

Laboratorijska Vježba 9. ASP.NET Core MVC Kontrola pristupa

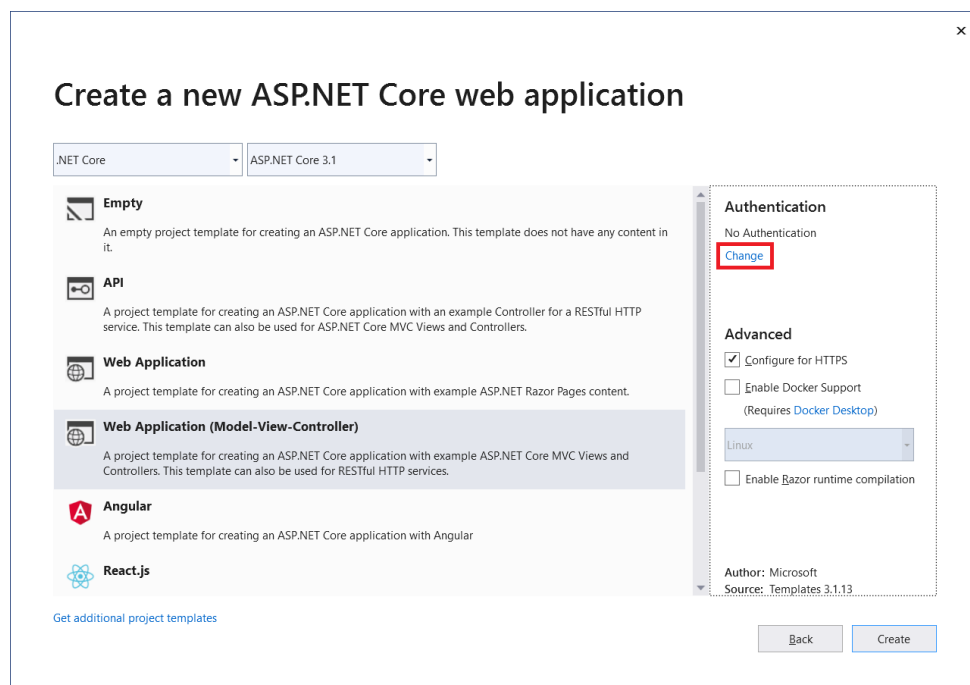
Cilj vježbe:

- Upoznavanje sa načinom automatskog kreiranja kontrole pristupa u ASP.NET aplikacijama;
- Vršenje selektivne kontrole pristupa korisnika korištenjem korisničkih uloga.

1. Specifikacija kontrole pristupa pri kreiranju aplikacije

Nakon dodavanja svih dijelova ASP.NET aplikacije, od sloja podataka do sloja pogleda, potrebno je još omogućiti kontrolu pristupa, jer bez korištenja uloga, svi korisnici imaju pristup svim dijelovima aplikacije bez ograničenja. Dodavanje kontrole pristupa u ASP.NET Core aplikaciji veoma je jednostavno, ali je neophodno omogućiti ga pri kreiranju aplikacije, pa će se nanovo kreirati ista aplikacija koja se koristila u prethodnim laboratorijskim vježbama nad kojom će onda biti izvršene neophodne izmjene kako bi se omogućila kontrola pristupa.

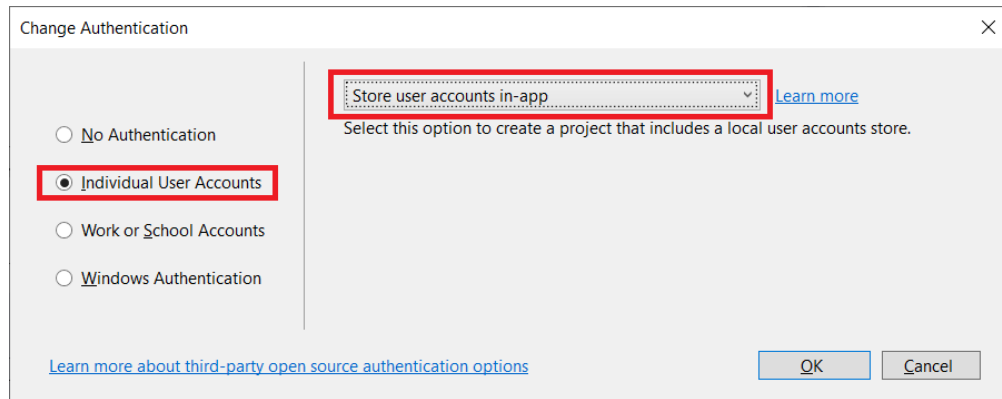
Potrebno je izvršiti iste korake za kreiranje nove ASP.NET aplikacije kao što je demonstrirano u laboratorijskoj vježbi 5, do koraka detaljne specifikacije informacija o aplikaciji. U tom koraku potrebno je, osim vrste ASP.NET Core aplikacije, definisati i da aplikacija posjeduje kontrolu pristupa, odabirom opcije **Authentication** → **Change** na način prikazan na Slici 1.



Slika 1. Odabir opcije za omogućavanje kontrole pristupa

Nakon toga na novom ekranu potrebno je odabrati vrstu kontrole pristupa. Kako bi se omogućila registracija i logovanje bez posredstva *Microsoft*-a i drugih organizacija, potrebno je odabrati opciju **Individual User Accounts**. Budući da korisnički računi neće biti spremeni na posebne sigurne lokacije na *Microsoft Azure* platformi, potrebno je odabrati opciju **Store**

user accounts in-app, na način prikazan na Slici 2. Na ovaj način specifikacija opcija za kontrolu pristupa je završena i aplikacija se može kreirati na isti način kao u prethodnim laboratorijskim vježbama.



Slika 2. Specifikacija informacija o kontroli pristupa

Nakon kreiranja aplikacije, potrebno je ponoviti postupak kreiranja modela, kontrolera i pogleda na isti način kao u prethodnim laboratorijskim vježbama. Kako se programski kod nove aplikacije razlikuje od starog, potrebno je izbjegavati kopiranje sadržaja svih file-ova i dodati samo ono što je neophodno. Ključne promjene u procesu ponovnog kreiranja aplikacije su:

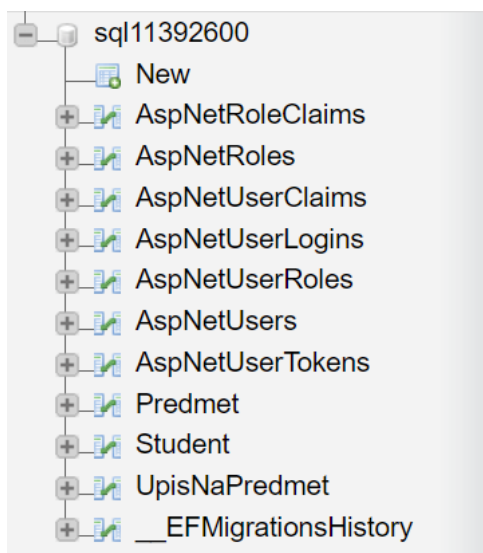
1. Aplikacija već ima predefinisani kontekst podataka. Ovaj kontekst može se iskoristiti pri *scaffold*-anju kontrolera bez potrebe za kreiranjem novog konteksta podataka.
2. Aplikacija ima definisane *Login* stranice u *Shared* folderu kojima se pristupa putem kontrola koje su definisane u *Layout* formi. Iz tog razloga sadržaj ove forme ne smije se brisati, već je samo potrebno dodati link na kontroler za predmete kao u prethodnim vježbama. Isto se odnosi na *Startup* klasu koja posjeduje definiciju korištenja autentifikacije u aplikaciji.
3. Pri definisanju konteksta podataka, u metodu *OnModelCreating* potrebno je dodati još jednu liniju koda kojom se definiše kreiranje tabela za upravljanje korisničkim ulogama. Metoda treba imati strukturu prikazanu u isječku koda ispod.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Student>().ToTable("Student");
    modelBuilder.Entity<UpisNaPredmet>().ToTable("UpisNaPredmet");
    modelBuilder.Entity<Predmet>().ToTable("Predmet");

    base.OnModelCreating(modelBuilder);
}
```

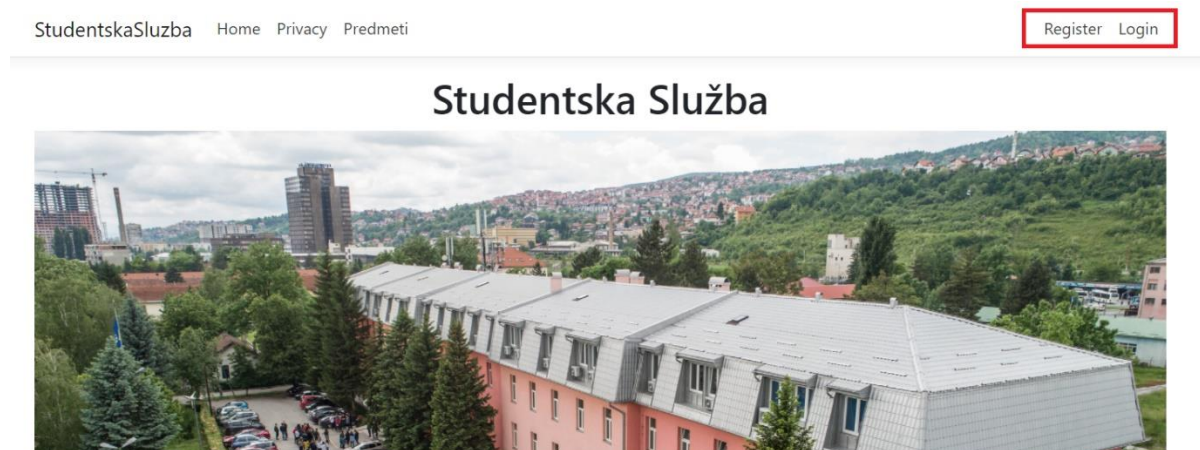
4. Baza podataka koja se koristi već ima definisane tabele za model klase podataka, ali ne i tabele za korisnike. Ako tabele za model klase već postoje, one se trebaju obrisati koristeći interfejs za pristup bazi podataka, a zatim je neophodno izvršiti novu migraciju na bazu podataka, čime se kreiraju nedostajuće tabele podataka.

Nakon što se izvrši migracija, u aplikaciju se dodaje veliki broj tabela važnih za ispravnu kontrolu pristupa, što je vidljivo iz prikaza strukture baze podataka na Slici 3. Ovime je proces rekonfiguracije aplikacije sa različitim postavkama u odnosu na prethodne laboratorijske vježbe završen.



Slika 3. Prikaz strukture baze podataka sa tabelama za autentifikaciju

Nakon što se izvrše svi koraci, aplikacija je spremna za korištenje i može se pokrenuti. Sada je izgled početne stranice nešto drukčiji, jer postoji opcija za registrovanje novih korisnika i logovanje u aplikaciju, kao što je vidljivo na Slici 4.



Aplikacija za registrovanje studenata i predmeta

Slika 4. Aplikacija sa omogućenim opcijama za kontrolu pristupa

Forme za registraciju i logovanje imaju svoje početne definicije i potpuno su funkcionalne¹. Opcije za potvrdu registracije putem emaila i resetovanje postojeće korisničke šifre samo su

¹ Detaljniji opis svih funkcionalnosti koje su automatski dodane kako bi se omogućila registracija i logovanje dostupan je na sljedećem linku: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>.

simulirane, ali je nivo funkcionalnosti dovoljan da bi se ovaj sistem mogao koristiti². Nakon registracije i logovanja, prikazuje se poruka da je korisnik ulogovan i daje mu se opcija za izlogovanje, na način prikazan na Slici 5.

StudentskaSluzba Home Privacy Predmeti

Hello ekrupalija1@etf.unsa.ba! Logout

Studentska Služba



Aplikacija za registrovanje studenata i predmeta

Slika 5. Prikaz interfejsa za ulogovanog korisnika

Nakon registracije korisnika, moguće ih je pronaći u bazi podataka, u tabeli **AspNetUsers**. Kao što je vidljivo na Slici 6, korisničke šifre za registrovane korisnike spašavaju se u polje *PasswordHash*, odnosno ne čuvaju se u *plain-text* formatu bez prethodnog šifriranja. Na ovaj način ostvaruje se dodatni nivo sigurnosti, jer se pri svakom pokušaju logovanja unesena šifra *hash*-ira a zatim se provjerava da li odgovara korisničkoj šifri koja je spašena u bazu podataka i na osnovu toga se odobrava/zabranjuje logovanje korisnika.

| + Options | | | | | | | |
|---|--|--------------------------------------|------------------|--------------------|-------------|-----------------|------------------|
| | | Id | UserName | NormalizedUserName | Email | NormalizedEmail | PasswordHash |
| <input type="checkbox"/> Edit Copy Delete | | 72cfad7e-1d53-4689-b0de-93e0b67042fb | ekrupalija1@etf. | EKRUPALIJA1@ETF.U | ekrupalija1 | EKRUPALIJA1@ | 1 AQAAAAEAACcQAA |
| <input type="checkbox"/> Edit Copy Delete | | e9b49933-967c-4d98-aeb2-ff287559307d | admin@etf.unsa | ADMIN@ETF.UNSA.BA | admin@etf | ADMIN@ETF.UN | 1 AQAAAAEAACcQAA |

Slika 6. Prikaz tabele korisnika u bazi podataka

Sada je neophodno dodati kontrolu pristupa na način da korisnici koji nisu ulogovani nemaju pristup kontroleru za predmete. Kontrola pristupa na jednom nivou vrlo je jednostavna – ukoliko je neophodno da korisnik bude ulogovan kako bi pristupio nekom kontroleru, dovoljno je dodati stereotip *Authorize* u zaglavlje tog kontrolera, na način prikazan u isječku koda ispod. Na taj način onemogućava se pristup tim stranicama (putem pogleda ili manuelnim unosom URL-a) ukoliko korisnik nije ulogovan u aplikaciju. Elementi na formi koji omogućavaju pristup tim stranicama i dalje su vidljivi, ali pri pokušaju pristupa korisnik se preusmjerava na ekran za logovanje.

² Za izmjenu validacije polja korisničke šifre može biti koristan sljedeći link: <https://www.c-sharpcorner.com/UploadFile/4b0136/how-to-customize-password-policy-in-Asp-Net-identity/>,

```
[Authorize]
public class PredmetsController : Controller
{
    private readonly ApplicationDbContext _context;

    public PredmetsController(ApplicationDbContext context)
    {
        _context = context;
    }
    ...
}
```

Ukoliko se elementi na formi (npr. stavka menija, dugme, link) žele sakriti korisnicima koji nisu ulogovani, potrebno je koristeći Razor sintaksu provjeriti da li je korisnik ulogovan u aplikaciju, pa ako jeste prikazati mu taj element. I ovo je vrlo jednostavno učiniti i prikazano je u isječku koda ispod, gdje se samo ulogovanim korisnicima dozvoljava prikaz stavke menija koja omogućava pristup kontroleru za predmete.

```
@if (User.Identity.IsAuthenticated)
{
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-
controller="Predmets" asp-action="Index">Predmeti</a>
    </li>
}
```





2. Korištenje korisničkih uloga

Autorizacija korisnika samo na jednom nivou najčešće nije dovoljna da bi kontrola pristupa bila potpuno konfigurisana. Najčešće postoji više različitih grupa korisnika, koji se u općem slučaju dijele na administratore (najčešće sistem ima samo jednog administratora), ostale autorizovane korisnike (koji mogu biti podijeljeni u jednu ili više grupa, npr. studenti, asistenti, profesori) i anonimne korisnike (koji se često nazivaju gostima i imaju ograničenu kontrolu pristupa svim funkcionalnostima). U nastavku će se izvršiti podjela registrovanih korisnika u dvije grupe – *Administrator* i *Korisnik* i na osnovu toga će se ograničiti pristup određenim funkcionalnostima.

Korisnička uloga nije nešto što korisnik može izabrati kada vrši registraciju na sistem. Ulogu mu dodjeljuje administrator sistema, odnosno korisnik koji ima pristup bazi podataka i može njegovom korisničkom računu dodijeliti ulogu. Iz tog razloga prvo je potrebno definisati uloge koje će postojati u sistemu, a zatim korisnicima dodijeliti različite uloge. U tu svrhu potrebno je imati barem dva registrovana korisnička računa, kako bi se mogla izvršiti provjera prava pristupa. Za definiciju korisničkih uloga potrebno je u bazi podataka odabrati tabelu **AspNetRoles**, a zatim dodati nove uloge³ tako da se dobije prikaz stanja u bazi podataka kao na Slici 7, odakle je vidljivo da postoje dvije definisane uloge – *Administrator* i *Korisnik*.

³ Ukoliko u tabeli već postoji jedna uloga, prvo je iz tabele *AspNetUserRoles* potrebno obrisati sve dodjele uloga korisnicima, a zatim obrisati ulogu iz tabele *AspNetRoles*.

+ Options

| | | | | Id | Name | NormalizedName | ConcurrencyStamp |
|--------------------------|--|--|--|----|---------------|----------------|------------------|
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 1 | Administrator | Administrator | NULL |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 2 | Korisnik | Korisnik | NULL |

Slika 7. Prikaz korisničkih uloga u bazi podataka

Nakon toga potrebno je dodijeliti prethodno definisane uloge registrovanim korisnicima. Potrebno je u tabelu **AspNetUserRoles** dodati dva unosa kojima se definišu ID korisnika i ID uloge koja mu se dodjeljuje. ID-eve korisnika moguće je pronaći u tabeli **AspNetUsers**. Nakon dodjele korisničkih uloga, dobiva se prikaz tabele *AspNetUserRoles* kao na Slici 8. Na ovaj način korisniku admin@etf.unsa.ba dodana je korisnička uloga *Administrator*, a korisniku ekrupalija1@etf.unsa.ba dodana je korisnička uloga *Korisnik*. Ovakvim postupkom može se definisati proizvoljan broj uloga, kao i njihova dodjela korisnicima.

| + Options | | | | | | | | |
|--------------------------|--|------|--|------|--------|--------|--------------------------------------|---|
| ↔ T ↔ | | | | | UserId | RoleId | | |
| <input type="checkbox"/> | | Edit | | Copy | | Delete | e9b49933-967c-4d98-aeb2-ff287559307d | 1 |
| <input type="checkbox"/> | | Edit | | Copy | | Delete | 72cfad7e-1d53-4689-b0de-93e0b67042fb | 2 |

Slika 8. Prikaz tabele dodjele korisničkih uloga u bazi podataka

Da bi se omogućilo korištenje uloga u ASP.NET Core aplikaciji, potrebno je u **Startup.cs** file gdje je prethodno definisano korištenje autorizacije specificirati i korištenje korisničkih uloga u metodi *ConfigureServices*, na način da se pri dodavanju identiteta doda i poziv metode **AddRoles<IdentityRole>**, tako da ova metoda ima izgled kao u isječku koda ispod.

```
// This method gets called by the runtime. Use this method to add services
// to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseMySQL(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
        .AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
    services.AddRazorPages();
}
```

Sada kada registrovani korisnici imaju svoje uloge, potrebno je ograničiti pristup kontrolerima na osnovu različitih uloga. Neka je potrebno da samo administrator ima prikazane opcije za dodavanje, editovanje i brisanje predmeta, a da obični korisnik može samo pregledati informacije o predmetu i njegove detalje. Prvenstveno je neophodno na nivou pogleda onemogućiti prikaz opcija za editovanje korisniku koji nije administrator, što se radi na način prikazan u isječku koda ispod.

```
<td>
    @if (User.IsInRole("Administrator"))
    {
        <a asp-action="Edit" asp-route-id="@item.ID">Edit</a>
        <nobr>|</nobr>
        <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
        <nobr>|</nobr>
    }
    <a asp-action="Details" asp-route-id="@item.ID">Details</a>
</td>
```

Na ovaj način ograničava se pristup na nivou pogleda, ali neophodno je onemogućiti korisniku da manuelno unese željenu rutu u web-pretraživač i na taj način pristupi formama kojima ne bi smio pristupati. Iz tog razloga u programski kod kontrolera potrebno je dodati stereotip kojim se označava da su sve metode osim metode *Index* i *Details* dozvoljene samo administratorima (a metode *Index* i *Details* dozvoljene i običnim, ali ne i neregistrovanim korisnicima). Različite deklaracije stereotipa u slučaju dozvola za jednu i više korisničkih uloga prikazane su u isječku koda ispod.

```
// GET: Predmets
[Authorize(Roles = "Administrator, Korisnik")]
public async Task<IActionResult> Index()
{
    return View(await _context.Predmet.ToListAsync());
}

// GET: Predmets/Create
[Authorize(Roles = "Administrator")]
public IActionResult Create()
{
    return View();
}
```

Ukoliko obični korisnik sada pokuša pristupiti formi za pregled predmeta, dobiva prikaz kao na Slici 9. Dozvoljen mu je samo pregled liste predmeta kao i detalja o svakom pojedinačnom predmetu, ali ne i editovanje. Ukoliko korisnik sada manuelno unese rutu **Predmets/Create** u web-pretraživač, dobiva prikaz kao na Slici 10, odnosno aplikacija prepoznaje da njegova korisnička uloga nije *Administrator* i kao takvom mu se ne dozvoljava pristup ovoj stranici.

Index

Naziv predmeta:

Broj ECTS bodova:

OOAD

5

[Details](#)

Slika 9. Prikaz forma s ograničenjima pristupa

localhost:44316/Identity/Account/AccessDenied

StudentskaSluzba Home Privacy Predmeti

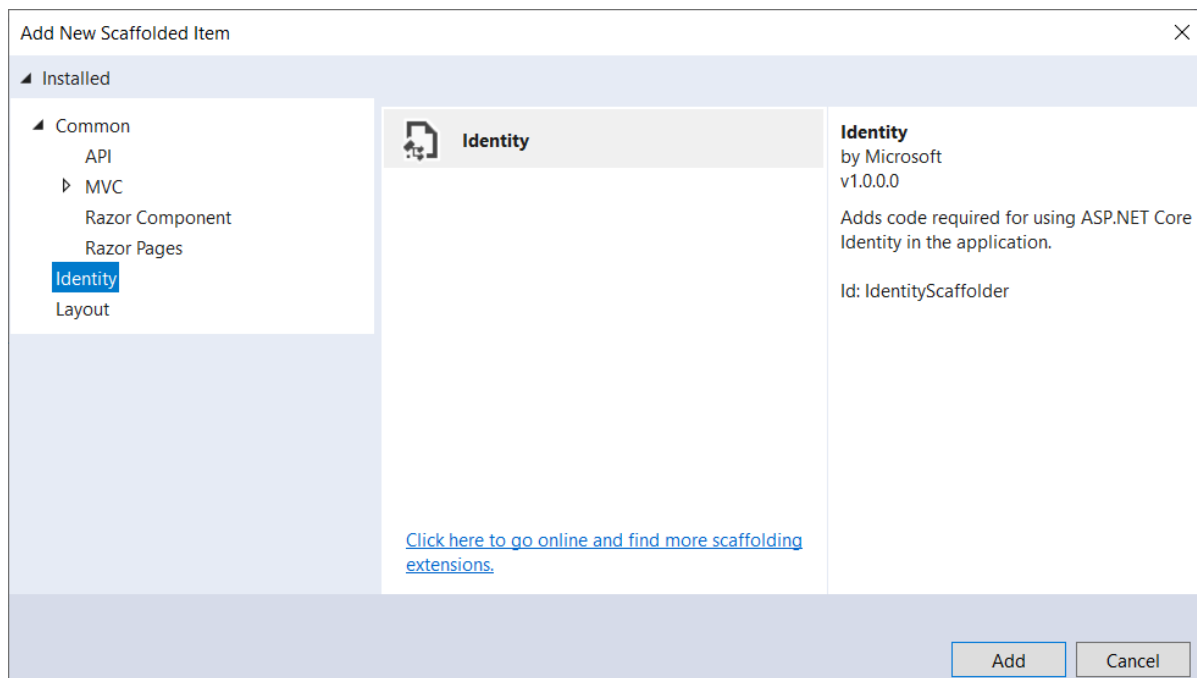
Hello ekrupalija1@etf.unsa.ba! Logout

Access denied

You do not have access to this resource.

Slika 10. Zabrana pristupa na osnovu korisničke uloge

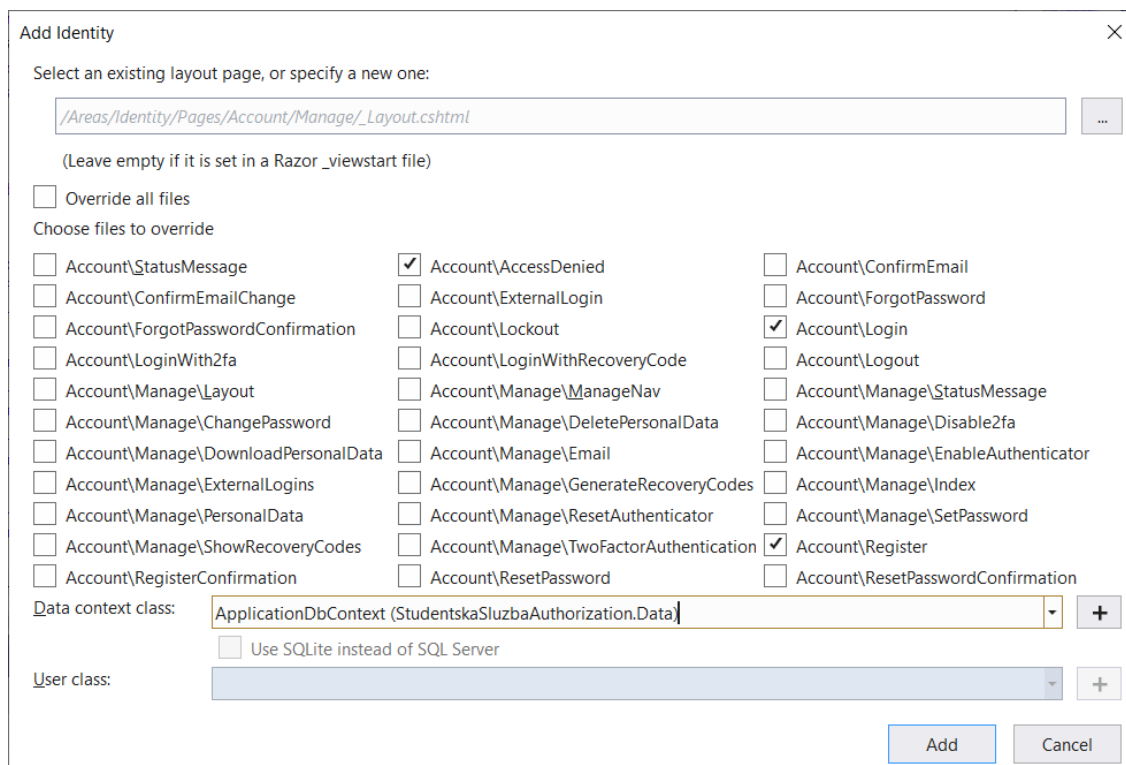
Stranice za registraciju, logovanje na aplikaciju i prikaz poruke o zabrani pristupa nisu dostupne u folderima gdje se nalaze pogledi. Da bi se dobio pristup ovim formama i stekla mogućnost editovanja njihovog sadržaja, potrebno je izvršiti desni klik na projekat, a zatim odabrati opciju **Add → New Scaffolded Item...** kao i pri dodavanju novog kontrolera, a zatim odabrati **Identity** objekat, na način prikazan na Slici 11. Na ovaj način vrši se automatsko generisanje formi za kontrolu pristupa koje se koriste u aplikaciji.



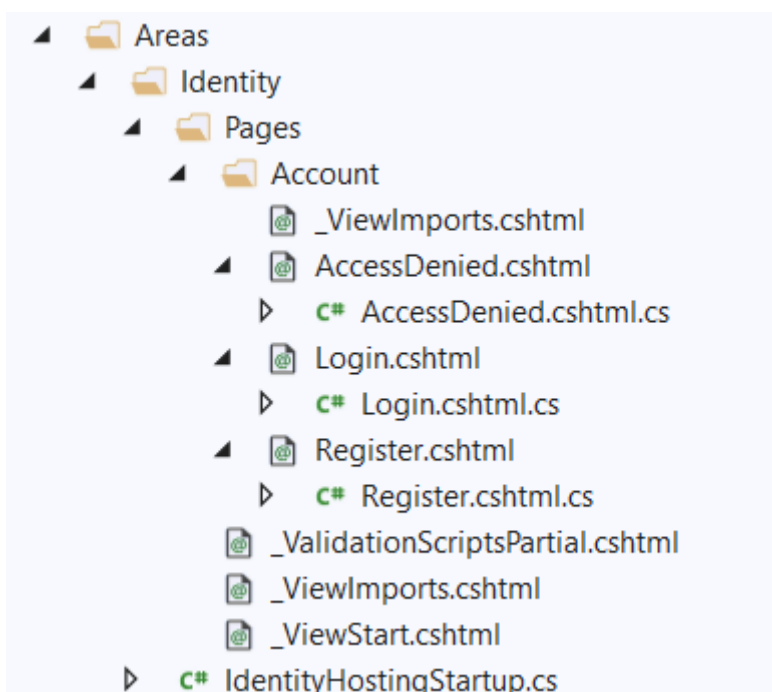
Slika 11. Odabir Identity objekta za automatsko generisanje

Budući da je *Identity* modul koji sadrži veliki broj formi i kontrolera za kontrolu pristupa, korisniku se nudi mogućnost odabira onih dijelova modula koje želi koristiti. Potrebno je odabrati **Account\Access Denied**, **Account>Login** i **Account\Register** funkcionalnosti i odabrati postojeći kontekst podataka, na način prikazan na Slici 12. Nakon toga započinje proces automatskog generisanja koda. Novododane forme nalaze se u folderu **Areas\Identity\Pages\Account**. Kao što je prikazano na Slici 13, u ovom folderu nalaze se pogledi *AccessDenied*, *Login* i *Register* koje je sada moguće editovati po želji.

Objektno Orijentisana Analiza i Dizajn



Slika 12. Odabir Identity modula koji se žele mijenjati



Slika 13. Prikaz novih formi za Identity module

Kao što je prikazano na Slici 13, *AccessDenied*, *Login* i *Register* ne posjeduju samo CSHTML kod kojim se definiše izgled, već i CS kod kojim se definišu same funkcionalnosti registracije i logovanja na aplikaciju. Izmjenom tog programskog koda moguće je proizvoljno definisati način potvrde emaila pri registraciji, resetovanje šifre putem emaila i sl., a ove funkcionalnosti se mogu i izbaciti i onemogućiti. Analiza ovih *file*-ova ostavlja se studentima za samostalno istraživanje.

3. Zadaci za samostalni rad

Programski kod aplikacije koja je kreirana u vježbi dostupan je u repozitoriju na sljedećem linku: <https://github.com/ehlymana/OOADVjezbe>, na *branchu* **lv9**.

1. Napraviti novu ASP.NET Core aplikaciju sa omogućenom kontrolom pristupa. Izvršiti neophodne izmjene kako bi se aplikacija iz prethodnih laboratorijskih vježbi ponašala na isti način uz kontrolu pristupa.
2. Ograničiti kontrolu pristupa tako da neregistrovani korisnici nemaju pristup akcijama *Create*, *Edit* i *Delete* kontrolera *Predmeti*. Onemogućiti pristup na nivou formi i na nivou kontrolera.
3. Napraviti tri korisničke uloge – *admin*, *studentska* i *student*. Studentima je potrebno onemogućiti pregled broja ECTS bodova za predmete (i na *Index* i na *Details* formama). Studentska služba može dodavati i editovati predmete, ali ih ne može brisati. Administratoru su dozvoljene sve mogućnosti. Definirati kontrolu pristupa na nivou formi i na nivou kontrolera.