ASP.NET Core MVC: authentikáció és authorizáció

A negyedik gyakorlat célja, hogy az eddigi funkcionalitásokat kiegészítsük felhasználókezeléssel és jogosultságkezeléssel. A felhasználók tudjanak regisztrálni, bejelentkezni, kijelentkezni. Látogatók (nem bejelentkezett felhasználók) a teendő listákat és azok elemeit csak megtekinteni tudják, módosítani, törölni, hozzáadni ne. A szerkesztési műveleteket csak a bejelentkezett felhasználók végezhessék el.

A felhasználókezeléshez az *ASP.NET Core Identity* keretrendszert fogjuk integrálni a TodoList projektbe.

1 Adatbázis kontextus, modell réteg

A felhasználói adatok tárolásának legegyszerűbb módja a lokális adatbázis entitásmodellen keresztül történő kezelése, ehhez a `Microsoft.AspNetCore.Identity.EntityFrameworkCore` NuGet csomagot adjuk hozzá a projektünkhöz.

A felhasználók tárolásához egy új entitás típust vegyünk fel a modell rétegbe, legyen a neve `ApplicationUser`. Ez az osztály a `Microsoft.AspNetCore.Identity` névtérben található `IdentityUser` típusból származzon, így tartalmazni fogja mindazokat a tulajdonságokat (pl. felhasználónév, jelszó), amelyeket az *ASP.NET Core Identity* elvár. (Részletes listáért ld. a dokumentációt.) Amennyiben további adatokat is szeretnénk minden felhasználóhoz tárolni, az `ApplicationUser` osztályt ezzel kiegészíthetnénk, de erre most nem lesz szükségünk.

A `TodoListDbContext` osztályt az `IdentityDbContext<ApplicationUser>` osztályból (névtér: `Microsoft.AspNetCore.Identity.EntityFrameworkCore`) származtassuk, így megörökli mindazokat az adattáblákat (`DbSet`-eket *code first* megközelítésben), amelyek a felhasználókezeléshez szükségesek. Ez a kezdetben szükségesnél több, összesen 7 entitáskollekciót fog az adatbázis-kontextusunkhoz adni, ugyanis támogatást nyújt pl. kétfaktoros authentikációhoz, RBAC (*role based access control*) alapú jogosultságkezeléshez, stb.

A kódunkat fordítsuk le, és az adatbázis rétegben történő változásokhoz generáljunk új migrációt a *Package Manager Console*-ban:

```
Add-Migration AddIdentity
```

A migráció alkalmazzuk is:

```
Update-Database
```

Ezt követően látni is fogjuk (pl. Visual Studioban *Sql Server Object Explorer* ablakban) az adatbázis sémájában az új táblákat, pl. az alapértelmezett nevén `AspNetUsers` tábla fogja a felhasználói rekordokat tartalmazni.

2 Program konfiguráció

A felhasználókezelést szükséges konfigurálnunk és az *IoC containerbe* regisztrálnunk, hogy használatba vehessük.

Konfiguráljuk az *ASP.NET Core Identity* keretrendszer által használandó alapértelmezett felhasználó és szerepkör (*role*) típusokat, utóbbira használjuk a keretrendszerben lévő `IdentityRole` ősosztályt, mivel nem szeretnénk most testre szabni. (Ebből is lehetne származtatni, hasonlóan az `IdentityUser`-hez.) Az `AddEntityFrameworkStores()` meghívásával regisztráljuk, hogy a `TodoListDbContext` típus fogja az identitásokat kezelni.

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequiredLength = 3;
    options.Password.RequiredUniqueChars = 0;
    options.Password.RequireLowercase = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
})
.AddEntityFrameworkStores<TodoListDbContext>();
```

A `Program` osztályban engedélyezzük a felhasználókezelést a `app.UseAuthentication()` eljárás meghívásával. Fontos, hogy az alábbi metódushívások sorrendje pontosan a következő legyen:

```
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
```

3 Nézetmodell réteg

Vegyünk fel egy `LoginViewModel` nézetmodell osztályt, amely a bejelentkezéskor elküldendő adatokat (felhasználónév, jelszó) tartalmazza. Mindkettő legyen kötelezően kitöltendő és a jelszó esetében attribútum annotációval jelezzük, hogy a megjelenítéskor jelszó mezőt generáljon az űrlap (`[DataType(DataType.Password)]`).

Vegyünk fel egy `RegisterViewModel` nézetmodell osztályt, amely a regisztrációkor elküldendő adatokat (felhasználónév, jelszó, jelszó megerősítése) tartalmazza. Mindhárom mező legyen kötelezően kitöltendő, és a jelszavak esetében itt is gondoskodjunk a jelszavak biztonságos megkéréséről. A `Compare` validátorral ellenőrizzük, hogy a megadott két jelszó egyezik-e.

4 Vezérlő réteg

Az *ASP.NET Core Identity* keretrendszerben a felhasználók bejelentkeztetését, és az azonosság nyilvántartását a `SignInManager` osztály felügyeli; a felhasználók kezelését a `UserManager` osztály végzi, ezek a `Microsoft.AspNetCore.Identity` névtérben találhatóak.

Vegyünk fel egy új vezérlő osztály `AccountController` néven, amely az authentikációért lesz felelős. Az osztály rendelkezzen egy `SignInManager<ApplicationUser>` és egy `UserManager<ApplicationUser>` példánnyal is, amelyek a konstruktoron keresztül az *IoC containerből* kerülnek befecskendezésre.

```
public class AccountController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;

    public AccountController(UserManager<ApplicationUser> userManager,
                             SignInManager<ApplicationUser> signInManager)
    {
        _userManager = userManager;
        _signInManager = signInManager;
    }

    // akciók
}
```

A következő akciókat valósítsuk meg:

- Definiáljunk egy `RedirectToLocal()` nevű metódust, amely egy opcionális `returnUrl` stringet vár paraméterül. Ezzel a metódussal gondoskodjunk róla, hogy bejelentkezés után a megfelelő, oldalon

belüli akcióra irányítsuk a felhasználót. Használjuk az `Url.IsLocalUrl()` metódust!
- A bejelentkezéshez a definiáljunk egy `Login()` metódust az űrlap kezdeti megjelenítésére, amely HTTP GET metóduson keresztül kiszolgálható. Paraméterként fogadjon egy opcionális `returnUrl` stringet, amely annak az oldalnak az URL-jét fogja tartalmazni, amelyre a bejelentkezés után visszairányítjuk a felhasználót. A `returnUrl`-t tároljuk a `ViewBag`ben.
- A már beküldött űrlap feldolgozására definiáljuk a `Login()` akció túlterhelését, amely HTTP POST metóduson keresztül kiszolgálható. Ez paraméterként fogadjon egy `LoginViewModel` példányt és egy `returnUrl` stringet. Amennyiben a nézetmodell valid, az `AccountController` osztályban tárolt `SignInManager` példány `PasswordSignInAsync()` eljárásával végezhetjük el az authentikációt. Eredményként egy `SignInResult` típusú struktúrát kapunk, amelynek `Succeeded` adattagja tartalmazza az a hitelesítés sikerességét. Sikeres authentikáció esetén az információ munkamenetben történő tárolása és szükséges sütik elhelyezése a kliens oldalon a keretrendszer feladata. Sikertelen bejelentkezés esetén az űrlapot tartalmazó nézetet küldjük vissza a felhasználónak.
- A bejelentkezéshez hasonlóan építsük fel a regisztrációt: definiáljuk a `Register()` akció két túlterhelését, amelyek közül az egyik egy `RegisterViewModel` nézetmodellt fogadjon paraméterként és csak HTTP POST metódussal legyen meghívható. Új felhasználó regisztrációját `UserManager` osztály `CreateAsync()` eljárásával végezhetjük el*, átadva paraméterként a létrehozott `ApplicationUser` példányt és a jelszót. A jelszó sózásáért, hasításáért a keretrendszer felel. Mindkét túlterhelés fogadjon egy `returnUrl` string paramétert, amit a bejelentkezéshez hasonlóan kezeljen.
- A kijelentkezéshez definiáljuk a `Logout()` akciót. A kijelentkeztetést a `SignInManager` osztály `SignOutAsync()` eljárásával végezhetjük el.

***Megjegyzés:** fontos, hogy a felhasználót ne mi adjunk közvetlenül az `AspNetUsers` táblához, hiszen azzal megkerülnénk a *ASP.NET Core Identity* keretrendszert, így a jelszó biztonságos hasítását is.

A `ListController` és az `ItemsController` vezérlő osztályokat lássuk el az `[Authorize]` attribútummal; az előbbi `Index` és `Details`, valamint utóbbi `DisplayImage` akcióját az `[AllowAnonymous]` attribútummal (`Microsoft.AspNetCore.Authorization` névtér). Így a szerkesztési műveletekhez csak bejelentkezett felhasználó férhet hozzá.

5 Nézet réteg

Definiáljuk a `Views/Account` könyvtárban a `Login.cshtml` és a `Register.cshtml` nézeteket, amelyek nézetmodellje `LoginViewModel`, valamint `RegisterViewModel` típusú legyen. A nézetek feladata ezen nézetmodellek űrlap megjelenítése a kitöltéshez.

A `Views/Shared/_Layout.cshtml` elrendező nézetben a látogató bejelentkezettségi státuszának megfelelően jelenítsük meg:

- a felhasználónevét és egy linket a kijelentkezésre;
- egy linket a bejelentkezésre, valamint a regisztrációra.

Hozzunk létre egy parciális nézetet: `Views/Shared/_PartialLogin`! Az authentikáció állapotától függő nézet kódját helyezzük ide.

A felhasználókezeléshez a nézet rétegben is kérhetjük 1-1 `SignInManager` és `UserManager` példány befecskendezését az *IoC containerből*:

```
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager
```

Ezt követően már könnyedén ellenőrizhető a látogató státusza:

```
@if (SignInManager.IsSignedIn(User)) { ... }
```

Bejelentkezett státusz esetén lekérhető a felhasználó neve:

```
@UserManager.GetUserName(User)
```

A parciális nézetet illesszük be az elrendező nézetbe, egy legördülő menübe a headerben.

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
```

```
        <li class="nav-item">
            <a asp-area="" asp-controller="Lists" asp-action="Index">Lists</a>
        </li>
    </ul>
    @await Html.PartialAsync("_LoginPartial")
</div>
```