WAF - 2. gyakorlat

A második gyakorlaton folytatjuk a TodoList fejlesztését, most már MVC architektúrában. Az eddigi listaelem modellt kiegészítjük egy adatbázisban tárolt képpel, illetve a webes felületen megjelenítjük a listákat és a hozzájuk tartozó elemeket. Az elemeket lehetőségünk lesz név vagy határidő szerint rendezni.

Amennyiben a *skeleton* projektből indulsz ki, folytasd a **Controller (vezérlő) réteg** fejeztnél a munkafüzetet.

Projekt létrehozása

A Visual Studio 2019 menüjében válasszuk az *ASP.NET Core Web Application* opciót!

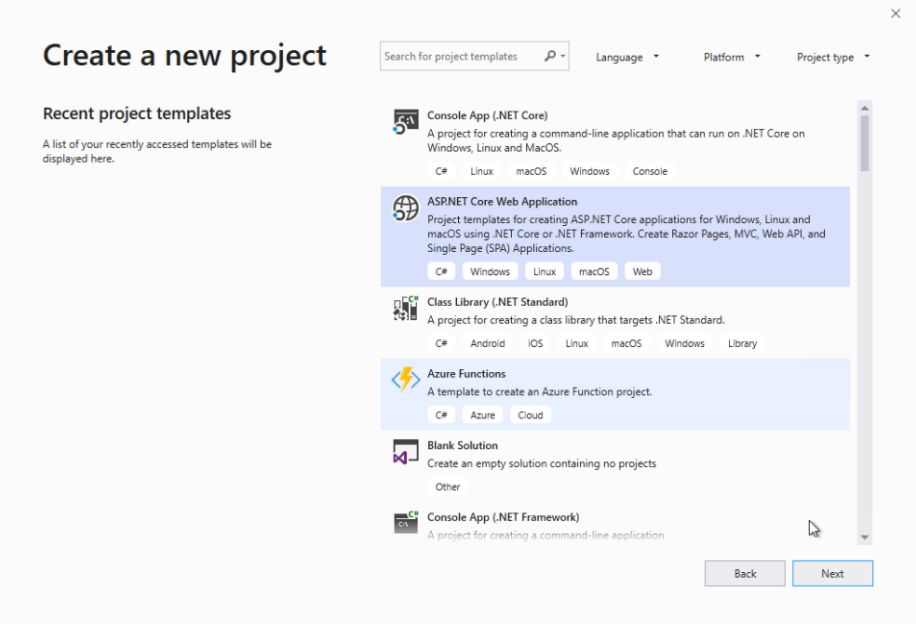

Figure 1: Új MVC webalkalmazás projekt létrehozása

A projekt létrehozásakor kapcsoljuk ki a HTTPS konfigurációt!

Modell réteg

A modell réteg korábbi osztályait / enumjait (`List`, `Item`, `DbType`, `DbInitializer`) az első gyakorlat utolsó állapotának megfelelően hozzuk létre az alkalmazás modell rétegében. Az `appsettings.json` fájlban definiált connection stringeket és a *DbType* értékét másoljuk át az új projektben automatikusan létrehozott `appsettings.json` fájlba.

Szolgáltatás-interface

Az előző gyakorlaton készített konzolos alkalmazás `TodoListService` osztályát is emeljük át a projektbe! A listákra vonatkozó műveletek közül az összes lista lekérésére (`GetLists`) és az azonosító alapján való lekérésre (`GetListById`) lesz szükség. Új metódsuként definiáljuk egy listaelem azonosító alapján történő lekérését (`GetItem(int id)`).

Startup.cs és az adatbázis-kontextus

Az ASP.NET Core keretrendszerben az ún. *service provider* egy IoC tárolóként (*IoC container*) funkcionál, azaz egy olyan *Inversion of Control* paradigmájú komponens, amely lehetőséget ad szolgáltatások megvalósításának dinamikus (futási idejű) betöltésére. Az IoC tároló egy központi regisztráció, amelyet minden programkomponens elérhet és felhasználhat.

A `Startup` osztály `ConfigureServices` metódusát egészítsük ki a `TodoListContext`ben található `OnConfiguring` metódus kódjával, amely az aktuális `DbType` alapján eldönti, hogy MSSQL szervert vagy SQLite-ot fog használni az adatbázis-kezeléshez. Az adatbázis kontextus IoC tárolóba történő regisztrációját elvégezhetjük a `ConfigureServices` metódus `IServiceCollection services` paraméterének `AddDbContext` eljárásával.

A `TodoListContext` osztályban az `OnConfiguring` metódusra ezután már nem lesz szükség, helyette egy `DbContextOptionsBuilder` típusú paramétert váró üres konstruktort hozzunk létre, amely meghívja a szülőosztály konstruktorát (`base`). Ilyen módon az adatbázis kontextus konfigurálását kívülről, függőségi befecskendezéssel (*dependency injection*) végezhetjük.

A `Startup` osztály `Configure` metódusa az eddigieken felül várjon egy `IServiceProvider` típusú paramétert! A `DbInitializer` osztály is várjon egy ilyen típusú paramétert, így az előbbiekben konfigurált és regisztrált adatbázis kontextust le tudjuk kérni itt is. Az IoC tárolóba regisztráljuk a `TodoListService` típust is (pl. `AddTransient`)!

Az osztályban lévő adatbázis-kontextust mostantól kérjük le ettől az objektumtól:
`TodoListContext _context = serviceProvider.GetRequiredService<TodoListContext>();`
A `DbInitializer` statikus osztály `Initialize` metódusát hívjuk meg a `ConfigureServices` metódus végén!

A 2. gyakorlathoz tartozó kiindulási *skeleton* projekt már tartalmazza az eddig leírt teendőket - a `TodoListService` osztályhoz történő `GetItem` metódus hozzáadását leszámítva.

Controller (vezérlő) réteg

Controller és nézetek létrehozása

A `Controllers` mappára jobbklikkelve az *Add Controller* menüpont alatt adhatunk a projekthez új controller osztályt. Adjuk meg az új controllerhez tartozó entitást és az adatbázis-kontextust. A legegyszerűbb, ha egyből nézetekkel együtt hozzuk létre a controllert, ekkor a `Views` mappában létrejön egy, a controllerünknek megfelelő nevű új mappa, amely az alapértelmezett CRUD műveletek mindegyikéhez tartalmaz egy-egy nézetet, amelyek `.cshtml` kiterjesztést kapnak. Hozzunk létre controllert és nézeteket a `List` entitáshoz!

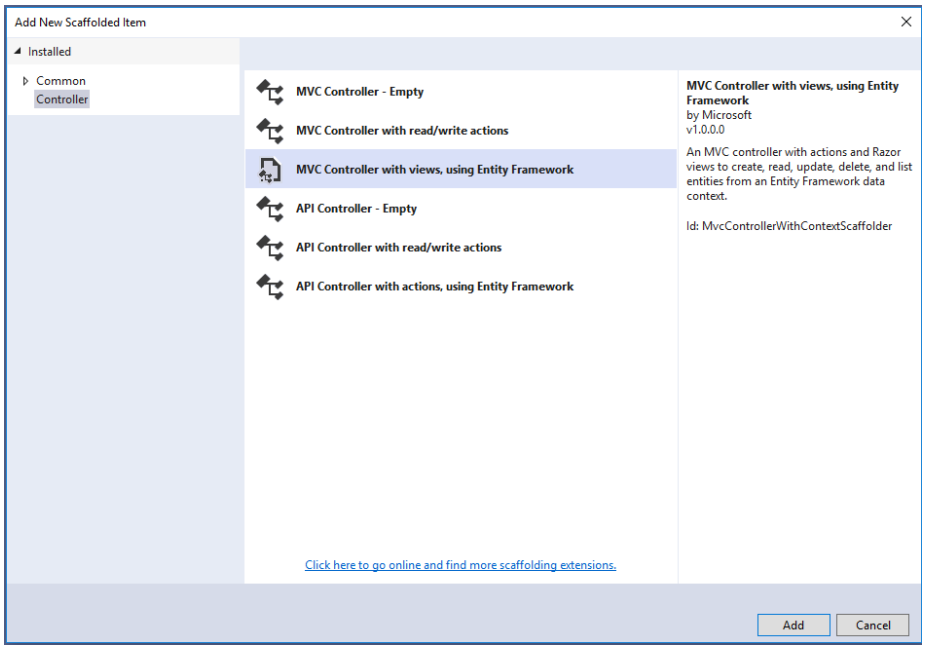

Figure 2: CRUD controller osztály generálása

A controller `TodoListContext` példány helyett egy `TodoListService` objektummal rendelkezzen. Ezt a vezérlő osztály konstruktora átveheti, és az IoC tároló automatikusan befecskendezi majd.

Megjegyzés: az első controller osztály generálásakor a projekthez automatikusan hozzáadásra kerül a `Microsoft.VisualStudio.Web.CodeGeneration.Design` NuGet package, erre a generáláshoz szüksége is van.

Listaentitások megjelenítése

A generált controllerbeli metódusok közül az `Index` felelős a listák megjelenítéséért, a nézet rétegben pedig az azonos nevű nézet (a controller által

generált metódusokhoz automatikusan létrejön egy-egy azonos nevű nézet, ha ezt az opciót választjuk). Az `Index` metódusban adjunk vissza egy nézetet, amely visszaadja a listákat (a service megfelelő metódusának meghívásával).

A nézetben a listákhoz adjunk egy-egy linket (`<a>`), amely a listához tartozó listaelemeket jeleníti meg!

Listaelemek megjelenítése

A listaelemek megjelenítését az `ListController` osztály `Details` metódusa végzi. A `Details` adja vissza a kapott azonosítónak megfelelő lista nézetét.

A `Details.cshtml` fájl tartalmát írjuk át úgy, hogy az elemeket táblázatos formában mutassa!

Listaelemek sorba rendezése

A `ListController` osztály `Details` metódusát egészítsük ki az elemek sorba rendezésével! A metódus várjon egy második paramétert, amely a rendezés típusát fogja meghatározni (`sortOrder`, típusa `String`). A `Details`hez tartozó nézetben az listaelemek nevét és határidejét megjelenítő oszlopok fejléceit (`Name` és `Deadline`) tegyük linkké (`<a>`), amelyek a `Details` akciót hívják, és a `sortOrder` paraméterhez egy-egy értéket kötnek, a következő módon: `asp-route-sortOrder="@ViewData["NameSortParm"]"` (a határidő esetében természetesen másik kulcsra lesz szükség, pl. `@ViewData["DeadlineSortParm"]`).

A `ViewData`ban lévő kulcsoknak a controllerben a `sortOrder` aktuális értékének megfelelően adjunk értéket! Ha a `sortOrder` üres string volt (ez lesz az alapértelmezett értéke), váltsuk át név szerint csökkenőre (pl. `name_desc`). Ennek megfelelően váltogassunk a határidő szerinti rendezések között a ViewData másik kulcsánál (pl. `deadline_asc` és `deadline_desc`)! Ezután a `sortOrder` értékétől függően rendezzük a korábban lekért listaobjektum elemeinek sorrendjét.

Képkezelés

A listaelemeket reprezentáló modell osztályt (`Item`) egészítsük ki egy új propertyvel: *Kép* (`Image`, típusa bájttömb). Listaelem létrehozásakor lehetőségünk lesz a webes felületen keresztül kiválasztani egy képet, amit beolvasás után bájttömbként tárolunk az adatbázisban. A korábbi `Add-Migration` paranccsal készítsünk új migrációt az adatbázis szerkezetének módosításához, illetve a `DbInitializer`ben az `EnsureCreated` helyett a `Migrate` metódust hívjuk meg!

Az adatbázishoz a `DbInitializer`en keresztül fogunk képeket adni. Hozzunk létre a projektben egy `App_Data` nevű mappát, ebben fogjuk tárolni a képeket. Az `appsettings.json`t egészítsük ki egy új kulcs-érték párral: `"ImageSource": "App_Data"`

A `DbInitializer` statikus osztály `Initialize` metódusa ezentúl várjon egy paramétert, amely megmondja, hogy hol kell keresnie a képeket (`imageDirectory`, típusa `string`)! Híváskor kérje le az `ImageSource` értékét az `appsettings.json`ből (ezt pl. a connection string lekéréséhez hasonlóan tehetjük meg). Az `Initialize` metódusban a listák és listaelemek létrehozását egészítsük ki a képek hozzáadásával:

1) Vizsgáljuk meg, hogy létezik-e az átvett könyvtár a `Directory` osztály `Exists` metódusa segítségével!
2) Ha létezik, a `Path` osztály `Combine` metódusával készítsük el a hozzáadni kívánt képek útvonalát!
3) Ha ez sikerült, és a fájlok léteznek (`File` osztály, `Exists` metódus), az inicializáló listában megadhatjuk az `Image` property értékét. Ez egy bájttömb, tehát a beolvasott képet konvertálni kell. Ezt a `File` osztály `ReadAllBytes` metódusával érhetjük el.

Megj.: a `Directory`, `Path` és `File` osztályok a `System.IO` névtérben találhatóak.

Képmegjelenítés

1) Az `ItemsControllert` egészítsük ki egy új akcióval, amely a képmegjelenítésről fog gondoskodni (`DisplayImage`, egy azonosítót vár paraméterül, és egy `IActionResult`ot ad vissza).
2) Kérjük le az azonosítóhoz tartozó listaelemet!
3) Adjunk vissza egy `FileResult` típusú eredményt, ehhez használjuk a `Controller` ősosztályból örökölt `File` metódust. Ez argumentumként a listaelemhez tartozó képet és annak MIME type-ját (`image/png`) várja!
4) A `ListsController` osztály `Details` nézetében lévő táblázathoz adjunk egy új oszlopot *Image* fejléccel! Ha a listaelemhez tartozik kép, azt jelenítsük meg (`<img>`), aminek a forrása (`src` attribútum) egy `@Url.Action`, ami az `ItemsController` osztály `DisplayImage` akcióját hívja!
5) A `wwwroot` mappában található `site.css` fájlt egészítsük ki egy `img.item` nevű szelektorral, ami megadja, hogy egy kép maximális szélessége és hosszúsága egyaránt 50 pixel! A képek `class` attribútumának ennek megfelelően legyen `item`˙az értéke.