Report on Community Solid Server and Demo on WebID-OIDC(Community Solid Server and Keycloak)

Ludovico Granata, Simone Persiani

May 2021

Contents

1 Report on Community Solid Server

The community-server project started in May 2020 as a replacement of the legacy node-solid-server; it's the open source alternative to the proprietary project Enterprise Solid Server by Inrupt (the startup cofounded by Tim Berners Lee). Inrupt is sponsoring 2 IMEC researchers and contributing 1 developer. The main contributors who actively work on the repository are @joachimvh and @RubenVerborgh.

1.1 Authentication

1.1.1 IdentityProvider is not ready

It's on the roadmap for the project to be both a ResourceServer and an IdentityProvider: as of now, the IdP implementation is still not production-ready (they expect it to be completed during the second quarter of 2021). Citing @RubenVerborgh:" it's not an Identity Provider (yet), but rather being a Resource Server and accepting/validating the token provided by an IDP".

If a user already has a WebId, then he/she can correctly authenticate against community-server. On the other hand, until the IdP part won't be properly implemented, users will have to use an external IdP implementation to register their new WebIds.

1.1.2 WebId-OIDC Authentication Protocol

Authentication via WebId-OIDC is already implemented, we were able to test it and we could provide a demo, see section 2.

This is a (maybe incomplete) list of checks currently performed during the authentication phase (not considering DPoP):

- is the required resource protected?
- does the HTTP request contain an "Authorization" header?
- does the "Authorization" header value start with "Bearer "?
- Access Token verification (@solid/identity-token-verifier):
  - is the token well formed?
  - is the domain of the issuer (iss claim's value) contained in the list of trusted issuers defined inside the WebId profile of the agent with the solid:oidcIssuer predicate?
  - is the token valid? (check is performed by the JOSE-JWT library)
    - is the token correctly encrypted by the expected IdP?
    - is the token already expired?
    - does the aud (audience) claim of the token (which could be either a string or an array) contain the "solid" string?

The class containing the implementation is src/authorization/WebAclAuthorizer.ts.

1.1.3 WebID-TLS Authentication Protocol

Authentication via WebId-TLS is currently NOT implemented since in July 2020 Ruben decided to start from WebId-OIDC and until now no further attempt was made to implement this feature (see this comment where WebId-TLS is said to be ”deprecated”).

The class to be extended would be src/authentication/UnsecureWebIdExtractor.ts.

1.2 Authorization

WAC protocol implementation can be found in src/authorization/WebAclAuthorizer.ts. It’s still not complete, as it lacks support for some authorization patterns.

1.2.1 Supported Patterns

```
  <rule> acl:agentClass foaf:Agent
  <rule> acl:agentClass acl:AuthenticatedAgent
  <rule> acl:agent <webid>
  <rule> acl:default <container_path>
  <rule> acl:accessTo <resource_path>
```

1.2.2 Unsupported Patterns

```
  <rule> acl:agentGroup <group_url>
  <rule> acl:agentClass <generic_class>
  <rule> acl:trustedApp [acl:origin <origin_url>]
```

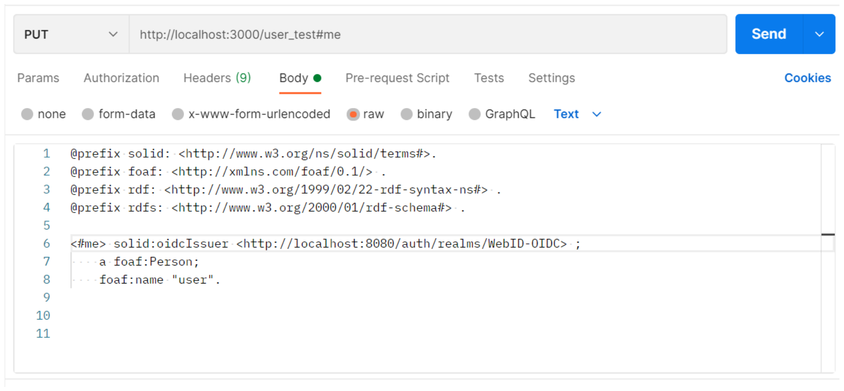
2 Demo WebID-OIDC : Keycloak, Community Solid Server and client-side Application

2.1 Setup Community Solid Server

1. start the Community Solid Server with Blazegraph as sparql endpoint:

```
node ./bin/server.js -c config/sparql-endpoint.json -s http://
    localhost:9999/blazegraph/sparql
```

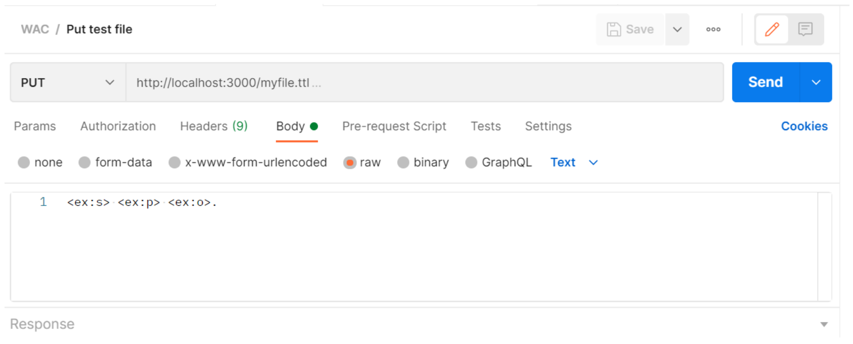
2. With postman (or any other way of doing HTTP requests) add a WebID profile to the Community Solid Server



http://localhost:3000/user_test#me

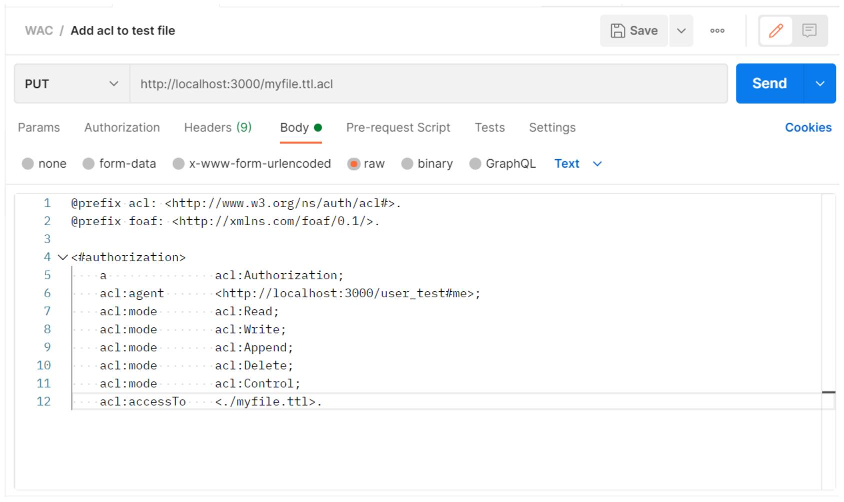
```
@prefix solid: <http://www.w3.org/ns/solid/terms#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#me> solid:oidcIssuer <http://localhost:8080/auth/realms/WebID-OIDC> ;
    a foaf:Person;
    foaf:name "user".
```

3. create the file "myfile.ttl" to test and also the file.ttl.acl that permits only a certain WebID to see it



http://localhost:3000/myfile.ttl ...

```
<ex:s> <ex:p> <ex:o>.
```

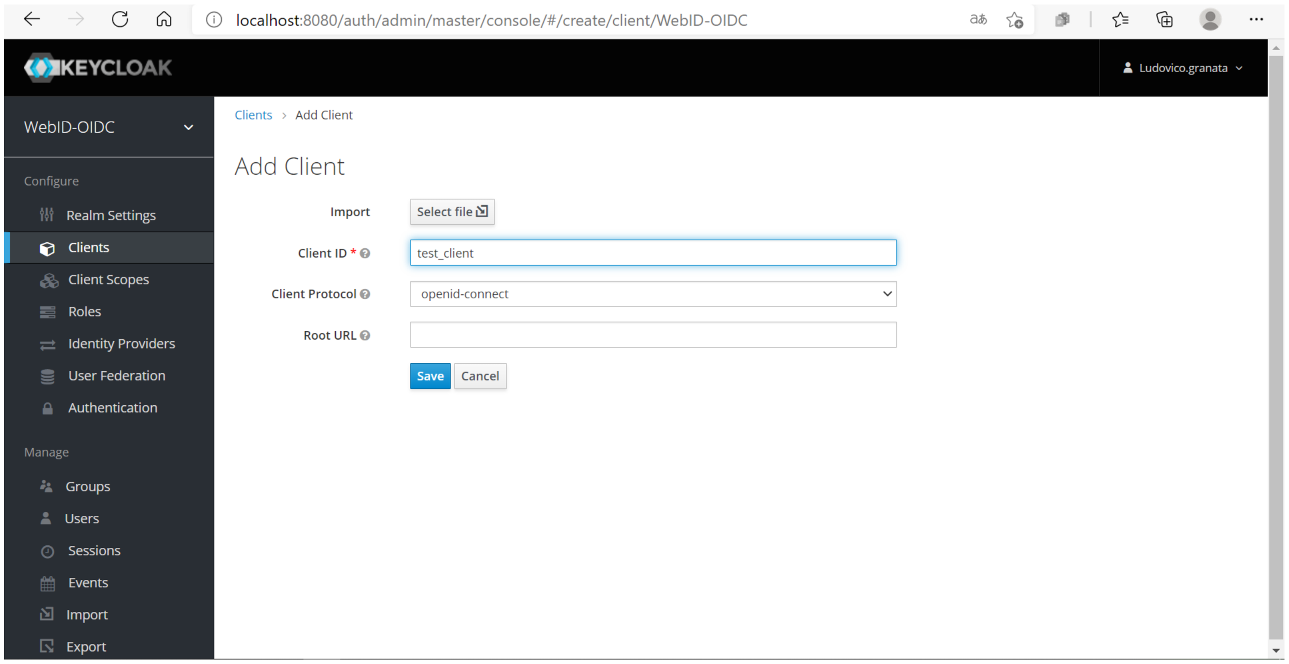


http://localhost:3000/myfile.ttl.acl

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<#authorization>
    a               acl:Authorization;
    acl:agent       <http://localhost:3000/user_test#me>;
    acl:mode        acl:Read;
    acl:mode        acl:Write;
    acl:mode        acl:Append;
    acl:mode        acl:Delete;
    acl:mode        acl:Control;
    acl:accessTo    <./myfile.ttl>.
```

2.2 Setup Keycloak

1. In the Realm "WebID-OIDC" add a new Client "test_client"



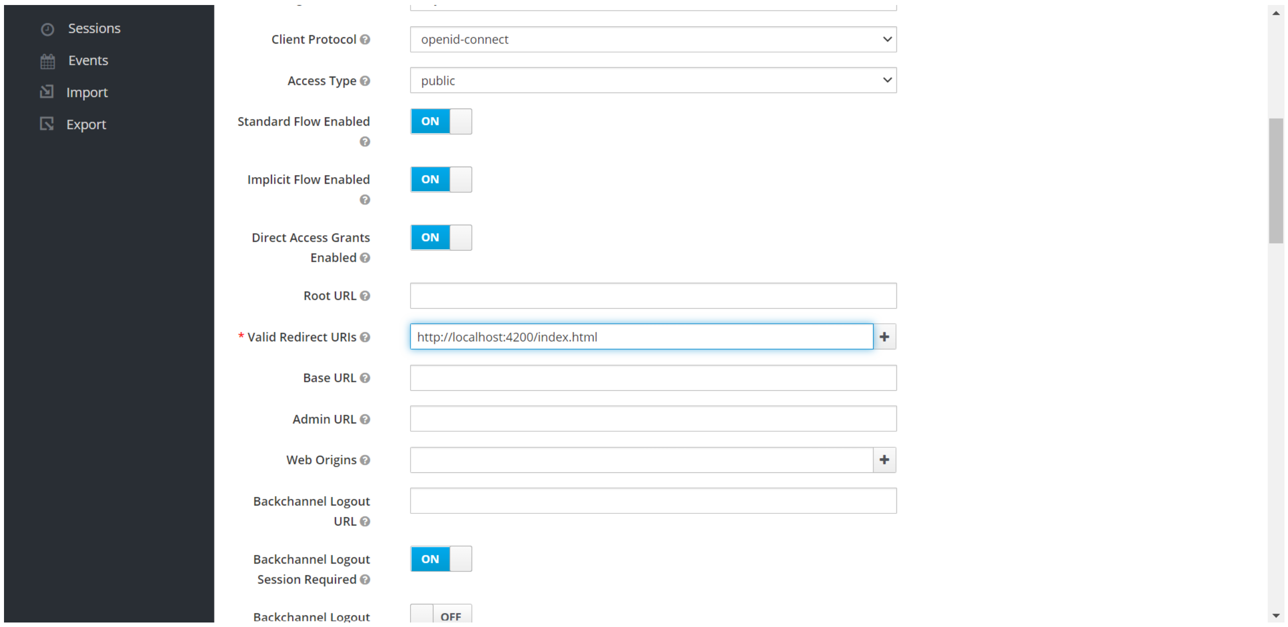
Import: Select file

Client ID *: test_client

Client Protocol: openid-connect

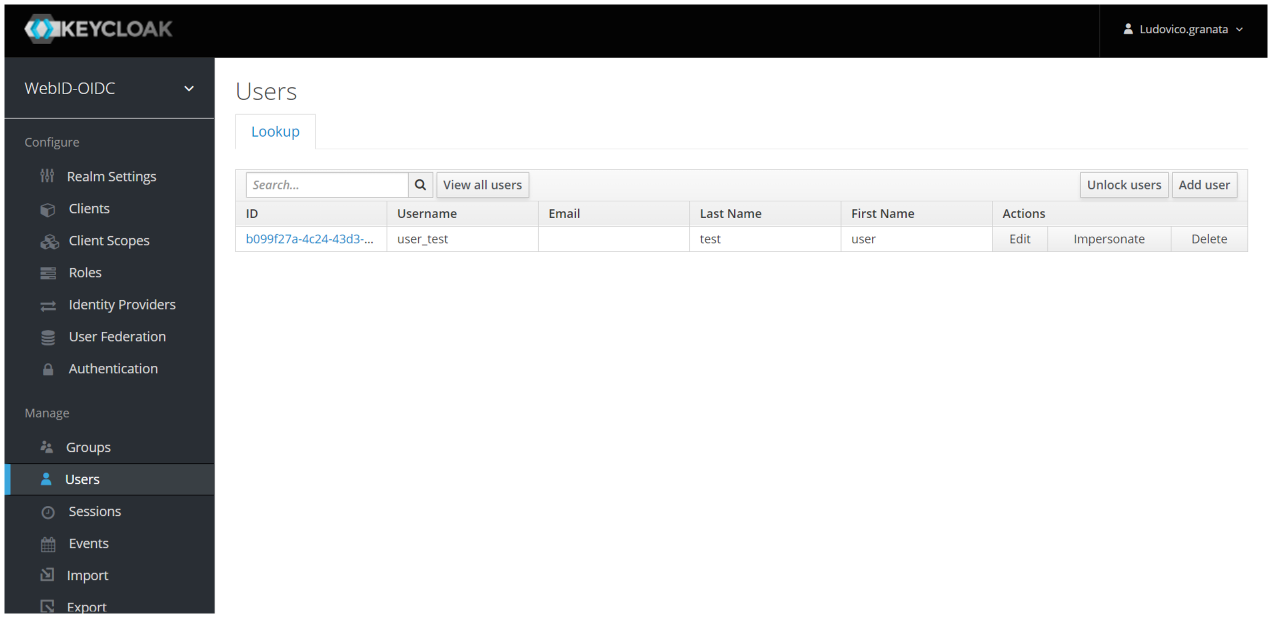
Root URL:

2. Make sure to enable the implicit flow and to put in the redirect url : http://localhost:4200/index.html (the URL of our client side application)



Client Protocol	openid-connect
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	ON
Direct Access Grants Enabled	ON
Root URL	
* Valid Redirect URIs	http://localhost:4200/index.html
Base URL	
Admin URL	
Web Origins	
Backchannel Logout URL	
Backchannel Logout Session Required	ON
Backchannel Logout	OFF

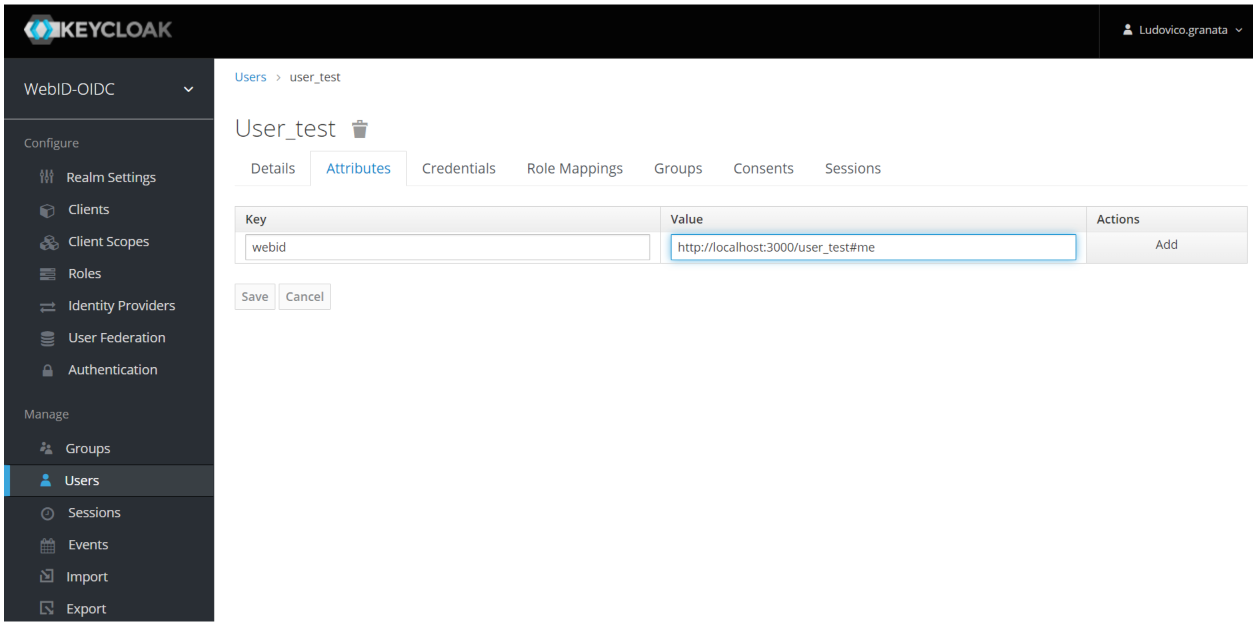
3. Add a user and set the password



Users

ID	Username	Email	Last Name	First Name	Actions		
b099f27a-4c24-43d3-...	user_test		test	user	Edit	Impersonate	Delete

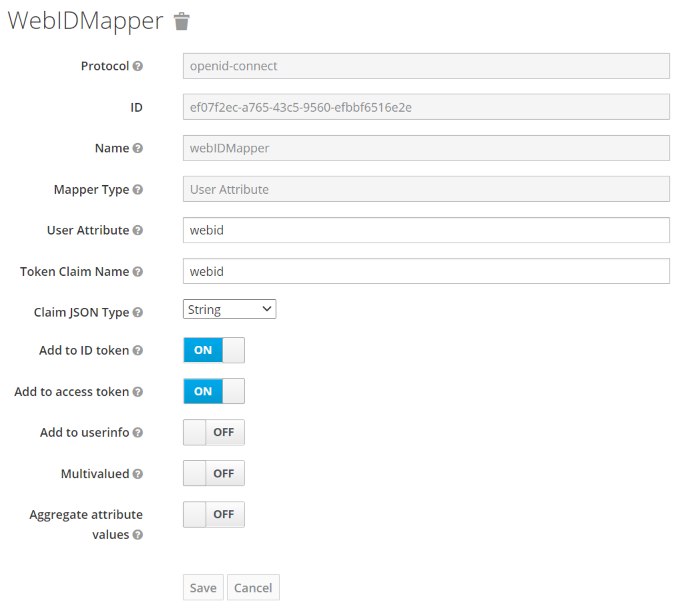
4. Add attribute with claim name webid and http://localhost:3000/user_test#me



User_test

Key	Value	Actions
webid	http://localhost:3000/user_test#me	Add

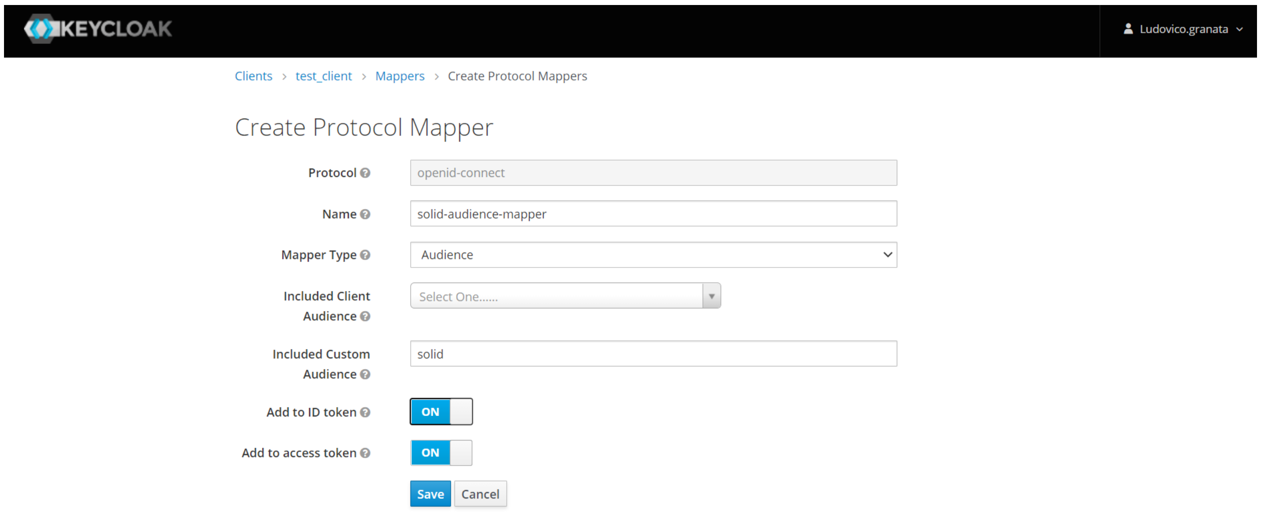
5. Add a protocol mapper for the webid (client → test_client → mappers → create protocol mapper)



WebIDMapper

Protocol	openid-connect
ID	ef07f2ec-a765-43c5-9560-efbbf6516e2e
Name	webIDMapper
Mapper Type	User Attribute
User Attribute	webid
Token Claim Name	webid
Claim JSON Type	String
Add to ID token	ON
Add to access token	ON
Add to userinfo	OFF
Multivalued	OFF
Aggregate attribute values	OFF

6. Add a protocol mapper for the webid (client → test_client → mappers → create protocol mapper)



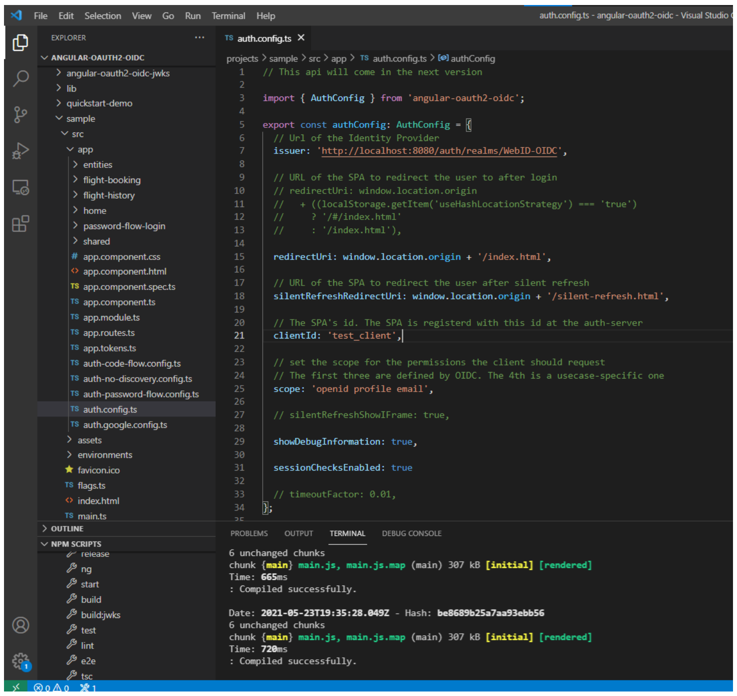
Create Protocol Mapper

Protocol	openid-connect
Name	solid-audience-mapper
Mapper Type	Audience
Included Client Audience	Select One......
Included Custom Audience	solid
Add to ID token	ON
Add to access token	ON

2.3 Setup Client Test Application

We have taken from github a simple client that work with Oauth2-OIDC protocol (https://github.com/manfredsteyer/angular-oauth2-oidc). To start it: npm run start

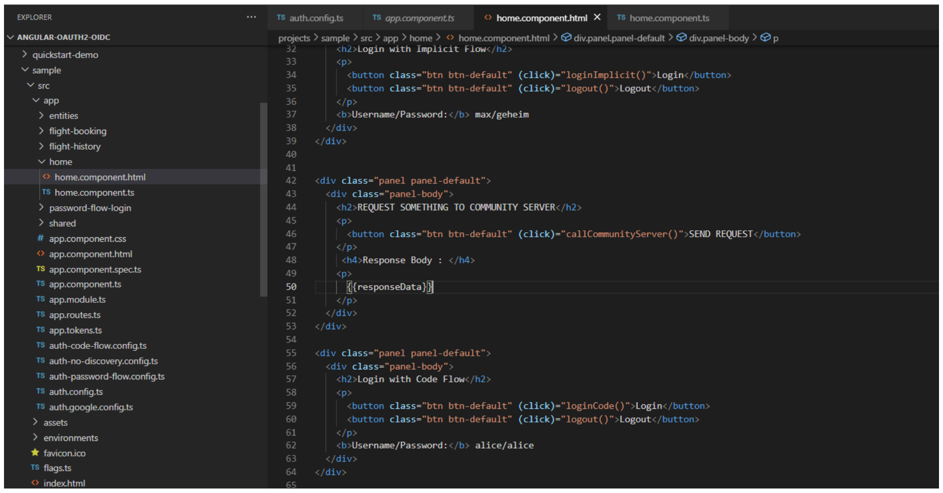
1. Change the auth.config file as follow:



```
// This api will come in the next version

import { AuthConfig } from 'angular-oauth2-oidc';

export const authConfig: AuthConfig = {
  // Url of the Identity Provider
  issuer: 'http://localhost:8080/auth/realms/WebID-OIDC',

  // URL of the SPA to redirect the user to after login
  // redirectUri: window.location.origin
  //   + ((localStorage.getItem('useHashLocationStrategy') === 'true')
  //     ? '/#/index.html'
  //     : '/index.html'),

  redirectUri: window.location.origin + '/index.html',

  // URL of the SPA to redirect the user after silent refresh
  silentRefreshRedirectUri: window.location.origin + '/silent-refresh.html',

  // The SPA's id. The SPA is registerd with this id at the auth-server
  clientId: 'test_client',

  // set the scope for the permissions the client should request
  // The first three are defined by OIDC. The 4th is a usecase-specific one
  scope: 'openid profile email',

  // silentRefreshShowIFrame: true,

  showDebugInformation: true,

  sessionChecksEnabled: true

  // timeoutFactor: 0.01,
};
```

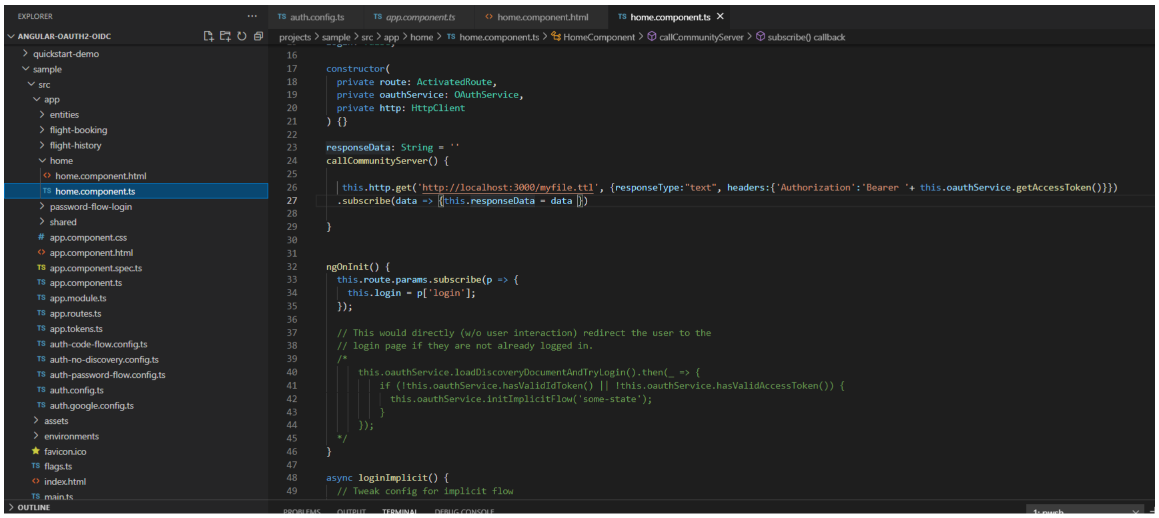
```
6 unchanged chunks
chunk {main} main.js, main.js.map (main) 307 kB [initial] [rendered]
Time: 665ms
: Compiled successfully.

Date: 2021-05-23T19:35:28.049Z - Hash: be8689b25a7aa93ebb56
6 unchanged chunks
chunk {main} main.js, main.js.map (main) 307 kB [initial] [rendered]
Time: 720ms
: Compiled successfully.
```

2. Add a simple button to make a request



```
    <h2>Login with Implicit Flow</h2>
    <p>
      <button class="btn btn-default" (click)="loginImplicit()">Login</button>
      <button class="btn btn-default" (click)="logout()">Logout</button>
    </p>
    <b>Username/Password:</b> max/geheim
  </div>
</div>


<div class="panel panel-default">
  <div class="panel-body">
    <h2>REQUEST SOMETHING TO COMMUNITY SERVER</h2>
    <p>
      <button class="btn btn-default" (click)="callCommunityServer()">SEND REQUEST</button>
    </p>
     <h4>Response Body : </h4>
    <p>
      {{responseData}}
    </p>
  </div>
</div>

<div class="panel panel-default">
  <div class="panel-body">
    <h2>Login with Code Flow</h2>
    <p>
      <button class="btn btn-default" (click)="loginCode()">Login</button>
      <button class="btn btn-default" (click)="logout()">Logout</button>
    </p>
    <b>Username/Password:</b> alice/alice
  </div>
</div>
```

3. Add the function to handle the http request in the file home.component.ts



```
  constructor(
    private route: ActivatedRoute,
    private oauthService: OAuthService,
    private http: HttpClient
  ) {}

  responseData: String = ''
  callCommunityServer() {

    this.http.get('http://localhost:3000/myfile.ttl', {responseType:"text", headers:{'Authorization':'Bearer '+ this.oauthService.getAccessToken()}})
    .subscribe(data => {this.responseData = data })

  }


  ngOnInit() {
    this.route.params.subscribe(p => {
      this.login = p['login'];
    });

    // This would directly (w/o user interaction) redirect the user to the
    // login page if they are not already logged in.
    /*
      this.oauthService.loadDiscoveryDocumentAndTryLogin().then(_ => {
        if (!this.oauthService.hasValidIdToken() || !this.oauthService.hasValidAccessToken()) {
          this.oauthService.initImplicitFlow('some-state');
        }
      });
    */
  }

  async loginImplicit() {
    // Tweak config for implicit flow
```

Now we can login and when we will make the request we will be recognized with our WebID and we will be allowed to see the resource.

3 What we could do next

- Provide an implementation of the WebId-TLS protocol;
- Complete the implementation of the WAC protocol (the part about groups);
- Help to finish the IdP part of community-server: this can be done either directly or indirectly (by solving unrelated issues so to ease the work of other developers).
- Add a mechanism to Keycloak so that during registration of a new user it automatically create a WebID.