Oauth

Contents

Khái niệm

Oauth không phải là 1 Api hoặc 1 service . Nó là 1 chuẩn Open để xác thực và bất kì ai cũng có thể triển khai nó.

Cụ thể hơn, Oauth là một chuẩn cho các app có thể được sử dụng để cung cấp cho các client app với "secure delegate access". Oauth làm việc trên Https và xác thực các thiết bị, Các Api, các server và các ứng dụng với access token hơn là sử dụng chứng chỉ (credential).

Có 2 phiên bản của Oauth 1.a và 2.0 . Những kĩ thuật của 2 version này khác nhau hoàn toàn và không được sử dụng với nhau, nó thì cũng ko tương thích ngược luôn.

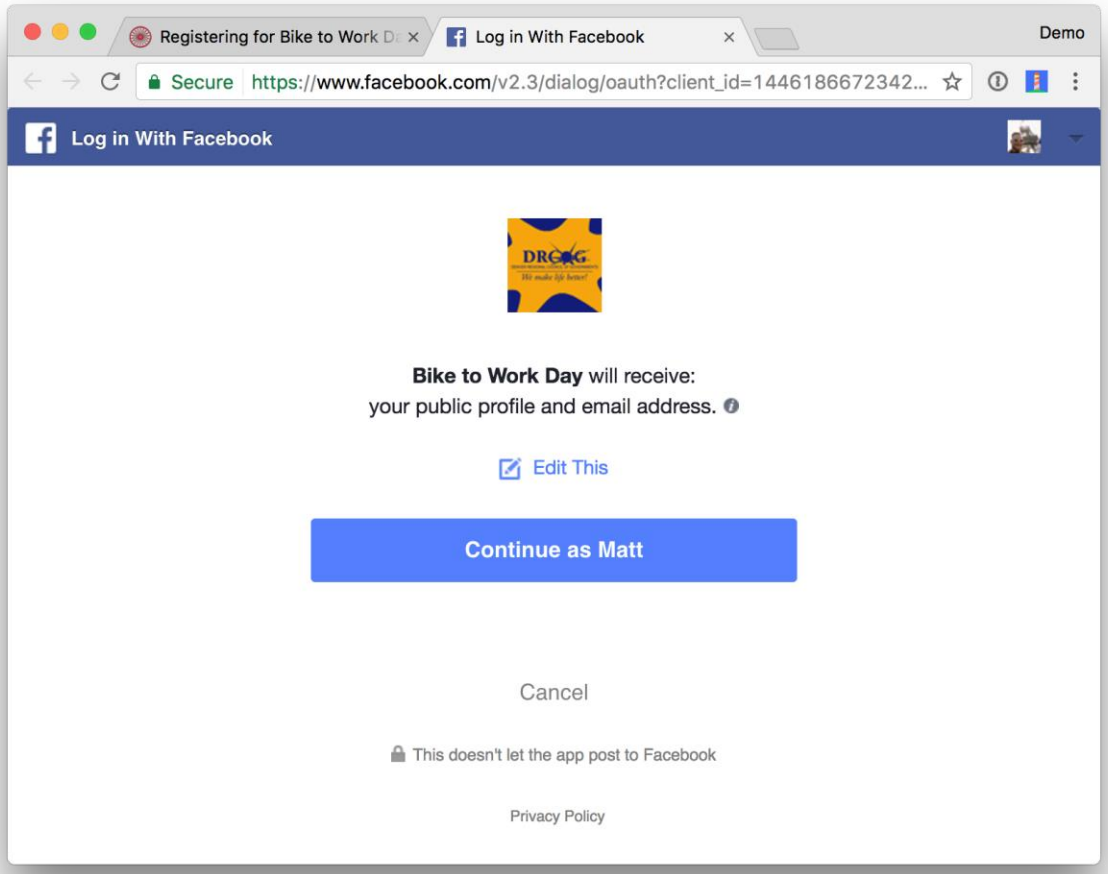
Vậy cái nào được sử dụng phổ biến? Ngày nay Oauth 2 được sử dụng rộng rãi . và ngay từ bây giờ bất kì nơi đâu tôi nói "Oauth" là đang nói về Oauth 2.0.

Oauth và ApIs

Có Sự thây đổi rất nhiều về việc build các APIs. Bây giờ hầu hết các develop đã chuyển sang Rest và stateless Apis . Tóm lại Rest là sử dụng giao thức Http chuyển Json trên network.

Các develop build nhiều Apis. Các API Economy là thông dụng và bạn có thể nghe nó ở bất cứ nơi đâu . các Company cần phải bảo vệ các Rest Apis theo cách cho phép nhiều thiết bị truy cập vào.

Đây à Oauth



Oauth là một ủy quyền xác thực framwork cho Rest/apis. Nó cho phép các app đạt được các quyền (scope) truy cập data của bạn mà không cần phải có password. Nó tách việc xác thực và hỗ trợ nhiều use . Hỗ trợ server to server , brower-based apps, mobile/native apps và consoles/TVs.

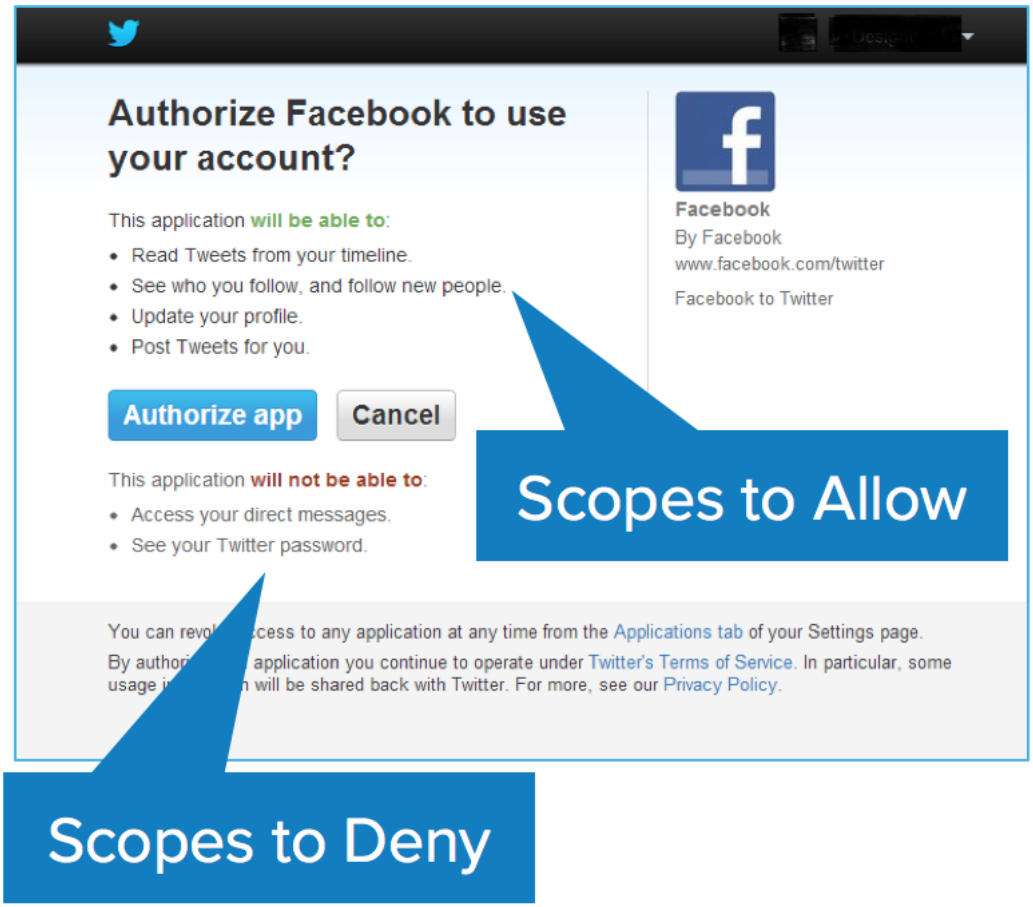
Các thành phần Chính

- Scope and consent (phạm vi và sự đồng ý)
- Actors (hành động)
- Client
- Token
- Authorization Server
- Flows

Scope and consent (phạm vi và sự đồng ý)

Scope là những gì bạn thấy xác thực trên màn hình khi mà 1 app yêu cầu Quyền hạn. Chúng là các gói quyền được yêu cầu khi mà client request 1 token. Những quyền này được code bởi các nhà phát triển khi viết code.

Bạn cũng có thể vào trang dashboard để xem ứng dụng nào bạn đã cấp quyền truy cập vào và thu hồi sự đồng ý (revoke consent)



Oauth Actor

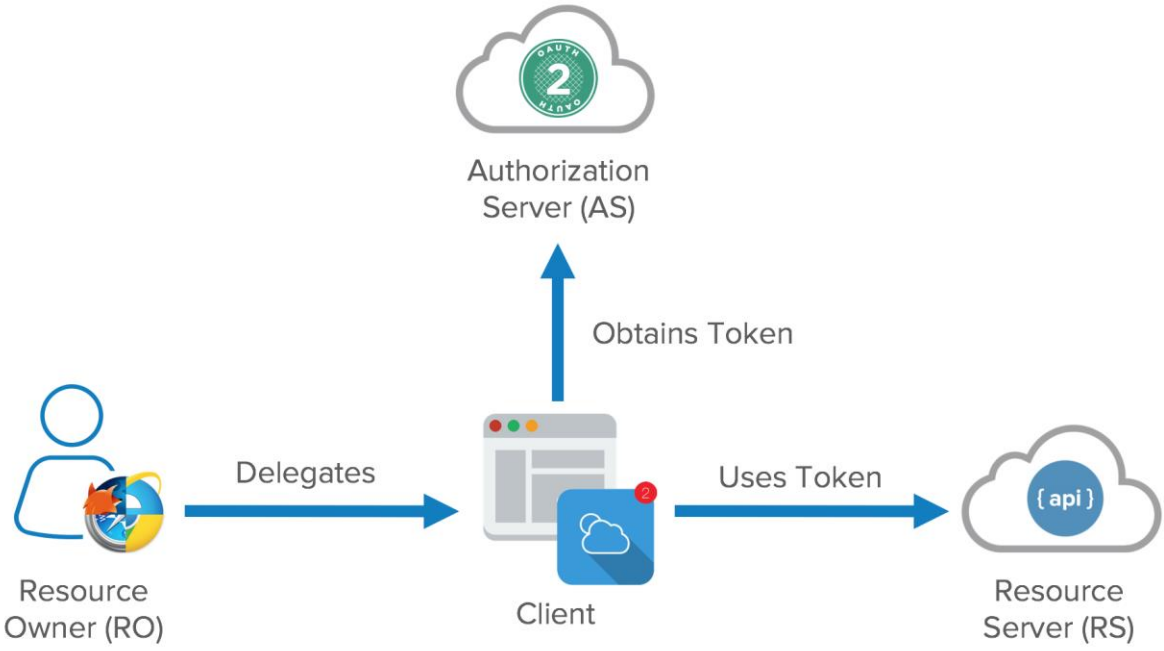
Các actors trong Oauth như sau

Resource Owner: Sở hưu data bên trong resource Owner. Vd: Tôi có resouce owner của tôi là Facebook profile.

Resource Server: Api nơi lưu trữ data application muôn truy cập

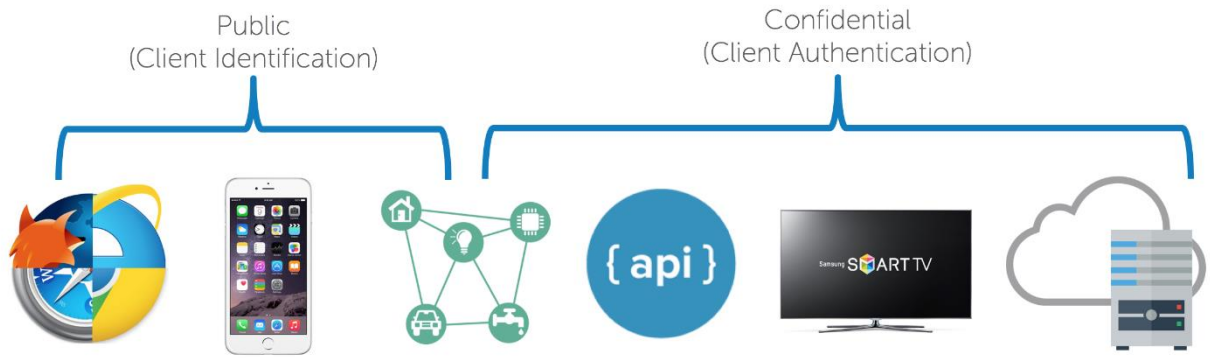
Client : Là ứng dụng bạn muốn truy cập vào

Authorization Server: Công cụ chính của Oauth (vd keyloak)



Resource Owner là vai trò nó có thể thay đổi với các chứng chỉ khác nhau có thể là enduser, hoặc là company.

Clients (Fontend): có thể là public hoặc bí mật (sử dụng secret key) . Nó có sự khác biệt khá lớn. Bí mật (confidential) clients có thể được tin tưởng để lưu secret key. Họ không chạy trên desktop hoặc thông qua app store. Chúng ta không thể đảo ngược chúng để get secret key. Họ cho chúng run tại nơi được bảo về enduser không thể truy cập được chúng.



Oauth Tokens

Access token là token mà client dùng để sử dụng để truy cập Resouce Server (API) . Token có thời gian sống theo phút giờ hoặc ngày. Chúng ta có thể get token bằng 2 cách dùng public hoặc confidential như đã nêu trên .

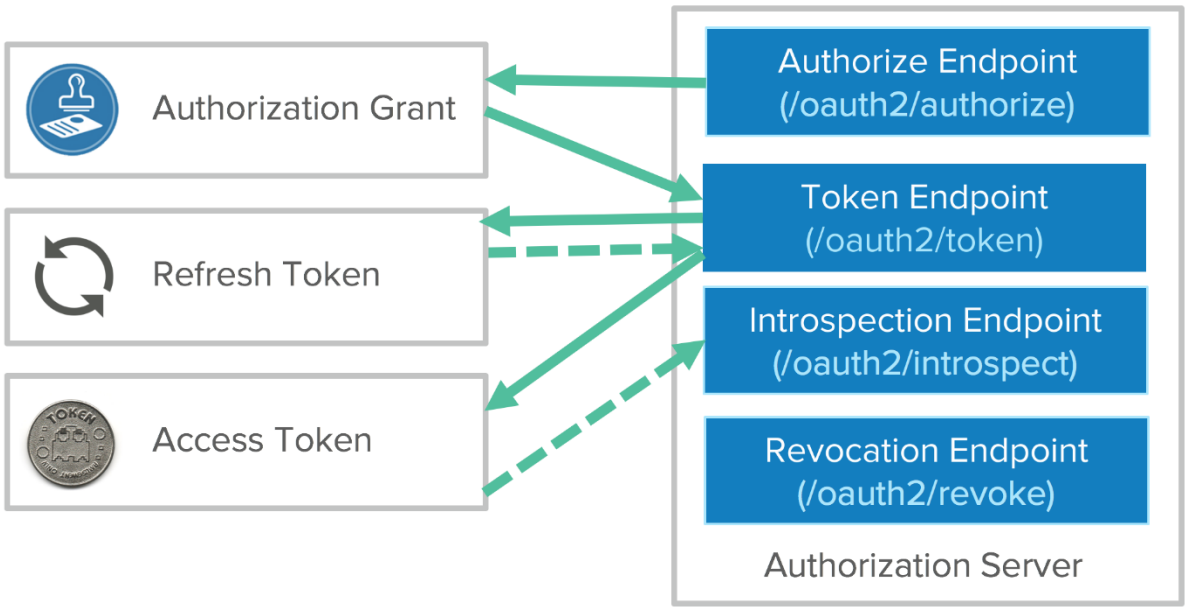
Còn 1 token khác đó là Refresh token. Cũng có thời gian tồn tại tương ứng với access-token tùy từng ứng dụng Oauth quy định. Như keyloak là chúng có thời gian sống giống nhau.


```
access_token:
"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJlUktVWG10TFhKMHBBNkxBS29aWko1ZlU0VDhCdmxKdERCb3pXanFFdnhjIn0.eyJleH
MmVmYS1hYjlmLTQ4MTUtYWE0Zi0yNjFhMzJkN2E0MjciLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODMvYXV0aC9yZWFsbXMvYmFlbGR1bmciLCJzdWI
XdDbGllbnQiLCJzZXNzaW9uX3N0YXRlIjoiYzA2MDNjYzItZmNmNi00NDgxLWEzY2MtYzJkMTgxZGQ5OWI2IiwiYWNyIjoiMSIsInNjb3BlIjoib3Blbml
cFPCziJoMrM3cRKsaEUZIYu00zRZ_S4ChKx5h3tuCCXCsCjxPLitz4wyFB_nCUbSLJuufGT_SZwLWUfC4krR5fi3e05UwdoLm9t2XHS2oHkdTHdP64JVQq
bMWIkOAo92dXpMxEvA2V6RyCIFBxOLELxjYwwgZXVLroLPB5WbeaM8-41QPUEQ9CW9sQndYklUm5_1J8Yeno9BFSlzjWoeyg8ChA5hhI-y-8B-1VzyecdRs
expires_in: 18000
id_token:
"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJlUktVWG10TFhKMHBBNkxBS29aWko1ZlU0VDhCdmxKdERCb3pXanFFdnhjIn0.eyJleH
NWQ5Zi1lMzRlLTQ4MTEtYTczNi0yMGE3YzU4MzBkNDUiLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODMvYXV0aC9yZWFsbXMvYmFlbGR1bmciLCJhdWC
iJJRCIsImF6cCI6Im5ld0NsaWVudCIsInNlc3Npb25fc3RhdGUiOiJjMDYwM2NjMi1mY2Y2LTQ0ODEtYTNjYy1jMmQxODFkZDk5YjYiLCJhdF9oYXNoIjoi
QuY29tIn0.gO3yytV9uKEM0Blw9mUU-
I3MEinXbolWY4IEvNtB6CI1QfMbHoWk0JnynGBHKSfXn2vayEaTl3ySIAJXzJmzQ2oxJAIkPmBAnCxMtLb6V8DBxfzVnDSUuUV7RSxKt3PjiVpehJc7KqyE
DKyegaUgbRYmnsR3UAHszgw4ciiZltmIIHiIN7VwEekXAh310sDHayrbrNm0dReojwV5UhMP2XNOExgsYFGoUDeezd0Xl7sA4NlicPIRZD0d6aAHS-2UTHU
"not-before-policy": 0
refresh_expires_in: 1800
refresh_token:
"eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIwZDkwY2JkNy03MTY0LTQyY2MtODhlMi1kMjE1ZTc5YWU4ZWEifQ.eyJleHAiOjE2Mjk
DM1IiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgzL2F1dGgvcmVhbG1zL2JhZWxkdW5nIiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgzL2F1dGgv
```


Ví dụ trên là 5 tiếng là 1800 giây. Dùng refresh token để get lại 1 access token mới khi nó đã expried hết hạn .

Có thể lưu mọi thứ trên payload của token bằng các claim của chúng

Tokens được lấy từ endpoint trên Authorization Server. Hai main endpoint chính là authorize endpoint và token endpoint. Chúng được tách biệt cho các trường hợp sử dụng khác nhau.



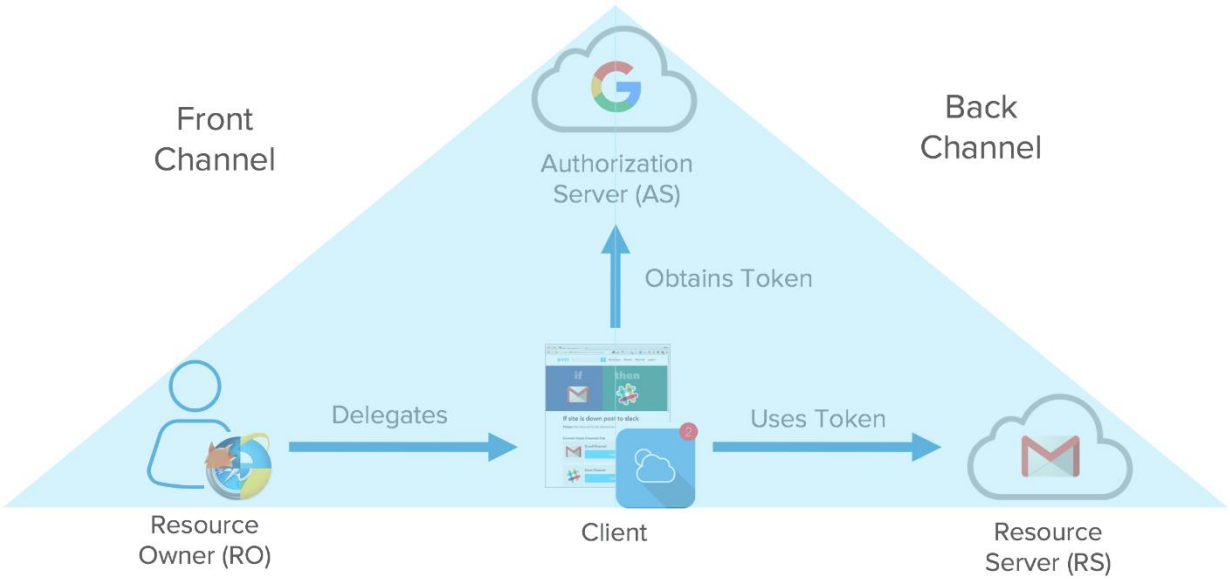
Bạn có thể sử dụng access token để truy cập vào Apis. Khi nó hết hạn, bạn sẽ quay lại token endpoint với refresh token để get mới 1 token access.

Front Back Channel

Để thực hiện được các hành động xác thực và cung cấp token thì xảy ra trên 2 chanel . font channel và back channel.

The font channel là những gì xảy ra trên browser. The brower chuyển hướng để user đi đến authorization server, the user nhận được sự chấp thuận. Sau khi user mang authorization grant về client thì client không cần dùng đến browser nữa trong qua trình yêu cầu cấp token.

Có nghĩa là token được tiêu thụ bởi client application nên nó có thể truy cập resources trên chính bản thân nó (kiểu thư gọi ajax vậy) . Chúng ta gọi back channel . the backchannel được gọi trực tiếp từ 1 http từ client đến resource server để trao đổi authorization grant cho token. Những channel được sử dụng vào các flow khác nhau dựa vào thiết bị mà bạn đang sử dụng



Cho ví dụ : Front channel flow nơi bạn xác thực thông qua user agent :

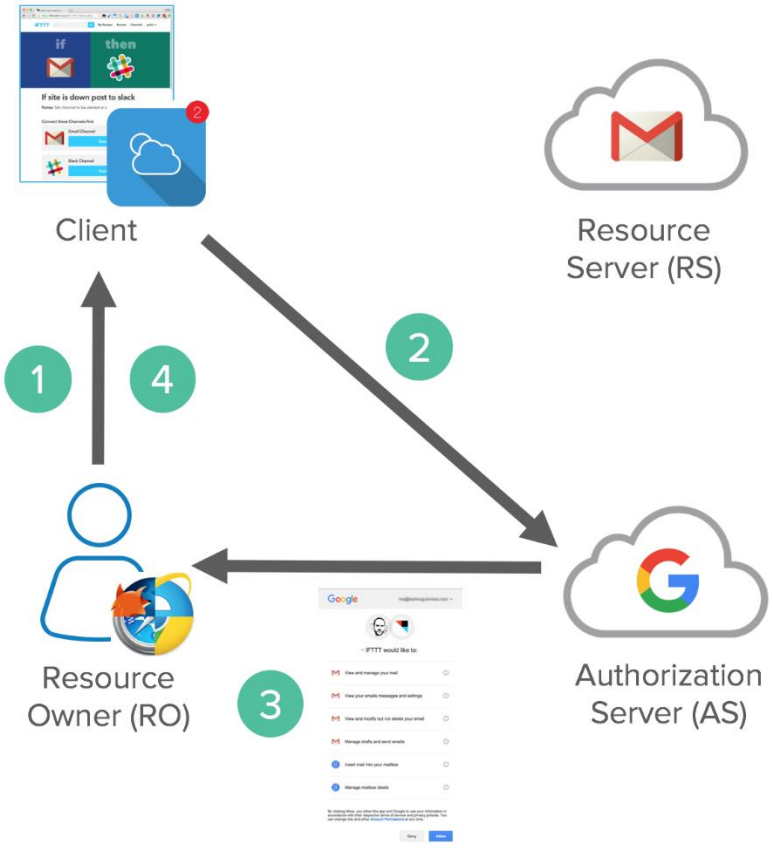
1 Resource Owner bắt đầu flow access đến resouce đang được bảo vệ

2 Client sends authorization request với các quyền scope mong muốn thông qua browser chuyển hướng đến Authorization Endpoint trên Authorization Server.

3 Authorization Server trả về 1 hợp thoại chấp thuận “Do you allow this application to have access to these scopes ?” “Bạn có cho phép ứng dụng này có quyền truy cập vào những phạm vi này không? ”

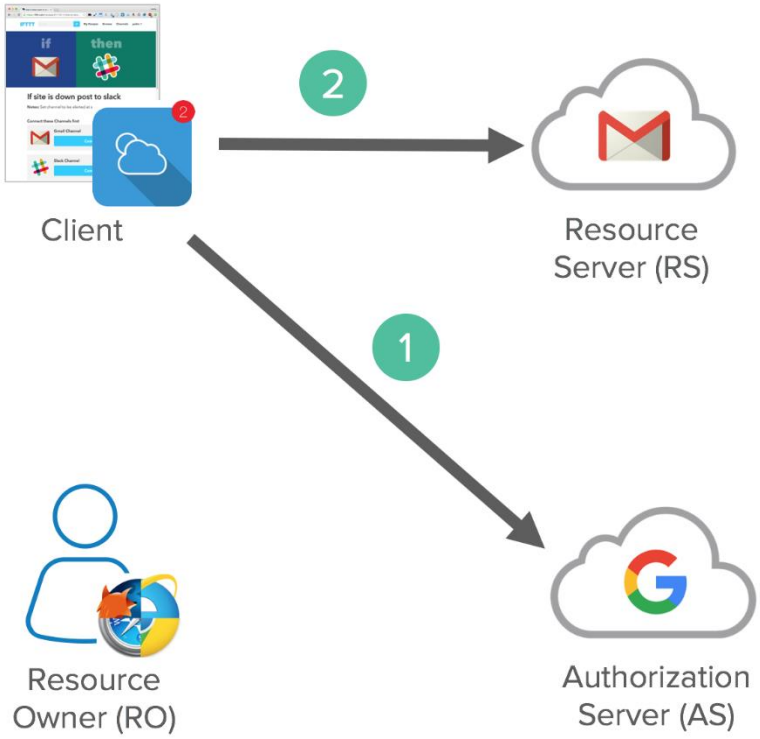
Dĩ nhiên bạn phải xác thực với application (client) này, nếu mà bạn chưa xác thực với Resouce server của bạn , nó sẽ bảo bạn login. Nếu bạn đã xác thực có trong cached session cookie, bản chỉ nhìn vào hợp thoại yêu cầu chấp thuận các scope quyền sẽ view lên và chấp thuận nó. Có nghĩa là chưa login sẽ login, nếu login từ trước rồi sẽ hiển thị hộp thoại view các quyền scope để bạn nhấn agree chấp nhận applicaton này sử dụng các quyền này lên Resouce server của bạn.

4 The authorization grant được truyền về application thông qua điều hướng trên brower. Đó là tất cả những gì trên Front channel.



Sau khi Front channel hoàn thành, Back channel xảy ra , trao đổi the authorization code cho 1 access token.

The Client application sends an access token request to the token endpoint on the Authorization Server with confidential client credentials and client id. This process exchanges an Authorization Code Grant for an Access Token and (optionally) a Refresh Token. Client accesses a protected resource with Access Token.



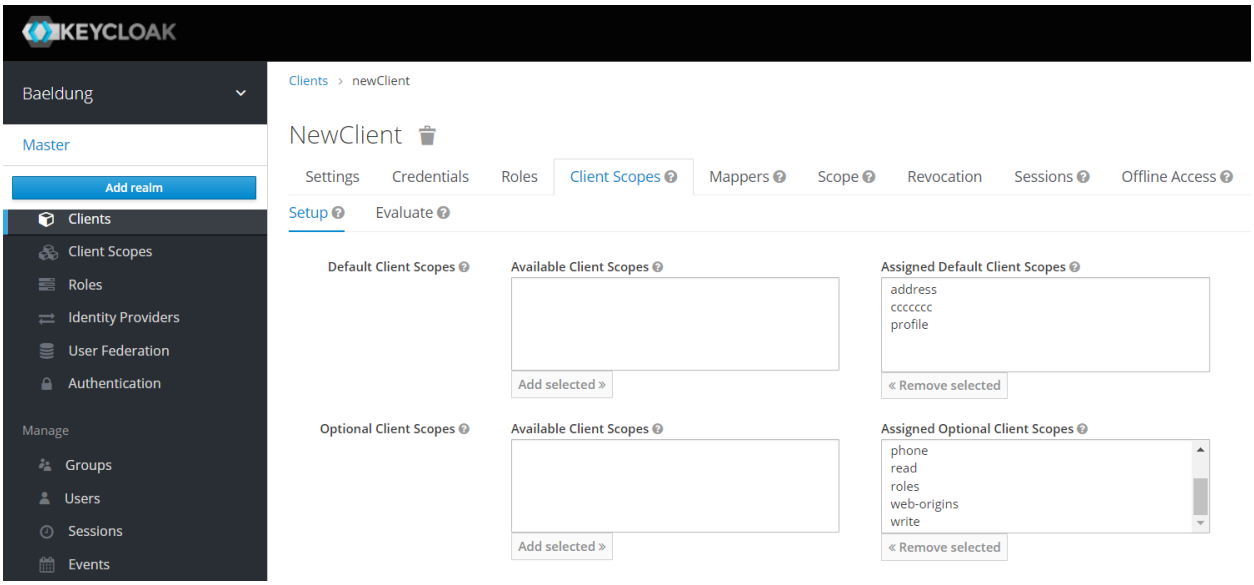
Giả Xác thực Với Oauth 2.0 Dùng OpenID

Login với Oauth đã trở nên phổ biến bởi Facebook và Twitter. Trong cái flow này, một client truy cập /me endpoint cùng với 1 access token. Tất cả chỉ là the client có truy cập resouce với 1 token. Nhiều người đã chế tạo ra endpoint fake lấy thông tin người khác với một access token . Không có chuẩn nào để get thông tin của user. Để giải quyết đều đó là dùng OpenID Connect. Nó sẽ tạo ra 1 id_Token cho client và một UserInfo endpoint để lấy user attribute. Khác với Saml , OIDC cung cấp 1 chuẩn set scope và claims cho việc định danh ví dụ: profile, email, address, and phone.

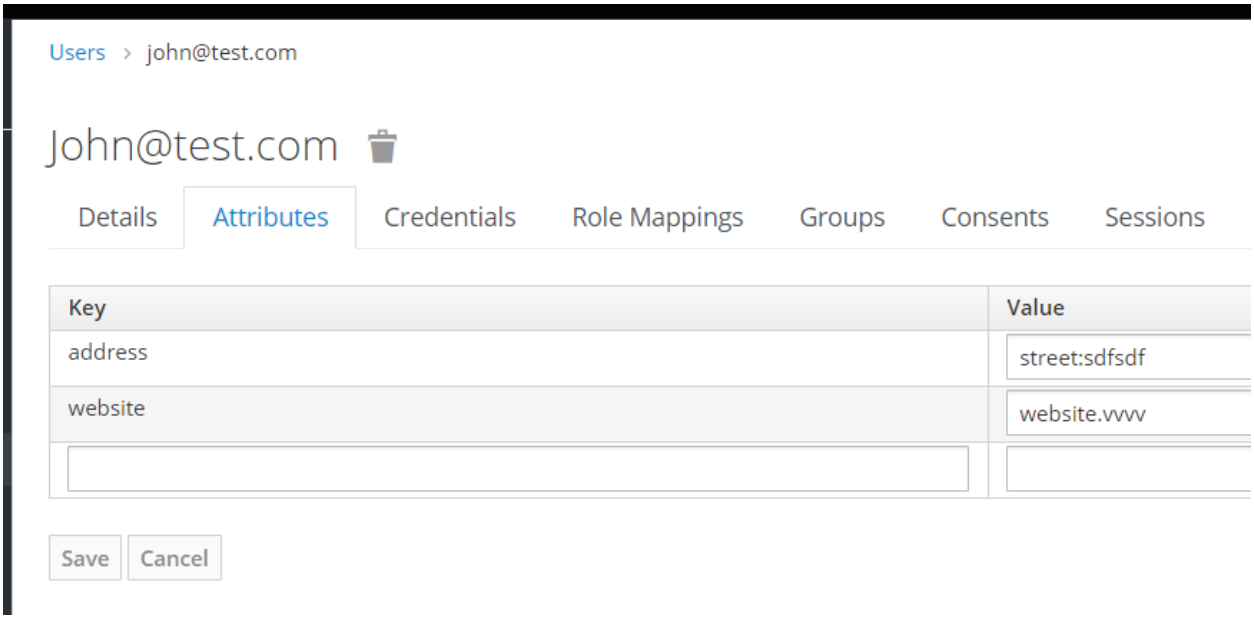
Set scope lúc yêu cầu page login:

```
login() {
    window.location.href =
    'http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/auth?response_type=code&&scope=openid%20read%20write&client_id=' +
    this._service.clientId + '&redirect_uri='+ this._service.redirectUri;
}
```

Yêu cầu scope option có trong cài đặt tự chọn của clients: openid read write

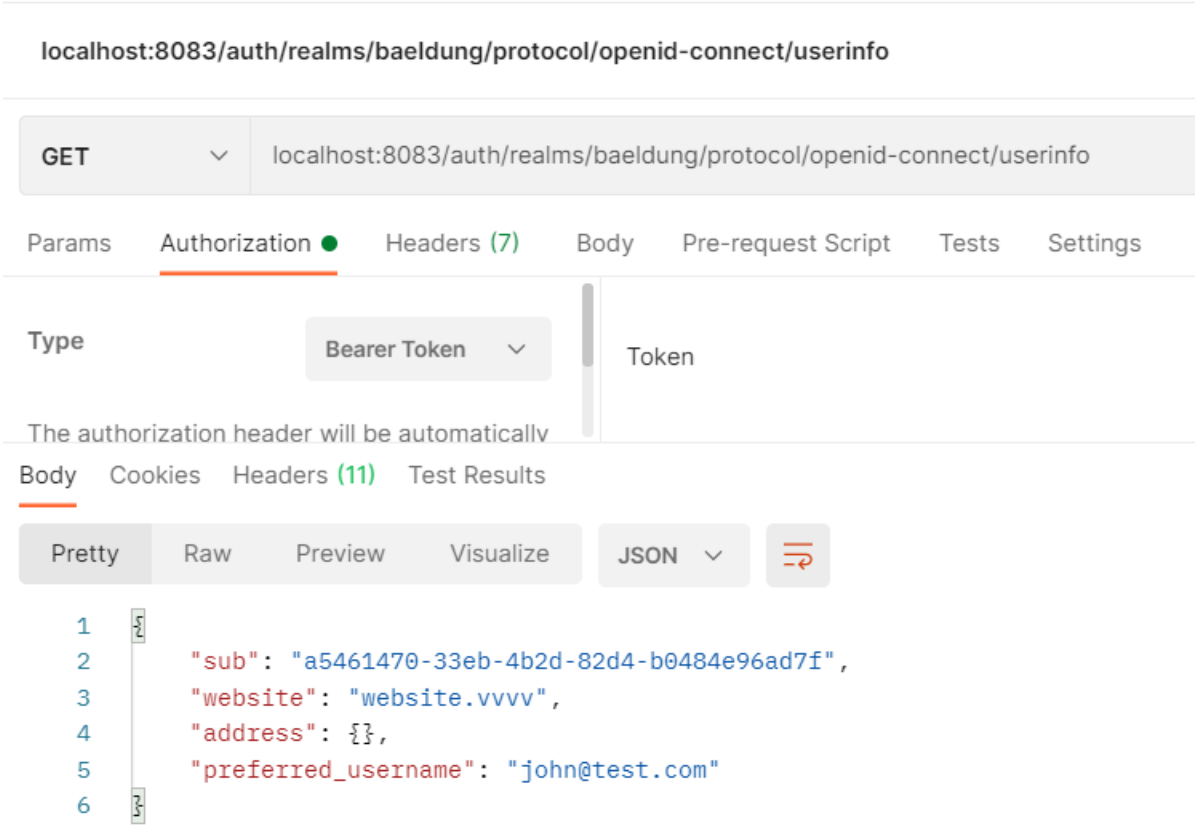


Set attribute cho user



Get thông tin user

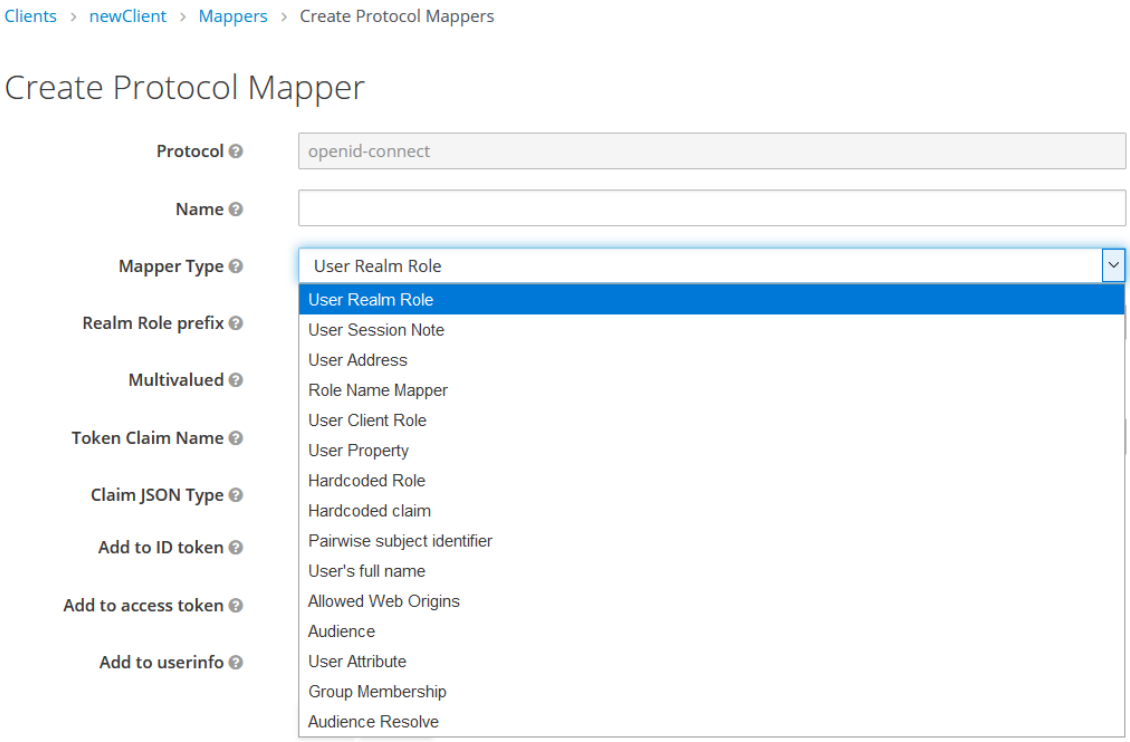
Get: localhost:8083/auth/realms/baeldung/protocol/openid-connect/userinfo



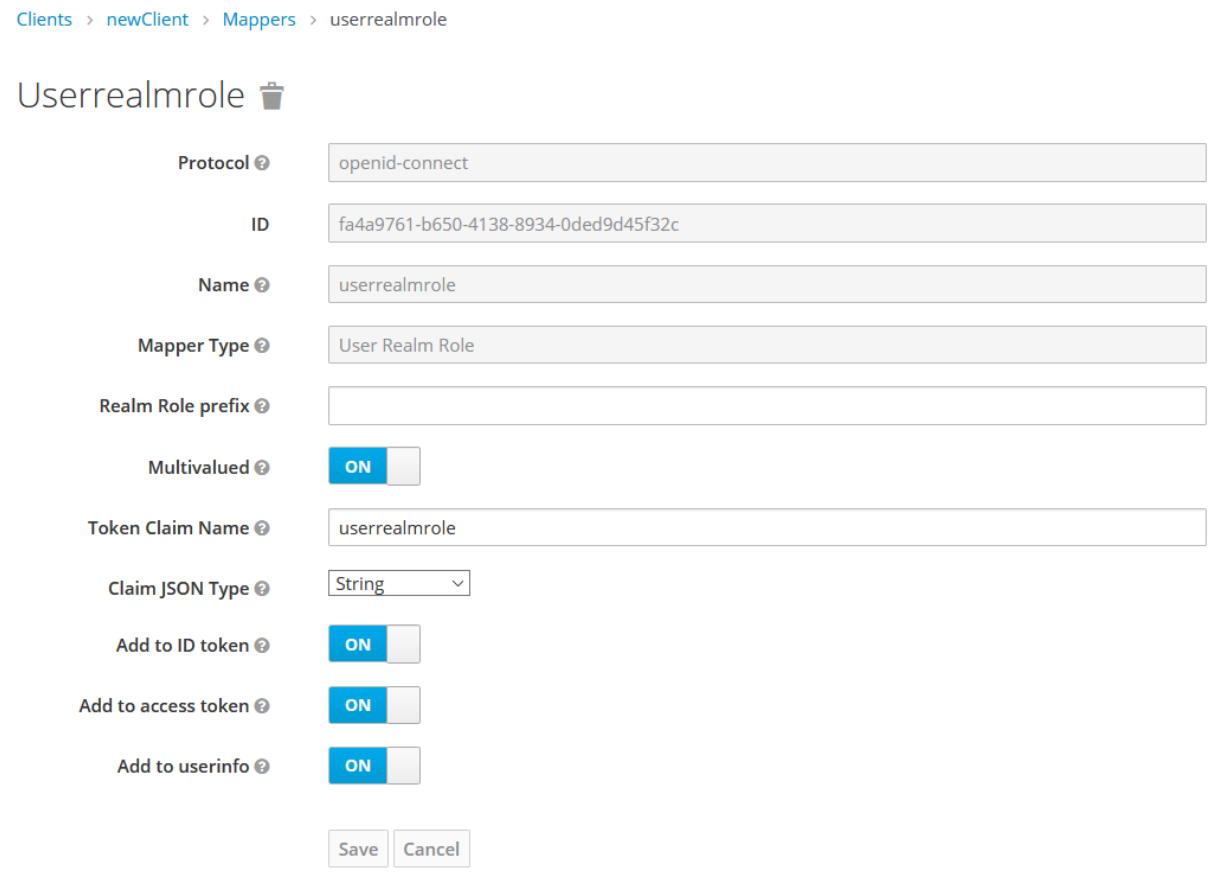
Custom Claims in the Token

Cấu hình trên app keyloak

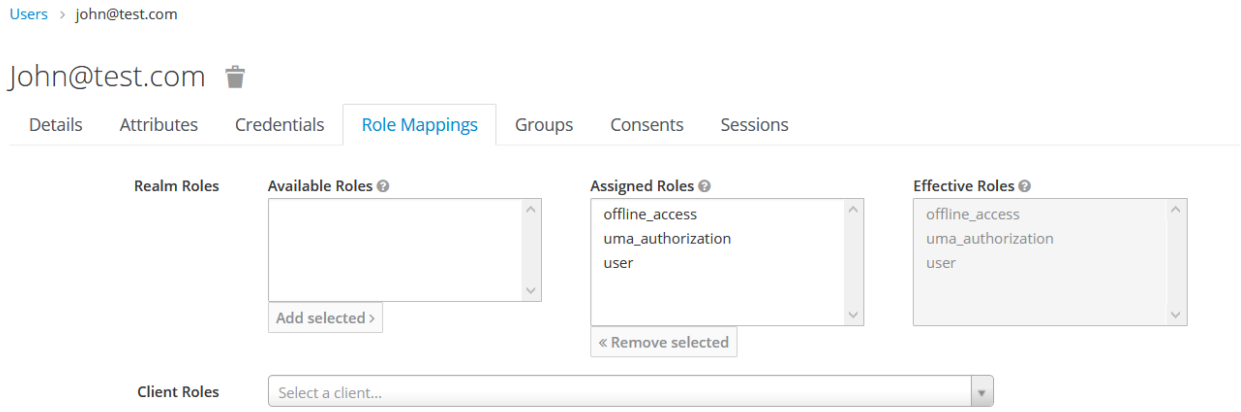
Nếu muốn thêm 1 claim và access token ta tạo ra 1 mapper ở menu Clients để đưa vào access token

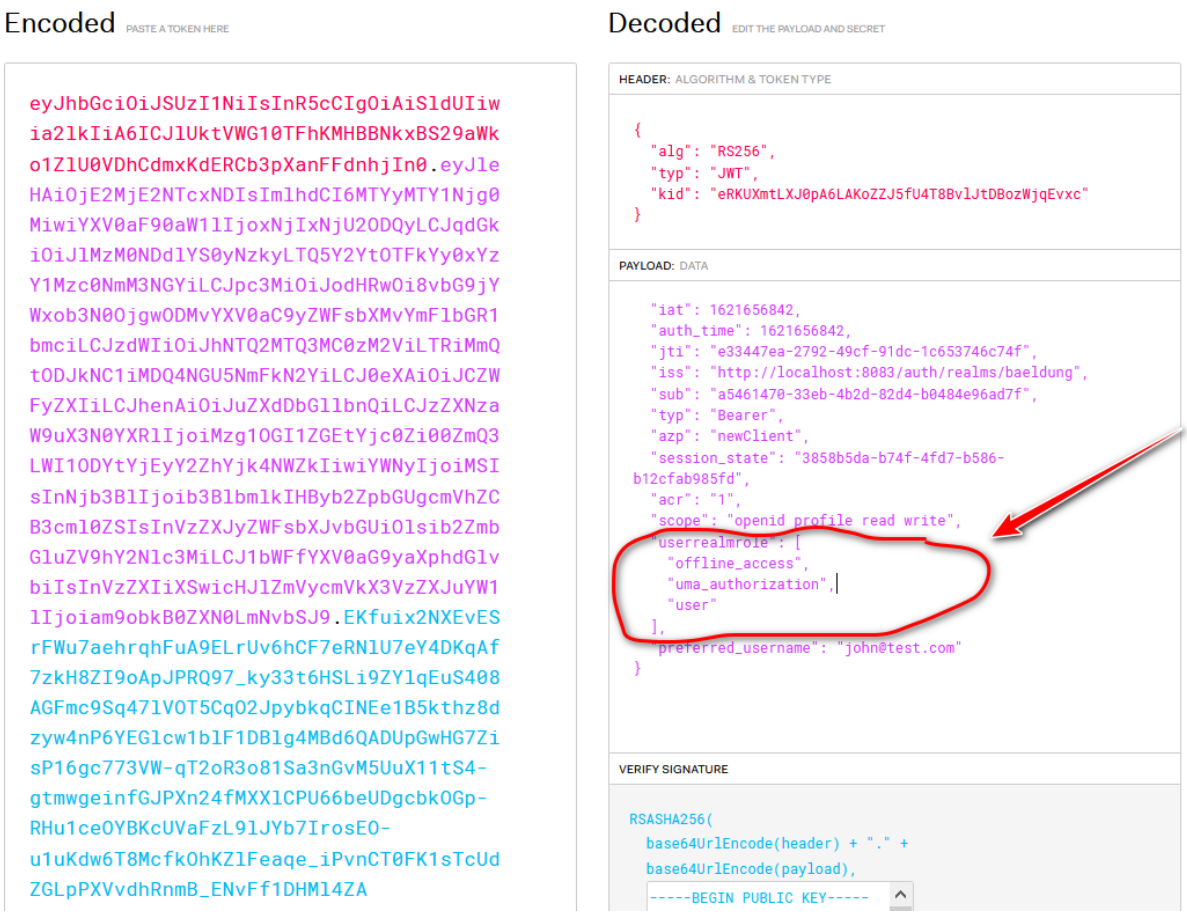


Ví dụ thêm User Realm Role là get các role được chỉ định của user mà application đang sử dụng “john@test.com”



Nó sẽ đưa role của ở user vào access token

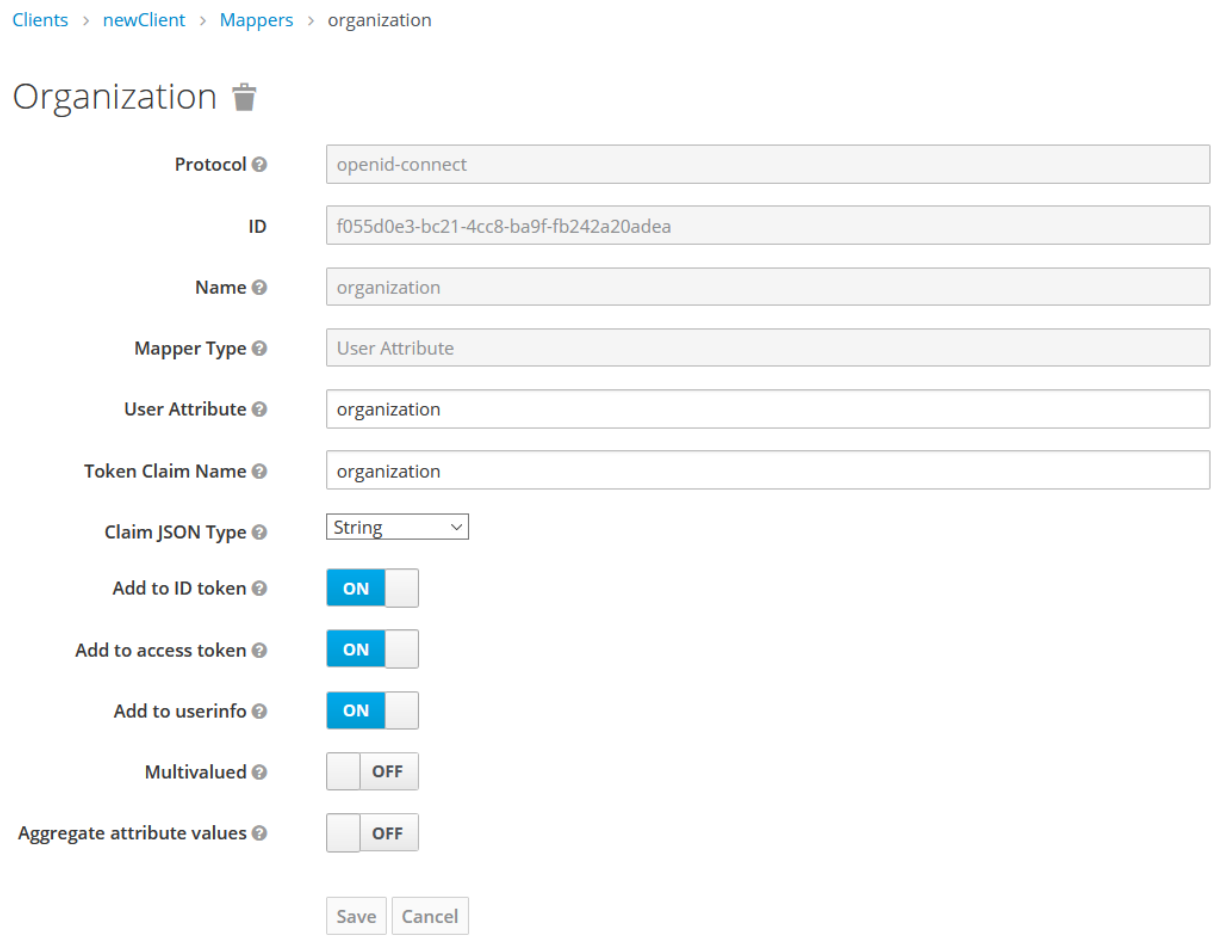




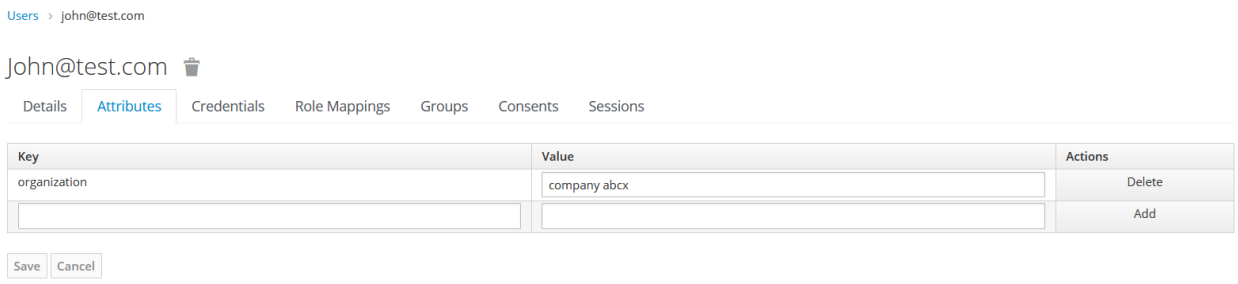
Còn nếu muốn đưa vào User attribute thì tạo ra Mappers Attribute mới trong Menu Clients hoặc là sử dụng các Mapper có sẵn trong keyloak Vd: mail, address, nicknam,...

Vd ta tạo mới 1 mapper type user attribute và sử dụng 1 cái Mapper có sẵn để đưa vào access token

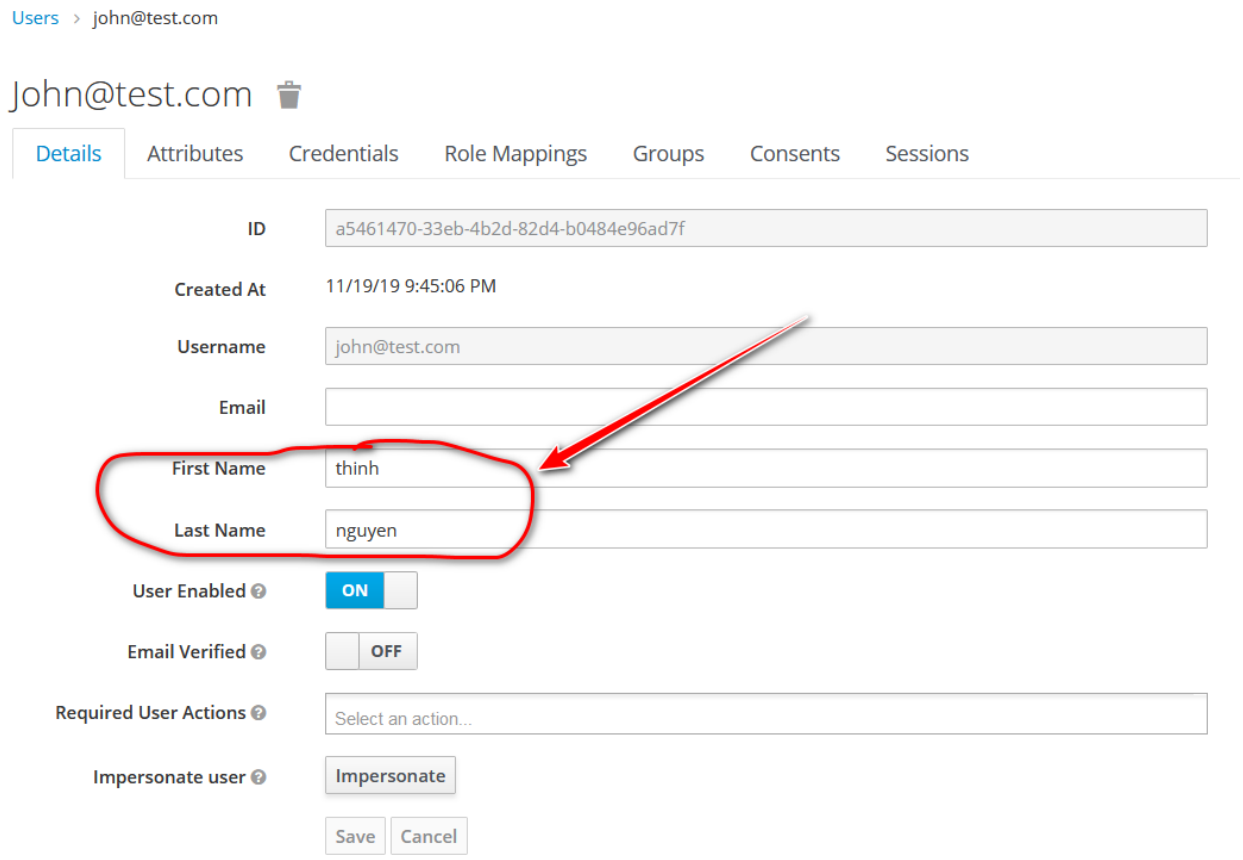
Tạo mapper mới



Thêm Attribute mới thêm vào Attribute của user đang login trên Application client (không phải user admin logint trên keyloak nhé)

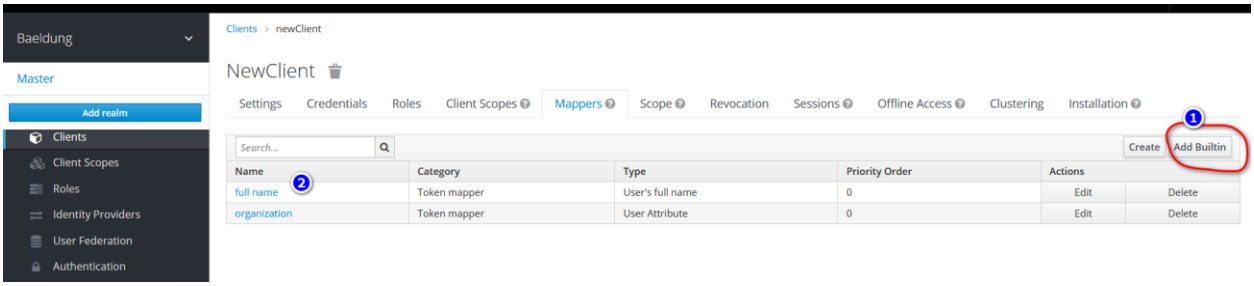


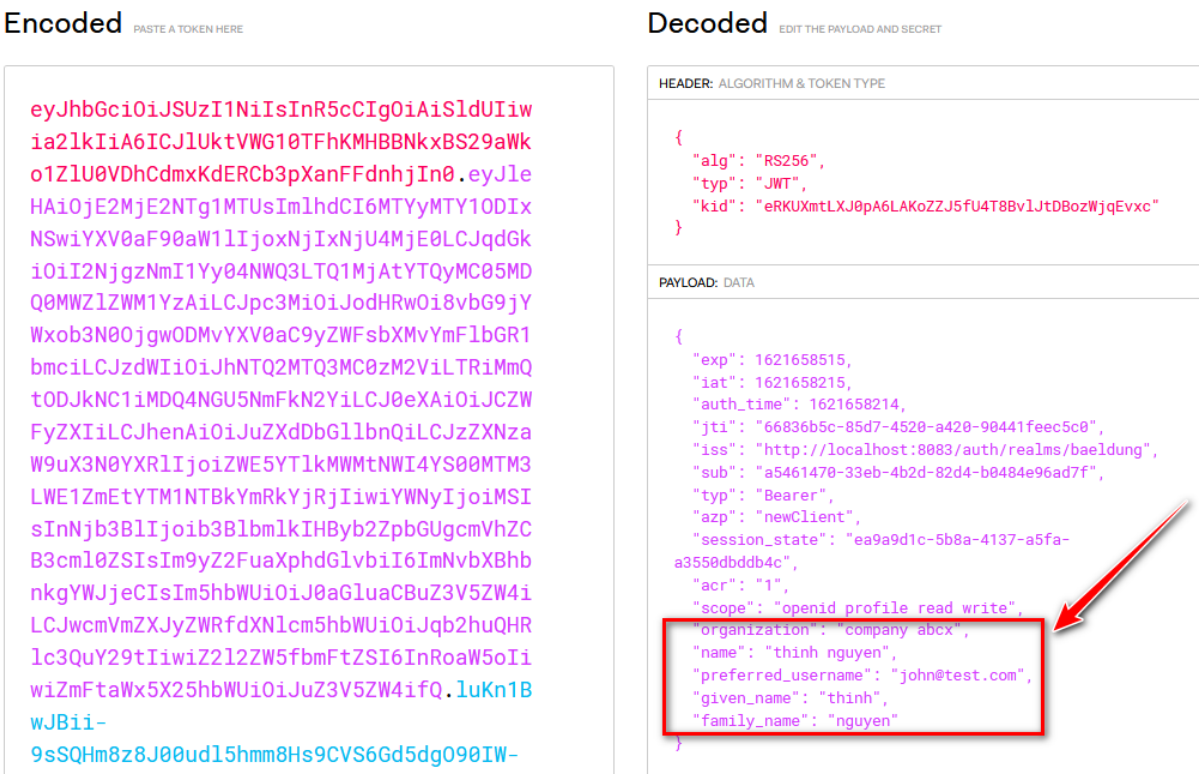
Sử dụng Mapper User Property có sẵn là full name để get first name và last name user đang login



Xong setting trên Clients để đưa vào trong các mapper vào trong newClient để trả về cho app client access token chứa các mapper này

Chọn 1 để ta add User mapper có sẵn là full name.





Cấu hình bằng file.json lúc start app authorization server

Ta có thể cấu hình sẵn trong App keyloak rồi export ra file.json đưa vào Spring boot.

Lưu ý là keyloak không export được users nên nếu có thể add user từ Code của server Authorization luôn nhưng nhớ add user sau khi đã cài đặt Realm nếu không nó sẽ tạo user trong realm admin

Ta cũng có thể thêm 2 mapper này vào mapper của newClient trong Clients:

```
656       "protocolMappers": [
657         {
658           "id": "6c10b4d2-4f69-4229-b62f-380919b38d7a",
659           "name": "full name",
660           "protocol": "openid-connect",
661           "protocolMapper": "oidc-full-name-mapper",
662           "consentRequired": false,
663           "config": {
664             "id.token.claim": "true",
665             "access.token.claim": "true",
666             "userinfo.token.claim": "true"
667           }
668         },
669         {
670           "id": "f055d0e3-bc21-4cc8-ba9f-fb242a20adea",
671           "name": "organization",
672           "protocol": "openid-connect",
673           "protocolMapper": "oidc-usermodel-attribute-mapper",
674           "consentRequired": false,
675           "config": {
676             "userinfo.token.claim": "true",
677             "user.attribute": "organization",
678             "id.token.claim": "true",
679             "access.token.claim": "true",
680             "claim.name": "organization",
681             "jsonType.label": "String"
682           }
683         }
684       ],
685       "defaultClientScopes": [
689       "optionalClientScopes": [
700     },
```


"protocolMappers": [

{

"id": "6c10b4d2-4f69-4229-b62f-380919b38d7a",

"name": "full name",

"protocol": "openid-connect",

"protocolMapper": "oidc-full-name-mapper",

"consentRequired": false,

"config": {

"id.token.claim": "true",

"access.token.claim": "true",

"userinfo.token.claim": "true"

}

},

{

```
    "id": "f055d0e3-bc21-4cc8-ba9f-fb242a20adea",
    "name": "organization",
    "protocol": "openid-connect",
    "protocolMapper": "oidc-usermodel-attribute-mapper",
    "consentRequired": false,
    "config": {
      "userinfo.token.claim": "true",
      "user.attribute": "organization",
      "id.token.claim": "true",
      "access.token.claim": "true",
      "claim.name": "organization",
      "jsonType.label": "String"
    }
  }
],
```

Configuration to Add/Remove/Rename Claims Trên Resouce Server

GetToken từ Controller

Có 2 đều lưu ý là mới chỉnh lại `antMatcher("/**")` như thế này nó mới bao gồm hết tất cả các request nếu không thì security sẽ từ chối request cứ báo là Cors sai (lỗi thật sự là không có filter security nào)

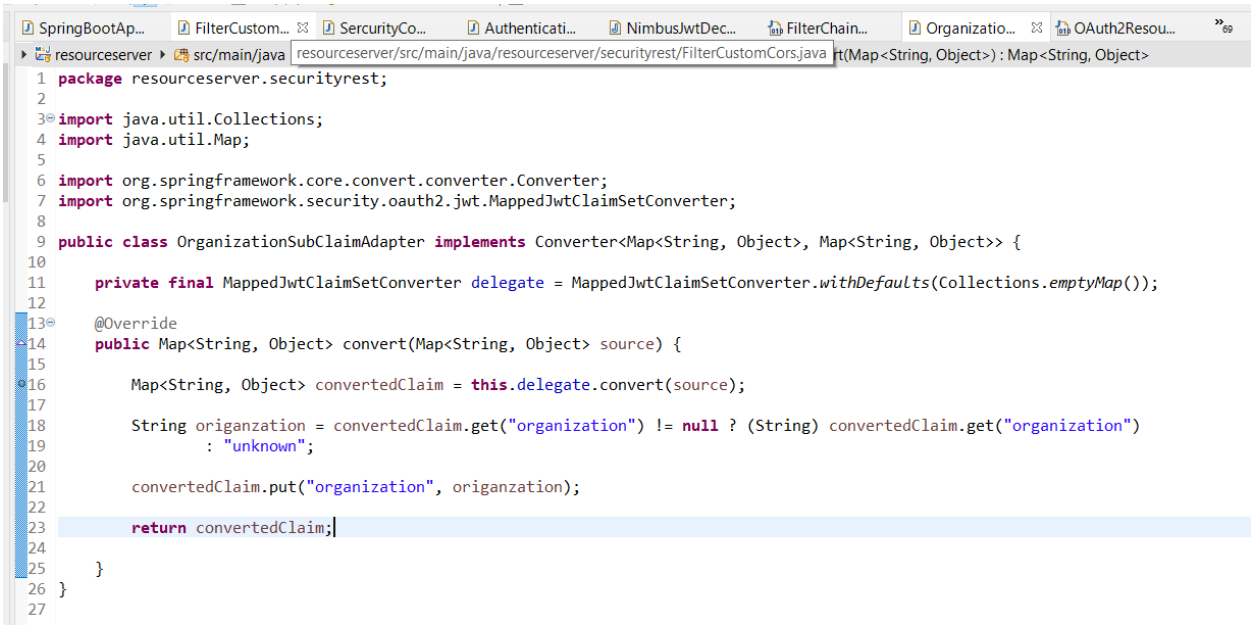
Thứ 2 là có thể xảy ra exception expried token .

```java

@RestController
@RequestMapping("/user")
public class AuthenticationServerClaims {


    @GetMapping("/info")
    Map<String, String> getUsernameWithOrigrantion(
            @AuthenticationPrincipal Jwt pricial) {

        if(pricial == null) {
            return Collections.singletonMap("No", "no");
        }
        Map<String, String> mapStr = new HashMap<String, String>();
        mapStr.put("user_name", pricial.getClaimAsString("preferred_username"));
        String origanzationStr = pricial.getClaim("organization");
        mapStr.put("organzation", origanzationStr);
        return mapStr;
    }
}

```

Configuration to Add/Remove/Rename Claims

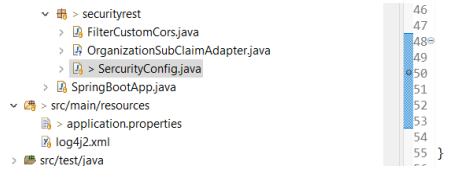
Ta chỉ cần custom lại hàm covert của JwtDecoder là ta có thể chỉnh sửa thêm xóa vv

Implements interface Convert của spring framwork



```java
package resourceserver.securityrest;

import java.util.Collections;
import java.util.Map;

import org.springframework.core.convert.converter.Converter;
import org.springframework.security.oauth2.jwt.MappedJwtClaimSetConverter;

public class OrganizationSubClaimAdapter implements Converter<Map<String, Object>, Map<String, Object>> {

    private final MappedJwtClaimSetConverter delegate = MappedJwtClaimSetConverter.withDefaults(Collections.emptyMap());

    @Override
    public Map<String, Object> convert(Map<String, Object> source) {

        Map<String, Object> convertedClaim = this.delegate.convert(source);

        String origanzation = convertedClaim.get("organization") != null ? (String) convertedClaim.get("organization")
                : "unknown";

        convertedClaim.put("organization", origanzation);

        return convertedClaim;

    }
}
```

Tạo ra Bean trong config sercutiry để nhúng convert vào JwtDecoder.



```
securityrest
  FilterCustomCors.java
  OrganizationSubClaimAdapter.java
  SercurityConfig.java
  SpringBootApp.java
src/main/resources
  application.properties
  log4j2.xml
src/test/java
```

```java
    }

    @Bean
    public JwtDecoder customJwtDecode(OAuth2ResourceServerProperties properties) {
        NimbusJwtDecoder jwtDecoder = NimbusJwtDecoder.withJwkSetUri(properties.getJwt().getJwkSetUri()).build();
        jwtDecoder.setClaimSetConverter(new OrganizationSubClaimAdapter());
        return jwtDecoder;
    }

}
```

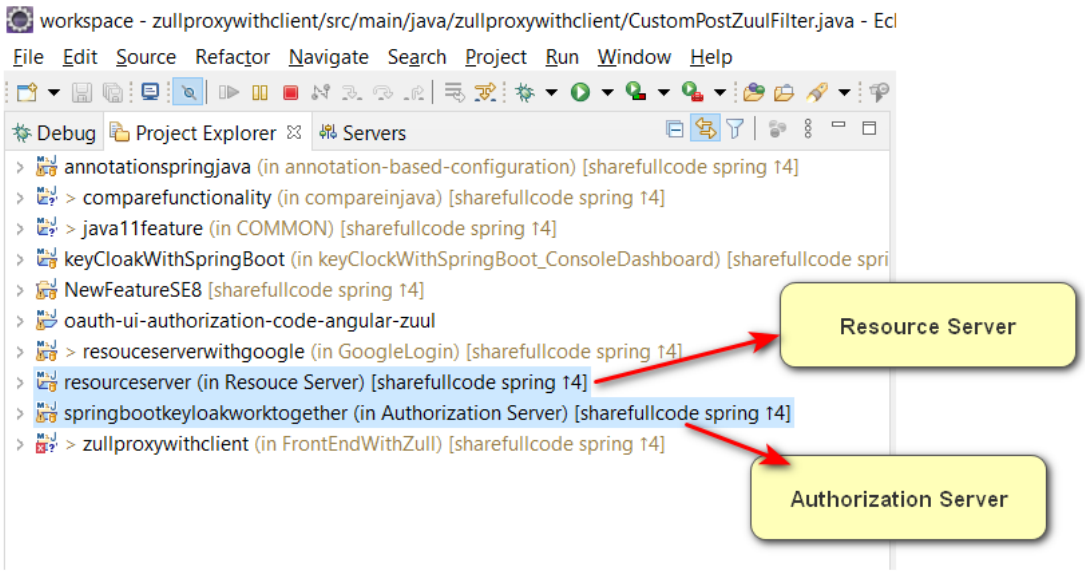
Ví Dụ 1 Cơ bản

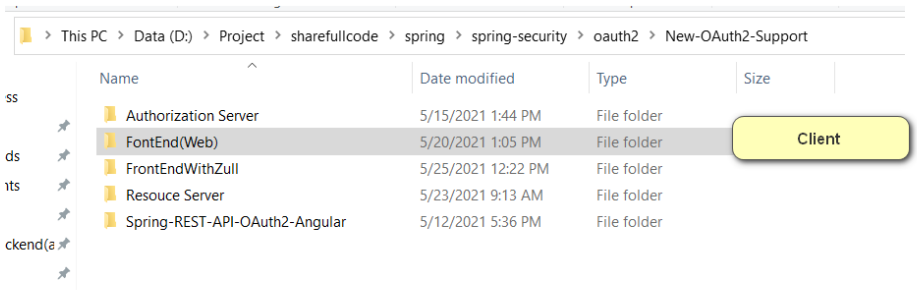
Tổng quan:

Client: Dùng angular 12 làm FrontEnd để gọi CRUD (localhost:8089)

Resource Server : Dùng spring boot + RestContoller + Customize Cors. (localhost: 8081)

Authorization Server: Dùng spring boot + Keyloak + cấu hình file .json (localhost:8083)





Flow:

Client nhấn login:

1) Sẽ gọi lấy code từ authorization server .

```
login() {
    window.location.href =
     'http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/auth?response_type=code&&scope=openid%20read%20write&client_id=' +
     this._service.clientId + '&redirect_uri='+ this._service.redirectUri;

     // login thanh công sẽ đi đến redirect_uri (http://localhost:8089) chính là page này luôn lúc này sẽ nhảy vào ngOnInit để lấy token
     // http://localhost:8089/ là trang home do app.module.ts quy định
    }
```

2) Authorization kiểm tra nếu login rồi sẽ trả code về nếu ko bắt buộc đăng nhập nhập ,Authorization server trả về code
3) Client sẽ gọi ajax để lấy access token từ Authorization Server bằng code đó.

```
ngOnInit(){
    this.isLoggedIn = this._service.checkCredentials();
    let i = window.location.href.indexOf('code');
    console.log("code: "+window.location.href);
    if(!this.isLoggedIn && i != -1){
        this._service.retrieveToken(window.location.href.substring(i + 5));
    }
}
```

```
retrieveToken(code:any){
  let params = new URLSearchParams();
  params.append('grant_type','authorization_code');
  params.append('client_id', this.clientId);
  params.append('client_secret', 'newClientSecret');
  params.append('redirect_uri', this.redirectUri);
  params.append('code',code);

  let headers = new HttpHeaders({'Content-type': 'application/x-www-form-urlencoded; charset=utf-8'});
  this._http.post('http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/token', params.toString(), { headers: headers })
  .subscribe(
    data => { this.saveToken(data); console.log(data)},
    err => alert('Invalid Credentials')
  );
}
```

4) Xong lưu access token, idtoken, refresh token vào Cookie.

```
saveToken(token:any){
  var expireDate = new Date().getTime() + (1000 * token.expires_in);
  Cookie.set("access_token", token.access_token, expireDate);
  Cookie.set("id_token", token.id_token, expireDate);
  var expireDateRefresh_token = new Date().getTime() + (1000 * token.refresh_expires_in);
  Cookie.set("refresh_token", token.refresh_token, expireDateRefresh_token);
  console.log('Obtained Access token');
  window.location.href = 'http://localhost:8089';
  this.getPropertyiInToken("organization ");
}
```

5) Lấy dữ liệu từ Resource Server thì gửi kem theo access token trong heaker trường beaber token.

```
getResource(resourceUrl:any) : Observable<any>{

  var headers = new HttpHeaders({'Content-type': 'application/x-www-form-urlencoded; charset=utf-8'
  , 'Authorization': 'Bearer '+Cookie.get('access_token')});

  return this._http.get(resourceUrl, { headers: headers }).pipe(
     catchError((err) => {
       console.log('error caught in service')
       console.error(err);

       //Handle the error here

       return throwError(err);    //Rethrow it back to component
     })
   )
}
```

6) Refresh Token để lấy access token mới từ Authorization

```
refreshToken(){
  let refresh_token = Cookie.get('refresh_token');
  let params = new URLSearchParams();
  params.append('grant_type','refresh_token');
  params.append('client_id', this.clientId);
  params.append('client_secret', 'newClientSecret');
  params.append('refresh_token', refresh_token);

  let headers = new HttpHeaders({'Content-type': 'application/x-www-form-urlencoded; charset=utf-8'});
  return this._http.post('http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/token', params.toString(), { headers: headers })
  .pipe(
    map(data =>{
      console.log(data);
      this.saveToken(data);
      return data;
      }),
    catchError(
      err => {
        console.log(err);
        return throwError(err);
      })
  )
}
```

7) Login thì gửi idtoken lấy từ cookie ra lên Authorization và xóa tất cả token có trong cookie.

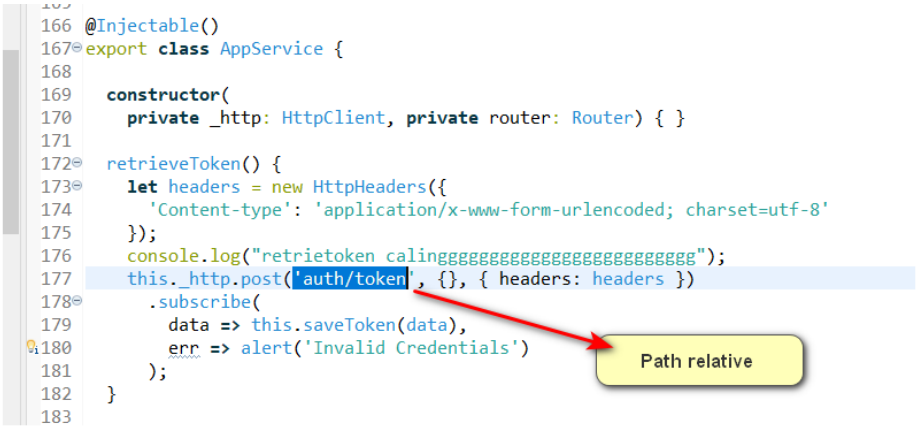
```
logout() {
  let token = Cookie.get('id_token');
  Cookie.delete('access_token');
  Cookie.delete('id_token');
  let logoutURL = "http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/logout?id_token_hint="
    + token
    + "&post_logout_redirect_uri=" + this.redirectUri;

  window.location.href = logoutURL;
}
```

Oauth Client + Zuul proxy

Khi truy cập từ Client đến Authorization ta cần phải truyền các giá trị mật nên cần phải che dấu nó đi , ở đây sử dụng Zuul của Spring Cloud ở phí client làm nhiệm vụ xử lí việc truyền para vào các request đế server , và nhận dữ mật lưu vào Cookie có mã hóa

Có 2 phần: Client vẫn là Angular , Zuul Filter cùng nằm chung port 8089 nằm chung 1 server trên cùng 1 spring boot

Client ta bỏ phần truyền param và bỏ phần truyền path chính xác cho nó chỉ cần **truyền các path relative** như thế này , và không còn truyền para như các bình thường nữa.



Phần tiếp theo nữa là ta phải chỉnh file anglar.json để khi build build đúng vị trí mà có thể tích hợp cùng với Zuul Spring boot chạy cùng 1 port

Mặc định là giốnh như thế này "outputPath": "dist/oauthApp" nó sẽ build ra dist thay vì vào thư mục của springboot

Phần Zuul

Thì ta chỉ cần file config route của nó

application.yml

```yaml
server:
    port: 8089
logging:
    level:
       org:
         springframework : debug
zuul:
  routes:
    auth/code:
      path: /auth/code/**
      sensitiveHeaders:
      url: http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/auth
    auth/token:
      path: /auth/token/**
      sensitiveHeaders:
      url: http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/token
    auth/refresh/revoke:
      path: /auth/refresh/revoke/**
      sensitiveHeaders:
      url: http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/logout
    auth/refresh:
      path: /auth/refresh/**
      sensitiveHeaders:
      url: http://localhost:8083/auth/realms/baeldung/protocol/openid-connect/token
    auth/redirect:
      path: /auth/redirect/**
      sensitiveHeaders:
      url: http://localhost:8089/
    auth/resources:
      path: /auth/resources/**
      sensitiveHeaders:
      url: http://localhost:8083/auth/resources/

  Servlet30WrapperFilter:
    pre:
      disable:true
```

Nếu truy cập các path nào ko có trong route thì nó cứ chạy qua thôi ko ảnh hưởng ví dụ như là truy tru cập lấy Data từ Resouce Server.

http://localhost:8081/resource-server/api/student

Cứ truy cập thỏi mấy chỉ những các path đó mới rơi vào route và được điều chỉ từ proxy Zuul.

Filter của Zuul

Sẽ có pre và post còn nữa nhưng ta sử dụng 2 loại filter này để:

Pre sẽ có nhiệm vụ chuyển đổi path relative thành path cố định , và truyền para cần thiết cho nó.

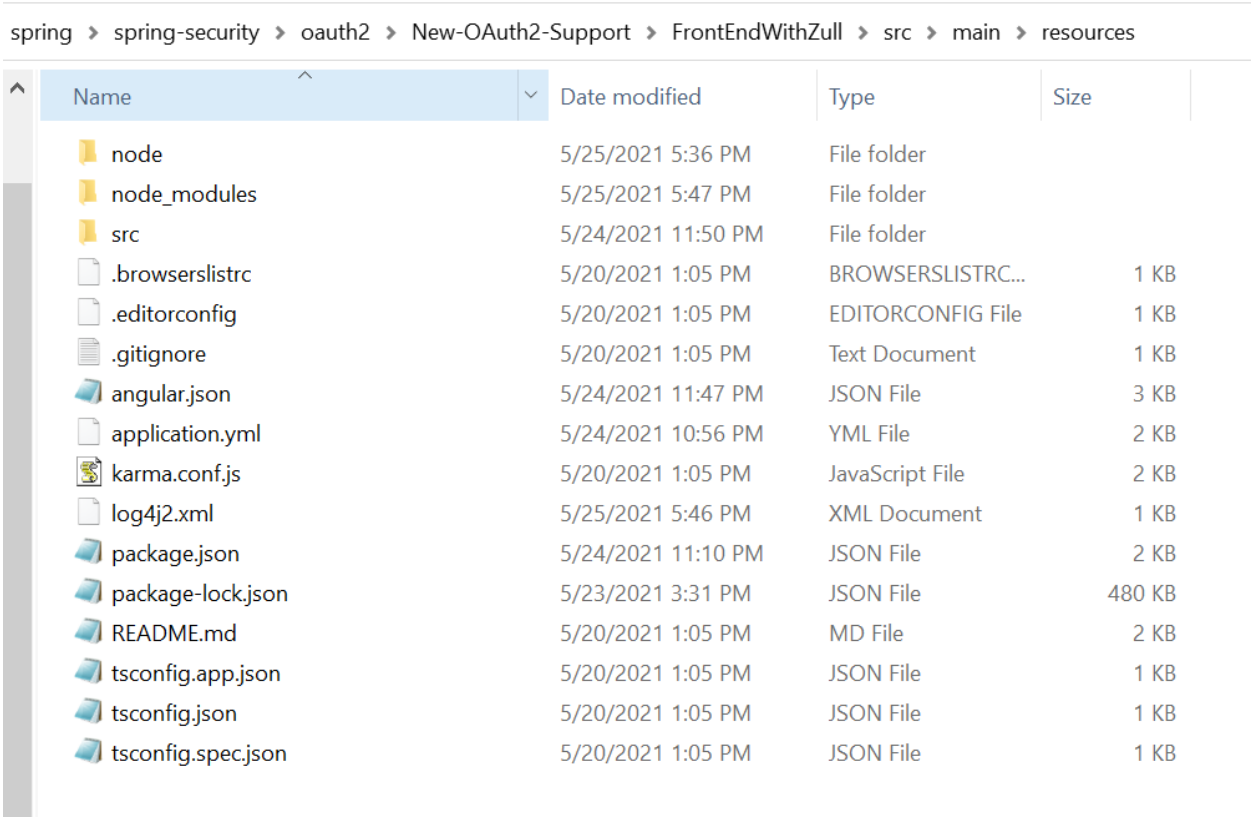
Post sẽ có nhiệm vụ nhận dữ liệu bảo mật từ Authorization để lưu vào Cookie . vd: code, accesstoken, refresh token, và cũng có nhiệm vụ xó chúng luôn khi login hoặc refresh token .

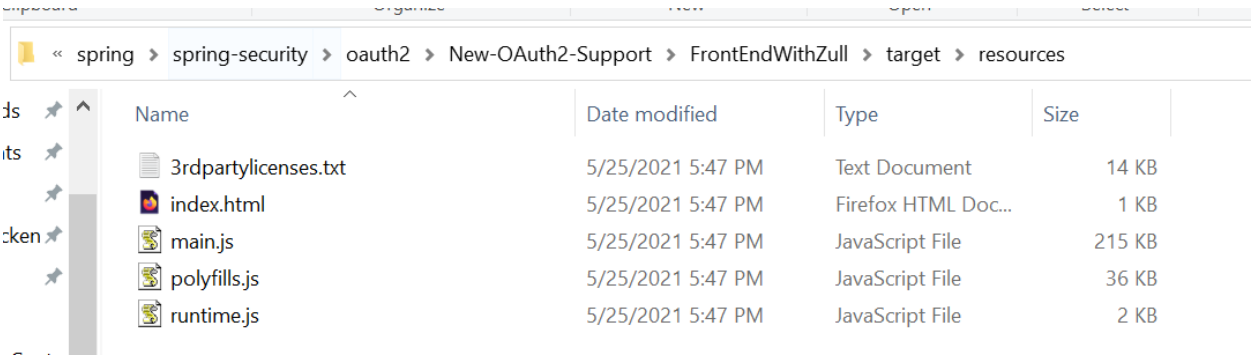
Cấu hình build trong file pom

Câu hình buidl thì ta có 2 plugin :

1 frontend-maven-plugin : Sẽ có nhiệm vụ tự **install** node, npm, run install, npm run build ra thư mục rouces của thư mục target. (cấu hình build ở file angular.json chỗ outputpath).

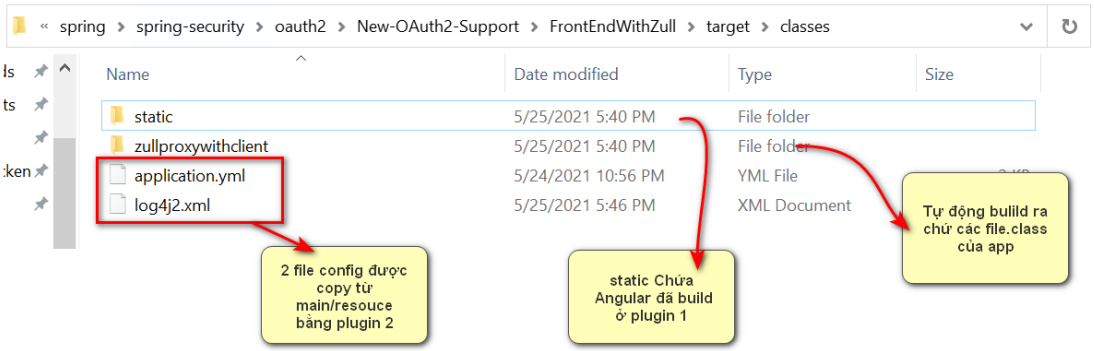
Tự động run instal node_modules và build ra thư mục targer/resource của app



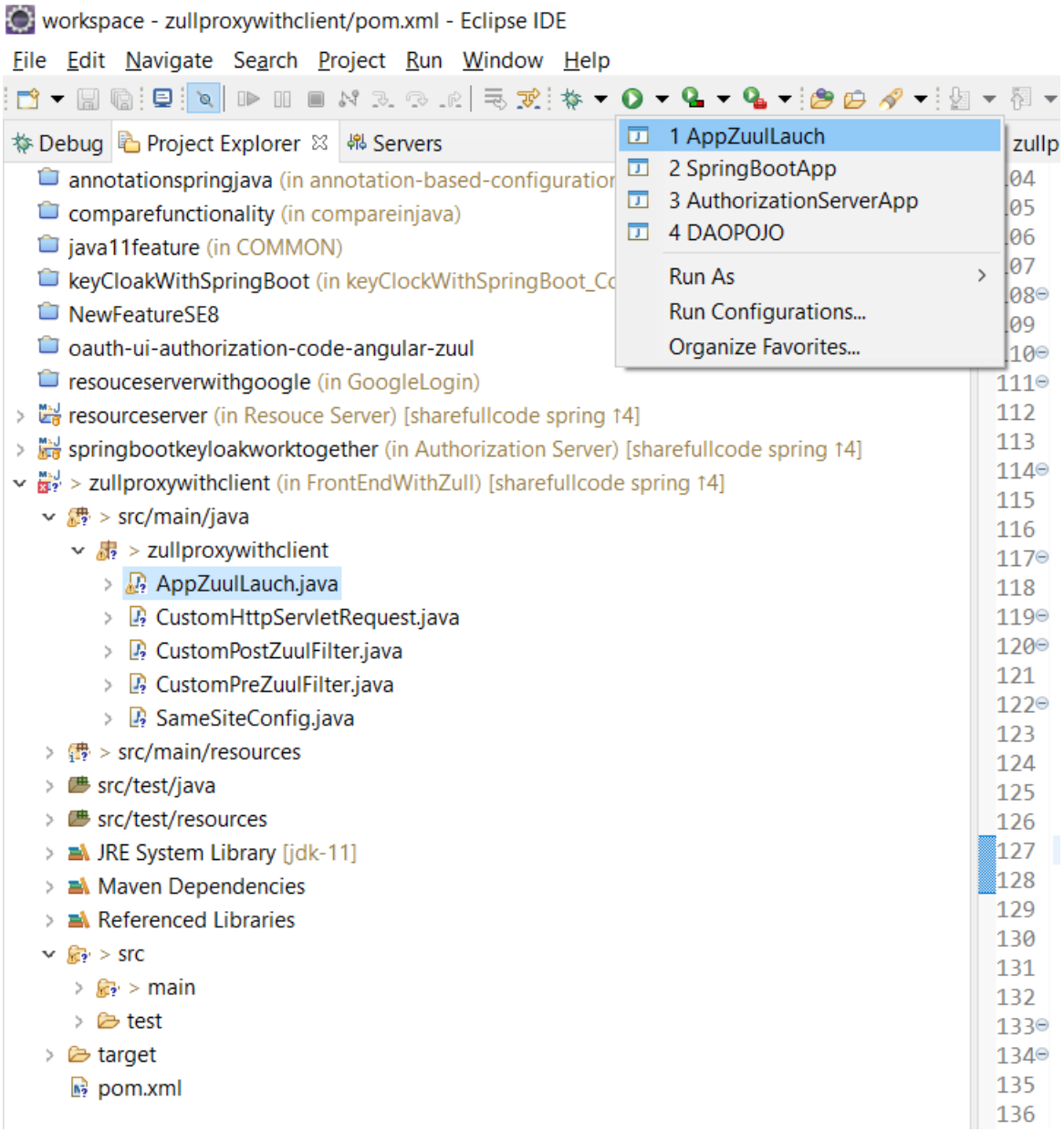


2 maven-resources-plugin: plugin này sẽ **copy file** đã cấu hình như file application.yml , log4j.xml bỏ vào thư mục classes để run đúng .

Build xong sẽ ra như thế này



Xong vào app spring boot và run lên



Tham khảo :

https://www.baeldung.com/spring-security-oauth

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