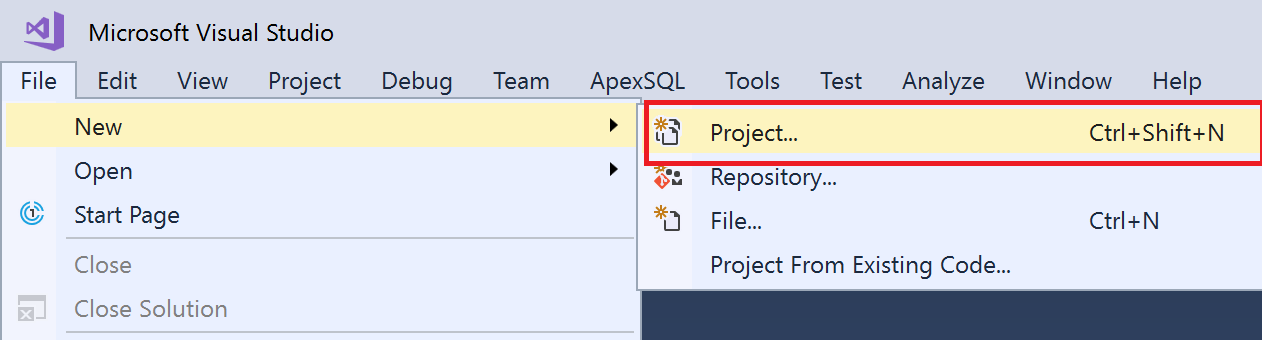
Ćwiczenie 1

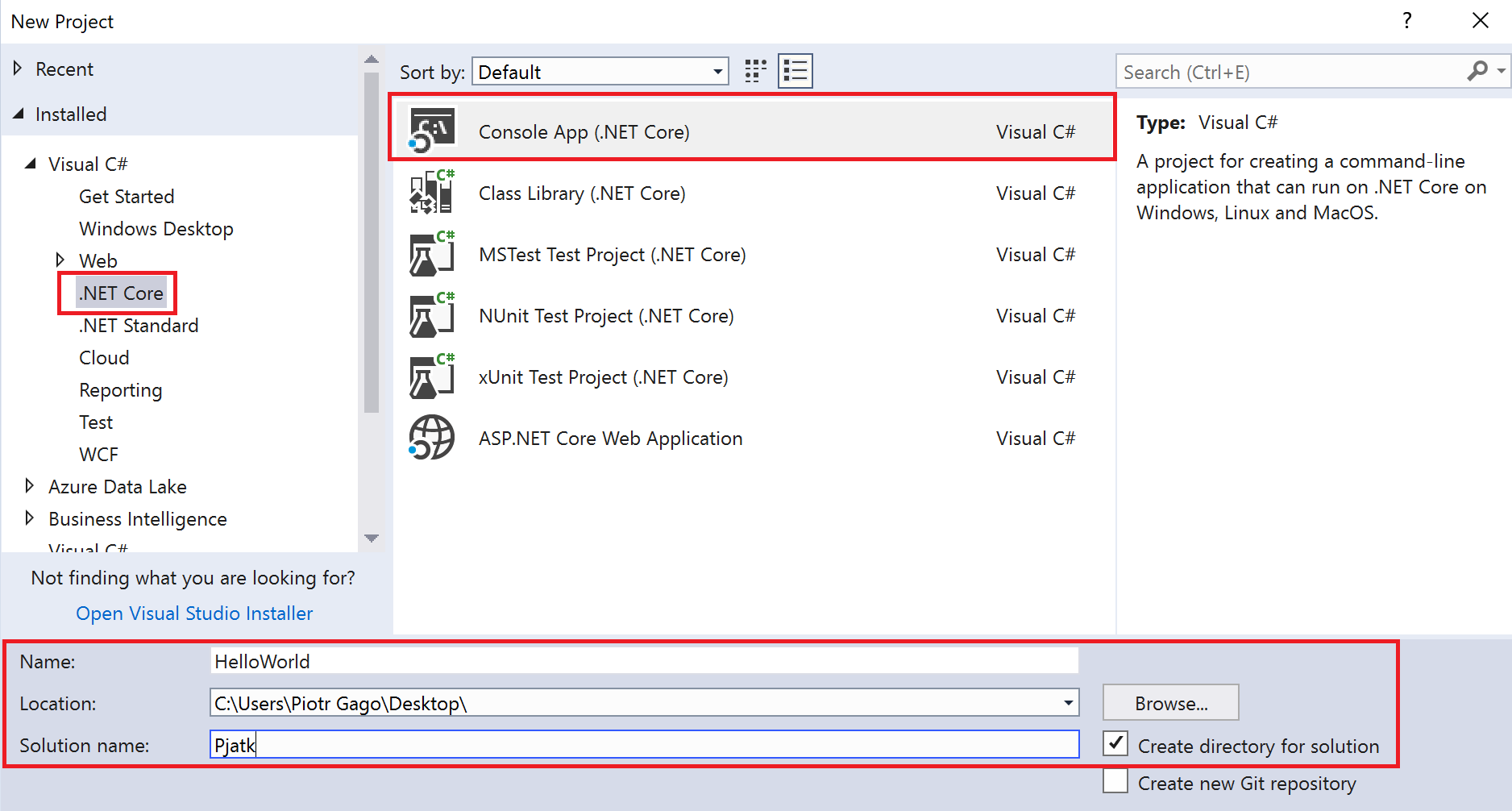
W niniejszym ćwiczeniu zapoznamy się z podstawami pracy z Visual Studio na przykładzie prostej aplikacji konsolowej.

## Tworzenie nowego projektu

1. Otwieramy program **Visual Studio 2017**. Po uruchomieniu programu mamy możliwość utworzenia konta Microsoft (nie jest to wymagane - pozwala na np. zapisywanie swoich ustawień VS w chmurze). W przeciwnym wypadku **klikamy “Not now, maybe later”**.
2. Następnie tworzymy nowy projekt. Wybieramy **File -> New -> Project...**

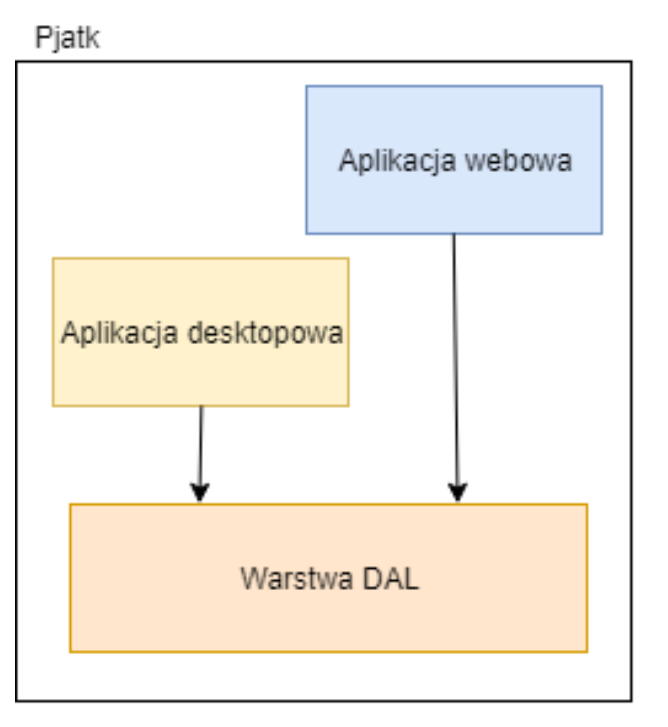


4. Wybieramy szablon nowego projektu. Następnie klikamy **Visual C# -> .NET Core -> Console App**. Tworzymy prostą aplikację konsolową. Klikamy checkbox **Create directory for solution** i uzupełniamy pola **Name i Solution** name jako **HelloWorld i Pjatk.** Proszę również za pomocą przycisku **Browse** wybrać odpowiednią lokalizację projektu - proszę wybrać Pulpit.



5. Dwa słowa o solucji. **Solution name**, czyli nazwa solucji jest jednym z pierwszych elementów nadawania architektury naszej aplikacji. Solucję możemy sobie wyobrazić jako kontener w ramach którego możemy przechowywać wiele różnych projektów.

6. Często będziecie pracowali nad aplikacją, która składa się z wielu różnych projektów. Np. pracujemy nad projektem dla naszej uczelni. Możemy posiadać solucję o nazwie Pjatk, a wewnątrz niej projekty o nazwach: **AplikacjaWebowa, AplikacjaDesktopowa, WarstwaDal** (warstwa *Data Access Layer* **- kod odpowiedzialny za komunikację z bazą danych**) it. AplikacjaWebowa i AplikacjaDesktopowa mogą korzystać z tej samej biblioteki DAL, dzięki czemu nie musimy kopiować kodu. Komunikacja z bazą danych dla obu aplikacji jest taka sama. Dzięki temu mogą współdzielić wspomnianą biblioteką.

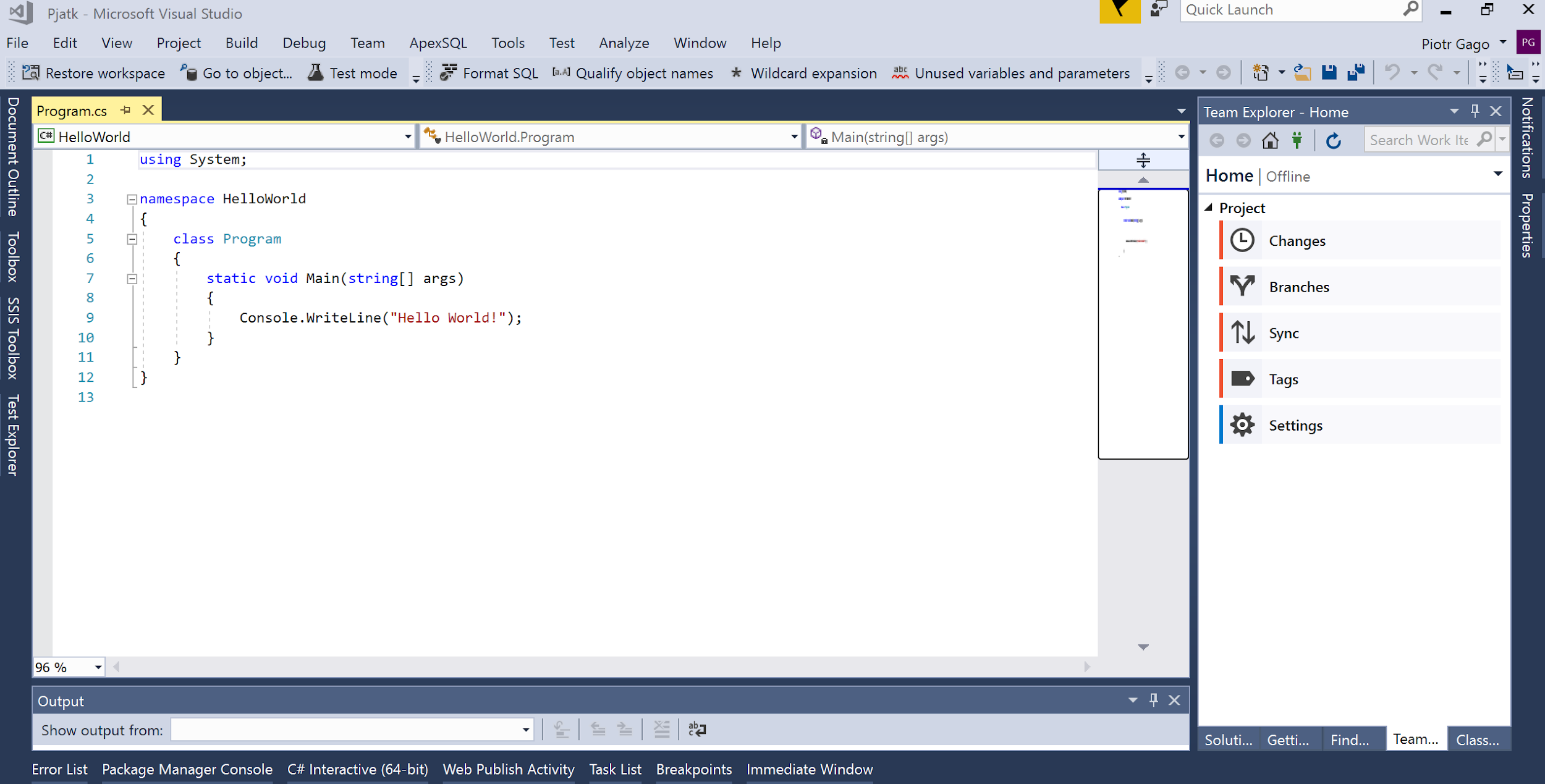


7. Pojedynczy projekt określamy jako ***Assembly***. Assembly możemy opisać jako najmniejszy element jaki możemy wydać/opublikować. Oznacza to, że często pojedyncze assembly będzie biblioteką - np. Warstwadal.dll (ang. *Dynamic Link Library*), która posiada własną wersję, nazwę autora i inne metadane. Jedno “assembly” może korzystać z innych tworząc graf zależności między różnymi bibliotekami.

8. Kolejny element na który należy zwrócić uwagę. Po lewej stronie na liście szablonów mamy możliwość wyboru wielu różnych języków i rodzajów projektów. Visual C# to jeden z nich, ale mamy również F#, Visual Basic, Python itd. Platforma .NET pozwala nam na pracę z różnymi językami i rodzajami projektów. Niezależnie od wybranego języka nasza aplikacja jest kompilowana do języka pośredniego (dosłownie IL - ang. *Intermediate Language*), który dopiero później jest uruchamiany przez platformę CLR (ang. *Common Language Runtime*). Warto wspomnieć, że platforma i cały opisany proces może przypominać JVM - wirtualną maszynę Javy. Faktycznie oba procesy są bardzo podobne.

## Uruchamianie projektu

1. Nasz projekt jest gotowy i domyślnie powinniśmy zobaczyć główne okno programu z klasą Program.cs i metodą Main. Jak widać kod jest tutaj bardzo podobny do języka Java.



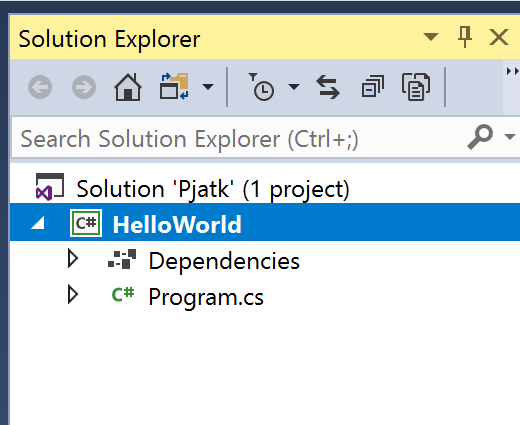
2. Spróbujmy teraz uruchomić aplikację. Klikamy Ctrl+F5, który pozwoli nam skompilować aplikację i ją uruchomić. Powinniśmy zobaczyć okno konsolowe podobne do poniższego.



3. Jak widać aplikacja została skompilowana i uruchomiona. Teraz zapoznamy się z podstawowymi funkcjami programu VS, które będą nam niezbędne podczas kolejnych ćwiczeń.

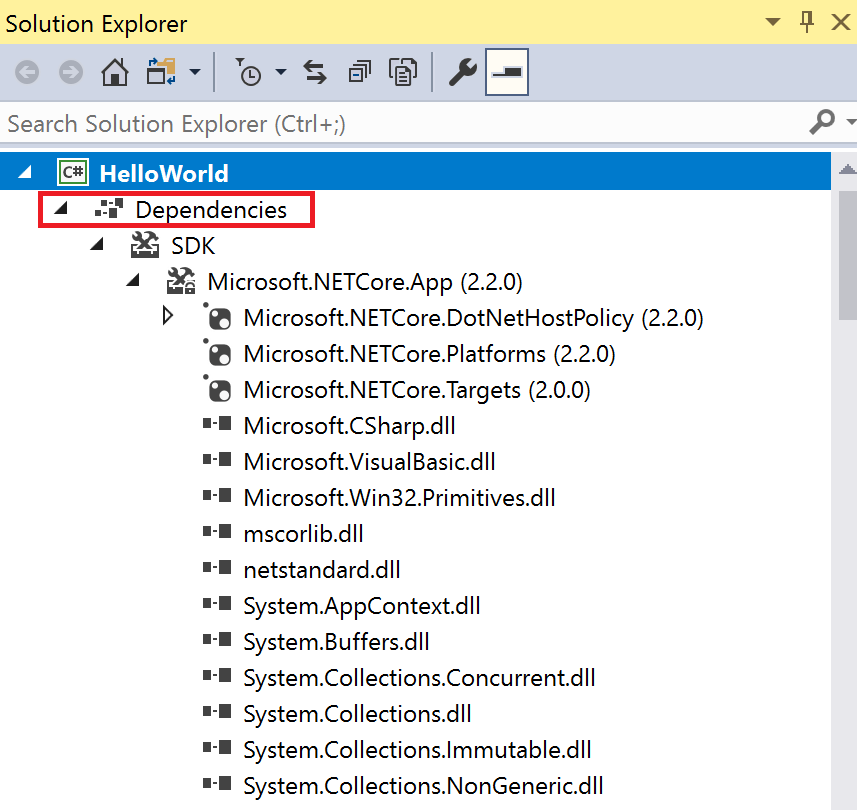
## Solution Explorer

1. Po prawej stronie głównego ekranu naszej aplikacji powinniśmy zobaczyć okno **Solution Explorer**.



2. Jest to jedno z najważniejszych okien naszej aplikacji. W tym oknie widzimy wszystkie elementy naszego projektu. Jak widać w obrębie solucji Pjatk mamy jeden projekt HelloWorld. Zgodnie z tym o czym mówiliśmy wcześniej moglibyśmy umieścić tutaj więcej projektów.

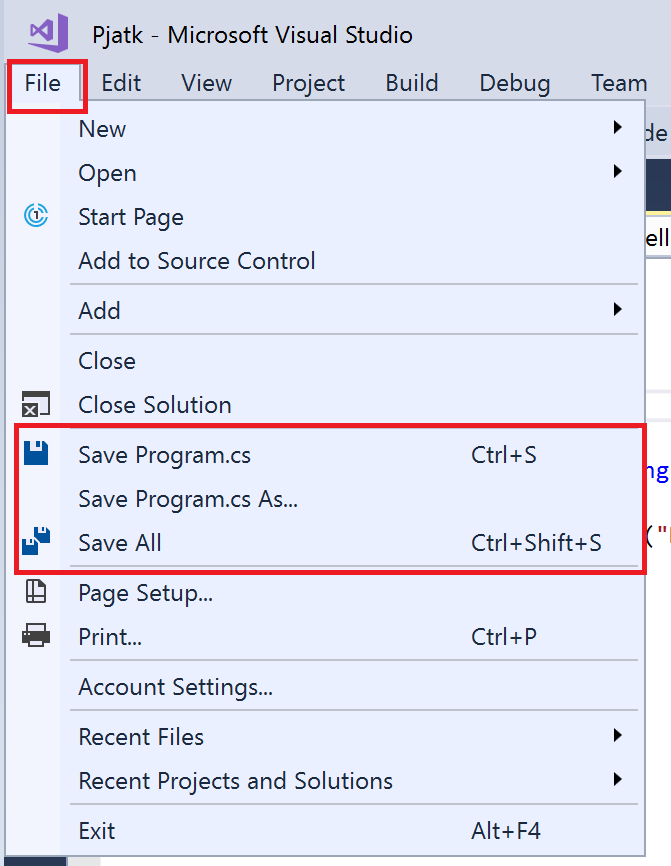
3. Rozwińmy zakładkę **Dependencies**.



4. Jak widać do naszego projekty mamy podpięte pewne domyślne biblioteki. W Javie większość domyślnych bibliotek zaczyna się od “java…”. W naszym wypadku większość domyślnych bibliotek składa się z nazwy “System…”. Wśród domyślnych bibliotek znajdziemy kolekcje, metody dostępu do plików i wiele więcej. Będziemy je poznawać w miarę naszych potrzeb. Możemy z powrotem zakładkę Dependencies.

## Otwieranie i zapisywanie projektu

1. W tej sekcji omówimy sobie w jaki sposób możemy zapisać cały projekt. Pierwsza opcja, która przychodzi do głowy to **File -> Save…**

****

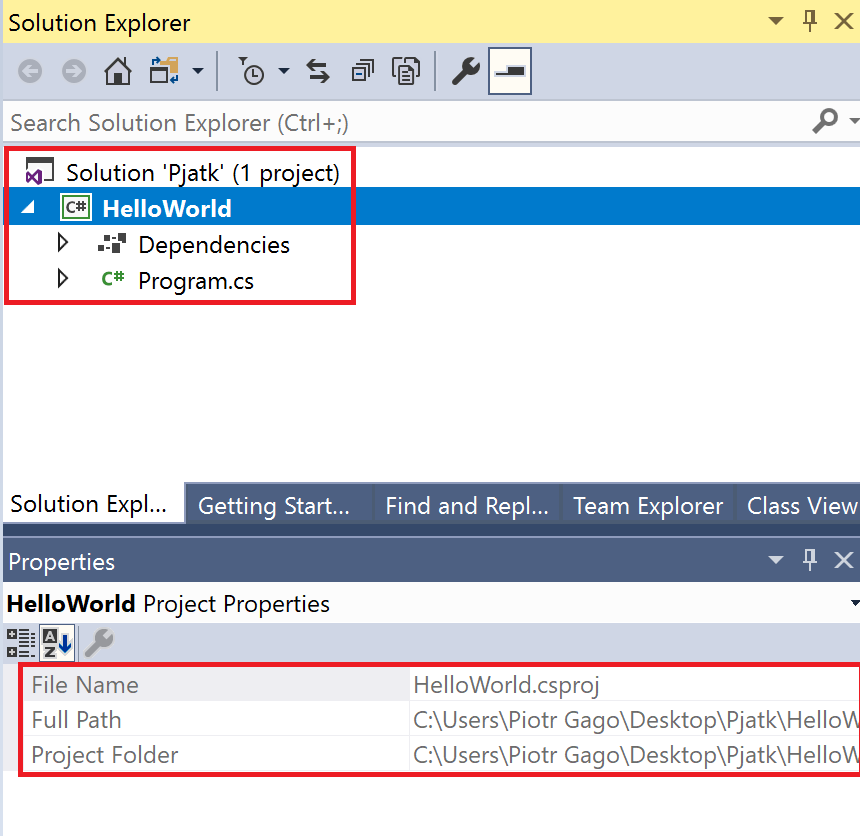
2. Należy pamiętać, że zaznaczone opcje służą generalnie do zapisania:

2a. **Save All** - zmian we wszystkich aktualnie otwartych plikach

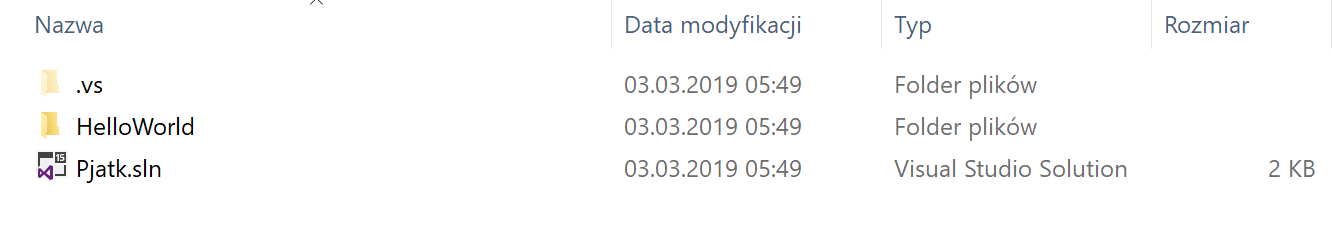
2b **Save Program.cs** - zapisania wyłącznie aktualnie otwartego pliku, a nie całego projektu

3. Należy o tym pamiętać, żeby przypadkiem nie przesłać lub nie zapisać wyłącznie części projektu. W takim wypadku nie będziemy w stanie go uruchomić.

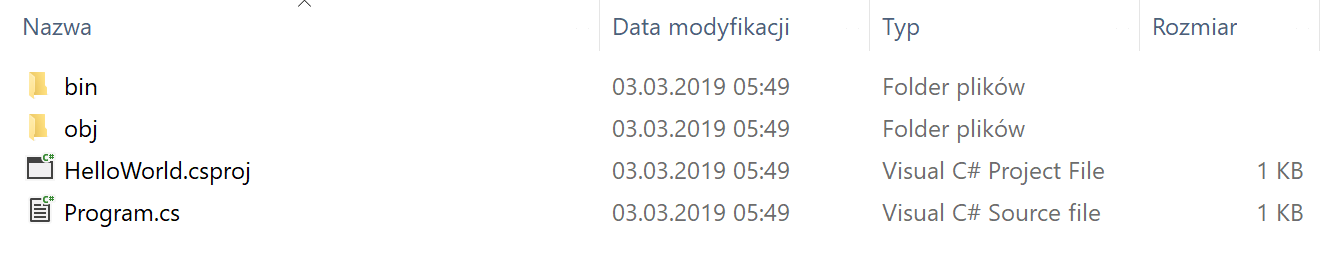
4. Zerknijmy w folderze na dysku jak wygląda nasz projekt. Pierwsza ważna uwaga: **jak znaleźć nasz projekt, jeśli nie wiemy gdzie go zapisaliśmy (my zapisaliśmy na pulpicie, ale często się zdarza, że stworzymy nowy projekt i nie pamiętamy gdzie). Możemy kliknąć i zaznaczyć nasz projekt w oknie Solution Explorer.** Następnie w oknie Properties (Właściwości) pojawią się dodatkowe informacje o aktualnym wybranym elemencie. W wypadku projektu znajdziemy tam pełną ścieżkę do naszego projektu.

****

5. Zerknijmy w takim razie na pulpit i zobaczmy co znajduje się w projekcie. W folderze o nazwie **Pjatk** (taki sam jak nazwa solucji) znajdziemy poniższe elementy:

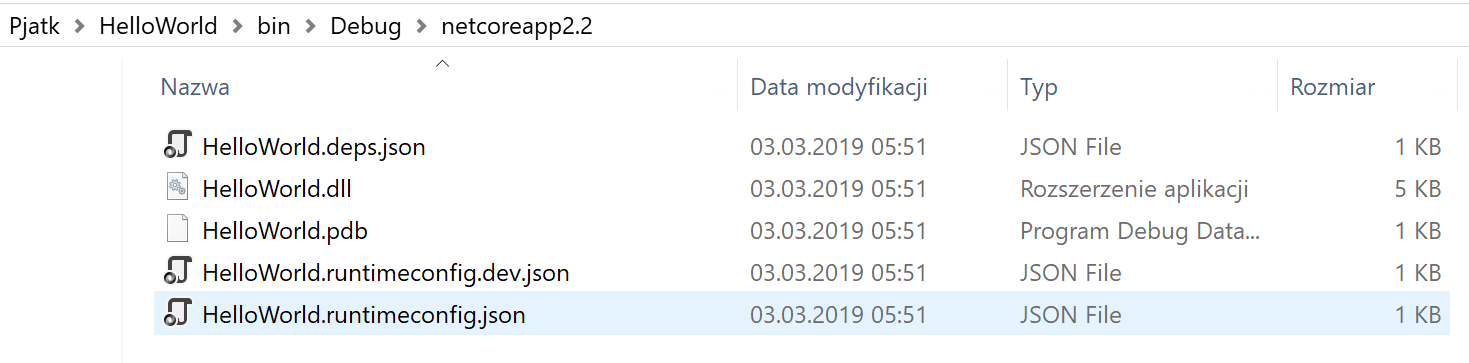


6. Folder **.vs** może nie być dla Was widoczny. Jest to folder ukryty i jego widoczność zależy od ustawień Windows. W środku znajdują się ustawienia Visual Studio. Ten folder nie musi (a często nie powinien) być kopiowany razem z projektem. Plik **Pjatk.sln** (sln - solution) to plik, który możemy otworzyć w Visual Studio. Możemy go traktować jak skrót, który pozwala “wciągnąć” w Visual Studio wszystkie projekty będące częścią solucji. Uwaga: nie jest to projekt. **Zapisanie i przesłanie wyłącznie pliku .sln nie pozwoli na odtworzenie projektu.** Folder **HelloWorld** to oczywiście nasz projekt. Sprawdźmy co znajduje się w środku.



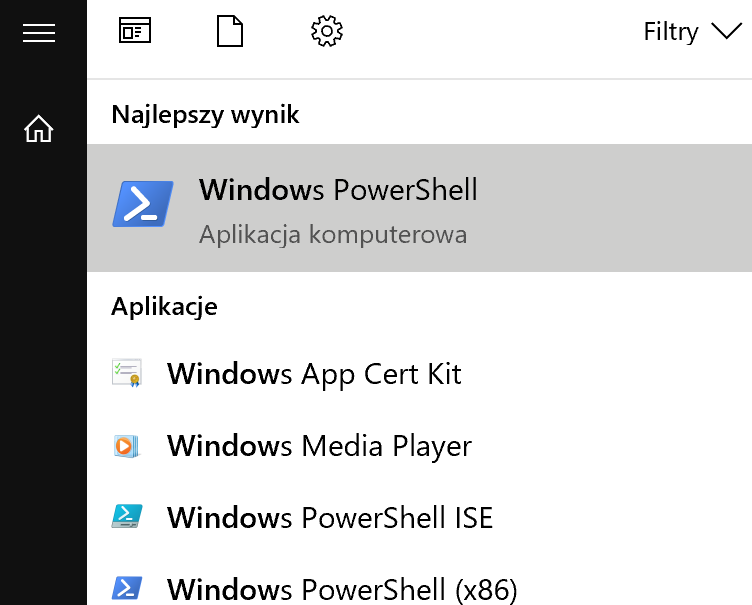
7. W środku projektu znajdziemy kilka plików i folderów. Program.cs - tego się spodziewaliśmy. Jest to nasza klasa. **HelloWorld.csproj** - **jest to plik, który** nie był widoczny bezpośrednio w Visual Studio. Jest jednak niezwykle ważny. Możemy go otworzyć w Visual Studio i stanowi zestaw instrukcji, które dokładnie określają co, w jakiej kolejności i w jaki sposób ma być kompilowane w naszym projekcie. Innymi słowy ten plik zbiera w całość wszystkie elementy projektu i pozwala VS na ich kompilację. Ten plik jest generowany, więc nie będziemy musieli go edytować ani tworzyć, ale zawsze musimy go skopiować.

8. Wejdźmy teraz do folderu bin (ang. *binary*). Wewnątrz znajdziemy ostatnio skompilowaną wersję naszej aplikacji - powstałą w momencie jej uruchomienia w Visual Studio.



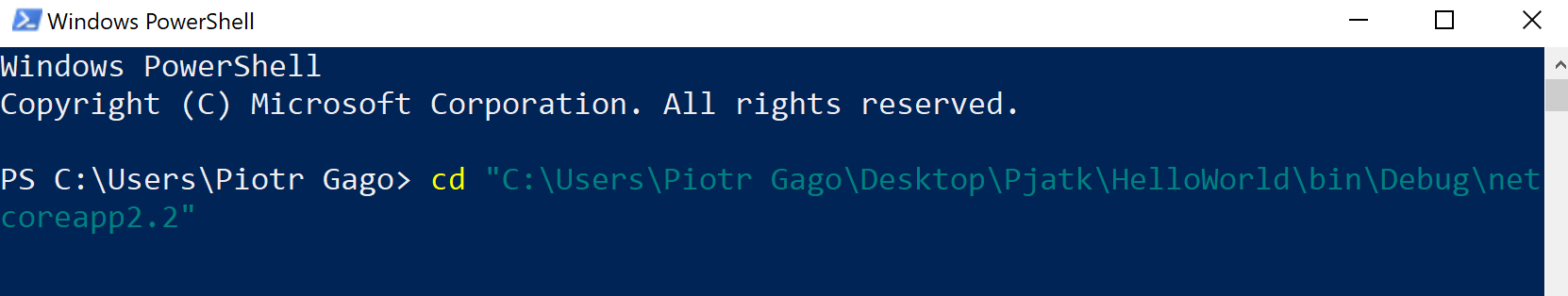
9. Właściwie najważniejszą rzeczą jest plik HelloWorld.dll. Tutaj znajduje się kod binarny naszej aplikacji w języku IL. Mamy ponadto trzy pliki **z końcówką .json**. Te pliki pozwalają na określenie konfiguracji, która może być później przez nas program pobrana. Nie są one niezbędne. Plik **HelloWorld.pdb** - zawiera pewne dodatkowe informacje związane z procesem debugowania aplikacji. Ten plik również w wersji ostatecznej (np. przekazywanej klientowi) nie jest niezbędny.

10. Skoro **HelloWorld.dll** to nasza aplikacja konsolowa to jak możemy ją uruchomić? Podobnie jak w przypadku aplikacji pisanych w Javie. O**twórzmy program/konsolę Windows Powershell.**

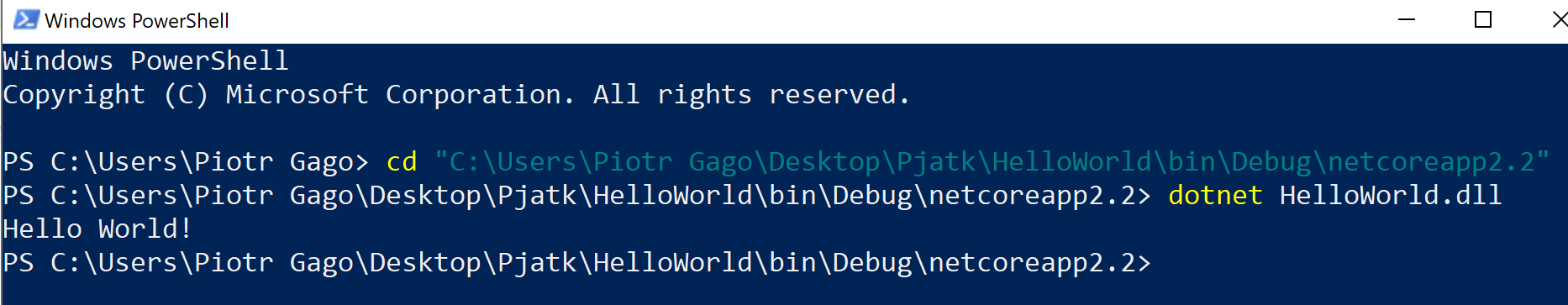


11. Kopiuje ścieżkę do miejsca gdzie znajduje się plik HelloWorld.dll i wewnątrz konsoli wpisuję komendę: **cd “C:\Users\Piotr Gago\Desktop\Pjatk\HelloWorld\bin\Debug\netcoreapp2.2”**

Oczywiście w miejsce mojej ścieżki **każdy wpisuje swoją.**

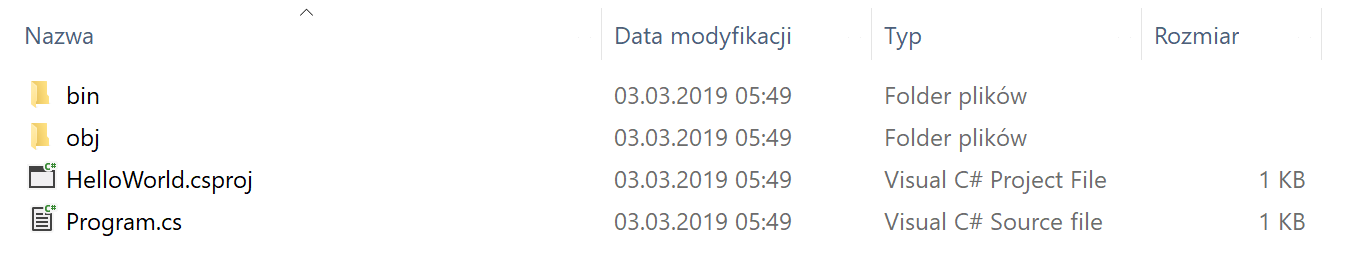


12. Komenda **cd** pozwoliła nam zmienić aktualny folder w jakim pracuje konsola. Następnie możemy uruchomić naszą aplikację. W tym celu wpisujemy komendę **dotnet HelloWorld.dll**



13. Jak widać nasza aplikacja została uruchomiona. Komenda **dotnet** pozwala wywołać CLR (ang. *Common Language Runtime*) i uruchomić z jego pomocą naszą aplikację. CLR za pomocą kompilacji just-in-time pozwala zamienić kod IL na kod maszynowy.

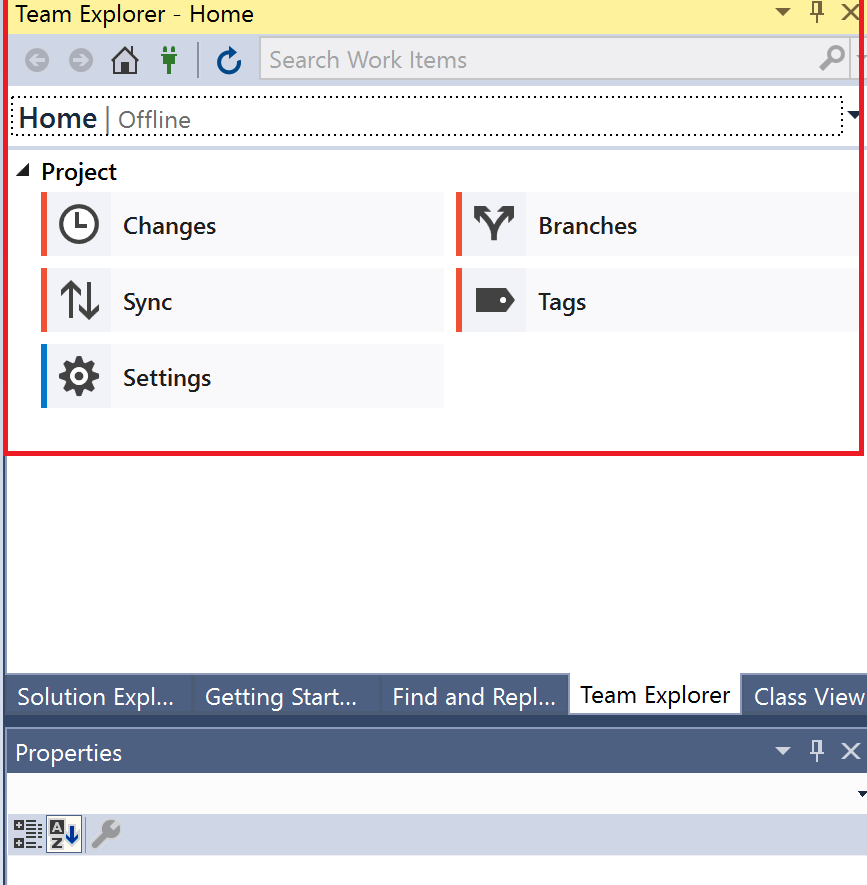
14. Wróćmy teraz z powrotem do głównego folderu naszej solucji.



15. Dwa ostatnie słowa o folderze **obj**. W tym folderze znajdziecie częściowe artefakty procesu kompilacji. Innymi foldery bin i obj mogą zostać pominięte w momencie kiedy przesyłacie aplikację lub chcecie ją zapisać. Oba foldery zostaną automatycznie wygenerowane w momencie kiedy ponownie skompiluje się projekt w Visual Studio. Wróćmy teraz do Visual Studio.

## Team Explorer

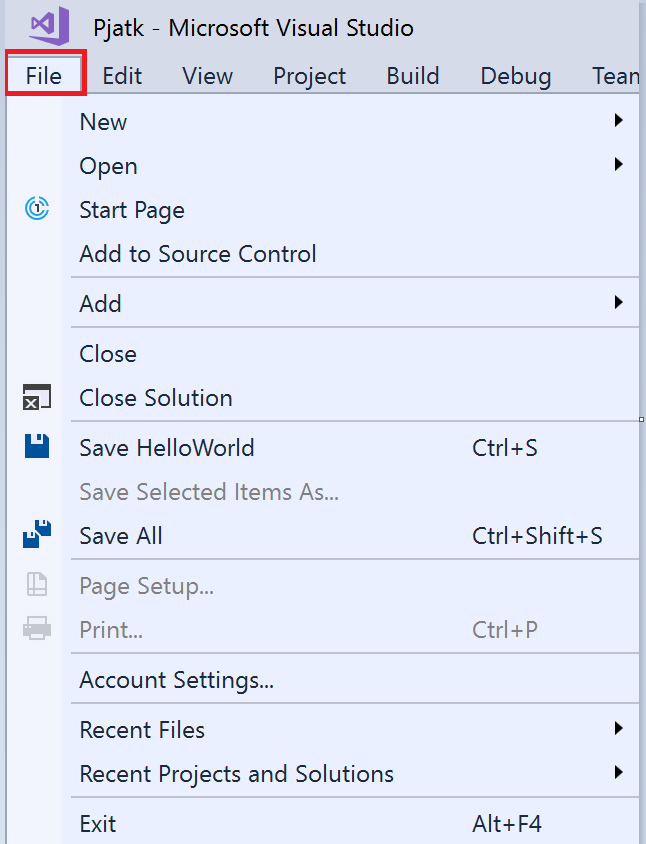
1. Kolejne okienko w Visual Studio, które będzie dla nas przydatne to Team Explorer.



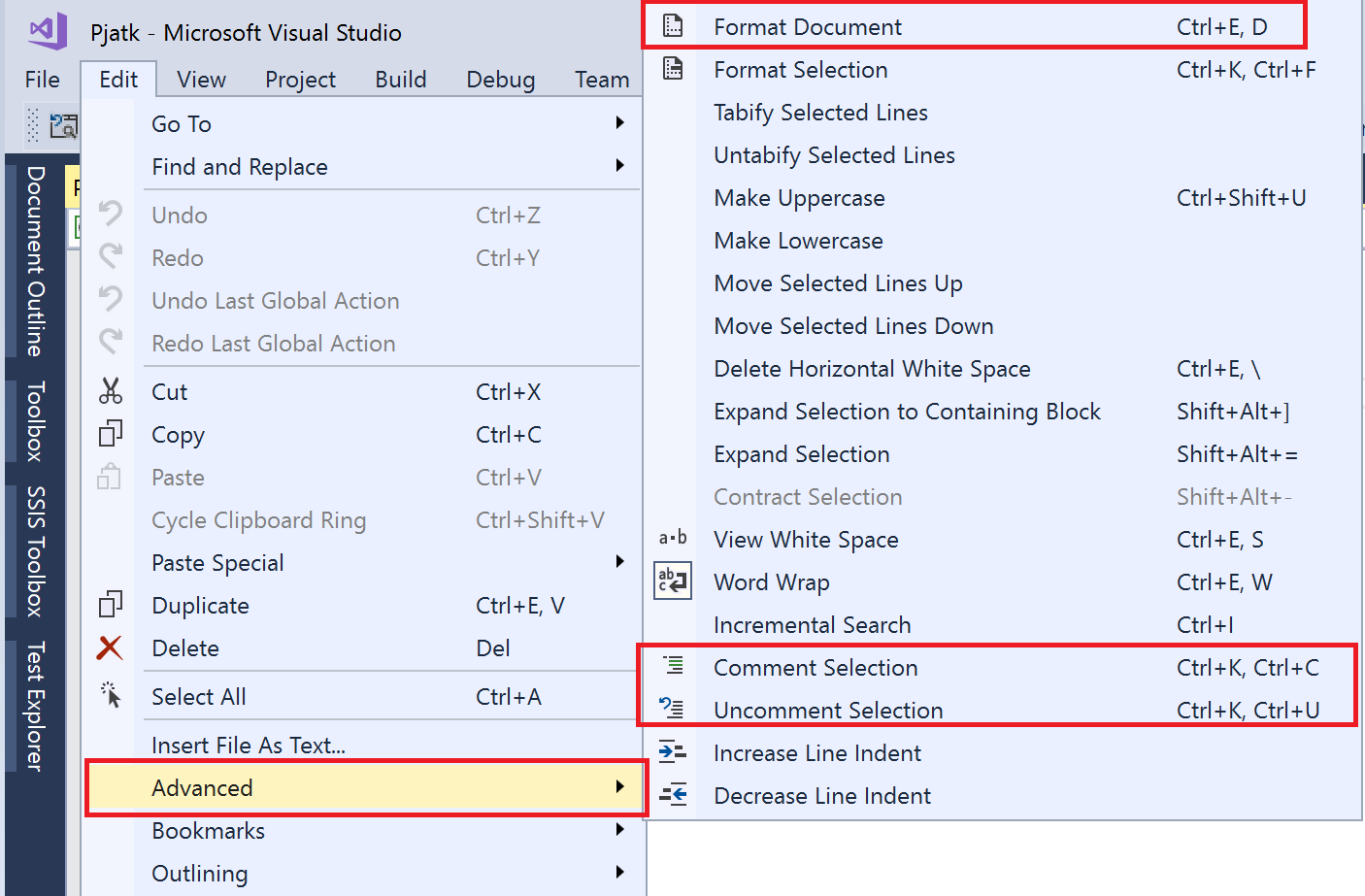
2. Zawartość tego okienko może być inna w Waszym przypadku. Na razie nie musicie się tym przejmować. Okno Team Explorer pozwala na komunikację z repozytorium kodu. Okno to pozwoli Wam zapisać projekt i późniejszą pracę z innymi uczestnikami projektu. Z czasem wspomnimy o wykorzystywanie repozytorium kodu i narzędzi takich jak Git.

## Inne okienka

1. Teraz zerkniemy w skrócie na inne okna, które mogą się nam przydać w Visual Studio. Zaczniemy od lewego górnego rogu.
2. **Menu File** - jak już wiemy - pozwoli nam nam m.in. na stworzenie nowego projektu lub otwarciu istniejącego.

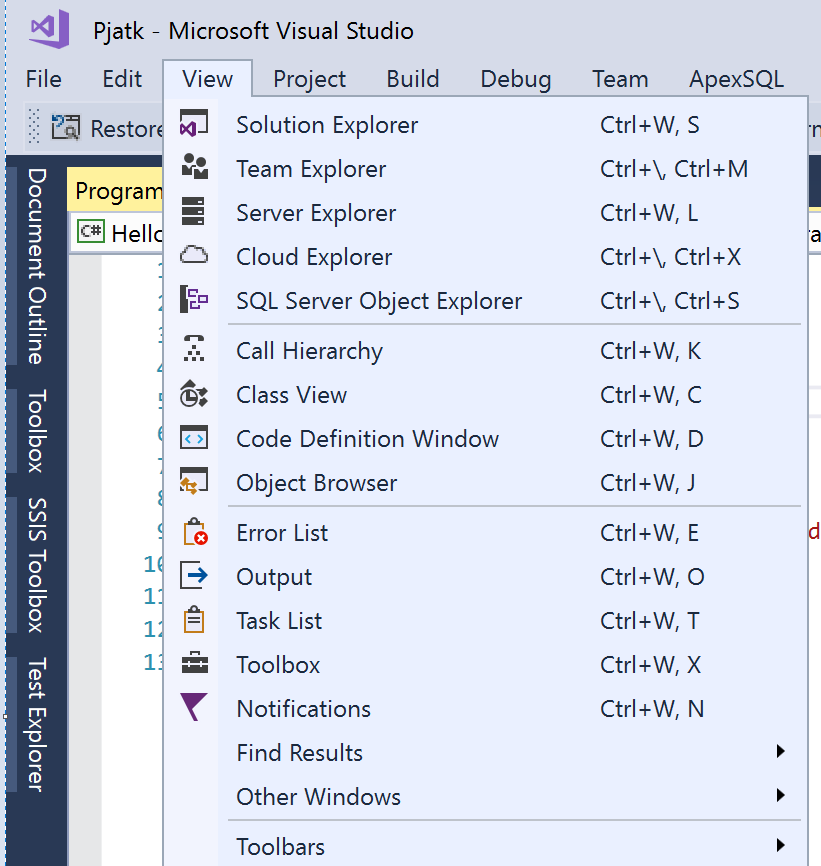


2. **Menu Edit** - w tym oknie znajdziemy typowe ustawienia znane z innych programów - m.in. wklejanie i kopiowanie. Na uwagę zasługuje kilka funkcji, które możemy znaleźć w zakładce **Edit -> Advanced**. Są to opcje **Format document i Comment/Uncomment Selection**. Czyli automatyczne formatowanie kodu i zakomentowywanie/odkomentowywanie.



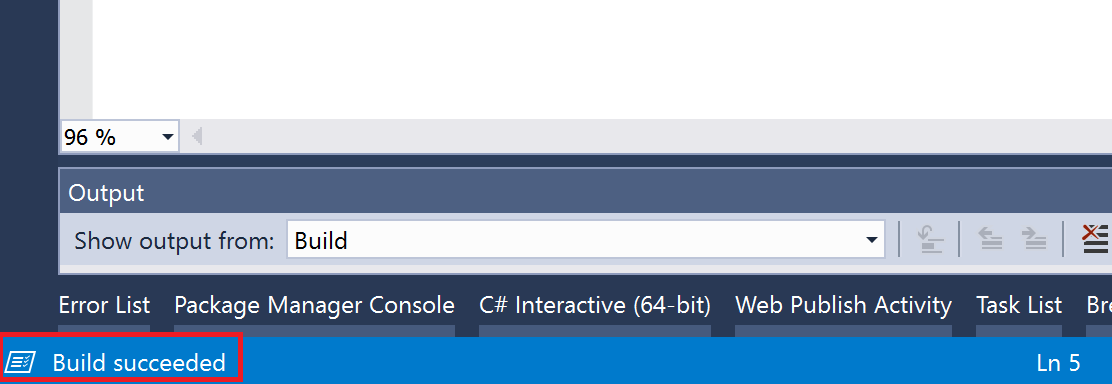
2. **Menu View** - również wiemy już, że pozwoli nam na przywrócenie dowolnego okna w Visual Studio. Jak widać w VS jest niezwyklu dużo różnego rodzaju okien. Tak naprawdę nas będzie interesowało jedynie kilka z nich - Solution Explorer, Team Explorer, Server Explorer (pozwalający na połączenie się różnymi źródłami danych) i Error List (tutaj znajdziemy listę błędów w naszej aplikacji).

Warto zauważyć, że niemal taką samą dodatkową ilość okien znajdziemy w zakładce View -> Other Windows. Jak już wspomniałem - nie przejmujemy się tym. Na razie korzystamy z konkretnych funkcji.



4. Nastepne menu to **Project**. Tej zakładki nie będziemy na razie omawiać. Jej zawartość zależy od aktualnego projektu.

5. Menu **Build**. To menu jest dla nas niezwykle ważne i pozwala na kompilowanie aplikacji. Bardzo często będziemy chcieli sprawdzić czy aplikacja dalej się kompiluje, ale niekoniecznie będziemy chcieli ją uruchamiać. Znajdziemy tutaj trzy pomocne funkcje: **Build**, **Build Solution** i **Clean**. Opcja **Build** pozwala nam skompilować całą solucję. Uwaga. Aplikacja nie zostanie uruchomiona po procesie kompilacji. Ponadto kompilacja obejmuje całą solucję i wszystkie projekty, które się w niej znajdują. Proszę ją uruchomić. Jeśli kod jest poprawny i aplikacja zawiera wszystkie niezbędne dla niej zależności otrzymamy komunikat **Build succeeded**.



6. Opcje **Rebuild** usuwa wszystkie poprzednie artefakty kompilacji i ponownie kompiluje projekt “od zera”. Opcja **Clean** wyłącznie usuwa wszystkie poprzednie artefakty kompilacji.

7. Pozostaje nam jeszcze kilka okien, które warto omówić - zostaną one omówione w trakcie ćwiczeń bezpośrednio z prowadzącym.