolowej za pomocą Visual Studio 2022.

Jeżeli nie masz komputera z systemem Windows albo chcesz korzystać z Visual Studio Code, możesz pominąć ten podrozdział, ponieważ ostateczny kod będzie ten sam.

Zarządzanie wieloma projektami w Visual Studio 2022

W Visual Studio 2022 istnieje koncepcja rozwiązania (ang. *solution*), które umożliwia jednoczesne otwieranie wielu projektów i zarządzanie nimi. Wykorzystamy tutaj rozwiązanie, aby zachowywać w nim dwa projekty tworzone w tym rozdziale.

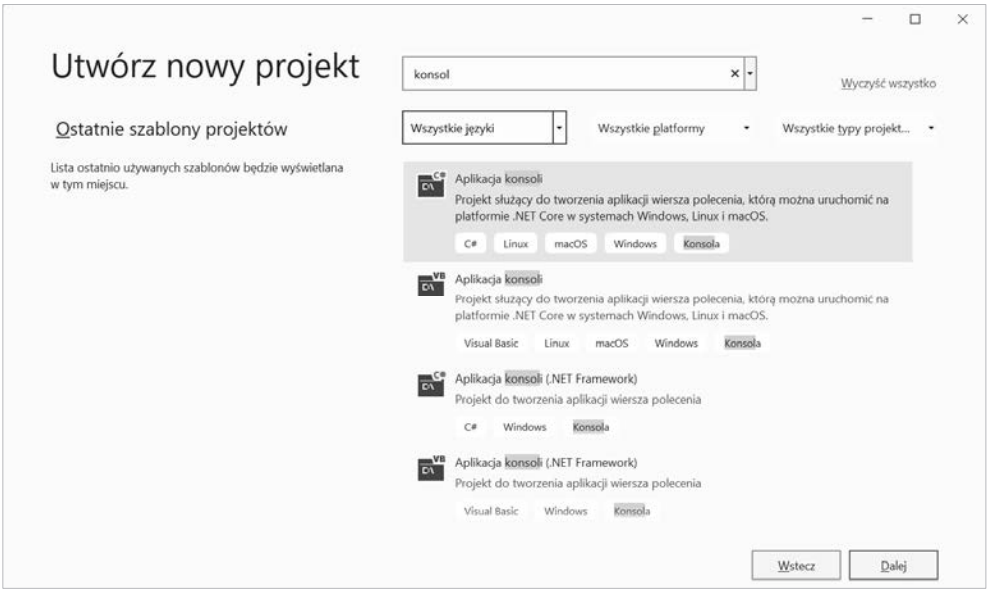
Pisanie kodu za pomocą Visual Studio 2022

Zaczynamy już pisanie kodu!

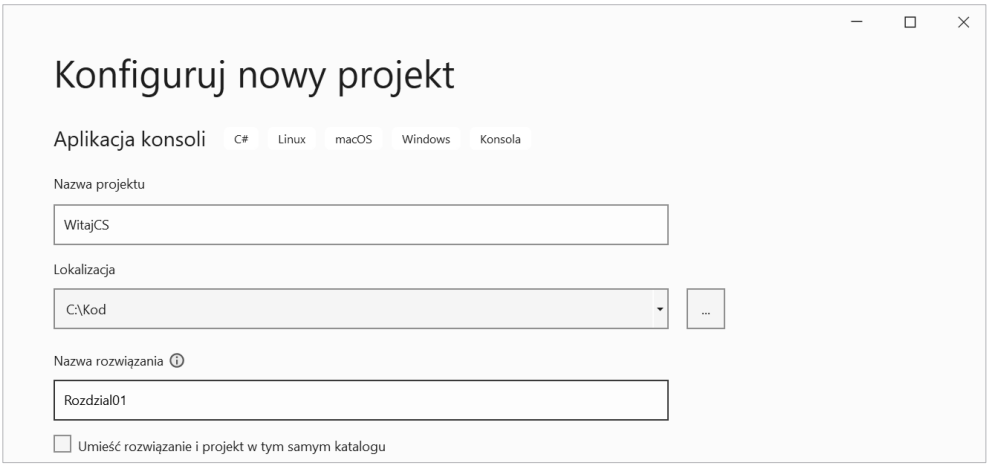
1. Uruchom Visual Studio 2022.
2. W oknie startowym kliknij przycisk *Utwórz nowy projekt*.
3. W oknie dialogowym *Utwórz nowy projekt*, w polu *Wyszukaj szablony*, wpisz słowo `konsol`, a z listy poniżej wybierz pozycję *Aplikacja konsoli*. Ważne jest, aby wybrać szablon dla języka C#, tak jak na rysunku 1.3, a nie dla innych języków, takich jak F# lub Visual Basic.
4. Kliknij przycisk *Dalej*.
5. W oknie dialogowym *Konfiguruj nowy projekt* wpisz nazwę projektu `WitajCS`, jako lokalizację podaj katalog `C:\Kod`, a w polu *Nazwa rozwiązania* wpisz nazwę `Rozdzial01`, tak jak na rysunku 1.4.
6. Kliknij przycisk *Dalej*.

> **Więcej informacji**
>
> Celowo wykorzystamy tutaj starszy szablon projektu pochodzący z .NET 5.0, aby zobaczyć pełen kod aplikacji konsoli. W kolejnym podrozdziale przygotujemy aplikację konsoli, wykorzystując do tego .NET 6.0, a wtedy zobaczysz, co się zmieniło.

7. W oknie dialogowym *Informacje dodatkowe* zwróć uwagę na to, że na liście rozwijanej *Platforma* dostępne są warianty .NET w wersji *Bieżąca* i *Długoterminowe wsparcie*. Wybierz z tej listy pozycję *.NET 5.0 (Bieżąca)* i kliknij przycisk *Utwórz*.

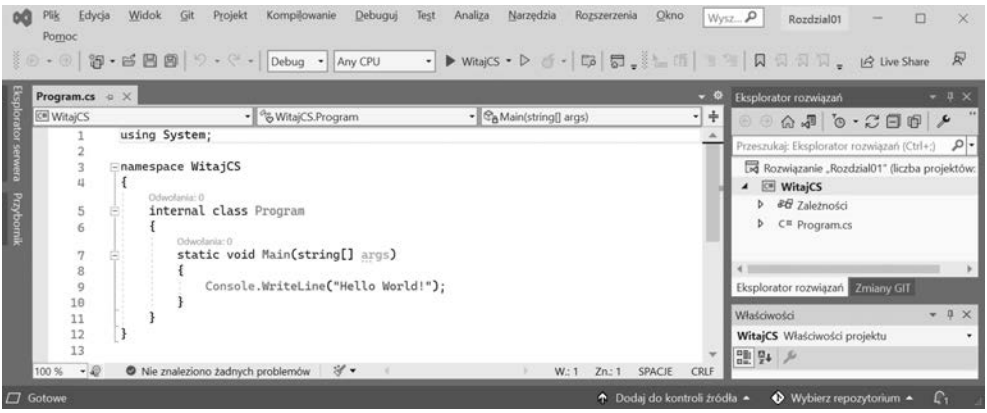


Rysunek 1.3. Wybieranie szablonu projektu aplikacji konsoli



Rysunek 1.4. Konfigurowanie nazwy i lokalizacji nowego projektu

8. W okienku *Eksplorator rozwiązań* kliknij dwukrotnie plik o nazwie *Program.cs*. Zwróć przy tym uwagę na to, że w *Eksploratorze rozwiązań* wyświetlany jest cały projekt *WitajCS*, tak jak na rysunku 1.5.
9. W pliku *Program.cs* zmień wiersz numer 9, tak żeby w konsoli wypisywany był tekst `Witaj, C#!`.

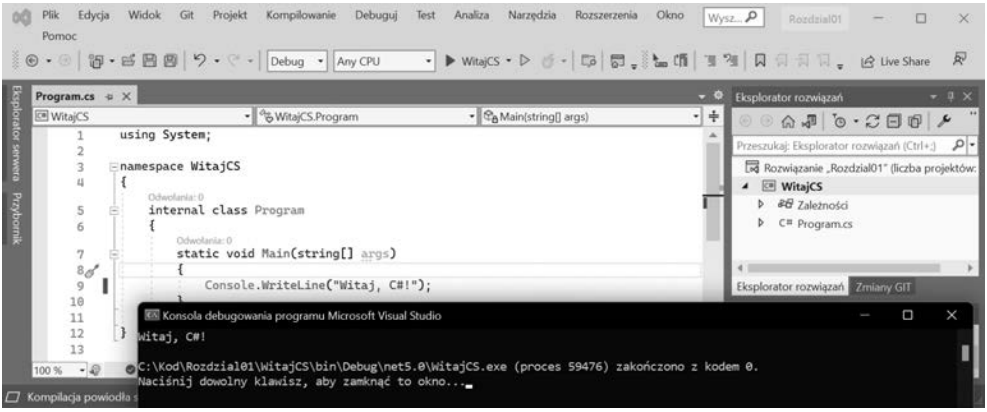


Rysunek 1.5. Edytowanie pliku Program.cs w Visual Studio 2022

Kompilowanie i uruchamianie kodu w Visual Studio

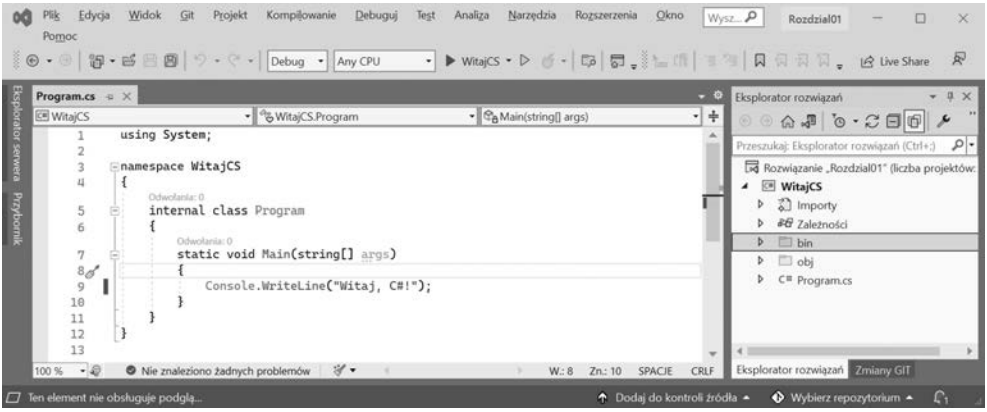
Następne zadanie polega na skompilowaniu i uruchomieniu kodu.

1. W Visual Studio wybierz z menu pozycję *Debuguj/Uruchom bez debugowania*.
2. Wynik działania programu zobaczysz w oknie konsoli, tak jak na rysunku 1.6.



Rysunek 1.6. Uruchomienie aplikacji konsoli w systemie Windows

3. Naciśnij dowolny klawisz, aby zamknąć okno konsoli i wrócić do Visual Studio.
4. W okienku *Eksplorator rozwiązań* kliknij projekt *WitajCS*, a następnie na pasku narzędzi okienka kliknij przycisk *Pokaż wszystkie pliki*. Zauważ, że teraz widoczne są też wygenerowane przez kompilator foldery *bin* i *obj*, takie jak na rysunku 1.7.



Rysunek 1.7. Wyświetlanie folderów i plików generowanych przez kompilator

Czym są foldery i pliki generowane przez kompilator?

Utworzone zostały dwa wygenerowane przez kompilator foldery o nazwach *bin* i *obj*. Nie musisz jeszcze zaglądać do tych folderów ani przeglądać zapisanych w nich plików. Musisz jednak wiedzieć, że kompilator w trakcie swojej pracy tworzy foldery i pliki tymczasowe. Możesz je w dowolnym momencie usunąć, a później zostaną one ponownie odtworzone. Programiści często w ten sposób „oczyszczają" projekt. W Visual Studio istnieje nawet pozycja menu *Kompilowanie/Wyczyść rozwiązanie*, która usuwa wszystkie tymczasowe pliki. Jest ona równoznaczna z wydaniem w Visual Studio Code polecenia `dotnet clean`.

- Folder *obj* zawiera po jednym skompilowanym pliku obiektu dla każdego pliku z kodem źródłowym. Takie obiekty nie zostały jeszcze połączone ze sobą w ramach pliku wykonywalnego.
- Folder *bin* zawiera po jednym binarnym pliku wykonywalnym dla każdej aplikacji lub biblioteki klas. Więcej szczegółów na ten temat znajdziesz w rozdziale 7., „Poznawanie typów .NET".

Pisanie programów najwyższego poziomu

Możesz sobie pomyśleć, że wypisanie prostego tekstu `Witaj, C#!` wymaga wpisania sporej ilości kodu.

Co prawda podstawowy kod został już wprowadzony przez szablon projektu, ale może istnieje jakiś prostszy sposób?

Okazuje się, że w C# 9 i nowszych wersjach języka istnieje coś, co jest nazywane **programem najwyższego poziomu** (ang. *top-level program*).

Porównajmy teraz aplikację konsoli przygotowaną przez szablon projektu:

```
using System;

namespace WitajCS
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

I kod programu najwyższego poziomu dla minimalnej aplikacji konsoli:

```
using System;

Console.WriteLine("Hello World!");
```

O wiele prostsze, prawda? Jeżeli musielibyśmy zacząć z pustym plikiem i samodzielnie wpisywać te wszystkie instrukcje, to ta wersja byłaby znacznie lepsza. Jak to jednak w ogóle działa?

Podczas kompilacji podstawowy kod definiujący przestrzeń nazw, klasę `Program` i metodę `Main` jest automatycznie generowany i umieszczany wokół wpisanych przez nas instrukcji.

Trzeba tylko zapamiętać dwie reguły dotyczące programów najwyższego poziomu:

- Ewentualne instrukcje `using` muszą znaleźć się na początku pliku.
- W projekcie może istnieć tylko jeden plik tego typu.

Zapisana na początku pliku instrukcja `using System;` nakazuje zaimportować przestrzeń nazw `System`. Dzięki temu możemy później skorzystać z instrukcji `Console.WriteLine`. Więcej informacji na temat przestrzeni nazw znajdziesz w następnym rozdziale.

Dodawanie drugiego projektu w Visual Studio 2022

Dodajmy teraz drugi projekt do naszego rozwiązania, aby przyjrzeć się programowi najwyższego poziomu.

1. W Visual Studio wybierz z menu pozycję *Plik/Dodaj/Nowy projekt…*.
2. W oknie dialogowym *Dodawanie nowego projektu*, na liście *Ostatnie szablony projektów*, wybierz pozycję *Aplikacja konsoli [C#]* i kliknij przycisk *Dalej*.
3. W oknie dialogowym *Konfiguruj nowy projekt*, w polu *Nazwa projektu*, wpisz `ProgramNajwyzszegoPoziomu`, a w polu *Lokalizacja* pozostaw wartość `C:\Kod\Rozdzial01` i kliknij przycisk *Dalej*.

4. W oknie dialogowym *Informacje dodatkowe* wybierz z listy rozwijanej pozycję *.NET 6.0 (Długoterminowe wsparcie)* i kliknij przycisk *Utwórz*.
5. W okienku *Eksplorator rozwiązań*, w projekcie *ProgramNajwyzszegoPoziomu*, kliknij dwukrotnie plik *Program.cs*, aby go otworzyć.
6. Zwróć uwagę na to, że teraz w pliku *Program.cs* znajduje się tylko wiersz komentarza oraz jedna instrukcja (co widać w poniższym kodzie), ponieważ wykorzystuje on funkcję programu najwyższego poziomu wprowadzoną w C# 9.

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

Wcześniej, gdy omawiałem koncepcję programu najwyższego poziomu, potrzebowaliśmy jeszcze instrukcji `using System;`. Dlaczego tutaj jej nie ma?

Niejawne importowanie przestrzeni nazw

Okazuje się, że co prawda nadal musimy zaimportować przestrzeń nazw *System*, ale tym razem robi to za nas nowa funkcja wprowadzona w C# 10. Sprawdźmy, jak to działa.

1. W okienku *Eksplorator rozwiązań* wybierz projekt *ProgramNajwyzszegoPoziomu* i kliknij przycisk *Pokaż wszystkie pliki*. Zwróć uwagę, że tutaj również pojawiają się foldery *bin* i *obj*.
2. Rozwiń folder *obj*, w nim rozwiń folder *Debug*, a potem jeszcze folder *net6.0* i otwórz plik o nazwie *ProgramNajwyzszegoPoziomu.GlobalUsings.g.cs*.
3. Zauważ, że ten plik jest automatycznie generowany przez kompilator na potrzeby projektów korzystających z .NET 6 i używa funkcji nazywanej **importem globalnym** wprowadzonej w C# 10. Pozwala ona globalnie importować powszechnie używane przestrzenie nazw, takie jak *System*, do wszystkich plików z kodem. Oto zawartość tego pliku:

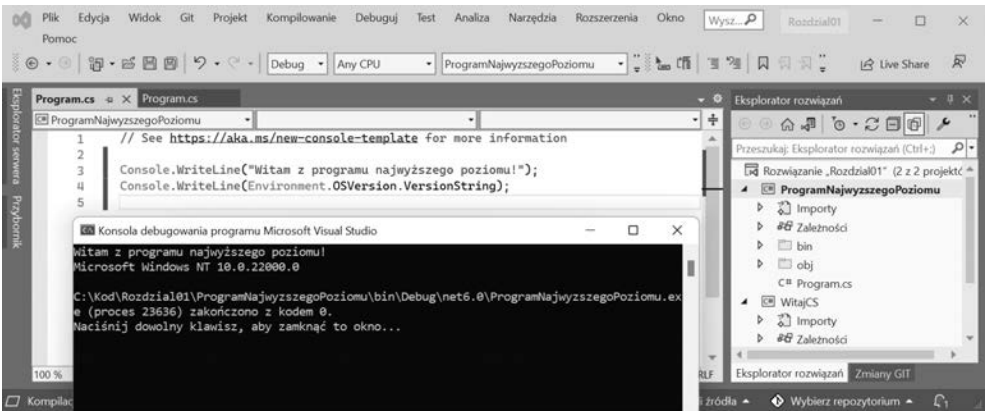
```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

> Więcej informacji na temat tej funkcji podam w następnym rozdziale. Na razie zapamiętaj, że poważną różnicą między .NET 5 i .NET 6 jest to, że wiele szablonów projektów, takich jak projekt aplikacji konsoli, wykorzystuje nowe funkcje języka, aby ukrywać powtarzalne części kodu.

4. W projekcie *ProgramNajwyzszegoPoziomu*, w pliku *Program.cs*, zmień instrukcje tak, żeby wypisywały inny komunikat oraz numer wersji systemu operacyjnego:

```
Console.WriteLine("Witam z programu najwyższego poziomu!");
Console.WriteLine(Environment.OSVersion.VersionString);
```

5. W okienku *Eksplorator rozwiązań* kliknij prawym przyciskiem myszy rozwiązanie *Rozdzial01* i z menu kontekstowego wybierz pozycję *Ustaw projekty startowe…*, a w otwartym oknie dialogowym wybierz opcję *Bieżące zaznaczenie* i kliknij przycisk *OK*.
6. W okienku *Eksplorator rozwiązań* kliknij projekt *ProgramNajwyzszegoPoziomu* (albo jego dowolny folder lub plik) i zauważ, że Visual Studio pogrubia nazwę tego projektu, informując, że jest on teraz projektem startowym.
7. Wybierz z menu pozycję *Debuguj/Uruchom bez debugowania*, aby uruchomić projekt *ProgramNajwyzszegoPoziomu*. Zwróć uwagę na podawane przez niego wyniki, przedstawione na rysunku 1.8.



Rysunek 1.8. Uruchomienie programu najwyższego poziomu w rozwiązaniu składającym się z dwóch projektów w Visual Studio dla Windows

Tworzenie aplikacji konsoli za pomocą Visual Studio Code

Zadaniem tego podrozdziału jest zaprezentowanie procedury tworzenia aplikacji konsoli za pomocą Visual Studio Code.

Jeżeli nie chcesz korzystać z Visual Studio Code ani z interaktywnych notatników .NET, to możesz pominąć ten podrozdział i od razu przejść do podrozdziału „Przeglądanie folderów i plików projektów".

Wszystkie instrukcje i zrzuty ekranu pochodzą z systemu Windows, ale te same działania sprawdzą się w Visual Studio Code działającym w systemach macOS lub Linux.

Główną różnicą będą działania wykonywane w wierszu poleceń, takie jak usuwanie plików. Zarówno polecenia, jak i ścieżki będą zapewne wyglądały inaczej w każdym z systemów:

Windows, macOS oraz Linux. Na szczęście polecenie `dotnet` działa tak samo we wszystkich systemach.

Zarządzanie wieloma projektami w Visual Studio Code

Visual Studio Code wykorzystuje koncepcję **przestrzeni roboczej** (ang. *workspace*), która umożliwia otwieranie wielu projektów jednocześnie i zarządzanie nimi. W tym podrozdziale wykorzystamy przestrzeń roboczą, w której utworzymy dwa projekty.

Pisanie kodu za pomocą Visual Studio Code

Zacznijmy zatem od pisania kodu!

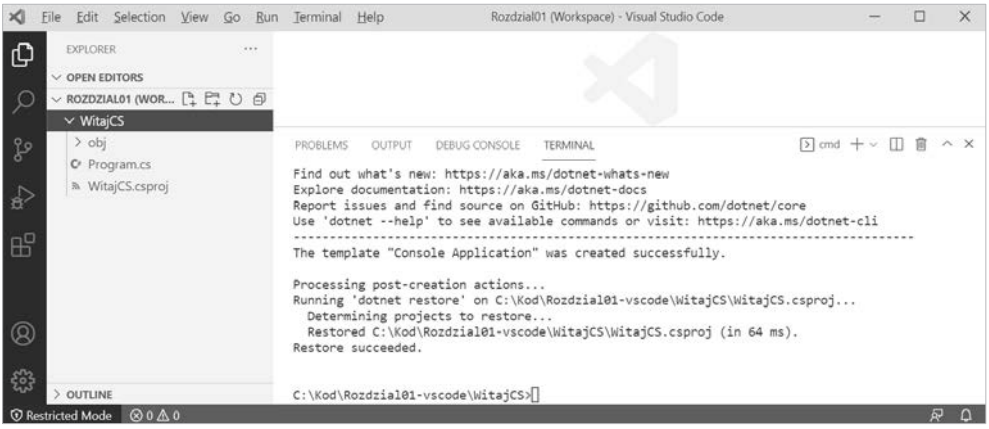
1. Uruchom Visual Studio Code.
2. Upewnij się, że nie masz otwartych żadnych plików, folderów ani przestrzeni roboczych.
3. Wybierz z menu pozycję *File/Save Workspace As…*.
4. W okienku dialogowym przejdź do swojego folderu użytkownika w systemie macOS (mój nazywa się *markjprice*) lub do folderu *Dokumenty* w systemie Windows albo do dowolnego innego folderu, w którym chcesz zapisać swoje projekty.
5. Kliknij przycisk tworzenia nowego folderu i nadaj folderowi nazwę *Kod*. (Jeżeli to samo ćwiczenie było już robione z Visual Studio 2022, to ten folder powinien już istnieć).
6. W folderze *Kod* utwórz nowy folder, o nazwie *Rozdzial01-vscode*.
7. W folderze *Rozdzial01-vscode* zapisz przestrzeń roboczą w pliku *Rozdzial01.code-workspace*.
8. Wybierz z menu pozycję *File/Add Folder to Workspace…* albo kliknij przycisk *Add Folder*.
9. W folderze *Rozdzial01-vscode* utwórz nowy folder, o nazwie *WitajCS*.
10. Zaznacz folder *WitajCS* i kliknij przycisk *Add*.
11. Wybierz z menu pozycję *View/Terminal.*

> **Więcej informacji**
>
> Celowo wykorzystamy tutaj starszy szablon projektu pochodzący z .NET 5.0, aby zobaczyć pełen kod aplikacji konsoli. W kolejnym podrozdziale przygotujemy aplikację konsoli, wykorzystując do tego .NET 6.0, a wtedy zobaczysz, co się zmieniło.

12. W panelu *TERMINAL* sprawdź, czy aktywnym folderem jest folder *WitajCS*, a następnie użyj polecenia `dotnet`, aby utworzyć nową aplikację konsoli używającą platformy .NET 5.0, stosując do tego poniższe polecenie:

```
dotnet new console -f net5.0
```

13. Zauważysz, że działające w wierszu poleceń polecenie `dotnet` utworzy w aktualnym folderze nowy projekt aplikacji konsoli, a w panelu *EXPLORER* pojawią się dwa nowo utworzone pliki: *WitajCS.csproj* i *Program.cs*, tak jak na rysunku 1.9.



Rysunek 1.9. W panelu EXPLORER widoczne są dwa nowo utworzone pliki i jeden folder

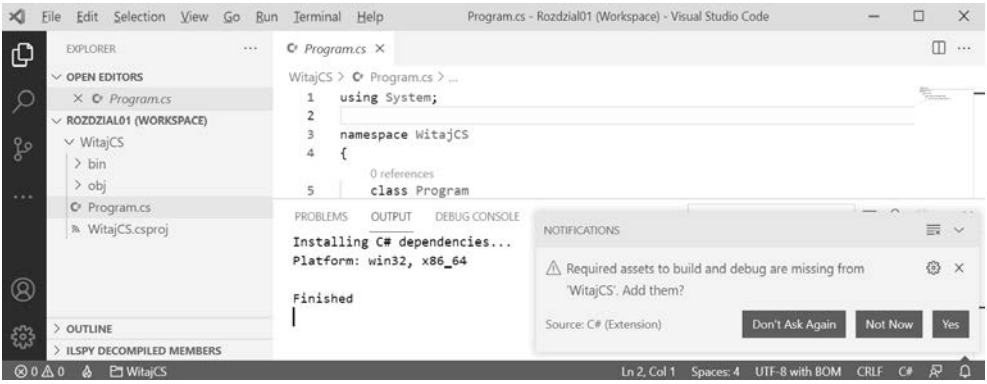
14. W panelu *EXPLORER* kliknij plik *Program.cs*, aby go otworzyć w edytorze. Za pierwszym razem Visual Studio Code może być zmuszone do pobrania i zainstalowania zależności związanych z językiem C#, takich jak pakiety OmniSharp, Razor Language Server oraz debugger .NET Core, o ile nie zostały one zainstalowane razem z rozszerzeniem dla języka C#. Visual Studio Code będzie pokazywać postępy instalowania tych elementów w panelu *OUTPUT*. Po zakończeniu instalowania pojawi się w nim komunikat `Finished`., tak jak na poniższym wydruku:

```
Installing C# dependencies...
Platform: win32, x86_64

Downloading package 'OmniSharp for Windows (.NET 4.6 / x64)' (36150
KB)..................... Done!
Validating download...
Integrity Check succeeded.
Installing package 'OmniSharp for Windows (.NET 4.6 / x64)'

Downloading package '.NET Core Debugger (Windows / x64)' (45048
KB)..................... Done!
Validating download...
Integrity Check succeeded.
Installing package '.NET Core Debugger (Windows / x64)'

Downloading package 'Razor Language Server (Windows / x64)' (52344
KB)..................... Done!
Installing package 'Razor Language Server (Windows / x64)'

Finished
```

Więcej informacji

Podany wyżej wydruk pochodzi z Visual Studio Code dla systemu Windows. W systemie macOS lub Linux ten wydruk będzie wyglądał nieco inaczej, ponieważ pobierane i instalowane będą komponenty właściwe dla tych systemów operacyjnych.

15. Kompilator utworzy foldery o nazwach *bin* i *obj*. Jeżeli zobaczysz komunikat mówiący o braku wymaganych elementów, kliknij przycisk *Yes*, tak jak na rysunku 1.10.



Rysunek 1.10. Komunikat o konieczności dodania elementów wymaganych do kompilowania i debugowania

16. Jeżeli ten komunikat zniknie, zanim zdążysz na niego zareagować, to możesz kliknąć ikonę dzwoneczka w prawym dolnym rogu, aby wyświetlić go ponownie.
17. Po kilku sekundach w panelu *EXPLORER* pojawi się folder o nazwie *.vscode*. Jak się przekonasz w rozdziale 4., „Pisanie, debugowanie i testowanie funkcji", zawartość tego folderu jest używana podczas debugowania.
18. W pliku *Program.cs* zmień 9. wiersz kodu, tak żeby w konsoli wypisywany był tekst „Witaj, C#!".

Dobra praktyka

Wybierz z menu pozycję *File/Auto Save*. Dzięki temu nie trzeba będzie pamiętać o zapisywaniu zmian w plikach przed każdym kompilowaniem aplikacji.

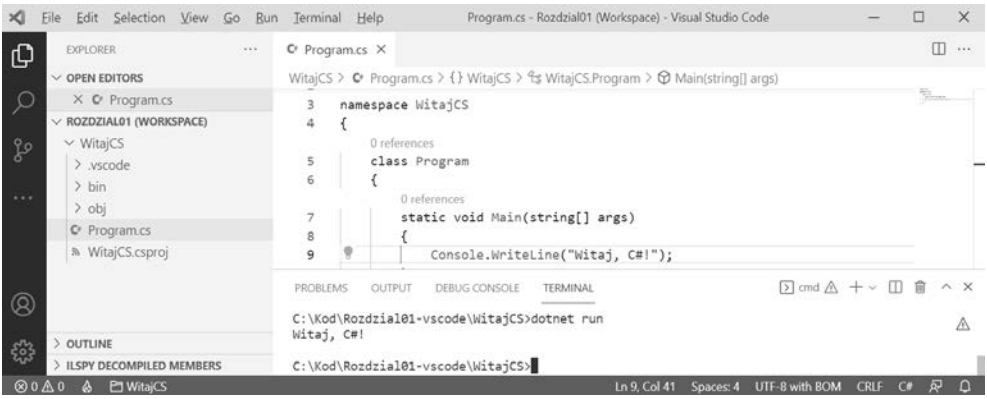
Kompilowanie i uruchamianie kodu za pomocą polecenia dotnet

Następnym zadaniem jest skompilowanie i uruchomienie kodu.

1. Wybierz z menu pozycję *View/Terminal*, a następnie wprowadź poniższe polecenie:

```
dotnet run
```

2. W panelu *TERMINAL* pojawią się wyniki pracy naszej aplikacji, takie jak na rysunku 1.11.



Rysunek 1.11. Wynik uruchomienia Twojej pierwszej aplikacji konsoli

Dodawanie drugiego projektu w Visual Studio Code

Dodajmy teraz drugi projekt do naszej przestrzeni roboczej, aby sprawdzić, jak wyglądają programy najwyższego poziomu.

1. W Visual Studio Code wybierz z menu pozycję *File/Add Folder to Workspace…*.
2. W folderze *Rozdzial01-vscode* utwórz nowy folder o nazwie *ProgramNajwyzszegoPoziomu*, zaznacz go i kliknij przycisk *Add*.
3. Wybierz z menu pozycję *Terminal/New Terminal*, a z rozwiniętej listy wybierz pozycję *ProgramNajwyzszegoPoziomu*. Możesz też w panelu EXPLORER kliknąć prawym przyciskiem myszy folder *ProgramNajwyzszegoPoziomu* i z menu kontekstowego wybrać pozycję *Open in Integrated Terminal*.
4. W panelu *TERMINAL* sprawdź, czy aktywny jest folder *ProgramNajwyzszegoPoziomu*, i wpisz polecenie tworzące nową aplikację konsoli:

```
dotnet new console
```

> **Dobra praktyka**
>
> Podczas używania przestrzeni roboczych musisz uważać przy wprowadzaniu poleceń w okienku *TERMINAL*. Przed wprowadzeniem potencjalnie niszczycielskich poleceń upewnij się, że aktywny jest właściwy folder! To właśnie dlatego nakazałem utworzyć nowy terminal dla folderu *ProgramNajwyzszegoPoziomu* jeszcze przed wprowadzeniem polecenia tworzącego nową aplikację konsoli.

5. Wybierz z menu pozycję *View/Command Palette*.
6. Wpisz słowo `omni`, a z rozwiniętej listy wybierz pozycję *OmniSharp: Select Project*.

7. Na liście rozwijanej znajdują się dwa projekty. Wybierz projekt *ProgramNajwyzszegoPoziomu*, a gdy pojawi się okienko dialogowe, kliknij przycisk *Yes*, aby dodać wymagane elementy do debugowania.

> **Dobra praktyka**
>
> Aby włączyć debugowanie i inne przydatne funkcje, takie jak formatowanie kodu oraz przechodzenie do definicji (ang. *Go to Definition*), musisz poinformować rozszerzenie OmniSharp, który projekt jest aktywnie rozwijany w Visual Studio Code. Możesz też szybko przełączać się między projektami, klikając nazwę projektu (lub folderu) na pasku statusu obok ikony płomienia.

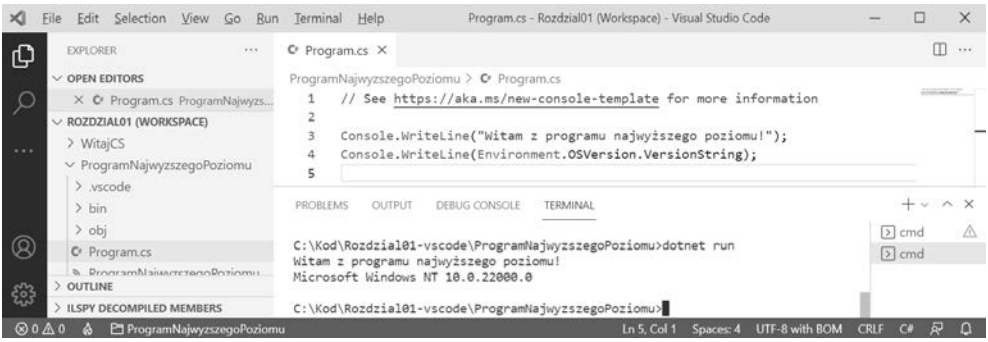
8. W panelu *EXPLORER*, w folderze *ProgramNajwyzszegoPoziomu*, zaznacz plik *Program.cs*, a następnie zmień widoczne w nim instrukcje tak, żeby wypisywały inny komunikat oraz podawały numer wersji systemu operacyjnego:

```
Console.WriteLine("Witam z programu najwyższego poziomu!");
Console.WriteLine(Environment.OSVersion.VersionString);
```

9. W panelu TERMINAL wprowadź widoczne poniżej polecenie uruchamiające program:

```
dotnet run
```

10. Zwróć uwagę na to, że wynik działania programu pojawia się w panelu *TERMINAL*, co widać też na rysunku 1.12.



Rysunek 1.12. Uruchomienie programu najwyższego poziomu w Visual Studio Code z przestrzenią roboczą zawierającą dwa projekty w systemie Windows

Jeżeli spróbujesz uruchomić ten program w systemie macOS Big Sur, to środowisko systemu operacyjnego będzie inne, co widać na poniższym wydruku:

```
Witam z programu najwyższego poziomu!
Unix 11.2.3
```

Zarządzanie wieloma plikami za pomocą Visual Studio Code

Jeżeli musisz pracować jednocześnie z wieloma plikami, to możesz umieścić je obok siebie w oknie edytora.

1. W panelu *EXPLORER* rozwiń oba projekty.
2. Otwórz pliki *Program.cs* pochodzące z dwóch projektów.
3. W oknie edytora kliknij, przytrzymaj i przeciągnij kartę jednego z plików, a potem ułóż ją tak, żeby oba pliki były widoczne jednocześnie.

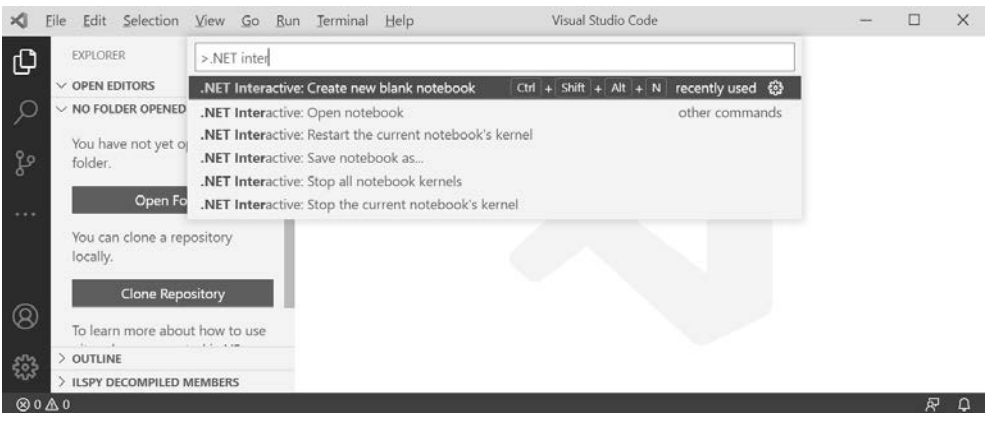
Badanie kodu w interaktywnych notatnikach .NET

Rozszerzenie .NET Interactive Notebooks ułatwia pisanie kodu jeszcze bardziej niż programy najwyższego poziomu. Do działania wymaga ono Visual Studio Code, a zatem jeżeli nie masz jeszcze zainstalowanego tego dodatku, to zainstaluj go teraz.

Tworzenie notatnika

Najpierw musimy utworzyć nowy notatnik.

1. W Visual Studio Code zamknij wszystkie otwarte przestrzenie robocze lub foldery.
2. Wybierz z menu pozycję *View/Command Palette*.
3. Wpisz słowa `.net inter`, a z listy wybierz pozycję *.NET Interactive: Create new blank notebook*, tak jak na rysunku 1.13.



Rysunek 1.13. Tworzenie nowego, pustego notatnika .NET

4. Gdy pojawi się lista dostępnych rozszerzeń, wybierz pozycję *Create as '.dib'*.

> **Więcej informacji**
>
> *.dib* to eksperymentalny format pliku zdefiniowany przez firmę Microsoft, który ma odróżniać interaktywne notatniki .NET od formatu *.ipynb* używanego przez interaktywne notatniki Pythona. To ostatnie rozszerzenie historycznie stosowane jest dla notatników Jupyter, które mogą przechowywać interaktywną (I) mieszankę danych, kodu Pythona (PY) oraz wyników, tworząc jeden plik notatnika (NB). W interaktywnych notatnikach .NET ta koncepcja została rozbudowana i pozwala tworzyć mieszankę kodu w językach C#, F#, SQL, HTML, JavaScript, składni Markdown oraz innych języków. Format *.dib* można uznać za poliglotę, ponieważ umożliwia mieszanie różnych języków. Możliwe jest też konwertowanie zawartości plików *.dib* i *.ipynb*.

5. Wybierz *C#* jako domyślny język używany w komórkach notatnika.
6. Jeżeli jest dostępna nowsza wersja rozszerzenia .NET Interactive, to starsza wersja zostanie usunięta, a nowsza pobrana i zainstalowana. Możesz wybrać z menu pozycję *View/Output*, a w okienku *Output* z listy rozwijanej wybrać pozycję *.NET Interactive: diagnostics*. Poczekaj cierpliwie, ponieważ może to zająć kilka minut. Nowo instalowany notatnik musi najpierw uruchomić własne środowisko .NET. Jeżeli po kilku minutach nic się nie zmieni, zamknij Visual Studio Code i uruchom je ponownie.
7. Po pobraniu i zainstalowaniu rozszerzenia .NET Interactive Notebooks w okienku OUTPUT pojawi się informacja o uruchomieniu procesu Kernel, taka jak na poniższym wydruku (w Twoim systemie numer portu i numer procesu na pewno będą inne):

```
Extension started for VS Code Stable.
...
Kernel process 12516 Port 59565 is using tunnel uri http://localhost:59565/
```

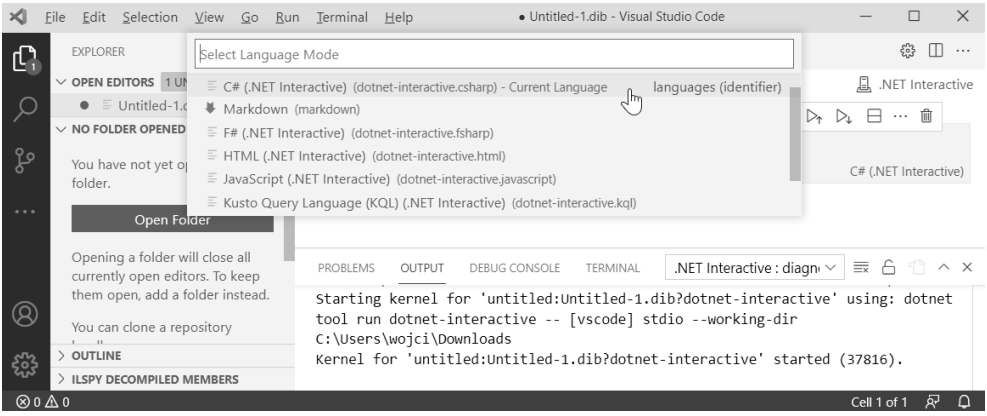
Pisanie i uruchamianie kodu w notatniku

Teraz możemy już zacząć pisać kod w komórkach notatnika.

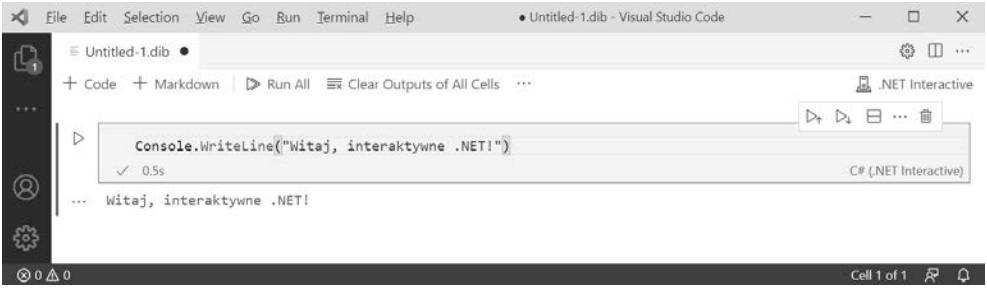
1. W pierwszej komórce powinna być już wybrana opcja *C# (.NET Interactive)*. Jeżeli aktywna jest inna opcja, to kliknij oznaczenie języka w prawym dolnym rogu komórki i wybierz z listy pozycję *C# (.NET Interactive)*, zwracając przy tym uwagę na dostępne możliwości wyboru, przedstawione na rysunku 1.14.
2. Wewnątrz komórki *C# (.NET Interactive)* wprowadź instrukcję wypisującą komunikat w konsoli (taką jak poniższa). Zauważ, że takiej instrukcji nie musisz zakończyć znakiem średnika, co jest wymagane w pełnej aplikacji.

```
Console.WriteLine("Witaj, interaktywne .NET!")
```

3. Kliknij przycisk *Execute Cell*, znajdujący się po lewej stronie komórki. Wynik działania kodu pojawi się na szarym tle poniżej tej komórki, tak jak na rysunku 1.15.



Rysunek 1.14. Zmiana języka dla komórki kodu w interaktywnym notatniku .NET



Rysunek 1.15. Uruchomienie kodu w notatniku i przeglądanie wyników jego pracy

Zapisywanie notatnika

Podobnie jak w przypadku innych plików, notatnik też dobrze jest zapisać przed przystąpieniem do dalszych działań.

1. Wybierz z menu pozycję *File/Save As…*.
2. Przejdź do folderu *Rozdzial01-vscode* i zapisz notatnik w pliku *Rozdzial01.dib*.
3. Zamknij kartę edytora z plikiem *Rozdzial01.dib*.

Dodawanie do notatnika składni Markdown i poleceń specjalnych

Istnieje możliwość zmieszania w komórkach tekstów z formatowaniem Markdown i kodem ze specjalnymi poleceniami.

1. Wybierz z menu pozycję *File/Open File…*, a następnie wybierz plik *Rozdzial01.dib*.

2. Jeżeli pojawi się okienko z pytaniem *Do you trust the authors of these files?*, odpowiedz kliknięciem przycisku *Open*, potwierdzając, że ufasz autorowi tego pliku.
3. Przesuń wskaźnik myszy na blok kodu i kliknij przycisk + *Markdown*, aby dodać komórkę z formatowaniem Markdown.
4. Wpisz nagłówek pierwszego poziomu, tak jak w poniższym tekście:

```
# Rozdział 1 – Cześć C#! Witaj, .NET!
Pisanie *formatowanego* **tekstu** i kodu to zabawa!
```

5. Kliknij parafkę w prawym górnym rogu komórki i przyjrzyj się sformatowanemu tekstowi.

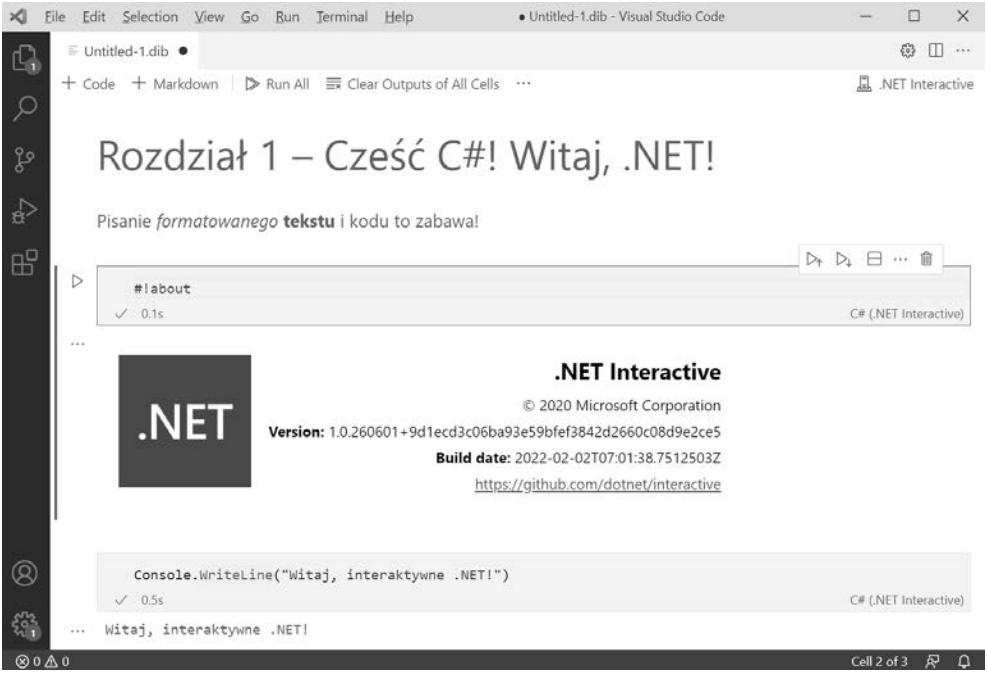
Więcej informacji

Jeżeli komórki są w złej kolejności, to możesz je odpowiednio ułożyć metodą przeciągnij i upuść.

6. Umieść wskaźnik myszy między komórką z kodem a komórką z tekstem Markdown, a następnie kliknij przycisk + *Code*.
7. Wpisz specjalne polecenie wypisujące numer wersji rozszerzenia .NET Interactive:

```
#!about
```

8. Kliknij przycisk *Execute Cell* i przyjrzyj się wynikom przedstawionym na rysunku 1.16.



Rysunek 1.16. Mieszanie formatowania Markdown, kodu i poleceń specjalnych w jednym interaktywnym notatniku .NET

Uruchamianie kodu w wielu komórkach

Gdy notatnik składa się z wielu komórek, musisz najpierw wykonać kod z poprzedzających komórek, aby ich kontekst był dostępny w komórkach następujących.

1. Na dole notatnika dodaj nową komórkę z kodem, a następnie wpisz w niej instrukcję deklarującą zmienną i przypisującą jej liczbę całkowitą:

```
int liczba = 8;
```

2. Na dole notatnika dodaj nową komórkę z kodem, a następnie wpisz w niej instrukcję wypisującą wartość zmiennej `liczba`:

```
Console.Writeline(liczba);
```

3. Zwróć uwagę na to, że druga komórka kodu nie wie nic na temat zmiennej `liczba`, ponieważ została ona zdefiniowana w innej komórce, czyli w innym kontekście, co widać też na rysunku 1.17.

Rysunek 1.17. Zmienna liczba nie istnieje w aktualnej komórce i jej kontekście

4. W pierwszej komórce kliknij przycisk *Execute Cell*, aby zadeklarować zmienną i przypisać jej wartość. Następnie w drugiej komórce również kliknij przycisk *Execute Cell*, aby wypisać wartość zmiennej `liczba`. Zauważ, że to rzeczywiście działa. (Możesz też w pierwszej komórce kliknąć przycisk *Execute Cell and Below*, aby wykonać kod w tej komórce i wszystkich komórkach poniżej).

> **Dobra praktyka**
>
> Jeżeli powiązany ze sobą kod podzielisz między dwie komórki, to pamiętaj o konieczności uruchomienia poprzedzającej komórki przed uruchomieniem następnej. Na samej górze notatnika dostępne są dwa przyciski — *Clear Output* (czyszczenie wyjść) i *Run All* (uruchom wszystkie). Są one bardzo przydatne, ponieważ kliknięcie najpierw pierwszego z nich, a potem drugiego sprawia, że wszystkie komórki z kodem zostaną poprawnie wykonane, pod warunkiem że zostały ułożone we właściwej kolejności.

Używanie interaktywnych notatników .NET do pisania kodu z tej książki

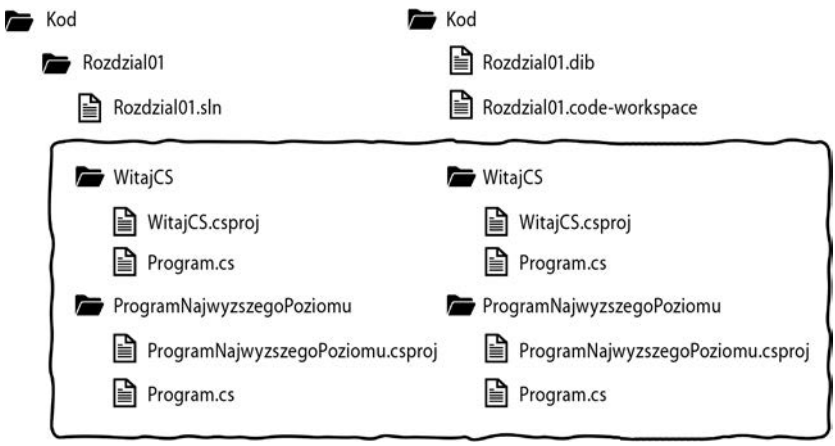
W pozostałych rozdziałach nie będę podawał instrukcji dotyczących używania notatników, ale w repozytorium GitHuba i w spolonizowanych kodach znajdują się odpowiednio przygotowane notatniki. Spodziewam się, że wielu czytelników będzie chciało skorzystać z przygotowanych notatników, aby uruchomić przykłady kodu omawiane w rozdziałach od 2. do 12. Jeżeli chcesz w ten sposób poznawać język, bez samodzielnego pisania kodu (nawet jeżeli to tylko aplikacja konsoli), skorzystaj z adresu *https://github.com/markjprice/cs10dotnet6/tree/main/notebooks* lub *https://ftp.helion.pl/przyklady/c10ne6.zip*.

Przeglądanie folderów i plików projektów

W tym rozdziale utworzyliśmy dwa projekty, a mianowicie *WitajCS* i *ProgramNajwyzszegoPoziomu*.

Visual Studio Code zarządza wieloma projektami za pomocą plików przestrzeni roboczych. Visual Studo 2022 zarządza projektami za pomocą rozwiązań. Poza tym przygotowaliśmy też interaktywny notatnik .NET.

W efekcie powstała struktura folderów z rysunku 1.18, którą będziemy powtarzać w kolejnych rozdziałach, choć tam będziemy tworzyć znacznie więcej różnych projektów.



Rysunek 1.18. Struktura folderów i plików dla dwóch projektów z tego rozdziału

Wspólne foldery i pliki

Mimo że pliki *.code-workspace* i *.sln* różnią się od siebie, to już foldery i pliki projektów takich jak *WitajCS* i *ProgramNajwyzszegoPoziomu* są identyczne w Visual Studio 2022 i Visual Studio Code. Oznacza to, że możesz wymieniać je między tymi dwoma edytorami kodu:

- W Visual Studio 2022 z otwartym rozwiązaniem wybierz z menu pozycję *Plik/Dodaj/Istniejący projekt…*, aby dodać do rozwiązania projekt utworzony w innym narzędziu.
- W Visual Studio Code z otwartą przestrzenią roboczą wybierz z menu pozycję *File/Add Folder to Workspace…*, aby dodać projekt utworzony przez inne narzędzie.

Dobra praktyka

Kod źródłowy, czyli pliki *.csproj* i *.cs*, są identyczne w obu wersjach, ale foldery *obj* i *bin* są automatycznie generowane przez kompilator, przez co mogą mieć odmienne numery wersji i powodować problemy. Jeżeli chcesz otwierać ten sam projekt zamiennie w Visual Studio 2022 i Visual Studio Code, to przed otwarciem go w edytorze usuń foldery *bin* i *obj*. To właśnie dlatego nakazałem utworzyć osobny folder dla projektów tworzonych w Visual Studio Code.

Kod w repozytorium GitHuba

Przykładowy kod z tej książki umieściłem w repozytorium GitHuba. Przygotowałem w nim osobne foldery dla Visual Studio Code i Visual Studio 2022, jak również osobne pliki interaktywnych notatników .NET. Oto dokładne umiejscowienie tych folderów:

- Rozwiązania dla Visual Studio 2022: *https://github.com/markjprice/cs10dotnet6/tree/main/vs4win*.
- Rozwiązania dla Visual Studio Code: *https://github.com/markjprice/cs10dotnet6/tree/main/vscode*.
- Pliki interaktywnych notatników .NET: *https://github.com/markjprice/cs10dotnet6/tree/main/notebooks*.

Dobra praktyka

Jeżeli zajdzie taka potrzeba, możesz wrócić do tego rozdziału, żeby przypomnieć sobie, jak należy tworzyć wiele projektów i zarządzać nimi w Twoim edytorze kodu.

Wykorzystywanie repozytorium GitHuba w tej książce

Git jest powszechnie używanym systemem kontroli wersji kodu źródłowego. GitHub to firma, witryna oraz aplikacja, które ułatwiają zarządzanie repozytoriami Gita. Firma Microsoft kupiła GitHub w 2018 roku, dzięki czemu dostępne funkcje są coraz mocniej integrowane w narzędzia Microsoftu.

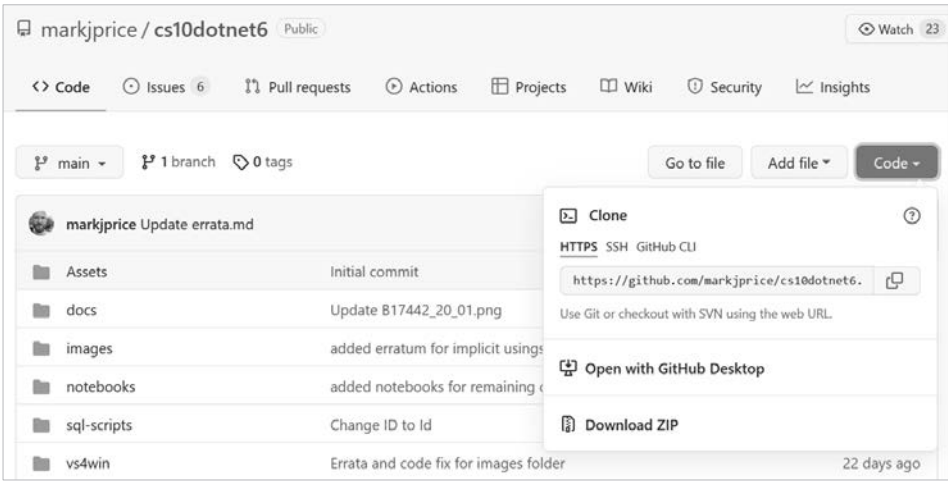
Na potrzeby tej książki przygotowałem repozytorium GitHuba i wykorzystałem je do:

- przechowywania kodu rozwiązań, które mogę aktualizować nawet po wydaniu książki;
- udostępniania dodatkowych materiałów do tej książki, takich jak poprawki, małe usprawnienia, listy przydatnych linków oraz dłuższe artykuły, które nie zmieściły się w książce.

Pobieranie kodu rozwiązań z repozytorium GitHuba

Używałem platformy GitHub do przechowywania rozwiązań wszystkich praktycznych ćwiczeń, jakie zamieszczam na końcu każdego z rozdziałów. Możesz je przejrzeć i pobrać pod adresem *https://github.com/markjprice/cs10dotnet6*.[1]

Jeżeli jednak chcesz po prostu pobrać wszystkie pliki przy użyciu Gita, to kliknij zielony przycisk *Code*, a potem wybierz opcję *Download ZIP*, tak jak na rysunku 1.19.



Rysunek 1.19. Pobieranie repozytorium jako pliku ZIP

[1] Spolszczoną wersję kodów dla całej książki można pobrać pod adresem *https://ftp.helion.pl/przyklady/c10ne6.zip — przyp. tłum.*

Zalecam też zapisanie sobie adresu repozytorium w zakładkach, ponieważ używam go również do publikowania poprawek i innych przydanych linków.

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 tym adresem możesz pobrać niezbędny instalator: *https://git-scm.com/download*.

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

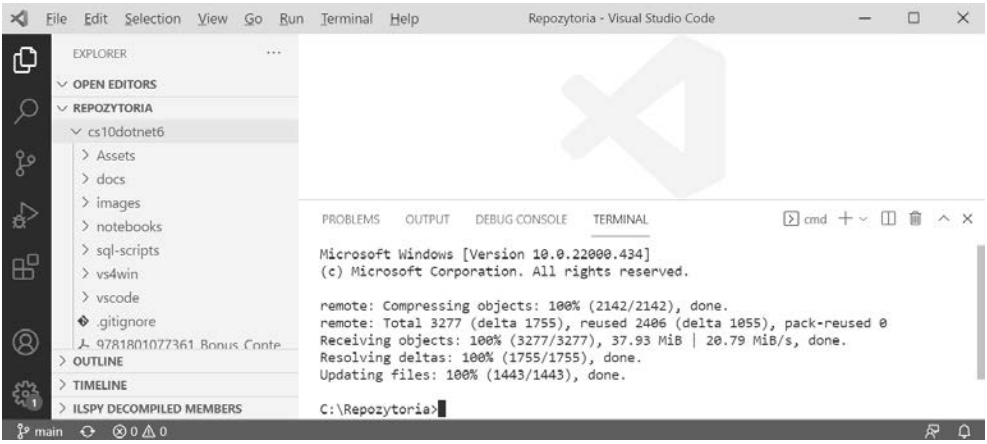
Klonowanie rozwiązań z tej książki z repozytorium kodu

Sklonujmy teraz z repozytorium kodu rozwiązanie przygotowane na potrzeby tej książki. W poniższych instrukcjach będę używał terminala z Visual Studio Code, ale te same polecenia możesz wpisać też w dowolnym innym wierszu poleceń lub oknie terminala.

1. W folderze *Dokumenty* utwórz folder o nazwie *Repozytoria* albo użyj dowolnej innej nazwy dla folderu przechowującego repozytoria Gita.
2. W Visual Studio Code otwórz folder *Repozytoria*.
3. Wybierz z menu pozycję *View/Terminal*, a następnie wprowadź poniższe polecenie:

```
git clone https://github.com/markjprice/cs10dotnet6.git
```

4. Klonowanie wszystkich rozwiązań dla poszczególnych rozdziałów na lokalny dysk twardy może zająć kilka minut, a całość będzie wyglądać tak jak na rysunku 1.20.



Rysunek 1.20. Klonowanie repozytorium za pomocą Visual Studio Code

Gdzie znaleźć pomoc?

W tym podrozdziale dowiesz się, gdzie w sieci WWW można znaleźć wartościowe informacje o programowaniu.

Przeglądanie dokumentacji Microsoftu

Najlepszym miejscem do poszukiwania pomocy dotyczącej narzędzi programistycznych i platform firmy Microsoft są strony Microsoft Docs dostępne pod adresem *https://docs.microsoft.com/*.

Uzyskiwanie pomocy dla narzędzia dotnet

W wierszu poleceń można poprosić narzędzie `dotnet` o podanie informacji na temat dostępnych poleceń.

1. Aby otworzyć oficjalną dokumentację polecenia `dotnet new` w oknie przeglądarki, wprowadź w terminalu Visual Studio Code poniższe polecenie:

```
dotnet help new
```

2. Aby wyświetlić treść pomocy w oknie wiersza poleceń, zastosuj opcję `-h` lub `--help`, tak jak w poniższym poleceniu:

```
dotnet new console -h
```

3. Na ekranie zobaczysz poniższy tekst (to wycinek całości):

```
Console Application (C#)
Author: Microsoft
Description: A project for creating a command-line application that can
run on .NET Core on Windows, Linux and macOS
Options:
  -f|--framework. The target framework for the project.
                      net6.0           - Target net6.0
                      net5.0           - Target net5.0
                      netcoreapp3.1.   - Target netcoreapp3.1
                      netcoreapp3.0.   - Target netcoreapp3.0
                  Default: net6.0

  --langVersion Sets langVersion in the created project file text –
 Optional
```

Przeglądanie definicji typów i ich elementów

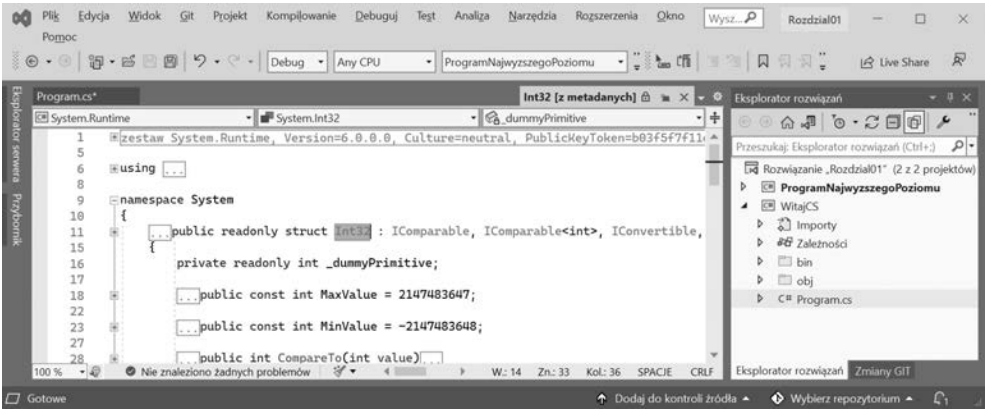
Kolejnym przydatnym klawiszem w Visual Studio Code oraz Visual Studio 2022 jest klawisz *F12*, aktywujący funkcję *Go To Definition* (Przejdź do definicji) Powoduje ona wyświetlenie publicznej definicji wybranego typu, która pochodzi z metadanych skompilowanego modułu.

Niektóre narzędzia (takie jak dekompilator ILSpy) wykonują nawet działania inżynierii wstecznej, żeby odtworzyć kod C# na podstawie metadanych oraz kodu IL.
Sprawdźmy teraz, jak można w praktyce skorzystać z funkcji *Przejdź do definicji*.

1. W Visual Studio 2022 lub Visual Studio Code otwórz rozwiązanie lub przestrzeń roboczą o nazwie *Rozdzial01*.
2. W projekcie *WitajCS*, w pliku *Program.cs*, w metodzie `Main`, wprowadź poniższą instrukcję, deklarującą zmienną typu `int` o nazwie `z`:

```
int z;
```

3. Kliknij typ `int`, a następnie naciśnij klawisz *F12* (albo kliknij prawym przyciskiem myszy i wybierz z menu kontekstowego pozycję *Przejdź do definicji* lub *Go To Definition*)
4. W nowo otwartym oknie z kodem możesz przejrzeć definicję typu `int`, której część jest widoczna na rysunku 1.21.



Rysunek 1.21. Metadane dla typu int

W ten sposób można się dowiedzieć, że typ `int`:

- Został zdefiniowany za pomocą słowa kluczowego `struct`.
- Znajduje się w module `System.Runtime`.
- Znajduje się w przestrzeni nazw `System`.
- Nosi nazwę `Int32`.
- A to oznacza, że jest aliasem dla typu `System.Int32`.
- Implementuje interfejs `IComparable`.
- Definiuje stałe określające minimalną i maksymalną wartość.
- Udostępnia takie metody jak `Parse`.

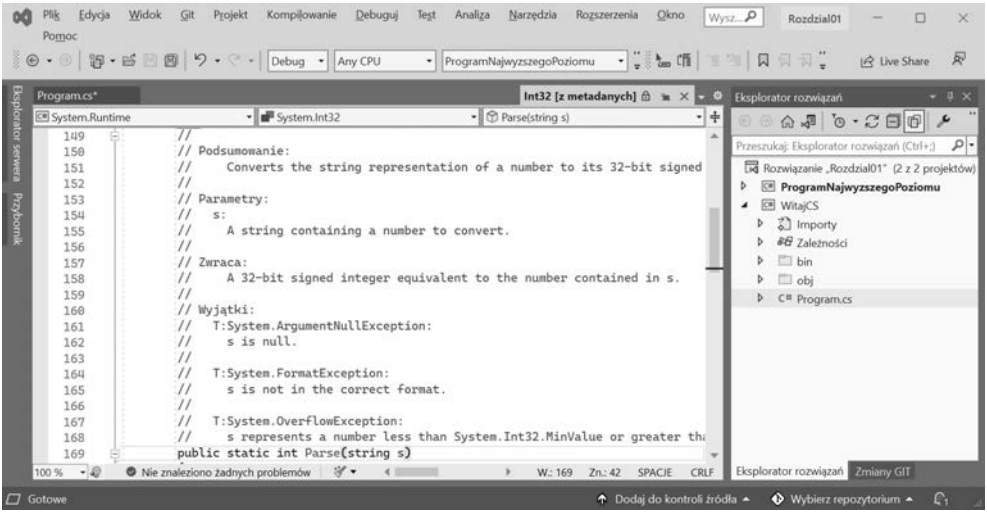
Dobra praktyka

Próbując użyć funkcji *Go To Definition* w Visual Studio Code, można czasami zobaczyć komunikat o błędzie „No definitione found", informujący, że nie znaleziono szukanej definicji. Zwykle wynika to z tego, że rozszerzenie *C#* nie wie nic o aktualnym projekcie. Trzeba wtedy wybrać z menu pozycję *View/Command Palette*, a potem wpisać `Omni` i wybrać polecenie *OmniSharp: Select Project*. Z wyświetlonej listy można wtedy wybrać projekt, z którym chcemy pracować.

Na razie funkcja *Idź do definicji* lub *Go To Definition* nie będzie szczególnie przydatna, ponieważ nie wiesz jeszcze, co oznaczają poszczególne pojęcia.

Już w połowie tej książki, po przeczytaniu rozdziałów od 2. do 6., poznasz język C# na tyle, żeby móc często korzystać z tej funkcji.

5. W oknie edytora kodu przewiń zawartość do metody `Parse` przyjmującej jeden parametr typu `string` (powinna zaczynać się w wierszu 169, natomiast opisujące ją komentarze znajdują się w wierszach od 149 do 168), tak jak na rysunku 1.22.



Rysunek 1.22. Komentarze do metody Parse z parametrem typu string

W komentarzach można zobaczyć, że Microsoft zapisał następujące informacje:

- Podsumowanie opisujące działanie metody.
- Parametry przyjmowane przez metodę, w tym przypadku wartość typu `string`.
- Wartość zwracana przez metodę, z podaniem typu danych.
- Trzy wyjątki, które mogą zostać rzucone przy wywołaniu tej metody (`ArgumentNullException`, `FormatException` i `OverflowException`). Teraz już wiesz, że każde wywołanie metody `Parse` trzeba umieścić w bloku `try`, który przechwyci ewentualnie rzucone wyjątki.

Mam nadzieję, że już nie możesz się doczekać, żeby dowiedzieć się, co to wszystko oznacza!

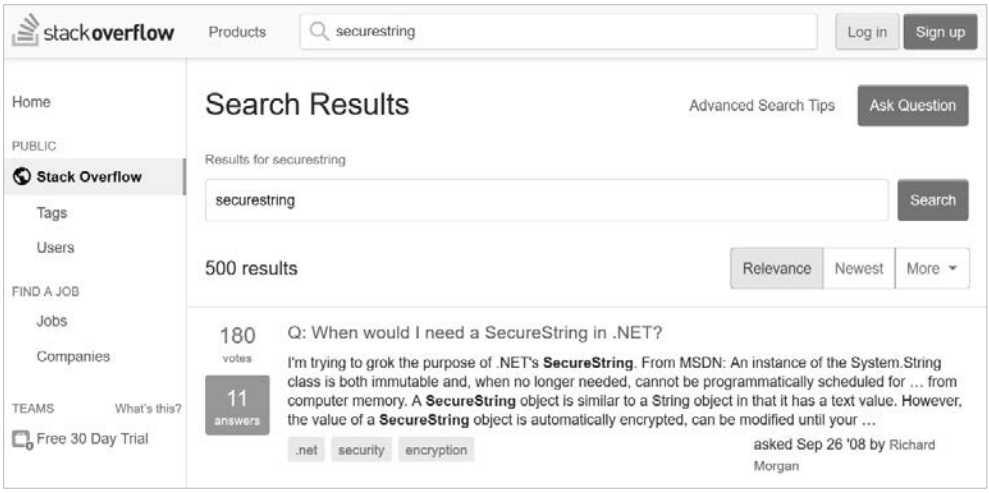
Jeszcze chwilka cierpliwości. Już prawie zakończyliśmy ten rozdział, a w kolejnym zajmiemy się szczegółami języka C#. Najpierw jednak musimy sprawdzić, gdzie jeszcze można szukać pomocy.

Poszukiwanie odpowiedzi na Stack Overflow

Stack Overflow to jedna z najpopularniejszych witryn WWW pozwalająca uzyskać odpowiedzi na trudne pytania dotyczące programowania. Jest ona tak popularna, że wyszukiwarki, np. DuckDuckGo, przygotowały specjalne metody tworzenia zapytań dla niej.

1. Uruchom swoją ulubioną przeglądarkę.
2. Otwórz stronę wyszukiwarki *duckduckgo.com* i wpisz poniższe zapytanie. Zwróć uwagę na otrzymane wyniki, które widnieją również na rysunku 1.23.

```
!so securestring
```



Rysunek 1.23. Wyniki wyszukiwania słowa securestring w serwisie Stack Overflow

Poszukiwanie odpowiedzi za pomocą Google

Możesz też skorzystać z wyszukiwarki Google i użyć jej zaawansowanych opcji, żeby zwiększyć prawdopodobieństwo uzyskania właściwych odpowiedzi na swoje pytania.

1. Przejdź na stronę Google.
2. Jeżeli szukasz informacji na temat **oczyszczania pamięci**, użyj pojęcia `garbage collection`, wpisując je do zapytania. W odpowiedzi zapewne zobaczysz wycinek

strony Wikipedii z definicją tego pojęcia w naukach komputerowych, i to w kilku wersjach językowych, w tym polskiej.

3. Możesz jednak poprawić to wyszukiwanie, ograniczając je do samych przydatnych witryn, np. Stack Overflow. Można też wykluczyć nieinteresujące nas języki programowania, takie jak C++, Rust albo Python. Z drugiej strony, można wymóc poszukiwanie odpowiedzi dla języka C# i platformy .NET, tak jak w poniższym zapytaniu:

```
garbage colection site:stackoverflow +C# -Java
```

Subskrybowanie blogów

Jeżeli chcesz być na bieżąco z rozwojem platformy .NET, to dobrze jest subskrybować oficjalnego bloga *.NET Blog*, prowadzonego przez zespoły inżynierów tworzących platformę (*https://blogs.msdn.microsoft.com/dotnet/*).

Filmy Scotta Hanselmana

Scot Hanselman z firmy Microsoft prowadzi doskonały kanał na YouTubie (*http://computerstufftheydidntteachyou.com*), gdzie opowiada o sprawach związanych z komputerami, o których nikt wcześniej Ci nie powiedział.

Polecam ten kanał każdemu, kto pracuje z komputerami.

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

Spróbuj odpowiedzieć na poniższe pytania. Pamiętaj, że choć większość odpowiedzi można znaleźć w treści tego rozdziału, to jednak w niektórych przypadkach konieczne będzie poszukanie informacji w sieci albo napisanie kawałka kodu.

1. Czy Visual Studio 2022 jest lepsze od Visual Studio Code?
2. Czy .NET 6 jest lepsza od .NET Framework?
3. Czym jest .NET Standard i dlaczego nadal jest to tak ważne?
4. Dlaczego programista może używać różnych języków (np. C# i F#) podczas pisania aplikacji działających w .NET Core?
5. Jak się nazywa metoda startowa w aplikacjach konsoli środowiska .NET i jak należy ją deklarować?

6. Czym jest program najwyższego poziomu i jak można w nim skorzystać z argumentów wiersza poleceń?
7. Co należy wpisać w wierszu poleceń, żeby skompilować i uruchomić kod źródłowy w C#?
8. Jakie są zalety używania interaktywnych notatników .NET do pisania kodu C#?
9. Gdzie należy szukać pomocy na temat wybranego słowa kluczowego języka C#?
10. Gdzie należy szukać rozwiązań typowych programistycznych problemów?

Więcej informacji

Odpowiedzi na wszystkie te pytania są dostępne w dodatku A.

Ćwiczenie 1.2 — ćwicz C# gdzie się da

Do ćwiczenia języka C# nie potrzebujesz Visual Studio 2022, Visual Studio for Mac ani Visual Studio Code. Wystarczy, że odwiedzisz stronę **.NET Fiddle** (*https://dotnetfiddle.net/*) i zaczniesz pisać kod.

Ćwiczenie 1.3 — dalsza lektura

Tematy do tej książki bardzo starannie dobrałem i starałem się uzyskać między nimi równowagę. Pozostałe przygotowane przeze mnie treści można znaleźć w repozytorium GitHuba utworzonym na potrzeby tej książki.

W niniejszej książce prezentuję najbardziej podstawowe informacje na temat języka C# i środowiska .NET, które są potrzebne wszystkim programistom. Zamiast większych przykładów lepiej było jednak podać linki do dokumentacji firmy Microsoft albo do wybranych artykułów publikowanych przez niezależnych autorów w sieci.

Skorzystaj z linków udostępnionych na poniższej stronie, aby dowiedzieć się więcej na tematy poruszane w tym rozdziale:

https://github.com/markjprice/cs10dotnet6/blob/main/book-links.md#chapter-1---hello-c-welcome-net

Podsumowanie

W tym rozdziale:

- przygotowaliśmy środowisko programistyczne;
- omówiliśmy różnice pomiędzy .NET, .NET Core, .NET Framework, Xamarin i .NET Standard;

- skorzystaliśmy z Visual Studio Code i .NET SDK oraz z Visual Studio 2022, aby przygotować prostą aplikację konsoli;
- użyliśmy interaktywnych notatników .NET, aby nauczyć się uruchamiać wycinki kodu C#;
- dowiedzieliśmy się, jak pobrać przykładowe kody dla tej książki z repozytorium GitHub;
- co najważniejsze, poznaliśmy sposoby poszukiwania pomocy.

W następnym rozdziale nauczymy się mówić w języku C#.

C# i .NET: najlepsze środowisko do programowania w najlepszym języku!

C# to jedno z najdoskonalszych dzieł Microsoftu. Co więcej, język ten, podobnie jak wieloplatformowy framework .NET, jest konsekwentnie rozwijany i wzbogacany. Coraz więcej profesjonalistów przekonuje się, że używanie C# jest przyjemne i satysfakcjonujące, a każda kolejna wersja przynosi liczne udogodnienia i nowe funkcjonalności. Aktualne wersje C# 10 i .NET 6 pozwalają bez trudu tworzyć rozbudowane serwisy internetowe czy wieloplatformowe aplikacje mobilne.

Ta książka jest kolejnym, gruntownie przebudowanym wydaniem cenionego podręcznika dla programistów #C. Dzięki niej nauczysz się najważniejszych zasad korzystania z tego języka. Znajdziesz tu drobiazgowe omówienie API środowiska .NET, jak również techniki pracy z systemem plików, asynchronicznymi strumieniami, serializacją i szyfrowaniem. Po przyswojeniu zagadnień związanych z językiem C# i aplikacjami konsoli dowiesz się, jak tworzyć praktyczne aplikacje i usługi z wykorzystaniem biblioteki ASP.NET Core, a także wzorzec MVC i technologię Blazor. Zapoznasz się z metodami stosowania wielozadaniowości do poprawy wydajności i skalowalności aplikacji. Przeczytasz też, w jaki sposób ASP.NET Core ułatwia pracę wielu zespołów programistów. Poszczególne zagadnienia zostały zilustrowane dokładnie wyjaśnionymi przykładami kodu, a dodatkowe ćwiczenia pozwolą Ci na utrwalenie zdobytych umiejętności.

Z tą książką nauczysz się:

- tworzyć własne typy w programowaniu zorientowanym obiektowo
- pisać, testować i debugować funkcje
- odczytywać dane i manipulować nimi za pomocą LINQ
- pracować z Entity Framework Core, a także Microsoft SQL Server i SQLite
- tworzyć usługi sieciowe i interfejsy użytkownika
- projektować aplikacje wieloplatformowe

Mark J. Price

specjalizuje się w programowaniu w języku C#. Pracuje w Microsofcie, tworzy rozwiązania dla Microsoft Azure. Zdał ponad 80 egzaminów Microsoftu. Zajmuje się też dydaktyką: prowadzi szkolenia wprowadzające do usług Digital Experience Platform, wiodącego systemu CMS.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9074-4

Cena: 159,00 zł

Packt