Proxy server co OpenID

Изработиле:

Александар Трајковски 151083

Стефан Стефановски 151028

Содржина

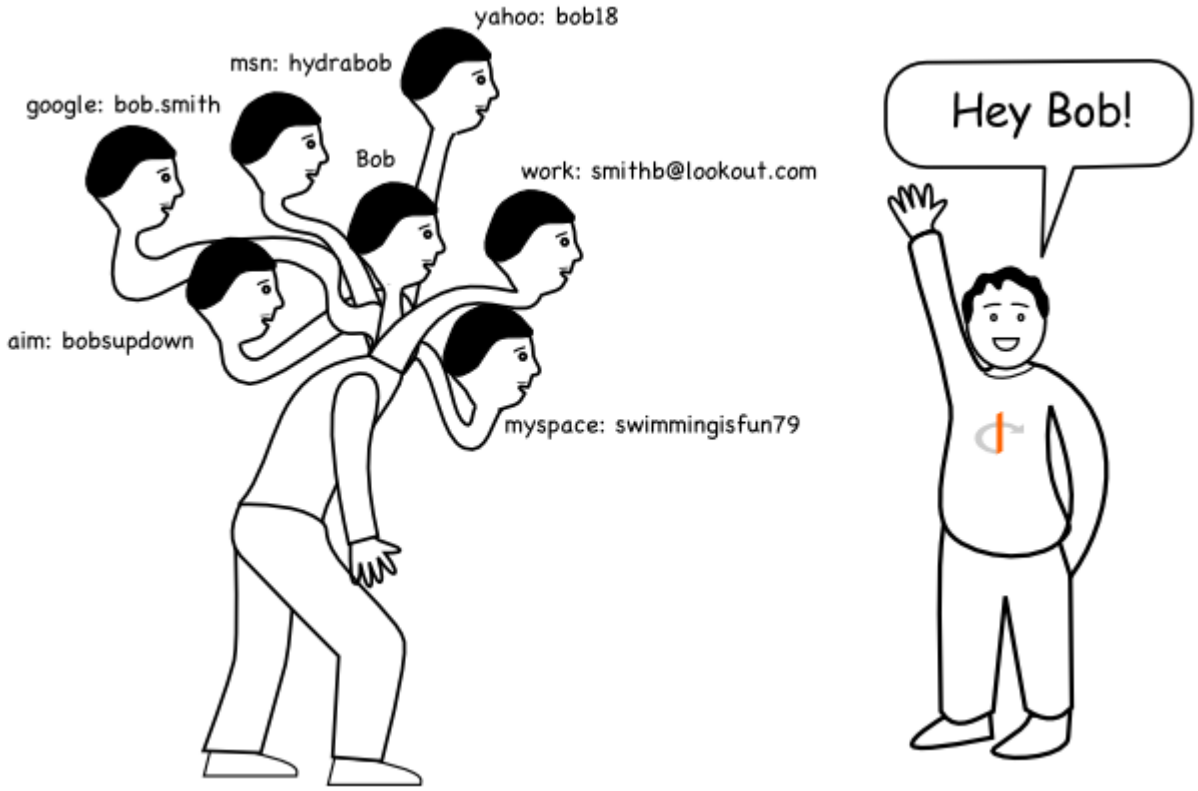
1. Вовед

Со развојот на технологијата луѓето денес постојано се поврзани за интернет. Тие користат најразлични апликации. Од нив огромен дел ги бараат истите податоци како e-mail, адреса, телефон, име и презиме. Регистрацијата на повеќе од една апликација може да биде напорна и премногу досадна. Заради овој проблем сé повеќе се прават апликации кои подржуваат OPENID connect.

Со зголемувањето на маркетот и користењето на личните податоци за секоја мала ситница се изложуваме на напади од малициозни лица кои вршат злоупотреба на истите тие податоци. За заштитата од таа опасност луѓето се приклонуваат кон користење на Proxy Server-и кои нудат анонимност и во некои случаи и bypassing на блокирани IP адреси во некој регион.

2. Што е OPENID?

OpenID е "возачка дозвола" за интернетот. Тој ни помага да добиеме пристап до OpenID подржани страници без претходна регистрација. Поефективен е од традиционалниот метод, бидејќи со него можеме да нагласиме кои информации сакаме страната да ги знае за нас (claims). Со ова се негира фрустрацијата за повторно и повторно пополнување на истите информации за секоја регистрација. Освен поедноставена и доста ефикасна регистрација, OpenID нуди и брза поедноставена најава. Потребно е да се запамети само една лозинка и едно корисничко име, за разлика од традиционалниот метод каде за секоја апликација и регистрација треба нова лозинка и ново корисничко име кои многу брзо се заборавааат.

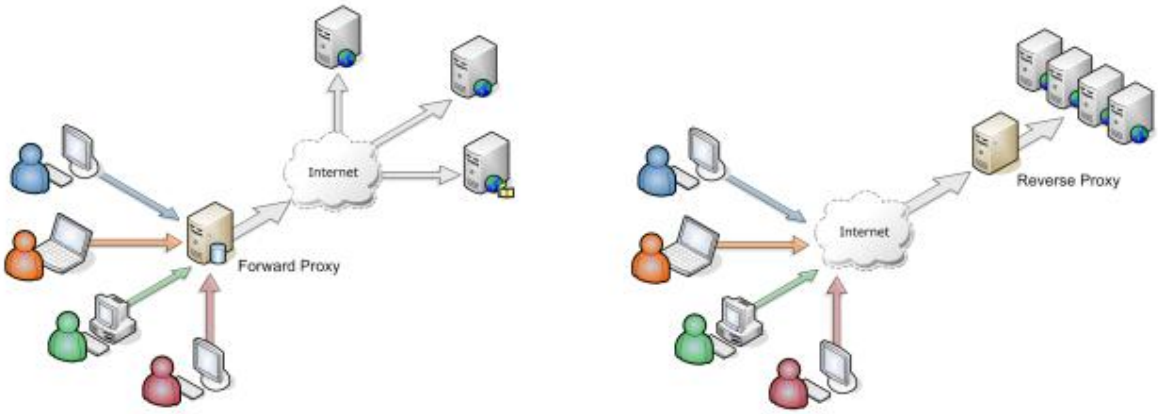


Варијанта на OpenID е OpenID Connect која ќе ја користиме во нашата апликација. Некои од клучните разлики е тоа што OpenID Connect користи OAuth 2.0 така да наместо само автентикација воедно се прави и авторизација. OpenID Connect овозможува пристап до инфомациите на корисниците во вид на JWT(Json Web Token) или така наречен ID token. Податоците вратени во тој токен се наречени claims/attributes.

3. Proxy Server

Proxy Server претставува сервер кој е посредник помеѓу клиентот и серверот кој ги чува информациите. Клиентот прво се конектира на Proxy Server и бара некоја услуга. По ова proxy server-от се конектира до серверот кој ја дава услугата и ја побарува услугата побарана од клиентот. Серверот ја враќа услугата на Proxy server-от која потоа ја препраќа на клиентот. Најчесто се користат за кеширање на побарувања со што се забрзува времето на одговор. Се користат и за прескокнување на блокирани страници како и за алатка која му дава анонимност на клиентот. Постојат два вида на proxies и тоа Forward и Reverse proxy.

Forward proxy се користи за праќање на барања од клиентот кон веб серверот. Reverse proxy се однесува како да е вистинскиот сервер, но во позадина без знаење на корисникот прима и праќа барања во внатрешна мрежа. Одговорите се вратени исто како што се добиени од внатрешната мрежа. За разлика од Forward proxy кој може да има пристап до многу други сервери, Reverse proxy има ограничен пристап до неколку сервери од внатрешната мрежа. Reverse proxy се користи начесто за енкрипција, балансирање на сообраќај, кеширање , компресија итн.



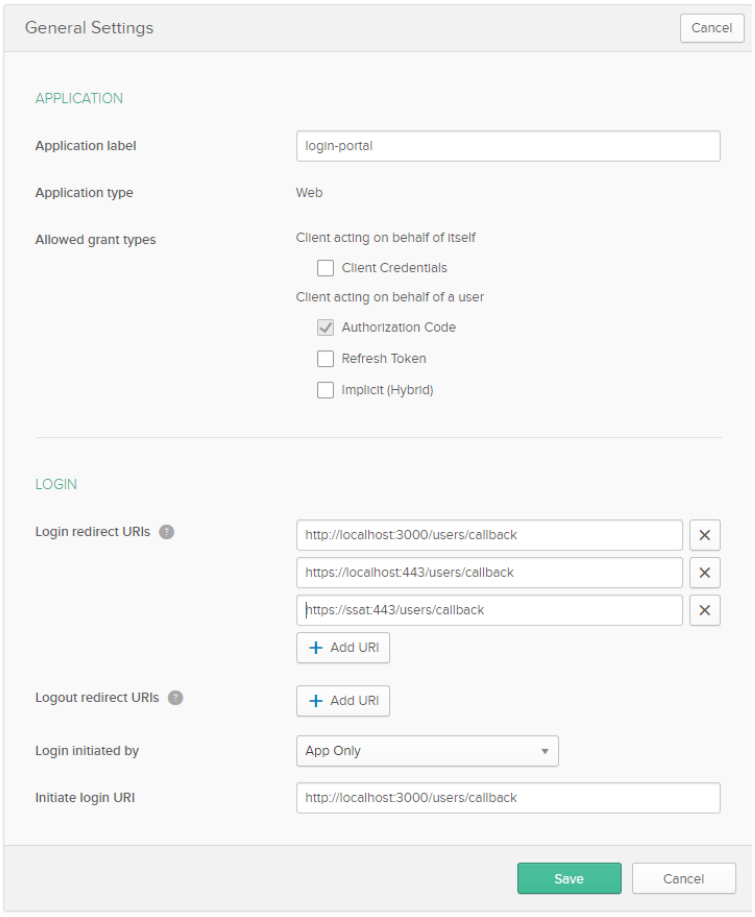
4. Имплементација на reverse proxy со OpenID Connect

Имплементирана е апликација со помош на nodeJS. За авторизација и автентикација е искористен OKTA кој претставува лиценциран издавач на OpenID Connect. Proxy server-от е подигнат на APACHE архитектура на Windows машина преку апликацијата Xampp.

4.1 NodeJS и OpenID connect имплементација

4.1.1 Креирање на девелоперска околина во OKTA и сетирање на апликација

Прво се регистриравме на https://developer.okta.com/signup/. Со завршената регистрација беше добиен пристап до околина за креирање на апликации. Од менито за алатки избравме креирање на апликации и беше креирана апликација со име login-portal. Конфигурацијата е прикажана на следната слика.



Во полето за login redirect се поставени 3 адреси. Од нив 2 те се за proxy server-от поставен на Apache, а едната е за апликацијата во NodeJS која ќе е подигната на localhost:3000.

По креирањето на апликацијата системот издаде Client ID и Client Secret кои подоцна ќе бидат искористени во апликацијата.

Како втор чекор беше направена апликација, достапна на GitHub, со помош на NodeJS и Express.js. Таа се состои од неколку routes, views и еден конфигурациски фајл app.js.

На следните слики се прикажани конфигурациите за OpenID Connect кои се содржат во app.js.

Главниот дел од апликацијата е сетирањето на oidc. Во полето за issuer ја поставуваме адресата која ја добивме со регистрацијата на Okta. Во делот за Client ID и ClientSecret ги поставуваме Client ID и Client Secret кои ни ги издаде

Okta при креирање на апликацијата. Полето за препраќање е поставен на localhost:3000, а callback на view dashboard. View dashboard е достапно само за корисници кои успешно се најавиле на апликацијата. Сесијата е заштитена со долг стринг од случајно генерирани карактери. За крај од околината за девелопирање на апликацијата добиваме API token за информациите преку API-> Token. Токенот се користи во oktaClient преку кој ги добиваме сите потребни информации за корисникот.

Други делови кои ги содржи app.js се конфигурации за грешки, неовластени страни за ненајавени корисници и прерутирање на соодветни страни како dashboard, home, logout, users.

```
var path = require('path');

var okta = require("@okta/okta-sdk-nodejs");
var ExpressOIDC = require("@okta/oidc-middleware").ExpressOIDC;
var logger = require('morgan');
const dashboardRouter = require("./routes/dashboard");
const publicRouter = require("./routes/public");
const usersRouter = require("./routes/users");


var app = express();

var oktaClient = new okta.Client({
  orgUrl: 'https://dev-930032.oktapreview.com',
  token: '00tHiTZxK7TYulw4lkT8uFWP1wgYM61WSaC9p_s4mz'
});
const oidc = new ExpressOIDC({
  issuer: "https://dev-930032.oktapreview.com/oauth2/default",
  client_id: '0oag4mpfwfLYkdefF0h7',
  client_secret: '6sMB7p51KiK8-PGaNED7pNa4whRa8YGBWKD8H0mz',
  redirect_uri: 'https://ssat:443/users/callback',
  scope: "openid profile",
  routes: {
    login: {
      path: "/users/login"
    },
    callback: {
      path: "/users/callback",
      defaultRedirect: "/dashboard"
    }
  }
});

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

app.use(express.static(path.join(__dirname, 'public')));
app.use(session({
  secret: 'e3t983t8919g83tupfqe82y138[9r0hahfs;ofh2817g1pfaubsu;ogusa;o821t9guabu;gsd;bgashfia]',
  resave: true,
  saveUninitialized: false
}));
app.use(oidc.router);
```

```
app.use((req, res, next) => {
  if (!req.userinfo) {
    return next();
  }

  oktaClient.getUser(req.userinfo.sub)
    .then(user => {
      req.user = user;
      res.locals.user = user;
      next();
    }).catch(err => {
      next(err);
    });
});
```

4.1.2 Конфигурација на Apache серверот

Со користење на XAMPP беше покренат сервер на Apache. За истиот да е OpenSSL reverse proxy беа направени неколку измени во конфигурациските фајлови.

Во apache httpd-ssl.conf беше воведен виртуелен сервер на порта 443 со следните подесувања.

```
<VirtualHost _default_:443>

#   General setup for the virtual host
DocumentRoot "D:/xampp/htdocs"
ServerName SSAT:443
ServerAdmin admin@example.com
ErrorLog "D:/xampp/apache/logs/error.log"
TransferLog "D:/xampp/apache/logs/access.log"

#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   Server Certificate:
SSLCertificateFile "conf/ssl.crt/SSAT.crt"

#   Server Private Key:
SSLCertificateKeyFile "conf/ssl.key/SSAT.key"

#   Server Certificate Chain:
SSLCertificateChainFile "conf/SSAT_ca-chain.crt"

</VirtualHost>
```

За сертификат е искористен групниот сертификат SSAT.cert заедно со приватниот клуч и ланчето.

Виртуелниот сервер дополнително е конфигуриран во httpd-vhosts.conf со следниве параметри.

```
<VirtualHost *:443>
    ProxyRequests off
    ServerName SSAT
    <Proxy *>
      Order deny,allow
      Allow from all
    </Proxy>

    <Location />
      ProxyPass http://localhost:3000/
      ProxyPassReverse http://localhost:3000/
    </Location>
```

SSLEngine on

SSLCertificateFile "D:\xampp\apache\conf\ssl.crt\SSAT.crt"

SSLCertificateKeyFile "D:\xampp\apache\conf\ssl.key\SSAT.key"

SSLCertificateChainFile "D:\xampp\apache\conf\SSAT_ca-chain.crt"

</VirtualHost>

Во него е поставен proxy pass и proxypassreverse на localhost:3000 каде е хостирана нашата NodeJS апликација. Со ова се овозможува apache серверот да е proxy server со OpenSSL конекција.

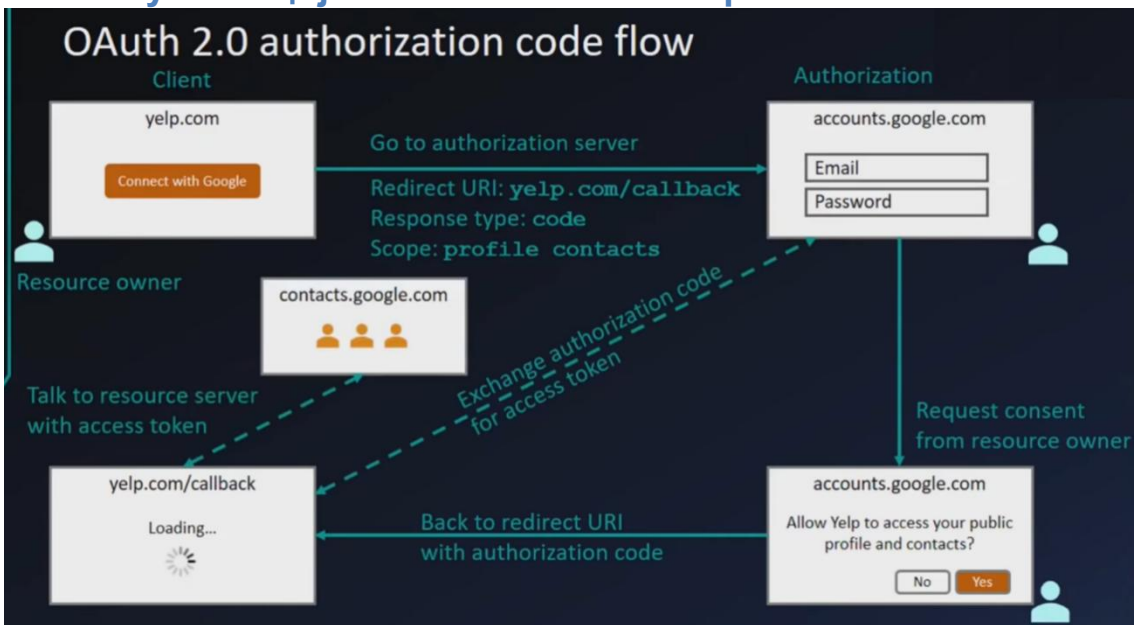
5. OpenID протоколот

За да го разбереме OpenID протоколот треба прво да го разбереме OAuth 2.0 протоколот бидејќи OpenID е всушност екстензија на OAuth 2.0.

Проблемот кој OAuth го решава е тој на делегирана авторизација, затоа се нарекува авторизациски слој. Делегирана авторизација е всушност кога една апликација ја авторизираме да пристапи до некои наши ресурси на друга апликација. На пример, го авторизираме *yelp.com* да ги пристапи напшите Gmail контакти. За нашето корисничко име и лозинка да не поминат преку *yelp.com*, користејќи OAuth ние можеме да се конектираме на нашиот Gmail акаунт и да го овластиме (авторизираме) yelp.com да ги пристапи нашите контакти. OAuth го постигнува токму ова на брз и ефикасен начин користејќи квери стринг параметри. Клиент апликацијата (во овој случај yelp.com) праќа барање до OAuth endpoint-от на Гмаил (во овој случај accounts.google.com) со неколку параметри: **response-type, redirect-uri, client-id** и **scope.**Овие параметри како и неколку други се строго дефинирани од OAuth протоколот. Да ги разгледаме:

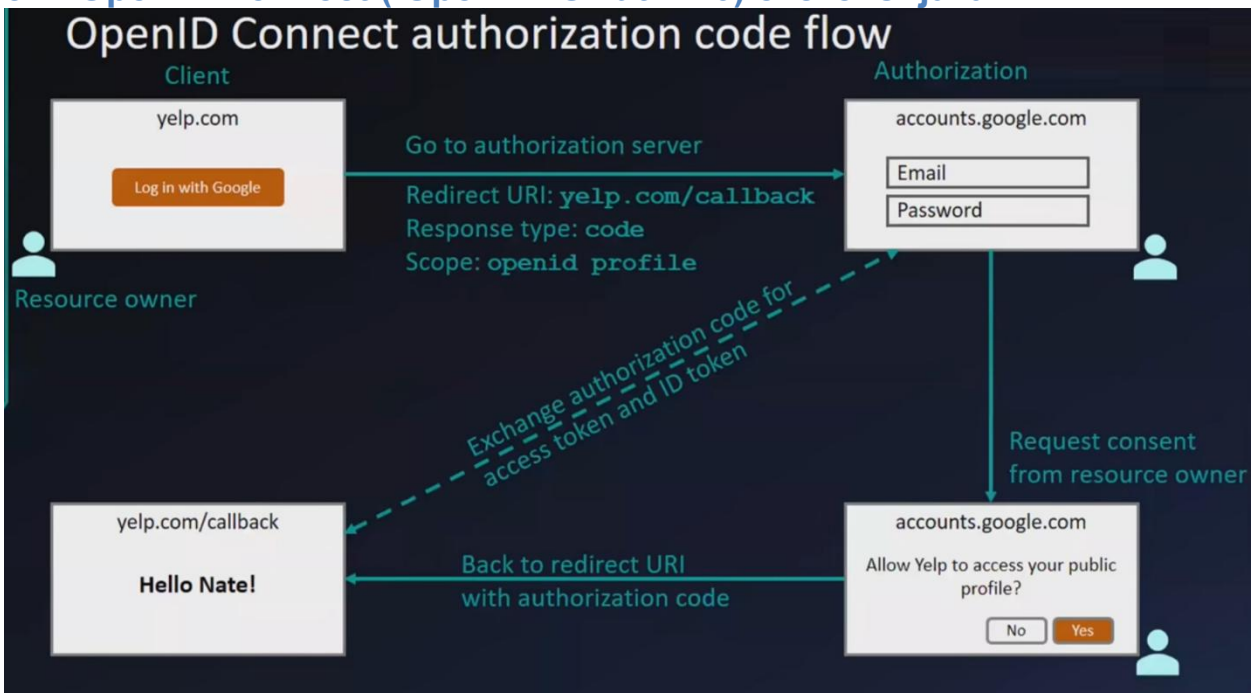
- Response-type – специфицира каква потврда ќе врати авторизацискиот сервер дека клиентската апликација е авторизирана. Може да биде access token, authorization code, id token итн.
- Redirect-uri – Специфицира на кој URI ќе не пренасочи авторизацискиот сервер откога ќе го авторизираме клиентот.
- Client-id – Идентификатор со кој авторизацискиот сервер препознава која клиентска апликација го праќа барањето. Овој ID се добива кога клиентот (yelp.com) ќе се регистрира со авторизацискиот сервер.
- Scope – Кои пермисии ги бара клиентот за пристап до нашите ресурси, на пример дозвола за читање, едитирање или бришење на контакти, име и презиме и други профилни информации и сл.

5.1. Визуелизација на текот на OAuth протоколот



Кога ќе кликнеме на *Connect with Google* копчето започнува текот на протоколот со генерирање на GET барање со претходно наведените параметри (и евентуално некои други). Нашиот прелистувач го прикажува одговорот на тоа барање и ја гледаме страната за најава на google.com. Откога ќе се најавиме се прикажува страната за авторизација. Google користејќи го client-id параметарот знае дека апликацијата која ги бара податоците е yelp.com и ги наведува пермисиите кои се побарани. Ние сега имаме избор дали да го овластиме yelp.com. При успешно овластување google го пренасочува прелистувачот на redirect-uri-то и додава уште еден параметар на тој uri кој претставува authorization code (бидејќи барањето специфицираше response-type: code), потоа yelp.com преку побезбеден заден канал го користи тој авторизациски код заедно со клиентски таен клиуч за да добие access токен (кој не се прикажува во прелистувачот за зголемена безбедност). Клиентот сега е авторизиран и завршува текот на протоколот.

5.2. OpenID Connect (OpenID+OAuth 2.0) екстензијата



Како што напоменавме OAuth се користи само за авторизација, но сепак пред појавата на OpenID некои платформи како Facebook, Google и сл. почнале да го овозможуваат OAuth протоколот да се користи и за автентикација, тоа е појавата на *Login with x* копчињата. Овие копчиња за социјална најава му овозможуваат на корисникот да се автентицира на трета апликација користејќи ги акредитивите од социјалната мрежа на безбеден начин. Бидејќи OAuth не бил наменет за автентикација овие имплементации не биле строго дефинирани и стандардизирани па затоа се појавил OpenID Connect стандардот кој точно дефинирал како клиентот може да го автентицира корисникот преку автентикациски сервер на трета страна.

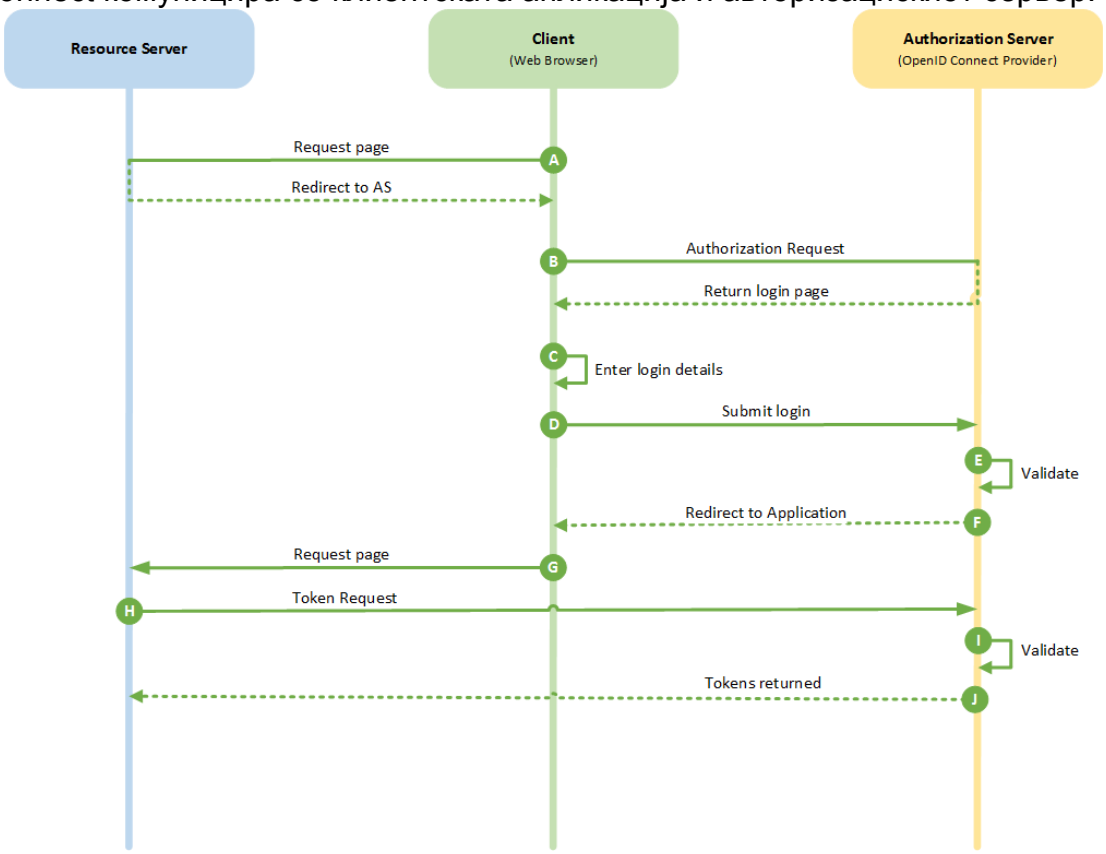
За таа цел се уште се користи OAuth но се додаваат некои параметри:

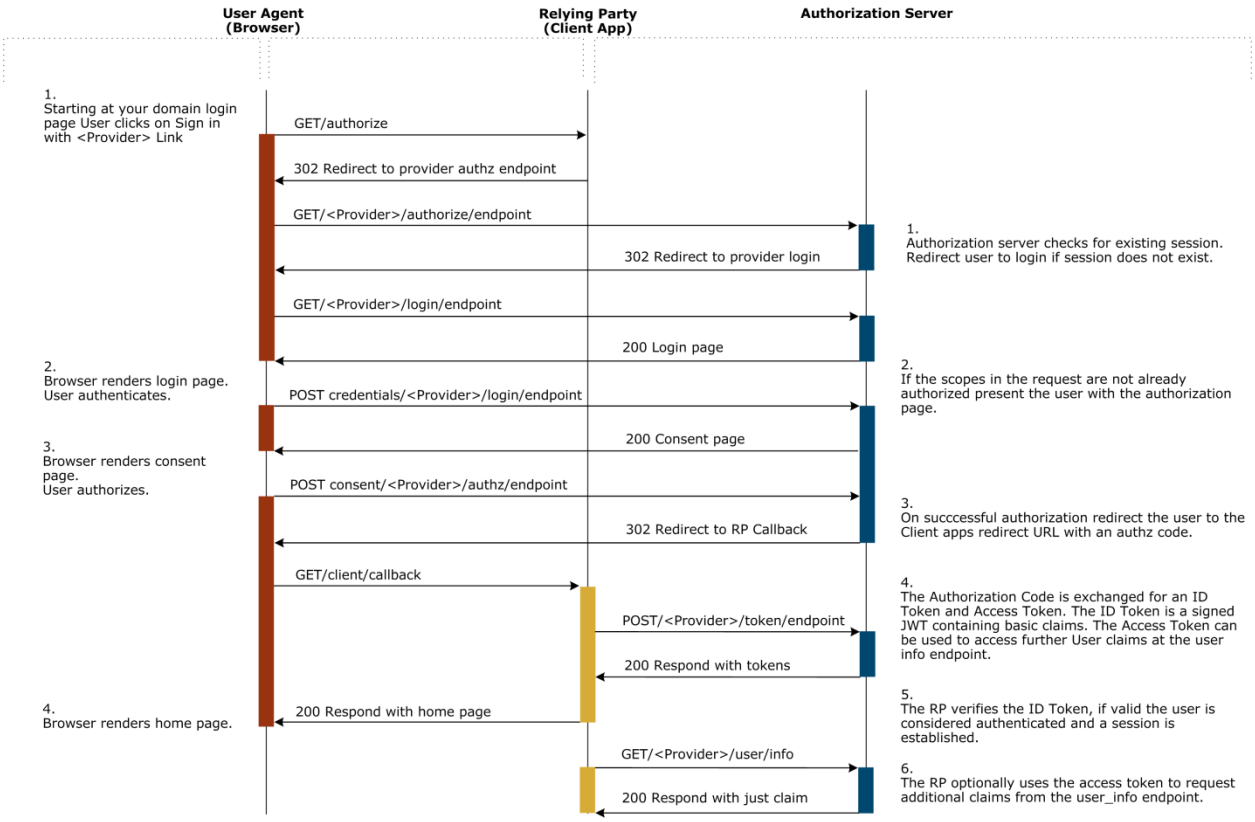
- ID token – JSON Web Token (jwt) стринг кој содржи потпишани и компактно кодирани информации за корисникот (JWT стандардот е целосна тема за себе).
- UserInfo endpoint – URI на кој клиентската апликација може да побара додатни информации за корисникот.
- Стандардно множество на scopes
- Стандардизирана имплементација

OpenID барањето е всушност OAuth 2.0 барање па затоа и текот на протоколот е исто како тој на OAuth со главната разлика дека во *scope* параметрот на барањетпе наведено *“openid”* и евентуално во *response-type* параметрот е наведено “*id-token”* (или поверојатно id-tokenot ќе се добие преку задниот канал користејќи го авторизацискиот код).

По успешната автентикација на автентикацискиот сервер, клиентската апликација може да го искористи вратениот id-token за да го автентицира корисникот локално. Со тоа завршува текот на OpenID протоколот.

На следните слики се прикажани flow дијаграмите за OpenID Connect. Првата слика е општа , додека втората ги прикажува сите повици за тоа како OpenID Connect комуницира со клиентската апликација и авторизацискиот сервер.





Литература

https://developer.okta.com/docs/api/resources/oidc

https://nodejs.org/en/docs/

http://expressjs.com/en/api.html

https://github.com/okta/okta-sdk-nodejs

https://www.apachefriends.org/faq_windows.html

https://httpd.apache.org/docs/2.4/vhosts/examples.html

[Видео] https://www.youtube.com/watch?v=996OiexHze0

https://auth0.com/docs/protocols/oidc

https://openid.net/specs/openid-connect-core-1_0.html

https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml

https://docs.axway.com/bundle/APIGateway_762_OAuthUserGuide_allOS_en_HTML5/page/Content/OAuthGuideTopics/OpenidImport/openid_flow.htm