EntityFramework, LINQ2Entities – laboratorium

a. **Z uwagi na tryb prowadzenia zajęć** na bieżąco dokumentuj wykonywane operacje w dokumencie INazwiskoMirroringLab1. Na koniec zajęć – niezależnie od etapu na którym będziesz wrzuć to sprawozdanie do zadania Dokumentacja-NaKoniecZajęć na moodlu.

I. Code First

a. Stwórz projekt typu ConsoleApplication .Net Core. Nazwij go INazwiskoProdutcEF

b. Dodaj klasę Product z polami int ProductID, string Name, int UnitsInStock. (Dodając do klasy property napisz prop I naciśnij dwa razy tabulator)

c. Stwórz klasę ProdContext dziedziczącą po DbContext. Klasa po której chcesz dziedziczyć nie będzie rozpoznawana. Żeby to rozwiązać musisz dodać do projektu pakiet EntityFrameworkCore (Project -> Manage NuGet Packages) lub z command-line'a przejdz do katalogu w ktorym masz project I wykonaj polecenie:

   dotnet add package Microsoft.EntityFrameworkCore

   Następnie dodaj w klasie using Microsoft.EntityFrameworkCore

   Od tego momentu klasa DbContext powinna być rozpoznawana.

d. Dodaj do klasy kontekstowej zbiór (DbSet) produktów i nazwij go Products.

e. W Mainie (plik Program.cs)

   i. poproś użytkownika o podanie nazwy produktu i zczytaj podana przez użytkownika nazwę (do pisania / czytania na/z konsoli: Console.WriteLine() Console.ReadLine())

   ii. zainstancjonuj obiekt produktu ustawiając mu nazwę na tą zczytaną od użytkownika: Product prod = new Product { ProductName = prodName } ;

   iii. Stwórz instancje ProdContext'u

   iv. dodaj zainstancjonowany obiekt do kontekstowej kolekcji Produktów (metoda add na kolekcji produktów kontekstu)

   v. zapisz zmiany na kontekście (metoda SaveChanges na kontekście)

   vi. Zbuduj i uruchom aplikacje

   vii. Dostaniesz wyjątek: No database provider has been configured. 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)

   viii. 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

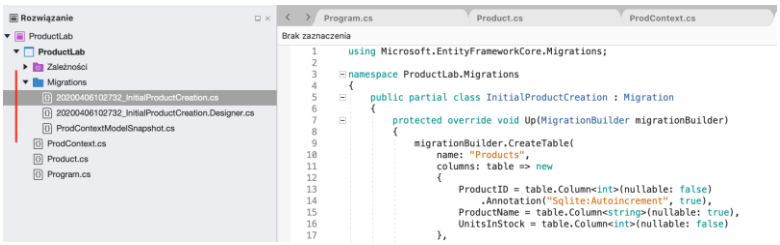
nazwie Product.db, wówczas metoda OnConfiguring powinna wyglądać następująco:

```
protected override void OnConfiguring(DbContextOptionsBuilder options) => options.UseSqlite("DataSource=Product.db");
```

Jak widać, póki co SQLite nie jest rozpoznawany. Żeby tak się stało, musimy dodać do projektu pakiet Microsoft.EntityFrameworkCore.Sqlite. Dodaj go do projektu tak jak robiliśmy to poprzednio. Po dodaniu, provider bazy danych Sqlite (a zatem metoda UseSqlite powinna być już widoczna.

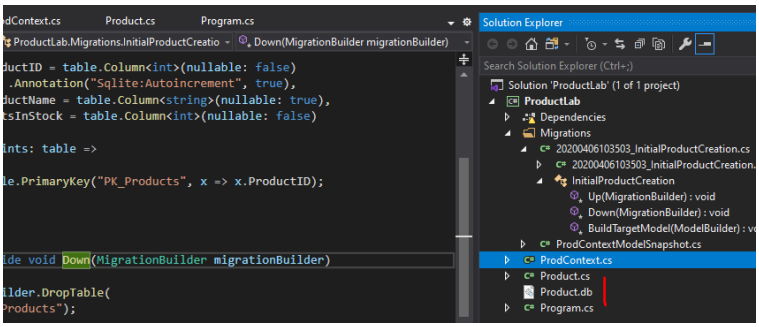
ix. Zbuduj i uruchom aplikacje.

x. Dostaniesz wyjątek mówiący o tym, że nie istnieje tabela produktów. No i to prawda – bo generalnie w ogóle nie istnieje baza danych z którą chcemy pracować. Żeby to rozwiązać musimy wykonać dwa kroki

1. Stworzyć migracje modelu: dotnet ef migrations add InitialProductCreation
    a. Jeśli powyższe zgłasza błąd o braku pakietu Microsoft.EntityFrameworkCore.Design – dodaj go do projektu tak jak robiliśmy to poprzednio
    b. Jeżeli powyższe zgłasza błąd o nieropoznawaniu opcji dotnet ef doinstaluj (najlepiej globalnie) entity framework toolsy poleceniem: dotnet tool install –global dotnet-ef
    c. Po wykonaniu migracji, w projekcie powinien pojawić się folder z kodem definiującym kolejne migracje modelu (na razie naszą początkową migracje)
    d. 

    e.

2. Odzwierciedlić zmiany w modelu definiowane przez kolejne migracje (na razie naszą początkową migrację) w bazie danych, czyli wykonać polecenie: dotnet ef database update
3. W tym momencie w plikach projektu powinien pojawić się plik sqlitowej bazy Product.db



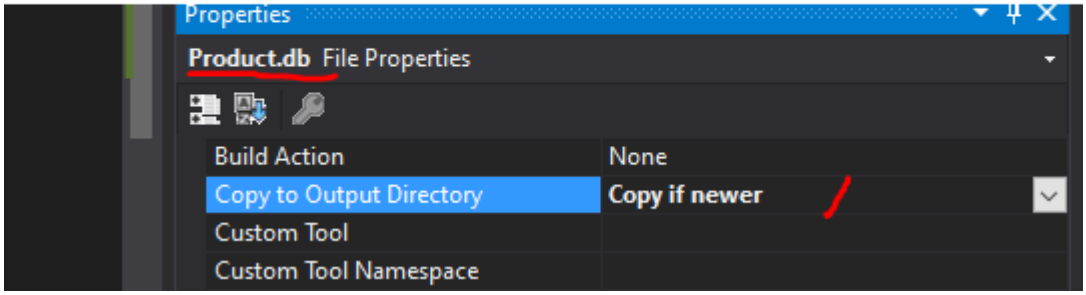
f. Dopisz w mainie fragment kodu pobierający oraz wyświetlający dostępne Produkty. Jeżeli przy pisaniu zapytania o produkty zgłasza błąd typu „polecenie select jest nieznane” dodaj do usingów System.Linq

g. Zbuduj, uruchom i przetestuj aplikacje:

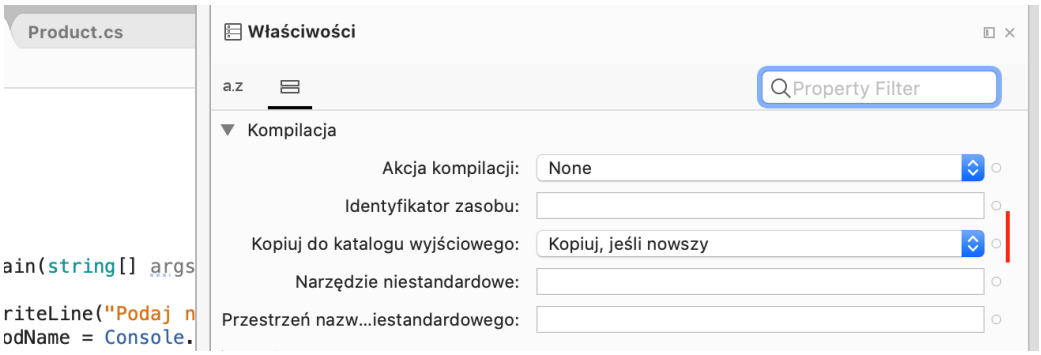
h. Ponownie dostaniesz wyjątek o braku tabeli Products. Mimo że plik bazy w projekcie istnieje (sam go tam EF dodał) i co więcej mimo ze tabela Produktów tam jest – jeżeli podepniesz się do tego pliku choćby commandlinowym narzędziem sqlite3 (lub dowolną inna przeglądarką plików sqlitowych) to ta tabela tam będzie:

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

j. Chodzi o to, że plik bazy co prawda jest dodany do projektu, ale on nie jest z automatu kopiowany do miejsca gdzie są wrzucane pliki wykonywalne aplikacji w momencie jej uruchamiania. Żeby to rozwiązać mamy dwie możliwości – albo w konfiguracji providera podać pełną scieżke do pliku Product.db albo prawy klik na tym pliku w Visual Studio - > Properties i zmieniamy opcję Copy To Output Directory z Do not copy Copy if newer

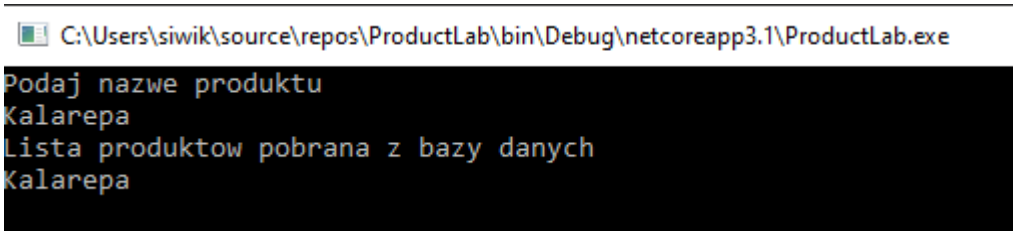
k.


l.



m. Po tej zmianie aplikacja powinna już zacząć działać zgodnie z oczekiwaniami. Tzn po zczytaniu nazwy - produkt powinien dodać się do bazy danych a następnei lista zapisanych produktów powinna zostać wypisana na konsoli

n.



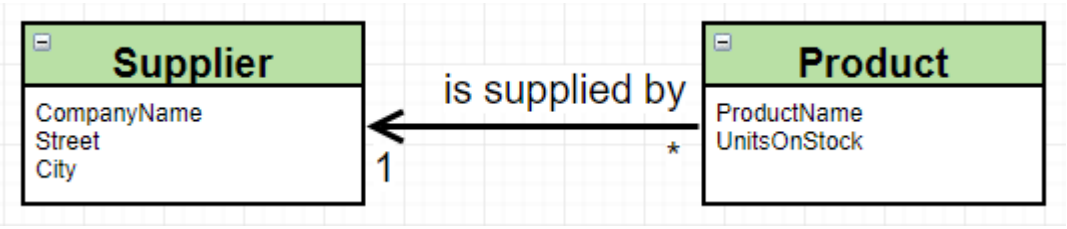
o. Oczywiście po podpięciu się do tego pliku bazy danych jakims „klientem” te produkty powinny być tam widoczne

p.

```
(base) MacBook-Pro-macbook:netcoreapp3.1 macbookpro$ sqlite3 Product.db
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
sqlite> select * from Products;
1|Marchew|0
2|Kapusta|0
sqlite>
```

q. Udokumentuj wykonane kroki oraz uzyskany rezultat (logi wywołań sqlowych,describe table/diagram z datagrip, select * from….)

II. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



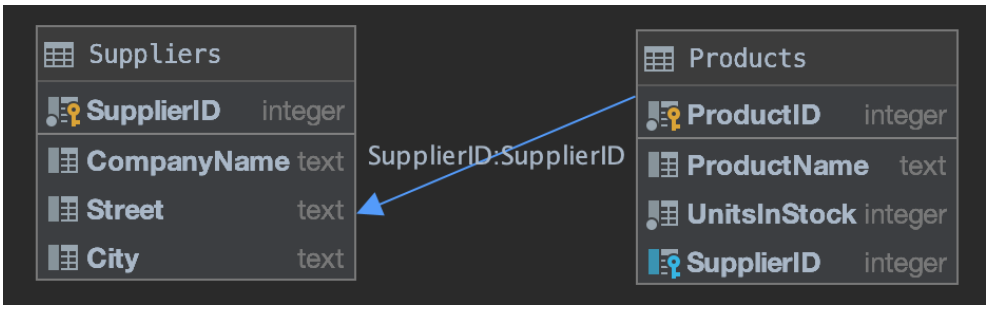
a. Jeśli nadziejesz się na błąd mówiący o tym że sqlite provider nie wspiera operacji typu AddForeignKey….. usuń plik z baza, i katalog z migracjami w projekcie i wykonaj migracje i update bazy od nowa.

b. Efekt który powinieneś uzyskać to utworzona tabel Produktów z kluczem obcym do Dostawcy:

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers" ("SupplierID") ON DELETE RESTRICT
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

c.

a w wersji „diagramowe”:



d. Dodaj do bazy nowy Produkt

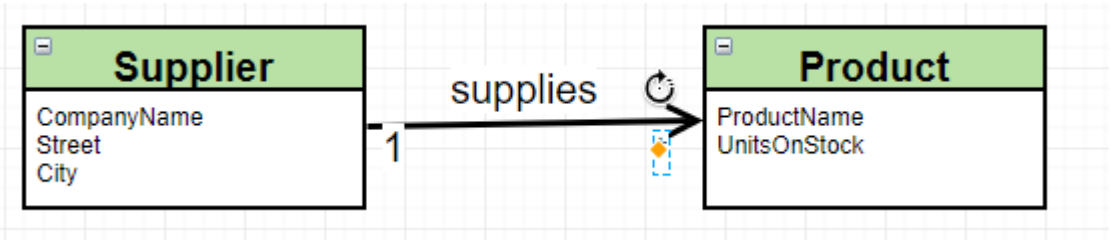
e. Stworz nowego dostawce.

f. Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego.

g. Wyświetl wszystkie produkty wraz z nazwą dostawcy

h. Udokumentuj wykonane kroki oraz uzyskany rezultat (ogi wywołań sqlowych,describe table/diagram z datagrip, select * from….)

III. Odwróć relacje zgodnie z poniższym schematem

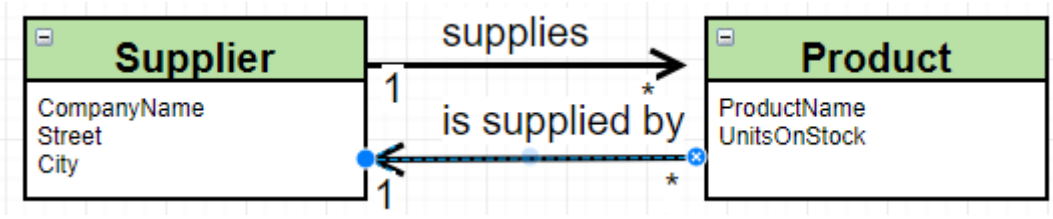


a. Stworz kilka produktow

b. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę

c. Udokumentuj wykonane kroki oraz uzyskane rezultaty w obu wariantach (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)

IV. Zamodeluj relacje dwustronną jak poniżej:



a. Tradycyjnie: Stworz kilka produktow

b. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę (pamiętaj o poprawnej obsłudze dwustronności relacji)

c. Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)

V. Dodaj klase Category z property int CategoryID, String Name oraz listą produktow

a. Zmodyfikuj produkty dodając wskazanie na kategorie do której należy.

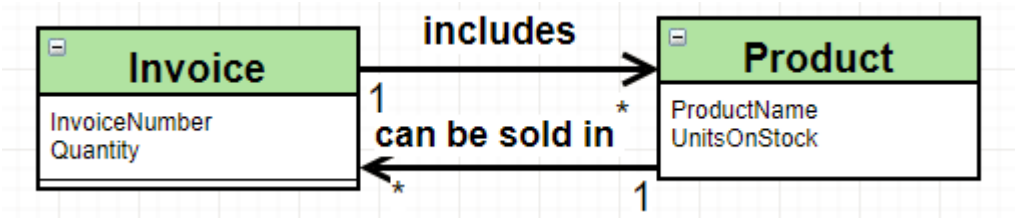
b. Stworz kilka produktow i kilka kategorii

c. Dodaj kilka produktów do wybranej kategorii

d. Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt

e. Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)

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



a. Stórz kilka produktów I “sprzedaj” je na kilku transakcjach.

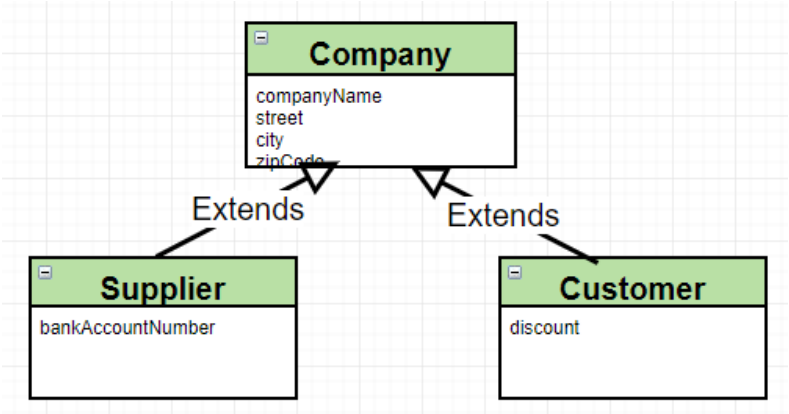
b. Pokaż produkty sprzedane w ramach wybranej faktury/transakcji

c. Pokaż faktury w ramach których był sprzedany wybrany produkt

d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

VII. Dziedziczenie

a. Wprowadź do modelu następującą hierarchie:



b. Dodaj i pobierz z bazy kilka firm obu rodzajów stosując po kolei trzy różne strategie mapowania dziedziczenia. TablePerHierarchy, TablePerType, TablePerClass – informacje znajdziesz np. tutaj: https://www.entityframeworktutorial.net/code-first/inheritance-strategy-in-code-first.aspx

c.

d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

VIII. Zadanie

a. Przygotuj aplikacje webowa korzystajaca z modelu przygotowanego podczas zajec w szczegolnosci wprowadzajac mozliwosc skladania zamowien na produkty.

b. Ocenie podlegać będzie:

   i. Zakres funkcjonalny (wg własnego pomysłu/inwencji)

   ii. Wykorzystanie mechanizmów EF

c. Przygotuj sprawozdanie omawiające przygotowaną aplikację, wykorzystane mechanizmy I sposób ich dzialania (wraz ze screenami).