Desarrollo e implementación de una aplicación de autenticación SSO en Node.js

Antonia Retamal Cañiz

Resumen—Gran parte de las aplicaciones y servicios web requieren que los usuarios se identifiquen mediante un método de control de acceso. Su implementación puede llegar a ser un problema para las organizaciones, ya que en ocasiones presentan vulnerabilidades de seguridad. A demás, la identificación también supone una dificultad para los usuarios debido a que deben memorizar un gran número de contraseñas. En este paper se propone la implementación y despliegue de un proveedor de autenticación, habilitado para dar acceso a la identidad del usuario a distintas aplicaciones web. Para su implementación se sigue el estándar del protocolo OAuth2, uno de los protocolos de seguridad más extendidos, que permite el acceso a los recursos de APIs (Application Programming Interface) en nombre del usuario a aplicaciones terceras. Para su correcta implementación se hace una evaluación de los ataques que puede sufrir el sistema y las técnicas para mitigarlos, ya que la seguridad es fundamental para garantizar la protección de los datos.

Palabras clave— Autenticación, Autorización, OAuth2, protocolo, proveedor, servidor de autorización, identificación, login, delegación a terceros, single sign-on, sso, control de acceso.

Abstract— A large part of web applications and services require users to authenticate using access control methods. Its implementation would be a problem for organizations because in some cases they present important security vulnerabilities. Moreover, the authentication would also be a problem for users due to the fact that they have to remember a significant number of passwords. This paper describes an implementation and integration of an authorization provider, following the OAuth2 standard, one of the most widely used delegation protocols. It provides user identity access management for different web applications. An attack and risk evaluation and its mitigation measures have been made for its proper implementation, since security is the main concern to guarantee the user data protection.

Index Terms— Authentication, Authorization, OAuth2, protocol, provider, authorization server, login, third-party delegation, single sign-on, sso, access control.

1 Introducción

A pesar de que existen múltiples métodos de autenticación de usuarios en los sistemas todos persiguen el mismo objetivo: autenticar que esa persona es realmente quien dice ser. A demás, algunas aplicaciones seguidamente comprueban si la identidad del usuario autenticado dispone de permiso para acceder al recurso solicitado, este proceso se conoce como autorización. Algunas organizaciones y organismos disponen de más de una aplicación o servicios web en los que el usuario se identifica. Dado el gran número de usuarios, aplicaciones y servicios, existe una solución que permite centralizar la gestión y administración de usuarios, reduciendo su mantenimiento, está estrategia se conoce como SSO (Single Sign-on).

Existen plataformas y aplicaciones llamadas API (interfaz de programación de aplicaciones) que se comunican con otras aplicaciones, facilitando el acceso a sus datos o servicios de forma segura. Una solución utilizada por las APIs y aplicaciones para externalizar la gestión de la autenticación de los usuarios y las contraseñas es el login social [1], en el que los usuarios se loguean a través de las redes sociales más populares, como Facebook o Google. De este modo se evita la memorización de las contraseñas por parte de los usuarios, lo que en ocasiones es una fuente de ataques de robo de los datos [2]. Para que el usuario inicie sesión a través de una red social que actúa como servidor de autenticación o proveedor de identidad, esta debe ser de confianza y familiarizada con el usuario ya que el usuario permite a un tercero, en este caso a una aplicación cliente en la que se identifica, acceder parcialmente a su información.

En este proyecto se propone la implementación de una aplicación proveedora de identidad y autenticación, habilitada para dar acceso a la identidad del usuario a otras páginas web. Llevar a cabo el proyecto supone un gran reto ya que el protocolo OAuth2, que permite la implementación del sistema propuesto fue publicado en el año 2012, en principio para homogeneizar la autorización de las APIs utilizadas por las aplicaciones web. No obstante, debido al crecimiento del uso de las aplicaciones y servicios web y de los dispositivos del IoT (Internet of Things) el uso del protocolo se ha extendido. Uno de los motivos principales que justifican este trabajo es su fácil integración con cualquier tipo de aplicaciones y nuevas tecnologías.

- *E-mail de contacto: antoniasqm@gmail.com*

- *Mención realizada: Tecnologías de la Información.*
- *Proyecto tutorizado por: Sergi Robles*
- *Curso 2017/2018*

1.1 Objetivos

El objetivo principal del proyecto es diseñar, implementar y desplegar un sistema de autenticación que proporcione a distintas webs la gestión del acceso de los usuarios a los recursos protegidos. Para alcanzar el objetivo principal se deberá en orden de prioridad decreciente cumplir los siguientes objetivos:

- Análisis de las soluciones y tecnologías existentes en el mercado que autentiquen a los usuarios de una aplicación web.
- Estudio de los lenguajes de programación, frameworks, estándares y librerías seleccionados para la implementación de la aplicación.
- Diseño e implementación del servidor OAuth 2.0 utilizando los lenguajes de programación y tecnologías seleccionados.
- Consideración de los ataques y brechas de seguridad de los sistemas de autenticación y búsqueda de las soluciones para su mitigación.
- Desarrollo de aplicaciones cliente de prueba para testear el correcto funcionamiento del sistema implementado.

1.2 Estructura del documento

Este document está estructurado de la siguiente manera. En el apartado 2 se exponen los protocolos y estándares de autenticación y autorización y las soluciones existentes en el mercado. En el apartado 3 se presenta la metodología y planificación del proyecto propuestos y utilizados. A continuación, en el apartado 4 se muestra el diseño y los aspectos clave de la implementación del proveedor de identidad. En el apartado 5 se exponen las pruebas realizadas y los resultados obtenidos. Finalmente, el apartado 6 concluye con la exposición de las conclusiones obtenidas y líneas futuras.

2 ESTADO DEL ARTE

Los protocolos de seguridad más utilizados para implementar el proveedor de identidad que permite autenticar y autorizar a las aplicaciones y usuarios son SAML, OAuth, LDAP y Open ID Connect. Dichos protocolos definen el esquema de comunicación y el flujo de datos entre el proveedor de identidad o servidor de autenticación y las aplicaciones cliente. En el primer punto se muestran los conceptos básicos del protocolo SAML2 y el protocolo OAuth2, dos de los protocolos más utilizados. En el segundo punto se listan las soluciones AaaS (Autenticación como un servicio) existentes en el mercado.

2.1 Protocolos de autenticación y autorización

a) SAML 2.0

SAML2.0 es un protocolo de autorización y autenticación que permite a las empresas la implementación de sistemas SSO (Single Sign-On), en aplicaciones empresa-empresa y empresa-consumidor. Según la especificación elaborada por Oasis [3], la infraestructura en SAML2.0 está compuesta por las siguientes entidades:

- el proveedor de identidad (idP), que es la autoridad SAML que autentica al usuario y por lo tanto gestiona la información de la identidad de los sujetos.
- el proveedor de servicios (SP), que es el consumidor SAML y responsable de proporcionar al usuario el acceso a un recurso o aplicación. Las acciones realizadas por el proveedor de servicios se basan en la información que le proporciona el proveedor de identidad.
- la aplicación cliente es la responsable de realizar las peticiones de autenticación al proveedor de identidad.

Como se observa en la *Figura 1*, la aplicación cliente solicita al proveedor de servicio la autenticación del usuario y este redirige al usuario al proveedor de identidad para validar la autenticación. Una vez el usuario es validado, el proveedor de identidad envía al proveedor de servicios la identidad del usuario, entre otros datos para validar y autorizar la aplicación cliente, como paso previo a dar acceso al recurso al usuario.

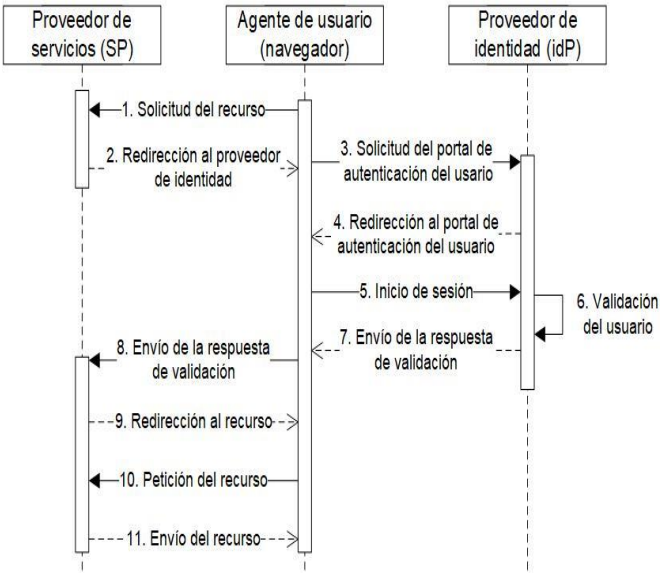

Fig. 1. Flujo de información del protocolo SAML2.

b) OAuth 2.0

OAuth2.0 es un protocolo de autorización orientado a otorgar a los propietarios de los recursos (los usuarios) el acceso a esos recursos restringidos. En el estándar del protocolo [4] se definen los distintos flujos de comunicación entre los actores implicados, los cuales son:

- el propietario del recurso o el usuario.
- el cliente, que es la aplicación que solicita el recurso.
- el servidor de autorización.
- el servidor de recursos, que contiene el recurso solicitado por el cliente (la aplicación).

En la *figura 2* se muestra la relación entre los diferentes actores descritos anteriormente.

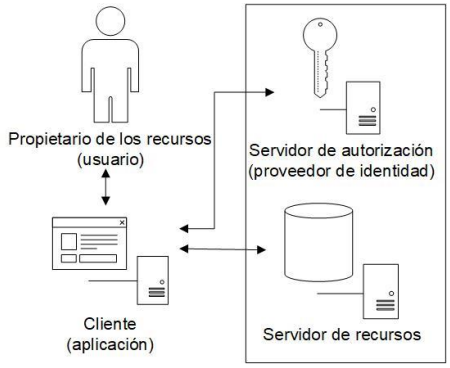

Fig. 2. Actores implicados en el protocolo OAuth2

La aplicación cliente debe estar registrada en el servidor de autenticación y tener asignada una id de cliente y una contraseña. En el diagrama de secuencia de la *figura 3* se muestra el flujo de datos de cuando un usuario presiona el botón de login o acceso y la aplicación cliente solicita al servidor de autorización la autenticación del usuario, aportándole la id de la aplicación y la URL de redirección. El servidor de autenticación redirecciona al usuario al portal de autenticación y si es válida le devuelve a la aplicación cliente el código de autorización para intercambiarlo por el token de acceso al recurso.

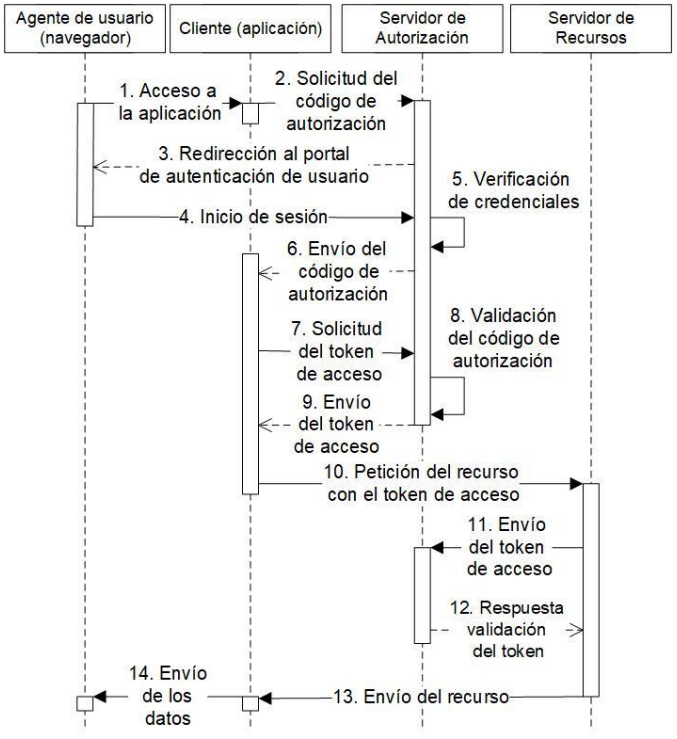

Fig. 3. Flujo de comunicación del protocol OAuth2

Dados los dos protocolos descritos anteriormente, los cuales cubren las necesidades de la aplicación a desarrollar, se ha seleccionado el estándar del protocolo OAuth2. La razón principal de la elección es por ser más simple de implementar y tener una curva de aprendizaje menor, dada su mayor estandarización.

2.2 Soluciones existentes

La autenticación de usuarios y el SSO son caros y complejos de desarrollar por parte de las empresas. Algunas de ellas, en vez de implementar el sistema que gestione la autenticación de usuarios y exponerse a posibles vulnerabilidades en sus aplicaciones contratan la autenticación como un servicio (AaaS). Los proveedores de autenticación existentes en el mercado, según la cantidad de usuarios, tienen asignados unas cuotas anuales y como ha sucedido en algunas ocasiones al dejar de ser servicios rentables pueden desaparecer, como ocurrió con la startup Stormpath [5].

Existen distintos proveedores de autenticación: mientras que algunos están dirigidos a la gestión interna de los negocios, otros se centran en la gestión de las aplicaciones que utilizan los clientes de un negocio. Algunos de los proveedores más populares son los siguientes:
Okta *1*: es un servicio que proporciona y simplifica la administración de los sistemas de identidad. Los productos que ofrece más destacados son SSO (Single Sign-On) y la autenticación multifactor.
Auth0 *2*: es una plataforma en la nube que ofrece la autenticación y la autorización como un servicio. Auth0 dispone de herramientas para simplificar la autenticación de las aplicaciones y APIs ya que hace uso de estándares como OAuth2.0, OpenID Connect, SAML 2.0, JSON Web Token o WS-Federation, ofreciendo SSO (Single Sign-On) a entornos empresariales.
Gluu *3*: es una plataforma de código abierto gratuita que proporciona a las organizaciones un servicio de autenticación y autorización para las aplicaciones web y móvil. Permite configurar SSO (Single Sign-On) en aplicaciones que tengan soporte para OpenID Connect, SAML o CAS para identidades federadas.
AWS Cognito *4*: es un servicio que proporciona control de acceso y gestión de usuarios para aplicaciones utilizando los estándares de administración de identidades.

3 METODOLOGÍA

3.1 Metodología

En este apartado se da una explicación general de la metodología aplicada y los entornos de desarrollo utilizados. Para el desarrollo del proyecto se ha seguido una metodología agile e iterativa. De acuerdo con Alistair Cockburn [6], la metodología iterativa sigue el mismo principio que la tradicional, con la diferencia que una vez se han completado las etapas de forma secuencial y se obtiene el resultado, este vuelve a repetir las etapas de nuevo. Esto permite mejorar el producto final incrementalmente. Tras el análisis de las características del proyecto se decidió poner en práctica esta metodología debido a que ofrece una mejora continua en el desarrollo del sistema y permite tener control del progreso del proyecto, al estar dividido en distintas etapas. A su vez, durante la fase de desarrollo se

1 Okta: https://www.okta.com/
2 Auth0: https://auth0.com/
3 Gluu: https://www.gluu.org/
4 AWS Cognito: https://aws.amazon.com/es/cognito

aplicó el método Kanban, que consiste en listar las tareas a realizar según los requisitos del proyecto, asignándole a cada tarea un peso de trabajo en horas. Este enfoque ha sido aplicado debido a que permite tener un mayor control de las tareas pendientes, en proceso y finalizadas y por lo tanto tener conocimiento del nivel de logro de los objetivos y requisitos establecidos. A fin de verificar que cada uno de los requisitos del proyecto se aplicasen correctamente, una vez ejecutadas y completadas las tareas vinculadas a un requisito, se han realizado distintas pruebas para detectar posibles errores y comprobar que el resultado obtenido se atiene a las funcionalidades esperadas. De este modo, no solo se han obtenido versiones funcionales en un período corto de tiempo, sino que también se ha comprobado mediante distintos tipos de test que las nuevas funciones implementadas no presentan errores ni brechas de seguridad.

Para tener control del código desarrollado se ha utilizado git como software de control de versiones. Al inicio del proyecto se utilizó Atom como IDE de desarrollo, no obstante, se decidió substituirlo por Visual Studio para poder depurar el código y solucionar los errores surgidos más eficazmente.

3.2 Planificación

Para el proyecto se ha estimado un tiempo de dedicación de 300 horas y un período de realización de 150 días. Para hacer una correcta planificación del proyecto este se ha dividido en distintas fases con un período de tiempo asignado. A priori, asignar el total de horas de duración de cada una de estas fases es una tarea complicada y debido a eso se ha hecho una estimación optimista, para disponer de un margen de tiempo.

En la *figura 4* se observa el diagrama de Gannt del proyecto, con las fechas de finalización esperadas de cada etapa.

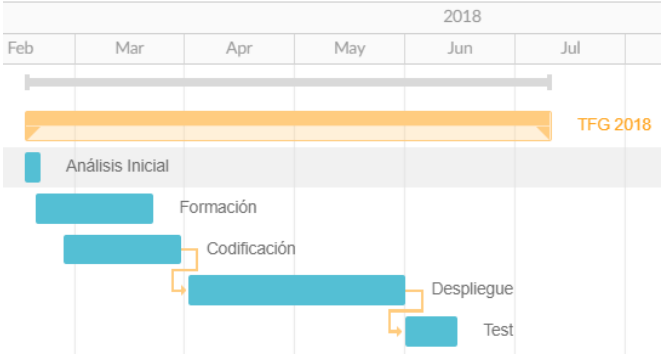

Fig. 4. Diagrama de Gannt del proyecto

El proyecto se replanificó ya que tuvo un desvío de una semana de retraso debido, en gran parte, a la tarea de creación de las aplicaciones cliente, perteneciente a la fase de despliegue. En la fase de planificación, a esta tarea se consideró destinar únicamente 10 horas de trabajo, siendo esta cifra en realidad mayor. Por consiguiente, se consideró incrementar en 10 horas el tiempo de dedicación de la fase de despliegue, lo que ocasionó no disponer de un margen adicional de tiempo para un futuro posible contratiempo.

4 Desarrollo

En este apartado se van a detallar las herramientas y lenguajes utilizados para desarrollar el sistema y su arquitectura simplificada. Dado que el objetivo principal del proyecto es la implementación de un sistema de autenticación se han tenido que investigar y seleccionar unos lenguajes de programación para crear la aplicación.

4.1 Fase de análisis

Durante la fase de análisis se definieron los requisitos iniciales, los cuales se dividen en funcionales y no funcionales y están listados en las *tablas 1 y 2*.

TABLA 1
Requisitos Funcionales

La aplicación debe permitir el registro de los usuarios mediante su correo electrónico.
La aplicación debe permitir la autenticación de los usuarios mediante su correo electrónico, cuenta de Facebook o Twitter.
La aplicación cliente debe redireccionar a los usuarios al portal de autenticación único al querer identificarse.
La aplicación debe permitir que los usuarios restauren la contraseña a través del correo proporcionado.
La aplicación permitirá al administrador del sistema de autenticación el registro de los dominios y aplicaciones web cliente restringidas a su uso.
Si el sistema detecta un ataque de fuerza bruta debe bloquear temporalmente la cuenta de usuario.
El sistema debe ser Single Sign-On.

TABLA 2
Requisitos No Funcionales

El sistema deberá desarrollarse siguiendo los patrones de programación y consideraciones de seguridad descritos en el estándar del protocolo OAuth2.0 [4].
Las credenciales de la base de datos deben ser almacenadas utilizando un algoritmo de encriptación hash.
El sistema deberá estar protegido contra ataques Man in the Middle, Cross Site Scripting (XSS) y ataques de robo de credenciales.
Las comunicaciones entre los servidores de autenticación, las aplicaciones y los clientes utilizarán el protocolo de encriptación https.
El correo dirigido al usuario con el token de restauración de contraseña debe enviarse utilizando un algoritmo de encriptación.
El sistema debe contar con un manual (“readme”) estructurado para su instalación y despliegue.

4.2 Tecnologías utilizadas

El servidor ha sido desarrollado utilizando Node.js, un entorno de programación del lado del servidor que se ejecuta sobre el intérprete Javascript. Según un estudio de rendimiento y escalabilidad [7], Node.js permite una programación asíncrona, realizando diferentes operaciones en paralelo, implicando un coste menor a dedicar a la infraestructura de servidores. A demás se ha decidido utilizar el framework Express porque facilita la construcción de aplicaciones web, especialmente la creación de rutas y la gestión de plantillas. A demás incorpora middlewares que permiten una mejor interacción con otras aplicaciones.
La base de datos utilizada es MongoDB, una base de datos noSQL orientada a documentos. La elección de MongoDB y de Node.js fue conjunta ya que esta aporta eficacia y rapidez al trabajar con objetos similares a JSON, igual que Javascript. La utilidad de la base de datos de la aplicación es la de almacenar los datos de los usuarios y el requisito más importante es la seguridad de los datos, las librerías de Node.js que permiten la interacción con MongoDB son suficientes para cubrir dichas necesidades.

Dado que el sistema es desplegado en un servidor localizado en una Raspberry se ha utilizado Nginx, un servidor web multiplataforma, ya que ofrece las funcionalidades de proxy inverso, balanceo de carga y soporte de HTTP sobre SSL, indispensable para la implementación del protocolo OAuth2 [8].

Dos librerías Node.js claves para la implementación del servidor utilizadas son OAuth2orize y Passport, siendo la primera un middleware para la gestión de la autorización y la segunda de la autenticación.

a) Auth2orize

La librería OAuth2orize gestiona la transacción OAuth2 entre la aplicación cliente y el servidor. En la *figura 5* se muestran los diferentes endpoints que han sido implementados para completar la transacción OAuth2 entre la aplicación cliente y el servidor:
- El endpoint de autorización al que son redirigidas las peticiones de las aplicaciones cliente verifica la identidad de la aplicación y comprueba si el usuario está autenticado en el sistema y en caso de no estarlo lo redirige a autenticarse. Finalmente, solicita el permiso al usuario para que la aplicación cliente acceda a sus datos, en caso de tratarse de una aplicación sin una confianza directa.
- El endpoint de decisión procesa la petición realizada al usuario para la solicitud del permiso de acceso a los datos por parte de la aplicación cliente.
- El endpoint de token verifica la identidad de la aplicación cliente que proporciona un código de autorización para obtener un token de acceso y acceder al recurso.

a) Passport

Las librerías Passport son un conjunto de estrategias que dan soporte a la autenticación de los usuarios. En el servidor se ha implementado la autenticación del usuario mediante el correo y contraseña y mediante las redes sociales.

4.3 Modelo de base de datos

Al inicio del proyecto se intento la instalación de la base de datos MongoDB en el dispositivo Raspberry Pi, donde se realiza el despliegue. No obstante, debido a que el dispositivo no soporta las plataformas x86 o amd64 no fue posible instalar una versión que no estuviera obsoleta. Como alternativa se optó por utilizar MongoDB Hosting (Mlab).

Las colecciones de la base de datos en Mlab son:
- Client (aplicaciones cliente registradas).
- User (usuarios registrados).
- AuthoritzationCode (código de intercambio por AccessToken).
- AccesToken (código para solicitar un recurso).
- Resources (datos accedidos por la aplicación cliente).

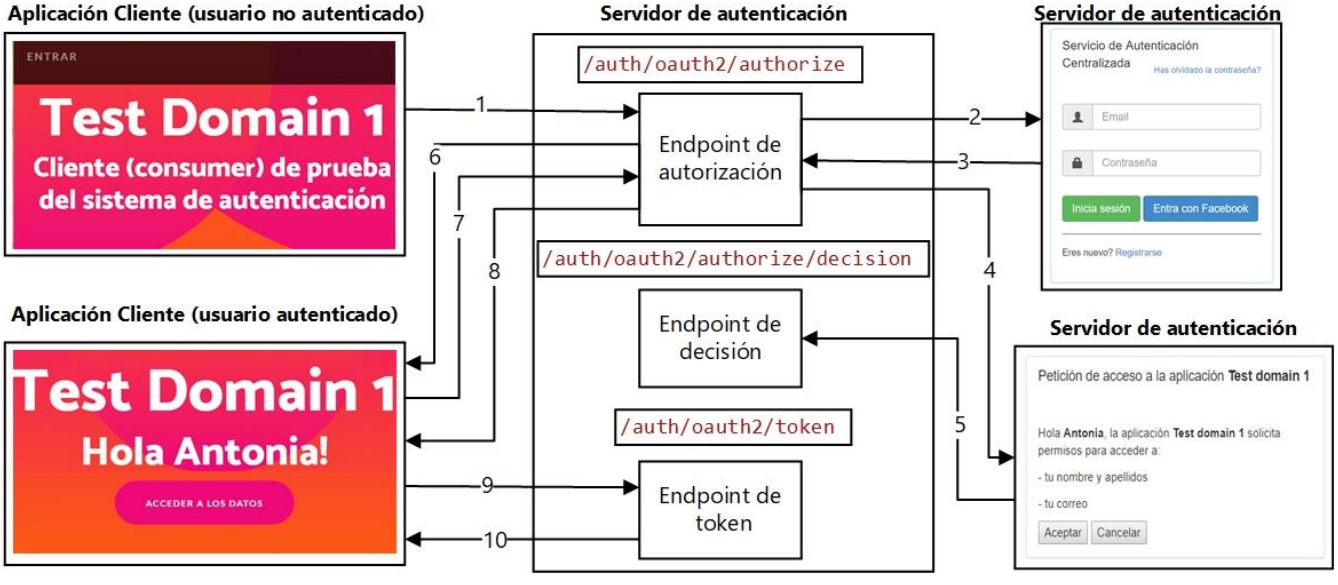

Fig. 5. Representación de los endpoints del servidor y la comunicación entre pantallas

5 RESULTADOS

En este apartado se da una breve explicación de cómo se llevan a cabo dos pruebas diferentes de la aplicación y los resultados obtenidos al ejecutarlas.

La primera prueba consiste en analizar el flujo de datos generado por el proveedor de autenticación y una aplicación cliente de prueba, a fin de compararlo con el flujo de los datos de la especificación OAuth2.0 [4] y determinar si el programa aplica los aspectos clave del protocolo.
La segunda prueba es un análisis del rendimiento de la aplicación en una situación de sobrecarga del sistema, cuando integra una base de datos MongoDB, la seleccionada para desarrollar el proyecto, y MySQL, una base relacional como alternativa.

La Raspberry Pi 3 Model B en la que se han realizado las pruebas de las aplicaciones cliente y el servidor de autenticación tiene las siguientes características:

- Procesador Cortex-A53 4 núcleos
- 1 GB LPDDR2 SDRAM 900 Mhz
- S.O. Raspbian GNU/Linux 9 (stretch)
- NodeJS v8.11.1
- Nginx 1.10.3

Se ha decidido realizar las pruebas en un hardware con las siguientes características ya que la cantidad de recursos está más limitada que en otros dispositivos y porque el sistema también es desplegado en la Raspberry Pi 3.

5.1 Prueba 1: Flujo de datos OAuth2

Esta prueba muestra un escenario de autenticación en el que un usuario de una aplicación cliente solicita acceso a un recurso protegido almacenado en la base de datos del servidor de autenticación desarrollado, el cual también actúa como entidad de servidor de recursos.
El objetivo de esta prueba es analizar el flujo de datos de la comunicación entre el proveedor de autenticación y la aplicación cliente, para detectar si el sistema cumple las especificaciones principales del protocolo Auth2.0 [4] y realizar un primer análisis de los riesgos y amenazas.

Para llevar a cabo la prueba se ha utilizado como aplicación cliente de test la herramienta Insomnia, que permite crear solicitudes HTTP y visualizar los detalles de las transacciones generadas.

La aplicación cliente, previamente registrada en el proveedor, desea acceder a los recursos protegidos a través de la ruta https://www.ssoauthentication.bid/api/resources, la cual devuelve "Unauthoritzen" cuando un usuario no está autenticado en una aplicación cliente. Para autenticar al usuario, la aplicación cliente envía una petición OAuth2.0 al servidor de autenticación. El cliente incluye en la petición la id de la aplicación, la contraseña de la aplicación y la URL de redirección donde la aplicación cliente desea recibir la respuesta. En la *figura 6* se muestra la petición creada en la herramienta Insomnia.

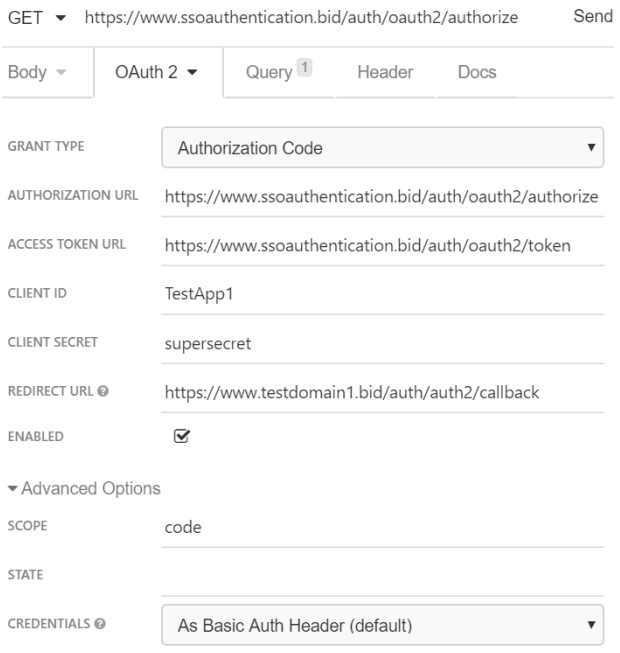

Fig. 6. Petición Get Auth2.0 enviada del cliente al servidor

En la *figura 7* se puede observar una parte parcial del código implementado que permite la autenticación OAuth2.0 de una aplicación Node.js cliente en el proveedor de autenticación.

```
passport.use(new OAuth2Strategy({
  authorizationURL: 'https://www.ssoauthentication.bid/' +
                    'auth/oauth2/authorize',
  tokenURL: 'https://www.ssoauthentication.bid/' +
                    'auth/oauth2/token',
  clientID: 'TestApp1',
  clientSecret: 'supersecret',
  callbackURL: 'https://www.testdomain1.bid/' +
                  'auth/auth2/callback',
  passReqToCallback: true
  },
  function (req, accessToken, refreshToken, profile, done) {
    User.findOrCreate({ id: profile.id }, function (err, user) {
      return done(err, user);
    });
  }
));
```

Fig. 7. Fragmento representativo del código de la aplicación cliente Node.js para la autenticación simultánea de los usuarios

El servidor de autenticación, al recibir la petición del cliente redirecciona al usuario al portal centralizado de autenticación, y el usuario previamente registrado introduce su correo electrónico y contraseña.

Fig. 8. Portal de autenticación al que es redireccionado el usuario

Una vez el usuario se identifica, el servidor de autenticación comprueba en el endpoint de autorización que tanto las credenciales del cliente como las del usuario son correctas y, ya validado, el servidor genera un código de autorización y un código de acceso con la identificación del usuario y de la aplicación cliente.


```
"code": "zKn4xsSNrJbOrCUJ",
"clientId": "TestApp1",
"redirectUri": "https://www.testdomain1.bid/auth/auth2/callback",
"userId": "7b3saeuHGorvdgeb",
"__v": 0
```


Fig. 9. Authoritzation Code generado en la base de datos Mongo.


```
"userId": "7b3saeuHGorvdgeb",
"clientId": "TestApp1",
"token":
"8mCZJ4DM9x5o8qsGJhzdfOKsm4Hft6AeENRu4Q33CEAL4QVOuDK7D31FwIgRXZecYovvUgoFyQIvuE
fWhUa29SaOg7C5nXgIvDGupSXp7Hfp5SngQwVvkgoG9lTtKt4L4ACYZuxvb8WVIWBgmvAyEbFar0KJG
ywkGnEZ2Y7Fxt84sx1kPvFBE7JTFUSNDLGXSZ92Ru9dWZRnATVz21uCYVogDPnszlbl1nZUuVe5EJvwq
DehtTrZGrtNtPchXi6e",
```


Fig. 10. Access Token generado en la base de datos Mongo.

El servidor devuelve a la url de redirección proporcionada por la aplicación cliente el token de acceso como respuesta de la petición de autenticación OAuth2.

```
> GET /?response_type=code HTTP/1.1
> Host: www.ssoauthentication.bid
> User-Agent: insomnia/5.16.6
> Authorization: Bearer
qEA1WjfMJi4JaxIdWv9DzjhFBSuHEwK4ZKbdBTrQCxTHyAMZRi
Nc2O6afDUC491g1sZXKWmppX6925wf6hTlNPYIWacq98vCiFi8
Pcvk61vQhsYu9wZVzjn64UacdCS75bmzyKs3CeArZl4KJL1HUV
Og5JUUTOclcoNF6BC55MbV9pdn0HPnO2hS1E1cQOjTPnzxb29O
uBL16z0caNktI3wjrUUrbvjNrhTr8BxsUVzDHXcp8pDMny3mVJ
Cv6yP0
> Accept: */*
```

Fig. 11. Recepción del token de acceso en la aplicación cliente

La aplicación cliente solicita el recurso incluyendo el token de acceso en la petición y el servidor de autenticación valida el token para devolver el recurso protegido. Como test de prueba, los recursos solicitados por el usuario son la lista de playas de la base de datos.

```
> GET /api/protectedresources HTTP/1.1
> Host: www.ssoauthentication.bid
> User-Agent: insomnia/5.16.2
> Cookie: connect.sid=s%3AqmgxfgcpwrSGDn66u2-
LE7iPrmxvbW7V.BAWplsFatjeEtxBc92TyEHQkboplrgRph5n2
pkdaNOM
> Authorization: Bearer
8mCZJ4DM9x5o8qsGJhzdfOKsm4Hft6AeENRu4Q33CEAL4QVOuD
K7D31FwIgRXZecYovvUgoFyQIvuEfWhUa29SaOg7C5nXgIvDGu
pSXp7Hfp5SngQwVvkgoG9lTtKt4L4ACYZuxvb8WVIWBgmvAyEb
Far0KJGywkGnEZ2Y7Fxt84sx1kPvFBE7JTFUSNDLGXSZ92Ru9d
WZRnATVz21uCYVogDPnszlbl1nZUuVe5EJvwqDehtTrZGrtNtP
chXi6e
> Accept: */*
```

Fig. 12. Petición del recurso protegido utilizando el token de acceso


```
[
  {
    "beach": {
      "nombre": "El Saler",
      "ciudad": "Valencia",
      "comunidad": "Valencia",
      "longitud": 300,
      "nudista": "si",
      "userid": "DBV5BtSP7TywrkgU"
    },
    "_id": "5b09b0c74ce4aa2a8348c20d",
    "__v": 0
  }
]
```


Fig. 13. Recurso protegido enviado por el proveedor de autenticación en formato JSON

5.2 Prueba 2: Análisis de rendimiento del servicio

En esta prueba de picos y de estrés se realiza la autenticación simultánea de varios centenares de usuarios diferentes en el servicio de autenticación para determinar el tiempo de respuesta y el comportamiento del rendimiento de la aplicación en situaciones de sobrecarga, uno de los factores que determinan la calidad del servicio.

La prueba se divide en dos partes, la primera es la autenticación de los usuarios en el servidor de autenticación integrando la base de datos MongoDB. En la segunda prueba se ha modificado la aplicación para integrar la base de datos MySQL y así poder observar el comportamiento de la aplicación en un entorno SQL y en uno no-SQL. En cada una de las bases de datos se ha importado un fichero csv con 1000 entradas de usuarios. Se ha escogido este número de muestras ya que se ha considerado que el sistema implementado sería utilizado por organizaciones con un menor número de usuarios simultáneos. A demás, como se observa en la *figura 14,* se ha creado una aplicación en C# con el objetivo de generar y enviar las peticiones de login de forma simultánea.

```
Uri address = new Uri("https://www
  .ssoauthentication.bid/login");
TextReader fReader = File.OpenText("users.csv");
var csv = new CsvReader(fReader);
csv.Configuration.HasHeaderRecord = false;
var records = csv.EnumerateRecords(new
  UsersResult());
List<Thread> threads = new List<Thread>();
int count = 0;
foreach (var r in records) {
  if (count >= 1000){
    break;
  }
  count++;
  Thread t = new Thread(() => RequestData(address
    ,ref lSessionTime, r.pass, r.username));
  threads.Add(t);
  t.Start();
}
foreach (var r in threads){
  r.Join();
}
```

Fig. 14. Fragmento del código en C# que crea los threads

La aplicación crea un thread por cada entrada de usuario del fichero csv y cada thread ejecuta el método "RequestData", que envía una petición al servidor de autenticación con los datos del usuario de la entrada. El entero count representa el número de peticiones simultáneas a enviar al servidor de autenticación.

En la *tabla 3* se muestra el tiempo de respuesta del sistema, desde que el usuario presiona el botón de login hasta que el sistema devuelve una respuesta.

TABLA 3
RESULTADOS DEL TIEMPO DE RESPUESTA DEL SERVICIO

Nº de peticiones simultáneas	Tiempo medio de respuesta por usuario (segundos)	
	MySQL	MongoDB
25	0,181 s	0,255 s
50	0,206 s	0,287 s
100	0,355 s	0,4319 s
200	0,545 s	0,566 s
400	1,297 s	1,287 s
800	3,497 s	3,439 s
900	3,296 s	4,353 s

En la comparativa de tiempos de respuesta la variación entre utilizar una base de datos MySQL o MongoDB es relativamente inapreciable. Según un estudio experimental de tiempos de respuestas y experiencia de usuario [9], un tiempo de respuesta de 1 segundo es el máximo aceptable para una aplicación, ya que a partir de ese valor el usuario es capaz de percibir el retraso. Por lo tanto, la interacción del usuario con el proveedor de autenticación deja de ser fluida a partir de 200 peticiones simultáneas.

También se ha evaluado como afecta la sobrecarga y el número de peticiones entrantes concurrentes en el sistema, para analizar la posible aparición de errores y falta de disponibilidad. Para ello se han calculado la diferencia entre el número total de peticiones de logueo enviadas y las respuestas validas recibidas, que representan los usuarios que se han logueado en el sistema con éxito. La aplicación almacena automáticamente dichos resultados en un fichero, los cuales son representados en la *tabla 4*.

TABLA 4
RESULTADOS DE LA DISPONIBILIDAD DEL SERVICIO

Nº peticiones simultáneas	Total de inicios de sesión validos				Nº de peticiones reenviadas	
	MySQL		MongoDB		MySQL	Mongo DB
25	18	72%	19	76%	7	6
50	38	76%	36	72%	12	14
100	76	76%	69	69%	24	31
200	144	72%	139	70%	56	43
400	273	68%	275	69%	127	121
800	508	64%	515	64%	292	285
900	498	55%	547	61%	402	353

Los resultados obtenidos muestran como el proveedor de autenticación no es capaz de atender 25 peticiones concurrentes, provocando que alrededor del 30% de las peticiones de inicio de sesión no se procesen. A medida que se incrementa este volumen de peticiones, las respuestas válidas del servidor disminuyen. Al ejecutar las pruebas también se observó que el número de peticiones reenviadas por cada usuario es de alrededor del 30%, ya que el tiempo de espera de la respuesta se incrementa.

6 CONCLUSIONES

El propósito del proyecto era implementar un sistema de autenticación en el lenguaje Node.js que permitiese a distintas aplicaciones cliente simplificar el proceso de la autenticación de los usuarios. Una vez finalizada y testeada la aplicación y mediante las pruebas realizadas que permiten observar una parte del comportamiento de la aplicación desarrollada, se puede concluir que el objetivo principal ha sido cumplido, ya que la aplicación implementa gran parte de las especificaciones del protocolo OAuth2.0. y satisface la mayoría de los requisitos propuestos en la fase inicial del proyecto.

No obstante, el desarrollo e implementación de un sistema de dichas características requiere un análisis más profundo de los riesgos y amenazas, ya que hasta las aplicaciones de las empresas más importantes presentan vulnerabilidades por un mal uso del protocolo. Unos investigadores de la universidad de Hong Kong publicaron un estudio [10] en el que examinaron más de 600 aplicaciones que implementaban OAuth 2.0 y encontraron que el 41% de las aplicaciones eran vulnerables a un exploit que desarrollaron.

Para desarrollar el sistema, se hizo un análisis de todos los ataques que podría sufrir y las medidas que se deberían aplicar para su prevención. Entre los ataques analizados, los más relevantes fueron el XSS (Cross-site scripting), en el que código Javascript es inyectado en la página web y el CSRF (Cross-site request forgery), un exploit malicioso en el que un atacante obtiene el token de acceso de un usuario.

Para los diferentes ataques se tomaron medidas correctoras, como la validación de los datos de los formularios, la utilización de un secreto por parte del cliente, el registro de las URIs de redirección de los clientes para evitar redirecciones a terceros o el uso de SSL para encriptar los tokens y credenciales. A pesar de que el sistema cumpliese las especificaciones podría igualmente ser atacado ya que no todas las medidas y operaciones están documentadas en las especificaciones, al no tratarse de un estándar completo.

Cabe destacar, como conclusión final, que crear un proveedor OAuth sin vulnerabilidades es una tarea difícil que consume mucho tiempo. A demás, uno de los problemas de las aplicaciones cliente que implementan el protocolo es su dependencia con el proveedor Oauth escogido, el cual, a pesar de solucionar el problema de la autenticación de los usuarios, los proveedores tienen el control sobre ellos.

Pese a tener desventajas, el protocolo aporta muchos beneficios, siendo los más destacados la disminución de la complejidad de la autenticación y una mayor experiencia del usuario.

6.1 Líneas futuras

Dado que no existe una única forma de diseñar e implementar el proveedor de autenticación OAuth2, en este proyecto se decidió implementar el servidor de autenticación y el servidor de recursos en un mismo servidor, que uniera las dos entidades. No obstante, una mejora sería separar dichas entidades para crear un proveedor multiservicio. Otra mejora, relacionada con el punto anterior, sería definir el alcance (scope) de cada aplicación cliente y, por lo tanto, limitar su acceso a la información restringida de cada usuario. Esto permitiría que los usuarios tuvieran conciencia y control de los datos a los que tienen acceso las aplicaciones terceras. Otra mejora del proveedor sería la integración del protocolo con otros protocolos como OpenID Connect, una capa de identidad por encima del protocolo OAuth2.0, que permite Single Sign-On, o SAML, entre otros.

AGRADECIMIENTOS

Primeramente, quisiera dar las gracias a Sergi Robles, mi tutor del proyecto por darme valiosos consejos e ideas.
A mí familia y pareja, por su apoyo, paciencia y ánimos a lo largo de estos años en los que he estudiado en la universidad. Finalmente quiero agradecer a la comunidad de desarrolladores de Node.js por su aportación de documentación y respuestas.

BIBLIOGRAFÍA

[1] R. Gafni, D. Nissim, "To social login or not login? - Exploring factors affecting the decision", *Issues in Informing Science and Information Technology*, vol. 11, pp. 57-72, 2014.

[2] T. Armerding, "The 17 biggest data breaches of the 21st century", Enero 2018. [Online]. Disponible: https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html

[3] L. Hal, B. Campbel B. Campbell, „Security Assertion Markup Language (SAML) V.2.0 Technical Overview", Marzo 2008. [Online] Disponible: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

[4] D. Hardt, "RFC6749: The OAuth2.0. Authorization Framework", Octubre 2012. [Online]. Disponible: https://tools.ietf.org/html/rfc6749

[5] J. Wagner, "Acquisition of Stormpath Exemplifies Dangers of Using Startup APIs", Mar 2017. [Online]. Disponible: https://www.programmableweb.com/news/acquisition-stormpath-exemplifies-dangers-using-startup-apis/analysis/2017/03/10 [Consultado el 17 de Mayo de 2018].

[6] A. Cockburn, "Incremental versus iterative development", Julio 2007. [Online]. Disponible: http://alistair.cockburn.us/Incremental+versus+iterative+development

[7] Lei, K., Ma, Y., & Tan, Z. "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js", 2014. IEEE 17th International Conference on Computational Science and Engineering, 661-668. [Consultado: 15 de abril de 2018]

[8] P. Prakash, R. Biju, M. Kamath, "Performance analysis of process driven and event driven web servers", Enero 2015. Intelligent Systems and Control (ISCO), IEEE 9th International Conference on.

[9] Echeverria, D. "Tiempo de Respuestas y Experiencia de Usuario, Estudio Experimental". 2016. Revista Latinoamericana de Ingeniería de Software, 4(5): 231-234, ISSN 2314-2642

[10] R. Yang, W. Cheong, T. Liu, "Signing into One Billion Mobile App Accounts Effortlessly with OAuth2.0", 2016. TheChinese University of Hong Kong.