



UNIVERSIDAD
REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado

CREACIÓN DE LOS MECANISMOS
NECESARIOS PARA LA CONCESIÓN DE
AUTORIZACIÓN ENTRE UNA APLICACIÓN Y
UNA API A TRAVÉS DE UNA
IMPLEMENTACIÓN DEL ESTÁNDAR OAUTH 2.0

Autor : Pedro Tello Sánchez

Tutor : Pedro de las Heras Quirós

Trabajo Fin de Grado

Creación de los mecanismos necesarios para la concesión de autorización entre una aplicación y una API a través de una implementación del estándar OAuth 2.0.

Autor : Pedro Tello Sánchez

Tutor : Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

Dedicado a aquellos que confiaron en mí.

Agradecimientos

Quiero agradecer este trabajo fin de grado y lo que este fin significa a aquellas personas que me han dado su apoyo, su esperanza, la paciencia y el cariño cuando los necesitaba, los recursos necesarios y muchas veces la motivación que me hacía falta para continuar, desde principio a fin en estos años.

Ellos saben quienes son y les estaré eternamente agradecido.

Resumen

Los tres pilares básicos de los que consta la seguridad de la información son: la confidencialidad, la integridad y la disponibilidad o autorización. Mientras que la confidencialidad es la propiedad que impide la divulgación de información a individuos o entidades no autorizados, la integridad es la propiedad que busca mantener los datos libres de modificaciones no autorizadas y por último, la disponibilidad o autorización es la cualidad de la información de encontrarse en disposición de quienes deben acceder a ella. Este proyecto ha utilizado la propiedad de disponibilidad o autorización, agregando un mecanismo de autenticación para el control de acceso.

Mientras que la autenticación trata de demostrar que una persona es quién dice ser, la autorización trata de discernir el grado de confianza que se concede para la obtención de un recurso o conjunto de recursos específico. El objetivo principal de este proyecto es la creación de los mecanismos necesarios para la concesión de autorización entre una aplicación y una API mediante una implementación del protocolo OAuth 2.0 (RFC 6749) [7]. Para ello se ha creado un mecanismo de control de acceso mediante usuario y contraseña, se ha realizado un proceso de gestión de alta y baja de aplicaciones y finalmente se ha creado desde cero el protocolo OAuth 2.0 utilizando la concesión por código de autorización.

Dicha implementación se ha realizado utilizando Java, JavaScript, HTML y CSS. Para desplegar la aplicación se ha utilizado el servidor de aplicaciones WildFly (en su versión 13). Como base de datos se ha utilizado PostgreSQL.

Este proyecto se ha realizado en el ámbito de la seguridad de la información y la ciberseguridad y trata de demostrar los aspectos positivos de una buena gestión en la autorización de recursos, como lo es la llevada a cabo por el protocolo OAuth 2.0.

Summary

The three basic pillars of which the security of information consists are: confidentiality, integrity and availability or authorization. While confidentiality is the property that prevents the disclosure of information to unauthorized individuals or entities, integrity is the property that seeks to keep the data free of unauthorized modifications and finally, availability or authorization is the quality of the information of be available to those who must access it. This project has used the availability or authorization property, adding an authentication mechanism for access control.

While authentication tries to prove that a person is who they say they are, the authorization tries to discern the degree of trust that is granted to obtain a specific resource or set of resources. The main objective of this project is the creation of the necessary mechanisms for granting authorization between an application and an API through an implementation of the OAuth 2.0 protocol (RFC 6749) [7]. To this end, an access control mechanism has been created through user and password, a process of managing applications has been registered and finally the OAuth 2.0 protocol has been created from scratch using the authorization code concession.

This implementation has been done using Java, JavaScript, HTML and CSS. The application server WildFly (v.13) was used to deploy the application. As a database, PostgreSQL has been used.

This project has been carried out in the field of information security and cybersecurity and tries to demonstrate the positive aspects of good management in the authorization of resources, such as the one carried out by the OAuth 2.0 protocol.

Índice general

1. Introducción	1
1.1. Historia	1
1.2. Diferencias entre autorización y autenticación	2
1.2.1. Autorización	2
1.2.2. Autenticación	3
1.3. Mecanismos de autorización y autenticación	4
1.3.1. SAML	4
1.3.2. OpenID Connect	5
1.3.3. OAuth 2.0	6
2. Objetivos	7
2.1. Objetivo general	7
2.2. Motivación	7
2.3. Objetivos específicos	7
2.4. Metodología	8
2.5. Planificación temporal	10
3. Estado del arte	11
3.1. Tecnologías utilizadas	11
3.1.1. Java	11
3.1.2. PostgreSQL	12
3.1.3. Gradle	12
3.1.4. JavaScript	13
3.1.5. JQuery	13

3.1.6. Ajax	13
3.1.7. Bootstrap	14
3.1.8. REST	15
3.2. Acceso al software del proyecto	15
4. Diseño	17
4.1. Protocolo abstracto de flujo	17
4.2. Tipo de concesión elegida: Código de autorización	19
4.3. Arquitectura general	19
4.4. Arquitectura de BBDD	21
4.4.1. Base de datos de la aplicación	21
4.4.2. Base de datos de la API	23
5. Implementación	27
5.1. Registro y login de usuarios en la aplicación	27
5.1.1. Login	28
5.1.2. Registro	33
5.1.3. Flujos de código	36
5.2. Registro de aplicaciones en la aplicación	38
5.2.1. Alta de aplicaciones	38
5.2.2. Baja de aplicaciones	41
5.2.3. Flujos de código	43
5.3. Proceso de autorización utilizando el estándar OAuth 2.0	47
5.3.1. Flujos de código	51
6. Resultados	55
7. Conclusiones	57
7.1. Conclusiones	57
7.2. Trabajos futuros	58
Bibliografía	61

Índice de figuras

1.1. Autorización VS Autenticación	2
1.2. OpenID Connect Protocol Suite	6
2.1. Esquema de las diferentes etapas	9
2.2. Diagrama de Gantt	10
4.1. Flujo Abstracto	18
4.2. RedOA2_Users	21
4.3. RedOA2_Application_Request	22
4.4. RedOA2_Application_Record	22
4.5. APIOA2_Users	23
4.6. APIOA2_Application_Request	24
4.7. APIOA2_Application_Record	24
4.8. APIOA2_Authorization_Base	25
5.1. Registro y Login	27
5.2. Login First Step	28
5.3. Login Second Step	29
5.4. Login Third Step OK	30
5.5. Login Third Step OK Admin	31
5.6. Login Third Step KO	31
5.7. Login Fourth Step	32
5.8. Register First Step	33
5.9. Register Second Step	34
5.10. Register Third Step	34

5.11. Register Fourth Step	35
5.12. Alta en API	38
5.13. Alta en API - Alta correcta	39
5.14. Alta en API - Alta ya realizada	40
5.15. Baja en API	41
5.16. Baja en API - Baja correcta	42
5.17. Baja en API - Baja no posible	42
5.18. Proceso de Autorización	47
5.19. Página principal	48
5.20. La aplicación no dispone de permisos	48
5.21. Web de login para el usuario	49

Capítulo 1

Introducción

Este Trabajo Fin de Grado se desarrolla en el ámbito de la seguridad y la preservación de la autenticación y de la autorización [8] en Internet. En este capítulo introduciremos los conceptos y tecnologías utilizadas a la vez que se presentará el esquema general de la custodia de la información.

1.1. Historia

Internet se ha convertido en el centro de nuestras vidas. Según un estudio realizado por la compañía Brandwatch ¹ en Abril de 2018 la población mundial era de 7.8 mil millones de personas de los cuales 4.2 mil millones de personas eran usuarios de Internet y 3.03 mil millones eran usuarios de redes sociales y/o estaban registrados en páginas web.

Con tal magnitud de usuarios generando datos en poco tiempo, se llegó a la conclusión, de que la información de los usuarios contenida en diferentes sitios web, podía ser de valor para otras entidades y se comenzaron a crear servicios que permitían el envío de determinada información bajo autorización del propietario de la misma. Así fue como nacieron las API que ofrecían servicios y las Aplicaciones que podrían consumir los mismos.

El problema surgió cuando para acceder a la información de un determinado usuario era necesario suministrar las credenciales de acceso de dicho usuario. Esa información era crítica y bajo ningún concepto se debería suministrar a alguien que no fuera la API o el usuario. Por tanto se necesitaba conseguir que los usuarios, propietarios de los datos, pudieran disponer de

¹www.brandwatch.com

un protocolo seguro mediante el cuál facilitar las credenciales únicamente a la API. Así se llegó a la conclusión de que era necesario encontrar métodos de autenticación que no pusieran en peligro la información personal del usuario a la vez que concedieran los permisos necesarios de autorización para acceder a la información específica que quisiera compartir el usuario.

A lo largo de los años han ido surgiendo diferentes tecnologías que iban proporcionando soluciones tanto al concepto de autorización como al concepto de autenticación.

1.2. Diferencias entre autorización y autenticación

Son dos conceptos que se suelen confundir con bastante frecuencia. Mientras que la autenticación se enfoca en determinar que el usuario es quien dice ser, la autorización se encarga de controlar qué acciones, ese usuario, puede realizar. La siguiente imagen 1.1 muestra de una forma más simple, el objeto que trata de realizar cada uno de los dos conceptos (autenticación izquierda, autorización derecha).

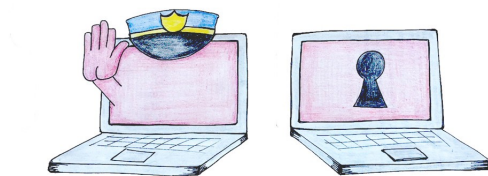


Figura 1.1: Autorización VS Autenticación

1.2.1. Autorización

La autorización se encarga de determinar si tenemos autoridad para hacer algo. Pensemos por ejemplo en las aplicaciones móviles en las que existe una versión Lite (gratuita) y una versión Pro (de pago). Un usuario gratuito está autorizado a acceder a un conjunto limitado de funcionalidades y no está autorizado a otras funcionalidades, mientras no pague la suscripción y se convierta en usuario *Premium*.

Cada vez que un usuario intenta acceder a una funcionalidad, la aplicación realiza un control para determinar si puede hacerlo o no. En segundo plano, lo que está ocurriendo es un proceso de autorización. Siempre que el usuario acceda a recursos para los que tiene permisos, no habrá problema. Sin embargo, en cuanto el usuario intente acceder a un recurso que solo está disponible para la versión Pro, se le denegará el permiso y comunmente se le invita a unirse al paquete *Premium*.

1.2.2. Autenticación

La autenticación se define como el proceso mediante el cual se verifica que un individuo, es quien dice ser. Un ejemplo sería, por ejemplo, el proceso de autenticación que realiza Google. Cuando se inicia sesión introduciendo un correo electrónico y contraseña estamos autenticándonos. Google está comprobando que la contraseña introducida coincide con la contraseña que se asoció a mi correo electrónico en el momento del registro. Si la contraseña coincide con el correo podremos acceder a la aplicación, sin embargo, si la contraseña no coincide con el correo, Google no permitirá el inicio de sesión porque no se puede asegurar que la persona sea quien dice ser.

Debido a que la posibilidad de que nos roben la contraseña existe, en los últimos años ha surgido la autenticación de dos factores. Dicha autenticación consiste en agregar un segundo paso para verificar que la persona es quien dice ser. Entonces, no solo se necesitaría la contraseña sino que se requeriría un paso adicional. La manera más común de realizar dicha autenticación consiste en enviar un mensaje al correo electrónico desde el cual se está intentando acceder o mediante el envío de un código al teléfono móvil de contacto.

Autenticación mediante certificados electrónicos

A partir de los avances tecnológicos actuales podemos predecir que las contraseñas ya no son un método confiable de autenticación de usuario. Este problema, combinado con la creciente amenaza de máquinas malintencionadas, hace que muchos expertos en TI se pregunten como pueden asegurarse de que solo los usuarios y dispositivos aprobados tengan acceso. La solución a este problema lo podemos encontrar en los certificados electrónicos.

La autenticación basada en certificados es el nombre que se le da a utilizar un certificado

digital para identificar a un usuario, máquina o dispositivo antes de otorgar acceso a un recurso, red, aplicación, etc. En el caso de la autenticación de usuario, a menudo se implementa en conjunto con métodos tradicionales como nombre de usuario y contraseña. A diferencia de algunas soluciones que solo funcionan para usuarios, como la biométrica y las contraseñas de un solo uso, la autenticación mediante certificado se puede utilizar para cualquier gestión final: usuarios, máquinas, dispositivos e incluso para gestionar la tecnología creciente de Internet of Things.

La autenticación basada en certificados hace uso de la infraestructura de clave pública (de sus siglas en inglés PKI) para generar un par de claves (una pública y una privada). La privada es personal e intransferible y la pública de libre acceso. De esta forma cualquier intento de acceso por parte del usuario se realizará utilizando la clave privada (firmando el mensaje) para que posteriormente la plataforma destino mediante un mecanismo de validación, haciendo uso de la clave pública de esta entidad, pueda descifrar dicho mensaje. En caso de poder descifrarlo correctamente se podrá afirmar con total certeza que el usuario es quien dice ser y se realizará correctamente la autenticación.

1.3. Mecanismos de autorización y autenticación

A continuación se van a exponer dos mecanismos de autorización y autenticación conjunta para finalizar con el mecanismo de autorización en el que vamos a basar este TFG.

1.3.1. SAML

Security Assertion Markup Language (SAML) es un estándar abierto que funciona transfiriendo la identidad del usuario desde un proveedor de identidad al proveedor del servicio. Esto se produce mediante el intercambio de documentos XML firmados digitalmente.

Un ejemplo de uso sería en el que un usuario inicia sesión en un sistema que actúa como proveedor de identidad. El usuario desea iniciar sesión en una aplicación (el proveedor de servicios). Los pasos serían los siguientes:

1. El usuario accede a la aplicación remota mediante un enlace en una intranet, un marcador o similar y se carga la aplicación.

2. La aplicación identifica el origen del usuario (por subdominio de la aplicación, dirección IP del usuario o similar) y redirige al usuario al proveedor de identidad, solicitando la autenticación. Esta es la solicitud de autenticación.
3. El usuario tiene una sesión de navegador activa existente con el proveedor de identidad o establece una, iniciando sesión en el proveedor de identidad.
4. El proveedor de identidad construye la respuesta de autenticación en forma de un documento XML que contiene el nombre de usuario o la dirección de correo electrónico del usuario, lo firma con un certificado X.509 y publica esta información al proveedor de servicios.
5. El proveedor de servicios, que ya conoce al proveedor de identidad y tiene una huella digital de certificado, recupera la respuesta de autenticación y la valida utilizando la huella digital del certificado.
6. Se establece la identidad del usuario y se le proporciona acceso a la aplicación.

SAML 2.0

SAML 2.0 [1] es una versión del estándar SAML para tramitar el intercambio de datos de autenticación y autorización entre dominios de seguridad. Es un protocolo basado en XML que utiliza tokens de seguridad que contienen aserciones para pasar información sobre un principal (generalmente un usuario final) entre un Proveedor de Identidad, y un Proveedor de Servicios.

SAML 2.0 fue ratificado como un estándar OASIS en marzo de 2005, reemplazando a SAML 1.1.

1.3.2. OpenID Connect

OpenID Connect [9] es un protocolo de autenticación implementada utilizando OAuth 2.0. El estándar está controlado por el OpenID Foundation. Permite a los clientes verificar la identidad del usuario final basándose en la autenticación realizada por un servidor de autorización, así como obtener información de perfil básica sobre el usuario final de manera interoperable y similar a REST.

OpenID Connect permite a los clientes de todo tipo, incluidos los clientes basados en web, móviles y de JavaScript, solicitar y recibir información sobre sesiones autenticadas y usuarios finales. El conjunto de especificaciones es extensible, lo que permite a los participantes utilizar funciones opcionales, como el cifrado de datos de identidad, el descubrimiento de proveedores de OpenID y la administración de sesiones, cuando sea conveniente para ellos.

Las especificaciones de OpenID Connect y las especificaciones en las que se basan se muestran en el diagrama a continuación:

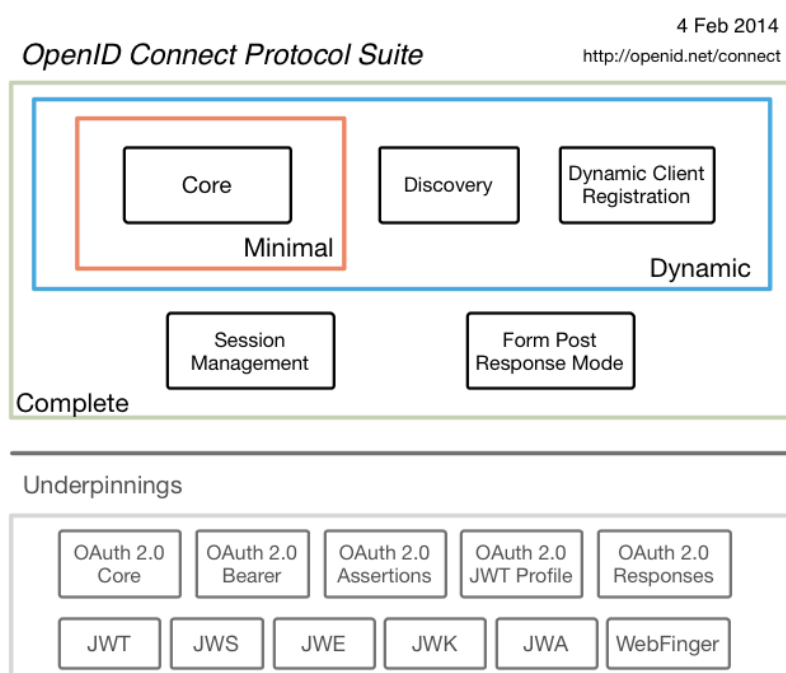


Figura 1.2: OpenID Connect Protocol Suite

1.3.3. OAuth 2.0

El estándar OAuth 2.0 [6] (RFC 6749) [7] es un *framework* de autorización que permite controlar el acceso por parte de las aplicaciones a los datos de los usuarios sin tener que proporcionar las credenciales. Según el estándar, el marco de autorización de OAuth 2.0 permite que una aplicación de terceros obtenga acceso limitado a un servicio HTTP, ya sea en nombre de un propietario de recursos mediante la organización de una interacción de aprobación entre el propietario del recurso y el servicio HTTP, o permitiendo a la aplicación de terceros obtener acceso en su propio nombre.

Capítulo 2

Objetivos

2.1. Objetivo general

Este trabajo fin de grado ha consistido en crear los mecanismos necesarios para la concesión de autorización entre una aplicación y una API a través de una implementación del estándar OAuth 2.0.

2.2. Motivación

Desde que comencé a trabajar en el mundo de la ciberseguridad he ido adquiriendo conocimientos de las principales técnicas, protocolos y estándares que son utilizados actualmente por las empresas y los profesionales en la actualidad. El estándar OAuth 2.0 llamó mucho mi atención por lo bien construido que estaba y la cantidad de sitios en los que se utilizaba. Después de esto me empecé a informar y decidí convertir la implementación de este estándar en mi TFG.

2.3. Objetivos específicos

Los objetivos específicos del proyecto son los siguientes:

1. Elaborar una propuesta de tecnología, arquitectura y límites del proyecto: Dicha propuesta respondería a la finalidad última del proyecto y pondría límites a los trabajos que se llevaría a cabo más adelante.

2. En base a la arquitectura realizar un estudio e implementar el esquema de base de datos a utilizar: Las conclusiones obtenidas serán en base a las necesidades iniciales del primer estudio y teniendo en cuenta en la medida de lo posible mejoras futuras.
3. Creación de la capa de *front* de la aplicación: Esta web deberá disponer de las características necesarias para que se puedan realizar las pruebas necesarias para el testeo de la aplicación. Se tendrán en cuenta características como establecer una interfaz amigable y *responsive*.
4. Administración del registro y login de usuarios en la aplicación: Creación de la funcionalidad específica para la correcta acción de registro y autenticación de usuarios en la aplicación, teniendo en cuenta diversas problemáticas que se puedan dar e intentar anticiparse, así como realizar el mejor control de errores posible.
5. Creación del proceso de registro de aplicaciones en la aplicación: Creación de la funcionalidad específica de alta y baja de aplicaciones en la API. De esta forma la aplicación podrá solicitar acceso a los recursos de los usuarios a la API.
6. Creación del proceso de autorización utilizando el estándar OAuth 2.0: Desarrollo de la funcionalidad específica del estándar OAuth 2.0 basándonos en la documentación establecida en la RFC 6749 [7].
7. Creación del *front* básico de gestión de usuarios de la API: Dicha web deberá disponer de una interfaz simple que permita realizar la autenticación de un usuario en la API.

2.4. Metodología

Este Trabajo Fin de Grado se ha realizado utilizando una metodología de realimentación o feedback basada en la metodología ágil SCRUM [3]. Se basa en que el producto final es el resultado de su propia evolución como consecuencia de las mejoras y problemas que se han ido identificando durante la creación del mismo. En el siguiente esquema podemos observar las diferentes etapas en formación rotativa que se han tenido en cuenta:

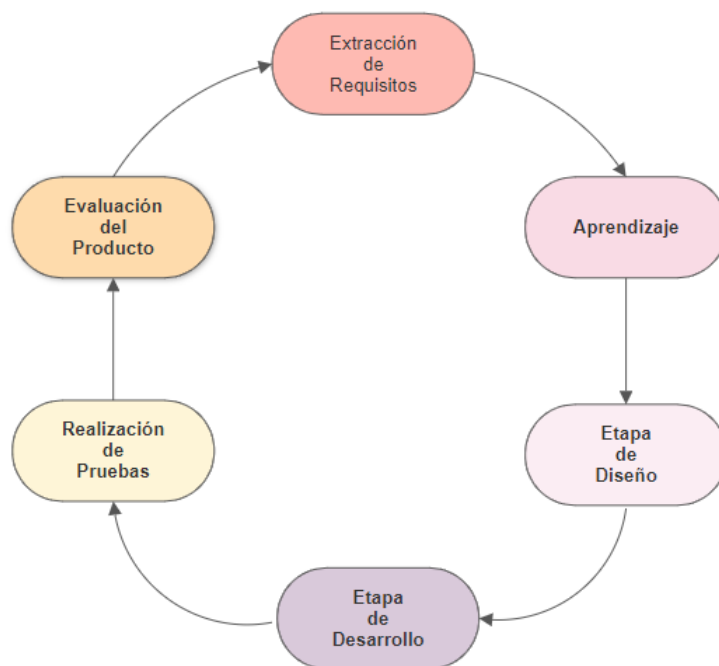


Figura 2.1: Esquema de las diferentes etapas

En la figura 2.1 podemos observar el flujo con las diferentes etapas que se ha seguido en el proyecto.

La etapa que podemos considerar inicial ha sido la de 'Aprendizaje' en la cuál se estudió el protocolo y se dedujeron los componentes que serían necesarios para que el modelo general tuviera sentido. De la mano fueron 'Etapa de diseño' en la cuál se decidió en un primer momento de qué manera se iban a juntar las piezas, para que más tarde, en 'Etapa de desarrollo', tuviéramos claro qué herramientas eran las óptimas. Una vez se tuvo una primera versión se continuó con la 'Realización de Pruebas' para comprobar el correcto funcionamiento del desarrollo y paralelamente se realizó una 'Evaluación del Producto'. Cuando ya se tuvo una idea general del producto parcialmente completo, se comparó con el resultado de la etapa 'Extracción de Requisitos' para, de esta forma, comprobar que se había logrado conseguir lo que se pretendía o aún faltaban cosas. En caso de que aún faltaran cosas se identificaban las diferencias obtenidas y se repetía el ciclo.

2.5. Planificación temporal

La duración de la realización del proyecto ha sido entorno a los 6.5 meses. A continuación se adjunta un diagrama de Gantt [4]:

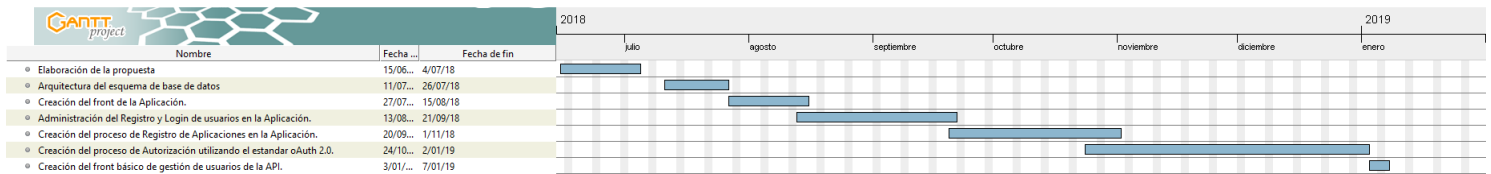


Figura 2.2: Diagrama de Gantt

Tal y como se observa en la figura 2.2, desde el 15 de Junio hasta el 7 de Enero la estimación de horas que he necesitado para la finalización del proyecto ha sido entorno a las 400 horas. De ese total de horas la mayoría han sido en las horas posteriores a la vuelta de la jornada laboral y durante los fines de semana, siendo respectivamente la media de horas de 1 a 2 y de 4 a 14.

Capítulo 3

Estado del arte

En este capítulo se expondrán las diferentes tecnologías y mecanismos utilizados durante la realización del proyecto. Alguno de ellos se han utilizado por disponer de conocimientos más profundos y ahorrarnos tiempo a la hora de llegar al objetivo final y otros por utilizar mecanismos nuevos y ampliar conocimientos.

3.1. Tecnologías utilizadas

3.1.1. Java

Java¹ es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o 'write once, run anywhere'), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. ² El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar el paradigma de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.

¹<https://www.java.com/es/>

²<https://www.oracle.com/technetwork/java/langenv-140151.html>

4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

3.1.2. PostgreSQL

PostgreSQL³ es un potente sistema de base de datos relacional de objetos de código abierto que utiliza y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas.

PostgreSQL se ha ganado una sólida reputación por su arquitectura probada, confiabilidad, integridad de datos, conjunto de características sólidas, extensibilidad y la dedicación de la comunidad de código abierto detrás del software para ofrecer constantemente soluciones innovadoras y de alto rendimiento. PostgreSQL se ejecuta en todos los principales sistemas operativos.

3.1.3. Gradle

Gradle⁴ es una herramienta de automatización de compilación de código abierto centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando un DSL Groovy o Kotlin.

1. Altamente personalizable: Gradle se modela de una manera que es personalizable y extensible de las formas más fundamentales.
2. Rápido: Gradle completa las tareas rápidamente, reutilizando los resultados de ejecuciones anteriores, procesando solo las entradas que han cambiado y ejecutando tareas en paralelo.
3. Potente: Gradle es la herramienta de compilación oficial para Android y es compatible con muchos lenguajes y tecnologías populares.

³<https://www.postgresql.org/>

⁴<https://gradle.org/>

3.1.4. JavaScript

JavaScript⁵ (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript⁶. Se define como orientado a objetos,⁷ basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

3.1.5. JQuery

jQuery⁷ es una biblioteca de JavaScript rápida, pequeña y con muchas funciones. Hace que cosas como la manipulación de documentos HTML, el manejo de eventos, la animación y Ajax sean mucho más simples con una API fácil de usar que funciona en una gran cantidad de navegadores. Con una combinación de versatilidad y extensibilidad, jQuery ha cambiado la forma en que millones de personas escriben JavaScript.

3.1.6. Ajax

Ajax⁸ acrónimo de Asynchronous JavaScript And XML [5] (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios

⁵<https://developer.mozilla.org/es/docs/Web/JavaScript>

⁶<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

⁷<https://jquery.com/>

⁸<https://web.archive.org/web/20080413132458/http://www.librosweb.es/ajax/index.html>

mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

JavaScript es un lenguaje de programación (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

3.1.7. Bootstrap

Bootstrap⁹ es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos *frameworks* web, solo se ocupa del desarrollo *front-end*.

Bootstrap es el segundo proyecto más destacado en GitHub y es usado por la NASA y la MSNBC entre otras organizaciones.

Es interesante el tratamiento web que se obtiene al utilizar Bootstrap, ya que por defecto cualquier página que se realice dispondrá de diseño web *responsive* o adaptativo que es una técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos.

⁹<https://getbootstrap.com/>

3.1.8. REST

La transferencia de estado representacional (en inglés representational state transfer) o REST, es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

3.2. Acceso al software del proyecto

Todo el código fuente del proyecto se encuentra disponible en GitHub en la dirección <https://github.com/Xevit/eclipse-workspace>. También se puede acceder al repositorio de la memoria a través del siguiente enlace <https://github.com/Xevit/Memo>.

Capítulo 4

Diseño

En este capítulo se describe detalladamente el proceso de diseño que se ha seguido para la realización del proyecto.

4.1. Protocolo abstracto de flujo

Para entender las decisiones que se han tomado durante la realización del proyecto es necesario que primero identifiquemos las características del protocolo OAuth 2.0. Dicho protocolo define cuatro roles [2]:

- **Propietario del recurso:** Usuario que autoriza a una aplicación para acceder a su cuenta. El acceso de la aplicación a la cuenta del usuario está limitado al 'alcance' de la autorización otorgada (por ejemplo lectura/escritura).
- **Cliente:** Aplicación que quiere acceder a la cuenta del usuario. Antes de que pueda hacerlo deber ser autorizado por el usuario, y la autorización debe ser validada por la API.
- **Servidor de recursos/Servidor de Autorización:** El servidor de recursos aloja las cuentas de usuario protegidas, y el servidor de autorización verifica la identidad del usuario y luego emite tokens de acceso a la aplicación.

El diagrama de la figura 4.1 muestra un protocolo abstracto de flujo:

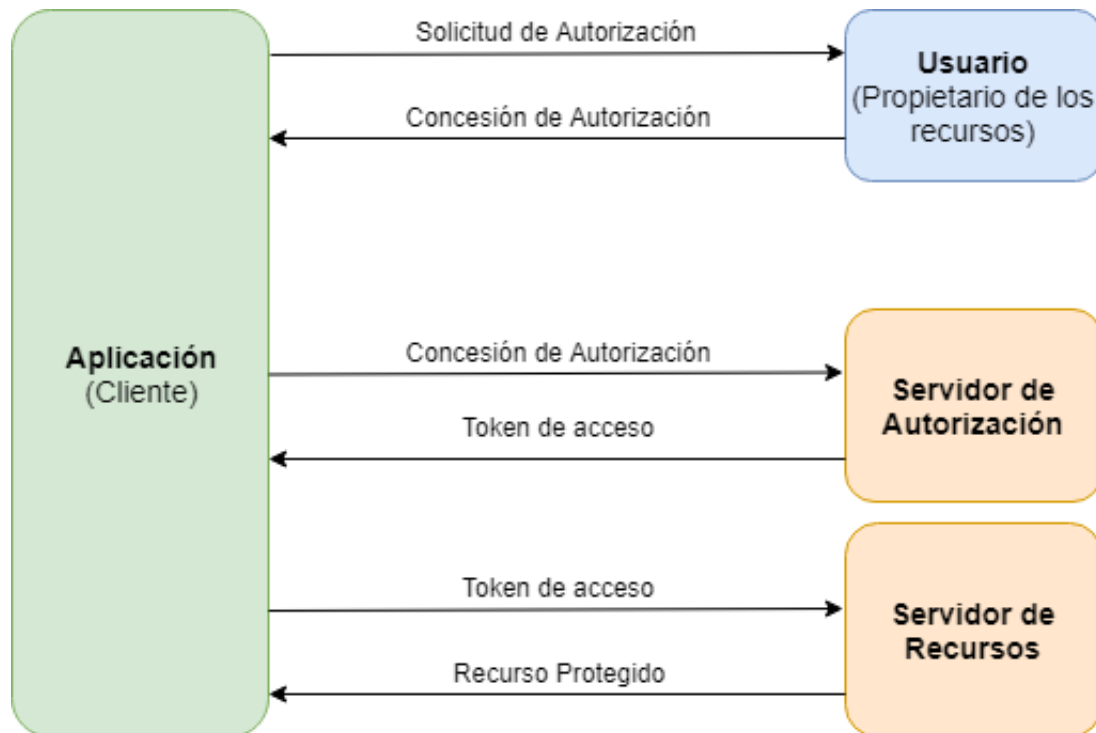


Figura 4.1: Flujo Abstracto

En el flujo de protocolo abstracto anterior 4.1, los primeros cuatro pasos cubren la obtención de una concesión de autorización y un token de acceso. El tipo de concesión de autorización depende del método utilizado por la aplicación para solicitar la autorización y los tipos de concesión admitidos por la API. OAuth 2.0 define cuatro tipos de concesión:

- **Código de autorización:** utilizado con aplicaciones del lado del servidor.
- **Implícito:** se utiliza con aplicaciones móviles o aplicaciones web (aplicaciones que se ejecutan en el dispositivo del usuario).
- **Credenciales de contraseña del propietario del recurso:** se utilizan con aplicaciones de confianza, como las que son propiedad del propio servicio.
- **Credenciales del cliente:** utilizadas con el acceso a la API de aplicaciones.

4.2. Tipo de concesión elegida: Código de autorización

El tipo de concesión de código de autorización es el más comúnmente usado porque está optimizado para aplicaciones del lado del servidor, donde el código fuente no se expone públicamente y se puede mantener la confidencialidad del secreto del cliente. Este es un flujo basado en la redirección, lo que significa que la aplicación debe ser capaz de interactuar con el 'agente del usuario' (es decir, el navegador web del usuario) y recibir códigos de autorización API que se enrutan a través del agente del usuario.

El siguiente punto 'Arquitectura general', pasa a describir los desarrollos mediante los flujos e interacciones entre los diferentes roles que se han realizado en este proyecto.

4.3. Arquitectura general

Para que se pueda tener en mente la arquitectura que finalmente ha tenido el proyecto se ha dividido el mismo en tres partes, cada una de las cuales hace referencia a las tres partes importantes que anteriormente enumeramos:

1. Administración del registro y login de usuarios en la aplicación:

- Partes implicadas → **Aplicación - Usuario.**
- Mediante este desarrollo se conseguirá disponer de un servicio de gestión de acceso a usuarios.
- Las opciones consistirán en una pestaña para realizar el acceso (login) y otra pestaña para llevar a cabo el registro de un nuevo usuario.
- No podrán existir dos usuarios con el mismo nombre.

2. Creación del proceso de registro de aplicaciones en la Aplicación: Creación de la funcionalidad específica de alta y baja de aplicaciones en la API:

- Partes implicadas → **Aplicación - API.**
- Funcionalidad a la que únicamente tendrá acceso el usuario administrador.

- Consistirá en el desarrollo de dos botones. Uno de ellos tendrá la capacidad de realizar el proceso de comunicación entre la aplicación y la API para registrar dicha aplicación en la lista de confianza de la API.
 - El segundo botón tendrá la capacidad de realizar el proceso de comunicación entre la aplicación y la API para eliminar nuestra aplicación de la lista de confianza de la API. Este botón se usará en el caso en el que no queramos que nuestra aplicación figure en la lista de confianza de la API.
3. Creación del proceso de autorización utilizando el estándar OAuth 2.0: Desarrollo de la funcionalidad específica del estándar OAuth 2.0 basándonos en la documentación establecida en la RFC 6749 [7]:
- Partes implicadas → **Aplicación - Usuario - API.**
 - Esta funcionalidad funcionará correctamente solamente en caso de que previamente hayamos dado de alta nuestra aplicación en la lista de confianza de la API (ver punto 2).
 - Consistirá en la creación de un botón que desarrolle la funcionalidad completa del protocolo OAuth 2.0.
 - Dentro de los pasos necesarios para la correcta ejecución del protocolo se encuentra la de mostrar una ventana para que el usuario se autentique contra la plataforma de la API. En caso de no estar dado de alta previamente en dicha API u obtener una autenticación fallida el proceso completo dará error y no se obtendrá token de acceso.

En el siguiente capítulo de implementación desarrollaremos cada una de estas tres partes.

4.4. Arquitectura de BBDD

La arquitectura de base de datos(BBDD) consta de dos partes diferenciadas. Por un lado tenemos las tablas de la BBDD de la aplicación (REDOA2) y por otro lado las tablas de la API (APIOA2).

4.4.1. Base de datos de la aplicación

Las tablas correspondientes a la aplicación son las siguientes:

REDOA2_USERS

La figura 4.2 es un ejemplo de la forma que va a tener la BBDD.

	123 id	ABC nombre	ABC usuario	ABC password	ABC email
1	2	pacol	pacol	1234	ptellos@gmail.com
2	5	Claudia	Claudia	1234	claudia@gmail.com

Figura 4.2: RedOA2_Users

En esta BBDD se van a almacenar los datos de los usuarios que se registren en la aplicación.

- **id**: Identificador del número de fila en el que se encuentra este registro.
- **nombre**: Nombre del usuario que se ha registrado en la plataforma.
- **usuario**: Usuario de acceso a la plataforma. En caso de realizar el login se deberá indicar este usuario.
- **password**: Contraseña de acceso a la plataforma. En caso de realizar el login se deberá indicar este password.
- **email**: Email de contacto del usuario registrado.

REDOA2_APPLICATION_REQUEST

La figura 4.3 es un ejemplo de la forma que va a tener la BBDD.

	ABC url_redirect	ABC code_secret	ABC client_id	ABC client_secret
1	http://localhost:8080/APIOAuth2	NDUwODM5Nzg=	9aw82jvhJ9PwUi0VD6GUt	7kt06XFXfw6DNWxTlhNBOAf

Figura 4.3: RedOA2_Application_Request

Una vez que se ha realizado el alta de una nueva aplicación en la API se nos facilitan los datos de la siguiente tabla.

- **url_redirect**: URL a la que se tiene que enviar la respuesta en la comunicación.
- **code_secret**: Código enviado al correo de la aplicación que quiere darse de alta en la API como factor primario de autenticación. Se almacena en Base 64.
- **client_id**: El CLIENT_ID lo emite la API y actúa como usuario a la hora de solicitar recursos.
- **client_secret**: El CODE_SECRET lo emite la API y actúa como una contraseña a la hora de solicitar recursos.

REDOA2_APPLICATION_RECORD

La figura 4.4 es un ejemplo de la forma que va a tener la BBDD.

	ABC url_redirect	ABC access_token	ABC token_type	123 expires_in	ABC refresh_token
1	http://localhost:8080/APIOAuth2	w2yGrBM1FSurjZrwJSBem	bearer	1,547,760,788,075	L8ImvdfgM0RVoYTnfer6p
2	http://localhost:8080/APIOAuth2	ISB6KFQbgzw2oyXXzB7Lg	bearer	1,547,761,089,482	hg1b9SpqjkWGaYBq5Hwez

Figura 4.4: RedOA2_Application_Record

Una vez que el proceso de autorización se ha realizado correctamente se escribe en esta tabla la información imprescindible para realizar las solicitudes.

- **url_redirect**: URL a la que se tiene que enviar la respuesta en la comunicación.

- **access_token**: Token de acceso proporcionado por la API.
- **token_type**: Token que indica que dicho token es de actualización y que se podrá utilizar para solicitar un nuevo token en el momento en el que se caduque.
- **expires_in**: Tiempo de expiración del Token.
- **refresh_token**: Token de actualización que servirá para solicitar un nuevo 'ACCESS_TOKEN' en el momento en que se supere el 'EXPIRES_IN'

4.4.2. Base de datos de la API

Las tablas correspondientes a la API son las siguientes:

APIOA2_USERS

La figura 4.5 es un ejemplo de la forma que va a tener la BBDD.

	123 id	ABC nombre	ABC usuario	ABC password	ABC email
1	1	ptellos	ptellos	1234	ptellos@gmail.com
2	2	admin	admin	1234	ptellos@gmail.com

Figura 4.5: APIOA2_Users

En esta BBDD se van a almacenar los datos de los usuarios que se registren en la API.

- **id**: Identificador del número de fila en el que se encuentra este registro.
- **nombre**: Nombre del usuario que se ha registrado en la plataforma.
- **usuario**: Usuario de acceso a la plataforma. En caso de realizar el login se deberá indicar este usuario.
- **password**: Contraseña de acceso a la plataforma. En caso de realizar el login se deberá indicar este password.
- **email**: Email de contacto del usuario registrado.

APIOA2_APPLICATION_REQUEST

La figura 4.6 es un ejemplo de la forma que va a tener la BBDD.

	ABC url_redirect	ABC code_secret	ABC client_id	ABC client_secret
1	http://localhost:8080/ApplicationOAuth2	NDUwODM5Nzg=	9aw82jvhJ9PwUi0VD6Gut	7kt06XFXfw6DNWxTlhNBOAf

Figura 4.6: APIOA2_Application_Request

Una vez que se ha realizado el alta de una nueva aplicación en la API se facilitan los datos de la siguiente tabla.

- **url_redirect**: URL a la que se tiene que enviar la respuesta en la comunicación.
- **code_secret**: Código enviado al correo de la aplicación que quiere darse de alta en la API como factor primario de autenticación. Se almacena en Base 64.
- **client_id**: El CLIENT_ID lo emite la API y actúa como usuario a la hora de solicitar recursos.
- **client_secret**: El CLIENT_SECRET lo emite la API y actúa como una contraseña a la hora de solicitar recursos.

APIOA2_APPLICATION_RECORD

La figura 4.7 es un ejemplo de la forma que va a tener la BBDD.

	ABC client_id	ABC client_secret	ABC access_token	ABC refresh_token	123 expires_in
1	bpfVLuSNe78y0CXmIMeYh	M91AeUrqUtSVUh3g3GwMkha	ve5zhTqyHLqR8UsJAzzUU	1LgzWehXwJSFTKwbrpAW9	1,547,580,631,558
2	bpfVLuSNe78y0CXmIMeYh	M91AeUrqUtSVUh3g3GwMkha	Rc2NAmww5IL4F46RffoaO	4rFxC31kS0Xn9unRABqEY	1,547,581,128,363

Figura 4.7: APIOA2_Application_Record

Una vez que el proceso de autorización se ha realizado correctamente se escribe en esta tabla la información imprescindible con la que la aplicación va a realizar las solicitudes.

- **client_id**: El CLIENT_ID lo emite la API y actúa como usuario a la hora de solicitar recursos.

- **client_secret**: El CLIENT_SECRET lo emite la API y actúa como una contraseña a la hora de solicitar recursos.
- **access_token**: Token de acceso proporcionado por la API.
- **refresh_token**: Token de actualización que servirá para solicitar un nuevo 'ACCESS_TOKEN' en el momento en que se supere el 'EXPIRES_IN'
- **expires_in**: Tiempo de expiración del Token.

APIOA2_AUTHORIZATION_BASE

La figura 4.8 es un ejemplo de la forma que va a tener la BBDD.

	ABC client_id	ABC authorization_code
1	bpfVLuSNe78y0CXmIMEyh	bBIQppW6Ri9luzymGTMmc
2	bpfVLuSNe78y0CXmIMEyh	v02CZp2u3PVqViBZKkc4C

Figura 4.8: APIOA2_Authorization Base

Tabla que almacena el paso intermedio en el que se recibe un código de autorización correcto.

- **client_id**: El CLIENT_ID lo emite la API y actúa como usuario a la hora de solicitar recursos.
- **authorization_code**: Clave intermedia en el proceso de autorización.

Capítulo 5

Implementación

Después de realizar el estudio del protocolo RFC 6749 [7], de haber identificado cada una de los tipos de concesión permitidas y de explicar la elección que se ha realizado, en el siguiente capítulo entraremos en profundidad en cada una de las partes en las que se ha dividido la implementación.

5.1. Registro y login de usuarios en la aplicación

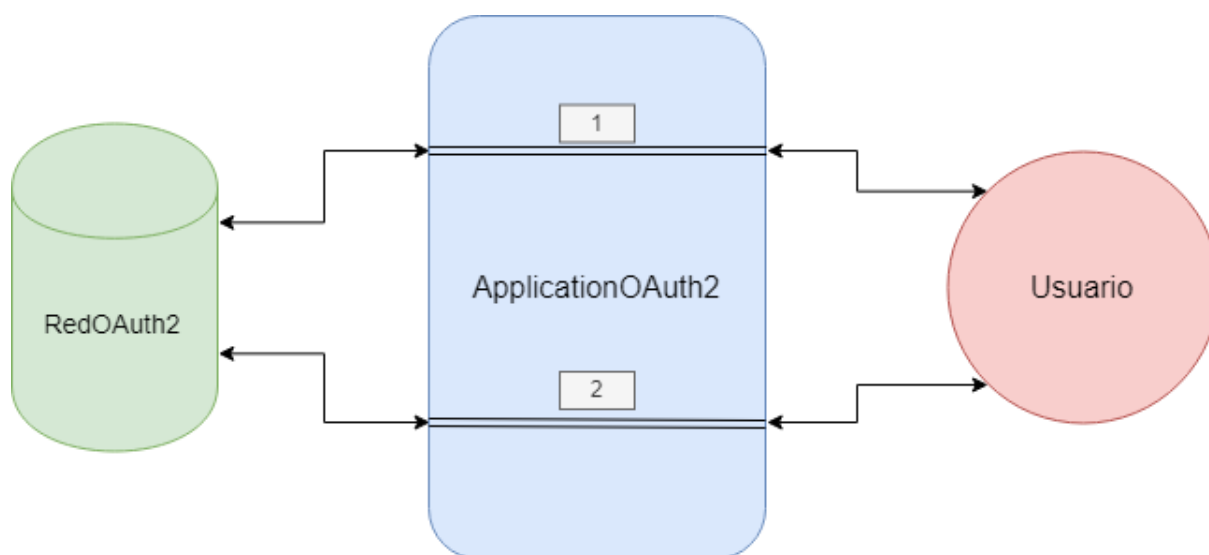


Figura 5.1: Registro y Login

En esta primera parte, tal y como se observa en la imagen 5.1 se puede observar que existen dos flujos que relacionan a la aplicación con el usuario. El primero (1) es el login de usuarios y el segundo (2) es el registro de usuarios en la aplicación. Las siguientes secciones desgranar cada uno de los dos procesos (el proceso de login y el proceso de registro) mostrando el *front* de la aplicación, para finalmente de forma conjunta mostrar los flujos de código.

5.1.1. Login

El proceso completo de login de un usuario sería el siguiente:

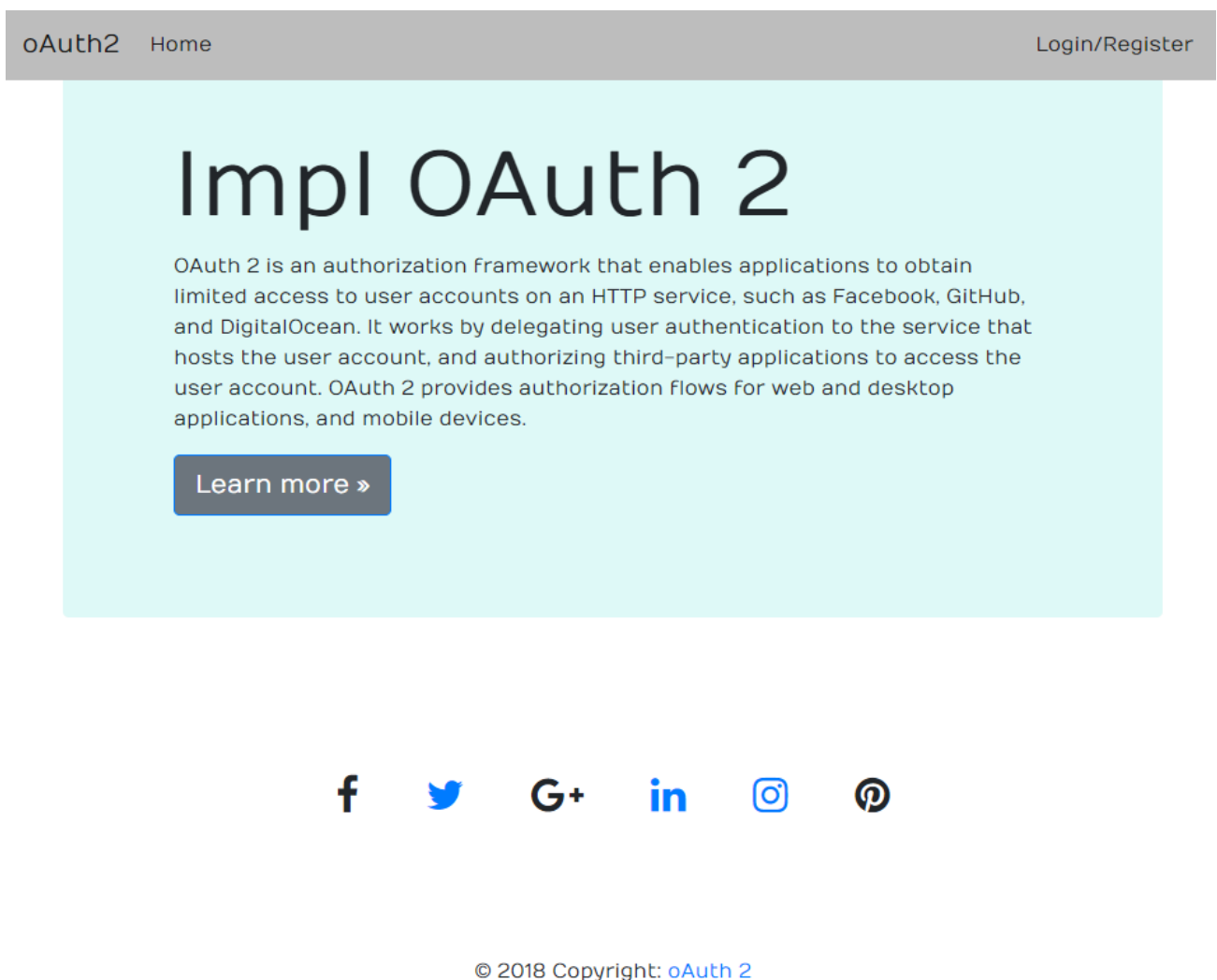


Figura 5.2: Login First Step

Esta figura consiste en la página web principal. Se puede ver un resumen de lo que es el

protocolo OAuth 2.0 y un botón mediante el cuál se accederá a la página principal del estándar. Arriba a la izquierda se dispone del botón 'Home' para volver a la página principal. Arriba a la derecha se tendrán los botones 'Login/Register' mediante los cuales se accederá a una nueva página que dispondrá de la capacidad de acceso a la aplicación.

Abajo se pueden observar iconos de las diferentes Redes Sociales (RRSS). Aquellos que estén en azul redireccionarán a la página personal del creador de este proyecto, en la RRSS específica. Estos iconos, junto con el 'Copyright' se ha establecido de esta manera para dar un aspecto atractivo y de empaque a la página principal.

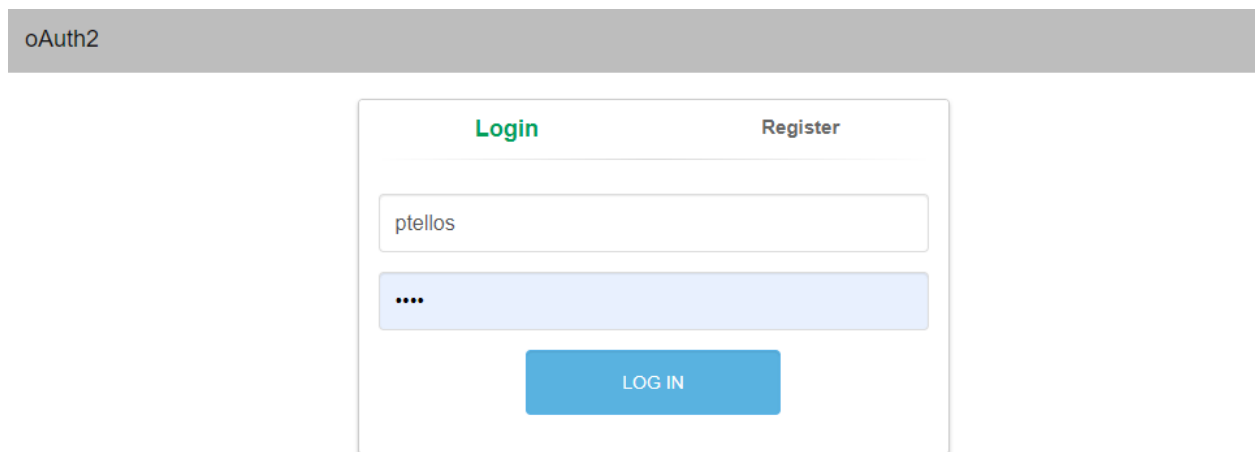
The image shows a web interface for OAuth2 authentication. At the top, there is a grey header bar with the text 'oAuth2'. Below this, there is a white rectangular box containing the login and registration form. The box has two tabs at the top: 'Login' (highlighted in green) and 'Register'. Inside the box, there is a text input field with the placeholder text 'ptellos'. Below the input field is a blue button with three dots '...'. At the bottom of the box is a large blue button labeled 'LOG IN'.

Figura 5.3: Login Second Step

Una vez se redireccione a la pantalla de acceso, se dispondrá de dos pestañas. En la primera tendremos el 'Login', mediante el cuál, después de escribir un nombre de usuario y contraseña dado de alta en la aplicación, se nos redireccionará a la página principal.

En caso de pulsar la pestaña 'Register' se accederá a un formulario que se deberá rellenar para dar de alta un nuevo usuario en la aplicación. En caso de que se intente acceder con un usuario que previamente estuviera dado de alta en la plataforma se indicará.

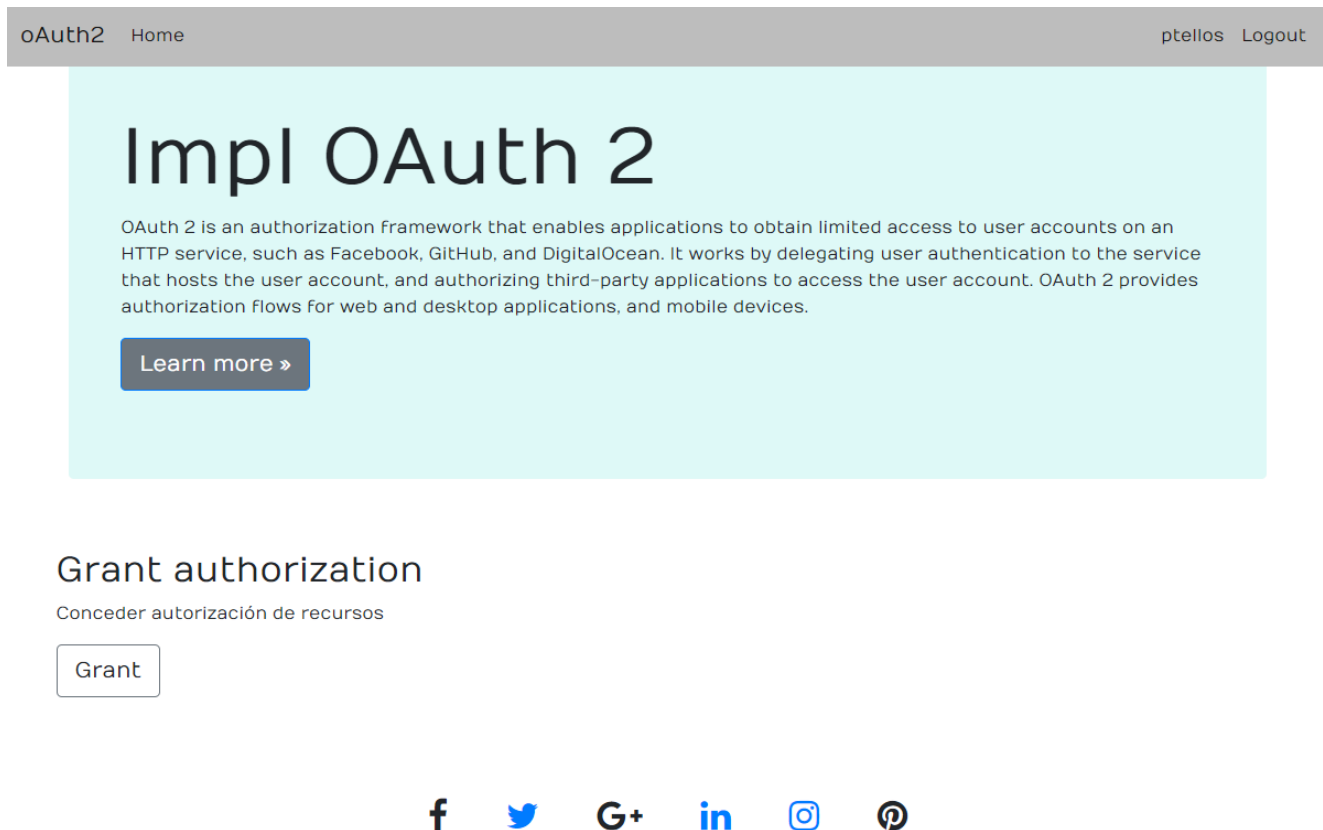


Figura 5.4: Login Third Step OK

En caso de que la autenticación haya sido correcta se redirigirá a la pantalla principal 5.4 mostrándose en la parte superior derecha el usuario con el que se haya autenticado en la aplicación y se abrirá una nueva opción, 'Grant authorization' para comenzar con el proceso de autorización de recursos.

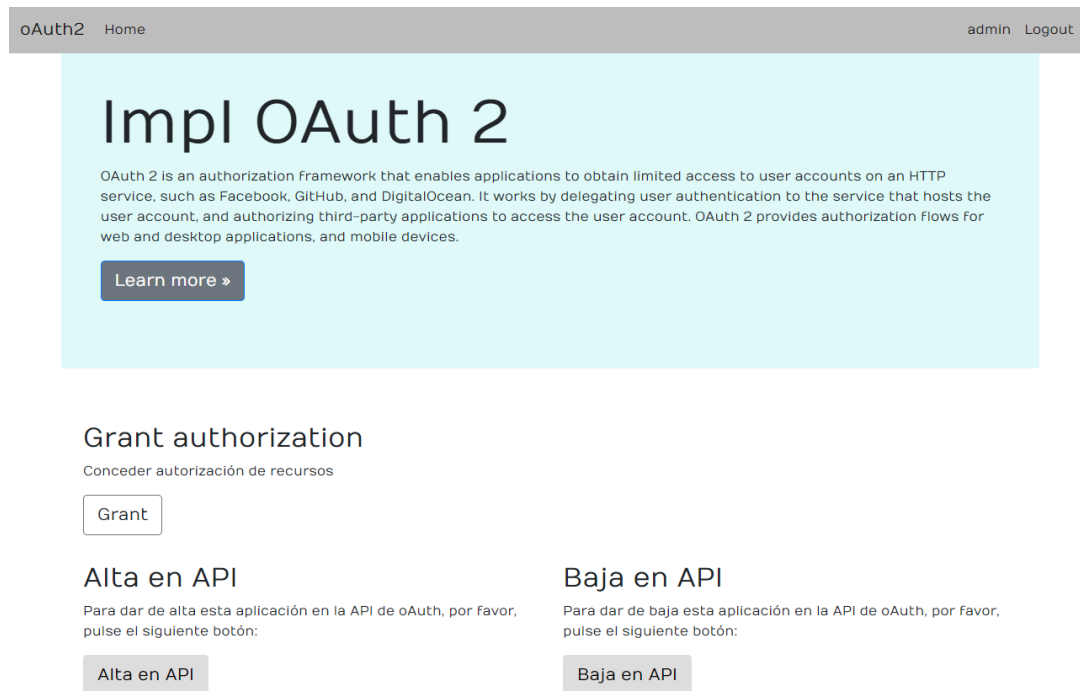


Figura 5.5: Login Third Step OK Admin

En caso de que se haya realizado el acceso utilizando para ello el usuario 'admin' 5.5, además de disponer de la pestaña de 'Grant authorization' se tendrán las pestañas de 'Alta en API' y 'Baja en API'.

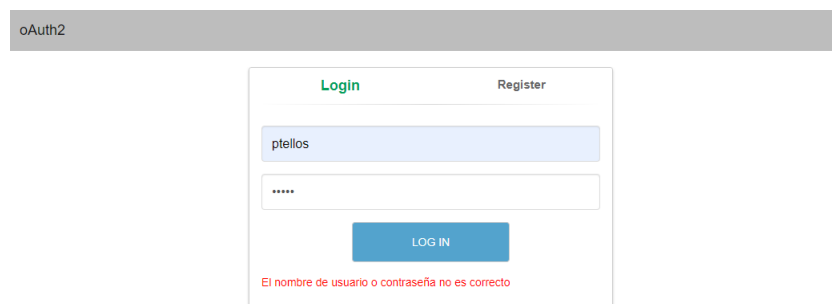


Figura 5.6: Login Third Step KO

En caso que el nombre de usuario o contraseña sean incorrectos aparecerá el mensaje 'El nombre de usuario o contraseña no es correcto'.

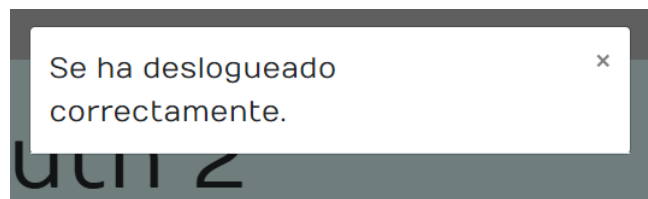


Figura 5.7: Login Fourth Step

En caso de querer salir de la aplicación se deberá pulsar el botón de 'Logout' y se nos informará, mediante el mensaje modal 'Se ha deslogueado correctamente.' que se ha realizado correctamente el logout.

5.1.2. Registro

El proceso completo de registro de un usuario sería el siguiente:

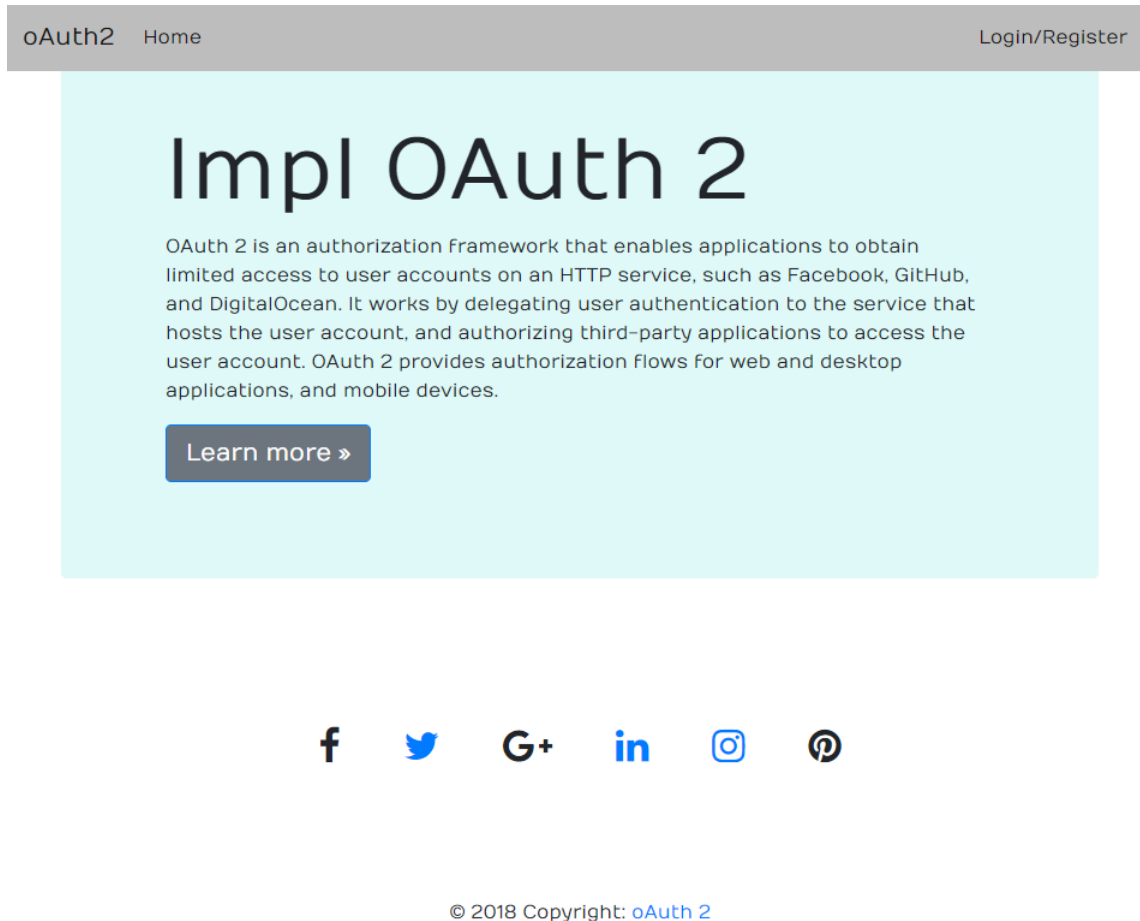
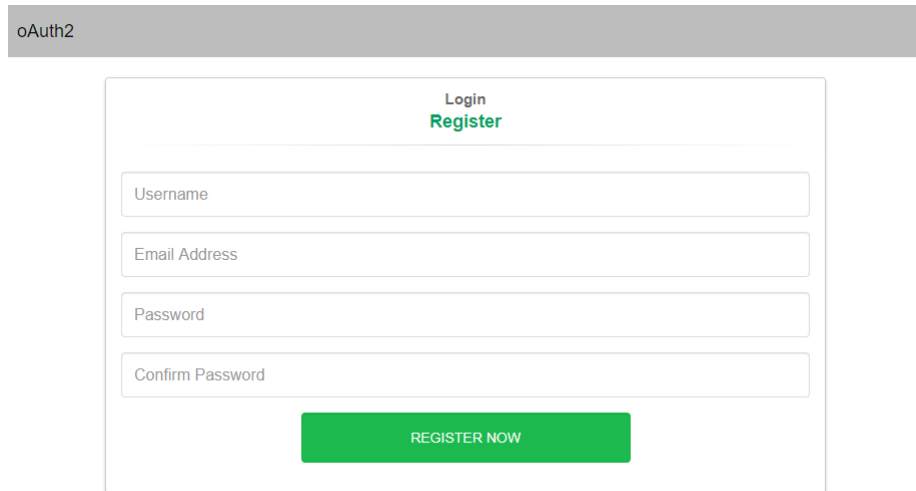


Figura 5.8: Register First Step

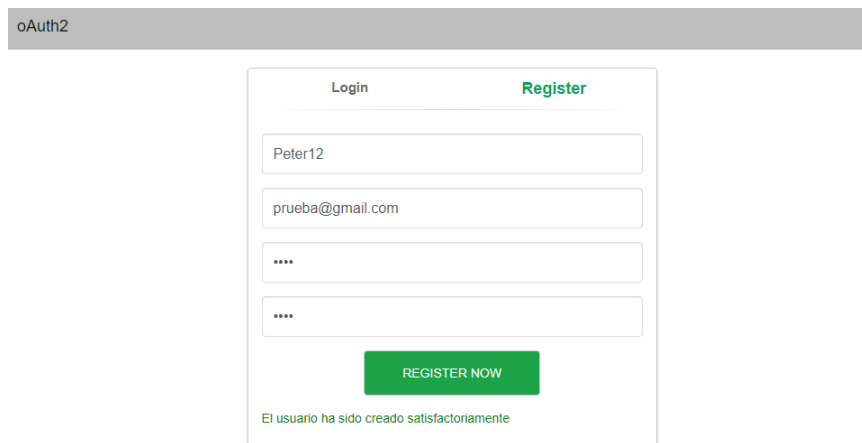
En esta figura se puede ver la web principal de la aplicación. Para acceder al registro se deberá pulsar sobre el botón 'Login/Register' en la parte superior derecha de la figura.



The screenshot shows a web interface with a grey header bar containing the text 'oAuth2'. Below the header is a white rectangular box. At the top of this box, there are two links: 'Login' and 'Register', with 'Register' highlighted in green. Below the links are four input fields stacked vertically: 'Username', 'Email Address', 'Password', and 'Confirm Password'. At the bottom of the box is a green button with the text 'REGISTER NOW' in white capital letters.

Figura 5.9: Register Second Step

Se nos redigirá a la pantalla de autenticación 5.9 o registro donde si pulsamos a la pestaña de 'Register' se podrá realizar un registro de usuario en la aplicación. Para ello se deberá rellenar el 'Username', el 'Email Address', la 'Password' y será necesario rellenar el 'Confirm Password' para confirmar la password.



The screenshot shows the same web interface as Figure 5.9, but now the 'Register' link is highlighted in green. The input fields are filled with the following text: 'Peter12' for Username, 'prueba@gmail.com' for Email Address, and two instances of '****' for Password and Confirm Password. The green 'REGISTER NOW' button is still present. Below the button, a green message reads 'El usuario ha sido creado satisfactoriamente'.

Figura 5.10: Register Third Step

En caso de que el registro sea exitoso se obtendrá un mensaje indicando que el proceso se ha realizado satisfactoriamente 5.10.

oAuth2

Login

Register

admin

prueba@gmail.com

....

....

REGISTER NOW

El usuario ya existe

Figura 5.11: Register Fourth Step

En caso de que el nombre que se intente dar de alta ya se encuentre registrado en la plataforma se obtendrá el mensaje de error 'El usuario ya existe' indicando que el usuario ya estaba dado de alta previamente 5.11.

5.1.3. Flujos de código

La página principal se encuentra en el archivo *index.html* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/index.html>). Al hacer click en el botón 'Login/Register' se redirecciona al JSP *login.jsp* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/login.jsp>). Este JSP tiene cargado un script llamado *ajax.json.log.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/ajax.json.log.js>) con dos funciones Ajax. La primera de ellas se activará al realizar el login y la segunda al realizar el registro.

1. La primera redireccionará a la URL **/login_usuario** que estará gestionada por la clase *LoginServlet.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/LoginServlet.java>). En esta clase se comprobará que exista el usuario y contraseña que nos envían. Para comprobar la existencia del usuario se llamará al método de la clase *DAOLogin.existUser* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAOLogin.java>):
 - a) En caso de existir se creará una cookie (básica) para la persistencia de la sesión y se devolverá la resolución de la clase al archivo *ajax.json.log.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/ajax.json.log.js>). Este redireccionará a la página principal. En la clase principal se podrá observar como se dispone de una sesión abierta con el usuario logueado además de tener disponible los botones de interacción con la aplicación. En caso de haberse realizado el login con el usuario 'admin' se dispondrá de los botones de Alta y Baja de Aplicaciones en la API.
 - b) En caso de no existir se devolverá la resolución de la clase al archivo *ajax.json.log.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/ajax.json.log.js>).

`ApplicationOAuth2/src/main/webapp/js/ajax-json-log.js`). Este mostrará un mensaje indicando que el usuario y/o contraseña no son válidos.

2. La segunda redireccionará a la URL **/registra_usuario** que estará gestionada por la clase *RegisterServlet.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/RegisterServlet.java>). En esta clase se comprobará que el usuario no existe a través del método de la clase *DAORegistro.existUser* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/RegisterServlet.java>):
 - a) En caso de que el usuario ya exista se finalizará el hilo de la clase y se devolverá la resolución al archivo *ajax-json-log.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/ajax-json-log.js>). Este mostrará un mensaje indicando que el usuario que se está intentando dar de alta ya existe en la Aplicación.
 - b) En caso de que el usuario no exista se realizará una llamada al método de la clase *DAORegistro.setUser* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAORegistro.java>) para que realice una inserción en BBDD de todos los campos que ha facilitado el usuario. Posteriormente finalizará el hilo de la clase y se devolverá la resolución al archivo *ajax-json-log.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/ajax-json-log.js>). Este mostrará un mensaje indicando que el usuario ha sido creado satisfactoriamente.

5.2. Registro de aplicaciones en la aplicación

Para que podamos utilizar el protocolo OAuth 2.0 previamente se tiene que cumplir el paso de que la aplicación que quiere solicitar recursos de un usuario en la API, esté dado de alta en dicha API.

En este punto entraremos en detalle en el registro de aplicaciones en la API así como la baja de dichas aplicaciones.

5.2.1. Alta de aplicaciones

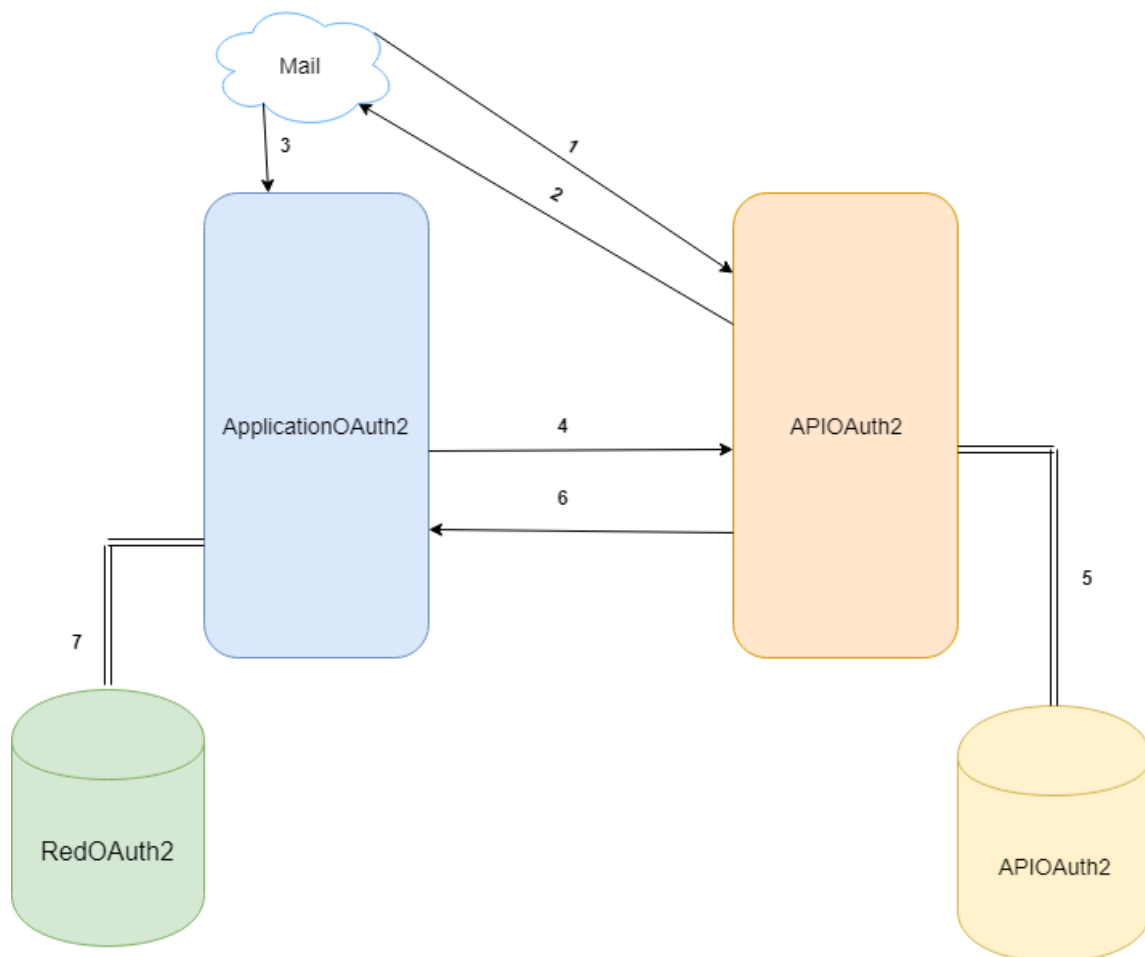


Figura 5.12: Alta en API

Este esquema 5.12 muestra el alta de la Aplicación en la API. Los diferentes pasos que se dan hasta llegar a dicho propósito serían los siguientes:

1. Se manda correo electrónico a la API para comenzar el proceso de alta. En dicho correo se deberá enviar la URL desde la cuál se va a hacer la petición de alta.
2. La API responde al correo electrónico enviando también un mail con un código en Base64. De esta forma se indicará que la aplicación cumple los requisitos impuestos por la API.
3. Se carga el código en Base64 en la aplicación.
4. Se manda la primera solicitud directa entre aplicación y API. Dicha solicitud se realizará mediante REST y llevará en la URL la aplicación que la realiza, así como el código en Base64.
5. La API lee la URL y va a buscar en su BBDD si existe alguna aplicación dada de alta que corresponda con los datos que se le mandan en la petición.
 - a) En caso de que la aplicación ya estuviera dada de alta se enviará un mensaje de vuelta a la aplicación indicando que ya estaba dada de alta.
 - b) En caso de que la aplicación no estuviera dada de alta en la API se generará un código **CLIENT_ID** y un código **CLIENT_SECRET**. Se almacenará en la BBDD de la API toda la información correspondiente a la misma y se responderá indicando que dicha aplicación no estaba dada de alta en la API y facilitándole los dos códigos generados.
6. En la parte de la aplicación se nos indicará el resultado del proceso del alta:
 - a) El siguiente mensaje 5.13 es el que se obtendrá en caso de que el alta se haya realizado correctamente.

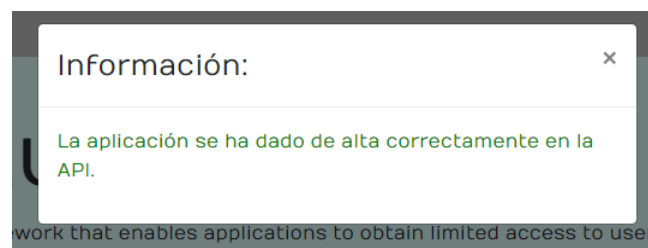


Figura 5.13: Alta en API - Alta correcta

- b) El siguiente mensaje 5.14 se obtendrá en caso de que la aplicación ya estuviera dada de alta en la API.

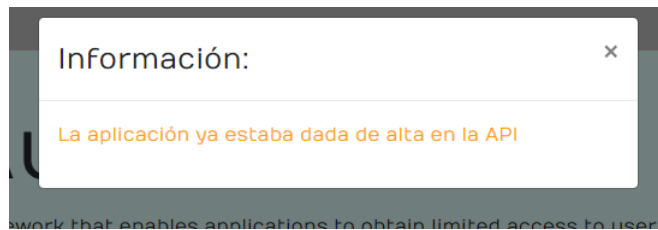


Figura 5.14: Alta en API - Alta ya realizada

7. Para finalizar, en caso de que la aplicación no estuviera dada de alta se almacenarán los datos facilitados por la API en la BBDD de la aplicación.

5.2.2. Baja de aplicaciones

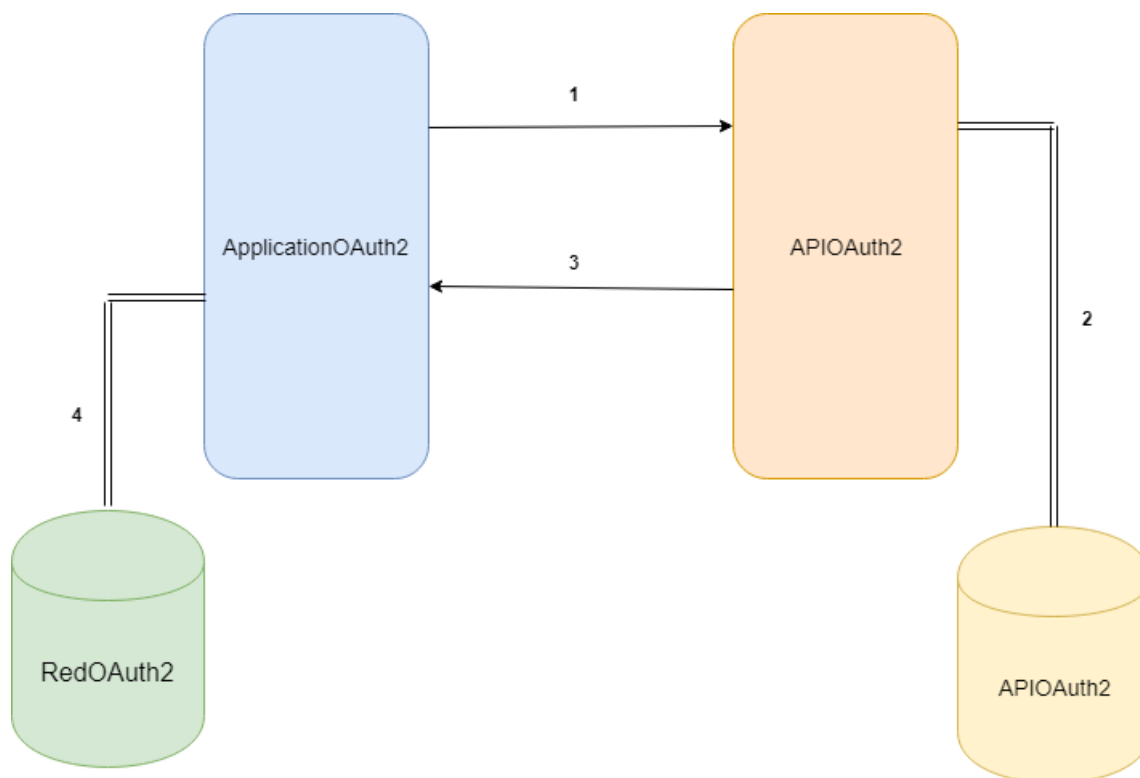


Figura 5.15: Baja en API

Este esquema 5.15 muestra la baja de la aplicación en la API. Los diferentes pasos que se dan hasta llegar a dicho propósito serían los siguientes:

1. Se manda una solicitud directa desde la aplicación a la API. Dicha solicitud se realizará mediante REST y llevará en la URL la aplicación que la realiza, así como el código inicial en Base64.
2. La API lee la URL y va a buscar en su BBDD si existe alguna aplicación dada de alta que corresponda con los datos que se le mandan en la petición.
 - a) En caso de que la aplicación estuviera dada de alta se borrará de la BBDD de la API toda la información correspondiente a la misma y se responderá indicando que dicha aplicación estaba dada de alta en la API.

5.2.3. Flujos de código

Alta en API

1. Se llama al archivo *apiModal.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/apiModal.js>) que tendrá una función Ajax mediante la cuál mandaremos un mensaje POST al recurso **ApplicationOAuth2/AltaEnAPI** que se encuentra en la clase *ApplicationRegistering.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationRegistering.java>) de la Aplicación.
2. Desde la clase *ApplicationRegistering.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationRegistering.java>) se llamará a la siguiente URL mediante mensaje GET:
`http://localhost:8080/APIOAuth2/Register? url_redirect=http://localhost:8080/ApplicationOAuth2/AltaEnAPI/Confiramation? code_secret=[CODE_SECRET]`
3. La anterior URL será manejada por la clase *ListenerRegistrationRequest.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ListenerRegistrationRequest.java>) de la API que llamará al método *DAOResisterApplication.existApp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOResisterApplication.java>) para comprobar si existe o no la aplicación.
 - a) Si la aplicación ya estaba dada de alta, se enviará el siguiente mensaje GET:
`http://localhost:8080/ApplicationOAuth2/AltaEnAPI/Confirmation?exist=true`
 - b) Si la aplicación no estaba dada de alta:
 - 1) Se llama a la clase *RandomStringGenerator.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/>

`java/com/ptellos/api/RandomStringGenerator.java`) para generar el `CLIENT_ID` y el `CLIENT_SECRET`.

- 2) Se llama al método *DAORegisterApplication.registerApplication* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAORegisterApplication.java>) para almacenar todos los valores en la BBDD de la API.

- 3) Se envía el siguiente mensaje GET:

```
http://localhost:8080/ApplicationOAuth2/AltaEnAPI/  
Confirmation?exist=false&clientId=[CLIENT_ID]&  
clientSecret=[CLIENT_SECRET]&url_redirect=http://  
localhost:8080/APIOAuth2
```

4. Tanto la respuesta de que la clase ya estaba creada como la de que no estaba creada se manejarán desde el recurso **ApplicationOAuth2/AltaEnAPI** en el método que maneja las peticiones GET de la clase *ApplicationRegistering.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationRegistering.java>).

a) En caso de que la aplicación ya estuviera creada, se mostrará el siguiente mensaje, 'La aplicación ya estaba dada de alta en la API'.

b) En caso de que la aplicación no estuviera dada de alta:

- 1) Se llamará al método *DAORegisterApplication.registerApplication* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAORegisterApplication.java>) para almacenar todos los valores en la BBDD de la aplicación.
- 2) Se mostrará el siguiente mensaje, 'La aplicación se ha dado de alta correctamente en la API'.

Baja en API

1. Se llama al archivo *apiModal.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/js/apiModal.js>) que tendrá una función Ajax mediante la cuál mandaremos un mensaje POST al recurso **ApplicationOAuth2/BajaEnAPI** que se encuentra en la clase *ApplicationUnsubscribe.java* de la Aplicación.
2. Desde la clase *ApplicationUnsubscribe.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationUnsubscribe.java>) se llamará a la siguiente URL mediante mensaje GET:
`http://localhost:8080/APIOAuth2/Unsubscribe?url_redirect=http://localhost:8080/ApplicationOAuth2/BajaEnAPI/Confirmation?code_secret=[CODE_SECRET]`
3. La anterior URL será manejada por la clase *ListenerUnsubscribeRequest.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ListenerUnsubscribeRequest.java>) de la API que llamará al método *DAOUnsubscribeApplication.existApp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOUnsubscribeApplication.java>) para comprobar si existe o no la aplicación.

a) Si la aplicación estaba dada de alta:

- 1) Se llama al método *DAOUnsubscribeApplication.unsubscribeApplication* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOUnsubscribeApplication.java>) para eliminar todos los valores en la BBDD de la API.
- 2) Se envía el siguiente mensaje GET:
`http://localhost:8080/ApplicationOAuth2/BajaEnAPI/Confirmation?exist=true&url_redirect=http://localhost`

:8080/APIOAuth2

b) Si la aplicación no estaba dada de alta, se enviará el siguiente mensaje GET:

**http://localhost:8080/ApplicationOAuth2/BajaEnAPI/
Confirmation?exist=false**

4. Tanto la respuesta de que la clase ya estaba creada como la de que no estaba creada se manejarán desde el recurso **ApplicationOAuth2/BajaEnAPI** en el método que maneja las peticiones GET de la clase *ApplicationUnsubscribe.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationUnsubscribe.java>).

a) En caso de que la aplicación estuviera dada de alta:

- 1) Se llamará al método *DAOUnsubscribeApplication.java* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAOUnsubscribeApplication.java>) para eliminar todos los valores de dicho registro en la BBDD de la aplicación.

- 2) Se mostrará el siguiente mensaje, 'La aplicación se ha dado de baja en la API'.

b) En caso de que la aplicación no estuviera dada de alta en la API, se mostrará el siguiente mensaje, 'La aplicación no está dada de alta en la API'.

5.3. Proceso de autorización utilizando el estándar OAuth 2.0

A continuación explicamos el proceso de autorización de OAuth 2.0 haciendo referencia a los pasos indicados en la figura con números:

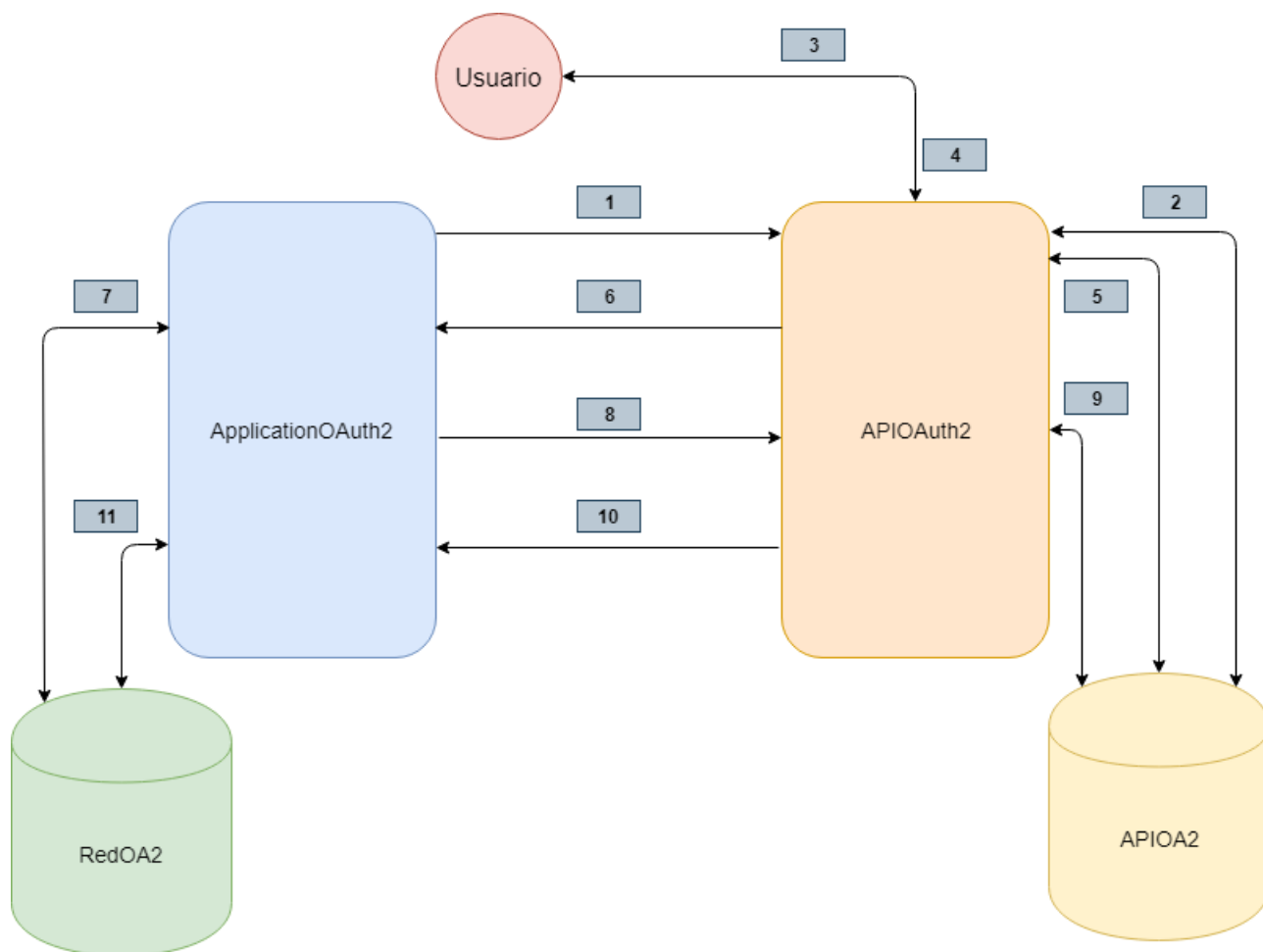


Figura 5.18: Proceso de Autorización

Este esquema 5.18 va a ayudar a entender todo el proceso de la concesión de autorización. Desde que se realiza la petición en la aplicación hasta que terminamos obteniendo las credenciales necesarias.

1. Se hace click en el botón 'Grant' de la pantalla principal de la aplicación. Se mandará una petición a la API de solicitud de concesión de autorización. En dicha petición se indicará el **CLIENT_ID** y la url de redirección.

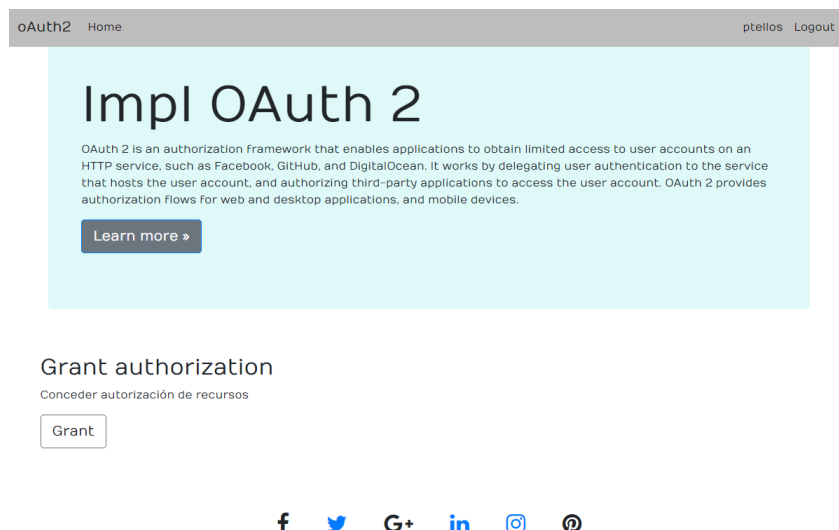


Figura 5.19: Página principal

2. La API gestionará dicha solicitud y buscará en su BBDD si la aplicación que está generando dicha solicitud está dada de alta en la API.
 - a) En caso de que exista se seguirá con el proceso.
 - b) En caso contrario se lanzará la siguiente ventana 5.20, indicando que la aplicación no dispone de permisos para la petición que está lanzando, tal y como se muestra en la siguiente imagen:

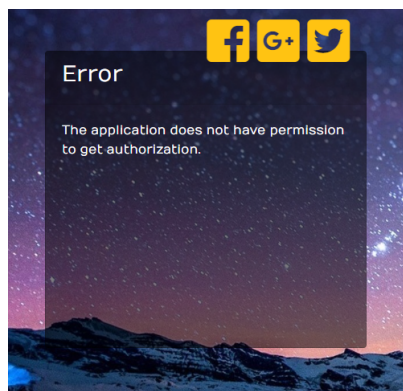


Figura 5.20: La aplicación no dispone de permisos

3. En caso de estar dada de alta en la API se lanzará una web de login 5.21 mediante la cuál el usuario que está intentando dar autorización a la aplicación se autenticará.

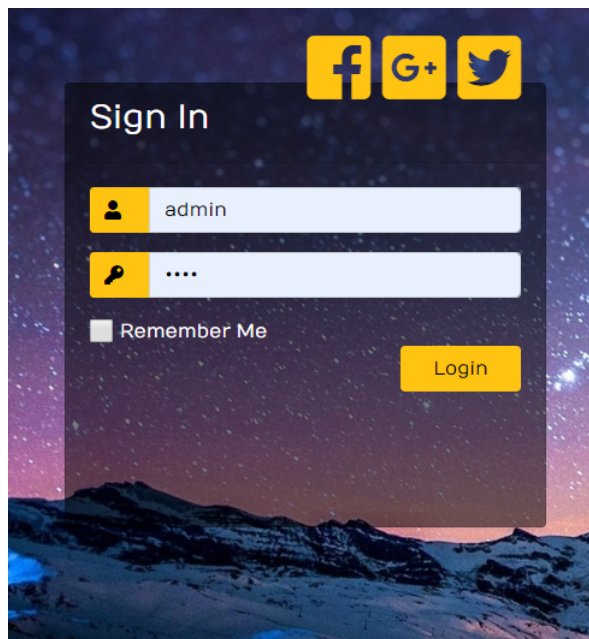


Figura 5.21: Web de login para el usuario

4. Se devolverá a la API un formulario con el usuario y contraseña.
5. La API comprobará buscando en su BBDD si el usuario que está intentando hacer login, existe. En caso de que exista se generará el **AUTHORIZATION_CODE** y se almacenará en la BBDD junto con el **CLIENT_ID**.
6. En caso de que el usuario exista la API realizará una redirección a la aplicación enviando el **CLIENT_ID** de la aplicación y el **AUTHORIZATION_CODE**.
7. En caso de que la aplicación identifique, en la URL de respuesta, el **CLIENT_ID** se gestionará dicha petición y almacenará en su BBDD los datos recibidos.
8. Se continuará con el proceso de concesión de autorización y esta vez la aplicación enviará en la URL, el **CLIENT_ID**, el **CLIENT_SECRET** y el **AUTHORIZATION_CODE** que se acaban de obtener.

9. La API gestionará la petición entrante y en caso de que la petición obtenida sea correcta, generará un **ACCESS_TOKEN** y almacenará todos los datos correspondientes a dicho proceso en su BBDD.
10. Se realizará un último envío desde la API que contendrá un JSON con los datos correspondientes al proceso de concesión de autorización correcta entre los que se encontrarán:
 - a) **ACCESS_TOKEN**: Token de acceso proporcionado por la API.
 - b) **TOKEN_TYPE**: Token de tipo 'bearer' que indica que dicho token es un token de actualización y que se podrá utilizar para solicitar un nuevo token en el momento en el que se caduque el **ACCESS_TOKEN**.
 - c) **EXPIRES_IN**: Tiempo de expiración del Token.
 - d) **REFRESH_TOKEN**: Token de actualización que servirá para solicitar un nuevo **ACCESS_TOKEN** en el momento en que se supere el **EXPIRES_IN**.
 - e) **SCOPE**: Alcance de los permisos que se han obtenido. En este caso se instanciará a 'read'.
11. Finalmente, para concluir el proceso de concesión de autorización, la aplicación almacenará en su BBDD todos los datos enviados por la API.

De esta forma la aplicación conseguirá obtener las credenciales de acceso necesarias para poder realizar consultas a la API y de poder acceder a la información del usuario que se encuentre dentro del rango de autorización establecido (en este caso solo se dispondrían de permisos para leer).

5.3.1. Flujos de código

1. Se pulsa el botón 'Grant' dentro de la página *index.html* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/webapp/index.html>) que se gestionará en **ApplicationOAuth2/GrantAuthorization** dentro del método *ApplicationGrant.doGet()* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationGrant.java>). Se preparará la redirección con la siguiente URL:

`http://localhost:8080/APIOAuth2/GrantAuthorization/FirstStep?response_type=code&client_id=[CLIENT_ID]&redirect_uri=http://localhost:8080/ApplicationOAuth2/ResponseAuthorization&scope=read`

2. El método *ListenerGrant.doGet()* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ListenerGrant.java>) será el encargada de analizar la respuesta y solo se analizará si el **response_type** es del tipo **code**. Se comprobará que la aplicación está dada de alta en la API a través del método *DAOSearchApplication.existApp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOSearchApplication.java>).

- a) En caso de que sí esté dada de alta, se redireccionará a la página:

`http://localhost:8080/APIOAuth2/login.jsp?client_id=[CLIENT_ID]`

- b) En caso de que no esté dada de alta, se redireccionará a la siguiente página que indicará que no se dispone de permisos para confiar en la aplicación:

`http://localhost:8080/APIOAuth2/loginOff.html`

3. La página *login.jsp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/webapp/login.jsp>) recogerá el nombre de usuario y contraseña y mediante una función Ajax en el archivo *login.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/webapp/js/login.js>)

com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/webapp/js/login.js) enviará los datos a través de un mensaje [POST] a la URL:

http://localhost:8080/APIOAuth2/ProcessGrant

a) En el método *ProcessGrant.doPost()* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ProcessGrant.java>) se comprobará si el usuario existe en la API a través del método *DAOLogin.existsUser* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOLogin.java>).

1) En caso de que exista se volverá a la función Ajax del archivo *login.js* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/webapp/js/login.js>) y se llamará a la función *callProcess* que redireccionará mediante el método [GET] a:

http://localhost:8080/APIOAuth2/ProcessGrant

b) En el método *ProcessGrant.doGet()* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ProcessGrant.java>):

- 1) Se generará el **AUTHORIZATION_CODE** a través de la clase *RandomStringGenerator* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/RandomStringGenerator.java>).
- 2) Se comprueba que existe el **CLIENT_ID** a través del método *DAOSetAuthorizationCode.existsApp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOSetAuthorizationCode.java>) y si existe se añadirá la dupla **CLIENT_ID / AUTHORIZATION_CODE** a la BBDD de la API a través del método *DAOSetAuthorizationCode.setApplication* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOSetAuthorizationCode.java>).
- 3) Se realizará una redirección a la aplicación para facilitarle el **AUTHORIZA-**

TION_CODE, a través de un mensaje [GET] a la URL:

`http://localhost:8080/ApplicationOAuth2/ResponseAuthorization?code=[AUTHORIZATION_CODE] & client_id=[CLIENT_ID]`

4. El método *ResponseAuthorization.doGet()* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ResponseAuthorization.java>) en el lado de la aplicación, procesará la respuesta:

- a) Se comprobará que el **CLIENT_ID** existe en la BBDD de la aplicación mediante el método *DAOSearch.existClient* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAOSearch.java>).
- b) Se solicitará el **CLIENT_SECRET** mediante el método *DAOSearch.returnClientSecret* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAOSearch.java>).
- c) Se realizará una redirección mediante un mensaje [GET] a la API con la siguiente URL:

`http://localhost:8080/APIOAuth2/GrantAuthorization/SecondStep?client_id=[CLIENT_ID] & client_secret=[CLIENT_SECRET] & code=[AUTHORIZATION_CODE] & redirect_uri=http://localhost:8080/ApplicationOAuth/GrantAuthorization`

5. El método *ListenerGrant.doGet()* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ListenerGrant.java>) en el lado de la API, procesará la respuesta:

- a) Se comprueba a través del método *DAOSearch.existApp* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOSearch.java>) en una primera instancia si

se tiene almacenado alguna aplicación que disponga de un **CLIENT_ID** asociado al **AUTHORIZATION_CODE** que nos están remitiendo y si eso se cumple, en una segunda instancia se comprobará si existe un **CLIENT_ID** asociado al **CLIENT_SECRET** que nos están facilitando.

- 1) En caso de que se cumpla a través del método *DAOInsert.setCorrectGrant* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/dao/DAOInsert.java>) se almacenará toda la información sobre dicha concesión de autorización.

- b) Mediante el método *ListenerGrant.responsePost()* (<https://github.com/Xevit/eclipse-workspace/blob/master/APIOAuth2/src/main/java/com/ptellos/api/ListenerGrant.java>) se creará un JSON que albergue toda la información para la finalización del proceso de concesión de autorización. Dicho JSON contendrá los siguientes datos:

request :

```
{'access_token': '[ACCESS_TOKEN]',
'token_type': 'bearer',
'expires_in': '[EXPIRES_IN]',
'refresh_token': '[REFRESH_TOKEN]',
'scope': 'read'}
```

- c) Se realizará mediante un método [POST] la siguiente llamada:

<http://localhost:8080/ApplicationOAuth/GrantAuthorization>

6. El método *ApplicationGrant.doPost()* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/application/ApplicationGrant.java>) en el lado de la aplicación, procesará la respuesta:

- a) Se extraerá la información contenida en el JSON y mediante el método *DAOValueApplicationRequest.setApplicationCredentials* (<https://github.com/Xevit/eclipse-workspace/blob/master/ApplicationOAuth2/src/main/java/com/ptellos/dao/DAOValueApplicationRequest.java>) se inicializarán dichos parámetros en la BBDD.

Capítulo 6

Resultados

El resultado final que se quería conseguir era la realización completa de un proceso de autorización utilizando el protocolo OAuth 2.0. Para ello los puntos esenciales que tenían que cumplirse eran los siguientes:

1. Realización de un proceso de autenticación de usuarios en la aplicación principal. Para esto se requería que los datos del usuario estuvieran almacenados en una BBDD a la cuál se tenía que acceder y comprobar si el nombre de usuario y contraseña estaban dados de alta como dupla. Se realiza la gestión tanto de los accesos correctos como de aquellos que resultan erróneos.
2. Realización de un proceso de alta y baja de aplicaciones en la API. Para este proceso resultaba imprescindible que dicha gestión estuviera llevada a cabo únicamente por el usuario administrador. Dentro de las competencias de las que dispondría el administrador, estaría las de realizar la petición de alta de la aplicación en la API y la de enviar la solicitud de baja de la lista de confianza de la API de la aplicación.
3. Realización de un proceso completo de autorización de la aplicación principal en la API, siguiendo para ello el estándar RFC 6749 [7]. El proceso completo tenía que basarse íntegramente en la concesión por código de autorización. El proceso ha consistido en conseguir un token mediante el cuál se accedía a un determinado grado de autorización para la consulta de recursos.

Aunque podemos concluir que el objeto principal de este proyecto se ha conseguido al

100 %, la creación de un proceso mediante el cual se pudiera ver en acción el uso del token de autorización hubiera sido, visualmente más atractivo.

Capítulo 7

Conclusiones

En los capítulos anteriores se ha presentado el estudio de la documentación y la motivación de realizar este proyecto. También se ha desarrollado una implementación de dicho protocolo (RFC 6749) [7] y analizado el comportamiento bajo diversas circunstancias.

7.1. Conclusiones

Tras la realización de las pruebas del correcto funcionamiento de la implementación podemos concluir que la solución desarrollada cubre las expectativas alcanzadas sobre la motivación inicial.

Se ha creado una web que simula el protocolo OAuth 2.0 y se ha realizado de la forma más intuitiva y completa que se ha podido idear.

El desarrollo que se ha seguido es el mismo que se utiliza en las empresas, utilizando patrones de nombres, activando archivos de log para su posterior estudio y facilidad en el análisis de la pila de código, utilizando un servidor de aplicaciones de código abierto (WildFly) y un sistema de automatización de construcción de código abierto (Gradle).

El estudio y desarrollo de este proyecto me han aportado un gran abanico de conocimientos del que antes no disponía, ya que la implementación no se ha realizado utilizando programas conocidos o con los que antes había trabajado y me sentía cómodo, si no que he preferido ir por la línea del si no usando herramientas nuevas que necesitaban de un estudio desde cero del producto. En lugar de Tomcat se ha usado WildFly, en lugar de Maven se ha usado Gradle y en lugar de Oracle he utilizado PostgreSQL

7.2. Trabajos futuros

Como trabajo futuro propondría dos líneas de desarrollo:

1. Por una parte y tal como se indicaba en la introducción, la autorización no deja de ser una de las dos ramas que deberían tenerse en cuenta para disponer de un mecanismo completo de seguridad. La otra rama, la de autenticación, se trata de forma simplificada en este proyecto. Por lo que un trabajo futuro podría ser:
 - a) La realización de un mecanismo de autenticación de usuarios. Dicho mecanismo podría basarse en Infraestructura de Clave Pública y combinarlo con otro mecanismo como puede ser el mecanismo OTP (One Time Password) que añade un nivel de seguridad adicional al solicitar una clave que se envía al teléfono o correo electrónico que tenga asociado el usuario.
 - b) La unión de este proyecto con la API que pueda proporcionar el OpenID Foundation.
2. Una segunda línea de desarrollo podría ser la de hacer más visual este proyecto pudiendo utilizar el ACCESS_TOKEN que se termina obteniendo para solicitar la información que se requiera que tenga disponible el usuario en la Web final de la API. El grado de información al que se podría acceder variaría teniendo en cuenta el 'scope' que se haya proporcionado.

Glosario

API Application Programming Interface. V, VII, 1, 2, 7, 8, 13, 17–20

BBDD Base de Datos. 20, 33, 35–38, 40–42, 44–46, 48–51

CSS Cascading Style Sheets. V

DSL Domain Specific Language. 12

HTML HyperText Markup Language. V

HTTP Hypertext Transfer Protocol. 6

JSON JavaScript Object Notation. 46, 50

MSNBC Canal de noticias estadounidense. 14

NASA National Aeronautics and Space Administration. 14

OASIS Open Architecture Scientific Information System. 5

OTP One Time Password. 54

PKI Public Key Infrastructure. 4

REST Representational State Transfer. 5, 15, 35, 37

RFC Request for Comments. V, VII, 6, 8, 20, 23, 51, 53

RRSS Redes Sociales. 25

TFG Trabajo Fin de Grado. 4

TI Tecnologías de la Información. 3

XML eXtensible Markup Language. 4, 5, 13, 14

Bibliografía

- [1] A. Anderson and H. Lockhart. Saml 2.0 profile of xacml. *OASIS, September*, 51(1.4), 2004.
- [2] M. Anicas. An introduction to oauth 2. *URL: <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>*, 23, 2014.
- [3] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001.
- [4] R. N. Chang, H. Franke, and C.-F. E. Wu. System and method on generating multi-dimensional trace files and visualizing them using multiple gantt charts, Oct. 31 2006. US Patent 7,131,113.
- [5] M. Eernisse. *Build Your Own AJAX Web Applications*. SitePoint Pty Ltd, 2006.
- [6] D. Hardt. The oauth 2.0 authorization framework. 2012.
- [7] D. Hardt. Rfc 6749: The oauth 2.0 authorization framework. *Internet Engineering Task Force (IETF)*, 10, 2012.
- [8] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. User authentication and authorization in the java/sup tm/platform. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pages 285–290. IEEE, 1999.
- [9] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. Openid connect core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 2014.