Podešvanje autentikacije/autorizacije u ASP.NET Web API framework-u

Inicijalna aplikacija

Inicijalna aplikacija je generisana pomoću Visual Studio template-a tako što se izabere .NET Web Application (.NET Framework) + ASP.NET Web API + Individual User Authentication.

Startup

Sve počinje iz Startup.cs klase koja se poziva prilikom pokretanja Owin middleware-a.

```
[assembly: OwinStartup(typeof(WebApp.Startup))]

namespace WebApp
{
    3 references | lukic-aleksandar, 5 days ago | 1 author, 1 change
    public partial class Startup
    {
        0 references | lukic-aleksandar, 5 days ago | 1 author, 1 change
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);
        }
    }
}
```

Okruženje će Configuration metodi proslediti objekat koji ima interfejs IAppBuilder koji nam omogućava dodavanje novih slojeva u middleware. Startup klasa je parcijalna i u svom drugom parcijalnom delu, koji se nalazi u App_Start/Startup.Auth.cs, sadrži metodu ConfigureAuth(IAppBuilder) u kojoj se podešava autentikacija/autorizacija.

```
public void ConfigureAuth(IAppBuilder app)
{
    // Configure the db context and user manager to use a single instance per request
    app.CreatePerOwinContext(ApplicationDbContext.Create);
    app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);

    // Enable the application to use a cookie to store information for the signed in user
    // and to use a cookie to temporarily store information about a user logging in with a third party login provider
    app.UseCookieAuthentication(new CookieAuthenticationOptions());
    app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

    // Configure the application for OAuth based flow
    PublicClientId = "self";
    OAuthOptions = new OAuthAuthorizationServerOptions
    {
        TokenEndpointPath = new PathString("/Token"),
        Provider = new ApplicationOAuthProvider(PublicClientId),
        AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),
        AccessTokenExpireTimeSpan = TimeSpan.FromDays(14),
        // In production mode set AllowInsecureHttp = false
        AllowInsecureHttp = true
    };

    // Enable the application to use bearer tokens to authenticate users
    app.UseOAuthBearerTokens(OAuthOptions);
}
```

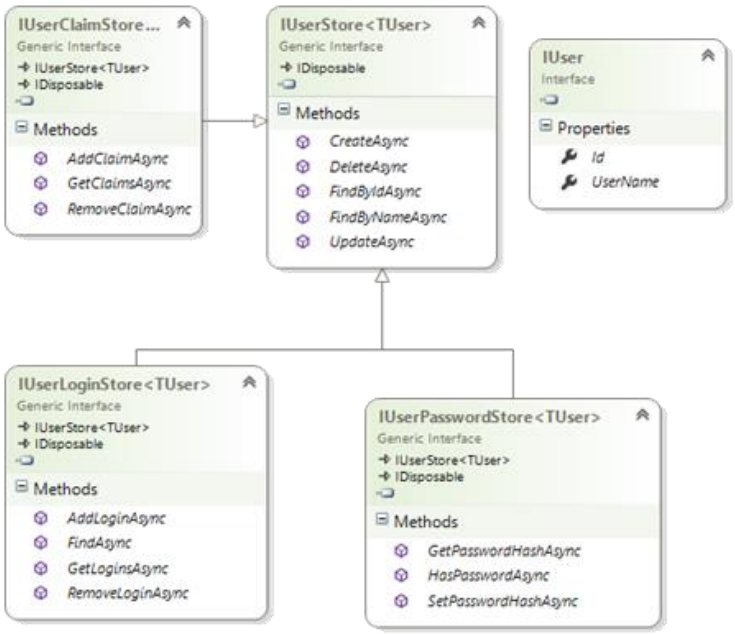
ConfigAuth metoda će ovde biti ukratko pojašnjena, dok su detalji navedeni u kasnijim sekcijama. U ovoj metodi je urađeno sledeće:

1. Registrovan je kontekst za bazu i navadeno je da se on kreira po korisničkom zahtevu.
2. Registrovan je menadžer korisnika i navedeno je da se on kreira po korisničkom zahtevu.
3. Omogućeno je korišćenje Cookie-a za čuvanje podataka o ulogovanom korisniku
4. Kreirane su opcije za OAuth autorizaciju i podešeno je korišćenje OAuth bearer tokena.

Rad sa korisnicima

ASP.NET Core Identity

Core Identity definiše skup interfejsa koji opisuju korisnika (IUser) i skladišta korisnika (IUserStore*).



Sa slike se može videti da svaki korisnik mora imati barem Id i UserName. Klasa koja u aplikaciji predstavlja korisnika treba da implementiraju ovaj interfejs i eventualno kreira dodatne atribute. Korisnici se čuvaju u UserStore-u, koji omogućava CRUD operacije nad njima. Na najvišem nivou hijerarhije se nalazi IUserStore<TUser> gde je TUser tip koji implementira IUser. Komponente koje implementiraju IUserPasswordStore interfejs koriste se za podršku lokalnih korisničkih naloga, dok komponente koje implementiraju IUserLoginStore se korsite za third party login kao što je Google. Pored ovih skladišta korisnika postoje i skladišta za Claim-ove i Role IUserClaimStore i IUserRoleStore.

Rad sa bazom

ASP.NET Identity nudi rad sa bazom pomoću Entity Framework-a. Polazna klasa koja služi za rad sa bazom je DbContext. ASP.NET Identity nudi IdentityDbContext klasu koja implementira sve potrebne

UserStore-ove. Dalje ćemo mi u taj kontekst dodavati sopstvene klase modela kako bi se sve nalazilo u jednoj bazi. UserStore koristi IUser interfejs, a ASP.NET Identity nudi default implementaciju ovog interfejsa kroz IdentityUser klasu, koju mi dalje nasleđujemo u ApplicationUser klasi gde se mogu dodavati dodatni atributi korisnika.

```
public class ApplicationUser : IdentityUser
{
    4 references | lukic-aleksandar, 5 days ago | 1 author, 1 change
    public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager, string authenticationType)
    {
        // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
        var userIdentity = await manager.CreateIdentityAsync(this, authenticationType);
        // Add custom user claims here
        return userIdentity;
    }
}
```

```
5 references | lukic-aleksandar, 5 days ago | 1 author, 1 change
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    1 reference | lukic-aleksandar, 5 days ago | 1 author, 1 change
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }

    1 reference | lukic-aleksandar, 5 days ago | 1 author, 1 change
    public static ApplicationDbContext Create()
    {
        return new ApplicationDbContext();
    }
}
```

UserManager

UserManager je konkretna klasa koja omogućava rad sa korisničkim informacijama. Ona je zadužena za hešovanje šifre, validaciju korisnika i rukovanje claim-ovima i rolama. U UserManagera se mogu dodati custom validatori korisnika, šifri i specifične heš funkcije. Pomoću ove klase se vrše CRUD operacije nad korisnicima, ne radi se direktno sa DbContext-om(što će moći da se vidi u narednim primerima).

```
public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)
{
    var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));
    // Configure validation logic for usernames
    manager.UserValidator = new UserValidator<ApplicationUser>(manager)
    {
        AllowOnlyAlphanumericUserNames = false,
        RequireUniqueEmail = true
    };
    // Configure validation logic for passwords
    manager.PasswordValidator = new PasswordValidator
    {
        RequiredLength = 6,
        RequireNonLetterOrDigit = true,
        RequireDigit = true,
        RequireLowercase = true,
        RequireUppercase = true,
    };
    var dataProtectionProvider = options.DataProtectionProvider;
    if (dataProtectionProvider != null)
    {
        manager.UserTokenProvider = new DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Create("ASP.NET Identity"));
    }
    return manager;
}
```

Registracija korisnika

Registracija korisnika je implementirana u AccountController klasi.

```
  // POST api/Account/Register
[AllowAnonymous]
[Route("Register")]
0 references | lukic-aleksandar, 5 days ago | 1 author, 1 change
public async Task<IHttpActionResult> Register(RegisterBindingModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = new ApplicationUser() { UserName = model.Email, Email = model.Email };

    IdentityResult result = await UserManager.CreateAsync(user, model.Password);

    if (!result.Succeeded)
    {
        return GetErrorResult(result);
    }

    return Ok();
}
```

Dodavanje OAuth autentikacije

Log in, odnosno traženje tokena se zadaje sa TokenEndpointPath atributom, a dužina trajanja sa AccessTokenExpireTimeSpan.

```
OAuthOptions = new OAuthAuthorizationServerOptions
{
    TokenEndpointPath = new PathString("/Token"),
    Provider = new ApplicationOAuthProvider(PublicClientId),
    AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),
    AccessTokenExpireTimeSpan = TimeSpan.FromDays(14),
    // In production mode set AllowInsecureHttp = false
    AllowInsecureHttp = true
};
```

Izmene radi podrške JWT tokena

U Startup.Auth.cs klasi dodata konfiguracija je podeljena u dve metode. Prva konfiguriše OAuth server, dok druga konfiguriše korišćenje JWT tokena za autorizaciju. Token će se dobijati na /oauth/token putanji, trajaće jedan dan, a njegov sadržaj je definisan u CustomJwtFormat klasi. Kao provider postavljena je klasa CustomOAuthProvider, koja sadrži metodu GrantResourceOwnerCredentials u kojoj se proverava postojanje korisnika, slika ispod.

```
private void ConfigureOAuthTokenGeneration(IAppBuilder app)
{
    app.CreatePerOwinContext(ApplicationDbContext.Create);
    app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);

    OAuthAuthorizationServerOptions OAuthServerOptions = new OAuthAuthorizationServerOptions()
    {
        //For Dev enviroment only (on production should be AllowInsecureHttp = false)
        AllowInsecureHttp = true,
        TokenEndpointPath = new PathString("/oauth/token"),
        AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
        Provider = new CustomOAuthProvider(),
        AccessTokenFormat = new CustomJwtFormat(ISSUER)
    };

    // OAuth 2.0 Bearer Access Token Generation
    app.UseOAuthAuthorizationServer(OAuthServerOptions);
}
1 reference | lukic-aleksandar, 1 day ago | 1 author, 1 change
private void ConfigureOAuthTokenConsumption(IAppBuilder app)
{
    string audienceId = ConfigurationManager.AppSettings["as:AudienceId"];
    byte[] audienceSecret = TextEncodings.Base64Url.Decode(ConfigurationManager.AppSettings["as:AudienceSecret"]);
    // Api controllers with an [Authorize] attribute will be validated with JWT
    app.UseJwtBearerAuthentication(
        new JwtBearerAuthenticationOptions
        {
            AuthenticationMode = AuthenticationMode.Active,
            AllowedAudiences = new[] { audienceId },
            IssuerSecurityTokenProviders = new IIssuerSecurityTokenProvider[]
            {
                new SymmetricKeyIssuerSecurityTokenProvider(ISSUER, audienceSecret)
            }
        });
}
```

Provera postojanja korisnika vrši se pomoću UserManager-a.

```
public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
{
    ApplicationUserManager userManager = context.OwinContext.GetUserManager<ApplicationUserManager>();

    ApplicationUser user = await userManager.FindAsync(context.UserName, context.Password);

    if (user == null)
    {
        context.SetError("invalid_grant", "The user name or password is incorrect.!!!!");
        return;
    }

    //if (!user.EmailConfirmed)
    //{
    //    context.SetError("invalid_grant", "AppUser did not confirm email.");
    //    return;
    //}

    ClaimsIdentity oAuthIdentity = await user.GenerateUserIdentityAsync(userManager, "JWT");

    var ticket = new AuthenticationTicket(oAuthIdentity, null);

    context.Validated(ticket);
}
```

Korisni linkovi:

https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth

https://developer.okta.com/blog/2018/06/29/what-is-the-oauth2-password-grant

https://odetocode.com/blogs/scott/archive/2013/11/25/asp-net-core-identity.aspx

https://odetocode.com/blogs/scott/archive/2014/01/03/asp-net-identity-with-the-entity-framework.aspx

https://benfoster.io/blog/how-to-write-owin-middleware-in-5-different-steps

https://coding.abel.nu/2014/05/whats-this-owin-stuff-about/

https://tools.ietf.org/html/rfc6749