Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

INDUSTRIA INGENIARITZA TEKNIKOKO ATALA

SECCIÓN INGENIERÍA TÉCNICA INDUSTRIAL

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

2015/2016

AdESMuS

MEMORIA

DATOS DE LA ALUMNA O DEL ALUMNO	**DATOS DEL DIRECTOR O DIRECTORA**
NOMBRE **FCO. JAVIER**	NOMBRE **MIKEL**
APELLIDOS **GARCÍA CARVAJAL**	APELLIDOS **VILLAMAÑE GIRONÉS**
	DEPARTAMENTO **LENGUAJES Y SISTEMAS INFORMÁTICOS**
FDO.: FECHA:	FDO.: FECHA:

Índice de contenidos

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

AdESMuS (**Ad**aptable **E**valuation **S**ystem Using **Mu**ltiple **S**ources) es una herramienta configurable que permite realizar la evaluación de los TFGs. Dispone de diferentes formas de evaluación, y permite realizar diferentes configuraciones para diferentes asignaturas en función de unos elementos evaluables previamente definidos. Para la evaluación de los elementos evaluables se utiliza el concepto de las rúbricas holísticas y analíticas, que permiten evaluar de forma escalar, el desarrollo de una tarea tanto a nivel de ejecución como de efectividad.

2 PLANTEAMIENTO INICIAL

En el siguiente apartado se tratan los apartados del planteamiento inicial del proyecto. Los objetivos marcados, la arquitectura utilizada, las herramientas usadas y el alcance del proyecto.

2.1 Objetivos

El objetivo principal del proyecto reside en definir e implementar para AdESMuS un sistema escalable que permita realizar transacciones de datos a través de la red sin necesidad de estar ligado a un único ámbito de ejecución. Todos los usuarios de AdESMuS podrán utilizar el sistema desde diferentes medios, como puede ser un móvil, una web o incluso desde aplicaciones externas como puede ser Moodle. Por ello, los objetivos marcados serán los siguientes:

- Definir la administración del sistema AdESMuS, entendiendo por administración a la parte en la que son registrados y manipulados todos los elementos que participan en el funcionamiento de AdESMuS, como pueden ser, los usuarios, centros, departamentos, asignaturas, grupos, etc.
- Definir e implementar un sistema que permita la interacción con los usuarios de AdESMuS a través de la red para la manipulación de sus datos. Una API.
- Definir e implementar la seguridad de una API.
- Definir e implementar un entorno de pruebas consistente que permita controlar el correcto funcionamiento del sistema.
- Y sobre todo, hacer de AdESMuS una herramienta fácil e intuitiva, que permita un acceso rápido a los datos.

2.2 Arquitectura

AdESMuS sigue una arquitectura basada en el modelo *REST* (**Re**presentational **S**tate **T**ransfer), *stateless* - sin estado, y se apoya completamente en el estándar HTTP para realizar las transacciones de los datos.

Denominada como Servicio web RESTful, la aplicación se basa en una comunicación cliente/servidor, donde el canal de comunicación se encuentra protegido por un *certificado SSL (https)* y el acceso a la API se realiza a través de *Tokens***,** los cuales, serán proporcionados por el servidor. El sistema también dispone de un control de acceso por *roles* que permiten determinar las capacidades del usuario para realizar acciones sobre la API.

Como se puede observar en la siguiente ilustración, existen dos zonas claramente diferenciadas y separadas por un canal de comunicación (Internet). Dicho canal de

comunicación se encuentra encriptado por un certificado SSL. Los usuarios pueden obtener sus datos desde diferentes dispositivos.

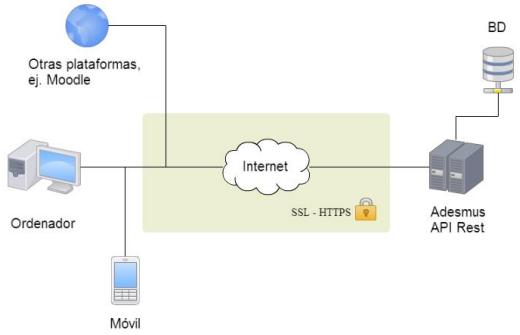

Ilustración 1 - Arquitectura REST

En la siguiente ilustración se representa una comunicación cliente/servidor, donde el usuario se loguea para obtener un Token. En ese momento, el sistema determina si los datos proporcionados por el usuario son correctos, y procede a enviarle al cliente un Token, con el cual, el usuario pueda solicitar información en las siguientes peticiones.

El principio de *stateless* de REST lo controlará el usuario consumidor del recurso. En cada petición solicitada a la API, el usuario necesitará proporcionar el Token que se le facilitó en el momento de la autenticación. De esta manera, el sistema sabe quién le está solicitando información y si tiene permiso para consumir el recurso solicitado.

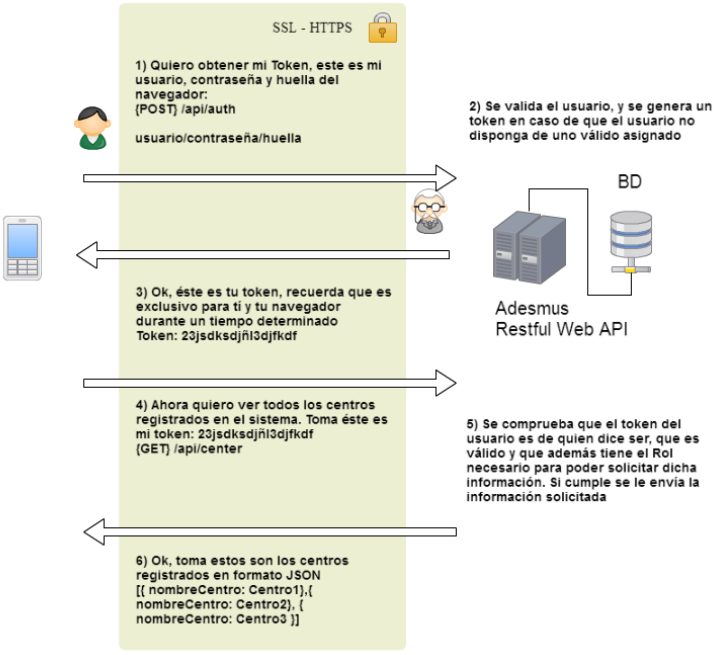

Ilustración 2 - Ejemplo de comunicación Servicio web RESTful

2.3 Herramientas

Durante el desarrollo del proyecto se hará uso de las siguientes herramientas:

Ordenador portátil y de sobremesa:
Imprescindibles para poder desarrollar el proyecto

Sistemas operativos
Se utilizarán principalmente dos sistemas operativos. Un sistema basado en Linux que hará las veces de servidor, y un sistema operativo Windows7 desde el cual se realizará la implementación del proyecto.

Sublime Text2:
Es el IDE seleccionado para la implementación. Es una herramienta de poco peso, extremadamente ágil, y con infinidad de extensiones.

Gantt Project
Herramienta utilizada para la generación de la planificación temporal.

Microsoft Office
Se utilizará para el desarrollo de la documentación del proyecto.

NodeJS
Es la librería seleccionada como base para el desarrollo de la API. Se trata de una librería JavaScript que se ejecuta en el lado del servidor, es asíncrona y está basada en eventos. Otra de las principales ventajas de la librería es que no necesita de terceras aplicaciones como puede ser Apache o Tomcat para su funcionamiento.

Cacoo
Herramienta online que permite realizar diagramas y organigramas.

Postman
Es la herramienta seleccionada para la creación y ejecución de las pruebas unitarias.

Google Drive
Se hará uso de esta herramienta para gestionar en la nube los backups que hagamos del proyecto y gestionar el acceso a todos los documentos que se vayan registrando.

Dia
Herramienta para realizar casos de uso y modelos de dominio.

2.4 Alcance

El proyecto se ha divido en las siguientes fases:

2.4.1 Fases del proyecto

Captura de requisitos

El primero de los pasos a seguir es saber las necesidades que se requieren en el proyecto para la consecución del mismo.

Análisis

Una vez que se tiene claro el qué se quiere hacer, se busca el cómo hacerlo. La búsqueda de información se dividirá por necesidades requeridas, inicialmente marcadas después de realizar las primeras reuniones con el director.

Seleccionar un sistema de gestión de bases de datos y un lenguaje de programación que cumplan con nuestras expectativas, hacer un estudio de la seguridad a implementar y determinar de qué manera realizar el transporte de los datos.

Diseño

Una vez seleccionadas las tecnologías que se van a utilizar, se estructura la aplicación unificando la problemática del punto anterior.

Implementación

Se modulariza la aplicación consiguiendo una abstracción en el código. La implementación del proyecto se dividirá en los siguientes prototipos:

- **Prototipo Nº1. Acceso a la documentación online**: Se creará la documentación online de la API, para que el usuario pueda consultar en todo momento de que manera realizar la comunicación con el servidor.

- **Prototipo Nº2. Acceso a la Landing Page de la API:** Se creará una página principal a modo de presentación de la API.

- **Prototipo Nº3. Registro y acceso al sistema:** Se implementarán las funcionalidades necesarias que permitan realizar un registro en el sistema mediante un usuario y contraseña, y posteriormente, con el usuario ya registrado, y mediante la operación de logueo, conseguir un Token válido para el acceso a los recursos disponibles.

- **Prototipo Nº4. Recursos para realizar operaciones sobre el Administrador, Docente y Estudiante:** Se implementarán todas las funcionalidades necesarias que permitan al sistema, añadir, editar, actualizar o eliminar usuarios de tipo Administrador, Docente o Estudiante, de la base de datos.

- **Prototipo Nº5. Recursos para realizar operaciones sobre las Asignaturas:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Asignaturas de la base de datos.

- **Prototipo Nº6. Recursos para realizar operaciones sobre los Centros:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Centros de la base de datos.

- **Prototipo Nº7. Recursos para realizar operaciones sobre los Departamentos:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Departamentos de la base de datos.

- **Prototipo Nº8. Recursos para realizar operaciones sobre las Titulaciones:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Titulaciones de la base de datos.

- **Prototipo Nº9. Recursos para realizar operaciones sobre los Grupos:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Grupos de la base de datos.

- **Prototipo Nº10. Recursos para realizar operaciones sobre las Matrículas:** Se implementarán las funcionalidades necesarias que permitan al sistema añadir, editar, actualizar o eliminar Matrículas de la base de datos.

Plan de pruebas

Se crean baterías de pruebas unitarias para comprobar el estado de la aplicación, garantizando de esta manera que el funcionamiento de la aplicación sea correcta en cualquiera de sus vertientes. AdESMuS no dispone de una interfaz gráfica por lo tanto ésta sección es muy importante para el correcto funcionamiento de la aplicación y se hará especial hincapié en ella durante el transcurso del proyecto.

Documentación

La documentación se irá completando a lo largo del proyecto con la información recabada a la finalización de las fases establecidas.

2.4.2 Esquema de descomposición del proyecto

La siguiente ilustración representa el esquema de descomposición del proyecto en diferentes tareas.

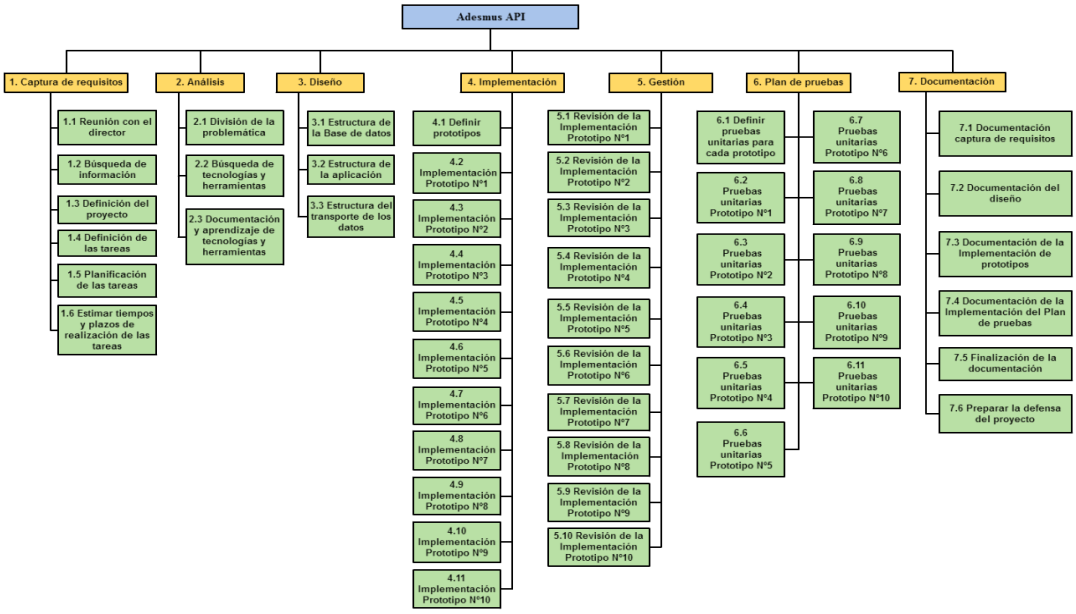

Ilustración 3 - EDT

2.4.3 Tareas

A continuación se definirán las tareas del EDT de una forma detallada:

1. Captura de requisitos

1.1. Paquete de trabajo: Reunión con el director

- **Duración:** 3 horas.
- **Descripción:** Primera toma de contacto con el director del proyecto para abordar las necesidades del proyecto.
- **Salidas/Entregables:** Documento que formaliza lo hablado durante la reunión.
- **Recursos necesarios:** Tener claro cuáles son las metas del proyecto y que es lo que se necesita para llegar a ellas.

1.2. Paquete de trabajo: Búsqueda de información

- **Duración:** 18 horas.

- **Descripción:** Una vez que se sabe qué es lo que se necesita, se busca información general al respecto. Documentación sobre APIs, protocolos de comunicación, seguridad en el transporte de datos. Buscar casos de éxito fielmente documentados y posibles alternativas.
- **Salidas/Entregables:** Información recabada en forma de enlaces o documentos.
- **Recursos necesarios:** Acceso a Internet para obtener la información.
- **Precedencias:** Paquete de trabajo 1.1

1.3. Paquete de trabajo: Definición del proyecto

- **Duración:** 9 horas.
- **Descripción:** Se define el esqueleto de la aplicación, unificando todas las tecnologías seleccionadas para el funcionamiento de la misma.
- **Salidas/Entregables:** Estructura básica del proyecto establecida.
- **Recursos necesarios:** Tener claro qué se quiere hacer y disponer de las tecnologías necesarias.
- **Precedencias:** Paquete de trabajo 1.1 y 1.2

1.4. Paquete de trabajo: Definición de las tareas

- **Duración:** 4 horas.
- **Descripción:** Definir las tareas que son necesarias para realizar el proyecto.
- **Salidas/Entregables:** Tareas definidas y perfectamente documentadas.
- **Recursos necesarios:** Tener claro qué es lo que se necesita y de qué manera llegar a conseguir los objetivos marcados.
- **Precedencias:** Paquete de trabajo 1.3

1.5. Paquete de trabajo: Planificación de las tareas

- **Duración:** 3 horas.
- **Descripción:** Establecer una hoja de ruta, indicando de qué manera se van a realizar las tareas y en qué orden.
- **Salidas/Entregables:** Planificación de ejecución de las tareas.
- **Recursos necesarios:** Tener definidas y documentadas las tareas necesarias.
- **Precedencias:** Paquete de trabajo 1.4

1.6. Paquete de trabajo: Estimar tiempos y plazos de realización de las tareas

- **Duración:** 3 horas.
- **Descripción:** Se realiza el cálculo aproximado del tiempo que será necesario invertir en la realización de las tareas.
- **Salidas/Entregables:** Planificación de ejecución de las tareas con tiempos de ejecución.
- **Recursos necesarios:** Planificación de las tareas.
- **Precedencias:** Paquete de trabajo 1.5

2. Análisis

2.1. Paquete de trabajo: División de la problemática

- **Duración:** 3 horas.
- **Descripción:** Analizar la problemática del proyecto y dividirlo en secciones atacables de forma independiente. "*Divide y vencerás"*.
- **Salidas/Entregables:** División del problema en pequeñas porciones.
- **Recursos necesarios:** Tener toda la información necesaria para determinar de qué manera realizar las divisiones del problema.
- **Precedencias:** Tareas de Gestión del 1.1 al 1.7

2.2. Paquete de trabajo: Búsqueda de tecnologías y herramientas

- **Duración:** 7 horas.
- **Descripción:** Realizar una búsqueda exhaustiva de las tecnologías y herramientas que se incorporarán al proyecto.
- **Salidas/Entregables:** Tener claro que las tecnologías y herramientas seleccionadas cumplan las necesidades del proyecto.
- **Recursos necesarios:** Cuales son las necesidades o requerimientos del proyecto.
- **Precedencias:** Paquete de trabajo 1.1

2.3. Paquete de trabajo: Documentación y aprendizaje de tecnologías y herramientas

- **Duración:** 8 horas.
- **Descripción:** Familiarizarse con el uso de las tecnologías y herramientas seleccionadas partiendo de toda la información obtenida en paquete de trabajo 2.2.
- **Salidas/Entregables:** Adquisición de un conocimiento básico que aumentará a medida que se desarrolle el proyecto.
- **Recursos necesarios:** Las tecnologías y herramientas seleccionadas.
- **Precedencias:** Paquete de trabajo 2.2

3. Diseño

3.1. Paquete de trabajo: Estructura de la Base de Datos

- **Duración:** 7 horas.
- **Descripción:** Inicialmente se parte de un modelo relacional ya existente. Se han rediseñado y añadido tablas nuevas, y se han actualizado campos. También se han añadido nuevas funcionalidades a las tablas (Triggers).
- **Entradas:** Se parte de un modelo relacional ya existente.
- **Salidas/Entregables:** Modelo completo de base de datos.
- **Recursos necesarios:** Modelo relacional existente y motor de base de datos.

- **Precedencias:** Previo análisis y conocimiento del modelo relacional ya existente.

3.2. Paquete de trabajo: Estructura de la aplicación

- **Duración:** 26 horas.
- **Descripción:** Se realiza un análisis de las necesidades para determinar la creación de una estructura eficiente, escalable y segura.
- **Salidas/Entregables:** Una estructura consolidada del proyecto.
- **Recursos necesarios:** Tener claro todas las necesidades del proyecto.
- **Precedencias:** Tareas de Gestión y Análisis.

3.3. Paquete de trabajo: Estructura del tratamiento y transporte de los datos

- **Duración:** 3 horas.
- **Descripción:** Se define la comunicación entre el cliente y el servidor a través de un modelo de transporte de datos. De qué manera se envían y reciben los datos.
- **Salidas/Entregables:** Diseño de la trama de datos para la comunicación entre el cliente y el servidor.
- **Recursos necesarios:** Tener claro el qué y de qué manera queremos enviar y recibir los datos a través de la capa de transporte.
- **Precedencias:** Paquete de trabajo 2.2 y 2.3

4. Implementación

4.1. Paquete de trabajo: Definir prototipos

- **Duración:** 6 horas.
- **Descripción:** Para facilitar la implementación, el sistema se dividirá en prototipos, en los cuales, se irán añadiendo nuevas funcionalidades hasta completar la aplicación.
- **Salidas/Entregables:** Definición de prototipos.
- **Recursos necesarios:** Tener claros los requerimientos del proyecto.
- **Precedencias:** Grupos de tareas 1, 2 y 3

4.2. Paquete de trabajo: Implementación Prototipo Nº1. *"Acceso a la documentación online"*

- **Duración:** 8 horas.
- **Descripción:** Se implementará una documentación online para la consulta de los recursos disponibles para el usuario.
- **Salidas/Entregables:** Definición de la documentación online.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3

4.3. Paquete de trabajo: Implementación Prototipo Nº2. *Módulo de comunicación: "Acceso a la Landing Page de la API"*

- **Duración:** 1 horas.
- **Descripción:** Se implementará una página de entrada a la API y de consulta pública. Todos los usuarios (incluidos anónimos) podrán acceder a ella para ver de qué se trata AdESMuS.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.4. Paquete de trabajo: Implementación Prototipo Nº 3. *Módulo de comunicación: "Registro y acceso al sistema"*

- **Duración:** 22 horas.
- **Descripción:** Se implementará la funcionalidad que permita al usuario registrarse en caso de tener permisos o realizar un login para acceder al sistema.

  /auth/register/ (recurso para el registro)
  /auth/login/ (recurso para realizar una identificación)

- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.5. Paquete de trabajo: Implementación Prototipo Nº 4. *Módulo de comunicación: "Recursos para realizar operaciones sobre el Administrador, Docente y Estudiante"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de los usuarios en el sistema a través de las siguientes rutas:

  /api/users/administrator/ (recurso administrador)
  /api/users/teaching/ (recurso docente)
  /api/users/student/ (recurso estudiante)

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.6. Paquete de trabajo: Implementación Prototipo Nº 5. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Asignaturas"*

- **Duración:** 20 horas.

- **Descripción:** Se implementarán los recursos necesarios para la gestión de las asignaturas a través de la siguiente ruta:

  /api/subject/

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.7. Paquete de trabajo: Implementación Prototipo Nº 6. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Centros"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de los Centros a través de la siguiente ruta:

  /api/center/

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.8. Paquete de trabajo: Implementación Prototipo Nº 7. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Departamentos"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de los Departamentos a través de la siguiente ruta:

  /api/department/

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.9. Paquete de trabajo: Implementación Prototipo Nº 8. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Titulaciones"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de las Titulaciones a través de la siguiente ruta:

  /api/degree/

En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.

- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.10. Paquete de trabajo: Implementación Prototipo Nº 9. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Grupos"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de los Grupos a través de la siguiente ruta:

  /api/group/

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3.

4.11. Paquete de trabajo: Implementación Prototipo Nº 10. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Matrículas"*

- **Duración:** 20 horas.
- **Descripción:** Se implementarán los recursos necesarios para la gestión de las Matrículas a través de la siguiente ruta:

  /api/enrollment/

  En función de los privilegios que el usuario disponga, éste podrá hacer uso de la diferente funcionalidad que se aporta bajo cada una de las rutas anteriormente nombradas.
- **Salidas/Entregables:** Definición de los recursos para el consumo de datos.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Grupos de tareas 1, 2 y 3

5. Gestión

5.1. Paquete de trabajo: Revisión de la implementación, Prototipo Nº1

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.2

5.2. Paquete de trabajo: Revisión de la implementación, Prototipo Nº2

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.3

5.3. Paquete de trabajo: Revisión de la implementación, Prototipo Nº3

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.4

5.4. Paquete de trabajo: Revisión de la implementación, Prototipo Nº4

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.5

5.5. Paquete de trabajo: Revisión de la implementación, Prototipo Nº5

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.6

5.6. Paquete de trabajo: Revisión de la implementación, Prototipo Nº6

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.7

5.7. Paquete de trabajo: Revisión de la implementación, Prototipo Nº7

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.8

5.8. Paquete de trabajo: Revisión de la implementación, Prototipo Nº8

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.9

5.9. Paquete de trabajo: Revisión de la implementación, Prototipo Nº9

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.10

5.10. Paquete de trabajo: Revisión de la implementación, Prototipo Nº10

- **Duración:** 1 hora.
- **Descripción:** Se revisará la implementación del prototipo.
- **Salidas/Entregables:** Implementación del prototipo revisada.
- **Recursos necesarios:** Prototipo implementado.
- **Precedencias:** Paquete de trabajo 4.11

6. Plan de pruebas

6.1. Paquete de trabajo: Definir Pruebas Unitarias (UT)

- **Duración:** 15 horas.
- **Descripción:** Se implementarán baterías de pruebas unitarias para comprobar la integridad y estado de la API. Con dichas baterías, se realizará una comprobación de cada uno de los recursos disponibles en la API, validando desde la entrada de datos, estado del usuario logueado en el sistema, permisos de ejecución sobre recursos o datos introducidos en el sistema. Se comprobarán los tres principales módulos en su totalidad: Módulo de comunicación, Modulo de coordinación y el Modulo de Gestión.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Tareas de gestión, análisis y diseño.

6.2. Paquete de trabajo: Pruebas Unitarias Prototipo Nº1. *Módulo de comunicación: "Acceso a la documentación online"*

- **Duración:** 5 horas.
- **Descripción:** Se implementa la documentación online para el consumo de datos de la API. Dicha documentación estará ligada directamente a cada una

de las cabeceras de las funciones, lo cual facilita su mantenimiento en un futuro, en caso de realizar modificaciones.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.2 y 5.1

6.3. Paquete de trabajo: Pruebas Unitarias Prototipo Nº2. *Módulo de comunicación: "Acceso a la Landing Page de la API"*

- **Duración:** 1 horas.
- **Descripción:** Se comprobará que la página principal de la API funciona correctamente.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.3 y 5.2

6.4. Paquete de trabajo: Pruebas Unitarias Prototipo Nº3. *Módulo de comunicación: "Registro y acceso al sistema"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.4 y 5.3

6.5. Paquete de trabajo: Pruebas Unitarias Prototipo Nº4. *Módulo de comunicación: "Recursos para realizar operaciones sobre el Administrador, Docente y Estudiante"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.5 y 5.4

6.6. Paquete de trabajo: Pruebas Unitarias Prototipo Nº5. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Asignaturas"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.

- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.6 y 5.5

6.7. Paquete de trabajo: Pruebas Unitarias Prototipo Nº6. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Centros"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.7 y 5.6

6.8. Paquete de trabajo: Pruebas Unitarias Prototipo Nº7. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Departamentos"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.8 y 5.7

6.9. Paquete de trabajo: Pruebas Unitarias Prototipo Nº8. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Titulaciones"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.9 y 5.8

6.10. Paquete de trabajo: Pruebas Unitarias Prototipo Nº9. *Módulo de comunicación: "Recursos para realizar operaciones sobre los Grupos"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.
- **Precedencias:** Paquete de trabajo 4.10 y 5.9

6.11. Paquete de trabajo: Pruebas Unitarias Prototipo Nº10. *Módulo de comunicación: "Recursos para realizar operaciones sobre las Matrículas"*

- **Duración:** 10 horas.
- **Descripción:** Se implementarán las pruebas unitarias que certifiquen una comprobación exhaustiva de cada uno de los recursos contenidos en el prototipo.
- **Salidas/Entregables:** Definición de pruebas unitarias.
- **Recursos necesarios:** Definición del prototipo.

7. Documentación

7.1. Paquete de trabajo: Documentación de la captura de requisitos

- **Duración:** 17 horas.
- **Descripción:** Unificar la información obtenida, para determinar el plan de progreso en la ejecución del proyecto.
- **Salidas/Entregables:** Documento con los objetivos del proyecto.
- **Recursos necesarios:** Todo lo necesario para crear el documento de objetivos del proyecto.
- **Precedencias:** Tarea 1.

7.2. Paquete de trabajo: Documentación del diseño

- **Duración:** 5 horas.
- **Descripción:** Al finalizar la tarea de Análisis se obtendrá la información recogida durante su ejecución y se procederá a la documentación de la misma.
- **Salidas/Entregables:** La memoria parcial.
- **Recursos necesarios:** La información obtenida durante la ejecución de la tarea de Análisis.
- **Precedencias:** Grupo de tareas 1, 2 y 3.

7.3. Paquete de trabajo: Documentación de la implementación de los prototipos

- **Duración:** 5 horas.
- **Descripción:** Al finalizar la implementación de los prototipos, se documentará cada prototipo de forma individual, explicando detalladamente su implementación.
- **Salidas/Entregables:** La memoria parcial.
- **Recursos necesarios:** Implementación de los prototipos.
- **Precedencias:** Todos los paquetes de trabajo de la tarea 4

7.4. Paquete de trabajo: Documentación de la implementación del Plan de pruebas

- **Duración:** 10 horas.

- **Descripción:** Al finalizar la ejecución del plan de pruebas unitarias se documentarán las pruebas realizadas, indicando de forma detallada cuales han sido las pruebas realizas.
- **Salidas/Entregables:** La memoria parcial.
- **Recursos necesarios:** Implementación del plan de pruebas unitarias.
- **Precedencias:** Todos los paquetes de trabajo de la tarea 6

7.5. Paquete de trabajo: Finalización de la documentación

- **Duración:** 20 horas.
- **Descripción:** Se completará la memoria del proyecto, incluyendo toda la información obtenida durante la ejecución del mismo. Tareas, diseños realizados y dificultades encontradas. Añadiendo también las conclusiones, valoraciones y experiencias.
- **Salidas/Entregables:** La memoria.
- **Recursos necesarios:** Documentación obtenida en la ejecución de las tareas.
- **Precedencias:** Grupos de tareas de 1, 2, 3, 4, 5 y 6.

7.6. Paquete de trabajo: Preparar la defensa del proyecto

- **Duración:** 20 horas.
- **Descripción:** Preparar la defensa del proyecto
- **Salidas/Entregables:** La defensa.
- **Recursos necesarios:** La memoria y el proyecto.
- **Precedencias:** Grupos de tareas de 1, 2, 3, 4, 5, 6 y los paquetes de trabajo 7.1, 7.2, 7.3, 7.4 y 7.5

2.4.4 Planificación temporal

En la siguiente ilustración se realiza un cálculo estimado del número de horas necesarias para la realización de las tareas mencionadas anteriormente, acompañado de un Gantt para definir en el tiempo la planificación establecida.

Tareas	Estimación
Captura de requisitos	**40 horas**
1.1.Reunión con el director	3 horas
1.2. Búsqueda de información	18 horas
1.3. Definición del proyecto	9 horas
1.4. Definición de las tareas	4 horas
1.5. Planificación de las tareas	3 horas
1.6. Estimar tiempos y plazos de realización de las tareas	3 horas
Análisis	**18 horas**
2.1. División de la problemática	3 horas
2.2. Búsqueda de tecnologías y herramientas	7 horas

2.3. Documentación y aprendizaje de tecnologías y herramientas	8 horas
Diseño	**36 horas**
3.1. Estructura de la base de datos	7 horas
3.2. Estructura de la aplicación	26 horas
3.3. Estructura de la capa de transporte	3 horas
Implementación	**177 horas**
4.1. Definir prototipos	6 horas
4.2. Implementación Prototipo Nº1	8 horas
4.3. Implementación Prototipo Nº2	1 hora
4.4. Implementación Prototipo Nº3	22 horas
4.5. Implementación Prototipo Nº4	20 horas
4.6. Implementación Prototipo Nº5	20 horas
4.7. Implementación Prototipo Nº6	20 horas
4.8. Implementación Prototipo Nº7	20 horas
4.9. Implementación Prototipo Nº8	20 horas
4.10. Implementación Prototipo Nº9	20 horas
4.11. Implementación Prototipo Nº10	20 horas
Gestión	**10 horas**
5.1 Revisión de la implementación Prototipo Nº1	1 hora
5.2 Revisión de la implementación Prototipo Nº2	1 hora
5.3 Revisión de la implementación Prototipo Nº3	1 hora
5.4 Revisión de la implementación Prototipo Nº4	1 hora
5.5 Revisión de la implementación Prototipo Nº5	1 hora
5.6 Revisión de la implementación Prototipo Nº6	1 hora
5.7 Revisión de la implementación Prototipo Nº7	1 hora
5.8 Revisión de la implementación Prototipo Nº8	1 hora
5.9 Revisión de la implementación Prototipo Nº9	1 hora
5.10 Revisión de la implementación Prototipo Nº10	1 hora
Plan de pruebas	**101 horas**
6.1. Definir pruebas unitarias	15 horas
6.2. Pruebas unitarias Prototipo Nº1	5 horas
6.3 Pruebas unitarias Prototipo Nº2	1 hora
6.4. Pruebas unitarias Prototipo Nº3	10 horas
6.5. Pruebas unitarias Prototipo Nº4	10 horas
6.6. Pruebas unitarias Prototipo Nº5	10 horas
6.7. Pruebas unitarias Prototipo Nº6	10 horas
6.8. Pruebas unitarias Prototipo Nº7	10 horas
6.9. Pruebas unitarias Prototipo Nº8	10 horas
6.10. Pruebas unitarias Prototipo Nº9	10 horas
6.11. Pruebas unitarias Prototipo Nº10	10 horas
Documentación	**77 horas**
7.1. Documentación captura de requisitos	17 horas
7.2. Documentación del diseño	5 horas
7.3. Documentación de la implementación de prototipos	5 horas
7.4. Documentación de la implementación del plan de pruebas	10 horas
7.5. Finalización de la documentación	20 horas
7.6. Preparar la defensa del proyecto	20 horas
Total	**459 horas**

Tabla 1 - Planificación temporal

La estimación indica que se tardarán 459 horas en completar el desarrollo del proyecto. Contando que se dedicará una media de 15 horas por semana aproximadamente, la duración del proyecto se extenderá a las 31 semanas.

El reparto del trabajo durante la semana no será equitativo, y los fines de semana o festivos también serán días de trabajo. En el siguiente Gantt se muestra la distribución en el tiempo del reparto de horas.

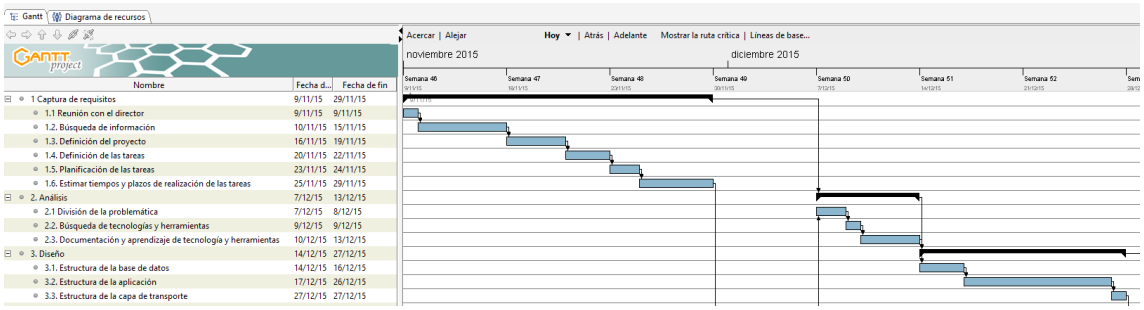

Ilustración 4 - Gantt (I)

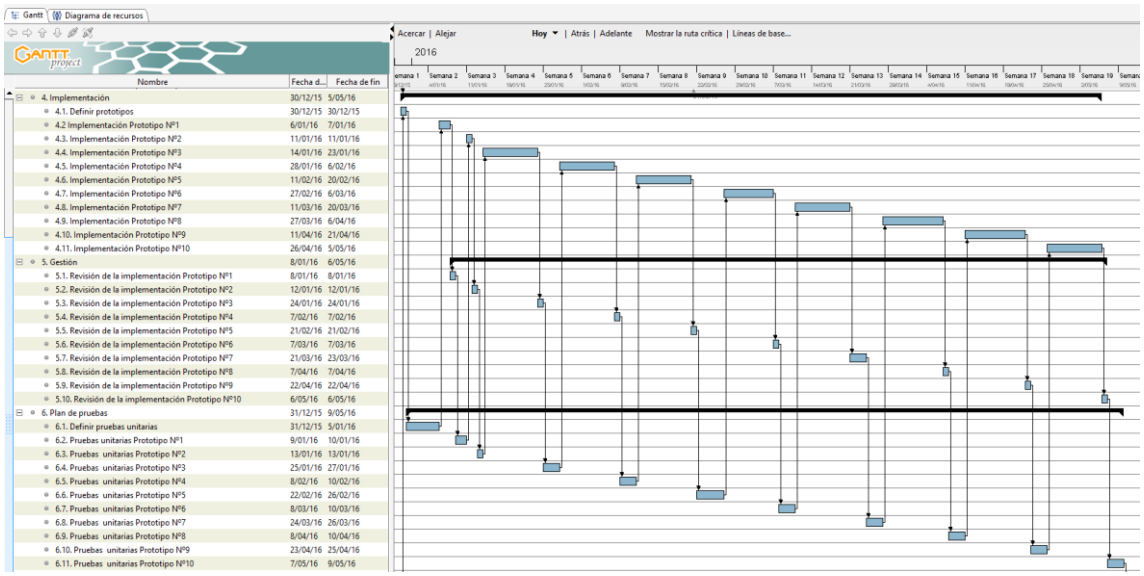

Ilustración 5 - Gantt (II)

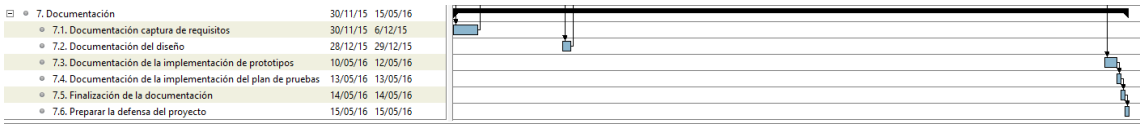

Ilustración 6 - Gantt (III)

2.4.5 Gestión de riesgos

En todo proyecto surgen imprevistos, los cuales, es importante tener identificados en su mayor medida, para poder determinar cómo responder ante ellos en caso de sufrirlos. No existe el riesgo 0 en un proyecto, y de la misma manera, tampoco se puede cubrir el 100% de los posibles riesgos. Es importante guardar un equilibrio y tratar de identificar aquellos riesgos que puedan ocasionar más problemas al desarrollo del proyecto.

Problemas con el Software

Probabilidad: Media.

Impacto: Alta.

Consecuencia: Retraso en la ejecución de las tareas establecidas.

Prevención: Tener siempre las aplicaciones utilizadas con la última versión recomendada por el proveedor de la o las soluciones usadas.

Plan de contingencia: Recuperar el tiempo perdido en siguientes etapas.

Pérdida de datos

Probabilidad: Media.

Impacto: Alta.

Consecuencia: Retraso en la ejecución de las tareas establecidas.

Prevención: Tener un plan de mantenimiento de copias de seguridad establecido. Realizar una copia completa cada sábado y copias incrementales durante la semana.

Plan de contingencia: Realizar copias completas o incrementales de forma semanal para asegurar de ésta manera la pérdida de la menor información.

Problemas con el Hardware

Probabilidad: Baja.

Impacto: Medio.

Consecuencia: Retraso en la ejecución de las tareas.

Prevención: Dar un buen uso al hardware. Tener el equipo en las condiciones óptimas para que su rendimiento sea el correcto. Tener vigiladas las temperaturas y controlar la correcta ventilación del sistema.

Plan de contingencia: Sustituir el hardware que ocasiona los problemas, y recuperar el tiempo perdido para entrar en plazos de ejecución de las tareas.

Caídas de la luz

Probabilidad: Baja.

Impacto: Alta.

Consecuencia: Pérdida de información si no ha sido guardada.

Prevención: Utilizar una SAI.

Plan de contingencia: Recuperar el último backup realizado.

Problemas externos; fallos en actualización de librerías, contacto con el soporte para resolución de incidencias

Probabilidad: Media.

Impacto: Alta.

Consecuencia: Retraso en la ejecución de las tareas.

Prevención: Mantener las librerías actualizadas y llevar un control del feedback proporcionado por el proveedor, de esta manera se podrá determinar en qué medida estos cambios pueden afectar al desarrollo del proyecto.

Plan de contingencia: Solucionar los problemas tramitando incidencias a los proveedores de las librerías si procede, y recuperar el tiempo perdido para entrar en plazos de ejecución de las tareas.

Fallar en los plazos de ejecución de las tareas

Probabilidad: Alta.

Impacto: Alta.

Consecuencia: No cumplir con lo establecido en la planificación temporal.

Prevención: Tener un margen de horas para poder utilizar en caso de que se sufran retrasos en la ejecución de las tareas.

Plan de contingencia: Recuperar el tiempo perdido en otras etapas en las que la carga de trabajo sea menor.

Problemas personales

Probabilidad: Baja.

Impacto: Alta.

Consecuencia. Retraso en la ejecución de las tareas.

Prevención: No se puede determinar la aparición de los problemas personales.

Plan de contingencia: Dentro de lo posible, si se sufren, intentar recuperar el tiempo perdido en las etapas posteriores.

Problemas de análisis y diseño

Probabilidad: Media-Alta.

Impacto: Alta.

Consecuencia: Retraso en la ejecución de las tareas.

Prevención: No existe el análisis y diseño perfecto, siempre se encontrarán formas más eficientes de hacer las cosas a medida que el desarrollo avanza. Se dedicará el tiempo necesario para realizar un buen análisis y diseño.

Plan de contingencia: Replantear la problemática y buscar alternativas viables para mejorar lo existente.

2.4.6 Viabilidad económica

Todo proyecto tiene un coste. A continuación se muestra el análisis de la viabilidad económica, lo que costará hacer el proyecto y de qué manera se piensa recuperar la inversión realizada. El análisis se divide en diferentes puntos a tratar.

Personal

El proyecto será realizado por una persona cualificada. Teniendo en cuenta que un analista/programador cobra 22 euros/hora y que el proyecto se ha realizado en 459 horas. El total del montante asciende a **10.098 euros**.

Hardware

Ordenador portátil: Se empleará un portátil exclusivamente para el desarrollo del proyecto valorado en 950 euros, con una vida media de 6 años y un uso del 40%.

- Amortización anual:
  950 euros / 6 años = **159 euros**.

- Gastos del portátil:

  ((159 euros x 13 semanas) x 40%) / 52 semanas = **16 euros**.

Ordenador de sobremesa: Se empleará un ordenador de sobremesa valorado en 1400 euros, con una vida media de 8 años y un uso del 20%.

- Amortización anual:
  1400 euros / 8 años = **175 euros**.

- Gastos del ordenador de sobremesa:

  ((175 euros x 13 semanas) x 20%) / 52 semanas = **9 euros**.

Software

Licencia Windows 7: Incluida en el precio de los ordenadores.

Linux, Ubuntu: Gratuita.

Licencia Office 2013: Incluida en el precio de los ordenadores.

Vmware Player: Gratuita.

Postman: Gratuita.

Sublime Text2: Gratuita.

Gantt Project: Gratuita.

Cacoo: Gratuita.

NodeJS: Gratuita.

Otros:

Gastos en desplazamientos, luz y fotocopias: La estimación de los gastos producidos por los desplazamientos para las reuniones, la luz consumida por los ordenadores y las fotocopias realizadas será de un 5% sobre el resto de gastos, y su valor asciende a 500 euros.

Gastos totales	Euros
Personal	***10.098***
Hardware	***25***
Ordenador portátil	*16*
Ordenador sobremesa	*9*
Software	*0*
Otros	***500***
Total	***10.623***

Tabla 2 - Gastos de evaluación económica

Recuperar la inversión:

El proyecto está destinado a un uso educativo, nunca a la venta. Por lo tanto, no se contempla el beneficio económico en el desarrollo del mismo.

3 ANTECEDENTES

La evolución de los Servicios web en los últimos tiempos ha sido considerable y con ello la aparición de nuevas tecnologías.

Frente al tradicional protocolo de comunicaciones SOAP (**S**imple **O**bject **A**ccess **P**rotocol) del año 1998, el cual, permite una comunicación entre dos objetos mediante el envío de datos XML, nacen otro tipo de tecnologías que vienen a resolver, mejorar o agilizar la problemática encontrada hasta el momento.

SOAP es un protocolo de comunicaciones muy potente, y puede utilizarse para infinidad de tareas, pero el hecho de que esté pensado para poder ser utilizado para casi todo, hace que muchas veces complique el desarrollo de las tareas sencillas.

La arquitectura *REST*, no siendo mucho más joven que *SOAP*, año 2000, define un estilo de arquitectura, en el cual se definen una serie de reglas, las cuales tienen que ser seguidas y cumplidas por aquellas aplicaciones que deseen incorporar *REST* en su sistema:

Cliente/servidor

Existe una separación clara de funcionalidades entre la parte del cliente y la parte servidora. Las dos partes serán totalmente independientes.

Stateless, "sin estado"

El servidor no reconoce la situación de un solicitante de un recurso una vez que éste devuelve la respuesta. Este detalle corre a cuenta del usuario, el cual deberá decir en cada solicitud al servidor quien es, mediante un usuario y contraseña, o mediante un Token de seguridad.

Cacheable

Mediante el atributo Cache-Control del encabezado se determina si se desea cachear las respuestas en local.

Sistema por capas

Independencia de capas. El usuario no tiene que saber por qué capas se mueven los datos que envía.

Interfaz uniforme

No importa quién solicite o reciba las peticiones, siempre y cuando se respete el punto intermedio, la interfaz, en la cual se definen de una manera determinada la manera en la que se deben enviar y recibir los datos. Todas las peticiones o respuestas que modifiquen el patrón original establecido no serán correctas. Teniendo claros estos conceptos, aparecen los *Servicios web RESTful*, que hacen referencia un Servicio web que sigue los principios de una arquitectura *REST*. Sus principales características son las siguientes:

URI del recurso

La API estará formada por recursos los cuales se traducen en Urls.

Representación del recurso

La respuesta de los datos puede darse en diferentes formatos como pueden ser JSON, XML o TXT, definiendo en la cabecera el parámetro "Content-type".

Operaciones

También llamados verbos de operación, los verbos determinan las operaciones que se pueden realizar sobre los recursos definidos en la API. Operaciones de GET, PUT, POST, DELETE, entre otros.

Usar un *Servicio web RESTful* o en su defecto *REST*, simplifica las cosas y las estandariza. Precisamente esa es su principal baza y la que le ha llevado hoy en día a convertirse en el protocolo de comunicaciones más usado en cuanto a los Servicios web se refiere, y por ende, la arquitectura seleccionada para el desarrollo del proyecto AdESMuS**.**

Entre los ejemplos de uso, los más destacados y conocidos, son las APIs de las diferentes redes sociales, como puede ser: Twitter, Facebook o Instagram.

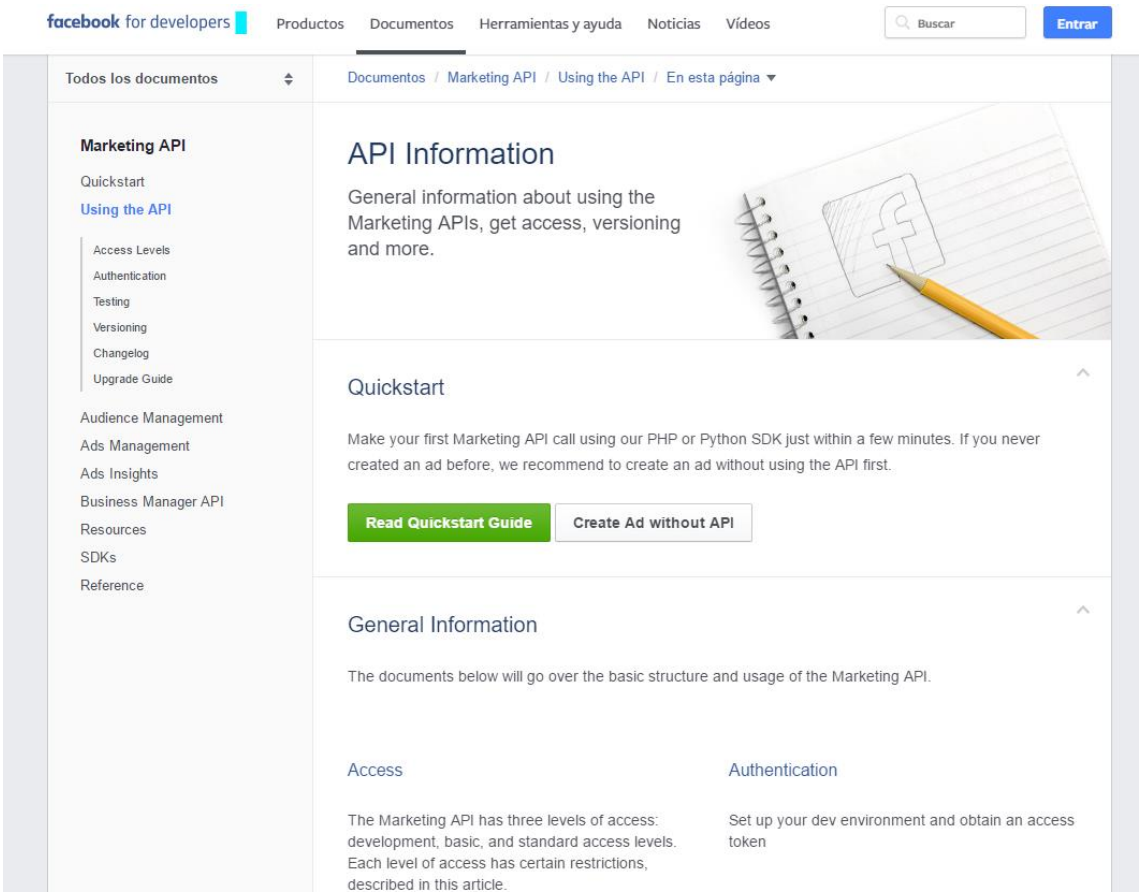

Ilustración 7 - Ejemplo de API, Facebook (https://developers.facebook.com/docs/marketing-api/using-the-api)

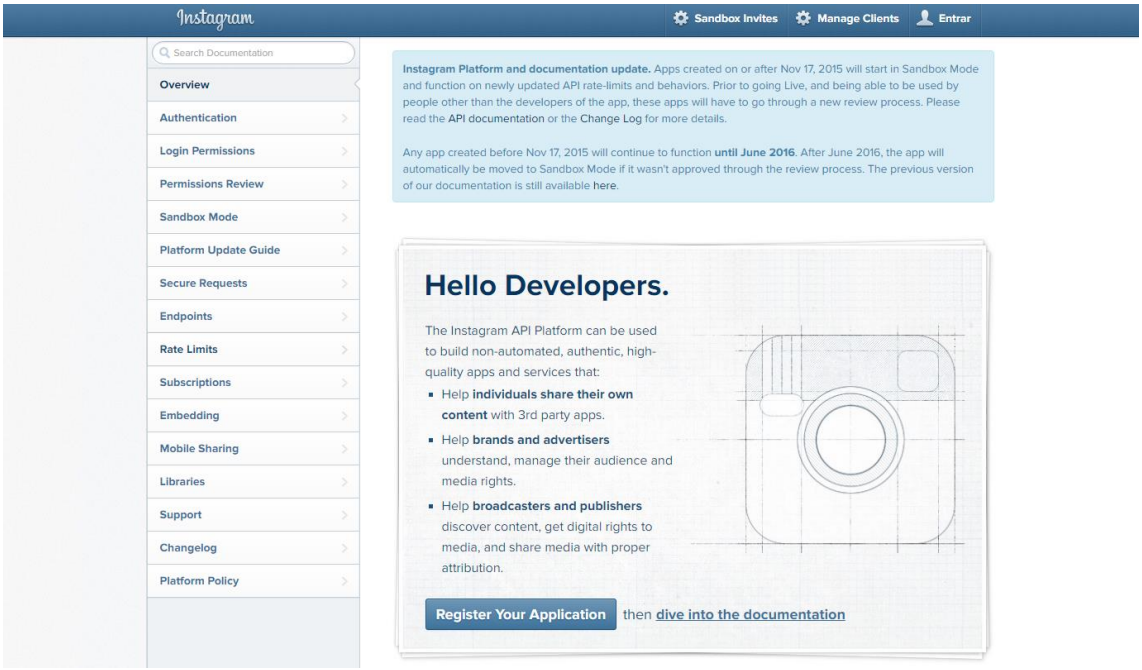

Ilustración 8 - Ejemplo de API, Instagram (https://www.instagram.com/developer/)

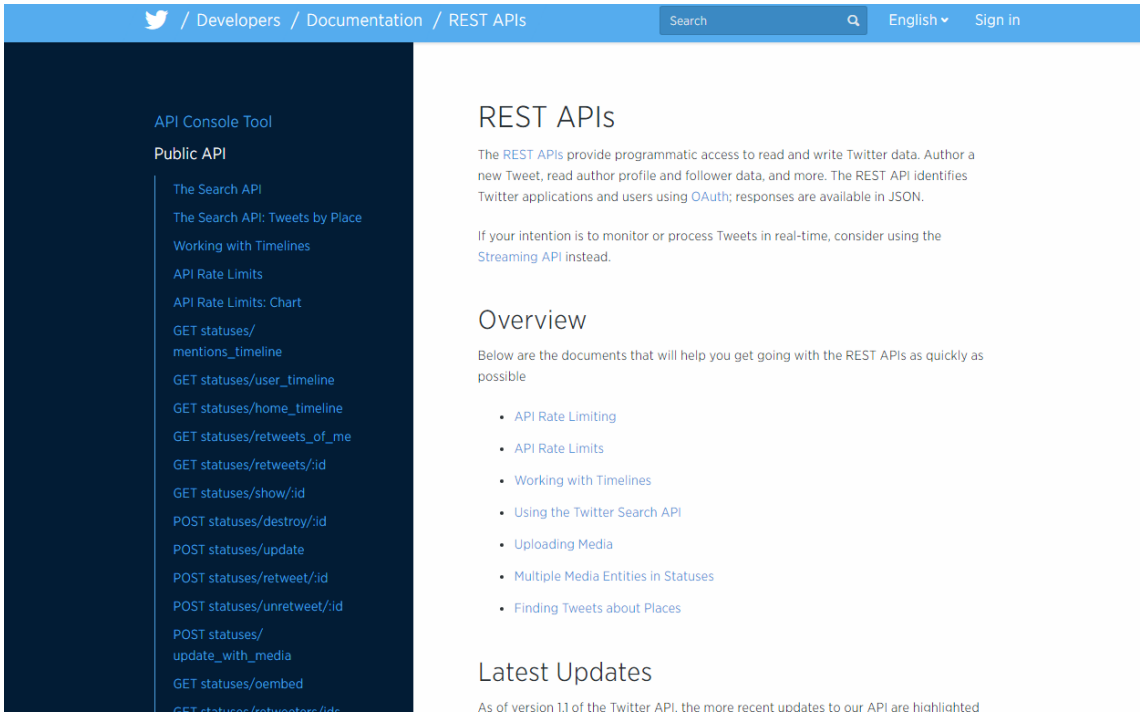

Ilustración 9 - Ejemplo de API, Twitter (https://dev.twitter.com/rest/public)

Hoy en día existen muchos lenguajes de programación que permiten mediante Frameworks o Microframeworks crear Servicios web RESTful.

Para el desarrollo de AdESMuS se ha pensado en una tecnología novedosa relativamente, pero no por ello menos potente. NodeJS es un entorno de JavaScript de alto rendimiento, que hace uso del motor v8 de JavaScript creado por Google, se ejecuta del lado del servidor y está basado en eventos. La principal ventana de NodeJS frente a otros lenguajes de programación, como puede ser PHP o Java, es que todas las operaciones de E/S se realizan de forma totalmente asíncrona, evitando los bloqueos. NodeJS no necesita tener instalado un servidor local para su funcionamiento. Dispone de módulos básicos que permiten implementar servidores web de una manera sencilla y rápida.

Todas las características citadas anteriormente, inclinaron la balanza a favor de la elección de NodeJS, que además, se sitúa como tecnología puntera en el desarrollo de APIs, dispone de una excelente comunidad y una amplia documentación.

4 CAPTURA DE REQUISITOS

En este apartado se mostrarán y explicaran: la jerarquía de los actores, los casos de uso y el modelo de dominio.

4.1 Jerarquía de actores y casos de uso

Jerarquía de actores

En la siguiente figura se muestra la jerarquía de los actores.

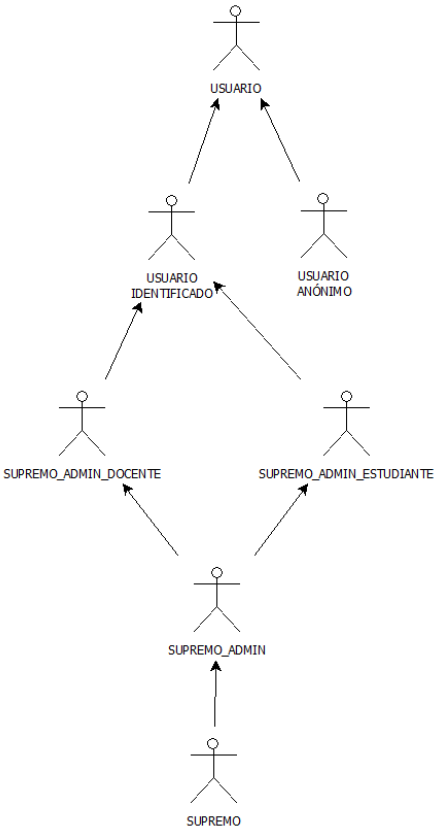

Ilustración 10 - Jerarquía de actores

Usuario

Es el actor que se encarga de ejecutar aquellas acciones, las cuales, se encuentran disponibles para cualquier tipo de usuario. Un usuario sin identificarse o un usuario identificado en el sistema (Supremo, Administrador, Docente y Estudiante).

Usuario anónimo

Es el actor que se encarga de ejecutar aquellas acciones, las cuales, se encuentran disponibles para aquellos usuarios que no se encuentren identificados en el sistema.

Usuario identificado

Es el actor que ejecuta aquellas acciones, que solamente están disponibles para aquellos usuarios que se encuentren identificados en el sistema con el rol de: Supremo, Administrador, Docente y Estudiante.

Supremo_Admin_Docente

Es el actor que se encarga de ejecutar aquellas acciones, las cuales, pueden ser ejecutadas por un usuario identificado en el sistema con el rol de: Supremo, Administrador o Docente.

Supremo_Admin_Estudiante

Es el actor que se encarga de ejecutar aquellas acciones, las cuales, pueden ser ejecutadas por un usuario identificado en el sistema con el rol de: Supremo, Administrador o Estudiante.

Supremo_Admin

Actor encargado de ejecutar las acciones que únicamente se encuentran disponibles para un usuario identificado en el sistema con el rol de: Supremo o Administrador.

Supremo

Es el actor que ejecuta todas aquellas acciones que únicamente se encuentran disponibles para un usuario identificado con rol: Supremo.

Casos de uso

En las siguientes figuras se muestra el modelo de los casos de uso separado por los actores.

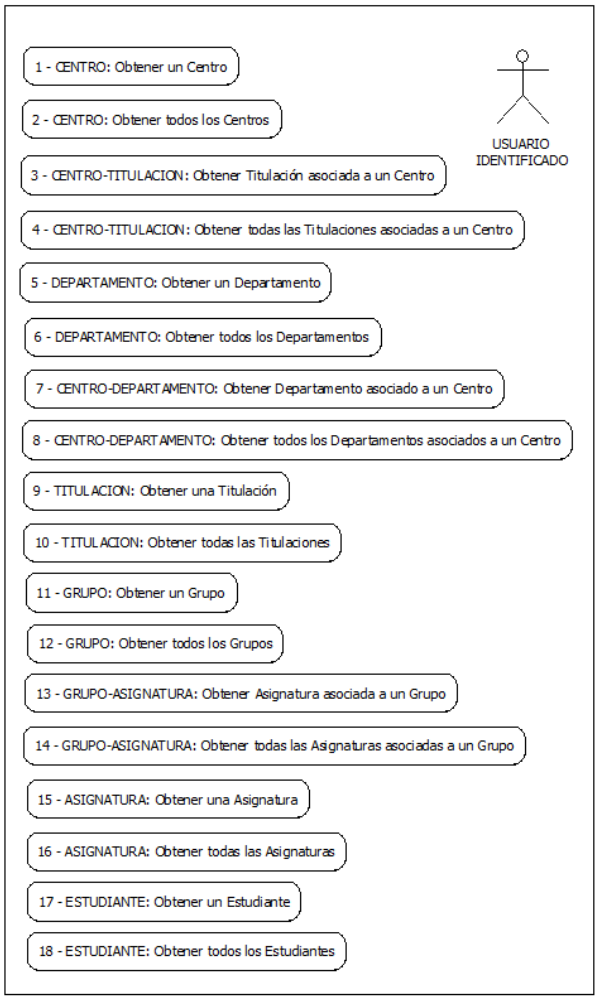

Ilustración 11 - Casos de uso (I)

1 – Obtener un centro

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos sobre un Centro.

2 – Obtener todos los centros

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todos los centros registrados en el sistema.

3 – Obtener Titulación asociada a un Centro

Recurso disponible para el actor *Usuario Identificado*, que permite obtener la asociación de una Titulación a un Centro.

4 – Obtener todas las Titulaciones asociadas a un Centro

Recurso disponible para el actor *Usuario Identificado*, que permite obtener todas las Titulaciones que se encuentran asociadas a un Centro.

5 – Obtener un Departamento

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de un Departamento.

6 – Obtener todos los Departamentos

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todos los departamentos registrados en el sistema.

7 – Obtener Departamento asociado a un Centro

Recurso disponible para el actor *Usuario Identificado*, que permite obtener la asociación de un Departamento a un Centro.

8 – Obtener todos los Departamentos asociados a un Centro

Recurso disponible para el actor *Usuario Identificado*, que permite obtener todos los Departamentos que se encuentran asociadas a un Centro.

9 – Obtener una Titulación

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos sobre una Titulación.

10 – Obtener todas las Titulaciones

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todas las Titulaciones registradas en el sistema.

11 – Obtener un Grupo

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos sobre un Grupo.

12 – Obtener todos los Grupos

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todos los Grupos registrados en el sistema.

13 – Obtener Asignatura asociada a un Grupo

Recurso disponible para el actor *Usuario Identificado*, que permite obtener la asociación de una Asignatura a un Grupo.

14 – Obtener todas las Asignaturas asociadas a un Grupo

Recurso disponible para el actor *Usuario Identificado*, que permite obtener todas las Asignaturas que se encuentran asociadas a un Grupo.

15 – Obtener una Asignatura

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos sobre una Asignatura.

16 – Obtener todas las Asignaturas

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todas las Asignaturas registradas en el sistema.

17 – Obtener un Estudiante

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos sobre un Estudiante.

18 – Obtener todos los Estudiantes

Recurso disponible para el actor *Usuario Identificado*, que permite obtener datos de todos los Estudiantes registrados en el sistema.

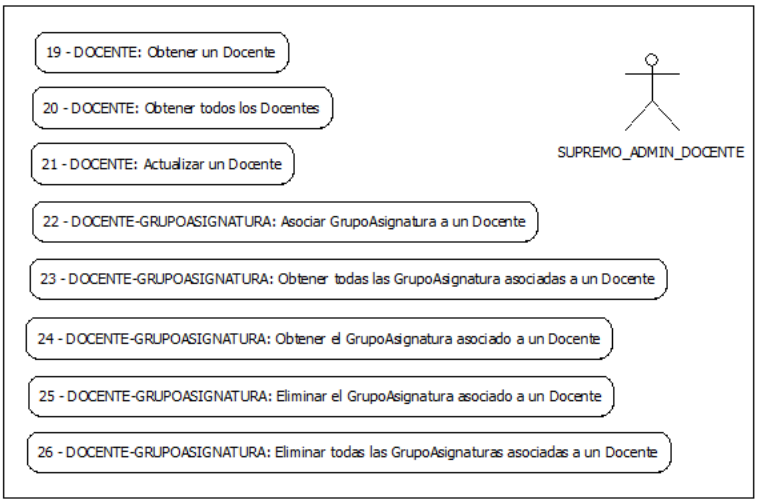

Ilustración 12 - Casos de uso (II)

19 – Obtener un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite obtener datos sobre un Docente.

20 – Obtener todos los Docentes

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite obtener datos de todos los Docentes registrados en el sistema.

21 – Actualizar un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite actualizar los datos de un Docente registrado en el sistema.

22 – Asociar GrupoAsignatura a un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite asociar un GrupoAsignatura a un Docente.

23 – Obtener todas las GrupoAsignatura asociadas a un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite obtener todas las asociaciones GrupoAsignatura de un Docente.

24 - Obtener el GrupoAsignatura asociado a un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite obtener la asociación de un GrupoAsignatura de un Docente.

25 – Eliminar el GrupoAsignatura asociado a un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite eliminar la asociación de un GrupoAsignatura a un Docente.

26 – Eliminar todas las GrupoAsignatura asociadas a un Docente

Recurso disponible para el actor *Supremo_Admin_Docente*, que permite eliminar todas las asociaciones de GrupoAsignatura que tenga un Docente.

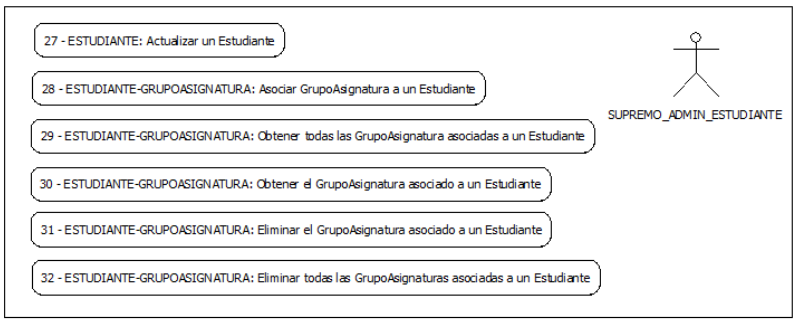

Ilustración 13 - Casos de uso (III)

27 – Actualizar un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite actualizar los datos de un Estudiante registrado en el sistema.

28 – Asociar GrupoAsignatura a un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite asociar un GrupoAsignatura a un Estudiante.

29 – Obtener todas las GrupoAsignatura asociadas a un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite obtener todas las asociaciones GrupoAsignatura de un Estudiante.

30 - Obtener el GrupoAsignatura asociado a un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite obtener la asociación de GrupoAsignatura de un Estudiante.

31 – Eliminar el GrupoAsignatura asociado a un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite eliminar la asociación de un GrupoAsignatura a un Estudiante.

32 – Eliminar todas las GrupoAsignatura asociadas a un Estudiante

Recurso disponible para el actor *Supremo_Admin_Estudiante*, que permite eliminar todas las asociaciones de GrupoAsignatura que tenga un Estudiante.

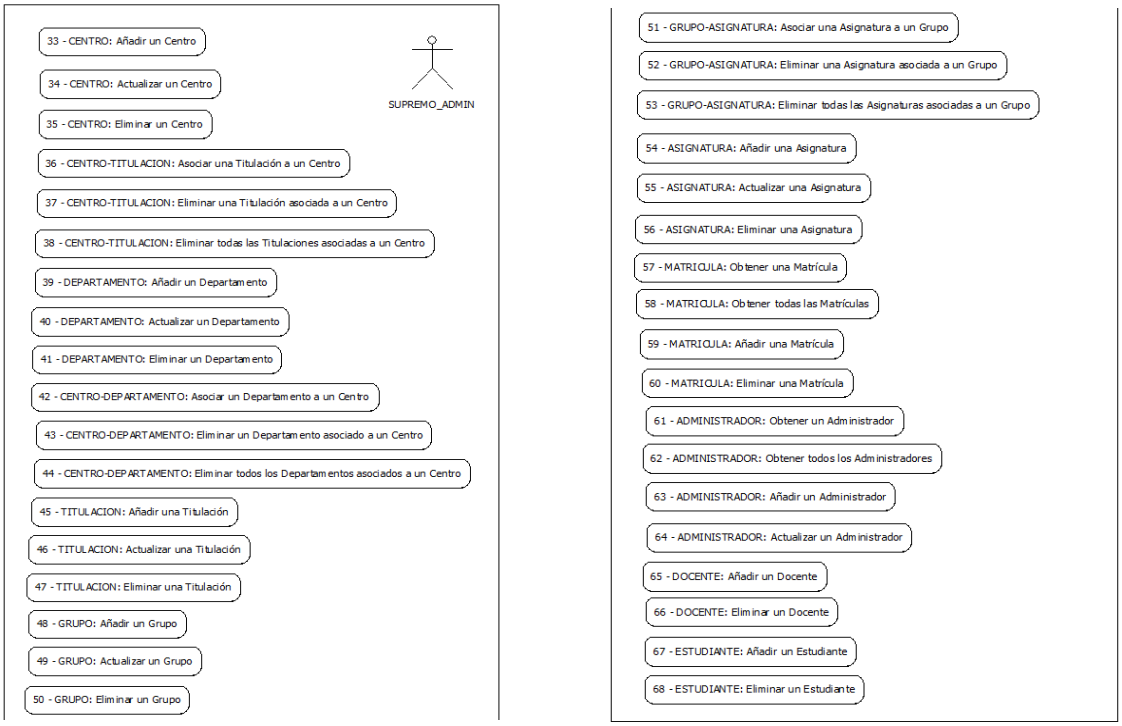

Ilustración 14 - Casos de uso (IV)

33 – Añadir un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Centro en el sistema.

34 – Actualizar un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de un Centro registrado en el sistema.

35 – Eliminar un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar un Centro registrado en el sistema.

36 – Asociar una Titulación a un Centro

Recurso disponible para el actor Supremo_Admin, que permite asociar una Titulación a un Centro.

37 – Eliminar una Titulación asociada a un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar la asociación de una Titulación asociada a un Centro.

38 – Eliminar todas las Titulaciones asociadas a un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar todas las Titulaciones asociadas a un Centro.

39 – Añadir un Departamento

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Departamento en el sistema.

40 – Actualizar un Departamento

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de un Departamento registrado en el sistema.

41 – Eliminar un Departamento

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar un Departamento del sistema.

42 – Asociar un Departamento a un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite asociar un Departamento a un Centro.

43 – Eliminar un Departamento asociado a un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar la asociación de un Departamento a un Centro.

44 – Eliminar todos los Departamentos asociados a un Centro

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar todos los Departamentos asociados a un Centro.

45 – Añadir una Titulación

Recurso disponible para el actor *Supremo_Admin*, que permite registrar una Titulación en el sistema.

46 – Actualizar una Titulación

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de una Titulación registrada en el sistema.

47 – Eliminar una Titulación

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar una Titulación registrada en el sistema.

48 – Añadir un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Grupo en el sistema.

49 – Actualizar un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de un Grupo registrado en el sistema.

50 – Eliminar un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar un Grupo registrado en el sistema.

51 – Asociar una Asignatura a un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite asociar una Asignatura a un Grupo.

52 – Eliminar una Asignatura asociada a un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar la asociación de una Asignatura a un Grupo.

53 – Eliminar todas las asignaturas asociadas a un Grupo

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar todas las Asignaturas asociadas a un Grupo.

54 – Añadir una Asignatura

Recurso disponible para el actor *Supremo_Admin*, que permite registrar una Asignatura en el sistema.

55 – Actualizar una Asignatura

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de una Asignatura registrada en el sistema.

56 – Eliminar una Asignatura

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar una Asignatura del sistema.

57 – Obtener una Matrícula

Recurso disponible para el actor *Supremo_Admin*, que permite obtener los datos de una Matrícula registrada en el sistema.

58 – Obtener todas las Matrículas

Recurso disponible para el actor *Supremo_Admin*, que permite obtener los datos de todas las Matriculas registradas en el sistema.

59 – Añadir una Matrícula

Recurso disponible para el actor *Supremo_Admin*, que permite registrar una Matrícula en el sistema.

60 – Eliminar una Matrícula

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar una Matrícula del sistema.

61 – Obtener un Administrador

Recurso disponible para el actor *Supremo_Admin*, que permite obtener los datos de un Administrador registrado en el sistema.

62 – Obtener todos los Administradores

Recurso disponible para el actor *Supremo_Admin*, que permite obtener los datos de todos los Administradores registrados en el sistema.

63 – Añadir un Administrador

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Administrador en el sistema.

64 – Actualizar un Administrador

Recurso disponible para el actor *Supremo_Admin*, que permite actualizar los datos de un Administrador registrado en el sistema.

65 – Añadir un Docente

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Docente en el sistema.

66 – Eliminar un Docente

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar un Docente registrado en el sistema.

67 – Añadir un Estudiante

Recurso disponible para el actor *Supremo_Admin*, que permite registrar un Estudiante en el sistema.

68 – Eliminar un Estudiante

Recurso disponible para el actor *Supremo_Admin*, que permite eliminar un Estudiante del sistema.

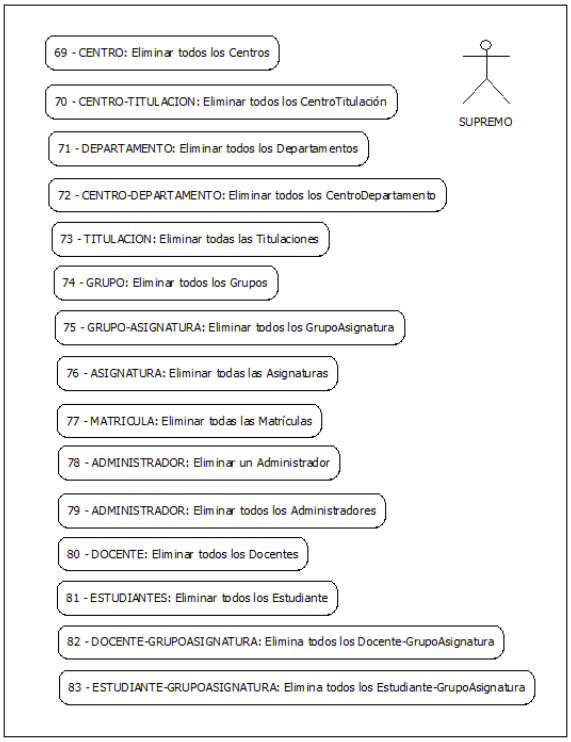

Ilustración 15 - Casos de uso (V)

69 – Eliminar todos los Centros

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Centros registrados en el sistema. Vaciado de la tabla.

70 – Eliminar todos los CentroTitulación

Recurso disponible para el actor *Supremo*, que permite eliminar todos los CentroTitulación registrados en el sistema. Vaciado de la tabla.

71 – Eliminar todos los Departamentos

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Departamentos registrados en el sistema. Vaciado de la tabla.

72 – Eliminar todos los CentroDepartamento

Recurso disponible para el actor *Supremo*, que permite eliminar todos los CentroDepartamento registrados en el sistema. Vaciado de la tabla.

73 – Eliminar todas las Titulaciones

Recurso disponible para el actor *Supremo*, que permite eliminar todas las Titulaciones registradas en el sistema. Vaciado de la tabla.

74 – Eliminar todos los Grupos

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Grupos registrados en el sistema. Vaciado de la tabla.

75 – Eliminar todos los GrupoAsignatura

Recurso disponible para el actor *Supremo*, que permite eliminar todos los GrupoAsignatura registrados en el sistema. Vaciado de la tabla.

76 – Eliminar todas las Asignaturas

Recurso disponible para el actor *Supremo*, que permite eliminar todas las Asignaturas registradas en el sistema. Vaciado de la tabla.

77 – Eliminar todas las Matrículas

Recurso disponible para el actor *Supremo*, que permite eliminar todas las Matrículas registradas en el sistema. Vaciado de la tabla.

78 – Eliminar un Administrador

Recurso disponible para el actor *Supremo*, que permite eliminar un Administrador registrado en el sistema.

79 – Eliminar todos los Administradores

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Administradores registrados en el sistema. Vaciado de la tabla.

80 – Eliminar todos los Docentes

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Docentes registrados en el sistema. Vaciado de la tabla.

81 – Eliminar todos los Estudiantes

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Estudiantes registrados en el sistema. Vaciado de la tabla.

82 – Eliminar todos los Docente-GrupoAsignatura

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Docente-GrupoAsignatura registrados en el sistema. Vaciado de la tabla.

83 – Eliminar todos los Estudiante-GrupoAsignatura

Recurso disponible para el actor *Supremo*, que permite eliminar todos los Estudiante-GrupoAsignatura registrados en el sistema. Vaciado de la tabla.

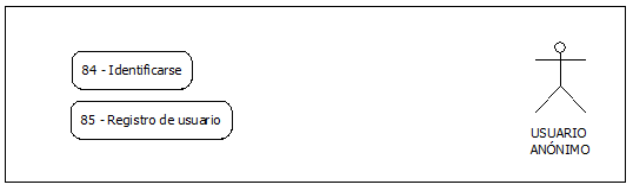

Ilustración 16 - Casos de uso (VI)

84 – Identificarse

Recurso disponible para el actor *Usuario Anónimo*, que permite identificarse en el sistema para obtener un Token válido y poder acceder a los recursos que tenga establecido su Rol de usuario.

85 – Registro de usuario

Recurso disponible para el actor *Usuario Anónimo*, que permite registrar el primer usuario del sistema, el cual será único y tendrá el Rol de Supremo. Después de realizar el primer registro, éste recurso quedará deshabilitado y será inaccesible para cualquier actor.

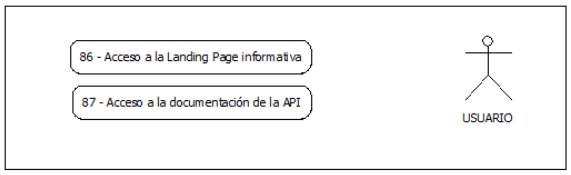

Ilustración 17 - Casos de uso (VII)

86 – Acceso a la Landing Page informativa

Todos los actores tendrán acceso a la visualización de la Landing Page, donde podrán encontrar un pequeño resumen de lo que es AdESMuS.

87 – Acceso a la documentación de la API

Todos los actores tendrán acceso al recurso que proporciona la documentación para la manipulación de la API.

4.2 Modelo de dominio, representación UML

En la siguiente ilustración se muestra el modelo de dominio de AdESMuS en su representación UML. A continuación se explicará de forma detallada el significado de cada una de las entidades y la relación que existe entre ellas.

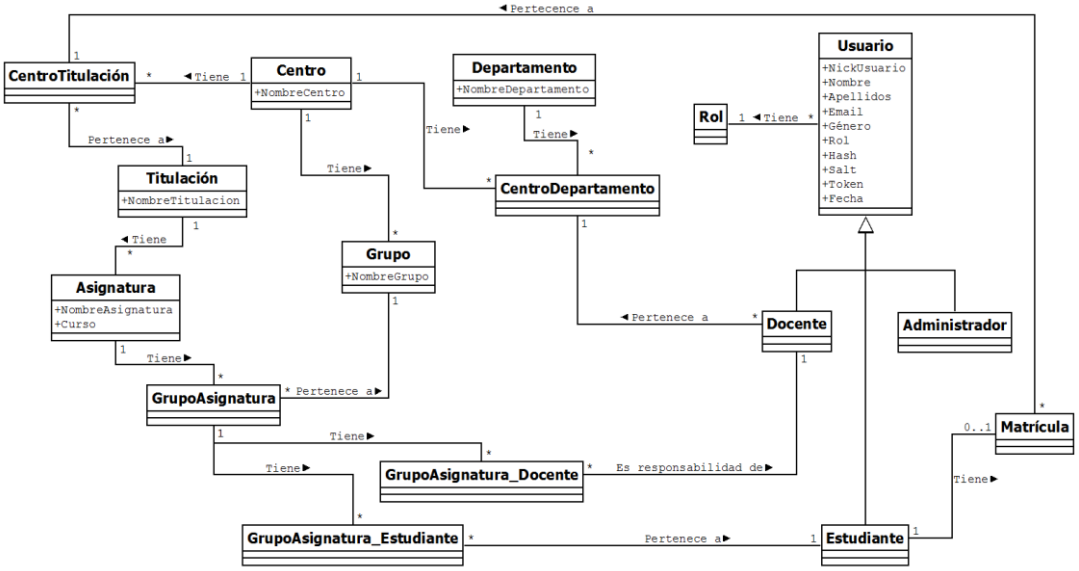

Ilustración 18 - Modelo de dominio, UML

Centro

La entidad Centro representa todos los Centros que se encuentran registrados en el sistema. Dispone de un parámetro *nombreCentro* que permite registrar el nombre del centro.

Titulación

La entidad Titulación representa las titulaciones que se encuentran registradas en el sistema. Dispone de un parámetro *nombreTitulacion* que permite registrar el nombre de la titulación.

CentroTitulacion

La entidad CentroTitulación representa las titulaciones que se encuentran asociadas a un centro.

Departamento

La entidad Departamento representa los departamentos que se encuentran registrados en el sistema. Dispone de un parámetro *nombreDepartamento* que permite registrar el nombre del departamento.

CentroDepartamento

La entidad CentroDepartamento representa los departamentos que se encuentran asociados a un centro.

Asignatura

La entidad Asignatura representa las asignaturas que se encuentran registradas en las diferentes titulaciones. Una asignatura tiene que estar asociada siempre a una titulación. Dispone de un parámetro *nombreAsignatura* que permite registrar el

nombre de la asignatura, y también un parámetro *curso* para determinar de qué curso es la asignatura.

Grupo

La entidad Grupo representa los grupos que se encuentran registrados en el sistema, los cuales pertenecen a uno o varios centros. Todos los grupos estarán asociados a un algún centro. Dispone de un parámetro *nombreGrupo* que permite registrar el nombre del grupo.

GrupoAsignatura

La entidad GrupoAsignatura representa los grupos que se encuentran asociados a las asignaturas.

Matrícula

La entidad Matrícula representa las matrículas registradas en el sistema. Todas las matrículas tendrán asignado un estudiante y un centrotitulación.

GrupoAsignatura_Docente

La entidad GrupoAsignatura_Docente representa las asociaciones de grupoasignatura que tienen vinculados los docentes.

GrupoAsignatura_Estudiante

La entidad GrupoAsignatura_Estudiante representa las asociaciones de grupoasignatura que tienen vinculados los estudiantes.

Usuario

La entidad de Usuario representa la información común de los diferentes tipos de usuarios. Se permitirá registrar un Nick de usuario, nombre, apellidos, email, género del usuario, un rol, y diversa información para su identificación y acceso al sistema, Hash, Salt, Token y fecha de registro.

Rol

La entidad Rol representa los roles disponibles en el sistema, los cuales se asignarán a los usuarios.

Docente

La entidad Docente representa a los docentes que se encuentran registrados en el sistema. La tabla docente hereda de la tabla de usuario.

Estudiante

La entidad Estudiante representa a los estudiantes que se encuentran registrados en el sistema. La tabla estudiante hereda de la tabla de usuario.

Administrador

La entidad Administrador representa a los administradores que se encuentran registrados en el sistema. La tabla de administrador hereda de la tabla de usuario.

5 ANÁLISIS Y DISEÑO

El apartado de análisis y diseño se divide en las siguientes partes.

5.1 Primera aproximación: Modularidad y seguridad

AdESMuS siguiendo los principios de la arquitectura REST se divide principalmente en tres grandes vertientes. En un extremo encontramos la parte servidora, en el otro extremo encontramos los clientes, los cuales son consumidores de los recursos ofrecidos por la API, y en el medio, se encuentra un canal de comunicación, por el cual, fluyen los datos.

Al tratarse de un sistema que se encuentra en todo momento expuesto a la red pública, la seguridad en cualquiera de las partes mencionadas en el punto anterior es extremadamente importante. Gran parte del proyecto estará focalizado a la seguridad, ya bien sea a nivel de servidor (autenticación mediante Token, roles, protección contra posibles ataques a la base de datos), o a nivel de transporte de datos (protección con certificado SSL, tráfico encriptado).

En la siguiente figura se muestra la división del proyecto en las diferentes partes que se explicarán a continuación.

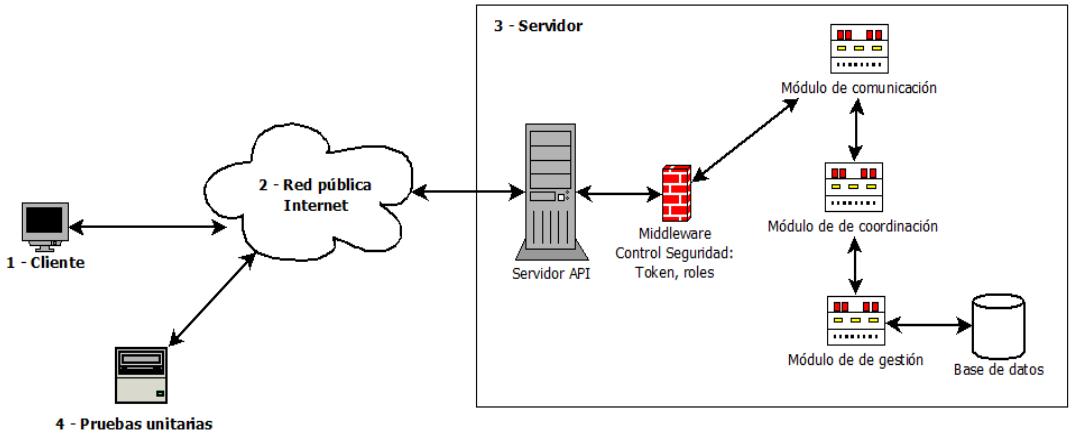

Ilustración 19 - Arquitectura de AdESMuS API

Cliente

Todas las solicitudes de datos que realice el cliente deben incorporar un Token válido en la cabecera de la petición. Este Token debe cumplir exhaustivamente una serie de

requisitos, los cuales, son establecidos en el módulo de seguridad para el control de Tokens que se encuentra en el servidor.

Uno de los parámetros alojados en el Token en el momento de su generación es una huella del navegador. Dicha huella, será enviada por el cliente cuando éste realice el logueo en el sistema. Además de la huella, el cliente debe enviar también los datos del usuario: nombre del usuario y contraseña.

Para la generación de las huellas digitales en la parte del cliente se usará la librería *fingerPrint*, la cual, permite realizar un mix de diferentes parámetros del navegador, como pueden ser, las extensiones que tiene instaladas el usuario, el lenguaje establecido en el navegador, zona horaria, resolución, navegador utilizado o lista de tipografías instaladas, entre otras. La librería generará el hash de forma automática.

Red pública, Internet

Para conseguir una comunicación segura entre los dos extremos, se utiliza un certificado SSL, el cual encripta la capa de transporte de datos, consiguiendo que los datos enviados a través del canal de comunicación sean inteligibles.

Servidor

La parte servidora contará con diferentes niveles o capas de acceso. Primeramente existirá un *Middleware* que hará las funciones de filtro o firewall y se encargará de decidir si las peticiones entrantes son válidas. Para ello, comprobará si el usuario ha enviado un token en la cabecera, y si éste Token cumple con los requisitos establecidos. Por ejemplo, comprobará la fecha de expiración del Token.

Una vez que la petición pasa la barrera del Middleware, es recibida por el *Módulo de comunicación*, o capa de presentación, el cual se encargará de determinar si el recurso solicitado existe en la API, y además, realizará la validación de los datos recibidos (parámetros embebidos en la Url, o parámetros enviados en el cuerpo de petición). Existan o no errores durante el proceso de la solicitud, el *Modulo de comunicación* formará una trama de datos de vuelta, donde dejará constancia del estado de la solicitud, junto al posible cuerpo de la trama con los datos solicitados.

El *Modulo de comunicación*, da por correcta la petición y delega la solicitud a un nivel inferior, el *Modulo de coordinación*, el cual se encarga de hacer las llamadas pertinentes al *Módulo de gestión* para obtener los datos que el usuario ha solicitado.

El *Módulo de gestión*, tiene como única finalidad atender las solicitudes del *Módulo de coordinación*, acceder a la base de datos y obtener los datos solicitados. El *Módulo de gestión* controlará también los diferentes ataques que pueda recibir la base de datos, por ejemplo, la inyección SQL.

5.2 Estructura interna de la aplicación

AdESMuS se creará bajo una estructura inicial proporcionada por el Framework de node.js llamado *Express*.

Express es un Framework de NodeJS basado en JavaScript para el desarrollo de aplicaciones Web: minimalista, flexible y robusto. Entre algunas de sus virtudes, destacar las siguientes:

- **Router Handler**: Posibilidad de utilizar verbos GET, POST, PUT, DELETE para atender peticiones.
- **Middleware**: Funciones que tienen acceso al objeto de solicitud (req) y al objeto de respuesta (res) y que se lanzan siempre que exista una petición. De esta manera podemos controlar las solicitudes antes de introducirlas en el sistema. Un buen ejemplo de uso para AdESMuS: la función *Middleware* se usa para la validación del Token en cada una de las solicitudes para el consumo de los recursos.

Las configuraciones, módulos y helpers creados para AdESMuS se acoplarán a la estructura ya definida en *Express*, manteniendo una jerarquía ordenada y eficiente, para que el sistema sea fácilmente escalable en el futuro.

5.3 Diagrama relacional de la Base de datos

El sistema de gestión de base de datos seleccionado es MySql, el cual, es un sistema relacional, multihilo y multiusuario. La siguiente ilustración muestra el diagrama relacional de la base de datos y las propiedades de las tablas.

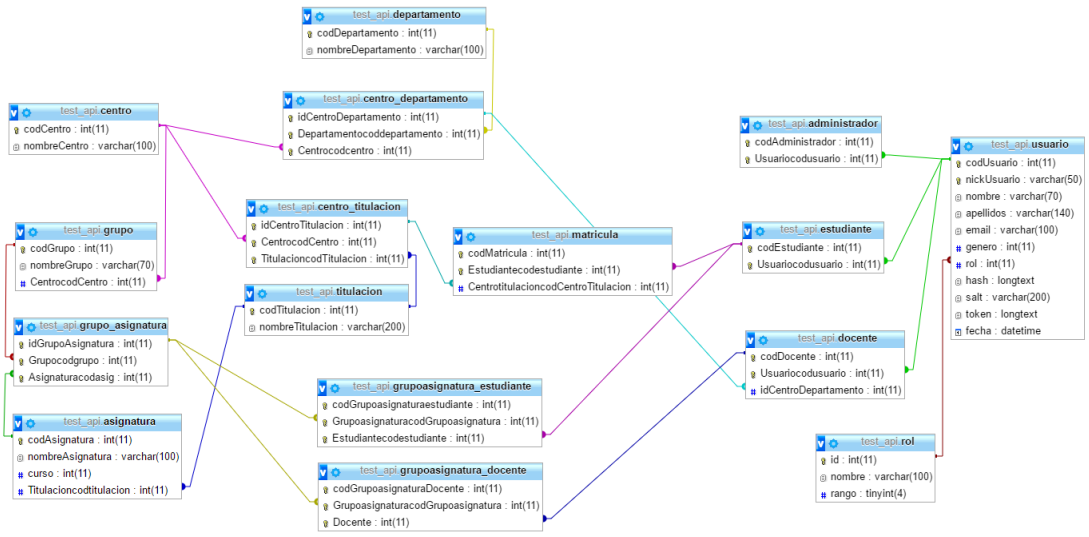

Ilustración 20 - Diagrama relacional de la Base de datos

6 DESARROLLO

En el apartado de desarrollo se explica con un lenguaje natural, lo que se ha hecho en el proyecto, justificando las decisiones tomadas.

6.1 Estructura de la aplicación y puesta en funcionamiento

Como ya se comentó en el punto anterior, AdESMuS parte de un esqueleto básico, el cual, es proporcionado por el Framework de NodeJS, Express. En la siguiente ilustración se puede observar de qué manera se distribuyen inicialmente las carpetas y los ficheros.

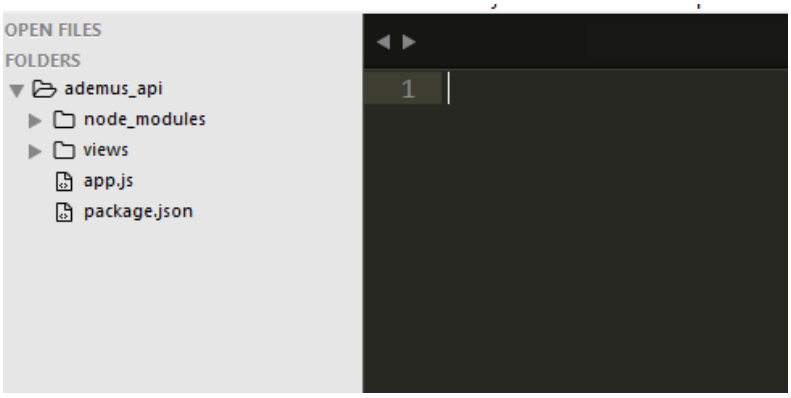

Ilustración 21 - Estructura básica de Express

Lo primero que llama la atención es la sencillez del Framework. El sistema dispone de un fichero de configuración llamado *app.js* donde se alojará toda la información referente a la configuración del servidor. Un fichero *package.json*, el cual, llevará el control de la inyección de las dependencias que se usen en el proyecto (librerías usadas). También dispone de una carpeta con los módulos del sistema llamada *node_modules* (contiene librerías de terceros), una carpeta con las vistas, *views*, y un motor de plantillas para la representación de los datos, por defecto *Jade*, aunque puede ser sustituido por otros.

Partiendo de esta base, se arranca el servidor desde la consola ejecutando el siguiente comando desde la carpeta raíz del proyecto: *nodejs app.js*. Express, trae por defecto configurado un controlador que atiende las peticiones GET por el puerto 3000 en la raíz "/". Cuando el usuario solicite un recurso por Url GET http://dominio:3000/, el servidor responderá con una cadena, en este caso, "Hello World".

Ilustración 22 - Ejemplo básico Express

Con el sencillo ejemplo definido en la ilustración anterior se ha conseguido definir, a grandes rasgos, una parte muy importante en el proyecto, las rutas. Puesto que AdESMuS no dispone de una interfaz gráfica para la interacción, se hace necesario e imprescindible establecer unas reglas en la definición de las rutas para poder realizar la conexión con el servidor y consumir los recursos que éste proporcione.

Un recurso, pone a disposición del cliente, una serie de funcionalidades que le permita obtener o modificar información contenida en un servidor. Estas funcionalidades son definidas y controladas en la parte servidora. Además, el cliente podrá utilizar los verbos *GET*, *POST*, *PUT* y *DELETE* para indicarle al servidor de qué manera desea consumir el recurso. Por ejemplo, el esquema completo de un recurso podría ser el siguiente:

GET http://dominio.com/students

Quiero obtener todos los estudiantes registrados.

GET http://dominio.com/students/123

Quiero obtener el estudiante con id 123.

POST http://dominio.com/students

Quiero registrar un estudiante en el sistema. Te envío los datos del registro en el cuerpo de la petición.

PUT http://dominio.com/students/123

Quiero actualizar el estudiante 123, te envío los datos en el cuerpo de la petición.

DELETE http://dominio.com/students

Quiero eliminar todos los estudiantes registrados.

DELETE http://dominio.com/students/123

Quiero eliminar el usuario 123.

Obviamente, no será tan sencillo obtener la información de un recurso. AdESMuS aplica diferentes técnicas para la protección del sistema y los datos, las cuales, se explicarán durante el desarrollo de este punto. Entre las técnicas aplicadas cabe destacar: autenticación basada por Tokens, encriptación del canal de comunicación, acceso limitado a recursos por control de roles de usuario, etc.

Una vez que el esqueleto básico se pone en funcionamiento, y se comprueba que el sistema de rutas funciona perfectamente, se organiza la jerarquía de la aplicación para adaptarla a las necesidades del proyecto.

AdESMuS, como se comentó en el apartado de Análisis y diseño, está formado por tres módulos claramente diferenciados.

El *Módulo de comunicación* o capa de presentación es el encargado de gestionar las peticiones entrantes y delegar las tareas al *Módulo de coordinación*. El *Módulo de coordinación* atiende a las solicitudes del *Módulo de comunicación* y se encarga de hacer las llamadas pertinentes al *Módulo de gestión* para obtener los datos solicitados. El *Módulo de gestión* atiende a las peticiones del *Módulo de coordinación*, y únicamente se encarga de realizar los accesos a la base de datos para obtener los datos solicitados.

Teniendo claro la funcionalidad de cada uno de los módulos, se define en la raíz de la aplicación una nueva carpeta llamada *adesmus_modules*. Esta carpeta contiene únicamente los módulos creados para la aplicación AdESMuS. Dicha carpeta será totalmente independiente a la carpeta de los módulos del sistema *node_modules*. En la carpeta *adesmus_modules* se crean tres nuevas carpetas, una por cada uno de los tres módulos citados anteriormente. En la siguiente ilustración se puede observar el resultado.

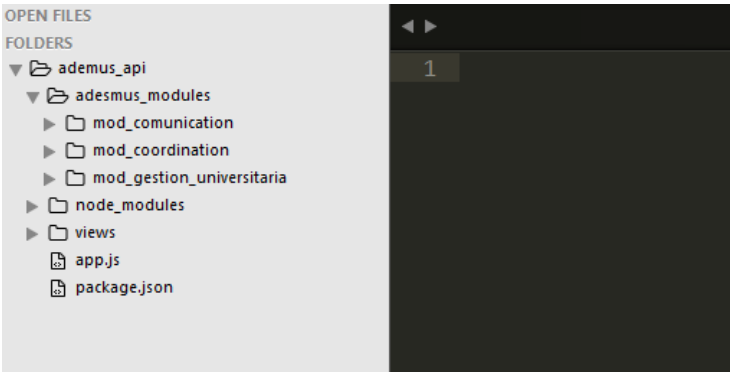

Ilustración 23 - Creación de carpetas para módulos AdESMuS

Además de establecer la jerarquía para los módulos citados anteriormente, también se crearán otros ficheros. Por ejemplo, el fichero *config.js***,** el cual, también se crea en la raíz del proyecto, y en él, se guarda en formato JSON (JavaScript Object Notation, formato de texto ligero para el intercambio de datos), información estática referente al proyecto, como puede ser, la información necesaria para realizar la conexión con la base de datos. Otras carpetas que se crean en la raíz del proyecto son: la carpeta llamada *helpers* que contendrá los scripts con las funcionalidades comunes al proyecto, la carpeta *certs* que contendrá los certificados creados para poder utilizar un canal de

comunicación encriptado (certificado SSL, https), y una última carpeta llamada *doc* que contendrá la documentación de la API. En la siguiente ilustración se muestra el resultado de la jerarquía completa.

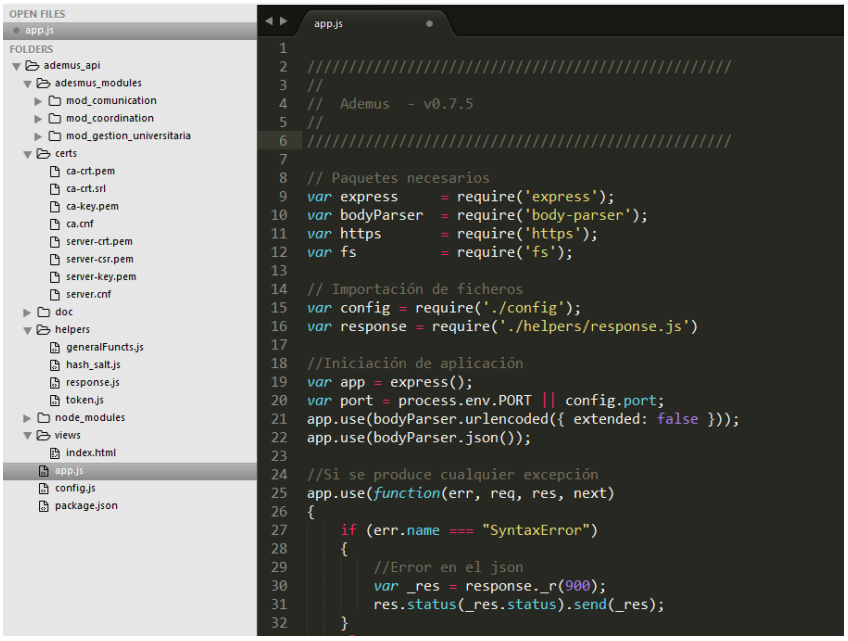

Ilustración 24 - Jerarquía completa

AdESMuS dispone de una base de datos a la cual se tiene acceso a través de NodeJS, por lo tanto, es necesario un conector que permita realizar la conexión entre el recurso y la base de datos. Para ello se ha utilizado la librería *node-mysql*, la cual se instala con el siguiente comando desde la consola: *$ npm install mysql*

AdESMuS controla la autenticación de los usuarios a través de Tokens. Estos Tokens, son generados por una librería, *jsonwebtoken*. Para realizar la instalación de la librería se ejecuta desde la consola el siguiente comando: *$ npm install jsonwebtoken*

Para realizar la documentación de la API, AdESMuS utiliza la herramienta *apidoc* la cual se instala desde la consola con el siguiente comando: *$ npm install apidoc -g*

Una vez que las librerías necesarias se encuentran instaladas, se configuran las rutas o recursos de la API. En la siguiente ilustración se muestra la jerarquía de las rutas asignadas en el fichero de configuración *app.js*.

```
// Módulos de comunicación, definición
var router_principal = require('./adesmus_modules/mod_comunication/router_principal');
var router_auth = require('./adesmus_modules/mod_comunication/router_auth');
var router_api = require('./adesmus_modules/mod_comunication/router_api');
var router_administradores = require('./adesmus_modules/mod_comunication/router_administradores');
var router_estudiantes = require('./adesmus_modules/mod_comunication/router_estudiantes');
var router_docentes = require('./adesmus_modules/mod_comunication/router_docentes');
var router_centros = require('./adesmus_modules/mod_comunication/router_centros');
```

```
var router_departamentos = require('./adesmus_modules/mod_comunication/router_departamentos');
var router_titulaciones = require('./adesmus_modules/mod_comunication/router_titulaciones');
var router_grupos = require('./adesmus_modules/mod_comunication/router_grupos');
var router_asignaturas = require('./adesmus_modules/mod_comunication/router_asignaturas');
var router_matriculas = require('./adesmus_modules/mod_comunication/router_matriculas');

//Rutas
app.use('/', router_principal);
app.use('/auth', router_auth);
app.use('/api', router_api);
app.use('/api/users/administrator', router_administradores);
app.use('/api/users/student', router_estudiantes);
app.use('/api/users/teaching', router_docentes);
app.use('/api/center', router_centros);
app.use('/api/department', router_departamentos);
app.use('/api/degree', router_titulaciones);
app.use('/api/group', router_grupos);
app.use('/api/subject', router_asignaturas);
app.use('/api/enrollment', router_matriculas);
```

Ilustración 25 – Definición de rutas o recursos

En la parte inferior de la ilustración anterior, se encuentran definidas las rutas, las cuales, son encapsuladas por su funcionalidad, por ejemplo, todo lo referente a los departamentos tendrá como inicio de ruta */api/department*. Otro ejemplo, todas las operaciones sobre cualquier tipo de usuario tendrá como inicio de ruta */api/users*.

Cada definición de la ruta, se acompaña de un segundo parámetro, el cual, apunta al controlador que contiene la funcionalidad de los recursos. Por ejemplo, *app.use('/', router_principal)* tiene como segundo parámetro la variable *router_principal*, esto quiere decir, si se observa en la parte superior de la ilustración anterior, el contenido de dicha variable no es más que un Path que apunta a una zona del sistema, concretamente a un fichero. Todas las definiciones de rutas apuntan al *Módulo de comunicación* puesto que es el encargado de gestionar las peticiones entrantes.

Es importante fijarse en la definición de *app.use('/api', router_api)*. Si se observan el resto de las definiciones de rutas, ésta puede resultar repetitiva, puesto que casi todas parten de */api*. En este momento entra en juego el *Middleware*.

Un *Middleware*, no es más que un punto de ejecución intermedio, un filtro, que permite controlar todas las peticiones que se realicen por debajo de una ruta estipulada. Es decir, cada vez que alguien solicite un recurso que contenga como raíz */api*, por ejemplo */api/department*, antes de que la petición pase al recurso solicitado, se filtra por el *Middleware*. En el caso de AdESMuS, en cada una de las peticiones que se hagan desde */api*, se comprueba la existencia del Token del usuario y su validez, si éste es correcto, el *Middleware* deja pasar la petición para acceder al recurso solicitado.

Una vez que las rutas ya están definidas, a continuación, se explicará de forma detallada la comunicación entre los diferentes módulos, y los problemas encontrados durante la implementación de los mismos.

6.2 Acceso a la Landing Page

El acceso a éste recurso es público, es decir, no es necesario encontrarse identificado en el sistema para poder acceder al contenido. Es un recurso que únicamente devuelve datos, por lo tanto el único verbo aceptado es el de GET. Recordar que todos los controladores se encuentran en la carpeta *mod_comunication*. En la siguiente ilustración se muestra el contenido del controlador *router_principal.js*.

```
router_principal.get(
	'/',
	function(req,res,next)
	{
		res.sendFile(path.join(__dirname, '../../views', 'index.html'));
	}
);
```

Ilustración 26 - Módulo de comunicación: Recurso / GET

Como se observa en la ilustración anterior, cuando la petición llega al controlador, el servidor prepara la respuesta, y para ello, accede a las vistas disponibles en la carpeta *views*, obtiene el contenido del fichero, y por el objeto res (response) le envía los datos al usuario. En la siguiente ilustración se muestra la plantilla definida que se envía al usuario cuando éste solicite el recurso.

```
<!DOCTYPE html>
<html>
<head>
	<title>AdESMuS</title>
</head>
<body>
	<h1>AdESMuS</h1>
</body>
</html>
```

Ilustración 27 - Módulo de comunicación: Plantilla HTML

6.3 Acceso a los recursos y seguridad

AdESMuS dispone de varias capas de seguridad. La **primera**, el *Token*, el cual identifica al usuario que realiza las peticiones al sistema. Es la llave con la que el usuario accede a los recursos de la API. La **segunda** capa de seguridad son los *Roles*. En el momento del registro al usuario se le asigna un rol (Administrador, Docente o Estudiante), el cual, le indica a qué recursos tiene acceso. Por ejemplo. Un usuario de tipo Estudiante, puede obtener los Docentes que se encuentran registrados en el sistema, pero nunca podrá registrar un Docente. Con la utilización de los roles, se permite encapsular mejor la

funcionalidad de cada uno de los actores. La **tercera** capa de seguridad se enfoca directamente en la encriptación del canal de comunicación mediante certificados SSL. La **cuarta** capa de seguridad, hace referencia a la forma en la que se guardan los datos sensibles del usuario. Por motivos de seguridad nunca se registrarán contraseñas en la base de datos. La **quinta** y última capa de seguridad hace referencia a la encriptación de los identificadores creados en los registros. El usuario, en la parte cliente, nunca trabaja con identificadores reales de las tablas, siempre con hashes reversibles, los cuales únicamente pueden ser encriptados y desencriptados por el sistema.

A continuación se explica cada una de las capas de seguridad en detalle, indicando en qué momento se aplica la capa de seguridad y de qué manera afecta al sistema su utilización.

Primera capa, el Token

Un Token, no es más que un objeto cifrado que contiene información, generalmente sensible, y el cual, es utilizado como identificador para poder acceder a los recursos que ofrece una API. La funcionalidad para la creación y verificación de los Tokens se encuentra en el fichero *token.js* dentro de la carpeta *helpers*. AdESMuS es la única entidad encargada de crear, cifrar y descifrar los Tokens, y por lo tanto, la única conocedora del contenido de los mismos. En el caso de AdESMuS el objeto Token generado dispone de los siguientes parámetros, los cuales, serán comprobados en el momento de la validación del Token.

- *Exp*: Fecha de expiración del Token. A partir de la fecha marcada, el Token dejará de estar vigente, y será un Token inválido.
- *Huella*: La huella única del navegador desde el cual se realiza la identificación.
- *Id*: Id del usuario que solicita la identificación.
- *Usuario*: Nombre del usuario que solicita la identificación.
- *Rol*: Rol del usuario que solicita la identificación. Supremo, Administrador, Docente o Estudiante.

Segunda capa, los Roles

Una vez que el Token es validado en el Middleware, la siguiente capa de seguridad hace referencia a los *Roles*, a través de los cuales, se establece que tipo de usuarios pueden hacer uso de un recurso y cuáles no. Por ejemplo, un usuario con *Rol* Estudiante, nunca podrá crear un Centro. Para crear un Centro el sistema determina que se necesita un usuario con *Rol* de Administrador.

En las siguientes tablas se muestra de forma detallada cuales son las operaciones que se pueden realizar sobre los recursos, en función de los roles asignados a cada funcionalidad.

Departamento	Sup.	Admin.	Doc.	Est.	Recurso
Obtener un Dpto	X	X	X	X	**GET** /api/department/id
Obtener todos los Dptos	X	X	X	X	**GET** /api/department
Actualizar un Dpto	X	X			**PUT** /api/department/id
Añadir un Dpto	X	X			**POST** /api/department
Eliminar un Dpto	X	X			**DELETE** /api/department/id
Eliminar todos los Dptos	X				**DELETE** /api/department

Tabla 3 - Operaciones sobre un Departamento

Centro	Sup.	Admin.	Doc.	Est.	Recurso
Obtener un Centro	X	X	X	X	**GET** /api/center/id
Obtener todos los Centros	X	X	X	X	**GET** /api/center
Actualizar un Centro	X	X			**PUT** /api/center/id
Añadir un Centro	X	X			**POST** /api/center
Eliminar un Centro	X	X			**DELETE** /api/center/id
Eliminar todos los Centros	X				**DELETE** /api/center

Tabla 4 - Operaciones sobre un Centro

CentroDepartamento	Sup.	Admin.	Doc.	Est.	Recurso
Asociar un Dpto a Centro	X	X			**GET** /api/center/idCentro/ department/idDpto
Obtener todos los Dptos asociados a un Centro	X	X	X	X	**GET** /api/center/idCentro/ department
Obtener Dpto asociado a un Centro	X	X	X	X	**GET** /api/center/idCentro/ department/idDpto
Eliminar asociación de un Dpto a un Centro	X	X			**DELETE** /api/center/idCentro/ department/idDpto
Eliminar todos los Dptos asociados a un Centro	X	X			**DELETE** /api/center/idCentro/ department/

Tabla 5 - Operaciones sobre un CentroDepartamento

El control de acceso por *Roles* se realiza en el *Módulo de comunicación*, el cual se lanza antes de la ejecución de cualquier operación. Por ejemplo, se quiere realizar la petición de añadir un Centro en el sistema. Para ello se llama al *Módulo de comunicación*, realizando una petición *POST /api/center*. En la siguiente ilustración se muestra la cabecera del método que recibe la petición del registro de un Centro.

```
router_api_centros.get(
        '/',
        GeneralFunctions.RequireRole([0,1,2,3]),
        function (req,res,next)
        {
                -- CODIGO--
        }
```

Ilustración 28 - Control de acceso por Roles

Se observa que en la definición de la cabecera hay tres parámetros. El primero indica el Path del recurso al que se accede. El segundo parámetro, hace referencia a los *Roles* requeridos para poder ejecutar la operación. Y en el último lugar, se encuentra la ejecución de la operación. La operación solo se ejecutará si pasa el filtro del método *RequireRole*, el cual, se encarga de revisar el *Rol* contenido dentro del *Token* proporcionado por el usuario. Si su *Rol* se encuentra dentro de la lista de los roles permitidos para ejecutar la operación (en este caso 0-Supremo, 1-Administrador, 2-Docente, 3-Estudiante), el método *RequireRole* deja paso para que la operación se ejecute. Si el *Rol* que tiene el usuario en el *Token* no se encuentra en la lista de los *Roles* admitidos para la ejecución de la operación, el sistema devuelve error, indicándole al usuario que no tiene los permisos adecuados para ejecutar la operación.

Tercera capa, encriptación del canal, certificado SSL

AdESMuS hace uso de un *certificado SSL* (https) que encripta el tráfico entre el cliente y el servidor. Esta capa de seguridad, es una medida extra, puesto que con la utilización de Tokens, los datos del usuario ya viajan cifrados por la red, salvo en el momento del registro del usuario, por ello, se implanta el certificado SSL.

Cuarta capa, registro de datos sensibles en la base de datos

Por motivos de seguridad obvios, no es viable registrar contraseñas en la base de datos, y por ende, tampoco es viable el registro de contraseñas encriptadas. ¿Por qué no? Fácil, si el sistema es vulnerado es posible que exista una pérdida de información, y además, si el sistema es crackeado, el atacante seguramente podrá obtener la contraseña usada para la encriptación y desencriptación de las contraseñas que se encuentran

almacenadas en la base de datos, porque claro, el sistema usa la misma contraseña para encriptar todas las contraseñas de los usuarios. Esto no es correcto.

Para afrontar esta situación, en AdESMuS se hace uso de Hashes y semillas. Los Hashes son secuencias alfanuméricas irreversibles. Y una semilla o Salt puede ser una contraseña alfanumérica de una longitud determinada, generada por un Random. En el momento del registro de un usuario el sistema genera una semilla única para dicho usuario, la cual junto a la contraseña del usuario, y una tercera contraseña definida en el sistema, da como resultado un Hash irreversible, el cual, se guardará en la base de datos junto a la semilla creada para el usuario. Cada usuario registrado en la base de datos tendrá asignado un Hash generado con una Salt diferente. Cuando el usuario se identifique, la contraseña llegará (encriptada siempre, https) al sistema, que obtendrá el usuario de la base de datos que quiere realizar la identificación, obtendrá el Salt que tiene asignado y junto a la contraseña definida en la configuración del sistema se genera un Hash. Bien, ahora existen dos Hashes, el que tiene el usuario en la base de datos asignado y el que se acaba de generar en la identificación. Se comparan los dos Hashes, si coinciden la identificación del usuario es correcta, si no coinciden, el usuario ha introducido una contraseña errónea lo que provoca que el Hash generado sea diferente al que tiene registrado.

¿Qué pasaría ahora con las contraseñas de la base de datos si el sistema se ve comprometido? Básicamente nada, aunque exista una pérdida de información (que también es preocupante), siempre será necesaria la contraseña del usuario para generar el Hash, y ésta contraseña solo se aporta en el momento de la identificación por el usuario.

Quinta capa, encriptación de identificadores

Para que la comunicación entre el cliente y el servidor sea fluida, es necesario el uso de identificadores que permitan identificar sobre que objetos de la API se desean realizan las operaciones. No es seguro el uso directo de éstos identificadores, nadie tiene porque saber que el identificador, por ejemplo, del Centro X es el 1050 porque de la misma manera que sabe de la existencia del id 1050, puede deducir que al menos existen otros 1049 Centros, que además, seguramente respeten el orden de identificación. En AdESMuS, el tratamiento de los identificadores se realiza en el *Módulo de comunicación*. Internamente el sistema trabaja con los identificadores reales, pero en el momento en que la información sale fuera del sistema, se filtran los identificadores y se encriptan. De la misma manera cuando llegue una petición, por ejemplo, para obtener la información del Centro con identificador *xkjdlks8983lksdf*, el *Módulo de comunicación* se encarga de desencriptar el identificador, comprobar que es correcto y posteriormente operar con él internamente sin encriptar.

Una vez explicadas las diferentes capas de seguridad de las que dispone el sistema AdESMuS, se detallarán las operaciones de registro e identificación de un usuario y el consumo de un recurso, explicando la problemática encontrada en el desarrollo y las soluciones aportadas.

Registro de un usuario

El acceso a este recurso es público, pero con matices. El sistema permite registrar diferentes tipos de usuarios (roles de usuarios): Supremo, Administradores, Docentes y Estudiantes. Pero no todos los usuarios se registran de la misma manera.

El sistema diferencia dos tipos de usuarios principalmente, por un lado está el usuario Supremo, y por otro lado, se encuentran los Administradores, Docentes y Estudiantes.

Inicialmente AdESMuS no dispone de usuarios registrados en la base de datos, y esto implica, que no se puedan realizar acciones en la API. Para que el sistema se active, es necesario registrar un usuario que tenga el rol de usuario Supremo. Este usuario será único en el sistema, y será el que a su vez, se encargue de registrar nuevos usuarios: Administradores, Docentes y Estudiantes.

Una vez que el usuario con rol de Supremo es registrado en el sistema, el recurso para realizar registros de usuarios con rol Supremo (/auth/register) se comenta en el código y queda deshabilitado. En el sistema no existe ningún recurso que permita eliminar el usuario con rol de Supremo. Este usuario será un usuario permanente en el sistema.

Para poder registrar otros tipos de usuarios (Administradores, Docentes o Estudiantes) se utilizará la funcionalidad proporcionada por la API en cada uno de los recursos. Por ejemplo, para registrar un Administrador, el recurso */api/users/administrator* dispone de un método que permite realizar el registro. Otras operaciones que se pueden realizar sobre un Administrador, serán las de: obtener un Administrador, actualizar un Administrador, o eliminar un Administrador.

El proceso para el registro de un usuario (independientemente de su rol) siempre será el mismo. A continuación, se explicará el proceso completo desde que se realiza la solicitud de registro, hasta que el sistema devuelve la respuesta al usuario indicándole que el registro ha sido satisfactorio.

Todo comienza con una solicitud al recurso. Para realizar un registro de un usuario con rol de Supremo, no es necesario indicarle al servidor quien realiza la petición, pero para realizar un registro de un usuario con rol de Administrador, Docente o Estudiante, es necesario identificarse en el sistema para obtener un Token válido.

En el proceso de identificación, un usuario con rol Administrador obtiene un Token válido, único y de duración determinada. Una vez que el Administrador dispone de un Token, puede proceder a realizar la solicitud de registro de un Docente. Para ello, llama al recurso */api/users/teaching* utilizando el verbo POST, y coloca en la cabecera de la petición el Token que le ha sido facilitado en la identificación. En el cuerpo de la petición, el Administrador incorpora en formato JSON los parámetros necesarios para el registro de un Docente y envía la solicitud.

El *Módulo de comunicación*, a través del *Middleware*, recibe la solicitud, y antes de procesarla y enviarla al recurso que registra el Docente, comprueba si la petición

contiene en su cabecera un Token. En caso de que el Token no exista, el Middleware cancelará la acción, y le enviará al usuario una respuesta.

El primero de los problemas encontrados fue determinar la forma en la que los datos eran devueltos al usuario. Toda petición tiene una respuesta, la cual, debe ser lo suficientemente clara, para determinar que ha pasado en el sistema. Puesto que todos los recursos del sistema devuelven algo siempre, había que definir una trama de datos común para todos los recursos.

Para definir la trama de datos, dentro de la carpeta */helpers* se crea un fichero llamado *response.js*, el cual, será el encargado de definir todas las tramas solicitadas. Dentro del fichero, se crea una plantilla de tipo JSON, donde se definen las cadenas que posteriormente serán sustituidas por el contenido que se desea enviar. En la siguiente ilustración se muestra la template en formato JSON. Cuando se solicite la trama de datos, se sustituirán las variables *_VAR0status_*, *_VAR1code_*, *_VAR2msg_* y *_VAR3data_* por el contenido que se desea enviar al usuario.

```
var templateGenerica =
        '{ "status" : _VAR0status_, "code" : _VAR1code_ , "msg" : _VAR2msg_ , "body" : _VAR3data_ }';
```

Ilustración 29 - Response.js: plantilla genérica

Con la definición de esta trama, se consigue que todas las respuestas sean uniformes, y que lo único que varíe en ellas, sea la información que se desea enviar. Si en un futuro hay que realizar modificaciones en la trama de datos, solamente habrá que cambiar la estructura de la plantilla, y todos los recursos devolverán la trama actualizada.

La trama de datos cuenta con cuatro parámetros. El parámetro *status* hace referencia a un código de estado (oficial HTTP). El segundo parámetro *code*, hace referencia a un código interno de la aplicación. El tercer parámetro *msg* indica a modo informativo el significado del parámetro *code.* Y finalmente *body* contiene la información solicitada por el usuario.

Dentro del fichero *response.js* también se han definido los tipos de códigos de error que pueden ocurrir. Cada uno de los elementos del JSON, define un tipo de error y contiene tres parámetros. Un código *status* genérico, el código interno asociado denominado *code* y un mensaje informativo *msg* del código interno definido. En la siguiente ilustración de pueden observar los códigos de errores que pueden darse en el sistema.

```
var codes =
        [
        //200
        { status : 200, code : 800, msg : "'Ok"' },
        { status : 200, code : 801, msg : "'Nok"' },
        { status : 200, code : 802, msg : "'Error interno"'},
```

```
        //400
        { status : 400, code : 900, msg : '"Error, json o parámetros recibidos incorrectos"'},
        //401
        { status : 401, code : 910, msg : '"Token Error, no existe token"'},
        { status : 401, code : 911, msg : '"Token Error, el token enviado no tiene usuario en BD"'},
        { status : 401, code : 912, msg : '"Token Error, el token no es válido"'},
        { status : 401, code : 913, msg : '"Login Error, los datos no son correctos"'},
        { status : 401, code : 914, msg : '"Token Error, los token no coinciden"'},
        //403
        { status : 403, code : 920, msg : '"Error, no tienes los permisos/rol necesarios para realizar la acción"'},
        //404
        { status : 404, code : 940, msg : '"Error, el verbo solicitado no existe para el recurso"'},
        //500
        { status : 500, code : 950, msg : '"Error, se ha producido un error interno en el servidor"'}
]
```

Ilustración 30 - Códigos de error internos

En la siguiente ilustración se muestra la lista de los códigos de estado HTTP oficiales utilizados en el desarrollo de AdESMuS y su significado. Estos códigos hacen referencia a la variable *status* de la trama de datos.

```
Códigos genéricos – Respuestas del servidor

200     : OK
        Respuesta ok del servidor

400 : Bad Request
        Los datos enviados por el cliente no son correctos. Faltan
        datos o el json no es correcto.

401 : Unauthorized
        Fallos en cuanto al login, registro, y petición de datos.
        Token incorrecto.

403 : Forbidden
        Permisos de rol para ejecutar la acción. El servidor reconoce
        el método pero el usuario no tiene acceso para ejecutarlo.

404 : Not Found
        No se encuentra el método llamado. La Url no es correcta.

500 : Internal Server Error
        Error interno del servidor.
```

Ilustración 31 - Códigos de error oficiales para HTTP

Dentro del fichero *response.js* también encontramos el método que gestiona las peticiones de las tramas solicitadas, como se muestra en la siguiente ilustración. Dicho método recibe por parámetro el código de error que se definirá en la trama y los datos que se asociarán al *body*. La función realiza una serie de comprobaciones y genera un nuevo objeto que contiene la trama completada con la información solicitada.

```
Response._r = function(
	pCodigo,
	pData)
{
	var _data = null;
	var _result = null;

	//Se comprueba si el usuario nos envía el código
	if (pCodigo === undefined ||
		pCodigo === null ||
		!isInt(pCodigo)) return _result;

	//Se obtiene el código de la lista de códigos
	var _c = buscarCodigo(pCodigo);
	if (_c === undefined ||
		_c === null) return _result;

	//Se convierten los datos de entrada para añadirlos en la plantilla
	if (pData !== undefined &&
		pData !== null) _data = JSON.stringify(pData);

	//Se genera el objeto con las variables a modificar
	var _mapaObjeto = {
		"_VAR0status_" : _c.status,
		"_VAR1code_" : _c.code,
		"_VAR2msg_" : _c.msg,
		"_VAR3data_" : _data
	};

	//Se construye el json con las variables colocadas
	_result = templateGenerica.replace(
		/_VAR1code_|_VAR2msg_|_VAR3data_|_VAR0status_/gi,
		function(pEncontradas) { return _mapaObjeto[pEncontradas]; }
	);
	return JSON.parse(_result);
}
```

Ilustración 32 - Response.js: Método que genera la trama de datos de vuelta

En la siguiente ilustración se muestra un ejemplo de la llamada al método que genera la trama con los datos.

```
var _res = response._r(900);
```

Ilustración 33 - Response.js: Llamada al método que genera la trama de datos

Una vez definida la generación de las tramas para la devolución de los datos, se continúa con el proceso del registro de un Docente. La petición ha pasado las pruebas del *Middleware*, el cual, ha validado el Token incorporado en la cabecera de la solicitud, y la petición llega al recurso */api/users/teaching*.

El Middleware */api* anteriormente nombrado, al igual que el recurso */api/users/teaching* se encuentran en el Módulo de comunicación.

Cuando la petición llega al recurso /api/users/teaching, el sistema comprueba el verbo de la petición para determinar cuál es la acción que se quiere realizar. GET, POST, PUT o DELETE.

Generalmente la diferencia entre POST y GET, es la forma en la que se envía la información al servidor (entre otras). El verbo GET se utiliza para solicitar información al servidor (si se envían parámetros, será a través de la propia Url, poco recomendado). POST es utilizado cuando se quiere enviar información al servidor para realizar una acción (los datos no se envían en la Url, sino en el cuerpo de la petición, puede parecer seguro, pero tampoco lo es). El verbo PUT le indica al sistema que la operación que se desea realizar es de actualización. De la misma manera, el verbo DELETE le indica al sistema que se desea eliminar un elemento del sistema.

La petición de registro de un Docente, va acompañada de un verbo POST. Por lo tanto, en el cuerpo de la petición deben encontrarse los datos del Docente que se va a registrar. El sistema comprueba si existen en el cuerpo de la petición todos los datos necesarios para realizar el registro. En caso de falte algún parámetro, se formará una trama de vuelta, indicando al usuario el resultado de la solicitud (errónea).

Los datos enviados son validados y el *Módulo de comunicación* solicita al *Módulo de coordinación* que le gestione la solicitud del registro. Para ello, llama a una función *AddDocente* que se encuentra en el fichero *functions_docente.js* dentro de la carpeta de *mod_coordination.*

Una vez la petición se encuentra en el *Módulo de coordinación*, se genera el objeto con la información del usuario que ha recibido del *Módulo de comunicación* y delega la tarea de registro el *Módulo de gestión*, el cual, su única misión es realizar todos los accesos a la base de datos, ya bien sea para obtener información, actualizar, registrar o eliminar elementos. El *Módulo de gestión* registra el objeto recibido por el *Módulo de coordinación*, y éste devuelve una respuesta, obteniendo diferente información de ella: el identificador del registro del Docente o el número de filas afectadas (muy útil sobre todo cuando se actualizan o eliminan objetos de la base de datos).

El *Módulo de coordinación* recibe la respuesta del *Módulo de gestión*, y éste a su vez, se la envía al *Módulo de comunicación*, que comprueba si ha existido algún error durante el proceso de registro, y genera la trama de datos para responder al usuario. En la trama de datos devuelta al Administrador se incluirá: el estado de la respuesta, y el identificador encriptado del registro que se acaba de realizar.

Identificación de un usuario

Una vez que se ha registrado un Docente en el sistema. Es el momento de que el propio Docente se identifique en el sistema para obtener un Token que le permita consumir los recursos que tenga asignados.

El recurso que permite realizar la identificación en el sistema es */auth/login*, que se encuentra en el *Módulo de comunicación* en el fichero *router_auth*, y atiende las solicitudes con el verbo POST. Además, la petición obtenida en el servidor debe contener en el cuerpo los datos de identificación del usuario, en este caso, del Docente: el Nick del usuario, su contraseña y la huella del navegador. Este es el único momento, como ya se comentó en puntos anteriores, que los datos de identificación del usuario viajan a través de la red. El certificado SSL (https) aplicado al canal de comunicación, permite que los datos del usuario que se transmiten siempre se encuentren encriptados.

Los datos recibidos en la solicitud son correctos, y el *Módulo de comunicación* solicita al *Módulo de coordinación* la tarea de la identificación del usuario. Para ello, le envía los datos que ha recibido en la solicitud y se queda a la espera de que éste le comunique una respuesta.

Otro de los problemas encontrados durante la implementación, fue el tratamiento de las esperas entre ejecuciones de tareas. NodeJS ejecuta las tareas de forma asíncrona, es decir, si ejecutamos una tarea, el sistema no espera a su finalización y continúa ejecutando el resto de las tareas que pueda tener pendientes. En la siguiente ilustración se muestra gráficamente el comportamiento de NodeJS frente a la ejecución de varias tareas asíncronas.

El servidor recibe las tareas a través un único hilo principal, y éste las envía a un pool de tareas. Estas tareas se ejecutan de forma asíncrona, mientras que el hilo principal puede seguir atendiendo otras peticiones entrantes. Cuando una tarea finalice, se lo hará saber al hilo principal a través de un CALLBACK, el cual puede ser personalizado para determinar la información que se quiere devolver. Generalmente la función CALLBACK debe devolver un parámetro que indique si ha existido algún error en la ejecución de la tarea, y otro parámetro que contenta los datos solicitados.

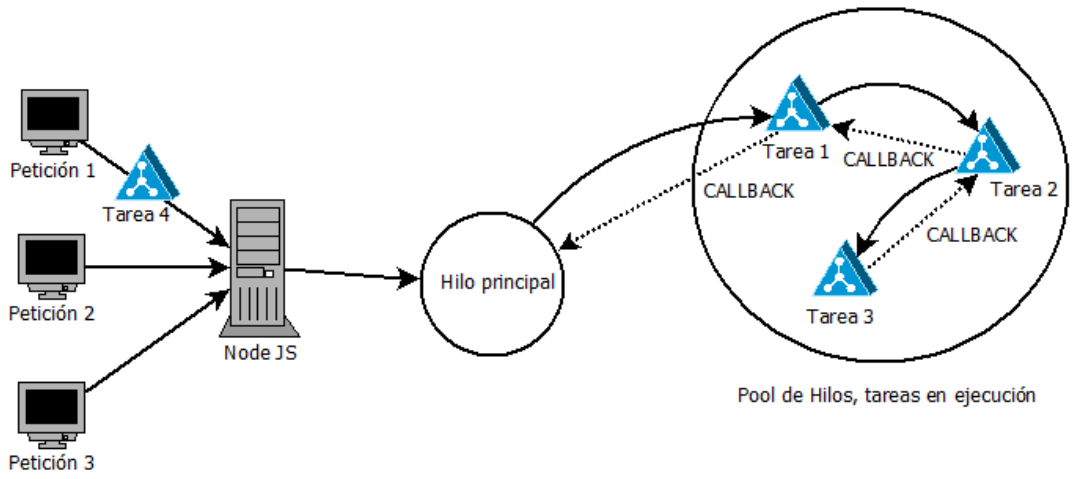

Ilustración 34- NodeJS: Tratamiento de las tareas asíncronas

AdESMuS necesita que la ejecución interna de las tareas sea síncrona, es decir, las tareas se deben esperar unas a otras. Por ello, la solución tomada fue utilizar

CALLBACKs y la anidación de funciones. En la siguiente ilustración se muestra un ejemplo de uso en forma de código. Se crea la primera tarea llamando a la función *FunciónX* pasando por parámetro tres elementos: pParémtro1, pParámetro2 y el CALLBACK. Este último será el encargado de gestionar la respuesta principal. Internamente dentro de la función se crea una nueva tarea *ObtenerInformación* que tiene como último parámetro una nueva función. Esta función contendrá la respuesta dada por el método *ObtenerInformación*. Una vez que se obtiene la información, se finaliza la ejecución de la tarea *FunciónX*, completando la respuesta del CALLBACK con la información obtenida de la respuesta del método *ObtenerInformación.*

```
exports.FunciónX = function(
        pParámetro1,
        pParámetro2,
        CALLBACK){
                Módulo.ObtenerInformación (
                        pParámetro1,
                        pParámetro2,
                        function(error, info)
                        {
                                //Respuesta de FunciónX
                                CALLBACK(null, info);
                        }
        }
```

Ilustración 35 - Ejemplo CALLBACK y funciones anidadas

Volviendo a la identificación del usuario en el *Módulo de coordinación*, y habiendo recibido los datos, se comprueba que el usuario que solicita el Token se encuentra registrado en la base de datos. Si el usuario no está registrado se genera una trama de vuelta con el error ocurrido para avisar al usuario del estado de su solicitud.

Una vez que se valida la existencia del usuario en la base de datos, el sistema comprueba si la contraseña introducida por el usuario es correcta. Como ya se comentó en el apartado del registro de un usuario, nunca se guardará una contraseña del usuario en la base de datos, únicamente se guarda un Hash y un Salt en el momento del registro.

Después de validar la contraseña del usuario, el sistema comprueba si el usuario tiene un Token asignado en la base de datos. Si el usuario no tiene asignado un Token se genera uno válido para un tiempo limitado y se le asigna solicitando el *Módulo de gestión* la actualización del usuario. Si el usuario ya dispone de un Token, el sistema comprueba si se trata de un Token válido, y si éste es correcto, se le envía al usuario. En caso de que el Token que tiene asignado el usuario sea correcto pero haya expirado, el sistema le asignará uno nuevo.

El sistema solamente genera Tokens cuando es necesario. Este era otro de los problemas a los que hacer frente. No pueden existir Tokens válidos en la red, que no se encuentren asociados a los usuarios. Todos los Tokens válidos en el sistema AdESMuS tendrán dueño, un usuario del sistema.

6.4 Acceso a la documentación Online

Para realizar la documentación de la API, se ha utilizado la herramienta *apidoc*, que permite crear una interfaz de usuario a partir de la generación de comentarios en el código.

La interfaz está totalmente ligada a las cabeceras de las funciones definidas en el código. De esta manera, facilita enormemente la tarea de mantener actualizada la documentación cuando se realicen cambios. Modificando únicamente el comentario añadido en la cabecera del método y ejecutando la instrucción *apidoc* en la consola, la documentación se actualizará automáticamente.

En el momento de la generación de la documentación se genera una carpeta llamada *doc* en la raíz del proyecto.

7 VERIFICACIÓN Y EVALUACIÓN

En el siguiente apartado se detallará la forma en la que ha sido probada la API y cuáles han sido los instrumentos utilizados para comprobar el correcto funcionamiento del sistema.

7.1 Definición de baterías de pruebas unitarias

Debido al gran volumen de recursos y funcionalidades definidas, se definen baterías de pruebas unitarias que permiten comprobar el estado de todas las funcionalidades definidas. Estas baterías se encuentran separadas en dos grandes grupos, el primer grupo focalizado a la parte de los usuarios, donde se probarán todas las operaciones que se puedan realizar entre usuarios, y el segundo, a la parte de gestión donde se comprobarán todas las operaciones que se puedan realizar en los Centros, Departamentos, Matrículas, etc.

En total han sido definidas en torno a 950 pruebas unitarias, cuyo tiempo aproximado de ejecución es de 50 segundos, obteniendo como resultado un resumen de cada una de las pruebas ejecutadas y el estado de las mismas. El tiempo invertido en el diseño de las pruebas unitarias ha sido elevado, pero se ha ganado en velocidad de desarrollo. Si se realiza una modificación en el sistema, en 50 segundos se puede saber si el sistema sigue funcionando correctamente.

Para definir las pruebas se ha utilizado una extensión de Google Chrome llamada *Postman*, la cual, permite realizar solicitudes HTTP. En la siguiente ilustración se muestra a modo ilustrativo cuál sería su estructura inicial con las baterías de pruebas definidas en la parte izquierda.

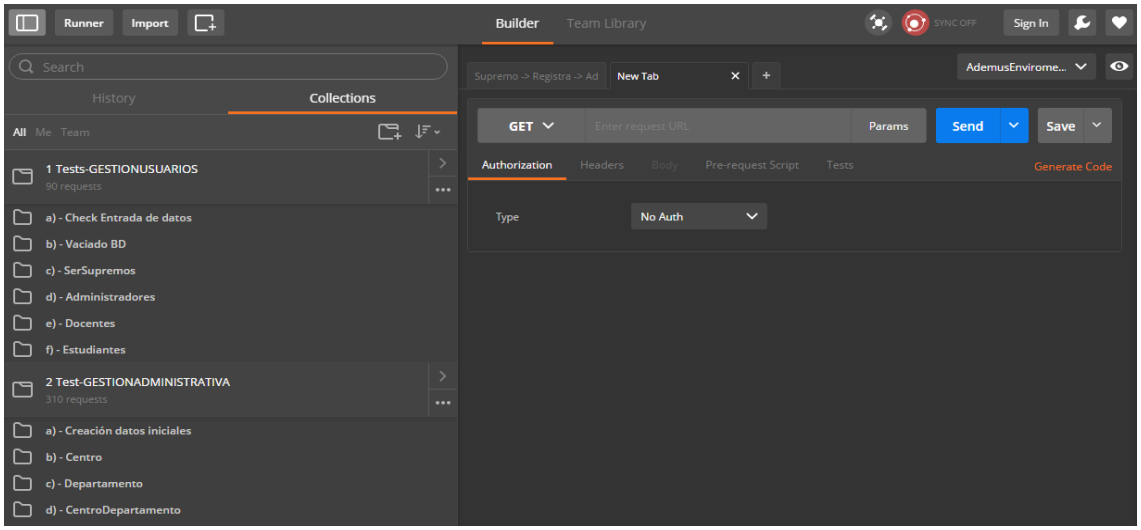

Ilustración 36 - Postman, primer vistazo

Para mejorar la gestión de la implementación de las pruebas, se crea un entorno de variables globales que permiten realizar modificaciones sobre el sistema de una forma rápida. Además de las variables globales, también se hará uso de variables locales. En la siguiente ilustración se muestran las variables globales definidas para el proyecto AdESMuS.

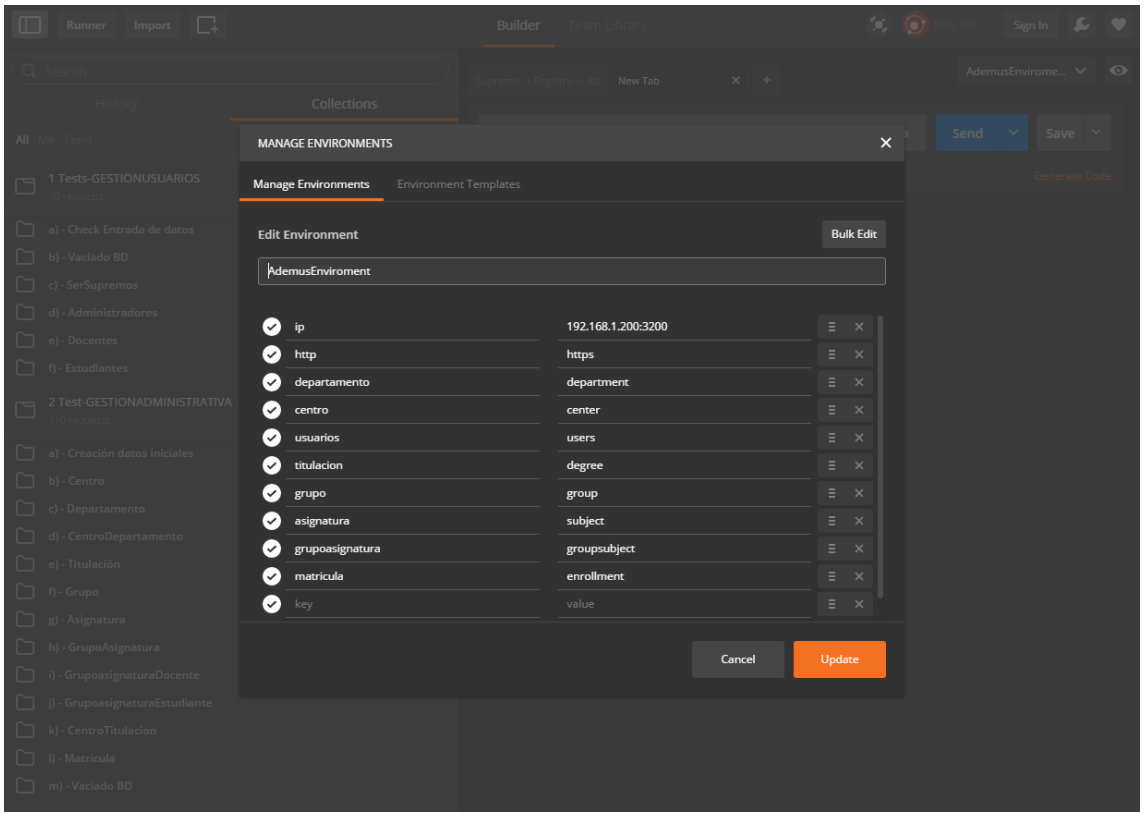

Ilustración 37 - Postman, definición de variables globales

Una vez definidas las variables globales, y antes de empezar a explicar las pruebas, se explicará el funcionamiento básico del proceso de ejecución de una prueba, comprobaciones realizadas y el resultado obtenido.

Para realizar un pequeño ejemplo se tomará como referencia el registro de un Centro. Para realizar el registro de un Centro es necesario tener un Token válido de un Administrador. Se da por hecho que previamente se ha realizado una prueba unitaria de identificación de un Administrador y el Token obtenido se ha guardado en una variable local *admin-token*.

En la siguiente ilustración se muestra la solicitud del consumo del recurso por medio de la Url (*{{http}}://{{ip}}/api/{{centro}}*), donde se observa que los parámetros que se encuentran entre llaves hacen referencia a las variables globales definidas. También se observa en la parte de la cabecera que se pasa por parámetro el *x-acess-token* que hace referencia al Token del Administrador guardado en la variable local *admin-token*.

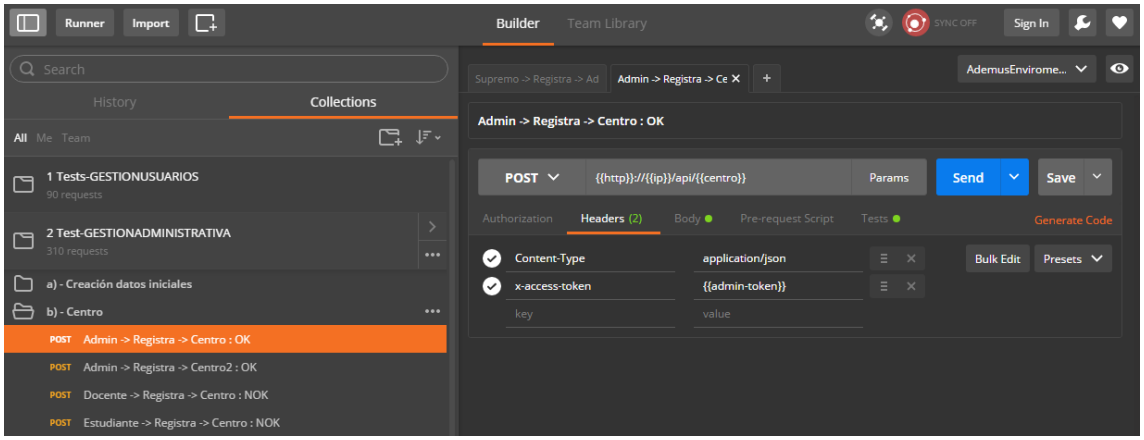

Ilustración 38 - Postman, definición de la petición (I)

En la parte del cuerpo de la petición, como se observa en la siguiente ilustración se definirá el objeto con los datos en formato JSON, que contendrá los parámetros necesarios para realizar el registro de un Centro, en este caso, solamente es necesario introducir el nombre del Centro.

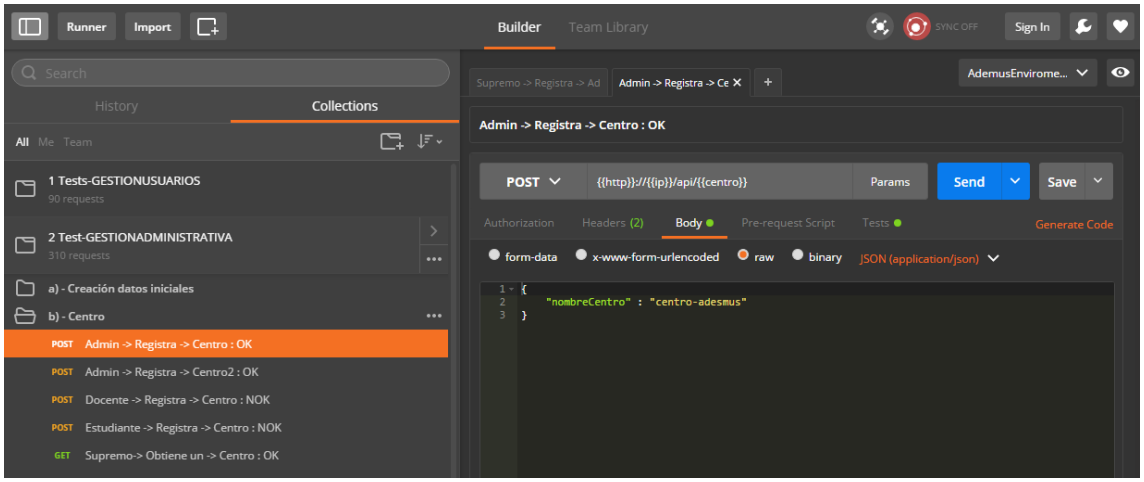

Ilustración 39 - Postman, definición de la petición (II)

Por último, en el apartado de las pruebas, como se observa en la siguiente ilustración, se encuentran definidas las comprobaciones que se desean aplicar al resultado de la petición. En el caso del Centro, para validar la prueba del registro es necesario que el sistema devuelva un *response* con código 200, y que también, internamente el objeto devuelvo (la trama de los datos) tenga un parámetro code con el valor de 800. Además el sistema guardará en la variable local *centro-id* el identificador obtenido del registro para utilizarlo en futuras pruebas.

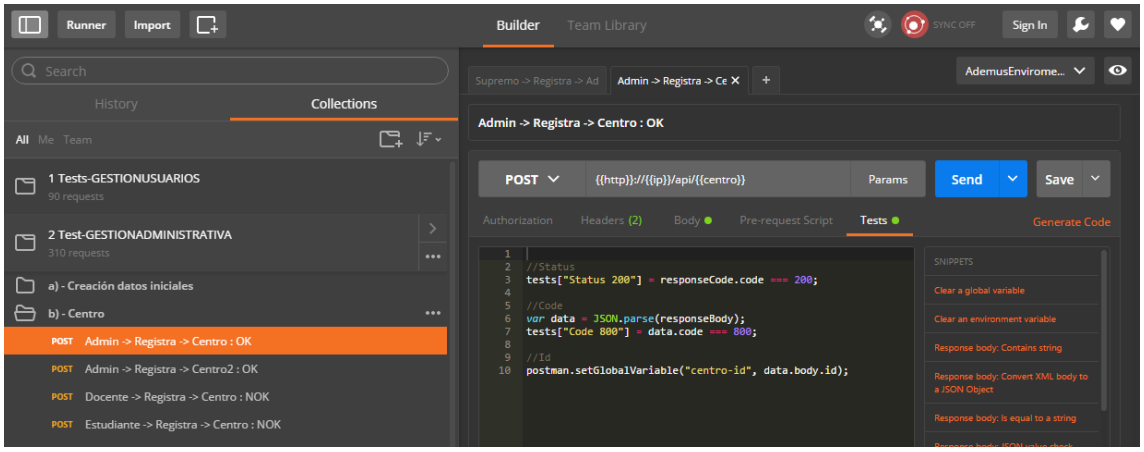

Ilustración 40 - Postman, definición de la petición (III)

El estado 200 de la respuesta viene explícito en la propia respuesta y además también se encuentra dentro de la trama de vuelta de datos junto al resto de información. En la siguiente ilustración se muestra el resultado obtenido de la petición. Se observa como el identificador obtenido se encuentra encriptado.

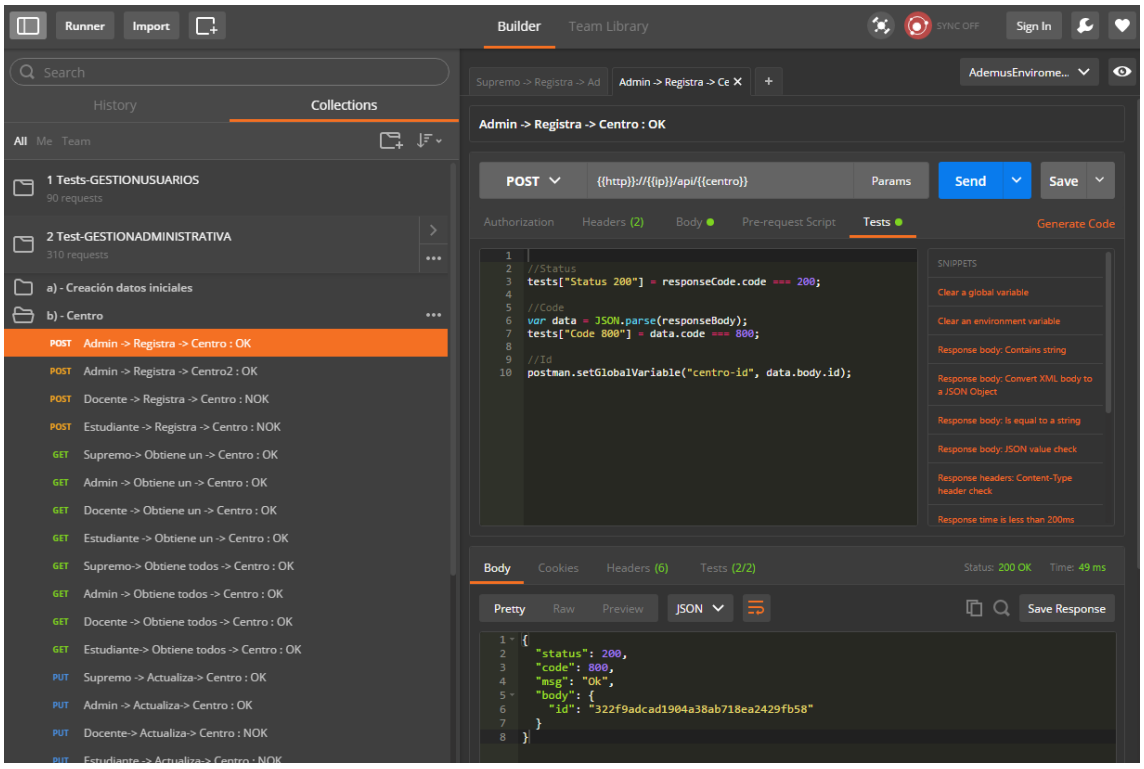

Ilustración 41 - Postman, definición de la petición (IV)

En la ilustración anterior, en la zona de la respuesta se muestra un apartado de *Test (2/2)* en verde, que indica que las pruebas definidas para la solicitud han pasado

correctamente. En la siguiente ilustración se muestra la manera en la que Postman representa el resultado de la ejecución de una única prueba unitaria.

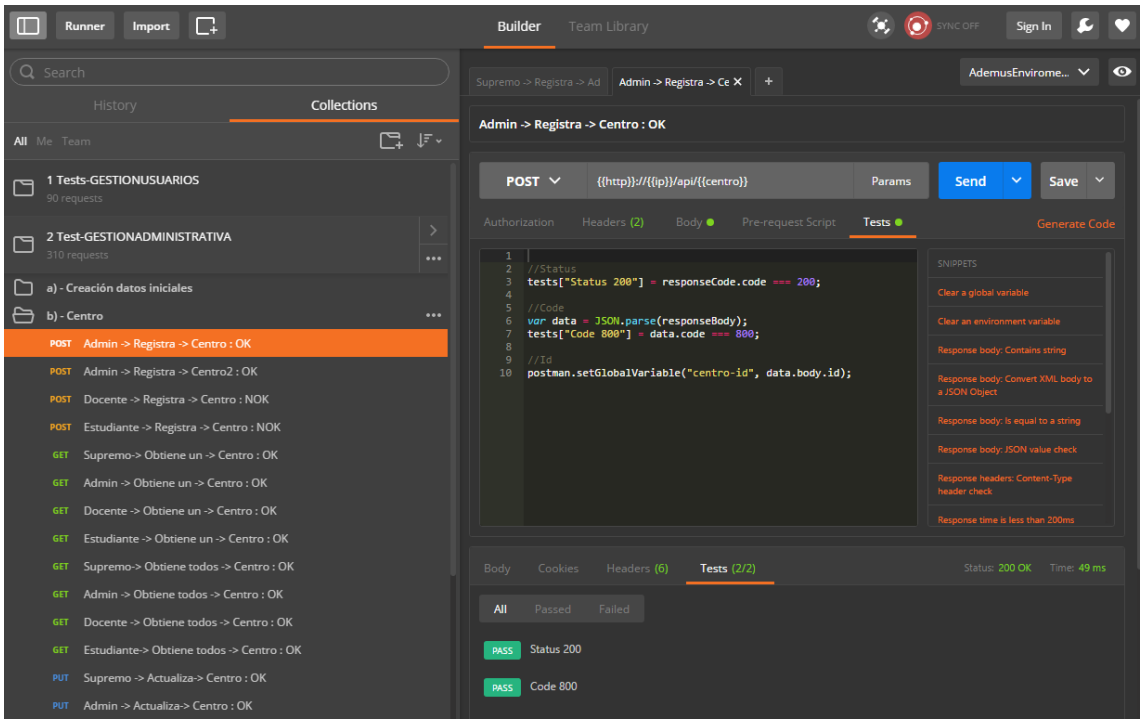

Ilustración 42 – Postman, definición de la petición (V)

Para entender de una manera rápida que prueba se está ejecutando, y cuál debe ser su resultado, en la definición del nombre de la prueba (marcado en naranja en la ilustración anterior) se encuentran cinco apartados, teniendo en cuenta que cada prueba definida tiene un resultado y pruebas asociadas.

POST Admin → Registra → Centro: OK

El primer apartado hace referencia al verbo utilizado en la solicitud, en este caso POST. El segundo apartado hace referencia al tipo de usuario que realiza la operación, un Administrador. El tercer apartado hace referencia a la acción que va a realizar, en este caso un registro. El cuarto parámetro indica a qué recurso va a llamar para realizar la operación. Y el quinto parámetro indica cual debería ser el resultado correcto. En el caso del Centro, un Administrador que quiera registrar un Centro, si introduce todos los datos necesarios para realizar la acción, el resultado de la operación debe ser un OK por parte del sistema.

Una vez explicado el funcionamiento de *Postman*, y habiendo definido las configuraciones necesarias para definir las pruebas, se explicarán los dos grandes bloques de pruebas unitarias mencionados anteriormente, los cuales, serán ejecutados de forma secuencial o individual, siempre respetando las dependencias de datos. Por ejemplo, para registrar un Docente es necesario saber previamente a que Centro y Departamento pertenece para poder realizar el registro.

7.2 Pruebas unitarias, usuarios

a) Check Entrada de datos

En este apartado se realizan las pruebas relacionadas con la duplicidad de registros, parámetros enviados a la API y control de identificaciones. Como se puede observar en la siguiente ilustración, primeramente se registra un usuario con rol de Supremo, y posteriormente se intenta crear un nuevo usuario Supremo, dando como resultado NOK, con código 200 porque la respuesta es correcta y el código 801 que según las especificaciones de los errores definidos en la API significa *Nok*. En determinadas situaciones no se aporta más información por motivos de seguridad. En el apartado de los Test se observa que *Test (2/2)* se encuentra en verde, esto indica que las pruebas definidas se ejecutaron correctamente. En este caso sabiendo que anteriormente se había creado otro usuario con Rol de Supremo, la ejecución de ésta tarea debe devolver NOK, y así es verificada en la definición de la prueba que dice que la respuesta cuando se ejecute dicha acción (previo registro de otro usuario con ese mismo nombre) debe devolver un código 801.

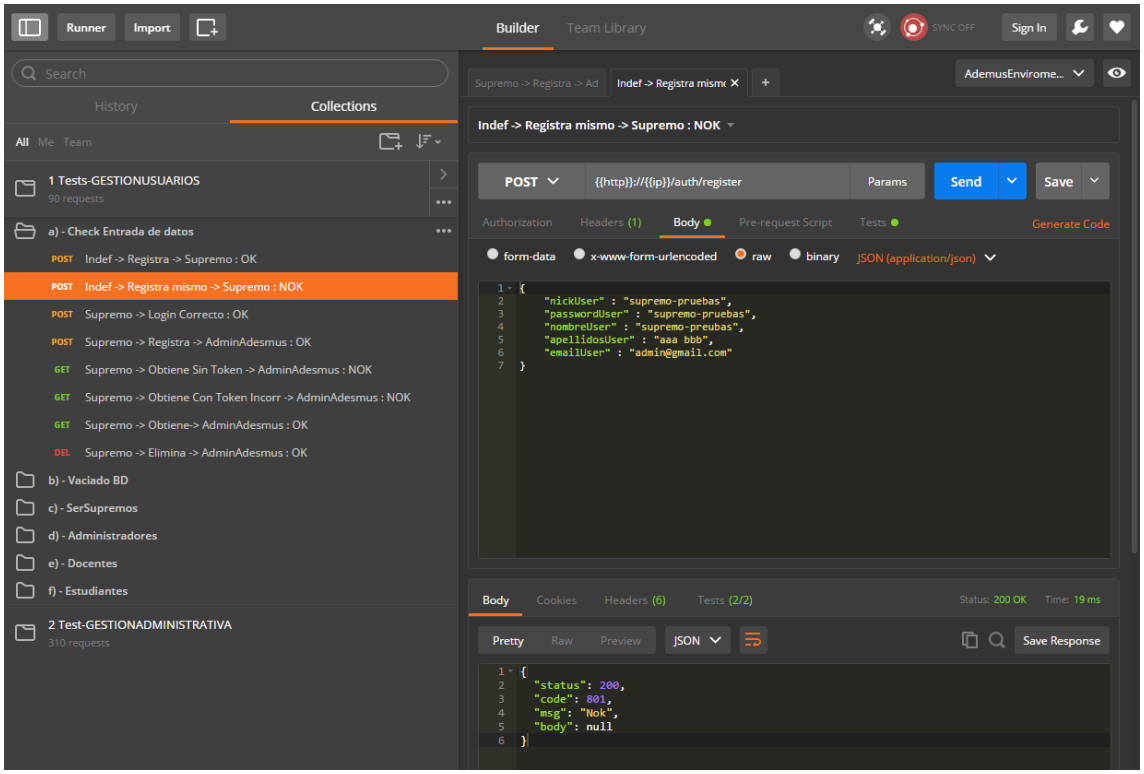

Ilustración 43 - Postman, pruebas unitarias - Check entrada de datos (I)

En la siguiente ilustración se observan otras pruebas como pueden ser la de obtener un Administrador (previamente creado por el propio usuario Supremo) sin especificar en la cabecera de la petición el Token y dando como respuesta error, debido a que en la

cabecera no se ha definido ningún Token. En la parte de los Test definidos para la petición también se observa *Test (2/2)* en verde, donde se ha controlado que la respuesta obtenida tiene que ser 200 pero con un código interno de 910, el cual, significa que no hay Token en la cabecera.

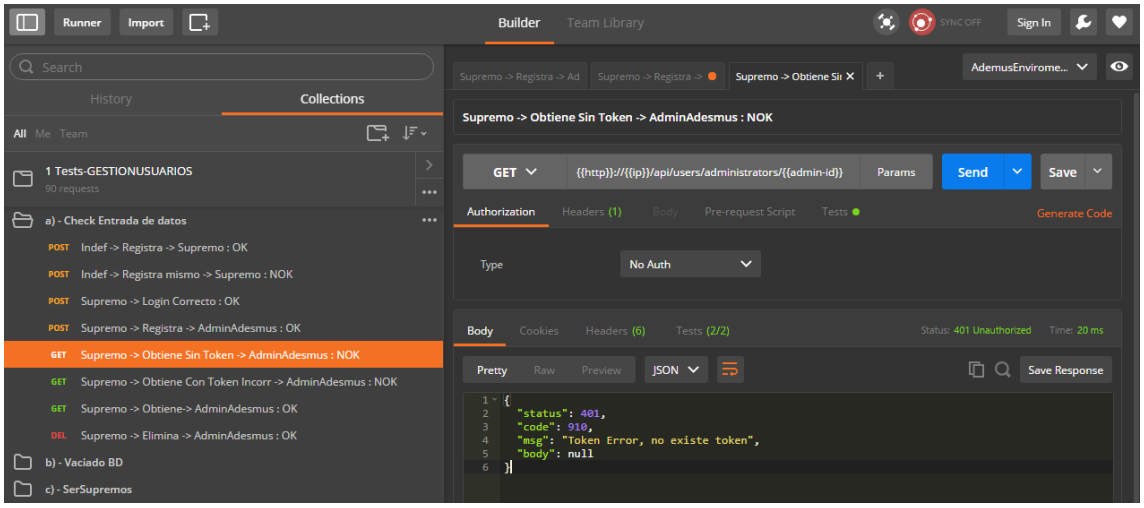

Ilustración 44 - Postman, pruebas unitarias - Check entrada de datos (II)

Si se intenta obtener un Administrador indicado un Token incorrecto en la cabecera de la petición, el sistema determina que el Token no es válido y responde al usuario con el código interno 912 que indica que el Token enviado no es correcto, como se muestra en la siguiente ilustración. En la parte de los Test se observa *Test (2/2)* en verde, que indica que la respuesta dada por el servidor es la que se ha definido en las pruebas, y por lo tanto es lo que debería de ocurrir.

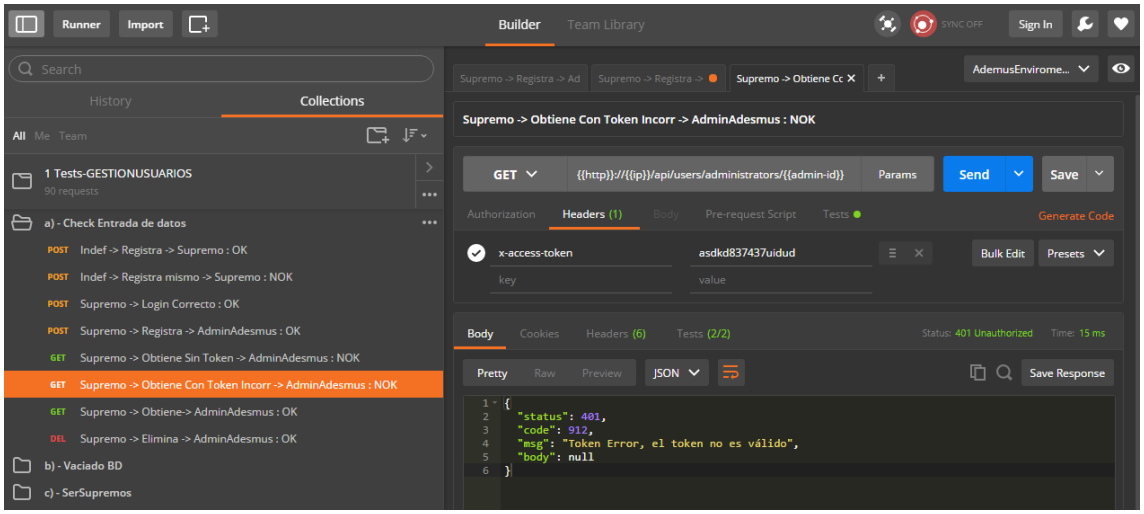

Ilustración 45 - Postman, pruebas unitarias - Check entrada de datos (III)

En la siguiente ilustración, el usuario Supremo, obtiene la información del Administrador, indicando su Token válido en la cabecera de la petición, y el sistema responde con la información del Administrador solicitado. En la Url se indica el administrador que se quiere obtener a través del parámetro *admin-id* cuyo identificador es previamente guardado en la operación del registro del Administrador. Recordar que el identificador guardado en *admin-id* está encriptado. En la zona de los Test se comprueba que el sistema devuelve una respuesta 200, y además, dentro del apartado *body* de la trama de datos de vuelta debe existir un elemento, en este caso el Administrador obtenido.

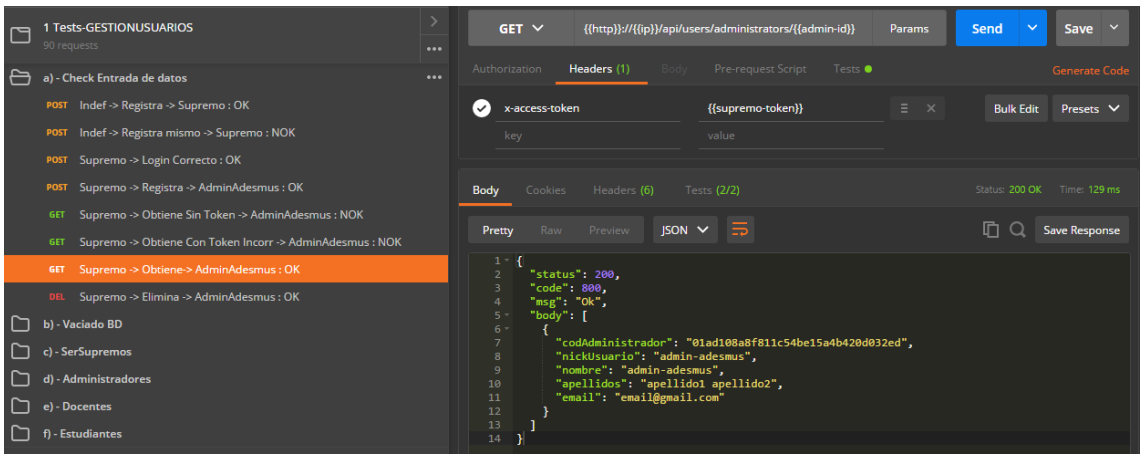

Ilustración 46 - Postman, pruebas unitarias - Check entrada de datos (IV)

Finalmente, en la siguiente ilustración se muestra como el usuario Supremo elimina el Administrador que ha registrado anteriormente, dando como resultado el número de filas afectadas en la eliminación. En la parte de los Test se comprueba que la acción de eliminación ha afectado a una línea del sistema.

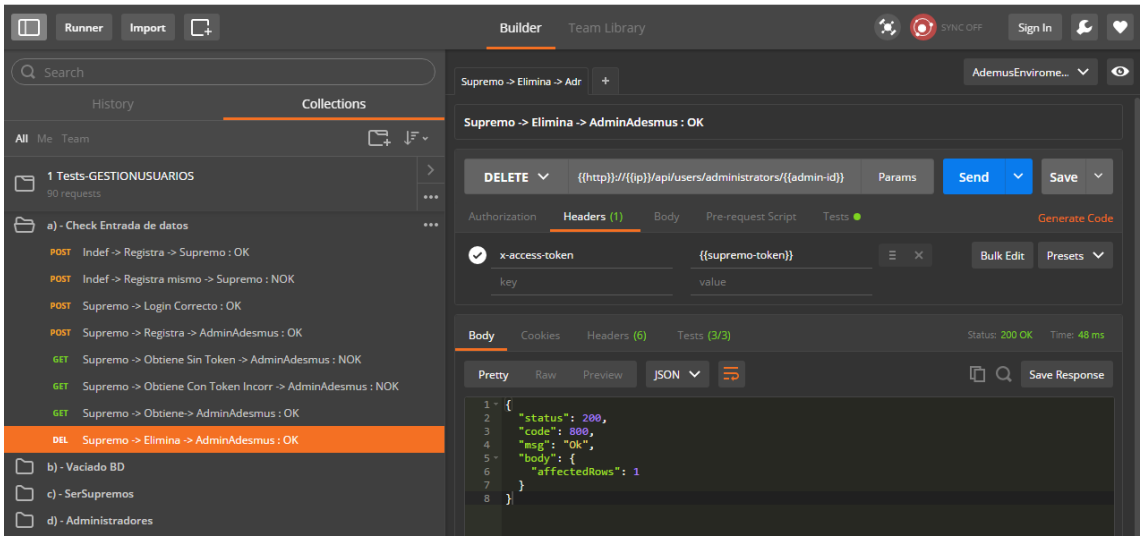

Ilustración 47 - Postman, pruebas unitarias - Check entrada de datos (V)

b) Vaciado BD

Se ha creado un apartado que permite eliminar todos los datos que existen en la base de datos. A excepción, del usuario Supremo, que no existe funcionalidad definida en la API para ser eliminado. Precisamente, para eliminar todos los registros de las tablas en la base de datos es necesario disponer de un Token con rol de Supremo. En la siguiente ilustración se muestran los recursos disponibles para el vaciado de todas las tablas y se observa el resultado de la ejecución de una de ellas, la eliminación de todos los docentes que se encuentran registrados en el sistema. En la parte de los Test se comprueba que se ha eliminado al menos un objeto de la tabla Docentes en la base de datos, además de comprobar que la respuesta tiene un código 200 y un código 800 interno que indica que la respuesta es correcta.

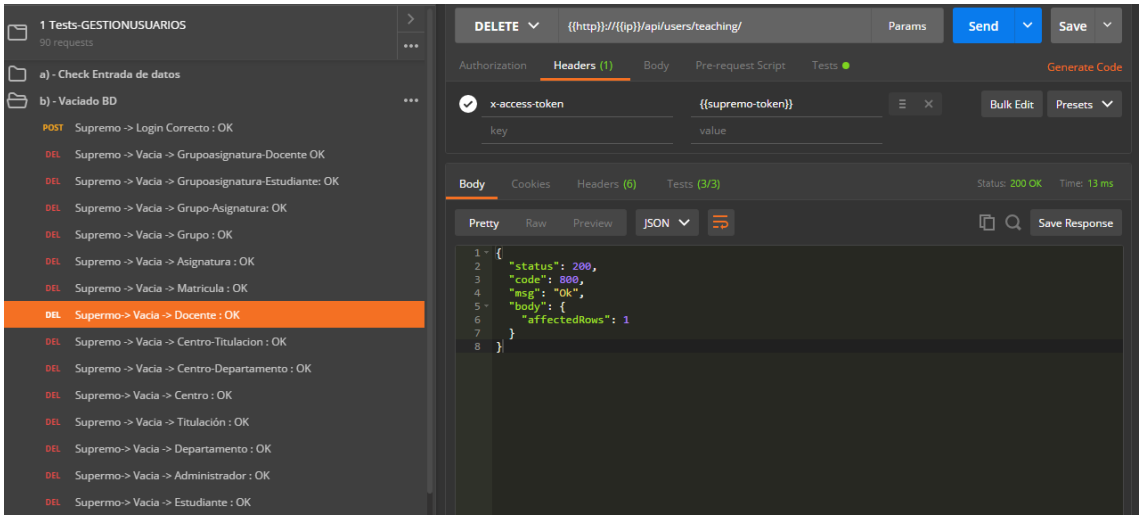

Ilustración 48 - Postman, pruebas unitarias - vaciado BD

c) Supremo

En el apartado de las pruebas del usuario Supremo, como se muestra en la siguiente ilustración, se muestra como el usuario introduciendo sus datos de forma errónea en la identificación obtiene un error, e introduciendo correctamente sus datos obtiene un Token, el cual, utilizará después para crear dos Administradores. Se comprobará también que el mismo Administrador (Nick de usuario) no puede registrarse por segunda vez. También se realizan las pruebas para obtener la información de un Administrador y todos los Administradores registrados en el sistema. Otra de las pruebas que se realizan es modificar el Administrador. Y para finalizar, el usuario Supremo elimina los dos Administradores creados en los pasos anteriores.

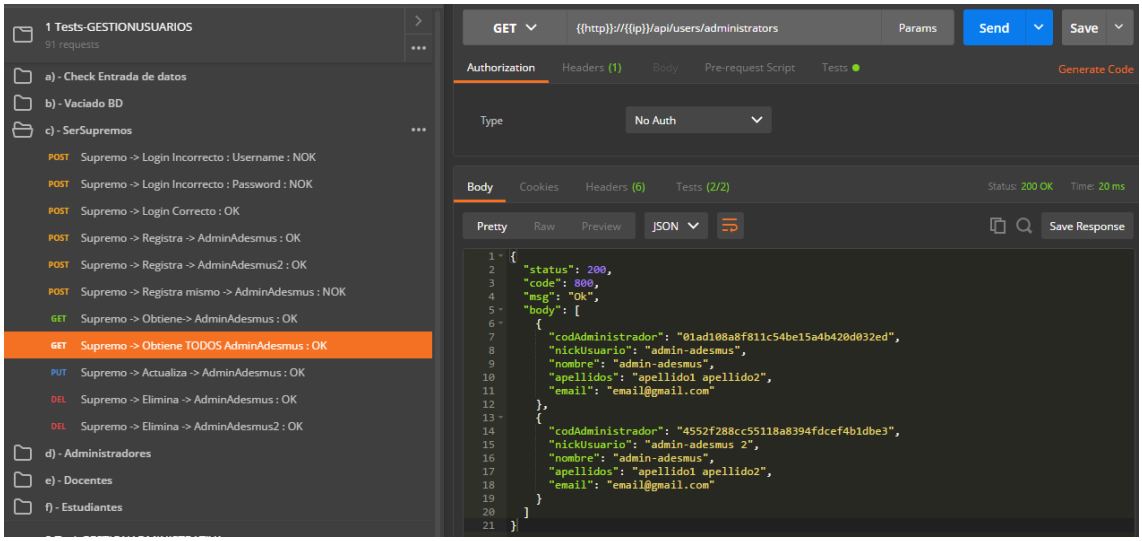

Ilustración 49 - Postman, pruebas unitarias - Supremo

d) Administradores

Dentro del apartado de pruebas para los Administradores, como se muestra en la siguiente ilustración, primeramente el usuario Supremo crea el Administrador. El propio Administrador realiza una identificación errónea, por nombre de usuario y contraseña. Posteriormente introduciendo correctamente los datos de identificación obtiene un Token válido. El propio Administrador crea un nuevo Administrador. Obtiene la información de sí mismo pasando por parámetro en la Url su identificador encriptado. Comprueba que puede obtener todos los Administradores registrados en el sistema. También se comprueba que el Administrador puede actualizar su perfil de Administrador pero obviamente no puede modificar el perfil de otro Administrador. Para terminar, el Administrador intenta eliminar otro Administrador y el sistema dice que no tiene los permisos necesarios para realizar dicha operación. Finalmente es el usuario Supremo el que realiza la operación de eliminar los dos Administradores creados en los pasos anteriores.

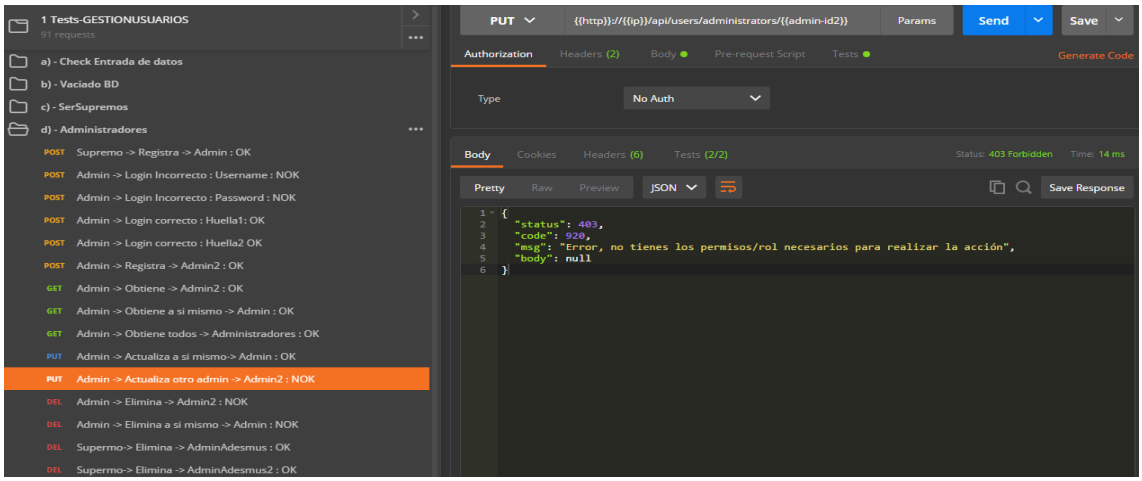

Ilustración 50 - Postman, pruebas unitarias - Administradores

e) Docentes

Para realizar las pruebas a un Docente, previamente es necesario registrar un Centro, un Departamento y asociar el Departamento al Centro. Estas operaciones EXTRA, no pueden ser realizadas por un Docente, por lo tanto las realiza el Administrador. Como se observa en la siguiente ilustración, se realiza la prueba de registro de un Docente por parte de un Administrador, pasando por parámetros en el *body* toda la información necesaria para su registro (incluido el centroDepartamentoId encriptado). Una vez que se ha realizado la prueba de registro del Docente, éste se identifica en el sistema, obtiene un Token válido para acceder a la API y comprueba que puede realizar las operaciones de obtener otros Docentes (incluido el mismo), obtener todos los Docentes registrados en el sistema, actualizar su perfil de Docente, pero no el de otros Docentes, de la misma manera que sucede en los Administradores. Se comprueba que el Docente no puede eliminar otros Docentes (incluido el mismo), y finalmente el usuario Supremo es el encargado de eliminar todos los registros realizados en los pasos anteriores.

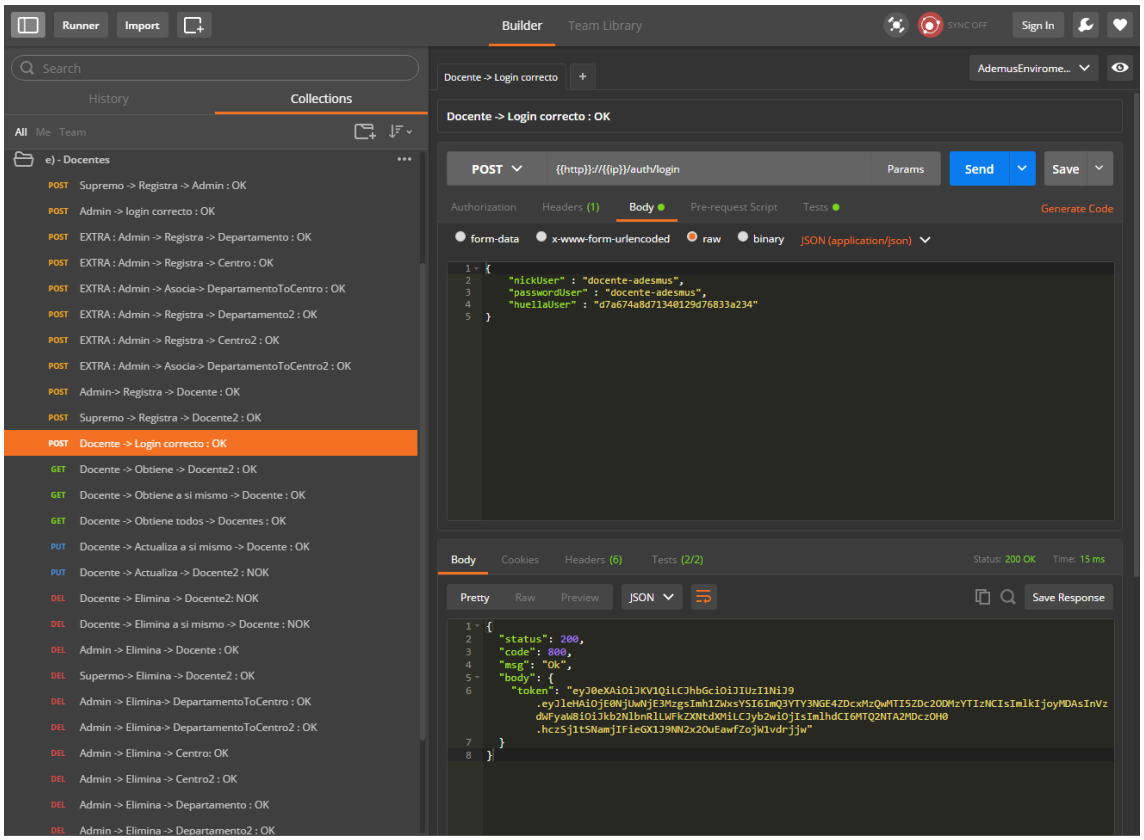

Ilustración 51 - Postman, pruebas unitarias - Docentes

f) Estudiantes

En la siguiente ilustración, el usuario con rol de Supremo registra un Administrador. El Administrador obtiene su Token válido y posteriormente registra dos Estudiantes. El primer Estudiante registrado realiza una identificación en el sistema y realiza las

operaciones de obtención de datos, obtiene la información de otro Estudiante, y también se obtiene a sí mismo. El Estudiante puede editar su perfil de usuario pero no puede editar el perfil de otros Estudiantes. El Estudiante intenta eliminar otros Estudiantes pero no tiene permisos para realizar éstas acciones en el sistema. Finalmente el usuario con rol de Supremo elimina los Estudiantes que ha registrado en los pasos anteriores.

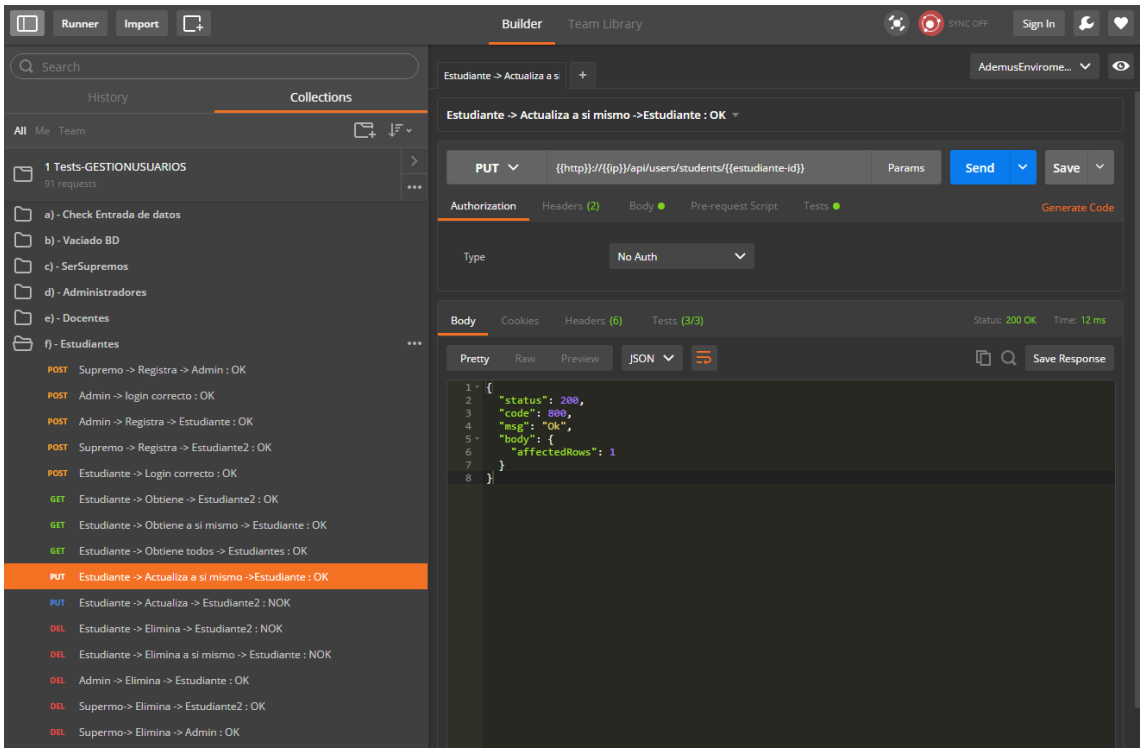

Ilustración 52 - Postman, pruebas unitarias - Estudiantes

7.3 Pruebas unitarias, gestión

a) Creación de datos iniciales

Inicialmente para la ejecución del apartado de pruebas de Gestión se realiza un registro previo de los usuarios que se usarán en el resto de las pruebas, como se puede observar en la siguiente ilustración, se registra un usuario Supremo (si no existe), el usuario Supremo a su vez registra un Administrador, y éste a su vez registra todos los datos necesarios para poder registrar dos Docentes y dos Estudiantes. Por último, todos los usuarios registrados realizan la identificación en el sistema y se obtienen sus Tokens, que son guardados en variables dentro de Postman para poder ser utilizados en futuras solicitudes.

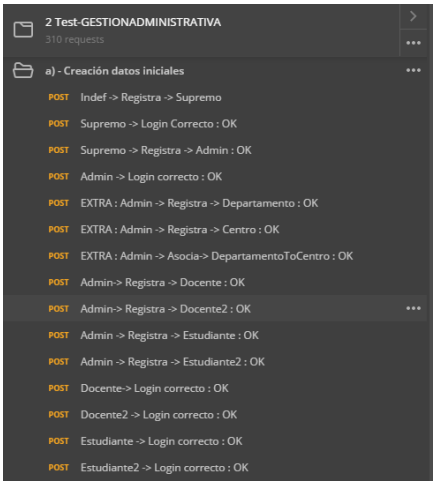

Ilustración 53 - Postman, pruebas unitarias, gestión, creación de datos iniciales

b) Centro

En este apartado se comprueban todas las operaciones disponibles para el Centro. El Administrador puede crear Centros, pero un Docente y un Estudiante no. Todos los usuarios identificados pueden obtener información de uno o varios Centros. Solamente los usuarios con rol Supremo y Administrador pueden modificar un Centro. Los usuarios con rol Estudiante y Docente no puede eliminar Centros registrados. En la siguiente ilustración se muestran todas las pruebas realizadas para el Centro, y además, el resultado de la ejecución de la prueba en la que un Docente intenta registrar un Centro.

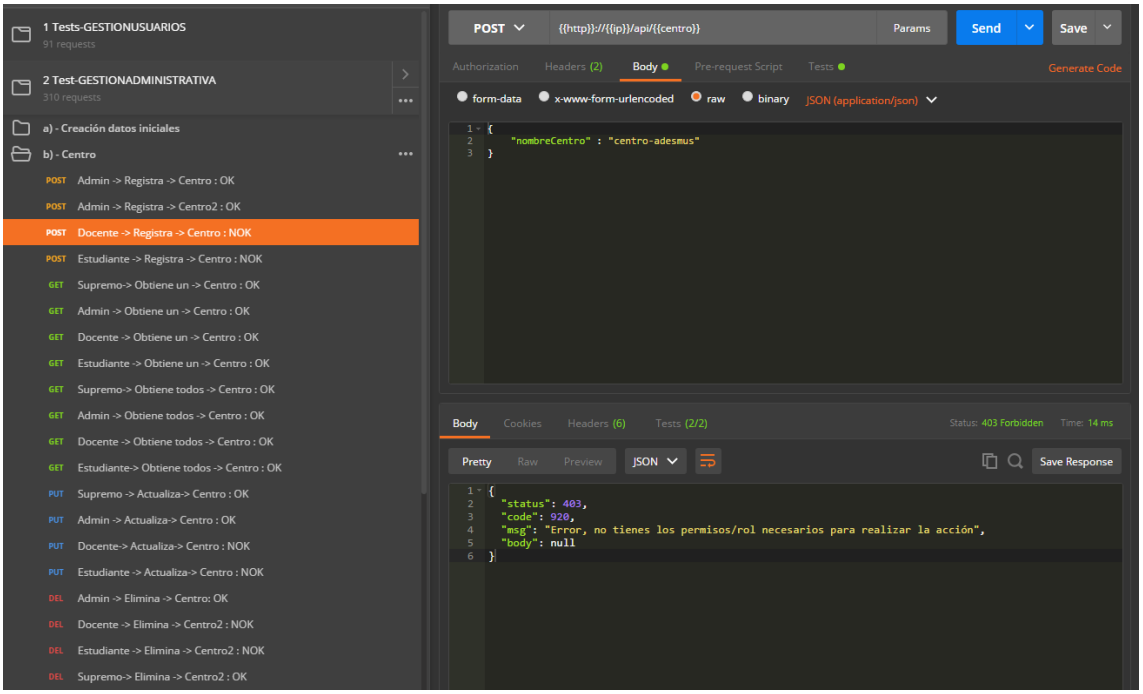

Ilustración 54 - Postman, pruebas unitarias, gestión, Centro

c) Departamento

En este apartado se verifican las operaciones sobre el Departamento. Solamente el usuario con rol Administrador y Supremo pueden registrar Departamentos en el sistema. En cambio, todos los usuarios identificados pueden obtener la información de uno o varios Departamentos registrados. Los usuarios con rol Estudiante y Docente no pueden realizar modificaciones en el Departamento. Los únicos roles que pueden eliminar Departamentos en el sistema son los de Supremo y Administrador. En la siguiente ilustración se muestran las pruebas del apartado Departamento, y el resultado de la ejecución de obtener todos los Departamentos registrados utilizando un Token con rol de Supremo.

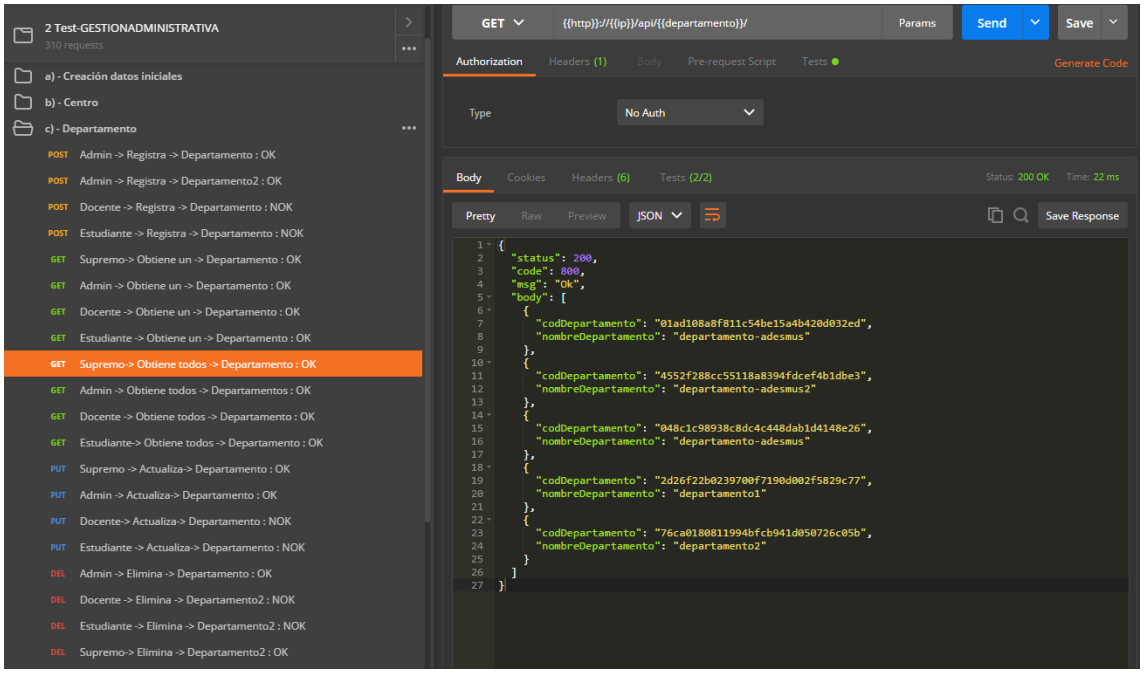

Ilustración 55 - Postman, pruebas unitarias, gestión, Departamento

d) CentroDepartamento

En este apartado se realizan todas las pruebas sobre el recurso que asocia los Departamentos a los Centros. El administrador crea los Departamentos y los Centros, y los asocia. Todos los roles de usuario identificado pueden obtener la información de los Departamentos que se encuentran asociados a un Centro. Finalmente el Administrador elimina los datos creados para la ejecución de éste apartado. En la siguiente ilustración se muestran las pruebas realizadas en el apartado de pruebas para el CentroDepartamento y además se muestra la ejecución de una de las pruebas donde un usuario con rol Docente obtiene la asociación de un Departamento que se encuentra asociado a un Centro.

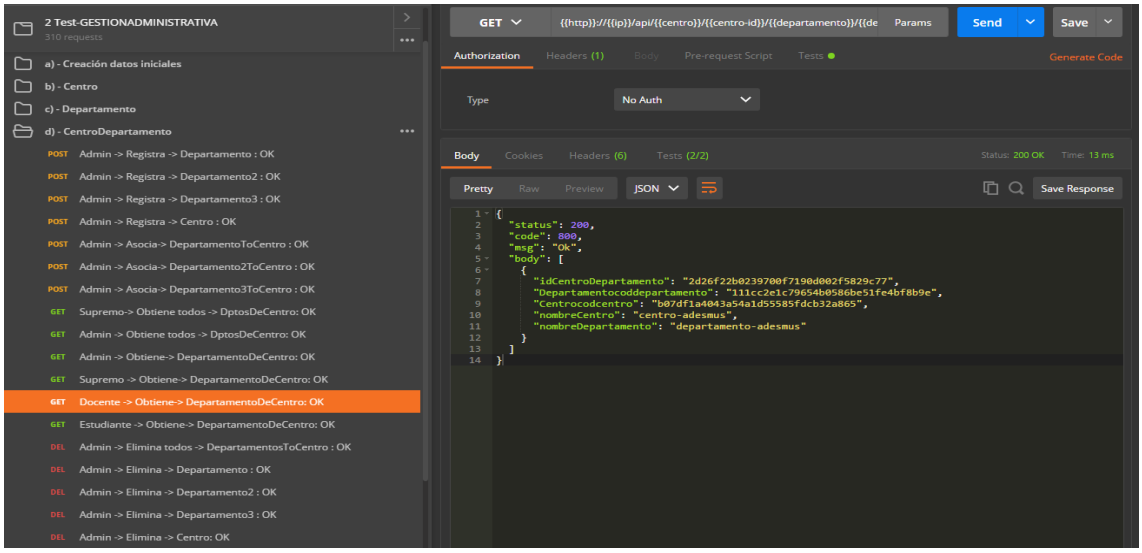

Ilustración 56 - Postman, pruebas unitarias, gestión, CentroDepartamento

e) Titulación

En este apartado se realizan todas las pruebas relacionadas con las Titulaciones. El usuario con rol Administrador registra una serie de Titulaciones. Se comprueba que el usuario con rol Docente y Estudiante no pueden realizar registros de Titulaciones. Todos los roles de usuarios identificados pueden obtener información sobre una o varias Titulaciones registradas en el sistema. Solamente los usuarios con rol de Administrador y Supremo pueden realizar modificaciones en las Titulaciones. Los usuarios con rol de Estudiante y Docente no pueden eliminar Titulaciones del sistema. En la siguiente ilustración se muestran las pruebas realizadas para el apartado de las Titulaciones, y además, la ejecución de una prueba teniendo en cuenta que el Token ha expirado en tiempo, dando como resultado, una respuesta errónea.

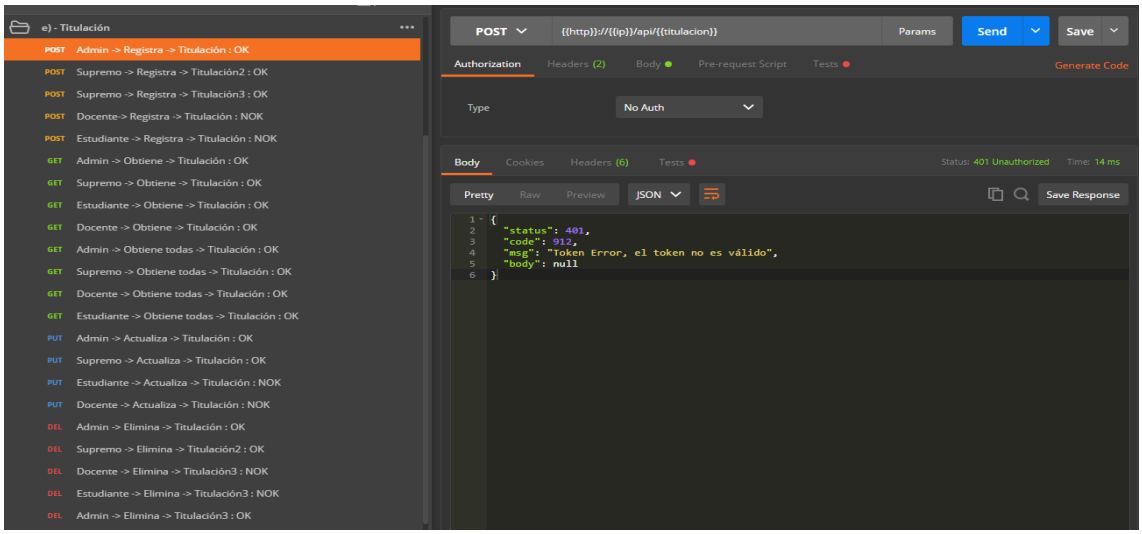

Ilustración 57 - Postman, pruebas unitarias, gestión, Titulación

f) Grupo

En este apartado se verifican todas las operaciones que se pueden realizar sobre un Grupo. Solamente los usuarios con rol Supremo y Administrador pueden realizar registros de Grupos en el sistema. Todos los roles de usuario identificado pueden obtener información sobre uno o varios Grupos registrados en el sistema. Los usuarios con rol Estudiante y Docente no pueden realizar modificaciones sobre los Grupos registrados. Solamente los usuarios con rol Administrador y Supremo pueden eliminar Grupos del sistema. En la siguiente ilustración se muestran todas las pruebas realizadas para el apartado de Grupos, y además, el resultado de la ejecución de una de las pruebas, donde un usuario con rol Administrador actualiza un Grupo, dando como resultado el número de filas afectadas en la actualización.

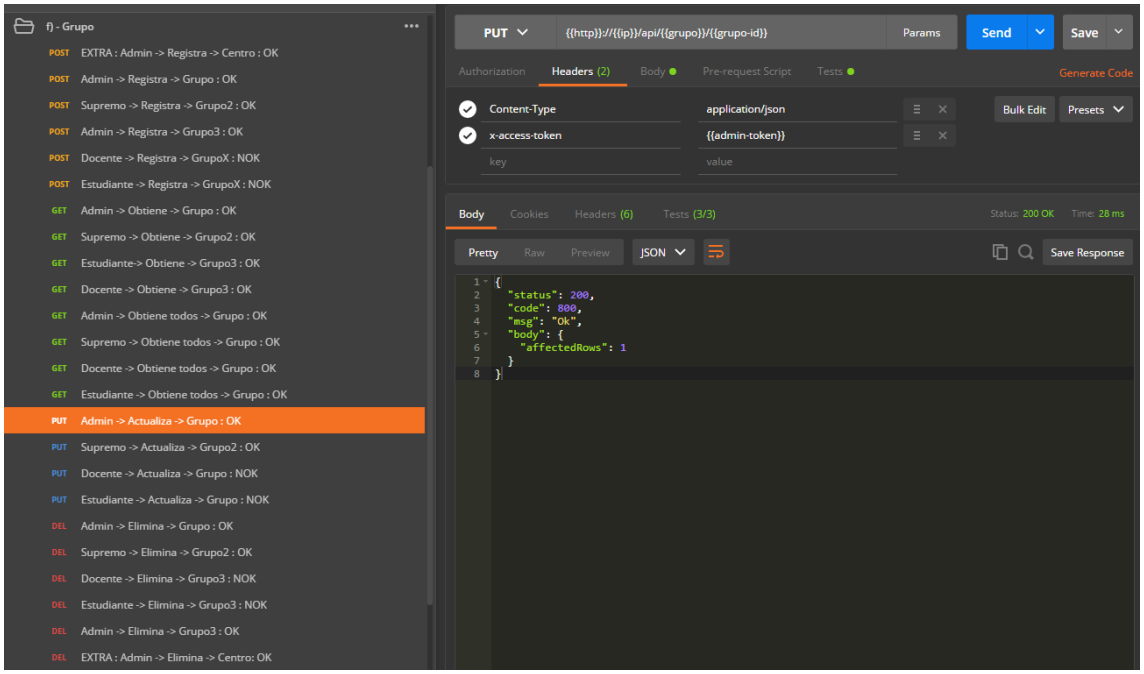

Ilustración 58 - Postman, pruebas unitarias, gestión, Grupo

g) Asignatura

En este apartado se verifican todas las operaciones que se realizan sobre las Asignaturas. El usuario con rol Administrador crea diferentes Titulaciones. Posteriormente, en el registro de las Asignaturas, indica para cada Asignatura a qué Titulación corresponde. Los usuarios con rol Estudiante y Docente no pueden registrar Asignaturas. Solamente pueden realizar modificaciones en las Asignaturas los usuarios con rol de Administrador o Supremo. La eliminación corre a cargo de un usuario con rol de Administrador o Supremo. En la siguiente ilustración se muestran todas las pruebas realizadas para el apartado de las Asignaturas, y además, el resultado de la ejecución de la prueba en la que un usuario con rol Administrador obtiene la información de todas las Asignaturas registradas.

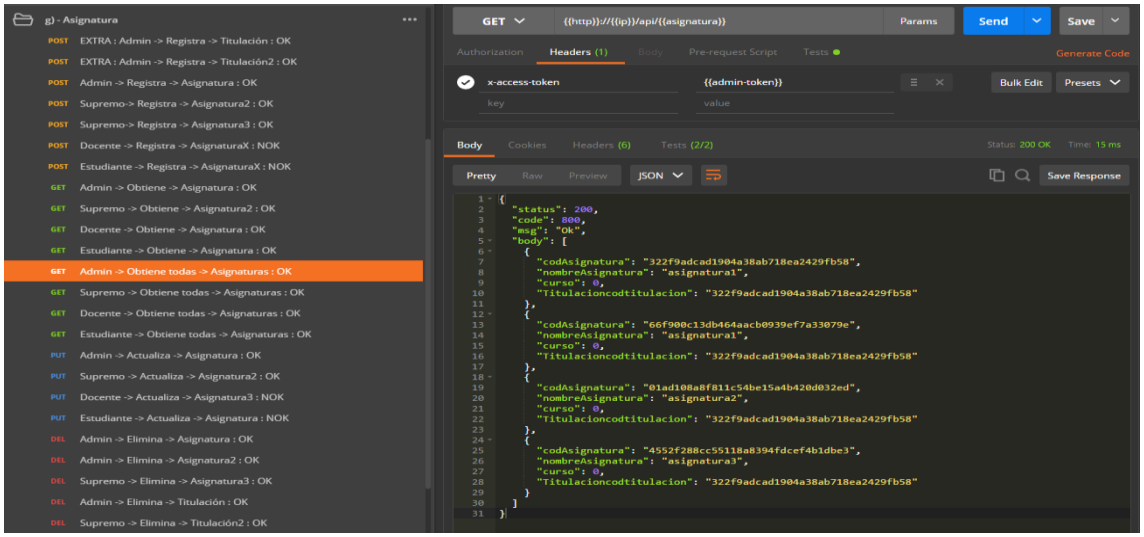

Ilustración 59 - Postman, pruebas unitarias, gestión, Asignatura

h) GrupoAsignatura

En el siguiente apartado se verifican todas las operaciones que se pueden realizar en la asociación de Asignaturas a Grupos. El usuario con rol Administrador registra un Centro y una Titulación. Posteriormente, para registrar un Grupo indica a qué Centro pertenece. El usuario con rol Administrador registra tres Asignaturas y las asocia al Grupo registrado en el paso anterior. Los usuarios con rol de Docente y Estudiante no pueden asociar Asignaturas a Grupos. Todos los roles de usuario identificado pueden obtener información de las Asignaturas asociadas a los diferentes Grupos registrados. Solamente los usuarios con rol Administrador y Supremo pueden eliminar las asociaciones de Asignaturas a Grupos. En la siguiente ilustración se muestran las pruebas realizadas para el apartado GrupoAsignatura, y además, se muestra la ejecución de una de las pruebas donde un usuario con rol de Supremo asocia una Asignatura a un Grupo.

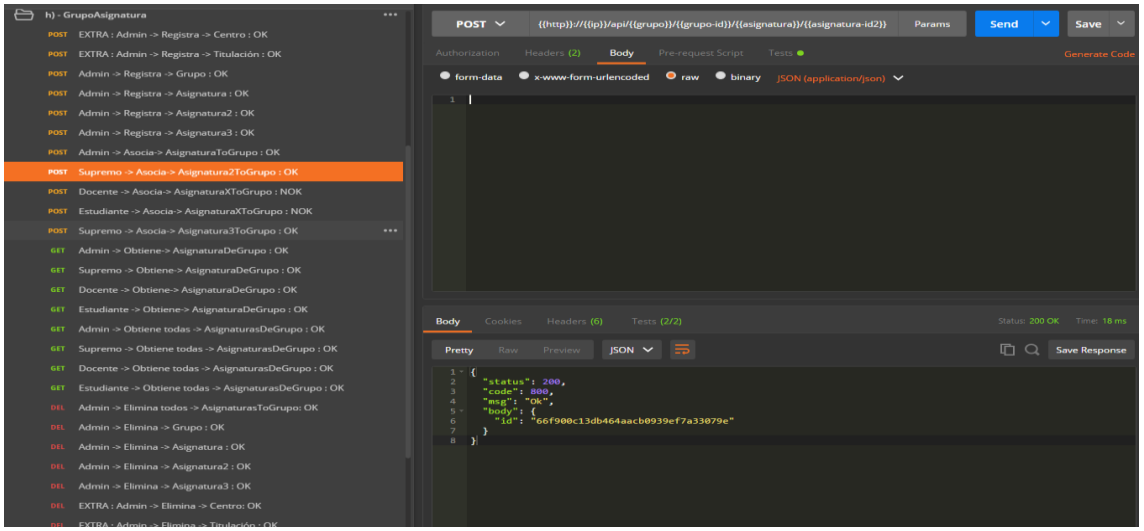

Ilustración 60 - Postman, pruebas unitarias, gestión, GrupoAsignatura

i) GrupoasignaturaDocente

En este apartado se verifican las operaciones que se realizan sobre las Asignaturas de Grupos que se encuentran vinculadas a un usuario con rol Docente. Los usuarios con rol Estudiante no podrán asociar a un usuario con rol Docente Asignaturas de Grupos. Tampoco un usuario con rol Docente podrá asociar Asignaturas de Grupos a otro usuario con rol Docente. Únicamente el propio usuario con rol Docente podrá asociarse a las Asignaturas de Grupos que cree oportuno. El usuario con rol Docente puede obtener la información de las Asignaturas de Grupos a las que está asociado, pero no podrá obtener la información de las Asignaturas de Grupos que tenga asociadas otro Docente. En cambio, el usuario con rol Administrador podrá acceder a la información de todas las Asignaturas de Grupos que tenga asociada cualquier Docente. Los propios usuarios con rol Docente podrán eliminar la asociación a dichas Asignaturas de Grupo, pero no podrán eliminar Asignaturas de Grupo que tenga asociado otro Docente. El usuario con rol de Supremo y Administrador podrán eliminar cualquier asociación de Asignaturas de Grupo a Docentes. En la siguiente ilustración se muestran todas las operaciones realizadas para el GrupoasignaturaDocente.

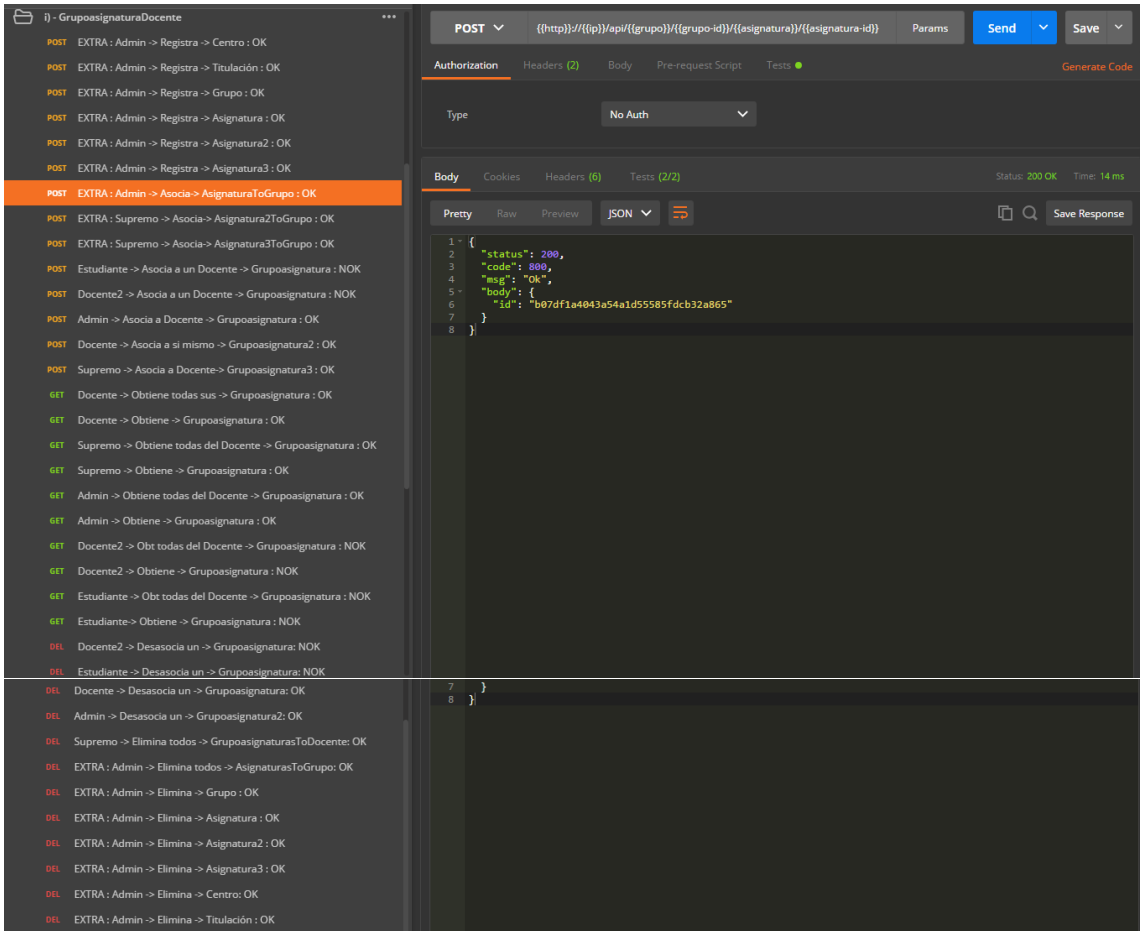

Ilustración 61 - Postman, pruebas unitarias, gestión, GrupoasignaturaDocente

j) GrupoasignaturaEstudiante

En este apartado se verifican las operaciones que se realizan sobre las Asignaturas de Grupos que se encuentran vinculadas al usuario con rol Estudiante. A la inversa de la situación en el apartado anterior, un usuario con rol Docente no pueden asociar a un Estudiante Asignaturas de Grupos, de la misma manera que un usuario con rol Estudiante no puede asociar Asignaturas de Grupo a otro Estudiante. Solamente el propio usuario con rol Estudiante puede asociarse a las Asignaturas de Grupo que considere oportuno. Los usuarios con rol Estudiante y Docente no podrán obtener información de las Asignaturas de Grupo que tenga asociadas otro Estudiante. Los usuarios con rol Administrador y Supremo podrán realizar todas las operaciones disponibles. En la eliminación, los Estudiantes podrán eliminar sus asociaciones a las Asignaturas de Grupos, pero no podrán eliminar asociaciones de otros Estudiantes. En la siguiente ilustración se muestran todas las operaciones realizadas para el GrupoasignaturaEstudiante.

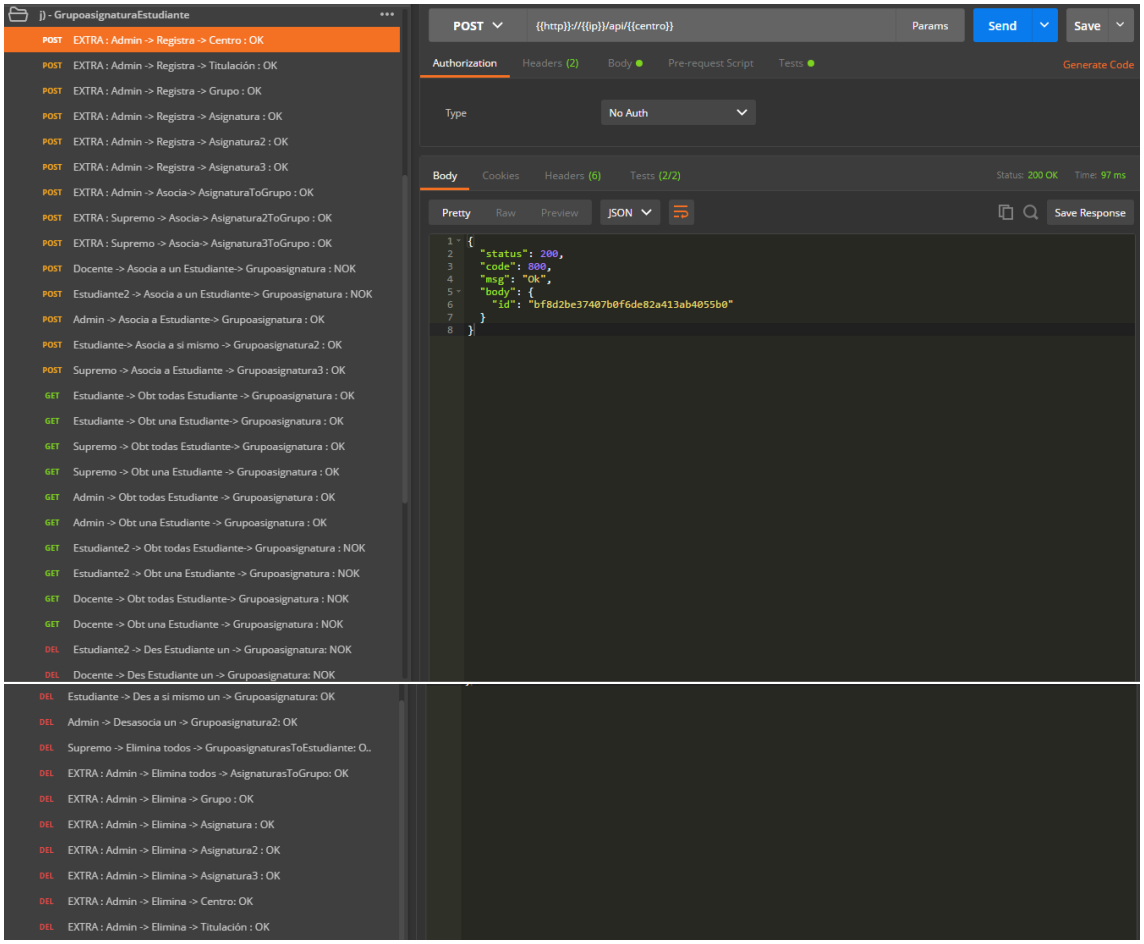

Ilustración 62 - Postman, pruebas unitarias, gestión, GrupoasignaturaEstudiante

k) CentroTitulación

En este apartado se verifican todas las pruebas que se realizan sobre las asociaciones de Titulaciones a los Centros registrados en el sistema. Los usuarios con rol Estudiante y

Administrador no pueden asociar una Titulación a un Centro. Todos los usuarios identificados pueden obtener información de las Titulaciones asociadas a cada Centro. Los usuarios con rol Docente y Estudiante no pueden eliminar las Titulaciones asociadas a los Centros. En la siguiente ilustración se muestran todas las operaciones realizadas para el CentroTitulación, y además, se muestra la ejecución de la prueba en la que un usuario con rol Docente obtiene todas las Titulaciones de un Centro.

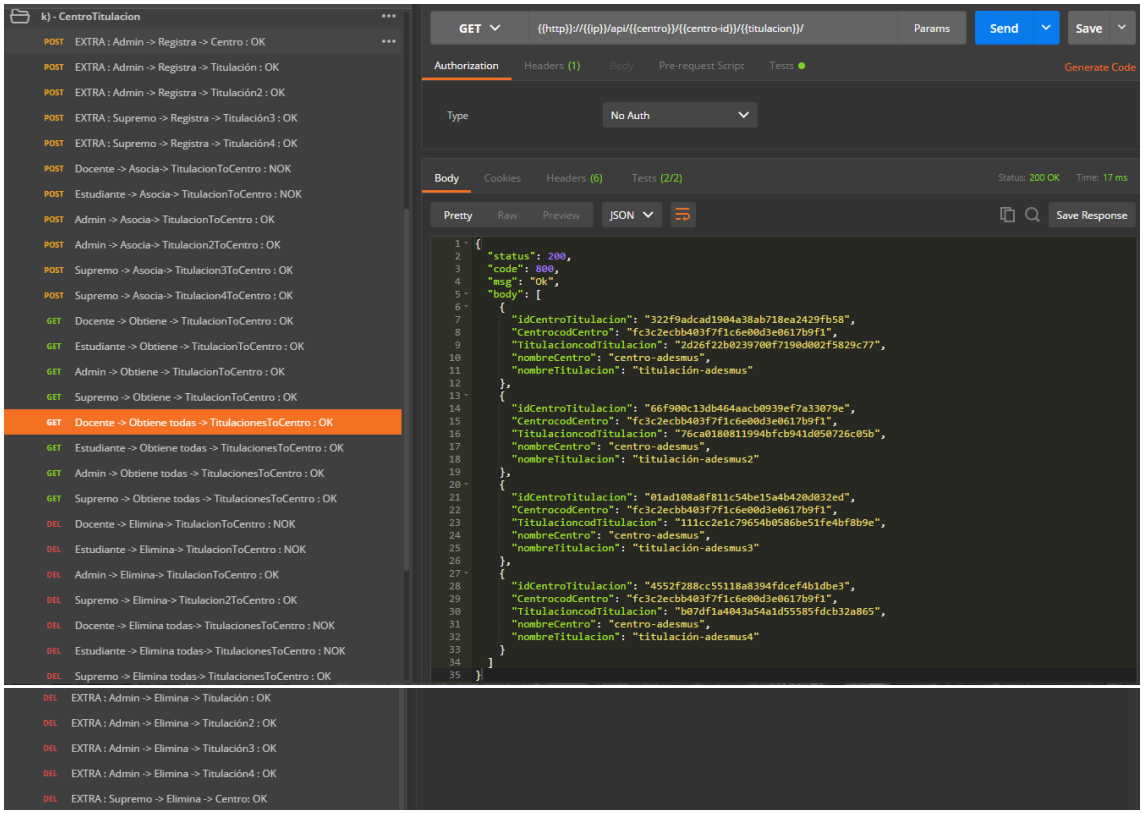

Ilustración 63 - Postman, pruebas unitarias, gestión, CentroTitulacion

l) Matrícula

En este apartado se verifican todas las pruebas que se realizan sobre las Matrículas. Los usuarios con rol Docente y Estudiante no pueden registrar Matrículas. Solamente los usuarios con rol Administrador y Supremo pueden conocer el contenido de las Matrículas que han sido registradas. Si fuera necesario conocer el contenido o estado de la matrícula, el usuario con rol Estudiante debe solicitar la información al Administrador. Los usuarios con rol Administrador y Supremo son los únicos con capacidad para eliminar Matrículas registradas en el sistema. En la siguiente ilustración se muestran las operaciones realizadas sobre las Matrículas, y además, el resultado de la ejecución de una de las pruebas donde el usuario con rol Administrador obtiene la información de una de las Matrículas existentes en el sistema.

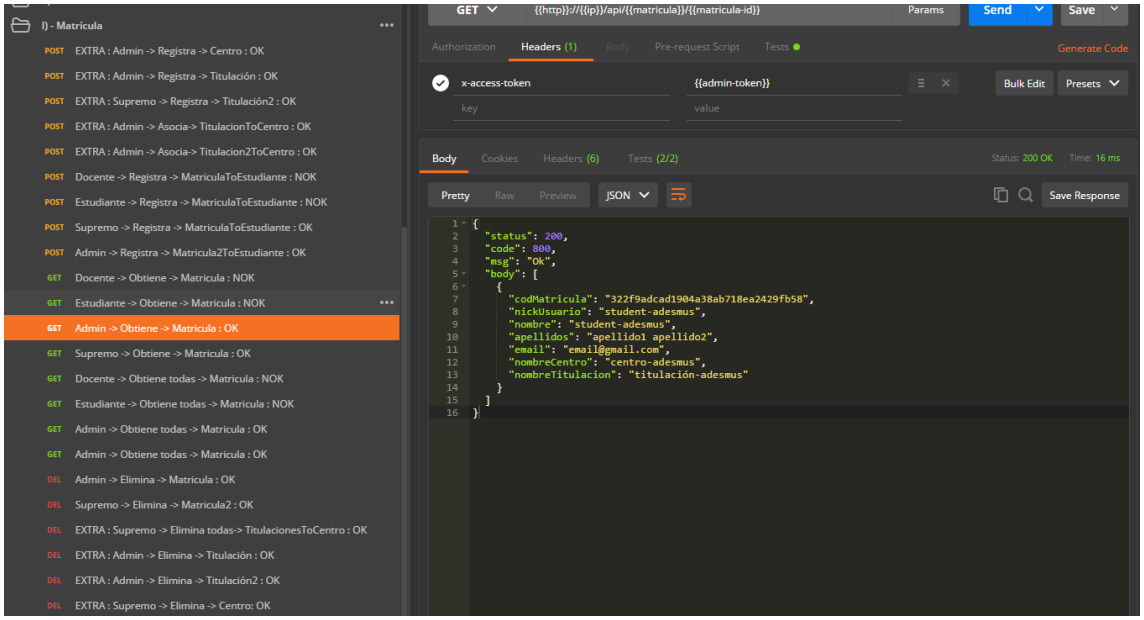

Ilustración 64 - Postman, pruebas unitarias, gestión, Matrícula

m) Vaciado BD

Para finalizar las pruebas, se vacían todas las tablas, con el fin de eliminar posibles registros que hayan quedado guardados durante las pruebas. En la siguiente ilustración se muestran las pruebas realizadas para el vaciado de las tablas de la base de datos. Solamente el usuario con rol Supremo pueden realizar las operaciones de vaciado. El orden de ejecución de las pruebas es importante, debido a la integridad referencial existente en la base de datos.

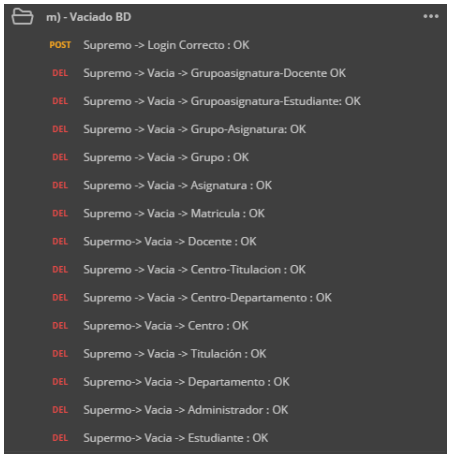

Ilustración 65 - Postman, pruebas unitarias, gestión, Vaciado BD

Para finalizar con el apartado de las pruebas, y una vez que se ha explicado de manera individual el funcionamiento de las mismas, sólo queda lanzar las dos baterías de pruebas y ver el resultado. En las dos siguientes ilustraciones se muestra el resultado de la ejecución de las dos baterías de pruebas: gestión usuarios y gestión administrativa.

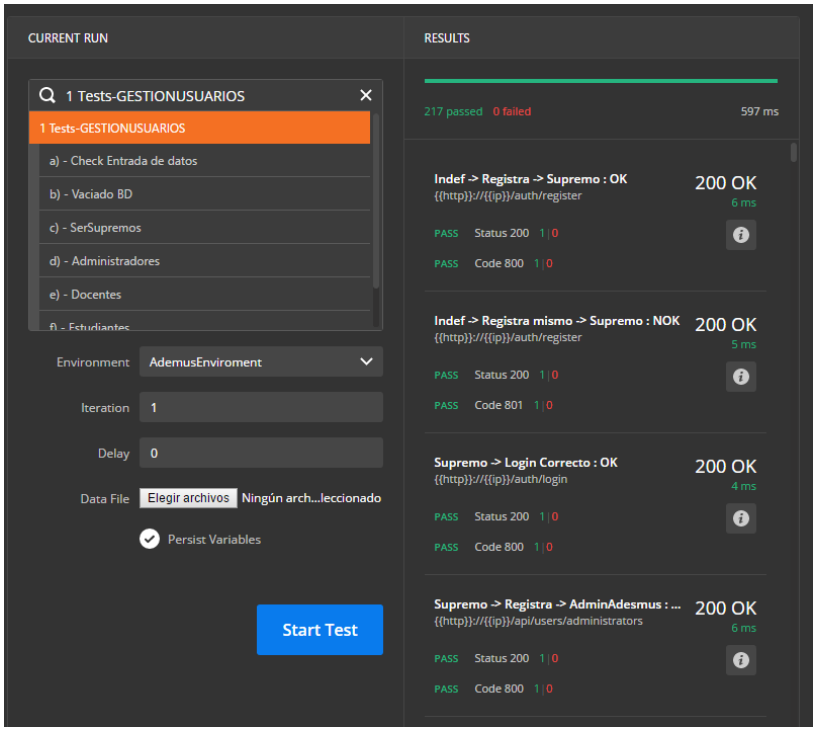

Ilustración 66 - Pruebas unitarias, Gestión usuarios

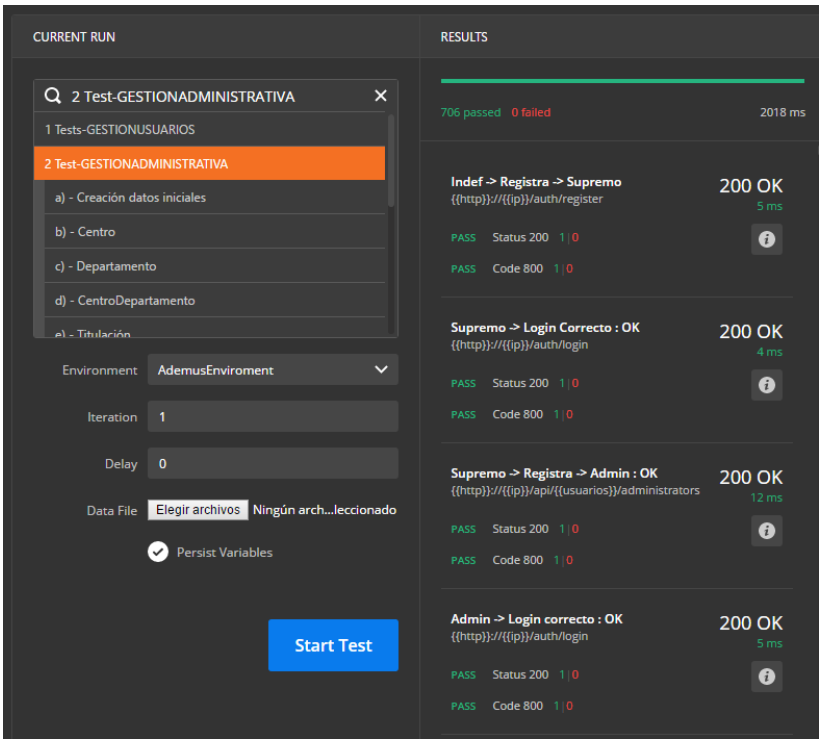

Ilustración 67 - Pruebas unitarias, Gestión administrativa

8 CONCLUSIONES Y TRABAJO FUTURO

En este apartado se tratan las conclusiones finales del proyecto, tanto a nivel de gestión, como desde un punto de vista crítico personal.

8.1 Conclusiones de gestión

En cuanto a la planificación temporal realizada al comienzo del proyecto, resaltar que se ha producido un retraso considerable en gran parte de las tareas establecidas en el EDT. Principalmente, debido a problemas personales, compaginar el trabajo y el desarrollo del proyecto. En cambio, en otras tareas el tiempo invertido ha sido algo inferior de lo esperado.

El apartado de *captura de requisitos* no se ha visto afectado en cuanto tiempo total de ejecución de las tareas establecidas. Los apartados de *análisis* y *diseño* tampoco se han visto afectados en cuanto al tiempo total de ejecución, pero cabe destacar, que aunque el tiempo total de la ejecución de las tareas cumple con lo establecido en la planificación temporal, algunas de éstas tareas fueron realizadas en menor tiempo, y otras, en las cuales se empleó más tiempo de lo debido. La búsqueda de tecnologías y herramientas se retrasó por la gran cantidad de información obtenida para su análisis, y el aprendizaje y familiarización de las herramientas fue más rápido debido al conocimiento previo aportado al comienzo del proyecto. El apartado de *implementación* se ha visto considerablemente afectado en cuanto a la ejecución de los tiempos establecidos. Debido a las problemáticas encontradas durante el desarrollo de los primeros prototipos implementados, donde se encontraron soluciones aplicables para el desarrollo del resto de los prototipos. El apartado del *plan de pruebas* también se ha visto afectado en los tiempos marcados, debido a la gran cantidad de pruebas a definir. La tarea de *documentación* en ocasiones ha sido dura, y también se ha visto afectada en los tiempos de ejecución establecidos inicialmente.

En cuanto a los riesgos, durante la ejecución del plan de pruebas se perdió la información de las pruebas definidas en *Postman*. Este riesgo ya estaba contemplado en la gestión de riesgos, se rescató la última copia de seguridad realizada, y se pudo continuar con la definición de las pruebas unitarias sin surtir efecto en los tiempos marcados en la planificación. Fue mínima la información perdida.

A continuación, se muestra una ilustración y una tabla con las diferencias de horas establecidas al comienzo del proyecto y a su finalización.

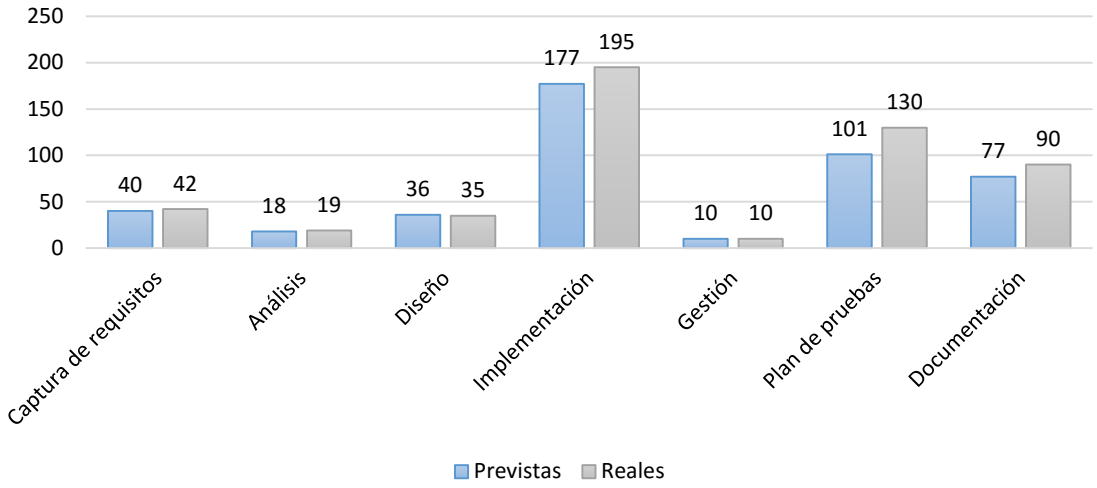

Ilustración 68 - Gráfica comparativa: horas estimas y reales del proyecto

Tareas	Estimadas (horas)	Reales (horas)
Captura de requisitos	40	42
Análisis	18	19
Diseño	36	35
Implementación	177	195
Gestión	10	10
Plan de pruebas	101	130
Documentación	77	90
TOTAL	**459**	**521**

Tabla 6 - Tabla comparativa: horas estimadas/reales, desarrollo del proyecto

De la tabla anterior, se deduce que el incremento a la finalización del proyecto asciende a 62 horas. Este incremento se ve afectado también en la evaluación económica realizada al inicio del proyecto. El gasto del personal finalmente asciende a 11.462 euros, y por ende, la amortización del hardware también se ve afectado ascendiendo a 29 euros. En la siguiente tabla se muestran los incrementos económicos que afectan a los gastos totales del proyecto.

	Estimados (euros)	Reales (euros)
Personal	**10.098**	**11.462**
Hardware	**25**	**29**
Ordenador portátil	16	19
Ordenador sobremesa	9	10
Software	**0**	**0**
Otros	**500**	**500**
Total	**10.623**	**11.991**

Tabla 7 - Tabla comparativa: gastos totales estimados y reales del proyecto

8.2 Conclusiones de objetivos

Una vez finalizado el proyecto, comentar que los objetivos se han cumplido satisfactoriamente. Se ha conseguido desarrollar un sistema administrable utilizando una arquitectura REST, a la cual, se le han implementado diferentes capas de seguridad para proporcionar un acceso seguro a los datos del sistema. También se ha cumplido el objetivo de definir un entorno de pruebas unitarias robusto. Otro de los objetivos marcados, era el uso de JavaScript en la parte servidora, el cual, debido a sus características como la ejecución asíncrona de tareas y velocidad, proporciona al usuario un acceso rápido a sus datos. El último de los objetivos cumplidos está relacionado con la usabilidad del sistema. Se ha desarrollado un sistema fácil de usar e intuitivo, que permite al usuario desde diferentes dispositivos acceder a sus datos registrados en el sistema.

8.3 Conclusiones personales

A nivel personal, y en cuanto a la parte técnica, resaltar sobre todo los conocimientos adquiridos en el desarrollo de una arquitectura REST bajo JavaScript. Observando el resultado y el excelente comportamiento del sistema desarrollado, valoro en un futuro cercano desarrollar e implementar un nuevo sistema REST para el entorno en el que trabajo.

En cuanto a la parte personal, cabe descartar sobre todo la capacidad de superación en diferentes momentos durante el desarrollo del proyecto. Ha sido complicado compaginar mi trabajo personal y el desarrollo de AdESMuS, pero ha sido infinitamente más gratificante ver el resultado y todo lo aprendido.

8.4 Conclusiones sobre líneas futuras

En cuanto a las futuras mejoras para el sistema AdESMuS, resaltar sobre todo la revisión de la gestión interna de la ejecución de funciones asíncronas. La anidación de los métodos y las esperas entre tareas en ocasiones ha sido ardua. Para evitar estos problemas mencionados una buena solución sería la utilización de promesas. Las *promesas* son un paradigma de la programación en JavaScript que permite trabajar mejor con funciones asíncronas. Entre sus principales ventajas destacar sobre todo que mejora la legibilidad del código, se evita la anidación de funciones y la utilización de CALLBACK´s, mejora la experiencia en el tratamiento de los errores puesto que el código se estructura de una forma más eficiente y agradable a la vista.

9 BIBLIOGRAFÍA

Rest

http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

http://www.tutorialspoint.com/restful/index.htm

NodeJS

https://nodejs.org/en/docs/

Express (NodeJS)

http://expressjs.com/es/guide/routing.html

Promesas

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promesa

ANEXO I. CASOS DE USO EXTENDIDOS

En éste apartado y de forma más detallada, se explicarán los casos de uso anteriormente citados. Para mejorar la identificación de los roles en las cabeceras de las siguientes tablas, se ha asignado un Alias a cada uno de los actores.

- SUPREMO → Supr
- SUPREMO_ADMIN → Admin
- SUPREMO_ADMIN_DOCENTE → Doc
- SUPREMO_ADMIN_ESTUDIANTE → Est
- USUARIO ANÓNIMO → Anon

Acceso a la información del sistema: Landing Page y documentación de la API

Nombre: Acceso a la información del sistema AdESMuS. Página principal informativa del sistema, y acceso a la información de la API.

Descripción: Todos los roles establecidos en el sistema AdESMuS tienen acceso a la información del sistema, ya sea la página principal del proyecto o la información de la API.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Acceso a la información: Landing Page y API	X	X	X	X	X

Tabla 8 - Anexo I, Casos de uso extendidos, Acceso a la información del sistema

Precondiciones: Tener acceso al recurso que permite obtener la información.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) Se realiza una solicitud al sistema para obtener el recurso.
2) El sistema devuelve la información solicitada.

Postcondiciones: Se ha obtenido la página principal de presentación del sistema AdESMuS o la información de la API.

Registro del usuario Supremo

Nombre: Registro de usuario con rol de Supremo.

Descripción: El primero de los usuarios que se introduce en el sistema será un usuario con rol de Supremo. Para realizar ésta operación no será necesario un Token. Una vez registrado el usuario, el método que permite el registro de éste tipo de usuarios queda deshabilitado (comentado por código).

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Registro usuario inicial Supremo					X

Tabla 9 - Anexo I, Casos de uso extendidos, Registro de usuario con rol Supremo

Precondiciones: Tener acceso al recurso que permite realizar el registro de un usuario con rol de Supremo.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba los datos recibidos para el registro de un usuario Supremo y realiza las verificaciones pertinentes para determinar si el registro es válido o no.

2) El sistema retorna el resultado de la operación del registro.

Postcondiciones: Se ha registrado un usuario con rol de Supremo en el sistema.

Identificación de un usuario

Nombre: Identificación de un usuario.

Descripción: Permite a un usuario anónimo identificarse en el sistema para obtener un Token válido.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Identificación en el sistema					X

Tabla 10 - Anexo I, Casos de uso extendidos, Identificación de usuarios

Precondiciones: Tener acceso al recurso que permite realizar la identificación.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba los datos de identificación del usuario, y en función del estado del mismo, obtendrá su Token válido asignado o se le proporcionará uno nuevo.

2) El sistema le retorna al usuario el Token con el que podrá realizar las consultas en el sistema.

Postcondiciones: El usuario obtiene un Token válido para acceder al sistema.

Añadir elementos

Nombre: Añadir un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Descripción: Permite registrar en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Añadir Centro	X	X			
Añadir Dpto.	X	X			
Añadir Titulación	X	X			
Añadir Grupo	X	X			
Añadir Asignatura	X	X			
Añadir Matrícula	X	X			
Añadir Administrador	X	X			
Añadir Docente	X	X			
Añadir Estudiante	X	X			

Tabla 11 - Anexo I, Casos de uso extendidos, Añadir elementos

Precondiciones: Disponer de un Token válido.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, comprueba si dispone del rol necesario para realizar la operación en el sistema.
2) Se añade en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Postcondiciones: Se ha añadido en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Obtener uno o varios elementos

Nombre: Obtener uno o todos los Centros, CentroTitulaciones, Departamentos, CentroDepartamentos, Titulaciones, Grupos, GrupoAsignaturas, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, DocenteGrupoasignaturas o EstudianteGrupoasignaturas.

Descripción: Permite obtener del sistema uno o todos los Centros, Titulaciones asociadas a un Centro, Departamentos, Departamentos asociados a un Centro, Titulaciones, Grupos, Asignaturas asociadas a un Grupo, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a un Docente o Grupoasignaturas asociadas a un Estudiante.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Obtener uno o todos los Centros	X	X	X	X	
Obtener una o todas las Titulaciones asociadas a un Centro	X	X	X	X	
Obtener uno o todos los Departamentos	X	X	X	X	
Obtener uno o todos los Departamentos asociados a un Centro	X	X	X	X	
Obtener una o todas las Titulaciones	X	X	X	X	
Obtener uno o todos los Grupos	X	X	X	X	
Obtener una o todas las Asignaturas asociadas a un Grupo	X	X	X	X	
Obtener una o todas las Asignaturas	X	X	X	X	
Obtener una o todas las Matrículas	X	X			
Obtener uno o todos los Administradores	X	X			
Obtener uno o todos los Docentes	X	X	X		
Obtener uno o todos los Estudiantes	X	X	X	X	
Obtener una o todas las Grupoasignaturas asociadas a un Docente	X	X	X		
Obtener una o todas las Grupoasignaturas asociadas a un Estudiante	X	X		X	

Tabla 12 - Anexo I, Casos de uso extendidos, Obtener elementos

Precondiciones: Disponer de un Token válido.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, comprueba si dispone del rol necesario para realizar la operación en el sistema.

2) Se obtiene del sistema uno o todos los Centros, Titulaciones asociadas a un Centro, Departamentos, Departamentos asociados a un Centro, Titulaciones, Grupos, Asignaturas asociadas a un Grupo, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a un Docente o Grupoasignaturas asociadas a un Estudiante.

Postcondiciones: Se ha obtenido del sistema la información de uno o todos los Centros, Titulaciones asociadas a un Centro, Departamentos, Departamentos asociados a un Centro, Titulaciones, Grupos, Asignaturas asociadas a un Grupo, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a un Docente o Grupoasignaturas asociadas a un Estudiante.

Actualizar elementos

Nombre: Actualizar un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Descripción: Permite actualizar en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Actualizar Centro	X	X			
Actualizar Departamento	X	X			
Actualizar Titulación	X	X			
Actualizar Grupo	X	X			
Actualizar Asignatura	X	X			
Actualizar Matrícula					
Actualizar Administrador	X	X			
Actualizar Docente	X	X	X		
Actualizar Estudiante	X	X		X	

Tabla 13 - Anexo I, Casos de uso extendidos, Actualizar elementos

Precondiciones: Disponer de un Token válido.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, comprueba si dispone del rol necesario para realizar la operación en el sistema.
2) Se actualiza en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Postcondiciones: Se ha actualizado en el sistema un Centro, Departamento, Titulación, Grupo, Asignatura, Matrícula, Administrador, Docente o Estudiante.

Asociar elementos

Nombre: Asociar un CentroTitulación, CentroDepartamento, GrupoAsignatura, DocenteGrupoasignatura, EstudianteGrupoasignatura.

Descripción: Permite asociar una Titulación a un Centro, un Departamento a un Centro, una Asignatura a un Grupo, un Grupoasignatura a un Docente o un Grupoasignatura a un Estudiante.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Asociar Titulación a un Centro	X	X			
Asociar Departamento a un Centro	X	X			
Asociar Asignatura a un Grupo	X	X			
Asociar Grupoasginatura a un Docente	X	X	X		
Asociar Grupoasignatura a un Estudiante	X	X		X	

Tabla 14 - Anexo I, Casos de uso extendidos, Asociar elementos

Precondiciones: Disponer de un Token válido.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, comprueba si dispone del rol necesario para realizar la operación en el sistema.

2) Se asocia en el sistema una Titulación a un Centro, un Departamento a un Centro, una Asignatura a un Grupo, un Grupoasignatura a un Docente o un Grupoasignatura a un Estudiante.

Postcondiciones: Se ha realizado la asociación en el sistema de una Titulación a un Centro, un Departamento a un Centro, una Asignatura a un Grupo, un Grupoasignatura a un Docente o un Grupoasignatura a un Estudiante.

Eliminar uno o varios elementos

Nombre: Eliminar Centro, CentroTitulaciones, Departamento, CentroDepartamentos, Titulación, Grupo, GrupoAsignaturas, Asignatura, Matrícula, Administrador, Docente, Estudiante, DocenteGrupoasignaturas o EstudianteGrupoasignaturas.

Descripción: Permite eliminar un Centro, una o todas las Titulaciones asociada a un Centro, un Departamento, uno o todos los Departamentos asociados a un Centro, una Titulación, un Grupo, una o todas las Asignaturas asociadas a un Grupo, una Matrícula, un Administrador, un Docente, un Estudiante, una o todas las Grupoasignaturas asociadas a un Docente o una o todas las Grupoasignaturas asociadas a un Estudiante.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Eliminar un Centro	X	X			
Eliminar Titulación/es asociadas a un Centro	X	X			
Eliminar un Departamento	X	X			
Eliminar Departamento/s asociados a un Centro	X	X			
Eliminar una Titulación	X	X			
Eliminar un Grupo	X	X			

Eliminar Asignatura/s asociadas a un Grupo	X	X			
Eliminar una Asignatura	X	X			
Eliminar una Matrícula	X	X			
Eliminar un Administrador	X				
Eliminar un Docente	X	X			
Eliminar un Estudiante	X	X			
Eliminar Grupoasignatura/s asociadas a un Docente	X	X	X		
Eliminar Grupoasignatura/s asociadas a un Estudiante	X	X		X	

Tabla 15 - Anexo I, Casos de uso extendidos, Eliminar elementos

Precondiciones: Disponer de un Token válido.

Requisitos no funcionales: No hay

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, comprueba si dispone del rol necesario para realizar la operación en el sistema.

2) Se eliminan del sistema un Centro, una o todas las Titulaciones asociada a un Centro, un Departamento, uno o todos los Departamentos asociados a un Centro, una Titulación, un Grupo, una o todas las Asignaturas asociadas a un Grupo, una Matrícula, un Administrador, un Docente, un Estudiante, una o todas las Grupoasignaturas asociadas a un Docente o una o todas las Grupoasignaturas asociadas a un Estudiante.

Postcondiciones: Se ha eliminado del sistema un Centro, una o todas las Titulaciones asociada a un Centro, un Departamento, uno o todos los Departamentos asociados a un Centro, una Titulación, un Grupo, una o todas las Asignaturas asociadas a un Grupo, una Matrícula, un Administrador, un Docente, un Estudiante, una o todas las Grupoasignaturas asociadas a un Docente o una o todas las Grupoasignaturas asociadas a un Estudiante.

Eliminar todos los elementos

Nombre: Eliminar todos los registros de Centros, CentroTitulaciones, Departamentos, CentroDepartamentos, Titulaciones, Grupos, GrupoAsignaturas, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, DocenteGrupoasignaturas o EstudianteGrupoasignaturas.

Descripción: Permite eliminar todos los registros de Centros, Titulaciones asociadas a los Centros, Departamentos, Departamentos asociados a los Centros, Titulaciones, Grupos, Asignaturas asociadas a los Grupos, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a Docentes o Grupoasignaturas asociadas a Estudiantes.

Actores:

	Actores				
Acción	**Supr**	**Admin**	**Doc**	**Est**	**Anon**
Eliminar todos los Centros	X				
Eliminar todos los CentroTitulación	X				
Eliminar todos los Departamentos	X				
Eliminar todos los CentroDepartamentos	X				
Eliminar todas las Titulaciones	X				
Eliminar todos los Grupos	X				
Eliminar todos los GrupoAsignaturas	X				
Eliminar todas las Asignaturas	X				
Eliminar todas las Matrículas	X				
Eliminar todos los Administradores	X				
Eliminar todos los Docentes	X				
Eliminar todos los Estudiantes	X				
Eliminar todos los DocenteGrupoasignatura	X				
Eliminar todos los EstudianteGrupoasignatura	X				

Tabla 16 - Anexo I, Casos de uso extendidos, Vaciar tablas

Precondiciones: Disponer de un Token con rol Supremo.

Requisitos no funcionales: No hay.

Flujo de eventos:

1) El sistema comprueba que el Token es válido, y además, si dispone de un rol de Supremo.

2) Se eliminan del sistema todos los registros de Centros, Titulaciones asociadas a los Centros, Departamentos, Departamentos asociados a los Centros, Titulaciones, Grupos, Asignaturas asociadas a los Grupos, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a Docentes o Grupoasignaturas asociadas a Estudiantes.

Postcondiciones: Se han eliminado del sistema todos los registros de Centros, Titulaciones asociadas a los Centros, Departamentos, Departamentos asociados a los Centros, Titulaciones, Grupos, Asignaturas asociadas a los Grupos, Asignaturas, Matrículas, Administradores, Docentes, Estudiantes, Grupoasignaturas asociadas a Docentes o Grupoasignaturas asociadas a Estudiantes.

ANEXO II. DIAGRAMAS DE SECUENCIA

AdESMuS puede ser ejecutado desde diferentes ámbitos. Al no disponer de una interfaz gráfica para realizar las peticiones, se ha definido en los diagramas de secuencia una interfaz virtual que engloba el conjunto de dispositivos o tecnologías que pueden hacer uso de la API. La API puede ser consumida desde diferentes medios: una aplicación de escritorio, una web, un sistema Moddle, etc.

Acceso a la información del sistema o Landing Page

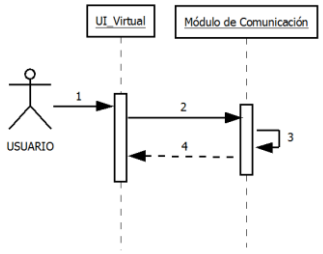

Ilustración 69 - Anexo II, Diagramas de secuencia, documentación del sistema

1) El usuario desde el medio en el que se encuentre solicita obtener la información general de la API.
2) El medio realiza una solicitud al recurso de la API que permite obtener la información general de AdESMuS (Landing Page).

   Mod_comunicación → router_principal.js → Recurso: GET '/'
3) El recurso o *Módulo de comunicación* recibe la petición, accede a la zona interna de vistas y obtiene el contenido de la vista solicitada.
4) El *Módulo de comunicación* genera la respuesta y se le envía al usuario.

Registro de un usuario Supremo

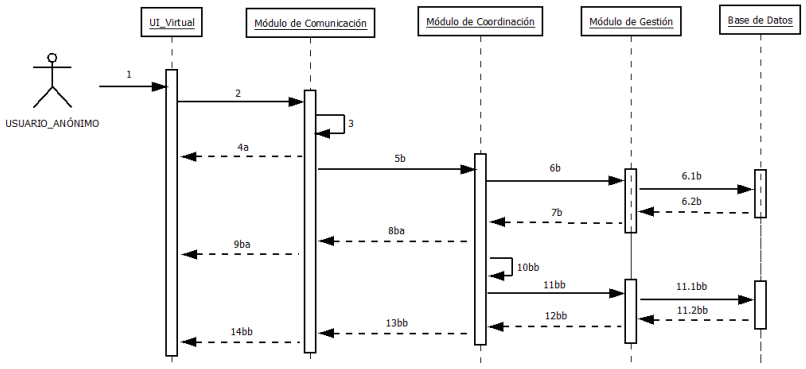

Ilustración 70- Anexo II, Diagramas de secuencia, registro de un usuario Supremo

1) El usuario desde el medio en el que se encuentre solicita el registro de un usuario con rol de Supremo.

2) El medio realiza una solicitud al recurso de la API que permite realizar el registro de un usuario con rol de Supremo. Para ello, le envía los datos del registro del usuario. No es necesario un Token para realizar ésta operación, y solamente se podrá registrar en el sistema un usuario con dicho rol.

   Mod_comunicación → router_auth.js → POST '/auth/register'.

3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos para poder continuar con el proceso de registro del usuario con rol de Supremo.

   [Si no se han recibido correctamente todos los datos]

   4a) El sistema devuelve una trama informativa indicando que los datos no son correctos.

   [Si se han recibido correctamente los datos]

   5b) El *Módulo de comunicación* delega al *Módulo de coordinación* el proceso del registro del usuario.

   Mod_coordinación → functions_usuario.js → registerUser(...)

   6b) El *Módulo de coordinación* solicita al *Módulo de gestión* un usuario con las características del usuario solicitante.

   Mod_gestión → usuario.js → findByNickUsuario(...)

   6.1b) El *Módulo de gestión* accede a la base de datos.

   6.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

   7b) El *Módulo de gestión* devuelve el resultado de la búsqueda del usuario.

      [Si el usuario con el Nick buscado ya existe]

      8ba) El *Módulo de coordinación* informa al *Módulo de comunicación* que ha existido un error.

      9ba) El *Módulo de comunicación* obtiene el error generado, genera la trama de datos y se la envía al usuario que ha realizado la petición.

      [Si el usuario con el Nick indicado no existe]

      10bb) El *Módulo de coordinación* genera el objeto con los datos del usuario.

      11bb) El *Módulo de coordinación* delega el registro al *Módulo de gestión* indicándole el objeto con los datos del usuario que se quiere registrar.

Mod_gestión → usuario.js → addUser(…)

11.1bb) El *Módulo de gestión* accede a la base de datos.

11.2bb) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

12bb) El *Módulo de gestión* notifica al *Módulo de coordinación* el resultado del registro del usuario.

13bb) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado del registro del usuario.

14bb) El *Módulo de comunicación* genera la trama de datos con el resultado del registro del usuario.

Identificación de un usuario

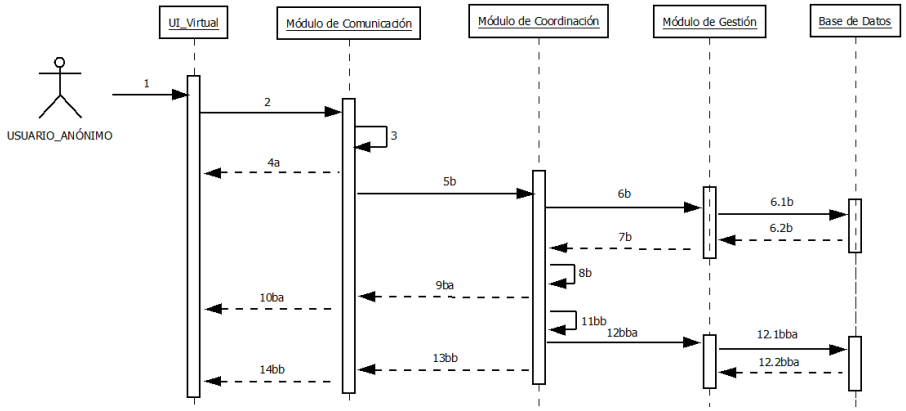

Ilustración 71- Anexo II, Diagramas de secuencia, identificación de un usuario

1) El usuario desde el medio en el que se encuentre aporta sus datos de identificación.
2) El medio realiza una solicitud al recurso de la API que permite realizar la identificación de un usuario en el sistema. Para ello, aporta los datos de identificación proporcionados por el usuario.

   Mod_comunicación → router_auth.js → POST '/auth/login'.
3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos.

   [Si no se han recibido correctamente los datos]

   4a) El *Módulo de comunicación* devuelve una trama informativa indicando que los datos no son correctos.

   [Si se han recibido correctamente los datos]

5b) El *Módulo de comunicación* delega al *Módulo de coordinación* el proceso de identificación.

Mod_coordinación → functions_usuario.js → loginUser(…)

6b) El *Módulo de coordinación solicita* al *Módulo de gestión* la búsqueda de un usuario que coincida con los datos del usuario que quiere realizar la identificación.

Mod_gestión → usuario.js → findByNickUsuario(…)

6.1b) El *Módulo de gestión* accede a la base de datos.

6.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

7b) El *Módulo de gestión* retorna el resultado de la búsqueda del usuario solicitado.

8b) Se comprueba el usuario obtenido y los datos de identificación que ha proporcionado el usuario.

[Si el usuario no existe || los datos de su identificación son incorrectos]

9ba) El *Módulo de coordinación* comunica al *Módulo de comunicación* que ha existido un error en el proceso y no se va a continuar.

10ba) El *Módulo de comunicación* recibe el error, genera la trama con la información y se la envía al usuario.

[Si el usuario existe && sus datos de identificación son correctos]

11bb) El *Módulo de coordinación* comprueba si el usuario tiene un Token asignado, y si éste es válido.

Helpers → token.js → verifyToken(…)

[Si el usuario no dispone de un Token, o el Token del que dispone no es válido]

12bba) Se genera un Token nuevo, válido durante un tiempo determinado y se realiza una petición al *Módulo de gestión* para registrar al usuario el nuevo Token generado.

Mod_gestión → usuario.js → updateUser(…)

12.1bba) El *Módulo de gestión* accede a la base de datos.

12.2bba) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

13bb) El *Módulo de coordinación* remite al *Módulo de comunicación* el Token del usuario.

14bb) El *Módulo de coordinación* genera la trama incluyendo el Token y se la envía al usuario.

Añadir, asociar y actualizar elementos

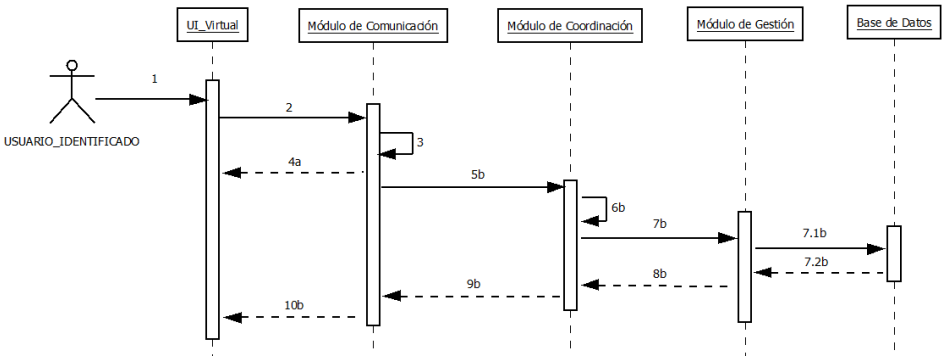

Ilustración 72- Anexo II, Diagr. de secuencia, añadir, asociar y actualizar elementos

Añadir elementos

1) El usuario desde el medio en el que se encuentre solicita añadir un elemento.
2) El medio realiza una solicitud al recurso de la API que permite añadir un elemento al sistema, y para ello, aporta en la solicitud, los datos del elemento que desea añadir junto a un Token válido.

   Mod_comunicación → router_centros.js → POST '/api/center'.
3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos, existe un Token válido y el usuario dispone de permisos para realizar operaciones sobre el recurso solicitado.

   [Si no se han recibido correctamente los datos || Token es inválido || sin permisos para ejecutar la acción]

   4a) El *Módulo de comunicación* devuelve una trama informativa indicando que se ha producido un error en la solicitud.

   [Si se han recibido correctamente los datos && Token es válido && el usuario tiene permisos de ejecución]

   5b) El *Módulo de comunicación* delega el registro al *Módulo de coordinación* y le envía los datos recibidos en la solicitud.

   Mod_coordinación → functions_centro.js → addCentro(…)

6b) El *Módulo de coordinación* genera el objeto con los datos que se desean registrar.

7b) El *Módulo de coordinación* delega el registro al *Módulo de gestión* indicándole el objeto que desea añadir a la base de datos.

Mod_gestión → centro.js → addCentro(…)

7.1b) El *Módulo de gestión* accede a la base de datos.

7.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

8b) El *Módulo de gestión* notifica el resultado del registro al *Módulo de coordinación*.

9b) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado del registro.

10b) El *Módulo de comunicación* genera la trama con el resultado de la operación.

Asociar elementos

1) El usuario desde el medio en el que se encuentre solicita la asociación de elementos.
2) El medio realiza una solicitud al recurso de la API que permite asociar elementos, y para ello, aporta en la solicitud los identificadores de los elementos, y un Token válido.

   Mod_com → router_centros.js → POST *'/center/(:idCen)/department/(:idDpto)'*
3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos, existe un Token válido y el usuario dispone de permisos para realizar operaciones sobre el recurso solicitado.

   [Si no se han recibido correctamente los datos || Token es inválido || sin permisos para ejecutar la acción]

   4a) El *Módulo de comunicación* devuelve una trama informativa indicando que se ha producido un error en la solicitud.

   [Si se han recibido correctamente los datos && Token es válido && el usuario tiene permisos de ejecución]

   5b) El *Módulo de comunicación* delega la operación de asociación al *Módulo de coordinación*.

   Mod_coord → functions_centro.js → associateDptoToCentro(…)

6b) El *Módulo de coordinación* genera el objeto con los elementos que se desean asociar.

7b) El *Módulo de coordinación* delega la operación de asociación al *Módulo de gestión* indicándole el objeto que desea añadir a la base de datos.

Mod_gestión → centro.js → associateDptoToCentro(…)

7.1b) El *Módulo de gestión* accede a la base de datos.

7.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

8b) El *Módulo de gestión* notifica el resultado de la asociación al *Módulo de coordinación*.

9b) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado obtenido.

10b) El *Módulo de comunicación* genera la trama con el resultado de la operación.

Actualizar elementos

1) El usuario desde el medio en el que se encuentre solicita la actualización de un elemento.

4) El medio realiza una solicitud al recurso de la API que permite actualizar un elemento determinado, y para ello, aporta en la solicitud el identificador del elemento que desea modificar, junto a la información modificada y un Token válido.

   Mod_comunicación → router_centros.js → PUT *'/api/center/(:idCentro)'*.

2) El *Módulo de comunicación* comprueba que los datos recibidos son correctos, existe un Token válido y el usuario dispone de permisos para realizar operaciones sobre el recurso solicitado.

   [Si no se han recibido correctamente los datos || Token es inválido || sin permisos para ejecutar la acción]

   4a) El *Módulo de comunicación* devuelve una trama informativa indicando que se ha producido un error en la solicitud.

   [Si se han recibido correctamente los datos && Token es válido && el usuario tiene permisos de ejecución]

   5b) El *Módulo de comunicación* delega la operación de actualización del elemento al *Módulo de coordinación*.

Mod_coordinación → functions_centro.js → updateCentro(…)

6b) El *Módulo de coordinación* genera el objeto con los parámetros indicados a modificar por el usuario.

7b) El *Módulo de coordinación* delega la actualización al *Módulo de gestión* indicándole el objeto que desea modificar de la base de datos.

Mod_gestión → centro.js → updateCentro(…)

7.1b) El *Módulo de gestión* accede a la base de datos.

7.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

8b) El *Módulo de gestión* notifica el resultado de la actualización al *Módulo de coordinación.*

9b) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado obtenido.

10b) El *Módulo de comunicación* genera la trama de datos con el resultado de la operación.

Obtener y eliminar elementos

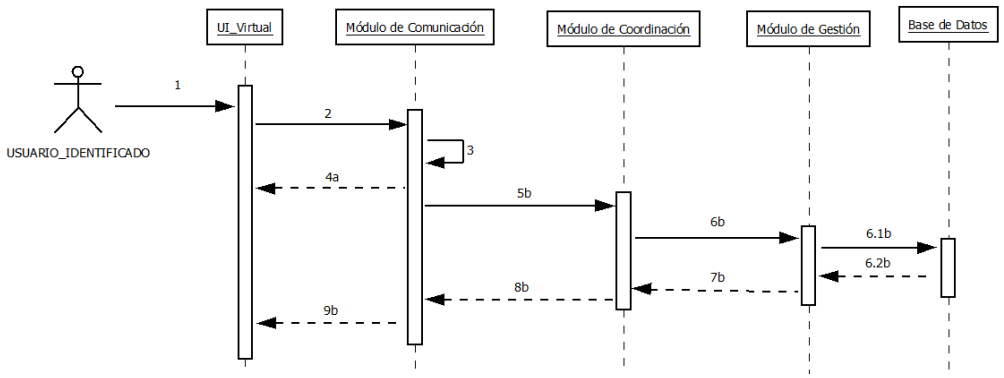

Ilustración 73- Anexo II, Diagramas de secuencia, obtener y eliminar elementos

Obtener uno o varios elementos

1) El usuario desde el medio en el que se encuentre solicita obtener una serie de elementos.

2) El medio realiza una solicitud al recurso de la API que permite obtener uno o varios elementos, y para ello, el usuario aporta un Token válido.

   Mod_comunicación → router_centros.js → GET '/api/center/(:idCentro)'

3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos, existe un Token válido y el usuario dispone de permisos para realizar operaciones sobre el recurso solicitado.

   [Si no se han recibido correctamente los datos || Token es inválido || sin permisos para ejecutar la acción]

   4a) El *Módulo de comunicación* devuelve una trama informativa indicando que se ha producido un error en la solicitud.

   [Si se han recibido correctamente los datos && Token es válido && el usuario tiene permisos de ejecución]

   5b) El *Módulo de comunicación* delega la operación de obtención de los datos al *Módulo de coordinación* enviándole la información de la solicitud.

   Mod_coordinación → functions_centro.js → findById(…)

   6b) El *Módulo de coordinación* realiza una petición al *Módulo de gestión* indicándole que quiere obtener una serie de datos.

   Mod_gestión → centro.js → findById(…)

   6.1b) El *Módulo de gestión* accede a la base de datos.

   6.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

   7b) El *Módulo de gestión* notifica el resultado de los datos obtenidos al *Módulo de coordinación*.

   8b) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado obtenido.

   9b) El *Módulo de comunicación* genera la trama de datos con el resultado de la operación.

Eliminar uno, varios o todos los elementos

1) El usuario desde el medio en el que se encuentre solicita la eliminación de un elemento.
2) El medio realiza una solicitud al recurso de la API que permite la eliminación de los elementos indicados, y para ello, aporta en la solicitud los datos necesarios para realizar dicha acción.

   Mod_comunicación → router_centros.js → DELETE '/api/center/(:idCentro)'

3) El *Módulo de comunicación* comprueba que los datos recibidos son correctos, existe un Token válido y el usuario dispone de permisos para realizar operaciones sobre el recurso solicitado.

[Si no se han recibido correctamente los datos || Token es inválido || sin permisos para ejecutar la acción]

4a) El *Módulo de comunicación* devuelve una trama informativa indicando que se ha producido un error en la solicitud.

[Si se han recibido correctamente los datos && Token es válido && el usuario tiene permisos de ejecución]

5b) El *Módulo de comunicación* delega la operación de eliminación del elemento al *Módulo de coordinación*.

Mod_coordinación → functions_centro.js → deleteCentro(…)

6b) El *Módulo de coordinación* delega la operación de eliminación al *Módulo de gestión* indicándole qué elemento se desea eliminar.

Mod_gestión → centro.js → deleteCentro(…)

6.1b) El *Módulo de gestión* accede a la base de datos.

6.2b) El *Módulo de gestión* obtiene de la base de datos el resultado de la operación realizada.

8b) El *Módulo de gestión* notifica el resultado de la eliminación al *Módulo de coordinación*.

9b) El *Módulo de coordinación* notifica al *Módulo de comunicación* el resultado obtenido.

10b) El *Módulo de comunicación* genera la trama con el resultado de la operación.