WAF - 1. gyakorlat

Az első gyakorlat keretében egy egyszerű konzolos alkalmazást fogunk elkészíteni Entity Framework Core alapú adatbázissal. Az alkalmazás egy modell rétegből fog állni, ami alapján létrehozzuk az adatbázist, valamint egy szolgáltatásokat tartalmazó interface-ből, amellyel az adatbázis-műveleteket végezzük el.

Az alkalmazás tennivalólistákat fog tartalmazni, amelyeket a nevük ír le. A listákhoz tetszőleges számú elem tartozhat, amelyek névvel, határidővel és opcionális leírással rendelkeznek.

Projekt létrehozása

A File → New → Project menüben válasszuk ki a C# nyelvű Console Application típusú projektet!

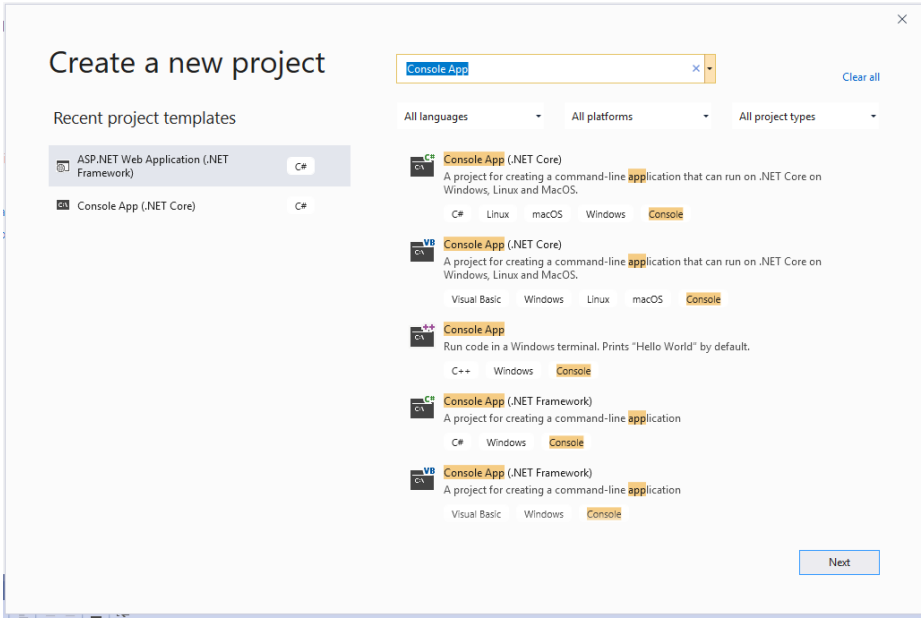

Figure 1: A legelső projekttípust válasszuk ki.

Modell réteg

A modell az adatbázistáblák leképezéseit, az entitás osztályokat fogja tartalmazni, valamint a leképezést és az adatbázisszerverhez való kapcsolódást létrehozó kontextusból. Ebben a rétegben készítünk még mintaadatokat, amelyekkel létrehozáskor töltjük fel az adatbázist.

Lista entitás

Készítsük el a listákat reprezentáló entitás osztályt!

1) Hozzuk létre a entitás osztályokat tartalmazó mappát a projektben!
2) Adjunk egy új osztályt a modellhez: `List.cs`
3) Tegyük publikussá az osztályt!
4) Hozzuk létre az osztályban a következő *propertyket*:
   - Azonosító (`ID`, típusa 32 bites egész szám): a tábla elsődleges kulcsa lesz. Ezt a `Key` annotációval jelezzük!
   - Név (`Name`, típusa szöveg): a `Required` annotációval jelezzük, hogy új lista hozzáadásakor kötelező a mezőt kitölteni, illetve garantáljuk a `MaxLength` annotációval, hogy a név nem lehet 30 karakternél hosszabb.

Adatbázis-kontextus létrehozása

Az adatbázis-kontextus felel az adatbázishoz való kapcsolódásért, valamint a C# kód és az adatbázistáblák egymásra való leképezéséért.

1) Hozzunk létre egy osztályt a kontextusnak! (`TodoListContext`)
2) Az osztály származzon a `DbContext` osztályból, amely a `Microsoft.EntityFrameworkCore` NuGet package letöltésével érhető el a csomaggal megegyező nevű névtérben.
3) Hozzuk létre a listák leképezését a kontextusban! Ezt egy `DbSet` típusú objektum példányosításával tehetjük meg, amelynek típusparamétere a `List`.
4) Írjuk felül a `DbContext`-ből származó `OnConfiguring` metódust! Egy `DbContextOptionsBuilder` típusú paramétert fog várni (`optionsBuilder`). Az `optionsBuilder UseSqlServer` metódusának meghívásával konfigurálhatjuk az adatbázist*. A metódus egy `connection stringet` vár, amelyet ebben a feladatban konstans stringként adunk meg a metódus hívásakor, pl.: `"Data Source=(localdb)\\MSSQLLocalDB;initial catalog=TodoListCore3;Trusted_Connection=True;MultipleActiveResultSets=True"`

A connection string fontos része a `Data Source`, amely megadja, hogy milyen szerverhez kapcsolódva hozzuk létre az adatbázist, valamint az `initial catalog`, amelyben az adatbázis nevét adjuk meg. Ebben a lépésben Microsoft SQL Serverrel dolgozunk.

* A `UseSqlServer` metódust a `Microsoft.EntityFrameworkCore.SqlServer` NuGet package telepítésével érhetjük el.

Adatbázis létrehozása migrációval

1) Telepítsük a `Microsoft.EntityFrameworkCore.Tools` NuGet packageet! Ebben elérhetőek az adatbázis *Package Manager Console*-ból való manipulációjához szükséges parancsok.
2) Nyissuk meg a View → Package Manager Console-t!

3) Hozzunk létre migrációt a következő paranccsal: `Add-Migration CreateList`
4) Az előző lépéssel létrejöttek az adatbázist leíró C# kódok. Az adatbázis tényleges létrehozásához futtassuk az `Update-Database` parancsot!
5) Az adatbázis tartalmát a View → SQL Server Object Explorer ablakban tekinthetjük meg.

Megjegyzés: az Entity Framework Core parancsait nem csak a Visual Studio *Package Manager Console*-jából érhetjük el, hanem az operációs rendszer termináljából is (pl. Linux operációs rendszer alatt), amennyiben telepítettük az EF Core konzolos eszközeit, amelyet megtehetünk lokálisan a projekthez, vagy globálisan a felhasználói fiókunkhoz. Javasolt az utóbbi, hiszen gyakran lehet szükségünk ezekre az eszközökre:

```
dotnet tool install --global dotnet-ef
```

Ilyenkor az előbbi két parancs megfelelője:

- `dotnet ef migrations add CreateList`
- `dotnet ef database update`

Listaelemek

Készítsük el a listaelemeket reprezentáló entitás osztályt! A elemek sok-az-egyhez kapcsolatban állnak a listákat tartalmazó táblával (egy listához több elem tartozhat, de egy elem csak egy listához).

1) Adjunk egy új osztályt a modellhez: `Item.cs`
2) Tegyük publikussá az osztályt!
3) Hozzuk létre az osztályban a következő propertyket:
   - Azonosító (`ID`, típusa 32 bites egész szám, elsődleges kulcs)
   - Név (`Name`, típusa szöveg, kötelező mező, maximális hossza 30 karakter)
   - Leírás (`Description`, típusa szöveg): a `DataType` annotáció `MultilineText` típusával garantáljuk, hogy többsoros lehessen a leírás!
   - Határidő (`Deadline`, típusa `DateTime`, kötelező mező)
   - A elemet tartalmazó lista azonosítója (`ListId`, típusa 32 bites egész szám, kötelező mező)
   - Az elemet tartalmazó lista (`List`, típusa `List`, virtuális mező).

Egészítsük ki a List osztályt a hozzá tartozó listaelemek csoportjával! Ehhez definiáljunk a List osztályban egy `ICollection<Item>` propertyt (`Items`).

Az adatbázis-kontextusban hozzunk létre egy `DbSet` példányt a listaelemek leképezéséhez!

A korábbi `Add-Migration` és `Update-Database` parancsok segítségével frissítsük az adatbázis szerkezetét!

Az adatbázis feltöltése mintaadatokkal

1) Hozzunk létre egy publikus, statikus osztályt `DbInitializer` néven!
2) Az osztálynak legyen egy publikus, statikus `Initialize` nevű metódusa, amely nem ad vissza semmit, és egy adatbázis-kontextust vár paraméterül.
3) Az `Initialize` metóduson belül győződjünk meg róla, hogy az adatbázis létezik (`EnsureCreated`).
4) Amennyiben az adatbázisban már vannak adatok (pl. a List tábla nem üres), térjük vissza.
5) Hozzunk létre 1-2 új listát néhány elemmel. Minden kötelező adatot adjunk meg.
6) A listákat adjuk hozzá az adatbázis `List` táblájához, amelyet az adatbázis-kontextuson keresztül érhetünk el.
7) Mentsük el a kontextus változtatásait (`SaveChanges`)!

Szolgáltatás-interface létrehozása

Ebben az interface-ben definiálunk műveleteket, amelyek az adatbázis-manipulációt végzik. Szigorúan véve a modell réteg része, nem képez külön réteget.

1) Adjunk a projekthez egy `Services` nevű mappát!
2) Készítsünk egy új osztályt a `Services`-ben (`TodoListService`)!
3) Tegyük publikussá az osztályt!
4) Az osztály konstruktora paraméterként kapjon egy adatbázis-kontextus példányt, amin keresztül az osztály metódusai el tudják érni az adatbázist.

CRUD műveletek

Definiáljunk néhány metódust, amelyek alapvető műveleteket végeznek az entitásokon! Négy művelethez definiálunk metódusokat: adatok hozzáadása, lekérése, módosítása és törlése.

1) **Create**
   - Legyen egy `AddItemToList` metódusunk, amely egy listaelemet vár paraméterül, és nem ad vissza semmit. Ha az elem nem `null`, adjuk hozzá az `Items` entitáshoz! Ne felejtsük el elmenteni a változtatást.
2) **Read**
   - Készítsünk egy `GetLists` nevű metódust, amely listázza a megadott stringet tartalmazó nevű listákat. A metódus egy alapértelmezetten üres stringet vár paraméterül, és listaentitások egy listáját adja vissza. Egy `Linq` lekérdezéssel kérjük le az adatbázisból azokat a listákat, amelyek nevében szerepel a paraméterként kapott string! A lista legyen a listanevek szerint növekvő sorrendbe rendezve.
   - Készítsünk egy `GetListByID` nevű metódust, amely egy egész számot vár paraméterként (egy azonosítót), és egy modellbeli listát ad vissza. Egy `Linq` lekérdezéssel kérjük le az adatbázisból azt a listát és elemeit, amelynek az azonosítója megegyezik a paraméterként kapott számmal!

- Definiáljunk egy `GetItemsByListID` nevű metódust, amely a megadott azonosítójú lista elemeit adja vissza! Egy `Linq` lekérdezéssel keressük ki az azonosítónak megfelelő listát, és adjuk vissza a hozzá tartozó elemeket!

3) **Update**
   - Definiáljunk egy `ChangeListName` nevű metódust, amellyel megváltoztathatjuk egy lista nevét! A metódus egy listaazonosítót és egy új nevet vár paraméterül, és nem ad vissza semmit. Egy `Linq` lekérdezéssel keressük meg az azonosítónak megfelelő listát, és változtassuk meg a nevét! Mentsük el a változtatást.
4) **Delete**
   - Definiáljunk egy `RemoveItemByName` nevű metódust, amely kitörli a megadott listából a keresett nevű listaelemet! A metódus egy lista azonosítót és elem nevet vár paraméterül, és nem ad vissza semmit. A név alapján egy `Linq` lekérdezéssel keressük ki a megadott nevű elemet, majd töröljük az `Items` entitásból! Mentsük el a változtatást.

A `TodoListService`-ben definiált műveleteket hívjuk meg a `Program.cs` osztály `Main` metódusában! Szükség lesz egy példányra a `TodoListContext`-ből, amivel meghívhatjuk az adatbázist feltöltő statikus metódust, valamint a service ezen keresztül fogja elérni az adatbázist.

SQLite támogatás

Tegyük cross-platformmá az alkalmazást: biztosítsunk lehetőséget SQLite szerverhez való kapcsolódásra!

1) A modellben hozzunk létre egy fájlt `DbType.cs` néven!
2) A fájlban legyen egy publikus, `DbType` nevű enum, amelynek legyen két értéke: `SqlServer` és `Sqlite`.
3) Adjunk a projekthez egy fájlt `appsettings.json` néven! Ebbe a fájlba emeljük át az MSSQL szerverhez tartozó connection stringet (a szekció neve legyen `ConnectionStrings`), valamint adjunk hozzá egy új connection stringet, amivel egy SQLite adatbázishoz lehet majd kapcsolódni! Pl. `"Data Source=TodoListCore3.db"`
4) Legyen még egy kulcs-érték párunk a fájlban, ahol a kulcs `DbType`, az érték (SQLite támogatás esetén) `Sqlite`.
5) Adjuk a projekthez a `Microsoft.Extensions.Configuration.Json` NuGet csomagot, amellyel könnyedén strukturáltan beolvashatunk JSON konfigurációs állományokat.
6) Az `OnConfiguring` metódust egészítsük ki úgy, hogy SQLite szerverhez is tudjunk kapcsolódni (`UseSqlite`)! Ehhez a `Microsoft.EntityFrameworkCore.Sqlite` NuGet package-re lesz szükség. A metódust egészítsük ki egy `ConfigurationBuilder` típusú objektummal, amely beállítja a projektkönyvtárat alapértelmezett útvonalnak (`SetBasePath(Directory.CurrentDirectory())`), és itt keresi a connection stringeket tartalmazó JSON fájlt (`AddJsonFile("appsettings.json")`)!

7) Készítsünk egy `IConfigurationRoot` típusú objektumot úgy, hogy meghívjuk az előző lépésben létrehozott objektum `Build` parancsát.
8) Ezen az objektumon keresztül elérhetjük az `appsettins.json DbType` kulcsához tartozó értéket (`GetValue`), valamint az ennek megfelelő connection stringet (`GetConnectionString`), amivel helyettesíthetjük az eddigi beégetett értéket.