&Bcy;&rcy;&ocy;&jsercy; &scy;&lcy;&ocy;&bcy;&ocy;&dcy;&ncy;&icy;&khcy; &mcy;&iecy;&scy;&tcy;&acy; &zcy;&acy; &scy;&iecy;&pcy;&tcy;&iecy;&mcy;&bcy;&acy;&rcy;&scy;&kcy;&icy; &ucy;&pcy;&icy;&scy;&ncy;&icy; &rcy;&ocy;&kcy; 2012**Univerzitet u Nišu, Elektronski fakultet**

**Katedra za računarstvo**

**Jovan Pešić**

Analiza i implementacija Single Sign On metode autentifikacione šeme

**Diplomski rad**

**Niš, 2021**

&Bcy;&rcy;&ocy;&jsercy; &scy;&lcy;&ocy;&bcy;&ocy;&dcy;&ncy;&icy;&khcy; &mcy;&iecy;&scy;&tcy;&acy; &zcy;&acy; &scy;&iecy;&pcy;&tcy;&iecy;&mcy;&bcy;&acy;&rcy;&scy;&kcy;&icy; &ucy;&pcy;&icy;&scy;&ncy;&icy; &rcy;&ocy;&kcy; 2012**Univerzitet u Nišu, Elektronski fakultet**

**Katedra za računarstvo**

Analiza i implementacija Single Sign On metode autentifikacione šeme

**Diplomski rad**

**Zadatak:**

**Mentor: Prof. dr Ivan Petković Kandidat: Jovan Pešić, 15301**

**Komisija:**

**1. \_\_\_\_Prof. dr Ivan Petković\_\_\_\_\_\_\_\_\_ Datum prijave: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2. \_\_\_\_Prof. dr Milorad Tošić \_\_\_\_\_\_\_\_ Datum predaje: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3. \_\_\_\_Doc. dr Petar Rajković \_\_\_\_\_\_\_\_ Datum odbrane: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Niš, 2021**

Sadržaj

[1. Uvod 5](#_Toc81329348)

[2. Single Sign On 6](#_Toc81329349)

[2.1. Šta je Single Sign On (SSO)? 6](#_Toc81329350)

[2.2. Prednosti Single Sign On-a 8](#_Toc81329351)

[2.3. Problemi Single Sign On-a 8](#_Toc81329352)

[3. OIDC i OAuth2 9](#_Toc81329353)

[3.1. Poređenje ključnih razlika između OIDC i SAML protokola 10](#_Toc81329354)

[3.2. OpenId Connect (OIDC) 11](#_Toc81329355)

[3.3. OAuth 2.0 protokol 11](#_Toc81329356)

[3.4. Primeri upotrebe tokova dozvole 15](#_Toc81329357)

[3.5. Tok Autorizacionim kodom (Authorization Code Flow) 16](#_Toc81329358)

[3.5.1. Zahtev za autentikaciju 17](#_Toc81329359)

[3.5.2. Uspešan odgovor nakon potvrde identiteta 18](#_Toc81329360)

[3.5.3. Zahtev za dobijanje tokena 18](#_Toc81329361)

[3.5.4. Odgovor Token Metode 19](#_Toc81329362)

[3.5.5. Validacija Tokena 19](#_Toc81329363)

[3.6. Implicitni Tok (Implicit Flow) 20](#_Toc81329364)

[3.6.1. Zahtev za autentikaciju 21](#_Toc81329365)

[3.6.2. Uspešan odgovor nakon potvrde identiteta 21](#_Toc81329366)

[3.7. Hibridni Tok 22](#_Toc81329367)

[3.8. Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE) 23](#_Toc81329368)

[3.8.1. Zahtev za autentikaciju 23](#_Toc81329369)

[3.8.2. Uspešan odgovor nakon potvrde identiteta 24](#_Toc81329370)

[3.8.3. Zahtev za dobijanje tokena 24](#_Toc81329371)

[3.8.4. Validacija tokena identiteta 25](#_Toc81329372)

[4. Single Sign Out 27](#_Toc81329373)

[4.1. Šta je Single Sing Out? 27](#_Toc81329374)

[4.2. OIDC Sign Out 27](#_Toc81329375)

[4.3. Klijentska aplikacija inicira Logout 28](#_Toc81329376)

[4.3.1. Session Management 29](#_Toc81329377)

[4.3.2. Front-channel Logout 29](#_Toc81329378)

[4.3.3. Back-channel Logout 29](#_Toc81329379)

[5. Arhitektura Sistema 32](#_Toc81329380)

[6. Identity Server Solution 34](#_Toc81329381)

[6.1. Identity Server 4 34](#_Toc81329382)

[6.1.1 Standardni scenariji veb aplikacija 34](#_Toc81329383)

[6.1.2. Discovery endpoint 36](#_Toc81329384)

[6.1.3. Identity Resource 36](#_Toc81329385)

[6.1.4 Klijenti 38](#_Toc81329386)

[6.1.5 MVC Klijent 38](#_Toc81329387)

[6.1.6 Angular Klijent 39](#_Toc81329388)

[6.1.7 Authorize Endpoint 40](#_Toc81329389)

[6.2 Sign In 40](#_Toc81329390)

[6.2.1 Tehnologije za implementaciju Sign-ona 41](#_Toc81329391)

[6.2.2 Sign In korisnički interfejs 41](#_Toc81329392)

[6.2.3 Validacija kredencijala i kreiranje kolačića sesije 42](#_Toc81329393)

[6.2.4 Endpoint za prijem korisničkih kredencijala 42](#_Toc81329394)

[6.3 Token Endpoint 43](#_Toc81329395)

[6.4 User Info Endpoint 44](#_Toc81329396)

[6.5 Profile Service 44](#_Toc81329397)

[6.6. IdentityServer postavka projekta 47](#_Toc81329398)

[7. Angular Client 49](#_Toc81329399)

[7.1 PKCE tok autorizacije 49](#_Toc81329400)

[7.2 Angular Lib for OpenID Connect & OAuth2 49](#_Toc81329401)

[8. MVC Client 52](#_Toc81329402)

[8.1. ClientMVC definisanje klijenta 53](#_Toc81329403)

[9. Zaključak 56](#_Toc81329404)

[Reference 57](#_Toc81329405)

# Uvod

Termin Single Sign On (SSO) se odnosi na mogućnost korisnika da prijavljivanjem na jednu aplikaciju unosom svojih kredencijala, bude automatski prijavljen I u ostalim aplikacijama, bez obzira na platformu, tehnologiju ili domen koji korisnik koristi.Korisnik se prijavljuje samo jednom pa otuda I naziv Single Sign On.

Na primer ako se prijavite na Google uslugu kao što je Gmail, automatski ćete biti autentikovani na Youtube, Google Analytics, Google Cloud I na ostale Google aplikacije.

U teorijskom delu ovog rada ćemo objasniti detaljnije šta je SSO, koje su prednosti I mane kao I protokole koji se koriste da bi se postigla ova funkcionalnost. Fokus će biti na implementaciji OpenId Connect – protokola gde ce biti detaljnije objašnjeno zbog čega se trenutno smatra kao najbolje rešenje za implementaciju Single Sign On-a

Da bi se postigao SSO uvodi se i pojam tokena identiteta (eng ID token) koji se koriste u cilju autentikacije korisnika na različitim aplikacijama. OpenID Connect protokol definiše nekoliko načina odnosno toka za pribavljanje ovih tokena, i u ovom radu će biti detaljnije objašnjeno koje tokove treba koristiti da bi se obezbedila maksimalna sigurnost u zavisnosti od prirode samih aplikacija.

Zatim ce biti teorijski obrađen pojam odjave korisnika iz aplikacija (Single Sign Out), koji su tipovi odjava kao i problemi koji se javljaju prilikom implementacije.

U praktičnom delu rada biće implementirana i objašnjena potrebna arhitektura sistema, kao i različite tehnologije korišćene u implementaciji u cilji demonstracije nezavisnosti Single Sign On-a od tehnologija.U realizaciji praktičnog dela rada koristiće se .Net Core, Angular, Microsoft SQL tehnologije.Takođe kroz primere će biti objašnjene i neke od biblioteka korišćene za implementaciju OpenId Connect protokola.

Radi demonstracije Single Sign On-a biće implementirane 2 Web aplikacije korišćenjem različitih tehnologija , kao i mogućnost korisnika da nakon sto se prijavi na jednu od aplikacija , bude automatski prijavljen i na drugu bez potrebe za ponovom prijavom, odnosno unosom kredencijala.

# Single Sign On

## Šta je Single Sign On (SSO)?

Single Sign On je metoda koja omogućava korisnicima da se autentikuju preko više aplikacija ili vebsajta, samo jednom koristeći iste kredencijale (na primer unosom korisničkog imena i lozinke).

Single Sign On autentikacija sada postaje tražena više nego ikada. U današnje vreme skoro svaka aplikacija ili vebsajt zahteva neku formu autentikacije da bi imala pristup zaštićenim resursima. Kako broj aplikacija i servisa raste, tako potreba za centralizovanim sistemom prijave postaje neophodna.

Koncept centralizovanog sistema prijave je poznat kao Ujedinjeni sistem identiteta (eng. Federated Identity).

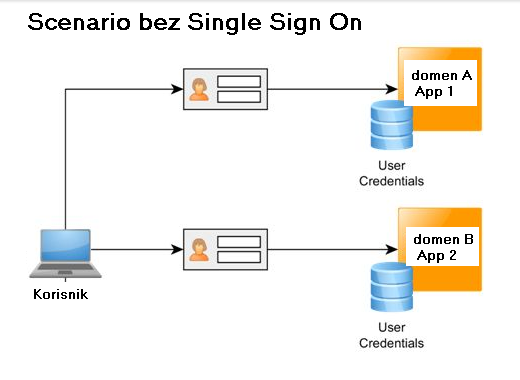
Ujedinjeni sistemi identiteta rešavaju nekoliko problema:

* Autentifikaciju
* Pravo pristupa resursima
* Razmena korisnickih podataka
* Upravljanje korisnicima

Aspekt Autentifikacije bavi se potvrđivanjem i validiranjem korisničkih podataka, tj. kredencijala, i utvrđivanjem identiteta korisnika.

Pravo pristupa resursima kao što ime kaže se bavi restrikcijama pristupa određene aplikacije određenim resursima (na primer da li aplikacija ili korisnik ima pravo da pristupi nekom resursu).

Razmena korisničkih podataka bavi se razmenom podataka kroz različite sisteme za upravljanje korisnicima. Na primer polja kao što su “Pravo ime“ mogu biti prisutna u više sistema. Ujedinjeni sistem identiteta sprečava dupliranje podataka povezivanjem povezanih atributa (eng. linking).



Slika 1. Scenario bez Signle Sign On-a

Upravljanje korisnicima je povezano sa samom administracijom (registracija novog korisnika, brisanje ili ažuriranje postojećeg korisnika).

Single Sign on je strogo povezan autentifikacionom delu ujedinjenog sistema identiteta. Kada se korisnik jednom autentikuje unosom kredencijala, preko ujedinjenog sistema identiteta se podaci o samom identitetu dele izmedju podsistema koji zahtevaju podatke.

U razvoju veb aplikacija pre ili kasnije se javlja potreba da korisnik koji se prijavio preko domena A bude automatski prijavljen i na aplikaciji sa domenom B (to je i svrha Single Sign On).

Bez Single Sign On mehanizma korisnik bi morao da se autentifikuje unešenjem kredencijala na svakoj aplikaciji (domenu) kojoj pristupa, bez obzira što te aplikacije čine jednu veću logičku celinu (Slika 1).

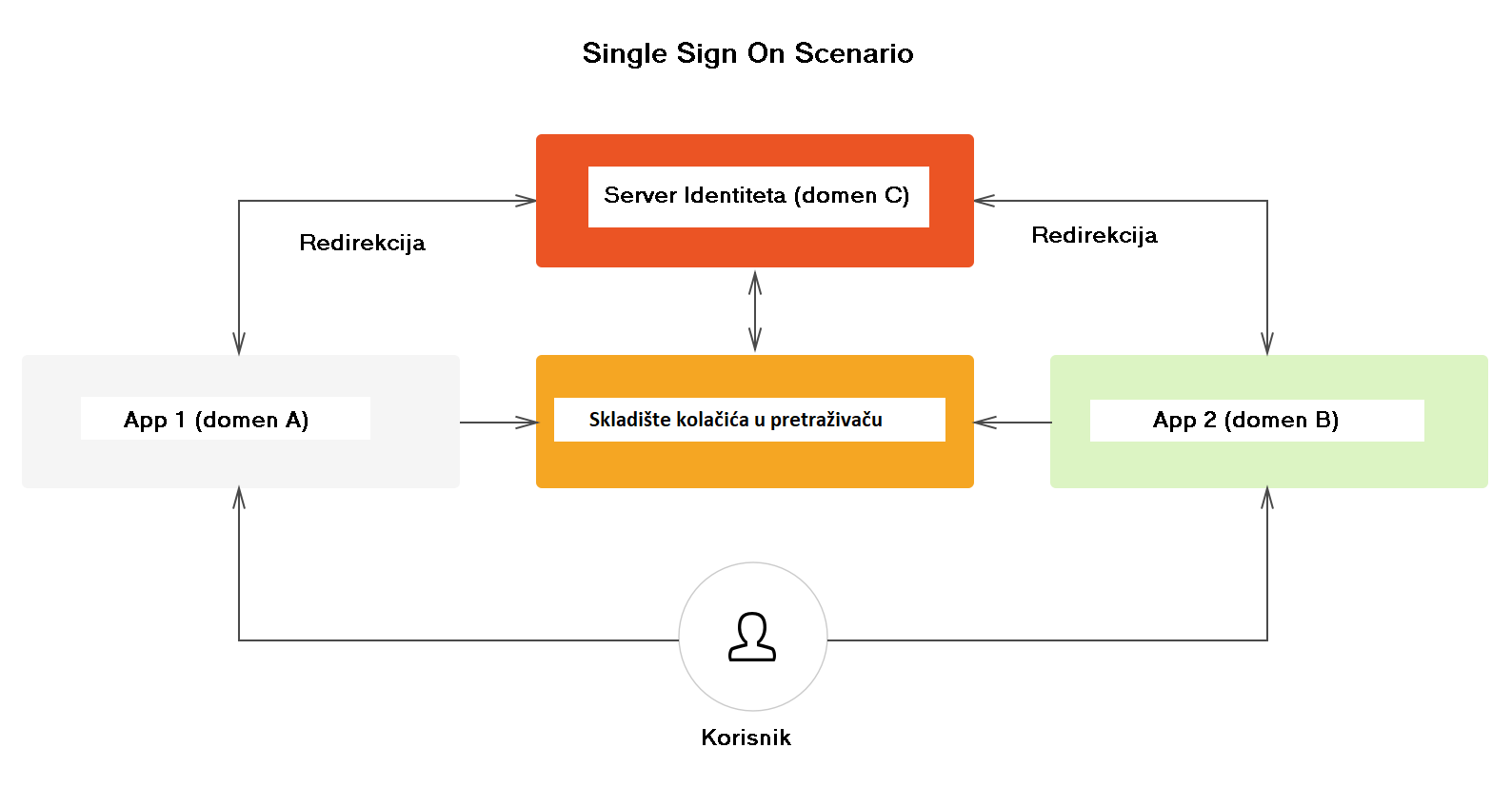
Očigledno resenje ovakvog problema jeste da informacije (identiteta) dele sesije između različitih domena. Međutim iz sigurnosnih razloga, veb pretraživači forsiraju politiku zvanu Ista politika porekla.

Ova politika definiše da kolačići (ili drugi podaci) mogu da dostupni samo njihovom vlasniku.

Drugim rečima domen A ne može da pristupi kolačićima domena B i obrnuto.

Postoje različiti Single Sign On protokoli koji omogućavaju deljenje informacija sesije (kolačića), ali suštinski koncept je isti. Postojanje Ujedinjenog Sistema Identiteta (na posebnom domenu C) preko koga je izvrsena autentikacija i preko koga se sesija deli sa ostalim domenima na neki način.

Kada korisnik ode na aplikaciju sa domenom A koja zahteva autentikaciju, automatski biva redirektovan na domen C (Ujedinjeni sistem identiteta). Kako je korisnik već prošao proces autentikacije na domenu C, na primer preko domena B, korisnik će biti vraćen na originalni domen A sa potrebnim informacijama identiteta bez potrebe za ponovnim unosom kredencijala (Slika 2) [1].



Slika 2. Scenario sa Sigle Sing On-om

Trenutno najpoznatiji protokoli za Single Sign On su: OpenID Connect, Facebook Connect, Security Assertion Markup Language (SAML), Microsoft Account [2].

Za dalji rad izabrana je implementacija OpenID connect (OIDC) protokola za postizanje Single Sign On-a.

Nakon uspešne autentikacije za dozvolu prava pristupa određenim resursima, koristiće se implementacija Oauth2 protokola (OAuth 2.0 je industrijski standardni protokol za autorizaciju).

## Prednosti Single Sign On-a

Korisnicima je glavna prednost što više ne moraju da pamte na desetine lozinki. Ovo čak oslobađa korisnika od upravljanja pojedinačnih lozinki na svakoj aplikaciji ponaosob, zbog čega se SSO prijave takođe smatraju i menadžerima lozinki. Smatrajući da je ovo tako zgodna alternativa i zato što štedi vreme i gnjavažu, SSO rešenja se koriste kako u privatnom tako i u profesionalnom kontekstu [3].

Kompanije koje primenjuju SSO u svom radu očekuju veću produktivnost svojih zaposlenih i manje poziva službe za pomoć zbog zaboravljenih lozinki. Zbog toga, IT ima manje posla i manje košta. Ištovremeno, IT osoblju olakšava dodeljivanje novih naloga novim zaposlenima ili brisanje pristupa bivšim zaposlenima.

Takođe, lozinku treba uneti samo na jednom interfejsu, a to smanjuje potencijal za napade (pecanje) u sistemu. Pod ovim okolnostima, kompanija može sebi priuštiti da usresredi rad na zastiti samo na jednom mestu. Kao što su na primer SSL sertifikati. Povećava brzinu tamo gde je napotrebnija. U okruženjima kao što su bolnice, odbrambena industrija i hitne službe, gde veliki broj ljudi i odeljenja zahteva brz i nesmetan pristup istim aplikacijama.

## Problemi Single Sign On-a

Sa druge strane, postoje određeni problemi i napori u primeni, kao i slabosti samog SSO. U osnovi, mogu se koristiti samo one usluge koje podržava odgovarajući sistem SSO. Ako sistem zakaže pristup povezanim aplikacijama biće ograničen ili nemoguć. U ovom slučaju SSO postaje takozvano usko grlo.

Takođe treba uzeti u obzir i sigurnost SSO prijave. Ako korisnik napusti svoju aplikaciju, a da nije izvršio odjavljivanje, treća strana bi teoretski mogla da iskoristi vreme do automatske odjave (eng. Single Sign Out), kako bi iskoristila pristup odobren putem SSO prijave. Pored toga, problematično je ako “glavna lozinka“ za SSO interfejs padne u pogrešne ruke. To napadaču daje trenutni pristup svim povezanim uslugama.

Postoji zabrinutost i zbog GDPR-a (eng. General Data Protection Regulation), u kojem su zahtevi za zaštitu ličnih podataka pooštreni u celoj Evropi od 25. Maja 2018. godine. Sada je potrebno dobiti izričit odgovor od korisnika da bi mogli da koriste SSO prijavu. U prošlosti je bio potreban i ovaj sporazum, ali su zakonitosti toliko promenjene da je situacija sada mnogo stroža.

S obzirom na ove potencijalne rizike, potrebno je obratiti posebnu pažnju na sigurnost podataka koji se čuvaju na strani servera. Ima smisla povećati sigurnost SSO funkcija koristeći na primer dvofaktorsku potvrdu identiteta (eng. two-factor authentication) ili druga rešenja kao što su pametne kartice (smart cards) ili tokeni.

Ako neko istraži SSO prijavu na Internetu, pronalazi relativno malo negativnih informacija o postupku višestruke potvrde identiteta. Umešto toga, godinama se to tretira kao sasvim prikladno otkriće za digitalno radno mešto, što se tiče udobnosti i sigurnosti podataka.

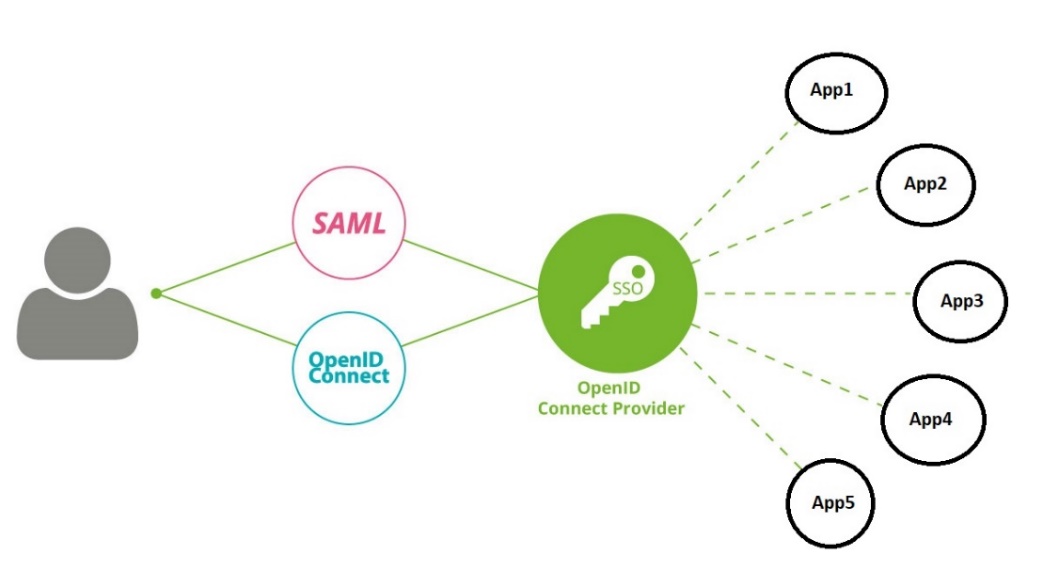
US-American Cloud Access Security Broker (CASB) hvali globalnu upotrebu usluga u oblaku u kompanijama, ali ištovremeno kritikuje relativno nisku upotrebu metoda za višestruku potvrdu identiteta (SSO prijava).

# OIDC i OAuth2

U predhodnom odeljku je objašnjen koncept Single Sign On-a (SSO) i rečeno da postoje različiti protokoli za implementaciju SSO-a (OpenIdConnect, Security Assertion Markup Language). Kao i da postoji potreba za centralizovanim sistemom prijave, odnosno Serverom Identiteta (eng. IdP - Identity Service Provider). Takođe je rečeno da je za dalji rad izabrana implementacija OIDC i OAuth2 protokola. OAuth2 omogućava aplikacijama pristup zaštićenim resursima korisćenjem tokena zvanim Access Tokens.

OIDC je izgrađen na vrhu OAuth2 protokola i koristi dodatni token u JWT (eng. Json Web Token) formatu, koji se naziva ID Token. Posebno je fokusiran na autentikaciju korisnika i široko se koristi za omogućavanje korisničkih prijava na veb sajtovima i u mobilnim aplikacijama. S obzirom da je OIDC nadogradnja OAuth2 protokola, predstavlja potpuno razvijen protokol za autentikaciju i dodelu prava pristupa (autorizaciju), koji intenzivno koristi sigurnosne JSON tokene za komunikaciju izmedju servisa, aplikacija i servera identiteta (IdP) [4].

SAML je nezavisan od OAuth2 protokola, i koristi XML format za razmenu poruka za potvrdu identiteta,za razliku od JWT formata koji koristi OIDC. Češće se koriti za pomoć korisnicima korporacija da se prijave u više aplikacija pomoću jedne prijave.



Slika 3. SAML i OpenID konekcija

## Poređenje ključnih razlika između OIDC i SAML protokola

Oba protokola se koriste za postizanje istih ciljeva, međutim sam način kako autentikuju korisnika se razlikuje, korišcenjem drugačijih tehnologija i metoda.

Suštinska razlika izmedju ovih protokola je ta što se OAuth2 koristi za kontrolu prava pristupa zaštićenim resursima (neka aplikacija ili web Api ili datoteka), dok su OIDC i SAML industrijski standardi za potvrdu identeta odnosno autentifikaciju. OAuth2 se koristi u potpuno drugačijim situacijama od druga dva standarda, a može se koristiti ištovremeno u kombinaciji sa OIDC-om i SAML-om, što uglavnom i jeste slučaj [5].

**RP** **nasuprot SP:** Oba protokola, RP (eng. Relying Party, termin za klijentsku aplikaciju) u terminologiji OIDC-a, i SP (eng. Service Provider) u slučaju SAML-a, redirektuju korisnika serveru identiteta (IdP ili OP). Razlika je u tome što SAMP nije bas kompatibilan sa određenim tipom aplikacija, kao što su na primer mobilne aplikacije, u poređenju sa OIDC-om koji jednako dobro funkcioniše kako sa veb tako i sa mobilnim aplikacijama.

**Format Poruka:** SAML koristi tvrdnje koje predstavljaju atribute autentikacije kao i takozvane izjave za dodelu prava pristupa i formatirane su korišćenjem XML formata. Sa druge strane OIDC koristi JSON Web tokene pod nazivom ID Tokeni u JWT formatu, koji pruža informacije za potvrdu identiteta.

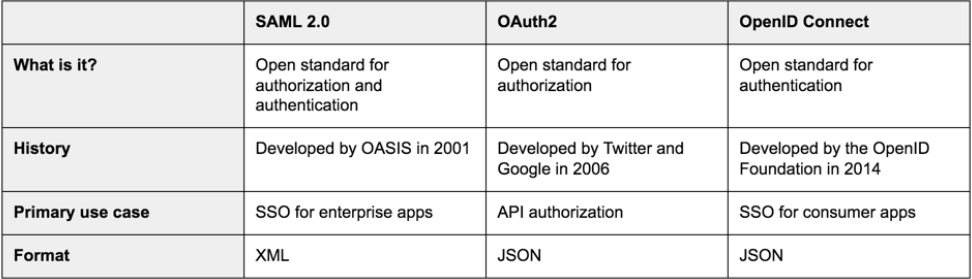
**Podrška za API:**  OIDC je nastao iz razloga što je SAML Xml format “Težak”, i ne može se dobro integrisati sa Api-ima. OIDC radi tako što koristi HTTP komunikacioni kanal za slanje “Lakih” JSON sigurnosnih tokena za potvrdu identiteta korisnika. Sa druge strane SAML koristi SOAP, koji je takođe protokolski sloj preko HTTP-a, međutim šalje “Teške” poruke u XML formatu za potvrdu identiteta korisnika.

**Autentikacija korišćenjem mobilnih aplikacija:** OIDC koristi RESTFUL komunikaciju za stvaranje i prosleđivanje “Laganih“ JSON tokena, koji se prosleđuju između servera identiteta (IdP) i pouzdane strane (eng. Relayng Party), odnosno klijentske aplikacije, što čini OIDC vodećim protokolom za mobilne aplikacije. Za razliku od SAML-a koji je uglavnom razvijen za autentikaciju veb aplikacija, a budući da koristi XML, nema prostora za buduću upotrebu u mobilnim aplikacijama, što daje prednost OIDC-u.

**Korisnikov Pristanak:** S obzirom da je OIDC izgrađen na vrhu OAuth2 protokola za podršku autentikacije, može pružiti ugrađeni sloj za prava pristupa (Autorizacije), koji podstiče korisnika da prvo pristane na ono što pružalac usluga (IdP) može obezbediti odnosno dati pristup. SAMl takođe može obezbediti tok saglasnosti, međutim to čini kodiranjem samog programera (eng. Hard-Coding), umesto da obezbedi tok saglasnosti kao deo standarda u svom protokolu.

**Statična Autentikacija:**  U SAML protokolu Server Identiteta (IdP) i Pouzdana strana (relayng party) moraju biti konfigurisani da se poznaju pre stvarnog prenosa podataka. Sa druge strane za OIDC to nije slučaj.

**Implementacija:** OIDC protokol je daleko jednostavnije implementirati od strane programera, što povećava upotrebljivost samog protokola za različite svrhe. Takođe, može biti lako implementiran od nule ili korišćenjem već gotovih biblioteka koje su dostupne kao pomoć i podrška developerima.



Slika 4. Tabela protokola

## OpenId Connect (OIDC)

Rečeno je da je OIDC sloj za potvrdu identiteta (autentikaciju) koji je izgrađen na vrhu (ekstenzija) protokola za autorizaciju (OAuth2). To znači da OIDC koristi iste komponente i arhitekturu kao i OAuth2 ali za potvrdu identiteta (za razliku od OAuth2 koji izdaje access tokene za dodelu prava pristupa), OIDC se koristi za izdavanje ID tokena i koristi iste tokove odobrenja (eng. Authorization Grant Flows) kao i OAuth2 [6].

OAuth je prvobitno bio zamišljen kao rešenje nečega što je poznato kao delegirani pristup (eng. Delegated access) što predstavlja omogućavanje aplikacijama da međusobno komuniciraju i dele informacije u ime samog korisnika.

Ovaj problem je zahtevao rešenje koje bi standardizovalo način na koji dve ili više nepovezane aplikacije dele lične korisničke podatke. Iako OAuth2 to rešava, on zapravo ne prenosi podatke koji mogu identifikovati korisnika koji trađi povlastice, odnosno pravo pristupa (access token ne sadži informacije o samom korisniku već o tome koja aplikacija ima prava pristupa određenim resursima). OIDC dodaje funkciju potvrde autentičnosti u OAuth -ov program za autorizaciju.

Pre nego što uđemo u detalje tokova odobrenja koji su definisani u specifikaciji OAuth2 protokola, prvo ćemo proći i objasniti koncepte i terminologiju koja se koristi u OAuth2 protokolu.

## OAuth 2.0 protokol

Sledeći koncepti se koriste u OAuth 2.0 [7]:

**Vlasnik Resursa (Resource Owner)**

Entitet sposoban da odobri pristup zaštićenom resursu. Kada je entitet osoba, ona se naziva korisnik.

**OAuth Klijent**

Nezavisna aplikacija koja želi pristup privatnim resursima vlasnika resursa. OAuth klijent može da šalje zahteve resursima u ime vlasnika resursa, nakon što mu vlasnik dodeli ovlašćenje. OAuth uvodi dve vrste klijenata:

* Poverljivi: Registrovani su kao klijentova tajna (client secret)
* Javni

U terminologiji OIDC-a cešto se za klijente koristi i termin poverljiva strana odnosno aplikacija (eng. Relayng Party RP).

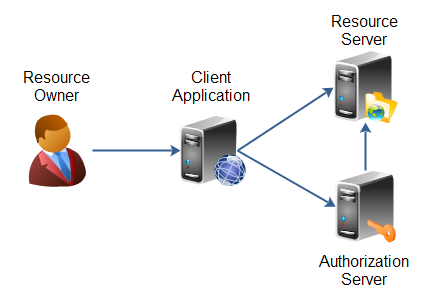
**OAuth server (IdP)**

Poznat kao server za autorizaciju (U OIDC je poznat po terminologiji Server Identiteta ili OpenID OP).

Server koji OAuth klijentima daje opseg i pravo pristupa zaštićenom resursu u ime vlasnika resursa. Server izdaje token pristupa (eng. access token), klijentu nakon što uspešno izvrši sledeće korake:

* Autentifikuje vlasnika resursa
* Validira zahtev i pravo za odobrenje
* Pribavlja Ovlašćenje vlasnika resursa

Server za autorizaciju takodje može biti i server resursa.



Slika 5. Koncept OAuth 2.0 protokola

**Token prava pristupa (Access Token)**

Niz karaktera koji predstavlja ovlašćenje koje je OAuth klijentu dodelio vlasnik resursa. Ovaj niz predstavlja određeni opseg kao i trajanje prava pristupa. Odobrava ga vlasnik resursa, a nameće OAuth Server.

**Zaštićeni resurs (Protected Resource)**

Zaštićeni resurs je resurs kome klijent pristupa kada server resursa validira access token, koji klijent šalje u zahtevu za određenim resursom.

**Server Resursa (Resource Server)**

Server koji hostuje zaštićene resurse. Koristi tokene prava pristupa (eng. access tokens) za validaciju i vraćanju resursa klijentu, ukoliko klijent ima prava pristupa na određeni resurs.

Server Resursa takodje može biti i OAuth Server.

**Token Osveženja (Refresh Token)**

Niz karaktera koji se koristi za dobijanje novog Tokena prava pristupa (access tokena). Token osveženja opciono izdaje server autorizacije klijentu zajedno sa tokenom prava pristupa. Kada istekne token prava pristupa, klijent može koristiti token osveženja da zatraži novi token pristupa koji se zasniva na istoj autorizaciji bez ponovnog uključivanja vlasnika resursa.

**Token Identiteta (ID Token)**

OIDC proširuje mogućnosti autentifikacije OAuth2 tako što uključuje takozvani Token identiteta izdat kao Json Veb Token (JWT). ID tokeni su konceptualno analogni ID karticama, jer sadrže skup tvrdnja (eng. claims) o korisniku, poput imena, e-pošte i ostalih ličnih podataka korisnika. Ovo se razlikuje od tokena prava pristupa (eng. access token) u tome što token prava pristupa ne uključuje nikakve informacije identiteta, već postoje za autorizaciju pristupa serverima resursa sa ograničenim opsegom pristupa. Token prava pristupa nije dokaz za potvrdu identiteta, zato što se tokeni prava pristupa mogu dobiti na više nacina. Zato se ID Tokeni koriste za potvrdu Identiteta korisnika.

**Kod Autorizacije (Authorization Code)**

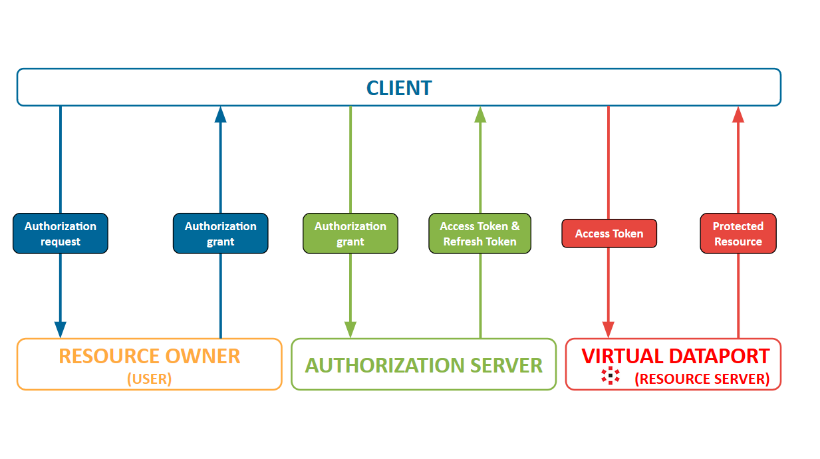
Kod koji server za autorizaciju generiše kada vlasnik resursa odobri zahtev i koristi se za validaciju klijenata pre izdavanja tokena prava pristupa (eng. access token-a).

**Dodela Odobrenja (Authorization Grant)**

Dozvola koja predstavlja ovlašćenje vlasnika resursa za pristup zaštićenim resursima. OAuth2 Klijenti koriste odobrenje za dobijanje tokena (access token, id token, refresh token). Postoje četiri tipa odobrenja autorizacije, odnosno tokova odobrenja:

* Tok Autorizacionim kodom (eng. Authorization Code Flow)
* Implicitni Tok (eng. Implicit Flow)
* Kredencijali Vlasnika resursa (eng. Resource Owner Password Credentials)
* Kredencijali Klijenta (eng. Client Credentials)
* Hibridni Tok (eng. Hybrid Flow) – Dodatno uveden od strane OIDC sa posebnim namenama
* Tok Autorizacionim kodom sa dokazom ključa za razmenu kodova (eng. Authorization Code Flow with Proof Key for Code Excange - PKCE)

Važno je napomenuti da ove tokove koristi i OIDC samo dodaje izdavanje tokena identiteta. U nastavku će biti objašnjen svaki od toka i koje su njihove prednosti i mane kao i preporučeni tipovi klijentske aplikacije za svaki od navedenih tokova.



Slika 6. Koncept toka odobrenja

S obzirom da je ovde fokus na Single Sign On-u koristeći OIDC protokol, detaljnije će biti objašnjeni samo tokovi koji se izvršavaju uz pomoć pretraživaća i samog korisnika sa ciljem autentikacije i izdavanja tokena identiteta.

U tokovima dozvole autorizacije sa takvom namenom spadaju:

* Tok Autorizacionim kodom (eng. Authorization Code Flow)
* Implicitni Tok (eng. Implicit Flow)
* Hibridni Tok (eng. Hybrid Flow) – Dodatno uveden od strane OIDC sa posebnim namenama
* Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (eng. Authorization Code Flow with Proof Key for Code Excange - PKCE)

Biće rečeno kada bi tačno trebalo da se koristi svaki od njih, uključujući primere gde bi mogao da se koristi neki tok, ali potencijalno može da izazove više problema nego što može da ih reši.

Još jedna stvar vredna napomene je da većina pružača usluga identiteta (OIDC OpenId providers) sada nudi biblioteke koje omogućavaju upotrebu ove funkcije, a da programeri nisu svesni svih detalja niskog nivoa implementacije.

## Primeri upotrebe tokova dozvole

Ukratko će biti napomenuto koji od tokova dozvole su preporučeni za korišćenje u zavistnosti od vrsta klijentskih aplikacija [8] .

**Veb Aplikacija sa serverskom stranom (eng. Web Application with dedicated server-side component)**

Veb aplikacije sa severskom stranom poput .Net MVC-a mogu sigurno čuvati klijentovu tajnu (eng. client secret, Confidential Client), tako da se za ovu vrstu aplikacija preporučuje korišćenje Toka Autorizacionim kodom.

**Desktop Aplikacija (eng. Desktop Native Application)**

Desktop aplikacija obično nema mogućnost da čuva tajnu klijenta, tako da se za ovakvu vrstu aplikacije preporucuje Tok Autorizacionim kodom sa dokazom ključa za razmenu kodova (eng. Authorization Code Flow with Proof Key for Code Excange - PKCE) sa javnim klijentom.

**Mobilna aplikacija (eng. Mobile Native Application)**

Za mobilnu aplikaciju takođe važi da ne može sigurno da čuva tajnu klienta, tako da se i za ovu vrstu aplikacije preporučuje Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (eng. Authorization Code Flow with Proof Key for Code Excange - PKCE) sa javnim klijentom.

**SPA(eng Single Page Applications) i Javascript aplikacije**

Ovakav tip aplikacija se izvršava u korisničkom pretraživaču što takodje ne predstavlja siguran način za čuvanje klijentske tajne. Za ovaj tip aplikacije može se koristiti Implicitni tok, medjutim on ima svoje sigurnosne rizike tako da se preporučuje korišćenje Toka Autorizacionim kodom sa dokazom kljuca za razmenu kodova sa javnim klijentom. (SPA rade tako što ne zahtevaju svaki put dovlacenje klijentskog koda u korisnikov pretraživač, vec koriste AJAX pozive za dinamičko osveženje stranice.Angular i React su jedni od poznatijih programskih jezika za implementaciju Single Page Aplikacija)

Mnoštvo aplikacija se neće tačno uklopiti u jednu od ovih kategorija. Pojedine situacije moraće se procenjivati na osnovu sopstvenih potreba i zahteva.

**Stvari koje treba uzeti u obzir:**

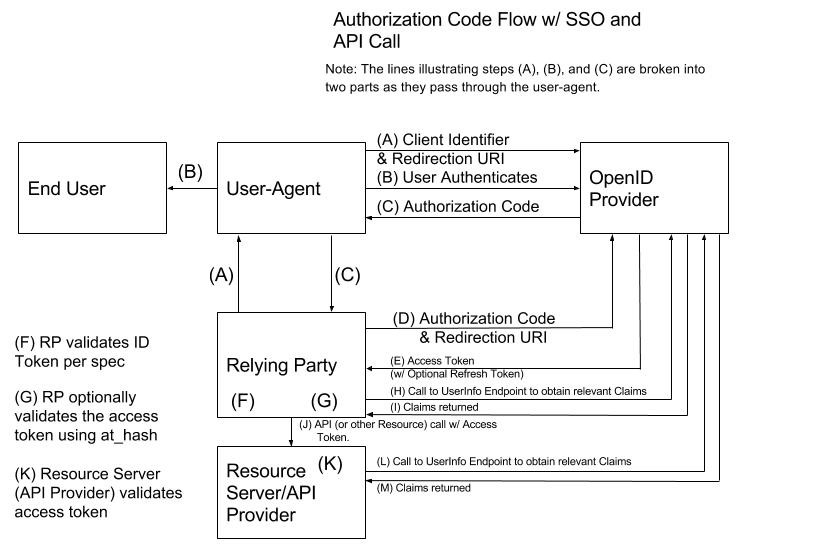
* Javni naspram poverljivih klijenata (Vi ste vlasnik klijentske aplikacije/kontrolišete aplikaciju u odnosu na nezavisnu aplikaciju, tj. aplikaciju treće strane)
* Tip Klijenata: veb aplikacija, desktop, mobilna, spa aplikacija.
* Identitet korisnika ili aplikacije (ili oboje)
* Želja za centralizovanim upravljackim tokom prijavljivanja

## Tok Autorizacionim kodom (Authorization Code Flow)

Sada će biti objašnjeno kako funkcioniše tok autorizacionim kodom i kako se korišćenjem ovog toka korisnik autentifikuje preko klijentske aplikacije (eng. Relayng Party) i dobija token identiteta i prava pristupa preko servera identiteta (IdP) [9].

Korišćenjem ovog toka se trebaju obezbediti sledeće stvari:

* Autentikacija krajnjeg korisnika
* Autentikacija klijentske aplikacije (Relayng Party RP u OIDC terminologiji), opciono
* Klijentska aplikacija ne vidi kredencijale (šifru) krajnjeg korisnika
* Pretraživač ne vidi tokene koji se prenose izmedju Servera identiteta (IdP) i Klijentske apikacije



Slika 7. Tok Autorizacionim kodom

Na početku korisnik inicira proces prijave tako što u pretraživaču kuca adresu svoje aplikacije (Relayng Party). Pretraživač tada upućuje zahtev klijentskoj aplikaciji.

### Zahtev za autentikaciju

Kada klijentska aplikacija (RP) primi zahtev, proverava da li je korisnik prošao proces autentikacije (da li ima izdat token identiteta). Ukoliko nije, tada Klijentska aplikacija redirektuje korisnika na Server Identiteta (OpenID Provider OP) koji korisniku vraća stranu za prijavljivanje, gde korisnik treba uneti svoje kredencijale (korisničko ime i lozinku).

GET/oidc/authorize?  
response\_type=code  
&scope=openid%20profile%20email  
&client\_id=s6BhdRkqt3  
&state=af0ifjsldkj  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1  
&nonce=fjsjsellesldska...  
Host: server.example.com

Klijentska aplikacija šalje zahtev za autorizaciju Serveru Identiteta tako što poziva autorizacionu metodu sa sledećim parametrima:

* **Response\_type = code** – Ovim parametrom se specificira da se radi o toku autorizacionim kodom i nakon što korisnik zavši proces autentikacije, server Identiteta vraća klijentskoj aplikaciji kod sa kojim se kasnije u kombinaciji sa klijentskom identifikacijom i šifrom dobijaju potrebni tokeni.
* **Scope = openid profile email** – Scope parametrom se zapravo specificiraju specifična prava, odnosno opseg pristupa sa kojima se izdaju tokeni i u sklopu OIDC-a se mogu zahtevati samo specifične informacije koje će biti izdate u tokenima. Svaki scope se navodi sa razmakom u zahtevu, i na primer openid nam kaže da želimo izdavanje identity tokena samo sa korisnikovim identifikacionim kodom. Profile, email scopom želimo da u tokenu identiteta dobijemo više informacija o korisniku poput emaila, slike, imena itd. Važno je napomenuti da se u Serveru identiteta mogu dodavati specifični opsezi i prava za same korisničke informacije unutar id tokena kao i za prava pristupa specifičnim resursima unutar access tokena.
* **Client\_Id** – Identifikator klijenta (privatni ili javni) koji je registrovan u serveru identiteta i preko koje zapravo Klijentska aplikacija vrši autentikaciju i autorizaciju. Prilikom registracije klijenata u serveru identiteta se zapravo konfigurišu svi ovi parametri i opsezi pristupa, a klijentska aplikacija mora da pruži iste parametre prilikom autorizacije.
* **State** –Ovaj parametar uglavnom služi da spreči takozvane CSRF (eng. Cross-site request forgery) napade i da se dodatno validira zahtev ka serveru identiteta. Klijentska aplikacija šalje random state parametar, nakon autentikacije server identiteta, klijentskoj aplikaciji vraća isti state kako bi utvrdila da je zapravo to odgovor na njen zahtev.
* **Redirect\_uri** – Url adresa klijentske aplikacije na kojoj server identiteta šalje kod nakon autentikacije.
* **Nonce** – Dodatna zaštita od napada. Klijent šalje nonce parametar, koji server identiteta ubacuje kao tvrdnju unutar tokena identiteta. Klijent kada primi token proverava da li se nonce vrednost slaže sa poslatom. Uvedeno je radi sprečavanja takozvanih replay napada.

Ukoliko autorizaciona metoda unutar servera identiteta utvrdi da korisnik nije prošao potrebnu autentikaciju, odnosno nije uneo korisničko ime ili lozinku (proverava se uglavnom informacija o sesiji, tj. da li je setovan kolačić identiteta), tada se prvo korisnik redirektuje na stranu za prijavljivanje.

### Uspešan odgovor nakon potvrde identiteta

Nakon uspešne autentikacije korisnika na strani prijave, Server Identiteta (OP) će redirektovati korisnika nazad na autorizacionu metodu, sa informacijama o sesiji (uglavnom u formi kolačića). Tada će metoda za autorizaciju vratiti HTTP redirekt (sa kodom autorizacije) klijentskoj aplikaciji.

HTTP/1.1 302 Found  
Location: <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsld

Kao rezultat , pretraživač šalje GET zahtev klijentskoj aplikaciji u sledećoj formi:

GET <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsldkj

### Zahtev za dobijanje tokena

U ovom trenutku, Klijentska aplikacija (RP) je primila kod autorizacije bez da je videla zapravo korisnikove kredencijale (korisničko ime i lozinku). Sada RP treba da dobavi tokene (id token, access token, refresh token) i ostale informacije iz Token metode unutar servera identiteta. Tako da sada RP salje zahtev Token metodi za pribavljanje potrebnih tokena.

POST /oidc/token HTTP/1.1  
Host: server.example.com  
Content-Type: application/x-www-form-urlencoded  
grant\_type=authorization\_code&  
code=SplxlOBeZQQYbYS6WxSbIA&  
redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb&  
client\_id=s6BhdRkqt3&  
client\_secret=blah\_blah\_blah\_1234

*Client\_secret* parametar je potreban ukoliko je RP zapravo registrovan kao privatni (eng. confidential) klijent.

Kao što vidimo, sada Klijentska aplikacija korišćenjem koda i klijenta zahteva izdavanje potrebnih tokena.

Zbog neophodne validacije osim koda, serveru identiteta se šalju i ostale klijentske informacije koje su registrovane kao deo klijenta (privatnog ili javnog) unutar servera identiteta (grant\_type, redirect\_uri, client\_id, client\_secret).

### Odgovor Token Metode

Nakon validacije koda i klijentskih parametra, server identiteta (OP) vraća klijentskoj aplikaciji (RP) ptrebne tokene u sledećem obliku:

HTTP/1.1 200 OK  
Content-Type: application/json  
Cache-Control: no-štore  
Pragma: no-cache  
{  
“access\_token”: “SlAV32hkKG”,  
“token\_type”: “Bearer”,  
“refresh\_token”: “8xLOxBtZp8”,  
“expires\_in”: 3600,  
“id\_token”: “eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc  
yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5  
NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ  
fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz  
AKfQ.ggW8hZ1EuVLuxNuuIJKX\_V8a\_OMXzR0EHR9R6jgdqrOOF4daGU96Sr\_P6q  
Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ  
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4\_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd  
QyHE5lcMiKPXfEIQILVq0pc\_E2DzL7emopWoaoZTF\_m0\_N0YzFC6g6EJbOEoRoS  
K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3\_osjzw2PAithfubEEBLuVVk4  
XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg”  
}

### Validacija Tokena

Kada klijentska aplikacija (RP) primi token identiteta (id\_token) on mora biti potvrđen, tj. dodatno validiran. Iako se opciono savetuje se da se ovaj deo ne preskače zato što ugrožava bezbednost.

U OIDC specifikaciji *Id\_token* se koristi radi provere identiteta na strani RP-a, dok se *access\_token* koristi za opcionalno preuzimanje dodatnih informacija o korisniku korišćenjem UserInfo metode na strani servera identiteta (OP), kao i za upućivanje poziva serverima resursa.

UserInfo metoda se koristi za dobavljanje dodatnih informacija o korisniku, koja nisu deo tokena identiteta.

GET /connect/userinfo

Authorization: Bearer <access\_token>

HTTP/1.1 200 OK

Content-Type: application/json

{

"sub": "248289761001",

"name": "Bob Smith",

"family\_name": "Smith",

"role": [

"user"

]

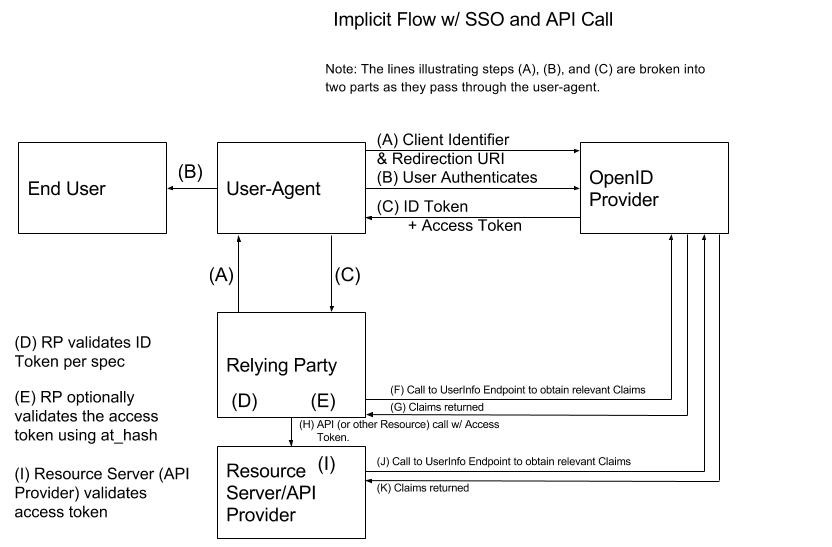
}

## Implicitni Tok (Implicit Flow)

Implicitni tok je sličan toku autorizacionim kodom samo je pojednostavljen i prvobitno se preporučivao native i javaskript aplikacijama, gde je token odmah vraćen aplikaciji, bez dodatnog koraka za razmenu autorizacionog koda.

S obzirom da native i javascript aplikacije ne mogu sigurno čuvati klijentsku tajnu (client secret), ovaj tok uglavno koristi javne klijente sa strane servera identiteta.

S obzirom da su koraci jako slični toku autorizacionom kodu, fokus će biti na objašnjenjuključnih razlika.



Slika 8. Implicitni Tok

### Zahtev za autentikaciju

GET /oidc/authorize?  
response\_type=id\_token%20token  
&client\_id=s6BhdRkqt3  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb  
&scope=openid%20profile  
&state=af0ifjsldkj  
&nonce=n-0S6\_WzA2Mj HTTP/1.1   
Host: server.example.com

Klijentska strana (eng. Relayng Party) upućuje *get* zahtev preko pretraživača, pozvajući */authorize* metodu na strani servera identiteta (OP).

Razlika u ovom zahtevu u odnosu na tok autorizacionim kodom jeste u parametru *response\_type* gde se ovoga puta navodi *id\_token* i opcionalno token. Ako se kao parametar unese *id\_token* to govori serveru identiteta da klijentu vrati token identiteta kao odgovor direktno kroz pretrazivač i da se radi o implicitnom toku. Dodatni opcioni parametar token govori serveru identiteta da klijentska aplikacija takođe želi i token za prava pristupa (*access\_token*), koji će takođe biti vraćen kroz pretrazivač kao parametar.

### Uspešan odgovor nakon potvrde identiteta

HTTP/1.1 302 Found  
Location: [https://client.example.org/cb#](https://client.example.org/cb)  
access\_token=SlAV32hkKG  
&token\_type=bearer  
&id\_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso  
&expires\_in=3600  
&state=af0ifjsldkj

Server Identiteta odgovara na zahtev u ovom formatu. Kao što se može videti ovoga puta se ne vraća kod vec se direktno prosleđuju tokeni.

Zatim pretraživač šalje redirektovani zahtev klijentskoj aplikaciji (RP).

POST <https://client.example.org/cb>

access\_token=SlAV32hkKG  
 &token\_type=bearer  
 &id\_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso  
 &expires\_in=3600  
 &state=af0ifjsldkj

Klijentska aplikacija (RP) presreće ovaj zahtev i vadi tokene ( i druge opcione parametre).

Važno je napomenuti da se ne preporučuje upotreba implicitnog toka (a neki serveri taj tok u potpunosti zabranjuju) zbog sigurnosnih rizika vraćanja tokena u HTTP preusmeravanju bez ikakve potvrde da ga je klijentska aplikacija primila.

Javni klijenti poput native i javascript aplikacija, sada bi trebalo da koriste Tok Autorizacionim kodom sa dokazom ključa za razmenu kodova (eng. Authorization Code Flow with Proof Key for Code Excange - PKCE), koji je napravljen da prevaziđe ove sigurnosne probleme za javne klijente servera identiteta.

## Hibridni Tok

Hibridni tok je kombinacija implicitnog toka i toka autorizacionim kodom. Svi koraci su isti s tim što se token identiteta vraća implicitnim tokom kroz pretraživač, a token prava pristupa preko autorizacionog koda [10].

Aplikacije koje su u stanju da bezbedno čuvaju klijentske tajne (MVC aplikacije), mogu imati koristi od upotreba hibridnog toka, koji omogućava klijentskoj aplikaciji (RP) trenutni pristup tokenu identiteta, a istovremeno obezbeđuje sigurno preuzimanje tokena za prava pristupa (access\_token, refresh\_token, takodje i novi id\_token) razmenom koda i klijentske tajne.

Ovo može biti korisno u situacijama kada klijentska aplikacija mora odmah pristupiti informacijama o korisniku, ali mora izvršiti određenu obradu i validaciju sa serverom identiteta pre nego što pristupi zaštićenim resursima.

S obzirom da smo objasnili korake u implicitnom toku i toku autorizacionim kodom, sada ćemo samo ukratko proći korake.

* Klijentska aplikacija salje zahtev za autentikaciju i autorizaciju redirekcijom na */authorize* metodu serevera identiteta. Ovoga puta parametar *response\_type* se postavlja na *id\_token code* što govori da se radi o hibridnom toku.
* Nakon autentikacije server identiteta vraća klijentskoj aplikaciji kod kao i token identiteta
* Klijentska aplikacija odmah može da iskoristi token identiteta da bi dobila podatke o klijentu
* Klijentska aplikacija zahteva token za pravo pristupa resursima (eng. access\_token) tako što serveru identiteta šalje kod i klijentsku tajnu radi dodatne validacije (poziva /token metodu na strani servera identiteta).
* Nakon što server identiteta validira zahtev preko predhodno poslatog koda i klijentske tajne, vraća klijentskoj aplikaciji potrebne tokene.
* Klijentska aplikacija pomoću tokena sada može pristupati zaštićenim resursima, kao i zahtevanjem dodatnih informacija o klijentu, pozivajući */userInfo* metodu na strani servera identiteta.

Takođe postoje nekoliko kombinacija hibridnog toka, definisana kroz *response\_type* parametar prilikom autentikacije i autorizacije.

* **response\_type = code** token – Server identiteta vraća kao http redirekt odgovor kod i token prava pristupa (access\_token)
* **response\_type = code id\_token** – Server identiteta vraća kao http redirekt odgovor, kod i token identiteta (id\_token)
* **response\_type = code id\_token token** – Server identiteta vraća kao http redirekt odgovor, kod token identiteta (id\_token) kao i token prava pristupa (access\_token)

## Tok Autorizacionim kodom sa dokazom kljuca za razmenu kodova (Authorization Code Flow with Proof Key for Code Excange - PKCE)

PKCE OIDC tok je specijalno dizajniran za autentikaciju native i korisnika mobilnih aplikacija.

Dizajniran je kao nadogradnja toka autorizacionim kodom, za aplikacije koje koriste javne klijente i koje ne mogu sigurno čuvati klijentsku tajnu (client\_secret). Ovaj tok se smatra najboljom praksom kada se koriste SPA (eng. Single Page Applications) ili aplikacije za mobilne uređaje. Ključna razlika između standardnog toka autorizacionim kodom je ta što aplikacije ne moraju da prosleđuju klijentsku tajnu (client\_secret).

PKCE smanjuje bezbednosne rizike za native aplikacije, jer klijentske tajne nisu potrebne u izvornom kodu, što ograničava izloženost obrnutom inženjeringu [11].

### Zahtev za autentikaciju

GET /oidc/authorize?response\_type=code  
&scope=openid%20profile%20email  
&client\_id=s6BhdRkqt3  
&state=af0ifjsldkj  
&code\_challange= PCBxoCJMdDloUVl1ctjvA6VNbY6fTg1P7PNhymbydM  
$ code\_challenge\_method=S256  
&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1  
Host: server.example.com

Kao što možemo videti zahtev je vrlo sličan toku autorizacionim kodom tako da cemo objasniti samo parametre specifične za PKCE tok.

* **code\_challange** – klijentska aplikacija na svojoj strani generiše random **code\_verifier,** zatim se taj niz karaktera hešira i ta vrednost se šalje kao code\_challange
* **code\_challange\_method** –korišćena hash metoda za heširanje code\_verifier (uglavnom se koristi SHA256)

### Uspešan odgovor nakon potvrde identiteta

Odgovor je isti kao kod toka autorizacionim kodom. Metoda za autorizaciju vratiti HTTP redirekt sa kodom autorizacije klijentskoj aplikaciji.

HTTP/1.1 302 Found  
Location: <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsld

Kao rezultat , pretraživač šalje GET zahtev klijentskoj aplikaciji u sledećoj formi:

GET <https://client.example.org/cb?>  
code=SplxlOBeZQQYbYS6WxSbIA  
&state=af0ifjsldkj

### Zahtev za dobijanje tokena

POST /oidc/token HTTP/1.1  
Host: server.example.com  
Content-Type: application/x-www-form-urlencoded  
grant\_type=authorization\_code&  
code=SplxlOBeZQQYbYS6WxSbIA&  
redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb&  
client\_id=s6BhdRkqt3&  
code\_verifier=shekrslsnj4kfdksfksk

Kao što možemo videti, sada u zahtevu za dobijanje tokena umešto klijentske tajne (client\_secret), klijentska aplikacija šalje izvorni string (niz karaktera) code\_verifier.

Sada server identiteta na osnovu dobijenih *code\_challenge* i *code\_challenge\_method* parametra iz predhodnog zahteva može primeniti heširanje istom metodom nad *code\_verifier* parametru i uporediti da li se nastala heširana vrednost slaže sa *code\_challange* parametrom, i time potvrdi klijentski zahtev i izda tokene.

Izdavanje tokena i dalji koraci su isti kao kod toka autorizacionim kodom (Slika 9).



Slika 9. PKCE Tok

### Validacija tokena identiteta

Po OIDC specifikaciji klijenti moraju validirati token identiteta na sledeći način [12]:

* Ako je token identiteta enkriptovan, dekriptovati ga pomoću ključeva i algoritama koje je klijent naveo tokom registracije, odnosno koje je server identiteta (OP) trebalo da koristi za enkriptovanje tokena identiteta. Ako je o enkripciji dogovoreno sa OP stranom u vreme registracije i token identiteta nije šifrovan, klijentska strana (eng. Relayng Party) treba da ga odbije.
* Identifikator izdavaoca tokena identiteta (OP) mora se tačno podudarati sa vrednošću **iss** tvrdnje (eng. claim) unutar tokena identiteta.
* Klijent mora da validira **aud** (eng. audience) tvrdnju unutar tokena identiteta koja treba da sadrži identifikator klijenta kome je token namenjen (**client\_id**). Aud tvrdnja moze da sadrži niz klijenata kome je token namenjen. Token identiteta mora biti odbijen ukoliko ne sadrži identifikator klijentske aplikacije ili ako sadrži identifikatore klijenata koje nisu od poverenja.
* Ako token identiteta sadrži više klijenata kome je namenjen u **aud** tvrdnji, klijentska aplikacija bi trebalo da verifikuje da je **azp** (predstavlja trenutno autorizovanu stranu) tvrdnja takođe prisutna u tokenu identiteta.
* Ako je **azp** (eng. authorized party) tvrdnja prisutna, onda klijent treba da verifikuje da je njena vrednost zapravo identifikator klijenta (**client\_id**).
* Ako je token identiteta primljen preko direktne komunikacije između klijenta i servera identiteta (što i jeste slučaj u OIDC tokovima), onda TSL serverska validacija može biti iskorišćena da validira izdavaoca tokena umesto provere potpisa tokena. Klijentska aplikacija mora validirati potpise svih drugih tokena identiteta prema **JWS** (eng. JSON Web Signature) specifikaciji, koristeći algoritme specificirane u zaglavlju (eng. header) tokena identiteta odnosno **alg** tvrdnji. Klijent mora koristiti ključ koji je dobio od izdavača tokena (OP).
* Unutar zaglavlja tokena identiteta **alg** tvrdnja bi trebalo da bude **RS256** ili algoritam poslat od strane klijentske aplikacije preko **id\_token\_signed\_response\_alg** parametra prilikom registracije sa serverom identiteta.
* Trenutno vreme mora biti pre vremena isteka tokena identiteta definisano u **exp** (eng. expiration) tvrdnji.
* Tvrdnja iat (eng. issued at) u tokenu identiteta predstavlja vreme izdavanja tokena i može se koristiti za odbijanje tokena koji su izdati predaleko od trenutnog vremena. Prihvatljivi opseg je specifičan za klijenta.
* Ako je **nonce** vrednost poslata prilikom zahteva za autentikaciju, onda **nonce** tvrdnja mora biti prisutna u tokenu identiteta, i proverava se da li je ta vrednost ista kao vrednost koju je klijent poslao prilikom autentikacije, uvedeno je radi sprečavanja takozvanih **replay** napada.

# Single Sign Out

Kao što je već rečeno, Single Sign On (SSO) je moguće postići oslanjajući se na neki centralni server identiteta koji nakon uspešne autentikacije pamti stanje sesije u vidu kolačića, i tokom sledeće prijave korisnika preko druge aplikacije, server identiteta proverava da li postoji sesija, odnosno kolačić, koji potvrđuje identitet korisnika. Samim tim govori serveru identiteta da je korisnik već izvršio proces autentikacije i dodeljuje mu potrebne tokene.

## Šta je Single Sing Out?

Single Sign Out nije tako jednostavno implementirati. Single Sign Out zahteva da server identiteta (server tokena) sadrži i listu aplikacija kojima je izdao tokene i koji protokol će koristiti za slanje zahtveva za odjavu. Korisnik se možda prijavio u različite aplikacije koristeći različite protokole (WsFed, SAML, OpenIDConnect). Svi ovi protokoli definišu neke pristupe za Single Sign Out, ali većina njih ne rezultira odjavom iz svih aplikacija. Deo problema jeste što se navedeni protokoli ne mogu osloniti na prisustvo pretraživača koji sluša takve poruke nakon inicijalne prijave korisnika (na primer mobilne aplikacije koriste pretraživac samo za postizanje Singl Sign On-a).

Moguće rešenje je prosleđivanje svih zahteva preko nekog dodatnog proxy servera koji ima status prijave korisnika, ali to nije bas skalabilno rešenje.

Za situacije gde nije moguće postići odjavu sa svih aplikacija (pre svega kod aplikacija koje ne koriste sve vreme pretraživač u svom radu), najbolja opcija je ne stvoriti utisak da se korisnik odjavljuje iz svih svojih aplikacija odjednom [13].

## OIDC Sign Out

Prema OIDC specifikaciji postoje tri metode za postizanje Single Sign Out-a, odnosno za okončanje korisničke sesije [14].

* **Session Management**
* **Front-channel Logout**
* **Back-channel Logout**

U nastavku će biti detaljnije objašnjeno kako funkcionišu ove tri metode kao i kako klijentska aplikacija RP (eng. Relaying Party) inicira zahtev za sign out prema serveru identiteta.

## Klijentska aplikacija inicira Logout

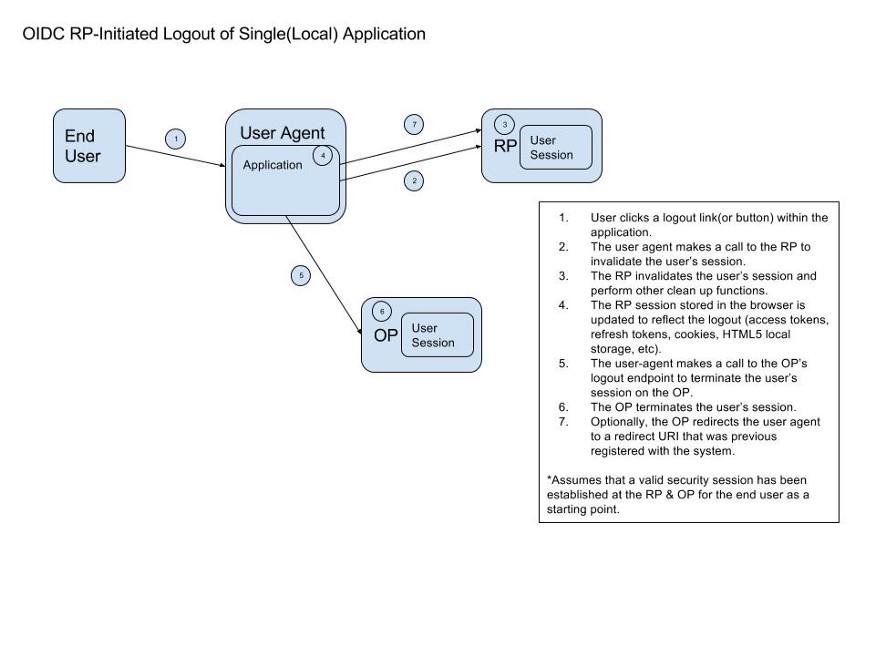
U ovom slučaju klientska aplikacija je odgovorna za brisanje internih tokena u svom skladištu. Nakon što je to završeno, klijentska strana redirektuje pretraživač na logout metodu unutar servera identiteta (OP). Logout metoda je definisana po OIDC Session Management Specifikaciji i može biti pribavljenja preko OP Discovery Endpoints (Po OIDC specifikaciji unutar Discovery Endpoints se nalazi spisak svih metoda koje pruža server identiteta poput authorize, token, logout, userInfo metoda).

Logout zahtev ka serveru identiteta izgleda ovako:

GET <https://idp.levvel.io/oidc/logout?id_token_hint=previvously_issued_jwt_id_token&>  
post\_logout\_redirect\_uri=https://app1.levvel.io/oidc/logged\_out&  
state=some\_random\_string

Parametri koji se šalju prilikom odjavljivanja, po specifikaciji su sledeći [15]:

* **id\_token\_hint** – Predhodno izdat token identiteta, nije obavezan, ali se preporučuje kao nagoveštaj o trenutnoj autentikacionoj sesiji sa klijentom.
* **post\_logout\_redirect\_uri** – Povratna url adresa klijentske aplikacije na koju će pretraživač biti redirektovan nakon što odjavljivanje bude izvršeno. Ova vrednost mora predhodno biti registrovana u podešavanjima klijenta na strani servera identiteta (OP). Takođe je opcionalni parameter.
* **State** – Vrednost koju klijentska aplikacija koristi za održavanje stanja između zahteva za odjavu i povratnog poziva navedenog kroz *post\_logout\_redirect\_uri* u cilju sprečavanja CSRF napada. Takođe je opcionalni parameter.



Slika 10. Klijentska aplikacija inicira Logout

Kada jedna od klijentskih aplikacija izda zahtev za odjavljivanje, korišćenjem jedne od tri metoda koje definise OIDC, možemo implementirati odjavljivanje ostalih klijentskih aplikacija.

### Session Management

Upravljanje sesijama je metoda za kontinuirano praćenje statusa prijavljivanja krajnjeg korisnika kod servera identiteta, tako da klijentska aplikacija može da odjavi krajnjeg korisnika koji se već odjavio sa servera identiteta (preko neke druge pouzdane klijentske aplikacije).

Jednostavno rečeno, to znači da klijentska aplikacija treba neprestano da pita server identiteta da li je krajnji korisnik jos uvek prijavljen. Ako server identiteta odgovori da korisnik nije prijavljen, onda bi klijentska aplikacija trebala da obriše sve lokalne podatke sesije i tokene povezane sa tim korisnikom.

Postoje dva načina na koja klijentska aplikacija može da proveri status prijavljivanja krajnjeg korisnika. Jedna od metoda je ponavljanje zahteva za potvrdu identiteta, ali to bi prouzrokovalo previše mrežnog saobraćaja i poremetilo, tj. odgodilo uslugu servera identiteta. Kako bi bilo poželjnije proveriti status bez uzrokovanja previše mrežnog saobraćaja, koristi se poboljšana tehnika koja uključuje iframe-ove. Jednom kada se sesija uspostavi, iframe klijentske aplikacije ce idalje ispitivati iframe na strani servera identiteta za status prijavljivanja krajnjeg korisnika.

### Front-channel Logout

Komunikacija preko Front-channel-a se izvršava preko pretraživača. Kada Klijentska aplikacija inicira odjavljivanje, server identiteta preko stranice za logout izrađuje iframe-ove koji se onose na adrese ostalih klijenata koji su registrovani u serveru identiteta. Na ovaj način, zahtev za odjavu za svaku klijentsku aplikaciju se javlja paralelno (umesto serijske sesije preusmeravanja da bi se izvršila odjava na svakoj klijentskoj aplikaciji).

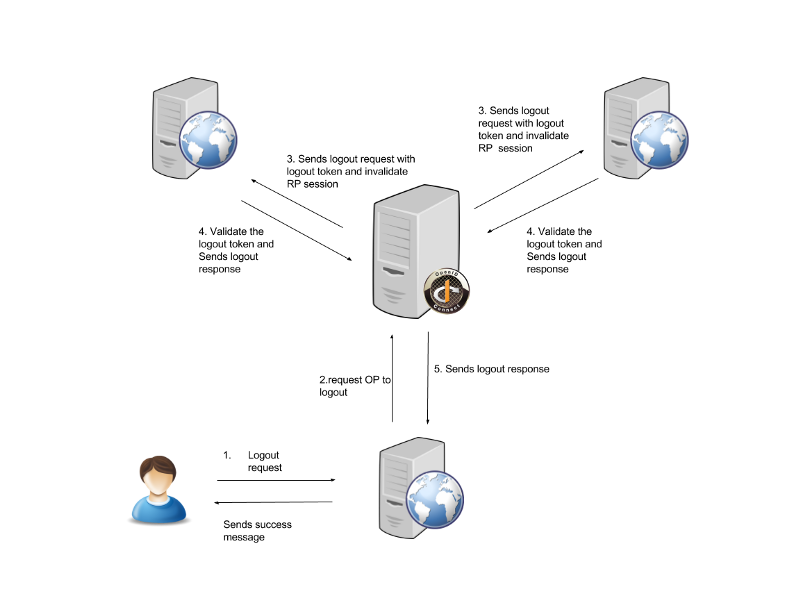
### Back-channel Logout

Ovaj mehanizam odjave je mnogo manje “lomljiv“ od ostalih metoda koji koriste front-channel komunikaciju, jer ne zavisi od aktivnosti pretraživača krajnjeg korisnika, već se komunikacija vrši tako što server identiteta prilikom odjave direktno poziva adrese svih aktivnih klijenata nakon okončanja sesije. Ovo se može činiti najboljom opcijom za OIDC odjavom. Međutim, nije prikladan za svaki scenario jer nameće neke nove zahteve sistemu [16].

Kada Klijentska aplikacija ili server identiteta pokrene akciju odjave, server identiteta pronađe sve klijentske aplikacije koje na serveru identiteta dele sesiju ištog korisnika. Server identiteta tada generiše poseban JWT token, koji je jako sličan tokenu identiteta, nazvan token za odjavu, koji sadrži relevantne tvrdnje za svaku klijentsku aplikaciju i šalje zahteve sa tim tokenima na metodama registrovanim u klijentskim aplikacijama kao krajnje tačke odjave (klijentske adrese za odjavu back-channel metodom moraju da budu registrovane i na strani servera identiteta).

POST /backchannel\_logout HTTP/1.1  
Host: rp.example.org  
Content-Type: application/x-www-form-urlencoded  
logout\_token=eyJhbGci ... .eyJpc3Mi ... .T3BlbklE ...

Po prijemu zahteva klijentska strana validira token za odjavu i ponistava lokalnu korisničku sesiju.



Slika 11. Back-channel Logout

**Prednosti Back-Channel-a**

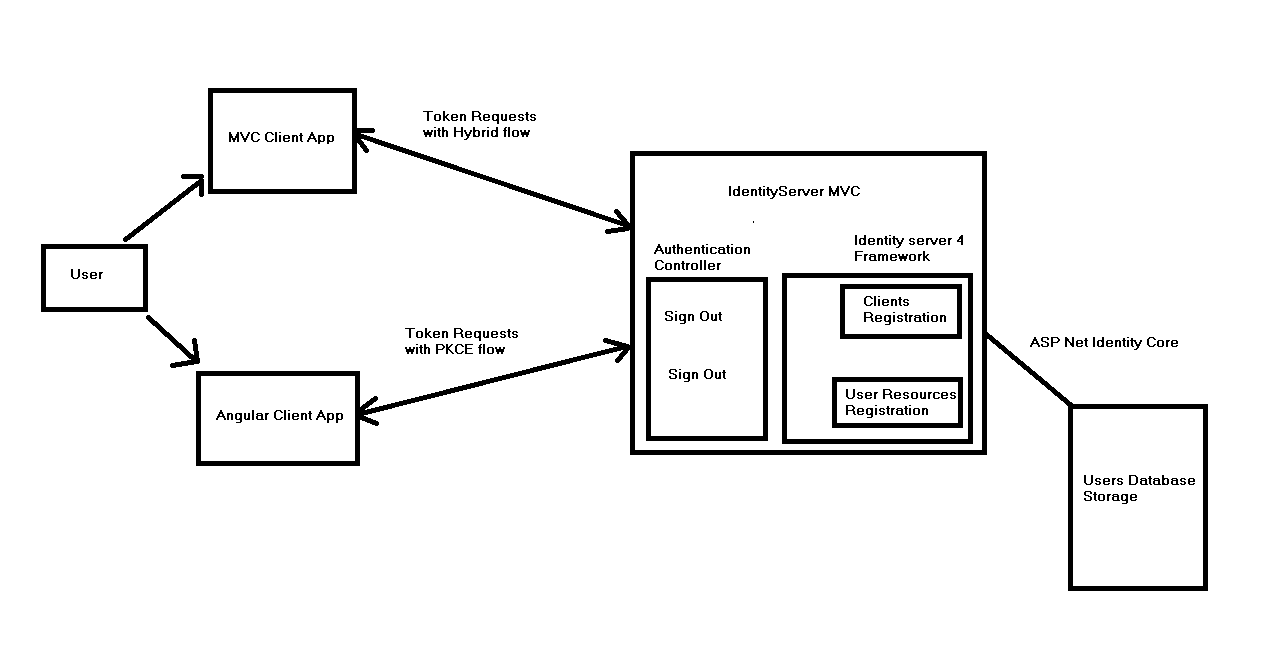
Ovaj mehanizam odjave je dosta pouzdaniji od ostalih metoda, jer uspeh zahteva za odjavu ne zavisi od aktivnosti sesije korisničkog pretraživača.

**Posledice Back-Channel-a**

Sistemu se nameću novi zahtevi, jer se moraju uspostaviti dodatne mrežne veze između servera za back-channel komunikaciju. Podaci o stanju sesije koji se čuvaju u kolačićima i u HTML5 skladištu, nisu dostupni preko back-channel komunikacije.

U Front-channel metodama, odjava je zapravo jednostavno brisanje kolačića i stanja u HTML5 memoriji. Složenije je u odjavi preko Back-channel komunikacije, jer klijentske aplikacije moraju da primene specifične metode za prekid korisničkih sesija po prijemu zahteva za odjavom.

# Arhitektura Sistema

****

Slika 1*2*. *Arhitektura sistema*

Cilj rada jeste pokazati i objasniti koncept Single Sign On-a, odnosno mogućnost korisnika da autentifikovanjem preko jedne aplikacije dobije pristup i drugim aplikacijama koje su deo sistema, bez potrebe da se ponovo autentifikuje, tj. unosi svoje kredencijale.

Sistem se sastoji od dve klijentske veb aplikacije (MVC Client, Angular Client), kao i Servera identiteta preko koje će biti omogućeno deljenje informacija identiteta korisnika (Identity Server).

**Tehnologije**

U daljem tekstu će biti ukratko navedene tehnologije pomoću kojih je implementiran 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 pribavljanje tokena identiteta koristiće se PKCE tok sa javnim klijentom koji je registrovan u serveru identiteta.
3. **Identity Server** – Serverska MVC aplikacija implementirana korišćenjem Microsoft .Net Core 5 Framework-a. Da bi se omogućilo serveru funkcionalnost servera veze između klijentskih aplikacija, potrebno je implementirati sledeće module:

* **OIDC i OAuth 2.0**- za implementaciju ovih protokola koristišćen je Identity Server 4 Framework ( Registracija klijanata i Korisničkih Resursa je deo ovog frameworka).
* **Sign In, Sign On** – za implementaciju je koristićen Microsoft .Net Core 5 Framework. Za ovu svrhu registrovan je Authentication kontroler sa SignIn, Sign Out metodama.
* **Autentikacija korisnika**– za autentifikaciju koristišćena je Asp Net Core Identity biblioteka, a za čuvanje korisničkih podataka iskorišćena je baza 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 aplikacije. Identity Server 4 je zvanično sertifikovan od strane OpenID Foundation-a, deo je i .Net Foundation-a i deluje u skladu sa njihovim kodeksom ponašanja i licensiran 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 različite tipove aplikacija [17].

### Standardni scenariji veb aplikacija

Tipično, aplikacije bi trebalo da podrže neke ili sve od sledećih scenarija (slika 13) [18]:

* 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 13. Tipovi aplikacija i scenarija

Za potrebu autentikacije je korišćena centralizovana .Net Core 5 Web aplikacija pod nazivom IdentityServer (slika 15), a preko nuget paketa je ubačen Identity Server 4 framework na sledeci način (slika 14):



Slika 14. Identity Server 4 nuget



Slika 15. IdentityServer .NET 5.0 Solution

U ovom radu je dat fokus na autentikaciji korisnika i postizanju Single Sign On-a korišćenjem kolačića i tokena identiteta (id\_token). Token za prava pristupa (access\_token) je korišćen isključivo za dobavljanje dodatnih korisničkih informacija.

Da bi se implementirao Single Sign On korišćenjem Identity Server 4 framework-a, potrebno je objasniti sledeće oblasti [19]:

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 16).

Koristi se takođe u automatizaciji sa klijentskom aplikacijom u pronalaženju ključnih endpointa po OIDC specifikaciji (authorize/token/userinfo).



Slika *16*. *Identity Server 4 / .well-known/openid-configuration*

### Identity Resource

Zadatak 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-a, email-a, adrese i tako dalje.
* **API resources** – predstavljaju neku funkcionalnost kojoj klijenti žele da pristupe. U glavnom su to Restful API aplikacije.

S obzirom da je fokus u ovom radu postizanje Single Sign On-a, uglavnom će biti fokus na resursima 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 može biti zahtevana od strane klijentske aplikacije, i koje se nalaze u tokenu identiteta.

Kada su objašnjavani autorizacioni tokovi u sklopu OIDC protokola, rečeno je da se **scope** parametar koristi za definisanje nekog opsega prava pristupa (što se tiče tokena identiteta, definiše koje će se tvrdnje o korisniku vratiti klijentskoj aplikaciji).

Takođe je rečeno 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 (primer 1) ali može se definisati i neka imenovana grupa koja objedinjuje više tvrdnji samog korisnika (primer 2).

namespace IdentityServer4.Models

{

public static class IdentityResources

{

public class OpenId : IdentityResource

{

public OpenId();

}

public class Profile : IdentityResource

{

public Profile();

}

public class Email : IdentityResource

{

public Email();

}

public class Phone : IdentityResource

{

public Phone();

}

public class Address : IdentityResource

{

public Address();

}

}

}

Primer *1*. *Identity Server 4 standardne tvrdnje*

new IdentityResource (

name: "custom.profile",

userClaims: new[] { "sub", "email", "username" },

displayName: "Your user identifier")

Primer *2*. *Identity Server 4 definisanje sopstvene grupe tvrdnji*

Na kraju treba definisati niz resursa odnosno grupe tvrdnji, sa kojom Server Identiteta raspolaže (primer 3).

public static IEnumerable<IdentityResource> Ids =>

new IdentityResource[]

{

new IdentityResources.OpenId(),

new IdentityResources.Profile(),

new IdentityResources.Email(),

new IdentityResource(

name: "custom.profile",

userClaims: new[] { "sub", "email", "username" },

displayName: "Your user identifier")

};

Primer *3*. *Identity Server 4 definisanje raspolozivih resursa unutar Servera Identiteta*

### Klijenti

U sklopu ovog rada za demonstraciju Single Sign On-a će se 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 moraju se registrovati klijentske aplikacije (eng. Relayng Party), i dodeliti im posebna prava pristupa (primer 4, primer 5).

new Client

{

ClientId = "ClientMVC",

ClientName = "Client MVC",

AllowedCorsOrigins = {"https://localhost:44344"},

ClientSecrets = { new Secret("MVCSecret".Sha256()) },

AllowedGrantTypes = GrantTypes.Hybrid,

RedirectUris = { "https://localhost:44344/signin-oidc" },

PostLogoutRedirectUris = { "https://localhost:44344/signout-callback-oidc" },

IdentityTokenLifetime = 1200,

AlwaysIncludeUserClaimsInIdToken = true,

AllowedScopes =

{

"openid",

"custom.profile"

},

RequireClientSecret = true,

AllowOfflineAccess = true,

RequirePkce = false

},

Primer *4*. *Identity Server 4 definisanje MVC klijenta*

### MVC Klijent

Ključni parametri definisani unutar MVC klijenta su dati u nastavku (primer 4):

* **ClientId –** Jedinstveni identifikator klijenta
* **AllowedGrantTypes –** Dozvoljenji tokovi autorizacije (Za MVC klijenta će se koristiti Hibridni tok, takođe treba napomenuti da klijent može da podrži više tokova)
* **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-a)
* **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 mogućnost da se više klijentskih aplikacija mogu autentikovati preko istog klijenata registrovanog u serveru identiteta. Zato ovaj parametar očekuje listu povratnih adresa.
* **PostLogoutRedirectUris –** Slično kao RedirectUris, samo se ovoga puta navodi adresa klijentske aplikacije nakon 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 postavljen ovaj parametar, klijentska aplikacija će dobiti i token osveženja (eng. refresh\_token), sa kojim može ponovo zahtevati tokene (access\_token, id\_token) bez potrebe da se ponovo autentifikuje.
* **AllowedScopes –** Predstavlja listu dozvoljenih opsega pristupa resursima. Ovde Identity server definiše pravo klijenta da pristupa i dobija određene resurse. U konkretnom slučaju, dozvoljeni korisnički resursi, koje klijentska aplikacija može da zahteva su **openId** i **custom.profile** (primer 1 i primer 2). To znači da će token identiteta izdat klijentskoj aplikaciji sadržati sledece tvrdnje: **sub** (tvrdnja u sklopu openId resursa korisnika), **email**, **username** (tvrdnje u sklopu samostalnog definisanog custom.profile resursa korisnika).

### Angular Klijent

new Client

{

ClientId = "ClientAngular",

ClientName = "Client Angular",

RedirectUris ={"http://localhost:4200"},

AllowedCorsOrigins = {"http://localhost:4200"},

AllowedGrantTypes = GrantTypes.Code,

AllowedScopes =

{

"openid",

"email"

},

PostLogoutRedirectUris = {"http://localhost:4200"},

IdentityTokenLifetime = 1200,

AlwaysIncludeUserClaimsInIdToken = true,

RequireClientSecret = false,

AllowOfflineAccess = true,

RequirePkce = true

}

Primer *5.* *Identity Server 4 definisanje Angular klijenta*

Potrebno je definisati i jednog klijenta za SPA aplikacuju unutar servera identiteta odnosno Relayng Party (primer 5).

* **AllowedGrantTypes** – Angular klijent koristi tok Autorizacionim kodom, zato je ovaj parametar definisan kao **GrantTypes.Code**.
* **AllowedScopes** – Angular klijent može u tokenu ideniteta zahtevati samo **sub** i **email** tvrdnje (to je definisano pomoću **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 se definiše da se ne vrši provera klijentske tajne.
* **RequirePkce** – ovim parametrom se definiše 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 autentifikacije kao što su identifikator klijenata (client\_id), dozvoljeni opseg prava pristupa tokena (scope), itd., a zatim proverava da li je korisnik prethodno prošao kroz proces autentifikacije tako što proverava i validira sesioni kolačić. Ukoliko kolačić nije validan ili nije prisutan, onda *authorize* metoda navigira pretrazivač 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 pretraživaču 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 je već objašnjeno, 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 [20].

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

* **Sam korisnički interfejs za unos potrebnih kredencijala (uglavnom email i šifra)**
* **Endpoint unutar 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 korišćen je ugrađeni **Razzor Engine** unutar same .Net Core 5 Identity Server aplikacije (korisnički interfejs nije uslovljen .Net tehnologijom tako da se može koristiti i Angular, React ili jednostavan HTML zajedno sa Javascriptom).

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

Za potrebnu validaciju kredencijala, menadžmentom korisnika, kao i izdavanjem potrebnih kolačića sesije korišćena je **AspNet Identity Core** biblioteka u sklopu .Net Core 5 framework-a.

### Sign In korisnički interfejs

@model IdentityServer.AuthenticationViewModels.SignInViewModel

@{

ViewData["Title"] = "Sign In";

}

<div class="row" style="margin:auto;margin-top:5%;text-align:center">

<div class="col-md-4"></div>

<div class="col-md-4">

<section>

<form **asp-route-returnurl**="@ViewData["ReturnUrl"]" method="post">

<h4>Use a local account to sign in.</h4>

<hr />

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Email"></label>

<input **asp-for**="Email" class="form-control" />

<span **asp-validation-for**="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Password"></label>

<input **asp-for**="Password" class="form-control" />

<span **asp-validation-for**="Password" class="text-danger"></span>

</div>

<div class="form-group">

<button type="submit" class="btn btn-success">Sign in</button>

</div>

</form>

</section>

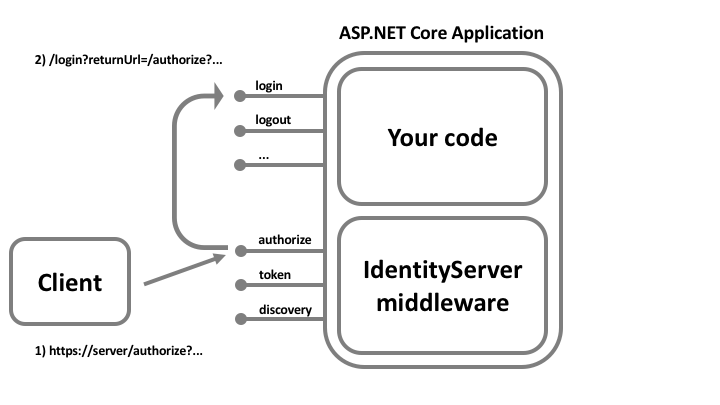
</div>

<div class="col-md-4"></div>

</div>

Primer *6* *IdentityServer Sign In korisnički interfejs*

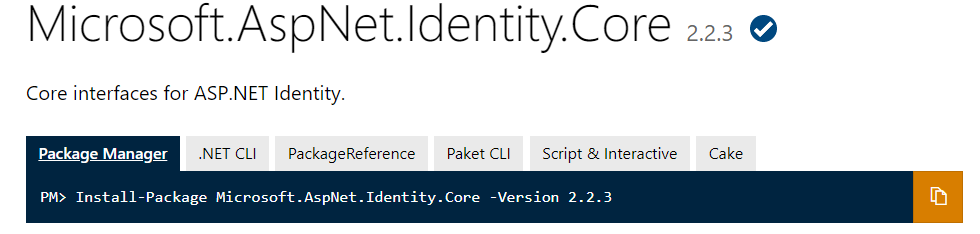
Kao što je napomenuto, za izradu Sing In korisničkog interfejsa korišćen je ugrađeni Razzor Engine u sklopu .Net Core 5 framework-a (primer 6) . Sastoji se od jednostavne post forme za unos korisničkih kredencijala kao i dodatnog **ReturnUrl (**slika17**)** parametara koji je zapravo url putanja do *authorize* metode, gde se korisnik redirektuje nakon formiranja kolačića sesije.



Slika 17. *IdentityServer 4 login*

### Validacija kredencijala i kreiranje kolačića sesije

Za validaciju kredencijala i generisanje kolačića sesije koristišćena je biblioteka (slika 18) za korisnički menadžment u sklopu .Net Core 5 frameworka, koja pruža potrebne olakšavajuće servise poput **UserManager**-a, **SignInManager**-a i metode poput **PasswordSignIn** za autentifikaciju korisnika i generisanje/brisanje kolačica.



Slika *18* *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 **IdentityServer** projekta. Za ovu svrhu napravljen je poseban kontroler po imenu *Authentication* kao i dve metode za **SignIn**. Jedna je **Get,** za slanje korisničkog interfejsa pretraživaču, i druga **Post,** za prijem korisničkih kredencijala (primer 7).

public class AuthenticationController : Controller

{

private readonly UserManager<AppUser> \_userManager;

private readonly SignInManager<AppUser> \_signInManager;

private readonly IIdentityServerInteractionService \_interaction;

public AuthenticationController(UserManager<AppUser> userManager,

SignInManager<AppUser> signInManager,

IIdentityServerInteractionService interaction)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_interaction = interaction;

}

[HttpGet]

[AllowAnonymous]

public IActionResult SignIn(string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> SignIn(SignInViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Email,

model.Password,

model.RememberMe);

if (result.Succeeded)

{

return Redirect(returnUrl);

}

}

return View(model);

}

}

Primer *7* *IdentityServer metoda za prijem korisničkih kredencijala*

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 autentifikacije (proverava prisutnost i validira kolačić sesije), ona će redirektovati korisnički pretrazivač na get akciju (u ovom 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 vratio kroz post formu nazad serverskoj metodi).

Za validaciju korisničkih kredencijala se koristi **Asp Net Identity Core** bibliotečka funkcija **PasswordSignInAsync** kroz **signInManager**, koja će 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 uspešno 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 šalje zahtev za dobijanje tokena tako što zajedno sa kodom šalje i **client\_id**, **client\_secret** (kada se koristi privatni klijent, u ovom slučaju MVC Client) ili **code\_verifier** (kada se koristi javni klijent sa PKCE tokom, u ovom 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 previše 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 4implementira posebnu metodu **connect/userinfo** koja vraća dodatne informacije o korisniku.

*UserInfo* metoda je zaštićena za anonimne korisnike i moguće 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 šalje pozivajući *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 će se naći unutar samih tokena na osnovu zahtevanog opsega tvrdnji definisanih unutar registrovanih tvrdnji 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 (primer 8).

public class ProfileService<TUser> : IProfileService where TUser : IdentityUser<int>, new()

{

private readonly UserManager<TUser> \_userManager;

public ProfileService(UserManager<TUser> userManager) …

public async **Task** GetProfileDataAsync(ProfileDataRequestContext context)

{

if (context.Caller.Equals("ClaimsProviderIdentityToken"))

{

//Add claims to identity token

If (context.RequestedResources.Resources.IdentityResources.Any(x => x.Name == "openid"))

{

var subClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "sub");

context.IssuedClaims.Add(subClaim);

}

if (context.RequestedResources.Resources.IdentityResources.Any(x => x.Name == "email"))

{

var emailClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "email");

context.IssuedClaims.Add(emailClaim);

}

if (context.RequestedResources.Resources.IdentityResources

.Any(x => x.Name == "custom.profile"))

{

var subClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "sub");

var usernameClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "username");

var emailClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "email");

context.IssuedClaims.Add(subClaim);

if (usernameClaim != null)

{

context.IssuedClaims.Add(usernameClaim);

}

context.IssuedClaims.Add(emailClaim);

}

}

if (context.Caller.Equals("UserInfoEndpoint"))

{

var subClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "sub");

if (subClaim != null)

{

var user = await \_userManager.FindByIdAsync(subClaim.Value);

if (user != null)

{

context.IssuedClaims = context.Subject.Claims.ToList();

var claims = await \_userManager.GetClaimsAsync(user);

context.IssuedClaims.AddRange(claims);

}

}

//return user info claims

}

if (context.Caller.Equals("ClaimsProviderAccessToken"))

{

context.IssuedClaims.AddRange(context.Subject.Claims);

//Add claims to access token

}

}

}

Primer 8 *Identity Server 4 ProfileService metoda za definisanje tvrdni unutar tokena*

**GetProfileDataAsync** će se pozvati automatski pre izdavanja tokena klijentskoj aplikaciji i ostaje na programeru da definiše pravila i uslove izdavanja specifičnih tvrdnji unutar tokena.

*GetProfileDataAsync* kao parametar zahteva **ProfileDataRequestContext** **context** (primer 9) u kome se nalaze potrebne informacije za validaciju i provere specifičnih 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 (mogu se definisati neke posebne tvrdnje za specificne klijente) itd.

public class ProfileDataRequestContext

{

public ProfileDataRequestContext() …;

public ValidatedRequest ValidatedRequest { get; set; }

public ClaimsPrincipal Subject { get; set; }

public IEnumerable<string> RequestedClaimTypes { get; set; }

public Client Client { get; set; }

public string Caller { get; set; }

public ResourceValidationResult RequestedResources { get; set; }

public List<Claim> IssuedClaims { get; set; }

}

Primer 9 *Identity Server 4 ProfileService zahtev za dobijanje tvrdnji unutar tokena*

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

Pomocu **context.Caller** parametra može se 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 određeni klijent može da zahteva preko **scope** parametra i koji su definisani 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 ove tvrdnje se mogu koristiti za kreiranje niza tvrdnji unutar tokena. Ukoliko unutar kolačića nije definisana tvrdnja koja je potrebna, na primer prilikom User Info endpointa, klijentska aplikacija može 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 takođe definiše metodu *IsActiveAsync* pomoću koje se može validirati da li postoji i da li je aktivan korisnik za koga se zahtevaju informacije o idenitetu i tokeni (primer 10).

public async Task IsActiveAsync(IsActiveContext context)

{

var subClaim = context.Subject.Claims.FirstOrDefault(x => x.Type == "sub");

if (subClaim == null)

{

context.IsActive = false;

return;

}

var user = await \_userManager.FindByIdAsync(subClaim.Value);

if (user == null)

{

context.IsActive = false;

return;

}

context.IsActive =true;

}

Primer 10 *Identity Server 4 ProfileService validacija korisnika pre izdavanja tokena*

## IdentityServer postavka projekta

Sada kada su definisalni potrebni resursi identiteta, klijenti, stranicu za prijavljivanje, kao i ProfileService, potrebno je sve povezati unutar IdentityServer projekta.

Za definisanje resursa identiteta i klijenta koristiće se klasa *Config* koja sadrži jedan niz dostupnih resursa identiteta kao i jedan niz klijenata (primer 11).

public class Config

{

public static IConfiguration Configuration { get; set; }

public Config(IConfiguration configuration)

{

Configuration = configuration;

}

public static IEnumerable<IdentityResource> Ids =>

new IdentityResource[]

{

new IdentityResources.OpenId(),

new IdentityResources.Profile(),

new IdentityResources.Email(),

new IdentityResource(

name: "custom.profile",

userClaims: new[] { "sub", "email", "username" },

displayName: "Your user identifier")

};

public static IEnumerable<Client> Clients =>

new Client[]

{

new Client { //MVC Client},

new Client {// Angular Client}

};

}

Primer 11*.* *IdentityServer Definisanje resursa identiteta I klijenata*

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

Zatim unutar Startup klase IdentityServer projekta treba omogućiti korišćenje ovog paketa (primer 12).

public void Configure(IApplicationBuilder app)

{

app.UseStaticFiles();

app.UseRouting();

app.UseIdentityServer();

app.UseAuthorization();

app.UseEndpoints(endpoints =>{…});

}

Primer 12 *IdentityServer integracija IdentityServer 4 framework-a*

Na kraju je potrebno registrovati IdentityServer4, klijente, resurse identiteta, Asp Net Identity integraciju, konekciju sa bazom gde će se čuvati informacije o korisnicima kao i ProfileService (primer 13).

public class Startup

{

public IWebHostEnvironment Environment { get; }

public Startup(IWebHostEnvironment environment, IConfiguration configuration)

{

Configuration = configuration;

Environment = environment;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<AppUser, AppRole>();

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

var builder = services.AddIdentityServer (

options =>

{

options.UserInteraction.LoginUrl = "/authentication/signIn";

options.UserInteraction.LogoutUrl = "/authentication/logout";

}

)

.AddDeveloperSigningCredential()

.AddInMemoryIdentityResources(Config.Ids)

.AddInMemoryClients(Config.Clients)

.AddJwtBearerClientAuthentication()

.AddAspNetIdentity<AppUser>()

.AddProfileService<ProfileService<AppUser>>();

}

}

Primer 13*. IdentityServer registrovanje servisa*

# Angular Client

Nakon što je implementiran server identiteta, sada će biti definisana klijentska aplikacija pod nazivom AngularCient koristeći Angular 11 Framework-a.

S obzirom da se ovakav tip aplikacije izvršava isključivo u korisničkom pretraživaču, za autentifikaciju i pribavljanje tokena će se koristiti PKCE tok sa prethodno definisanim javnim klijentom na strani servera identiteta (javni klijent i PKCE tok s obzirom da ovakav tip klijentske aplikacije ne može sigurno čuvati klijentsku tajnu).

## PKCE tok autorizacije

Pre nego što se krene sa implementacijom Angular klijenta, ukratko će biti objašnjeno koji su koraci PKCE toka potrebni za uspešnu autentifikaciju i pribavljanje tokena identiteta.

* **Authorize –** Prvi korak autentifikacije jeste poziv authorize metode na strani servera identiteta i slanje potrebnih parametara poput identifikatora klijenta (client\_id), opsega zahtevanih tvrdnji unutar tokena identiteta (scope), heširana random vrednost od strane klijentske aplikacije (code\_challange), url klijentse aplikacije gde će se poslati potrebni kod nakon valididacije, itd.
* **Sign In** **–** Korisnički interfejs za unos kredencijala ukoliko korisnik pre toga nije prošao kroz proces autentifikacije (nije postavljen kolačić sesije). Ovaj korak je implementiran samo na strani servera identiteta i nije povezana sa klijentskom aplikacijom.
* **Token –** Slanje zahteva za dobijanje potrebnih tokena sa strane klijentske aplikacije, pozivom token metode na strani servera identiteta. Da bi se uspešno dobili tokeni, klijentska aplikacija mora poslati parametre za validaciju na strani servera identiteta, poput koda dobijenog kao rezultat authorize metode (code), identifikator klijenta (client\_id), izvornog random niza karaktera (*code\_verifier* koji zapravo predstavlja hesirani code=challange predhodno poslatog kroz authorize metodu) itd.
* **Validacija Tokena –** Nakon dobijanja tokena identiteta, potrebno je da klijentska aplikacija izvrši potrebne sigurnosne validacije (poput proveravanja nonce, **aud** tvrdnji).

## Angular Lib for OpenID Connect & OAuth2

Za implementaciju svih neophodnih koraka autentikacije sa PKCE tokom, korišćena je angular biblioteka pod nazivom angular-oidc-client, koja je zvanično sertifikovana angular biblioteka od srane OIDC fondacije pod MIT licensom [21].

Da bi koristili ovu biblioteku potrebno je skinuti npm paket unutar node modula angular aplikacije.

**npm install angular-auth-oidc-client**

**Podešavanje klijentskih parametara korišćenjem angular-oidc-client biblioteke**

Unutar App modula angular klijentske aplikacije potrebno je registrovati oidc klijenta, na sličan način na koji je to odrađeno unutar servera identiteta (primer 14).

Import { AuthModule, LogLevel, OidcConfigService, OidcSecurityService }

from 'angular-auth-oidc-client';

export function configureAuth(oidcConfigService: OidcConfigService) {

return () => oidcConfigService.withConfig ({

clientId: 'ClientAngular',

redirectUrl: window.location.origin,

responseType: 'code',

scope: 'openid email',

logLevel: LogLevel.None,

autoUserinfo: true,

storage: window.localStorage,

maxIdTokenIatOffsetAllowedInSeconds: 60 \* 60 \* 12,

stsServer: 'https://localhost:44333'

});

}

@NgModule({

declarations: [

AppComponent

],

imports:[

AuthModule.forRoot(),

],

providers:

[

OidcConfigService,

{

provide: APP\_INITIALIZER,

useFactory: configureAuth,

deps:[OidcConfigService],

multi: true,

}

],

bootstrap:[AppComponent]

})

export class AppModule { }

Primer 14 *AngularClient registrovanje oidc klijenta*

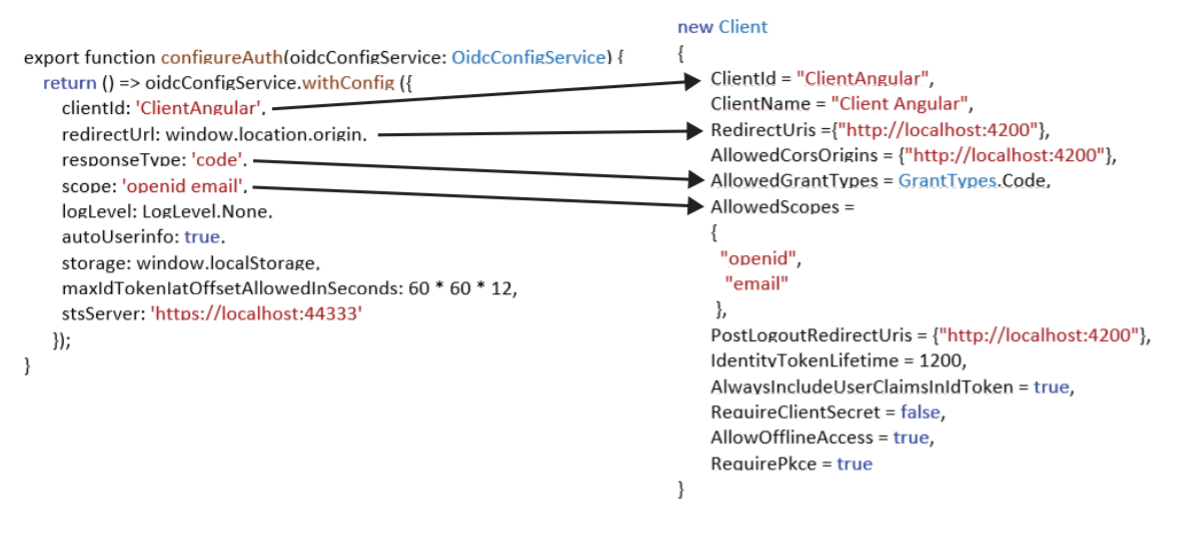
Kao što se može videte, postupak registrovanja klijenta na strani angular klijentske aplikacije je jako sličan registraciji klijenta na strani servera identiteta.

U nastavku su dati potrebni parametri:

* **stsServer:** Url adresa servera identiteta.
* **redirectUrl:** Url adresa klijentske aplikacije na koju će se redirektovati server identiteta sa kodom, nakon authorize metode i uspešne autentifikacije.
* **clientId:** Identifikator klijenta.
* **scope:** Zahtevani opsezi tvrdnji unutar tokena identiteta registrovanih kao resursi identiteta unutar servera identiteta.
* **responseType:** Identifikator korišćenog toka autorizacije, u ovom slučaju se radi o toku autorizacionim kodom (ukoliko se radi o toku autorizacionim kodom po default-u podrazumeva da je reč o PKCE toku).
* **autoUserInfo:** Setovanjem ovog parametra, klijentska biblioteka će nakon dobijanja tokena automatski pozvati userInfo metodu unutar servera identiteta za dobijanje dodatnih informacija o korisničkom identitetu.
* **storage:** Ovim parametrom se definiše skladište za čuvanje tokena. Može se koristiti HTML 5 localStorage, sessionStorage ili se definisati sopstveno skladište.
* **maxIdTokenIatOffsetAllowedInSeconds:** Parametar validacije tokena u odnosu na razliku vremena od kada je token izdat preko servera identiteta do prijema na strani klijentske aplikacije.

Ova biblioteka će u pozadini izvršiti i neophodne validacije tokena identiteta. Informacije o potrebnim metodama toka autorizacije poput **authorize/token/userInfo**, biblioteka dobija korišćenjem **.well-known/openid-configuration** endpoint-a, definisane na strani servera identiteta korišćenjem Identity Server 4 framework-a.

Da bi se klijentska aplikacija uspešno autentifikovala potrebno je da se klijentski parametri registrovani na strani angular klijenta i servera identiteta poklapaju (slika 19).



Slika *19. ClientAngular klijent definisan na strani klijentske aplikacije I servera identiteta*

Nakon što je registrovan klijent, koriščenjem metoda OidcSecurityService-a implementiranog od srane angular-oidc-client biblioteke, može se jednostavno upravljati procesom autentifikacije (primer 15).

* Pozivom **oidcSecurityService.authorize()** započinje se proces autentifikacije.
* Može se izvršiti proveru da li je korisnik uspešno autentifikovan koriščenjem **oidcSecurityService.checkAuth()** metode.
* Potrebne informacije i tvrdnje o identitetu korisnika može se dobiti korišćenjem **oidcSecurityService.userData$** polja.
* Pozivom **oidcSecurityService.logoffAndRevokeTokens()** metode, mogu se obrisati tokeni iz skladišta i izvršiti odjavljivanje korisnika na strani servera identiteta.

export class HomeComponent implements OnInit

{

constructor(private oidcSecurityService: OidcSecurityService) { }

ngOnInit(): void

{

}

login() {

this.oidcSecurityService.authorize();

}

isAuth() {

this.oidcSecurityService.checkAuth().subscribe((auth) => console.log('is authenticated', auth));

}

getUserInfo() {

this.oidcSecurityService.userData$.subscribe((user) => {

console.log("User -->", user);

})

}

signOut() {

this.oidcSecurityService.logoffAndRevokeTokens().subscribe();

}

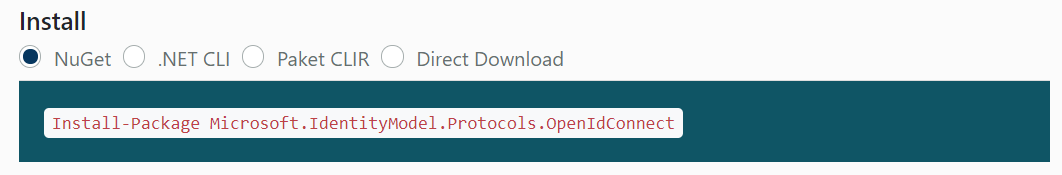
}

Primer 15 *angular-oidc-client metode OidcSecurityService-a za upravljanje autentikacijom*

# MVC Client

Druga klijentska aplikacija pod nazivom ClientMVC je implementirana korišćenjem .Net Core 5 Framewok-a. S obzirom da ovakav tip veb aplikacije ima svoju serversku stranu i može bezbedno čuvati klijentsku tajnu, koristiće se poverljiv klijent registrovan u serveru identiteta u kombinaciji sa hibridnim tokom autorizacije.

Kao pomoć u implemetiranju hibridnog oidc toka, koristišćena je Microsoft-ova biblioteka OpenIdConnect i potrebno je u ClientMVC projektu instalirati nuget paket (slika 20)



Slika *20 ClientMVC OpenIdConnect nuget*

## ClientMVC definisanje klijenta

Nakon što je instaliran OpenIdConnect nuget paket, potrebno je definisati klijenta unutar startup klase preko koga se vrši tok autentikacije (primer 16).

public void ConfigureServices(IServiceCollection services)

{

services.AddAuthentication().AddCookie()

.AddOpenIdConnect(options =>

{

options.ClientId = "ClientMVC";

options.SignInScheme = "Cookies";

options.Authority = "https://localhost:44333";

options.ClientSecret = "MVCSecret";

options.ResponseType = "code id\_token";

options.SaveTokens = true;

options.GetClaimsFromUserInfoEndpoint = true;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidAudience = "ClientMVC"

};

options.Scope.Clear();

options.Scope.Add("openid");

options.Scope.Add("custom.profile");

});

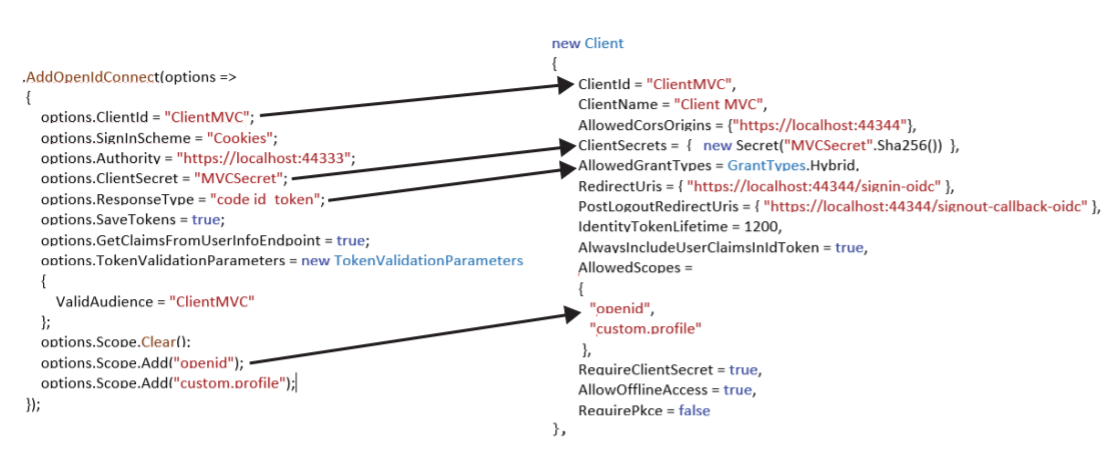
}

Primer *16 ClientMVC OpenIdConnect definisanje klijenta*

Definisanje klijenta se vrši pozivom .AddOpenIdConnect metode sa sledećim parametrima (primer 16):

* **options.SignInScheme –** predstavlja internu šemu autentikacije (ovakav tip aplikacije uglavnom koristi kolačiće za autentifikaciju i autorizaciju) koja se koristi u ClientMVC projeku. Jednom kada se klijentska aplikacija uspešno autentifikuje preko servera identiteta i dobije tokene, ona generiše svoje interne kolačiće sa tvrdnjama dobijenim iz tokena.
* **options.Authority –** Url adresa servera identiteta
* **options.ClientId** **–** Identifikator klijenta registrovanog u serveru identiteta.
* **options.ClientSecret** **–** Klijentska tajna potrebna za dobijanje tokena nakon prijema autorizacionog koda.
* **options.ResponseType** **–** Predstavlja tip oidc toka autorizacije i u ovom slučaju je setovan na “code id\_token”, što označava da se radi o hibridnom toku i da će server identiteta nakon autentifikacije dobiti kod i token identiteta pomoću kojih se uz pomoc klijentske tajne dobijaju tokeni pozivom token metode unutar servera identiteta.
* **options.SaveTokens** **–** Pomoću ovog parametra se definiše da li će se tokeni čuvati u internim kolačićima klijentske aplikacije (poput HTML 5 localStorage, sessionStorage unutar angular aplikacije).
* **options.GetClaimsForUserInfoEndpoints** **–** Setovanjem ovog parametra se odlučuje da li će klijentska aplikacija, nakon dobijanja tokena, pozvati UserInfo metodu na strani servera identiteta da bi odbila dodatne tvrdnje o identitetu korisnika.
* **options.TokenValidationParameters** **–** Pomoću ove opcije mogu se izvršiti dodatne validacije tokena identiteta poput ispitivanja **aud** tvrdnje, odnosno ispitivanja kome je token namenjen.
* **options.Scope** – Scope nizom se definiše opseg tvrdnji, odnosno resursa, koje će se naći u tokenu identiteta, i koje su predhodno registrovane unutar servera identiteta. Ova biblioteka će po default-u postaviti “openid profile” opsege resursa identiteta i ukoliko želimo da postavimo drugaćije opsege potrebno je predhodno pozvati Clear() metodu koja prazni niz, i nakon toga setovati nove opsege pozivom metode add (options.Scope.Add(“openid”), options.Scope.Add(“custom.profile”)).

Da bi se klijentska aplikacija uspešno autentifikovala potrebno je da se klijentski parametri registrovani na strani MVC klijenta i servera identiteta poklapaju (slika 21).

****

Slika *21 ClientMVC klijent definisan na strani klijentske aplikacije I servera identiteta*

Takođe se može primetiti da **RedirectUris** parametar (url adresa ClientMVC aplikacije na kojoj će server identiteta vratiti odgovor nakon uspešne autentifikacije) na strani servera identiteta poziva metodu **signin-oidc**. Ova metoda je implementirana u klijentskoj aplikaciji od strane **Microsoft OpenIdConnect** biblioteke i predstavlja middleware za dobijanje koda autorizacije i slanja token poziva [22].

Nakon što je definisan klijent u Startup klasi, potrebno je napraviti metodu za početak procesa autentikacije. Nakon što OpenIdConnect biblioteka preuzme i validira tokene, unutar kontrolera može se preko HttpContext-a pristupiti tokenima i izvući potrebne korisničke tvrdnje (primer 17).

namespace ClientMVC.Controllers

{

[Authorize]

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

[AllowAnonymous]

public IActionResult Login()

{

return Challenge(new Microsoft.AspNetCore.Authentication.AuthenticationProperties

{

RedirectUri = "/Home/Index",

}, "OpenIdConnect");

}

public async Task<IActionResult> Privacy()

{

var accessToken = await HttpContext.GetTokenAsync("access\_token");

var id\_token = await HttpContext.GetTokenAsync("id\_token");

var claims = HttpContext.User.Claims;

return View();

}

}

}

Primer *17 ClientMVC startovanje autentikacije I pristupanje tokenima I korisničkim tvrdnjama*

# Zaključak

Korisnička imena i lozinke glavna su meta sajber kriminala.Svaki put kada se korisnik prijavi na novu aplikaciju, to je prilika za hakere. Single Sign On smanjuje broj površina za napad jer se korisnici prijavljuju samo jednom dnevno i koriste samo jedan skup kredencijala.

Smanjivanje prijave na jedan skup kredencijala poboljšava bezbednost preduzeća.Kada zaposleni moraju da koriste zasebne lozinke za svaku aplikaciju, obično to ne rade. U stvari, 59% korisnika koristi iste ili slične lozinke na više naloga.Zbog toga ukoliko haker dobije pristup preko jedne loše zaštićene aplikacije verovatno ce moći da pristupi i drugim korporativnim sistemima.

U teorijskom delu ovog rada fokus je bio na razumevanju potreba za Single Sign On- om kao i njegova realizacija korišćenjem OpenId Connect protokola.

Dodatna paznja je bila na sigurnosti takvog sistema s toga je više paznje bilo posvećeno na validaciji tokena identiteta kao i tokova autorizacije koje definise OpenId Connect protokol u cilju bezbednog pribavljanja tokena i autentikaciji korisnika.

S obzirom da OpenId Connect definiše više tokova za dobijanje tokena identiteta u ovom radu je dato objašnjenja za svaki tok koji su uslovi i kriterijumi za najbezbedniji odabir u zavisnosti od prirode same aplikacije, s toga smo u praktičnom primeru koristili 2 različita toka za pribavljanje tokena (PKCE za Angular Client, Hibridni Tok za MVC Client).

Takođje je detaljno objašnjena uloga Servera Identiteta u cilju implementacije Singe Sign On-a kao i terminologije definisane OpenId Connect i OAuth2 protokolima.

U praktičnom delu rada je kroz primere detaljno objasnjena potrebna arhitektura ovakvog sistema kao i koriščenje gotovih biblioteka u cilju povećanja sigurnosti sistema kao i štednji vremena za samostalnu implementaciju.

U praktičnom primeru može se zaključiti da ukoliko se korisnik bude prijavio preko jedne od klijentskih aplikcija na primer Angular Client, zbog postojanja servera identiteta, tokena identiteta kao i kolačića sesije, korisnik se može redirektovati na drugu klijentsku aplikaciju (MVC Client) bez potrebe za ponovnom autentikacijom. Samim tim se povećava korisnikov doživljaj sistema.

# Reference

[1] Sebastian Peyrott, Auth0 September 23. 2015. What is and how does single sign on work. Available at: https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/ [Accessed 12 April 2021].

[2] Digital Guide Ionos, October 04. 2019. Single Sign On Available at: <https://www.ionos.com/digitalguide/server/tools/single-sign-on/> [Accessed 25 April 2021].

[3] Taina Teravainen Definiton of Single Sign On Available at: <https://searchsecurity.techtarget.com/definition/single-sign-on> [Accessed 03 May 2021].

[4] Okta Concepts of OAuth and OpenId Available at: <https://developer.okta.com/docs/concepts/oauth-openid/> [Accessed 13 May 2021].

[5] Vidizmo Authentication Protocols: OpenID Connect vs. SAML Available at: <https://blog.vidizmo.com/authentication-protocols-openid-connect-vs.-saml> [Accessed 22 May 2021].

[6] OpenId What is OpenID Connect. Available at: <https://openid.net/connect/> [Accessed 01 June 2021].

[7] IBM Security OAuth 2.0 concepts. Available at: <https://www.ibm.com/docs/en/sva/9.0.7?topic=SSPREK_9.0.7/com.ibm.isam.doc/config/concept/con_oauth_20_concepts.html> [Accessed 08 June 2021].

[8] Robert Broeckelmann May 21, 2017. When To Use Which (OAuth2) Grants and (OIDC) Flows. Available at: <https://medium.com/@robert.broeckelmann/when-to-use-which-oauth2-grants-and-oidc-flows-ec6a5c00d864> [Accessed 15 June 2021].

[9] Robert Broeckelmann Mar 25, 2017. SAML2 vs JWT: Understanding OpenID Connect. Available at: <https://medium.com/@robert.broeckelmann/saml2-vs-jwt-understanding-openid-connect-part-2-f361ca867baa> [Accessed 17 June 2021].

[10] Curity Using the OpenID Connect Hybrid Flow. Available at: <https://curity.io/resources/learn/hybrid-flow/> [Accessed 22 June 2021].

[11] Auth0 Authorization Code Flow with Proof Key for Code Exchange (PKCE). Available at: <https://auth0.com/docs/flows/authorization-code-flow-with-proof-key-for-code-exchange-pkce> [Accessed 28 June 2021].

[12] OpenId Specification 3.1.3.7. ID Token Validation. Available at: <https://openid.net/specs/openid-connect-core-1_0.html#IDTokenValidation> [Accessed 05 July 2021].

[13] Mrochon August 04, 2016. Single SignOn is easy, Single Signout is not. Available at: <https://mrochon.azurewebsites.net/2016/08/04/single-signon-is-easy-single-signout-is-not/> [Accessed 08 July 2021].

[14] Ashen De Silva August 27, 2018. OpenID Connect — Session Management. Available at: <https://ashend.medium.com/openid-connect-session-management-d0c8e7dc252b> [Accessed 15 July 2021].

[15] OpenId Specification OpenID Connect RP-Initiated Logout. Available at: <https://openid.net/specs/openid-connect-rpinitiated-1_0.html> [Accessed 20 July 2021].

[16] Ashen De Silva December 04, 2018. OpenID Connect Backchannel Logout. Available at: <https://ashend.medium.com/openid-connect-backchannel-logout-144a3198d2a> [Accessed 28 July 2021].

[17] Github July 07, 2021. IdentityServer4. Available at: <https://github.com/IdentityServer/IdentityServer4> [Accessed 03 August 2021].

[18] Microsoft May 13, 2020. IdentityServer for cloud-native applications. Available at: <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/identity-server> [Accessed 10 August 2021].

[19] Docs.IdentityServer Welcome to IdentityServer4 (latest) Available at: <https://docs.identityserver.io/en/latest/index.html> [Accessed 17 August 2021].

[20] Docs.IdentityServer IdentityServer4 Sign In Topic. Available at: <https://docs.identityserver.io/en/latest/topics/signin.html> [Accessed 20 August 2021].

[21] NpmJs Damien Bod August 20, 2021. Angular Lib for OpenID Connect & OAuth2. Available at: <https://www.npmjs.com/package/angular-auth-oidc-client> [Accessed 22 August 2021].

[22] Github August 19, 2021. Use OpenId Connect with Dotnet Core & OneLogin. Available at: <https://github.com/onelogin/openid-connect-dotnet-core-sample> [Accessed 28 August 2021].