WAF - 5. gyakorlat

Az ötödik gyakorlat célja, hogy egy webszolgáltatást és egy azt feldolgozó klienst nyújtsunk a korábban elkészített adatbázishoz. A WPF alapú kliensben szeretnénk a meglévő adatokat megjeleníteni, illetve követni a weboldalon keresztül eszközölt változtatásokat.

Amennyiben a skeleton projektből indulsz ki, folytasd a ***WebApi*** *fejeztnél a munkafüzetet.*

Refaktorálás

Mivel a webszolgáltatás és a weboldal is ugyanazokat az entitás modelleket használná, ezért érdemes egy külön projektet létrehozni a perzisztencia rétegnek, így elkerüljük a kód ismétlést. Az új projekt *Class Library (.NET Standard)* legyen, így .NET Core és .NET Framework alkalmazásokban is felhasználható lesz.

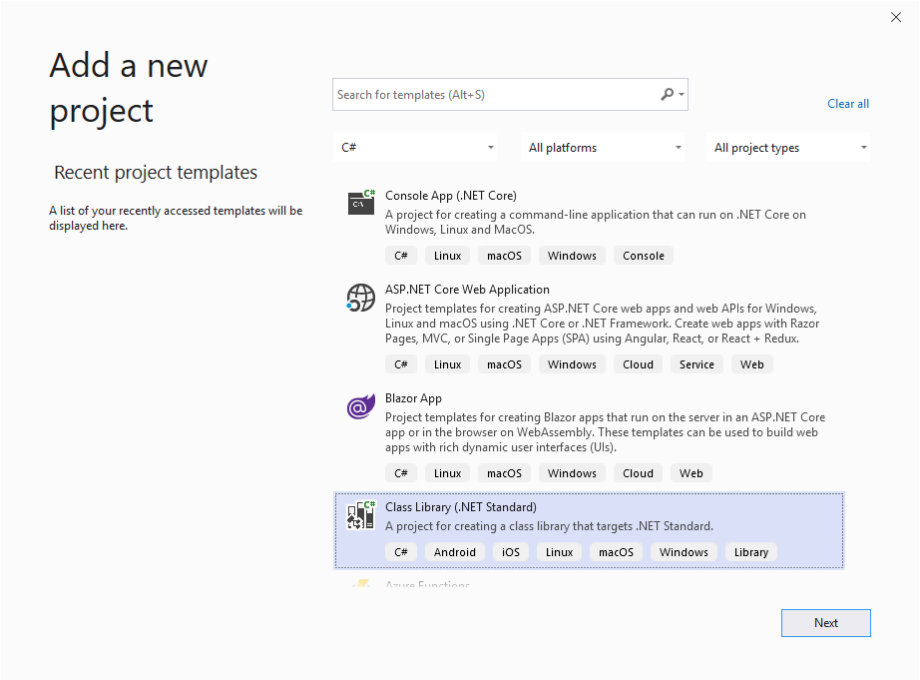

Figure 1: Projekt létrehozása a perzisztencia rétegnek

Emeljük át ebbe a projektbe a perzisztencia réteghez tartozo osztályokat:

- ApplicationUser
- DbInitializer
- DbType
- List
- Item
- TodoListDbContext
- Migrations mappa
- Services mappa

Adjuk hozzá az új projekthez az alábbi használt nuget csomagokat*, illetve írjuk át a névtereket az új struktúrát tükrözve.

- *Microsoft.EntityFrameworkCore*
- *Microsoft.AspNetCore.Identity.EntityFrameworkCore*

**Előfordulhat, hogy néhány csomag verzió .NET Standard 2.1-et igényel. A projekt tulajdonságaiban tudjuk átállítani a Target Framework-öt*

Adjuk hozzá a weboldal projekthez függőségként az új projektünket a *dependencies → add reference* által.

Mivel szeretnénk, hogy a webszolgáltatás és a weboldal is ugyanazt az adatbázist érje el, ezért ha *Sqlite*-ot használunk módosítsuk az adatbázis fájl útvonalát relatív egy mappával feljebb.

Miután javítottunk minden névtér elérést, ellenőrizzük, hogy ugyanúgy működik-e a weboldalunk mint eddig.

Webszolgáltatás - WebApi

Hozzunk létre egy új projektet a webszolgáltatásnak az *ASP.NET Core Web Application → API* template segítségével.

Mint ahogy a weboldal esetében is, az egyszerűség kedvéért kapcsoljuk ki a HTTPS-t

Miután tanulmányoztuk a projektben létrejött példakódot, töröljuk azt. Ehhez a projekthez is adjuk hozzá függőségként a perzisztencia réteget a korábban ismertetett módon, illetve állítsuk be az adatbázis elérést a weboldalhoz hasonlóan a `Startup.cs` és az `appsettings.json`-ban.

Vezérlők

Két végpontot szeretnék a webszolgáltatással biztosítani. Egy `Lists`-et mellyel az összes listát, illetve egy `Items`-et mellyel a paraméterül megadott azonosítóju listához tartozó összes elemet tudjuk lekérni. Ezekhez hozzunk létre kontrollereket a weboldalnál mar ismertetett módon, ügyelve arra, hogy az adatbázist ne közvetlenül a kontextuson hanem a `TodoListService` osztályon keresztül érjük el.

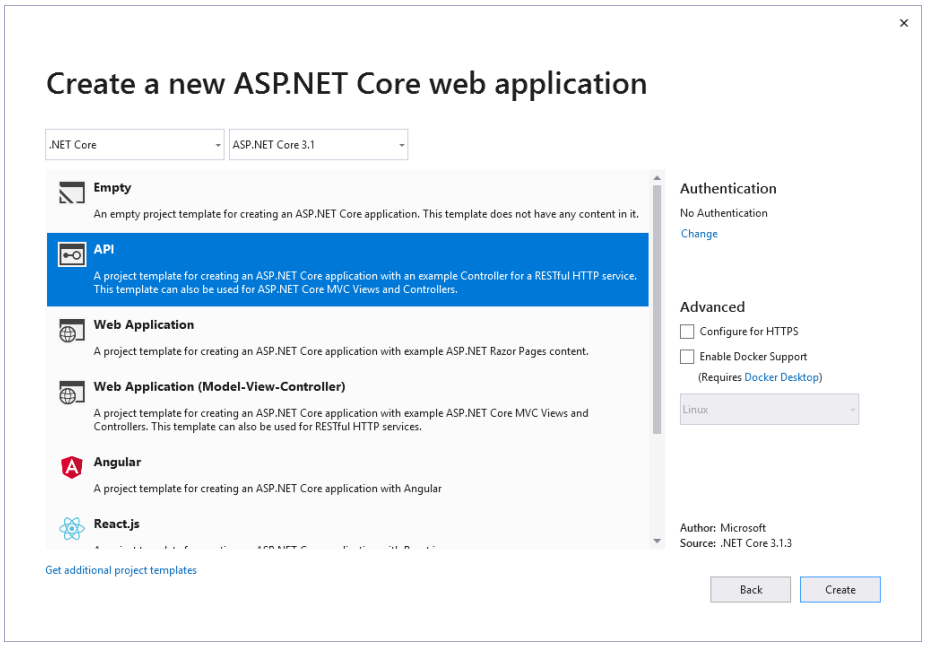

Figure 2: Projekt létrehozása a webszolgáltatásnak

A webszolgáltatásoknál különösen fontosak a válaszul küldött *hibakódok*, ezért ügyeljünk arra, hogy ha egy nem létező azonosítót kapunk az elem lekérdező végpontnál akkor egy ennek megfelelő *404 Not Found* státusz kóddal térjunk vissza pl. `return NotFound()`

Manuális tesztelés

Mielőtt megírnánk a klienst, teszteljük le a Webszolgáltatást egy erre alkalmas eszközzel (*Postman*, *httprepl*). Tegyük indulóvá az új projektet, a tulajdonságaiban pedig jegyezzük fel az alkalmazásunk portját, illetve kapcsoljuk ki a *Launch browsert*, mivel nem szeretnénk böngészőt indítani a projekt indulásakor. Amennyiben nem írtuk át, a végpontjaink el vannak látva egy *api* prefixszel. Figyeljük meg, hogy az alap beállitas szerint a végpontok JSON kódolásban küldik el az objektumokat. Ha megpróbálunk lekérni egy listához tartozó elemeket, azt tapasztalhatjuk, hogy a JSON szerializáló önmagára hivatkozó kört talált ezért hibát dob. Ez az entitás modellünkben szereplő navigációs tulajdonságok miatt van. A szerializáló egy elemről visszajut az azt tartalmazó listába ami szintén tartalmazza az összes elemet, így egy végtelen ciklust generál.

Adatátviteli objektumok

A kliens és a szerver közti kommunikáció általában úgynevezett adatátviteli objektumokkal(*Data Transfer Object - DTO*) történik. Ezek az adott kérésre szabott

objektumok, melyek csak az átküldendő információt tartalmazzák. Esetünkben az entitásmodelljeink a navigációs tulajdonságokat kihagyva megfelelnek erre a célra, hozzunk létre, nekik megfeleltethető DTO-kat a perzisztencia rétegben és a továbbiakban őket használjuk a szerver és a kliens közti kommunikációra az entitásmodellek helyett.

Megjegyzés: A DTO-kban praktikus konverziós operátorokat implementálni a típusok közti megfeleltetésekre

Asztali alkalmazás - WPF

A korábbi félévekből már ismerősnek kell lenni a keretrendszernek, így itt kevésbe lesz részletes a leírás. Hozzunk létre egy új projektet a kliensnek.

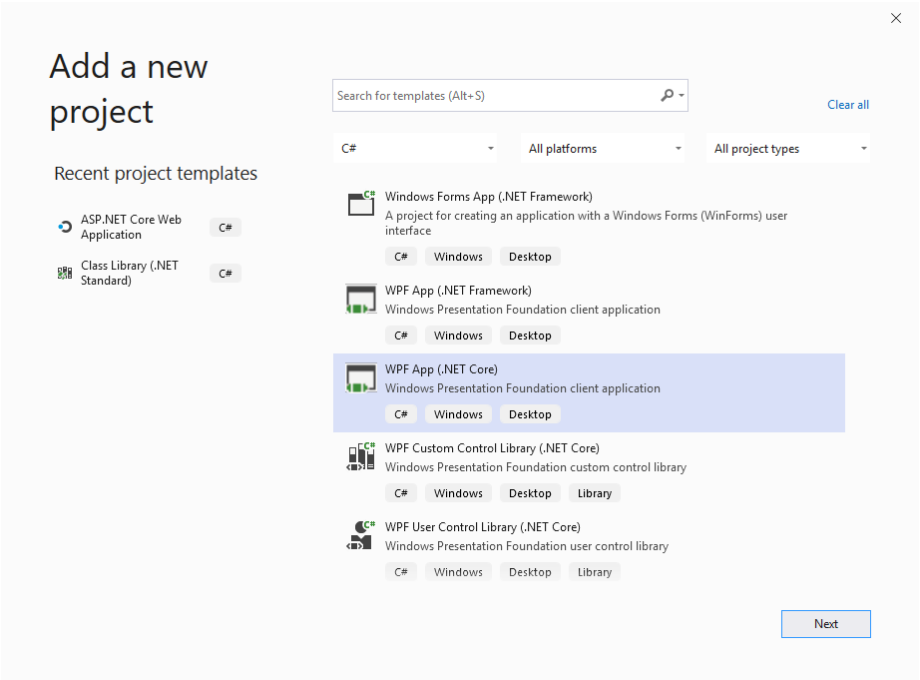

Figure 3: Projekt létrehozása a kliensnek

MVVM architektúrában fogunk dolgozni, ezért hozzunk létre három mappát ezen rétegeknek(*Model*, *View*, *ViewModel*) Mivel a DTO-k a perzisztencia réteg projektjében kaptak helyet így állitsuk be azt a projekt függőségeként. Megjegyzendő, hogy ezen osztályoknak akár saját projektet is létre hozhatnánk csökkentve így a perzisztencia réteghez való szoros függést. A korábban feljegyzett elérési útvonalat a webszolgáltatáshoz helyezzük el egy új konfigurációs fájlba. *Add → New item → Application Configuration File*

Modell réteg

Hozzunk létre egy szervíz osztályt, ami a webszolgáltatással történő kommunikációért felel. A kérések elvégzésére a `HttpClient` osztályra lesz szükségünk, állítsuk be, a majd később konstruktoron keresztül átadott elérési útvonalat a `BaseAddress` tulajdonságban. Valosítsuk meg a kettő végpontot lekérdező metódusokat. A `HttpClient GetAsync(string endpoint)` metódusával tudunk *GET* kéréseket intézni. Az `Items` végpontunkhoz tartozó paraméter megadásához használjuk a `QueryHelpers.AddQueryString` metódust, mely a `Microsoft.AspNetCore.WebUtilities` csomagban érhető el. A válasz `HttpResponseMessage` tartalmazza a kapott státuszkódot és a tartalmat JSON stringként amit még deszerializálnunk kell. Ehhez adjuk hozza a projekthez a `Microsoft.AspNet.WebApi.Client` csomagot, ami kiegészíti a `HttpContent`-et egy ezt elősegítendő `ReadAsAsync` metódussal. Ha nem várt státuszkódot kapunk, jelezzük ezt a hívó félnek egy saját kivételtípus dobásával.

Nézet réteg

A főablakban szeretnénk megjeleníteni egy a listákat tartalmazó `ListBox` elemet és egy `DataGrid`-ben jelenítsük meg az aktuálisan kiválasztott listához tartozó elemeket, azok részleteivel együtt. *Megjegyzés: Amennyiben szeretnénk használni egy esemény és parancs összeköttetését elősegítő `Interaction.Triggers`-t, adjuk hozzá a projekthez a `Microsoft.Xaml.Behaviors.Wpf` csomagot*

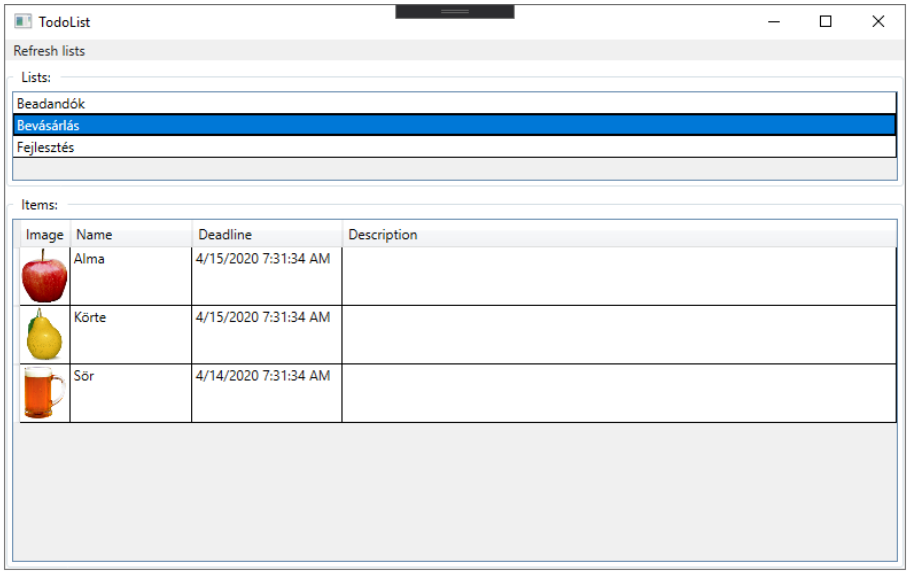

Figure 4: A főablak egy lehetséges kinézete

Nézetmodell réteg

Használjuk a korábbi félévekben megismert `DelegateCommand`, `ViewModelBase` és `MessageEventArgs` osztályokat. Hozzunk létre a nézetünkhöz egy nézetmodell osztályt. A korábban megírt szervíz osztály segítségével már az ablak megjelenítésekor kérjük le a listákat, melyekkel töltsük fel a nézethez kötött ehhez tartozó `ObservableCollection`-t. Esetleg tegyük lehetővé a listák frissítését egy fájl menüben. Kezeljük az esetlegesen kapott kivételeket egy `MessageBox`-al mellyel informáljuk a felhasználót az aktuális hibáról.

Alkalmazás réteg

A konfigurációs fájlból kérjuk le a webszolgaltatás elérési útvonalát: `ConfigurationManager.AppSettings["baseAddress"]`, majd ezzel konstruáljuk meg a szervíz osztályunkat, azzal pedig a fő ablakhoz tartozó nézetmodellt.

Állítsuk be mindkét vagy akár mindhárom projektet induló projektnek a solution tulajdonságainal és utána teszteljük le működésüket.