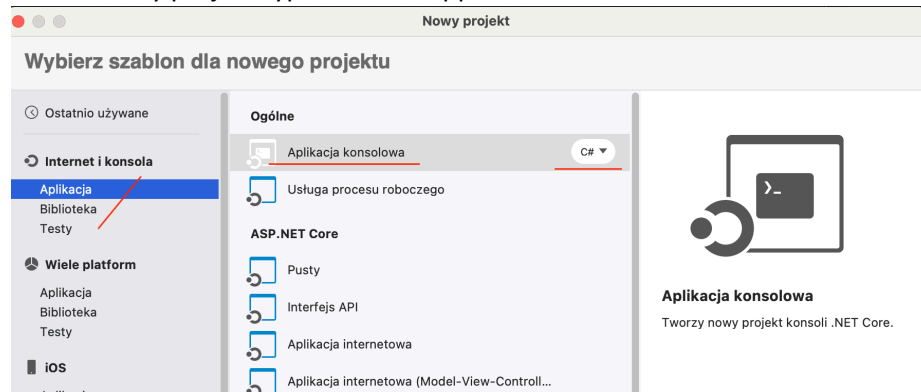


# Wprowadzenie do EntityFramework.Core

1. Proszę o bieżące dokumentowanie wykonywanych operacji w dokumencie ImieNazwiskoEFLab. Na koniec zajęć – niezależnie od etapu na którym będziesz proszę o upload tego dokumentu do zadania EfektyPracyNaKoniecZajęć na moodlu.

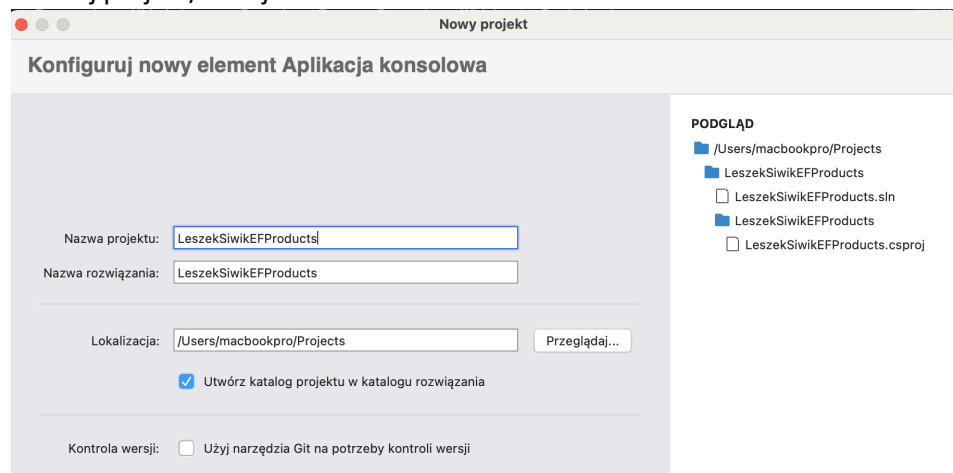
## I. HelloWorld:

- a. Uruchom Visual Studio
- b. Stwórz nowy projekt typu Console Application

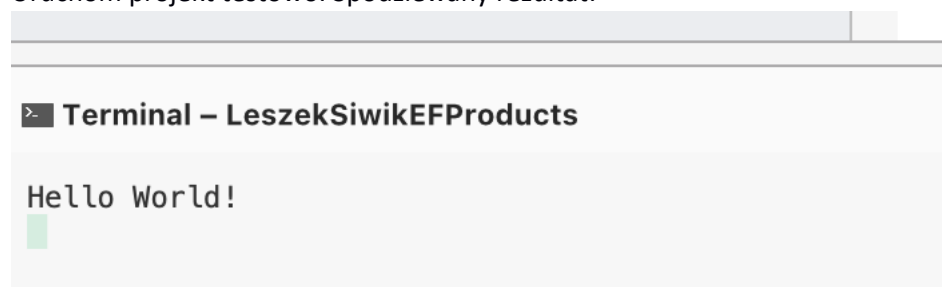


Wersja frameworka z którym chcemy pracować to .NET Core 6.0/7.0

- c. Nazwij projekt/solucję: ImieNazwiskoEFProducts



- d.
- e. Uruchom projekt testowo. Spodziewany rezultat:



- f. Dodaj do projektu klasę Product

- i. Usuujemy domyślny konstruktor (jeśli jest – obecny na macu, nieobecny na windowsach)
- ii. Dodajemy do klasy produktu trzy publiczne property (prop + tab tab)
  1. Int ProductID
  2. String ProductName
  3. int UnitsOnStock

```

namespace LeszekSiwikEFProducts
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
    }
}

```

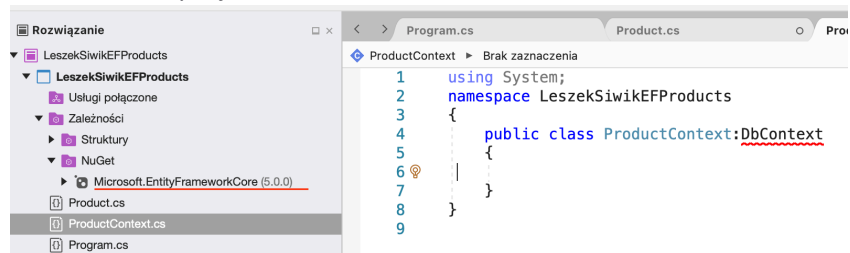
- 4.
- iii. Dodaj do projektu klasę ProductContext. Będzie to klasa EntityFrameworkowa która będzie „zarządzać” obiektami klasy Product, zapisywać i odczytywać je z bazy danych etc. W tym celu:
  1. Usuujemy z niej domyślny konstruktor (jeśli jest)
  2. Klasa musi dziedziczyć po DbContext (operator dziedziczenia to :).
  3. Klasa po której chcesz dziedziczyć nie będzie rozpoznawana. Żeby to rozwiązać potrzebujemy dodać EntityFramework do projektu. Zrobimy to z command-line’a /terminala. Otwieramy terminal systemowy, przechodzimy do katalogu w którym znajduje się projekt (miejsce w którym znajduje się projekt .csproj)

```

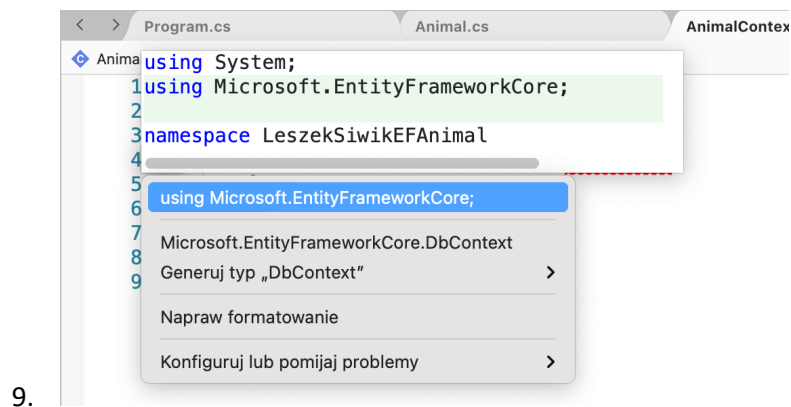
(base) MacBook-Pro-macbook:~ macbookpro$ cd Projects/LeszekSiwikEFProducts/LeszekSiwikEFProducts
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ ls -al
total 24
drwxr-xr-x  7 macbookpro  staff  224 18 lis 09:25 .
drwxr-xr-x  5 macbookpro  staff  160 18 lis 09:19 ..
-rw-r--r--  1 macbookpro  staff  178 18 lis 09:19 LeszekSiwikEFProducts.csproj
-rw-r--r--  1 macbookpro  staff  135 18 lis 09:25 Product.cs
-rw-r--r--  1 macbookpro  staff  203 18 lis 09:19 Program.cs
drwxr-xr-x  3 macbookpro  staff   96 18 lis 09:19 bin
drwxr-xr-x  8 macbookpro  staff  256 18 lis 09:20 obj
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$

```

- 4.
5. I wykonujemy polecenie:
  - a. dotnet add package Microsoft.EntityFrameworkCore
6. Potwierdzeniem że się powiodło jest obecność tego pakietu w zależnościach projektu:



- 7.
8. Niestety jak widać to nie wystarczy do tego żeby klasa DbContext po której chcemy dziedziczyć zaczęła być rozpoznawana. Ale teraz w menu kontekstowym pojawia się możliwość „dociągnięcia” do projektu odpowiedniego „usinga”:



9.

10. Po dołączeniu tego usinga problem z „widocznością” klasy DbContext powinien zostać rozwiązany i możemy przystąpić do implementacji klasy ProductContext. Klasa ta powinna zawierać publiczne property kolekcji obiektów którymi będzie zarządzać. Kolekcje powinny być kolekcjami typu DbSet (czyli w naszym przypadku DbSet<Product> Products

```
namespace LeszekSiwikEFProducts
{
    public class ProductContext:DbContext
    {
        public DbSet<Product> Products { get; set; }
    }
}
```

11.

12. Żeby model został odzwierciedlony w bazie danych trzeba:
- Przygotować kod odpowiedzialny za migracje modelu
  - Wykonać update struktury bazy danych na podstawie modelu (a w zasadzie kodu odpowiedzialnego za migracje modelu)
13. Żeby przygotować kod odpowiedzialny za migracje idziemy do miejsca gdzie mamy źródła projektu (dokładnie tam gdzie instalowaliśmy pakiet EntityFrameworkCore i wykonujemy polecenia:
- dotnet ef migrations add InitProductDatabase
  - Jeżeli powyższe zgłasza błąd o nierozpoznawaniu opcji dotnet ef doinstaluj (najlepiej globalnie) entity framework tools poleceniem:

**dotnet tool install --global dotnet-ef**

- Przy pierwszej próbie dostaniesz prawdopodobnie następujący błąd:

```
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ dotnet ef migrations add InitProductDatabase
Build started...
Build succeeded.
Your startup project 'LeszekSiwikEFProducts' doesn't reference Microsoft.EntityFrameworkCore.Design. This package is required for the Entity Framework Core Tools to work. Ensure your startup project is correct, install the package, and try again.
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$
```

- No to zgodnie z treścią błędu doinstalowujemy do projektu pakiet Microsoft.EntityFrameworkCore.Design czyli wykonujemy polecenie:

dotnet add package Microsoft.EntityFrameworkCore.Design

a następnie powtarzamy polecenie

dotnet ef migrations add InitProductDatabase

e. No to aktualnie dostaniemy prawdopodobnie błąd:

```
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.AddMigration.<c__DisplayClass0_0.<.ctor>b__0()
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.<c__DisplayClass3_0'1.<.Execute>b__0()
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.Execute(Action action)
No database provider has been configured for this DbContext. A provider can be configured by overriding the 'DbContext.OnConfiguring' method or by using 'AddDbContext' on the application service provider. If 'AddDbContext' is used, then also ensure that your DbContext type accepts a DbContextOptions<TContext> object in its constructor and passes it to the base constructor for DbContext.
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$
```

- i. No I to prawda – nigdzie w projekcie nie definiowaliśmy z jakiej bazy chcemy korzystać (nie tylko jak się ma nazywać, gdzie jest zlokalizowana tylko w ogóle z jakiego typu db chcemy korzystać (MSQL, SQLite etc)
- ii. No to skonfigurujmy nasz kontekst, żeby wiedział do jakiej bazy chcemy się łączyć. Jednym ze sposobów jest nadpisanie w klasie naszego kontekstu metody OnConfiguring. Załóżmy, że chcemy skorzystać z bazy SQLite o nazwie ProductsDatabase.db, wówczas metoda OnConfiguring powinna wyglądać następująco:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    base.OnConfiguring(optionsBuilder);
    optionsBuilder.UseSqlite("DataSource=ProductsDatabase");
}
```

iii. Czyli aktualnie moja klasa kontekstowa wygląda następująco:

```
namespace LeszekSiwikEFProducts
{
    public class ProductContext:DbContext
    {
        public DbSet<Product> Products { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            base.OnConfiguring(optionsBuilder);
            optionsBuilder.UseSqlite("DataSource=ProductsDatabase");
        }
    }
}
```

- iv. Jak widać, póki co SQLite nie jest rozpoznawany. Żeby tak się stało, musimy dodać do projektu pakiet Microsoft.EntityFrameworkCore.Sqlite. Dodajemy go do projektu dokładnie tak jak robiliśmy to poprzednio. Czyli:

```
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.AddMigration.<c__DisplayClass0_0.<.ctor>b__0()
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.<c__DisplayClass3_0'1.<.Execute>b__0()
at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.Execute(Action action)
No database provider has been configured for this DbContext. A provider can be configured by overriding the 'DbContext.OnConfiguring' method or by using 'AddDbContext' on the application service provider. If 'AddDbContext' is used, then also ensure that your DbContext type accepts a DbContextOptions<TContext> object in its constructor and passes it to the base constructor for DbContext.
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

- v. Po dodaniu, providera bazy danych Sqlite metoda UseSqlite powinna być już widoczna, a operacja dodania migracji powinna już przejść bez problemów. W efekcie w projekcie powinien pojawić się podkatalog Migrations z definicjami naszych migracji:



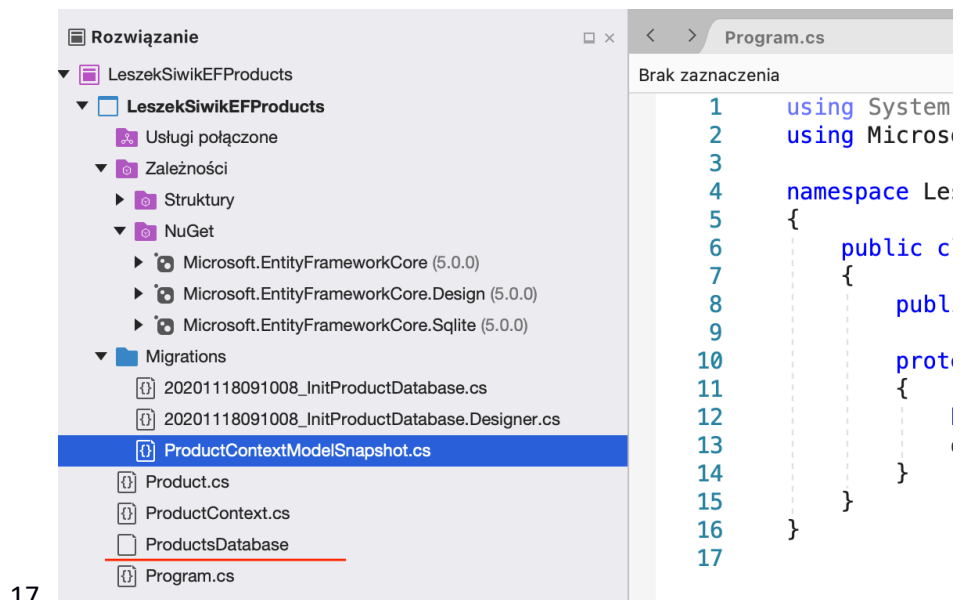
14. No to kolej na operacje updatu bazy danych na podstawie naszych migracji. A zatem wracamy do terminala i wykonujemy polecenie:

**dotnet ef database update.**

15. Efekt powinien być jak poniżej:

```
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ dotnet ef database update
Build started...
Build succeeded.
Applying migration '20201118091008_InitProductDatabase'.
Done.
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$
```

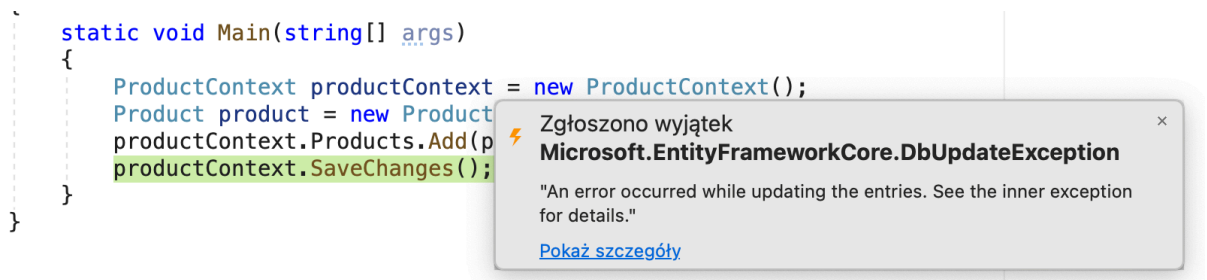
16. A w efekcie, w strukturze plików projektu powinien pojawić się plik bazy danych o nazwie takiej jak podaliśmy konfigurując providera bazy (jeśli trzymaliśmy się instrukcji to podawaliśmy tam nazwę ProductsDatabase.db



17.

18. Spróbujemy teraz napisać fragment kodu który będzie odpowiedzialny za dodanie produktu do bazy a następnie za pobranie wszystkich danych o produktach i wyświetlenie ich w konsoli

19. A zatem wędrujemy do funkcji Main w pliku Program.cs. Usuwamy linijkę odpowiedzialną za wydruk napisu HelloWorlds i dodajemy następujące linie kodu:
20. Tworzymy instancję ProductContextu:  
**ProductContext productContext = new ProductContext();**
21. Tworzymy instancję produktu – niech się nazywa Flamaster:  
**Product product = new Product { ProductName = "Flamaster" };**
22. Dodajemy nasz flamaster do kolekcji produktów w productContextcie:  
**productContext.Products.Add(product);**
23. Zapisujemy zmiany w kontekście:  
**productContext.SaveChanges();**
24. Zbudujmy i uruchommy projekt. Najprawdopodobniej dostaniesz wyjątek:



25. Jak popatrzymy w szczegóły to zobaczymy że nie ma tabeli Products:

ErrorCode	ErrorCode	...
HResult	-2147467259	int
HelpLink	(null)	string
InnerException	(null)	System.Exception
Message	"SQLite Error 1: 'no such table: Products!'"	string
Source	"Microsoft.Data.Sqlite"	string
SQLiteErrorCode	1	int
SQLiteExtendedErrorCode	1	int

26. Ale jednocześnie jak podejmiemy się dowolnym klientem pod plik ProductsDatabase to zobaczymy że:

```
[(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ sqlite3 ProductsDatabase
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
[sqlite> .tables
Products          __EFMigrationsHistory
sqlite> ]
```

27. Czyli tabela Products istnieje. Co więcej jak zobaczymy jej strukturę to jest ona taka jakbyśmy oczekiwali:

```
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ sqlite3 ProductsDatabase
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
[sqlite> .tables
Products          __EFMigrationsHistory
[sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
  "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
  "ProductName" TEXT NULL,
  "UnitsOnStock" INTEGER NOT NULL
);
[sqlite> PRAGMA table_info(Products)
[ ...> ;
0|ProductID|INTEGER|1||1
1|ProductName|TEXT|0||0
2|UnitsOnStock|INTEGER|1||0
sqlite> █
```

28. W czym więc rzecz. Otóż w momencie uruchomienia aplikacja korzysta nie z tego pliku ProductsDatabase który widzimy w źródłach projektu ale z pliku który „siedzi” w podkatalogu w którym siedzą pliki wykonywalne aplikacji. Czyli w podkatalogu bin/Debug/netcoreapp3.1/

```
(base) MacBook-Pro-macbook:LeszekSiwikEFProducts macbookpro$ cd bin/Debug/netcoreapp3.1/
(base) MacBook-Pro-macbook:netcoreapp3.1 macbookpro$ ls -al
total 10856
drwxr-xr-x  36 macbookpro  staff   1152 18 lis 10:19 .
drwxr-xr-x   3 macbookpro  staff    96 18 lis 09:19 ..
-rwxr-xr-x   1 macbookpro  staff 256952 2 cze 13:07 Humanizer.dll
-rw-r--r--   1 macbookpro  staff 23857 18 lis 10:11 LeszekSiwikEFProducts.deps.json
-rw-r--r--   1 macbookpro  staff 11776 18 lis 10:19 LeszekSiwikEFProducts.dll
-rw-r--r--   1 macbookpro  staff 2176 18 lis 10:19 LeszekSiwikEFProducts.pdb
-rw-r--r--   1 macbookpro  staff 163 18 lis 10:09 LeszekSiwikEFProducts.runtimeconfig.dev.json
-rw-r--r--   1 macbookpro  staff 146 18 lis 10:09 LeszekSiwikEFProducts.runtimeconfig.json
-rwxr-xr-x   1 macbookpro  staff 151432 24 paź 06:39 Microsoft.Data.Sqlite.dll
-rwxr-xr-x   1 macbookpro  staff 23424 17 cze 04:37 Microsoft.DotNet.PlatformAbstractions.dll
-rwxr-xr-x   1 macbookpro  staff 28552 24 paź 06:39 Microsoft.EntityFrameworkCore.Abstractions.dll
-rwxr-xr-x   1 macbookpro  staff 298376 24 paź 06:40 Microsoft.EntityFrameworkCore.Design.dll
-rwxr-xr-x   1 macbookpro  staff 1252232 24 paź 06:40 Microsoft.EntityFrameworkCore.Relational.dll
-rwxr-xr-x   1 macbookpro  staff 182144 24 paź 06:40 Microsoft.EntityFrameworkCore.Sqlite.dll
-rwxr-xr-x   1 macbookpro  staff 1822600 24 paź 06:40 Microsoft.EntityFrameworkCore.dll
-rwxr-xr-x   1 macbookpro  staff 26504 19 paź 18:48 Microsoft.Extensions.Caching.Abstractions.dll
-rwxr-xr-x   1 macbookpro  staff 32648 19 paź 18:48 Microsoft.Extensions.Caching.Memory.dll
-rwxr-xr-x   1 macbookpro  staff 20360 19 paź 18:48 Microsoft.Extensions.Configuration.Abstractions.dll
-rwxr-xr-x   1 macbookpro  staff 44416 19 paź 18:48 Microsoft.Extensions.DependencyInjection.Abstractions.dll
-rwxr-xr-x   1 macbookpro  staff 75648 19 paź 18:48 Microsoft.Extensions.DependencyInjection.dll
-rwxr-xr-x   1 macbookpro  staff 69000 19 paź 18:49 Microsoft.Extensions.DependencyModel.dll
-rwxr-xr-x   1 macbookpro  staff 52616 19 paź 18:48 Microsoft.Extensions.Logging.Abstractions.dll
-rwxr-xr-x   1 macbookpro  staff 42376 19 paź 18:53 Microsoft.Extensions.Logging.dll
-rwxr-xr-x   1 macbookpro  staff 55176 19 paź 18:48 Microsoft.Extensions.Options.dll
-rwxr-xr-x   1 macbookpro  staff 39296 19 paź 18:48 Microsoft.Extensions.Primitives.dll
-rw-r--r--   1 macbookpro  staff    0 18 lis 10:19 ProductsDatabase
-rwxr-xr-x   1 macbookpro  staff 6144 3 wrz 17:13 SQLitePCLRaw.Batteries_v2.dll
-rwxr-xr-x   1 macbookpro  staff 46592 3 wrz 17:13 SQLitePCLRaw.core.dll
-rwxr-xr-x   1 macbookpro  staff 5632 3 wrz 17:13 SQLitePCLRaw.nativeLibrary.dll
-rwxr-xr-x   1 macbookpro  staff 57844 3 wrz 17:13 SQLitePCLRaw.provider.dynamic_cdecl.dll
-rwxr-xr-x   1 macbookpro  staff 180912 19 paź 18:37 System.Collections.Immutable.dll
-rwxr-xr-x   1 macbookpro  staff 80992 19 paź 18:37 System.ComponentModel.Annotations.dll
-rwxr-xr-x   1 macbookpro  staff 83336 19 paź 18:39 System.Diagnostics.DiagnosticSource.dll
-rwxr-xr-x   1 macbookpro  staff 168328 19 paź 18:46 System.Text.Encoding.Web.dll
-rwxr-xr-x   1 macbookpro  staff 351624 19 paź 18:47 System.Text.Json.dll
drwxr-xr-x  15 macbookpro  staff   480 18 lis 10:09 runtimes
(base) MacBook-Pro-macbook:netcoreapp3.1 macbookpro$ █
```

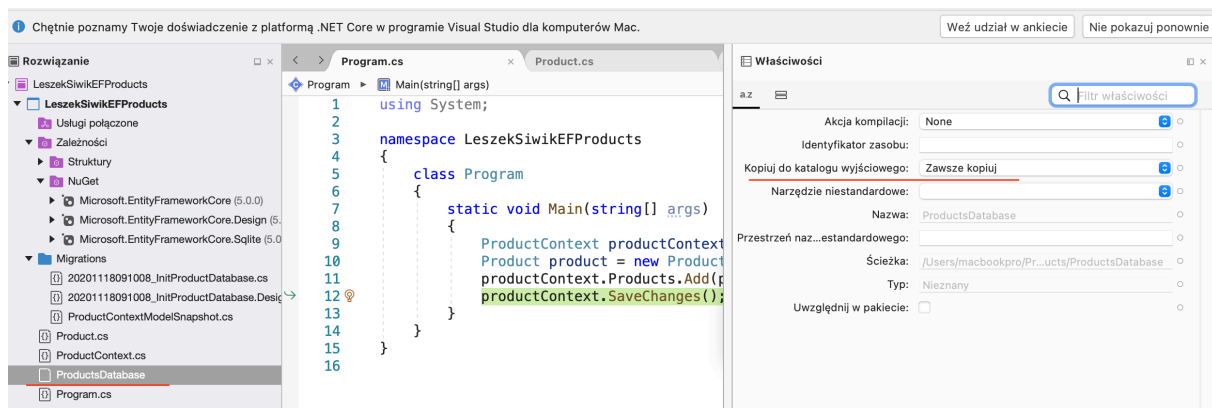
29.

30. A jak podepnijemy się „klientem” do tego pliku to zobaczymy że rzeczywiście nie ma tam żadnych tabel:

```
/Users/macbookpro/Projects/LeszekSiwikEFProducts/LeszekSiwikEFProducts/bin/Debug/netcoreapp3.1
(base) MacBook-Pro-macbook:netcoreapp3.1 macbookpro$ sqlite3 ProductsDatabase
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
[sqlite> .tables
sqlite> █
```

31. Żeby to rozwiązać robimy „right-click” na pliku ProductsDatabase w VisualStudio, wybieramy właściwości, i dla opcji „Kopiuje do katalogu wyjściowego” wybieramy opcję „Zawsze Kopiuje”





32. I teraz jeśli zbudujemy i uruchomimy projekt to po pierwsze nie dostaniemy żadnych wyjątków. A po drugie jeśli podejmiemy się do pliku ProductsDatabase (tego znajdującego się w katalogu z plikami wykonywalnymi) to zobaczymy że w bazie są tabele (w szczególności tabela Products) i że nasz Flamaster został dodany do bazy.

```

/Users/macbookpro/Projects/LeszekSiwikEFProducts/LeszekSiwikEFProducts/bin/Debug/netcoreapp3.1
(base) MacBook-Pro-macbook:netcoreapp3.1 macbookpro$ sqlite3 ProductsDatabase
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
sqlite> .tables
sqlite> .tables
Products          __EFMigrationsHistory
sqlite> select * from Products;
1|Flamaster|0
sqlite>

```

33. To napiszmy fragment kodu który pobierze wszystkie produkty z bazy i wypisze je na konsoli. Idziemy zatem do naszego Main'a i pod kodem który pisaliśmy wcześniej dopisujemy:

34. Zapytanie Linq'owe którego zadaniem jest pobranie nazw wszystkich produktów:

```

var query = from prod in productContext.Products
            select prod.ProductName;

```

35. Jeśli dostaniesz błąd mówiący że nie znaleziono elementu „Select” trzeba „dociągnąć” do projektu namespace System.Linq (czyli na górze pliku dodajemy: **using System.Linq;** )

36. Następnie przechodzimy w pętli foreach po wynikach zapytania i drukujemy na konsoli „imiona” zwierzątek.

```

foreach (var pName in query)
{
    Console.WriteLine(pName);
}

```

37. Czyli w całości mój main wygląda aktualnie następująco:



```

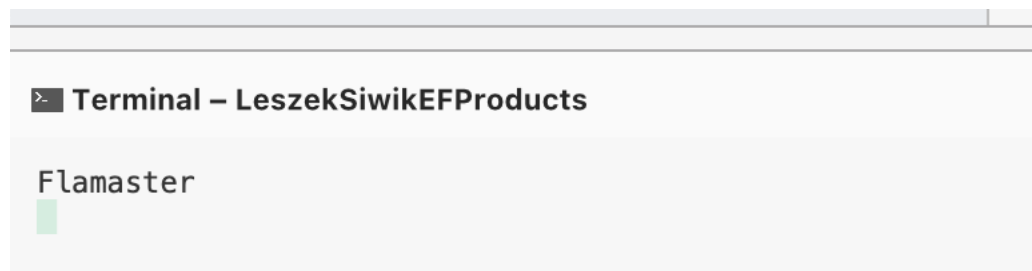
namespace LeszekSiwikEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();
            Product product = new Product { ProductName = "Flamaster" };
            productContext.Products.Add(product);
            productContext.SaveChanges();

            var query = from prod in productContext.Products
                        select prod.ProductName;

            foreach (var pName in query)
            {
                Console.WriteLine(pName);
            }
        }
    }
}

```

38. Zbudujmy i uruchommy aplikacje. Oczekiwany efekt:



```

> Terminal - LeszekSiwikEFProducts

Flamaster

```

39. Szatu może nie ma, ale wydruk potwierdza, że rozwiązanie działa i że jesteśmy w stanie wstawić i pobrać dane z bazy danych.
40. Uważnym czytelnikom dziwne może się wydać dlaczego na wydruku mamy jeden produkt, podczas gdy uruchamiamy aplikację już po raz kolejny, a przy każdym uruchomieniu wykonywany jest m.in. kod odpowiedzialny za dodanie naszego flamastra do bazy. Więć „na logike” na wydruku tych Flamastrów powinno być więcej. Wynika to z tego, że we właściwościach pliku bazodanowego ProductsDatabase ustawiliśmy żeby plik, który mamy w źródłach projektu był za każdym razem kopiowany do katalogu z plikami wykonywalnymi, a więc przy każdym uruchomieniu nadpisujemy plik bazy danych i tracimy zawartość, którą już tam potencjalnie mieliśmy. Proponuje zmienić ten mechanizm kopiowania z „Zawsze kopiuj” na „Kopiuj jeśli nowszy”. Dzięki temu plik w katalogu „wykonywalnym” zostanie nadpisany tylko jeśli zmieni się struktura bazy danych w plikach źródłowych. Po tej zmianie i ponownym uruchomieniu powinniśmy już zobaczyć na wydruku dwa Flamastry, przy kolejnym uruchomieniu trzy itd.

**Terminal – LeszekSiwikEFProducts**

```
Flamaster  
Flamaster  
Flamaster
```

41. Żeby móc łatwiej rozróżniać te nasze produkty, proponuje fragment kodu odpowiedzialny za wstawianie produktu do bazy zmienić w taki sposób, żeby zamiast wstawiania za każdym razem produktu z wpisaną na sztywno nazwą, zapytać użytkownika o nazwę wstawianego produktu. (Zczytanie wartości ze standardowego wejścia (z konsoli) to metoda `Console.ReadLine()`). Reasumując mój main wygląda aktualnie następująco:

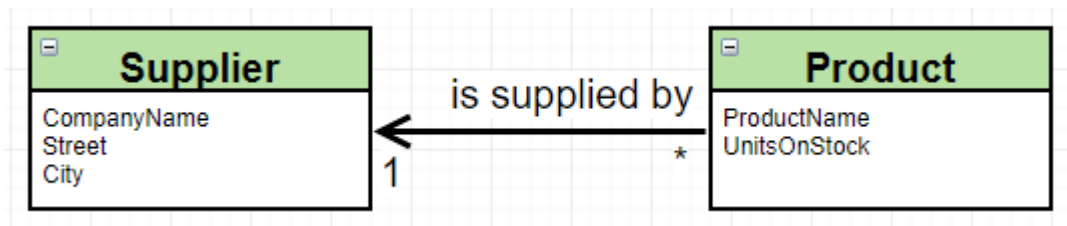
```
namespace LeszekSiwikEFProducts  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Podaj nazwę produktu");  
            string prodName = Console.ReadLine();  
  
            Console.WriteLine("Poniżej lista produktów zarejestrowanych w naszej bazie danych");  
            ProductContext productContext = new ProductContext();  
            Product product = new Product { ProductName = prodName };  
            productContext.Products.Add(product);  
            productContext.SaveChanges();  
  
            var query = from prod in productContext.Products  
                        select prod.ProductName;  
  
            foreach (var pName in query)  
            {  
                Console.WriteLine(pName);  
            }  
        }  
    }  
}
```

42. A działanie jak poniżej:

**Terminal – LeszekSiwikEFProducts**

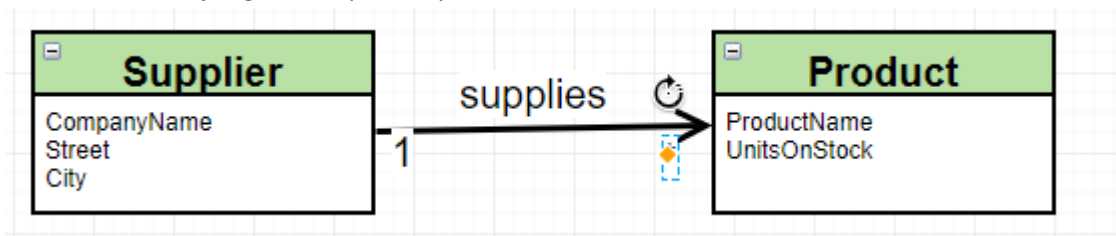
```
Podaj nazwę produktu  
Nozyczki  
Poniżej lista produktów zarejestrowanych w naszej bazie danych  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Zeszyt  
Nozyczki
```

43. Od tego momentu pracujemy bardziej samodzielnie.  
44. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



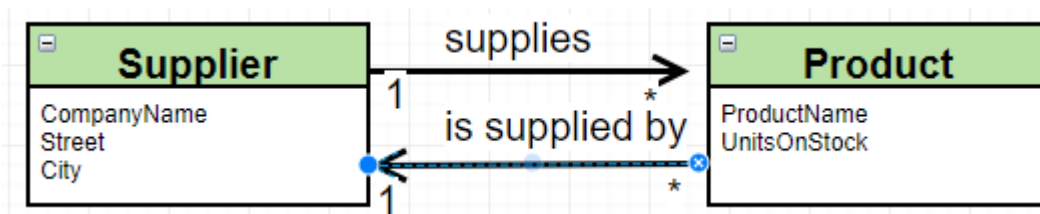
- Stworz nowego dostawce.
- Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego.
- Udokumentuj wykonane kroki oraz uzyskany rezultat (.schema table/diagram z datagrip, select \* from....)

II. Odwróć relacje zgodnie z poniższym schematem



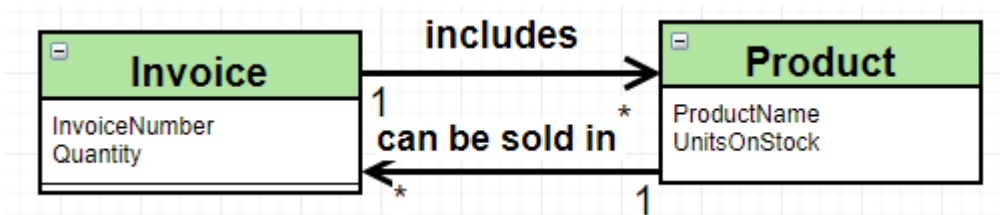
- Stworz kilka produktow
- Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select \* from....)**

III. Zamodeluj relacje dwustronną jak poniżej:



- Tradycyjnie: Stworz kilka produktow
- Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select \* from....)**

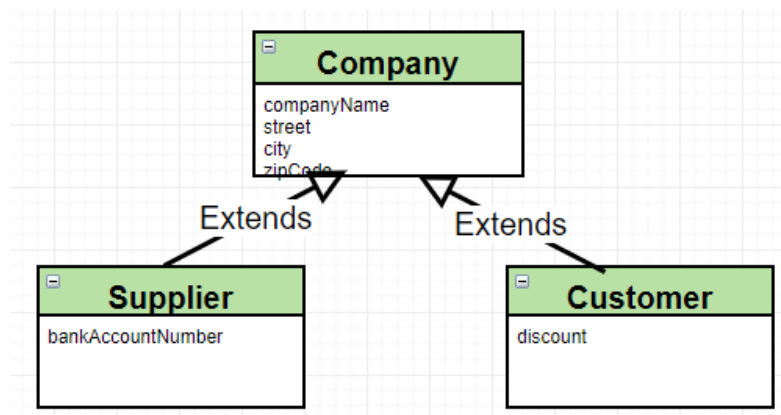
IV. Zamodeluj relacje wiele-do-wielu, jak poniżej:



- Stórz kilka produktów i "sprzedaj" je na kilku transakcjach.
- Pokaż produkty sprzedane w ramach wybranej faktury/transakcji
- Pokaż faktury w ramach których był sprzedany wybrany produkt
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select \* from....)**

#### V. Dziedziczenie

- Wprowadź do modelu poniższą hierarchie dziedziczenia używając strategii Table-Per-Hierarchy:



- Dodaj i pobierz z bazy kilka firm obu rodzajów stosując po kolei trzy różne strategie mapowania dziedziczenia.
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select \* from....)**

#### VI. Zamodeluj tę samą hierarchie dziedziczenia, ale tym razem użyj strategii Table-Per-Type

- Dodaj i pobierz z bazy kilka firm obu rodzajów
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select \* from....)**
- Porównaj obie strategie modelowania dziedziczenia