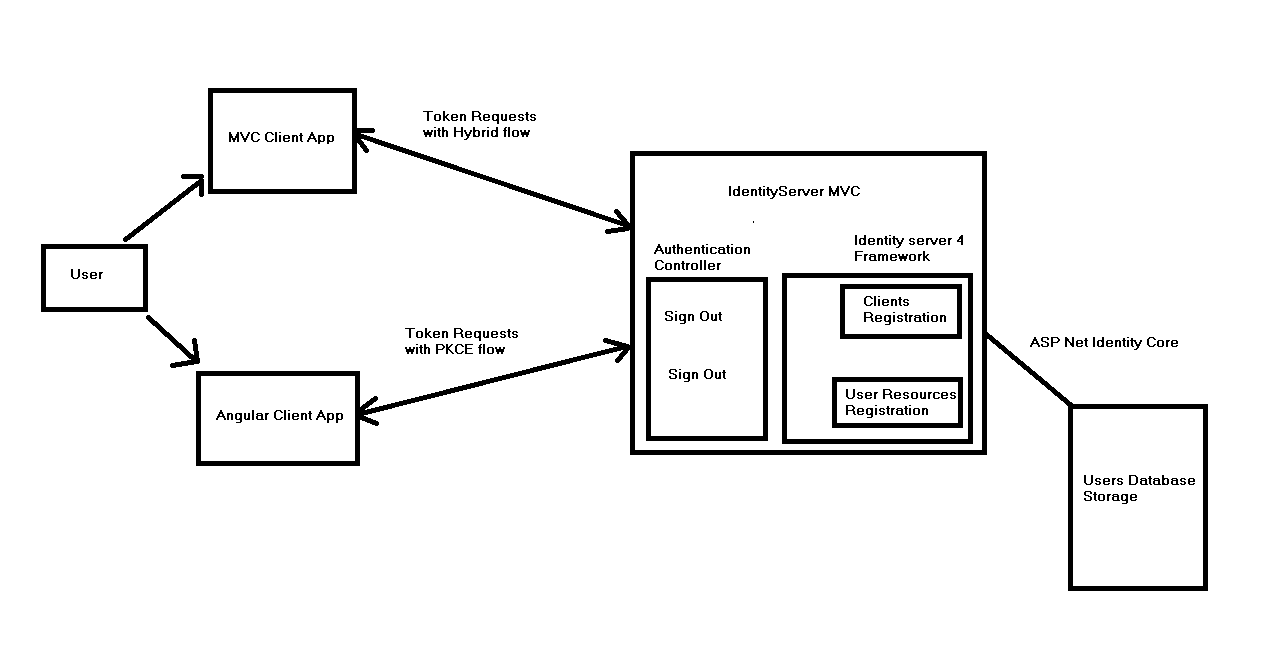
**Arhitektura**

****

Slika 1. *Arhitektura sistema*

Pre nego što krenemo u praktičnu implementaciju Single Sign On-a , ukratko ćemo dati uvid u arhitekturu željene implementacije kao tehnologije koje ćemo koristiti u ovom radu.

Kao što smo napomenuli cilj ovog rada jeste pokazati i objasniti koncept Single Sign On-a , odnosno mogućnost korisnika da autentikovanjem preko jedne aplikacije ima pristup i drugim aplikacijama koje su deo sistema, bez potrebe da se ponovo autentikuje odnosno unosi svoje kredencijale.

Naš sistem se sastoji od dve klijentske veb aplikacije (MVC Client , Angular Client) , kao i Servera identiteta preko koje ce biti omoguceno deljenje informacija identiteta korisnika (IdentityServer).

**Tehnologije**

Sada ćemo ukratko navesti tehnologije pomoću kojih ćemo implementirati naš sistem.

1. **MVC Client**- Klijentska aplikacija implementirana korišćenjem Microsoft .Net Core 5 Framework-a. S obzirom da predstavlja standardnu veb aplikaciju i ima svoju bezbednu stranu (Backend), koristiće Hibridni tok (Registracija klijenata ovog tipa u serveru identiteta se definiše kao privatni klijent sa klijentskom tajnom) u cilju dobijanja tokena identiteta.
2. **Angular Client**- Klijentska aplikacija implementirana korišćenjem Angular 11 Framework-a. S obzirom da je ova klijentska aplikacija Single Page Application i izvršava se isključivo u korisničkom pretraživaču, nema mogućnost za sigurno čuvanje klijentske tajne. Iz tog razloga za primavljanje tokena identiteta koristiće se PKCE tok sa javnim klijentom koga registrujemo u našem serveru identiteta.
3. **Identity Server** – Serverska MVC aplikacija implementirana korišćenjem Microsoft .Net Core 5 Framework-a. Da bi omogućili našem serveru funkcionalnost servera veze između klijentskih aplikacija potrebno je implementirati sledeće module:

* **OIDC i OAuth 2.0**- za implementaciju ovih protokola koristićemo Identity Server 4 framework ( Registracija klijanata i Korisničkih Resursa je deo ovog frameworka)
* **Sign In, Sign On** – za implementaciju koristićemo Microsoft .Net Core 5 Framework. Za ovu svrhu registrovaćemo Authentication kontroler sa SignIn, Sign Out metodama.
* **Autentikacija korisnika**– za autentikaciju koristićemo Asp Net Core Identity biblioteku a za čuvanje korisničkih podataka Microsoft SQL server.

**Identity Server Solution**

**Identity Server 4**

Identity server je besplatan OpenId Connect (OIDC) i OAuth 2.0 framework za ASP.Net Core. Osnovali i odrzavali su ga Dominick Baier i Brock Allen.

Identity server 4 uključuje sve implementacije protokola i moguće proširivosti potrebne za integrisanje autentifikacije zasnovane na tokenima, Single Sign On-a kao i kontrolu pristupa API resursima vaše aplikacije.Identity server 4 je zvanično sertifikovan od strane OpenID Foundation i stoga je u skladu sa specifikacijama.Deo je i .Net Foundation i deluje u skladu sa njihovim kodeksom ponašanja. Licensiran je pod Apache 2 (licensa koju je odobrio OSI).

Identity server 4 je dizajniran da pruži uobičajeni način za potvrdu identiteta, bilo da su to Veb,Native mobilne ili API aplikacije.Identity server 4 se može koristiti za primenu Single Sign On za više aplikacija i tipova aplikacija.

**Standardni scenariji veb aplikacija**

Tipično, aplikacije bi trebalo da podrže neke ili sve od sledećih scenarija (slika 2):

* Korisnik pristupa veb aplikaciji preko pretraživača
* Korisnik pristupa Web API resursima preko aplikacija orijentisanje na pretraživaču
* Korisnik preko mobilnih native aplikacija pristupa Web API resursima
* Neke druge aplikacije pristupaju Web API resursima (bez aktivnog korisnika ili korisničkog interfejsa)



Slika 2. Tipovi aplikacija I scenarija

Za potrebu autentikacije cemo koristiti centralizovanu .Net Core 5 Web aplikaciju pod nazivom IdentityServer(slika 4) , a preko nuget paketa ubacujemo Identity Server 4 framework na sledeci način (slika 3):



Slika 3. Identity Server 4 nuget



Slika 4. IdentityServer .NET 5.0 Solution

U ovom radu ćemo se isključivo fokusirati na autentikaciji korisnika i postizanju single sign on-a korišćenjem kolačića i tokena identiteta (id\_token). Token za prava pristupa(access\_token) ćemo koristiti isključivo za dobavljanje dodatnih korisničkih informacija.

Da bi implementirali Single Sign On korišćenjem Identity Server 4 framework-a potrebno je objasniti sledeće oblasti:

1. Metode implementirane od strane Identity server 4 frameworka:

* Discovery Endpoint
* Authorize Endpoint
* Token Endpoint
* UserInfo Endpoint
* End Session Endpoint

1. Reference:

* Identity Resource
* Client
* Profile Service
* ASP.NET Identity support

**Discovery endpoint**

Discovery dokument se može naći na adresi <https://baseaddress/.well-known/openid-configuration> i sadrži informacije o endpointima,ključnim materijalima i prednostima unutar identity servera (slika 5).

Koristi se takođe u automatizaciji sa klijentskom aplikaciom u pronalazenju ključnih endpointa po oidc specifikaciji (authorize/token/userinfo).



Slika *5*. *Identity Server 4 / .well-known/openid-configuration*

**Identity Resource**

Zadatak našeg servera identiteta jeste kontrola pristupa različitim resursima preko tokena.

Dva osnovna tipa resursa u okviru servera identiteta jesu:

* **Identity Resources** – predstavljaju tvrdnje o korisniku poput id,email , address i tako dalje.
* **API resources** – predstavljaju neku funkcionalnost kojoj klijenti zele da pristupe.U glavnom su to restful API aplikacije.

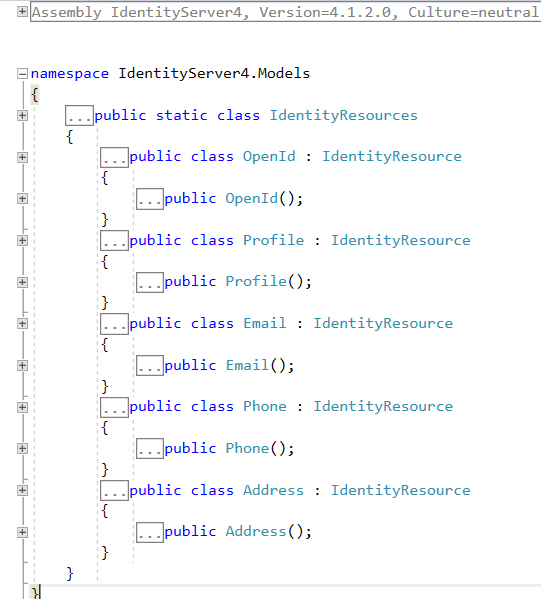
S obzirom da je fokus u ovom radu postizanje Single Sign On-a , uglavnom cemo se fokusirati na resurse identiteta (eng Identity Resources) i kako se informacije o korisniku preko tokena identiteta razmenjuju između različitih klijentskih aplikacija i servera ideniteta u cilju postizanja Single Sign Out-a.

Resursi identiteta u Identity Server-u predstavljaju zapravo imenovanu grupu tvrdnji koja moze biti zahtevana od stranje klijentske aplikacije , i koje se nalaze u tokenu identiteta.

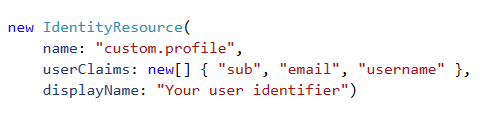
Kada smo objasnjavali autorizacione tokove u sklopu OIDC protokola , rekli smo da se **scope** parametar koristi za definisanje nekog opsega prava pristupa (što se tiče tokena identiteta , definise koje ce se tvrdnje o korisniku vratiti klijentskoj aplikaciji).

Takođe smo rekli da bi se izdao token identiteta, klijentska aplikacija prilikom toka autorizacije mora unutar **scope** parametra poslati **openid** tvrdnju, koja predstavlja minimalnu grupu resursa identiteta da bi se token izdao i sadrzi **sub** tvrdnju koja uglavnom predstavlja jedinstveni identifikator korisnika u bazi.

Server identiteta pruža nekoliko standardnih tvrdnji(slika 6) ali možemo definisati i sami neku imenovanu grupu koja objedinjuje vise tvrdnji samog korisnika (slika 7).



Slika *6*. *Identity Server 4 standardne tvrdnje*



Slika *7*. *Identity Server 4 definisanje sopstvene grupe tvrdnji*

Na kraju treba definisati niz resursa odnosno grupe tvrdnji , sa kojom Server Identiteta raspolaže (slika 8)



Slika *8*. *Identity Server 4 definisanje raspolozivih resursa unutar Servera Identiteta*

**Klijenti**

U sklopu ovog rada za demonstraciju Single Sign On-a ćemo koristiti dve web klijentske aplikacije:

* **ClientMVC-** Web aplikacija implementirana u .Net Core 5 Framework-u
* **AngularClient-** Web SPA aplikacija implementirana u Angular 11 Framework-u

Da bi se postigao efekat Single Sign On-a unutar Servera identiteta moramo registrovati klijentske aplikacije (eng Relayng Party) kao i dodeliti im posebna prava pristupa (slika 9, slika 10)



Slika *9*. *Identity Server 4 definisanje MVC klijenta*

**MVC Klijent**

Sada ćemo objasniti ključne parametre definisanih unutar MVC klijenta (slika 9):

* **ClientId –** Jedinstveni identifikator klijenta
* **AllowedGrantTypes-** Dozvoljenji tokovi autorizacije ( Za MVC klijenta ćemo koristiti Hibridni tok, takođe treba napomenuti da klijent može da podrži vise toka )
* **AllowedCorsOrigins-** U nekim slučajevima klijentske aplikacije tokom procesa autentikacije koriste **AJAX** pozive , tako da je potrebno registrovati validne domene klijentskih aplikacije zbog **CORS** politike.
* **ClientSecrets –** Klijentska tajna (koristi se za registrovanje klijenata od poverenja i namenjena je aplikacijama koje mogu sigurno čuvati tajnu poput MVC Web aplikacije koja ima svoj Backend Deo i ne izvršava se isključivo u pretraživaću poput Single Page Application
* **RedirectUris** – Adrese klijentskih aplikacija gde server identiteta nakon potrebne autentikacije vraća kod ili token identiteta (u slučaju hibridnog toka vraća se kod i id\_token). Treba napomenuti da postoji mogucnost da se više klijentskih aplikacija mogu autentikovati preko istog klijenata registrovanog u serveru identiteta i zato ovaj parametar očekuje listu povratnih adresa.
* **PostLogoutRedirectUris-** Slično kao RedirectUris samo se ovoga puta navodi adresa klijentske aplikacije nako izvršene odjave
* **IdentityTokenLifetime –** Trajanje izdatog tokena identiteta u sekundama
* **RequireClientSecret-** Identity server zahteva da klijentska aplikacija tokom zahteva za izdavanje tokena mora dostaviti klijentsku tajnu.
* **AllowOfflineAccess –** Ukoliko je ovaj parametar postavljen , klijentska aplikacija ce dobiti i token osveženja (eng refresh\_token), sa kojim može ponovo zahtevati tokene (access\_token, id\_token) bez potrebe da se ponovo autentikuje.
* **AllowedScopes -**Predstavlja listu dozvoljenih opsega pristupa resursima. Ovde Identity server definiše pravo klijenta da pristupa i dobija odredjene resurse. U konkretnom slučaju dozvoljeni korisnički resursi , koje klijentska aplikacija može da zahteva su **openId** i **custom.profile** (slika 6 i slika 7). To znači da ce token identiteta izdat klijentskoj aplikaciji sadržati sledece tvrdnje: **sub** (tvrdnja u sklopu openId resursa korisnika), **email**, **username** (tvrdnje u sklopu našeg samostalnog definisanog custom.profile resursa korisnika).

**Angular Klijent**



Slika *10.* *Identity Server 4 definisanje Angular klijenta*

Sada ćemo definisati i jednog klijenta za Angular aplikaciju. Kao što možemo videti dosta je slično predhodnom definisanom MVC klijentu, a sada cemo objasniti ključne razlike (slika 10).

* **AllowedGrantTypes**- Angular klijent koristi tok Autorizacionim kodom, zato ovaj parametar definišemo kao **GrantTypes.Code**.
* **AllowedScopes**- Angular klijent moze u tokenu ideniteta zahtevati samo **sub** i **email** tvrdnje (to smo definisali pomocu **openid** i **email** korisničkih resursa
* **RequireClientSecret**- Angular klijent je definisan kao javni , što znači da tokom validacije pre izdavanja tokena, klijentska tajna neće biti potrebna.Ovim parametrom definišemo da se ne vrši provera klijentske tajne.
* **RequirePkce**- ovim parametrom definišemo da se koristi **PKCE** tok autorizacije za pribavljanje tokena

**Authorize Endpoint**

Authorize endpoint unutar Identity server 4 biblioteke se koristi za validaciju klijentske aplikacije. Prvi korak u procesu autentikacije klijentske aplikacije jeste poziv Authorize metode unutar servera identiteta.

Authorize endpoint prvo validira neophodne parametre koje je klijentska aplikacija poslala u toku procesa autentikacije kao što su identifikator klijenata (client\_id) , dozvoljeni opseg prava pristupa tokena (scope), a zatim proverava da li je korisnik prethodno prosao kroz process autentikacije tako što proverava I validira sesioni kolačić. Ukoliko kolačić nije validan ili nije prisutan , onda authorize metoda navigira pretrazivac na stranu za prijavljivanje I dodaje parameter ReturnUrl koji sadrzi url authorize metode na koju opet biva redirektovan nakon unosa korisničkih kredencijala I postavljanjem kolačica sesije.

Kada se zadovolje svi uslovi authorize metoda vraća pretraživacu potrebne parametre preko 302 Http redirect-a sa Kodom (ukoliko se koristio tok autorizacionim kodom) I Tokenom identiteta (ukoliko se koristio Hibridni ili Implicitni tok).

**Sign In**

Kao što smo objasnili Authorize endpoint unutar Identity Server 4 biblioteke proverava da li je korisnik prošao kroz proces autentikacije tako što proverava kolačić sesije.Međutim sam proces autentikacije korisnika i generisanje potrebnog kolačića sesije nije u domenu biblioteke i prepušteno je sopstvenoj implementaciji.

Da bi obezbedili krajnjem korisniku interfejs za autentikaciju potrebno je implementirati nekoliko stvari:

* **Sam korisnički interfejs za unos potrebnih kredencijala (uglavnom Email i Šifra)**
* **Endpoint unutar našeg IdentityServer solutiona za prijem korisničkih kredencijala**
* **Potrebnu validaciju korisničkih kredencijala i izdavanje kolačića sesije**

**Tehnologije za implementaciju Sign-ona**

Što se tiče samog korisničkog interfejsa koristicemo ugrađeni **Razzor Engine** unutar same .Net Core 5 IdentityServer aplikacije (Treba napomenuti da korisnički interfejs nije uslovljen .Net tehnologijom tako da mozemo koristiti i Angular,React ili jednostavan HTML zajedno sa Javascriptom)

Endpoint za prijem korisničkih kredencijala je u sklopu našeg IdentityServer projekta i za svrhu autentikacije napravićemo **Authentication** kontroler zajedno sa **SignIn** endpointom za prijem.

Za potrebnu validaciju kredencijala, menagementom usera, kao i izdavanjem potrebnih kolačića sesije koristicemo **AspNet Identity Core** biblioteku u sklopu .Net Core 5 framework-a

**Sign In korisnički interfejs**

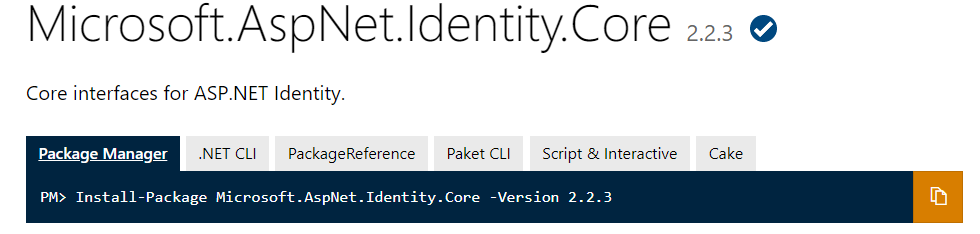


Slika *11* *IdentityServer Sign In korisnički interfejs*

Kao što smo napomenuli za izradu Sing In korisničkog interfejsa koristicemo ugrađeni Razzor Engine u sklopu .Net Core 5 framework-a (slika 11) . Sastoji se od jednostavne post forme za unos korisničkih kredencijala kao i dodatnog **ReturnUrl** parametara koji je zapravo url putanja do authorize metode, gde se redirektujemo nakon formiranja kolačića sesije.

**Validacija kredencijala i kreiranje kolačića sesije**

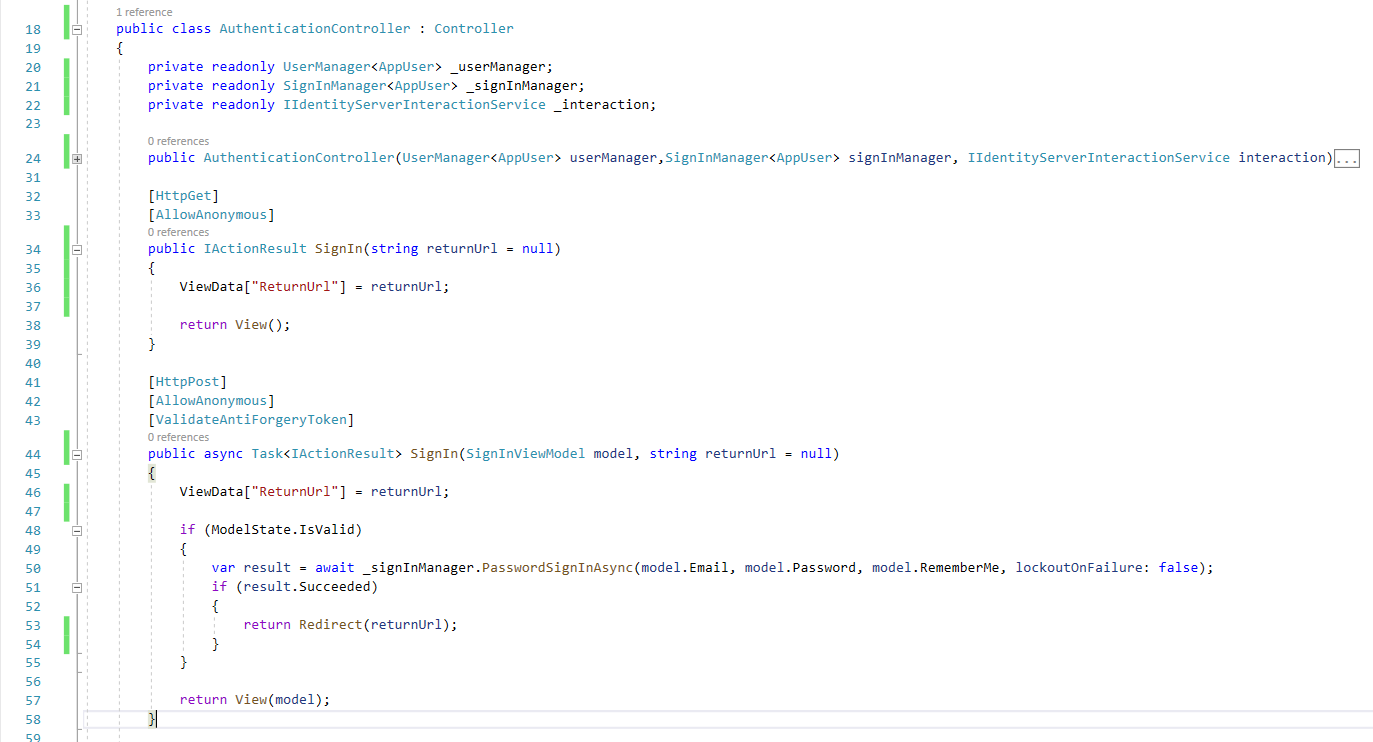
Za validaciju kredencijala i generisanje kolačića sesije koristicemo biblioteku (slika 12) za korisnički management u sklopu .Net Core 5 frameworka, koja nam pruza potrebne olakšavajuce servise poput **UserManager**-a, **SignInManager**-a i metode poput **PasswordSignIn** za autentikaciju korisnika i generisanje/brisanje kolačica.



Slika *12* *IdentityServer AspNet Identity Core nuget*

**Endpoint za prijem korisničkih kredencijala**

Na kraju lepak koji spaja korisnički interfejs i potrebnu validaciju korisničkih kredencijala jeste metoda unutar naseg **IdentityServer** projekta. Za ovu svrhu napravicemo poseban kontroler po imenu Authentication kao i dve metode za **SignIn**.Jedna je **Get** za slanje korisničkog interfejsa pretraživaču i drugu **Post** za prijem korisničkih kredencijala (slika 13).



Slika *13* *IdentityServer metoda za prijem korisničkih kredencijala*

Kao što smo napomenuli , prvi korak u toku autorizacije klijentske aplikacije jeste poziv **autorize** metode u sklopu **identity server 4** framework-a. Ukoliko authorize metoda detektuje da korisnik nije prošao proces autentikacije (proverava prisutnost i validira kolačić sesije) , ona ce redirektovati korisnički pretrazivač na get akciju (u nasem slučaju get **SignIn** metodu) sa **returnUrl** parametrom koji predstavlja url za povratak na authorize metodu sa parametrima.

**SignIn Get** metoda jednostavno preko ViewData prosledjuje returnUrl parametar i vraća korisnički interfejs.

**SignIn Post** metoda prihvata korisničke kredencijale kao i **returnUrl** parametar (returnUrl se zapravo samo preneo korisničkom interfejsu a zatim se isti samo vratio kroz post formu nazad serverskoj metodi)

Za validaciju korisničkih kredencijala jednostavno koristimo **Asp Net Identity Core** bibliotečku funkciju **PasswordSignInAsync** kroz **signInManager** , koja ce nakon validacije generisati potreban kolačić sesije.

Nakon generisanja kolačića dovoljno je jednostavno odraditi redirekt nazad ka **Authorize** metodi preko **returnUrl**-a , koja će ovoga puta uspesno validirati korisnika i vratiti klijentskoj aplikaciji kod ili token identiteta zavisno od tipa toka autorizacije preko koga klijent pristupa ( Authorization Code, Hybrid,Implicit flow )

**Token Endpoint**

Kada klijentska aplikacija dobije nazad kod autorizacije od strane Servera Identiteta, sa tim kodom salje zahtev za dobijanje tokena tako što zajedno sa kodom salje I **client\_id**, **client\_secret** (kada se koristi privatni klijent, u našem slucaju MVC Client) ili **code\_verifier** ( kada se koristi javni klijent sa PKCE tokom, u našem slučaju angular client).

Identity server 4 framework za ovu svrhu implementira Post **/connect/token** metodu koja proverava I validira zahtev I nakon toga vraća klijentskoj aplikaciji potrebne tokene (id\_token, access\_token, refresh\_token)

**User Info Endpoint**

S obzirmo da je token identiteta JWT token I da se skladišti u korisnički pretraživač , ne preporučuje se da zauzima previse memorije, odnosno da sadrži previše informacija (tvrdnji) o identitetu korisnika. Zbog toga se pokazalo kao najbolja praksa da token identiteta sadrži samo neophodne tvrdnje. Ukoliko su klijentskoj aplikaciji potrebne dodatne informacije o identitetu korisnika , **Identity Server 4** implementira posebnu metodu **connect/userinfo** koja vraća dodatne informacije o korisniku.

UserInfo metoda je zaštićena za anonimne korisnike i moguce je pristupiti samo ukoliko klijentska aplikacija poseduje token za prava pristupa (access\_token) , koji je dobila nakon Token endpointa.

Zahtev od strane klijentske aplikacije se jednostavno salje gađajuci Get userinfo metodu , i slanjem tokena za prava pristupa u zaglavlju http zahteva unutar Authorization polja.

GET /connect/userinfo

Authorization: Bearer <access\_token>

**Profile Service**

Pre nego što server identiteta izda tokene klijentskoj aplikaciji potrebno je definisati koje tvrdnje ce se naci unutar samih tokena na osnovu zahtevanog opsega tvrdnji definisanih unutar registrovanih tvrdni kao i registrovanih resursa identiteta.

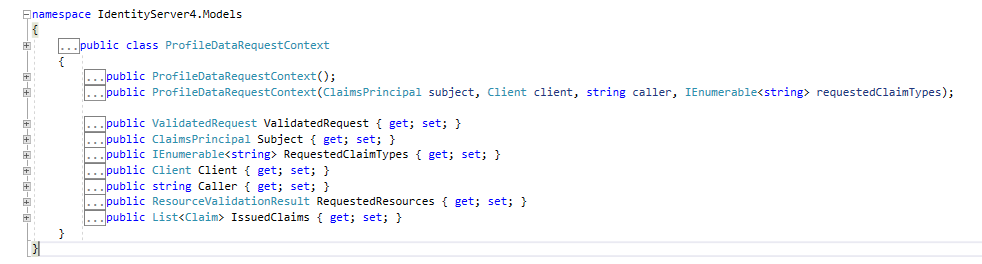
Identity server 4 framework definiše poseban interfejs za kontrolu i definisanje tvrdnji pod nazivom IProfileService i potrebno ga je implementirati na sledeci način (slika 14).



Slika *14* *Identity Server 4 ProfileService metoda za definisanje tvrdni unutar tokena*

**GetProfileDataAsync** ce se pozvati automatski pre izdavanja tokena klijentskoj aplikaciji i ostaje na nama da definišemo pravila i uslove izdavanja specificnih tvrdnji unutar tokena.

GetProfileDataAsync kao parametar zahteva **ProfileDataRequestContext** **context** (slika 15) u kome se nalaze potrebne informacije za validaciju i provere specificnih zahteva poput tipa tokena za koji se poziva ( da li se radi o generisanju tokena identiteta ili user info zahtev ili zahtev za token prava pristupa) , kao i na primer identifikator klijentske aplikacije koja zahteva tokene (mozemo definisati neke posebne tvrdnje za specificne klijente) itd.



Slika *15* *Identity Server 4 ProfileService zahtev za dobijanje tvrdnji unutar tokena*

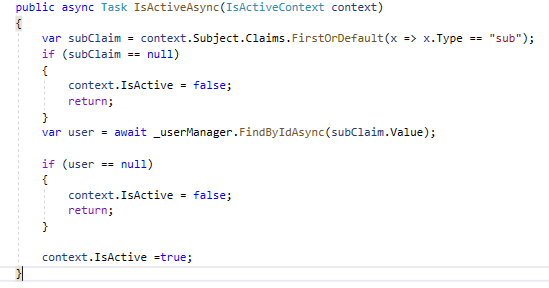
Na kraju tvrdnje koje ce biti ubačene u zahtevani token se definisu unutar **context.IssuedClaims** niza koga treba napuniti potrebnim tvrdnjama.

Pomocu **context.Caller** parametra mozemo utvrditi tip tokena koji se zahteva ( id\_token, access\_token ili userInfo endpoint) i u skladu s tim ispitati potrebne uslove.

Unutar **context.RequestedResources** se nalaze informacije o dozvoljenim grupama tvrdnja (openid, email, custom.profile) koje odredjeni klijent moze da zahteva preko **scope** parametra i koje smo definisali u sklopu registracije klijenata i resursa identiteta.

Unutar **context.Subject.Claims** parametra se nalaze zapravo tvrdnje izvučene iz sesionog kolačića definisanog pomoću ASP Net Identity core biblioteke i mozemo koristiti ove tvrdnje za kreiranje niza tvrdnji unutar tokena. Ukoliko unutar kolačića nije definisana tvrdnja koja nama treba , na primer prilikom User Info endpointa, klijentska aplikacija moze zahtevati dodatne tvrdnje koje se ne nalaze unutar kolačića sesije. U te svrhe može se iskoristiti **UserManager** u sklopu ASP Net Identity Core biblioteke da bi se izvukle dodatne tvrdnje o korisniku.

IProfileService takodje definise metodu IsActiveAsync pomocu koje mozemo validirati da li postoji i da li je aktivan korisnik za koga se zahtevaju informacije o idenitetu i tokeni (slika 16).

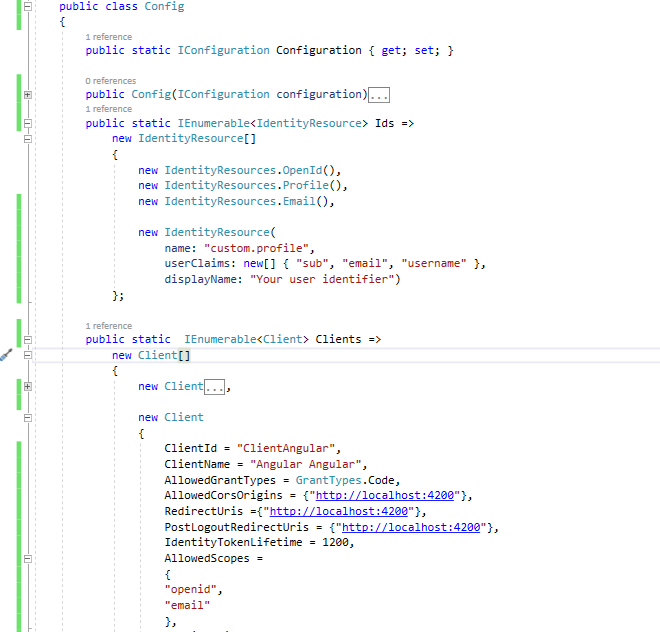


Slika *16* *Identity Server 4 ProfileService validacija korisnika pre izdavanja tokena*

**IdentityServer postavka projekta**

Sada kada smo definisali potrebne resurse identiteta , klijente , stranicu za prijavljivanje, kao i ProfileService , potrebno je sve povezati unutar IdentityServer projekta.

Za definisanje resursa identiteta i klijenta koristicemo klasu config koja sadrzi jedan niz dostupnih resursa identiteta kao i jedan niz klijenata (slika 17).



Slika *17* *IdentityServer Definisanje resursa identiteta I klijenata*

*Da bi unutar projekta koristili IdentityServer 4 framework potrebno je instalirati nuget paket (objasnjeno u predhonoj sekciji).*

*Zatim unutar Startup klase našeg IdentityServer projekta treba omoguciti korišćenje ovog paketa (slika18).*



Slika *18* *IdentityServer integracija IdentityServer 4 framework-a*

*Na kraju je potrebno registrovati IdentityServer4, klijente, resurse identiteta, Asp Net Identity integraciju , konekciju sa bazom gde ćemo čuvati informacije o korisnicima kao I ProfileService (slika 19).*

**

Slika *19 IdentityServer registrovanje servisa*