.NET Okruženje

Vjezba 11 – Autentikacija i autorizacija

Sadržaj

Autorizacija i autentikacija

Pri kreiranju ASP.NET Core MVC aplikacije, automatski se otvara mogućnost za korištenje ugrađenog user membership modula – Owin autentikacijskog modula. Navedeni autentikacijski modul je vrlo dobro koncipiran, te je moguće na relativno jednostavan način proširiti osnovne funkcionalnosti po želji. Dodatno, moguće je i potpuno samostalno implementirati sve potrebne autentikacijske i autorizacijske protokole, no to izlazi izvan dosega ovog materijala.

Autentikacija

Autentikacija se sastoji od:

- Traženja od korisnika da unese korisničko ime i lozinku, čime dokazuje da je upravo taj korisnik za kojeg se iskazuje.
    - Owin mehanizam omogućava jednostavno dodavanje popularnih otvorenih mehanizama za autentikaciju kao što je Google, Facebook, Twitter i Microsoft.
    - Automatski je omogućena i samostalna registracija korisnika
- Pri izvršavanju neke akcije controllera, moguće je definirati da samo ulogirani korisnici imaju pravo izvršavati tu akciju. Ukoliko postojeći korisnik nije prijavljen (anoniman je), tada se korisnika preusmjeri na stranicu za login, i po uspješnom loginu ga se vrati gdje je bio

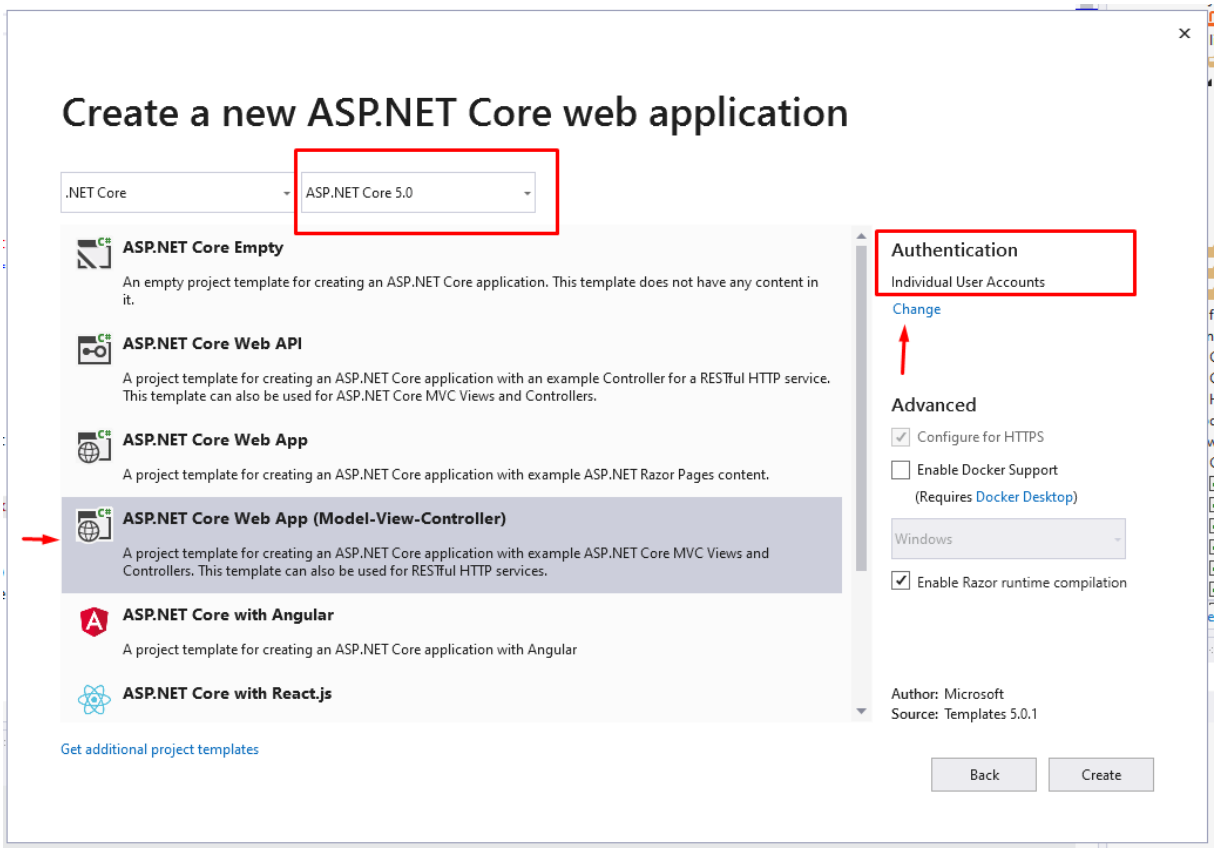
Po inicijalnim postavkama, Owin mehanizam radi na način da samostalno kreira posebnu bazu podataka i u njoj sprema podatke o korisnicima, njihovim ulogama i ostalim potrebnim podacima za autentikaciju i autorizaciju. Međutim, takva arhitektura nije baš optimalna jer je često potrebno povezivati domenski specifične entitete s entitetima koji predstavljaju korisnika (User). Također, često želimo dodati specifična polja uz User entitet (primjerice, JMBAG ako su korisnici studenti). Iz tog razloga, najoptimalnija opcija za omogućavanje autentikacije koja će biti dovoljno proširiva i fleksibilna, a opet zadržati automatski dobiveni set funkcionalnosti je da naša DbContext klasa naslijedi iz IdentityDbContext klase te samim time naslijedi i sve potrebne tablice u bazi podataka. Da bi to ispravno konfigurirali, potrebno je napraviti niz manjih izmjena na postojećem projektu.

Neke od izmjena su:

1. U Models projekt dodati **AppUser** klasu koja nasljeđuje **IdentityUser**
2. Klasa `ClientManagerDbContext` treba naslijediti klasu **IdentityDbContext<AppUser>**
3. Instalirati paket *Microsoft.AspNetCore.Identity.EntityFrameworkCore* u DAL projekt i paket *Microsoft.Extensions.Identity.Stores* u Model projekt, te *Microsoft.AspNetCore.Identity.UI* u Web projekt
4. Pokrenuti migracijske skripte i sinkronizaciju baze
    a. Osigurati da se koristi ispravni *startup project* i ispravni *project name*
5. Modificirati Startup.cs (vidi dolje)
    a. Za razlike može poslužiti diff ckecker alat (ima i raznih online rješenja)
6. Uočiti folder „Areas“ (vidi dolje)
7. Uočiti razliku u „_Layout.cshtml“ (vidi dolje)

Postoji još nekoliko manjih izmjena, ali njih je najbolje uočiti na način da se kreira nova ASP.NET Core Web aplikacija (verzija ASP.NET 5.0) i postavi se autentikacija prilikom kreiranja te se prouče razlike u

strukturi datoteka i Startup.cs datoteci. Kreiranje nove aplikacije preporučeno je sa sljedećim postavkama:



Način autentikacije koji je u ovom kontekstu potreban je „Individual User Accounts“, te ga je potrebno izabrati klikom na „Change Authentication“.

Važno: pogledati gdje se sve koristi **IdentityUser** klasa u novokreiranom projektu, i potrebno ju je zamijeniti sa **AppUser** u vježbi. *Hint: ne zaboraviti _LoginPartial.*

Zadatak 11.1

Prema gornjim koracima (pažljivo ih pročitati!) postići to da se aplikacija uredno pokreće, te da ranije implementirane funkcionalnosti (CRUD operacije nad klijentima) rade uredno.

- Obaviti registraciju te prijavu s novokreiranim korisnikom

OAuth

Model Owin autentikacije automatski podržava i autentikaciju OAuth mehanizmima kao što je primjerice Google OAuth2 mehanizam ili Facebook i slični poznati i popularni servisi. Kako bi omogućili ovakav način autentikacije u aplikaciji, potrebno je napraviti sljedećih nekoliko koraka:

- Omogućiti SSL za projekt čak i u dev okruženju (localhost)
  - Ovo je postavljeno u novoj verziji Visual Studio alata
- Kreirati vlastitu aplikaciju na Google ili Facebook stranici
  - Dobaviti ClientId i ClientSecret

- Postaviti ispravne autentikacijske podatke u Startup.cs datoteci

Postavljanje SSL

Pošto komunikacija sa servisima kao što su Google i Facebook zahtjeva HTTPS protokol, potrebno je i u našem projektu postaviti da se isključivo koristi HTTPS protokol. U pravilu, na svim stranicama gdje je moguće obaviti neku vrstu prijave ili slanja osjetljivih podataka, obvezno je koristiti HTTPS protokol.

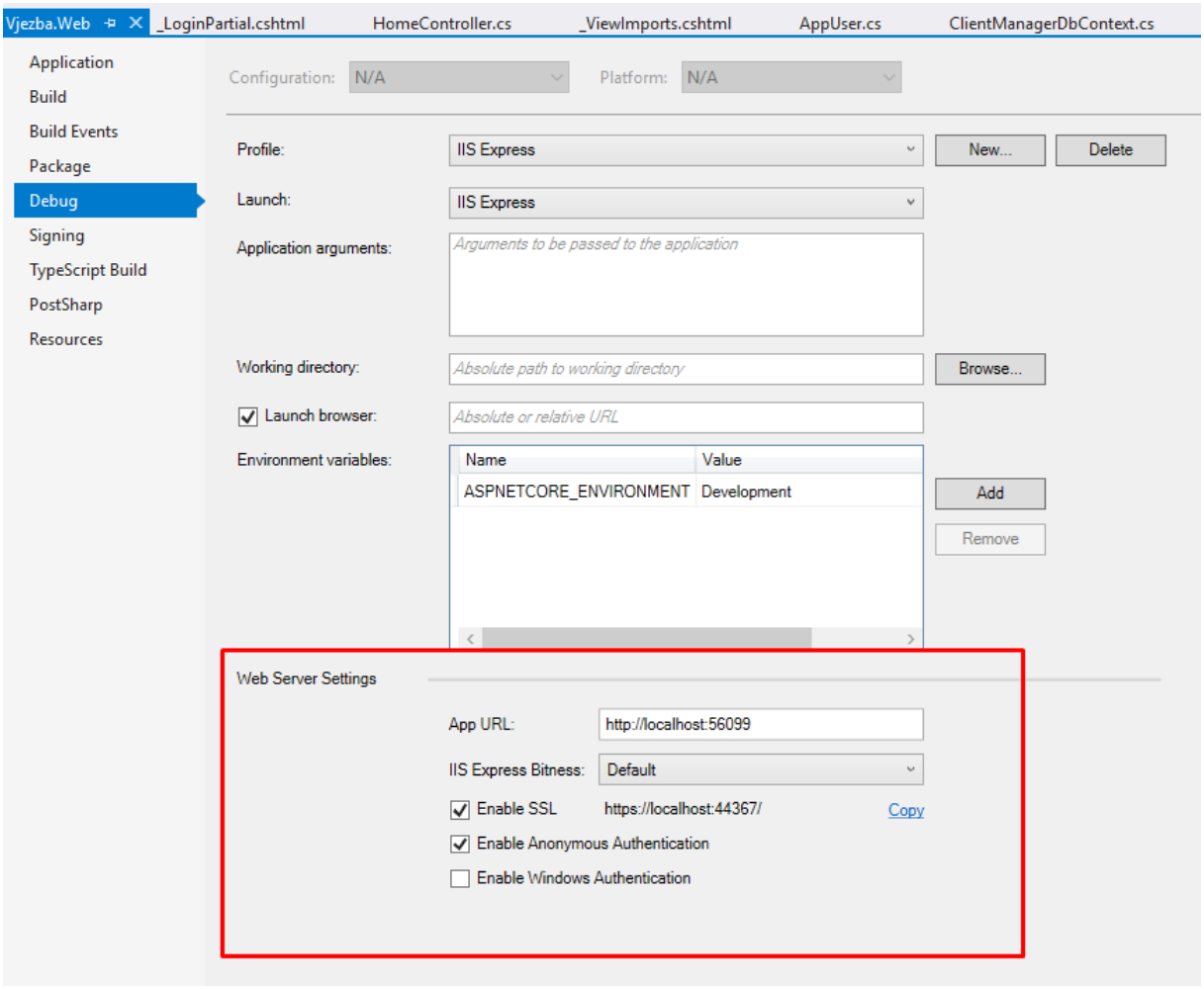
Zadatak 11.2

Postaviti HTTPS protokol u vlastitoj aplikaciji. **Napomena**: obvezno ove postavke pažljivo podesiti i prvo pročitati redom što je potrebno napraviti!

1. Odabrati projekt **Vjezba.Web** i desnim klikom otvoriti postavke
    a. Odabrati opcije pod **Debug**
2. Uvjeriti se da je SSL omogućen
    a. Kopirati URL kojem port najvjerojatnije počinje s 443
3. Uvjeriti se da je sve postavljeno dobro na način da se pokrene aplikacija i potvrdi da je nemoguće pristupiti stranicama preko HTTP protokola, nego isključivo preko HTTPS protokola.

Ukoliko se koristi Visual Studio 2019, dobra je šansa da je to već postavljeno.

Primjer kako mogu izgledati postavke (port se može razlikovati):



Kreiranje vlastite aplikacije na Google ili FB servisu

Kreirati vlastiti google ili FB account (prema želji) koristeći sljedeća uputstva:

- FB: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/facebook-logins?view=aspnetcore-5.0
- Google: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/google-logins?view=aspnetcore-5.0

Zadatak 11.3

Prema uputstvima s gornjih poveznica omogućiti Google/FB login.

- Uvjeriti se da je moguće obaviti registraciju i nakon toga prijavu putem Google/FB servisa u vlastitoj aplikaciji
- Provjeriti nalaze li se podaci u tablici AspNetUsers
- **Napomena**: dovoljno je odabrati jedan od servisa (ili Google ili FB)

Proširenje osnovne AppUser tablice

Naravno, osnovno (inicijalno) ponašanje nije uvijek dostatno za implementaciju funkcionalnosti aplikacije. Iz tog razloga omogućeno je jednostavno proširenje tablice AppUser kako bi unijeli dodatne informacije o korisnicima. Ono što je potrebno napraviti je u klasi Vjezba.Model.AppUser

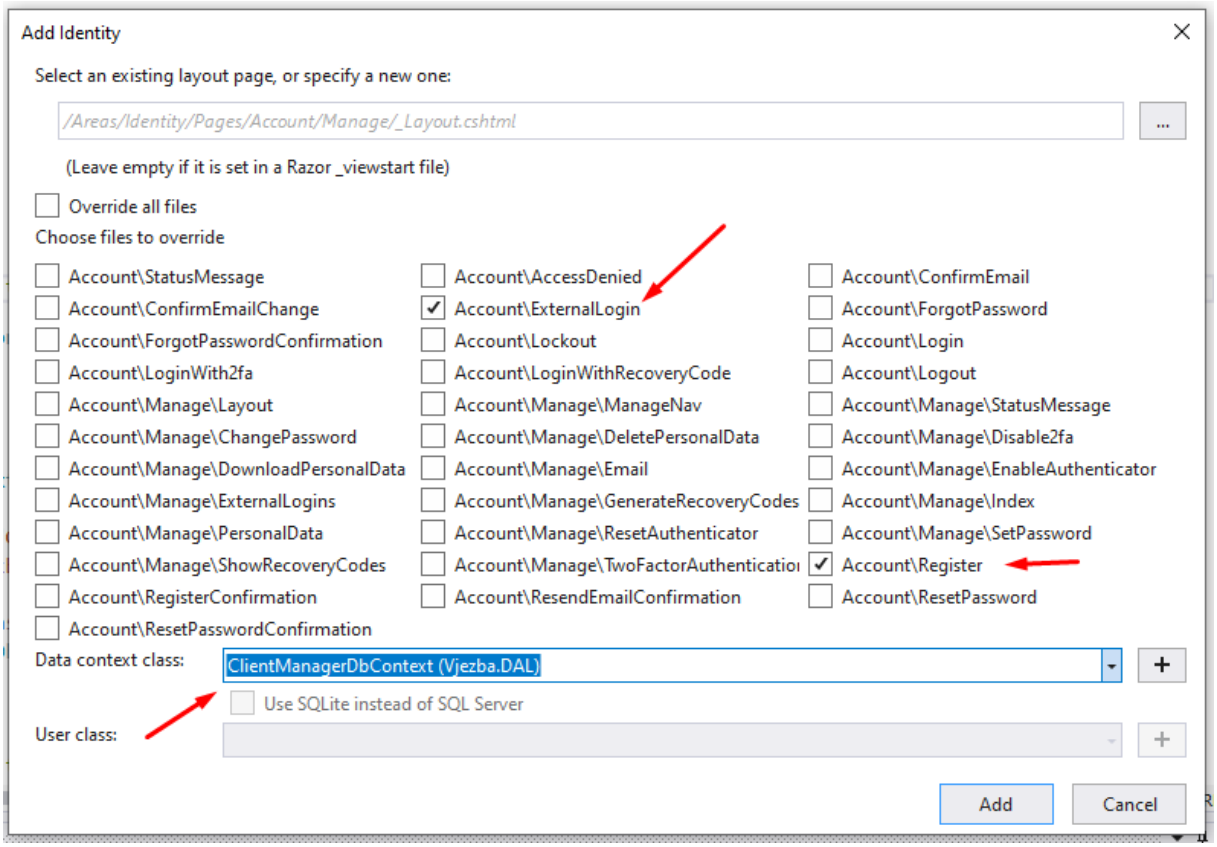
dodati željena svojstva i izvršiti potrebne migracijske skripte, nakon čega će se tražena polja nalaziti u bazi podataka. Međutim, samo s tim izmjenama nismo omogućili korisnicima da unesu te podatke prilikom registracije ili prilikom registracije putem vanjskih servisa. Da bi se to omogućilo, potrebno je dodati željena polja za unos na formama za registraciju.

S obzirom da .NET Core Auth mehanizam automatski generira potrebne forme i stranice za Identity, potrebno je generirati odgovarajuće view datoteke kako bi se mogle modificirati. Datoteke je moguće generirati kroz UI od Visual Studio alata:

- Desni klik na Web project -> Add - > New Scaffolded Item -> „Jahač“ Identity

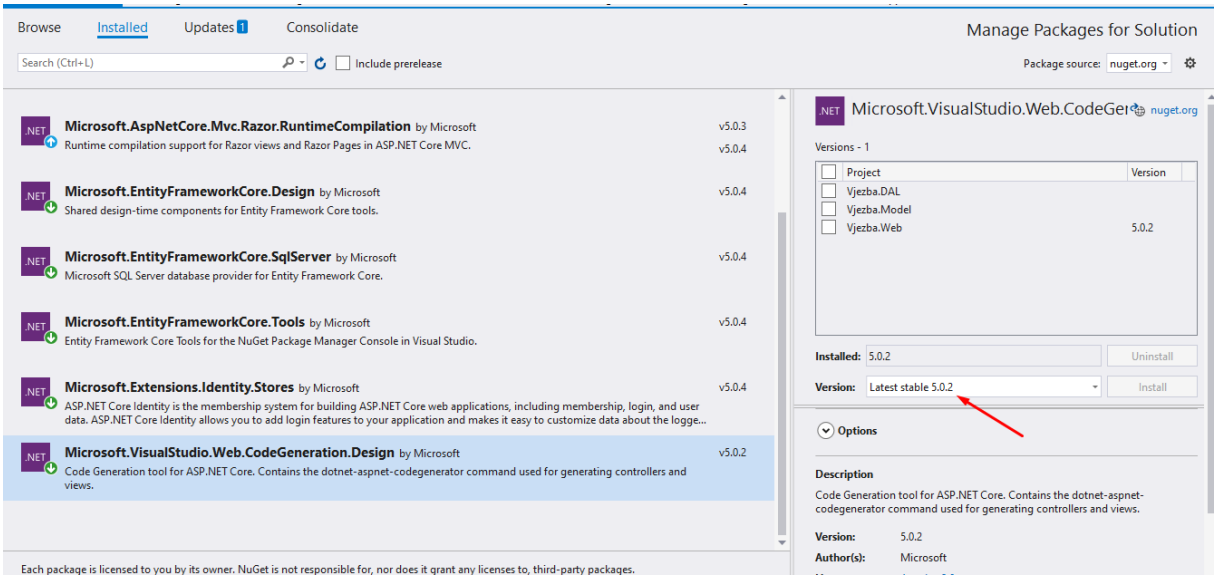
Nakon toga odabrati:

- Account/Register
- Account/ExternalLogin



Po potrebi moguće je prilagoditi i ostale view datoteke, ali za potrebe ovog zadatka dovoljno je odabrati gore navedene.

Napomena: Ukoliko VS2019 izbacuje exception, osigurati da je verzija biblioteke za CodeGeneration 5.0.2 (NuGet package manager GUI se aktivira desnim klikom na projekt, i odabirom *Manage Nuget Packages*):



Zadatak 11.4

Proširiti klasu **AppUser** sa dva specifična polja – **OIB** i **JMBG**, te omogućiti korisnicima unos tih vrijednosti na ispravnim mjestima.

- Dodati migracijske skripte i osvježiti bazu podataka
  - Prilikom generiranja migracije osigurati da se samo ta polja osvježe (možda migracija generira i dodatna osvježavanja polja, njih obrisati prije poziva dotnet ef database update)
- Forma koja se aktivira nakon prve prijave korisnika preko vanjskog servisa ukoliko korisnik ne postoji u sustavu (registracija preko vanjskog sustava) se nalazi u predlošku koji se zove **Account/ExternalLogin.cshtml**
  - Samostalno dodati dva nova polja za **OIB** i **JMBG**
    - Polja trebaju biti obvezna, točne duljine 11 i 13 znakova, samo se sastojati od brojeva (validacija – gdje ju dodati?)
  - Modificirati **ViewModel** koji se koristi na toj formi na ispravan način
  - Modificirati potrebne akcije u ExternalLogin.cshtml.cs klasi
- Forma koja se aktivira ako se klikne na „Register“ link na stranici se nalazi u predlošku koji se zove **Register.cshtml**
  - Napraviti slične promjene kao na prošloj formi

Autorizacija korisnika

Naravno, sama autentikacija ne bi imala smisla ako ne bi mogli ograničiti pristup za željene akcije i controllere samo prijavljenim korisnicima. Možemo ograničiti pristup na razini akcije controllera ili na razini cijelog controllera (primjenjuje se na sve akcije u controlleru):

QuizController.cs (na razini akcije)

```
[Authorize]
public ActionResult Create()
{
    ...
}

[Authorize]
[HttpPost]
public ActionResult Create(Quiz model)
{
    ...
}
```

QuizController.cs (na razini cijelog controllera)

```
    [Authorize]
    public class QuizController : Controller
    {
        public ActionResult Index()
        {
            var context = new QuizManagerDbContext();
       ...
```

S druge strane, ukoliko unatoč eksplicitno navedenom zahtjevu za autorizaciju nad samom Controller klasom želimo dopustiti da se nekoj akciji pristupi bez obzira na to je li korisnik prijavljen ili ne, može se koristiti sljedeća anotacija:

AccountController.cs

```
[AllowAnonymous]
public ActionResult Login(string returnUrl)
{
    ViewBag.ReturnUrl = returnUrl;
    return View();
}
```

Informacije o trenutnom korisniku

Često se može dogoditi da nam je u nekom trenutku u kodu potrebna informacija o tome koji točno korisnik je ulogiran - ne samo da li je trenutni korisnik ulogiran ili ne. To možemo postići sljedećim kodom (primjerice, u controlleru po želji):

Controller

```
[Authorize]
public class ClientController : Controller
{
    private ClientManagerDbContext _dbContext;
    private UserManager<AppUser> _userManager;

    public ClientController(ClientManagerDbContext dbContext, UserManager<AppUs
er> userManager)
    {
        this._dbContext = dbContext;
        this._userManager = userManager;
    }

    public IActionResult Index(string query = null)
    {
        var userId = this._userManager.GetUserId(base.User);
```

Međutim, može se primjetiti da je ovakav poziv prilično nespretan, te je najbolja praksa staviti svojstvo UserId u bazni Controller koje obavlja cijeli ovaj kod.

Zadatak 11.5

Implementirati autorizaciju na primjeru ClientController-a na način da:

- Bilo tko može pregledavati podatke o klijentima (bilo da je prijavljen ili ne) – akcije **Index** i **Details,** pretraga
- Samo prijavljeni korisnici mogu uređivati podatke (**Create**, **Edit**, **Delete**)
- Dodati bazni controller kojeg će naslijediti svi drugi
    - U baznom controlleru dodati string UserId getter property
- Proširiti **Client** sa podacima CreatedById, UpdatedById i modificirati ispravno ClientController da u slučaju kreiranja ili izmjene objekta se zapiše tko ga je modificirao/kreirao
    - Zapisati ili username ili userId u tablicu (po izboru)
        - Nije nužno da je to ForeignKey – čak bolje i ako nije

Autorizacija po ulogama

Za razliku od autentikacije, autorizacija omogućava samo nekim prijavljenim korisnicima da imaju pristup nekim akcijama i nekim controllerima. Za to se brine upravljanje ulogama. Kako bi omogućili autorizaciju po ulogama (role), potrebno je napraviti sljedeće:

- U bazi napuniti podatke o ulogama i o tome koji korisnik ima koju ulogu
    - Uloge se, primjerice, mogu napuniti u **Seed** metodi
    - Koji korisnik ima koju ulogu je moguće napraviti ručno u bazi (za ovu vježbu)
- Anotirati željene akcije/controllere s definicijom koji korisnici smiju pristupiti

Autorizacijske anotacije

Nakon gornje implementacije, potrebno je na željene akcije controllera (ili cijelokupne controllere) postaviti ovakve anotacije:

QuizController.cs

```
[Authorize(Roles = "Admin,StandardUser")]
public ActionResult Details(int id)
{
    using (var context = new QuizManagerDbContext())
    {
        var quizRepo = new QuizRepository(context);
        return View(quizRepo.Get(id));
    }
}

[Authorize(Roles = "Admin")]
public ActionResult Create()
{
    FillDropDownValues();
    return View();
}
```

Zadatak 11.6

Dodati uloge Admin i Manager, te ograničiti pristup akcijama Controller-a:

- Brisati klijenta može samo Admin
- Izmjenjivati i dodavati klijenta može Manager ili Admin
- Bilo koji prijavljeni korisnik neovisno od uloge može pregledavati detalje klijenta
- Svi mogu pregledati tablični prikaz klijenata (i pretraživati)

Dio zadatka je također pomučiti se i otkriti što treba dodati da bi role bile omogućene u aplikaciji. Dodatno, nakon što se korisnika doda u rolu, potrebno je napraviti logout-login kako bi se rola „aplicirala“.

Način demonstracije rješenja: otvoriti 3 različita browsera, i u jednom se ulogirati kao admin, u drugom kao manager, u trećem kao 'obični' korisnik