Eötvös Loránd Tudományegyetem
Informatikai Kar

Webes alkalmazások fejlesztése

6. előadás

Autentikáció és autorizáció (ASP.NET Core)

Cserép Máté
mcserep@inf.elte.hu
http://mcserep.web.elte.hu

Autentikáció és autorizáció

Autentikáció

- Egy weblap felhasználó kezelése számos elemmel rendelkezik, amelyeket biztonságosan kell megvalósítanunk:
  - *regisztráció, adatmódosítás*
  - *bejelentkezés*, *kijelentkezés*, *automatikus bejelentkeztetés*
  - *elfelejtett jelszó/azonosító kezelése*
    - mivel a jelszót mindig kódolva tároljuk, ezért elfelejtett jelszó esetén a felhasználónak biztosítani kell új jelszó létrehozását (amennyiben meggyőződtünk az azonosságáról)
  - *extra funkcionalitások*: felhasználói csoportok, láthatóságok kezelése, e-mailben történő megerősítés …

Autentikáció és autorizáció

ASP.NET Identity

- A felhasználói autentikáció összetettsége miatt az ASP.NET egy kész rendszert biztosít számunkra, ez az *Identity*
  - biztosítja a felhasználó-kezeléshez szükséges funkciókat az adatkezeléstől a felületig
  - a felhasználói adatok tárolására/elérésére számos lehetőséget ad, használhatunk lokális megoldásokat (pl. adatbázis, *Windows Authentication*), más szolgáltatások fiókkezelését (pl. *Microsoft Account*, *Twitter*, *Facebook*), általánosságban támogatott az *OAuth* és *OpenId*.
  - az alap funkcionalitás a **`Microsoft.AspNetCore.Identity`** programkönyvtárból (*NuGet* csomagból) érhető el, az adattárolást további csomagok biztosítják.

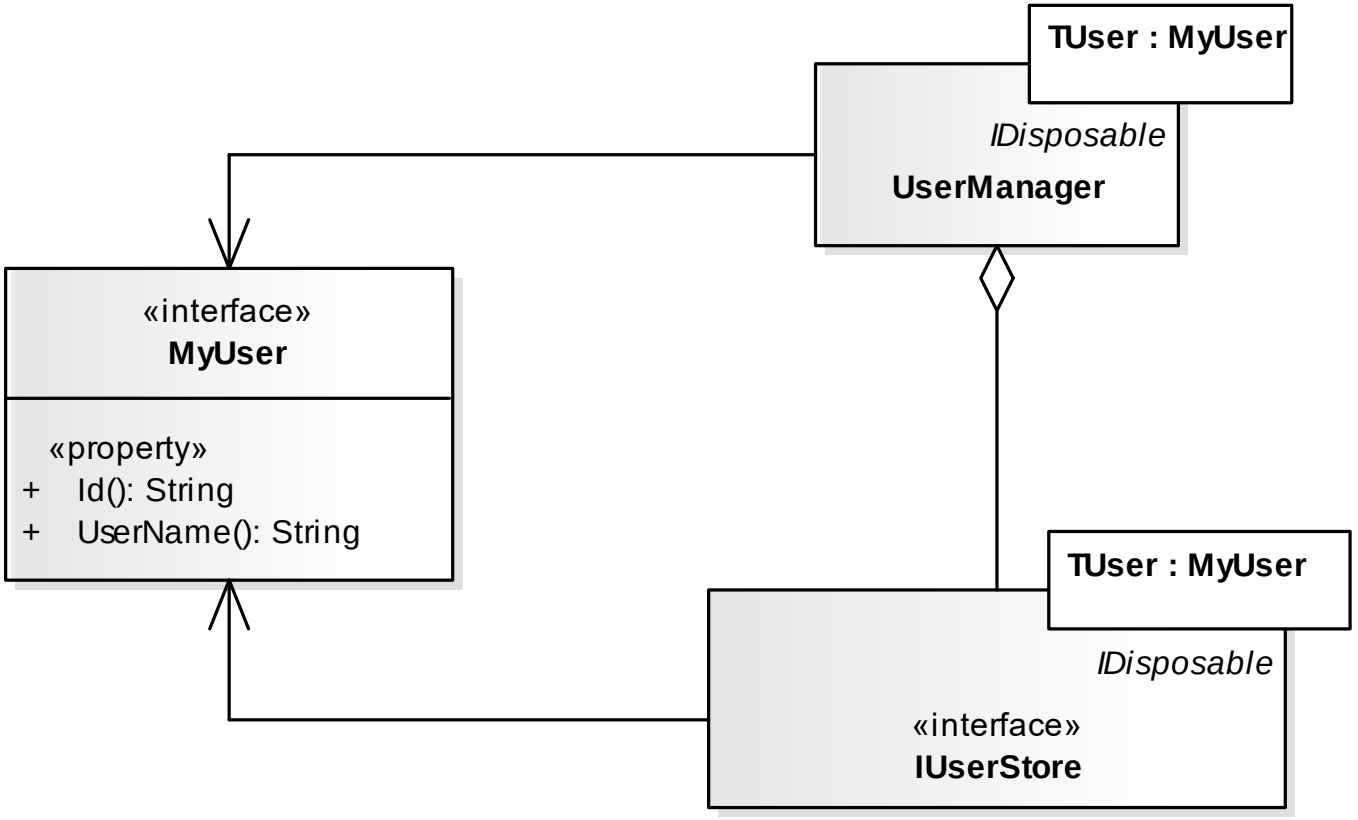
Autentikáció és autorizáció

Felhasználók kezelése

- A felhasználók kezelését a **`UserManager`** osztály felügyeli, amely bármilyen speciális felhasználótípust kezelni tud, műveletei az aszinkron végrehajtást támogatják
  - a felhasználót regisztrálhatjuk (**`CreateAsync`**), azonosíthatjuk (**`FindByIdAsync`**, **`FindByEmailAsync`**, stb.), módosíthatjuk (**`ChangePasswordAsync`**), stb.
  - a felhasználó tetszőleges típus lehet, amelyet reprezentálja a szükséges információt (jelszó, e-mail cím, stb.)
  - a menedzser osztály egy adattáron (**`IUserStore`**) keresztül kezeli a felhasználókat, vagyis ez szabja meg a konkrét felhasználó kezelési megoldást
  - minden osztály a felhasználó típusra specializált

Autentikáció és autorizáció

Felhasználók kezelése



Autentikáció és autorizáció

Felhasználók kezelése

- Pl.:

```
public class MyUser { … }
    // a felhasználó típusa
public class MyStore : IUserStore<MyUser> { … }
    // az adattár típusa, a felhasználóra
    // specializálva

UserManager<MyUser> manager =
    new UserManager<MyUser>(new MyStore(), …);
    // felhasználókezelő, a felhasználóra
    // specializálva

MyUser user = await manager.FindByNameAsync(name);
    // felhasználó keresése (név, jelszó alapján)
```

Autentikáció és autorizáció

Adattárolás entitásmodellel

- A felhasználói adatok tárolásának legegyszerűbb módja lokális adatbázis entitásmodellen keresztül történő kezelése (**`Microsoft.AspNetCore.Identity.EntityFrameworkCore`**)
  - az entitásmodell (**`IdentityDbContext`**) névvel és jelszóval ellátott felhasználókat tud kezelni (**`IdentityUser`**)
    - az **`IdentityUser`** specializálható, bármilyen tulajdonsággal bővíthető a felhasználó
    - a kódban megadott adatok alapján hozza létre az adatbázis szerkezetet (*code first*)
    - biztosítja a jelszavak kódolását (időfüggő sózással)
  - az entitásmodell a csatolt **`UserStore`** osztállyal használható

Autentikáció és autorizáció

Adattárolás entitásmodellel

- Pl.:

```
IndentityDbContext<IdentityUser> context = new …;
    // adatbázis entitásmodell
UserStore<IdentityUser> store =
    new UserStore<IdentityUser>(context);
    // adattár
UserManager<IdentityUser> userManager =
    new UserManager<IdentiyUser>(store);
    // felhasználókezelő

IdentityUser user = await userManager
                              .FindByNameAsync(name);
    // felhasználó keresése
```

Autentikáció és autorizáció

Felhasználók hitelesítése

- A felhasználók bejelentkeztetését, és az azonosság nyilvántartását a **`SignInManager`** osztály felügyeli, amely szintén a felhasználó típusára specializált és aszinkron műveleteket biztosít
  - a felhasználónév-jelszó páros alapú direkt bejelentkeztetést a **`PasswordSignInAsync`**, a kijelentkeztetést a **`SignOutAsync`** műveletek kezelik, amelyeknek megadható, hogy perzisztens legyen-e a bejelentkezés (azaz jegyezze meg az adatokat), valamint hibás adatok esetén kizárásra kerüljön-e a felhasználó (*lockout*)
  - támogatja a külső *autentikációs providereken* keresztüli hitelesítést (**`ExternalLoginSignInAsync`**) és a két faktoros autentikációt (**`TwoFactorAuthenticatorSignInAsync`**)

Autentikáció és autorizáció

Felhasználó bejelentkeztetés

- Bejelentkeztetés:

```
public async Task<IActionResult> Login(…) {
    // …
    var result = await signInManager
        .PasswordSignInAsync(userName, password,
                             remember, false);
    // bejelentkeztetés
    if(result.Succeeded) { … }
}
```

- Kijelentkeztetés:

```
public async Task<IActionResult> Logout() {
    await signInManager.SignOutAsync();
}
```

Autentikáció és autorizáció

Hitelesítés konfigurációja

- Annak érdekében, hogy a funkciók a rendelkezésünkre álljanak, az alkalmazás számára konfigurálnunk kell az **`Identity`** működését
  - ezt a **`Startup`** osztály **`ConfigureServices`** művelete végzi, amely paraméterben megkapja a szolgáltatások kollekcióját (**`IServiceCollection`**), amely bővíthető és beállítható
  - az **`AddIdentity<MyUser>`** segítségével felvehetjük az autentikációs szolgáltatást a gyűjteménybe
  - az **`AddUserStore<MyStore>`** megadja az alkalmazandó felhasználói adattár típusát
  - az **`AddEntityFrameworkStores<MyDbContext>`** *Entity Framework Core* alapú adattárolást állít be és megadja a használandó adatbázis entitás kontextus típusát

Autentikáció és autorizáció

Hitelesítés konfigurációja

- Pl.:

```
services.AddIdentity<MyUser>()
    .AddEntityFrameworkStores<MyDbContext>()
    .AddDefaultTokenProviders();

// beállítások
services.Configure<IdentityOptions>(options => {
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = false;
    options.Password.RequiredUniqueChars = 3;

    options.User.RequireUniqueEmail = true;
});
```

Autentikáció és autorizáció

Hitelesítés konfigurációja

- A konfigurált felhasználókezelést engedélyeznünk is kell, ezt ezt a **`Startup`** osztály **`Configure`** műveletében végezhetjük el:

```
app.UseAuthentication();
```

- Fontos azonban, hogy a **`Configure`** eljárásban az alábbi utasítások sorrendje pontos az alábbi legyen:

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

Autentikáció és autorizáció

Hozzáférés korlátozás

- Az így bejelentkezett felhasználót szintén a **`HttpContext`** osztályon keresztül kezelhetjük
  - a **`User.Identity.IsAuthenticated`** tulajdonság jelzi, hogy van-e bejelentkezett felhasználó
  - a felhasználó nevét a **`User.Identity.Name`** tulajdonságon keresztül érhetjük el
  - pl.:

```
if (User.Identity.IsAuthenticated) {
    IdentityUser user = await userManager.
        FindByNameAsync(User.Identity.Name);
        // lekérjük a bejelentkezett felhasználót
}
```

Autentikáció és autorizáció

Példa

Feladat: Valósítsuk az utazási ügynökség weblapjának felhasználó kezelési funkcióját.

- a *ASP.NET Core Identity* segítségével valósítjuk meg, a **`TravelAgencyContext`** entitás kontextus típus specializálja **`IdentityDbContext`** típust. A **`Guest`** az **`IdentityUser`** specializációja, teljes névvel és címmel kiegészítve.
- A felhasználókezelést az **`AccountService`** szolgáltatás helyett az integrált **`UserManager`** és **`SignInManager`** szolgáltatásokra bízzuk.
- egészítsük ki a **`Startup`** osztályt a konfigurációhoz
- továbbra is meghagyjuk a regisztráció nélküli foglalást, ilyenkor automatikusan regisztrálunk egy új felhasználót

Autentikáció és autorizáció

Hozzáférés korlátozás

- Erőforrások hozzáférése korlátozható a felhasználókra
  - az **`Authorize`** attribútum alkalmazható vezérlőkre, illetve akcióműveletekre, így csak megfelelő autentikáció után vehető igénybe az erőforrás
  - pl.:

```
[Authorize] // csak a bejelentkezett felhasználó
            // férhet hozzá
public IActionResult ManageAccount(){ … }
```

  - a hozzáférés korlátozható felhasználókra (**`Users`**) és szerepekre (**`Roles`**)
  - teljes vezérlő korlátozása esetén felszabadíthatunk műveleteket (az **`AllowAnonymous`** attribútummal)

Autentikáció és autorizáció

Hozzáférés korlátozás

- Pl.:

```
[Authorize] // érvényes az összes műveletre
public class AccountController : Controller
{
    public IActionResult ManageAccount(){ … }

    [Authorize(Roles = "admin")]
    // ehhez csak rendszergazda férhet hozzá
    public IActionResult ManageAllAccounts(){ … }

    [AllowAnonymous] // ehhez bárki hozzáférhet
    public IActionResult Login(){ … }
    …
}
```

Autentikáció és autorizáció

Biztonságos kommunikáció

- Lehetőségünk van biztonságos kommunikációra, az üzenetek titkosítására *Transport Layer Security* (*TLS*, *SSL*) segítségével
  - kizárja a kommunikáció lehallgatását, a munkamenet lopást (jó eséllyel), beállítása a webszerverre tartozik
  - a weblap elvárhatja a biztonságos csatornát egy erőforrásra (a **`RequireHttps`** attribútummal), vagy ellenőrizheti a meglétét a **`Request.IsHttps`** tulajdonsággal
- Pl.:

```
[Authorize]
[RequireHttps] // csak biztonságos adatközlés
               // mellett érhető el
public class AccountController : Controller { … }
```