Proyecto Fin de Grado
Ingeniería de Telecomunicación

Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle

Autor: Alejandro Monago Ruiz

Tutor: Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado
Ingeniería de Telecomunicación

Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle

Autor:
Alejandro Monago Ruiz

Tutor:
Antonio Jesús Sierra Collado
Profesor titular

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle

Autor: Alejandro Monago Ruiz

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

A mi familia

A mis amigos

A mis profesores

A mis compañeros de trabajo

Resumen

En el trabajo que se expone a continuación se ha desarrollado un servicio web API REST, éste ha sido realizado haciendo uso del lenguaje de programación JAVA, el cual es explotado por diversas tecnologías de gran relevancia en el desarrollo web y que han sido aplicadas a este proyecto.

REST (Representational State Transfer) es el modelo de transferencia de datos en el cual se basa esta implementación. Éste es un conjunto de principios de arquitectura para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc.) y sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

En particular, REST hace uso del protocolo HTTP y del pequeño conjunto de operaciones definidas para el mismo (POST, GET, PUT y DELETE) a través de una sintaxis universal como son las URIs. Estas premisas son las que hacen de REST una arquitectura que presenta una gran escalabilidad y desacoplamiento para los sistemas que puedan hacer uso de ella.

La estructura de este desarrollo ha sido realizada a través del software para la gestión y construcción de proyectos Maven. Éste se basa en patrones y estándares y trabaja con arquetipos, los cuales son configurables a través de su fichero POM. En este fichero reside la consiguración necesaria para gestionar librerías, compilar, empaquetar o generar la documentación del proyecto. Maven trabaja con las dependencias que son definidas en el POM, descargándolas y almacenándolas en un repositorio común a todos los proyectos que hagan uso del software, de esta manera se evita la duplicación de librerías y se consigue además el árbol de dependencias que necesite cada librería para su correcto funcionamiento.

El modelado del proyecto está estructurado en capas diferenciadas (Fachada, negocio y datos) consiguiendo así desacoplar la funcionalidad de cada una de estas partes. Además, para cada uno de los servicios se han creado interfaces con la definición de sus métodos y sus correspondientes implementaciones por separado.

La lógica interna de la aplicación recae sobre el Framework Spring, el cual es el encargado de escuchar las peticiones que lleguen al servicio web y dar una respuesta. Para ello se necesita tener configurado un servlet principal, llamado normalmente DispatcherServlet, que recibe todas las peticiones HTTP que llegan a la aplicación y distribuye los procesos para llevar cabo el flujo necesario para la generación de la respuesta.

El servlet se apoya en un contenedor que se carga en tiempo de despliegue y que es configurado a través del fichero “application-context”. Este fichero es de tipo XML y en él se recogen en forma de bean todos los recursos de la aplicación. Cuando la aplicación sea desplegada, el framework leerá este fichero e instanciará todos aquellos objetos que así hayan sido definidos para su posterior uso en la aplicación. El framework además permite definir los bean en las propias clases a través de anotaciones, y llevará a cabo la creación y destrucción de las instancias de los objetos en relación a cómo se definan en el contexto.

Para el acceso a los datos de la aplicación se ha añadido el framework de persistencia Hibernate, el cuál implementa la interfaz común JPA, que es una abstracción de JDBC que permite comunicar el modelo de objetos del aplicativo con el modelo de tablas de la base datos. Hibernate actúa de manera transparente al usuario y le aporta escalabilidad, eficiencia y facilidades a la hora de hacer consultas en la base de datos. Se ha usado además Criteria, que es una API creada por Hibernate y que permite construir las querys de base de datos absolutamente orientadas a objetos.

Para almacenar los datos a los cuales accederá el Servicio Web se ha creado una base de datos ORACLE, haciendo uso de la versión 11.2. Oracle es un motor de base de datos relacional, multiplataforma y que puede ejecutarse en multitud de sistemas operativos: Linux, Mac, Windows, etc. Además, el encoding con el que se

ha configurado la base de datos es UTF8, el cual es capaz de representar cualquier carácter Unicode mediante símbolos de longitud variable e incluye la especificación ASCII.

Para la implementación de la seguridad se ha escogido JSON Web Token(JWT), que es un estándar abierto basado en JSON para la generación de tokens de acceso. En la autenticación con token, el usuario se identifica bien con un usuario/contraseña o mediante una única clave y la aplicación web le devuelve una firma codificada que el usuario usará en las cabeceras de cada una de las peticiones HTTP. En el proyecto se usa HS512 como algoritmo de codificación.

Haciendo uso de esta estructura y queriendo ejemplificarla, se ha dado forma a un servicio en línea orientado al ámbito académico, en el cual se definen difirentes recursos para la obtención de datos del alumnado y los centros. El usuario enviará peticiones acompañadas de JSON estructurados de la manera que exige el servicio y recibirá una respuesta inmediata con la información que se solicita también en formato en JSON.

Abstract

In the work described below has developed a web service API REST, this has been done using the programming language JAVA, which is exploited by various technologies of great relevance in web development and have been applied to this project.

REST (Representational State Transfer) is the data transfer model on which this implementation is based. This is a set of architectural principles to describe any interface between systems that directly uses HTTP to obtain data or indicate the execution of operations on the data, in any format (XML, JSON, etc.) and without the additional abstractions of protocols based on message exchange patterns, such as SOAP.

In particular, REST makes use of the HTTP protocol and the small set of operations defined for it (POST, GET, PUT and DELETE) through a universal syntax such as URIs. These premises are what make REST an architecture that presents great scalability and decoupling for the systems that can make use of it.

The structure of this service has been made through the software for the management and construction of Maven projects. It is based on patterns and standards and works with archetypes, which are configurable through its POM file. In this file resides the achievement necessary to manage libraries, compile, package or generate documentation. Maven works with the dependencies that are defined in the POM, downloading them and storing them in a repository common to all the projects that make use of the software, in this way the duplication of libraries is avoided and the dependencies tree that each library needs for its correct operation is also obtained.

The modeling of the project is structured in differentiated layers (Façade, business and data) achieving this way to decouple the functionality of each one of these parts. In addition, for each of the services interfaces have been created with the definition of their methods and their corresponding implementations separately.

The internal logic of the application falls on the Spring Framework, which is responsible for listening to the requests that arrive at the web service and give an answer. In order to do this, it is necessary to have configured a main servlet, normally called DispatcherServlet, which receives all the HTTP requests that arrive to the application and distributes the processes to carry out the necessary flow for the generation of the response.

The servlet is supported by a container that is loaded at deployment time and that is configured through the "application-context" file. This file is of XML type and in it are collected in bean form all the resources of the application. When the application is deployed, the framework will read this file and instantiate all those objects that have been defined for later use in the application. The framework also allows to define the bean in the classes themselves through annotations, and will carry out the creation and destruction of the instances of the objects in relation to how they are defined in the context.

For the access to the application data the HIBERNATE persistence framework has been added, which implements the common JPA interface, which is a JDBC abstraction that allows to communicate the objects model of the application with the tables model of the database. Hibernate acts transparently to the user and provides scalability, efficiency and ease when making queries in the database. Criteria has also been used, which is an API created by Hibernate and which allows the database queries to be built absolutely object-oriented.

To store the data that the web service will access, an ORACLE database has been created, making use of version 11.2. Oracle is a relational, multiplatform database engine that can be run on a multitude of operating systems: Linux, Mac, Windows, etc. In addition, the encoding with which the database has been configured is UTF8, which is capable of representing any Unicode character using symbols of variable length and includes the ASCII specification.

For the implementation of security, JSON Web Token (JWT) has been chosen, which is an open standard based on JSON for the generation of access tokens. In token authentication, the user is identified either with a user/password or by a unique key and the web application returns an encrypted signature that the user will use in the headers of each HTTP request. In the project HS512 is used as an encoding algorithm.

Making use of this structure and wanting to exemplify it, an online service oriented to the academic field has been created, in which different resources are defined for obtaining data from students and schools. The user will send requests accompanied by JSON structured in the way required by the service and will receive an immediate response with the information also requested in JSON format.

Índice

Índice de Tablas

Índice de Figuras

1 OBJETIVOS

"Nada es especialmente difícil si lo dividimos en tareas pequeñas"

- Henry Ford -

Debido a mi experiencia y aprendizaje en el ámbito laboral nace la idea de este proyecto. He desarrollado un servicio web para un cliente y uso concreto, lo que ha generado en mí el deseo de poner en práctica este trabajo, aplicándolo al ámbito académico e implementado nuevas tecnologías como Spring Framework o JSON Web Token.

Bajo esta premisa, me marco los siguientes objetivos:

- Definir el conjunto de recursos de los que va a disponer el Servicio Web, declarando la estructura que debe tener la llamada para consumirlos y la respuesta que va a devolver. Al quererlo enmarcar en la obtención de datos académicos, resulta de gran importancia que entre ellos se incluya:
  - Recursos para la obtención de datos personales del alumnado.
  - Recursos para la obtención de datos académicos del alumnado.
  - Recursos para la obtención de datos en lo referente a Titulaciones y centros.
- Confeccionar un modelo de datos relacional para dar cabida a un conjunto de información que se ajuste a la realidad y que será consumida por el Servicio Web. Para ello será necesario la realización de diversas tablas compuestas por una clave primaria, datos estructurados por tipo y claves foráneas que las relacionen entre ellas.
- Crear una base de datos Oracle, cuyo encoding será UTF8, al igual que la respuesta que se quiere devolver por el Servicio Web. Se deberán crear los scripts que generen las tablas indicadas en el anterior apartado y ejecutarlos sobre la misma. Además, será necesario crear y ejecutar también los scripts de inserción de datos para realizar unas pruebas correctas sobre los recursos.
- Desarrollar un Servicio Web API REST que implemente los recursos definidos en el primero de los puntos y que haga uso del modelo de datos que se ha mencionado en los siguientes. Para ello se marcan los siguientes subobjetivos:
  - La estrucutra de directorios del proyecto será realizada con el software Maven. Para ello de escogerá un arquetipo Spring MVC (Modelo Vista Controlador) que forme el cuerpo del Servicio Web. Además, gracias a este software se podrán incluir y descargar las librerías que se necesiten a través de un correcto uso de su fichero de configuración POM. Otro de los beneficios a explotar por esta tecnología será la fácil compilación del proyecto para el uso y despligue del aplicativo.
  - La lógica funcional del Servicio Web estará soportada por Spring Framework, de manera que será éste el encargado de levantar los recursos del aplicativo en su servlet principal, escuchar las peticiones y

redirigir el flujo de procesos para la obtención de la respuesta. En el contenedor de Spring se configurará la conexión con base de datos y se declará la creación de beans a través de anotaciones. En el caso de los servicios, se declarará una interfaz por cada uno de ellos, en la que se definan los métodos que posee y por otra parte se realizará la implementación de los mismos. Para que estos servicios puedan ser usados en cualquier parte del aplicativo, se inyectarán estos beans a través también de anotaciones sobre las clases que así lo necesiten.

  - El control y acceso a los datos estará delegado al framework Hibernate. Para ello será necesario configurar los objetos POJO que tendrán correspondencia con las diferentes tablas de base de datos a través de anotaciones que caractericen al propio objeto y sus atributos. Éste framewok trabajará de manera transparente al usuario, de esta forma se podrán configurar las consultas haciendo uso de los objetos y sus atributos, y será Hibernate quien comprondrá las querys a ejecutar por Oracle en el modelo relacional definido en base de datos.

  - La seguridad del Servicio Web estará a cargo de JSON Web Token. Se deberá definir un recurso específico para la generación de tokens de acceso. Este recurso comprobará que en la llamada al mismo se incorpore la clave de seguridad necasaria para la obtención del token, si ésta es correcta se devolverá al usario un token de acceso codificado según el modelo de JSON Web Token a través de claims. Una vez se tenga este token de acceso, el usuario deberá incorporarlo a la cabecera de las peticiones HTTP que se realicen a cada uno de los recursos de obtención de datos definidos en el primer punto. En cada uno de los recursos se validará que sea un token válido y que éste no haya expirado según el tiempo establecido en su creación.

- Identificar aquellos errores que puedan darse al hacer uso del servicio, definirlos y controlarlos dentro del Servicio Web, de manera que el consumidor de los recursos pueda identificar el problema.

- Realizar una batería de pruebas completa, en la que se pongan de manifiesto cada uno de los errores declarados en el punto anterior y se valide el comportamiento correcto para cada uno de los recursos del Servicio Web.

A continuación, se muestra el esquema del Servicio Web expuesto para su desarrollo.

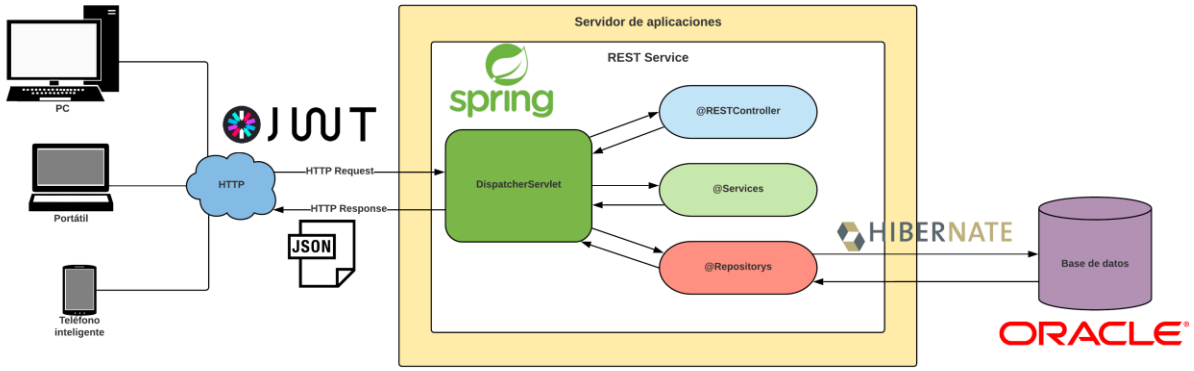

Ilustración 1 – Esquema del Servicio Web que se pretende desarrollar

En este capítulo de se han marcado una serie de objetivos a cumplir para el desarrollo del Servicio Web que constituye la meta final de este proyecto. En el siguiente apartado se va a introducir el trabajo con un breve resumen de la comunicación entre sistemas a lo largo de la historia, valiendo el final de la misma como motivante para la realización de este desarrollo.

2 INTRODUCCIÓN

"Lo último que uno sabe, es por donde empezar"

- Blaise Pascal -

La comunicación entre sistemas es y ha sido algo importante en el mundo de la tecnología, y ésta se ha visto limitada por la naturaleza de los mismos. El poder comunicar dos sistemas en distintas máquinas y desarrollados en tecnologías diferentes ha supuesto un reto. Lo primero sería encontrar una vía de comunicación, lo cual resolvió internet; y lo segundo que ambas máquinas aun pudiendo utilizar diferentes tecnologías interpretasen los mensajes. Para dar solución a esto último nacen los Servicios Web, una tecnología que utilizando protocolos y estándares sea capaz de comunicar dos sistemas desarrollados incluso en distintas plataformas y lenguajes de programación. Los primeros que surgieron fueron los conocidos como RMI, DCOM o CORBA.

Ante la necesidad de dar escalabilidad y fluidez a los sistemas se estudió mucho sobre cómo hacer para que una aplicación estuviese corriendo en máquinas distintas, de manera que se pudiesen comunicar entre ellas y se apoyasen la una en la otra.

Aportando una solución a esto nace RMI (Java Remote Method Invocation), el cuál es un mecanismo ofrecido por Java para invocar un método situado en otra máquina de manera remota. Esta herramienta es únicamente válida si se quieren comunicar dos sistemas que están diseñados con tecnología Java, ya que da un especial protagonismo al paso de objetos por referencia. Este objeto estará accesible a través de la red pudiendo ser consultado éste y sus métodos por otra máquina a través de peticiones en un puerto TCP. Sin embargo, la principal desventaja de RMI es que este no soporta la comunicación con otros sistemas que no estén corriendo aplicaciones desarrolladas en Java.

El modelo de comunicación DCOM (Distributed Component Object Model) fue creado por Microsoft para conseguir también la interoperabilidad del software, es decir, que dos aplicaciones ubicadas en máquinas diferentes pudiesen comunicarse incluso si estas estaban desarrolladas en lenguajes diferentes. En él se establece un mecanismo confiable entre cliente y servidor, sin importar donde estén situados, de manera que el cliente de un objeto DCOM mira ese objeto como si estuviera en su propio espacio de direcciones, este principio es llamado transparencia de ubicación.

DCOM compitió durante muchos años con otra tecnología anterior denominada CORBA (Common Object Request Broker Architecture). Ésta fue creada por el OMG (Object Management Group), una organización sin ánimo de lucro que perseguía acelerar la introducción del software orientado a objetos y reducir los costes de las aplicaciones distribuidas. CORBA define un escenario en el cuál los clientes son objetos que pueden hacer solicitudes a una implementación de un objeto en un servidor. Este cliente y servidor no necesariamente se encontrarían en el mismo espacio de direcciones o en la misma máquina. El mecanismo a través del cual se comunicaban ambos es ORB (Object Request Broke) que utiliza un estándar para convertir las estructuras de

datos en un flujo de bytes, conservando el orden de los bytes entre distintas arquitecturas. Estas tecnologías acabaron en desuso debido a las numerosas incompatibilidades a las que se fueron enfrentando, dificultades a la hora de configurar los nodos de conexión o frente al firewall de los equipos que las ejecutaban. Para dar solución a estos problemas se creó SOAP (Simple Object Access Protocol), la idea era separar completamente las máquinas entre sí, sin conexiones permanentes entre clientes y servidores. Cada producto publicaría aquello que podías hacer a través de direcciones web y comunicación mediante HTTP conjunta con otros estándares.

SOAP fue creado por IBM o Microsoft, entre otros. Estas grandes compañías se interesaron por un protocolo creado por David Winer en 1998 llamado XML-RPC. En este protocolo definía que tanto cliente como servidor podían realizar llamadas a un servidor web mediante HTTP. Los mensajes debían seguir un formato de etiquetas en lenguaje XML.

Las mayores ventajas que posee SOAP son que no está asociado a ningún lenguaje de programación ni protocolo de transporte, a fin de cuenta es simplemente un fichero XML que puede ser interpretado por cualquiera de ellos. No está atado a ninguna infraestructura de objetos distribuidos, utiliza estándares existentes, como lo son HTTP o SMTP; y permite la interoperabilidad entre múltiples entornos.

Después de éste surgiría REST, un servicio web mucho más flexible, ya que permite transmitir cualquier tipo de dato. La comunicación es realizada exclusivamente a través del protocolo HTTP, indicando en el Header Content type el tipo de datos que se transmite, pudiéndose así transmitir no sólo ficheros XML como en el caso de SOAP sino además JSON, binarios, documentos, etc. REST utiliza los métodos definidos en HTTP para comunicarse (GET, POST, PUT, DELETE, PATCH) y también los códigos nativos del protocolo (200,404,403…).

Grandes referentes de las comunicaciones como Facebook, Amazon, Twitter o MEGA utilizan API REST en sus aplicaciones, y es que en la actualidad es raro imaginarse una aplicación web sin una de estas.

Motivado por estos precedentes se lleva a cabo el siguiente proyecto, implementando un API REST utilizando tecnologías muy explotadas en el sector laboral. En el siguiente capítulo se pondrán de manifiesto cada una de ellas, destacando y ejemplificando todas sus características.

3 ESTADO DEL ARTE

"La formulación de un problema, es más importante que su solución"

- Albert Einstein -

Para abordar el proyecto presentado, es fundamental realizar un análisis de la tecnología que se implementa en el mismo.

En este apartado se presenta cada una de ellas, explicando en qué consisten y que ventajas proporcionan, los móviles que las han llevado a ser una pieza importante en el desarrollo de aplicaciones y que las hace distintas al resto de soluciones del mercado.

3.1 Servicios Web

El consorcio W3C (World Wide Web Consortium) define un servicio web como un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable. Cuenta con una interfaz descrita en un formato procesable por un equipo informático (específicamente en WSDL), a través de la que es posible interactuar con el mismo mediante el intercambio de mensajes SOAP, típicamente transmitidos usando serialización XML sobre HTTP conjuntamente con otros estándares web.

3.1.1 Tipos de Servicios Web

Los servicios web que han sido más relevantes pueden ser clasificados de la siguiente manera:

Remote Procedure Calls (RPC, Llamadas a Procedimientos Remotos): están basados en RPC y presentan una interfaz de llamada a procedimientos remotos y funciones distribuidas. Es una comunicación nodo a nodo entre cliente y servidor, donde el cliente solicita que se ejecute cierto procedimiento o función y el servidor envía la respuesta. Las primeras referencias de servicios web estaban basadas en esta versión, sin embargo, las numerosas problemáticas por el acoplamiento entre los sistemas y el nacimiento de nuevas tecnologías, han quedado a éste casi olvidado.

Service Oriented Architecture (SOA, Arquitectura Orientada a Servicios): es una arquitectura de aplicación en la cual todas las funciones están definidas como servicios independientes con interfaces invocables que pueden ser llamados en secuencias bien definidas para formar los procesos de negocio. Al contrario que los Servicios Web basados en RPC, este estilo es débilmente acoplado, lo cual es preferible ya que se centra en los servicios proporcionados por el documento WSDL, más que en los detalles de implementación con los distintos sistemas. El más relevante sería SOAP (Simple Object Access Protocol).

REST (Representational State Transfer): es un conjunto de principios de arquitectura para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc.) y sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

En los siguientes apartados se va a desarrollar el concepto de los servicios web más usados en la actualidad, SOAP y REST.

3.1.2 Características de SOAP

SOAP es un protocolo sobre el que se establece el intercambio de información con el servicio web, es decir, ofrece las directrices básicas a través de las cuales un servicio web se puede construir. Este protocolo está basado en XML y formado por tres partes:

- Envelope: el cual define qué hay en el mensaje y cómo procesarlo.
- Conjunto de reglas de codificación para expresar instancias de tipos de datos.
- La convención para representar llamadas a procedimientos y respuestas.

En la arquitectura de un servicio web que implementa este protocolo también se pueden diferenciar tres partes:

- Proveedor del servicio.
- Solicitante.
- Publicador.

El proveedor de servicios envía al publicador del servicio un fichero WSDL con la definición del servicio web. El solicitante interactúa con el publicador, descubre quién es el proveedor (protocolo WSDL) y contacta con él (protocolo SOAP). El proveedor valida la petición de servicio y envía el dato estructurado en formato XML utilizando el protocolo SOAP. El fichero XML es validado de nuevo por el que pide el servicio utilizando un fichero XSD.

Dentro de esta arquitectura tendría cabida definir dos estándares importantes en esta estructuración:

WSDL (Web Services Description Language): es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios web.

UDDI (Universal Description, Discovery and Integration): protocolo para publicar la información de los servicios web. Permite comprobar qué servicios web están disponibles.

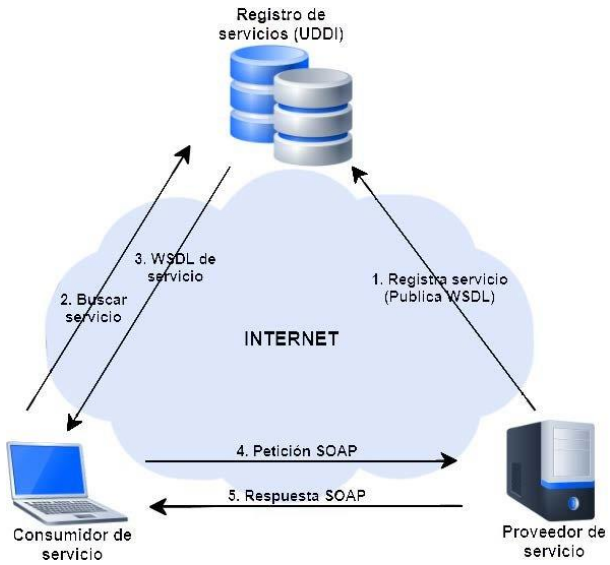

Ilustración 2 – Servicio web basado en SOAP

3.1.3 Características de REST

REST (Representational State Transfer) es un estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la Web. El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP.

REST no es más que una colección de recursos definidos y diseccionados. El término a menudo es utilizado para describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional como hace SOAP. Cabe destacar que es posible diseñar un sistema de acuerdo con la arquitectura

propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en cambio seguir un modelo RPC.

REST es un estilo de arquitectura basado en estándares como son HTTP, URL, la representación de los recursos: XML/HTML/GIF/JPEG/etc. y los tipos MIME: text/xml, text/html, …

La motivación de REST es la de capturar las características de la Web que la han hecho tan exitosa.

REST se ha centrado en explotar el éxito de la Web, que no es más que el uso de formatos de mensaje extensibles, estándares y un esquema de direccionamiento global.

En particular, el concepto central de la Web es un espacio de URIs unificado. Las URIs identifican recursos, los cuales son objetos conceptuales. La representación de tales objetos se distribuye por medio de mensajes a través de la Web. Este sistema es extremadamente desacoplado.

Las características principales de un modelo REST serían las siguientes:

- **Escalabilidad**. La variedad de sistemas y de clientes crece continuamente, pero cualquiera de ellos puede acceder a través de la Web. Gracias al protocolo HTTP, pueden interactuar con cualquier servidor HTTP sin ninguna configuración especial.
- **Independencia**. Los clientes y servidores pueden tener puestas en funcionamiento complejas. Diseñar un protocolo que permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs.
- **Compatibilidad**. En ocasiones existen componentes intermedios que dificultan la comunicación entre sistemas, como pueden ser los firewalls. Las organizaciones protegen sus redes mediante firewalls y cierran casi todos los puertos TCP salvo el 80, el que usan los navegadores web. REST al utilizar HTTP sobre Transmission Control Protocol (TCP) en el puerto de red 80 no resulta bloqueado. Es importante señalar que los servicios web se pueden utilizar sobre cualquier protocolo, sin embargo, TCP es el más común.
- **Identificación de recursos**. REST utiliza una sintaxis universal como es el uso de URIs. HTTP es un protocolo centrado en URIs, donde los recursos son los objetos lógicos a los que se le envían mensajes.
- **Protocolo cliente/servidor sin estado**. Cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.
- **Operaciones bien definidas**. HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

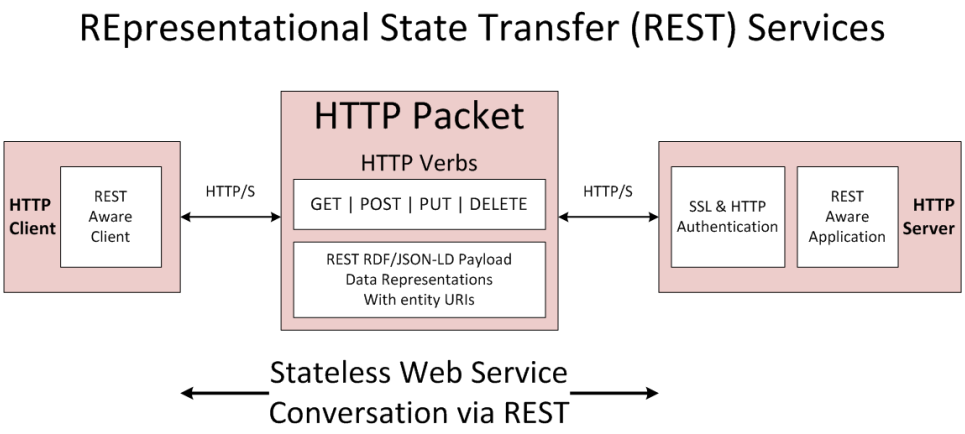

Ilustración 3 – Servicio web basado en REST

3.1.4 Comparativa entre REST y SOAP

Es inevitable entrar a comparar los dos tipos de comunicación entre sistemas más usados en la actualidad. De manera general se puede resaltar de SOAP que es fuertemente acoplado, pudiendo ser testado y depurado antes de poner en marcha la aplicación. Mientras que de REST se puede declarar que su potencial recae en su escalabilidad en todo tipo de sistemas y su escaso consumo de recursos debido al limitado número de operaciones y al esquema de direccionamiento unificado. En la siguiente tabla se muestran de manera detallada algunas de sus características más importantes.

REST	SOAP
• Las operaciones se definen en los mensajes. • Una dirección única para cada instancia del proceso. • Cada objeto soporta las operaciones estándares definidas. • Componentes débilmente acoplados. • Bajo consumo de recursos. • Las instancias del proceso son creadas explícitamente. • El cliente no necesita información de enrutamiento a partir de la URI inicial. • Los clientes pueden tener una interfaz "listener" (escuchadora) genérica para las notificaciones. • Generalmente fácil de construir y adoptar.	• Las operaciones son definidas como puertos WSDL. • Dirección única para todas las operaciones. • Múltiples instancias del proceso comparten la misma operación. • Componentes fuertemente acoplados. • Fácil (generalmente) de utilizar. • La depuración es posible. • Las operaciones complejas pueden ser escondidas detrás de una fachada. • Envolver APIs existentes es sencillo • Incrementa la privacidad. • Herramientas de desarrollo.

Tabla 1 – Características REST y SOAP

Para profundizar más en sus aspectos tecnológicos y buscar diferencias entre ambos, se expone la siguiente tabla.

	REST	SOAP
Tecnología	Interacción dirigida por el usuario por medio de formularios.	Flujo de eventos orquestados.
	Pocas operaciones con muchos recursos	Muchas operaciones con pocos recursos.
	Mecanismo consistente de nombrado de recursos (URI).	Falta de un mecanismo de nombrado.
	Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia.	Se centra en el diseño de aplicaciones distribuidas.
Protocolo	Síncrono.	Síncrono y Asíncrono.
	XML auto descriptivo.	Tipado fuerte, XML Schema.

	HTTP.	Independiente del transporte.
	HTTP es un protocolo de aplicación.	HTTP es un protocolo de transporte.
Descripción del servicio	Confía en documentos orientados al usuario que define las direcciones de petición y las respuestas.	WSDL.
	Interactuar con el servicio supone horas de testeado y depuración de URIs.	Se pueden construir automáticamente stubs (clientes) por medio del WSDL.
	No es necesario el tipado fuerte, si ambos lados están de acuerdo con el contenido.	Tipado fuerte.
	WADL propuesto en noviembre de 2006.	WSDL 2.0.
Gestión del estado	El servidor no tiene estado (stateless).	El servidor puede mantener el estado de la conversación.
	Los recursos contienen datos y enlaces representando transiciones a estados válidos.	Los mensajes solo contienen datos.
	Los clientes mantienen el estado siguiendo los enlaces.	Los clientes mantienen el estado suponiendo el estado del servicio.
	Técnicas para añadir sesiones: Cookies	Técnicas para añadir sesiones: Cabecera de sesión (no estándar)
Seguridad	HTTPS.	WS-Security.
	Comunicación punto a punto segura.	Comunicación origen a destino segura.
Metodología de diseño	Identificar recursos a ser expuestos como servicios.	Listar las operaciones del servicio en el documento WSDL.
	Definir URLs para direccionarlos.	Definir un modelo de datos para el contenido de los mensajes.
	Distinguir los recursos de solo lectura (GET) de los modificables (POST,PUT,DELETE).	Elegir un protocolo de transporte apropiado y definir las correspondientes políticas QoS, de seguridad y transaccional.
	Implementar e implantar el servidor Web.	Implementar e implantar el contenedor del servicio Web.

Tabla 2 – Diferencias destacables entre REST y SOAP

3.2 Maven

Maven es una herramienta para la gestión y construcción de proyectos java, que se basa en el concepto POM (Proyect Object Model). Con Maven se pueden generar arquetipos, gestionar librerías, compilar, empaquetar, generar documentación…

Antes de la llegada de Maven, un desarrollador dedicaba gran parte de tiempo en comprender la estructura y peculiaridades de un proyecto, analizar que librerías utilizaba el código, donde incluirlas, que dependencias de compilación hacían falta en el mismo. Todo este trabajo de build es reemplazado con la llegada de Maven.

Maven se basa en patrones y estándares y trabaja con arquetipos, los cuales son configurables a través de su fichero protagonista, el pom.xml. Desde el cuál es posible configurar muchos aspectos de nuestro proyecto y del cual se hablará en profundidad más adelante.

3.2.1 Ciclo de vida

Maven define tres ciclos de build con etapas diferenciadas. Cada una de las etapas es una instrucción Maven, la cual se ejecutaría llamándola con *mvn <instruccion>*. Cuando se ejecuta dicha instrucción, Maven irá verificando cada una de las fases hasta llegar a la del comando utilizado.

Ciclo de vida por defecto

– **Validación (validate):** Validar que el proyecto es correcto.

– **Compilación (compile).**

– **Test (test):** Probar el código fuente usando un framework de pruebas unitarias.

– **Empaquetar (package):** Empaquetar el código compilado y transformarlo en algún formato tipo .jar o .war.

– **Pruebas de integración (integration-test):** Procesar y desplegar el código en algún entorno donde se puedan ejecutar las pruebas de integración.

– **Verificar** que el código empaquetado es válido y cumple los criterios de calidad (**verify**).

– **Instalar** el código empaquetado en el repositorio local de Maven, para usarlo como dependencia de otros proyectos (**install**).

– **Desplegar** el código a un entorno (**deploy**).

Ilustración 4 – Ciclo de vida Maven

También existen otros ciclos o metas y aunque pueden ser llamadas cuando se quiera no forman parte del ciclo de vida por defecto. Como son:

Ciclo de limpieza

clean: elimina todos los ficheros generados por construcciones anteriores.

Ilustración 5 – Ciclo de limpieza Maven

Ciclo de documentación

site: genera la página web de documentación del proyecto.

site-deploy: despliega la página de documentación en el servidor indicado.

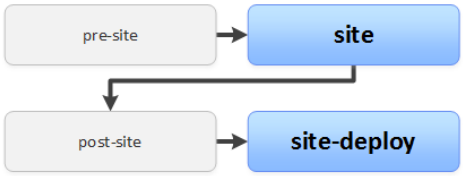

Ilustración 6 – Ciclo de documentación Maven

3.2.2 POM

POM es un modelo de objeto para un proyecto, o como describe la documentación de Maven:

"*El fichero "pom.xml" es el núcleo de configuración de un proyecto Maven. Simplemente es un fichero de configuración, que contiene la mayoría de la información necesaria para construir (build) un proyecto al gusto del desarrollador*"

El pom al ser un fichero xml está compuesto por un conjunto de etiquetas que dan forma al proyecto. La primera de ellas es la etiqueta <**project** >, que engloba todo el pom y en la que se define la arquitectura del xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
```

La primera etiqueta que forma el cuerpo del archivo es la etiqueta <**modelVersion**>. Esta indica la versión del pom, para que funcione con las versiones de Maven 2 y 3 debe tener el valor "4.0.0".

```
<modelVersion>4.0.0</modelVersion>
```

Las siguientes son las llamadas etiquetas básicas, entre las que se puede destacar:

- <**groupId**>: Suele ser el nombre o la web de la organización. Aunque no es necesario que tenga

puntos de separación, se recomienda para actúe como paquete de Java.

- **<artifactId>**: El nombre del artefacto.
- **<name>**: El nombre del proyecto.
- **<version>**: La versión de del proyecto. Por defecto se suele usar "1.0.0". Aunque es posible usar la metodología de versiones que más nos guste.
- **<packaging>**: Con ella se indica cómo se desea que sea empaquetado el proyecto cuando Maven lo construya. Si por ejemplo se usa como valor "jar", se nos creará una biblioteca de Java. Otro ejemplo sería definirlo como "war", que sería un empaquetado web para desplegar en un servidor.

En el proyecto que se presenta se hace uso de los siguientes valores:

```
<groupId>org.es.wsus</groupId>
<artifactId>wsus</artifactId>
<name>wsus</name>
<packaging>war</packaging>
<version>1.0.0</version>
```

Siguiendo con la enumeración de las etiquetas, otra muy usada es **<properties>** que engloba un conjunto de otras etiquetas. Lo que englobe esta etiqueta <properties> será el conjunto de variables globales que podrán ser usadas en otras partes del fichero POM. Un ejemplo de gran utilidad sería formar etiquetas con las versiones de las dependencias que se incorporarán al proyecto, siendo visibles de cara a futuras actualizaciones de las mismas.

```
<properties>
<java-version>1.8</java-version>
<org.springframework-version>4.3.14.RELEASE</org.springframework-version>
<hibernate.version>4.3.6.Final</hibernate.version>
<org.slf4j-version>1.7.5</org.slf4j-version>
<jackson.databind-version>2.5.2</jackson.databind-version>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

En el proyecto se definen etiquetas para las versiones que se utilizarán para Java, el Framework Spring, Hibernate, el log slj4j, Jackson-Databind y la codificación del proyecto.

Con la etiqueta **<repositories>** se indica cuáles serán los repositorios en los que Maven debe buscar las librerías para nuestro proyecto. En este proyecto se ha configurado de la siguiente forma:

```
<repositories>
  <repository>
    <id>central</id>
    <name>Maven Repository Switchboard</name>
    <layout>default</layout>
    <url>http://repo1.maven.org/maven2</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

La etiqueta **<dependencies>** engloba a todas sus etiquetas hijas **<dependency>** en las que se definen los artefactos que quiere que Maven descargue e incorpore a nuestro proyecto, siendo ésta una de las funcionalidades más potentes que nos aporta. A modo de ejemplo:

```
<dependencies>
      <dependency>
```

```
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.2.0.3</version>
      </dependency>

<!-- Jackson -->
      <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.databind-version}</version>
      </dependency>
<dependencies>
```

Con las etiquetas **<gropuId>**, **<artifactId>** y **<version>** es posible identificar la biblioteca y la versión. Además, en la segunda de ellas se ve como se hace uso de una de las variables globales que se han definido anteriormente en la etiqueta **<properties>**.

Además de éstas, se puede añadir la etiqueta **<scope>**, que sirve para indicar a Maven cuando se quiere que utilice una biblioteca u otra. Es decir, si se quiere que sólo la añada al proyecto cuando se compile en pruebas(test), sólo en tiempo de ejecución (runtime)… Si no se especifica ningún valor, por defecto Maven la asigna (compile), compila la dependencia para que esté disponible en todos los classpaths.

Otra etiqueta muy usada es **<exclusions>**, a través de la cual se puede indicar a Maven que librería heredada de las que se han incorporado, no se quiere añadir al proyecto. Esto puede ser muy útil a la hora de evitar conflictos entre librerías y da pie a explicar la potencia de Maven para ahorrar el buscar no sólo la librería que se quiere, sino también las que necesita la propia librería para su correcto funcionamiento.

3.2.3 El repositorio de Maven

Una vez definidas todas las dependencias, Maven las descarga y las guarda todas en un mismo repositorio, generalmente <USER_HOME>/.m2/repository. Aunque se puede cambiar si se desea.

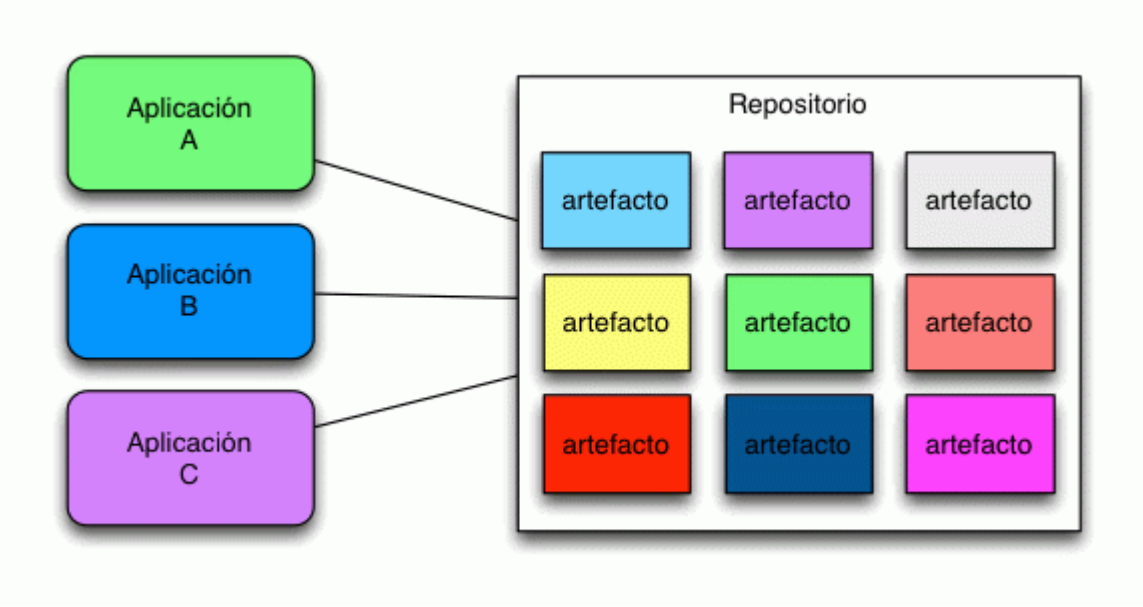

Ilustración 7 – Ejemplificación de repositorio Maven

Como se puede ver en la imagen, este repositorio es compartido por todos los proyectos, evitando jars duplicados y guardando las distintas versiones de cada una de las librerías que son necesarias sin que exista

ningún conflicto con el uso de las mismas en los distintos proyectos que generen.

3.3 Spring Framework

Un Framework son un conjunto de clases y soluciones ya implementadas para su uso con el fin de estandarizar, agilizar y resolver los problemas.

El Framework Spring nos permite desarrollar aplicaciones de manera más eficaz y rápida, ahorrando a su vez muchas líneas de código.

En un modelo de aplicación web habitual, sin hacer uso de algún framework de gestión de las acciones de la web, el cliente envía peticiones HTTP al servidor y éste tendrá configurados distintos servlets para dar una respuesta al cliente. De esta manera, la aplicación estará formada por un número determinado de servlets que desarrollarán las distintas funcionalidades definidas para la misma.

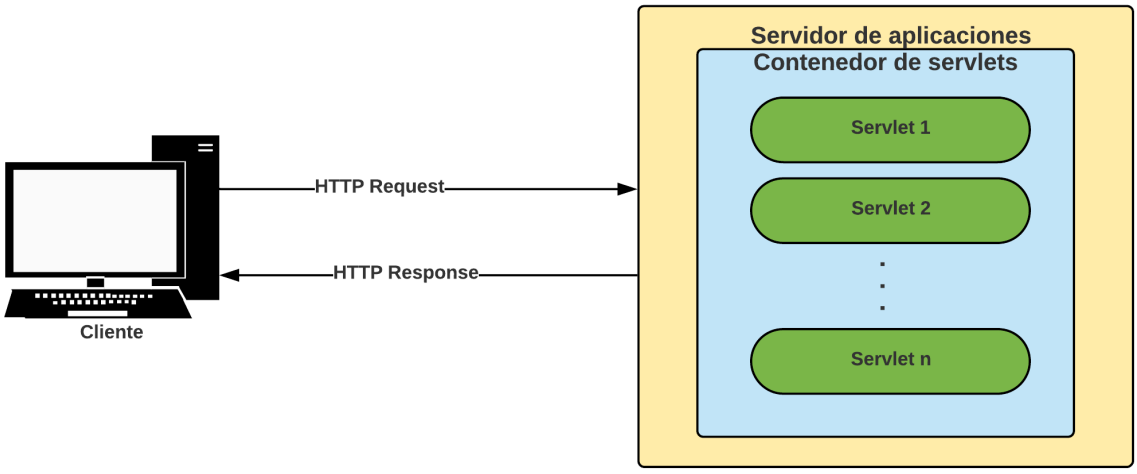

Ilustración 8 – Estructura de aplicación web basada en servlets

El framework de Spring sin embargo solo necesita tener configurado un servlet principal, llamado normalmente DispatcherServlet, que recibe todas las peticiones HTTP que llegan a la aplicación, es decir se recoge toda la funcionalidad en un único servlet.

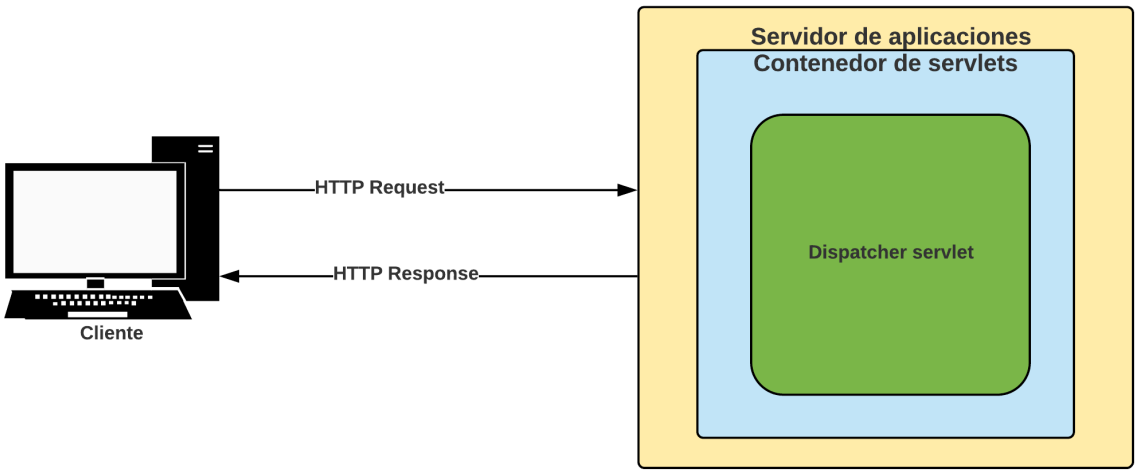

Ilustración 9 – Estructura de aplicación web basada en Spring

Para que esto sea posible el servlet se apoya en un contenedor que se carga en el despliegue de la aplicación y el cual es configurado a través del fichero normalmente nombrado "application-context". Este fichero es de tipo XML y en él se recogen en forma de bean todos los recursos de la aplicación. Cuando la aplicación sea desplegada, el framework leerá este fichero e instanciará todos aquellos objetos que así hayan sido definidos para su posterior uso en la aplicación.

El framework llevará a cabo la creación y destrucción de las instancias de los objetos en relación a cómo se definan en el contexto. De esta manera se encuentran los siguientes ámbitos de creación de bean:

- **Singleton**: Es el ámbito por defecto de Spring, es decir, si no se especifica el tipo en la creación del bean, Spring lo creará con este ámbito. El contenedor de Spring creará una única instancia compartida de la clase designada por este bean, por lo que siempre que se solicite este bean se estará inyectando el mismo objeto.
- **Prototype**: El contenedor de Spring creará una nueva instancia del objeto descrito por el bean cada vez que se le solicite el mismo. En algunos casos puede ser necesario, pero hay que tener en cuenta que no se debe abusar de este tipo puesto que puede causar una pérdida de rendimiento en la aplicación.
- **Request**: El contenedor de Spring creará una nueva instancia del objeto definido por el bean cada vez que reciba un HTTP request.
- **Session**: El contenedor de Spring creará una nueva instancia del objeto definido por el bean para cada una de las sesiones HTTP y entregará esa misma instancia cada vez que reciba una petición dentro de la misma sesión.

3.3.1 Módulos

Spring se apoya en diferentes módulos con sus respectivas funcionalidades. Los cuales presentan el siguiente esquema:

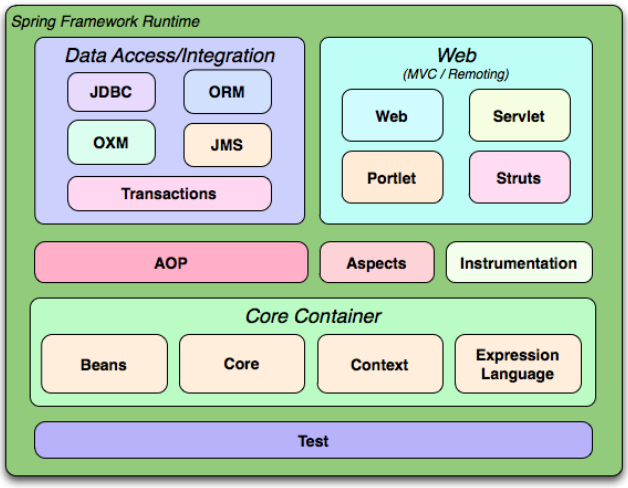

Ilustración 10 – Modulación de Spring Framework

Uno de los más importante es el módulo de core, que permite crear e inyectar beans de cara al diseño de inversión del control (IoC) que proporciona Spring. Este diseño consiste en especificar respuestas y acciones deseadas ante sucesos para que algún tipo de entidad o arquitectura externa lo ejecute, sin necesidad de preocuparse de qué manera se lleva a cabo.

Otro módulo importante de cara a la implementación llevada a cabo en este proyecto ha sido el Web, Spring integra en este módulo el Modelo Vista Controlador(MVC) y la definición interna de servlets.

3.3.2 Creación de contexto y beans

El principal componente del framework Spring es el contenedor de inversión de control (IoC), este contenedor es el encargado de administrar los *beans*, un bean es un objeto Java que es administrado por Spring, es decir, la tarea de instanciar, inicializar y destruir objetos será delegada a este contenedor, el mismo también realiza otras tareas como la inyección de dependencias (DI).

Para poder usar este contenedor se tiene que configurar, la forma tradicional de hacerlo es a través del archivo XML de configuración de Spring, aunque es posible utilizar anotaciones y código Java.

Una vez definido el contexto es posible formar su cuerpo, añadiéndole beans a través de etiquetas y pudiendo éstos ser usados en cualquier parte de la aplicación. El XML se mostraría de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:beans="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
     xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
          http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd" >

<beans:bean id="servicioA" class="es.prueba.ServicioA"></beans:bean>

<beans:bean id="servicioB" class="es.prueba.ServicioB"></beans:bean>

</beans:beans>
```

Como se puede ver, se hace referencia a la clase y se le aporta un id para acceder a ella:

```
ServicioA servicioA = (ServicioA) contexto.getBean("servicioA");

System.out.println(servicioA.mensaje());

ServicioA servicioB = (ServicioB) contexto.getBean("servicioB");

System.out.println(servicioB.mensaje());
```

Sin embargo, se tiene el problema de que, si se quiere declarar un gran número de componentes, el fichero xml se puede hacer excesivamente pesado en su manejo y lectura. Para solucionar esto, Spring da la posibilidad de usar anotaciones en las clases, de manera que se puede obtener el mismo resultado, pero de manera más cómoda. Para ejemplificar el caso anterior con anotaciones se tiene que usar la etiqueta @Service.

```
@Service
public class ServicioA {

     public String mensaje(){
          return "Hola ServicioA";
     }

}
```

Una vez hecho esto, se debe indicar al xml que cargue todas las anotaciones creadas. Para ello se usan dos etiquetas especiales:

- **<context:annotation-config />** : con la que se le informa al framework que se van a usar anotaciones en el código.
- **<context:component-scan />** : se le indica al framework la ruta de paquetes que debe escanear en busca de clases para crear sus beans.

```
<context:annotation-config/>
<context:component-scan base-package="es.prueba" />
```

3.3.3 Spring estereotipos y anotaciones

Spring ha definido multitud de etiquetas y anotaciones con el fin de categorizar componentes dentro del código y asignarles un cometido.

Estereotipos

@Component: es el estereotipo principal, indica que la clase anotada es un componente o bean de Spring.

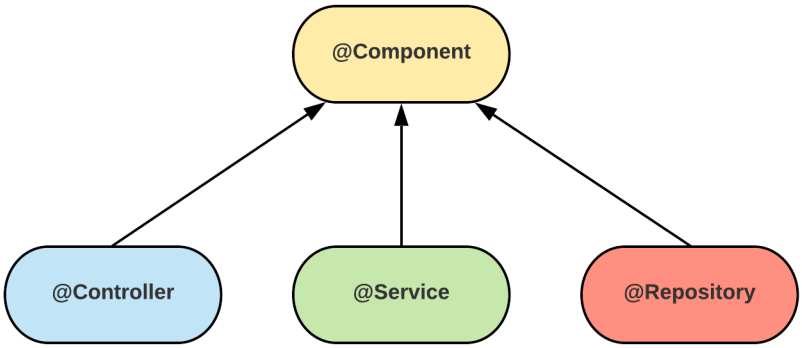

Ilustración 11 – Estereotipos Spring Framework

@Repository: Es el estereotipo que tiene como función dar de alta un bean para que implemente el patrón repositorio, que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. Al marcar el bean con esta anotación, Spring aporta servicios transversales como conversión de tipos de excepciones.

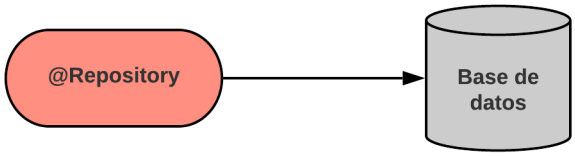

Ilustración 12 – Estereotipo @Repository en Spring Framework

@Service: Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Su tarea fundamental es la de agregador.

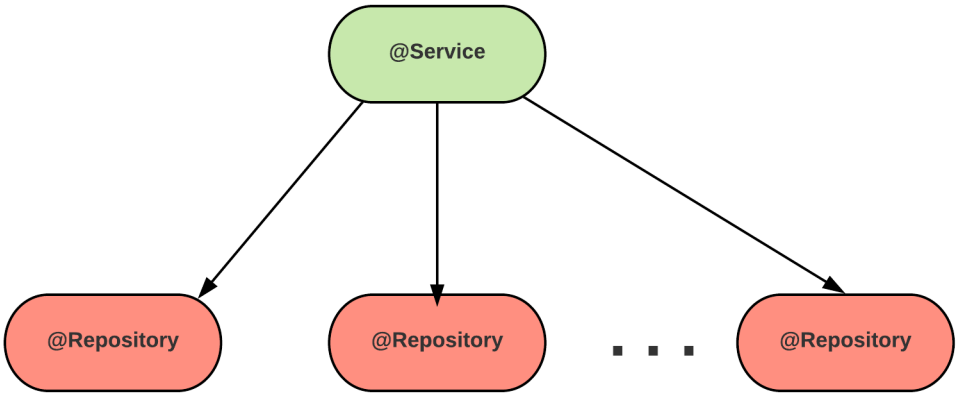

Ilustración 13 - Estereotipo @Service en Spring Framework

@Controller: El último de los estereotipos, es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. Para ello se apoya habitualmente en algún motor de plantillas o librería de etiquetas que facilitan la creación de páginas.

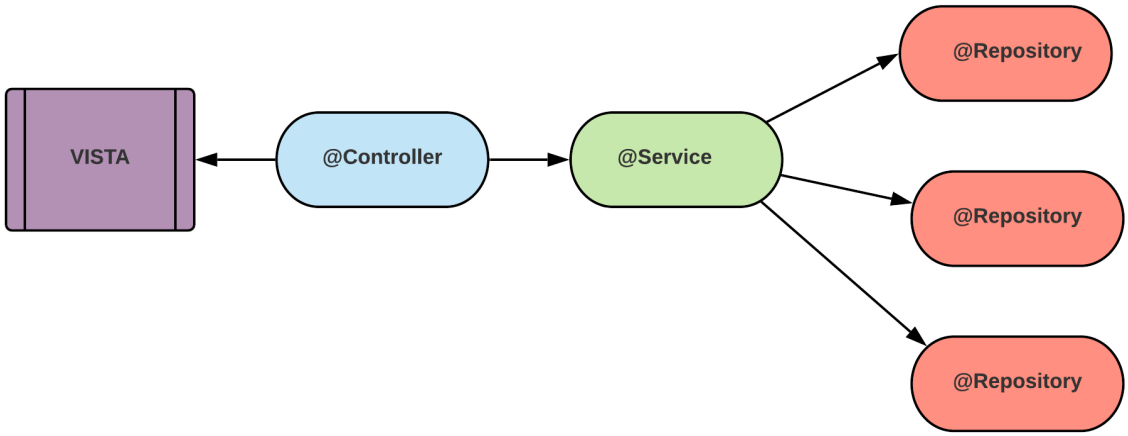

Ilustración 14 - Estereotipo @Controller en Spring Framework

Anotaciones

Una de las anotaciones más importantes de Spring es **@Autowired**, con ella es posible inyectar un componente como puede ser un servicio o un bean en la clase que se desea hacer uso de él. Por ejemplo:

Se declara una interfaz:

```
public interface AlumnosSvc {

     AlumnosJson buscarAlumno(String dni) throws AppException;
     List<AlumnosJson> buscarAlumnos(List<String> listaDnis) throws AppException;
}
```

Se implementan los métodos de la interfaz:

```
@Transactional
@Service
public class AlumnosSvcImpl implements AlumnosSvc {


private final Logger logger = LoggerFactory.getLogger(AlumnosSvcImpl.class);
	/**
	 * Obtener Alumno por dni de BBDD
	 */
	public AlumnosJson buscarAlumno(String dni) throws AppException{
		//IMPLEMENTACIÓN
	}

	/**
	 * Obtener Alumnos por dni de BBDD
	 */
	public List<AlumnosJson> buscarAlumnos(List<String> listaDnis) throws
AppException{
		//IMPLEMENTACIÓN
	}


}
```

Se inyecta la interfaz en otra clase con "@Autowired":

```
@Service("restService")
public class RestServiceImpl implements RestService {


	@Autowired(required = true)
	 private AlumnosSvc alumnosSvc;
```

Una vez inyectado el recurso en la clase, es posible hacer uso de sus propiedades y métodos:

```
alumnos = alumnosSvc.buscarAlumnos(listaDnis);
```

Si dentro de la interfaz en lugar de métodos se tienen declarados servicios mediante la etiqueta @Service, es posible especificar cuál de ellos se quiere inyectar con la etiqueta **@Qualifier**

```
public interface AlumnosSvc {

@Service("obtenerDNI")
public class ObtenerDNI implements AlumnosSvc {
}

@Service("ingresarAlumno")
public class IngresarAlumno implements AlumnosSvc {
}


}
```

Si se quiere inyectar ObtenerDNI:

```
@Service("restService")
public class RestServiceImpl implements RestService {


     @Autowired(required = true)
     @Qualifier("obtenerDNI")
      private AlumnosSvc alumnosSvc;
```

Otra posibilidad sería realizarlo a través de la anotación **@Resource:**

```
@Service("restService")
public class RestServiceImpl implements RestService {

     @Resource("obtenerDNI")
      private AlumnosSvc alumnosSvc;
```

@RestController. Con esta etiqueta Spring facilita mucho el escenario de un API REST.

En primer lugar, se debe configurar un servlet de escucha en el fichero web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/jav
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Ilustración 15 – Configuración de servlet de escucha con Spring Framework

Y añadir esta etiqueta en el fichero que actuará como controlador del servicio:

```
@RestController
public class RestController {

     @RequestMapping(value="uri")
      public ObjetoRespuesta metodoController {
        //…
      return objeto;
      }
}
```

Otra manera de formar un API REST a través de Spring sería con el uso del estereotipo **@Controller**, pero si se usa éste, es necesario añadir a cada uno de los métodos que se definen en el controlador la etiqueta **@ResponseBody**:

```
@Controller
public class RestController {

     @RequestMapping(value="uri")
      @ResponseBody
      public ObjetoRespuesta metodoController {
        //…
      return objeto;
      }
}
```

3.4 JPA (Java Persistence API)

El desarrollo que se realiza en una aplicación de negocio JAVA es orientado a objetos, sin embargo, las bases de datos relacionales almacenan la información en forma de tabla con sus respectivas filas y columnas. Para realizar una correlación entre ambos se necesita de una interfaz que permita guardar la información que se maneja dentro de la aplicación JAVA en el modelo relacional de base de datos.

JPA es una abstracción de JDBC que permite comunicar ambos modelos de manera sencilla, realizando la conversión entre objetos y tablas. Esta conversión se llama ORM (Mapeo Relacional de Objetos) y puede configurarse mediante un xml o anotaciones. Toda esta relación es transparente el desarrollador y para ello se deben crear los objetos de una manera singular, a éstos se les llama entidades.

JPA es una interfaz común y generalmente ésta es implementada por frameworks de persistencia, en este proyecto se ha elegido Hibernate. De esta manera será Hibernate el que realice el trabajo, pero siempre funcionando bajo la API de JPA.

3.4.1 Anotaciones principales

Cuando se habla de una entidad, se hace referencia en cualquier caso a un objeto POJO (Plain Old Java Object), es decir un objeto sencillo que no extiende de otro ni implementa funcionalidad fuera del mismo.

Una entidad se caracteriza por ser una clase de primer nivel, no ser final, proporcionar un constructor e implementar la interfaz java.io.Serializable.

Cualquier entidad JPA presenta numerosas etiquetas que la caracterizan, las necesarias para formar una entidad sencilla serían:

- **@Entity**: informa al proveedor de persistencia que cada instancia de esta clase es una entidad.
- **@Table:** permite configurar el nombre de la tabla que se está mapeando con dicha entidad. Para ello se hace uso del atributo name.
- **@Colum:** acompañará a cada uno de los atributos de la entidad y permitirá configurar el nombre de la columna a la que dicho atributo hace referencia dentro del sistema relacional. Esta etiqueta posee numerosos atributos, donde se puede indicar al framework propiedades que debe tener en cuenta para la columna.
- **@Id:** acompañará aquel atributo que permita diferenciar las distintas entidades de manera que cada una sea única, normalmente acompaña al que se asocie a la clave primaria de la tabla que se esté mapeando.
- **@JoinColumn:** permite configurar aquel atributo que contiene una clave foránea a otra tabla.
- **@OneToMany:** esta etiqueta irá acompañando a la anterior en el caso de que el atributo de la entidad puede referenciar a más de un atributo de la tabla a la que hace referencia. A la hora de obtener estos valores en una consulta, se pueden diferenciar dos tipos:

- **EAGER:** se puede definir como un tipo de lectura temprana, es decir, en la consulta del objeto se obtienen todos los valores de las entidades que están relacionadas con la entidad. Esto es desaconsejable en el caso de asociaciones en las que se puedan ver involucrados muchos objetos, ya que puede suponer un problema de rendimiento para la aplicación.
- **LAZY:** se define como lectura demorada, permite obtener un objeto de base de datos sin los valores de la relación a la que hace referencia el atributo. Es muy útil de cara al rendimiento de la aplicación ya que evita obtener ciertas propiedades que puedan no ser necesarias en la creación del objeto. Lo que se inicializaría es una asociación, de manera que, si se quiere acceder a la información en algún momento, se obtendría.

3.5 Hibernate

Hibernate es un framework de persistencia de software libre que implementa la gestión de la API de persistencia de Java. Este framework actúa de manera transparente al usuario y le aporta escalabilidad, eficiencia y facilidades a la hora de hacer consultas en la base de datos.

En lo referente a la flexibilidad y escalabilidad, hibernate está diseñado para adaptarse a cualquier esquema y modelo de datos, además ofrece la relación inversa, pudiendo así obtener un modelo de tablas relacional a partir de la definición de entidades en Java.

Hibernate presenta un sistema de doble caché, el segundo es totalmente configurable. Estos sistemas se definen como nivel 1 y nivel 2.

El First Level Caché es el que presenta automáticamente Hibernate cuando dentro de una transacción se interactúa con la base de datos, en éste caso se mantienen en memoria los objetos que fueron cargados y si más adelante en el flujo del proceso se vuelven a necesitar van a ser retornados desde la cache, ahorrando accesos sobre la base de datos. Se puede considerar como una caché de corta duración ya que es válido solamente entre el begin y el commit de una transacción, de forma aislada a las demás.

El Second Level Cache permite ser configurado de cara a la mejora del rendimiento. La diferencia fundamental es que éste tipo de caché es válida para todas las transacciones y puede persistir en memoria durante todo el tiempo en que el aplicativo esté online, se puede considerar como una caché global.

Hibernate genera las consultas SQL y libera al desarrollador del manejo más primario de las mimas, además posee un lenguaje de consulta propio denominado HQL (Hibernate Query Language) y de una API para construir consultas conocida como Criteria.

3.5.1 Criteria

Criteria es una API creada por hibernate pensada específicamente para facilitar las consultas a base de datos. La principal ventaja que tiene es que la forma de construir las querys de base de datos es absolutamente orientada a objetos. Esto es de gran utilidad de cara a dar soporte a un formulario de búsqueda que presente multitud de campos, ya que nos evitaría líneas de comprobación para cada uno de los campos, simplificando mucho el código de las consultas.

La mecánica es sencilla, se crea una instancia de Criteria para un objeto en concreto y haciendo uso de métodos de esta instancia se da forma a las restricciones de búsqueda en torno a ese objeto.

```
Criteria crit = sess.createCriteria(Cat.class);
```

Para limitar el número de resultados:

```
crit.setMaxResults(50);
```

Añadir criterios individuales:

```
crit.add( Restrictions.like("name", "Fritz%") )
```

```
crit.add( Restrictions.between("weight", minWeight, maxWeight) )
```

Para disyunciones:

```
crit.add( Restrictions.or(
    Restrictions.eq( "age", new Integer(0) ),
    Restrictions.isNull("age")
  ) )
```

Ordenar resultados:

```
crit.addOrder( Order.asc("name") )
```

Con esto se puede conseguir cualquier consulta básica, pero el abanico de posibilidades es mucho más extenso pudiendo incluso introducir en ellas código sql, crear proyecciones o formar subconsultas. La siguiente ilustración presenta un ejemplo de consulta en el servicio web que se presenta.

```
public List<Titulaciones> obtenerTitulacionesPorCentro(List<String> centros) throws AppException{
    List<Titulaciones> hs = null;

    try{
        Criteria criteria = sessionFactory.getCurrentSession().createCriteria(Titulaciones.class);
        criteria.createAlias("centros", "c");
        Disjunction matchDisjunction = Restrictions.disjunction();
        for (String centro : centros) {
            matchDisjunction.add(Restrictions.ilike("c.centNmNombre", centro, MatchMode.ANYWHERE));
        }
        criteria.add(matchDisjunction);
        hs = (List<Titulaciones>) criteria.list();
    }catch(Exception e){
        logger.error(e.getMessage(), e);
        throw new AppException(AppException.ERROR_10, "Ha ocurrido un error al obtener las titulaciones.",e);
    }

    return hs;
}
```

Ilustración 16 – Ejemplificación de consulta a través de Criteria

3.6 JWT (JSON Web Token)

La autenticación basada en token es una referencia en el desarrollo de aplicaciones web, ya que presenta algunas ventajas respecto a la autenticación más común, en la que se guardan en sesión los datos del usuario.

En la autenticación con token, el usuario se identifica bien con un usuario/contraseña o mediante una única clave y la aplicación web le devuelve una especie de firma cifrada que el usuario usará en las cabeceras de cada una de las peticiones HTTP.

Esta información no se tiene que almacenar en la parte del servidor, como en el caso de la autenticación a través de sesión, sino que se guarda en la parte del cliente y es la aplicación la que comprobará si es válida en cada una de las peticiones.

A su vez, con ello se gana en escalabilidad, pudiendo usar cualquier tecnología (Web, Android, iOS…) para hacer uso de la aplicación, sin diferenciar entre ellas en el servidor.

El token que envía como respuesta la aplicación tiene un tiempo de vida el cuál se debe configurar acorde a lo que interese.

El JWT está formado por tres cadenas separadas por punto:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJ
zdWIiOiIiLCJleHAiOjE1MzU5NjcwMTgsImlhdCI
6MTUzNTg4MDYxOH0.aaavOmuvUE2AMbkzMRef7Gx
6i9IEgcYnY3XNPZXhNZGhj7L3EFjJWgyTB4j4phi
tRY_AGbXo1N4teshYTAn8QA
```


Ilustración 17 – Estructura JWT

- **Header**

Es la primera parte del token, está formada por el tipo de token y el algoritmo de codificación utilizado:

```
{
"typ": "JWT",
 "alg": "HS512"
}
```

Ilustración 18 – Header JWT

- **Payload**

Está compuesto por atributos llamados Claims, existen:

- **iss**: especifica la tarea para la que se va a usar el token.
- **sub**: presenta información del usuario.
- **aud**: indica para que se emite el token. Es útil en caso de que la aplicación tenga varios servicios que se quieren distinguir.
- **iat**: indica la fecha en la que el token fue creado.
- **exp**: indica el tiempo de expiración del token, se calcula a partir del iat.
- **nbf**: indica el tiempo en el que el token no será válido hasta que no transcurra.
- **jti**: identificador único del token. Es utilizado en aplicaciones con diferentes proveedores.

Los más comunes son **sub**, **iat** y **exp**. Para el proyecto en cuestión, en el que no se hace login por usuario:

```
{
  "sub": "",
  "exp": 1535967018,
  "iat": 1535880618
}
```

Ilustración 19 – Payload JWT más común

También admite campos personalizados, por ejemplo:

```
{
  "sub": "",
  "exp": 1535967018,
  "iat": 1535880618,
  "rol": 1,
  "seg": "alta"
}
```

Ilustración 20 – Payload JWT configurable

- **Signature**

La firma es la última de las tres partes, está formada por la información del Header y el Payload codificada en Base64, más una clave secreta que se configura en la propia aplicación.

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) secret base64 encoded
```

Ilustración 21 – Signature JWT

En este capítulo se han puesto en situación cada una de las tecnologías protagonistas del desarrollo a realizar. REST destaca sobre el resto por su gran desacople con los sistemas en los que se implanta, utilizando el protocolo extendido de HTTP y su recogido número de instrucciones, nos permite establecer comunicaciones y transferencias de datos sin importar el formato de los mismos. Mediante Maven, es posible configurar y estructurar todo un proyecto, evadiendo al desarrollador de dificultades a la hora de incorporar librerías o compilar el proyecto. El Framework de Spring aporta toda la lógica funcional dentro de la aplicación, realizando un diseño capaz de especificar respuestas y acciones deseadas ante sucesos para que algún tipo de entidad o arquitectura externa lo ejecute. Hibernate permite al desarrollador realizar consultas orientadas a los objetos y actuará de manera transparente convirtiendo esas consultas a las sentencias necesarias para el cliente de base de datos. Al añadir JSON Web Token conseguimos aportar a nuestro servicio una seguridad que no acumula carga de sesiones en nuestro servidor ni necesita de un modelo de usuarios para el uso del servicio. En el siguiente capítulo se pondrán en práctica cada uno de estas tecnologías y se detallarán los pasos a seguir para el desarrollo del Servicio Web.

4 DESARROLLO DE LA APLICACIÓN

"Las que conducen y arrastran al mundo no son las máquinas, sino las ideas."

- Victor Hugo -

El escenario para el desarrollo de la aplicación está formado por la base de datos y la API con acceso a la misma. En este apartado se va a profundizar en cada una de las partes, explicando sus configuraciones, diseño y características.

4.1 Base de datos

La API REST consultará los datos de la propia base de datos del organismo del que forme parte. A modo de ejemplificar esto, ha sido necesario crear una base de datos e introducirle la información necesaria. Para ello, se ha optado por crear una BBDD Oracle en un entorno local, que es donde se encontrará desplegada la aplicación con el fin de desarrollar las pruebas.

4.1.1 Creación y configuración

Para crear la BBDD Oracle, se ha optado por la versión 11g de la misma.

Para ello, se debe descargar el cliente de Oracle (Oracle Database 11g Express Edition) e instalarlo en el ordenador. Al proceder con la instalación se pedirá un usuario/contraseña con el fin de configurar el cliente.

Una vez instalado, es necesario ejecutar el asistente de Oracle en el apartado Get Started.

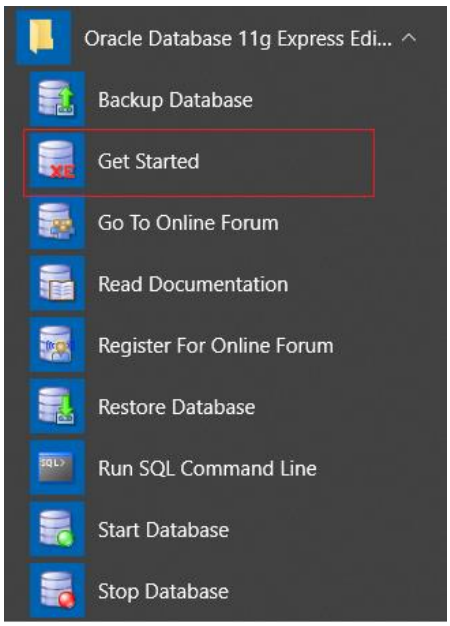

Ilustración 22 – Puesta en marcha base de datos Oracle

Se abrirá la interfaz de Oracle para la configuración de la base de datos, desde la cual se creará el espacio de trabajo para la aplicación.

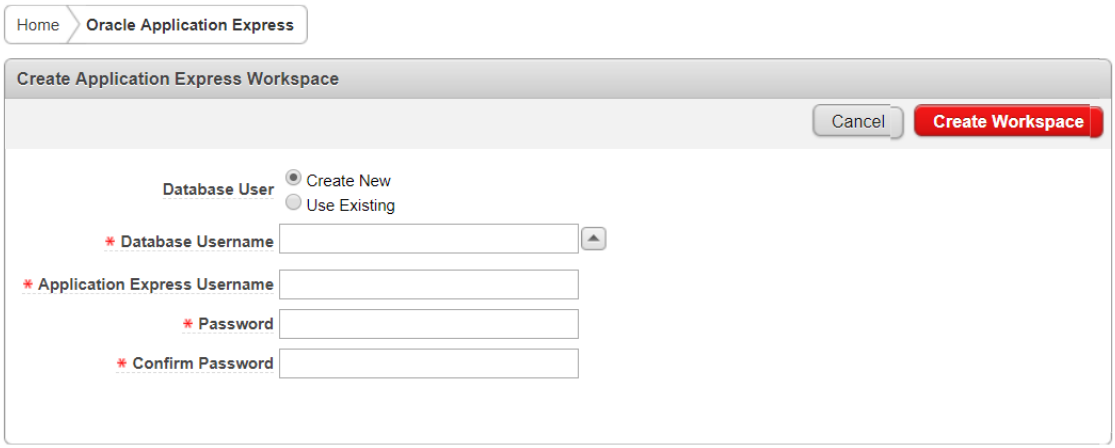

Ilustración 23 – Creación base de datos Oracle

Para la conexión con la BBDD y la realización de acciones sobre la misma se he usado Oracle SQL Developer.

Se creará la nueva conexión según lo definido en el asistente de Oracle.

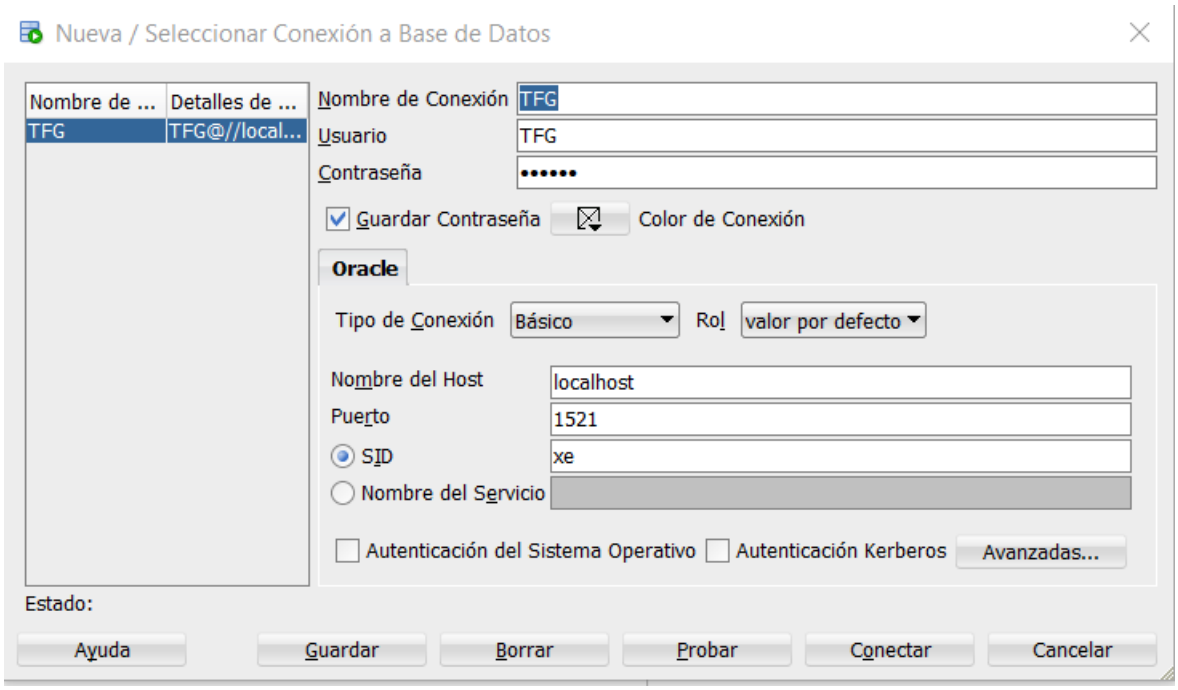

Ilustración 24 – Conexión con la base de datos Oracle

4.1.2 Modelo de datos

- La tabla **WSUS_ALUM_ALUMNOS**, se encargará de guardar los datos personales de cada alumno.

ESPECIFICACIÓN DE WSUS_ALUM_ALUMNOS			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
ALUM_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
ALUM_NM_NOMBRE	VARCHAR2(50)	S	Nombre del alumno
ALUM_LI_APELLIDO1	VARCHAR2(50)	S	Primer apellido del alumno
ALUM_LI_APELLIDO2	VARCHAR2(50)	S	Segundo apellido del alumno
ALUM_LI_DNI	VARCHAR2(25)	S	Documento de identidad del alumno
ALUM_LI_DOMICILIO_FAMILIAR	VARCHAR2(100)	S	Domicilio familiar del alumno
ALUM_LI_DOMICILIO_CURSO	VARCHAR2(100)	S	Domicilio donde reside el alumno durante el curso
ALUM_LI_TELEFONO	VARCHAR2(20)	S	Número de teléfono del alumno
ALUM_LI_CORREO_E	VARCHAR2(50)	S	Correo electrónico del alumno
ALUM_FH_CREACION	DATE	S	Campo auditoría: fecha creación
ALUM_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
ALUM_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
ALUM_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación
Observaciones: • **Secuencia:** se creará la secuencia WSUS_ALUM_SC01 sobre la columna ALUM_CS_CODIGO. • **Trigger:** se creará el trigger WSUS_ALUM_AUDITORIA para rellenar los campos de auditoría.			

Tabla 3 – Especificación de WSUS_ALUM_ALUMNOS

- La tabla **WSUS_CENT_CENTROS**, se encargará de almacenar la información de los centros.

ESPECIFICACIÓN DE WSUS_CENT_CENTROS			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
CENT_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
CENT_NM_NOMBRE	VARCHAR2(250)	S	Nombre del centro
CENT_LI_DIRECCION	VARCHAR2(250)	S	Dirección del centro
CENT_LI_CORREO_E	VARCHAR2(250)	S	Correo electrónico del centro
CENT_LI_TELEFONO	VARCHAR2(100)	S	Teléfono del centro
CENT_LI_WEB	VARCHAR2(250)	S	Página web del centro
CENT_FH_CREACION	DATE	S	Campo auditoría: fecha creación
CENT_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
CENT_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
CENT_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación
Observaciones: • **Secuencia:** se creará la secuencia WSUS_CENT_SC01 sobre la columna CENT_CS_CODIGO. • **Trigger:** se creará el trigger WSUS_CENT_AUDITORIA para rellenar los campos de auditoría.			

Tabla 4 - Especificación de WSUS_CENT_CENTROS

- La tabla **WSUS_TITU_TITULACIONES**, se encargará de almacenar los nombres de las diferentes titulaciones asociadas a los centros.

ESPECIFICACIÓN DE WSUS_TITU_TITULACIONES			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
TITU_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
CENT_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_CENT_CENTROS
TITU_NM_NOMBRE	VARCHAR2(250)	S	Nombre de la titulación
TITU_FH_CREACION	DATE	S	Campo auditoría: fecha creación
TITU_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
TITU_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
TITU_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación
Observaciones: • **Clave ajena:** CENT_CS_CODIGO: clave ajena a la tabla WSUS_CENT_CENTROS. • **Secuencia:** se creará la secuencia WSUS_CENT_SC01 sobre la columna CENT_CS_CODIGO. • **Trigger:** se creará el trigger WSUS_CENT_AUDITORIA para rellenar los campos de auditoría.			

Tabla 5 - Especificación de WSUS_TITU_TITULACIONES

- La tabla **WSUS_ASIG_ASIGNATURAS**, se encargará de almacenar la información de las asignaturas asociadas a las titulaciones.

ESPECIFICACIÓN DE WSUS_ASIG_ASIGNATURAS			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
ASIG_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
TITU_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_TITU_TITULACIONES
ASIG_NM_NOMBRE	VARCHAR2(100)	S	Nombre de la asignatura
ASIG_NU_CREDITOS	NUMBER(3,1)	S	Número de créditos de la asignatura
ASIG_FH_CREACION	DATE	S	Campo auditoría: fecha creación
ASIG_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
ASIG_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
ASIG_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación
Observaciones: • **Clave ajena:** TITU_CS_CODIGO: clave ajena a la tabla WSUS_TITU_TITULACIONES. • **Secuencia:** se creará la secuencia WSUS_ TITU_SC01 sobre la columna TITU_CS_CODIGO. • **Trigger:** se creará el trigger WSUS_ TITU_AUDITORIA para rellenar los campos de auditoría.			

Tabla 6 - Especificación de WSUS_ASIG_ASIGNATURAS

- La tabla **WSUS_CURS_CURSOS**, se encargará de almacenar los diferentes cursos académicos.

ESPECIFICACIÓN DE WSUS_CURS_CURSOS			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
CURS_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
CURS_NM_NOMBRE	VARCHAR2(10)	S	Nombre del curso
CURS_FH_CREACION	DATE	S	Campo auditoría: fecha creación
CURS_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
CURS_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
CURS_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación
Observaciones: • **Secuencia:** se creará la secuencia WSUS_ CURS_SC01 sobre la columna CURS_CS_CODIGO. • **Trigger:** se creará el trigger WSUS_ CURS_AUDITORIA para rellenar los campos de auditoría.			

Tabla 7 – Especificación de WSUS_CURS_CURSOS

- La tabla **WSUS_CALI_CALIFICACIONES**, se encargará de almacenar las calificaciones de cada alumno por curso académico y asignatura.

ESPECIFICACIÓN DE WSUS_CALI_CALIFICACIONES			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
CALI_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
ALUM_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_ALUM_ALUMNOS
ASIG_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_ASIG_ASIGNATURAS
CURS_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_CURS_CURSOS
CALI_NU_CALIFICACION	NUMBER(3,1)	S	Calificación obtenida por el alumno
CURS_FH_CREACION	DATE	S	Campo auditoría: fecha creación
CURS_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
CURS_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
CURS_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación

Observaciones:

- **Clave ajena:** ALUM_CS_CODIGO: clave ajena a la tabla WSUS_ALUM_ALUMNOS.
  ASIG_CS_CODIGO: clave ajena a la tabla WSUS_ASIG_ASIGNATURAS.
  CURS_CS_CODIGO: clave ajena a la tabla WSUS_CURS_CURSOS.
- **Secuencia:** se creará la secuencia WSUS_ CALI_SC01 sobre la columna CALI_CS_CODIGO.
- **Trigger:** se creará el trigger WSUS_ CALI_AUDITORIA para rellenar los campos de auditoría.

Tabla 8 – Especificación de WSUS_CALI_CALIFICACIONES

- La tabla **WSUS_MATR_MATRICULACIONES**, se encargará de almacenar la información relativa a las matriculaciones de los alumnos.

ESPECIFICACIÓN DE WSUS_MATR_MATRICULACIONES			
COLUMNA	**TIPO**	**OBLIG.**	**COMENTARIO DE LA COLUMNA**
MATR_CS_CODIGO	NUMBER(12,0)	S	Clave primaria generada a partir de una secuencia
ALUM_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_ALUM_ALUMNOS
ASIG_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_ASIG_ASIGNATURAS
CURS_CS_CODIGO	NUMBER(12,0)	S	Referencia a la tabla WSUS_CURS_CURSOS
MATR_FH_CREACION	DATE	S	Campo auditoría: fecha creación
MATR_FH_MODIFICACION	DATE	S	Campo auditoría: usuario creación
MATR_LI_CREACION	VARCHAR2(100)	S	Campo auditoría: fecha modificación
MATR_LI_MODIFICACION	VARCHAR2(100)	S	Campo auditoría: usuario modificación

Observaciones:

- **Clave ajena:** ALUM_CS_CODIGO: clave ajena a la tabla WSUS_ALUM_ALUMNOS.
  ASIG_CS_CODIGO: clave ajena a la tabla WSUS_ASIG_ASIGNATURAS.
  CURS_CS_CODIGO: clave ajena a la tabla WSUS_CURS_CURSOS.
- **Secuencia:** se creará la secuencia WSUS_ MATR_SC01 sobre la columna MATR_CS_CODIGO.
- **Trigger:** se creará el trigger WSUS_ MATR_AUDITORIA para rellenar los campos de auditoría.

Tabla 9 – Especificación de WSUS_MATR_MATRICULACIONES

Como resultado de estas especificaciones, se obtiene un modelo de datos como el siguiente:

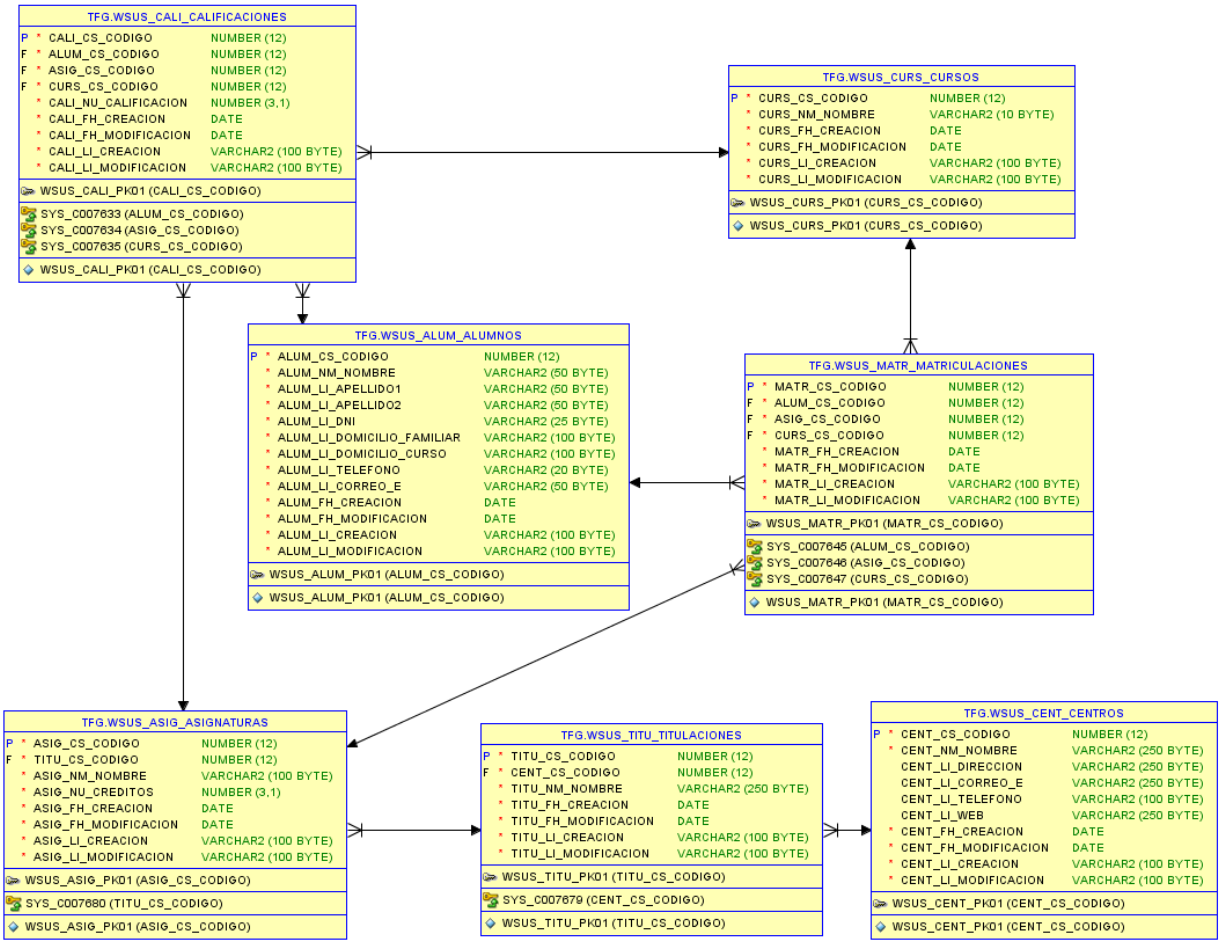

Ilustración 25 – Modelo de datos completo

4.1.3 Trigger de auditoría

En cada uno de los scripts de creación de las tablas comentadas en el anterior apartado, se ha generado un trigger de auditoria. Este se lanzará en cada inserción de datos en la tabla para comprobar que los campos de auditoria, es decir, usuario de creación/modificación y fecha de creación/modificación vengan rellenos. De no ser así, insertará en esos campos el usuario conectado a la base de datos y la fecha en la que se esté realizando, respetando en todo momento si es una primera inserción o una actualización de un registro. La siguiente imagen muestra como ejemplo el trigger para la tabla WSUS_ALUM_ALUMNOS.

4.1.4 Generación de los datos

Para poder hacer uso del servicio web, es necesario que la base de datos contenga información y que esta se acerque lo máximo posible a la realidad. Para ello se ha tratado de configurar estos datos en base a la experiencia académica, tomando como ejemplo la Universidad de Sevilla en lo que algunas tablas se refiere.

4.1.4.1 Generación de datos personales

Para tener una cantidad óptima de datos referente al alumnado, se ha necesitado de una herramienta para la generación de datos personales.

En esta herramienta es posible elegir el tipo de dato que se quiere generar, el número de registros y el tipo de fichero de salida que se desea.

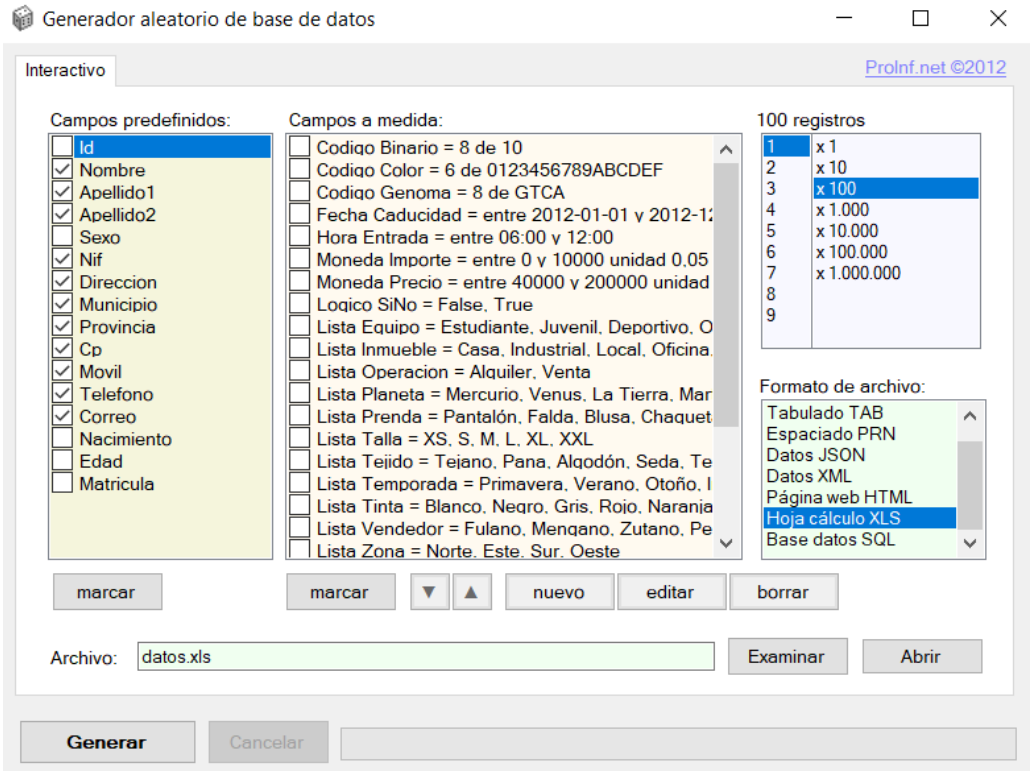

Ilustración 26 – Programa de generación de datos

En este proyecto, se ha optado por la generación en formato de hoja de cálculo, para así poder tratar estos datos de manera que se ajustasen al modelo de datos y al ámbito de la Universidad de Sevilla; además se ha tenido que hacer especial hincapié en que el alumno pudiese disponer de un domicilio familiar y un domicilio de residencia. Algo que es trascendente a la hora de por ejemplo la concesión de alguna ayuda económica en caso de que estos no coincidan y además se encuentren a una distancia superior a la que se estipule en dicha ayuda.

Con esta Excel se han creado los inserts que darían cuerpo a la tabla WSUS_ALUM_ALUMNOS.

4.1.4.2 Generación de datos académicos

Para dar forma a las tablas que recogen la información de los centros y las titulaciones, se han obtenido los datos de la web de la Universidad de Sevilla, incluyéndolos en su totalidad.

Las tablas de matriculaciones y calificaciones son las que más datos deben alojar, además los que más se deben acercar a la realidad de cara a obtener unos resultados lo más verídicos posible. Esto supone poner una especial atención en que las asignaturas de las que el alumno se matricule ronden los 60 créditos que conforman un curso académico, aunque se estén compartiendo de distinto curso. Y que además siga una lógica

de matriculación entre los cursos de manera escalonada. Es por ello, que se ha decidido acotar estas tablas para ejemplificar únicamente los estudios que se realizan en la Escuela Superior de Ingeniería. Lo cual se considera suficiente para mostrar la funcionalidad del servicio web.

Para conformar estos datos, se ha hecho uso de una hoja de cálculo para poder así generar las calificaciones de manera automática. A aquellos alumnos que aprobaban una asignatura, se le ha asignado otra del curso siguiente de un número de créditos similar, con fin de ajustar la suma del total de créditos matriculados en ese curso académico.

Para cada titulación de la escuela se ha seguido un mismo patrón, creando las matriculaciones y calificaciones de 10 alumnos de primer curso, 10 de segundo curso, 10 de tercero y por último 10 de cuarto curso. Así como sus matriculaciones en el curso siguiente a raíz de las calificaciones obtenidas. Teniendo en total un estudio para 320 alumnos en el curso académico de 2017/2018, así como sus matriculaciones en el curso siguiente. Esto proporcionará información suficiente de cara al estudio de una concesión de ayuda.

4.2 Servicio Web

4.2.1 Estructura

El servicio web estará desplegado en el servidor propio del organismo de educación y atenderá las solicitudes consultando en la base de datos del organismo. El esquema será como el que se presenta a continuación.

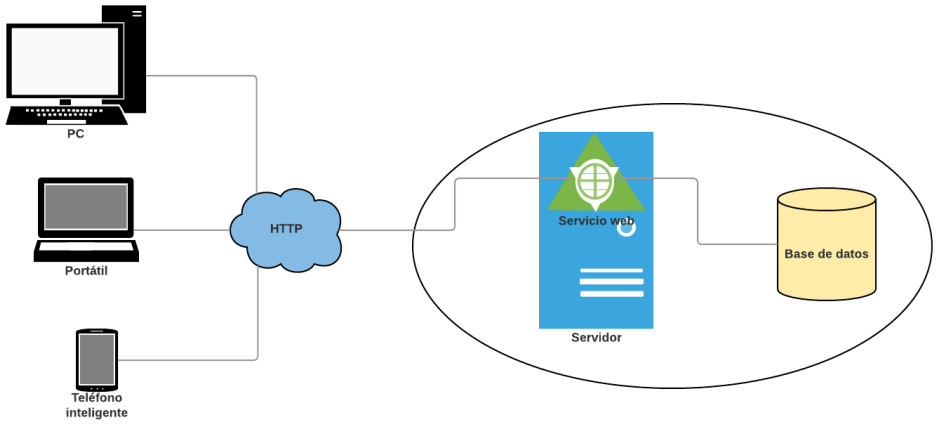

Ilustración 27 – Modelo del servicio

4.2.2 Creación de Proyecto Maven

El proyecto se ha modelado en una arquitectura Maven, con el fin de partir desde una estructura base y de explotar todas las ventajas que esto aporta.

Como punto de partida, al crear el Proyecto se ha elegido el arquetipo base spring-mvc-archetype, desde el cuál es posible desarrollar el resto de paqueterías necesarias.

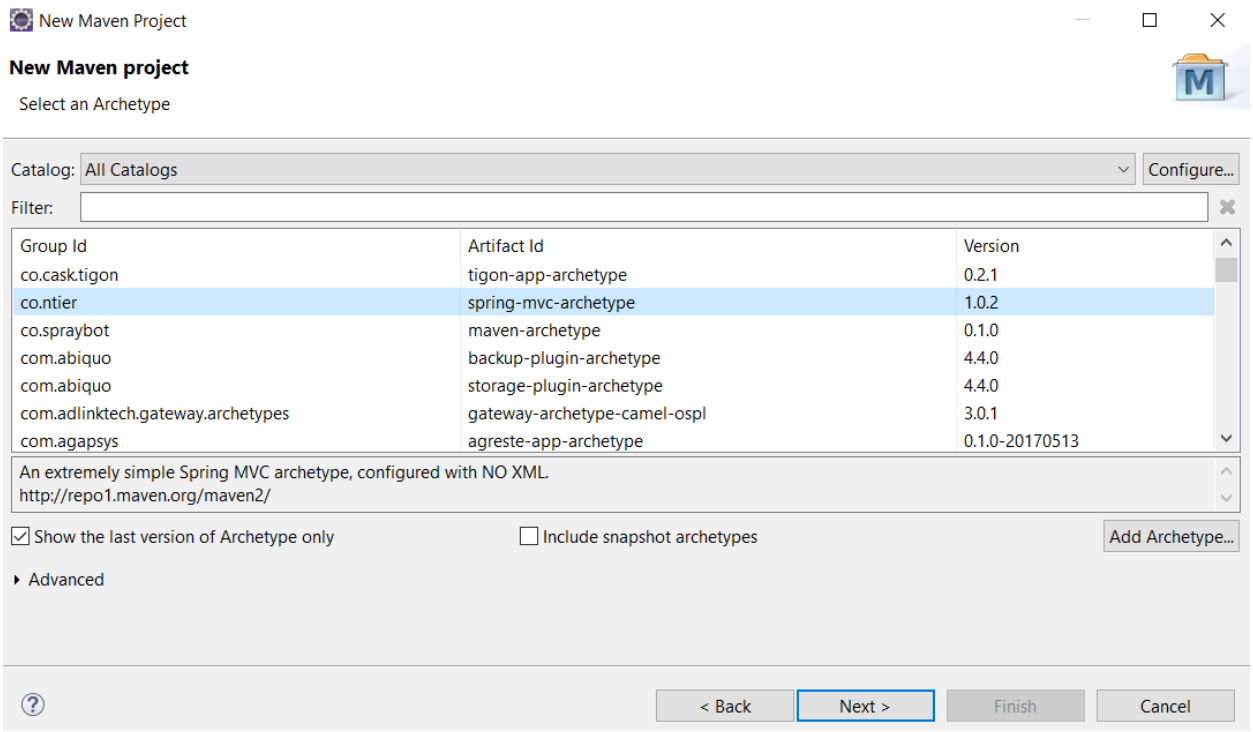

Ilustración 28 – Creación del proyecto a través de arquetipo

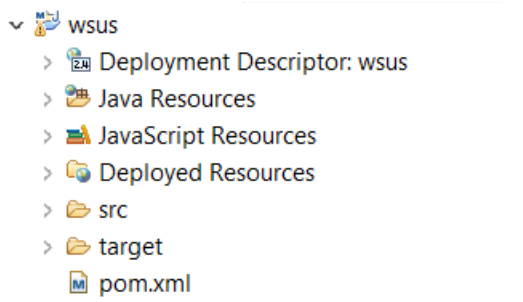

Ilustración 29 – Estructura de directorios del arquetipo spring-mvc-archetype

4.2.3 Edición del fichero POM y creación del fichero settings

En el POM del proyecto que se genera ya se encuentran las dependencias de Spring o javax.servlet, las cuales servirán de apoyo para realizar el servicio web. También incluye una configuración básica para el proyecto, como son el empaquetado de compilación que se va a usar, las versiones de java o spring y repositorios por defecto para las dependencias que se quieran añadir.

Para la realización del proyecto se han llevado a cabo las siguientes modificaciones:

- **Dependencias:**

Se han añadido las siguientes dependencias:

- OJDBC (ojdb6): para la conexión con la BBDD.
- Hibernate (hibernate-core): de cara a trabajar sobre la BBDD.
- JSON (Jackson-databind, geoson-jackson, json): para trabajar con los JSON.
- Logging (slf4j): para la creación de un fichero de log.

- **Propiedades:**

En este apartado se han modificado las versiones de Java y Spring que se van a utilizar a unas más recientes de las que se prestan con el arquetipo.

Además, se han configurado las versiones de Hibernate, Jackson Databind y sl4j que se van a implementar. Así como la configuración del encoding del proyecto.

```
<properties>
    <java-version>1.8</java-version>
    <org.springframework-version>4.3.14.RELEASE</org.springframework-version>
    <hibernate.version>4.3.6.Final</hibernate.version>
    <org.slf4j-version>1.7.5</org.slf4j-version>
    <jackson.databind-version>2.5.2</jackson.databind-version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

Ilustración 30 – Configuración de versiones en el POM

- **Fichero settings.xml:**

Este fichero es aquel en el que se apoya Maven al compilar el proyecto, en él se pueden indicar distintas propiedades que el compilador debe tener en cuenta para construir. En el caso que nos compete, se han configurado los parámetros de conexión con la BBDD mediante las etiquetas necesarias, el resultado sería el siguiente:

```
<!--
    Base de datos y pool de conexiones
-->
<!-- Pool de conexiones con la base de datos. -->
<db.wsus.pool>oracle.jdbc.pool.OracleConnectionPoolDataSource
</db.wsus.pool>
<!-- Driver para la conexión con la base de datos. -->
<db.wsus.driver>oracle.jdbc.driver.OracleDriver</db.wsus.driver>
<!-- Cadena de conexión con la base de datos. -->
<db.wsus.url>jdbc:oracle:thin:@localhost:1521:XE</db.wsus.url>
<!-- Usuario de conexión con la base de datos. -->
<db.wsus.user>TFG</db.wsus.user>
<!-- Contraseña de conexión con la base de datos. -->
<db.wsus.pass>TFG</db.wsus.pass>
<!-- Dialecto hibernate. -->
<db.wsus.dialect>org.hibernate.dialect.Oracle10gDialect</db.wsus.dialect>

<db.wsus.show_sql>true</db.wsus.show_sql>
<db.wsus.format_sql>false</db.wsus.format_sql>
<db.wsus.auth>Container</db.wsus.auth>
<!-- Conexiones máximas inactivas -->
<db.wsus.max_idle>30</db.wsus.max_idle>
<!-- Tiempo de espera (ms)-->
<db.wsus.max_wait>10000</db.wsus.max_wait>
<!-- Número máximo de conexiones activas. -->
<db.wsus.max_active>100</db.wsus.max_active>
```

Ilustración 31 – Configuración de conexión en fichero settings

4.2.4 Creación de servlet de escucha y conexión con BBDD

Para que el servicio web pueda atender a las llamadas será necesario crear un servlet de escucha, como ya se ha expuesto en apartados anteriores.

El siguiente paso será configurar este servlet, para ello se ha creado el fichero servlet-context.xml, el cual se debe asociar al servlet en el fichero web.xml. Este fichero será el "application-context" del proyecto.

En este fichero de configuración se van a crear los beans en los que se apoyará el modelo de datos:

- **dataSource**: se define así la interfaz de acceso a la base de datos. Para ello se hace uso de las variables definidas en el settings.

```
<beans:bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <beans:property name="driverClassName" value="${db.wsus.driver}" />
    <beans:property name="url" value="${db.wsus.url}"/>
    <beans:property name="username" value="${db.wsus.user}" />
    <beans:property name="password" value="${db.wsus.pass}"/>
    <beans:property name="maxActive" value="${db.wsus.max_active}"/>
    <beans:property name="maxWait" value="${db.wsus.max_wait}"/>
    <beans:property name="maxIdle" value="${db.wsus.max_idle}"/>
</beans:bean>
```

Ilustración 32 – Definición de dataSource en el contexto de Spring

- **sessionFactory:** este bean permitirá obtener la sesión para trabajar con Hibernate. En él se especifican como propiedades una referencia al dataSource, la paquetería donde se definen las entidades y parámetros para la configuración de Hibernate.

```
<beans:bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" >
    <beans:property name="dataSource" ref="dataSource"/>
    <beans:property name="packagesToScan">
        <beans:list>
            <beans:value>es.ja.wsus.model</beans:value>
        </beans:list>
    </beans:property>
    <beans:property name="hibernateProperties">
        <beans:props>
            <beans:prop key="hibernate.dialect">${db.wsus.dialect}</beans:prop>
            <beans:prop key="hibernate.show_sql">${db.wsus.show_sql}</beans:prop>
            <beans:prop key="hibernate.format_sql">${db.wsus.format_sql}</beans:prop>
        </beans:props>
    </beans:property>
</beans:bean>
```

Ilustración 33 – Definición de sessionFactory en el contexto de Spring

4.2.5 Creación de entidades a través de Hibernate

Una vez creada la conexión con base de datos, se van a generar las entidades con las que se trabajará. Para ello se ha hecho uso de la extensión de **Hibernate / Jboss Tools**.

Una vez que está instalada la extensión en Eclipse, se debe abrir la perspectiva "Hibernate Perspective". Lo siguiente será editar la configuración con nuestra base de datos desde el apartado "Add Configuration".

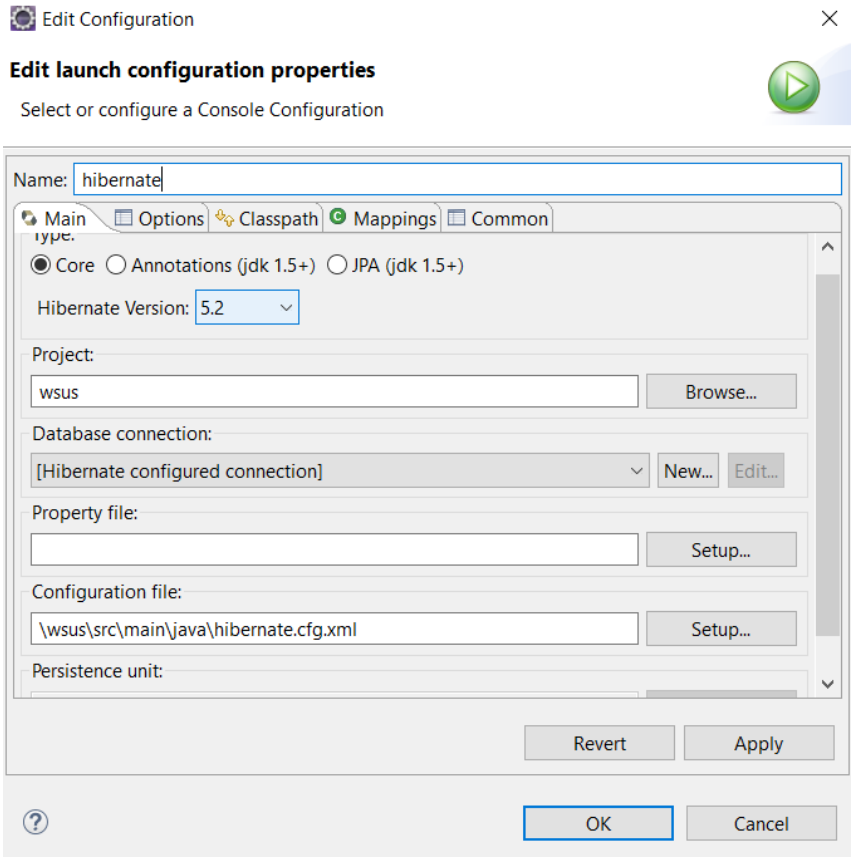

Ilustración 34 – Creación del lanzador de Hibernate

En esta configuración se tienen 3 apartados importantes:

- Project: donde se debe elegir el proyecto en el cual se tiene creada la configuración con base de datos y se quieren crear las entidades.
- Database Connection: donde se indica que se use la configuración hibernate que anteriormente se ha creado.
- Configuration file: que será el archivo que se tendrá que añadir con los parámetros de conexión en caso de no tenerlos ya previamente definidos. En este proyecto sólo se tendría que dar un nombre y una ruta donde generarse y cogerá la configuración de lo que se ha definido en el apartado anterior.

Una vez terminada esta configuración, se tendrá acceso a la base de datos.

Lo siguiente será editar el generador de código, para ello se accede a "Hibernate code generation":

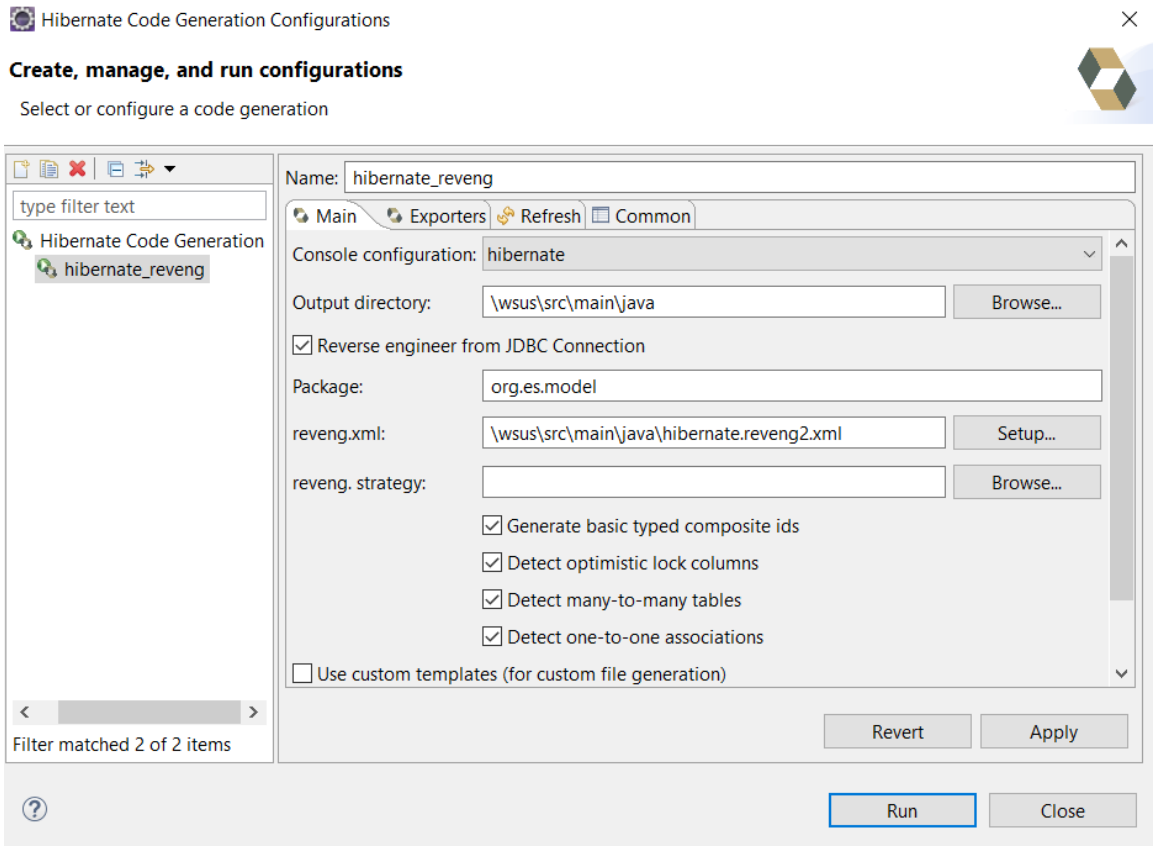

Ilustración 35 – Configuración del lanzador de Hibernate

- Console configuration: se indicará la conexión que se ha creado en el paso anterior.
- Output directory: la ruta en la que se quiere que se creen.
- Package: la paquetería en la que se encontrarán dentro del proyecto java.
- reveng.xml: se debe pulsar en Setup para indicar aquellas tablas con las que se quiere trabajar.

En la pestaña de "Exporters" se indicará el tipo de archivo que se quiere generar, para crear las entidades interesa el tipo Domain code (.java).

Una vez se ejecuta esta configuración se obtienen los mapeos a las tablas creados a través de Hibernate.

4.2.6 Creación del controlador, servicios y daos

De cara a mejorar la escalabilidad del código, se ha realizado un diseño estable y con vistas a futuros cambios, separando la fechada (controlador), la lógica de negocio (servicios) y la capa de acceso de datos (dao).

4.2.6.1 Controlador

A lo largo del proyecto, Spring toma todo el protagonismo, un ejemplo de ello es el controlador de escucha del servicio REST.

Este no es más que una clase java, a la que se le añade la etiqueta @Controller. La cuál es acompañada en los distintos servicios que se declaran de las etiquetas:

@RequestMapping: para definir la url y el tipo de petición.

@ResponseBody: para formar la respuesta de nuestro método.

Cada uno de los métodos que definen los servicios, reciben como parámetros de entrada el objeto request de la petición y los datos que acompañan a la misma. Estos datos se catalogarán de distinta forma según el tipo de la llamada que atienda el servicio, de tal manera que:

Tipo **GET**: se define el parámetro de entrada antecedido por la etiqueta **@RequestParam**.

Tipo **POST**: se define el parámetro de entrada antecedido por la etiqueta **@RequestBody**.

El framework de Spring se encargará de obtener de la petición estos parámetros para que pueden ser usados dentro del método.

En los distintos métodos se verifica que se cumpla con la seguridad JWT, se da forma al objeto que procesará el servicio a través de los parámetros de la petición y se controlan las distintas excepciones. Dándole una salida de errores al usuario a través de una respuesta con el código HTTP indicado y en algunos casos acompañándose de un mensaje informativo.

4.2.6.2 Capa de negocio

En esta parte de la aplicación se van a llevar a cabo todas las operaciones de procesado de la información. Esta capa se divide en distintas clases, de cara a separar la funcionalidad. Así se cuenta con un total de cinco servicios.

El servicio principal será al que llamarán todos y cada uno de los métodos definidos en el controlador. Este es un servicio de paso, el cuál ramificará cada una de las demandas al servicio en concreto que trabaje con los objetos que se demandan.

El resto de servicios están orientados a procesar la información, realizar consultas a la capa dao y formar la respuesta que entregará al servicio principal.

Los servicios están construidos a través de la etiqueta @Service de Spring y formados por una interfaz en la que se recojan aquellos métodos públicos que podrán ser solicitados fuera del mismo y una implementación donde se desarrollan éstos.

Además, en ellos se desarrollan métodos privados para el uso exclusivo de los mismos por otros métodos del servicio.

4.2.6.3 Capa de datos

La capa de datos está formada por aquellas clases que son catalogadas como DAO. Cada una de ellas se caracteriza por ir acompañada por la etiqueta @Repository (“Nombre”), la cual generará un bean para la clase de cara a poder ser solicitada por la capa de negocio y además le aportará servicios transversales para el manejo de los datos y control de excepciones.

En cada una de estas clases se inyecta el bean sessionFactory que ha sido definido en el contexto de Spring y gracias a ello será posible acceder a cada una de las tablas definidas en el modelo de datos.

Además, en todos ellos se hace uso de hibernate a través de Criteria, explotando así las posibilidades que nos da el Framework. En la siguiente imagen se muestra una de estas consultas.

4.2.7 Incorporación de seguridad JWT

Para incorporar la seguridad Json Web Token al servicio web, se va a necesitar de un servicio que sea accesible y se encargue de la generación del token para el usuario. Además de otra implementación para la comprobación del token en cada uno de los servicios de demanda de información.

- **Generación del token**

El servicio encargado de generar el token esperará que el cuerpo de la petición contenga un json formado por una línea *clave: valor*, de la cual intentará extraer el parámetro *clave*. De no contener el json esta clave, devolverá un error indicando que el json es incorrecto.

En el caso de que el json sea correcto y si disponga de esta clave, se extraerá la misma. Lo siguiente será comprobar que esta clave es la definida en la configuración del servicio web y en caso de ser afirmativo, dotar de un tiempo de expiración al token y codificar todo con una clave secreta en HS512.

Este token codificado será el que debe usar el usuario de cara a utilizar los distintos servicios.

- **Comprobación del token**

En cada uno de los recursos de los que dispone el servicio web, lo que se llevará a cabo en primer lugar es extraer la cabecera Authorization de la petición. De ella se descarta la cabecera Bearer para obtener el token y éste se codificará usando la clave secreta del encoding. Al hacer esto se obtienen los claims o privilegios del token.

Si se ha realizado correctamente, se comprobará que esté dentro del tiempo prefijado en la generación del token. Si está en el intervalo de tiempo adecuado, se proseguirá con la lectura del cuerpo de datos de la petición, la obtención y envío de la respuesta.

4.2.8 Ejemplificación de llamada al servicio mediante código java

Con el fin de ejemplificar una llamada el servicio web desde otra aplicación que lo pueda consumir, se ha realizado la implementación de una clase ejecutable que haga uso de uno de sus servicios.

En primer lugar, se ha definido la url de conexión.

```
HttpURLConnection conn = null;
BufferedReader br = null;
String value;
String token = "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIiLCJleHAiOjE1MzU5NjcwMTg

try {
    String uri = "http://localhost:8080/wsus/rest/alumnos";
    String charset = "UTF-8";
    URL url = new URL(uri);
    conn = (HttpURLConnection) url.openConnection();
    // conn.setRequestMethod(AppConstantes.GET);
    conn.setRequestMethod(AppConstantes.POST);
    conn.setRequestProperty(AppConstantes.ACCEPT, AppConstantes.APP_JSON);
    conn.setRequestProperty("Authorization", token);
```

Ilustración 36 – Código Java para la llamada a un servicio (I)

Se declara la uri a la que se realizará la petición HTTP y se abre la conexión e indica el tipo de la misma. A esta conexión se le añade como propiedad que admita formato JSON en sus datos y la cabecera de Autorización. Esta última estará formada por el protocolo Bearer que es el usado por JWT y el token que previamente se haya obtenido.

El siguiente paso será confeccionar el cuerpo de la petición.

```
//Creamos el cuerpo con los datos
conn.setDoOutput(true);
conn.setRequestProperty("Accept-Charset", charset);
conn.setRequestProperty("Content-Type", "application/json;charset=" + charset);
String urlParameters = "{\"dni\": [\"12722114D\",\"43052549E\"]}";
try (OutputStream output = conn.getOutputStream()) {
    output.write(urlParameters.getBytes(charset));
}
```

Ilustración 37 - Código Java para la llamada a un servicio (II)

Se le indica la codificación, que en este caso será UTF-8 y el tipo de contenido que se va a enviar. Posteriormente se introduce el json que recibirá el servicio web.

Por último, se realiza la llamada, se conforma el buffer de lectura, procesa la información recibida convirtiéndola en un objeto json y se muestra el resultado.

```
//Llamamos
if (conn.getResponseCode() != HttpStatus.OK.value()) {
    throw new Exception("Error al obtener el servicio");
}

br = new BufferedReader(new InputStreamReader(conn.getInputStream(), AppConstantes.ENCOD_UTF8));
StringBuilder sb = new StringBuilder();
String aux;

while ((aux = br.readLine()) != null) {
    sb.append(aux);
}

value = sb.toString();
ObjectMapper mapper = new ObjectMapper();
Object json = mapper.readValue(value, Object.class);
String imprimir = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(json);

System.out.println(imprimir);
```

Ilustración 38 - Código Java para la llamada a un servicio (III)

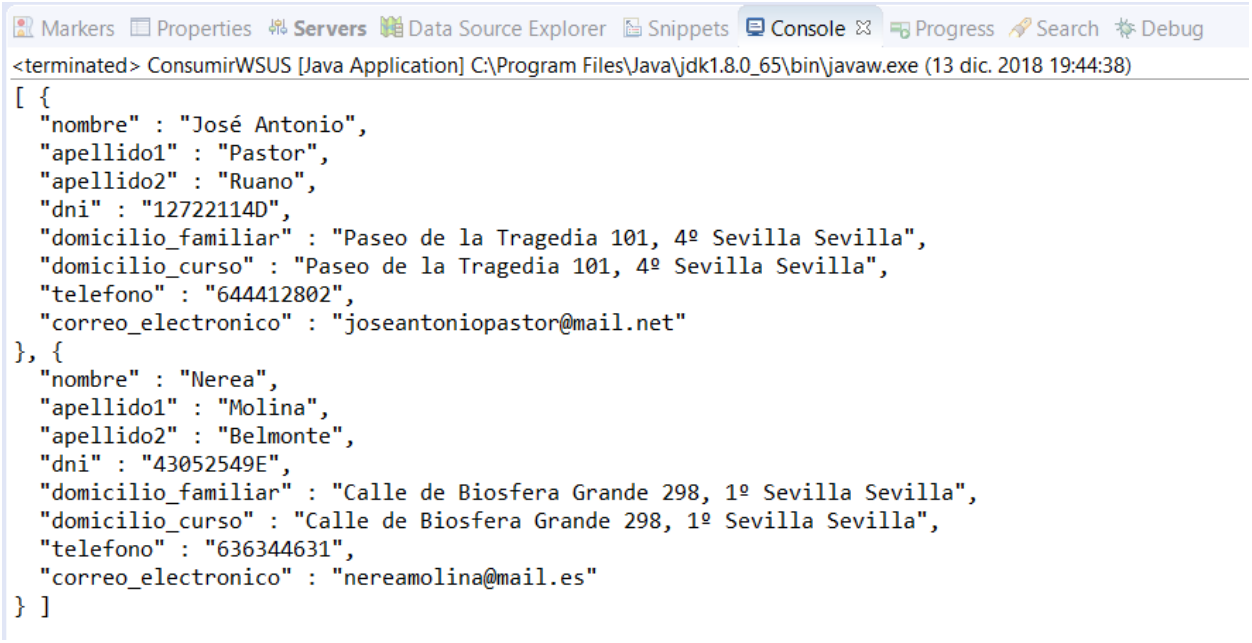

Ilustración 39 – Respuesta de la llamada a un servicio

4.3 Funcionalidad

En este apartado se explican y ejemplifican los diversos recursos que forman parte del Servicio Web. Se han creado conforme a lo que sería una demanda de datos de interés de cara a la resolución de trámites. En primer lugar, se presenta un resumen general de todos ellos y posteriormente se profundiza en cada uno, ilustrándolos mediante los datos con los que solicitaría el recurso y la respuesta que se obtendría.

4.3.1 Conexión al servicio web

Para hacer peticiones al servicio web, simplemente se deben realizar peticiones HTTP (GET, POST…) a los diferentes recursos del servicio con unos parámetros de entrada y un token de acceso. Exceptuando el recurso de Obtener Token, en el que el token no es necesario. Estos son los recursos disponibles, sus rutas correspondientes, así como el método y el tipo de datos de entrada/salida.

Recurso	Ruta	Método	Entrada	Salida
Obtener Token	/wsus/rest/refrescarToken	POST	Json	Json
Obtener alumno por DNI	/wsus/rest/alumno?dni={valor}	GET	String	Json
Obtener lista de alumnos por DNI	/wsus/rest/alumnos	POST	Json	Json
Obtener calificaciones por alumnos y curso académico	/wsus/rest/calificaciones	POST	Json	Json
Obtener matriculaciones por alumnos y curso académico	/wsus/rest/matriculaciones	POST	Json	Json
Obtener resumen del alumno de cara al curso indicado	/wsus/rest/resumen	POST	Json	Json
Obtener titulaciones por nombre del centro	/wsus/rest/titulacionesPorCentro	POST	Json	Json
Obtener titulaciones por nombre de la titulación	/wsus/rest/titulaciones	POST	Json	Json

Tabla 10 – Desglose de recursos del Servicio Web

4.3.2 Obtener Token

Para obtener el Token de acceso se debe hacer una petición POST al recurso Obtener Token con ruta "/wsus/rest/refrescarToken" y un JSON de entrada que sigue este formato:

```
{"tokenClave": "<contraseña>"}
```

Se debe sustituir <contraseña> por la contraseña que se le proporcione previamente al usuario. Si no hay ningún error, devolverá un JSON donde estará el token de acceso, que se debe guardar para las posteriores llamadas al servicio web.

La respuesta en caso de éxito será:

```
{
"token": "arYgTcj5h4…"
}
```

Si se ha producido un error:

```
{"mensaje": "Mensaje de error"}
```

El token de acceso tiene un tiempo de expiración configurable, una vez haya pasado este tiempo desde la generación del mismo se debe volver a crear otro token.

4.3.3 Obtener alumno por DNI

Este recurso presenta como utilidad devolver los datos del alumno con dni igual al que se le pasa por parámetro. Se debe hacer una petición GET con ruta "/wsus/rest/alumno?dni={valor}", donde dni es un String que se envía como parámetro:

Debe tener siempre el formato "00000000X". Por ejemplo:

```
/wsus/rest/alumno?dni=32454678F
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
{
  "nombre": "Rubén",
  "apellido1": "Jiménez",
  "apellido2": "Salazar",
  "dni": "41892816H",
  "domicilio_familiar": "Avenida Fortuna 84, 1º Morón de la Frontera Sevilla",
  "domicilio_curso": "Avenida Fortuna 84, 1º Morón de la Frontera Sevilla",
  "telefono": "666045430",
  "correo_electronico": "rubenjimenez@email.es"
}
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{"mensaje": "Mensaje de error"}
```

4.3.4 Obtener alumnos por DNI

Para obtener los datos de más de un alumno se debe hacer uso del siguiente recurso, para ello se informa en la petición de varios DNI y el servicio devolverá la información de aquellos que encuentre en la base de datos. Se debe hacer una petición POST con ruta "/wsus/rest/alumnos", donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:

```
{
   "dni": [
                "12722114D",
                "43052549E"
            ]
}
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
[
    {
    "nombre": "José Antonio",
    "apellido1": "Pastor",
    "apellido2": "Ruano",
    "dni": "12722114D",
    "domicilio_familiar": "Paseo de la Tragedia 101, 4º Sevilla Sevilla",
    "domicilio_curso": "Paseo de la Tragedia 101, 4º Sevilla Sevilla",
    "telefono": "644412802",
    "correo_electronico": "joseantoniopastor@mail.net"
  },
    {
    "nombre": "Nerea",
    "apellido1": "Molina",
    "apellido2": "Belmonte",
    "dni": "43052549E",
    "domicilio_familiar": "Calle de Biosfera Grande 298, 1º Sevilla Sevilla",
    "domicilio_curso": "Calle de Biosfera Grande 298, 1º Sevilla Sevilla",
    "telefono": "636344631",
    "correo_electronico": "nereamolina@mail.es"
  }
]
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{"mensaje": "Mensaje de error"}
```

4.3.5 Obtener calificaciones por alumnos y curso académico

Este recurso será de utilidad para obtener las calificaciones filtrando por dni del alumno y curso académico. Para ello se le envían los datos de interés en formato JSON. Se debe hacer una petición POST con ruta "/wsus/rest/calificaciones", donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:

```
{
   "dni": [
                "12722114D",
                "43052549E"
            ],

    "curso": [
                "2017/2018"
            ]

}
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
[
    {
    "nombre": "José Antonio Pastor Ruano",
    "curso": "2017/2018",
    "titulacion": "Grado en Ingeniería Electrónica, Robótica y Mecatrónica",
    "calificaciones":      {
      "Física II": 7,
      "Matemáticas I": 7.2,
      "Estadística e Investigación Operativa": 0.9,
      "Física I": 3.6,
      "Matemáticas III": 0.6,
      "Expresión Gráfica": 3.5,
      "Química": 0.3,
      "Empresa": 7.1,
      "Matemáticas II": 9,
      "Informática": 1.6
    }
  },
    {
    "nombre": "Nerea Molina Belmonte",
    "curso": "2017/2018",
    "titulacion": "Grado en Ingeniería Electrónica, Robótica y Mecatrónica",
    "calificaciones":      {
      "Física II": 7.1,
      "Matemáticas I": 10,
      "Estadística e Investigación Operativa": 3.3,
      "Física I": 3.3,
      "Matemáticas III": 8.9,
      "Expresión Gráfica": 4.7,
      "Química": 0.9,
      "Empresa": 8.9,
```

```
      "Matemáticas II": 7.4,
      "Informática": 9.1
    }
  }
]
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{"mensaje": "Mensaje de error"}
```

4.3.6 Obtener matriculaciones por alumnos y curso académico

Para tener acceso a las matriculaciones de los alumnos se hará uso de este recurso, en él se informará del conjunto de DNI y curso académico que se quiere consultar. Para ello se le envían los datos de interés en formato JSON. Se debe hacer una petición POST con ruta "/wsus/rest/matriculaciones", donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:

```
{
  "dni": [
            "12722114D",
            "43052549E"
          ],

  "curso": [
            "2017/2018",
            "2018/2019"
]
}
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
[
    {
    "nombre": "José Antonio Pastor Ruano",
    "titulacion": "Grado en Ingeniería Electrónica, Robótica y Mecatrónica",
    "cursos":    {
      "2017/2018":      {
        "matriculaciones":        {
          "Física II": 6,
          "Matemáticas I": 6,
          "Estadística e Investigación Operativa": 6,
          "Física I": 6,
          "Matemáticas III": 6,
          "Expresión Gráfica": 6,
          "Química": 6,
          "Empresa": 6,
          "Matemáticas II": 6,
```

```
        "Informática": 6
      },
      "totalCreditos": 60
    },
    "2018/2019":      {
      "matriculaciones":        {
        "Teoría de Máquinas y Mecanismos": 6,
        "Resistencia de Materiales": 6,
        "Estadística e Investigación Operativa": 6,
        "Física I": 6,
        "Matemáticas III": 6,
        "Expresión Gráfica": 6,
        "Química": 6,
        "Fundamentos de Electrónica": 6,
        "Ampliacion de Matemáticas": 6,
        "Informática": 6
      },
      "totalCreditos": 60
    }
  }
},
  {
  "nombre": "Nerea Molina Belmonte",
  "titulacion": "Grado en Ingeniería Electrónica, Robótica y Mecatrónica",
  "cursos":     {
    "2017/2018":      {
      "matriculaciones":        {
        "Física II": 6,
        "Matemáticas I": 6,
        "Estadística e Investigación Operativa": 6,
        "Física I": 6,
        "Matemáticas III": 6,
        "Expresión Gráfica": 6,
        "Química": 6,
        "Empresa": 6,
        "Matemáticas II": 6,
        "Informática": 6
      },
      "totalCreditos": 60
    },
    "2018/2019":      {
      "matriculaciones":        {
        "Fundamentos de Electrónica": 6,
        "Fundamentos de Computadores": 6,
        "Estadística e Investigación Operativa": 6,
        "Física I": 6,
        "Fundamentos de Control": 6,
        "Expresión Gráfica": 6,
        "Química": 6,
        "Automatización Industrial": 6,
        "Resistencia de Materiales": 6,
        "Teoría de Circuitos": 6
      },
      "totalCreditos": 60
    }
  }
```

```
  }
]
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{“mensaje”: “Mensaje de error”}
```

4.3.7 Obtener resumen por alumno

Para obtener un resumen informativo (créditos matriculados, aprobados, suspensos, porcentaje aprobados, residencia fuera del domicilio familiar o no) filtrando por dni del alumno y el curso académico. Para ello se le envían los datos de interés en formato JSON. Se debe hacer una petición POST con ruta “/wsus/rest/resumen”, donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:

```
{
  "dni": [
              "12722114D",
              "59967014B"
           ],

  "curso": [
              "2018/2019"
  ]
}
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
[
   {
   "nombre": "Raquel Mulet Villalba",
   "dni": "59967014B",
   "titulacion": "Grado en Ingeniería de Tecnologías Industriales",
   "cursos":     {
     "2017/2018":       {
       "creditosMatriculados": 60,
       "creditosAprobados": 40.5,
       "creditosSuspensos": 19.5,
       "porcentajeAprobados": "68.00%",
       "notaMedia": 6.1
     },
     "2018/2019":       {
       "creditosMatriculados": 57,
       "ResideFuera": "Si"
     }
   }
 },
   {
```

```
    "nombre": "José Antonio Pastor Ruano",
    "dni": "12722114D",
    "titulacion": "Grado en Ingeniería Electrónica, Robótica y Mecatrónica",
    "cursos":      {
      "2017/2018":        {
        "creditosMatriculados": 60,
        "creditosAprobados": 24,
        "creditosSuspensos": 36,
        "porcentajeAprobados": "40.00%",
        "notaMedia": 4.1
      },
      "2018/2019":        {
        "creditosMatriculados": 60,
        "ResideFuera": "No"
      }
    }
  }
]
```


En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:


```
{"mensaje": "Mensaje de error"}
```


4.3.8 Obtener titulaciones filtrando por nombre del centro

Para obtener las titulaciones que se imparten en los centros filtrando por una cadena de texto que esté contenida en el nombre del centro, la búsqueda se hará sin importar mayúsculas y minúsculas. Para ello se le envían los datos de interés en formato JSON. Se debe hacer una petición POST con ruta "/wsus/rest/titulacionesPorCentro", donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:


```
{
   "centros": [
                  "MeD"
              ]
}
```


También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:


```
[{
  "nombre": "FACULTAD DE MEDICINA",
  "direccion": "AVDA. DOCTOR FEDRIANI, S/N C.P: 41009 (SEVILLA)",
  "correoElectronico": "facmedinfo@us.es",
  "telefono": "95.455.98.30/29/25",
  "web": "http://www.medicina.us.es/",
  "titulaciones":    [
    "Grado en Medicina",
    "Grado en Biomedicina Básica y Experimental",
    "Máster Universitario en Investigación Médica: Clínica y Experimental"
  ]
}]
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{“mensaje”: “Mensaje de error”}
```

4.3.9 Obtener titulaciones filtrando por nombre de la titulación

Para obtener las titulaciones que se imparten en los centros filtrando por una cadena de texto que esté contenida en el nombre de la titulación, la búsqueda se hará sin importar mayúsculas y minúsculas. Para ello se le envían los datos de interés en formato JSON. Se debe hacer una petición POST con ruta “/wsus/rest/titulaciones”, donde los datos se envían en el cuerpo de la petición:

Debe tener siempre el formato:

```
{
   "titulaciones": [
                  "teleco",
                  "podo"
               ]
}
```

También se debe añadir a la petición el token de acceso anteriormente generado. Si no hay ningún error, devolverá un JSON con el siguiente formato:

```
[
    {
    "nombre": "ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA",
    "direccion": "CAMINO DESCUBRIMIENTOS, S/N.- ISLA CARTUJA C.P: 41092 (SEVILLA)",
    "correoElectronico": "jhidalgo@esi.us.es",
    "telefono": "95.448.61.00/61.50/73.96",
    "web": "http://www.etsi.us.es",
    "titulaciones":      [
      "Grado en Ingeniería de las Tecnologías de Telecomunicación",
      "Máster Universitario en Ingeniería de Telecomunicación"
    ]
  },
    {
    "nombre": "FACULTAD DE ENFERMERÍA, FISIOTERAPIA Y PODOLOGÍA",
    "direccion": "AVENZOAR, 6 C.P: 41009 (SEVILLA)",
    "correoElectronico": "banuls@us.es",
    "telefono": "95.455.14.71/72/68/79.05/64.80",
    "web": "http://www.fefp.us.es",
    "titulaciones": ["Grado en Podología"]
  }
]
```

En caso de no obtener ningún resultado, no devolverá nada. Se controlará mediante el código HTTP 204 – NO CONTENT.

En caso de producirse un error, se devolverá un JSON con el siguiente formato:

```
{“mensaje”: “Mensaje de error”}
```

En este capítulo se ha hecho mención a cada uno de los pasos seguidos para el desarrollo del Servicio Web. Para ello se ha comenzado explicando los pasos realizados para la creación de una base de datos Oracle que dará cabida a los datos que consumirá el servicio. Además, se ha mostrado el modelo de datos creado para poder realizar las pruebas, detallando cada una de las tablas y aportando una visión global del esquema relacional conseguido. Se han detallado también los pasos ejecutados para la creación del proyecto de cero con un arquetipo Spring MVC y estructurándolo y configurándolo gracias al software Maven. Se ha seguido así con la implantación de Spring para atender las peticiones y generar la respuesta; y con Hibernate como principal artífice de las consultas y comunicación con la base de datos. Se cierra el capítulo mostrando la configuración de los servicios necesarios para la generación y comprobación de tokens de acceso JSON e introduciendo la funcionalidad de todos los recursos de los que dispondrá el Servicio Web.

En el siguiente capítulo se van a exponer cada uno de los errores definidos y controlados por el aplicativo y se llevará a cabo una batería de pruebas para cada uno de los recursos, en donde pueda verse reflejado el correcto funcionamiento de cada uno de ellos.

5 PRUEBAS

"No menos que el saber me place el dudar."

- Dante Alighieri -

En este apartado se van a exponer y desarrollar las distintas pruebas que se han llevado a cabo de cara a validar el correcto funcionamiento del servicio web.

En un primer punto se van a enumerar el conjunto de errores controlados que arroja la aplicación, mientras que en el segundo apartado se pondrá de manifiesto llevando a cabo una pila de pruebas que se ha ejecutado con los resultados obtenidos en cada caso.

5.1 Errores

El servicio web implementa un conjunto de errores controlados de cara al usuario, para que de esta manera pueda saber en todo momento el porqué de no obtener la respuesta deseada. Se van a enumerar todos ellos en los siguientes puntos.

5.1.1 Datos vacíos

Si el usuario haciendo uso de cualquiera de los servicios que se ofrecen no completa el cuerpo de la petición, se le informará mediante el siguiente mensaje.

{"mensaje": "Los parámetros de entrada son obligatorios"}

5.1.2 JSON incorrecto

Si el usuario haciendo uso de cualquiera de los servicios que se ofrecen completa el cuerpo de petición con un formato de JSON incorrecto se le informará a través del siguiente mensaje.

{"mensaje": "El Json de entrada tiene un formato inválido"}

5.1.3 Clave de generación incorrecta

Si el usuario haciendo uso del servicio para la generación de token completa el cuerpo de la petición respetando el formato, pero con una clave de generación incorrecta, será informado mediante el siguiente mensaje.

{"mensaje": "Clave de generación de token incorrecta"}

5.1.4 Token obligatorio

Si el usuario haciendo uso de los servicios para la obtención de datos no acompaña la petición con el token en su cabecera, le será informado a través del siguiente mensaje.

{"mensaje": "El token es obligatorio"}

5.1.5 Token incorrecto

Si el usuario haciendo uso de los servicios para la obtención de datos acompaña la petición con el token en su cabecera, pero éste es incorrecto, será notificado a través del siguiente mensaje.

{"mensaje": "El token introducido no es correcto"}

5.1.6 Token expirado

Si el usuario haciendo uso de los servicios para la obtención de datos acompaña la petición de un token que ha sobrepasado el tiempo de vida configurado en la aplicación, será informado mediante el siguiente mensaje.

{"mensaje": "El token ha expirado"}

Por lo que el usuario tendrá que volver a solicitar un nuevo token de acceso.

5.1.7 Obtención de datos

Si ocurre un error en la aplicación a la hora de obtener los datos y formar la respuesta, el usuario será notificado a través del siguiente mensaje.

{"mensaje": "Error al obtener los datos, consulte con el administrador"}

5.2 Test

En este apartado se van a realizar las pruebas oportunas para comprobar el correcto funcionamiento de la aplicación. Para ello se va a proceder a probar cada servicio de manera ordenada modificando la configuración de la llamada al servicio o los parámetros de entrada. Para realizar los test se ha usado el software SoapUI.

5.2.1 Refrescar token

- **Cuerpo de la petición vacío.**

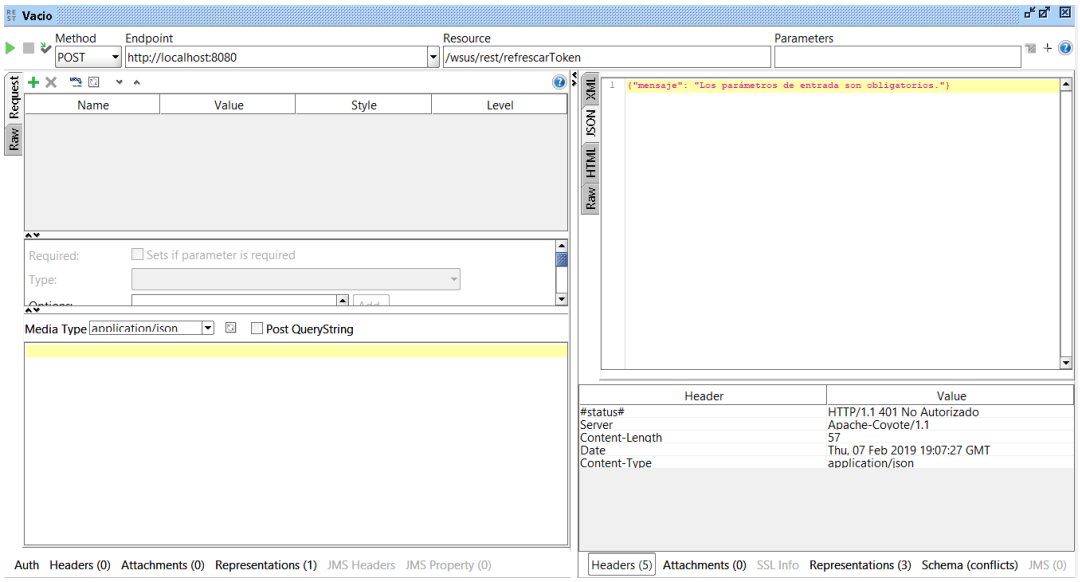

Ilustración 40 – Refrescar token (Cuerpo vacío)

- **Formato del cuerpo de la petición inválido**

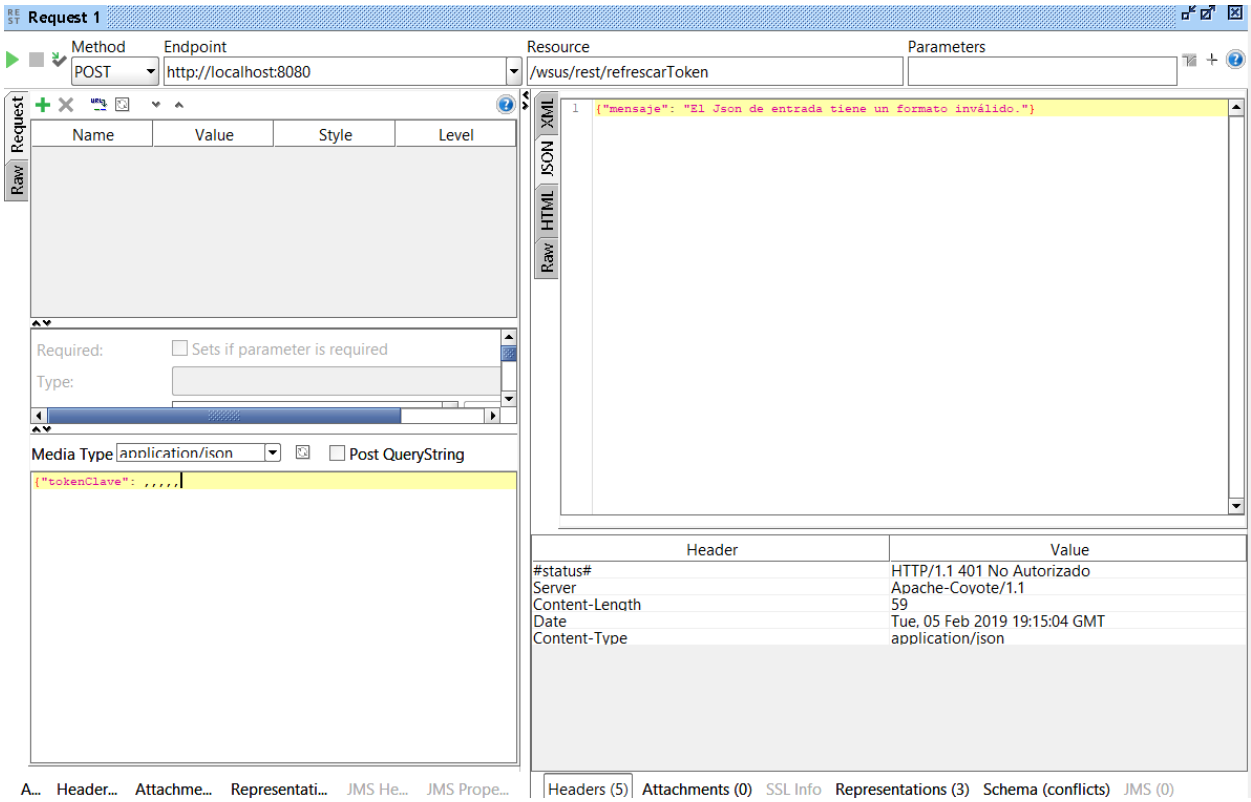

Ilustración 41 - Refrescar token (Formato inválido)

- **Clave incorrecta**

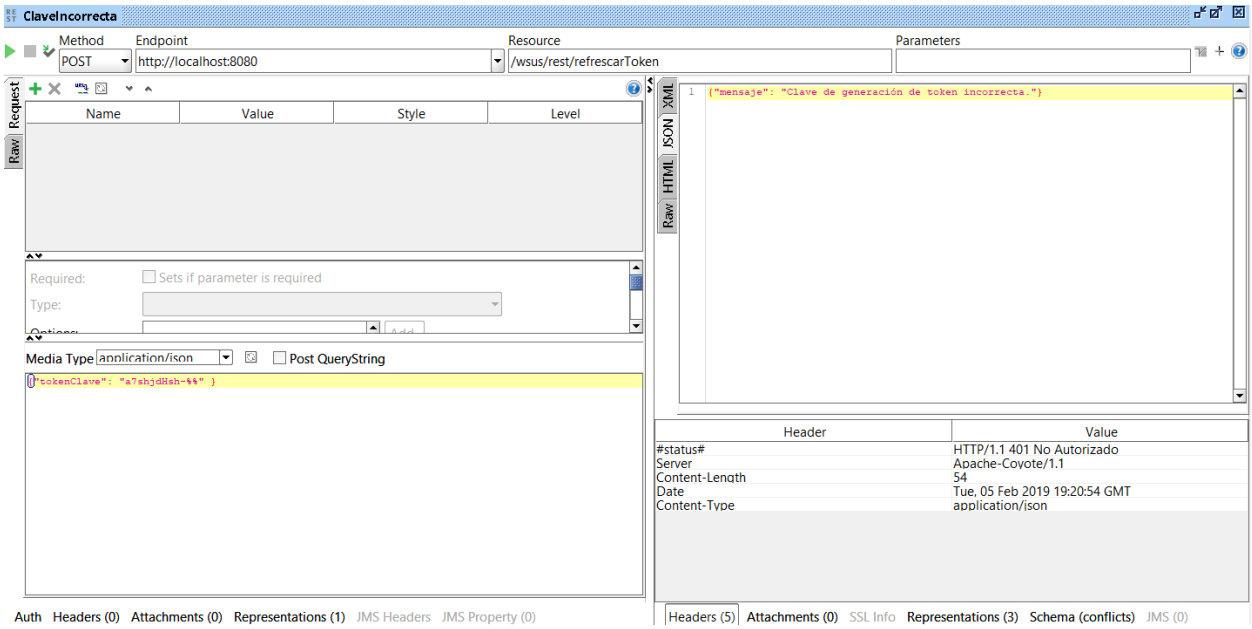

Ilustración 42 - Refrescar token (Clave incorrecta)

- **Clave correcta**

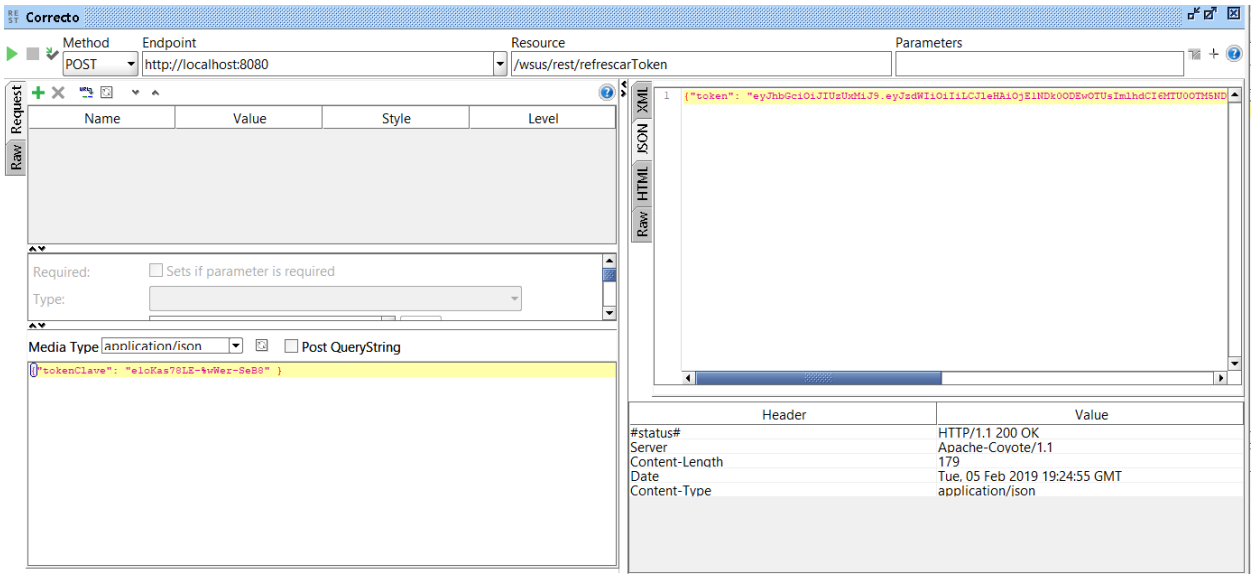

Ilustración 43 - Refrescar token (Clave correcta)

5.2.2 Obtener alumno por DNI

- **Sin parámetro**

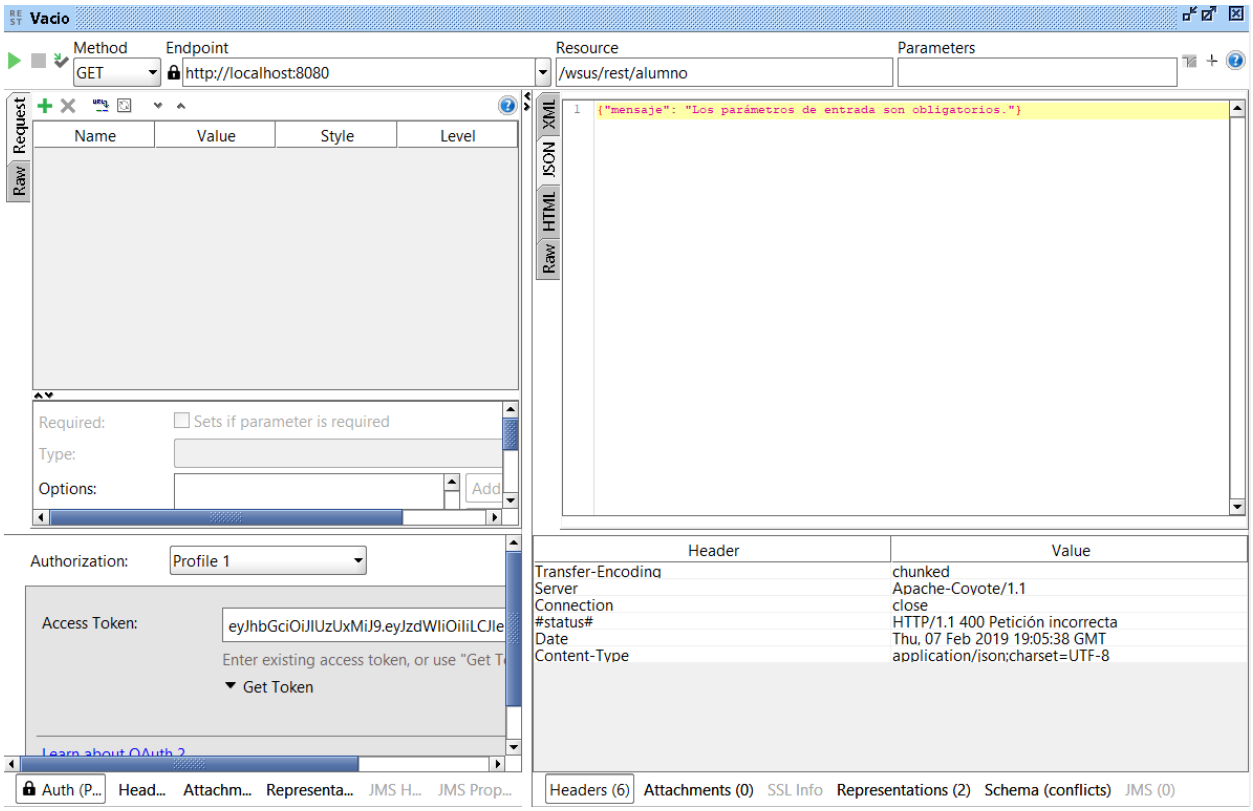

Ilustración 44 – Obtener alumno por DNI (Sin parámetro)

- **Token vacío**

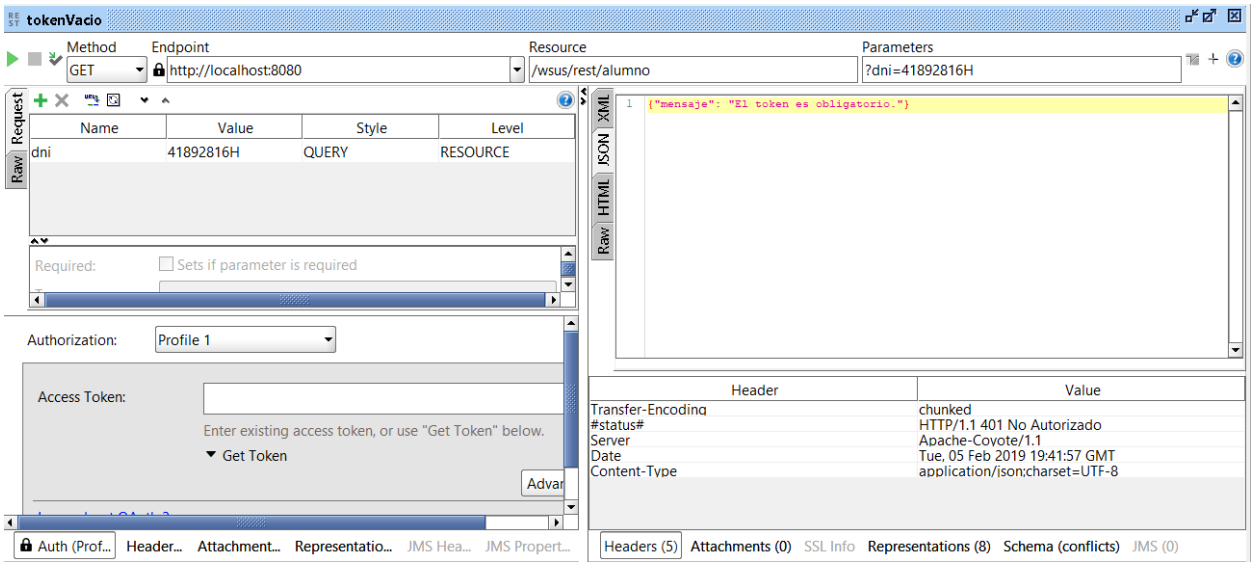

Ilustración 45 - Obtener alumno por DNI (Token vacío)

- **Token incorrecto**

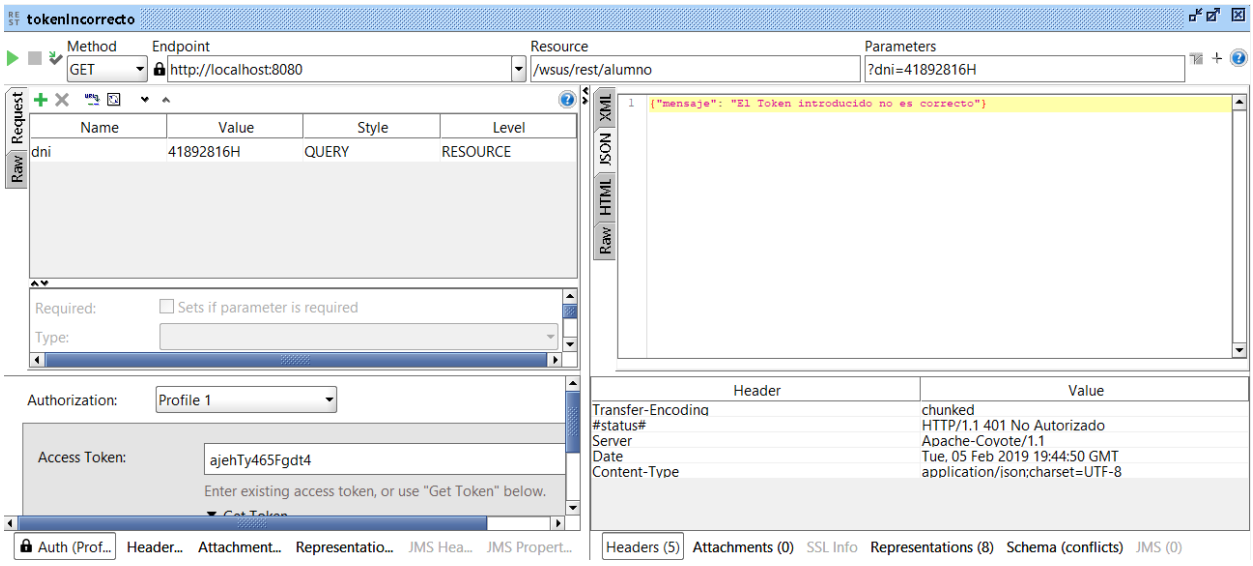

Ilustración 46 - Obtener alumno por DNI (Token incorrecto)

- **Token expirado**

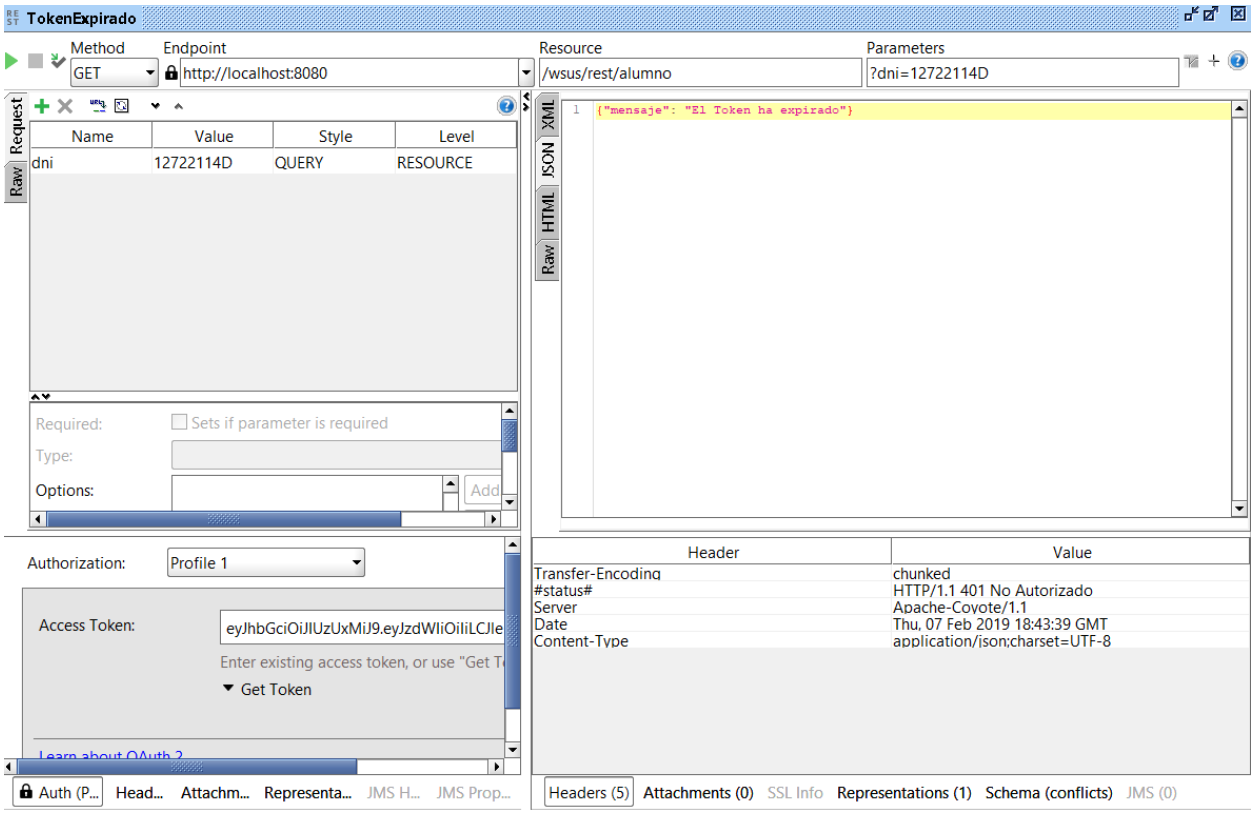

Ilustración 47 - Obtener alumno por DNI (Token expirado)

- **Token correcto**

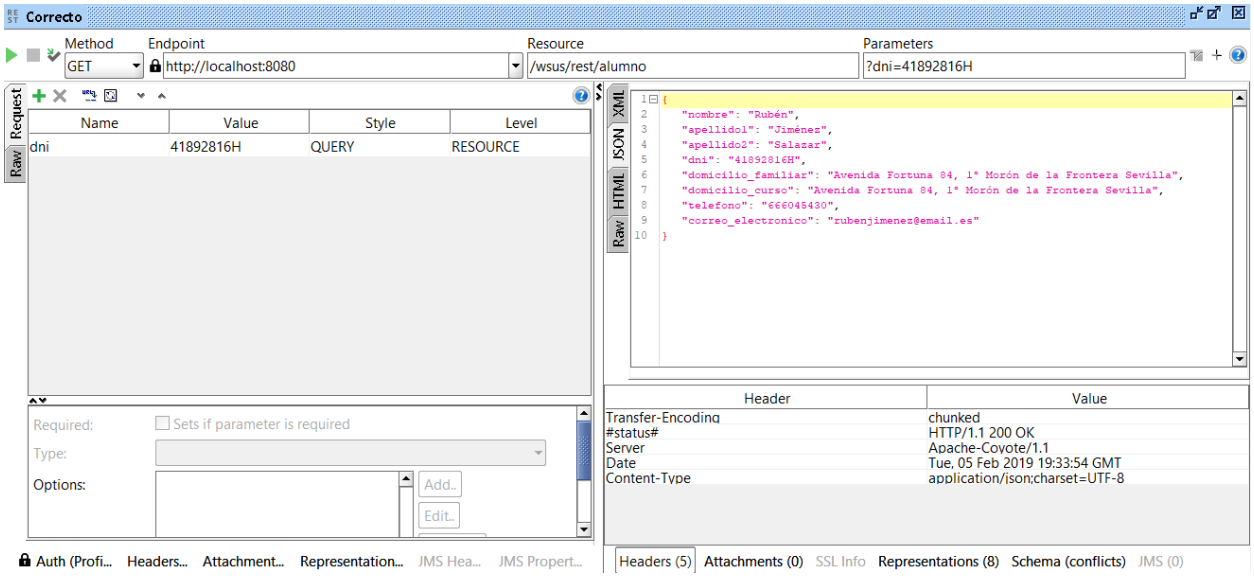

Ilustración 48 - Obtener alumno por DNI (Token correcto)

5.2.3 Obtener lista de alumnos por DNI

- **Token vacío**

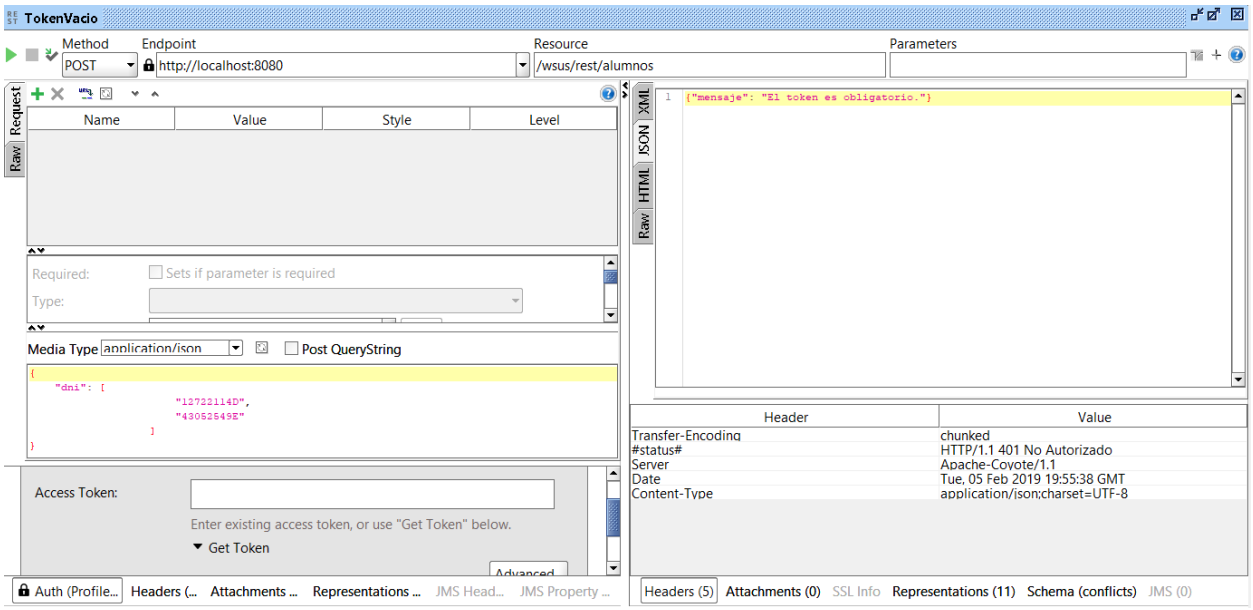

Ilustración 49 – Obtener lista de alumnos por DNI (Token vacío)

- **Token incorrecto**

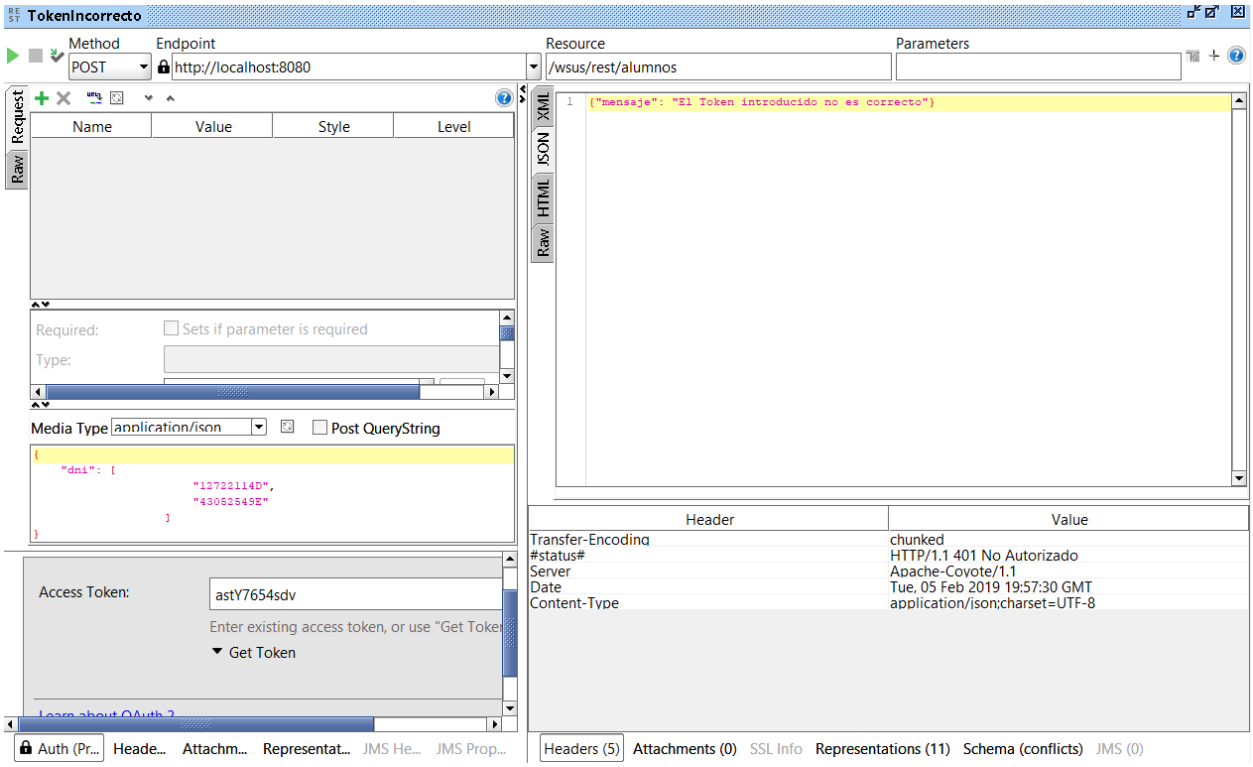

Ilustración 50 - Obtener lista de alumnos por DNI (Token incorrecto)

- **Token expirado**

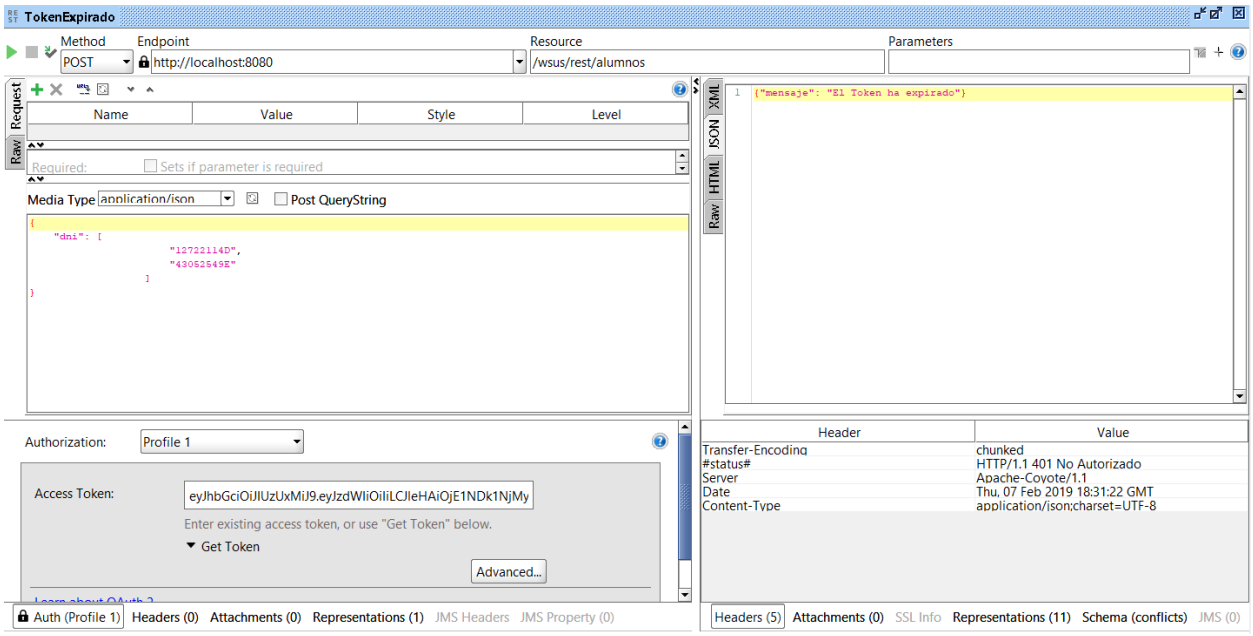

Ilustración 51 - Obtener lista de alumnos por DNI (Token expirado)

- **Cuerpo vacío**

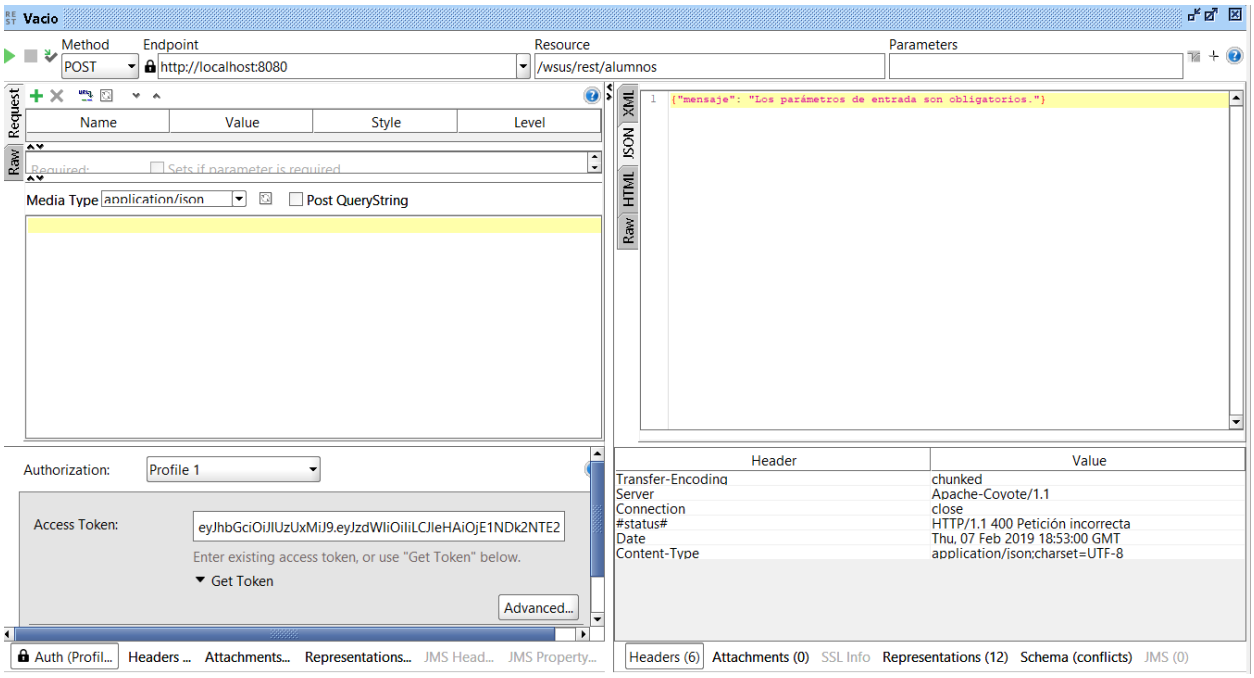

Ilustración 52 - Obtener lista de alumnos por DNI (Cuerpo vacío)

- **Formato incorrecto**

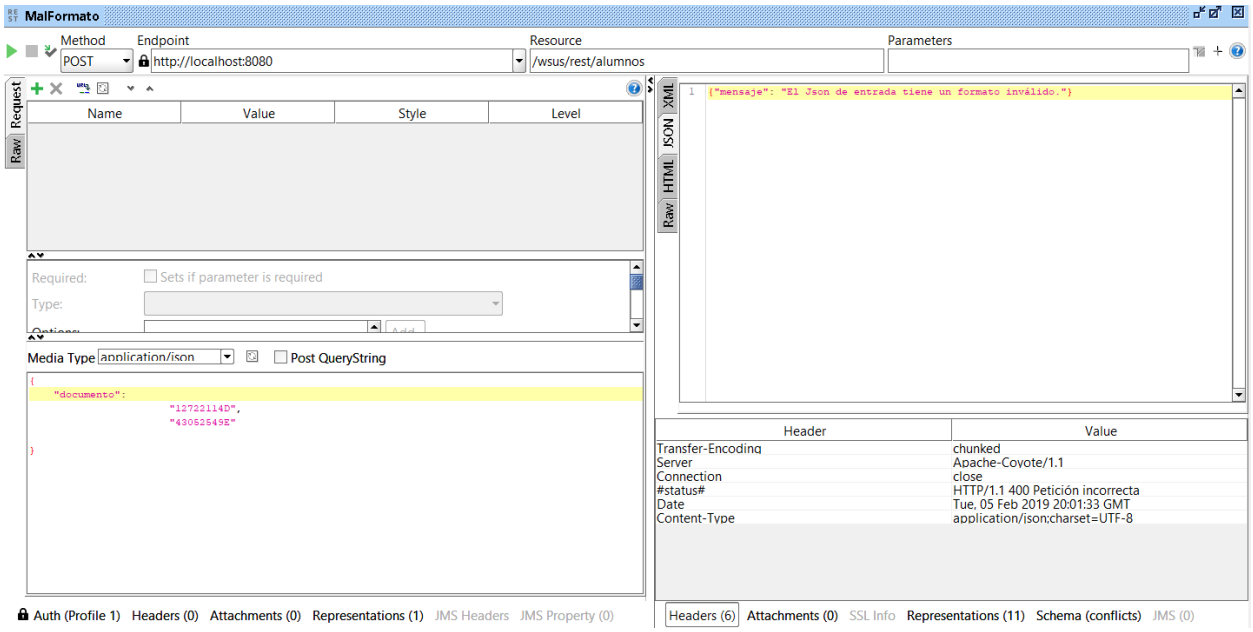

Ilustración 53 - Obtener lista de alumnos por DNI (Formato incorrecto)

- **Correcto**

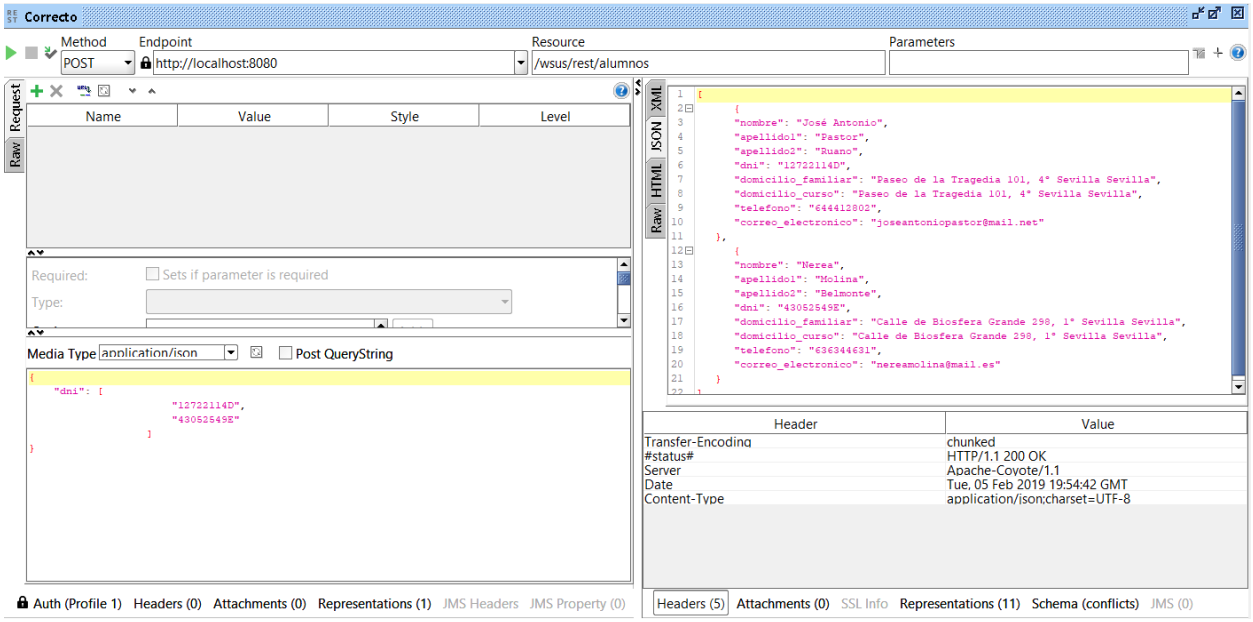

Ilustración 54 - Obtener lista de alumnos por DNI (Correcto)

5.2.4 Obtener calificaciones por DNI del alumno

- **Token vacío**

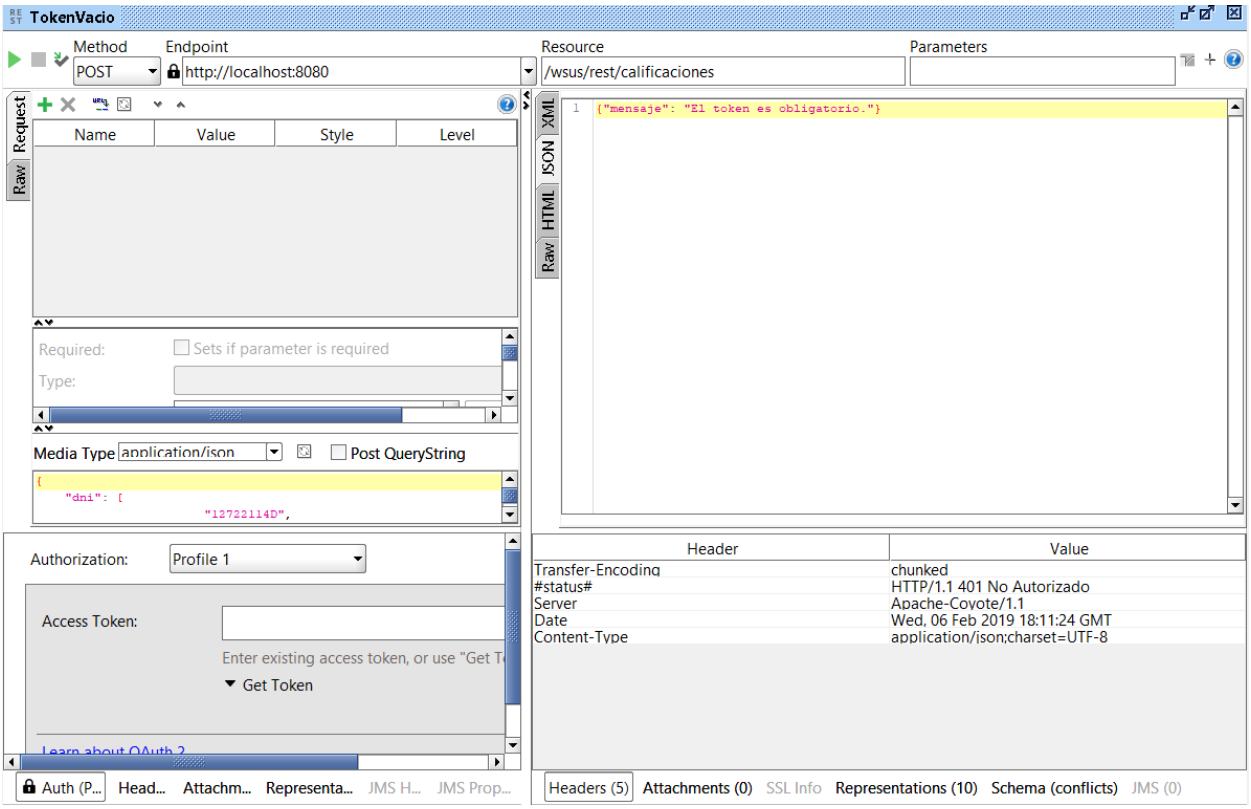

Ilustración 55 – Obtener calificaciones por DNI (Token vacío)

- **Token incorrecto**

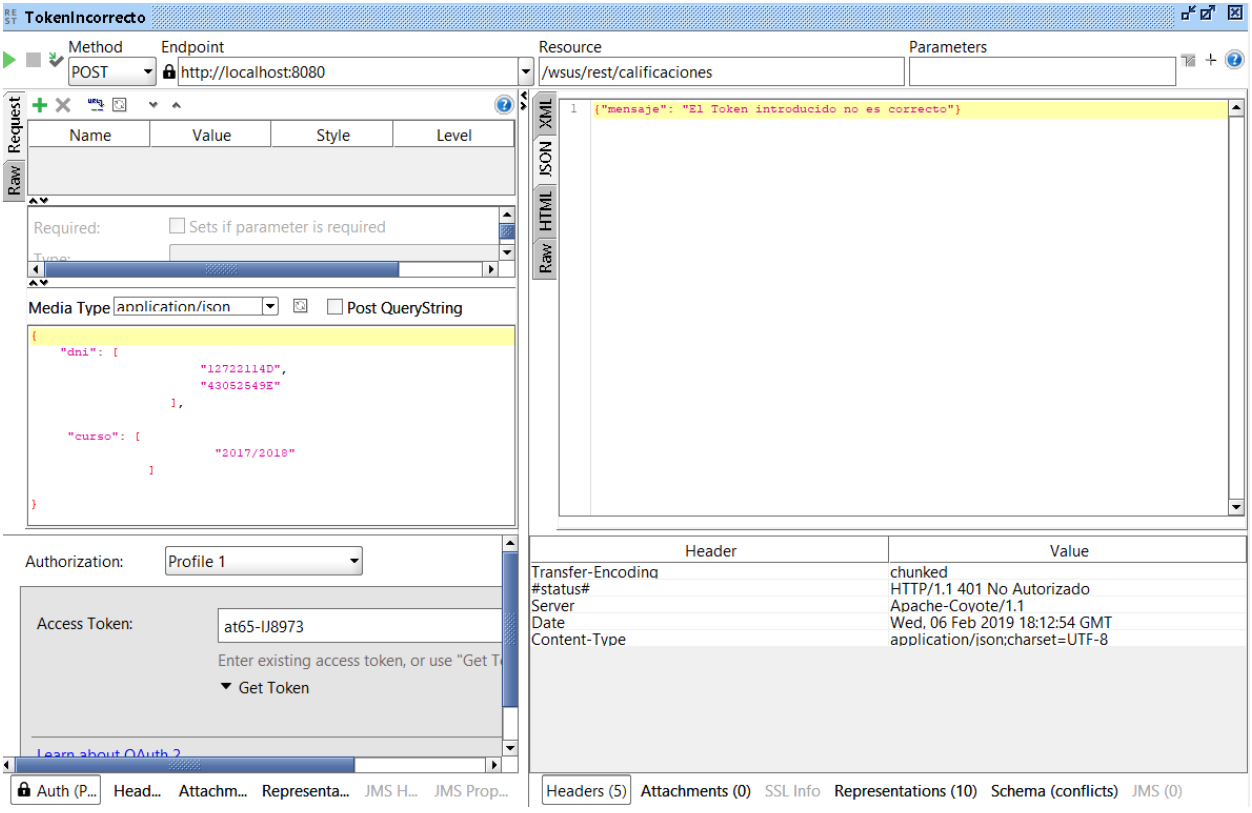

Ilustración 56 - Obtener calificaciones por DNI (Token incorrecto)

- **Token expirado**

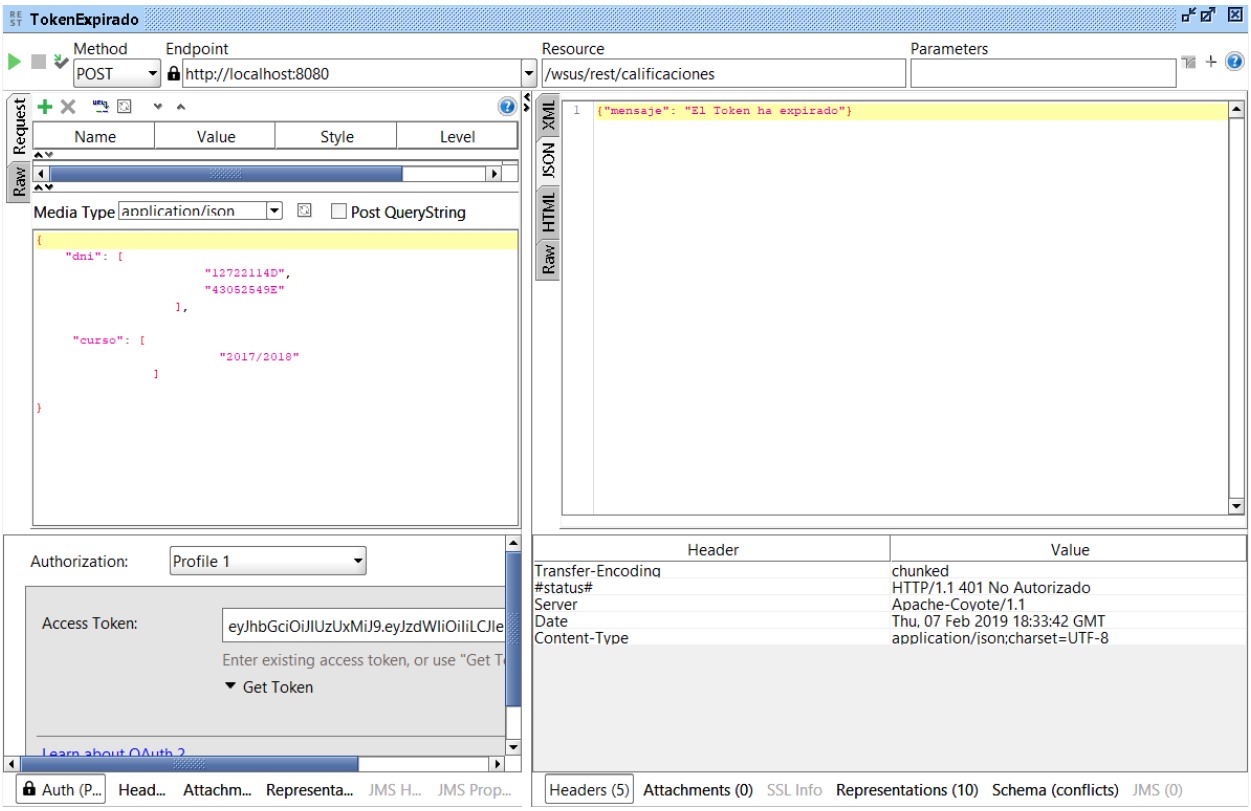

Ilustración 57 - Obtener calificaciones por DNI (Token expirado)

- **Cuerpo vacío**

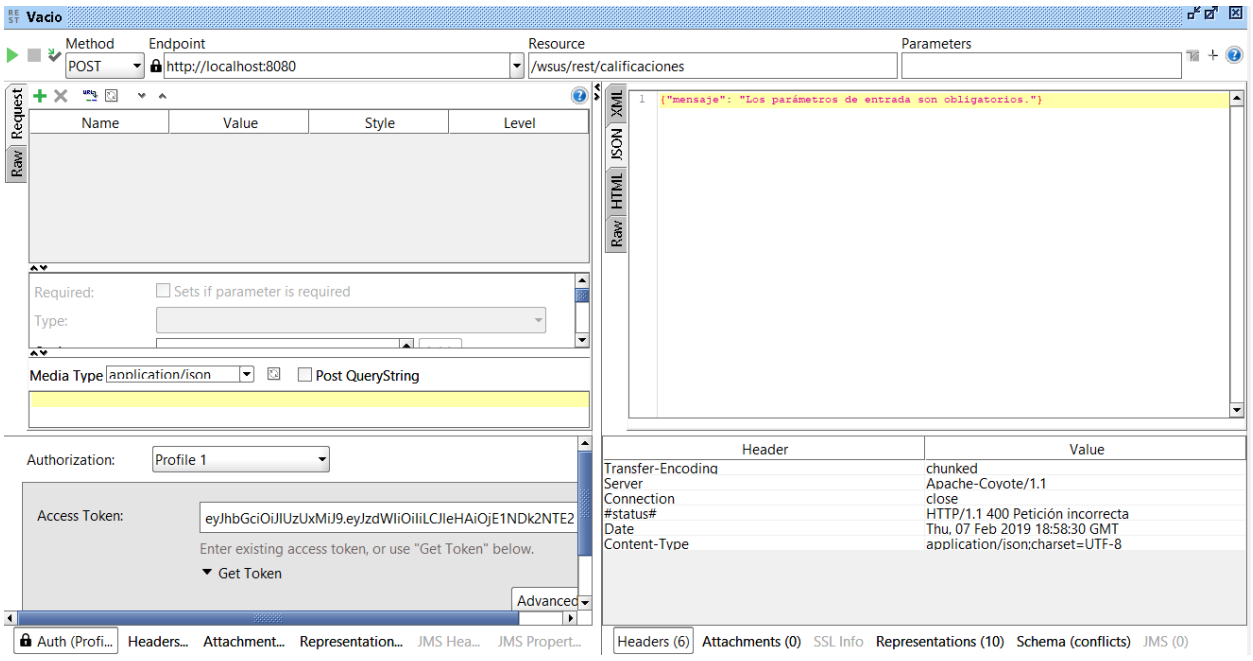

Ilustración 58 - Obtener calificaciones por DNI (Cuerpo vacío)

- **Formato incorrecto**

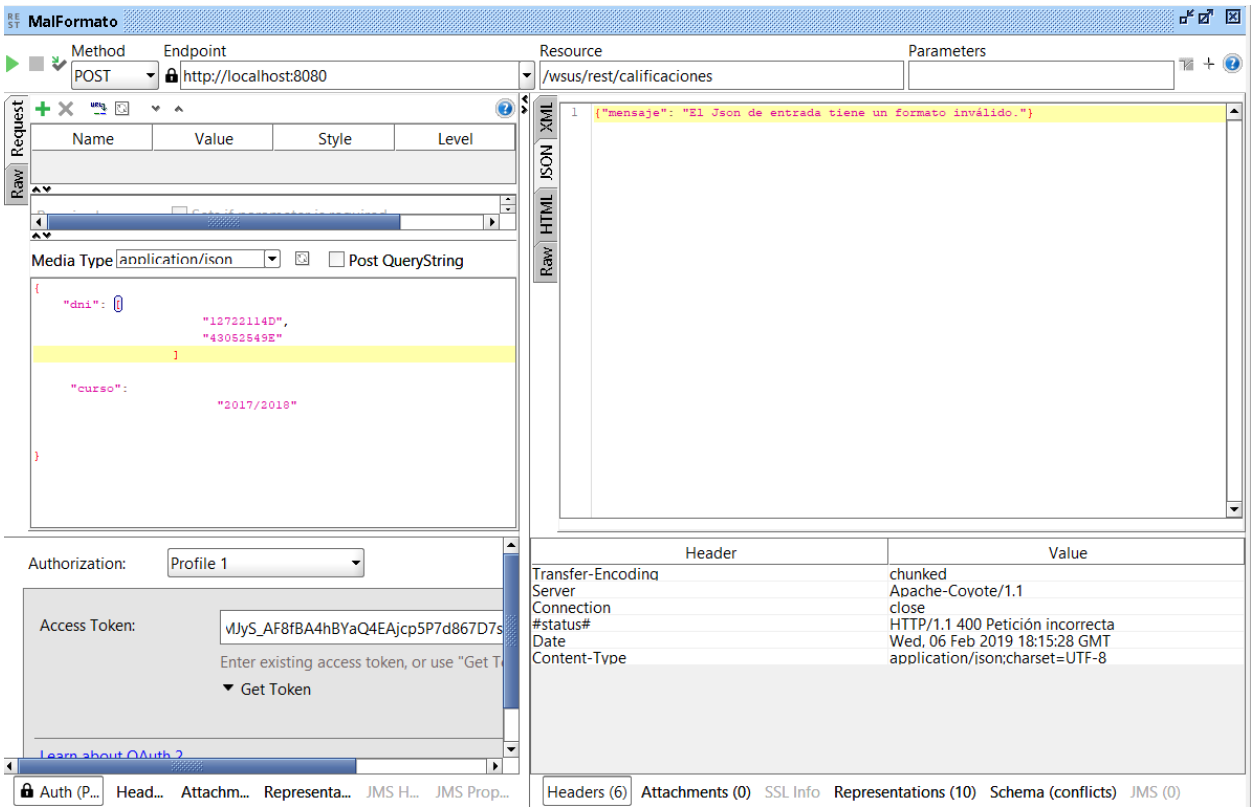

Ilustración 59 - Obtener calificaciones por DNI (Formato incorrecto)

- **Correcto**

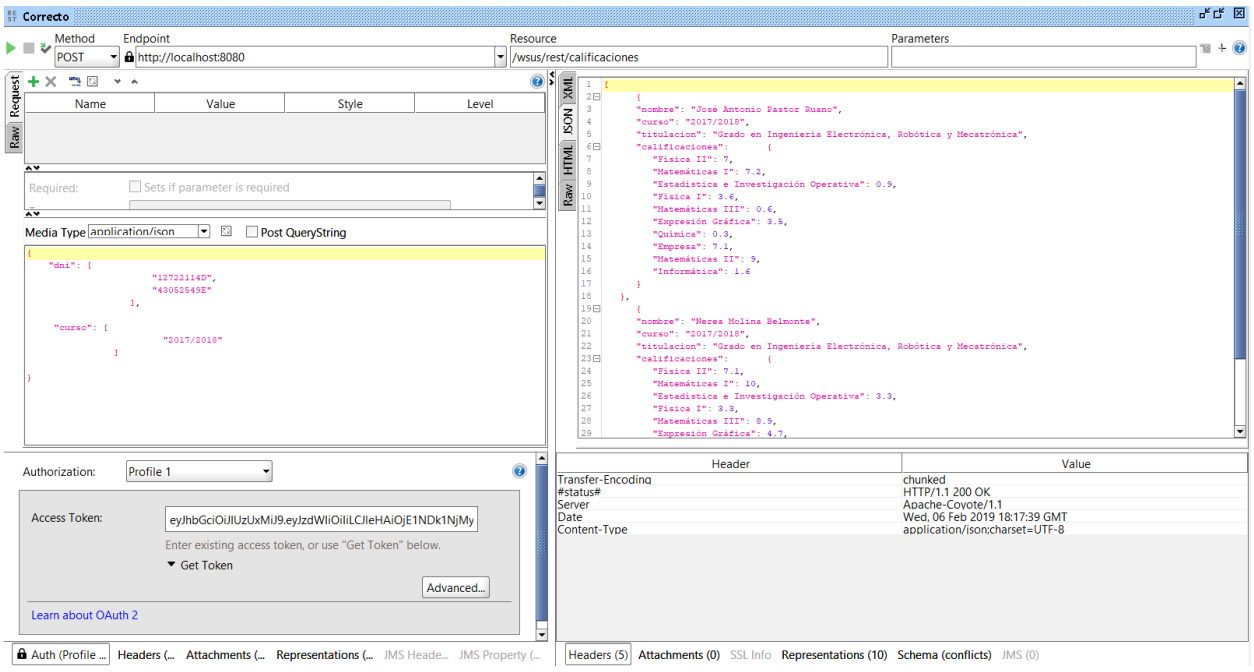

Ilustración 60 - Obtener calificaciones por DNI (Correcto)

5.2.5 Obtener matriculaciones por DNI del alumno

- **Token vacío**

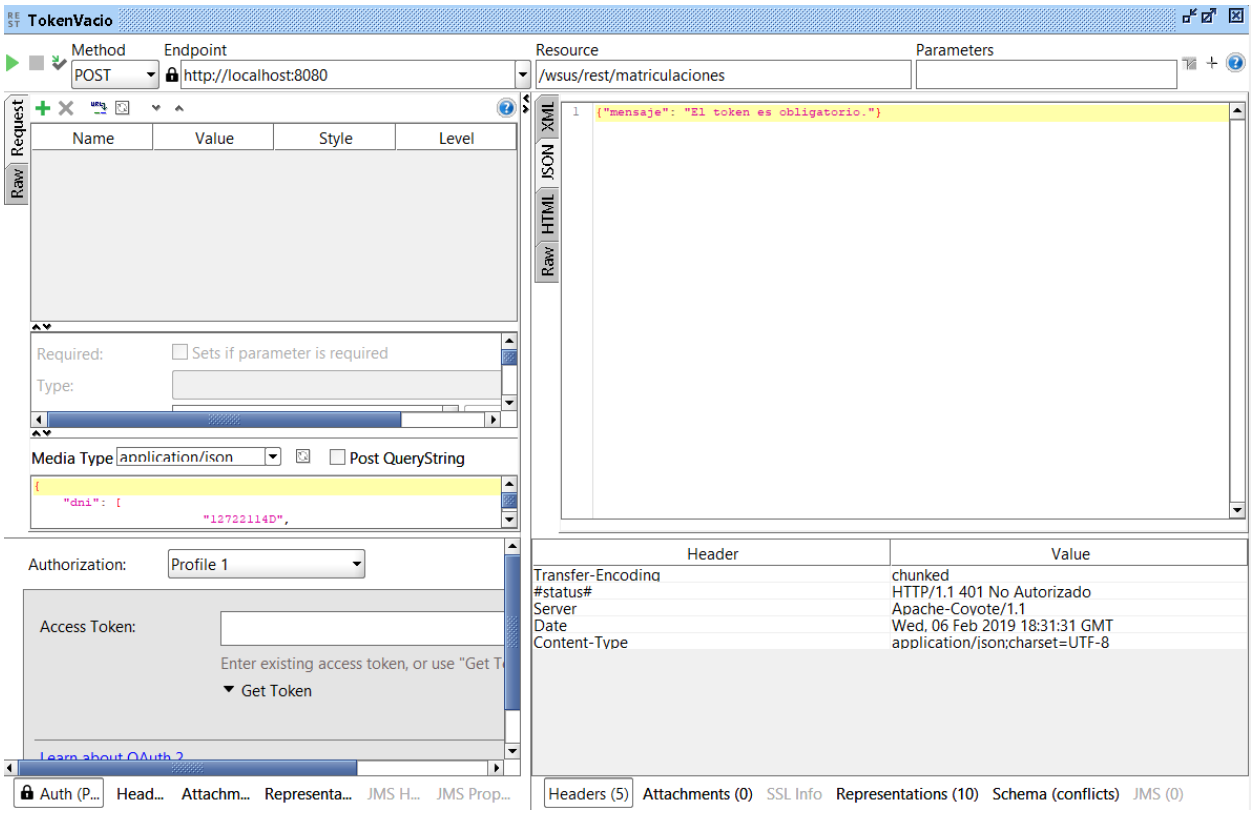

Ilustración 61 - Obtener matriculaciones por DNI (Token vacío)

- **Token incorrecto**

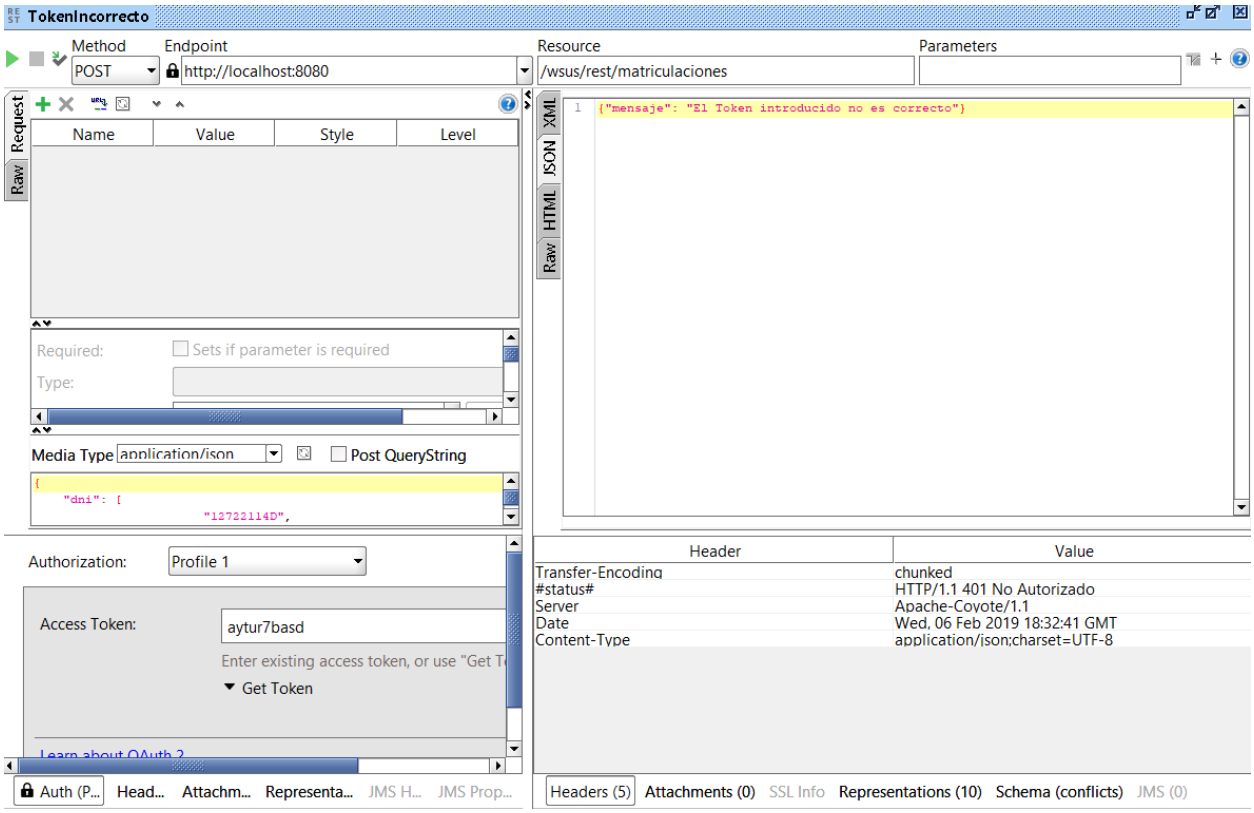

Ilustración 62 - Obtener matriculaciones por DNI (Token incorrecto)

- **Token expirado**

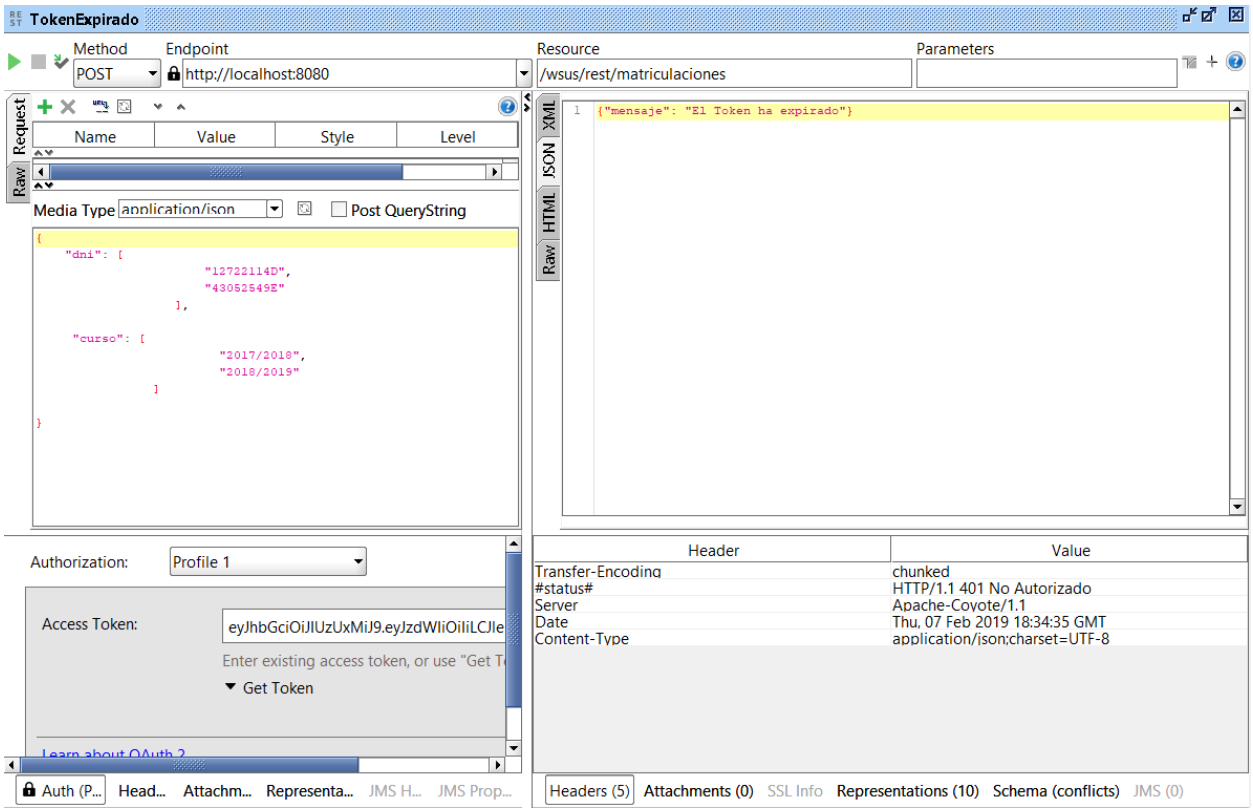

Ilustración 63 - Obtener matriculaciones por DNI (Token expirado)

- **Cuerpo vacío**

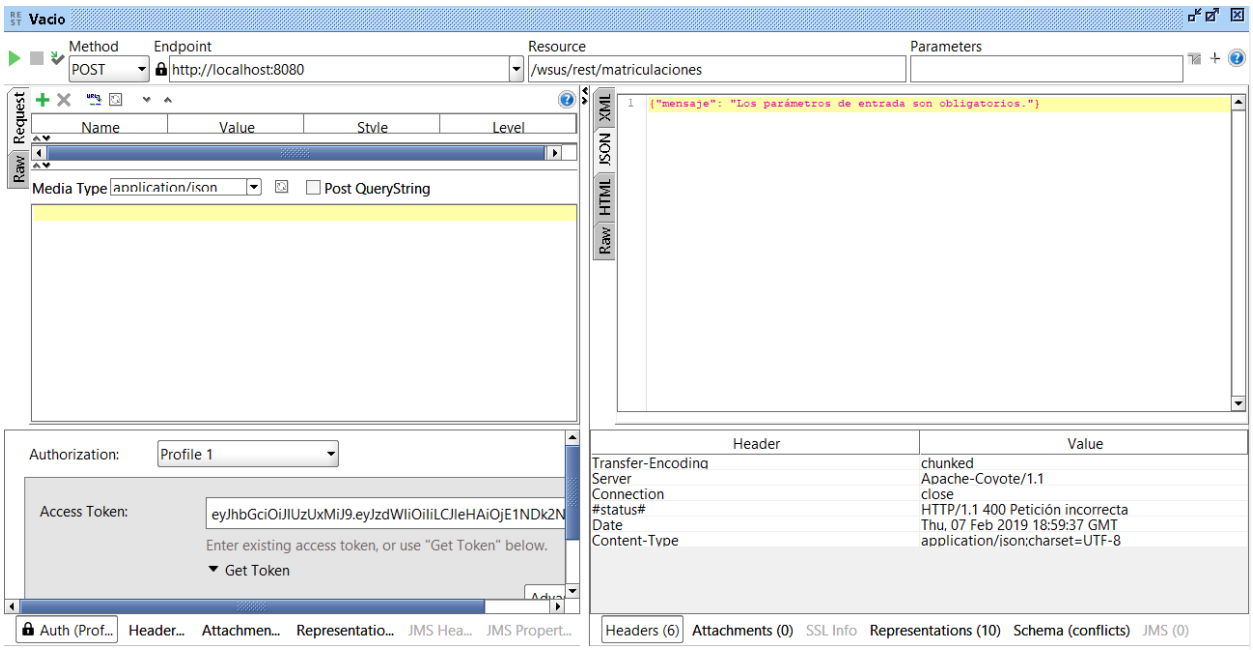

Ilustración 64 - Obtener matriculaciones por DNI (Cuerpo vacío)

- **Formato incorrecto**

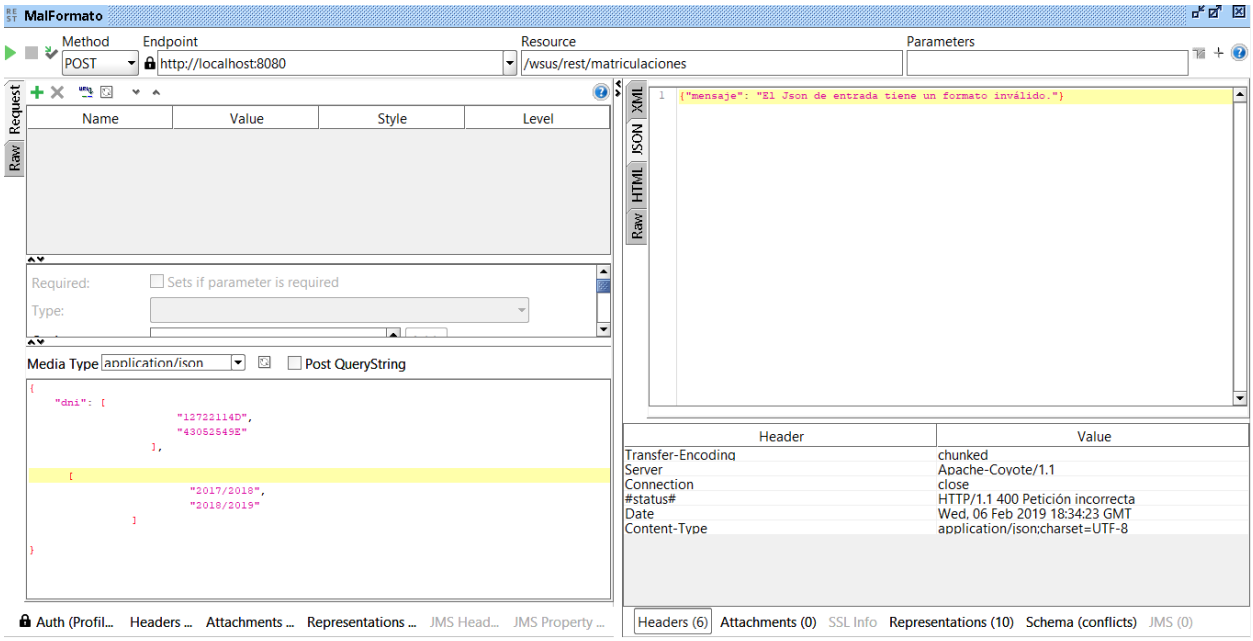

Ilustración 65 - Obtener matriculaciones por DNI (Formato incorrecto)

- **Correcto**

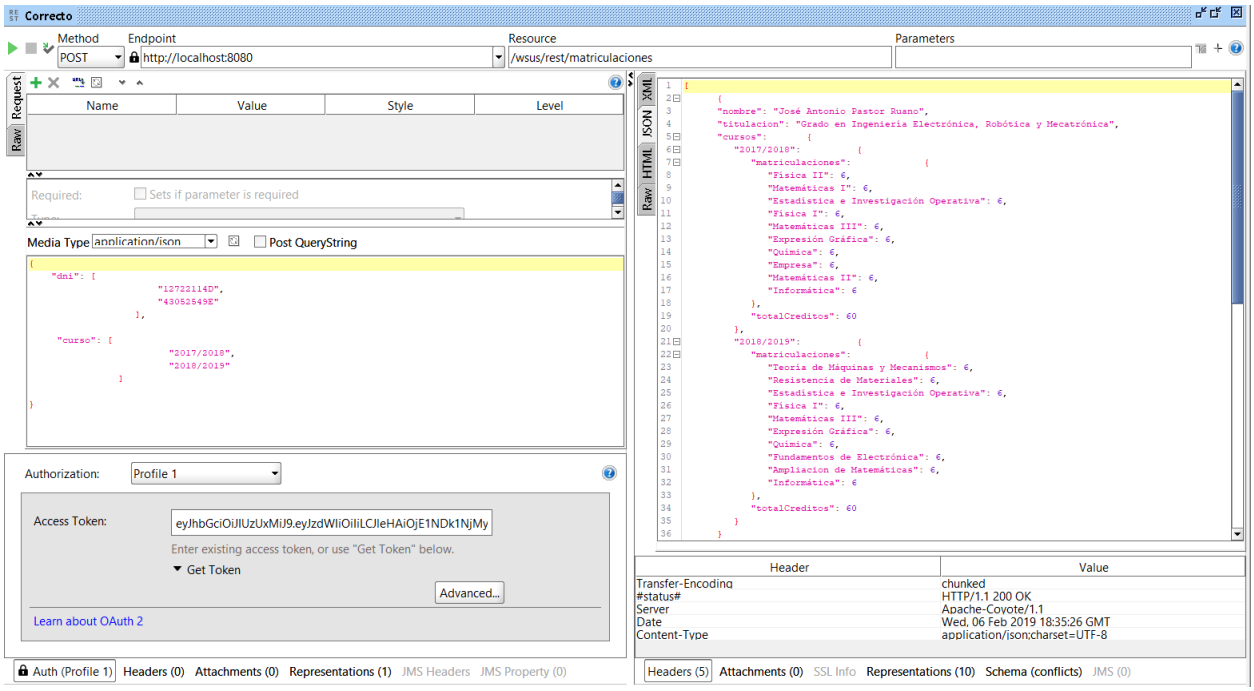

Ilustración 66 - Obtener matriculaciones por DNI (Correcto)

5.2.6 Obtener resumen por DNI del alumno y curso académico

- **Token vacío**

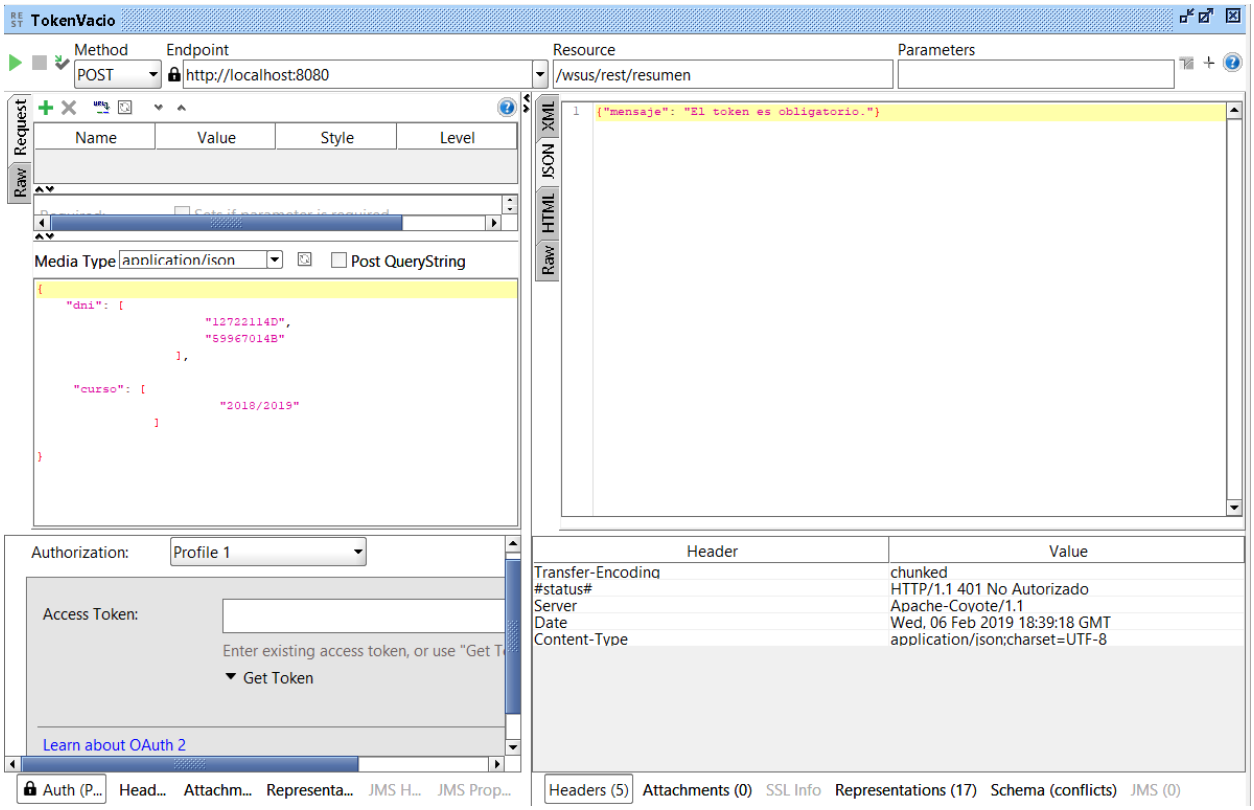

Ilustración 67 – Obtener resumen por DNI y curso académico (Token vacío)

- **Token incorrecto**

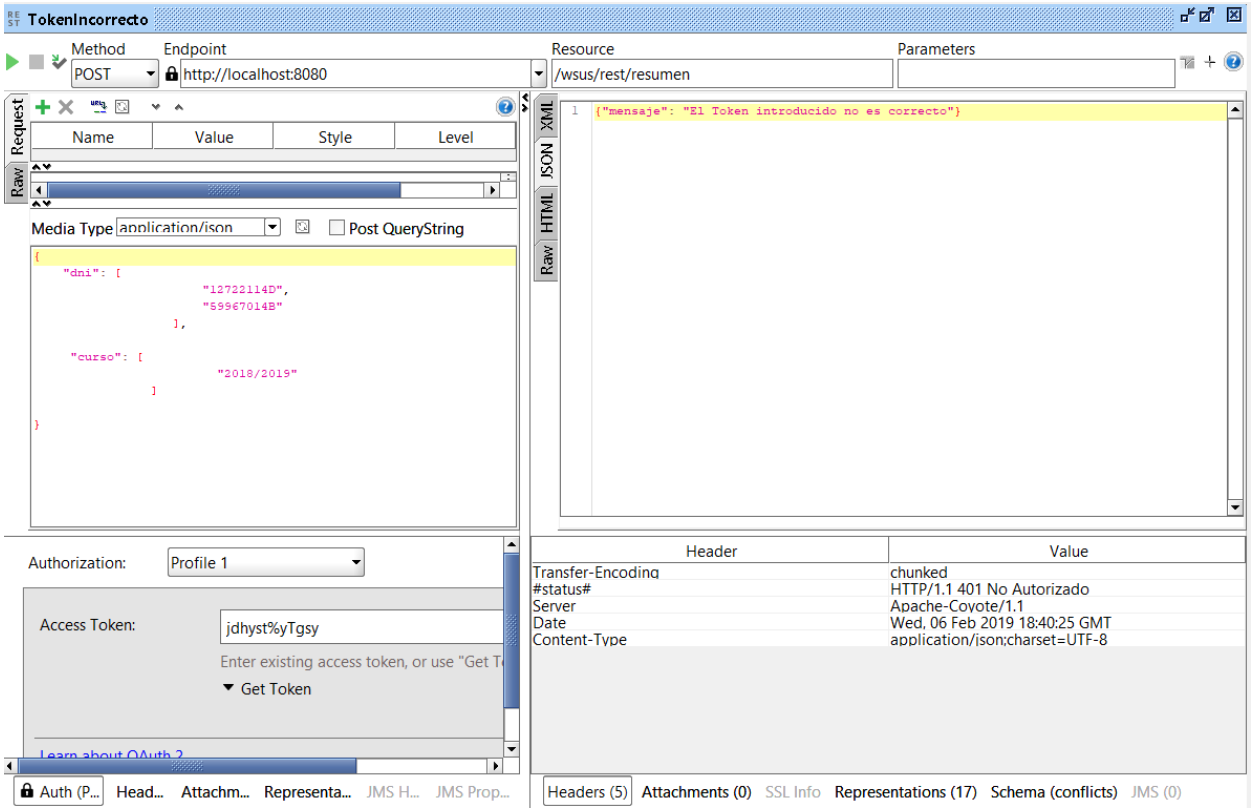

Ilustración 68 - Obtener resumen por DNI y curso académico (Token incorrecto)

- **Token expirado**

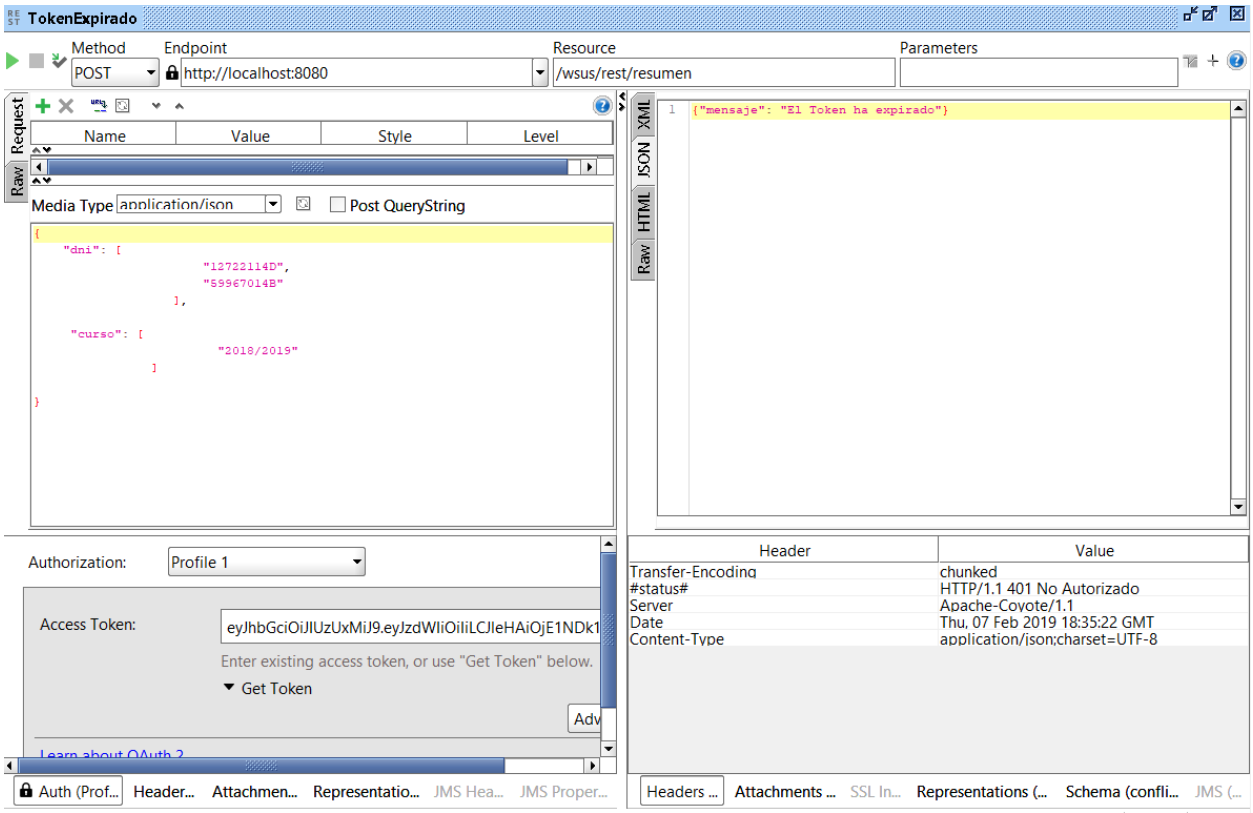

Ilustración 69 - Obtener resumen por DNI y curso académico (Token expirado)

- **Cuerpo vacío**

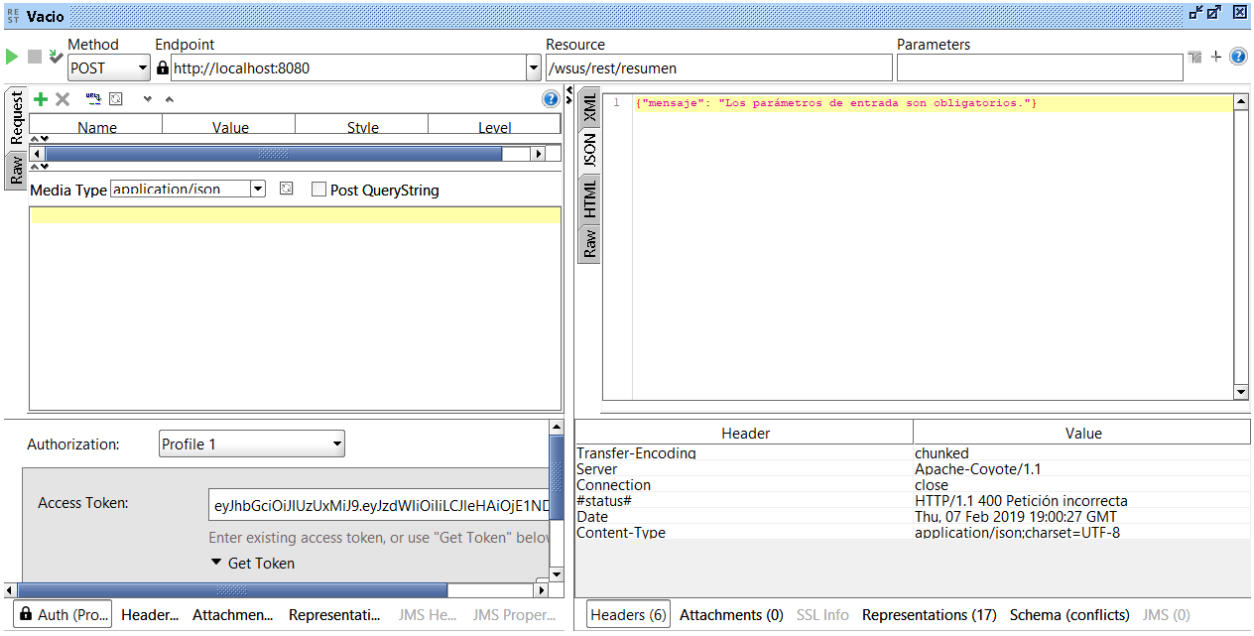

Ilustración 70 - Obtener resumen por DNI y curso académico (Cuerpo vacío)

- **Formato incorrecto**

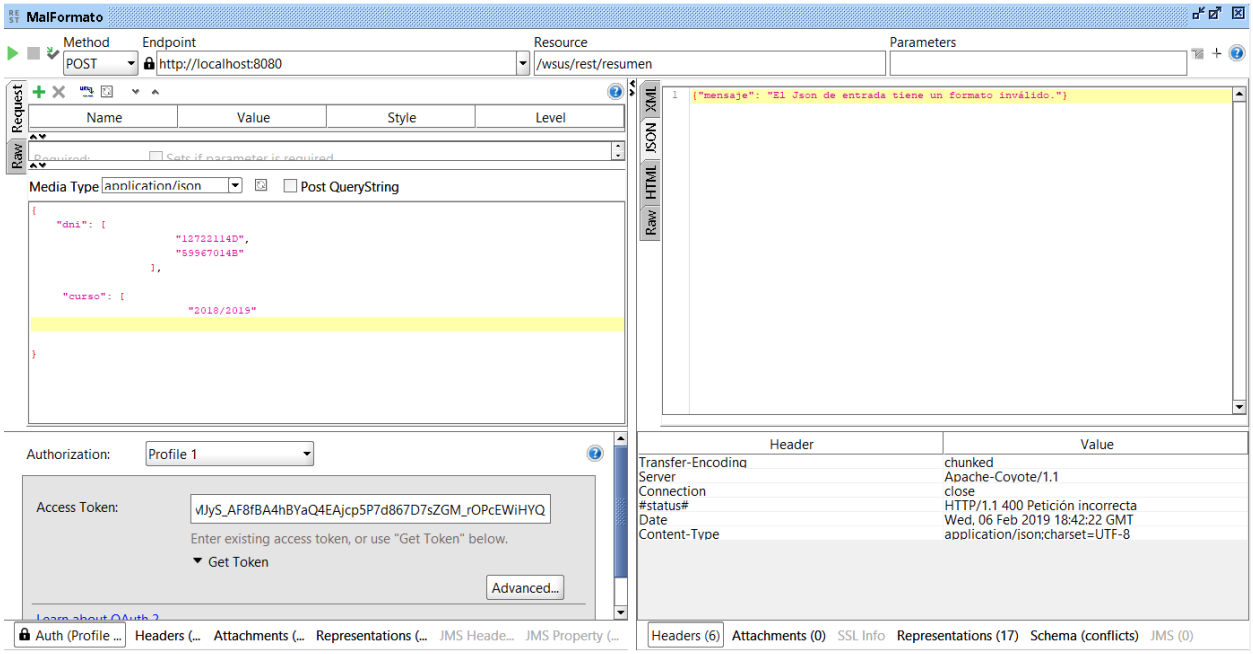

Ilustración 71 - Obtener resumen por DNI y curso académico (Formato incorrecto)

- **Correcto**

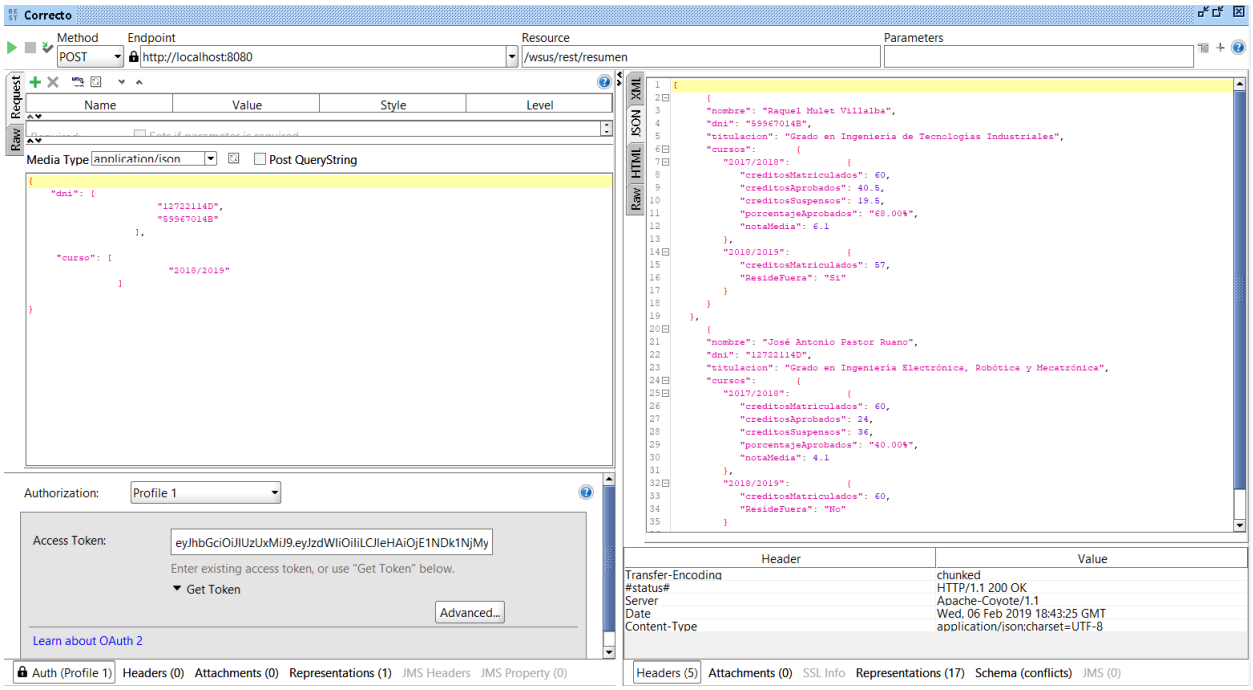

Ilustración 72 - Obtener resumen por DNI y curso académico (Correcto)

5.2.7 Obtener titulaciones por nombre del centro

- **Token vacío**

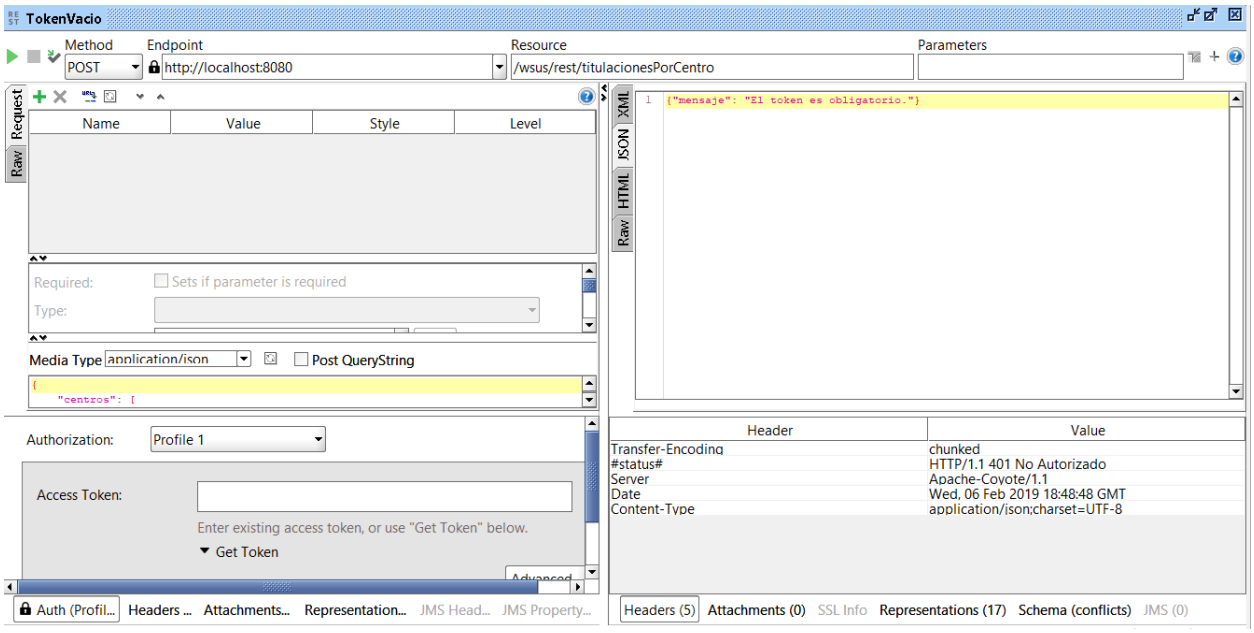

Ilustración 73 – Obtener titulaciones por nombre del centro (Token vacío)

- **Token incorrecto**

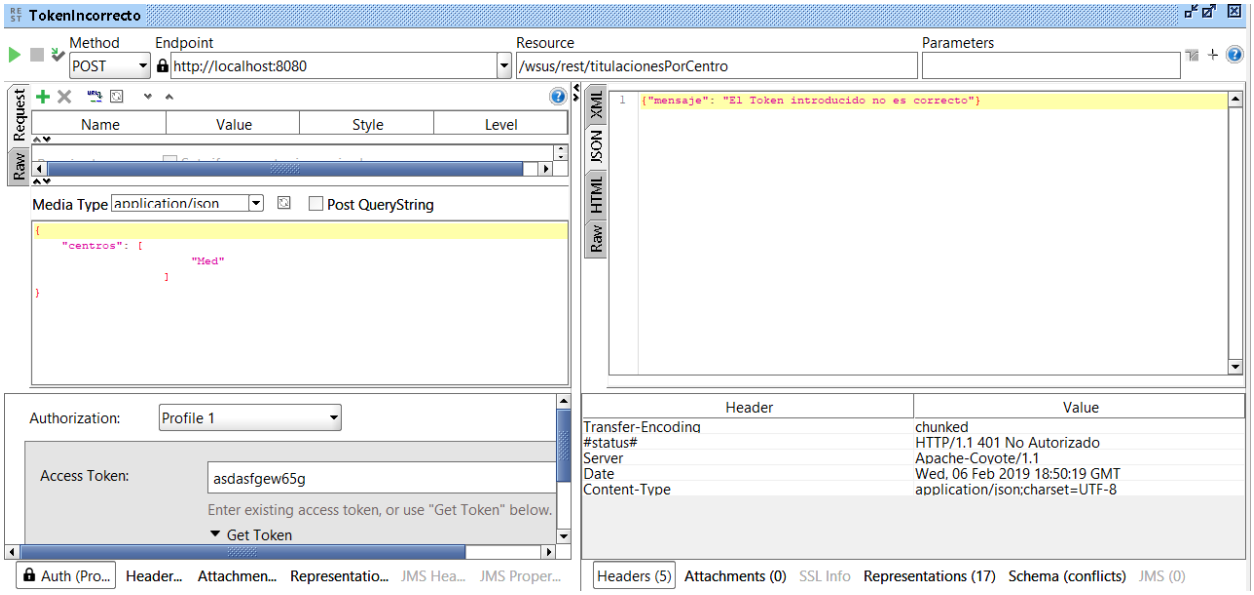

Ilustración 74 - Obtener titulaciones por nombre del centro (Token incorrecto)

- **Token expirado**

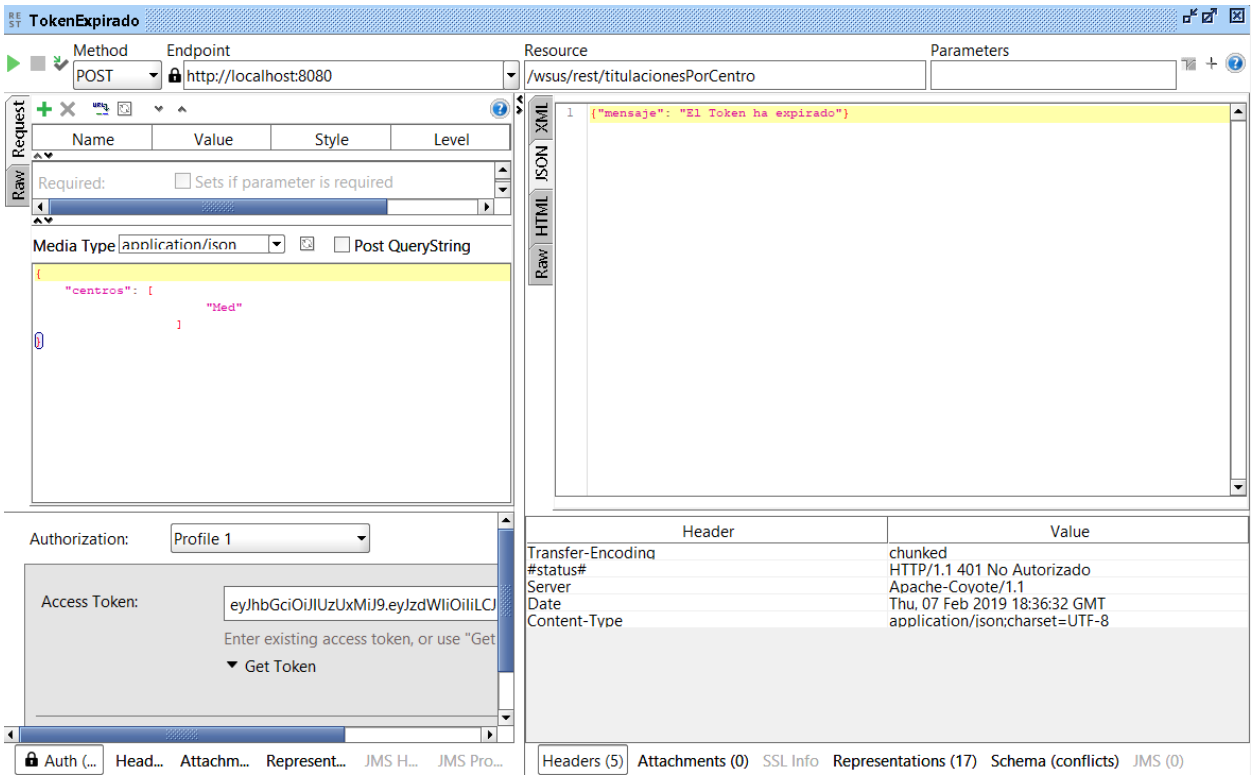

Ilustración 75 - Obtener titulaciones por nombre del centro (Token expirado)

- **Cuerpo vacío**

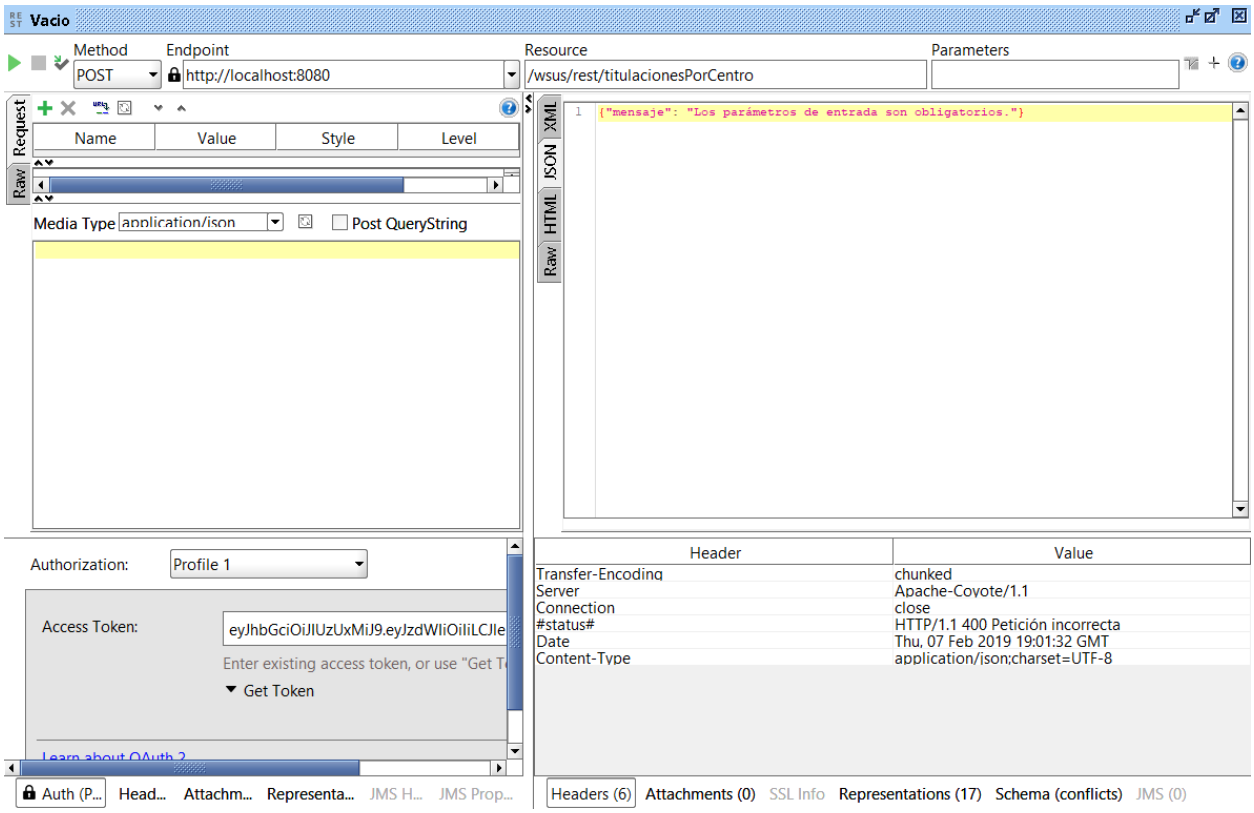

Ilustración 76 - Obtener titulaciones por nombre del centro (Cuerpo vacío)

- **Formato incorrecto**

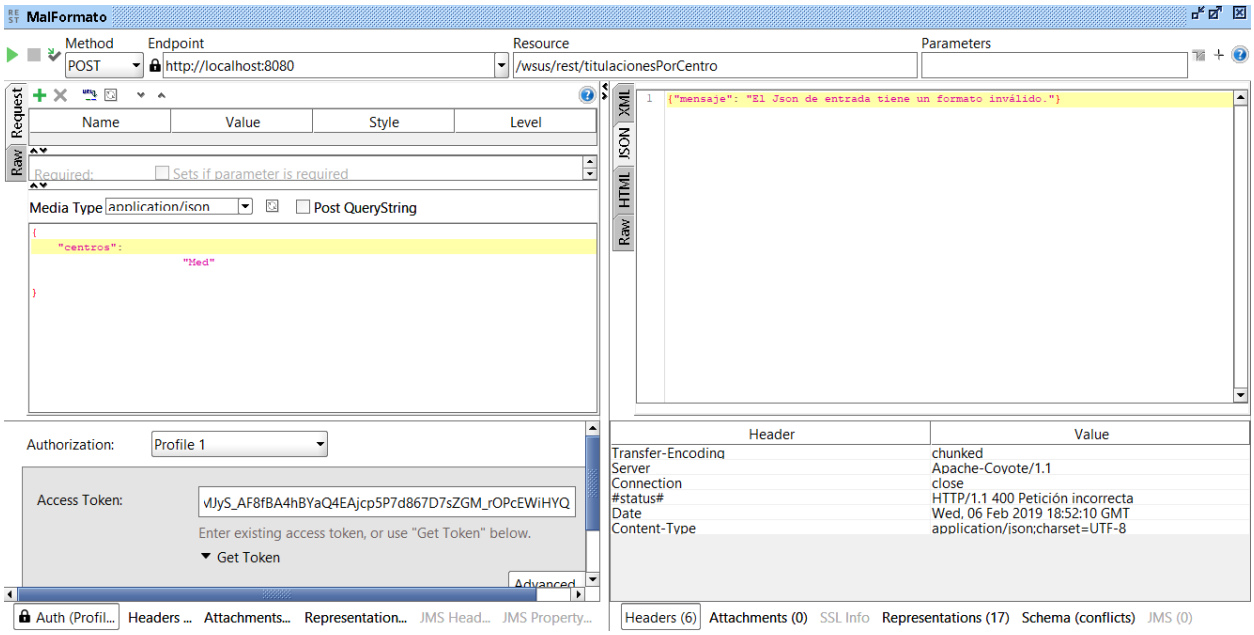

Ilustración 77 - Obtener titulaciones por nombre del centro (Formato incorrecto)

- **Correcto**

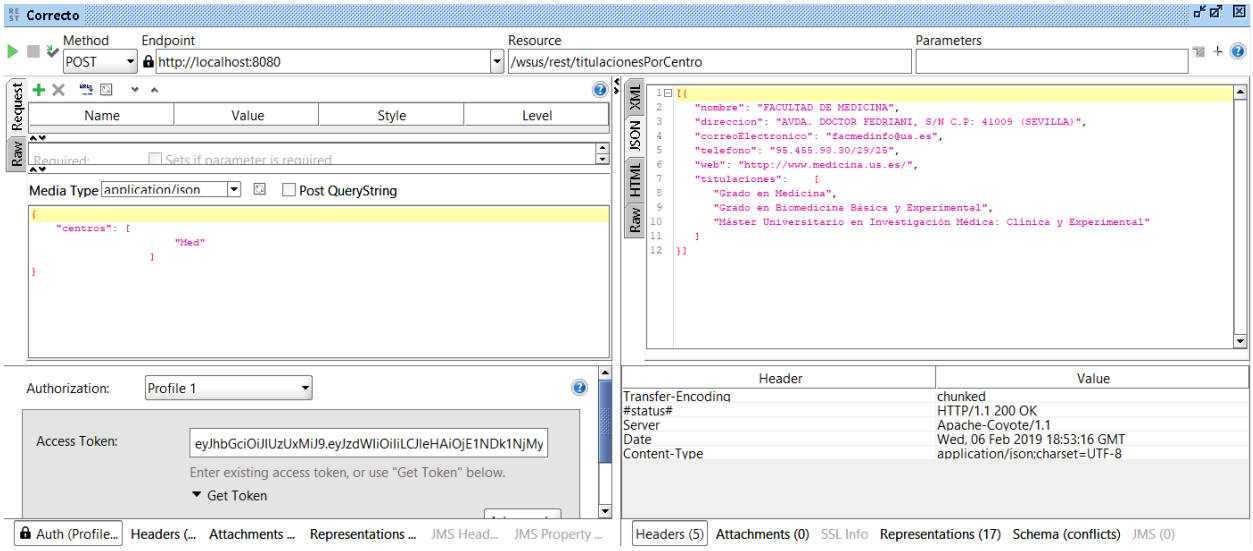

Ilustración 78 - Obtener titulaciones por nombre del centro (Correcto)

5.2.8 Obtener titulaciones por nombre de titulaciones

- **Token vacío**

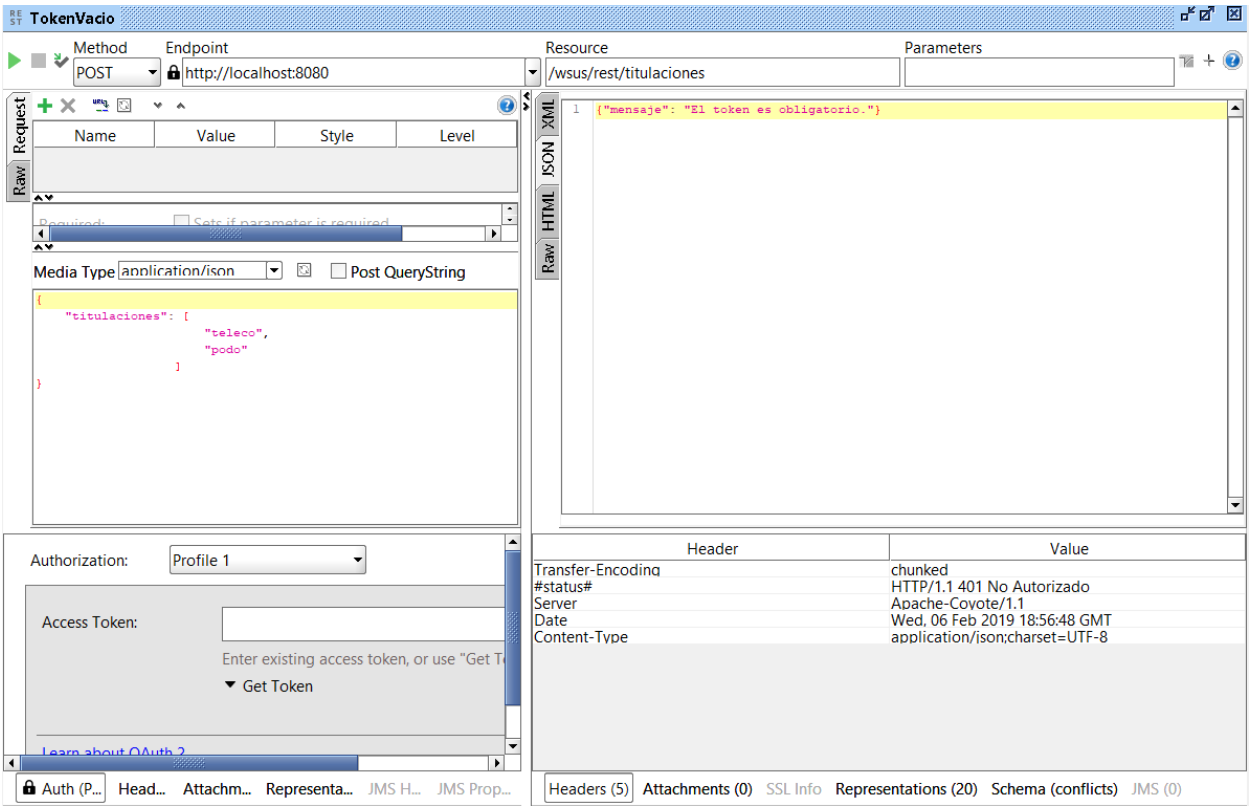

Ilustración 79 – Obtener titulaciones por nombre de titulaciones (Token vacío)

- **Token incorrecto**

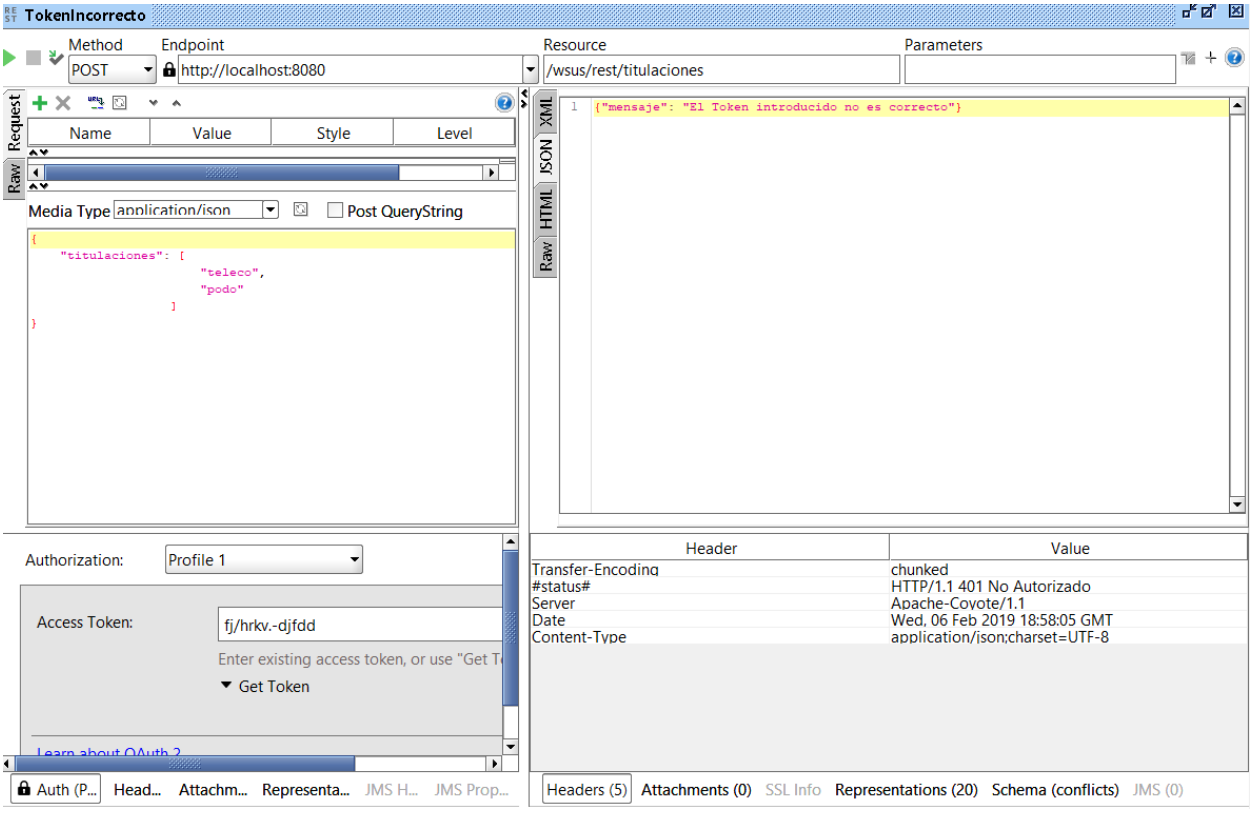

Ilustración 80 - Obtener titulaciones por nombre de titulaciones (Token incorrecto)

- **Token expirado**

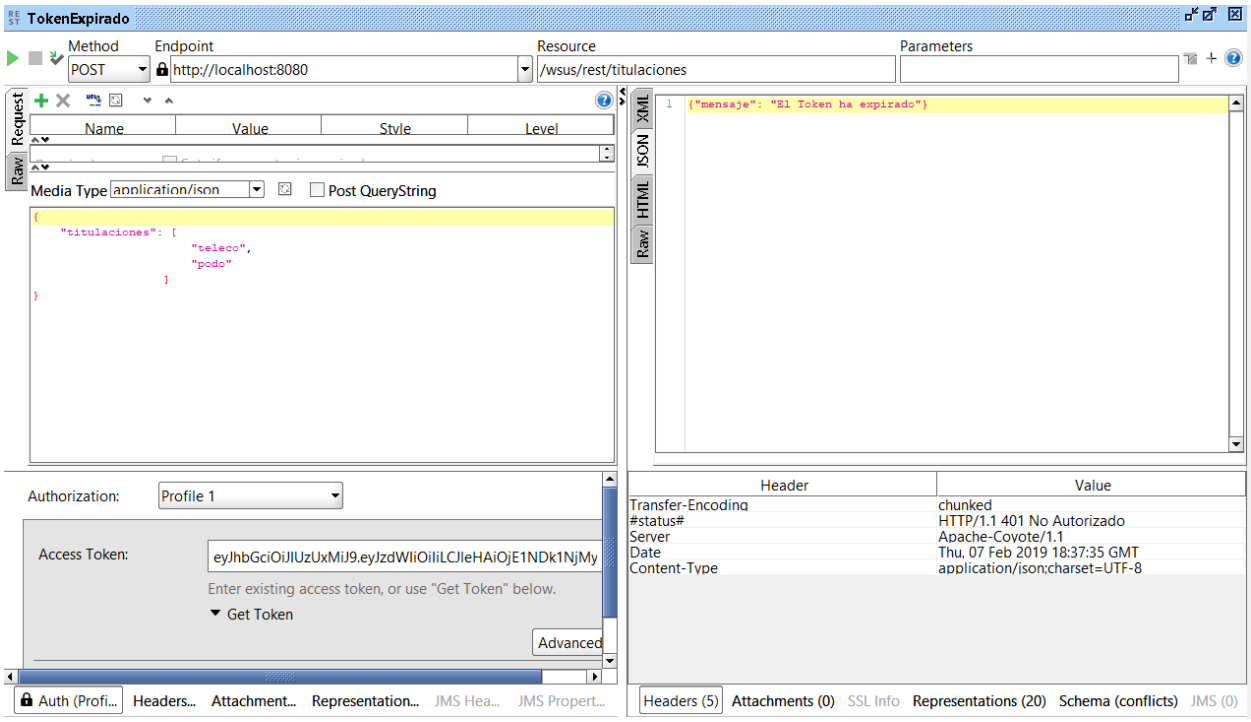

Ilustración 81 - Obtener titulaciones por nombre de titulaciones (Token expirado)

- **Cuerpo vacío**

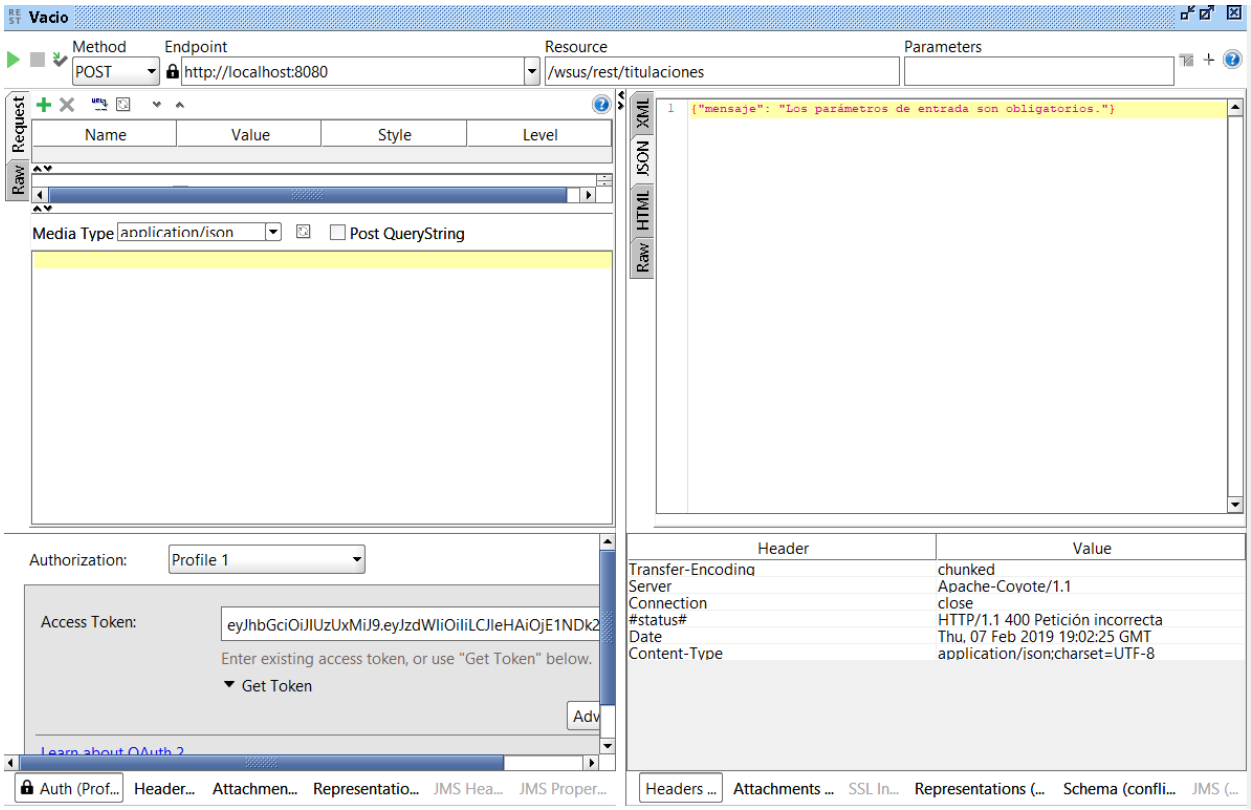

Ilustración 82 - Obtener titulaciones por nombre de titulaciones (Cuerpo vacío)

- **Formato incorrecto**

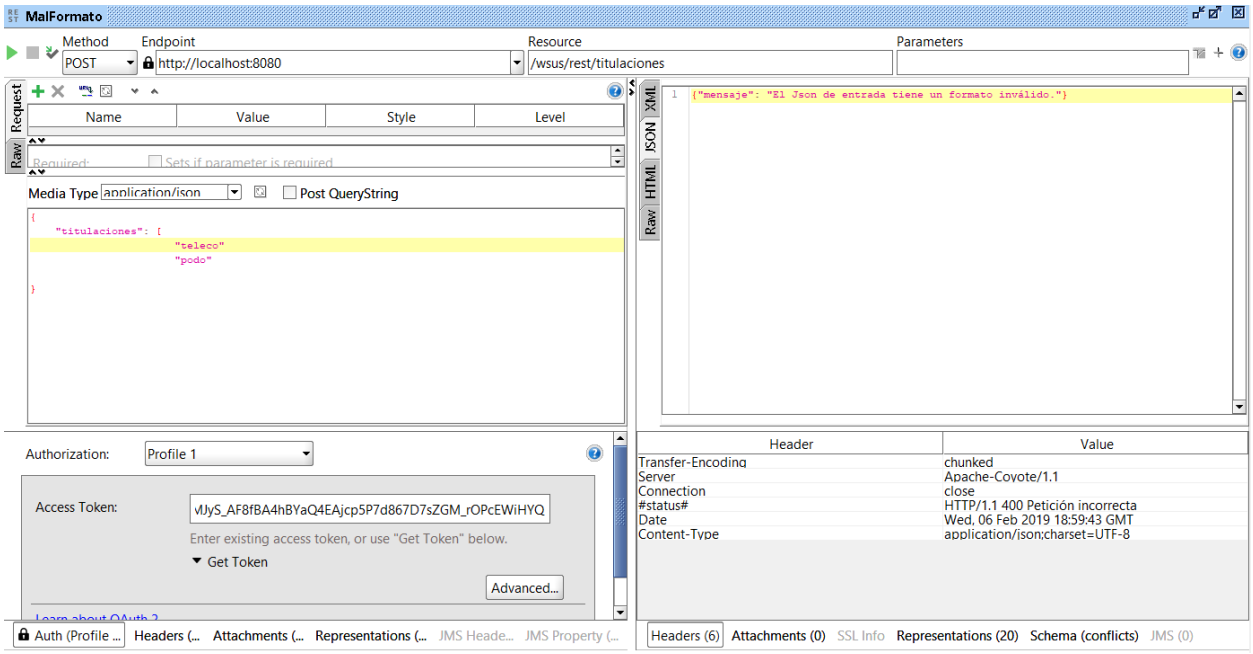

Ilustración 83 - Obtener titulaciones por nombre de titulaciones (Formato incorrecto)

- **Correcto**

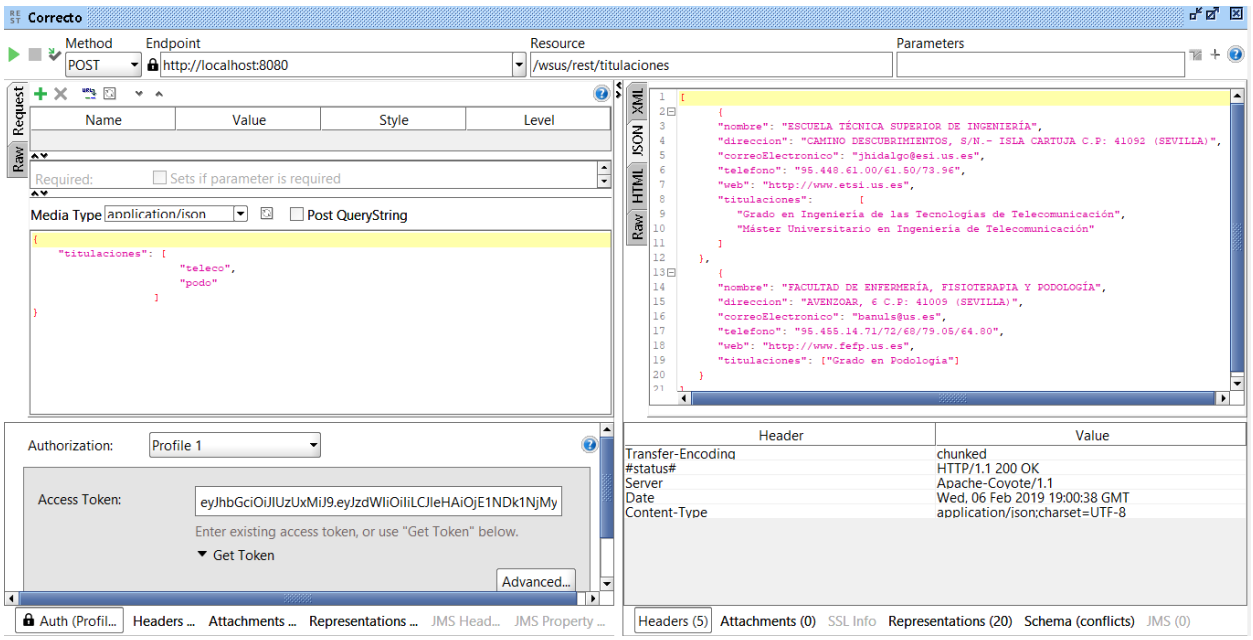

Ilustración 84 - Obtener titulaciones por nombre de titulaciones (Correcto)

En el este capítulo se han detallado cada uno de los casos de pruebas para certificar el correcto funcionamiento del Servicio Web. Se han podido observar el conjunto de respuestas devueltas por cada uno de los recursos ante las distintas situaciones que se pueden dar ante este escenario. El siguiente capítulo está reservado a la valoración del trabajo realizado, revisando los objetivos marcados al comienzo del mismo y justificando las metas conseguidas.

6 CONCLUSIONES

Si tu negocio no está en internet, tu negocio no existe

Bill Gates

En este capítulo de conclusiones me gustaría repasar cada uno de los objetivos marcados al comienzo del trabajo, para poder así evaluar desde una visión global todo lo que se ha definido en él.

La meta principal marcada al comienzo de este trabajo considero que está justificadamente cumplida. Se perseguía poner en práctica los conocimientos obtenidos en la experiencia laboral, ahondar en las tecnologías que he descubierto y materializar un Servicio Web dotado de todas ellas y aplicado al ámbito académico.

Para conseguir esta meta principal antes se han tenido que cubrir también cada uno de los objetivos que se citan al comienzo. Si hacemos un repaso por cada uno de ellos podremos hacer un juicio de valor.

Se ha conseguido definir un modelo de datos relacional para su consumo por parte del Servicio Web. Para éste se han declarado un conjunto de recursos, de manera que han quedado recogidos los puntos principales señalados en los objetivos. Estos recursos han sido definidos en cuanto a su funcionalidad, formato de la petición y formato de respuesta.

Se han generado además un conjunto de datos ajustados a la realidad y se han volcado en el modelo de tablas definido para una base de datos Oracle mediante scripts de inserción.

A la hora de dotar de una estructura al Servicio Web se ha seguido el objetivo marcado, haciendo uso del software Maven. Con éste, se ha podido partir de un arquetipo adaptado a las necesidades que se exponían, cubriendo y configurando además todas sus dependencias y parámetros en el archivo POM.

El Framework de Spring ha constituido la "red neuronal" del Servicio Web. Ha dispuesto de un servlet de escucha que atiende y redirecciona todas las peticiones haciéndoselas llegar al controlador para que ejecute el método definido en cada recurso. Además, ha dado forma a toda la capa de negocio, declarando servicios y creando beans para los mismos, permitiendo así la comunicación entre todas las partes del aplicativo. Con Spring además se ha definido la conexión a base de datos, declarando un bean sessionFactory que ha sido explotado por Hibernate.

Con el uso de Hibernate se cumple otro de los objetivos marcados al inicio de este trabajo. Con esto framework se ha conseguido crear las entidades que relacionan el modelo orientado a objeto de Java con el modelo relacional de base de datos definido en Oracle. Además, se ha hecho uso Criteria para realizar consultas siempre orientadas a los objetos que se manejan dentro del código y encargándose Hibernate de formar las querys a ejecutar en el modelo relacional.

Mediante JSON Web Token se ha cubierto el objetivo de dotar de una capa de seguridad al aplicativo. Se han declarado recursos y métodos para la obtención de tokens por parte del usuario y su posterior validación por parte del sistema. Para la generación del token de acceso, el usuario debe aportar la clave privada en su petición, el sistema reconoce y valida esta clave y facilita al usuario un token de acceso codificado para que éste sea incorporado en la cabecera de las peticiones a los recursos para la obtención de datos. Este token de acceso se ha configurado con un tiempo de 24 horas de expiración, a partir del cual el token quedará inválido y el usuario deberá volver a solicitar uno nuevo.

Para certificar el correcto funcionamiento del Servicio Web, se ha acompañado este desarrollo de una batería de pruebas detallada para cada uno de los recursos, obteniéndose los resultados esperados.

REFERENCIAS

[1] Wikipedia, «Java Remote Method Invocation,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_Remote_Method_Invocation. [Último acceso: 18 Marzo 2019].

[2] javaTpoint, «RMI (Remote Method Invocation),» [En línea]. Available: https://www.javatpoint.com/RMI. [Último acceso: 20 Marzo 2019].

[3] Wikipedia, «CORBA,» [En línea]. Available: https://es.wikipedia.org/wiki/CORBA. [Último acceso: Marzo 18 2019].

[4] D. Lázaro, «Introducción a los Web Services,» [En línea]. Available: https://diego.com.es/introduccion-a-los-web-services. [Último acceso: 5 Febrero 2019].

[5] desarrolloweb, «Historia de los Web Services,» [En línea]. Available: https://desarrolloweb.com/articulos/1883.php. [Último acceso: 5 Febrero 2019].

[6] Wikipedia, «Modelo de Objetos de Componentes Distribuidos,» [En línea]. Available: https://es.wikipedia.org/wiki/Modelo_de_Objetos_de_Componentes_Distribuidos. [Último acceso: 5 Febrero 2019].

[7] M. Saffirio, «¿Qué son los Web Services?,» [En línea]. Available: https://msaffirio.wordpress.com/2006/02/05/%C2%BFque-son-los-web-services/. [Último acceso: 5 Febrero 2019].

[8] Wikipedia, «Servicio web,» [En línea]. Available: https://es.wikipedia.org/wiki/Servicio_web. [Último acceso: 5 Febrero 2019].

[9] Medium, «¿Qué es un Servicio Web?,» [En línea]. Available: https://medium.com/grupo-carricay/qu%C3%A9-es-un-servicio-web-510be516863. [Último acceso: 3 Marzo 2019].

[10] desarrolloweb, «SOAP (Simple Object Access Protocol),» [En línea]. Available: https://desarrolloweb.com/articulos/1557.php. [Último acceso: 3 Marzo 2019].

[11] Wikipedia, «Simple Object Access Protocol,» [En línea]. Available: https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol. [Último acceso: 3 Marzo 2019].

[12] Wikipedia, «Llamada a procedimiento remoto,» [En línea]. Available: https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto. [Último acceso: 3 Marzo 2019].

[13] F. J. Sierra Ceballos, Java Interfaces gráficas y aplicaciones para internet 4ª Edición, Ra-Ma®, 2015.

[14] Wikipedia, «Arquitectura orientada a servicios,» [En línea]. Available: https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios. [Último acceso: 5 Abril 2019].

[15] Wikipedia, «Transferencia de Estado Representacional,» [En línea]. Available: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional. [Último acceso: 5 Abril 2019].

[16] Nutrients, «REST Services and Reports,» [En línea]. Available: https://nutrients.readthedocs.io/en/latest/02_dir/$_02-core-12-rest-prov.html. [Último acceso: 5 Abril 2019].

[17] arquitecturajava, «¿ Que es REST ?,» [En línea]. Available: https://www.arquitecturajava.com/que-es-rest/. [Último acceso: 5 Abril 2019].

[18] E. Etayo, «WebServices: SOAP vs REST: ¿Cual usar en cada caso?,» [En línea]. Available: http://estebanetayo.es/webservices-soap-vs-rest-%C2%BFcual-usar-en-cada-caso. [Último acceso: abril 5 2019].

[19] «Generador aleatorio de base de datos,» proinf, [En línea]. Available: https://proinf.net/permalink/generador_aleatorio_de_base_de_datos. [Último acceso: 16 Agosto 2018].

[20] «Oracle Database 11g: Información,» ORACLE, [En línea]. Available: https://www.oracle.com/technetwork/es/documentation/317497-esa.pdf. [Último acceso: 20 Agosto 2018].

[21] «Cambiar el juego de carácteres de la base de datos,» DATAPRIX, [En línea]. Available: http://www.dataprix.com/blogs/il-masacratore/oracle10g-cambiar-juego-car-cteres-base-datos. [Último acceso: Agosto 21 2018].

[22] «Change NLS_NCHAR_CHARACTERSET,» ORACLE, [En línea]. Available: https://community.oracle.com/thread/308100. [Último acceso: 21 Agosto 2018].

[23] «Configuración de una base de datos Oracle,» IBM, [En línea]. Available: https://www.ibm.com/support/knowledgecenter/es/SSYMRC_6.0.3. [Último acceso: 19 Agosto 2018].

[24] «Setting the NLS_LANG Environment Variable for Oracle Databases,» ORACLE, [En línea]. Available: https://docs.oracle.com/cd/E12102_01/books/AnyInstAdm784/AnyInstAdmPreInstall18.html. [Último acceso: 21 Agosto 2018].

[25] «Apache Maven,» Maven, [En línea]. Available: https://maven.apache.org/. [Último acceso: Marzo 22 2019].

[26] Adictos al trabajo, «Maven, nunca antes resultó tan fácil compilar, empaquetar,» [En línea]. Available: https://www.adictosaltrabajo.com/2006/09/19/maven/. [Último acceso: 5 Septiembre 2018].

[27] GENBETA, «¿Qué es Maven?,» [En línea]. Available: https://www.genbeta.com/desarrollo/que-es-maven. [Último acceso: 23 Marzo 2019].

[28] jarroba, «Maven,» [En línea]. Available: https://jarroba.com/maven/. [Último acceso: 23 Marzo 2019].

[29] Wikipedia, «Maven,» [En línea]. Available: https://es.wikipedia.org/wiki/Maven. [Último acceso: 22 Marzo 2019].

[30] davidmarco, «Introducción a JPA 2.0 (I),» [En línea]. Available: http://www.davidmarco.es/articulo/introduccion-a-jpa-2-0-i. [Último acceso: 25 Agosto 2018].

[31] davidmarco, «Introducción a JPA 2.0 (II),» [En línea]. Available: http://www.davidmarco.es/articulo/introduccion-a-jpa-2-0-ii. [Último acceso: 25 Agosto 2018].

[32] HIBERNATE, «Hibernate,» [En línea]. Available: https://hibernate.org/. [Último acceso: 18 Febrero 2019].

[33] HIBERNATE, «Consultas por Criterios,» [En línea]. Available: https://docs.jboss.org/hibernate/orm/3.3/reference/es-ES/html/querycriteria.html. [Último acceso: 30 Agosto 2018].

[34] Spring, «Guides,» [En línea]. Available: https://spring.io/guides. [Último acceso: 3 Abril 2019].

[35] arquitecturajava, «Spring,» [En línea]. Available: https://www.arquitecturajava.com/categoria/spring/. [Último acceso: 11 Mayo 2019].

[36] Universidad de Alicante, «Introducción a Spring,» [En línea]. Available: http://www.jtech.ua.es/j2ee/publico/spring-2012-13/sesion01-apuntes.html. [Último acceso: 19 Octubre 2018].

[37] Universidad de Valencia, «Desarrollando una aplicación Spring Framework MVC v4 + JPA paso a paso,» [En línea]. Available: https://www.uv.es/grimo/teaching/SpringMVCv4PasoAPaso/. [Último acceso: 5 Septiembre 2018].

[38] Tutor de Programación, «Spring Configuración de Beans,» [En línea]. Available: http://acodigo.blogspot.com/2017/02/spring-configuracion-de-beans.html. [Último acceso: 22 Agosto 2018].

[39] Programar en cloud, «Cómo Implementar La Seguridad De Tu API 1,» [En línea]. Available: https://programar.cloud/post/como-implementar-la-seguridad-en-tu-api-parte-1/. [Último acceso: 26 Agosto 2018].

[40] Programar en cloud, «Cómo Implementar La Seguridad De Tu API 2,» [En línea]. Available: https://programar.cloud/post/como-implementar-la-seguridad-en-tu-api-parte-2/. [Último acceso: 26 Agosto 2018].

[41] C. Azustre, «¿Qué es la autenticación basada en Token?,» [En línea]. Available: https://carlosazaustre.es/que-es-la-autenticacion-con-token/. [Último acceso: 6 Septiembre 2018].

[42] JWT, «Introduction to JSON Web Tokens,» [En línea]. Available: https://jwt.io/introduction/. [Último acceso: 6 Septiembre 2018].

[43] stackoverflow, «HTTP POST using JSON in Java,» [En línea]. Available: https://stackoverflow.com/questions/7181534/http-post-using-json-in-java. [Último acceso: 8 Septiembre 2018].

[44] SMARTBEAR, «https://www.soapui.org/,» [En línea]. [Último acceso: 8 Septiembre 2018].

ANEXOS

En los siguientes anexos, se tratan diferentes aspectos importantes en la replicación del contenido de este proyecto, generalmente de carácter técnico que quedan fuera del diseño del sistema y pertenecen a la implementación concreta que se ha utilizado. Por ello, será en estos anexos donde se encontrará el código utilizado.

1. Codificación de base de datos

Para solucionar problemas de codificación a la hora de crear las tablas e introducir los datos correctamente en UTF-8, se ejecutaron las siguientes líneas de código.

```
sqlplus / as sysdba
STARTUP MOUNT exclusive RESTRICT
ALTER SYSTEM ENABLE RESTRICTED SESSION;
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
ALTER SYSTEM SET AQ_TM_PROCESSES=0;
ALTER DATABASE OPEN ;
update props$
set value$='UTF8'
where name in ('NLS_CHARACTERSET', 'NLS_NCHAR_CHARACTERSET');
commit;
SHUTDOWN IMMEDIATE;
```

2. Script de creación de tabla

En este punto se pone de manifiesto un ejemplo de creación de tabla de los que se han ejecutado en el proyecto.

```
SPOOL WSUS.07_CREATE_TABLE_WSUS_MATR_MATRICULACIONES.LOG;
SET serveroutput on size 1000000;
SET linesize 1000;
PROMPT "SE CREA LA TABLA WSUS_MATR_MATRICULACIONES"

CREATE TABLE WSUS_MATR_MATRICULACIONES(
     MATR_CS_CODIGO NUMBER(12,0) NOT NULL,
     ALUM_CS_CODIGO NUMBER(12,0) NOT NULL,
     ASIG_CS_CODIGO NUMBER(12,0) NOT NULL,
     CURS_CS_CODIGO NUMBER(12,0) NOT NULL,
     MATR_FH_CREACION DATE NOT NULL,
     MATR_FH_MODIFICACION DATE NOT NULL,
     MATR_LI_CREACION VARCHAR2(100) NOT NULL,
     MATR_LI_MODIFICACION VARCHAR2(100) NOT NULL
);
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.MATR_CS_CODIGO IS 'Clave primaria
generada a partir de una secuencia'
```

```
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.ALUM_CS_CODIGO IS 'Referencia a la tabla
WSUS_ALUM_ALUMNOS'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.ASIG_CS_CODIGO IS 'Referencia a la tabla
WSUS_ASIG_ASIGNATURAS'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.CURS_CS_CODIGO IS 'Referencia a la tabla
WSUS_CURS_CURSOS'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.MATR_FH_CREACION IS 'Campo auditoría:
fecha creación'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.MATR_LI_CREACION IS 'Campo auditoría:
usuario creación'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.MATR_FH_MODIFICACION IS 'Campo auditoría:
fecha modificación'
;
COMMENT ON COLUMN WSUS_MATR_MATRICULACIONES.MATR_LI_MODIFICACION IS 'Campo auditoría:
usuario modificación'
;

CREATE UNIQUE INDEX WSUS_MATR_PK01 ON WSUS_MATR_MATRICULACIONES(MATR_CS_CODIGO);

ALTER TABLE WSUS_MATR_MATRICULACIONES ADD CONSTRAINT WSUS_MATR_PK01 PRIMARY KEY
(MATR_CS_CODIGO) USING INDEX;

ALTER TABLE WSUS_MATR_MATRICULACIONES ADD FOREIGN KEY (ALUM_CS_CODIGO) REFERENCES
WSUS_ALUM_ALUMNOS(ALUM_CS_CODIGO);
ALTER TABLE WSUS_MATR_MATRICULACIONES ADD FOREIGN KEY (ASIG_CS_CODIGO) REFERENCES
WSUS_ASIG_ASIGNATURAS(ASIG_CS_CODIGO);
ALTER TABLE WSUS_MATR_MATRICULACIONES ADD FOREIGN KEY (CURS_CS_CODIGO) REFERENCES
WSUS_CURS_CURSOS(CURS_CS_CODIGO);

CREATE SEQUENCE WSUS_MATR_SC01
     START WITH 1
     MAXVALUE 999999999999999999999999999
     MINVALUE 1
     NOCYCLE
     CACHE 20
     NOORDER;

--DROP TRIGGER AUDITORIA;

CREATE OR REPLACE TRIGGER WSUS_MATR_AUDITORIA
 BEFORE INSERT OR UPDATE
 ON WSUS_MATR_MATRICULACIONES  FOR EACH ROW
DECLARE

   v_username varchar2(50);
BEGIN
  IF UPDATING THEN
    IF  :new.MATR_LI_MODIFICACION is null then
     -- Busca el usuario que actuliza la tabla
     SELECT user INTO v_username
     FROM dual;
     -- Actualiza la fecha de modificacion con la fecha de base de datos
     -- Actualiza el usuario que modifica la tabla
     :new.MATR_LI_MODIFICACION := v_username;
```

```
   end if;
   :new.MATR_FH_MODIFICACION:= sysdate;
  ELSIF INSERTING THEN
    IF  :new.MATR_LI_CREACION is null then
     -- Busca el usuario que crea la tabla
     SELECT user INTO v_username
     FROM dual;
     -- Actualiza la fecha con la fecha de base de datos
     -- Actualiza el usuario que crea la tabla
     :new.MATR_LI_CREACION := v_username;
     :new.MATR_LI_MODIFICACION := v_username;
   end if;
   :new.MATR_FH_CREACION:= sysdate;
   :new.MATR_FH_MODIFICACION:= sysdate;
  END IF;
END;
/
SHOW ERRORS;


SPOOL OFF;
```

3. Entidad u objeto POJO

En este apartado se presenta el aspecto de la definición de una entidad del modelo, en concreto la entidad de Matriculaciones.java

```
@Entity
@Table(name = "WSUS_MATR_MATRICULACIONES", schema = "TFG")
public class Matriculaciones implements java.io.Serializable {

     /**
      *
      */
     private static final long serialVersionUID = 1L;
     private long matrCsCodigo;
     private Alumnos alumnos;
     private Asignaturas asignaturas;
     private Cursos cursos;
     private Date matrFhCreacion;
     private Date matrFhModificacion;
     private String matrLiCreacion;
     private String matrLiModificacion;

     public Matriculaciones() {
     }

     public Matriculaciones(long matrCsCodigo, Alumnos alumnos,
               Asignaturas asignaturas, Cursos cursos, Date matrFhCreacion,
               Date matrFhModificacion, String matrLiCreacion, String
matrLiModificacion) {
          this.matrCsCodigo = matrCsCodigo;
          this.alumnos = alumnos;
          this.asignaturas = asignaturas;
          this.cursos = cursos;
          this.matrFhCreacion = matrFhCreacion;
          this.matrFhModificacion = matrFhModificacion;
          this.matrLiCreacion = matrLiCreacion;
```

```
          this.matrLiModificacion = matrLiModificacion;
     }

     @Id

     @Column(name = "MATR_CS_CODIGO", unique = true, nullable = false, precision =
12, scale = 0)
     public long getMatrCsCodigo() {
          return this.matrCsCodigo;
     }

     public void setMatrCsCodigo(long matrCsCodigo) {
          this.matrCsCodigo = matrCsCodigo;
     }

     @ManyToOne
     @JoinColumn(name = "ALUM_CS_CODIGO", nullable = false)
     public Alumnos getAlumnos() {
          return this.alumnos;
     }

     public void setAlumnos(Alumnos alumnos) {
          this.alumnos = alumnos;
     }

     @ManyToOne
     @JoinColumn(name = "ASIG_CS_CODIGO", nullable = false)
     public Asignaturas getAsignaturas() {
          return this.asignaturas;
     }

     public void setAsignaturas(Asignaturas asignaturas) {
          this.asignaturas = asignaturas;
     }

     @ManyToOne
     @JoinColumn(name = "CURS_CS_CODIGO", nullable = false)
     public Cursos getCursos() {
          return this.cursos;
     }

     public void setCursos(Cursos cursos) {
          this.cursos = cursos;
     }

     @Temporal(TemporalType.DATE)
     @Column(name = "MATR_FH_CREACION", nullable = false, length = 7)
     public Date getMatrFhCreacion() {
          return this.matrFhCreacion;
     }

     public void setMatrFhCreacion(Date matrFhCreacion) {
          this.matrFhCreacion = matrFhCreacion;
     }

     @Temporal(TemporalType.DATE)
     @Column(name = "MATR_FH_MODIFICACION", nullable = false, length = 7)
     public Date getMatrFhModificacion() {
          return this.matrFhModificacion;
     }
```

```
	public void setMatrFhModificacion(Date matrFhModificacion) {
		this.matrFhModificacion = matrFhModificacion;
	}

	@Column(name = "MATR_LI_CREACION", nullable = false, length = 100)
	public String getMatrLiCreacion() {
		return this.matrLiCreacion;
	}

	public void setMatrLiCreacion(String matrLiCreacion) {
		this.matrLiCreacion = matrLiCreacion;
	}

	@Column(name = "MATR_LI_MODIFICACION", nullable = false, length = 100)
	public String getMatrLiModificacion() {
		return this.matrLiModificacion;
	}

	public void setMatrLiModificacion(String matrLiModificacion) {
		this.matrLiModificacion = matrLiModificacion;
	}

}
```

4. Controlador

En este apartado se ejemplifica el controlador del servicio web, incluyendo únicamente uno de sus recursos, concretamente el de obtener matriculaciones por alumnos y curso.

```
@Controller
public class RestController {

	private final Logger logger = LoggerFactory.getLogger(RestController.class);

	public static final ObjectMapper JSON_MAPPER = new ObjectMapper();

	@Autowired(required = true)
	private RestService restService;

	@Autowired
	private JwtTokenUtil jwtTokenUtil;

	@Autowired
	 private TimeProvider timeProvider;
/**
	 * Servicio que devuelve las matriculaciones de aquellos alumnos que cumplan
con los dni y cursos que se pasan como parámetros
	 * @param request
	 * @param datos
	 * @return
	 * @throws AppException
	 */
	@RequestMapping(value = RestURIConstantes.GET_MATRICULACIONES, method =
RequestMethod.POST)
	@ResponseBody
	public ResponseEntity<MappingJacksonValue>
getMatriculacionesPorAlumnosCurso(HttpServletRequest request,
@RequestBody(required=false) String datos) throws AppException {
```

```
          MappingJacksonValue value = null;
          HttpStatus status = null;
          try {

               //Comprobamos Token
               ResponseEntity<String> seguridad = comprobarToken(request);

               if(seguridad != null){
                    if(seguridad.getStatusCode().equals(HttpStatus.OK)){
                         if(datos != null && !datos.isEmpty()){

                              //Realizamos la consulta
                         try {
                                   JSONObject jsonEntrada= new JSONObject(datos);
                                   @SuppressWarnings("unchecked")
                                   LinkedHashMap<String, List<String>> mapa=
JSON_MAPPER.readValue(jsonEntrada.toString(), LinkedHashMap.class);
                                   value = new
MappingJacksonValue(restService.obtenerMatriculaciones(mapa.get("dni"),mapa.get("curs
o")));
                         } catch (AppException e) {
                              throw new AppException(AppException.ERROR_10,
ConstantesApi.ERROR_OBTENER_DATOS);
                         }catch (Exception e){
                              throw new AppException(AppException.ERROR_5,
ConstantesApi.EXEP_JSON_INCORRECTO);
                         }
                         //Comprobamos resultados de al consulta para generar
respuesta
                         if(value.getValue() != null &&
!value.getValue().toString().isEmpty()){
                         status = HttpStatus.OK;
                         }

                         else{
                         status = HttpStatus.NO_CONTENT;
                         }

                         }else{
                              value = mergeError(ConstantesApi.EXEP_DATOS_VACIOS);
                              status =  HttpStatus.BAD_REQUEST;
                         }
                    }
                    else if
(seguridad.getStatusCode().equals(HttpStatus.NOT_ACCEPTABLE)){
                         value = mergeError(ConstantesApi.TOKEN_EXPIRADO);
                         status =  HttpStatus.UNAUTHORIZED;
                    } else {
                         value = mergeError(ConstantesApi.TOKEN_ERRONEO);
                         status = HttpStatus.UNAUTHORIZED;
                    }

               } else {
                    value = mergeError(ConstantesApi.EXEP_TOKEN_VACIO);
                    status = HttpStatus.UNAUTHORIZED;
               }


          } catch (Exception e) {
               logger.debug(e.getMessage());
```

```
                value = mergeError(e.getMessage());
                status =  HttpStatus.BAD_REQUEST;
                enviarRespuesta(value, status);
           }

           return enviarRespuesta(value, status);
     }


/**
      * Método para enviar la respuesta
      * @param value
      * @param status
      */
     private ResponseEntity<MappingJacksonValue> enviarRespuesta(MappingJacksonValue
value, HttpStatus status){
           HttpHeaders headers = new HttpHeaders();
           ResponseEntity<MappingJacksonValue> response = null;
           headers.setContentType(new MediaType(AppConstantes.APP,
AppConstantes.JSON, Charset.forName(AppConstantes.ENCOD_UTF8)));
           response = new ResponseEntity<MappingJacksonValue>(value, headers,
status);
           return response;
     }

     /**
     /**
      * Maneja los errores de la respuesta del REST
      * @param codigo
      * @param mensaje
      * @return
      */
     private MappingJacksonValue mergeError(String mensaje) {
           ErrorJson error = new ErrorJson();
           error.setMensaje(mensaje);
           return new MappingJacksonValue(error);
     }
}
```

5. Recurso de generación de token

```
/**
      * Metodo utilizado para la generacion del token
      *
      * @return ResponseEntity<String> devuelve un json como string con el siguiente
      *         formato {status:"OK",token:"Token"}
      */
@RequestMapping(value = RestURIConstantes.GET_GENERAR_TOKEN, method =
RequestMethod.POST)
public ResponseEntity<String> obtenerNuevoToken(@RequestBody(required=false) String
datos) {

     HttpHeaders headers = new HttpHeaders();
     headers.setContentType(MediaType.parseMediaType("application/json"));
     ResponseEntity<String> response = new ResponseEntity<>(HttpStatus.OK);

     JSONObject json;
     try {
```

```
          if (datos != null && !datos.isEmpty()) {
               try {
                    json = new JSONObject(datos);
               } catch (Exception e) {
                    response = new
ResponseEntity<String>(jsonResponseError(ConstantesApi.EXEP_JSON_INCORRECTO),
headers, HttpStatus.UNAUTHORIZED);
                    throw new AppException(AppException.ERROR_5,
ConstantesApi.EXEP_JSON_INCORRECTO);
               }
          } else {
               response = new ResponseEntity<String>(
                         jsonResponseError(ConstantesApi.EXEP_DATOS_VACIOS),
headers, HttpStatus.UNAUTHORIZED);
               throw new AppException(AppException.ERROR_4,
ConstantesApi.EXEP_DATOS_VACIOS);
          }

          String clave;
          try {
               clave = json.get(ConstantesApi.CLAVE_TOKEN).toString();
          } catch (Exception e) {
               response = new ResponseEntity<String>(
                         jsonResponseError(ConstantesApi.EXEP_DATOS_INCORRECTOS),
headers, HttpStatus.UNAUTHORIZED);
               throw new AppException(AppException.ERROR_8,
ConstantesApi.EXEP_DATOS_INCORRECTOS);
          }

          if(clave!=null && !clave.isEmpty() &&
clave.equals(ConstantesApi.CLAVE_GENERACION_TOKEN)) {
               String token = "";
               // Generacion de token
               Map<String, Object> claims = new HashMap<>();
               String jsonString = new String();

               Integer expi = 86400; // Asignamos un dia por defecto

               token = jwtTokenUtil.doGenerateToken(claims, jsonString, expi);

               // generacion del objeto json de salida
               JSONObject jsonResponse = new JSONObject();
               jsonResponse.put("token", token);
               response = new ResponseEntity<>(jsonResponse.toString(), headers,
HttpStatus.OK);
          }
          else {
               response = new ResponseEntity<String>(
jsonResponseError(ConstantesApi.EXEP_CLAVE_GENERACION_INCORRECTA), headers,
                              HttpStatus.UNAUTHORIZED);
               throw new AppException(AppException.ERROR_9,
ConstantesApi.EXEP_CLAVE_GENERACION_INCORRECTA);
          }

     } catch (AppException e) {
          logger.debug(e.getMessage());
     }

     return response;
}
```

6. Recurso de comprobación de token

```
/**
      * Metodo utilizado para comprobar el token y devolver el contenido del token
      *
      * @param request
      *            comprueba si el token es valido
      * @return ResponseEntity<String> devuelve el contenido del token si es valido
      */
private ResponseEntity<String> comprobarToken(HttpServletRequest request) {

     ResponseEntity<String> resultado = null;
     String token = request.getHeader("Authorization");

     if(token != null){
     try {
     // eliminamos los 7 primeros caracteres (Bareer ) para quedarnos con el token
          Claims = jwtTokenUtil.getAllClaimsFromToken(token.substring(7));
          String jsonString = claims.getSubject();
          final Date expiration = claims.getExpiration();

          if (expiration.before(timeProvider.now())) {
               resultado = new ResponseEntity<>(ConstantesApi.TOKEN_EXPIRADO,
HttpStatus.NOT_ACCEPTABLE);
          } else {
               resultado = new ResponseEntity<>(jsonString, HttpStatus.OK);
          }
     } catch (ExpiredJwtException ex) {
          resultado = new ResponseEntity<>(ConstantesApi.TOKEN_EXPIRADO,
HttpStatus.NOT_ACCEPTABLE);
          logger.debug(ex.getMessage());
     } catch (Exception e) {
          resultado = new ResponseEntity<>(ConstantesApi.TOKEN_ERRONEO,
HttpStatus.UNAUTHORIZED);
          logger.debug(e.getMessage());
     }
     }
     return resultado;

                                                                              }
```

7. Servicio de negocio principal

A continuación, se aporta el código que compone la implementación del servicio raíz del aplicativo. Desde éste se redireccionan las peticiones a los servicios particulares o se ejecutan acciones que engloben a más de un servicio en particular como es la obtención del resumen.

```
@Service("restService")
public class RestServiceImpl implements RestService {

     private final Logger logger = LoggerFactory.getLogger(RestServiceImpl.class);

     @Autowired(required = true)
     private AlumnosSvc alumnosSvc;

     @Autowired(required = true)
```

```
	private CalificacionesSvc calificacionesSvc;

	@Autowired(required = true)
	private MatriculacionesSvc matriculacionesSvc;

	@Autowired(required = true)
	private TitulacionesSvc titulacionesSvc;

	@Autowired(required = true)
	private CalificacionesDAO calificacionesDAO;

	@Autowired(required = true)
	private MatriculacionesDAO matriculacionesDAO;

	/**
	 * Servicio para obtener alumnos por dni
	 */
	public AlumnosJson buscarAlumno(String dni) throws AppException{
		AlumnosJson alumno = null;
		try {
				alumno = alumnosSvc.buscarAlumno(dni);

			} catch (AppException e) {
				throw e;
			} catch (Exception e) {
				logger.error(e.getMessage(), e);
				throw new AppException(AppException.ERROR_10, "No se han
podido obtener Alumno", e);
			}
			return alumno;
	}

	/**
	 * Servicio para obtener alumnos por dni
	 */
	public List<AlumnosJson> buscarAlumnos(List<String> listaDnis) throws
AppException{
		List<AlumnosJson> alumnos = null;
		try {
				alumnos = alumnosSvc.buscarAlumnos(listaDnis);

			} catch (AppException e) {
				throw e;
			} catch (Exception e) {
				logger.error(e.getMessage(), e);
				throw new AppException(AppException.ERROR_10, "No se han
podido obtener Alumno", e);
			}
			return alumnos;
	}

	/**
	 * Servicio para obtener calificaciones de alumnos
	 */
	public List<CalificacionesJson> obtenerCalificaciones(List<String> listaDnis,
List<String> curso) throws AppException{
		List<CalificacionesJson> calificaciones = null;
		try {
			calificaciones =
calificacionesSvc.obtenerCalificaciones(listaDnis,curso.get(0));
```

```
                    } catch (AppException e) {
                         throw e;
                    } catch (Exception e) {
                         logger.error(e.getMessage(), e);
                         throw new AppException(AppException.ERROR_10, "No se han
podido obtener calificaciones", e);
                    }
                    return calificaciones;
     }

     /**
      * Servicio para obtener matriculaciones de alumnos
      */
     public List<MatriculacionesJson> obtenerMatriculaciones(List<String> listaDnis,
List<String> curso) throws AppException{
          List<MatriculacionesJson> matriculaciones = null;
          try {
               matriculaciones =
matriculacionesSvc.obtenerMatriculaciones(listaDnis,curso);

                    } catch (AppException e) {
                         throw e;
                    } catch (Exception e) {
                         logger.error(e.getMessage(), e);
                         throw new AppException(AppException.ERROR_10, "No se han
podido obtener matriculaciones", e);
                    }
                    return matriculaciones;
     }

     /**
      * Servicio para obtener matriculaciones de alumnos
      */
     public List<ResumenJson> obtenerResumen(List<String> listaDnis, List<String>
curso) throws AppException {
          List<ResumenJson> resumenes = null;
          List<Calificaciones> calificaciones = null;
          List<Matriculaciones> matriculaciones = null;
          LinkedHashMap<String, ResumenJson> mapa = new LinkedHashMap<>();
          LinkedHashMap<String, LinkedHashMap<String, BigDecimal>> notas = new
LinkedHashMap<>();
          try {
               String[] p = curso.get(0).split("/");
               String cursoAnterior = Long.toString(Long.parseLong(p[0]) - 1) + "/"
                         + Long.toString(Long.parseLong(p[1]) - 1);
               curso.add(cursoAnterior);
               calificaciones = calificacionesDAO.obtenerCalificaciones(listaDnis,
cursoAnterior);
               matriculaciones =
matriculacionesDAO.obtenerMatriculaciones(listaDnis, curso);
               if (!CollectionUtils.isEmpty(matriculaciones) &&
!CollectionUtils.isEmpty(calificaciones)) {
                    resumenes = new ArrayList<>();
                    for (Matriculaciones mat : matriculaciones) {
                         String key = mat.getAlumnos().getAlumLiDni();
                         if (mapa.containsKey(key)) {
                              ResumenJson rjson = mapa.get(key);
                              if
(rjson.getCursos().containsKey(mat.getCursos().getCursNmNombre())) {
```

```
                                        BigDecimal creditos =
mat.getAsignaturas().getAsigNuCreditos();
                                        BigDecimal creditosMatriculados = (BigDecimal)
rjson.getCursos()

     .get(mat.getCursos().getCursNmNombre()).get("creditosMatriculados");

     rjson.getCursos().get(mat.getCursos().getCursNmNombre()).put("creditosMatricula
dos",
                                                creditosMatriculados.add(creditos));
                                   } else {
                                        LinkedHashMap<String, Object> datos = new
LinkedHashMap<>();
                                        datos.put("creditosMatriculados",
mat.getAsignaturas().getAsigNuCreditos());

     rjson.getCursos().put(mat.getCursos().getCursNmNombre(), datos);
                                   }

                              } else {
                                   ResumenJson resumenJson = new ResumenJson();

     resumenJson.setNombre(concatenarNombre(mat.getAlumnos()));
                                   resumenJson.setDni(mat.getAlumnos().getAlumLiDni());

     resumenJson.setTitulacion(mat.getAsignaturas().getTitulaciones().getTituNmNombr
e());
                                   LinkedHashMap<String, LinkedHashMap<String, Object>>
cursos = new LinkedHashMap<>();
                                   LinkedHashMap<String, Object> datos = new
LinkedHashMap<>();
                                   datos.put("creditosMatriculados",
mat.getAsignaturas().getAsigNuCreditos());
                                   cursos.put(mat.getCursos().getCursNmNombre(), datos);
                                   resumenJson.setCursos(cursos);
                                   mapa.put(key, resumenJson);

                              }

                         }

                         for (Calificaciones cal : calificaciones) {
                              String key = cal.getAlumnos().getAlumLiDni();
                              if (notas.containsKey(key)) {
                                   LinkedHashMap<String, BigDecimal> ndatos =
notas.get(key);
                                   BigDecimal numero = ndatos.get("numero");
                                   BigDecimal suma = ndatos.get("suma");
                                   ndatos.put("numero", numero.add(new BigDecimal(1)));
                                   ndatos.put("suma",
suma.add(cal.getCaliNuCalificacion()));

                              } else {
                                   LinkedHashMap<String, BigDecimal> ndatos = new
LinkedHashMap<>();
                                   ndatos.put("numero", new BigDecimal(1));
                                   ndatos.put("suma", cal.getCaliNuCalificacion());
                                   notas.put(key, ndatos);

                                   ResumenJson rjson = mapa.get(key);
```

```
                                String fuera = null;
                                LinkedHashMap<String, LinkedHashMap<String, Object>>
cursos = rjson.getCursos();
                                LinkedHashMap<String, Object> datos =
cursos.get(curso.get(0));

     if(cal.getAlumnos().getAlumLiDomicilioCurso().equals(cal.getAlumnos().getAlumLi
DomicilioFamiliar())){
                                     fuera = "No";
                                }else{
                                     fuera = "Si";
                                }
                                datos.put("ResideFuera", fuera);
                           }
                           ResumenJson rjson = mapa.get(key);
                           LinkedHashMap<String, LinkedHashMap<String, Object>> cursos
= rjson.getCursos();
                           LinkedHashMap<String, Object> datos =
cursos.get(cal.getCursos().getCursNmNombre());
                           if (cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) == 0
                                     || cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) > 0
                                                &&
datos.containsKey("creditosAprobados")) {
                                BigDecimal creditosAprobados = (BigDecimal)
datos.get("creditosAprobados");
                                datos.put("creditosAprobados",
creditosAprobados.add(cal.getAsignaturas().getAsigNuCreditos()));
                           }
                           else if (cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) == 0
                                     || cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) > 0
                                                &&
!datos.containsKey("creditosAprobados")) {
                                datos.put("creditosAprobados",
cal.getAsignaturas().getAsigNuCreditos());
                           }
                           else if (cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) < 0
                                     && datos.containsKey("creditosSuspensos")) {
                                BigDecimal creditosAprobados = (BigDecimal)
datos.get("creditosSuspensos");
                                datos.put("creditosSuspensos",
creditosAprobados.add(cal.getAsignaturas().getAsigNuCreditos()));
                           }
                           else if (cal.getCaliNuCalificacion().compareTo(new
BigDecimal(5)) < 0
                                     && !datos.containsKey("creditosSuspensos")) {
                                datos.put("creditosSuspensos",
cal.getAsignaturas().getAsigNuCreditos());
                           }
                           if (!datos.containsKey("creditosAprobados")) {
                                datos.put("creditosAprobados", new BigDecimal(0));
                           }
                           if (!datos.containsKey("creditosSuspensos")) {
                                datos.put("creditosSuspensos", new BigDecimal(0));
                           }
```

```
                        BigDecimal aprobados = (BigDecimal)
datos.get("creditosAprobados");
                        BigDecimal suspensos = (BigDecimal)
datos.get("creditosSuspensos");
                        BigDecimal porcentaje =
aprobados.divide(aprobados.add(suspensos), 2, RoundingMode.HALF_UP);
                        datos.put("porcentajeAprobados",
String.valueOf(porcentaje.multiply(new BigDecimal(100))) + "%");
                        LinkedHashMap<String, BigDecimal> ndatos = notas.get(key);
                        BigDecimal notaMedia =
ndatos.get("suma").divide(ndatos.get("numero"), 1, RoundingMode.HALF_UP);
                        datos.put("notaMedia", notaMedia);

                    }
                    resumenes.addAll(mapa.values());
                }
            } catch (AppException e) {
                throw e;
            } catch (Exception e) {
                logger.error(e.getMessage(), e);
                throw new AppException(AppException.ERROR_10, "No se han podido
obtener los resúmenes", e);
            }
            return resumenes;
        }

        /**
         * Servicio para obtener titulaciones por nombre del centro
         *    */
        public List<TitulacionesJson> obtenerTitulacionesPorCentro(List<String>
centros) throws AppException{
            List<TitulacionesJson> titulaciones = null;
            try {
                titulaciones = titulacionesSvc.obtenerTitulacionesPorCentro(centros);

                    } catch (AppException e) {
                        throw e;
                    } catch (Exception e) {
                        logger.error(e.getMessage(), e);
                        throw new AppException(AppException.ERROR_10, "No se han
podido obtener Titulaciones", e);
                    }
                    return titulaciones;
        }

        /**
         * Servicio para obtener titulaciones por nombre titulacion
         *    */
        public List<TitulacionesJson> obtenerTitulaciones(List<String> titulaciones)
throws AppException{
            List<TitulacionesJson> titulacionesJson = null;
            try {
                titulacionesJson = titulacionesSvc.obtenerTitulaciones(titulaciones);

                    } catch (AppException e) {
                        throw e;
                    } catch (Exception e) {
                        logger.error(e.getMessage(), e);
                        throw new AppException(AppException.ERROR_10, "No se han
podido obtener Titulaciones", e);
```

```
                    }
                    return titulacionesJson;
     }

     private String concatenarNombre(Alumnos alumno) throws AppException{
           String nombreCompleto = alumno.getAlumNmNombre()+"
".concat(alumno.getAlumLiApellido1())+" ".concat(alumno.getAlumLiApellido2());
                return nombreCompleto;
          }



}
```

8. Servicio (Negocio) para la construcción de la respuesta

En este punto se adjunta el código correspondiente a uno de los servicios de la capa de negocio para la formación de la respuesta, en concreto el servicio para la obtención de matriculaciones.

```
@Transactional
@Service
public class MatriculacionesSvcImpl implements MatriculacionesSvc {

     private final Logger logger =
LoggerFactory.getLogger(MatriculacionesSvcImpl.class);

     @Autowired(required = true)
     private MatriculacionesDAO matriculacionesDAO;


     /**
      * Obtener Matriculaciones por alumnos y curso de BBDD
      */
     public List<MatriculacionesJson> obtenerMatriculaciones(List<String> listaDnis,
List<String> curso) throws AppException{
          List<MatriculacionesJson> matJson = null;
          LinkedHashMap<String, MatriculacionesJson> mapa = new LinkedHashMap<>();
          try {
               List<Matriculaciones> matriculaciones =
matriculacionesDAO.obtenerMatriculaciones(listaDnis, curso);
               if (!CollectionUtils.isEmpty(matriculaciones)) {
                    matJson = new ArrayList<MatriculacionesJson>();
               for (Matriculaciones mat : matriculaciones) {
                    String key = mat.getAlumnos().getAlumLiDni();
                    if(mapa.containsKey(key)){
                         MatriculacionesJson mjson = mapa.get(key);
                         LinkedHashMap<String, BigDecimal> matri = new
LinkedHashMap<>();
                         matri.put(mat.getAsignaturas().getAsigNmNombre(),
mat.getAsignaturas().getAsigNuCreditos());

     if(mjson.getCursos().containsKey(mat.getCursos().getCursNmNombre())){
                         AsignaturasMatJson asig =
mjson.getCursos().get(mat.getCursos().getCursNmNombre());

     asig.getMatriculaciones().put(mat.getAsignaturas().getAsigNmNombre(),
mat.getAsignaturas().getAsigNuCreditos());
```

```
     asig.setTotalCreditos(asig.getTotalCreditos().add(mat.getAsignaturas().getAsigN
uCreditos()));
                         }else{
                              AsignaturasMatJson asignaturas = new
AsignaturasMatJson();
                              asignaturas.setMatriculaciones(matri);

     asignaturas.setTotalCreditos(mat.getAsignaturas().getAsigNuCreditos());
                         mjson.getCursos().put(mat.getCursos().getCursNmNombre(),
asignaturas);
                         }

                    }
                    else{
                         MatriculacionesJson mjson = new MatriculacionesJson();
                         mjson.setNombre(concatenarNombre(mat.getAlumnos()));

     mjson.setTitulacion(mat.getAsignaturas().getTitulaciones().getTituNmNombre());
                         LinkedHashMap<String, BigDecimal> matri = new
LinkedHashMap<>();
                         matri.put(mat.getAsignaturas().getAsigNmNombre(),
mat.getAsignaturas().getAsigNuCreditos());
                         LinkedHashMap<String, AsignaturasMatJson> c = new
LinkedHashMap<>();
                         AsignaturasMatJson asignaturas = new AsignaturasMatJson();
                         asignaturas.setMatriculaciones(matri);

     asignaturas.setTotalCreditos(mat.getAsignaturas().getAsigNuCreditos());
                         c.put(mat.getCursos().getCursNmNombre(), asignaturas);
                         mjson.setCursos(c);
                         mapa.put(key, mjson);

                    }
               }
               matJson.addAll(mapa.values());
               }
                    } catch (AppException e) {
                         throw e;
                    } catch (Exception e) {
                         logger.error(e.getMessage(), e);
                         throw new AppException(AppException.ERROR_10, "No se han
podido obtener Alumnos", e);
                    }
                    return matJson;
     }
     private String concatenarNombre(Alumnos alumno) throws AppException{
      String nombreCompleto = alumno.getAlumNmNombre()+"
".concat(alumno.getAlumLiApellido1())+" ".concat(alumno.getAlumLiApellido2());
          return nombreCompleto;
     }

}
```

9. Servicio de acceso a datos

Este apartado se acompaña del código correspondiente de un servicio de acceso a datos, para seguir en consecuencia a los apartados anteriores se muestra el acceso a la tabla de matriculaciones, cerrando así la ejemplificación de un flujo completo de obtención de datos a través de un recurso de los definidos.

```
@Repository("matriculacionesDAO")
@Transactional
public class MatriculacionesDAOImpl implements MatriculacionesDAO {

private final Logger logger = LoggerFactory.getLogger(MatriculacionesDAOImpl.class);

     @Autowired
    private SessionFactory sessionFactory;


     public List<Matriculaciones> obtenerMatriculaciones(List<String> listaDnis,
List<String> curso) throws AppException{
          List<Matriculaciones> hs = null;

          try{
               Criteria criteria =
sessionFactory.getCurrentSession().createCriteria(Matriculaciones.class);
               criteria.createAlias("alumnos", "a");
               criteria.createAlias("cursos", "c");
               criteria.add(Restrictions.in("a.alumLiDni", listaDnis));
               criteria.add(Restrictions.in("c.cursNmNombre", curso));
               hs = (List<Matriculaciones>) criteria.list();
          }catch(Exception e){
               logger.error(e.getMessage(), e);
               throw new AppException(AppException.ERROR_10, "Ha ocurrido un error
al obtener las matriculaciones.",e);
          }

         return hs;
     }
}
```