

Wprowadzenie do EntityFramework.Core

Instrukcja zawiera dwie części

Część I – mocno „przewodnikowa” – aby pomóc Państwu dojść „do czegoś działającego”.

Część II – (od punktu II włącznie) to co stanowi przedmiot Państwa (bardziej) samodzielnej pracy (w tym zadania domowego jeśli nie uda się skończyć w trakcie zajęć).

Na koniec zajęć proszę o umieszczenie w moodlu screenshot’a pokazującego to co udało się zrealizować w trakcie zajęć

I. Część przewodnikowa:

a. Zweryfikujmy na początek jaką mamy do dyspozycji wersję dotnet frameworka.

Będzie nam to potrzebne na późniejszych etapach pracy

```
● siwik@fedora:~/Labs/LSiwikEFLab-2$ dotnet --version
7.0.118
○ siwik@fedora:~/Labs/LSiwikEFLab-2$
```

b. Uruchommy sobie jakiś edytor (kodu) (np. Visual Studio Code)

c. Stwórzmy katalog dla naszego rozwiązania/aplikacji - u mnie będzie to LSiwikEFLab.

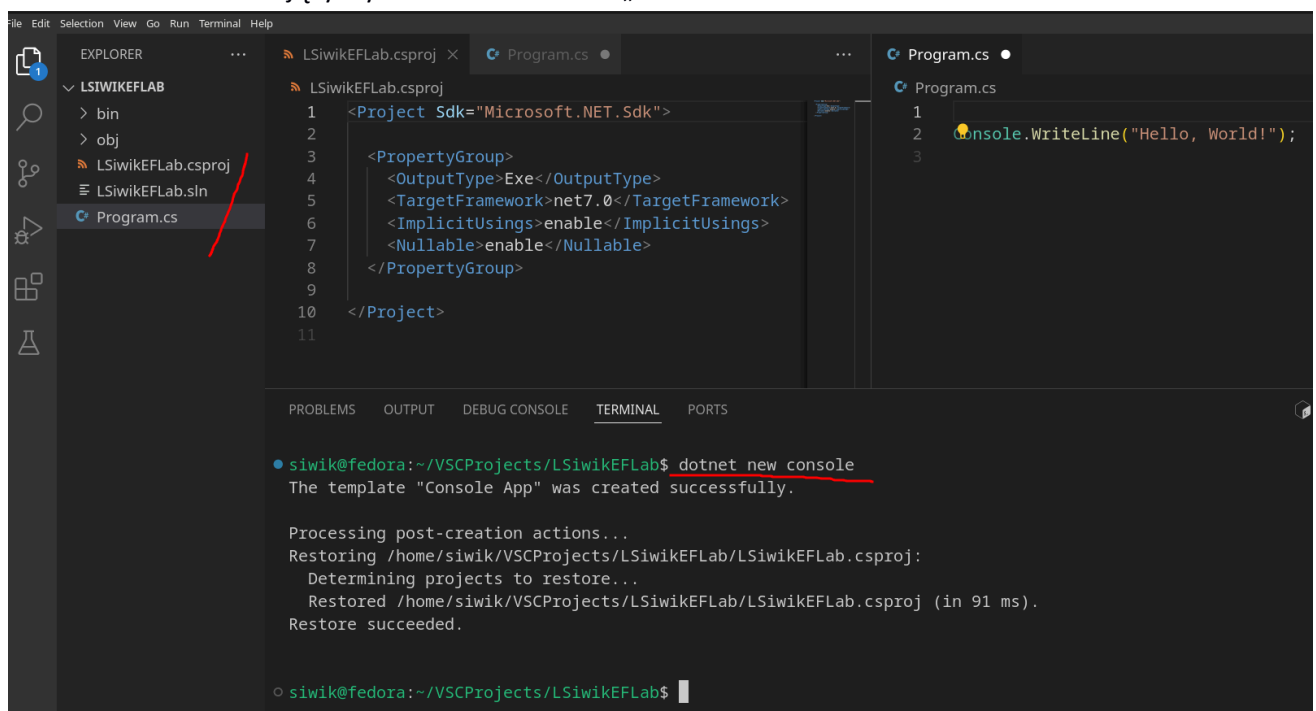
d. Otwórzmy sobie terminal i stwórzmy / zainicjujmy sobie w stworzonym katalogu .netową aplikację konsolową, czyli wykonujemy:

dotnet new console

e. W rezultacie, w katalogu naszego rozwiązania powinien nam się stworzyć m.in.

“manifest” naszego projektu (plik .csproj) oraz przykładowy plik Program.cs

zawierający wywołanie drukowania „Hello World” na konsoli:



The screenshot shows the Visual Studio Code interface with the LSiwikEFLab project open. The Explorer pane on the left shows the project structure: bin, obj, LSiwikEFLab.csproj, LSiwikEFLab.sln, and Program.cs. The main editor area shows the LSiwikEFLab.csproj file with the following content:

```
<?xml version='1.0' encoding='utf-8'>
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net7.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
</Project>
```

The Program.cs file shows the following content:

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

The terminal at the bottom shows the output of the 'dotnet new console' command:

```
● siwik@fedora:~/VSCProjects/LSiwikEFLab$ dotnet new console
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring /home/siwik/VSCProjects/LSiwikEFLab/LSiwikEFLab.csproj:
  Determining projects to restore...
  Restored /home/siwik/VSCProjects/LSiwikEFLab/LSiwikEFLab.csproj (in 91 ms).
Restore succeeded.

○ siwik@fedora:~/VSCProjects/LSiwikEFLab$
```

f.

- g. No to spróbujmy ten program testowo zbudować:

```
• siwik@fedora:~/Labs/LSiwikEFLab$ dotnet build
MSBuild version 17.4.8+6918b863a for .NET
Determining projects to restore...
All projects are up-to-date for restore.
LSiwikEFLab -> /home/siwik/Labs/LSiwikEFLab/bin/Debug/net7.0/LSiwikEFLab.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.24
○ siwik@fedora:~/Labs/LSiwikEFLab$
```

- h.
- i. I uruchomić:

```
• siwik@fedora:~/Labs/LSiwikEFLab$ dotnet run
Hello, World!
○ siwik@fedora:~/Labs/LSiwikEFLab$
```

- j.
- k. No więc wygląda, że wszystko tak jak miało być, więc idźmy dalej.

- l. Dodajmy do projektu klasę Product (nowy plik Product.cs i w nim definicja klasy) i dodajemy do klasy produktu trzy publiczne property

```
public int ProductID
public String? ProductName
public int UnitsOnStock
```

(jeśli mamy doinstalowany do VSC extension C# DevKit lub piszemy w Visual Studio możesz trochę sobie pomóc pisząc prop + tab tab)

```
public class Product
{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }    public
int UnitsInStock { get; set; }
}
```

- m. Dodajmy do projektu klasę ProdContext, która będzie odpowiedzialna za zarządzanie obiektami persystentnymi w naszej aplikacji.

- n. A zatem, nowy plik prodContext.cs, i w nim klasa ProdContext, która po pierwsze musi być klasą dziedziczącą po EntityFramework'owym DbContext

```
public class ProdContext: DbContext{}
```

- o. Tutaj musimy wykonać trochę dodatkowej pracy konfiguracyjnej. Otóż jeśli aktualnie spróbujemy zbudować naszą aplikację dostaniemy błąd z informacją o braku referencji / nierozpoznawalności klasy DbContext.

```
Program.cs • Product.cs • ProdContext.cs x LSiwikEFLab.csproj
ProdContext.cs > ProdContext
0 references
1 public class ProdContext: DbContext{
2     ...
3 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash

❯ siwik@fedora:~/Labs/LSiwikEFLab$ dotnet build
MSBuild version 17.4.8+6918b863a for .NET
Determining projects to restore...
All projects are up-to-date for restore.
/home/siwik/Labs/LSiwikEFLab/ProdContext.cs(1,27): error CS0246: The type or namespace name '
not be found (are you missing a using directive or an assembly reference?) [/home/siwik/Labs/
kEFLab.csproj]

Build FAILED.

/home/siwik/Labs/LSiwikEFLab/ProdContext.cs(1,27): error CS0246: The type or namespace name '
not be found (are you missing a using directive or an assembly reference?) [/home/siwik/Labs/
kEFLab.csproj]
0 Warning(s)
1 Error(s)
```

- p.
- q. No to spróbujmy, zgodnie z sugestią / komunikatem błędu dodać using'a do entity frameworka i przebudować projekt:

```
Program.cs • Product.cs • ProdContext.cs x LSiwikEFLab.csproj
ProdContext.cs > ...
1 using Microsoft.EntityFrameworkCore;
0 references
2 public class ProdContext: DbContext{
3
4 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + v [ ] [ ] ... ^ x

❯ siwik@fedora:~/Labs/LSiwikEFLab$ dotnet build
MSBuild version 17.4.8+6918b863a for .NET
Determining projects to restore...
All projects are up-to-date for restore.
/home/siwik/Labs/LSiwikEFLab/ProdContext.cs(1,17): error CS0234: The type or namespace name 'EntityFrameworkCore'
does not exist in the namespace 'Microsoft' (are you missing an assembly reference?) [/home/siwik/Labs/LSi
wikEFLab/LSiwikEFLab.csproj]
/home/siwik/Labs/LSiwikEFLab/ProdContext.cs(2,27): error CS0246: The type or namespace name 'DbContext' could
not be found (are you missing a using directive or an assembly reference?) [/home/siwik/Labs/LSiwikEFLab/LSi
wikEFLab.csproj]

Build FAILED.
```

- r. Jak widać, samo dodanie dyrektywy using problemu nie rozwiązuje, a to dlatego że EntityFramework z którego pochodzi klasa DbContext nie jest dodawany “z marszu” do każdego projektu, bo i nie każdy projekt tego frameworka potrzebuje.
- s. Musimy zatem dodać EntityFramework (Core) do naszego projektu.
- t. W ogólnym przypadku wystarczyłoby polecenie:
- u.

dotnet add package Microsoft.EntityFrameworkCore

- v. Natomiast, samo w sobie (bez żadnych dodatkowych opcji) polecenie to będzie próbowało pobrać i dodać do projektu najnowszą wersję EF, co skończy się błędem mówiącym o braku jego zgodności z pozostałymi elementami projektu:

```

info : CACHE https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore/page/8.0.0-preview.7.23375.4/9.0.0-preview.3.24172.4.json
info : Restoring packages for /home/siwik/Labs/LSiwiKEFLab/LSiwiKEFLab.csproj...
error: NU1202: Package Microsoft.EntityFrameworkCore 8.0.4 is not compatible with net7.0 (.NETCoreApp,Version=v7.0). Package Microsoft.EntityFrameworkCore 8.0.4 supports: net8.0 (.NETCoreApp,Version=v8.0)
error: Package 'Microsoft.EntityFrameworkCore' is incompatible with 'all' frameworks in project '/home/siwik/Labs/LSiwiKEFLab/LSiwiKEFLab.csproj'.
siwik@fedora:~/Labs/LSiwiKEFLab$

```

- w. Zgodnie z tym jak sprawdziliśmy to przed przystąpieniem do realizacji ćwiczenia, na naszych komputerach dostępny jest dotnet framework w wersji 7, pod taką wersję frameworka założony/stworzony został nasz projekt i w konsekwencji pod taką też wersję .net frameworka musimy pobrać i dodać sam EntityFramework
- x. Dodajemy zatem w naszym poleceniu wersję EF, którą chcemy pobrać (ostatnią zgodną z .net frameworkiem 7)

dotnet add package Microsoft.EntityFrameworkCore --version=7.0.7

- y. I to powinno już pójść ok (tam w wywołaniu są dwa minusy w sensie minus minus version, ale jak znam życie edytor / konwerter do pdfa sklei to do długiej pauzy ☺)
- z. W konsekwencji, po wykonaniu tego polecenia, w manifeście projektu powinniśmy zobaczyć dodaną zależność do EF, w samym projekcie powinna pojawić się sama biblioteka, a próba zbudowania projektu powinna aktualnie zakończyć się powodzeniem:

aa.

```

LSiwiKEFLab.csproj - LSiwiKEFLab - Visual Studio Code
EXPLORER
  LSiwiKEFLAB
    bin / Debug / net7.0
    LSiwiKEFLab
    LSiwiKEFLab.deps.json
    LSiwiKEFLab.dll
    LSiwiKEFLab.pdb
    LSiwiKEFLab.runtimeconfig.json
    Microsoft.EntityFrameworkCore.Abstractions.dll
    Microsoft.EntityFrameworkCore.dll
    Microsoft.Extensions.Caching.Abstractions.dll
    Microsoft.Extensions.Caching.Memory.dll
    Microsoft.Extensions.DependencyInjection.Abstractions.dll
    Microsoft.Extensions.DependencyInjection.dll
    Microsoft.Extensions.Logging.Abstractions.dll
    Microsoft.Extensions.Logging.dll
    Microsoft.Extensions.Options.dll
    Microsoft.Extensions.Primitives.dll
    obj
      Debug
      LSiwiKEFLab.csproj.nuget.dgspec.json
      LSiwiKEFLab.csproj.nuget.g.props
      LSiwiKEFLab.csproj.nuget.g.targets
      project.assets.json
      project.nuget.cache
      LSiwiKEFLab.csproj
      ProdContext.cs
      Product.cs
      Program.cs
    OUTLINE
    TIMELINE
  LSiwiKEFLab.csproj
    1 <Project Sdk="Microsoft.NET.Sdk">
    2
    3 <PropertyGroup>
    4   <OutputType>Exe</OutputType>
    5   <TargetFramework>net7.0</TargetFramework>
    6   <ImplicitUsings>enable</ImplicitUsings>
    7   <Nullable>enable</Nullable>
    8 </PropertyGroup>
    9
    10 <ItemGroup>
    11   <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.7" />
    12 </ItemGroup>
    13
    14 </Project>
    15
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  siwik@fedora:~/Labs/LSiwiKEFLab$
  siwik@fedora:~/Labs/LSiwiKEFLab$
  siwik@fedora:~/Labs/LSiwiKEFLab$
  siwik@fedora:~/Labs/LSiwiKEFLab$ dotnet build
  MSBuild version 17.4.8+6918b863a for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  LSiwiKEFLab -> /home/siwik/Labs/LSiwiKEFLab/bin/Debug/net7.0/LSiwiKEFLab.dll
  Build succeeded.
    0 Warning(s)
    0 Error(s)
  Time Elapsed 00:00:02.75
  siwik@fedora:~/Labs/LSiwiKEFLab$

```

- bb. No to idźmy dalej. Klasa kontekstowa powinna zawierać property będące kolekcją (typu DbSet) obiektów którymi EF będzie zarządzać. Dodajemy zatem taką kolekcję do naszego kontekstu:

```

using Microsoft.EntityFrameworkCore;

public class ProdContext: DbContext{
    public DbSet<Product> Products { get; set; }
}

```

- cc. Żeby model został odzwierciedlony w bazie danych musimy:
- Przygotować kod odpowiedzialny za migrację modelu oraz
 - Wykonać update struktury bazy danych na podstawie modelu (a w zasadzie kodu odpowiedzialnego za migrację modelu).
- dd. No to spróbujmy to zrobić.
- ee. Żeby przygotować kod odpowiedzialny za migrację wykonujemy (w katalogu projektu) polecenie:

dotnet ef migrations add InitProductDatabase

- ff. Przy pierwszym wywołaniu narzędzia dotnet ef prawdopodobnie zwrócony zostanie błąd / informacja o nierozpoznaniu opcji dotnet ef.
- gg. Żeby to rozwiązać musimy doinstalować "entity framework toolsy" (oczywiście dostosowane do posiadanej wersji .NET frameworka) poleceniem:

dotnet tool install --global dotnet-ef --version=7.0.7

```
● siwik@fedora:~/Labs/LSiwikiEFLab$ dotnet tool install --global dotnet-ef --version=7.0.7
You can invoke the tool using the following command: dotnet-ef
Tool 'dotnet-ef' (version '7.0.7') was successfully installed.
○ siwik@fedora:~/Labs/LSiwikiEFLab$
```

- hh. I powtarzamy próbę przygotowania kodu migracji modelu:

dotnet ef migrations add InitProductDatabase

- ii. Aktualnie, narzędzie zgłasza, że w projekcie brakuje mu pakietu . Design z entity frameworka. No to doinstalujemy je analogicznie jak robiliśmy to wcześniej:

dotnet add package Microsoft.EntityFrameworkCore.Design --version=7.0.

- jj. I powtarzamy polecenie

dotnet ef migrations add InitProductDatabase

- kk. Tym razem dostaniemy prawdopodobnie błąd mówiący o braku skonfigurowanego providera bazy danych dla naszego kontekstu:

```
xtType, String namespace)
  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.
○ siwik@fedora:~/Labs/LSiwikiEFLab$
```

- ll. No I to prawda – póki co, nigdzie w projekcie nie definiowaliśmy do tej pory z jakiej bazy chcemy korzystać (nie tylko jak się ma nazywać, gdzie jest zlokalizowana ale w ogóle z jakiego rodzaju db chcemy korzystać (MSQL, SQLite etc)
- mm. No to skonfigurujemy nasz kontekst, tak żeby wiedział do jakiej bazy chcemy się łączyć.
- nn. Jednym ze sposobów jest nadpisanie w klasie naszego kontekstu metody OnConfiguring. Załóżmy, że chcemy skorzystać z bazy SQLite o nazwie MyProductDatabase.db, wówczas metoda OnConfiguring powinna wyglądać następująco:

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

oo. Czyli aktualnie nasza klasa kontekstowa wygląda następująco:

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : DbContext
{
    public DbSet<Product> Products { get; set; }

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

pp. No to skoro dokonfigurowaliśmy kontekst, to spróbujmy (ponownie) z przygotowaniem migracji:

dotnet ef migrations add InitProductDatabase

qq. Niestety, ciągle “coś nie tak”, a próba przebudowania projektu kończy się następującym błędem:

```
siwik@fedora:~/Labs/LSiwikEFLab$ dotnet build
MSBuild version 17.4.8+6918b863a for .NET
Determining projects to restore...
All projects are up-to-date for restore.
/home/siwik/Labs/LSiwikEFLab/ProdContext.cs(9,24): error CS1061: 'DbContextOptionsBuilder' does not contain a definition for 'UseSqlite' and no accessible extension method 'UseSqlite' accepting a first argument of type 'DbContextOptionsBuilder' could be found (are you missing a using directive or an assembly reference?) [/home/siwik/Labs/LSiwikEFLab/LSiwikEFLab.csproj]
Build FAILED.
```

rr. A to z kolei dlatego, że metoda UseSqlite pochodzi z dedykowanego pakietu .Sqlite dostarczającego odpowiedniego providera do Entity Frameworka, który musimy dodać do projektu analogicznie jak poprzednio: czyli (tam ponownie są dwa minusy przed version ☺)

dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version=7.0.7

ss. No i próbujemy z migracją:

dotnet ef migrations add InitProductDatabase

tt. I wygląda, że tym razem wszystko przebiega ok. Uff ☺

```
ProdContext.cs - LSIWIKFLAB - Visual Studio Code

EXPLORER
  LSIWIKFLAB
    bin
    Migrations
      20240423142944_InitProductDatabase.cs
      20240423142944_InitProductDatabase.Designer.cs
      ProdContextModelSnapshot.cs
    obj
      Debug
        LSIWIKFLAB.csproj.EntityFrameworkCore.targets
        LSIWIKFLAB.csproj.nuget.dgspec.json
        LSIWIKFLAB.csproj.nuget.g.props
        LSIWIKFLAB.csproj.nuget.g.targets
        project.assets.json
        project.nuget.cache
      LSIWIKFLAB.csproj
      ProdContext.cs
      Product.cs
      Program.cs

ProdContext.cs
  ProdContext
    OnConfiguring
      1 using Microsoft.EntityFrameworkCore;
      2 public class ProdContext : DbContext
      3 {
      4     public DbSet<Product> Products { get; set; }
      5
      6     protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      7     {
      8         base.OnConfiguring(optionsBuilder);
      9         optionsBuilder.UseSqlite("DataSource=MyProductDatabase");
      10     }
      11 }

TERMINAL
  bash
  tent hash 21FRzcJhaTrlv7kTrqr/ltFcSQM2TyuTTPhUcj08H73od7Bb3QraNW90c71UucNI/245XPkKZ64fp7/70sKCSg==.
  info : Installed SQLitePCLRaw.lib.e_sqlite3 2.1.4 from https://api.nuget.org/v3/index.json with con
  tent hash 2C9Q9eX7CPLveJA0rIhf9RXAvu+7nWZu1A2MdG6SD/NOu26TakGgLIinsbc0JAspGijF0o3HoN79rx8a368fBg==.
  info : Package 'Microsoft.EntityFrameworkCore.Sqlite' is compatible with all the specified framewor
  ks in project '/home/siwik/Labs/LSIWIKFLAB/LSIWIKFLAB.csproj'.
  info : PackageReference for package 'Microsoft.EntityFrameworkCore.Sqlite' version '7.0.7' added to
  file '/home/siwik/Labs/LSIWIKFLAB/LSIWIKFLAB.csproj'.
  info : Generating MSBuild file /home/siwik/Labs/LSIWIKFLAB/obj/LSIWIKFLAB.csproj.nuget.g.targets.
  info : Writing assets file to disk. Path: /home/siwik/Labs/LSIWIKFLAB/obj/project.assets.json
  log : Restored /home/siwik/Labs/LSIWIKFLAB/LSIWIKFLAB.csproj (in 2.72 sec).
  • siwik@fedora:~/Labs/LSIWIKFLAB$ dotnet ef migrations add InitProductDatabase
  Build started...
  Build succeeded.
  Done. To undo this action, use 'ef migrations remove'
  • siwik@fedora:~/Labs/LSIWIKFLAB$
```

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

dotnet ef database update.

vv. No i pięknie :). Samo polecenie wykonało się bez problemów, a w katalogu projektu pojawił się plik naszej bazy danych:

```
ProdContext.cs - LSIWIKFLAB - Visual Studio Code

EXPLORER
  LSIWIKFLAB
    bin
    Migrations
      20240423142944_InitProductDatabase.cs
      20240423142944_InitProductDatabase.Designer.cs
      ProdContextModelSnapshot.cs
    obj
      Debug
        LSIWIKFLAB.csproj.EntityFrameworkCore.targets
        LSIWIKFLAB.csproj.nuget.dgspec.json
        LSIWIKFLAB.csproj.nuget.g.props
        LSIWIKFLAB.csproj.nuget.g.targets
        project.assets.json
        project.nuget.cache
      LSIWIKFLAB.csproj
      MyProductDatabase
      ProdContext.cs
      Product.cs
      Program.cs

ProdContext.cs
  ProdContext
    OnConfiguring
      1 using Microsoft.EntityFrameworkCore;
      2 public class ProdContext : DbContext
      3 {
      4     public DbSet<Product> Products { get; set; }
      5
      6     protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      7     {
      8         base.OnConfiguring(optionsBuilder);
      9         optionsBuilder.UseSqlite("DataSource=MyProductDatabase");
      10     }
      11 }

TERMINAL
  bash
  info : PackageReference for package 'Microsoft.EntityFrameworkCore.Sqlite' version '7.0.7' added to
  file '/home/siwik/Labs/LSIWIKFLAB/LSIWIKFLAB.csproj'.
  info : Generating MSBuild file /home/siwik/Labs/LSIWIKFLAB/obj/LSIWIKFLAB.csproj.nuget.g.targets.
  info : Writing assets file to disk. Path: /home/siwik/Labs/LSIWIKFLAB/obj/project.assets.json
  log : Restored /home/siwik/Labs/LSIWIKFLAB/LSIWIKFLAB.csproj (in 2.72 sec).
  • siwik@fedora:~/Labs/LSIWIKFLAB$ dotnet ef migrations add InitProductDatabase
  Build started...
  Build succeeded.
  Done. To undo this action, use 'ef migrations remove'
  • siwik@fedora:~/Labs/LSIWIKFLAB$ dotnet ef database update
  Build started...
  Build succeeded.
  Applying migration '20240423142944_InitProductDatabase'.
  Done.
  • siwik@fedora:~/Labs/LSIWIKFLAB$
```

ww. To spróbujmy 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

xx. A zatem wędrujemy do naszego Program.cs. Usuwamy linijkę odpowiedzialną za wydruk napisu HelloWorlds i dodajemy następujące linie kodu:

i. Tworzymy instancję ProductContextu:

ProductContext productContext = new ProductContext();

ii. Tworzymy instancję produktu – niech się nazywa Flamaster:

Product product = new Product { ProductName = "Flamaster" };

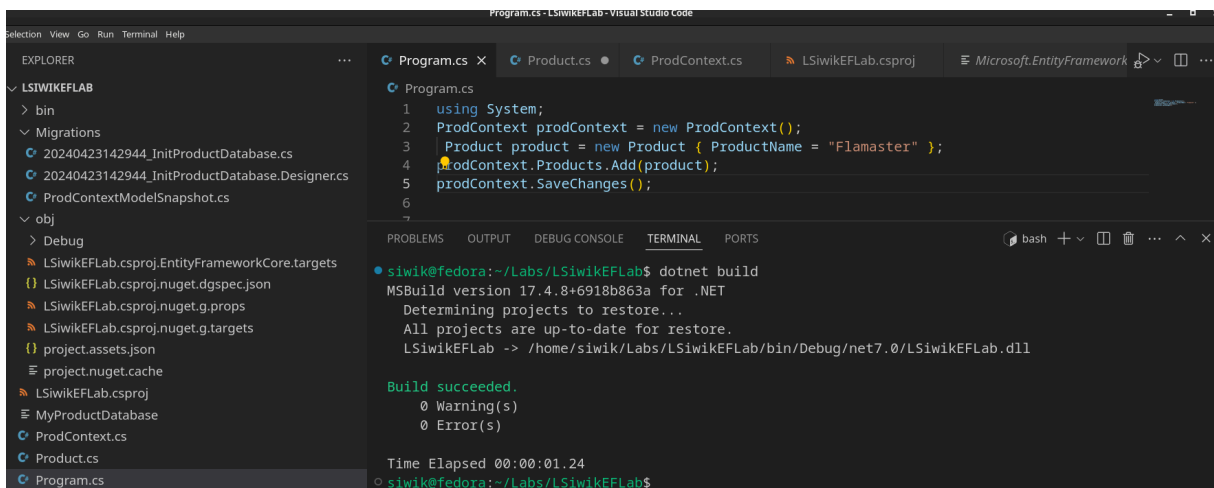
iii. Dodajemy nasz flamaster do kolekcji produktów w productContextcie:

productContext.Products.Add(product);

iv. Zapisujemy zmiany w kontekście:

productContext.SaveChanges();

yy. Zbudujmy na tym etapie i uruchommy projekt.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows the project structure for 'LSiwikEFLab'. The main editor displays the 'Program.cs' file with the following code:

```
1 using System;
2 ProductContext prodContext = new ProductContext();
3 Product product = new Product { ProductName = "Flamaster" };
4 prodContext.Products.Add(product);
5 prodContext.SaveChanges();
6
```

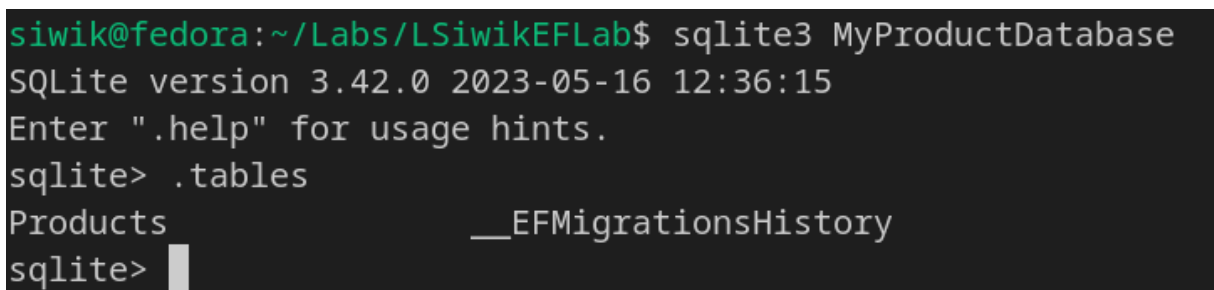
Below the code editor, the TERMINAL pane shows the output of a 'dotnet build' command:

```
siwik@fedora:~/Labs/LSiwikEFLab$ dotnet build
MSBuild version 17.4.8+6918b863a for .NET
Determining projects to restore...
All projects are up-to-date for restore.
LSiwikEFLab -> /home/siwik/Labs/LSiwikEFLab/bin/Debug/net7.0/LSiwikEFLab.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.24
siwik@fedora:~/Labs/LSiwikEFLab$
```

zz. Skoro wygląda ok, to spróbujmy podpiąć się dowolnym klientem pod plik MyProductDatabase i zobaczmy co tam mamy:



The screenshot shows a terminal window with the following commands and output:

```
siwik@fedora:~/Labs/LSiwikEFLab$ sqlite3 MyProductDatabase
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite> .tables
Products                __EFMigrationsHistory
sqlite>
```

aaa. No więc wygląda, że "coś" działa :) No to podglądnijmy strukturę tabeli Products:


```

siwik@fedora:~/Labs/LSiwikEFLab$ sqlite3 MyProductDatabase
SQLite version 3.42.0 2023-05-16 12:36:15
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,
  "UnitsInStock" INTEGER NOT NULL
);
sqlite>

```

bbb. Wygląda OK, w tym sensie, że odpowiada temu co mieliśmy zdefiniowane w naszej klasie Product, przypomnę:

```

public class Product{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }
}

```

ccc. No to nie pozostaje nic innego jak uruchomić raz czy drugi naszą aplikację i zobaczyć co dzieje się w bazie danych:

```

• siwik@fedora:~/Labs/LSiwikEFLab$ dotnet run
• siwik@fedora:~/Labs/LSiwikEFLab$ sqlite3 MyProductDatabase
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite> select * from Products;
1|Flamaster|0
sqlite> .exit
• siwik@fedora:~/Labs/LSiwikEFLab$ dotnet run
○ siwik@fedora:~/Labs/LSiwikEFLab$ sqlite3 MyProductDatabase
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite> select * from Products;
1|Flamaster|0
2|Flamaster|0
sqlite>

```

danych i wypisze je na konsole.

- i. Idziemy zatem do naszego Program.cs i pod kodem który pisaliśmy wcześniej dopisujemy:

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

```
var query = from prod in prodContext.Products
            select prod.ProductName;
```

- iii. Trzeba na tym etapie dodać do programu **using** do **System.Linq**

v. Następnie przechodzimy w pętli po wynikach zapytania i drukujemy na konsolę nazwy naszych produktów:.

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

- v. Czyli w całości mój Program.cs wygląda aktualnie następująco:

```
C# Program.cs • C# Product.cs • C# ProdContext.cs LSiwikEFLab.csproj
```

```
C# Program.cs
1  using System;
2  using System.Linq;
3  ProdContext prodContext = new ProdContext();
4  Product product = new Product { ProductName = "Flamaster" };
5  prodContext.Products.Add(product);
6  prodContext.SaveChanges();
7
8  var query = from prod in prodContext.Products
9              select prod.ProductName;
10
11 foreach (var pName in query)
12 {
13     Console.WriteLine(pName);
14 }
15
```

eee. To zbudujemy i uruchomimy naszą aplikację.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
```

```
Build succeeded.  
0 Warning(s)  
0 Error(s)
```

Time Elapsed 00:00:01.33

```
● siwik@fedora: ~/Labs/LSiwikEFLab$ dotnet run  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster
```

```
○ siwik@fedora: ~/Labs/LSiwikEFLab$
```

fff. No więc zostaliśmy hurtownikami Flamastrów :)

ggg. Żeby móc łatwiej rozróżniać, nasze produkty, zmodyfikujmy może nasz Program tak, żeby przed dodaniem do bazy danych użytkownik został zapytany o nazwę produktu.

i. No więc dodajemy gdzieś na początku linijki typu:

```
Console.WriteLine("Podaj nazwę produktu: ");  
String? prodName = Console.ReadLine();
```

ii. Podmieńmy odpowiednio tworzenie produktu na taki o podanej przez użytkownika nazwie, zbudujmy i uruchommy naszą aplikację:

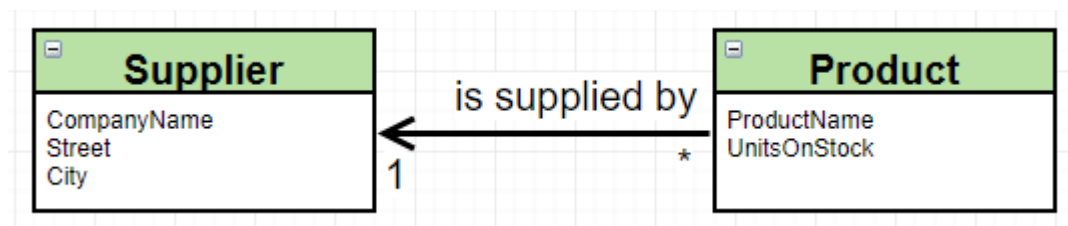
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
bash + v [ ] [x] ..  
  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Flamaster  
Krzeslo  
Kredki  
siwik@fedora: ~/Labs/LSiwikiEFLab$
```

hhh. No więc wygląda, że wszystko (mniej więcej :) tak jakbyśmy chcieli.

iii. Od tego momentu pracujemy **bardziej samodzielnie**.

II. Część samodzielna:

a. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej

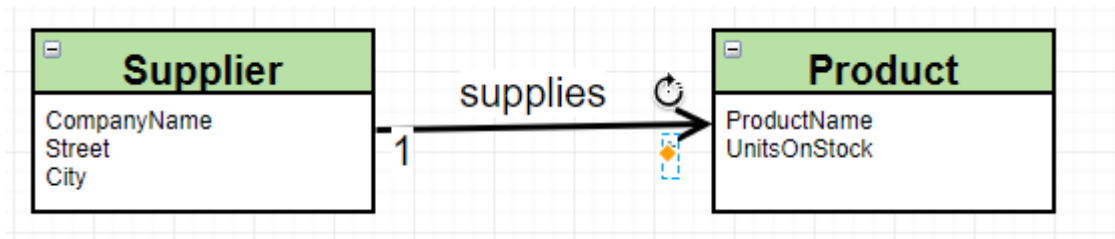


i. Stwórz nowego dostawcę.

ii. Znajdź poprzednio wprowadzony produkt i ustaw jego dostawcę na właśnie dodanego.

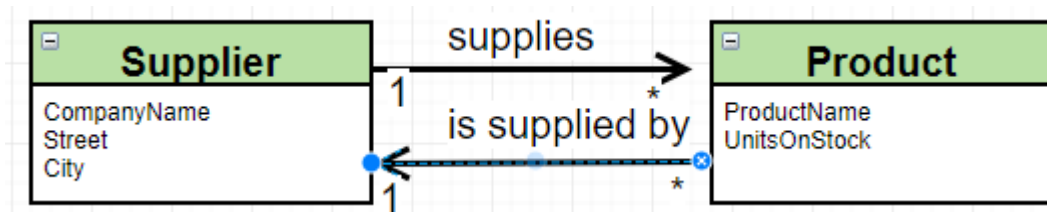
iii. Udokumentuj wykonane kroki oraz uzyskany rezultat (.schema table/diagram z datagrip, select * from....)

b. Odwróć relację zgodnie z poniższym schematem



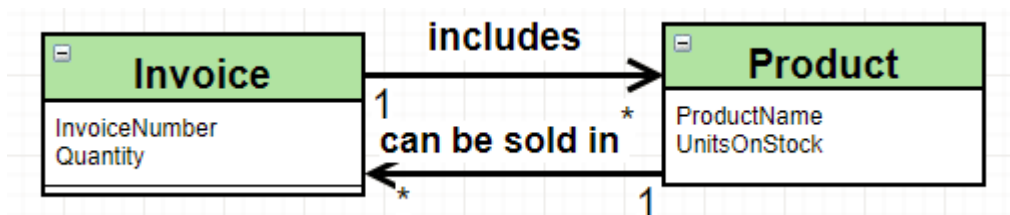
- Stwórz kilka produktów
- Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select * from....)**

c. Zamodeluj relację dwustronną jak poniżej:



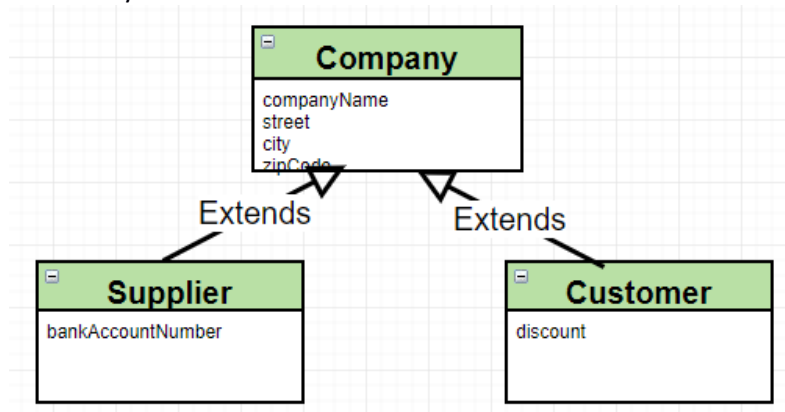
- Tradycyjnie: Stwórz kilka produktów
- Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select * from....)**

d. Zamodeluj relację 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 sprzedany został wybrany produkt
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select * from....)**

- e. Wprowadź do modelu poniższą hierarchie dziedziczenia używając startegii Table-Per-Hierarchy:



- i. Dodaj i pobierz z bazy danych kilka firm obu rodzajów
 - ii. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select * from....)**
- f. Zamodeluj tę samą hierarchię dziedziczenia, ale tym razem użyj strategii Table-Per-Type
- i. Dodaj i pobierz z bazy danych kilka firm obu rodzajów
 - ii. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (.schema table/diagram z datagrip, select * from....)**
- g. Porównaj (i skomentuj/opisz w raporcie) obie strategie modelowania dziedziczenia