WAF - 3. gyakorlat

A harmadik gyakorlat célja, hogy az eddigi megjelenítési funkcionalitás mellé adatbeviteli és szerkesztési lehetőséget is kínáljunk: legyen lehetőség új teendő listák létrehozására, a meglévők átnevezésére vagy akár törlésére. A listákba vehessünk fel új elemeket (kép feltöltésével), a meglévőket szerkeszthessük, helyezhessük át másik listába vagy törölhessük.

Controllerek és nézetek létrehozása

Hacsak a 2. gyakorlat kapcsán nem generáltuk le az összes CRUD műveletet a vezérlőkbe, akkor tegyük meg most. A `Controllers` mappára jobbklikkelve az *Add Controller* menüpont alatt adhatunk a projekthez új controller osztályt.

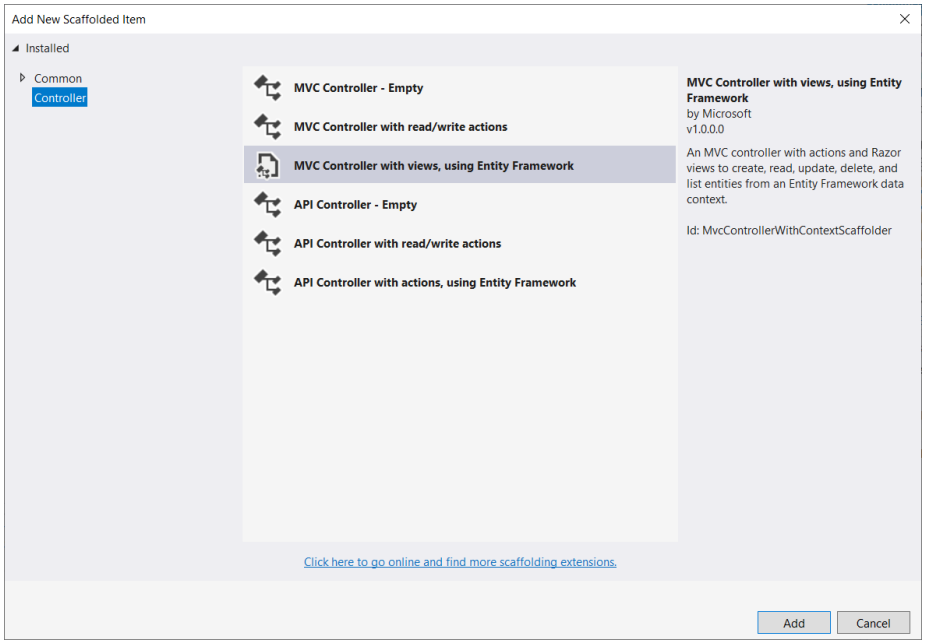

Figure 1: Controller osztály generálása

Válasszuk az *MVC Controller with views, using Entity Framework* opciót, ilyen módon a Visual Studio legenerálja nekünk egy Entity Framework entitás típushoz tartozó CRUD controllert (*scaffolding*). CRUD controllernek az olyan vezérlőket nevezzük, amelyek a CRUD (*create-read-update-delete*) perzisztálási tervezési mintára illeszkedve biztosítják a 4 alapműveletet.

Szükséges megadnunk, hogy melyik adatbázis kontextuson keresztül, melyik entitás típushoz szeretnénk vezérlő osztályt generálni, továbbá megadhatjuk,

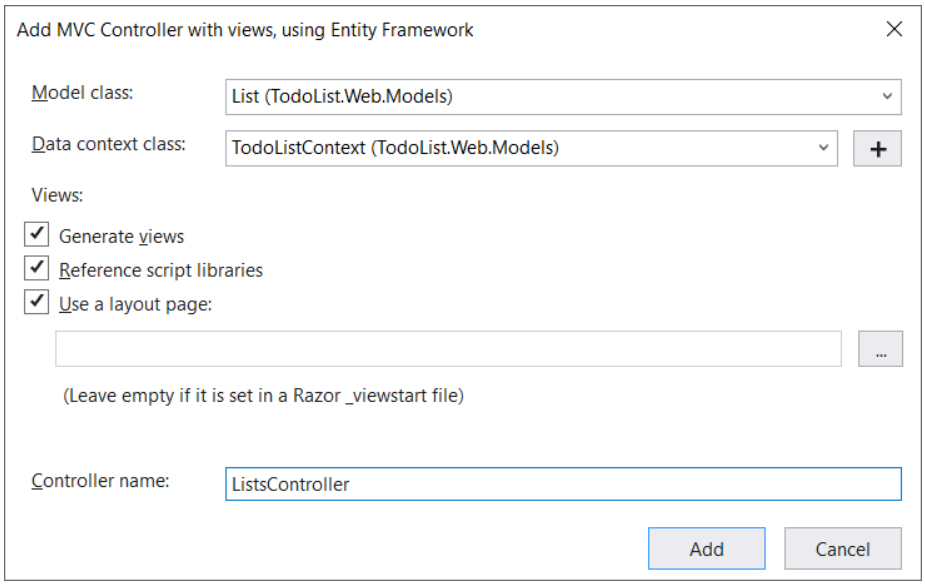

Figure 2: CRUD controller osztály generálása EF entitás osztály alapján

hogy:

- generáljon nézeteket is a Visual Studio;
- a nézetekben hivatkozza a kliens oldali validációhoz szükséges JavaScript kódot;
- milyen elrendező nézetet (*layout*) használjon. Amennyiben ezt üresen hagyjuk, a `_ViewStart.cshtml` fájlban megadott alapértelmezett beállítás jut majd érvényre.
- az elkészítendő controller osztály nevét.

A generálás után a `Controllers` mappában találjuk a vezérlő osztályt, a `Views` mappában pedig 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` és az `Item` entitáshoz is!

Fontos: a scaffolding előtt mindenképpen fordítsuk le az alkalmazásunkat, ugyanis az entitás típusok (és tulajdonságaik) az utolsó sikeres fordítás eredményéből kerülnek megállapításra.

Generált controllerek és nézetek áttekintése

Érdemes áttekinteni, hogy milyen akciókat hozott létre a generálás és mi ezek tartalma. A következő műveletekkel fogunk találkozni:

- *Index*: adott típusú entitás objektumok felsorolása
- *Details*: egyetlen entitás objektum részletes megjelenítése
- *Create*: új entitás objektum létrehozása
- *Edit*: létező entitás objektum szerkesztése
- *Delete*: létező entitás objektum eltávolítása

A szerkesztő műveletekhez (*Create*, *Edit*) két akció tartozik:

- az egyik, HTTP GET metóduson keresztül betölti a létrehozó vagy a szerkesztő űrlapot;
- a másik, HTTP POST metóduson keresztül fogadja az elküldött űrlap tartalmát és feldolgozza azt. Hiba esetén újból megjeleníti az űrlapot.

A törlés (*Delete*) akcióhoz egy kiegészítő, segéd akció is tartozik: a *DeleteConfirmed* akció a törlés megerősítésére szolgál. Az űrlapokat célzó XSRF támadások ellen a már generált kód is *anti forgery token* alkalmazásával védekezik.

A műveletek tartalmazzák a modellek propertyjeinek típusa és validációs annotációk szerinti ellenőrzéseket, a nem megfelelő bemenetet elutasítják és visszaküldik a felhasználónak az űrlapot javításra.

Futtassuk az alkalmazást és próbáljuk ki a működését: máris lehetőségünk van a teendő listák és az elemek alapvető megtekintésére és kezelésére (létrehozás, szerkesztés, törlés).

Controllerek és nézetek módosítása

Módosítsuk a generált controller osztályokat, hogy a `TodoListContext` példány helyett egy `TodoListService` objektummal rendelkezzen. Ezt a vezérlő osztály konstruktora átveheti, és az IoC tároló automatikusan befecskendezi majd. Írjuk át az összes akció törzsét úgy, hogy az adatbázissal ne közvetlenül, hanem a köztes *service* rétegen keresztül kommunikáljanak.

Emeljük át a 2. gyakorlat megoldásában elkészített teendő elem listázást. Ehhez egyszerűen a cseréljük le a `ListsController` osztály `Details` akcióját és a hozzá tartozó nézetet a korábban elkészítettre.

Helyezzünk el egy új linket a `ListsController` vezérlő `Details.cshtml` nézetében, amellyel egy új elem vehető fel a listába. Ezt irányítson a vezérlő (még létrehozandó) `CreateItem` akciójára és adja át a lista azonosítóját `id` argumentumaként.

```
<a asp-action="CreateItem" asp-route-id="@Model.Id">Add item</a>
```

Egészítsük ki a `ListsController` osztályt egy `CreateItem(int id)` akcióval, amely a paraméterként átvett listához ad hozzá egy új teendő elemet. Ehhez egyszerűen irányítsunk át az `ItemsController` vezérlő `Create` akciójára, de a `TempData`* tárolón keresztül adjuk át a lista azonosítóját.

```
public IActionResult CreateItem(int id)
{
    TempData["ListId"] = id;
    return RedirectToAction("Create", "Items");
}
```

Az `ItemsController` vezérlő `Create` akcióját módosítsuk, hogy a konstruált `SelectList`-ben az alapértelmezetten kiválasztott elem a megfelelő lista legyen. Ilyen módon új teendő elem hozzáadásakor a legördülő menüben alapértelmezetten a megfelelő lista kerül kiválasztásra, nem a legelső.

```
ViewData["Lists"] = new SelectList(_service.GetLists(), "Id", "Name", TempData["ListId"]);
```

Töröljük az `ItemsController` osztály `Index` és `Details` akciót, ugyanis ezekre nem lesz szükségünk. Töröljük a vonatkozó nézet fájlokat is, továbbá a többi nézetből az esetlegesen rájuk mutató hivatkozásokat (`<a>`).

* *Míg a `ViewData` és a `ViewBag` konténerek a vezérlők és a nézetek közötti adatátadásra szolgálnak (a nézetmodell mellett), addig a `TempData` konténer az akciók közötti adat átadást támogatja, akár vezérlők között is. Hasznos lehet akciók közötti átirányítás esetén, vagy amennyiben két oldalbetöltés között kell adatokat ideiglenesen megőrizni. (A háttérben rövid idejű munkamenetekkel (*session*) dolgozik. A munkamenetekkel és az állapotmegőrzéssel később foglalkozunk részletesen.)*

Képek feltöltése

Valósítsuk meg, hogy a teendő elemekhez létrehozáskor és szerkesztéskor legyen lehetőség egy képet felölteni. Mivel a megjelenítéskor PNG állományokra számítunk, ezért garantáljuk, hogy felölteni is csak ilyen formátumú képeket lehessen.

Az `ItemsController` vezérlő `Create` és `Edit` akcióit, valamint a vonatkozó nézeteket kell módosítanunk. Egészítsük ki a két metódus paraméterlistáját egy `IFormFile image` paraméterrel, egyben az `item` paraméterhez ne *bind*-oljuk tovább az `Image` propertyt. Amennyiben a feltöltött `IFormFile image` objektum nem `null` és a feltöltött fájl nem üres (a `Length` property értéke pozitív), akkor nyissunk egy új `MemoryStream` típusú adatfolyamat, amelybe betölthetjük a feltöltött fájlt. Innen a bájt tömböt már könnyedén az `item` entitás objektumba másolhatjuk.

```
using (var stream = new MemoryStream())
{
    image.CopyTo(stream);
    item.Image = stream.ToArray();
}
```

Az `Edit` akcióban ezen felül gondoskodnunk kell arról, hogy ha egy teendő elem módosításakor nem töltünk fel új képet, a frissítés ne írja felül az elemhez tartozó

korábbi képet (ha volt ilyen), mivel ilyen esetben az `IFormFile` paraméter értéke `null` lesz. Definiáljunk a *service* rétegben egy új metódust, amely az elem frissítésekor kihagyja a képet a propertyk közül.

```
_context.Update(item);
_context.Entry(item).Property("Image").IsModified = false;
_context.SaveChanges();
```

Gondoskodjunk róla, hogy a vezérlő az `IFormFile` paraméter értékétől függően hívja meg a megfelelő metódust a service-ből!

A nézetekben a `<form>` elemet ki kell egészítenünk a `enctype="multipart/form-data"` attribútummal, hogy a fájlfeltöltést az űrlap számára engedélyezzük. A fájl tallózó mezőt a következő módon definiálhatjuk:

```
<input asp-for="Image" type="file" accept="image/png" />
```

Valósítsuk meg, hogy szerkesztéskor jelenjen meg az aktuális kép a fájl tallózó beviteli mező felett, hasonlóan a `ListsController` osztály `Details.cshtml` nézetéhez. Ügyeljünk arra, hogy a kép megléte opcionális.

Megjegyzés: a beviteli mezőben a `accept="image/png"` attribútum csak kliens oldali megszorítást jelent a feltölthető fájlok típusára, és ilyen módon könnyen megkerülhető. Amennyiben mindenképpen garantálni szeretnénk, hogy csak PNG formátumú fájlt lehessen feltölteni, szerver oldali ellenőrzésre is szükség van, de ez nem triviális. Ellenőrizhetjük például, hogy a fájl a PNG állományoktól elvárt fejléccel kezdődik-e (az első 8 bájt rendre: `137 80 78 71 13 10 26 10`), azonban ez sem garantálja, hogy kép például nem sérült-e.

Erőforrások kötegelése és méretcsökkentése

Nagyobb webalkalmazások számos CSS és JavaScript állományt betölthetnek a weboldallal együtt. A probléma ezzel, hogy minden egyes fájl lekéréséhez új HTTP kapcsolatot kell felépíteni a kliens (a böngésző) és a szerver között. Így tulajdonképpen nem is a fájlok mérete, hanem azok mennyisége ronthatja már a weboldal betöltésének idejét.

A megoldást ezen erőforrás állományok kötegelt, egyetlen állományoként történő letöltése jelenti (angol szakszóval *bundling*). További optimalizációt jelenthet, ha a CSS és a JavaScript kódfájlokból a fölösleges whitespace karaktereket eltávolítjuk, ezzel csökkentve a méretüket (*minifying*).

Ez a feladat jól automatizálható, egy ASP.NET Core webalkalmazás esetében a `BuildBundlerMinifier` NuGet csomag lehet ebben segítségünkre, amely a projekt minden fordításakor a megadott konfiguráció szerint kötegeli és "tömöríti" a CSS és JavaScript állományainkat.

Adjuk hozzá a nevezett NuGet csomagot a projektünkhöz, valamint egy `bundleconfig.json` konfiguráció állományt a projektkönyvtár gyökerébe az

alábbi tartalommal:

```
[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  }
]
```

Ezzel megadtuk, hogy a `site.css` állományból kell elkészíteni a `site.min.css` *minified* CSS fájlt (több, mint 50%-os méret csökkenés). Több input fájl megadásával kötegelést is végezhetnénk.

Megadhatjuk azt is, hogy fejlesztés közben az eredeti CSS és JavaScript erőforrás állományokkal dolgozzunk (a forrásukat könnyen át tudjuk tekinteni hibakeresés esetén), de éles környezetben az optimalizált változatot használjuk. Ehhez a `_Layout.cshtml` elrendező nézetben a következő módon hivatkozzuk a fájlt:

```
<environment include="Development">
    <link rel="stylesheet" href="~/css/site.css" />
</environment>
<environment exclude="Development">
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
</environment>
```

Megjegyzés: hasonló elven a képfájlok betöltése is optimalizálható, amennyiben sok kis kép helyett az összes képet egy nagyobb állományra másoljuk és úgy töltjük le, ezeket *sprite*-oknak nevezzük. Erre azonban a CSS kódot is fel kell készíteni, így ezt most nem tárgyaljuk részletesebben.