

[github](#) - [Stack Overflow](#) - [LinkedIn](#) - [MVA](#) - a developer braindump on .Net, c#, Java, Javascript ...

{"@id":"cedric-dumont.com"}

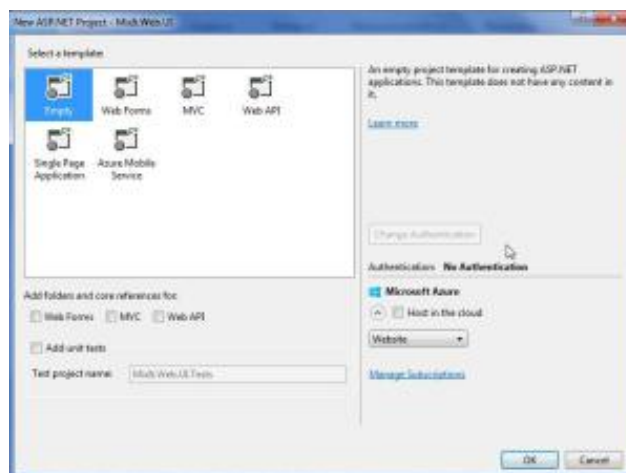
IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 4

Home (<http://cedric-dumont.com/tutorials/identityserver-v3-membershipreboot-angularjs-webapi-2-and-mvc-mix-it-introduction/>)

In this part, we will create a web based UI using **AngularJs** (<https://angularjs.org/>) and **SemanticUI** (<http://semantic-ui.com/>). I am not going to dive in details, but will explain the solution I build up to use Identityserver.v3 as the Authentication mechanism and memberShipReboot as the User Account repository.

Step 1 : create a new Empty Web Project

Create a new Empty web project with no authentication



(<https://cedricdumont.files.wordpress.com/2014/12/4-1.jpg>)

Step 2 : install Semantic-UI and AngularJS

Download and extract Semantic-Ui folder. In the solution, I've created a 'content' and a 'script' folder. From the 'dist' folder located in the extracted semanti-ui file, get the *themes* folder and the file *semantic.css*. Copy them in the 'content' folder of the solution

Also get the *semantic.js* file in that 'dist' directory and copy it to the solution's 'script' folder.

You must also download *angular.js* file and *angular-route.js* file from angularjs web site and copy them in the *script* folder of the solution

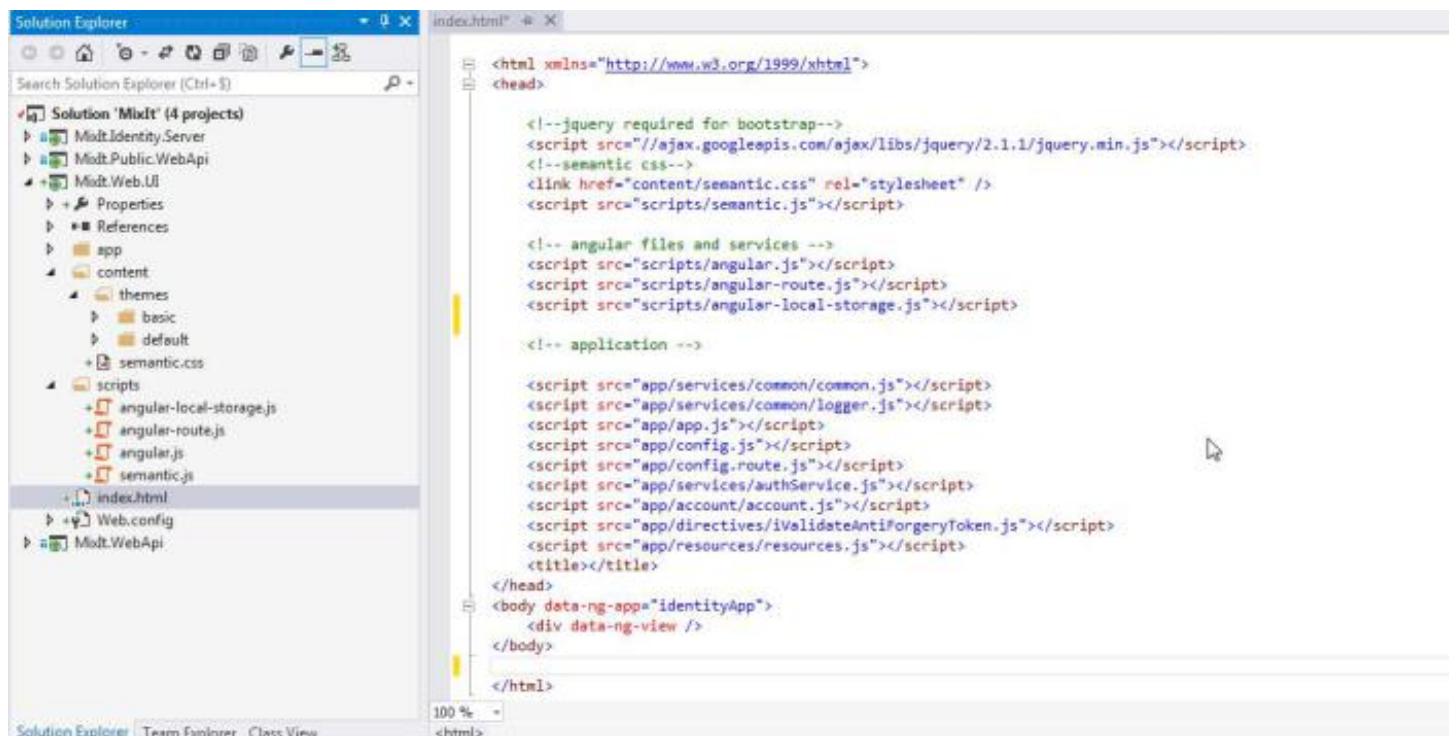
N.B.: You can also use CDN for that

I also use a local storage service (<https://github.com/grevory/angular-local-storage> (<https://github.com/grevory/angular-local-storage>)) to store the Auth token received from Identityserver.v3.

So download and copy that file in the *script* directory.

Create an index.html file and reference all these scripts. Don't forget JQuery (required for semantic-UI).

You should end up with something like this:



(<https://cedricdumont.files.wordpress.com/2014/12/4-2.jpg>)

The app folder will contain our angularJs app. We will name it IdentityApp. Check in the body tag of the index.html file where it is referenced.

Now that all framework are configure, let's start with the angular app.

Step 3 : the angular App

If you open the app folder you will see the following:

- the app.js file in which I reference the dependencies

```
1 | var app = angular.module('identityApp', [
2 |     'ngRoute',
3 |     'LocalStorageModule',
4 |     'common'
5 | ]);
```

- The *common* service is a convenience service (kind of a bag) that has dependencies on the *logger* service, the *\$q* service. It also provides some helper methods. (This pattern is stolen from **John Papa's hottowel** (<http://www.johnpapa.net/hottowel/>))

- The `config.route.js` file is for mapping routes to the controllers. In this application, I have 2 routes :
 '`/account`' that forwards to the account controller and template
 '`/resources`' route that forwards to the resource controller and template
- the `config.js` file adds a config objects. In there, I've configured the different url to my api's and also the url to the identityserver (your urls might be different)

```

1  var apiUrl = 'http://localhost:10073/ (http://localhost:10073/)';
2  var idsrvUrl = 'https://localhost:44305/identity/ (https://localhost:44305/identity/)';
3  var publicApiUrl = 'http://localhost:14117/ (http://localhost:14117/)';
4
5  var api = {
6      account: apiUrl + 'api/account/',
7      connect: idsrvUrl + 'connect/token',
8      resource : publicApiUrl + 'resource/'
9  };

```

- the `account` and `resources` directory contains both an `.html` file (the views) and a `.js` file (the controller for each view)
- the `services` folder contains the `common` and `logger` service files and also the **authService**.
- The `authService` contains the methods to create accounts and also login and logout.

the create function calls our 'private api'.

```

1  register: function (credentials) {
2      return $http.post(config.api.account + 'create',
3          {
4              Username: credentials.userName,
5              Password: credentials.password, Email: credentials.email
6          })
7      .success(function (data, status, headers, config) {
8      });
9  };
10 }

```

The login function get a token from the identityserver and store it in the session.

```

1  login: function (credentials) {
2      var user = { grant_type: 'password', username: credentials.userName };
3
4      //required for Idsrv V3 call (required by spec)
5      var urlEncodedUrl = {
6          'Content-Type': 'application/x-www-form-urlencoded',
7          'Authorization': 'Basic SWRlbnRpdHlXZWJVSTpzZWNYZXQ='
8      };
9
10     return $http({
11         method: 'POST', url: config.api.connect, headers:urlEncodedUrl,
12         data: user,
13         transformRequest: function (str) {
14             var str = [];
15             for (var p in obj)
16                 str.push(encodeURIComponent(p) + "=" + encodeURIComponent(obj[p]));
17             return str.join("&");
18         }
19     });
20 }

```

```

17         }).success(function (data, status, headers, config) {
18             Session.create(data.access_token, credentials.userName, '
19             localStorageService.set('bearerToken', data.access_token)
20         }).error(function (data, status, headers, config) {
21             Session.destroy();
22
23         });
24     },

```

There is also an authInterceptor configured to add the beare token on each request if present.

```

1  request: function (config) {
2      console.log('bearer : ');
3      console.log(localStorageService.get('bearerToken'));
4      config.headers = config.headers || {};
5      if (localStorageService.get('bearerToken')) {
6          config.headers.Authorization = 'Bearer '
7              + localStorageService.get('bearerToken');
8      }
9      return config;
10 }

```

- the app/account/account.js file contains the controller that depends on the AuthService.

all function call the corresponding AuthService method and manages some scope variable for the view to display correctly. These views are only implemented to display the functionalities.

```

1  function register(credentials) {
2      log(credentials);
3      AuthService.register(credentials).then(function (data, status, head
4          $rootScope.$broadcast(AUTH_EVENTS.registrationSuccess);
5          successLogger('registered');
6          resetView();
7      }, function () {
8          $rootScope.$broadcast(AUTH_EVENTS.registrationFailed);
9          errorLogger('notregistered');
10     });
11 };
12
13 function login(credentials) {
14     AuthService.login(credentials).then(function () {
15         $rootScope.$broadcast(AUTH_EVENTS.loginSuccess, [credentials.e
16         successLogger('logged in ' + AuthService.isAuthenticated());
17         vm.authenticated = true;
18
19     }, function () {
20         $rootScope.$broadcast(AUTH_EVENTS.loginFailed);
21         vm.authenticated = false;
22     });
23 };
24
25 function logout() {
26     AuthService.logout();
27     vm.authenticated = false;
28
29 };

```

- the app/resource/resource.js file just get a resource based on it's id.

if your are not authenticated the get will fail.

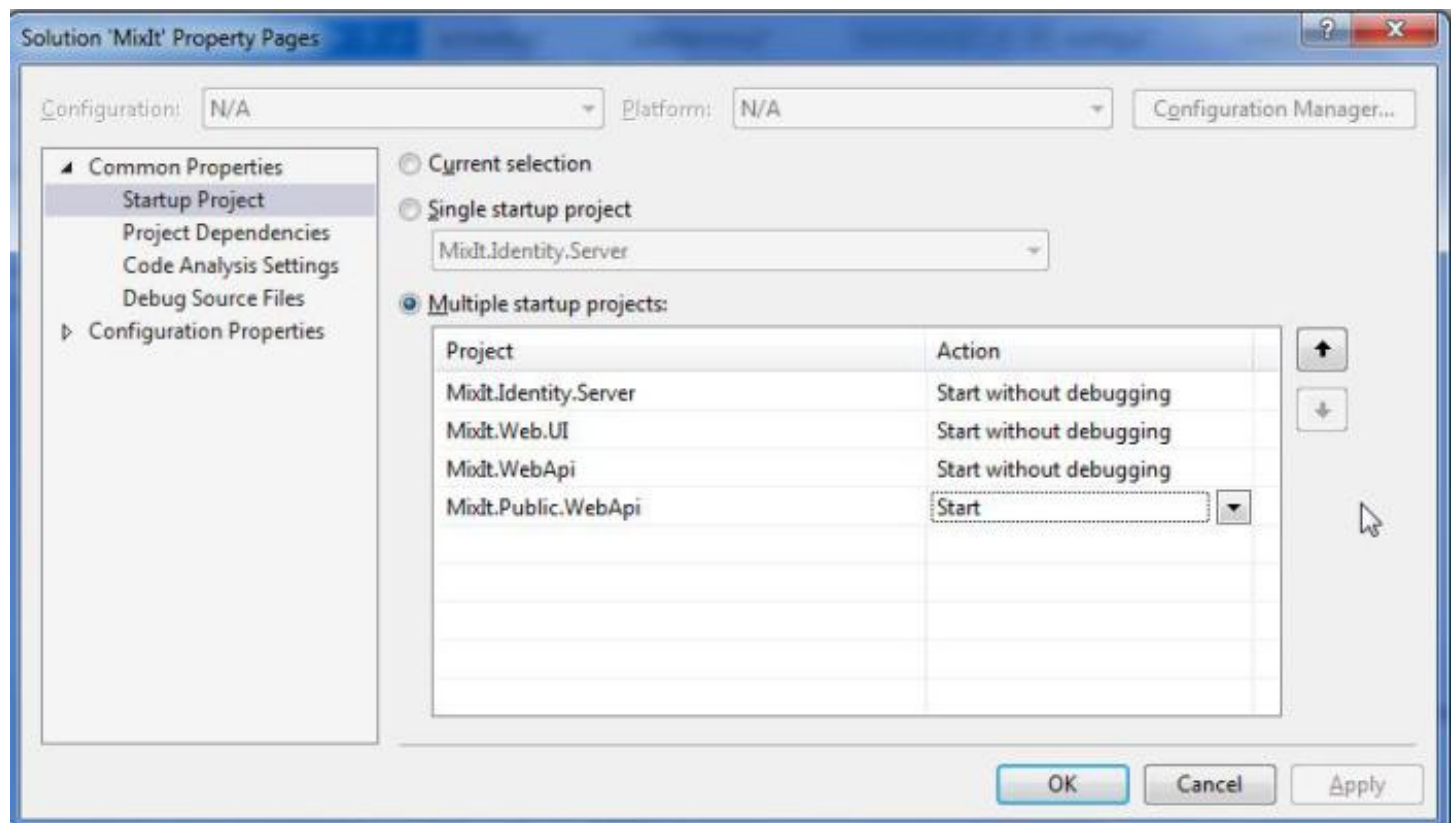
if you are authenticated, the resource will display.

Once again, the UI is really simple for convenience. The getResource method just call te `$http.get()` and the `authInterceptor` will set the bearer token if present. so we do not care about headers here.

Step 4 : Test It

To test the application, you must run all projects together. Therfor, you need to set the solution with multiple startup project.

Right click on the solution and select *Set Startup Projects ...*



<https://cedricdumont.files.wordpress.com/2014/12/4-3.jpg>

Then hit F5

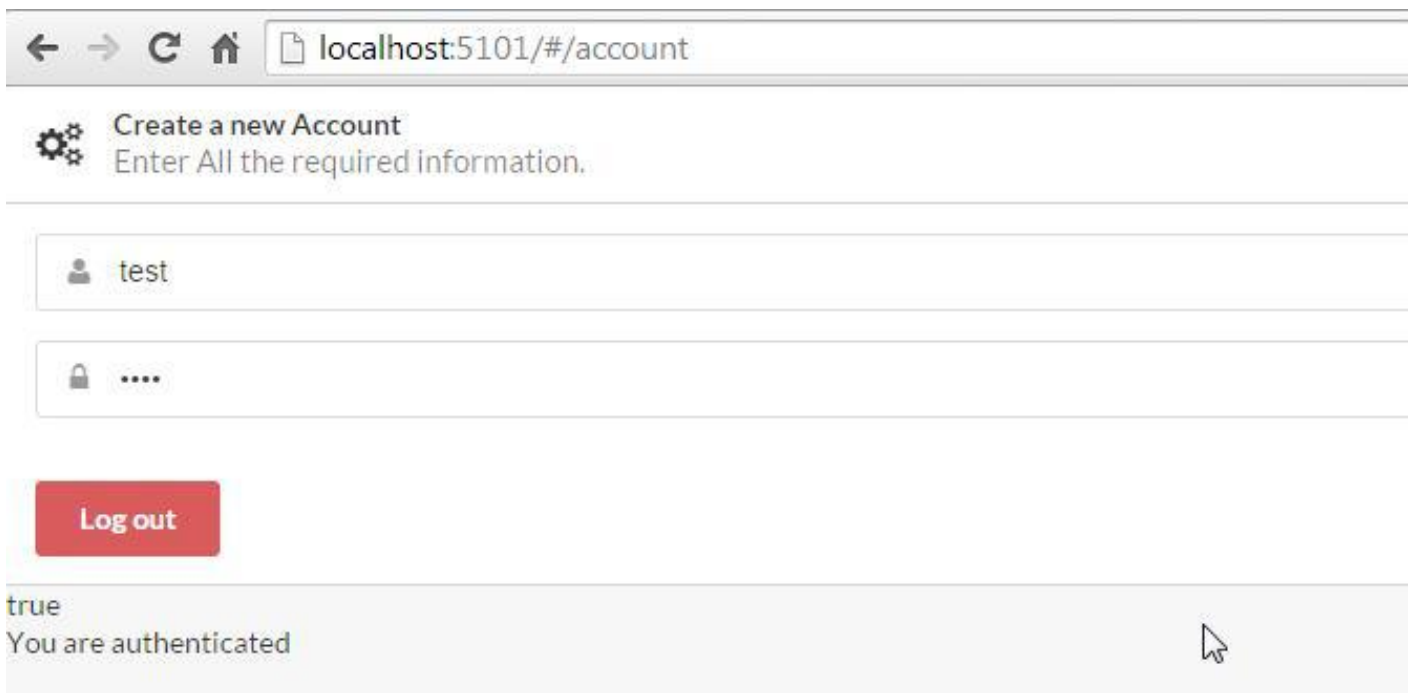
You will have 4 tabs that will open in your browser.

locate the tab showing the web UI and enter <http://localhost:5101/#/resources> (<http://localhost:5101/#/resources>) in the adress bar.

enter '1' in the inputfield and normally, you should have the message that we could not load the resource.

now enter <http://localhost:5101/#/account> (<http://localhost:5101/#/account>) in the adress bar and connect with user 'test' and password 'test' (or another user you created with fiddler.

you should see this if everything went well:



← → ↻ 🏠

⚙️ **Create a new Account**
Enter All the required information.

👤 test

🔒

Log out

true
You are authenticated

<https://cedricdumont.files.wordpress.com/2014/12/4-5.jpg>

then enter the following in the adress bar : <http://localhost:5101/#/resources>
(<http://localhost:5101/#/resources>)

if you cannot load the resource, then open the developer console (CTRL + SHIFT + I) and check if you have the following error:

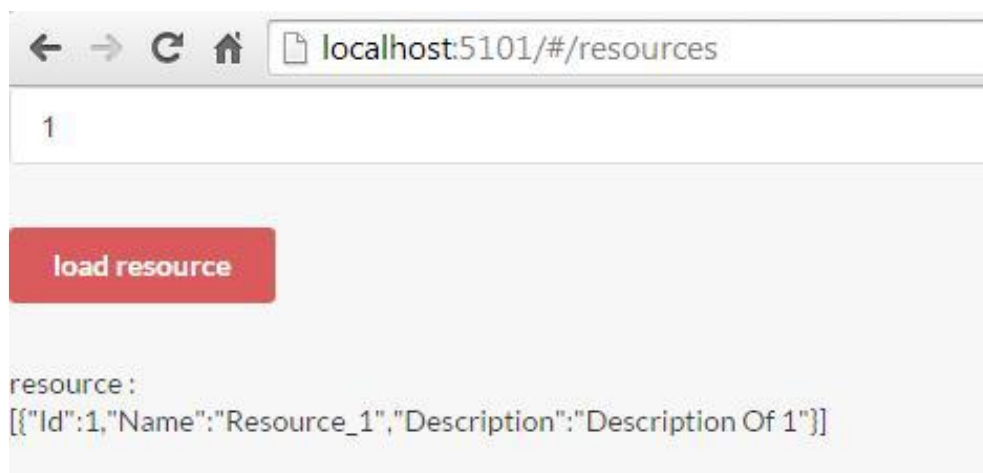


```
XMLHttpRequest cannot load http://localhost:14117/resource/1. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:5101' is therefore not allowed access. The response had HTTP status code 405. (index):1
```

<https://cedricdumont.files.wordpress.com/2014/12/4-6.jpg>

If so, don't forget to change the Cors settings in your controller and check in startup that it's enabled.

you should see the following result:



← → ↻ 🏠

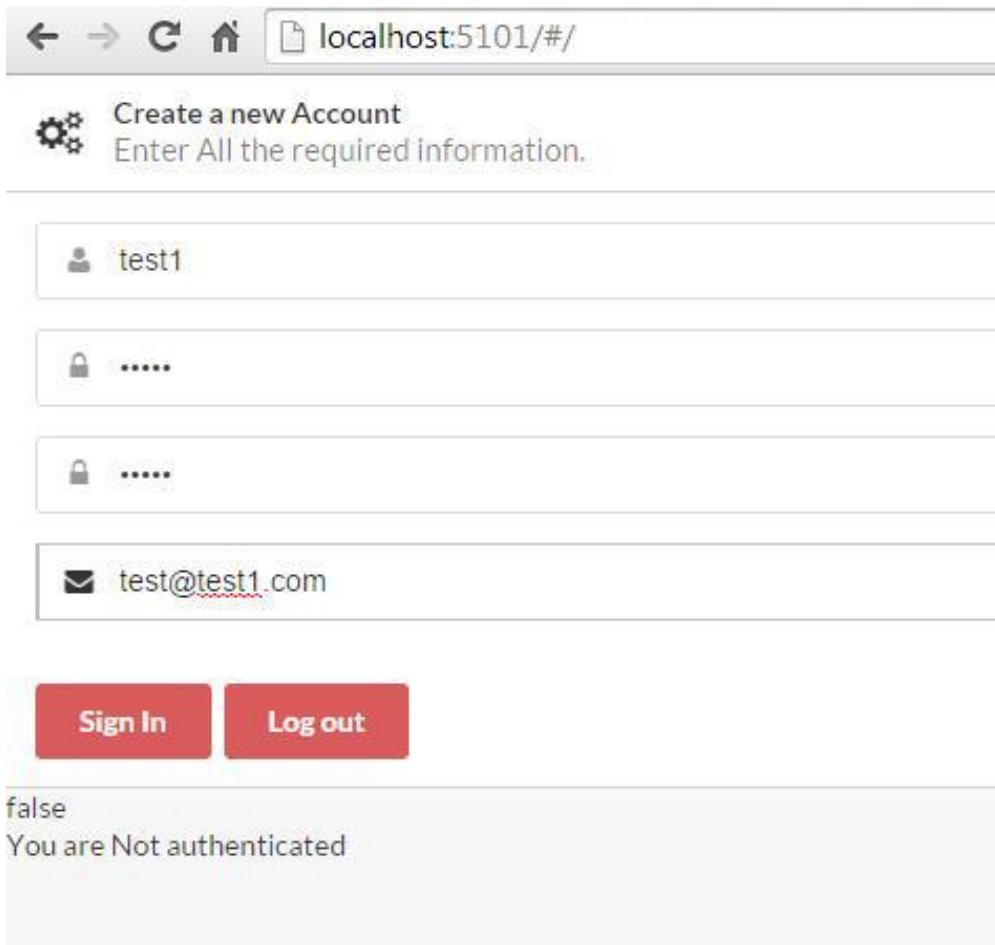
1

load resource

resource :
[{"Id":1,"Name":"Resource_1","Description":"Description Of 1"}]

<https://cedricdumont.files.wordpress.com/2014/12/4-7.jpg>

Let's create a new account (test1 with password test1)



← → ↻ 🏠 localhost:5101/#/

⚙️ **Create a new Account**
Enter All the required information.

👤 test1

🔒

🔒

✉️ test@test1.com

Sign In **Log out**

false
You are Not authenticated

<https://cedricdumont.files.wordpress.com/2014/12/4-8.jpg>

then log into it with credentials test1/test1 and get the resource, it should go fine.

Now Here I have got a question : Is it the right way to do it. Because here, I need to put in the client application (javascript) the clientId and client secret. I could use another flow (like implicit flow) but then the user would be redirected to a consent screen to authorize the application (I am in) to access the application. It's a bit weared here.

When I found a better way to do, I will update this doc. please feel free to add comments.

10 thoughts on “IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! : Part 4”

1. Pingback: **IdentityServer.v3, MembershipReboot, AngularJs, WebApi 2 and MVC : Mix It ! | {"@id":"cedric-dumont.com"}**

2.

John says:

January 29, 2015 at 17:42

Hi, i'm testing your samples and i'm getting an error on app/directives/iValidateAntiForgeryToken.js as nosuch file exists. Where can i find it? Does it has something to do with <http://www.fredonism.com/archive/protect-your-web-api-from-csrf-attacks.aspx>?

Best Regards

REPLY

1.

cedricdumontc says:

January 29, 2015 at 18:44

you can delete it. it requires using of mvc, but here i didn t want to use mvc. so i left this for the moment. i m still looking for an anti forgery protection but don t know if its necessary in this configuration . i will delete it from the sources

REPLY

3.

John says:

January 30, 2015 at 12:16

It's working now with the exception that localStorageService.set('bearerToken', data.access_token) is throwing an error and later no Bearer Token is sento to the API. Any clue? I've just downloaded the angular-local-storage.js file...

REPLY

4.

cedricdumontc says:

January 30, 2015 at 14:43

what is the error? I see that angular-local-storage is already configured in the sources

REPLY

5.

John says:

January 30, 2015 at 20:26

It gives me null value in angular-local-storage.js when setting the store for any value... I've commented the LocalStorage writes and used the Session Object instead...

REPLY

6.

Patrick says:

February 25, 2015 at 20:40

Hi Cedric, first of all – great post! I think the “Resource Owner Password Credentials” flow is not recommended to use in a SPA because you expose the client secret. There is an authorization / authentication endpoint in idsvr3 where you can either request access tokens

or authorization codes (implicit or authorization code flow). I believe the implicit flow is the right choice because it is the only way to request a token without exposing your client secret in your js code.

cheers

REPLY

1.

cedricdumontc says:

February 25, 2015 at 20:45

i totally agree and i related this in the post. it s not good to expose the client secret and it brings us back to as if we use basic authentication. but what i want is not to be redirected to the id server login page but to use a login page in my app.

REPLY

7.

John says:

February 25, 2015 at 20:52

That's why the Implicit flow (by design) does not use any client secret, isn' it? And i guess that the Resource Owner Password Flow is suited when you can achieve Direct Authentication, otherwise we will end up with browser redirection's with your Identity Provider/Outthorization Server...

REPLY

8.

Patrick says:

February 25, 2015 at 22:15

Yes, i think the only way to use your own login form without beeing redirected is to do the authentication in your web api backend. There you can use the client secret in a secure way. But i don't like the idea of mixing authentication with business logic. But what's the downside of redirecting to a login page? I think you can customize the idsrv3 login page and maybe you could even integrate the login form into your app so you don't have to redirect? Auth0 has this feature... maybe idsrv3 too?

REPLY

[Blog at WordPress.com.](#) — [The Sequential Theme.](#)

© Follow

Follow {"@id":"cedric-dumont.com"}

Build a website with WordPress.com

