قبلت للنشر في مجلة جامعة البعث (حمص–سوريا) بالمجلد 40 لعام 2018 م.

حماية خدمات وب نقل تمثيل الحالة (RESTful) باستخدام البرتوكول OpenID Connect

الدُكتورة زينب خلوف*

الملخّص

تعدّ خَدَمَاتُ وب نَقْلْ تَمْثيل الْحَالَة (RESTful Web Services) مُناسبة لإنشاء واجهات برمجة التطبيقات (RESTful APIs) لمُستخدمين مُنتشرين على الإنترنت بدونِ متطلباتٍ مُكلفة مُقارنة بخدمات الوب المُعتمدة على البرتوكول البسيط للوصول للكائنات (SOAP) [1]. يتمّ في هذه الخدمات اعتبار البيانات و التوابع كموارد لكل منها معرّف URI، ويُمكن الوصول لكلّ مورد من خلال عمليات البرتوكول HTTP. تُمثَّل الموارد التي ستُنقل إلى المُستخدم وفقاً لصيغة من معيار الامتدادات المُتعددة الأغراض لبريد الانترنت (MIME).

يوجد عدةُ معايير وتقنيات لحماية خدمات وب نقل تمثيل الحالة و للتوثّق من هوية مستخدمي الخدمة، تمايزت بين حلول تتطلبُ وجودَ قاعدة بيانات محليّة للمستخدمين و أُخرى تعتمد على مُخدّمات خارجية موثوقة.

يقدّم هذا البحث دراسة نظريّة وعمليّة توضّح الرسائل المختلفة للمعيار Open ID Connect وهو أحد المعايير الحديثة للتوثّق من هوية المستخدم بوساطة مخدّم سماحيّة خارجي أو مايُسمى مزود هوية، مبني فوق برتوكول السماحية المفتوح (Open Authorization OAuth 2.0).

الكلمات المفتاحية:

أمن النظم و الشّبكات الحاسوبية، خدمات وب نقل تمثيل الحالة، Open ID Connect، oAuth، مزوّد الهوية.

* أستاذ مساعد-قسم هندسة الشبكات و النظم الحاسوبية- كلية الهندسة المعلوماتية-جامعة البعث

Securing Representational State Transfer (RESTful) Web Services Using Open ID Connect

Dr. Zainab Khallouf*

Representational state transfer (RESTful) web services are lightweight web services suitable for creating APIs for users spread across the Internet without costly requirements compared with SOAP-based web services [1]. In RESTful web services data and functions are considered as resources transmitted in multipurpose internet mail extension (MIME) format. Moreover, these resources are identified by uniform resource identifiers (URIs) and accessed with HTTP operations.

Many standards and techniques exists to authenticate web service users, some techniques use centralized authentication local database while others rely on external authentication server.

This paper presents theoretical and practical analysis of Open ID Connect messages: One of the modern standard to authenticate users based on the protocol open authorization (oAuth) 2.0.

Keywords: Systems and computer networks security, restful web services, oAuth, openid connect, identity provider.

* Associate Professor, department of systems and computer networks engineering, faculty of informatics engineering, Al-Baath University, Homs, Syria.

مقدمة والهدف من البحث:

تمكّن خدماتُ وب نقل تمثيل الحالة (RESTful Web Services) من إنشاء واجهات برمجة التطبيقات (APIs) لمُستخدمين مُنتشرين على الانترنت بدونِ متطلباتٍ مُكلفة مُقارنة بخدمات الوب المعتمدة على البرتوكول البسيط للوصول للكائنات (SOAP)[1].

تستندُ خدمات وب نقلُ تمثيل الحالة على اعتبار البيانات و التوابع كموارد، وعلى تفاعل المُستخدم مع خدمة الوب من خلال عمليات البرتوكول HTTP حيث يُحدَدُ لكل مورد رابط (URI). تُمثَّل الموارد التي ستُنقل إلى المستخدم وفقاً لصيغة من معيار الامتدادات المتعددة الأغراض لبريد الانترنت (MIME).

يتطلبُ الاستخدام الآمن لخدمات الوب التوثّق من هويّة المُستخدم ولعلّ إنشاء قاعدة بيانات محليّة لحسابات المُستخدمين ولإمكانيات الحماية (Credentials) لهم يعدّ من أكثر الحلول استخداماً، لكن للتوثّق المحلّي محدوديات عدّة ومنها:

- قد لايحبذُ المستخدم التسجيل (Sign Up) و إنشاء حساب على موقع وب (مثل موقع تجارة الكترونية) فنجدهُ قد ترك الموقع لهذا السبب.
- إنشاء وإدارة قاعدة بيانات لكل تطبيق في الشركات مُتعددة التطبيقات يُمكن أن يتحول لكابوس بالنسبة للمدير.

تفويض التوثق من المُستخدم و تأمين الهوية إلى خدمة مُخصصة تدعى مزود الهوية (Identity Provider (IdP)) أتى كحلٍ لهذه المحدوديات. كحالة خاصة يُمكن لشركات مثل غوغل، تويتر، فيس بوك أن تلعب دور مزوّد هوية، كما يمكن بناء خدمة مزود هوية (IdP) في شركة للموظفين و للمتعاقدين للوصول لتطبيقات الانترنت.

يقدّمُ البحث دراسة نظريّة وعمليّة تُحلل الرسائل المختلفة للمعيار Open ID Connect [9] و هو أحد المعايير الحديثة للتوثّق من هوية المستخدم بوساطة مخدّم سماحيّة خارجي أو مايسمى مزود هوية، اعتماداً على برتوكول السماحية المفتوح (Open Authorization OAuth 2.0). يقدّم هَذَا الْبَحْثُ دراسة تطبيقية لتحليل رسائل المعيار بعد التقاطها.

تمّ تنظيم هذه الورقة كالآتي: يقدّم القسم الأول دراسة نظرية للبرتوكول OAuth 2.0، ويركزُ القسم الثاني على المعيار Open ID Connect، يليه قسمٌ عملي، لنُلخّص نتيجة البحث والتوجهاتِ المستقبلية في القسم الرابع.

البرتوكول OAuth2.0 protocol [2]:

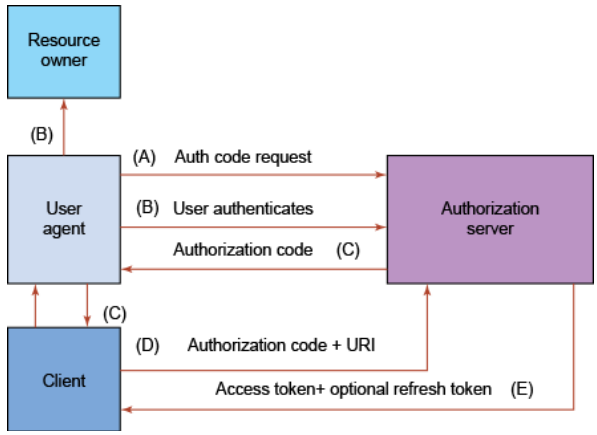
OAuth هو برتوكول مفتوح لِلسَّمَاحِيَّة مقيّس في RFC 6749، إصداره الْحَالِيُّ هُوَ الْإِصْدَارُ الثاني الذي يبسّط رسائل الإصدار الأول من البرتوكول و يأخذ بالاعتبار حالات استخدام أكثر تنوعاً. يمكّن البرتوكول المُستخدم (مالك الموارد Resource Owner) من منح سماحيّة بشكل آمن لطرف يُسمى وفقاً لمصطلحات البرتوكول "زبون" (Client) –يُمكن أن يكون تطبيق وب، تطبيق سطح مكتب، أو تطبيق موبايل– ليصل بشكل آمن لموارد تخصُّ المستخدم على مزود خدمة (Service Provider) أو مخدّم الموارد (Resource Server). على سبيل المثال، يمكن للمُستخدم "بوب" السّماح لتطبيق زبون مثل تطبيق طباعة ملفات الوصول لملفاته الموجودة على مخدّم موارد، و تُمنح السّماحية للزبون من قبل مخدّم سَمَاحِيَّة (Authorization server) بعد مُوافقة المستخدم [10]. رداً على طلب السّماحية يُرسل مخدّم السماحيّة لتطبيق الزبون رمز وصول (Access Token) يُمكنه من الوصول لمخدّم الموارد لصالح المُستخدم علماً أنّ البرتوكول يعرّف الأنواع الآتية من الرموز (Tokens):

- رمز الوصول (Access Token): يمكّن تطبيق الزبون من الوصول للموارد المحمية.
- رمز التحديث (Token Refresh) لتحديث رمز الوصول.
- الرمز الحامل (Bearer Token): وهو رمز حماية يمكّن مالكه من الوصول للموارد دون أن يُثبت امتلاكه لأية مفاتيح.

حالات منح السماحية (Grant Flows) في OAuth 2.0 [2]:

من الإضافات الأساسية التي قدمها OAuth 2.0 مقارنة بـ OAuth 1.0 تعريفُ أربع حالاتٍ لمنح السماحيّة لكل منها استخدامات محددة.

- منح رمز السماحية (Authorization Code Grant): يُستخدم لتطبيقات الزبون التي يمكنها الاحتفاظ بِكَلِمَةِ سِرٍّ مِثْلُ التَّطْبِيقَاتِ الَّتِي تَعْمَلُ على مخدم وب (Web Server Applications)، وهو التدفق الأكثر استخداماً. يوضّح الشّكل 1 خطوات هذا التدفق:



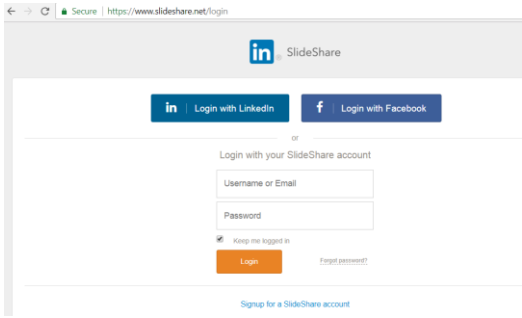
الشكل1: تدفق منح رمز السماحية (Authorization Code Grant Flow) في OAuth2 [3]

A. يوجّه تطبيق الزبون عميل المُستخدم (User-Agent) (المُستعرض) إلى نقطة السماحية (Authorization Endpoint) عند مُخدّم السماحية، مضمنّاً في رابط التوجيه الوسائط الآتية: مُعرفه (Client ID)، المجال (Scope) الذي يُحدد البرتوكول و أية معلومات إضافية مطلوبة من قبل الزبون عن المُستخدم، بالإضافة لحالة محلّية (Local State) (للربط بين الرد و الطلب)، و رابط إعادة توجيه (Redirection URI) الذي يمثل الرابط التي يجب إعادة المعلومات المطلوبة إليه (عنوان تطبيق الزبون عادة). الشكل العام لهذا الطلب كالتالي:

```
GET {Authorization Endpoint}
 ?response_type=code
```

```
&client_id={Client ID}
&redirect_uri={Redirect URI}
&scope={Scopes}
&state={Arbitrary String}
HTTP/1.1
HOST: {Authorization Server}
```

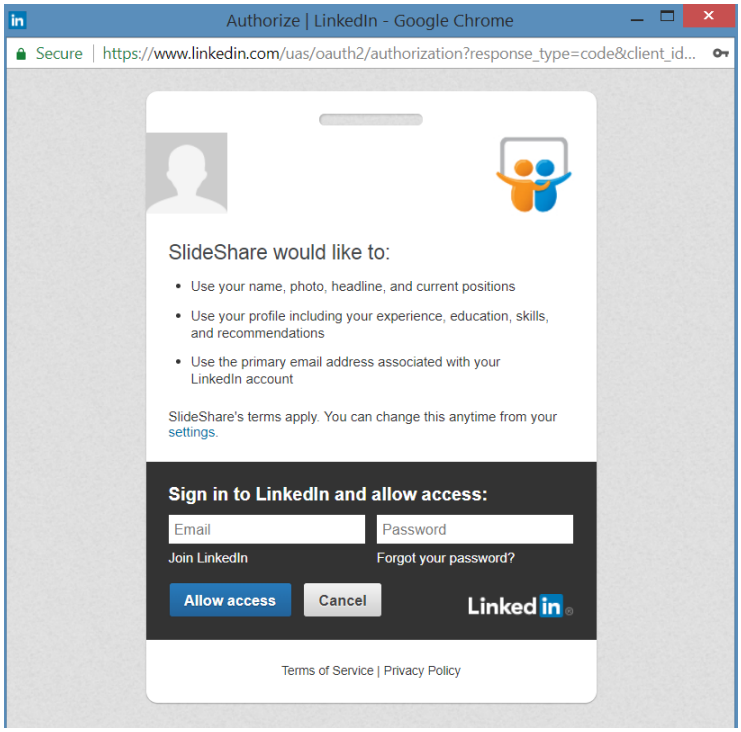
B. يوثّق مُخدّم السماحية مالك الموارد (المستخدم) و يتأكد من موافقته على وصول تطبيق الزبون إلى الموارد. يُمْكِنُ مُلَاحظَةُ هَذَا الخطوة فِي بَعْضِ الْمَوَاقِعِ الَّتِي تَعْتَمِدُ عَلَى مُزَوَّدَاتِ خَدَمَةٍ لِتَوْثِيقِ مُسْتَخْدَمِيهَا مثل: Pinterest ، و SlideShare كما يوضّح الشكل 2:



الشكل2: إمكانية اسناد تَوْثِيقَ الْمُسْتَخْدَمِ إِلَى مُزَوَّدٍ هوية (LinkedIn)

C. في حال منح مالك الموارد حق الوصول لتطبيق الزبون (الشكل 3)، يرسل مخدّم السماحية رد من نوع إعادة توجيه (HTTP 302) يتضمنُ رمز السماحيّة (Authorization Code) و أيّ حالة تمّ تضمينها من قبل الزبون مُسبقاً كما يوضّح الطلب التالي:

```
HTTP/1.1 302 Found
Location: {Redirect URI}
 ?code={Authorization Code}
 &state={Arbitrary String}
```



الشكل 3: طلب السماحية من مالك الموارد

D. يُرْسِلُ الزَّبونُ طَلَب إِلَى نُقْطَةِ الرُّموزِ (Token Endpoint) في مخدّم السماحية يطلب فيه رمز الوصول (Access Token) مُضمناً رمز السماحية و رابط إعادة التوجيه.

```
POST {Token Endpoint} HTTP/1.1
Host: {Authorization Server}
Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code
&code={Authorization Code}
&redirect_uri={Redirect URI}
```

E. يتوثق المخدّم من هوية الزبون و يتأكد أنّ رابط إعادة التوجيه يطابق رابط إعادة التوجيه في الخطوة A، ليرد بعد ذلك برمز وصول

(Access Token) وبشكل اختياري برمز تحديث (Refresh Token) بالإضافة لمجموعة من المعلومات:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "{Access Token}",
  "token_type": "{Token Type}",
  "refresh_token": "{Refresh Token}",  // -
Optional
  ...
}
```

- المنح الضمني (Implicit grant): يُستخدم هذا النمط في التطبيقات التي لايُمكنها الاحتفاظ بإمكانيات الحماية (Credentials) بشكل آمن مثل تطبيق جافا سكريب يعمل ضمن المستعرض. لايتم مَنَحَ رَمْزُ تَحْدِيث فِي هَذَا النَّمَطِ وَ لايتم أَيْضًا التَّوَثُّقِ مِن هُوِيَّةِ الزَّبونِ عِنْدَ مُخَدِّمِ الْوُثُوقِيَّةِ ، كذلك بعكس النمط السابق يستلم تطبيق الزبون رمز الوصول كنتيجة مباشرة لطلب السماحيّة مما يعرّض هذا الرمز للكشف من قبل مالك الموارد أو أي تطبيق يعمل على نفس المضيف.
- المنح باستخدام كلمة سر مالك الموارد (Resource Owner Password Credentials Grant): نمط التدفق الوحيد الذي يتمّ فيه ارسال اسم وكلمة سر المستخدم من قبل تطبيق الزبون للحصول على رمز الوصول و يتطلب ثقة عالية بين مالك الموارد (المستخدم) و تطبيق الزبون.
- المنح باستخدام امكانيات الزبون (Client credentials grant): تُستخدم في هذا النمط إمكانيات حماية تطبيق الزبون (المعرّف و كلمة السّر) أو أي طريقة توثّق أخرى للوصول لموارد تخص تطبيق الزبون نفسه.

المعيار OpenID Connect [5]:

أصبحَ المعيار OpenID Connect معياراً رائداً لتقنية التسجيل لمرة واحدة (Single Sign–On) و لتأمين الهوية (Identity Provision) بُني فوق البرتوكول OAuth 2.0 و إصداره الحالي 1.0. نُشر المعيار OpenID Connect في العام 2014، ورُغم أنّه ليس المعيار الأول لمزود الهوية (IdP) إلا أنه الأفضل من حيث بساطة الاستخدام، مُستفيداً من الدروس التي تم تعلمُها من جهود سابقة مثل SAML، OpenID 1.0 و OpenID 2.0.

وفقاً لهذا المعيار، يُفوض تطبيق الزبون التوثّق من هوية المستخدم إلى مخدّم سماحيّة أو مايسمى "مزود OpenID" ((OpenID Provider (OP))، ليحصلَ بعد ذلك على رمز الهوية للمستخدم باستخدام البرتوكول OAuth 2.0 حيث تُعتبر هوية المستخدم كمورد مقارنة بمصطلحات البرتوكول OAuth 2.0. يتمّ تمثيل هوية المُستخدم أو مايسمى رمزُ هويّة (ID Token) برمز JSON Web Token (JWT) موقّع الكترونياً من قبل المزود ((OpenID Provider (OP)) ليتم التحقق من التوقيع الالكتروني من قبل الأطراف المعنية كما يمكن تشفير الرمز بشكل اختياري لضمان السرية. يؤكّد رمز الهوية على هوية المُستخدم –الفاعل (Subject) (sub)–، كذلك يُحدد الهيئة المصدرة (Authority Issuing) (iss)، وتطبيق الزبون (Particular Audience) (aud)، بالإضافة لعناصر أخرى مثل مميز (nonce)، و وسمة زمنية تدلّ على وقت التوثيق (auth_time)، كذلك كيف تمّ التوثيق من حيث مدى قوته (acr)، أيضاً متى أُصدر الرمز (iat) و متى ينتهي (exp) ومعلومات اختيارية عن المستخدم مثل الاسم و عنوان البريد. كما يبين المثال:

```
{
 "sub"      : "alice",
 "iss"      : "https://openid.c2id.com",
 "aud"      : "client-12345",
 "nonce"    : "n-0S6_WzA2Mj",
 "auth_time" : 1311280969,
 "acr"      : "c2id.loa.hisec",
 "iat"      : 1311280970,
 "exp"      : 1311281970,
}
```

يَتمُّ ترميز التَّرْوِيسَةَ، التَّصْرِيحَاتِ، وَالتَّوْقِيعَ بِاسْتِخْدَام URL–safe string 64 ليكون بالإمكان تمريره كوسيط في رابط URL.

كيف يطلبُ الزبون (Client) والمُسمى الطرف المُعْتَمِد ((Relying Party (RP) رمز الهوية؟ [5]

بداية يتمّ التوثّق من هوية المُستخدم عند مزود الهوية (Identity Provider)، و عادة تُستخدم نوافذ المستعرض المنبثقة التي تستخدمها تطبيقات الوب لتوجيه المستخدم إلى مزود الهوية (IdP) (الشكل 3). من المهم أن يتم التوثق من المستخدم في سياق موثوق و آمن.

يُمكن أن يطلب تطبيق الزبون رمز هوية المُستخدم اعتماداً على البرتوكول OAuth 2.0 وفقاً لنمط من الأنماط الآتية:

- نمط رمز السماحية (Authorisation Code Flow): من أكثر الأنماط استخداماً، يبدأ بإعادة توجيه المُستعرض إلى مزوّد الهويّة (OP) لتوثيق المُستخدم و لنيل مُوافقته، من ثمّ يحصل الزبون على رمز سماحيّة يُمكنه من طلب رمز وصول إلى المورد (هوية المُستخدم). يؤمّن هذا النمط حماية مُثلى حيث أن الرموز لاتظهر في المستعرض كذلك يتم تَوْثِيق تَطْبِيق الزَّبون.
- النمط الضمني (Implicit Flow): للتطبيقات المعتمدة على JavaScript في المستعرض. في هذه الحالة يتم استلام رمز الهوية (ID Token) –ورمز الوصول توافقاً مع OAuth–مُباشرة مع رد إعادة التوجيه (Redirection Response) من المزوّد ولايتمّ إرسال طلب إضافيّ للحصول على رمز الهوية كما في النمط السابق.

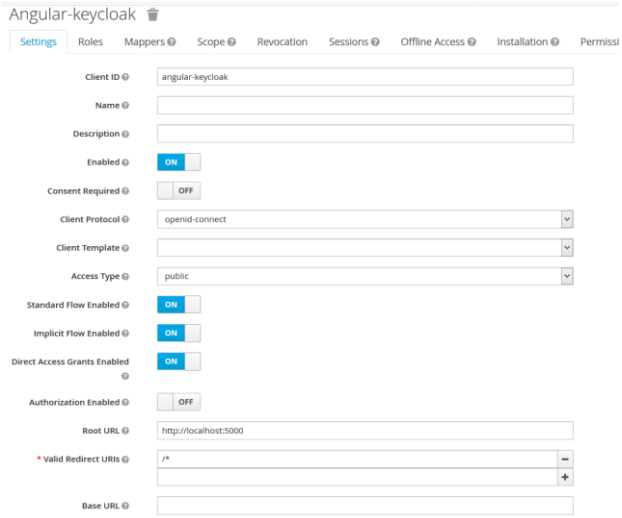
- النمط الهجين (Hybrid Flow): نادراً ما يُستخدم و يَسْمَحُ لِوَاجِهَةِ التَّطْبِيقِ (Front–End) و للأساس (Back–End) أن يستلما رموز بشكل مُستقل كل منهما عن الآخر.

تحليل عملي لرسائل المعيار OpenID Connect:

أسندنا التوثّق من هوية المُستخدم في هذه الدراسة العملية إلى المخدّم Keycloak [8] وهو مخدّم مفتوح المصدر لإدارة الهوية و للتحكم بالوصول واختبرنا حالتين: في الأولى يتم استخدام المنح الضمني أما في الثانية نمط منح السماحية.

- الحالة الأولى: تحليل نمط السماحيّة الضمني

لدراسة نمط السماحيّة الضمني في المعيار OpenID Connect بشكل عمليّ حققنا تطبيق مكوّن من قسمين: تطبيق أساس (Back-End) باستخدام الإطار ASP .NET Core MVC يتضمن خدمة وب و تطبيق واجهة (Front-End) بُني باستخدام Angular5 ليمكّن مستخدم موثّق من قبل مخدم السماحيّة من استعراض مجموعة من الكُتب عن طريق استدعاء طريقة من خدمة الوب في التطبيق الأساس. نوّد الإشارة إلى أنّه رغمّ أن تحقيق تطبيق مشابه موضّح في [6]، إلاّ أننا استخدمنا إصدار مستقل من مخدّم السماحية وطريقة مختلفة لتحقيق الواجهة. لبناء الثقة بين Keycloak و التطبيق تمّ إنشاء زبون عند المخدم معرّف بـ: angular-keycloak كما يوضّح الشكل 4:



الشكل 4: تعريف التطبيق الزبون عند مزود الهوية

أضفنا في تطبيق ASP .NET Core MVC معرّف التطبيق و رابط مزوّد الهوية في الملف appsettings.json:

```
"Jwt":{
   "Authority": "https://localhost:8443/auth/realms/master",
   "Audiance": "angular-keycloak",
   "Issuer": "https://localhost:8443/auth/realms/master"
 },
```

و حددنا أنّ التوثق سيتم من خلال رمز JWT في الصف Startup.cs .

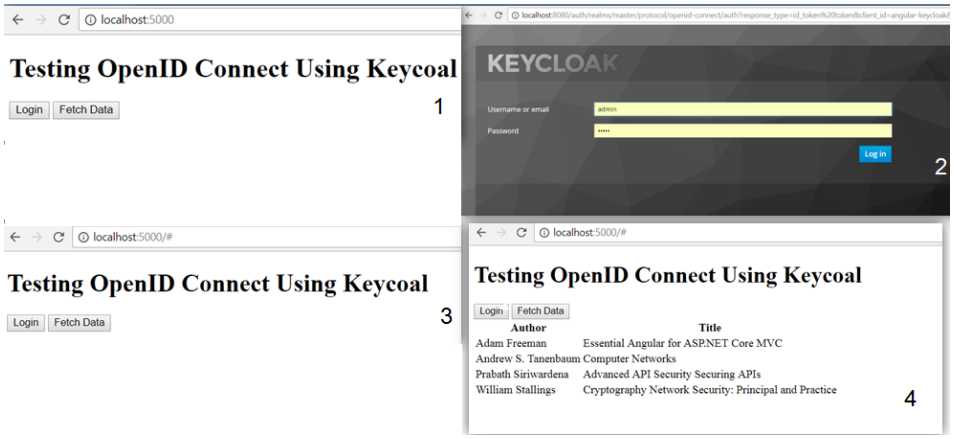
```
...
namespace hello_world
{
   public class Startup
   {
    ...
   public void ConfigureServices(IServiceCollection services)
   {
     services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
       .AddJwtBearer(options =>
       {
        options.Authority = Configuration["Jwt:Authority"];
        options.Audience = Configuration["Jwt:Audience"];
        ...
       });
     services.AddMvc();
     }
      ...

}
```

وكذلك وَسَمَّنَا خِدْمَةُ الوب بِالْكَلِمَةِ الْمِفْتَاحِيَّةِ Authorize. أما في طبقة العرض (Angular) أضفنا ملف التعريف auth.config.ts ذو المحتوى التالي:

```
import { AuthConfig } from 'angular-oauth2-oidc';
export const authConfig: AuthConfig = {
 // Url of the Identity Provider
 issuer: 'http://localhost:8080/auth/realms/master',
 // URL of the SPA to redirect the user to after login
 redirectUri: window.location.origin,
 clientId: 'angular-keycloak',
 scope: 'openid profile email',
 requireHttps: false
}
```

و أضفنا تعليمات للملف app.component.ts بأنّه سيتم استخدام النَّمَطُ الضِّمْنِيُّ للمعيار (initImplicitFlow) و بأنّ الوصول لخدمة الوب يتطلب إرسال رمز JWT يتم الحصول عليه من مخدم keycloak عند نجاح التوثق من المستخدم. يوضّح الشكل 5 مراحل التنفيذ:

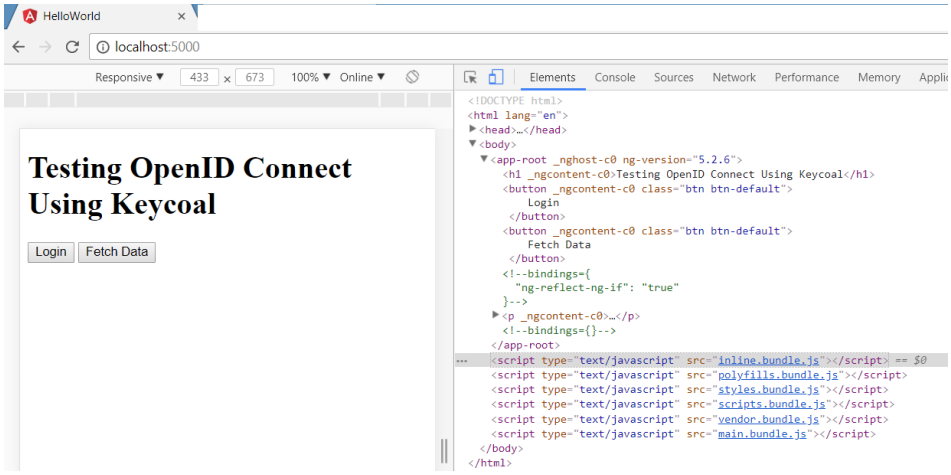


الشكل 5: مراحل تنفيذ التطبيق الأول لاختبار المنح الضمني

كما يبيّن الشكل 5، بداية تظهر النافذة الأولى المُتضمنة لزِرّيّ تَحَكُّم: الأول Login ليتم توثيق المستخدم من قبل مزود الهوية keycloak و الثاني Fetch data لإحضار البيانات من خدمة الوب. عندما يختار المستخدم الزر Login

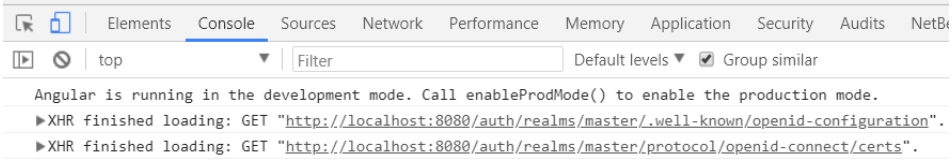
يَتِمُّ تَوْجِيهُ الْمُسْتَعْرِضِ إِلَى مُزَوَّد الهوية، و بعد أن يتم توثيق المستخدم بشكل صحيح يعاد توجيه المُستعرض إلى التطبيق الواجهة ليصبح بإمكان المستخدم طلب البيانات من خدمة الوب.

لتحليل رسائل البرتوكول OpenID Connect التقطنا الرزم باستخدام البرنامج Wireshark و استخدمنا الأداة Chrome DevTools [4] لتتبع الطلبات الصادرة من واجهة التطبيق. يبيّن الشّكل 6 واجهة التطبيق و الأداة Chrome DevTools:



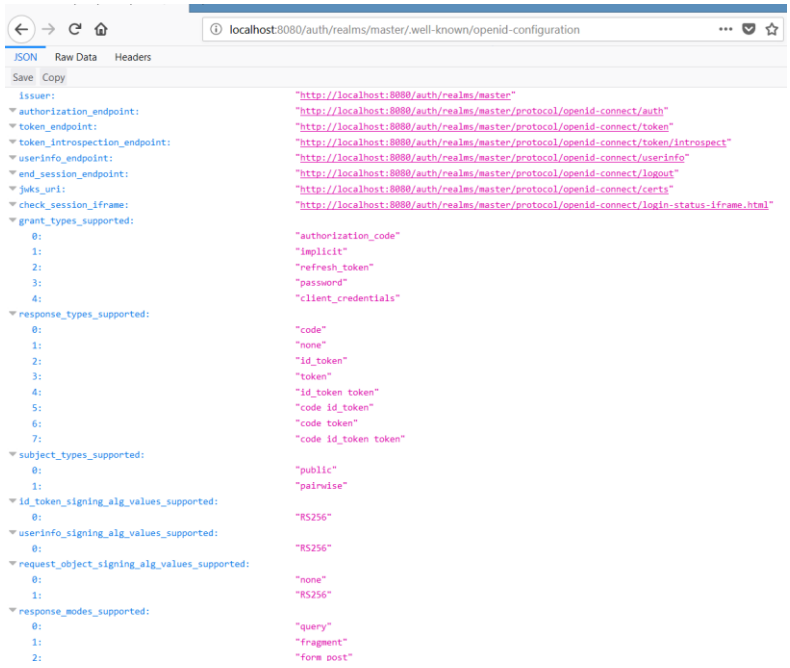
الشكل 6: واجهة التطبيق

لاحظنا أنه تمّ طلب ملفين من المخدم Keycloak (الشكل 7):



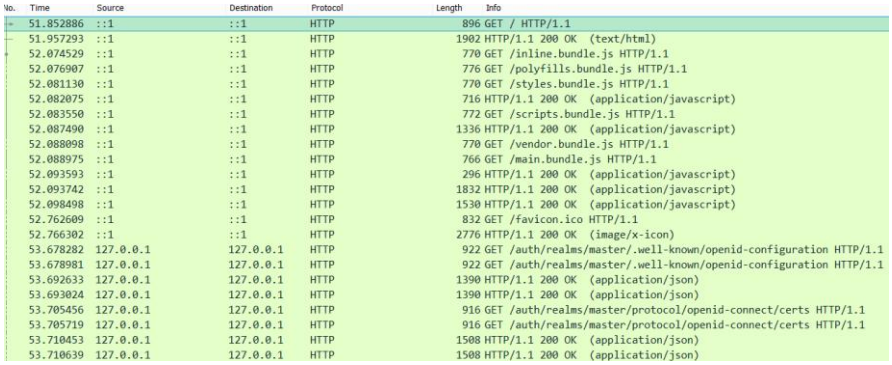
الشكل 7: تتبع طلب المعلومات الأساسية من مخدمّ السماحية

يتضمنُ الملف الأول التعريفات الأساسية عند مخدّم السماحية مثل عنوان النقطة المسؤولة عن توثيق المستخدم (Auth) و النقطة التي تمنح الرموز (Tokens) كما يوضّح الشكل 8، و الرابط الثاني يوصل للشهادة الرقمية لمخدّم السماحيّة.



الشكل 8: المعلومات الأساسية

بالتوازي مع تنفيذ التطبيق تمّ تشغيل أداة التقاط الرزم كما يوضح الشكل 9:



الشكل 9: تشغيل أداة التقاط الرزم

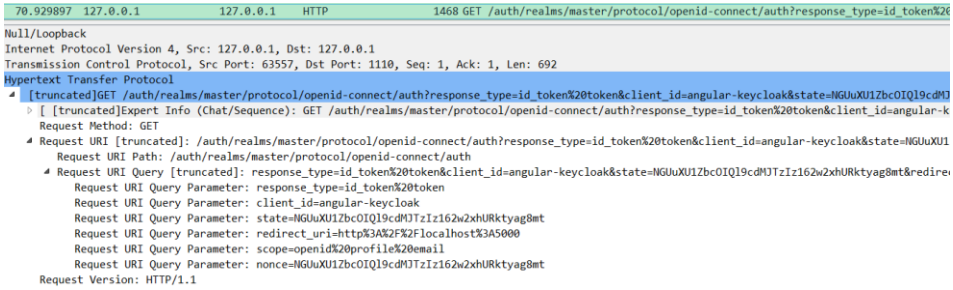
النقر على زر التسجيل (Login):

عند النقر على زر التسجيل في واجهة التطبيق يتم إرسال الطلب التالي إلى نقطة السماحية في البرتوكول:

```
[truncated]Expert Info (Chat/Sequence):
GET /auth/realms/master/protocol/openid-connect/auth
```

مع مجموعة وسائط الاستعلام (Request URI Query Parameter) الآتية و الموضحة أيضاً في الشكل 10 لطلب رمز الوصول و رمز الهوية:

```
Request URI Query [truncated]:
response_type=id_token%20token&
client_id=angular-keycloak&
state=NGUuXU1ZbcOIQl9cdMJTzIz162w2xhURktyag8mt&
redirect_uri=http%3A%2F%2Flocalhost%3A5000&
scope=openid%20profile%20email&
nonce=NGUuXU1ZbcOIQl9cdMJTzIz
```



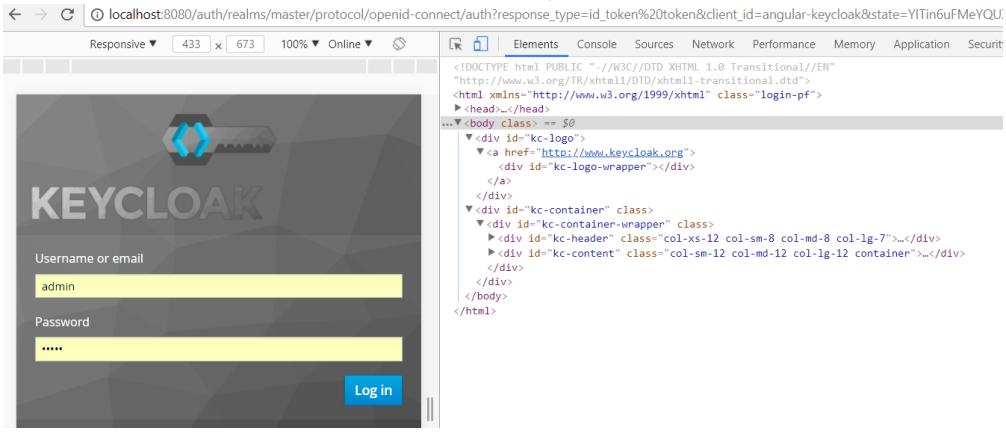
الشكل 10: الطلب الابتدائي لرمز الوصول و لرمز الهوية

- response_type: قيمته id_token مما يدل على طلب رمز الهوية من نقطة السماحية (Authorization Endpoint) في البرتوكول.
- client_id: معرّف الزبون عند مخدم السماحية (angular-keycloak).
- state: قيمة مميزة لربط الرد مع الطلب.
- redirect_uri: عنوان التطبيق الذي ستتم اعادة التوجيه إليه بعد نجاح توثيق المستخدم و إرسال المعلومات المطلوبة إليه.
- scope: وقيمته (openid%20profile%20email). openid يعني أن هذا طلب برتوكول openid و بدون هذه القيمة لايتمّ التعامل مع هذا الطلب كطلب برتوكول openid (أي القيمة اجبارية)، أما profile و email فهي قيّم اختيارية الأولى تطلب منح الوصول للمعلومات المتعلقة بالمستخدم (Claims)

و المتاحة من خلال نقطة النهاية للمستخدم (UserInfo Endpoint) مع رمز الوصول (Access Token) و القيمة الثانية تطلب الوصول لعنوان البريد الالكتروني.

- nonce: لمنع هجمة إعادة الإرسال.

يؤدي هذا الطلب لعرض الواجهة المبينة في الشكل 11:



الشكل 11: توثيق المستخدم من قبل مزود الهوية

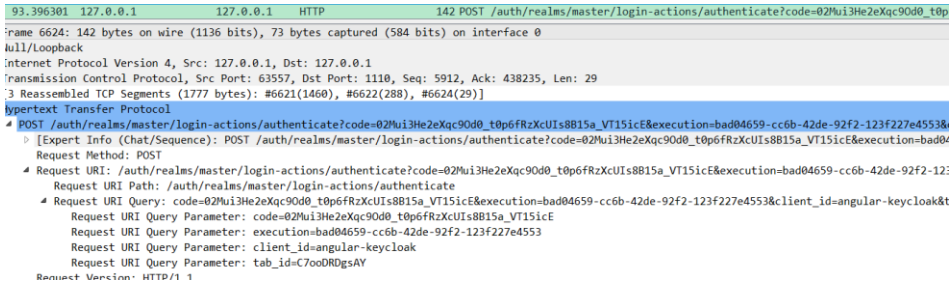
لاحظنا أن وسمة النموذج (form) تتضمن وسيط (Code) يمكن أن يستخدم للحماية من هجمات مثل تزوير الطلبات عبر المواقع (Cross-Site Request (CSRF Forgery:

```
<form id="kc-form-login" class="form-horizontal" onsubmit="login.disabled =
true; return true;" action="http://localhost:8080/auth/realms/master/login-
actions/authenticate?code=02Mui3He2eXqc9Od0_t0p6fRzXcUIs8B15a_VT
...
```

بعد أن يملأ المستخدم النموذج مدخلاً اسم المستخدم وكلمة السر و يضغط زر الارسال، يتمّ إرسال طلب التوثيق التالي للمخدّم و الموضّح أيضاً في الشكل 12:

```
POST/auth/realms/master/login-actions/authenticate?
code=02Mui3He2eXqc9Od0_t0p6fRzXcUIs8B15a_VT15icE&
execution=bad04659-cc6b-42de-92f2-123f227e4553&
client_id=angular-keycloak&
tab_id=C7ooDRDgsAY HTTP/1.1\r\n
```

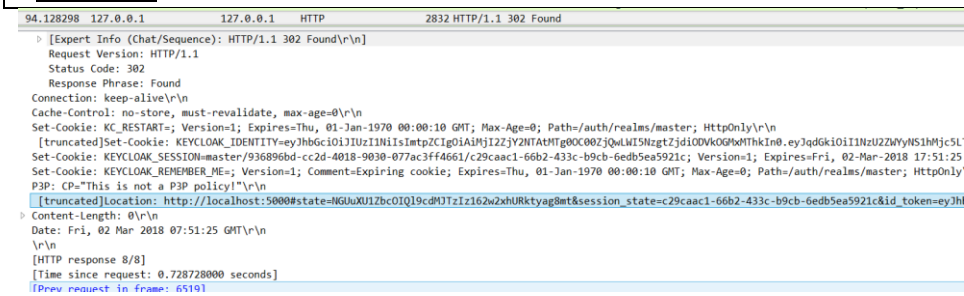
...
HTML Form URL Encoded: application/x-www-form-urlencoded
"Form item: "username" = "admin
"Form item: "password" = "admin



الشكل 12: إرسال اسم المستخدم و كلمة السّر

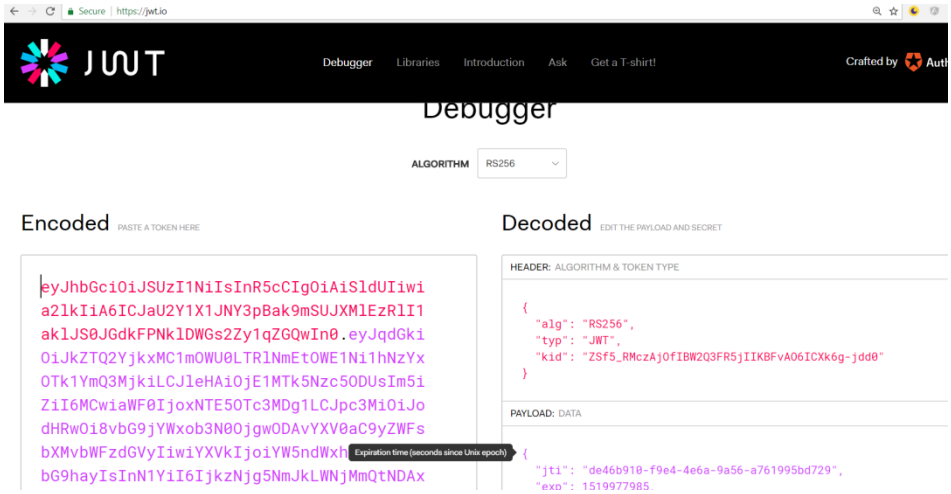
يؤدي نجاح التوثق من هوية المستخدم إلى إرسال الرموز إلى تطبيق الزبون من خلال إعادة توجيه (الشكل 13):

Expert Info (Chat/Sequence): HTTP/1.1 302 Found\r\n
http://localhost:5000#
state=NGUuXU1ZbcOIQl9cdyag8mt
&session_state=c29caac1−66b2
&id_token= eyJhbGciOiJSUzI1…21URN6Qxsb279…4tiS…
&access_token=….FsbXMvbWFzdGVyIiwiYXVkIjoiYW5ndWxhci1rZXljbG9hayIsInN1YiI6IjkzNjg5NmJkLp1Wq3ITToBqERLnQ
&token_type=bearer
&expires_in=900&not−before−policy=0



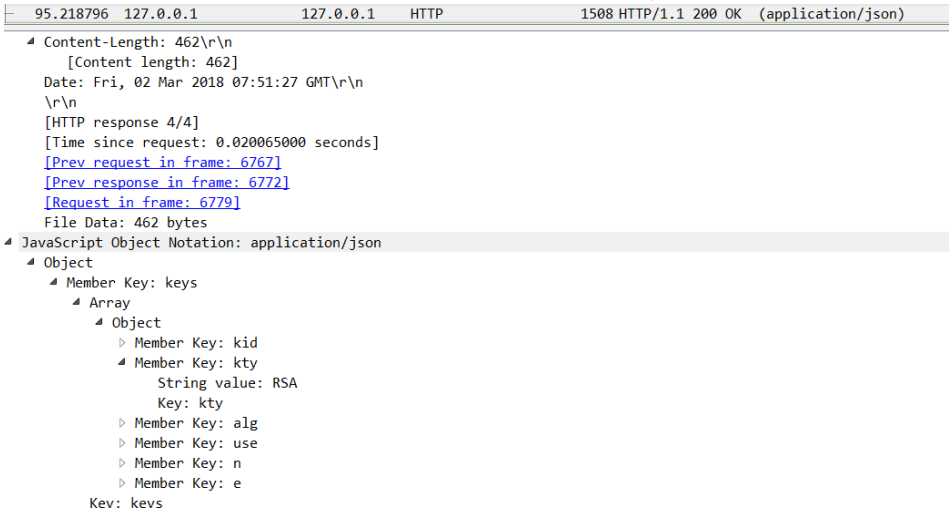
الشكل 13: إرسال رمز الوصول و رمز الهوية لتطبيق الزبون

يمكننا فك ترميز رمز الهوية من خلال موقع مثل (jwt.io) كما يوضح الشكل 14:



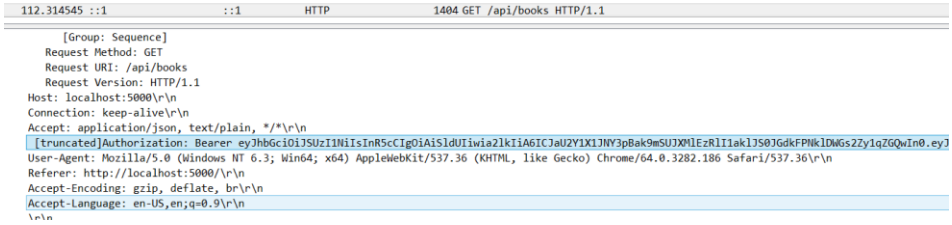
الشكل 14: فك ترميز jwt

نلاحظ بعد ذلك إعادة إرسال معلومات الشهادة الرقمية (الشكل 15) حتى يتأكد تطبيق الزبون من التوقيع الالكتروني.



الشكل 15: مكونات الشهادة الرقمية

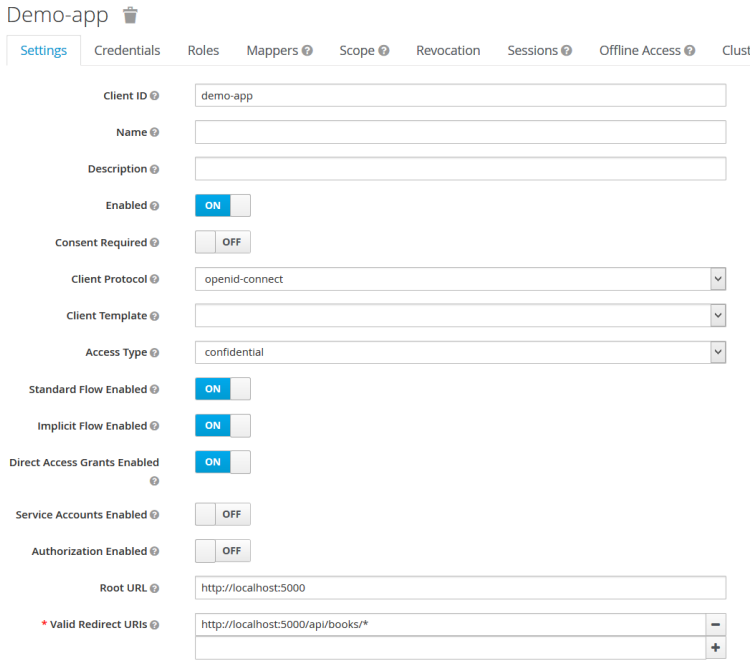
وفي الخطوة الأخيرة يتم ارسال الطلب مع رمز الوصول إلى خدمة الوب ليأتي الرد بالبيانات (الشكل 16):



الشكل 16: إحضار البيانات

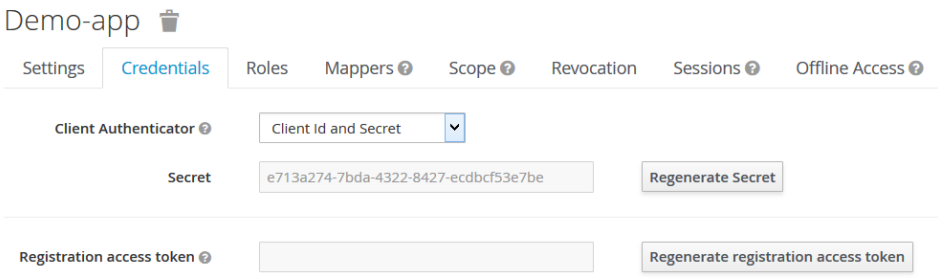
- الحالة الثانية: تحليل نمط منح السماحية

لتحليل نمط منح السماحية حققنا تطبيق ASP .NET Core MVC يتضمنُ خدمة وب بدون طبقة العرض المستخدمة في الحالة الأولى. عرفنا التطبيق في مخدم السماحيّة (demo-app) كما يوضح الشكل 17:



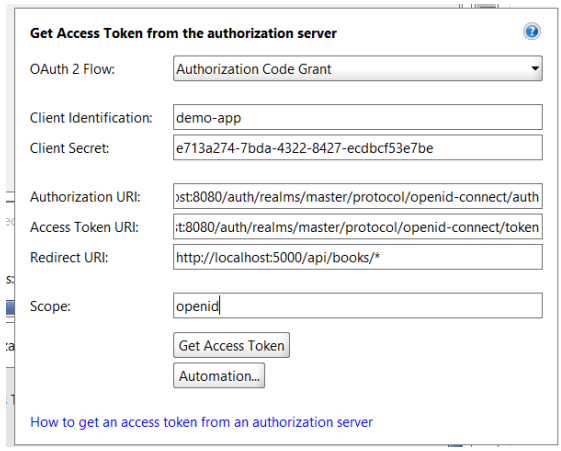
الشكل 17: تعريف التطبيق في حالة الاختبار الثانية

وعرفنا له كلمة سر على المخدّم لأنّ تطبيق الزبون في هذه الحالة يحتفظ بإمكانيات حماية ليتم التوثق من هويته أيضاً من قبل مخدّم السماحية (الشكل 18) :



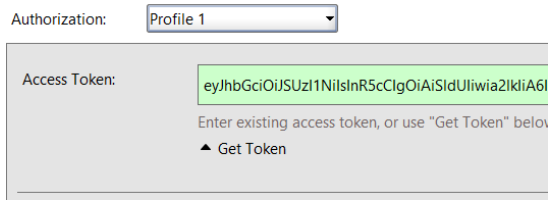
الشكل 18: اسم وكلمة سر تطبيق الزبون

استخدمنا الأداة soapui [7] للحصول على رمز وصول من المخدّم Keycloak كما يوضح الشكل 19. هذه الأداة تحاكي تطبيق الزبون في الحصول على رمز وصول يمكن أن نستخدمه لاحقاً عند استدعاء الخدمة من ضمن نفس الأداة.



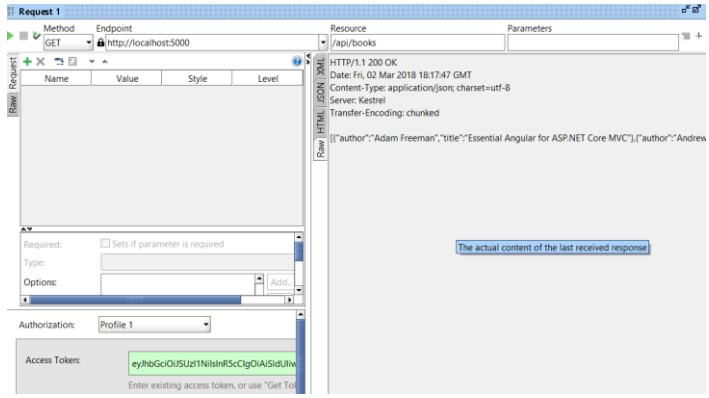
الشكل 19: استخدام soapui للحصول على رمز وصول من مخدم السماحيّة

عند إرسال الطلب من soapui، تظهر نافذة التوثق من هوية المستخدم عند مخدّم السماحيّة وعند نجاح التوثق من هوية المستخدم يتم الحصول على رمز الوصول كما يوضح الشكل 20:



الشكل 20: الحصول على رمز الهوية

استدعينا بعد ذلك الطريقة التي تسمح باستعراض الكتب في خدمة الوب بعد ارفاق رمز الوصول مع الطلب و قمنا بالتقاط الرزم بالتوازي مع الطلب (الشكل 21).



الشكل 21: استدعاء خدمة الوب

تحليل رسائل البرتوكول:

بداية يتم إرسال الطلب التالي إلى نقطة السماحية في البرتوكول لطلب رمز سماحيّة.

يتضمن الطلب عدد من الوسائط تم شرح دلالتها سابقاً.

```
Request Method: GET
Request URI: /auth/realms/master/protocol/openid-connect/auth?
scope=openid&
response_type=code&
redirect_uri=http%3A%2F%2Flocalhost%3A5000%2Fapi%2Fbooks%2F*&
client_id=demo-app
```

كرد يتم إرسال نموذج توثيق المستخدم عند مخدّم السماحيّة كما في الحالة السابقة أيضاً.

عندما يسجّل المستخدم في النافذة يتم إرسال الطلب التالي:

```
POST /auth/realms/master/login-actions/authenticate?
code=NkTHAzuBo1MrvDrfXibfsHqD09wRCshnpYEnYSmd6oE&execution=bad0
4659-cc6b-42de-92f2-123f227e4553&
client_id=demo-app&
tab_id=DU99wii0ky8 HTTP/1.1\r\n
Referer: http://localhost:8080/auth/realms/master/protocol/openid-
connect/auth?scope=openid&response_type=code&redirect_uri=http%3A%2F%
```

```
2Flocalhost%3A5000%2Fapi%2Fbooks%2F*&client_id=demo-app\r\n
…
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "username" = "admin"
Key: username
Value: admin
Form item: "password" = "admin"
Key: password
Value: admin
```

و عند نجاح التوثق من هوية المستخدم يتم إرسال رمز السماحية كما هو موضح:

```
HTTP/1.1 302 Found\r\n
…
[truncated]Location:
http://localhost:5000/api/books/*?session_state=60389ae9-77b8-4857-
987e-
035093457f80&code=eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
..91kbksL8q_O4PFkWjJlIEg.xSV0Vy8oxerfu1En2KRVcqLKequ-
RukB_OCSx_pxDiJylcBsAASRJnR
```

بعد ذلك يتوجه الزبون إلى نقطة منح الرموز في مخدّم السماحيّة مضمناً اسم الزبون وكلمة سره حتى يتم التوثق من هوية الزبون من قبل مخدم السماحية ويرسل رمز السماحية الذي حصل عليه من الرسالة السابقة:

```
POST /auth/realms/master/protocol/openid-connect/token HTTP/1.1\r\n
                                                                    …
HTML Form URL Encoded: application/x-www-form-urlencoded
"Form item: "client_secret" = "e713a274-…cf53e7be
…
"Form item: "grant_type" = "authorization_code
Key: grant_type
Value: authorization_code
"*/Form item: "redirect_uri" = "http://localhost:5000/api/books
```

Key: redirect_uri

*/Value: http://localhost:5000/api/books

item: "code" =

"eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..91kbksL8q_O4PFk

WjJIIEg....3gqf-53d

Key: code

Value [truncated]:

eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..91kbksL8q_O4PFk

WjJIIEg.xSV0Vy8oxerfu1En2KRVcqLKequ-

RukB_OCSx_pxDiJylcBsAASRJnRho7bYkzDpqzLRN0STgYMxNcWrcbKcW9-

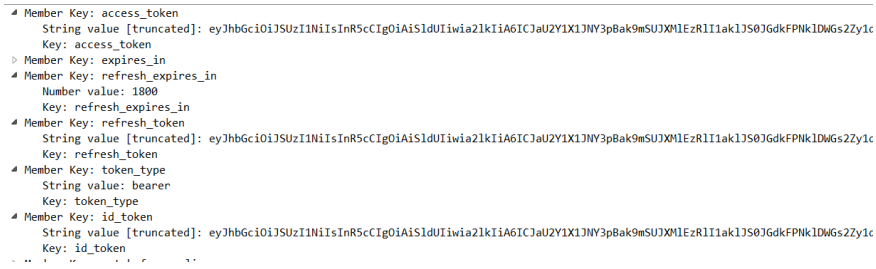
IsauEj0PUsQRt-V-lOzemXgVXrDPN5u_vEdWTmiTOS4uwiAfg9s3gqf-53d4D

"Form item: "client_id" = "demo-app

Key: client_id

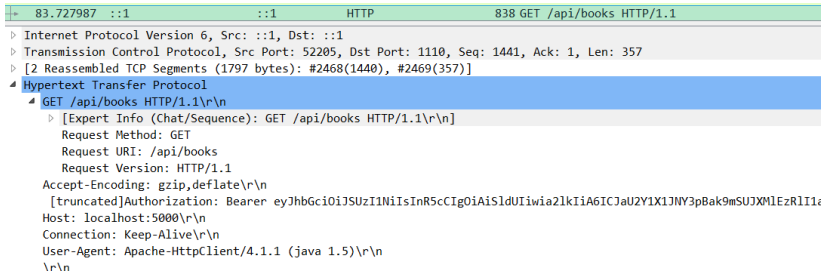
Value: demo-app

كرد يصل للزبون رمز وصول ورمز هوية كما توضح الرسالة التالية (الشكل 22) ، لاحظنا أن للرموز الثلاثة نفس القيمة.



الشكل 22: وصول رمز الوصول و رمز الهوية لتطبيق الزبون

بعد الحصول على رمز الوصول يصبح بإمكان التطبيق استدعاء الخدمة (الشكل 23).



الشكل 23: استدعاء الخدمة

الخاتمة و التوصيات المستقبلية:

قدمنا في هذا البحث دراسة عملية و نظرية لرسائل المعيار OpenID Connect و هو معيار رائد لتقنية التسجيل لمرة واحدة (Single Sign-On) و لتأمين الهوية (Identity Provision) بُني فوق البرتوكول OAuth 2.0 و إصداره الحالي 1.0. يمكن في الآفاق المستقبلية لتطوير هذا العمل التركيز على جوانب تقييم الأداء و أمن المعيار بحد ذاته.

المراجع:

[1]. The Java EE 7 Tutorial. Release 7 for Java EE Platform, E39031-01, June 2013.

URL: http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

[2]. Understanding OAuth2, by Johann Reinke.

URL: http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/

Last retrieved: 2/5/2018.

[3]. OAUTH 2.0 Server.

URL: http://jlabusch.github.io/oauth2-server/, Last accessed: 2/5/2018.

[4]. Chrome DevTools Overview - Google Chrome

URL: https://developer.chrome.com/devtools, Last accessed: 2/5/2018.

[5]. OpenID Connect explained

URL: https://connect2id.com/learn/openid-connect, Last accessed: 2/5/2018.

[6]. ASP.Net Core & Angular OpenID Connect using Keycloak, by Xavier Hahn. Sep 26, 2017.

URL: https://medium.com/@xavier.hahn/asp-net-core-angular-openid-connect-using-keycloak-6437948c008

Last accessed: 2/5/2018.

[7]. The Complete API Test Automation Framework for SOAP, REST and More. URL: https://www.soapui.org/

Last accessed: 3/5/2018.

[8]. Keycloak: Open Source Identity and Access Management For Modern Applications and Services. URL: https://www.keycloak.org/

Last accessed: 3/5/2018.
[9]. OpenID Connect, http://openid.net/connect/
Last retrieved: 2/5/2018.
[10]. Andres C.Salazar and Rene Enriquez (2014), RESTful Java Web Services Security, Packt Publishing.