Eötvös Loránd Tudományegyetem
Informatikai Kar

Webes alkalmazások fejlesztése

7. előadás

Autentikáció és autorizáció (ASP.NET)

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Autentikáció és autorizáció

Autentikáció

- Egy weblap felhasználó kezelése számos elemmel rendelkezik, amelyek biztonságosra 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*), vagy más szolgáltatások fiókkezelését (pl. *Microsoft Account*, *Twitter*, *Facebook*)
  - az alap funkcionalitás az `AspNet.Identity.Core` programkönyvtárból (*NuGet* csomagból) érhető el, az adattárolást további csomagok biztosítják

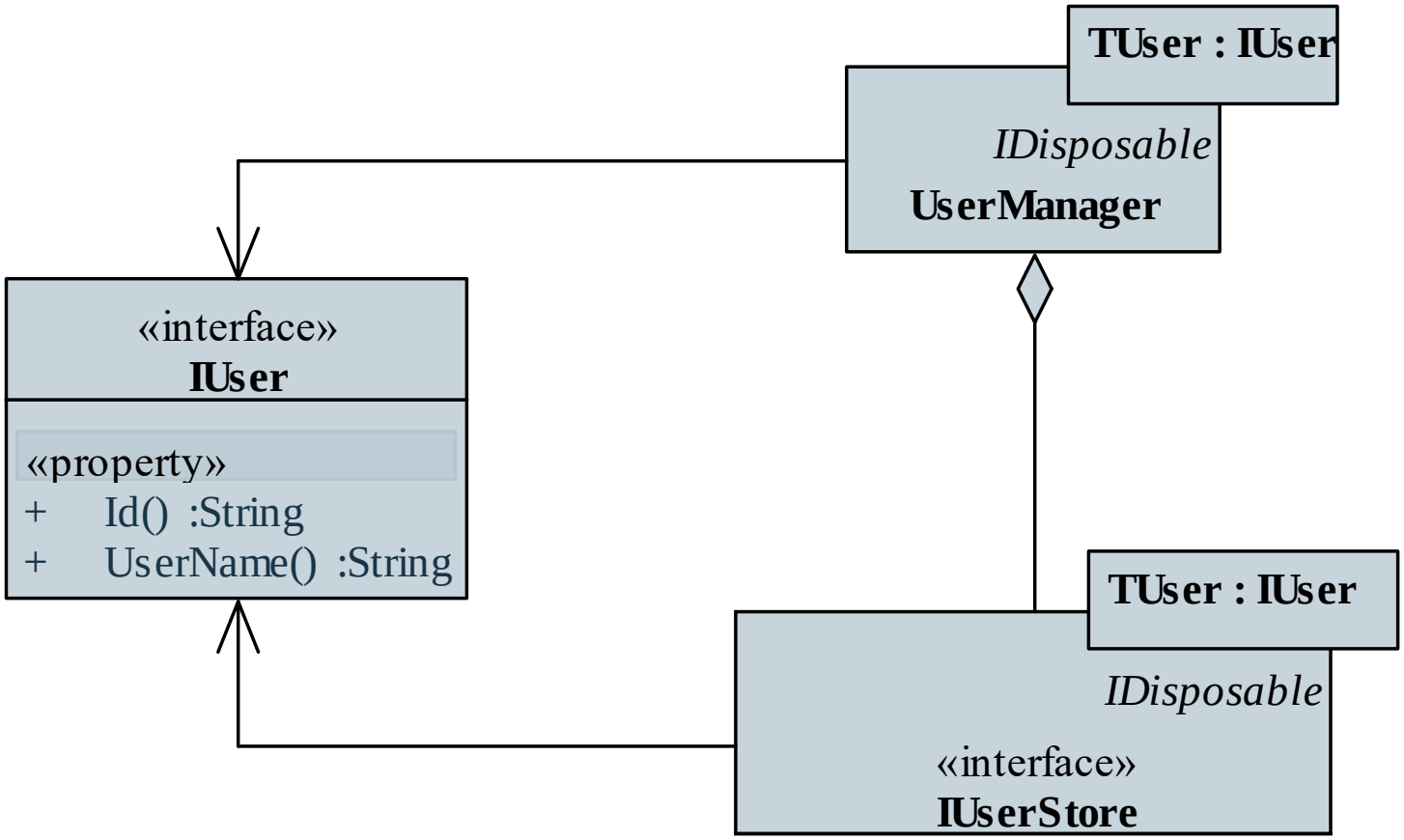
Autentikáció és autorizáció

Felhasználók kezelése

- A felhasználók kezelését a **`UserManager`** osztály felügyeli, amely bármilyen speciális felhasználótípust kezelni tud
  - a felhasználót regisztrálhatjuk (**`Create`**), azonosíthatjuk (**`Find`**), módosíthatjuk (**`ChangePassword`**), …
  - a felhasználó egy **`IUser`** interfészt megvalósító osztály, amelyet bármilyen további információval (jelszó, e-mail cím, …) kiegészíthetünk
  - 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 : IUser { … }
   // 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 = manager.Find(name, password);
   // felhasználó keresése (név, jelszó alapján)
```

Autentikáció és autorizáció

Adattárolás entitásmodellel

- A felhasználói adatok tárolásának legegyszerűbb módja lokális adatbázis entitásmodellen keresztül történő kezelése (**`AspNet.Identity.EntityFramework`**)
  - az entitásmodell (**`IdentityDbContext`**) névvel és jelszóval ellátott felhasználókat tud kezelni (**`IdentityUser`**)
    - szintén 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> manager =
   new UserManager<IdentiyUser>(store);
   // felhasználókezelő

IdentityUser user = manager.Find(name, password);
   // felhasználó keresése
```

Autentikáció és autorizáció

Adattárolás entitásmodellel

- Az adatokat célszerű egy olyan adatbázisban eltárolni, amelyet direkt autentikációs célokra használunk
  - az **`IdentityDbContext`** konstruktorában megadjuk az adatbázis kapcsolódási adatait (*connection string*), amelyet a konfigurációban (**`web.config`**) helyezünk el
  - alapértelmezett a **`DefaultConnection`**, ezt nem kell külön megadnunk, pl.:

```
<connectionStrings>
    <add name="DefaultConnection"
         connectionString="…" providerName="…" />
    …
</connectionStrings>
```

Autentikáció és autorizáció

OWIN

- A felhasználó egységes bejelentkeztetését, és az azonosság nyilvántartását az *OWIN* (*Open Web Interface for .NET*) programozási felület biztosítja (**`Microsoft.Owin`** csomag)
  - a be- és kijelentkeztetés a **`HttpContext`** osztály **`GetOwinContext().Authentication`** tulajdonsága segítségével történik (**`Microsoft.Owin.Host.SystemWeb`** csomag)
  - a bejelentkeztetést a **`SignIn`**, a kijelentkeztetést a **`SignOut`** műveletek kezelik, amelyeknek megadható az azonosítás nyilvántartás módja (pl. süti alapú), illetve, hogy perzisztens legyen-e a bejelentkezés (azaz jegyezze meg az adatokat)
  - a bejelentkezéshez követelés szükséges (**`ClaimsIdentity`**)

Autentikáció és autorizáció

Felhasználó bejelentkeztetés

- Pl.:

```
ClaimsIdentity claims = manager.CreateIdentity(
    user,
    DefaultAuthenticationTypes.ApplicationCookie);
    // felhasználói követelesések létrehozása, süti
    // alapú kezeléssel
HttpContext.GetOwinContext().Authentication.
    SignIn(claims);
    // bejelentkeztetés
…
HttpContext.GetOwinContext().Authentication.
    SignOut(DefaultAuthenticationTypes.
            ApplicationCookie);
    // kijelentkeztetés
```

Autentikáció és autorizáció

Felhasználó bejelentkeztetés

- Annak érdekében, hogy a funkciók a rendelkezésünkre álljanak, az alkalmazás számára konfigurálnunk kell az **`Identity`** működését
  - ezt alapértelmezetten a **`Startup`** osztály **`Configure`** művelete végzi, amely paraméterben megkapja az autentikáció konfigurációs objektumát (**`IAppBuilder`**)
  - a konfigurációs objektumnak kell megadnunk a megfelelő autentikációs lehetőségeket, pl.:

```
public void Configure(IAppBuilder app){
    app.UseCookieAuthentication(
        new CookieAuthenticationOptions { … });
        // süti alapú azonosítás engedélyezése
}
```

Autentikáció és autorizáció

Hozzáférés korlátozás

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

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

Autentikáció és autorizáció

Példa

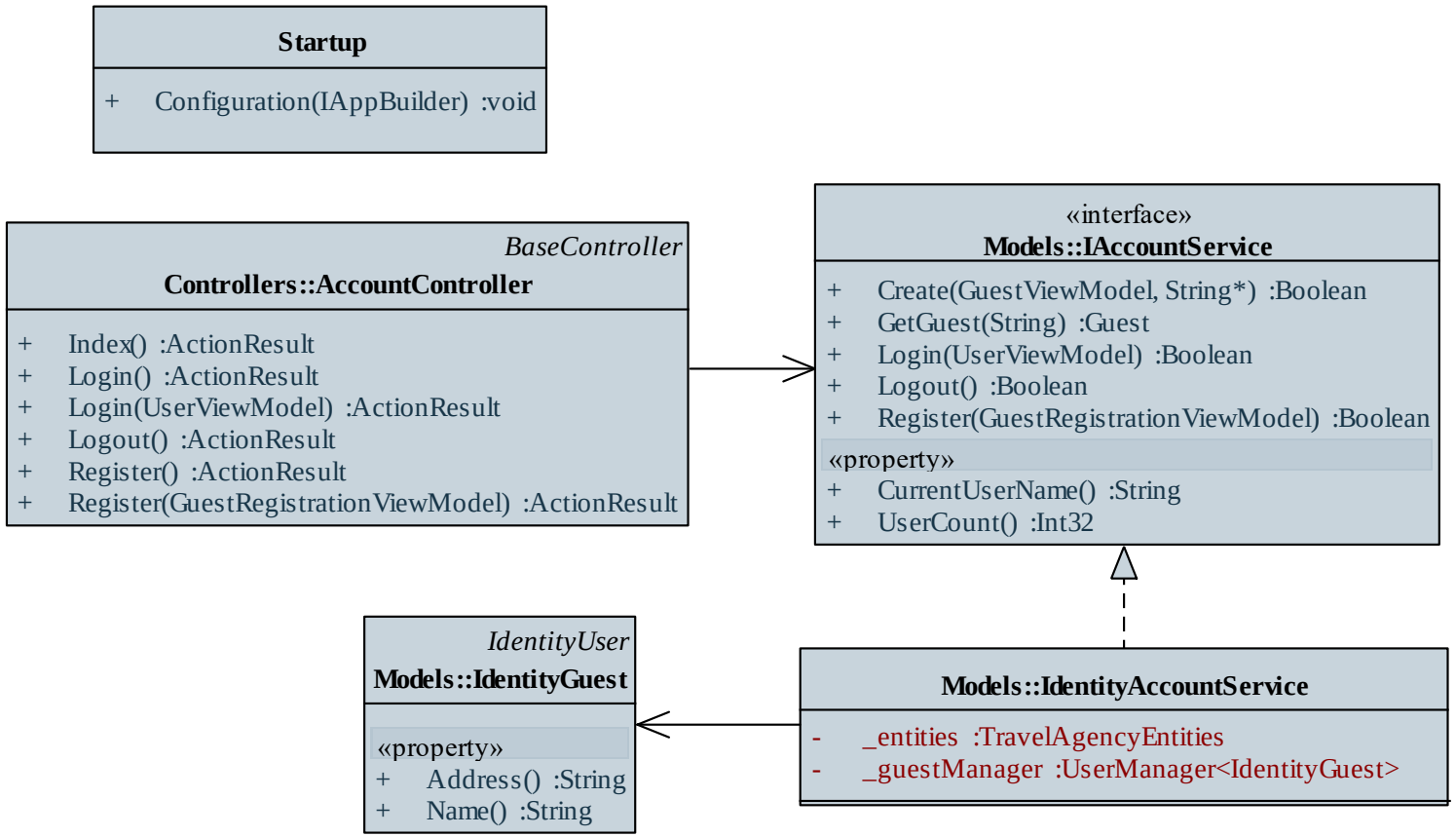
Feladat: Valósítsuk az utazási ügynökség weblapjának felhasználó kezelési funkcióját.

- a felhasználókezelést Identity és OWIN segítségével valósítjuk meg, a felhasználói adatok a **`TravelAgencyAuthentication`** adatbázisban tároljuk (egy új kapcsolattal a konfigurációban)
- megvalósítunk egy új változatát a felhasználókezelésnek (**`IdentityAccountService`**), az alap felhasználói adatokat kiegészítjük a címmel és a teljes névvel (**`IdentityGuest`**)
- felvesszük a **`Startup`** osztályt a konfigurációhoz
- továbbra is meghagyjuk a regisztráció nélküli foglalást

Autentikáció és autorizáció

Példa

Tervezés:



Autentikáció és autorizáció

Példa

Megvalósítás (`IdentityAccountService.cs`*):*

```
public Boolean Login(UserViewModel user) {
    …
    // megkeressük a felhasználót
    IdentityGuest identityGuest =
        _guestManager.Find(user.UserName,
                           user.UserPassword);
    if (identityGuest == null)
        return false;

    // ha valaki már bejelentkezett,
    // kijelentkeztetjük
    HttpContext.Current.GetOwinContext()
        .Authentication.SignOut(…);
```

Autentikáció és autorizáció

Példa

Megvalósítás (`IdentityAccountService.cs`*):*

```
    // bejelentkeztetjük az új felhasználót
    ClaimsIdentity claimsIdentity =
      _guestManager.CreateIdentity(
           identityGuest, …);
    HttpContext.Current.GetOwinContext().
       Authentication.SignIn(
          new AuthenticationProperties {
             IsPersistent = rememberMe },
          claimsIdentity);
       // perzisztens bejelentkezést állítunk be,
       // amennyiben megjegyzést kért
    …
}
```

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 ActionResult 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 ActionResult ManageAccount(){ … }

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

    [AllowAnonymous] // ehhez bárki hozzáférhet
    public ActionResult 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.IsSecureConnection`** tulajdonsággal
- Pl.:

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