Universidad de las Ciencias Informáticas

Facultad 3

Implementación del protocolo de autorización Open Authorization (Oauth) para Bosón

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

Mebys Ferrer Hernández

Tutores:

Ing. Abraham Calás Torres

La Habana, junio de 2016


"El futuro de nuestra patria tiene que ser necesariamente un futuro de hombres de ciencia, tiene que ser un futuro de hombres de pensamiento, porque precisamente es lo que más estamos sembrando; o que más estamos sembrando son oportunidades a la inteligencia (...)"

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autora:

Mebys Ferrer Hernández

Tutor:

Ing. Abraham Calás Torres

_____

Firma

_____

Firma

Agradecimientos

Yo Mebys Ferrer Hernández le agradezco:

En primer lugar, a las personas que más quiero en el mundo, a mi mamá y a mi abuela por todo el apoyo que me han brindado y por el esfuerzo y sacrificio que han tenido que hacer para que llegara este momento.

A mi futuro esposo Rey por ser mi novio, mi amigo y mi confidente, por estar a mi lado cada vez que lo necesito, por los momentos inolvidables que hemos compartido, por su paciencia, su comprensión y su cariño.

A mi tía Iliana por ayudarme siempre que la necesito.

A mi papá, a mi prima, a Jorge.

A mi suegra María por apoyarme y acogerme como si fuera una más de su familia.

A mis amigas Neysi, Dailét y Karla por haber compartido tan buenos momentos.

A mi mejor amiga Abueli por haber confiado en mí y estar siempre presente a pesar de la distancia.

A todos mis amigos del 3503 en especial a Alexei y Yordan por haberme aguantado todos estos años.

A la revolución por permitirme la oportunidad de estudiar en esta universidad.

A todos los que se encuentran aquí hoy, a los que creyeron en mí y a los que de una forma u otra colaboraron con la realización de este trabajo, MUCHAS GRACIAS.

Dedicatoria

Dedico el presente trabajo de diploma a mi mamá y a mi abuela porque sin su apoyo incondicional hoy no estaría aquí.

.

Resumen

En la Universidad de Ciencias Informáticas se lleva a cabo una intensa tarea de producción de software. La seguridad de los mismos ha cautivado la atención de los centros de desarrollo convirtiéndose en un tema activo de investigación. Dentro de la seguridad, la autenticación y autorización por parte de los usuarios es uno de los elementos claves que ha mantenido un desarrollo constante, surgiendo novedosas formas para autenticar y autorizar que ofrecen altos niveles de seguridad.

El presente trabajo propone la implementación de un protocolo de autenticación y autorización mayormente enfocado en el control de acceso a recursos protegidos por parte de los usuarios, que se adapte a las condiciones y características del marco de trabajo Bosón. Es aquí donde OAuth jugará un papel de gran importancia. OAuth es una tecnología emergente, que cubre los principales requisitos de seguridad aportando confidencialidad, integridad y autenticación a la hora de mantener una comunicación con una API y acceder a los recursos que esta ofrece.

Para su desarrollo se hace un estudio de la evolución que ha tenido la especificación del estándar OAuth hasta llegar a su versión más actual. Se define la arquitectura del sistema, así como un conjunto de artefactos necesarios para lograr la implementación. Para evitar inconsistencias durante el desarrollo y cumplir con los objetivos de la investigación se validan los requisitos y el diseño. Por último, se realizan pruebas al sistema que verifican su correcto funcionamiento y se discuten los resultados obtenidos de la aplicación en Bosón.

Palabras Claves:

autenticación, autorización, estándares, recurso, protocolo, seguridad, OAuth

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

La seguridad informática es un factor fundamental en cualquier empresa. Continuamente, muchas personas intentan acceder a los ordenadores ajenos para obtener datos no autorizados. Ese acceso no autorizado a una red informática o a los equipos que engloba, puede ocasionar graves problemas principalmente para las empresas, ocasionando graves pérdidas**,** tanto económicas como de credibilidad.

Es esencial que expertos en la materia controlen la seguridad informática de una empresa. Hay que mantener un estado de alerta permanente, ya que la seguridad es un proceso continuo que las empresas no pueden permitirse dejar a un lado, sino que deben enfocar su atención en corregir las vulnerabilidades que poseen para hacer frente a posibles ataques informáticos. La seguridad informática permite asegurar la integridad y privacidad de la información de un sistema informático y sus usuarios.

Dentro de los procesos claves que se gestionan en los sistemas de seguridad se encuentran los de autenticación, autorización y auditoría (Suhendra, 2011). El proceso de autenticación es el encargado de asegurar que el usuario es quien dice ser. Después que el usuario es autenticado, la autorización es el proceso de concesión de privilegios basado en la identidad, por lo que la fortaleza que presenten la autenticación y la autorización influirá en que tan seguro pueda ser un sistema.

La Universidad de las Ciencias Informáticas (UCI) cuenta con una red de centros de desarrollo entre los que se encuentra el Centro de Informatización de Entidades (CEIGE). Uno de los productos que se desarrollan en CEIGE es el marco de trabajo Bosón. Este servirá como arquitectura de referencia para construir sistemas informáticos orientados a la web, debido a que especifica las buenas prácticas para el desarrollo, así como la definición de patrones y tecnologías a utilizar. Está formado por componentes desarrollados sobre Symfony 2, y estos responden a la mayoría de los requisitos tecnológicos de un amplio espectro de aplicaciones. Bosón se presenta como la solución que permitirá establecer un formato de trabajo común entre las diversas soluciones de desarrollo, al aportarles componentes previamente construidos y listos para ser utilizados. (Calas, 2015)

Bosón redefine la seguridad de Symfony2, para gestionar la autenticación implementa un proceso de autenticación que está regido por el estándar internacional SAML. Bosón actualmente con el uso de este protocolo no cuenta con un mecanismo para efectuar un control de acceso a sus recursos protegidos mediante la gestión de políticas de autorización, afectando la seguridad de la información que viaja a través de estos servicios web. La técnica utilizada para acceder a un recurso protegido desde un

sistema externo es con la autenticación por usuario y contraseña, teniendo el usuario que compartir las credenciales de su cuenta en Bosón en una aplicación de terceros donde si esta es hackeada se pueden ver comprometidas las credenciales del usuario. Por tal motivo, el proceso de autenticación y autorización de Bosón puede estar sometido a diversos ataques con el objetivo de acceder a los servicios o recursos que brinda.

Por tanto, se plantea el siguiente **problema a resolver:** ¿Cómo fortalecer los procesos de autenticación y autorización en el marco de trabajo Bosón?

Teniendo como **objeto de estudio** los procesos de autenticación y autorización y el **campo de acción** la autenticación y autorización mediante el protocolo OAuth.

Con el desarrollo de la presente investigación se persigue lograr como **objetivo general** desarrollar un componente basado en el protocolo de autorización OAuth para fortalecer los procesos de autenticación y autorización del marco de trabajo Bosón.

Para darle cumplimiento al objetivo general se plantearon los siguientes **objetivos específicos**:

- Confeccionar el marco teórico conceptual de la investigación a partir de una búsqueda y revisión bibliográfica de Open Authorization (Oauth).
- Realizar el análisis y diseño del componente de seguridad utilizando Open Authorization (Oauth) en el marco de trabajo Bosón.
- Implementar el protocolo de autorización Open Authorization (Oauth) en el marco de trabajo Bosón
- Validar la solución propuesta mediante pruebas de caja blanca y el cuasiexperimento como técnica de validación.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de investigación**:

- Revisión de bibliografía actualizada para la elaboración del marco teórico-conceptual.
- Estudio de los estándares de autenticación y autorización que se utilizan en la actualidad.
- Definición de los requerimientos funcionales y no funcionales para el módulo de Seguridad.
- Planificación y diseño de las funcionalidades a implementar.
- Implementación de las funcionalidades diseñadas.
- Realización de las pruebas para comprobar que la gestión de la seguridad cumpla con todos los requerimientos especificados.

Se tiene como **idea a defender** de la presente investigación: Si se implementa el protocolo de autorización Oauth se logrará fortalecer los procesos de autenticación y autorización en el marco de trabajo Bosón.

Para dar cumplimiento a las tareas de investigación se emplearon los **Métodos Científicos** que se clasifican en**:**

Teóricos:

- **Análisis Histórico-Lógico**: permitió comprender de forma más clara la esencia del objeto de estudio y su concepción histórica. Para ello se realizó un análisis de la evolución del protocolo OAuth.
- **Analítico-Sintético:** para el estudio y análisis de la bibliografía, permitiendo una correcta formulación del problema y de los objetivos, la elaboración de las conclusiones y recomendaciones logrando resultados satisfactorios en la investigación.

Empíricos:

- **Entrevistas:** para obtener información relacionada con la propuesta de solución, así como toda la información referente al funcionamiento del marco de trabajo Bosón.

Este documento se encuentra organizado en tres capítulos, a continuación, se brinda una breve descripción de los mismos:

Capítulo 1: Fundamentación Teórica: Se describen los aspectos y conceptos asociados al dominio del problema a resolver, siendo estos esenciales para entender el entorno del mismo. Se presenta el estado del arte de las técnicas implicadas en el objeto de estudio y el conjunto de tecnologías involucradas en el desarrollo de la propuesta y se realiza la justificación de las seleccionadas para la solución del problema.

Capítulo 2: Propuesta de solución: Se hace una descripción de la propuesta de solución basada en el protocolo de seguridad OAuth para garantizar mejoras en la seguridad del marco de trabajo Bosón; así como la explicación de los principales productos de trabajo de la metodología empleada. Se exponen además los patrones empleados durante el desarrollo, así como algunos rasgos esenciales en el funcionamiento del marco de trabajo.

Capítulo 3: Validación de la solución: Se incluye en este capítulo un estudio de los conceptos fundamentales de las pruebas de software con sus clasificaciones. Se muestran los resultados de la validación del modelo, utilizando la técnica de cuasiexperimento.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En este capítulo se aborda acerca de la seguridad en aplicaciones web, haciendo énfasis dentro de esta en las características principales del proceso de autenticación y autorización. Posteriormente se hace un estudio del estado del arte, donde se analiza la evolución que ha tenido el protocolo Oauth hasta llegar a su versión más actual Oauth 2.0. Se brinda una descripción de varios métodos de autorización delegada que existían antes de OAuth y que sirvieron de inspiración a la hora de crear el estándar. Además, se da una explicación más detallada de lo que hace OAuth 2.0, mostrando los diferentes flujos propuestos que soportará la implementación de una forma sencilla e ilustrativa. Se describen las tecnologías, herramientas y técnicas que serán empleadas para la implementación del protocolo.

1.2 Seguridad en aplicaciones web

Los requerimientos primordiales de los sistemas informáticos que desempeñan tareas importantes son los mecanismos de seguridad adecuados a la información que se intenta proteger. El término seguridad informática define el conjunto de métodos y herramientas destinados a proteger los bienes o activos informáticos de una institución (Pfleeger, y otros, 2006). Un sistema de seguridad es una aplicación destinada a implementar mecanismos de seguridad, que definen responsabilidades y reglas a seguir, para evitar amenazas o minimizar sus efectos en caso que estas se materialicen (Avilés, 2015).

La información es el bien más preciado con que cuentan las organizaciones. Ante los riesgos que enfrentan los Sistemas Informáticos se requiere de la implementación de políticas de control de acceso eficientes; que incluyan la autenticación, autorización y auditoría (Suhendra, 2011) como procesos fundamentales.

Estos procesos tienen que lograr el cumplimiento de los objetivos básicos de la seguridad, lograr la confidencialidad, integridad y disponibilidad de la información (Pfleeger, 2006).

Cuando se desarrollan aplicaciones web, por lo general el desarrollo se enfoca más a la funcionalidad que a la seguridad. A la hora de desarrollar Bosón, los programadores suelen obviar cuestiones importantes para la seguridad como la protección de los datos, debido a la urgencia de ofrecer nuevas funcionalidades y servicios. Esto representa una vulnerabilidad para el sistema e incrementa la probabilidad de éxito de incidentes de

seguridad, razón por la cual, se crean nuevos programas, estrategias, mecanismos, políticas, prácticas y protocolos de seguridad, dirigidos a proteger este tipo de software.

1.3 Autenticación y autorización

La palabra autenticar es sinónimo de legalizar, legitimar, certificar y refrendar. Según el Diccionario de la Lengua Española SM en su versión multimedia, autenticar es *"autorizar o legalizar. Acreditar o dar fe con autoridad legal de la verdad de un hecho o de un documento"* (Real Academia Española, 2001). Para la gestión de la seguridad informática, particularmente en las aplicaciones web, la autenticación es el proceso con el que se verifica la identidad de un usuario o sistema externo que trate de acceder a un recurso. Este proceso suele requerir la presentación de algún tipo de credenciales. La mayoría de los sistemas, entre ellos Bosón, lo implementan solicitando un usuario y una contraseña.

Por otra parte, autorizar, según el Diccionario de la Lengua Española SM en su versión multimedia es *"dar autoridad, facultad o derecho para hacer algo. Permitir la realización de algo"* (Real Academia Española, 2001). También puede interpretarse como facultar, delegar, calificar, consentir o permitir. Este concepto, aplicado a la gestión de la seguridad de las aplicaciones web, define el proceso que, habiendo sido comprobada la validez de las credenciales de los usuarios y sistemas externos, se encarga de verificar los privilegios de acceso de estos sobre la información.

1.3.1. Sistema centralizado de autenticación o Single Sign-On

Un agente de Inicio de Sesión Único (SSO por sus siglas en inglés Single Sign-On), es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación. Este tipo de procedimiento, simplifica y centraliza el control de acceso de todas las aplicaciones de la institución. Además, aumenta la velocidad de los procesos de autenticación y autorización y simplifica el manejo de claves, aspectos que aumentan los niveles de seguridad y logran un mejor rendimiento del sistema (Roebuck, 2012).

La implementación de un SSO soluciona el problema de tener que autenticarse en múltiples aplicaciones, resuelve los inconvenientes de tener datos y recursos personales repartidos en diferentes sistemas y contribuye con la gestión de la seguridad. No obstante, cada institución puede desarrollar su propio SSO para integrar todas sus aplicaciones. Esto se convierte en un problema cuando se quieren integrar tecnologías de diferentes corporaciones, desarrolladas bajo procedimientos SSO diferentes.

Por tal motivo, se han creado estándares de autenticación y autorización que implementan un mismo SSO en todas las aplicaciones web publicadas en Internet.

1.3.2. Estándares de autenticación y autorización

Un estándar, según el Diccionario de informática e Internet de Microsoft, es una *"guía técnica definida por una organización no comercial o gubernamental de reconocido prestigio que se utiliza para uniformar una determinada área de desarrollo de hardware o software"* (Microsoft, 2003). Por tanto, un estándar de autenticación y autorización se define como un modelo o patrón que agrupa características comunes para la identificación y verificación de privilegios de usuarios en una aplicación. Esta investigación estará enmarcada en el estándar de autorización OAuth, donde se realiza un estudio de la evolución que ha tenido hasta llegar a su versión más actual.

1.4 Open Authorization(OAuth)

Antes de que se redactara la especificación del estándar OAuth, muchos sitios web de la industria eran conscientes de la necesidad de fortalecer sus procesos de autorización, por lo que diseñaron e implementaron sus propios protocolos (Boyd, 2012). En la medida que dichos protocolos representan el estudio preliminar que condujo a OAuth 1.0, y finalmente a OAuth 2.0.

A continuación se procede a reseñar algunos de los mismos con objeto de subrayar los elementos finales constituyentes de la versión más reciente del estándar OAuth.

1.4.1 AuthSub

AuthSub es un proceso de autorización desarrollado por Google para aplicaciones web que tuvieran la necesidad de acceder a servicios protegidos por un usuario de Google. Este proceso permite a la aplicación obtener acceso al servicio sin llegar a manejar las credenciales del usuario (Google, 2012).

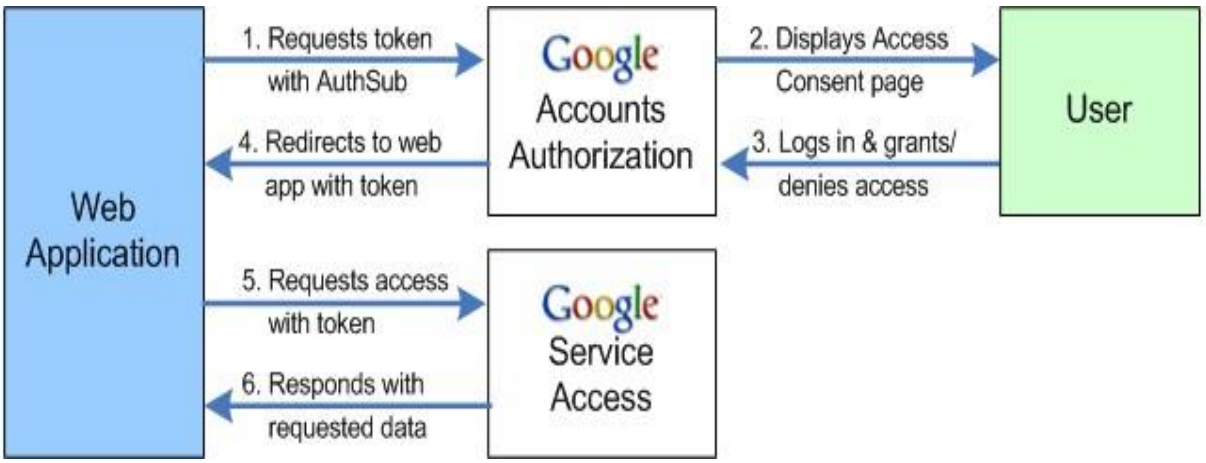

Figura 1. Proceso de Autorización AuthSub
Fuente: (developers google)

En la Figura 2 se puede observar el proceso de autorización seguido, que posee los siguientes pasos:

1. Cuando la aplicación necesita acceder a un servicio de un usuario de Google, hace una llamada utilizando AuthSub al Servicio de Autorización de Google.
2. El Servicio de Autorización responde mostrando una página de Google donde se le notifica de la petición de acceso a sus servicios. Para ello el usuario ha de estar autenticado.
3. El usuario acepta o rechaza la petición y se le redirige a la aplicación web inicial o a Google, según su decisión.
4. En la redirección a la aplicación web inicial se incluye un token de autorización de un solo uso, que se podrá cambiar por un token de más duración.
5. La aplicación web adjunta el token recibido al enviar la petición al servicio de Google, para poder actuar en nombre del usuario.
6. Si el servicio de Google reconoce el token, devuelve la información solicitada.

1.4.2 BBAuth

Browser-Based Authentication (BBAuth) es un proceso que permite a las aplicaciones pedir permiso a los usuarios de Yahoo! para poder acceder a sus datos También implementa un proceso de Single Sign-On para poder autenticar a los usuarios en aplicaciones externas. Aunque Yahoo! propone el uso de este método, también da soporte a la especificación estándar de OAuth 1.0 (Siriwardena, 2014).

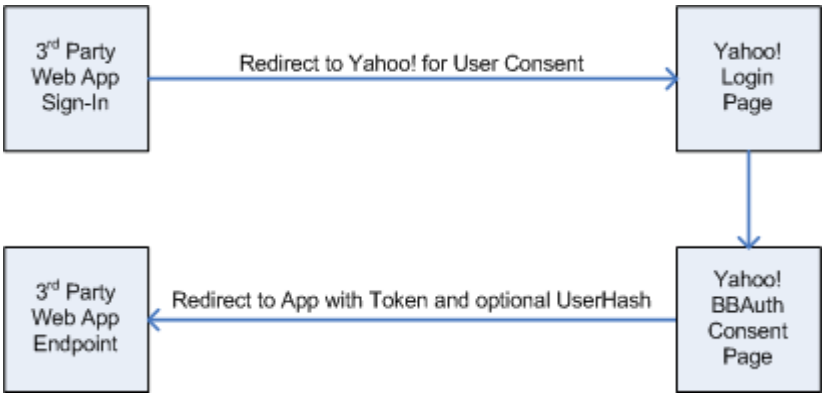

Figura 2 Proceso BBAuth
Fuente: (Siriwardena, 2014)

En la Figura 3 se puede observar el flujo que implementa BBAuth:

1. Tras haber registrado la aplicación de terceros en Yahoo!, se redirige al usuario a una URL de Yahoo! específica.
2. Si el usuario no se había autenticado en Yahoo!, lo hace y pasa a una página donde puede elegir si conceder los permisos a la primera aplicación.
3. Se redirige a la aplicación inicial incluyendo los datos necesarios para que ésta pueda acceder a la información que solicitaba desde el principio.

1.4.3 OAuth 1.0

OAuth 1.0 es un protocolo abierto que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, web y móvil. Como los anteriores métodos de autorización, permite a una aplicación de terceros acceder al recurso protegido de un usuario, con su permiso (HAMMER-LAHAV, 2010).

Debido a problemas de seguridad, se trabajó en una versión 1.0a no estándar que soluciona estos problemas, que actualmente utiliza por ejemplo Twitter, aunque su uso ha decaído debido a la aparición de OAuth 2.0. Los pasos de este proceso, representados en la Figura 4, se explican a continuación:

1. La aplicación solicita una petición de token.
2. El proveedor de servicios se la concede.
3. La aplicación dirige al usuario al proveedor de servicios, donde se autentica.
4. El usuario permite a la aplicación acceder al servicio protegido y es dirigido a la aplicación cliente.
5. La aplicación cliente, con la autorización del usuario, pide un token de acceso.
6. El proveedor de servicios le proporciona el token de acceso a la aplicación.
7. Ahora la aplicación puede acceder a los servicios protegidos.

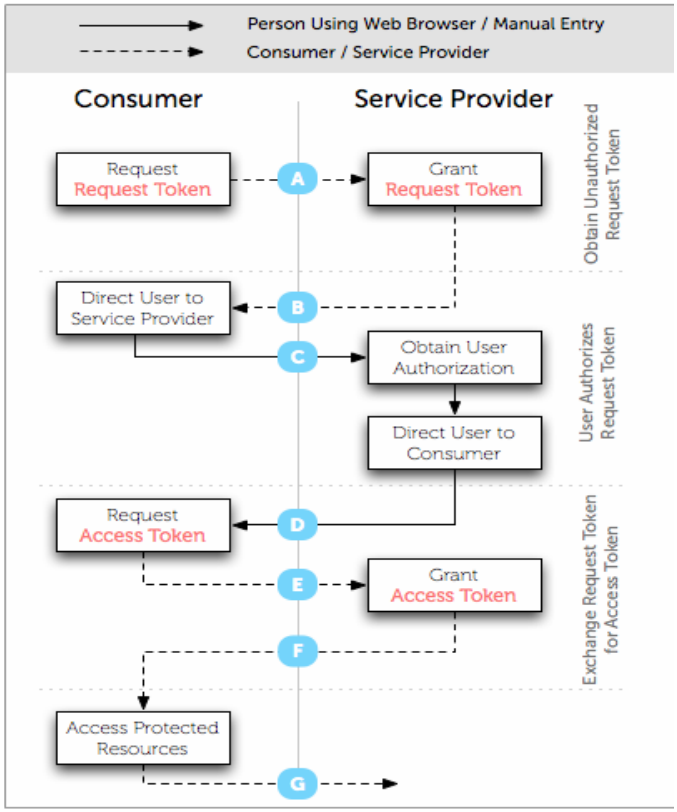

Figura 3 Proceso de autorización OAuth v1.0a
Fuente: (HAMMER-LAHAV, 2010)

1.4.4 Otros

Cuando se creó OAuth, en Internet convivían varios mecanismos para proporcionar una capa de seguridad a las APIs. Algunas de estas soluciones siguen utilizándose (las mencionadas anteriormente) aunque siempre proponen como alternativa el uso de OAuth.

Otras soluciones como OpenAuth, FlickrAuth y FacebookAuth se han retirado y se han sustituido por OAuth, pero todas ellas han contribuido a construir lo que es OAuth actualmente, ya que tienen muchos puntos en común y que acabara apareciendo un método estándar era inminente (Oauth, 2013).

OAuth 2.0 aparece con el fin de reemplazar a OAuth 1.0 enfocándose en la simplificación de los flujos de autorización para aplicaciones (Oauth, 2013). Se abordará de forma más detallada en el resto del documento las especificaciones del estándar.

1.5 Oauth 2.0

Antes de entrar en la descripción del trabajo realizado es necesario explicar en qué consiste OAuth 2.0.

Como se ha expuesto, su objetivo es añadir una capa de seguridad a los servicios que ofrece una API, a la vez que protege los datos de los usuarios, pudiendo un usuario conceder un acceso limitado a aplicaciones de terceros. OAuth propone proteger estos servicios y datos (a partir de ahora recursos) pidiendo un código (token de acceso) del que se puede deducir la identidad de quién accede y en nombre de quién accede.

1.5.1 Roles

OAuth define cuatro roles (Hardt, y otros, 2012):

- **Propietario del recurso:** entidad capacitada para conceder acceso a un recurso protegido.
- **Servidor de Recurso:** el servidor que aloja los recursos protegidos, siendo capaz de aceptar y responder a las solicitudes de recursos que le llegan de un cliente.
- **Cliente:** una aplicación capaz de hacer peticiones a recursos protegidos, en nombre del propietario del recurso, y con su autorización.
- **Servidor de Autorización:** es el servidor encargado de emitir tokens de acceso a clientes, los cuales han tenido que obtener previamente autorización del propietario del recurso.

1.5.2 Flujo

A continuación, se muestra el flujo general para la obtención de un recurso mediante OAuth 2.0 (Hardt, y otros, 2012) :

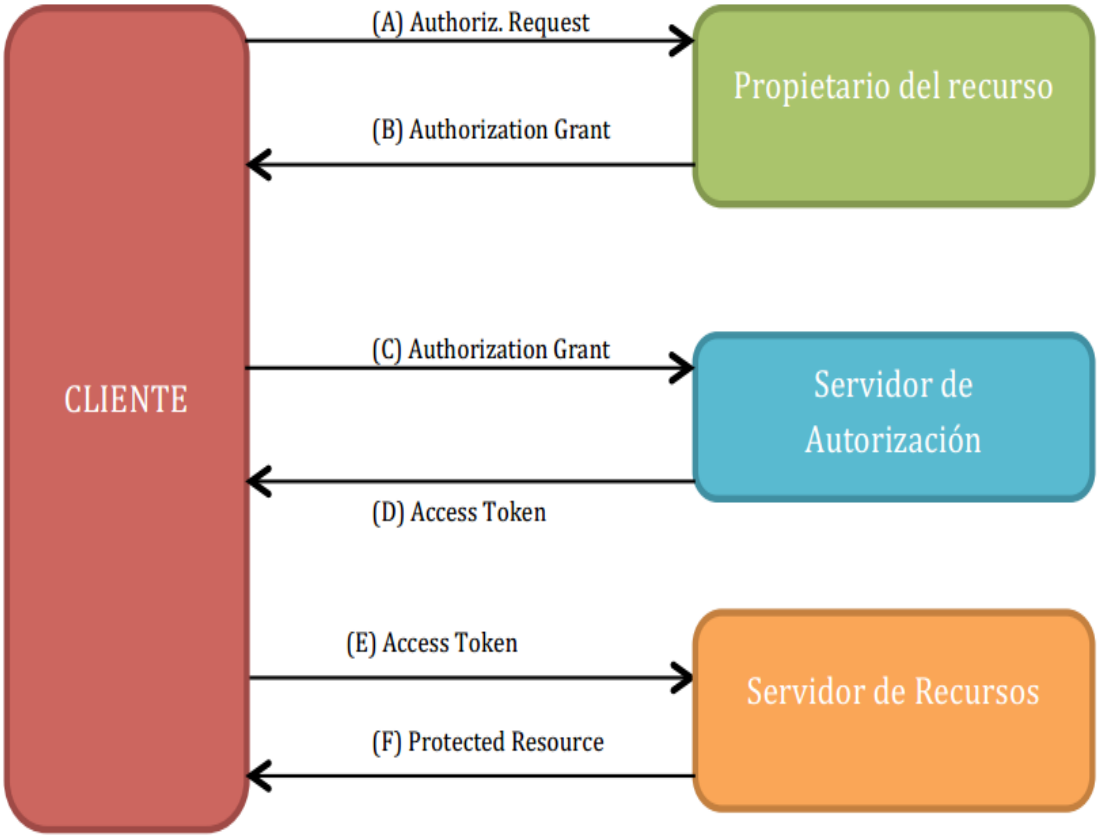

Figura 4. Funcionamiento del estándar Oauth
Fuente: (Hardt, y otros, 2012)

(A) El cliente solicita la autorización del propietario del recurso. La petición de autorización puede ser hecha directamente al propietario del recurso (como se muestra), o de manera indirecta, la cual es preferible, usando como intermediario el servidor de autorización

(B) El cliente recibe una concesión de autorización (authorization grant), la cual representa la autorización del propietario del recurso.

(C) El Cliente solicita un token de acceso mediante la autenticación en el Servidor de Autorización presentando el authorization grant.

(D) El Servidor de Autorización autentica al cliente y valida el authorization grant, y si es válido emite un token de acceso.

(E) El Cliente realiza una petición de un recurso protegido al Servidor de Recursos presentando el token de acceso.

(F) El Servidor de Recursos valida el token de acceso recibido, y si es válido, sirve la solicitud.

1.5.3 Concesión de autorización (authorization grant).

Un authorization grant no es más que una credencial que representa la autorización dada por el propietario del recurso para que se pueda acceder a sus recursos, la cual va a ser usada por el cliente para obtener un token de acceso. OAuth 2.0 define, principalmente, 4 formas en las que la aplicación cliente puede obtener autorización por parte del propietario del recurso. Estas son (Hardt, y otros, 2012):

- **Authorization Code:** se obtiene a través de un servidor de autorización, que hace de intermediario entre el cliente y el propietario del recurso.
- **Implicit**: es una versión simplificada del anterior, optimizada para entornos que usen lenguajes de script, capaces de ejecutar programas en una página web al ser visualizados en un navegador de internet, como pueda ser JavaScript.
- **Resource Owner Password Credentials**: se usan las credenciales del propietario del recurso (por ejemplo, nombre de usuario y password) para obtener el authorization grant, siempre y cuando exista una relación de confianza entre el cliente y el propietario del recurso.
- **Client Credentials:** las credenciales del cliente pueden ser usadas como authorization grant cuando el alcance de la autorización está limitado a los recursos protegidos que están bajo el control del cliente, o estos han sido previamente establecidos con un Servidor de Autorización.

Para el desarrollo de la solución se utilizará el código de autorización (Authorization Code) ya que está pensado para clientes confidenciales, es decir, que son capaces de ocultar sus credenciales cuando se autentican en el servidor de autorización.

1.5.4 Registro de clientes

Antes de iniciar el protocolo, el cliente tiene que estar registrado en el Servidor de Autorización. Los medios por los cuales el cliente se registra en el Servidor de Autorización están fuera del alcance de esta especificación. El registro de clientes en Servidores de Autorización no requiere una interacción directa entre el Cliente y el Servidor de Autorización. Si el registro es soportado por el Servidor de Autorización, este puede recurrir a otros medios para establecer la relación de confianza y la obtención de las propiedades requeridas del cliente (URL1 de redirección, tipo de cliente, authorization grant que se va a usar)

- Cuando se registra un cliente, el propietario o administrador de ese cliente debe: Especificar el tipo de cliente

[1] URL (Localizador Uniforme de Recursos): es la dirección específica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que estos puedan ser localizados o identificados.

- Ofrecer una URL para la redirección del cliente
- Incluir otro tipo de información que sea requerida por el Servidor de Autorización, por ejemplo: nombre de la aplicación y descripción

1.5.3.1 Tipos de clientes

OAuth define dos tipos de cliente (Hardt, y otros, 2012), basados en su habilidad para autenticarse de manera segura con el Servidor de Autorización.

- **Confidencial:** son los clientes capaces de mantener la confidencialidad de sus credenciales, por ejemplo, un cliente implementado en un servidor seguro con acceso restringido a las credenciales del cliente. También puede ser capaz de garantizar la seguridad de las credenciales por otros medios.
- **Público:** son clientes incapaces de mantener la confidencialidad de sus credenciales, por ejemplo, un cliente que se ejecuta en un dispositivo del propietario del recurso, como puede ser una aplicación nativa instalada o una aplicación web basada en navegador.

La designación del tipo de cliente se basa en la definición de autenticación segura que se haya implementado en el Servidor de Autorización y en los niveles de exposición de las credenciales del cliente. OAuth ha sido diseñado en torno a los siguientes perfiles de cliente (Hardt, y otros, 2012):

- **Aplicación Web:** una aplicación web es un cliente confidencial que es ejecutado en un servidor web. Los propietarios de recursos acceden al cliente a través de una interfaz de usuario HTML, representada en un user-agent (agente de usuario) en el dispositivo del propietario del recurso. Las credenciales del cliente, así como cualquier token de acceso emitido al cliente son guardados en el servidor web y no son accesibles por el propietario del recurso.
- **Aplicación basada en user-agent**: una aplicación de usuario basada en user-agent es un cliente público en el que el código del cliente se descarga de un servidor web y se ejecuta dentro de un agente de usuario (user-agent) en el dispositivo del propietario del recurso. Los datos y credenciales del protocolo son fácilmente accesibles (y usualmente visibles) al propietario del recurso. Dado que estas aplicaciones residen en el agente de usuario del propietario del recurso, pueden hacer un uso transparente de las capacidades del agente de usuario al solicitar la autorización.
- **Aplicaciones Nativas:** Una aplicación nativa es un cliente público instalado y ejecutado en el dispositivo del propietario del recurso. Los datos y credenciales del protocolo son accesibles por el propietario del recurso. Se asume que cualquier credencial de autenticación del cliente incluido en la aplicación puede

ser extraída. Por otro lado, las credenciales emitidas dinámicamente, tales como el token de acceso, sí tienen un nivel aceptable de protección, garantizando como mínimo que esas credenciales están protegidas de servidores hostiles que puedan interactuar con la aplicación, o de aplicaciones dentro del mismo dispositivo.

Identificador del cliente

El Servidor de Autorización otorga al cliente, durante el proceso de registro, un identificador de cliente, que no es secreto, está expuesto al propietario del recurso, y no debe ser usado en solitario para la autenticación del cliente (Hardt, y otros, 2012).

Cuando el cliente interactúe con el servidor de autorización directamente, este tendrá que poder autenticarse con las credenciales que se le hayan asignado.

Autenticación del cliente

Si el cliente es de tipo confidencial, el Servidor de Autorización y el Cliente deben establecer un método de autenticación adecuado a los requerimientos de seguridad del Servidor de Autorización. Este puede aceptar cualquier tipo de autenticación de clientes siempre que concuerde con los requerimientos de seguridad. Los clientes confidenciales suelen establecer una serie de credenciales de cliente que se usan para la autenticación con el Servidor de Autorización. El Servidor de Autorización no debe de hacer suposiciones sobre el tipo de cliente o aceptar el tipo de información recibida sin haber establecido previamente una relación de confianza con el cliente o su desarrollador. El Servidor de Autorización puede establecer métodos de autenticación con clientes de tipo público. Sin embargo, no debería confiar en autenticaciones de clientes públicos con el propósito de identificar al cliente. El cliente no puede usar más de un método de autenticación en cada petición.

1.5.5 Clientes no registrados

La especificación de OAuth 2.0 no excluye el uso de clientes no registrados. Sin embargo, el uso de este tipo de clientes queda fuera del alcance de esta especificación, ya que requiere de análisis de seguridad adicionales y de la revisión de su impacto interoperacional (Hardt, y otros, 2012).

1.5.6 Accediendo a recursos protegidos

El tipo de token de acceso proporciona al cliente la información requerida para utilizarlo de manera satisfactoria a la hora de realizar la petición de un recurso al Servidor de Recursos. El Servidor de Recursos debe validar el token recibido y asegurarse de que no está expirado y de que el scope cubre al recurso solicitado. El método por el cual el Servidor de Recurso valida un token de acceso está fuera del alcance de la

especificación de OAuth 2.0, pero normalmente conlleva una interacción entre el Servidor de Autorización y el Servidor de Recursos. La forma en la que el cliente utiliza el token de acceso para autenticarse con el Servidor de Recursos depende del tipo de token emitido por el Servidor de Autorización. Normalmente involucra el uso de la cabecera HTTP Authorization, usando un esquema de autenticación definido para cada tipo de token (Hardt, y otros, 2012).

Una vez se obtenga el token de acceso se podrá utilizar para acceder a un recurso proporcionándolo en la petición del mismo. Este token de acceso poseerá un conjunto de permisos que representan los datos o tareas que puede realizar, lo que en OAuth se denomina scope. El cliente podrá, bien especificar una serie de permisos al solicitar el token o bien no especificarlo, con lo que se supondrá que se solicitan los permisos por defecto. Si el cliente intenta acceder a un recurso con un token que no tenga los permisos requeridos por ese recurso, se le denegará el acceso.

1.6.5.1 Tipos de tokens de acceso

El tipo de token de acceso proporciona al cliente la información necesaria para utilizarlo de manera satisfactoria a la hora de realizar una petición al Servidor de Recursos de un recurso protegido. El cliente no debe de usar tokens de acceso si no comprende o confía en el tipo del token proporcionado (Hardt, y otros, 2012).

1.6.5.1.1Tokens Portadores (Bearer Tokens)

Un token de portador (bearer token) es un token de seguridad con la propiedad de que cualquier entidad (cliente) en posesión de un token (un portador, bearer) puede usarlo de una manera única, es decir, ningún otro bearer podrá usar ese mismo token de la misma manera. El uso de bearer tokens debe de implicar la utilización de mecanismos de transporte seguro y de cifrado de datos. (Hardt, y otros, 2012)

Existen tres métodos por los cuales un cliente debe de ser capaz de enviar un bearer token en la petición al Servidor de Recursos de un recurso compartido, no pudiendo usar más de uno en cada petición. Estos tres métodos son:

1. **Authorization Request Header Field:** Cuando se envía el token en el campo “Authorization” de la cabecera HTTP de la petición del recurso, el cliente está haciendo uso del esquema de autenticación Bearer para transmitir el token de acceso. Los Servidores de Recursos están obligados a soportar este método de envío de tokens, siendo el método más recomendado a usar y el que se usará en la solución.
2. **Form-Encoded-Body Parameter:** También es posible enviar el token de acceso en el cuerpo de la petición HTTP. Este método no se debe usar, excepto

en el contexto de aplicaciones donde el navegador de los participantes no tenga acceso al campo Authorization de la cabecera HTTP de la petición.

3. **URI Query Parameter:** El último método es enviar el token de acceso directamente en la URI de la petición HTTP. Debido a las debilidades de seguridad asociadas con este método, tales como la modificación del token, repetición de tokens o la redirección de la URI destino del token, ya que existe una alta probabilidad de que la URL que contiene el token de acceso sea accesible por atacantes externos. Por ello, este método sólo debe de usarse en el caso de que sea imposible de enviar el token de acceso mediante alguno de los otros dos métodos.

1.6 Metodología de desarrollo de software

Se seleccionó la metodología aprobada por la universidad para la actividad productiva de la misma de tal forma que se adapte a su ciclo de vida definido, la cual consiste en una variación de la metodología Proceso Unificado Ágil (AUP por sus siglas en inglés). Quedando estructurada en tres fases las cuales son detalladas a continuación:

Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración Construcción Transición	EJecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Figura 5. Ciclo de vida de la Metodología AUP-versión UCI
Fuente: (CEIGE, 2015)

1.7 Herramientas y tecnologías a utilizar

A continuación, se presentan las herramientas y tecnologías utilizadas en la implementación del protocolo.

1.7.1 Lenguaje de modelado

Lenguaje Unificado de Modelado (UML): Este lenguaje prescribe un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan; posibilitando así visualizar, especificar y documentar los artefactos o toda información que se obtiene o modifica durante un proceso de desarrollo de software, además de poder utilizarse para modelar distintos tipos de sistemas de software, hardware y organizaciones del mundo real (Larman, 2003).

Principales características:

- Permite modelar sistemas haciendo uso de técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.

1.7.2 Herramienta Case

Visual Paradigm para UML 8.0 es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, haciéndolas mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (Visual Paradigm, 2014).

1.7.3 Lenguajes de programación

PHP 5.4.4-14: PHP es el acrónimo recursivo del inglés *Hypertext Pre-processor* (Pre-procesador de hipertextos), es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución del PHP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores. (PHP, 2015)

1.7.4 Marcos de trabajo

Symfony 2.7: Es un marco de trabajo que ayuda a simplificar el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporciona estructura al código fuente, forzando

al desarrollador a crear código legible y fácil de mantener. Por último, facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Symfony es un completo marco de trabajo diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (symfony.es, 2007) (Eguiluz, 2013)**.**

Entre las características destacadas que ofrece a los desarrolladores de productos de software se encuentran las siguientes:

- Fácil de instalar y configurar en la mayoría de plataformas.
- Independiente del sistema gestor de bases de datos.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.

1.7.5 Doctrine 2.0

ORM Doctrine 2.3

Doctrine es un marco de trabajo que proporciona persistencia transparente de objetos PHP, el cual se sitúa en la parte superior de una capa de abstracción de base de datos (DBAL, por sus siglas en inglés, Database Abstraction Layer). Una de sus principales características es que cuenta con la opción de escribir las consultas de base de datos en un lenguaje de consulta estructurado (SQL, por sus siglas en inglés Structured Query Language) orientado a objetos, llamado lenguaje de consulta Doctrine (DQL2, por sus siglas en inglés, Doctrine Query Language) (Doctrine, 2014). Es una librería completa y configurable, además, viene integrada por defecto con Symfony2 y presenta entre sus características principales las siguientes:

- Generación automática del modelo.
- Posibilidades de trabajar con YAML.
- Relaciones entre entidades.

1.7.6 Entorno de Desarrollo Integrado (IDE)

PhpStorm8.0: PhpStorm llena un vacío existente en el mercado, como un Entorno de Desarrollo Integrado (IDE) inteligente para desarrollar aplicaciones en Pre-procesador

de hipertextos (PHP), proporcionando herramientas esenciales como refactorización, análisis de código y comprobación de errores (Packt Publishing, 2013).
Las principales novedades en PhpStorm 8.0 incluyen:

- Soporte para PHP 5.3, incluyendo espacios de nombre.
- Depuración con cero configuraciones con todos los navegadores
- Compatibilidad con el marco de trabajo Symfony
- Editores de Lenguaje de Consulta Estructurado (SQL) con resultados editables
- Soporte para Lenguaje Marcado de Hipertextos (HTML5).

1.7.8 Servidor de Aplicaciones

Apache 2.2.22-13: Apache es el servidor web por excelencia, su facilidad de configuración, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. Se ejecuta en gran cantidad de sistemas operativos, lo que lo hace prácticamente universal. Es una tecnología gratuita, código abierto y altamente configurable de diseño modular por lo que resulta muy sencillo ampliar sus capacidades. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor y es posible configurarlo para que ejecute un determinado script cuando esto suceda (Project, 2015).

1.8 Patrones

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y proveen facilidades para crear un software reutilizable de buena calidad. Cada patrón describe un problema que ocurre repetidamente en el entorno, y describe el núcleo de la solución a ese problema, de tal forma que esta pueda ser usada un millón de veces, sin hacer el mismo trabajo dos veces (Larman, 2003).
A continuación, se muestran los patrones utilizados en la solución.

1.8.1 Patrón de arquitectura

Patrón arquitectónico Modelo-Vista-Controlador (MVC)

Dentro de los patrones arquitectónicos existentes, se encuentra el patrón de arquitectura de las aplicaciones de software MVC, el cual resulta uno de los más empleados para el desarrollo de aplicaciones web. Este patrón o modelo, separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. El patrón de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es una página de Lenguaje de Marcado de Hipertexto

(HTML) y el código que provee datos dinámicos a la página, el modelo incluye el Sistema de Gestión de Base de Datos (SGBD) y la lógica de negocio, y el controlador es el responsable de recibir y atender los eventos de entrada desde la vista (Fernández Romero, y otros, 2012) (Buschmann, y otros, 1996).

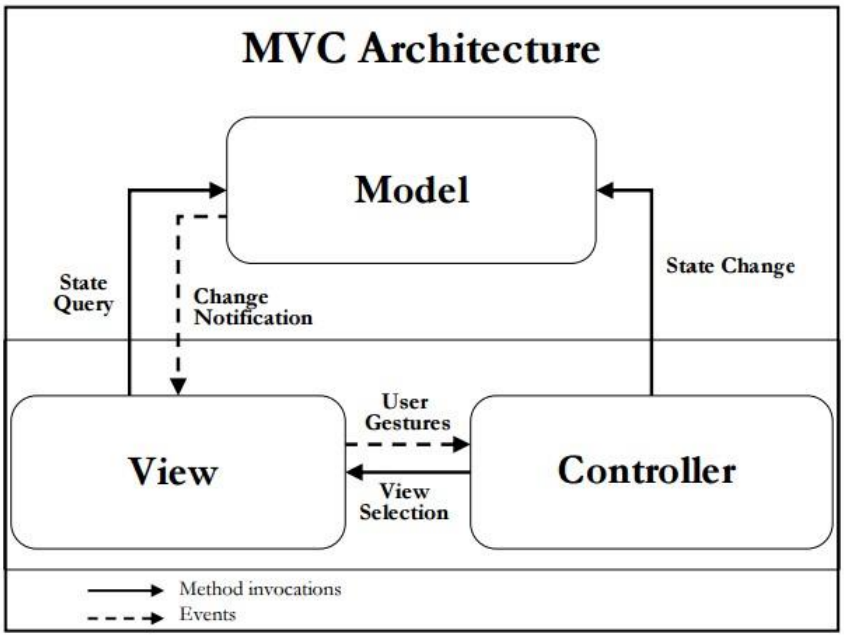

Figura 6. Patrón Modelo- Vista- Controlador
Fuente: (Liu, 2007)

1.8.2 Patrones de diseño

Para el desarrollo del módulo se aplican los patrones de diseño *GRASP*, los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos. Entre los más conocidos y utilizados para la solución están (Pressman, 2009):

- **Experto**: cada clase dentro del módulo tiene la responsabilidad de utilizar únicamente la información que ella misma posee para realizar la labor para la que fue concebida. Tal es el caso de las clases del modelo que son las encargadas de toda la lógica del acceso a los datos.
- **Creador**: identifica quien debe ser el responsable de la creación de nuevos objetos o clases, donde la nueva clase deberá ser creada por la clase que tiene toda la información necesaria para realizar la acción, que usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de clase y contiene o agrega la clase.
- **Alta cohesión**: cada clase que pertenece al modelo tiene como responsabilidad fundamental realizar las labores que solo le competen a ella y que no son desempeñadas por otros elementos del diseño. Dentro de esas labores se

encuentra crear las consultas de acceso a los datos. Todas las clases están agrupadas por las funcionalidades que realizan.

- **Bajo acoplamiento**: las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista, lo que proporciona que la dependencia en este caso sea baja.
- **Controlador**: el módulo tiene clases controladoras que se encargan de atender todas las peticiones y pasar los datos de la misma a las clases del modelo para su procesamiento. Al mismo tiempo es la clase que se encarga de enviar las respuestas a la vista.

La aplicación de estos patrones se ve reflejada con la asignación de las responsabilidades por clases para realizar solo las funcionalidades correspondientes a la información que las mismas contienen evitando una sobrecarga. Además, se da la responsabilidad de crear instancias de otras clases solamente a aquellas que contengan a las mismas y se modelan clases controladoras, encargadas de controlar el flujo de las operaciones del sistema. El diseño de la aplicación permite la interacción entre las mismas propiciando un bajo acoplamiento y en consecuencia la reutilización.

1.9 Validación de los requisitos

El resultado del trabajo realizado es una consecuencia de la ingeniería de requisitos y es evaluada su calidad en la fase de validación. La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto (Pressman, 2009). Para llevar a cabo este proceso, se aplicaron las siguientes técnicas de validación de requisitos:

- **Revisión técnica formal**: la revisión de las especificaciones de los requerimientos del proceso de generación de código fuente fue realizada con la analista principal del proyecto. Con la utilización de esta técnica se valida que no existan errores en el contenido o malas interpretaciones, información incompleta, inconsistencias y que los requisitos no sean contradictorios, imposibles o inalcanzables (Pressman, 2009), dando como resultado que fueran aprobados los que estaban descritos de forma correcta, clara y consistente.

- **Generación de casos de prueba (test de requisitos)**: fueron definidos y diseñados casos de pruebas para cada requerimiento especificado, con el objetivo de verificar el cumplimiento de los mismos. La utilización de esta técnica ofreció los siguientes resultados: fueron identificados los posibles escenarios de los requisitos, así como

los juegos de datos de los campos determinados en estos, se validaron y aprobaron los que estaban bien enunciados, descritos y consistentes (Pressman, 2009).

1.10 Métricas para validar el diseño

Lorenz y Kidd dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase, examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización (Pressman, 2009) (Lorenz, y otros, 1994).

Por la relevancia para la investigación se considera oportuno aplicar de Lorenz y Kidd la métrica TOC, pues permite visualizar si se distribuyen correctamente las asignaciones de responsabilidades entre las clases, verificándose así la cohesión y armonía entre las mismas, y de Chidamber y Kemerer las métricas RC para evaluar el grado de acoplamiento entre las clases.

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- **Responsabilidad:** consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** consiste en el grado de reutilización de presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de “reutilización”.
- **Complejidad del mantenimiento:** consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costos y la planificación del proyecto.
- **Cantidad de pruebas:** consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

1.10.1 Métrica Tamaño Operacional de Clase (TOC)

TOC: está dada por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- **Responsabilidad:** un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación:** un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** un aumento del TOC implica una disminución del grado de reutilización de la clase.

A continuación, se muestra en la siguiente tabla la aplicación de la métrica TOC, donde CP se refiere a la cantidad de procedimientos:

Atributo de calidad	Categoría	Criterio
Responsabilidad	Baja	CP < =Promedio
	Media	Promedio<=CP<=2* Promedio
	Alta	CP> 2* Promedio
Complejidad de implementación	Baja	CP< =Promedio
	Media	Promedio<=CP<=2* Promedio
	Alta	CP> 2* Promedio
Reutilización	Baja	CP> 2*Promedio
	Media	Promedio<=CP<=2* Promedio
	Alta	CP<= Promedio

Tabla 1. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC
Fuente: (Elaboración propia)

1.10.2 Métrica Relación entre Clases (RC)

RC: está dada por el número de relaciones de uso de una clase con otra. Permite evaluar los atributos de calidad: acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas de unidad necesarias de cada clase, teniendo en cuenta las relaciones existentes entre ellas:

- Acoplamiento: un aumento de las RC implica un aumento del Acoplamiento de la clase.
- Complejidad de mantenimiento: un aumento de las RC implica un aumento de la complejidad del mantenimiento de la clase.
- Reutilización: un aumento de las RC implica una disminución en el grado de reutilización de la clase.
- Cantidad de pruebas: un aumento de las RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

A continuación, se muestra en la siguiente tabla la aplicación de la métrica RC, donde CRU se refiere a la cantidad de relaciones de uso.

Atributo de calidad	Categoría	Criterio
Acoplamiento	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento	Baja	CRU<= Promedio
	Media	Promedio<=CRU<=2* Promedio
	Alta	CRU> 2* Promedio
Reutilización	Baja	CRU>2* Promedio
	Media	Promedio<=CRU<= 2* Promedio
	Alta	CRU<= Promedio
Cantidad de pruebas	Baja	CRU<= Promedio
	Media	Promedio<=CRU<=2* Promedio
	Alta	CRU> 2* Promedio

Tabla 2. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC
Fuente: (Elaboración propia)

1.11 Pruebas

Pruebas de caja blanca

El método de prueba de caja blanca, denominado a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de software puede obtener casos de prueba que: (1) garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales; y (4) ejerciten las estructuras internas de datos para asegurar su validez (Pressman, 2009).

Para ejecutar este tipo de pruebas según (Pressman, 2009) se utilizará la técnica del camino básico: esta técnica permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

1.12 Conclusiones Parciales

Mediante el estudio realizado en este capítulo se arribó las siguientes conclusiones:

- Después del estudio se define hacia donde está orientada la problemática de la investigación, definiéndose que Bosón no cuenta con un mecanismo de control

de acceso; determinándose la necesidad de la implementación de un protocolo para poder llevar a cabo una autorización delegada.

- El estudio del protocolo OAuth y la especificación del mismo permitió obtener un mejor entendimiento y los elementos necesarios para su posterior implementación.
- La metodología, las herramientas y lenguajes de programación seleccionados son los adecuados para la implementación del protocolo ya que además de ser los utilizados en el marco de trabajo Bosón, responden a la soberanía tecnológica estableciendo las bases para el desarrollo de la propuesta de solución.

Capítulo 2. Desarrollo de la solución

2.1 Introducción

En el presente capítulo se realiza una propuesta de solución para darle cumplimiento al objetivo general planteado y se describen las características que debe poseer el sistema a desarrollar; además se enumeran los requisitos funcionales y no funcionales con los que debe cumplir el mismo, así como su diseño de clases.

2.2 Descripción de la solución

Para la implementación del protocolo, se tiene la siguiente propuesta de solución:

- Cuando el usuario quiere acceder a Bosón a través de una aplicación cliente, ésta redirecciona al usuario a la portada de Bosón.
- Si el usuario no está autenticado, se autentica en ese momento con su cuenta de Bosón.
- Bosón luego le pregunta al usuario si quiere autorizar a la aplicación cliente, y le da a escoger al usuario los permisos que le dará a la misma.
- Cuando se presiona el botón “Autorizar”, se crea un “Access Token” (Vale para acceso) y un “Access Token Secret” (Vale para acceso secreto). Son como contraseñas, pero sólo permiten que la aplicación cliente acceda a la cuenta del usuario y haga las acciones que le permitió.

2.3 Requisitos del software

Los requisitos para implementar el protocolo OAuth2 son los descritos en la especificación del mismo, los cuáles se definen a continuación:

2.3.1 Requisitos funcionales

Los requerimientos funcionales son capacidades o condiciones que un sistema determinado debe cumplir. Seguidamente se enumeran los que se han identificado para el desarrollo de este componente.

Gestionar Cliente		
Identificador	**Nombre**	**Descripción**
RF_1	Adicionar Cliente	Permite adicionar un nuevo cliente al sistema.

RF_2	Eliminar cliente	Permite eliminar un cliente existente en el sistema a partir de su identificador.
RF_3	Listar clientes	Permite mostrar en un listado todos los clientes existentes en el sistema.
Identificador	**Nombre**	**Descripción**
RF_4	Redireccionar usuario	El cliente podrá dirigir al usuario al punto de autorización, que se encontrará en una URL.
RF_5	Autenticar	El cliente tendrá que autenticarse Con los parámetros client_id y client_secret.
RF_6	Autorizar	El servidor será capaz de mostrar al usuario una pantalla donde decidir si autorizar o no al cliente.
RF_7	Generar código de autorización	Se redirecciona al usuario a la aplicación cliente con un código de autorización.
RF_8	Validar código de autorización	Verificar si el código de autorización es válido.
RF_9	Generar token de acceso	El servidor de autorización será capaz de crear y almacenar un token de acceso si se le proporcionan los parámetros adecuados.

Tabla 3 Requisitos funcionales
Fuente: (Elaboración propia)

2.3.2 Historias de usuario

De cada uno de estos requisitos funcionales identificados fueron diseñadas sus historias de usuario.

A continuación, se muestra en la tabla, el ejemplo de la historia de usuario (HU) del requisito “Adicionar cliente”, el resto de ellas serán mostradas en los anexos.

Número: 45	**Nombre del requisito:** Adicionar cliente
Programador: Mebys Ferrer Hernández	**Iteración Asignada:**
Prioridad: Alta	**Tiempo Estimado:** 8 horas
Riesgo en Desarrollo:	**Tiempo Real:** 8 horas
Descripción: Este requisito permite adicionar un cliente al sistema. El sistema debe mostrar la opción de adicionar cliente, donde debe llenar los campos.	
Observaciones: N/A	
Prototipo de interfaz: N/A	

Tabla 4. HU del requisito “Adicionar cliente”
Fuente: (Elaboración propia)

2.3.3 Requisitos no funcionales

A continuación, se representan los requisitos no funcionales del sistema:

Usabilidad (USB)

El componente podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Rendimiento (REN)

Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 3 segundos.

Seguridad (SEG)

Autenticación (Contraseña de acceso). Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

Portabilidad (POR)

El componente debe ser multiplataforma, haciendo énfasis en Linux y Windows.

Soporte (SOP)

La aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

2.4 Arquitectura de software

Para el desarrollo de la solución se mantendrá la arquitectura en capas, basada en el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual según (Fernández Romero, y otros, 2012) es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo.
Para el sistema se adoptó una arquitectura multiniveles, específicamente de 2 niveles o capas, dichas capas son:

- **Capa de Lógica negocio:** La capa de lógica negocio está formada por interfaces y clases que definen cada uno de los componentes de negocio.
- **Capa de datos:** La capa de acceso a datos es la encargada de encapsular todos y cada uno de los componentes de acceso a datos, y es la encargada de acceder a los mismos.

Dado que lo que se va a implementar es un proveedor de autorizaciones solo se va a trabajar en la parte del servidor sin necesidad de interactuar con la capa de presentación, la misma será implementada por la aplicación cliente que vaya a utilizar el servidor de autorización OAuth implementado en Bosón.

Ventajas:

- Desarrollos paralelos (en cada capa).
- Aplicaciones más robustas debido al encapsulamiento.
- Mantenimiento y soporte más sencillo: es más sencillo cambiar un componente que modificar una aplicación monolítica.
- Mayor flexibilidad: se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad.

2.4.1 Patrón de arquitectura

En la arquitectura del sistema se hace uso del patrón arquitectónico MVC. A este patrón arquitectónico de software se le añaden las clases necesarias para la implementación del protocolo.

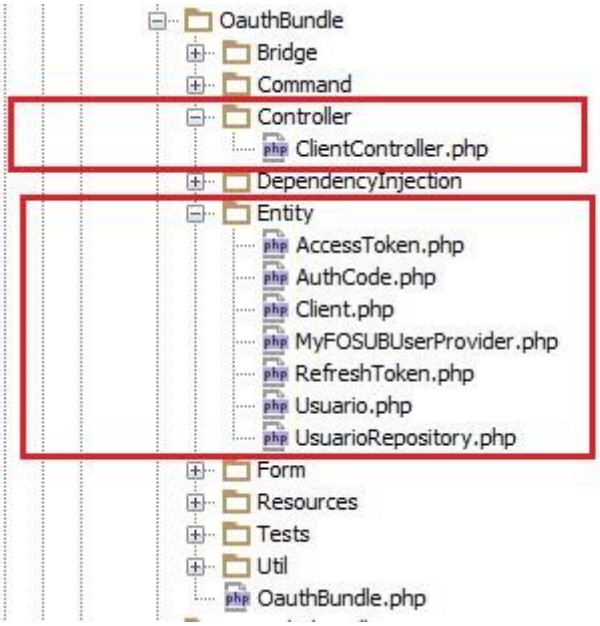

Figura 7. Ubicación de las clases controladoras y entidades
Fuente: (Elaboración propia)

- Controller: En esta carpeta se encuentra la clase Controladora que es la que contiene el código necesario para responder a las funcionalidades que se solicitan en la aplicación.
- Entity: Posee los ficheros con las Entidades del Modelo que son las encargadas de trabajar con los datos.

2.5 Modelo del diseño

A continuación, se muestra el diagrama de clases del diseño en el que se representan las clases con que cuenta el sistema, así como sus atributos y funcionalidades y la relación existente entre ellas.

2.5.1 Diagrama de clases con estereotipos web

A continuación, se presenta el diagrama de clases del diseño y la descripción de las clases que intervienen en el mismo.

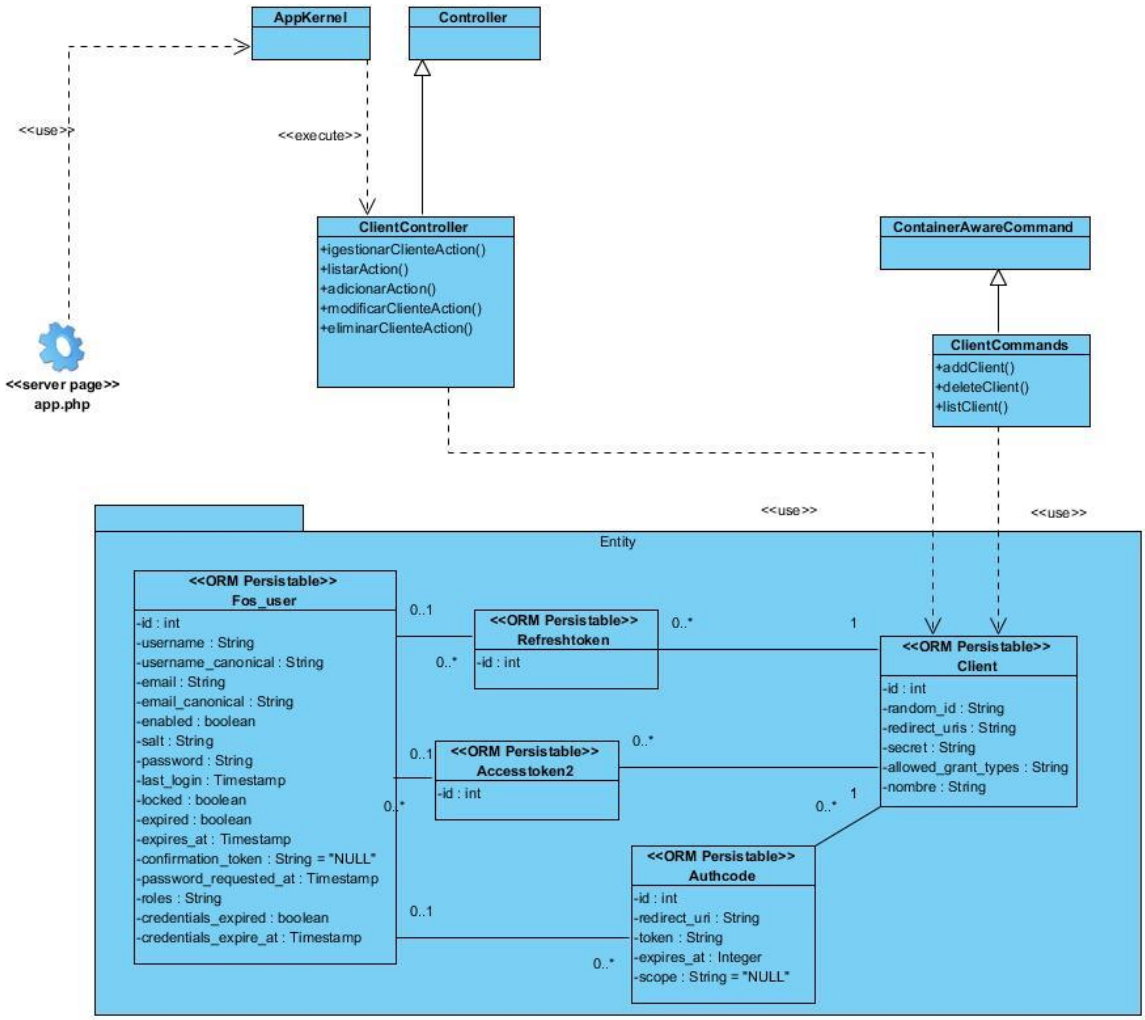

Figura 8. Diagrama de clases del diseño
Fuente: (Elaboración propia)

Tokens: OAuth utiliza tokens en lugar de credenciales de usuario para acceder a los recursos. Un token generalmente es una cadena aleatoria alfanumérica que es única, difícil de adivinar, y que tiene que usarse en par con el Consumer Secret para evitar el abuso del token. OAuth define dos tipos de tokens: Request (petición) y Access (acceso).

ClientController: Se definen las funcionalidades para manejar la gestión de dichas clases: eliminar, adicionar, modificar, listar. La clase controladora es la encargada de gestionar el acceso a la lógica de negocio y controlar todo el proceso.

Fos_user: Clase para la gestión de usuario. Se utiliza la implementación de FOSUserBundle, un Bundle de tercero para la gestión de usuarios.

Client: Clase para la gestión de clientes. Un cliente es la aplicación que solicita acceso a un servidor de recursos. Es la aplicación que quiere acceder al recurso en nombre del usuario.

ClientCommands: Permite crear clientes mediante la terminal introduciendo un comando con los parámetros solicitados.

Authcode: Clase que provee el código de autorización cuando el cliente quiere solicitar el acceso a los recursos protegidos en nombre de un usuario, es decir, una tercera parte.
Accesstoken: Clase que permite identificar quién está intentando acceder al cliente y en nombre de quién. El token es enviado por el cliente como un parámetro al servidor. Tiene una vida útil limitada, que se define por el servidor de autorización.
Refreshtoken: Este token se emite con el token de acceso, pero a diferencia de éste, no se envía en cada solicitud desde el cliente al servidor de recursos. Simplemente sirve para ser enviado al servidor de autorización para renovar el token de acceso cuando ha expirado.

2.6 Patrones de diseño

A continuación, se muestran los patrones utilizados en la solución.

2.6.2 Patrones de diseño GRASP

A continuación, se muestra la aplicación de los patrones de diseño en la propuesta de solución.

- **Experto**: Se evidencia en las clases modelos, ejemplo: la clase *Client*, la cual contiene toda la información referente a la manipulación de los datos de los clientes.
- **Creador**: En Symfony2 en la clase Controller se definen y ejecutan un conjunto de acciones en las que se crean los objetos de las clases que representan las entidades, evidenciando de este modo que la clase *ClientController* se encarga de crear las instancias de la clase *Client*, para usar las funcionalidades de ésta.
- **Alta cohesión**: Se encuentra evidenciado en la implementación de las clases que forman parte de la capa del Modelo, las cuales están formadas por diferentes funcionalidades que se encuentran estrechamente relacionadas (Informáticas, 2014). Ejemplo: la entidad *Authcode* contiene varias funcionalidades estrechamente relacionadas con los datos que maneja.
- **Controlador**: En Symfony2 todas las peticiones web son manejadas por el controlador frontal, que es el único punto de entrada de toda la aplicación en un entorno determinado, el cual se encuentra en la carpeta web de cada proyecto de Symfony2. También este patrón se evidencia en la solución a través de las clases controladoras que se encuentran en la carpeta Controller perteneciente a cada bundle de la solución.
- **Bajo acoplamiento**: Esta característica permite potenciar la reutilización y disminuye la dependencia entre las clases. En el sistema la clase *ClientController* al acceder a las entidades a través del método *getRepository*,

asegura que exista un grado moderado de acoplamiento entre las clases pues no existen dependencias directas entre ellas.

2.7 Modelo de datos

Uno de los artefactos generados en la disciplina de Análisis y diseño es el modelo de datos. Un modelo de datos es una serie de conceptos que puede utilizarse para describir los datos de acuerdo con reglas y convenios predefinidos y luego ser manipulados (Jiménez, 2011). Se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema, así como la correlación entre las clases de diseño y las estructuras de datos persistentes. En otras palabras, permite describir las estructuras de datos de la base de datos, su tipo, descripción y la forma en que se relacionan. A continuación, se presenta el modelo de datos del protocolo implementado el cual cuenta con un total de 5 tablas.

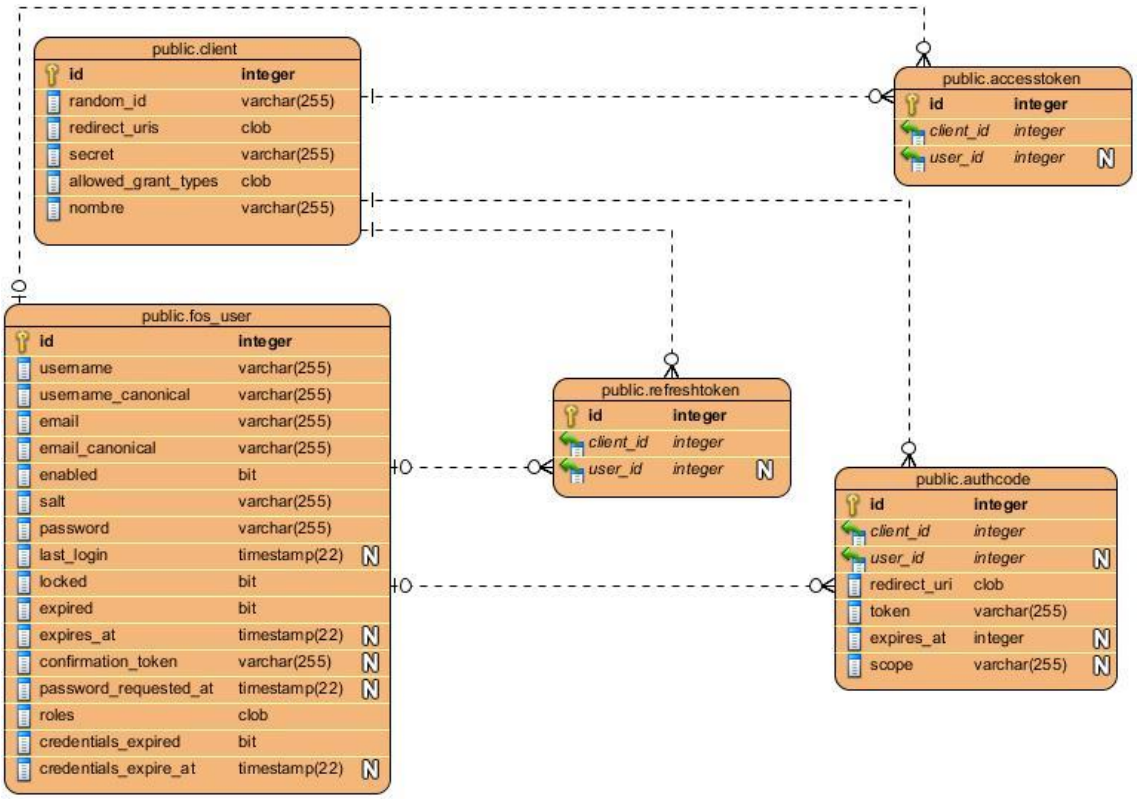

Figura 9. Modelo de Datos.
Fuente: (Elaboración propia)

2.8 Verificación del diseño

Para verificar la calidad del diseño de la solución se emplearon las métricas RC, TOC aplicadas a los indicadores: responsabilidad, complejidad de implementación y reutilización, clasificándolos en altos, medios o bajos.

2.8.1 Métrica Tamaño Operacional de Clase (TOC)

A continuación, se muestran las clases del sistema evaluadas en los atributos de calidad mencionados. La responsabilidad y la complejidad son inversamente proporcionales a la reutilización, por lo que, a mayor responsabilidad y complejidad de implementación de una clase, menor será su nivel de reutilización.

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
ClientController	5	Alta	Alta	Baja
ClientCommand	3	Media	Media	Media
Usuario	0	Baja	Baja	Alta
AccessToken	3	Media	Media	Media
AuthCode	2	Baja	Baja	Alta
Client	2	Baja	Baja	Alta
RefreshToken	2	Baja	Baja	Alta

Tabla 5. Evaluación de las clases del sistema mediante la métrica TOC
Fuente: (Elaboración propia)

2.8.1.1 Resultados de la aplicación de la métrica TOC

En el gráfico de la fig. 11 se puede observar el resultado de la evaluación de los atributos de la métrica TOC.

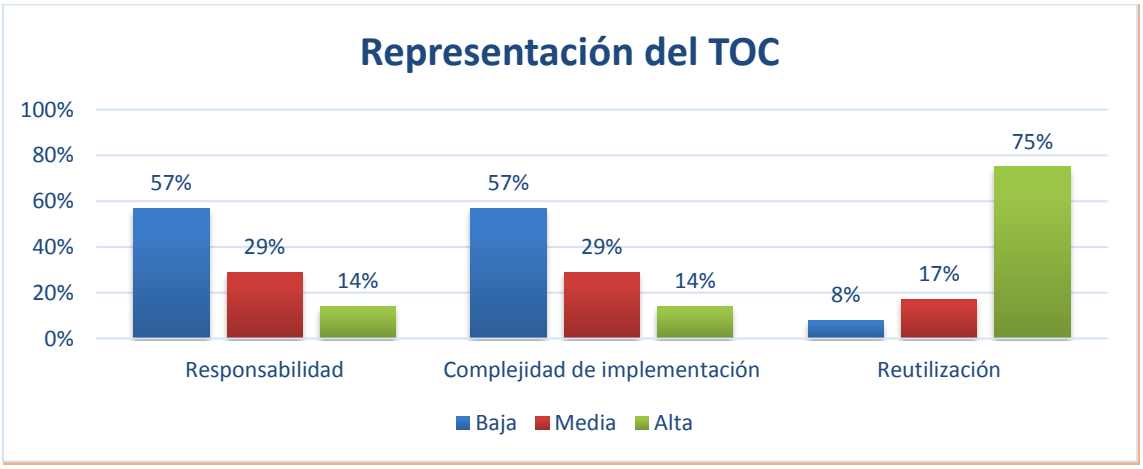

Figura 10. Representación de los resultados de la métrica TOC
Fuente: (Elaboración propia)

Como resultado de la aplicación de la métrica TOC se evidencia que las clases del sistema poseen una baja responsabilidad, baja complejidad de implementación y alta reutilización, por lo que el diseño de las clases en cuanto a cantidad de funcionalidades es satisfactorio.

2.8.2 Métrica Relación entre Clases (RC)

A continuación, se muestran las clases del sistema, evaluadas en los atributos de calidad mencionados.

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
ClientController	1	Bajo	Baja	Alta	Baja
ClientCommand	1	Bajo	Baja	Alta	Baja
Usuario	1	Bajo	Baja	Alta	Baja
AccessToken	2	Medio	Baja	Alta	Baja
AuthCode	3	Alto	Media	Media	Media
Client	5	Alto	Alta	Baja	Alta
RefreshToken	2	Medio	Baja	Alta	Baja

Tabla 6 Evaluación de las clases del sistema mediante la métrica RC
Fuente: (Elaboración propia)

2.8.2.1 Resultados de la aplicación de la métrica RC

En el gráfico de la fig. 12 se puede observar el resultado de la evaluación de los atributos de la métrica RC.

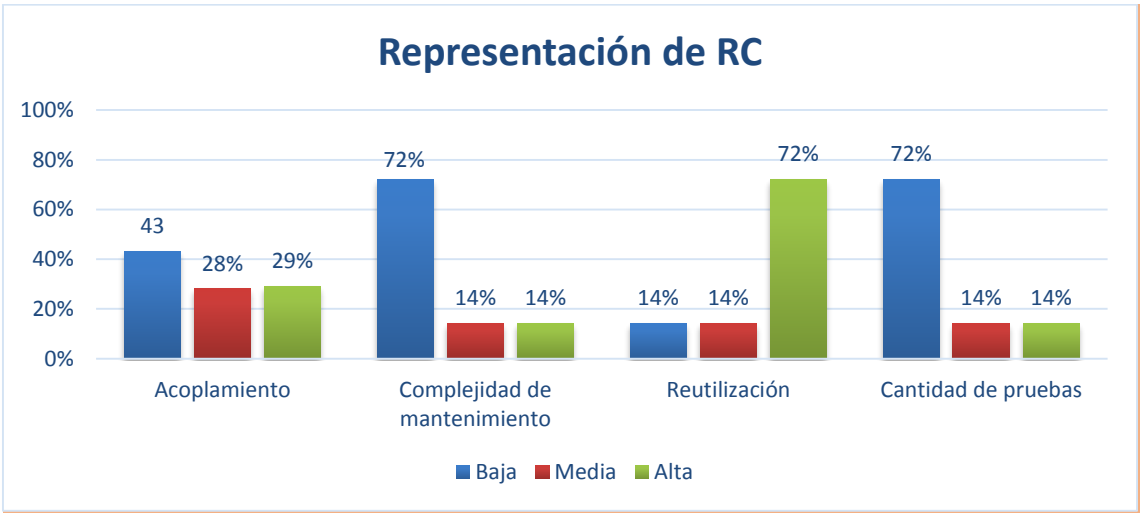

Figura 11. Representación de los resultados de la métrica RC
Fuente: (Elaboración propia)

Como resultado de la aplicación de la métrica RC se evidencia que las clases del sistema poseen un bajo acoplamiento, baja complejidad de mantenimiento, alta reutilización y una baja cantidad de pruebas, por lo que, de forma general, arrojó resultados favorables.

2.9 Conclusiones del capítulo

Al finalizar el presente capítulo se arriba a las siguientes conclusiones:

- La obtención de los requisitos funcionales y no funcionales, permitió definir el comportamiento y restricciones de la especificación del protocolo para su implementación.
- Después de realizar el análisis de la especificación del protocolo, quedaron definidas las Historias de Usuarios proporcionando una comprensión detallada de las funcionalidades del mismo.
- La utilización de las métricas del diseño TOC y RC evidenció que el diseño de las clases presenta baja responsabilidad y complejidad, lo cual favorece la reutilización, la alta cohesión y el bajo acoplamiento, pudiéndose comprobar una correcta asignación de responsabilidades.

Capítulo 3: Implementación y Prueba

3.1 Introducción

En el presente capítulo se describen los estándares de codificación que presenta el protocolo implementado, así como los resultados obtenidos en el proceso de autenticación y autorización de Bosón al aplicar el protocolo OAuth, utilizando el cuasi-experimento como técnica de validación.

3.2 Diagrama de componentes

A continuación, se muestra el diagrama de componente

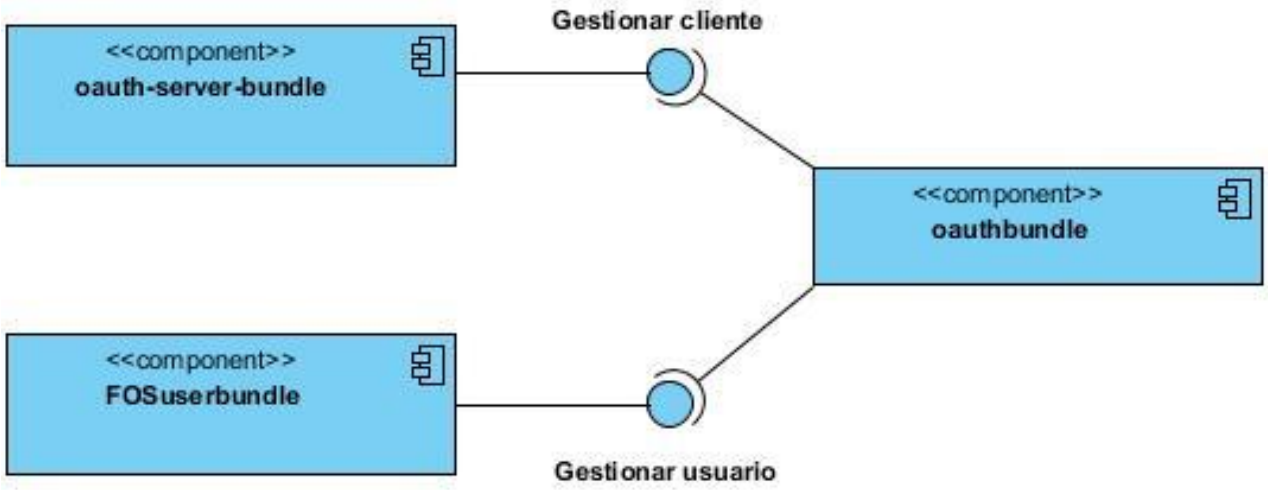

Figura 12 Diagrama de Componentes
Fuente: (Elaboración propia)

Para la implementación del protocolo fue necesario consumir servicios de dos bundles de terceros. FOSuserbundle para la gestión de usuarios y oauth-server-bundle para el registro de los clientes.

3.3 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. El código fuente debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, se establece un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de

software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento según se explica en (Microsoft, 2015).

Para el desarrollo de la solución, se utilizarán estándares y normas de codificación, los cuales se muestran a continuación:

Estándares de nomenclatura.

- **Nomenclatura de clases:** Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing.

Ejemplo:

- **Nomenclatura según el tipo de clases.**
  - **Clase Controladora**

Las clases controladoras después del nombre llevan la palabra: “Controller”. Ejemplo: ClientController

- **Nomenclatura de las funciones.**

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido el sufijo “Action”.

Ejemplo: adicionarAction

- **Nomenclatura de variables.**

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing. Ejemplo: $peticion

Normas de comentarios.

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.

Estilo del código

En la implementación, al escribir las sentencias en php el código debe usar cuatro espacios como indentación no tabuladores.

Las llaves de apertura de las clases y métodos deben ir en la línea siguiente, y las llaves de cierre deben ir en la línea siguiente al cuerpo de la clase.

La visibilidad debe estar declarada en todas las propiedades y métodos.

Los nombres de los métodos no deberán usar un guión bajo como prefijo para indicar si son privados o protegidos.

Los nombres de métodos no deben estar declarados con un espacio después del nombre del método. La llave de apertura debe situarse en su propia línea, y la llave de cierre debe ir en la línea siguiente al cuerpo del método. No debe haber ningún espacio después del paréntesis de apertura, y no debe haber ningún espacio antes del paréntesis de cierre.

3.4 Pruebas de software

3.4.1 Pruebas de caja blanca

Para aplicar las pruebas de caja blanca se empleó la técnica del camino básico. Esta permitió obtener una medida de la complejidad lógica para el diseño de los casos de pruebas y usar dicha medida como guía para la definición de un conjunto básico de caminos de ejecución. Se tomó como ejemplo el método *adicionarAction()*, perteneciente a la clase *ClientController*. Para ello se procede a enumerar las sentencias del código y a partir del mismo se construye el grafo de flujo asociado.

```
public function adicionarAction()
{
    $em = $this->getDoctrine()->getManager();
1   $nombre=$this->getRequest()->request->get('nombre');
    $redirectUri=$this->getRequest()->request->get('redirectUris');
    $clientes = $em->getRepository('OauthBundle:Client')->findAll();

2   foreach ($clientes as $comp) {

3       if (strtoupper($comp->getNombre()) == $nombre || strtolower($comp->getNombre()) == $nombre) {

4           return new Response('{"success": false, "data":{ "textResponse":
            "Ya existe un cliente con este nombre."}}');
        }
    }

    $this->container->get('client.bridge')->adicionarCliente($nombre, $redirectUri);
5   return new Response(json_encode(array("success" => true,
        "data" => array("textResponse" => 'Cliente eliminado'))));
}
```

Figura 13 Código fuente del método adicionarAction
Fuente: (Elaboración propia)

Partiendo del método antes mencionado se obtuvo el siguiente grafo de flujo.

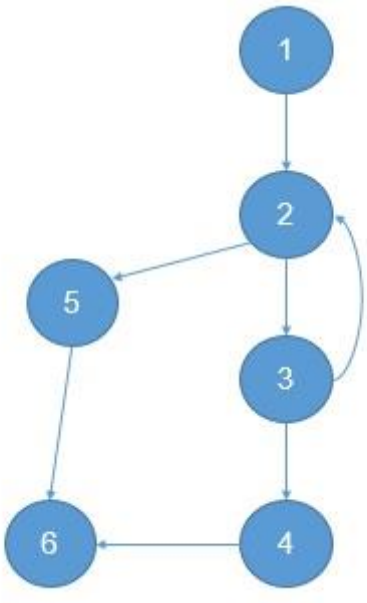

Figura 14. Grafo de flujo del método adicionarAction
Fuente: (Elaboración propia)

Luego de haber realizado la construcción del grafo de flujo se procede a calcular la complejidad ciclomática mediante tres fórmulas que se describen a continuación, las cuales deben exponer el mismo resultado para asegurar que el cálculo de la complejidad es correcto. Cada fórmula V (G) representa el valor del cálculo.

V(G) = (A - N) + 2

Donde A es el número de aristas y N es el número de nodos contenidos en el grafo.

V (G) = (7- 6) + 2

V (G) = 3

V (G) = P + 1

Donde P es el número de nodos predicados contenidos en el grafo. Los nodos predicados son aquellos que parten de dos o más aristas.

V (G) = 2 + 1

V (G) = 3

V (G) = R

Donde R es la cantidad total de regiones. V (G) = 3

El número de regiones del grafo es igual a la complejidad ciclomática.

El cálculo efectuado anteriormente dio como resultado el mismo valor en todos los casos, la complejidad ciclomática es de 3, este valor indica que existen 3 posibles caminos por donde el flujo puede circular y determina el número de casos de prueba que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. A continuación, se representan los caminos básicos por los que puede transitar el flujo:

- Camino 1**:** 1 - 2 - 5 – 6

- Camino 2: 1 - 2 - 3- 2 - 3 - 4 – 6
- Camino 3: 1 - 2 - 3- 2 - 5 - 6

Entrada	$nombre, $redirectUri
Resultados esperados	Adicionar un cliente.
Condiciones	Que no exista un cliente con ese nombre.

Tabla 7. Caso de prueba para el camino 1
Fuente: (Elaboración propia)

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica camino básico se concluye que los mismos fueron probados satisfactoriamente demostrando que el código generado no presenta ciclos infinitos y no existe código innecesario en el sistema desarrollado.

3.5 Validación de las variables de la investigación

Para demostrar que el protocolo propuesto contribuye al fortalecimiento de la seguridad en Bosón, se hace necesario aplicar un proceso de validación. En la presente investigación se utiliza el cuasi experimento como técnica de validación, para comprobar la efectividad de la propuesta.

Los cuasi experimentos son diseños experimentales, donde al igual que en los experimentos verdaderos se manipulan deliberadamente una o más variables independientes sobre una o más variables dependientes, dentro de una situación de control. Se diferencian, sin embargo, por el control de la situación experimental. Los cuasi experimentos se diferencian esencialmente en que la asignación de los grupos no se hace en forma aleatoria, sino que ya están formados antes del experimento y son independientes de este (Grau, y otros, 2004).

Para la aplicación del cuasi experimento, se define como tipología a utilizar el cuasi experimento pre y postprueba, la cual a partir de las observaciones iniciales permite analizar la equivalencia con las observaciones finales. La aplicación de esta técnica en este trabajo, se hace con el objetivo de analizar los efectos que tiene la implementación del nuevo protocolo, en el aumento de la seguridad del proceso de autenticación y autorización de Bosón. La Tabla 8 muestra el diseño cuasi experimental realizado y la simbología utilizada es la siguiente:

G: Grupo de participantes. En este caso se toma como grupo el proceso de autenticación y autorización de Bosón, asociado a dos momentos (Antes de la aplicación del protocolo y después de la aplicación del protocolo).

X: Tratamiento o estímulo. En este caso X corresponde a los cambios realizados en Bosón correspondientes a la aplicación del protocolo de autenticación y autorización propuesto.

O: Observación.

	Antes		**Después**
G_1	O_1	X	O_2

Tabla 8 Diseño cuasi experimental propuesto
Fuente: (Elaboración propia)

3.4.1 Análisis de los resultados

En el primer momento (O_1), se obtuvieron las siguientes observaciones del proceso de autenticación y autorización.

- Para que una aplicación cliente pueda acceder a los recursos protegidos de Bosón (API), el usuario tiene que compartir sus credenciales con la misma (usuario y contraseña), por lo que si es hackeada el atacante puede obtener toda la información del usuario.
- Las aplicaciones de terceros obtienen un acceso demasiado amplio a los recursos protegidos, dejando a su propietario sin ninguna capacidad para restringir la duración de ese acceso, limitar el acceso a sólo un subconjunto de recursos o incluso poder revocar ese acceso por parte de la aplicación a sus recursos.
- Los servidores necesitan soportar autenticación vía password, a pesar de los problemas de seguridad que esto mismo supone. Pueden efectuarse ataques de fuerza bruta, utilizando miles de nombres de usuarios y contraseñas que aprovechan la utilización de claves no seguras. Además, pueden tener lugar ataques de diccionario, que tomando grandes listas de palabras pueden codificarlas de miles de formas distintas para encontrar la contraseña correcta.

En el segundo momento (O_2) luego de la aplicación del modelo, se obtuvieron los siguientes resultados en el proceso de autenticación y autorización:

- Los usuarios no tienen que crear una nueva cuenta en una aplicación de terceros que quiera acceder a sus recursos protegidos en Bosón, sino que se autentican usando su cuenta de Bosón.

- El cliente solicita acceso a los recursos, controlados por el propietario del recurso y almacenado en un Servidor de Recursos, usando para ello unas credenciales no tan comprometedoras para el propietario del recurso (Token de acceso).
- Los ataques que se produzcan en las aplicaciones de terceros con el objetivo de apoderarse de las claves de usuario no tendrán efecto negativo, aun cuando tengan éxito, ya que las credenciales de los usuarios se almacenan en el servidor de Bosón.

Teniendo en cuenta los resultados del análisis del proceso de autenticación y autorización antes y después de la aplicación del protocolo OAuth, se tiene la siguiente comparación en cuanto al nivel de seguridad.

El proceso de autenticación y autorización antes de la aplicación del protocolo OAuth se encontraba limitado, al no poder registrar aplicaciones de terceros y no tener los recursos protegidos bajo seguridad. El protocolo utilizado para la autenticación hacía vulnerable el proceso, al estar sometido a diferentes ataques que ponen en peligro los recursos del sistema. Proporcionar acceso (temporal o permanente) a los recursos es determinante en el aumento de la seguridad que presente la autenticación y autorización. El estándar anterior no realiza un control de acceso a los recursos, por lo que no resulta adecuado para su utilización en aplicaciones que manejan recursos protegidos y requieren mayor nivel de seguridad.

Luego de aplicada la solución, se incorpora el registro de aplicaciones de terceros, para que accedan de forma segura a los recursos. Se elimina el peligro de los ataques que intentan apoderarse de las contraseñas de los usuarios que se registran en una aplicación cliente. Los usuarios pueden conceder un acceso restringido a los recursos que poseen a un cliente externo añadiendo una capa de seguridad a los servicios que ofrece una API, a la vez que protege los datos de los usuarios.

En el análisis anterior se pudo evidenciar que el segundo momento correspondiente al protocolo desarrollado presenta mayores niveles de fortaleza que el escenario original, demostrando el cumplimiento del problema de investigación planteado.

3.5 Conclusiones parciales

Al concluir el presente capítulo se evidencia que:

- Los estándares de codificación empleados en la implementación permitieron ganar en legibilidad, claridad y mayor entendimiento del código por parte de los desarrolladores de Bosón.

- Las pruebas de caja blanca efectuadas a los métodos principales del protocolo mediante la técnica “camino básico” permitieron un estudio de la operatividad del mismo, esto se refiere a que todos los caminos se ejecutan al menos una vez.
- Se validaron las variables que forman parte del problema de la investigación, demostrando que el protocolo implementado ofrece una mayor seguridad para Bosón cumpliéndose con el problema planteado.

Conclusiones generales

Para resolver el problema planteado, en el presente trabajo de diploma se propusieron objetivos, los cuales fueron cumplidos al llevar a cabo una serie de tareas, tras las cuales se puede arribar de forma general a las siguientes conclusiones:

- A través del estudio del arte se fundamentó la necesidad de desarrollar un protocolo que realice control de acceso a los recursos protegidos de Bosón.
- Mediante el estudio de la especificación del estándar, se logró definir las funcionalidades que mejores se adaptan a Bosón.
- Las tecnologías y herramientas seleccionadas, así como la utilización de variación AUP-UCI como metodología de desarrollo, facilitaron la implementación del protocolo, generándose así cada uno de los artefactos de la misma.
- Con la utilización de patrones arquitectónicos y de diseño, se logró implementar el protocolo de acuerdo a los estándares y modelos utilizados en el desarrollo de software que responden a la especificación del estándar.
- Las métricas orientadas al diseño de clases aplicadas al componente, devolvieron valores satisfactorios, validando que se hizo un correcto diseño de la solución propuesta.
- La aplicación de la prueba de caja blanca y del cuasiexperimento, permitió validar que la propuesta de solución cumple con la especificación del estándar y que se podrá usar en aras de fortalecer la seguridad en Bosón.

Por todo lo antes mencionado se evidencia el cumplimiento de los objetivos propuestos en el presente trabajo de diploma, lo cual conlleva a un cumplimiento del objetivo general

Bibliografía

Avilés, Gabriel Gallardo. 2015. *Seguridad en Bases de Datos y Aplicaciones Web.* s.l. : IT Campus Academy, 2015. ISBN 1511544473.

Boyd, Ryan. 2012. *Getting Started with OAuth 2.0.* s.l. : "O'Reilly Media, Inc.", 2012, 2012. ISBN 1449311601, 9781449311605.

Buschmann, F, y otros. 1996. *Pattern-Oriented Software Architecture: A System of Patterns.* 1996.

Calas, Abraham. 2015. *ARQUITECTURA DE REFERENCIA PARA PHP.* La Habana : s.n., 2015.

Doctrine. 2014. Doctrine. [En línea] 2014. [Citado el: 2016 de abril de 15.] http://docs.doctrine- project.org/en/2.0.x/reference/introduction.html..

Eguiluz, Javier. 2013. *Desarrollo web ágil con Symfony2.* s.l. : easybook, 2013.

Fernández Romero, Yenisleidy y Díaz, González, Yanette. 2012. REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES. *Patrón Modelo-Vista-Controlador.* [En línea] 2012. [Citado el: 25 de 3 de 2015.] http://revistatelematica.cujae.edu.cu/index.php/tele/article/viewFile/15/10.

Google. 2012. Google Identity Platform . [En línea] 2012. https://developers.google.com/identity/protocols/AuthSub.

Grau, y otros. 2004. *Metodología de la investigación.* Ibagué -Tolima : Segunda, 2004. ISBN: 958-8028-10-8..

HAMMER-LAHAV, Eran. 2010. *The oauth 1.0 protocol.* 2010. ISSN: 2070-1721.

Hardt, Dick y Jones, Mike. 2012. *The OAuth 2.0 Authorization Framework.* 2012. 2070-1721.

Headquarters, Company. 2006. Visual Paradigm. *10 Reasons to Choose Visual Paradigm.* [En línea] 2006. http://www.visual-paradigm.com/aboutus/10reasons.jsp..

Jiménez, Susana Alejandra López. 2011. *Instituto Tecnológico Superior de los Reyes.* 2011.

Larman, Craig. 2003. *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado.* s.l. : Pearson Educación, 2003. ISBN 8420534382.

Liu, Wendy. 2007. *MVC (Model-View-Controller).* 2007.

Lorenz, Mark y Kidd, Jeff. 1994. *Object-Oriented Software Metrics.* 1994. 9780131792920.

Microsoft. 2003. *DICCIONARIO DE INFORMATICA E INTERNET.* s.l. : S.A. MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, 2003. 9788448138608.

—. 2015. Microsoft Developer Network. *Microsoft Developer Network.* [En línea] Microsoft, 2015. [Citado el: 4 de 5 de 2015.] https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.

Oauth. 2013. OAuth 2.0. [En línea] 2 de abril de 2013. [Citado el: 12 de abril de 2016.] http://oauth.net/2.

—. 2013. OAuth Community Site. [En línea] 02 de Abril de 2013. [Citado el: 12 de abril de 2016.] http://oauth.net.

Packt Publishing. 2013. *Instant PhpStorm Starter.* Birmingham, UK : Livery Place, 2013.

Pfleeger, Charles P. y Pfleeger, Shari Lawrence. 2006. *Security in computing.* s.l. : Prentice Hall Professional, 2006. ISBN: 9780130355485.

PHP, Manual. 2015. PHP: ¿Qué es PHP? [En línea] 2015. [Citado el: 2 de mayo de 2016.] http://www.php.net/manual/es/intro-whatis.php.

Pressman, Roger S. 2009. *Ingeniería del Software Un enfoque práctico.* New York : McGraw-Hill, 2009. 978–0–07–337597–7.

Project, The Apache HTTP Server. 2015. Welcome! - The Apache HTTP Server. [En línea] 2015. [Citado el: 2 de mayo de 2016.]] http://httpd.apache.org..

Real Academia Española. 2001. Diccionario de la Real Academia Española de la Lengua. [En línea] 2001. [Citado el: 19 de junio de 2016.] http://lema.rae.es/drae/.

Roebuck, Kevin. 2012. *Single sign-on (SSO): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors.* s.l. : Emereo Publishing, 2012, 2012. ISBN 1743046642, 9781743046647.

Siriwardena, Prabath. 2014. *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE.* s.l. : Apress, 2014, 2014. 1430268174, 9781430268178.

Suhendra, Vivy. 2011. *A Survey on Access Control Deployment.* Berlin : Springer Berlin Heidelberg, 2011. pp. 11-20..

symfony.es. 2007. Symfony.es. *¿Qué es Symfony?* [En línea] 2007. http://symfony.es/pagina/que-es-symfony/.

Visual Paradigm. 2014. Visual Paradigm for UML. UML & SysML Toolset. [En línea] 2014. http://www.visual-paradigm.com/features/uml-and-sysml-modeling/..

Zakaria, Hosny. 2009. *Metrics for Aspect-Oriented Software Design.* Cairo : s.n., 2009.