UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

GRADO EN INGENIERÍA INFORMÁTICA

Aplicación para la Gestión Docente del Departamento de Informática y Sistemas

Tanausú Jorge Suárez Martín

Tutores

Agustín Javier Salgado de la Nuez

Nelson Manuel Monzón López

Las Palmas de Gran Canaria, enero de 2020

Índice

Agradecimientos

En este apartado me gustaría agradecer todo el trabajo realizado para poder conseguir que el proyecto que aquí se presenta como un trabajo final de grado se pudiera cumplir.

En primer lugar, agradecer a mis tutores ya que, sin su apoyo incondicional y su enorme trabajo, tanto días como noches, han realizado un enorme esfuerzo para que este proyecto saliera adelante y agradecer, también, sus ideas y enseñanzas que me han ayudado afrontar un proyecto con semejante magnitud. Por otro lado, también agradecer de igual forma al profesor Octavio Mayor Gonzáles por su enseñanzas y consejos para formar la base de datos en la que se apoya todo el proyecto.

Por último, un especial agradecimiento a todos aquellos profesores que me han acompañado por algunas de las etapas de mi enseñanza y que me han formado como profesional en mi titulación dentro de la escuela de ingeniería informática y que a partir de ahora me tocará seguir aprendiendo de forma autónoma.

1. Introducción

Anualmente cada departamento de la Universidad de Las Palmas de Gran Canaria (ULPGC) debe confeccionar su Plan de Ordenación Docente (POD). El POD representa la asignación del profesorado a las asignaturas encomendadas a dicho departamento.

En la actualidad, el Departamento de Informática y Sistemas (DIS) de la ULPGC cuenta con una aplicación web para la confección del POD. Dicha herramienta es funcional, pero tiene algunas limitaciones: principalmente en cuanto a las comprobaciones, importación y exportación de los datos. Estas operaciones deben realizarse de forma manual por parte del administrador. La aplicación debe cumplir con la normativa vigente. Si se produce algún cambio normativo esto obliga a reflejarlo en la aplicación. Por tanto, cualquier aplicación que gestione el POD debe ser versátil y poco restrictiva.

En el contexto del Trabajo Fin de Grado se ha desarrollado una aplicación que automatiza el proceso de confección del POD por parte del profesorado, así como la automatización de los procesos de importación y exportación de la demanda docente; disminuyendo las horas de trabajo de los responsables de esta tarea. La nueva aplicación presenta una serie de ideas nuevas, un diseño mejorado (responsive), la incorporación de filtros y buscadores para simplificar la cantidad de información mostrada.

La aplicación tiene una arquitectura cliente-servidor. Ambas partes, trabajan conjuntamente para ofrecer al cliente un servicio completo y facilidades en la gestión del POD. Este sistema trabaja con un modelo REST que permite cambiar o sustituir algunas de las partes o añadir nuevos clientes, en otras palabras, nos permite ser más escalables.

El desarrollo de todas las actividades que comprenden este TFG asciende a un total de 300 horas: incluyendo documentación, desarrollo de la aplicación y presentación de la defensa. Algunas de las actividades a destacar han sido la creación de producto backlog, dónde se ha realizado el análisis de la situación de la actual aplicación para valorar las necesidades y sus requerimientos. Sprint Zero, donde se han configurado todas la herramientas y tecnologías que se han utilizado durante el proyecto. El desarrollo de la funcionalidad de la aplicación se ha dividido en tres entregables, o incrementos, que serán explicados con mayor detalle más adelante.

El apartado desarrollo, de esta memoria, está dividida en tres bloques: análisis, diseño e implementación. La parte de análisis será donde se describirá la plataforma actual. Este apartado describirá los casos de usos, el modelo entidad-relación de la base de datos y el formato que debe tener los archivos de importación y exportación. El apartado de diseño describe los aspectos tanto del cliente y servidor, de cómo se comunican dichas entidades, así como aspecto de diseño de la interfaz de usuario, tablas de base de datos o de un sistema REST. Por último, el apartado de implementación describe aquellos elementos más importantes de las que cuenta la plataforma para la gestión y organización de todo el sistema, así como las facilidades que se dan a los usuarios.

Al final de este documento se presenta el presupuesto que correspondería con el desarrollo de la aplicación, así como un conjunto de anexos para que un usuario pueda ejecutar tanto cliente, como servidor.

2. Contexto y motivación

En la actualidad existe una aplicación para la confección del POD del DIS. Ésta presenta una serie de inconvenientes, como es, que dicha aplicación es gestionada por una única persona dejando la responsabilidad total del trabajo sobre ella. Además, la exportación del POD resultante requiere de una manipulación previa a su incorporación a las bases de datos centrales de la ULPGC.

La evolución de las distintas tecnologías web puede facilitar la incorporación de nuevas funcionalidades al sistema existente que aporte más comodidad al mantenimiento y comprensión del código. Es por ello, que se plantea esta nueva aplicación con una tecnología ya madura y en continua evolución, como es el caso de Angular y Python, que va más allá de la actual, desarrollada principalmente en HTML, CSS y Javascript.

El nuevo prototipo de aplicación que aquí se plantea pretende ser una evolución de la ya existente donde mejore los procesos de automatización de las importaciones y exportaciones de los datos. Además, de una nueva interfaz gráfica más intuitiva y de fácil uso, como las mejoras de las búsquedas o la facilidad en la integración de este sistema en otro, debido a la división entre la vista y el modelo, desarrollado por un sistema REST.

3. Estado del arte

Actualmente, la aplicación encargada de la confección del POD del DIS de la ULPGC, aunque funcional ha quedado algo obsoleta. Dicha aplicación deja en manos de los usuarios toda la responsabilidad de gestión, organización o cumplimiento de las normativas. En Imagen 1. POD - demanda docente, se puede visualizar un ejemplo del actual aspecto que tiene la aplicación actual donde en una única tabla se representan todos los datos de cada grupo de cada asignatura. En la columna que muestra las *horas cubiertas* se representa con tres colores diferentes si dichas horas están cubiertas (verdes), están sobrepasadas (amarillo) o no se han cubierto aún (rojo).

Titulación	Área	Asignatura	Curso	Tipo	Cuatr.	Grupo	Tipo Docencia	Horas Grupo	¿Horas Cubiertas?
Doble Grado en Ingeniería Informática y ADE	C	Trabajo Fin de Grado	6	Ob	1C	146	Teoría	18	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	01	Teoría	30	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	41	Prácticas en Laboratorio	30	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	42	Prácticas en Laboratorio	30	+0
Grado en Ingeniería Eléctrica	C	Informática y Programación	1	BR	1C	01	Teoría	30	+0
Grado en Ingeniería Eléctrica	C	Informática y Programación	1	BR	1C	41	Prácticas en Laboratorio	30	+0
Grado en Ingeniería Eléctrica	C	Informática y Programación	1	BR	1C	42	Prácticas en Laboratorio	30	+0
Grado en ingeniería Electrónica Industrial y Automática	C	Informática y Programación	1	BR	1C	01	Teoría	30	+0
Grado en ingeniería Electrónica Industrial y Automática	C	Informática y Programación	1	BR	1C	41	Prácticas en Laboratorio	30	+0
Grado en ingeniería Electrónica Industrial y Automática	C	Informática y Programación	1	BR	1C	42	Prácticas en Laboratorio	30	+0
Grado en ingeniería Electrónica Industrial y Automática	C	Informática y Programación	1	BR	1C	43	Prácticas en Laboratorio	30	+0
Grado en ingeniería Electrónica Industrial y Automática	C	Informática y Programación	1	BR	1C	44	Prácticas en Laboratorio	30	+0
Grado en Ingeniería en Tecnología Naval	C	Informática y Programación	1	BR	2C	01	Teoría	30	+0
Grado en Ingeniería en Tecnología Naval	C	Informática y Programación	1	BR	2C	41	Prácticas en Laboratorio	30	+0
Grado en Ingeniería en Tecnología Naval	C	Informática y Programación	1	BR	2C	42	Prácticas en Laboratorio	30	+0

Imagen 1. POD - demanda docente

Una de las primeras ideas que surgiere dicha aplicación es la necesidad de incorporar un buscador (para facilitar la búsqueda de los grupos), o bien, filtros para seleccionar algunos grupos con ciertas restricciones impuestas por el usuario. En algunas ocasiones supone un inconveniente para los usuarios que deseen localizar un grupo en específico o filtrar por su área de conocimiento.

En la Tabla 1. Comparativa entre APP gestión docente y POD, se puede observar una comparativa entre el prototipo desarrollado en este TFG ("*App gestión docente*") y la actual aplicación que está en funcionamiento ("*POD*").

Tabla 1. Comparativa entre APP gestión docente y POD

	App gestión docente	**POD**
Diseño	Angular 7	HTML
Diseño de tablas	Contenido fraccionado (scroll)	Contenido en una única tabla (bloque)
Cabecera de las tablas	Siempre visible	Estático
Colores que identifica el estado de las horas cubiertas (rojo, verde, amarillo)	✓	✓
Navegación	Menú con acceso directo al contenido	Contenido accesible a través de una página principal
Buscador para las asignaturas	✓	-
Buscador para los profesores	✓	-
Filtros de búsqueda	✓	-
Las filas de las tablas aparecen ordenadas	✓	-
Importación de Carga docente desde un archivo (ej: Excel)	✓	-
Importación de PDA desde un archivo (ej: Excel)	✓	-
Importación de lista de profesores desde un archivo (ej: Excel)	✓	-
Indicador de horas seleccionadas	✓	-

4. Objetivos

Entre los objetivos generales que se busca en este Trabajo Fin de Grado es que el alumno sea capaz de forma individualizada y con la ayuda de sus tutores, presentar y defender públicamente un proyecto de carácter técnico. De esta forma, se completará los estudios de Grado de Ingeniería Informática para enfrentarse a un ámbito laboral con todas y cada una de las competencias adquiridas a lo largo de la titulación.

Como objetivos más específicos del desarrollo del TFG encontraríamos:

- La aplicación de metodologías de ingeniería de software para documentar el proceso de desarrollo del proyecto, especificando el análisis, diseño y programación e implementación de un sistema.
- La implementación de un servicio de gestión que permita la recopilación de la selección de la carga docente del profesorado de un departamento de la ULPGC.
- El diseño e implementación de una interfaz de usuario que permita relacionar los conjuntos de datos del profesorado y las materias de la oferta disponible de los distintos centros.
- La implementación de un sistema de interacción del usuario con el graficado que le permita visualizar la selección realizada por el profesorado, así como el horario resultante, verificación y prueba de sistema sobre datos reales. La implementación de una herramienta que sea fácilmente modificable y ampliable en el futuro.

5. Competencias

Durante el desarrollo del TFG se han adquirido una serie de competencias, además de las competencias generales, las de la ULPGC, las del título y los objetivos descrito en el apartado 4. En el siguiente punto se definen las competencias relacionadas con el bloque de la intensificación de "*tecnologías de la información*".

TI01

"Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones" [14.12].

En este caso se trata de comprender el tipo de necesidades académicas necesita el profesorado para realizar la selección de la docencia, así como las necesidades del administrador para tener un sistema que automatice la gestión de los datos.

TI02.

"Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados" *[14.12].*

El software desarrollado ha cumplido con la planificación y estimaciones fijadas.

TI03

"Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas". [14.12]

En referencia a esta competencia y el desarrollo de la aplicación pensada en el usuario, se ha propuesto una aplicación lo mayor intuitiva posible, así como la accesibilidad a cualquier funcionalidad con los posibles menos *click's*, además de tener en cuenta las necesidades del usuario, propuesta por algunas personas que utilizan habitualmente la plataforma, como son mis tutores.

TI05

"Capacidad para seleccionar, desplegar, integrar y gestionar sistemas de información que satisfagan las necesidades de la organización, con los criterios de coste y calidad identificados" [14.12]

Con anterioridad al desarrollo de la aplicación, se ha hecho un estudio (análisis) del sistema de información en el que está integrada la aplicación del POD anterior.

TI06

*"*Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil*." [14.12]*

Las competencias TI05 y TI06 cumplen en el caso la aplicación se lanza bajo las necesidades del profesorado para seleccionar las horas de docencia, visualizar las horas seleccionadas.

TFG01

"*Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas*". *[14.12]*

El desarrollo de la aplicación, su descripción en esta memoria y defensa del trabajo ante el tribunal comprendería la adquisición de esta competencia.

6. Recursos utilizados

En este apartado se describen los recursos que se han utilizado para el desarrollo de la aplicación cliente-servidor de este TFG.

Hardware y software

En el proceso de desarrollo de la aplicación se ha utilizado un portátil con un sistema operativo *Windows 10 64bit*. Las características técnicas del equipo son, 12 GB de memoria RAM, 256 GB de disco sólido y procesador intel core i7 de 7th generación. Estas prestaciones son suficientes para soportar dos entornos al mismo tiempo, donde los requerimientos son menores.

También se ha hecho pruebas en otros entornos de trabajo para llevar a cabo algunas tareas como ejecución del cliente y del servidor. En estas pruebas se ha empleado un sistema con Windows 10, 8 GB de RAM, intel core i5, disco duro de 1TB.

Entorno de desarrollo (IDE)

En el desarrollo de la interfaz gráfica se ha usado el IDE WebStrom, debido a que este entorno está desarrollado para proyectos similares en los cuales se utiliza Javascript.

En el desarrollo del servidor se ha utilizado otro IDE de la misma familia, Pycharm, éste entorno específico para Python es la mejor opción en desarrollo por su cantidad y calidad de atajos que proporciona al desarrollador.

Control de versiones

En el control de versiones se ha utilizado Git y GitHub que proporcionan una seguridad para el desarrollador de guardar el código de forma externa al equipo local, además de controles de versiones, pudiendo así retroceder a puntos donde el código funciona correctamente, agilizando así los trabajos.

Lenguajes de programación

En una parte fundamental del desarrollo se encuentran los lenguajes de programación que aquí se definen, del lado del cliente la principal tecnología es *Angular 7*, éste es un framework de desarrollo creado por Google para aplicaciones principalmente en *TypeScript*, de código abierto. Por otra parte, del lado del servidor se encuentra Python, un lenguaje de programación de propósito general, que facilita la interacción con la gestión de peticiones HTTP y las ordenes SQL para la gestión de la base de datos.

Framework

Por otra parte, los lenguajes anteriormente descritos se apoyan principalmente en dos framework de desarrollo en los que se encuentra Boostrap y Flask. Boostrap es una librería de código abierto para el diseño de sitio y aplicaciones web. Flask es un framework escrito en Python que permite la creación de aplicaciones web, aunque su propósito en este proyecto ha sido el de proveer un modelo de base de datos para la emigración a distintos gestores, (SQLite, MySQL, MaríaDB, PosgreSQL) y también usar las especificaciones de WSGI (Web Server Gateway Interface) que nos permite realizar peticiones HTTP de forma sencilla y rápida, perfecto para un sistema REST.

Postman

Postman es una herramienta software para verificar, monitorizar y simular las peticiones del funcionamiento de una API REST. En este caso, ha sido utilizado para testear las respuestas del backend desarrollado con Flask y comprobar que todas estas peticiones sean las esperadas.

7. Planificación

El proyecto está desarrollado en un total de 300 horas, aproximadamente, en las cuales se divide en distintas etapas como se muestra en la Tabla 2. Planificación del proyecto. La creación de la pila de producto (backlog) es la primera tarea, en ella, se realiza un análisis de la situación de la actual plataforma y de las limitaciones que presenta hoy en día. Además, en este punto se lleva a cabo una definición de los requisitos preliminares del usuario y sus necesidades.

Una vez hecho el análisis previo sobre la situación que se encuentra la actual plataforma es hora de definir, a priori, los requisitos que debe cumplir la nueva plataforma. Posteriormente se ha diseñado un modelo preliminar de la base de datos y de la interfaz de usuario. Se ha definido las necesidades software de la aplicación: instalación y configuración de extensiones y de paquetes de Python, Flask y Angular 7.

En la tercera parte se desarrolla, de forma incremental (3 entregables), la plataforma atendiendo a las necesidades identificadas en las fases anteriores.

Fases	*Tiempo estimado (horas)*	*Tiempo real (horas)*	*Actividades*
Creación de la pila de producto (backlog)	20	10	Análisis de la situación actual (10 h).
			Requisitos de usuario y priorización de necesidades de la APP (10 h)
Sprint Zero	40	50	Definición de requisitos del software. (10 h)
			Diseño de la base de datos. (15 h)
			Diseño de la interfaz de usuario. (15 h)
Fase de desarrollo de las distintas Entregas	200	320	La estimación del desarrollo en tres versiones que sucederán en fases de implementación de la iteración con la base de datos, acceso a de distintos usuarios en

			cada fase e implementación iterativa de la interfaz de usuario según las necesidades que se estimen. La estimación de puesta a producción es la siguiente: - Entrega 1 (80 h) - Entrega 2 (60 h) - Entrega 3 (60 h)
Documentación/ presentación	40	40	Documentación de la aplicación desarrollada. Edición de la memoria del proyecto realizado. Preparación de la defensa pública.

Tabla 2. Planificación del proyecto

Como se puede observar en la Tabla 2. Planificación del proyecto, podemos apreciar que las horas estimadas con las horas reales no coinciden, extendiéndose la duración del proyecto hasta las 420 horas, muy por encima de las 300 horas planificadas. La razón de este hecho es debido en gran parte a la poca experiencia en (1) diseño e implementación de la base de datos, (2) en un nuevo lenguaje y (3) el desconocimiento de la estructura interna del DIS.

8. Desarrollo

En esta etapa se ha desarrollado las distintas funcionalidades de la plataforma, tanto en el *backend* y como en el *frontend*. De forma incremental y según la planificación explicada en el apartado anterior, cada funcionalidad de la aplicación ha llevado la misma dinámica de trabajo, donde se identifican las fases de análisis, diseño e implementación.

8.1 Análisis

En el apartado de análisis se pondrá el foco sobre que las características de la actual aplicación y de las necesidades (funcionalidades) que debe de tener la nueva plataforma.

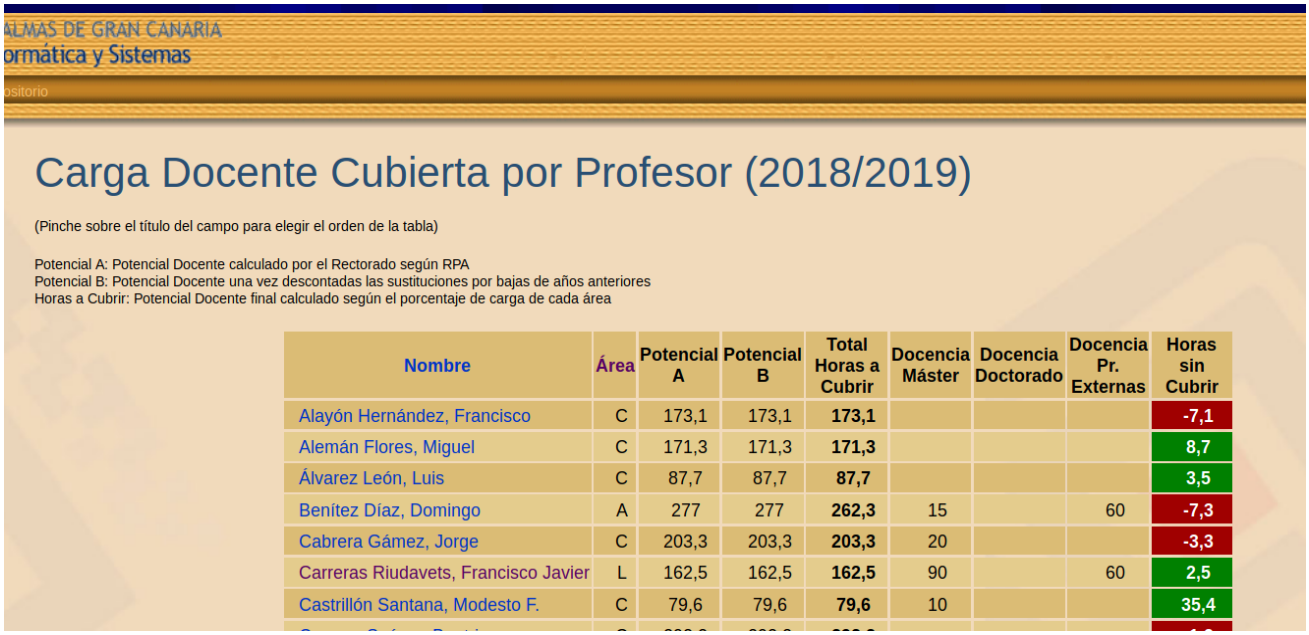
8.1.1 Plataforma actual

La aplicación del POD presenta una página principal con diferentes enlaces a los distintos contenidos de la herramienta. Uno de esos enlaces es la página que muestra la demanda docente del DIS. La información está tabulada y mediante un código de colores, mencionado en el apartado del estado del arte, se visualiza el número de horas cubiertas por cada asignatura. La Imagen 2. Demanda docente del POD presenta la información de los grupos que se le demanda al departamento, donde se indica la titulación, área de conocimiento, curso y tipo de la asignatura, el código y tipo del grupo (práctica de aula, teoría, práctica de laboratorio, etcétera). Para disponer de una mayor información acerca de qué profesores imparten dicha materia y el número de horas que se ha seleccionado, solo debemos picar sobre el nombre de la asignatura.

Titulación	Área	Asignatura	Curso	Tipo	Cuatr.	Grupo	Tipo Docencia	Horas Grupo	¿Horas Cubiertas?
Doble Grado en Ingeniería Informática y ADE	C	Trabajo Fin de Grado	6	Ob	1C	146	Teoría	18	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	01	Teoría	30	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	41	Prácticas en Laboratorio	30	+0
Grado en Ingeniería Civil	A	Informática y Programación	1	BR	2C	42	Prácticas en Laboratorio	30	+0

Imagen 2. Demanda docente del POD

De igual forma que ocurre con la demanda docente, la carga docente presenta la misma estructura. La tabla de carga docente presenta campos distintos como se puede observar en la Imagen 3. Carga docente POD. Clicando sobre el nombre del profesor podemos ver con mayor detalle las asignaturas que tiene asignado.



Nombre	Área	Potencial A	Potencial B	Total Horas a Cubrir	Docencia Máster	Docencia Doctorado	Docencia Pr. Externas	Horas sin Cubrir
Alayón Hernández, Francisco	C	173,1	173,1	**173,1**				-7,1
Alemán Flores, Miguel	C	171,3	171,3	**171,3**				8,7
Álvarez León, Luis	C	87,7	87,7	**87,7**				3,5
Benítez Díaz, Domingo	A	277	277	**262,3**	15		60	-7,3
Cabrera Gámez, Jorge	C	203,3	203,3	**203,3**	20			-3,3
Carreras Riudavets, Francisco Javier	L	162,5	162,5	**162,5**	90		60	2,5
Castrillón Santana, Modesto F.	C	79,6	79,6	**79,6**	10			35,4

Imagen 3. Carga docente POD

Cada profesor se asigna un horario de tutorías. Es por ello, que en la plataforma hay una página, muy similar a las anteriores, en el que se lista a los profesores, especificando el número de horas que se han cubierto por tutorías y el número de horas asignadas, Imagen 4. Lista de tutorías POD.

Tutorías Cubiertas por Profesor (2018/2019)

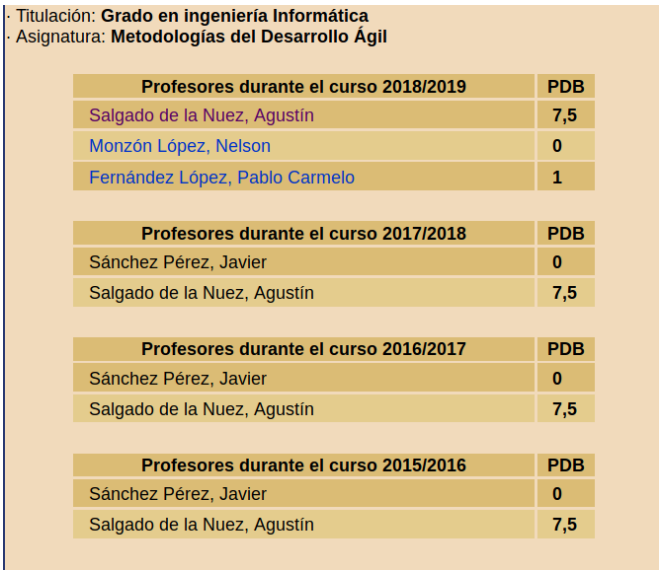
(Pinche sobre el título del campo para elegir el orden de la tabla)

Nombre	Área	Horas a Cubrir	Horas Cubiertas	Horas sin Cubrir
Alayón Hernández, Francisco	C	4,37	4,5	+0,13
Alemán Flores, Miguel	C	4,34	4,5	+0,16
Álvarez León, Luis	C	4,22	4,5	+0,28
Benítez Díaz, Domingo	A	5,66	6	+0,34
Cabrera Gámez, Jorge	C	4,15	4,5	+0,35
Carreras Riudavets, Francisco Javier	L	3,33	4	+0,67
Castrillón Santana, Modesto F.	C	4,02	4	-0,02
Correas Suárez, Beatriz	C	5,48	6	+0,52
Cuenca Hernández, Carmelo	A	4,24	4,5	+0,26
de Blasio García, Gabriel	C	2,93	3	+0,07
del Ser, Javier	C	0	0	+0
Domínguez Brito, Antonio	A	3,23	4	+0,77

Imagen 4. Lista de tutorías POD

Otra parte importante de la plataforma es el listado de profesores que son coordinadores de las asignatura o responsables de prácticas. Al igual que en las otras páginas, la información se representa en tablas, con dos códigos de colores: rojo para las asignaturas que aún no han sido seleccionadas y verdes para las que ya han sido cubiertas.

La aplicación actual cuenta con un historial de asignatura, donde se puede observar los profesores han impartido dicha asignatura en los años anteriores. En la Imagen 5. Historial



· Titulación: **Grado en ingeniería Informática**
· Asignatura: **Metodologías del Desarrollo Ágil**

Profesores durante el curso 2018/2019	PDB
Salgado de la Nuez, Agustín	7,5
Monzón López, Nelson	0
Fernández López, Pablo Carmelo	1

Profesores durante el curso 2017/2018	PDB
Sánchez Pérez, Javier	0
Salgado de la Nuez, Agustín	7,5

Profesores durante el curso 2016/2017	PDB
Sánchez Pérez, Javier	0
Salgado de la Nuez, Agustín	7,5

Profesores durante el curso 2015/2016	PDB
Sánchez Pérez, Javier	0
Salgado de la Nuez, Agustín	7,5

Imagen 5. Historial de asignaturas del POD

de asignaturas del POD, podemos ver un ejemplo para la asignatura de Metodología de Desarrollo Ágil.

8.1.2 Casos de usos

Existen dos roles dentro de la aplicación que aquí se presenta, el profesor y el administrador. Un usuario podrá tener ambos roles al mismo tiempo. Es por ello, que la interfaz gráfica está dividida en tres bloques: la cuenta del usuario, la administración y el POD.

Tabla 3. Control de acceso al POD

Bloques	**Administrador**	**Profesor**
Mi cuenta *(Perfil, Mi solicitud, mi Tutorías, Mi coordinación, Mi Carga docente)*	Solo al perfil	Todo el contenido
Administración *(Base de datos, Usuario, Solicitudes)*	Todo el contenido	Sin acceso
POD *(Demanda docente, Carga docente, tutorías, Coordinador de asignatura, Proyecto docente)*	Todo el contenido	Todo el contenido

El bloque de *Administración* se encargará de la gestión de usuario, de la base de datos y de las asignaciones de grupos. En la Imagen 6. Casos de uso del Administrador se puede observar todos los casos de usos relativos al rol de administrador.

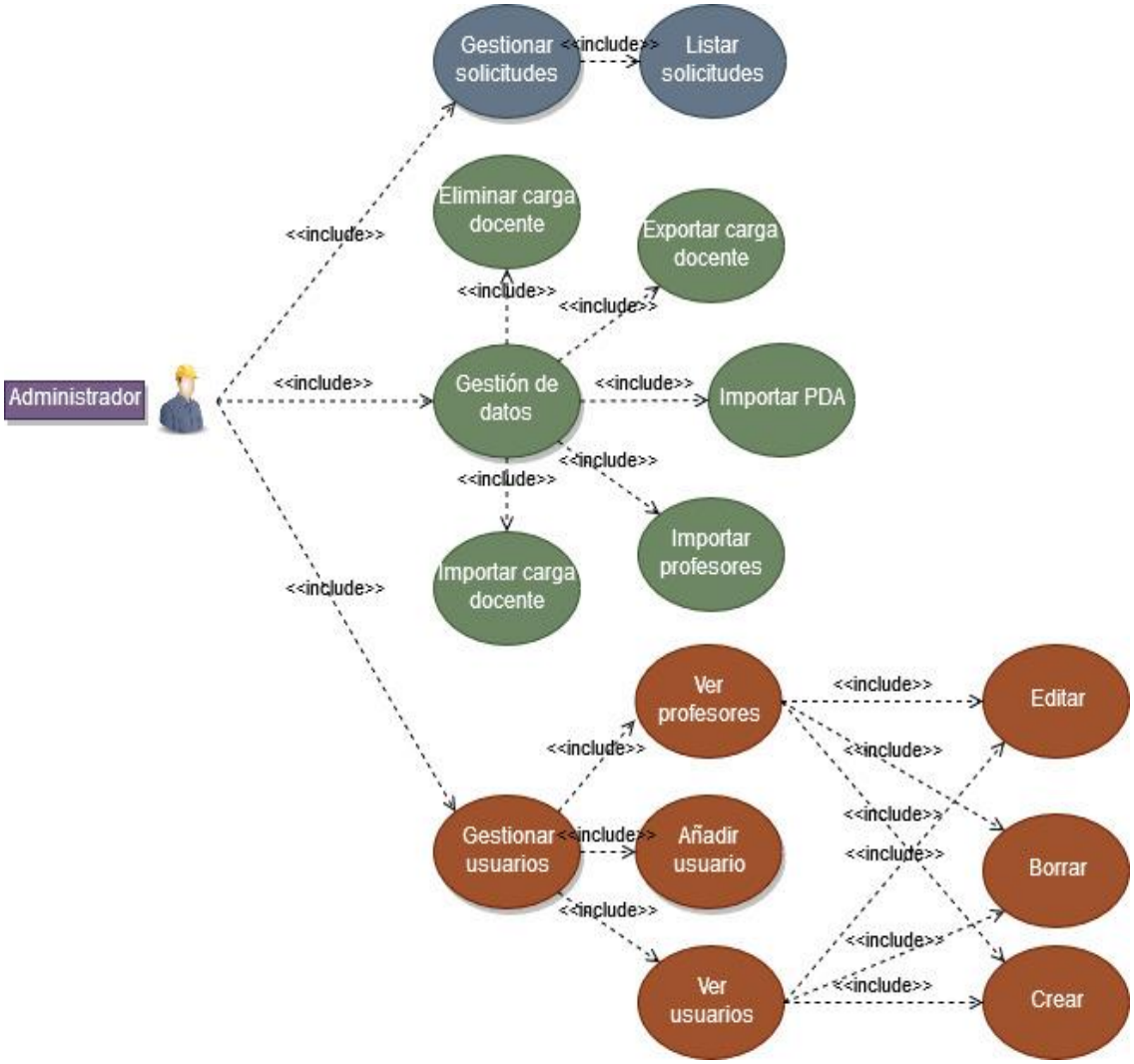


Imagen 6. Casos de uso del Administrador

El bloque de *Mi cuenta*, a excepción de *mi perfil*, solo será accesible para el profesor. Este bloque permite al profesor acceder a la gestión de sus tutorías, sus grupos asignados y las solicitudes realizadas hasta el momento. En la Imagen 7. Casos de usos del profesor. podemos observar los casos de uso del rol *profesor.*

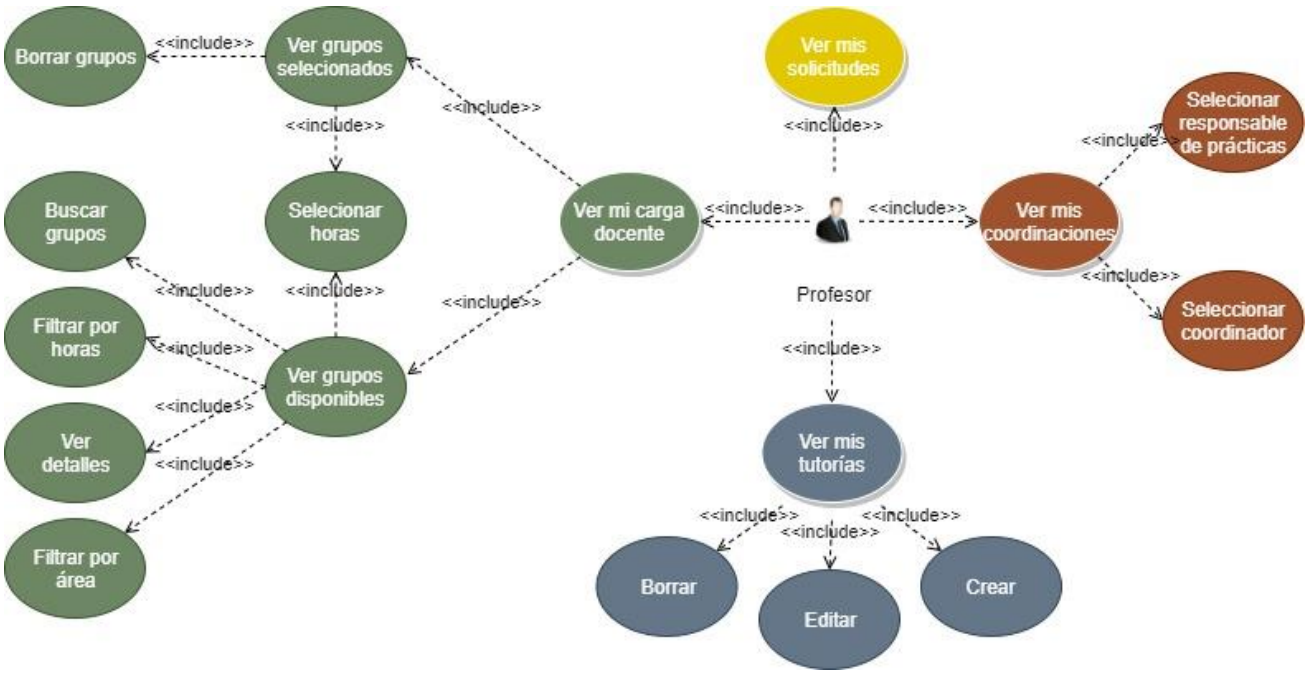


Imagen 7. Casos de usos del profesor.

El último bloque denominado *POD* será visible para todos los usuarios y tiene como objetivo informar sobre el estado actual del POD. Es decir, qué profesor está asignado a cada grupo, cuáles son las tutorías del profesorado, quién imparte docencia en cada grupo y quienes son los coordinadores y responsables de prácticas de las distintas asignaturas. Esta información es de carácter informativo por lo que tendrá permisos de sólo lectura. En la Imagen 8. Casos de usos de acceso y salida de la aplicación e Imagen 9. Casos de usos comunes al profesor/administrador podemos observar las acciones comunes a los roles administrador y profesor.

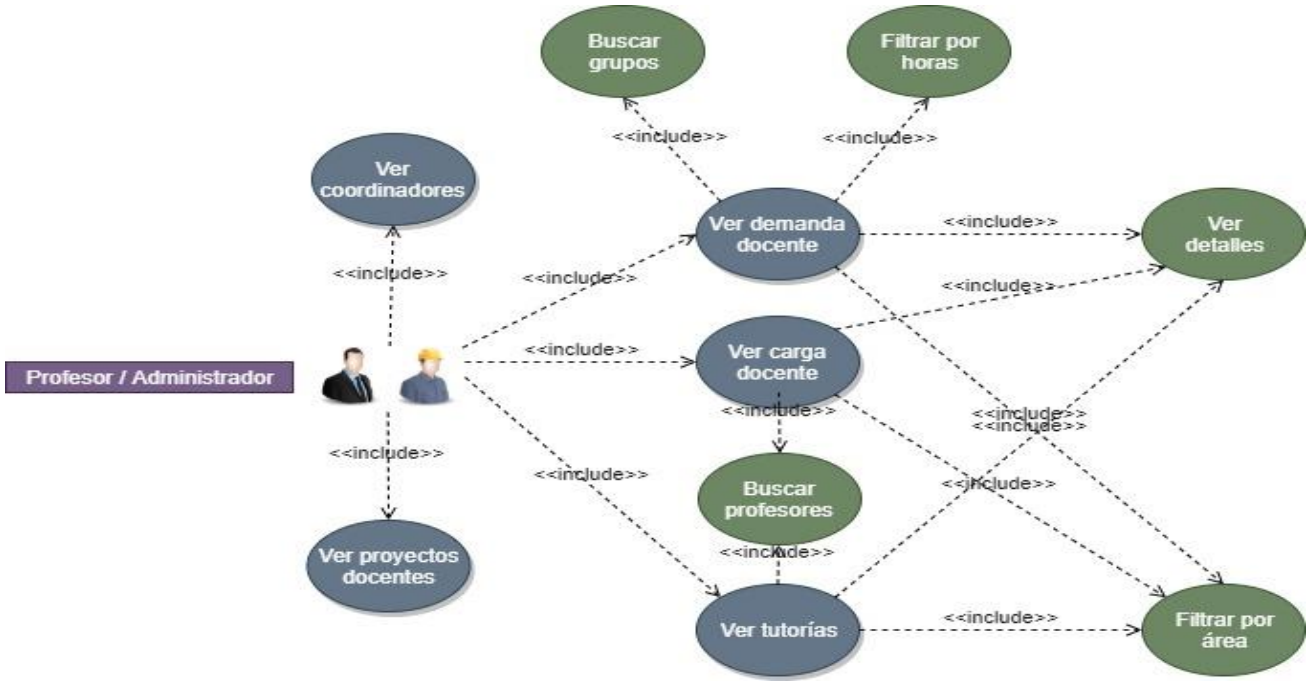


Imagen 9. Casos de usos comunes al profesor/administrador

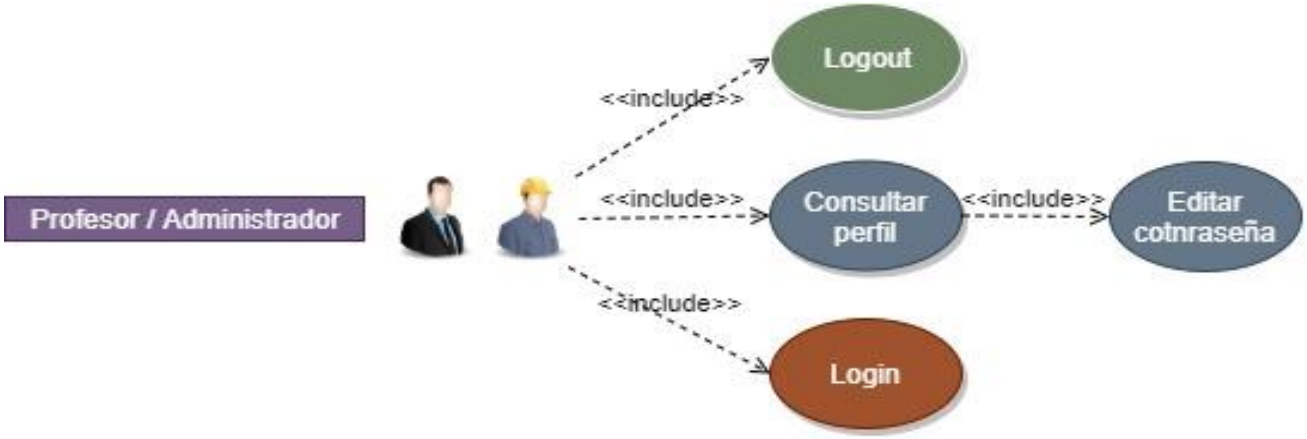


Imagen 8. Casos de usos de acceso y salida de la aplicación

8.1.3 Modelo Entidad-Relación

La base de datos está desarrollada en un sistema relacional que se puede representar mediante un modelo Entidad-Relación[1]. Una entidad -representada en un cuadro azul en la Imagen 10. Entidad-relación- representa una tabla en la base de datos. Dentro de este modelo existe relaciones entre las entidades representadas con rombos y uniones con doble líneas, o líneas simples. Las líneas dobles indican una relación fuerte, en otras palabras, una entidad no existe sin la otra.

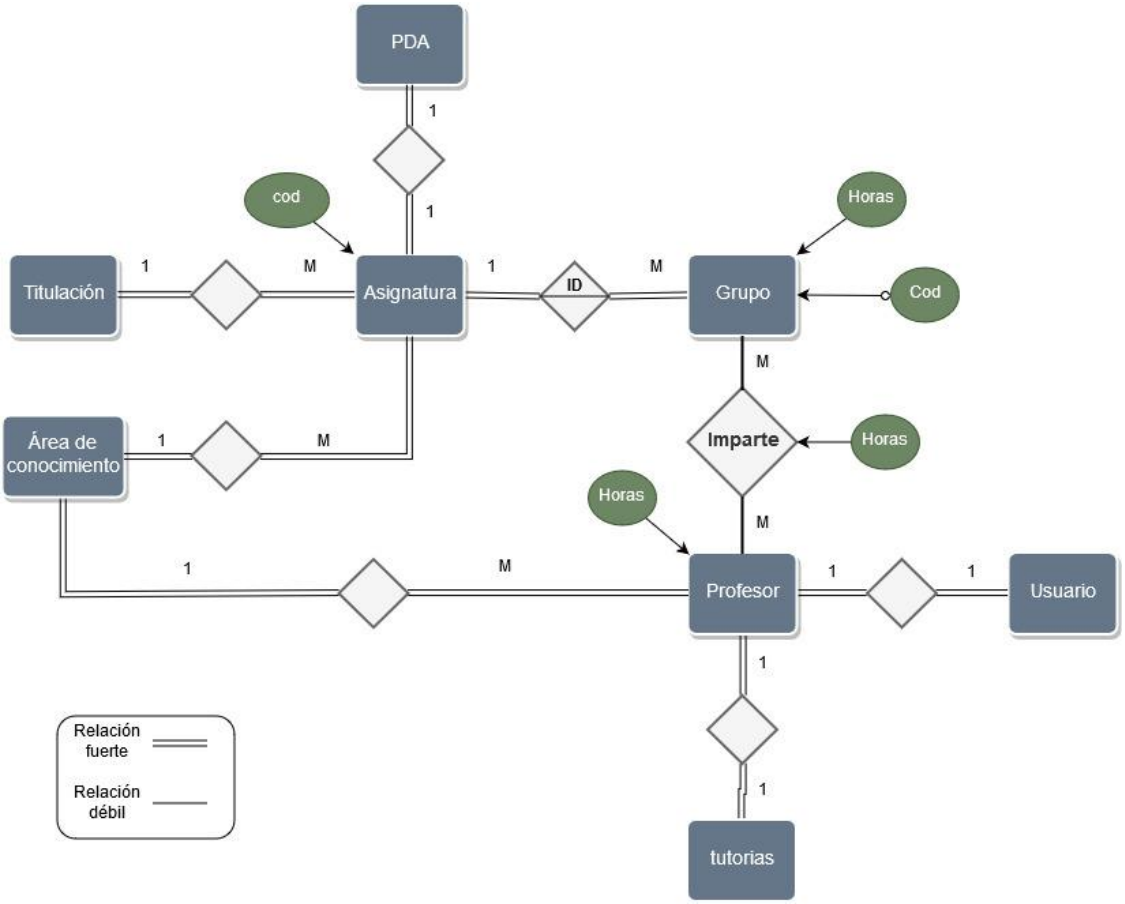


Imagen 10. Entidad-relación

Por ejemplo, una asignatura no puede existir si no existe una titulación o un área de conocimiento. Al contrario, si no existe ninguna asignatura, tampoco existe una titulación;

[1] Un modelo de entidad-relación es una herramienta para el modelo de datos, la cual permite representar entidades de una Base de Datos – Wikipedia [14.15]

ésta no puede quedar sin ninguna asignatura asignada. En el caso de las relaciones simples, las relaciones pueden existir independientemente.

Otro aspecto para tener en cuenta es la cardinalidad. La cardinalidad se representa con un valor numérico encima de la relación, indicando cuántas relaciones puede tener un registro de cada entidad en el otro. Por ejemplo, un área de conocimiento puede tener muchas asignaturas registradas; por ello se representa con una “M” (muchos). Al contrario, una asignatura sólo pertenece a un área de conocimiento y nunca a más de uno; por ello se representa con un “1”.

Así en la Imagen 10. Entidad-relación se puede observar las distintas entidades que forman la base de datos, así como sus relaciones y la cardinalidad de éstas. Lo más destacado en este modelo es la relación entre asignatura y grupo. La entidad grupo presenta en uno de sus atributos una pequeña bola (atributo “cod”). Esto nos está indicando que dicho atributo es un identificador débil, debido a que cada asignatura tiene un grupo y que dicho identificador es repetido en otra asignatura. Por ejemplo, pueden existir dos asignaturas distintas con un código de grupo 17, que serán completamente distintos. Para evitar esta situación de ambigüedad dentro de dicha relación se representa con la palabra “ID” y una línea horizontal dentro del rombo de relación. Esto hace referencia a que dicho identificador débil sea la combinación entre el código de la asignatura y el código del grupo para que pueda existir la entidad grupo. En otras palabras, la entidad grupo tendrán una clave primaria que será combinación entre el código de grupo y la clave foránea del código de la asignatura que está registrado ese grupo.

Otro aspecto que destacar en este modelo E-R es la relación entre la entidad grupo y profesor, donde dicha relación se le denomina con el nombre “imparte” que tendrá como atributo *“horas”*. Esta relación será una tabla nueva en la base de datos que relacionará muchos grupos con muchos profesores. En otras palabras, un profesor podrá impartir a ningún grupo o a varios; los grupos podrán ser impartidos por varios o ningún profesor. Esta situación puede parecer un poco extraña pero lo cierto es que se puede dar el caso; cuando los datos han sido cargados en la plataforma y aun ningún profesor ha seleccionado alguna de las horas de ese grupo que serán registrados en el atributo “hora” de la relación “imparte”.

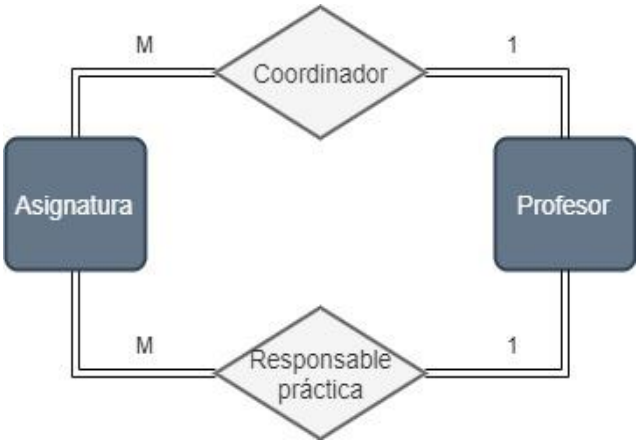


Imagen 11. Modelo entidad relación-asignatura/profesor

Para mayor claridad se muestra en Imagen 11. Modelo entidad relación-asignatura/profesor la relación de coordinador y responsable de práctica entre asignatura y profesores.

Debido a la naturaleza de la información, todas las entidades y relaciones, excepto la relación *imparte, coordinador y responsable de práctica*, generará un conjunto de datos estáticos. Es decir, éstos serán los mismos desde que se importan en la base de datos hasta que estos son exportados. Esto puede ser una particularidad del sistema y una de las razones por la cual se ha decidido por un sistema en concreto de gestión de la base de datos. La relación *imparte, coordinador y responsable de práctica*, son las únicas relaciones que crecerán con la interacción de los usuarios en la aplicación hasta que cada uno de los profesores selecciones su carga docente.

8.1.4 Archivos de Importación y Exportación

La importación de datos se lleva a cabo desde tres fuentes distintas (tres ficheros Excel con extensión *xlsx*) y un fichero de exportación. Esta idea fue la principal motivación al desarrollo de esta aplicación: la automatización de la entrada/salida de los datos del POD hacia una base de datos relacional. En este apartado se muestran los formatos de entrada y salida. Cualquier otro formato causaría un error, el servidor rechazaría la entrada de dichos datos devolviendo un error 500 o directamente, no importando los datos.

En el caso del archivo de importación del esquema de la base de datos debe ser igual a como se comenta a continuación. Para que la importación de datos funcione correctamente, los títulos de los campos deben de estar en la primera fila y sin tildes. Los nombres deben ser: “curso academico”, “cod titulacion”, “cod plan”, “cod especialidad”, “acronimo titulacion”, “nombre titulacion”, “centro imparticion”, “cod asignatura”, “nombre asignatura”, “curso asignatura”, “cuatrimestre asignatura”, “tipo asignatura”, “tipo de grupo”, “dni” (no es imprescindible), “horas”, “cod area”, “nomber area”, “venia” (no es imprescindible).

El formato de entrada del archivo Excel para el listado de profesores responde de igual forma al explicado anteriormente, pero con los campos que se muestran en la Imagen 12. formato de entra del listado de profesores:

dni	nombre	apellidos	potencial	horas tutorias	Cod Area	docencia master	docencia doctorado	practicas externas
44742831	profesor1	primer1, seg	250	15	35	15		
44742832	profesor2	primer1, seg	240	15	35		15	

Imagen 12. formato de entra del listado de profesores

El formato de entrada del archivo Excel del PDA tiene el formato que se muestra en la Imagen 13. formato de entrada de proyecto docente.

Cod Asignatura	Cod Area	Estado	Observaciones
40304	35	Aprobado	
40404	35	Suspendido	

Imagen 13. formato de entrada de proyecto docente

Es importante saber que si intentamos importar los PDA de asignaturas que no existen dentro de la base de datos, no se importarán, debido a que no se puede registrar un PDA de una asignatura inexistente. Lo mismo ocurre con el área de conocimiento de los profesores y PDA, si no existe no se importará.

Curso Acad.	Codigo Titulacion	Código Plan	Código Espec.	Código Asign.	Codigo Grupo	DNI (sin letra)	Num Horas	Código del Área Conoc	Venia
201920	4003	40	0	40304	17		60	35	
201920	4003	40	0	40304	18		60	35	

Imagen 14. Formato de archivo exportación

Cuando el administrador desee recoger los datos introducidos por la interacción del profesorado del departamento con la aplicación podrá realizarlo desde el bloque de *Administración* en la sección *exportar.* Previamente deberá aprobar las solicitudes que considere convenientes; entonces todas aquellas aprobadas se verán en el fichero que se empezará a descargar de forma automática y con los campos que vemos en la Imagen 14. Formato de archivo exportación. Dicho formato responde a requerimientos del DIS de la ULPGC. Aquellos profesores que hayan seleccionado algún grupo, del cual no coincida el área de conocimiento de la asignatura que pertenece y el área de conocimiento del profesor, aparecerá en la columna de *venia* la indicación de 'Sí'. En caso de que su área de conocimiento sea el mismo, aparecerá un 'No'.

8.2 Diseño

A partir de la información recopilada en la fase de análisis, el siguiente paso será la realización del diseño del sistema a implementar.

8.2.1 Aplicación Cliente-Servidor

La aplicación desarrollada para la gestión docente del DIS se encuentra dividida por dos partes siguiendo la arquitectura cliente-servidor.

El frontend, (o cliente) está desarrollada en *Angular* y su objetivo es interactuar con el usuario para facilitarle las labores de selección, ordenación docente y automatización de gestión. Ofrece una interfaz limpia y agradable, además de herramientas de búsqueda y de selección de ficheros de importación y exportación al administrador, gestión de los usuarios y gestión de peticiones por parte del profesorado.

El backend, está desarrollado en Python en base a una arquitectura REST, encargada de gestionar las peticiones desde uno, o más clientes. Las peticiones recibidas serán recogidas por un controlador (1), y posteriormente, se realizará una consulta al modelo (2), y éste, a la base de datos. La base de datos devolverá dicha respuesta al controlador para que responda nuevamente al cliente con un código de estado en la cabecera y un formato JSON[2] en el cuerpo, como se puede observar con más claridad en el diagrama de la Imagen 15. Esquema del servidor.

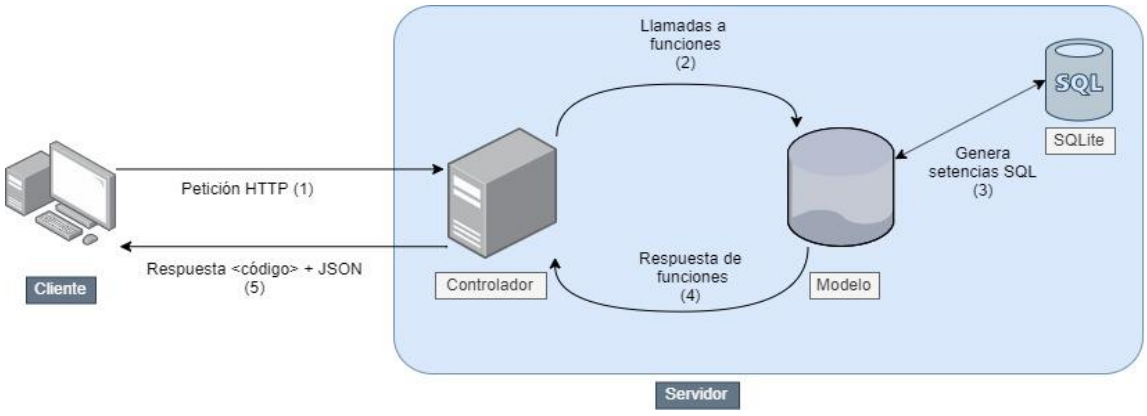


Imagen 15. Esquema del servidor

[2] Javascript Object Anotation – Notación de objecto de Javascript.

Otras de las funcionalidades que implementa el servidor es la posibilidad de importar varios archivos Excel con formato *.xlsx*. Dicha funcionalidad, fue propuesta desde el comienzo del proyecto para facilitar al administrador la gestión de la importación y exportación de la demanda docente del DIS. Hay que tener en cuenta que la arquitectura REST[3] que aquí se implementa se ejecuta sobre el protocolo de comunicación HTTP[4], sin estados. Esto quiere decir que tanto el cliente como el servidor no recuerda las peticiones que se han realizado anteriormente.

Angular utiliza un componente denominado *servicios*, que serán los encargados de controlar las peticiones que se realicen sobre el servidor según la interacción que tenga el usuario sobre la aplicación: como es el hecho de autentificarte, comprobar el token de sesión o pedir algún tipo de dato, etc. Las peticiones recibidas desde los servicios de Angular serán recibidas a través de la librería Flask. Esta librería facilita el uso de los protocolos HTTP, a través de los métodos GET, POST, PUT, DELETE para la comunicación a través de la red. A través de la cabecera de la petición y los datos enviados en el cuerpo responderá con cierto contenido (si fuese el caso) y un código en la cabecera: recurso creado (201), datos no encontrados (404), error de autentificación (401) o por que se ha producido algún error interno en el servidor (500). Una vez recibida la petición a través de Flask, el controlador desarrollado en Python será el encargado de obtener los datos solicitados al modelo; éste a su vez a la base de datos desarrollada en SQLite y devueltos otra vez al cliente en formato JSON.

Las comunicaciones generadas desde el modelo y SQLite se realizan a través de la librería SQAlchemy, es quién se encarga de generar sentencias SQL. La ventaja de esta librería es que podemos definir un cualquier objeto con identificadores, claves primarias, claves ajenas y el resto de campo. Dicha librería genera la sentencia SQL necesaria, independiente del gestor de base de datos que estemos utilizando, abriendo la posibilidad si fuese el caso de montar una base de datos en SQLite, MaríaDB, PosgreSQL o MySQL (que son las que actualmente soporta - Ver conexión a estas bases de datos y a otras en 14.13).

[3] Representation State transfer
[4] Protocolo de comunicación que permite las transferencias de información

8.2.2 Aspecto de la Interfaz de Usuario

En el siguiente apartado se describirá el diseño propuesto en el prototipo de aplicación. Teniendo en cuenta el aspecto del diseño, la interfaz gráfica tiene los colores y tonalidades azules oscuros típicos característicos de la Universidad de Las Palmas de Gran Canaria. La página de acceso sigue la misma tendencia, dividiendo la pantalla en dos partes, una con el azul oscuro degradado con el logotipo de la universidad y otra con un aspecto sencillo del formulario de acceso, como podemos ver en Imagen 60. Página de acceso La página principal, o dashboard, está formada por tres bloques.

El bloque localizado en la parte superior es la cabecera (ver Imagen 16. Navegador de escritorio). Ésta se mantendrá estática en la parte superior de la página, con el logotipo del DIS (parte superior derecha) y el logotipo de la ULPGC (parte superior izquierda).

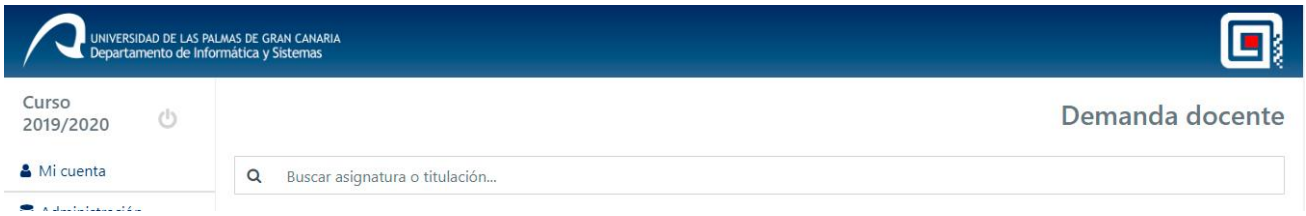


Imagen 16. Navegador de escritorio

El logotipo de la ULPGC contiene el logo del "*pensador*" y a su lado el nombre del DIS, que desaparecerá cuando por requerimiento del tamaño de la pantalla lo necesite. Como se ve en la Imagen 17. Navegador móvil.

El segundo bloque se encuentra el navegador situado en el lado izquierdo, de forma estática en la aplicación. Éste tendrá dos tipos comportamiento, el primero, será ocultarse cuando por requerimiento de la pantalla sea necesario Imagen 17. Navegador móvil. Cuando esto se produzca, en la cabecera de la aplicación aparecerá un icono con el cual podemos desplegar nuevamente el menú, provocando que el resto de la pantalla se oscurezca Imagen 18. Navegador desplegado. El segundo comportamiento que tiene el navegador será ocultar aquellas entradas que no se encuentre enfocadas, ocultándose automáticamente cada vez que se abre o se cierra un bloque (POD, Administración o Mi cuenta). En la navegación se encuentra el botón para cerrar la sesión; éste desplegará una ventana emergente que pedirá la confirmación de la acción.

Imagen 17. Navegador móvil

El tercer y último bloque del dashboard, es la parte dinámica de la aplicación donde se verá todo el contenido de la aplicación.

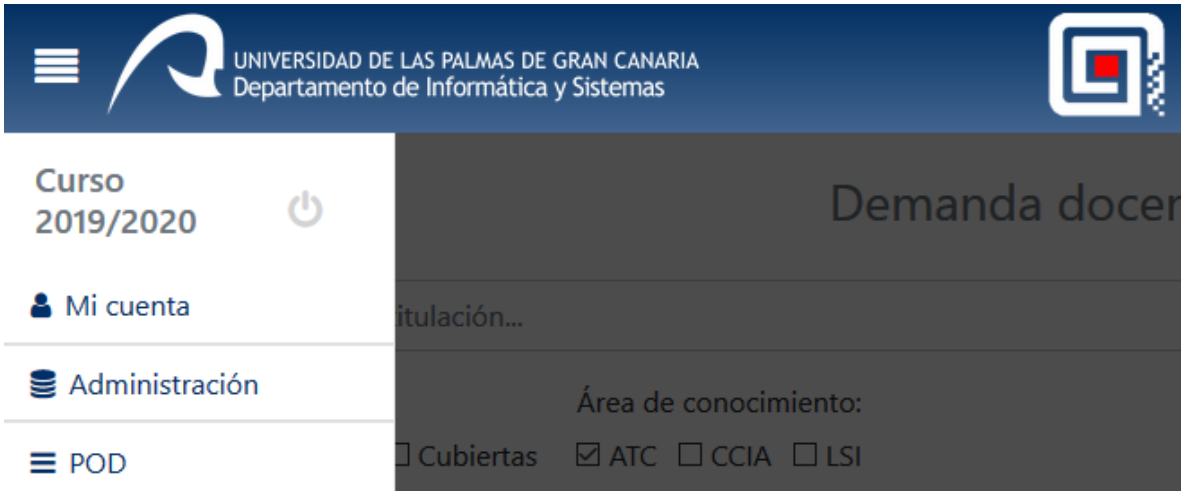


Imagen 18. Navegador desplegado

8.2.3 Tablas de la Base de Datos

Con el diagrama Entidad-Relación visto en el apartado 8.1.3 podemos realizar la tabulación del esquema en cualquier base de datos. En el apartado 8.1.3 hemos visto las distintas entidades y las relaciones que existen entre ellas. En este punto se diseña, las tablas y las relaciones, los campos de cada una de las entidades y lo más importante, las claves primarias, claves foráneas y claves combinadas. La base de datos ha sido desarrollada en SQLite y Python, lo que nos permite definir los objetos y de forma automática generar las sentencias SQL gracias a la librería SQLAlchemy de Flask. Este diseño, nos permitirá adaptarnos a distintos gestores de base de datos que deseemos implementar.

En la Imagen 19. Base de datos se encuentra la traducción del modelo Entidad-Relación a las tablas, así como sus relaciones y sus campos.

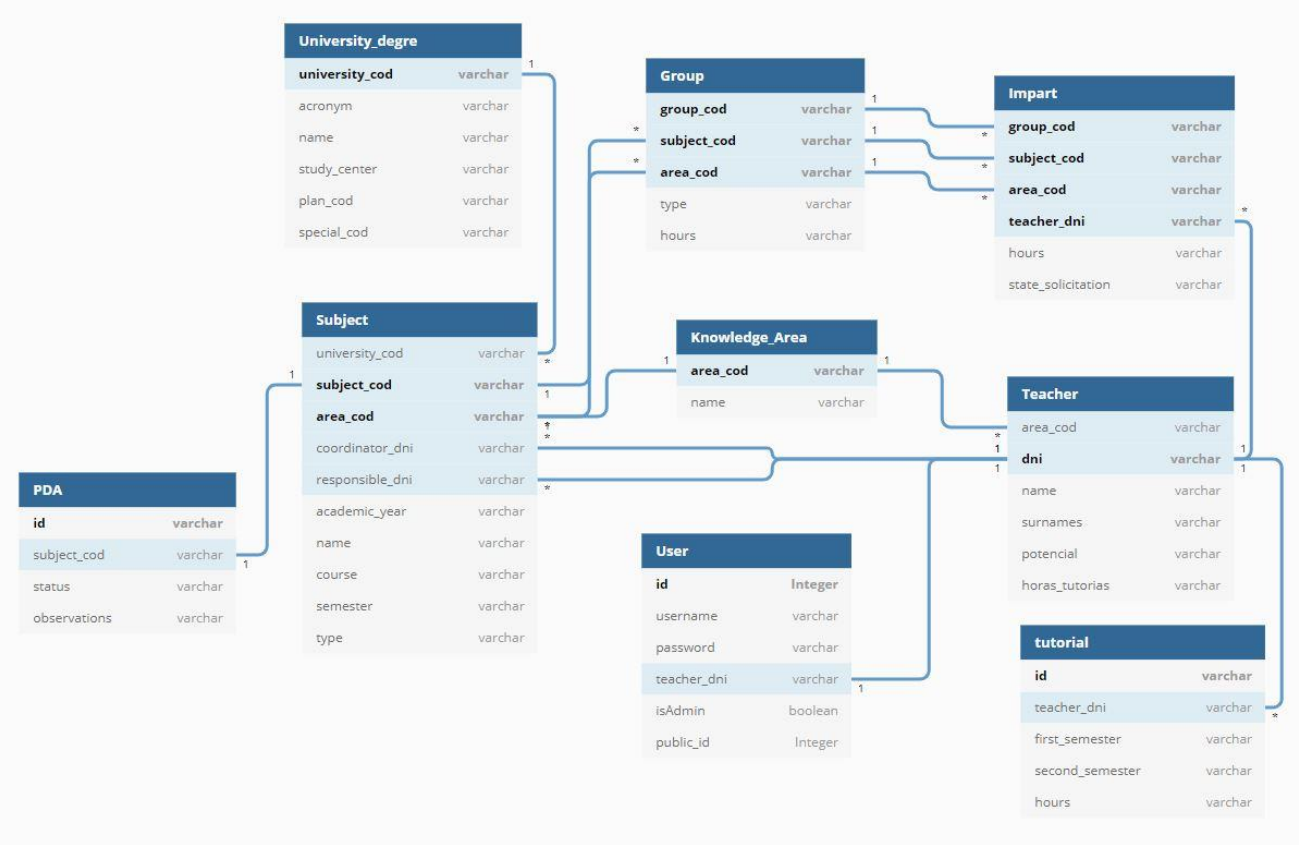


Imagen 19. Base de datos

Esto supone una gran diferencia de organización entre la plataforma que actualmente se utiliza y este prototipo que se presenta aquí, como se puede diferenciar en Imagen 20. Actual base de datos DIS.

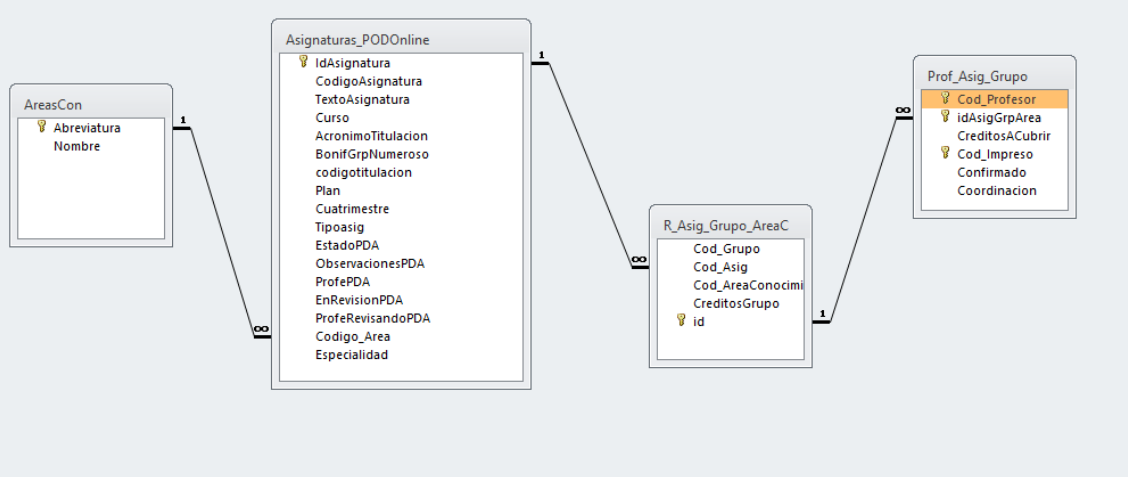


Imagen 20. Actual base de datos DIS

8.2.4 API Restful

La Tabla 4. API muestra el servicio Restful que ofrece el backend:

Tabla 4. API

Método	API	Descripción
Carga docente		
GET	/teacher_load	Obtiene la lista de la carga docente de cada profesor. Cada elemento de la lista representa los objetos de área de conocimiento y el profesor.
GET	/teacher_load/<*dni*>	Devuelve la carga docente del profesor con el "dni". El objecto devuelto es el profesor, la asignatura, el grupo, el área de conocimiento y el título universitario
POST	/teacher_load	Crea una nueva carga impartición de un profesor; *éste utiliza el token para identificarse*. Debe enviarse un JSON con "a*rea_cod"*, "*subject_cod*"*, "grupo_cod".*
DELETE	/teacher_load/<*área_cod*>/<*subject_cod*>/<*group_cod*>	Elimina la impartición de un profesor de un grupo. Los parámetros permitidos son: el código del área de conocimiento, el código de la asignatura y el código del grupo.
PUT	/teacher_load/<área_cod>/<subject_cod>/<group_cod> Ó /teacher_load/<área_cod>/<subject_cod>/<group_cod>/<*dni*>	Actualiza la impartición de un grupo. Los parámetros permitidos son el código del área de conocimiento, el código de la asignatura y el código del grupo. Los datos enviados permitidos son: "*impart_hours: string*", "approved: true | false", "*rejected: true* | *false*"

GET	/teacher_load/request	Devuelve todas las solicitudes de impartición. Los objetos devueltos son: *Profesor* e *impartición.*
	Grupos	
GET	/groups	Devuelve una lista con grupos con el número de horas cubiertas a través de la variable “group_cover_hours”. Los objetos devueltos son: *Asignatura, grupo, área de conocimiento, título universitario.*
GET	/group/<*área_cod*>/<*subject_cod*>/<*group_cod*>	Devuelve un solo grupo con una lista de profesores que imparten dicha asignatura y las horas cubiertas: “group_cover_hours”. Los objetos que devuelve son: *Asignaturas, Títulos universitario, áreas de conocimientos, grupos* y lista de *profesores.*
	Asignaturas	
GET	/subjects	Devuelve una lista de asignaturas. Los objetos devueltos de cada elemento de la lista son: *Títulos universitarios, áreas de conocimientos de cada asignatura*.
GET	/subject/<*subjecct_cod*>/<*area_cod*>	Devuelve una sola asignatura. Los objetos devueltos son: *Asignatura, área de conocimiento.*
GET	/subject/pda	Devuelve una lista de proyecto docente. Los objetos devueltos son: *PDA, Títulos universitarios, asignaturas, coordinador, área de conocimiento*
GET	/subject/coordinator	Devuelve una lista de coordinadores de asignatura. Los objetos devueltos son: *Asignatura, título universitario, área de conocimiento, coordinador.*

GET	/subject/coordinator/<*dni*>	Devuelve un listado de asignatura de un coordinador. Los objetos devueltos son: *Títulos universitarios, asignatura, área de conocimiento*
POST	/subject/coordinator/<*dni*>	Crea un coordinador para una asignatura. Debe enviarse un JSON con "a*rea_cod"*, "*subject_cod*".
GET	/subject/responsible	Devuelve una lista de responsable de prácticas de las asignaturas.
GET	/subject/responsible/<*dni*>	Devuelve un listado de asignatura de un responsable de práctica. Los objetos devueltos son: *asignatura, titulación, área de conocimiento.*
POST	/subject/responsable/<dni>	Crea un responsable de práctica de una asignatura. Debe enviarse un JSON con "*área_cod*", "*subject_cod*"
	Usuario	
POST	/Login	Autentificación de usuario. Es necesario enviar "*username*" y "*password*". Si es correcta esta combinación el método devolverá un token.
GET	/currentUser	Devuelve el objeto *user*.
GET	/users	Obtiene una lista de usuarios
GET	/user/<*username*>	Devuelve un usuario identificado por el "username". Los objetos devueltos son: *usuario* y *profesor, si lo fuese.*
DELETE	/user/<username>	Elimina un usuario, excepto si es el usuario "admin".
PUT	/user/<username>	Actualiza datos del usuario.
POST	/sign-in	Registra a un nuevo usuario, si éste también es profesor, es creado. Debe enviarse un JSON con "username", "*admin*", "password", de forma opcional:

		"name", "surnames", "potential", "tutorial_hours", "área_cod"
Profesor		
GET	/teacher	Devuelve una lista de profesores.
GET	/teacher/<*dni*>	Devuelve un profesor. Devuelve el número de horas que ha cubierto el profesor a través de la variable "group_cover_hours"
PUT	/teacher/<*dni*>	Actualiza los datos de un profesor. Debe enviarse un JSON con: "teacher_dni", "teacher_name", "teacher_surnames", "área_cod", "tutorial_hours", "teacher_potential", "teacher_master", "teacher_doctorate", "teacher_practice"
DELETE	/teacher/<*dni*>	Elimina un profesor
GET	/teacher/tutorial	Devuelve una lista de los profesores que tenga tutorías asignadas. Los objetos devueltos son: *profesor, tutorías, áreas de conocimientos.*
GET	/teacher/tutorial/<*dni*>	Devuelve las tutorías de un profesor.
POST	/teacher/tutorial/<teacher_dni>	Crea el horario de tutorías de un profesor.
Base de datos		
POST	/database	Carga el archivo de importación Excel con el esquema de la base de datos [8.1.4]
GET	/database	Descarga el esquema de la base de datos. Según recoge el punto [8.1.4]
DELETE	/database	Elimina el contenido de las tablas de la base de datos.

POST	/database/teacher	Carga el archivo de importación Excel con el contenido de un listado de profesores. Según recoge el punto 0
POST	/database/pda	Carga el archivo de importación Excel con el contenido de un listado de profesores.

Para hacer usos de todas las funciones es necesario autentificarse. Si el token es válido devuelve el valor pedido, si no lo es lanzará un error desautorización. En el apartado 8.2.6 se describe con mayor detalle. A continuación, en

Tabla 5. formatos de los objetos devueltos, se muestra el formato de cada uno de los objetos de la base de datos que devuelve la API.

Tabla 5. formatos de los objetos devueltos

Objeto	*JSON*
Impartición	{“group_cod”: String, “subject_cod”: String, “area_cod”: String, “teacher_dni”: String, “assigned_hours”: String, “approved”: boolean, “rejected”: boolean}
Asignatura	{“teacher_dni”: string, “teacher_name”: string, “teacher_area_cod”: string, “teacher_surnames”: string, “teacher_potential”: string, “tutorial_hours”: string, “other_teaching”: string}
Grupo	{“group_cod”: string, “subject_cod”: string, “área_cod”: string, “group_type”: string, “group_hours”: string}
Profesor	{“teacher_dni”: string, “teacher_name”: string, “teacher_area_cod”: string, “teacher_surnames”: string, “teacher_potential”: string, “tutorial_hours”: string, “other_teaching”: float}
Área de conocimiento	{“area_cod”: string, “área_name”: string}
Título universitario	{“university_cod”: string, “university_acronym”: string, “university_name”: string, “university_study_cente”: string, “university_plan_cod”: string, “university_special_cod”: string}

PDA	{“id”: entero, “subject_cod”: string, “área_cod”: string, “subject_name”: string, “pda_status”: string, “pda_observe”: string}
Usuario	{“id”: entero, “username”: string, “teacher_dni”: string, “isAdmin”: boolean}
Tutorías	{“first_semester”: Array<string>, “second_semester”: Array<string>, hours: string}

8.2.5 Rutas del frontend

El frontend utiliza algunos elementos de “Angular 7” llamados componentes. Algunos de estos componentes se utilizan como base para albergar y construir las vistas en base a otros componentes. Es por ello, que en este siguiente apartado se describirá las rutas que siguen los componentes bases para mostrar las distintas vistas.

Los componentes bases se estructuran de forma jerárquica (en forma de árbol), donde desde el archivo *app.component.html* (nodo raíz) se genera o se renderiza el resto de los componentes. En el segundo nivel se encuentra dos nodos hijos (o dos componentes bases), la página de Login y la página dashboard. Una vez, que el usuario haya accedido con sus credenciales, será redirigido a dashboard y éste generará un conjunto de nuevos nodos hijos.

```
const routes: Routes = [
  { path: '', component: LoginComponent},
  { path: 'login', component: LoginComponent},
  {
    path: 'dashboard',
    component: DashboardComponent,
    children: [
      // PDO pages
      { path: 'teacher-demand', component: TeacherDemandComponent, canActivate: [AuthGuard] },
      { path: 'teacher-load', component: TeacherLoadComponent, canActivate: [AuthGuard] },
      { path: 'teacher-tutorial', component: TeacherTutorialComponent, canActivate: [AuthGuard] },
      { path: 'teacher-tutorial-details/:id', component: TeacherTutorialDetailsComponent, canActivate: [AuthGuard] },
      { path: 'subject-coordinator', component: SubjectCoordinatorComponent, canActivate: [AuthGuard]},
```

Imagen 21. Rutas de la interfaz gráfica

Tal como se puede observar en la Imagen 21. Rutas de la interfaz gráfica (archivo *app-routing.module.ts*) la palabra reservada *children* generará todos los componentes dentro de

dashboard. También se han añadido páginas *not-found* para indicar al usuario cuando una ruta que no es válida.

Por último, se puede observar que en cada path (dirección URL), component (Elemento que se debe renderizar) se encuentra otro parámetro denominado *canActivate* que llama al servicio *AuthGuard*. Esto es así para proteger el acceso a las distintas vistas de la aplicación para los usuarios que hayan sido logueados. Por ejemplo, si un usuario intentará acceder a una de las páginas a través de la URL, éste sería redirigido a la página de acceso.

8.2.6 Sistema de autentificación de usuario

Para hacer uso de cualquier recurso de la base de datos es necesario autentificarse como un usuario válido. Por ello es necesario pasar al recurso */Login* las clave "username" y "password" dentro de del cuerpo de una petición HTTP con formato JSON. Si este usuario es correcto el recurso */Login* devuelve un identificador denominado token que debe ser colocado en la cabecera de todas las peticiones que se realicen al servidor. Este identificador se utiliza para identificar al usuario que está haciendo uso de algún recurso; es necesario renovarlo cada 30 minutos (la aplicación dará un aviso y redireccionará al usuario a la página de acceso).

La base de datos contiene una tabla denominada *user*. En ella existe un campo denominado "*public_id*" que es utilizada para generar el token desde la clase *token_required* para identificar al usuario.

La clase *token_required* (fichero *router.py*) es un decorador de Python que se colocado encima de los recursos que quieran ser protegidos, de tal forma, el usuario debe ser autentificado mediante token. [14.11]

8.2.7 Sistema de archivo de la Interfaz Gráfica

Para poder entender la estructura del frontend es necesario explicar brevemente como está organizada las carpetas y sus subcarpetas. Angular está orientado a páginas o

aplicaciones de una sola página y representar a los elementos como “componentes”. El código está organizado en función de esos “componentes” para que puedan ser usados con el simple hecho de invocarlos. Por ello, dentro de la carpeta *disManagerApp-client* (repositorio GitHub [14.1]) se puede observar la organización característica de Angular (e2e, node_modules, src, assets, environments, etc).

Dentro de la carpeta *app* se encuentran todos los componentes de Angular con los cuales hemos estado trabajando todo el tiempo. La carpeta *app* está organizada en cuatro carpetas y seis ficheros. El archivo *app.component.html* es la vista principal de la aplicación. En ella se renderizará todos los componentes que hayan sido invocados. El archivo *app.component.ts* es el controlador de la vista encargado de lanzarla cuando sea invocado. El archivo *app.component.scss* es el encargado de darle estilos a la vista.

El archivo *app.module.ts* es el más importante de Angular, ya que aquí se declaran todos los elementos con los que cuenta la aplicación. Por último, dentro de los ficheros, se encuentra el archivo *app-routing.module.ts*. Éste se trata de un controlador para realizar la navegación entre los distintos componentes de Angular. Aunque Angular no fue pensado para este propósito, nos permite realizar navegación a través de jerarquías en forma de árbol gracias al componente `<router-outlet></router-outlet>`. Éste identifica de forma automática si el componente que se intenta renderizar es uno de los hijos de algún nodo padre o es un nodo padre.

Las carpetas que se encuentran en el directorio *app* son *components, pages, pipes, services*. La carpeta *components*, están todos los componentes que forman la página, como son elementos comunes (Login, cabecera, navegación, dashboard, etc), formularios o ventanas emergentes (modal).

En la carpeta *pages* se encuentran todos los componentes que se utilizan para enrutar al frontend. Estos componentes están organizados en tres bloques principales: (i) POD (común para todos los usuarios), (ii) Administración (solo disponible para el administrador) y (iii) Mi perfil.

La carpeta *pipes* alberga a los componentes especiales de Angular con los que interactúa el usuario de forma inconsciente. Por ejemplo, para organizar por nombre de asignatura las

tablas, por nombre el listado de profesores, por área de conocimiento o búsqueda por nombres.

La carpeta *servicios* contiene otro componente especial de Angular. Aquí se encuentran todas las comunicaciones y servicios que son utilizados para recuperar los datos desde el back-end. El archivo *rest.service.ts* es el encargado de recoger todos los datos de la API [0] del back-end. El archivo *header.service.ts* es el encargado de construir las cabeceras con el token del usuario, una vez identificado. El archivo *authentication.service.ts* se encarga de guardar las credenciales y token del usuario que en ese momento está logueado en la página, y el que interactúa con el localstorage del navegador para almacenar las cookies de la sesión del usuario. También borra dicha información cuando el usuario haga *salga* o finalice el tiempo del token y deba volver a loguearse para volver a comprobar su autenticidad. Por último, el archivo *auth-guard.service.ts* se encarga de comprobar que el token es válido cada vez que se acceda algún recurso de la página. Este servicio es invocado desde *app-routing.module.ts*, así cuando el token caduque, al usuario es redirigido a la página Login.

8.2.8 Sistema de archivos del servidor

El sistema de archivos se aloja dentro de la carpeta *disManagerApp-server* (repositorio GitHub [14.2]). El archivo principal para la ejecución del backend se encuentra en el archivo raíz *dismanagerapp.py.* Desde aquí, el método main de Python *_name_* se encarga de lanzar el servidor Flask para recoger las peticiones Restful del cliente.

Dentro del directorio *src* encontramos los diferentes ficheros que se describen a continuación:

En el archivo *_init_.py* se encuentra las configuraciones del servidor, como es la palabra secreta con la que se realiza el cifrado hash para el token del usuario. Además, en este archivo se encuentran las importaciones de los paquetes que son necesarios para el correcto funcionamiento del sistema, así como, de la importación de la configuración de la base de datos que podemos encontrar en el archivo *config.py* localizado en la raíz del proyecto.

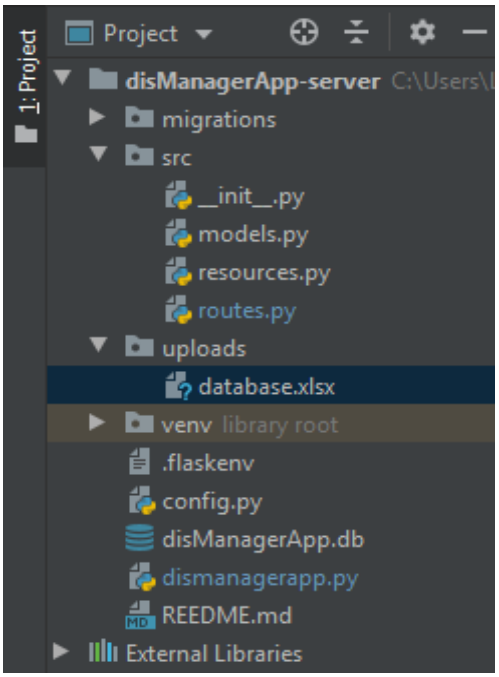


Imagen 22. Sistema de archivo backend

En el archivo *models.py* se encuentran los modelos y entidades de la base de datos, así como sus relaciones que posteriormente son migradas a la base de datos.

En el archivo *recources.py* están todas las funciones auxiliares de las cuales se soportan el resto de los métodos. Entre ellos se encuentran los métodos de tabulación de los datos obtenidos desde los ficheros importados (Excel). En el archivo *routes.py* se encuentra la API del servicio Restful. En base al protocolo HTTP en el cual el usuario realiza las peticiones, éste hace el papel de controlador del servidor. El directorio *uploads* es el lugar donde se guardan los archivos exportados/importados Excel de forma temporal. La entidad *disManagerApp.db* es la base de datos autogenerada desde el archivo *config.py en SQLite.*

8.3 Implementación

Una vez descritas las etapas de análisis y diseño, explicaremos las funciones más relevantes realizadas en la fase de implementación de la aplicación.

8.3.1 Página de acceso

La principal entrada y única a la plataforma es la *Página de acceso*. El diseño sigue la apariencia de la página de la ULPGC e incluye el logotipo oficial, como se puede observar en la Imagen 60. Página de acceso.

Para acceder a la plataforma debemos de tener las credenciales: usuario y contraseña. Estas credenciales son introducidas en el formulario del Login y son proporcionadas por el DIS. Si el acceso se había hecho dentro de la última hora, la interfaz gráfica comprobará la validez del token enviándoselo al servidor (la duración de la sesión). Entonces se nos redireccionará directamente a la plataforma debido a que el token de la sesión esta guardada en el local Storage del navegador. En caso contrario, si no hemos iniciado la sesión, deberemos de introducir las credenciales en el formulario. El formulario realizará la validación de la información introducida mostrando mensajes de error si fuera necesario, como podemos observar en Imagen 23. Validación Página de Acceso.

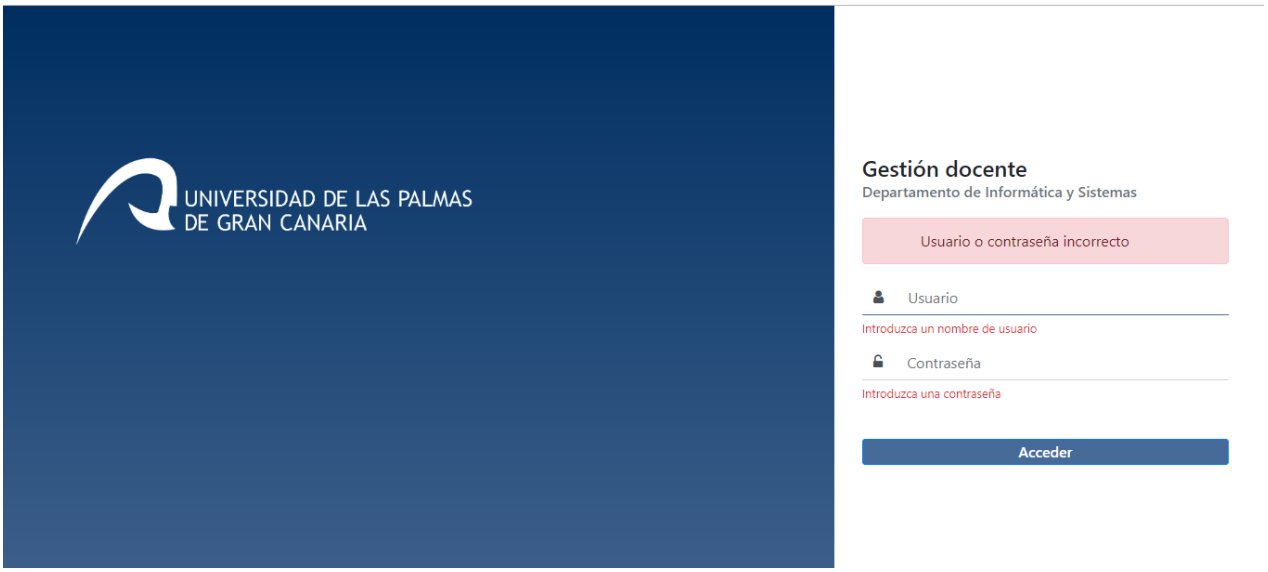


Imagen 23. Validación Página de Acceso

Cuando las credenciales son correctas, el botón de acceder nos informará que “Esperemos” hasta que finalice la validación. Este proceso es enviado en formato de JSON con el servicio *authentication.service.ts* al servidor. En caso de ser correcto nos enviará un token de sesión y datos del usuario que será guardado en local Storager. Posteriormente redireccionado a la página de demanda docente como vemos en el código de la Imagen 24.

```
44      this.auth.login(this.loginForm.value)
45        .subscribe(
46          next: data => {
47            this.auth.setToken(data.token);
48            this.auth.setUser(data.user);
49            this.router.navigate( commands: ['/dashboard/teacher-demand']);
50
51          },
52          error: error => {
53            this.activateLoad = false;
54            this.alert.loginIncorrect = true;
55          }
56        );
```

Imagen 24. Código de acceso Angular

Código de acceso Angular

El proceso descrito anterior es recogido por el servidor. El encargado de esto es la Api */Login* como se puede observar en la Imagen 25. Código de login del servidor. El algoritmo obtiene el objeto usuario de la base de datos a través del *username,* y posteriormente, realiza el *hash* de la contraseña para verificar que ésta coincide (línea 355) con la que se encuentra almacenada en la base de datos de forma cifrada. Si todo es correcto, se genera un token gracias a la librería de Flask-JWT donde se utiliza información como la *public_id* del usuario y una palabra secreta (que podemos encontrar en el archivo _init_.py) para generarlo con una duración de una hora aproximadamente. Cada una de las páginas a las que deseamos acceder están protegidas por el servicio *auth-guard.service.ts* para comprobar que dicho token es válido, en cuyo caso permitirá el acceso a la información.

```python
@app.route('/login', methods=['POST'])
def login():
    try:
        auth = request.get_json()

        if not auth or not auth['username'] or not auth['password']:
            return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})

        user = User.get(username=auth['username'])
        if not user:
            return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})

        if check_password_hash(user.password, auth['password']):
            token = jwt.encode({'public_id': user.public_id, 'exp': datetime.utcnow() + timedelta(minutes=60)},
                               app.config['SECRET_KEY'])

            user_dict = user.to_dict()
            if user.teacher:
                teacher_dict = build_dict(teacher=user.teacher, area=user.teacher.knowledgeArea)
                user_dict.update(teacher_dict)

            return response({'token': token.decode('UTF-8'), 'user': user_dict})

        return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})
```

Imagen 25. Código de login del servidor

8.3.2 Gestión de la Carga Docente

La página de gestión de la base de datos se encuentra dentro del bloque *Administración* y sólo es visible para el usuario administrador. En este bloque se realizarán labores de importación, exportación de ficheros y la eliminación de los datos que contiene las tablas en

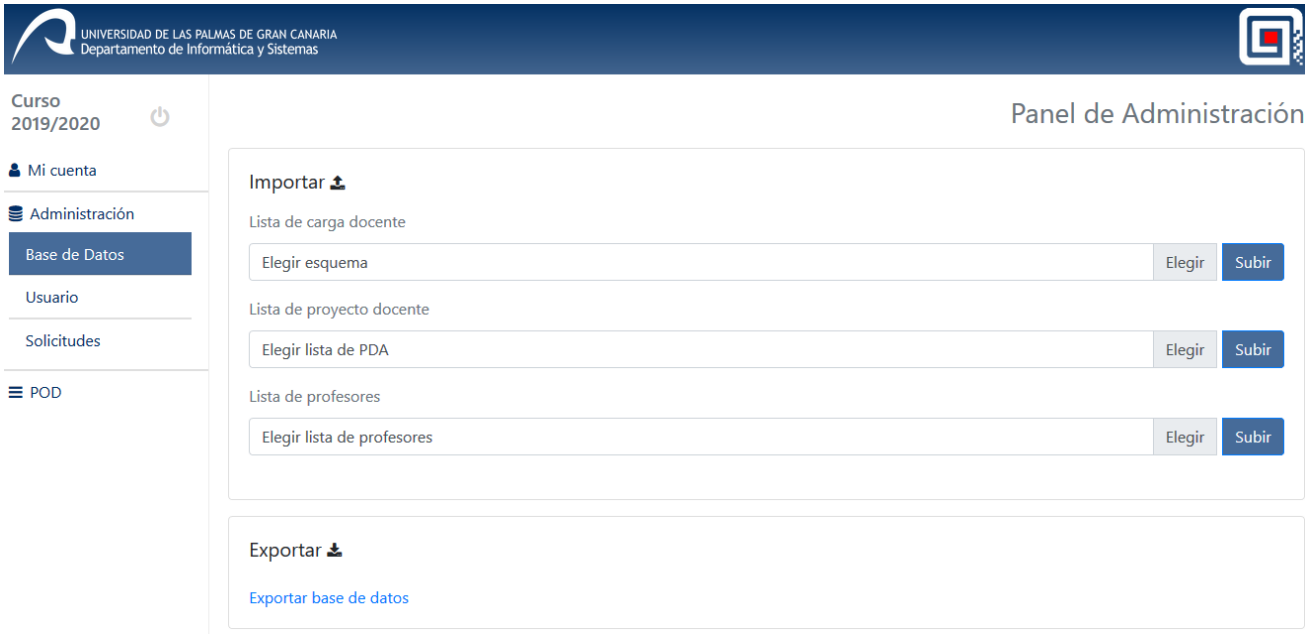


Imagen 26. Página de administración de BD

la base de datos. La sección de importación como se puede observar en la Imagen 26. Página de administración de BD nos permite importar tres grupos de datos: *la carga docente*, *los proyectos docentes* y *el listado de profesores*.

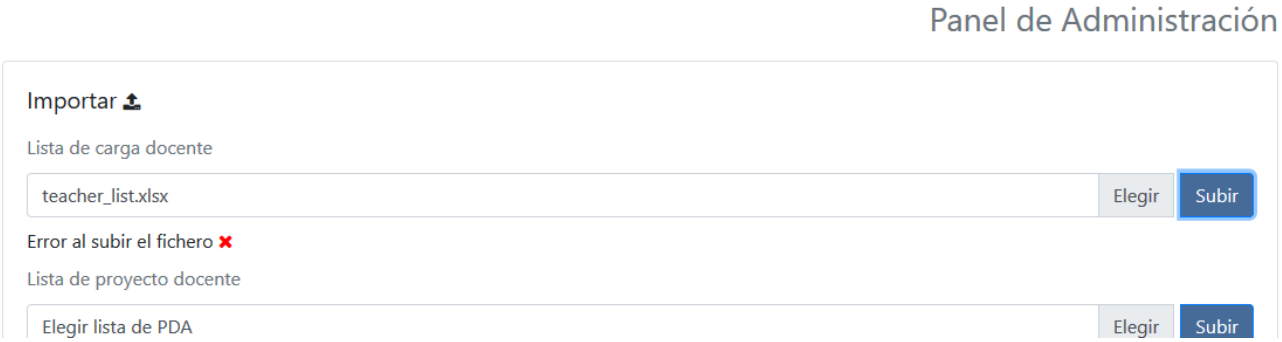


Imagen 27. Error importación

El fichero de importación de la *carga docente* se adapta a requerimientos expresos del DIS, donde la estructura de dicho fichero se indica en el punto 8.1.4. Si los ficheros de

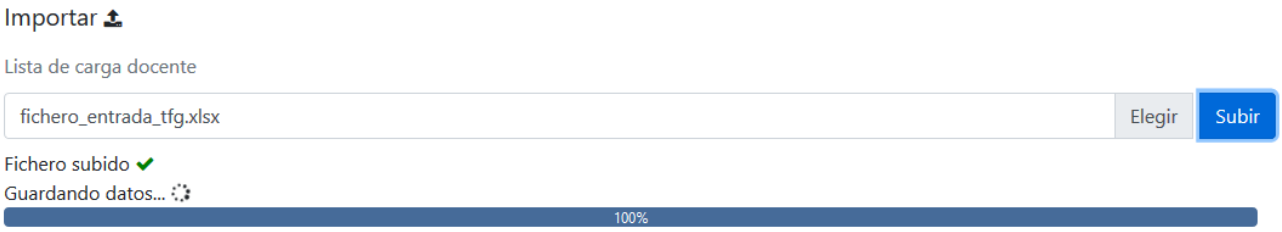


Imagen 29. Carga de fichero satisfactorio

importación no corresponden con lo descrito en este apartado o existe algún error en la importación, nos mostrará un error como podemos ver en la Imagen 27. Error importación. En caso contrario, el proceso que seguirá el programa será indicarnos si el fichero ha sido subido al servidor, Imagen 29. Carga de fichero satisfactorio. Si los datos han sido guardados en la base de datos nos muestra un mensaje, Imagen 28. Carga de datos satisfactorio.

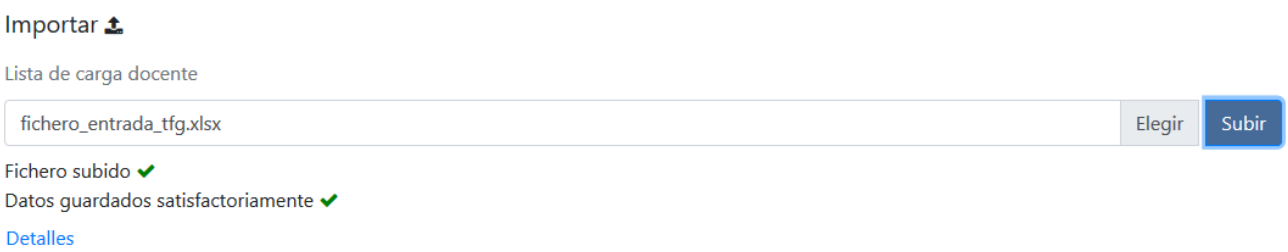


Imagen 28. Carga de datos satisfactorio

Si todo ha tenido éxito, se nos mostrará un botón de *detalles.* Cliqueando nos mostrará una ventaja emergente con la cual podemos verificar de forma rápida si todas las filas del fichero Excel han sido importadas, como se ve en la Imagen 30. Detalles de la importación

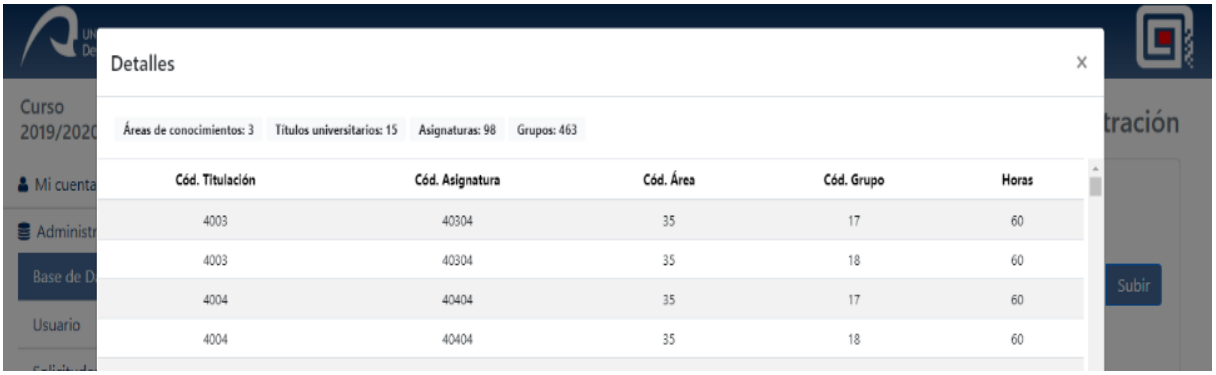


Imagen 30. Detalles de la importación

Hay que tener en cuenta que para importar la lista de PDA y el listado de profesores antes debemos tener importado el esquema de la base de datos. Dado que ambas listas dependen del área de conocimiento y de las asignaturas, en el caso del PDA.

Todo este proceso es recogido por el servidor. Los ficheros les llegan a través de los *servicios* de Angular. En el caso especial de un fichero, es necesario utilizar el objeto de Javascript, FormData() que está dentro del servicio *rest.services.ts*. En la Imagen 31. Código

```
35    postLoadScheme(fileToUpload: File): Observable<any> {
36      const formData = new FormData();
37      formData.append( name: 'file', fileToUpload, fileToUpload.name);
38      return this.http.post( url: this.endpoint + '/database', formData,
39        options: {
40          headers: this.header.builHeaderFile(),
41          observe: 'events',
42          reportProgress: true
43        });
44    }
```

Imagen 31. Código de importación de esquema

de importación de esquema se muestra el código que se encarga de utilizar la API del servidor a través del método POST para enviar el archivo. La función *builHeaderFile()* del objeto *header* construye la cabecera de la petición añadiendo el token del usuario. El método *postLoadScheme* devolverá un observable (un objeto especial de Angular) con la información de la importación devuelta por el servidor.

La petición que se genera anteriormente es recogida por el controlador del servidor, el archivo *router.py* a través de la API */database* y el método *POST* como se puede ver en Imagen 32. Método upload_database.

Este método comprueba si la petición contiene algún fichero. Si no lo contiene informa al usuario; en el caso que exista, el fichero es renombrado y guardado para ser pasado al método *openxlsx.* Este método se encargará de abrir, recoger todos los datos del fichero y convertirlo en un JSON, donde la clave de éste será el título de la columna del Excel.

Posteriormente, la construcción hecha en formato JSON es pasado al método *import_schema* (fichero *resource.py)* donde se hará la importación a la base de datos devolviendo la información de los registros guardados. Dicha información es devuelta al usuario con un código de éxito, por defecto de 201. Una vez, terminado todo el proceso de importación en la base de datos, se borrará el archivo.

```
603     @app.route('/database', methods=['POST'])
604     @token_required
605     def upload_database(user):
606         try:
607             if user.isAdmin:
608                 # check if the post request has the file part
609                 if 'file' not in request.files:
610                     return response({'message': 'File not found'}, 404)
611
612                 request_file = request.files['file']
613                 if request_file and allowed_file(request_file.filename):
614                     filename = secure_filename(request_file.filename)
615                     filename_dir = Resource.join_file(filename)
616                     request_file.save(filename_dir)
617
618                     data_file = Resource.openxlsx(filename_dir)  # return
619                     data_saved = import_schema(data_file)
620                     os.remove(filename_dir)
621
622                     return response({'message': data_saved})
623                 else:
624                     return response({'message': 'File not found'}, 404)
625         except Exception as ex:
626             return response({'message': ex}, 500)
```

Imagen 32. Método upload_database

Este proceso es el mismo que se realiza para la importación del PDA y de la lista de profesores, con sus respectivos métodos y servicios.

Si deseamos eliminar los datos importados o los generados por la interacción de la demanda docente, podemos ir a la sección eliminar esquemas. Se nos mostrará un mensaje que debemos confirmar, y posteriormente un mensaje que nos mostrará el proceso como vemos en la Imagen 33. Borrar datos de la Base de Datos.

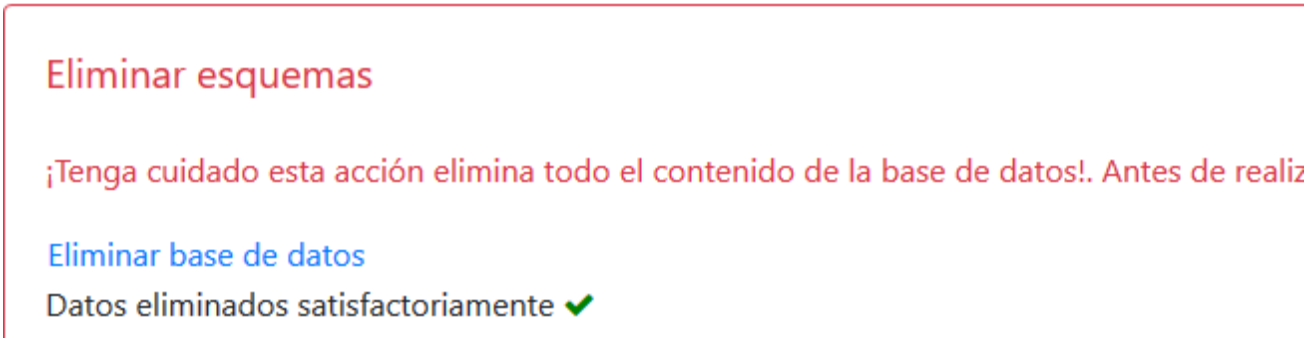


Imagen 33. Borrar datos de la Base de Datos

Otro aspecto importante, es la exportación de los datos donde el formato del fichero se describe en el apartado 8.1.4; esta operación se podrá llevar a cabo desde la sección *Exportar*. El servidor desde la API */database* con el método GET generará el archivo de exportación con el formato predefinido por el DIS. El código que recoge la petición de exportación es el que se puede ver en la Imagen 34. Código exportación

```
@app.route('/database', methods=['GET'])
@token_required
def download_database(user):
    try:
        return send_file(export_schema(), cache_timeout=-1)
    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 34. Código exportación

El método *donwload_database* utiliza el método especial de Flask, *send_file()* que devuelve al cliente el fichero generado por la función *export_schema()* que se encuentra en el fichero *Resource.py*. La opción, *cache_timeout*, que recoge en el método *send_file(),* evita que la respuesta del servidor sea cacheada en el navegador. El método export_schema() (archivo *Resource.py)* se encarga de crear el fichero con extensión *xlsx*, añadir los campos

a la cabecera y recoger los datos de la base de datos; plasmarlos en dicho archivo para posteriormente ser devueltos al usuario.

8.3.3 Administración de usuarios

La página de administración de usuarios está disponible en el bloque de "administración" y de acceso exclusivo para el administrador. Las funcionalidades de esta página se basan en la visualización de los usuarios que contiene la plataforma. Se muestra un conjunto de detalles, así como actualizar o eliminar dichos usuarios; como se puede ver en la Imagen 35. Visualización de usuarios e Imagen 36. Visualización de profesores

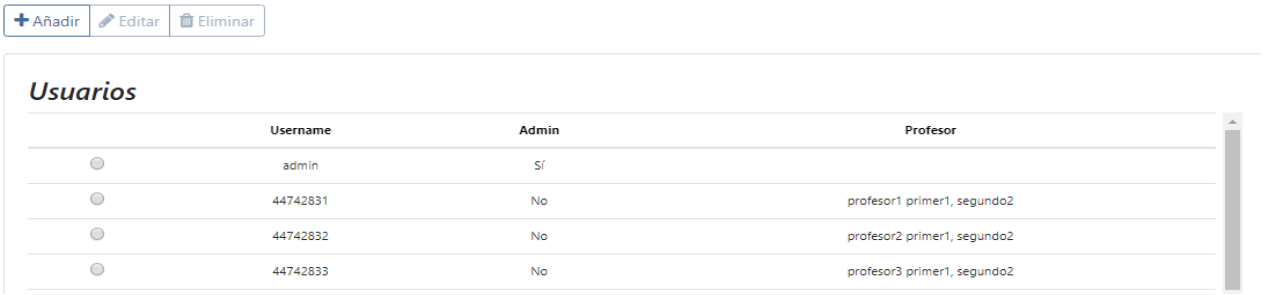


Imagen 35. Visualización de usuarios

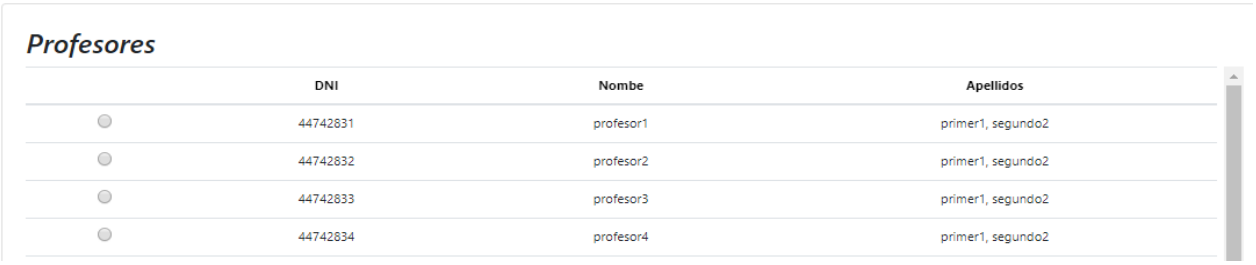


Imagen 36. Visualización de profesores

En la parte superior se encuentra el grupo de botones para realizar distintas acciones. Algunos de esos botones se mantendrán deshabilitados hasta que algunas de las filas de algunas de las tablas sean seleccionadas. Para añadir un conjunto o lista de profesores se realizará desde la página *Base de Datos.* Por el contrario, si necesitamos añadir un usuario o un profesor, podremos realizarlo desde esta sección cliqueando el botón añadir. Inmediatamente se desplegará una ventana emergente con un formulario, que una vez pasada la validación se añadirá a la base de datos. En la Imagen 37. Validación del usuario se puede observar el formulario y los avisos que se mostrarán en la validación.

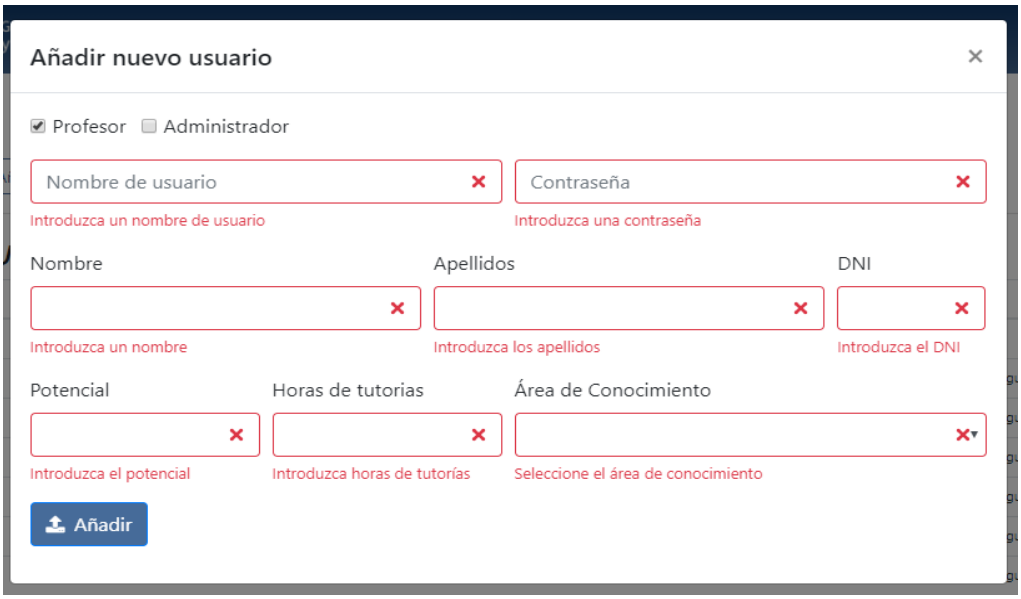


Imagen 37. Validación del usuario

El formulario es estructurado en un JSON y enviado al servidor que es recogido por el siguiente método que se muestra en la Imagen 38. Código de registro de usuario/profesor La función contempla dos casos. El primer caso, sólo se hará el registro de un usuario. El segundo caso, hará el registro de un usuario y profesor. Los datos mínimos necesarios serán el *username*, *password* y *admin*. Con esta información se crea un objeto usuario asignándole

```python
@app.route('/sign-in', methods=['POST'])
def create_user():
    try:
        data = request.get_json()
        if data and contains_keys(['username', 'password', 'admin'], data.keys()):
            user = User.get(username=data['username'])
            if user:
                return response({'message': 'User already exists'}, 301)
            user = User(data['username'], generate_password(data), data['admin'], public_id())
            if 'dni' in data.keys() and data['dni']:
                teacher = Teacher.get(data['dni'])
                if teacher:
                    user.teacher = teacher
                    if user.save():
                        return response({'message': 'User and Teacher have been created!'})
                    else:
                        return response({'message': 'User and Teacher could not have been created!'}, 401)

                if contains_keys(['name', 'surnames', 'potential', 'tutorial_hours', 'area_cod'], data.keys()):
                    area = KnowledgeArea.get(data['area_cod'])
                    if not area:
                        return response({'message': 'Knowledge Area could not been find'}, 404)

                    teacher = Teacher(data['dni'], data['name'], data['surnames'], data['potential'],
                                      data['tutorial_hours']
                                      , area)
                    user.teacher = teacher
                    if teacher.save():
                        return response({'message': 'Teacher created'})
            else:
                if user.save():
                    return response({'message': 'User created'})

        return response({'message': 'User could not have been created!'}, 404)
    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 38. Código de registro de usuario/profesor

una id pública (con la función hash 256) y cifrando la contraseña del usuario. Además, si contiene la información de un profesor se asignará ese usuario al profesor que se cree. Hay que tener en cuenta que, si dicha área de conocimiento no existe no se creará al profesor.

Para editar o eliminar un usuario o un profesor, debemos seleccionarlo en algunas de las listas. Estas acciones están controladas por la directiva de *Angular 7* llamada *ngModel* que establece una bidireccionalidad de los datos. De tal forma, cada vez que alguna de las filas es seleccionada, el valor de ésta es guardada en la variable seleccionada por esta directiva. En el caso de seleccionar una fila de la lista de profesores se guardará con la variable *teacherSelected.* En el caso de un usuario, *userSelected,* como se ve en Imagen 39.

```
<tbody>
<tr *ngFor="let user of tableUser">
  <td><input type="radio" name="buttonEdit" [value]="{userSelected: user}" [(ngModel)]="valueRadioButton" ></td>
  <td class="align-middle">{{user?.username}}</td>
  <td class="align-middle" *ngIf="user.isAdmin">Sí</td>
  <td class="align-middle" *ngIf="!user.isAdmin">No</td>
  <td class="align-middle">{{user?.teacher_name}} {{user?.teacher_surnames}}</td>
</tr>
</tbody>
```

Imagen 39. formulario de lista de usuario

formulario de lista de usuario

Así, una vez seleccionamos editar o eliminar llamará a las funciones que podemos ver en la Imagen 40. Funciones de editar y borrar usuarios y profesores Esto provocará la aparición de una ventana emergente para editar un usuario pasándole como referencia la variable de la directiva *ngModel*. En el caso de eliminar, una ventana emergente para confirmar la eliminación de la fila seleccionada, si aceptamos, se lanzará un evento que invocará a la función *confirmDelete*.

```
buttonEdit() {
  if (this.valueRadioButton == null) { return; }
  if (this.valueRadioButton.teacherSelected != null) { $('#modal-teacher-edit').modal('show'); } else
  if (this.valueRadioButton.userSelected != null) { $('#modal-user-edit').modal('show'); }
}

confirmDelete($event: boolean) {
  if ($event) {
    if (this.valueRadioButton.userSelected != null) {
      this.rest.deleteUser(this.valueRadioButton.userSelected.username).subscribe( next: value => this.loadList());
    } else {
      this.rest.deleteTeacher(this.valueRadioButton.teacherSelected.teacher_dni).subscribe( next: value => this.loadList());
    }
  }
}
```

Imagen 40. Funciones de editar y borrar usuarios y profesores

```python
@app.route('/user/<username>', methods=['PUT'])
@token_required
def update_user(user, username=None):
    u = User.get(username=username)
    data = request.get_json()
    if not u or not data:
        return response({'message': 'User not found'}, 404)

    u.update(data)
    if u.save():
        return response({'message': 'User successfully update'})

    return response({'message': 'User not could update'}, 404)
```

Imagen 42. Código actualización de usuario

```python
@app.route('/teacher/<dni>', methods=['PUT'])
@token_required
def update_teacher(user, dni=None):
    try:
        data=request.get_json()
        if not data and not dni:
            return response({'message': 'Data not found'}, 404)

        teacher = Teacher.get(dni)
        if teacher:
            teacher.update(data)
            if teacher.save():
                return response({'message': 'User successfully update'})

        return response({'message': 'User not could update'}, 404)
    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 41. Código de actualización de profesor

Cuando se confirme la edición del usuario, el servicio de Angular enviará los datos en formato JSON al servidor. El código que recoge esta petición de actualización es la que podemos ver en la Imagen 42. Código actualización de usuario e Imagen 41. Código de actualización de profesor y que posteriormente, es enviado al modelo de la base de datos.

8.3.4 Solicitudes

La página de solicitudes, disponible solamente para el administrador, es la encargada de revisar las solicitudes enviadas por los profesores. Las solicitudes se muestran en una tabla, donde se indica el grupo, las horas y el profesor que la ha seleccionado. También se indica si es necesario venia para alguno de los profesores.

El administrador debe aprobar las solicitudes si quiere que estén representados en el archivo de exportación. Los botones que aparecen en la cabecera se encuentran deshabilitados hasta que sea seleccionada al menos una de las solicitudes.

En la primera columna de la tabla se encuentra un *checkbox* para seleccionar cada una de las filas de la tabla. Estos checkbox's se comportarán de forma diferente según la interacción del usuario. El primer comportamiento es el que se describe en la Imagen 43. Algoritmo de selección completo. Está relacionado con el *checkbox* que se encuentra en la

```
private selectedAllGroup($event) {
  this.availableBtn = $event.target.checked;
  this.allSelected = $event.target.checked;
  for (const group of this.teacherRequest) { group.activedChanger = $event.target.checked; }
}
```

Imagen 43. Algoritmo de selección completo

```
private changeCheckboxGroup($event) {
  if (!$event.target.checked && this.allSelected) { this.allSelected = false; }
  if (this.activedAllGroup()) { this.allSelected = true; }
  for (const group of this.teacherRequest) {
    if (group.activedChanger) { this.availableBtn = true; break; } else { this.availableBtn = false; } }
}
```

Imagen 44. Algoritmo de selección

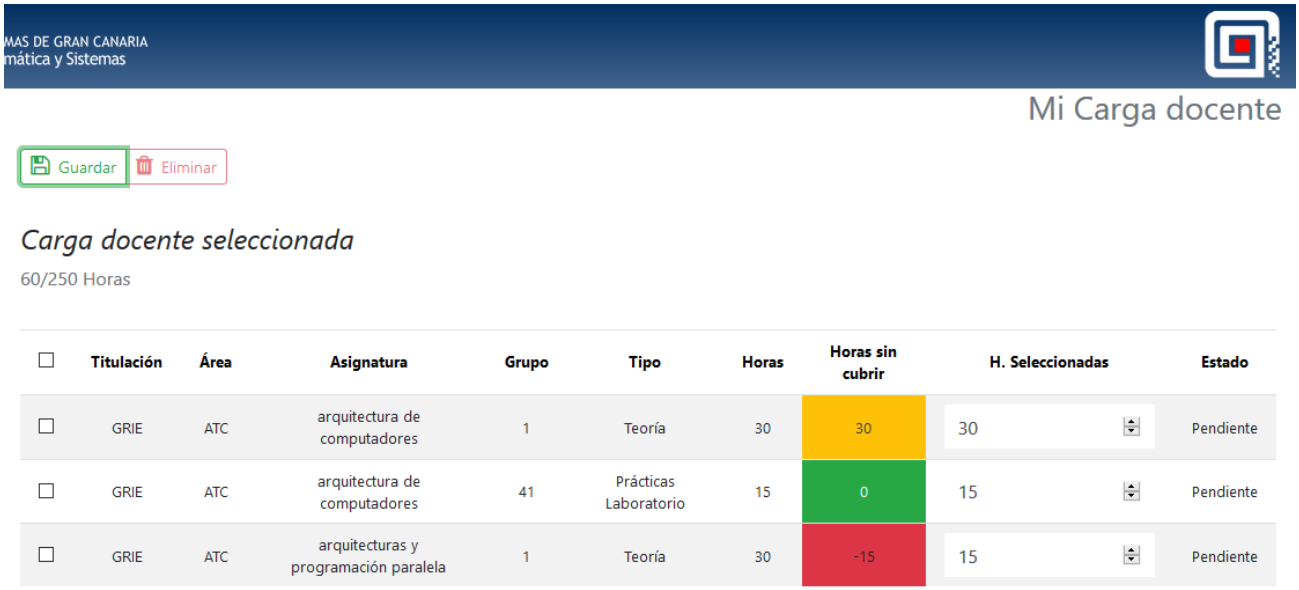
primera fila, primera columna de la lista, que se utiliza para seleccionar o deseleccionar todas las solicitudes. El segundo comportamiento está relacionado con los *checkbox*'s que se encuentran en cada solicitud y que describe el algoritmo que se puede ver en la Imagen 44. Algoritmo de selección. Este código además de activar el botón de la cabecera también modifica el *checkbox* principal (de la primera fila y primera columna) para que se seleccione si hemos seleccionado todas las solicitudes o se deseleccione si hemos deseleccionado todas las solicitudes.

8.3.5 Carga docente

La página *Mi carga docente* está destinada solamente para que los profesores puedan seleccionar el número de horas que desean impartir en los distintos grupos. Considerada una de las páginas principales para el profesorado; está estructurada en dos bloques. El primero, la carga docente seleccionada que muestra el listado de grupos que el profesor ya ha sido seleccionado, en cuyo caso que de no existir aparecerá un mensaje informativo.

Vista de la interfaz de usuario

En la Imagen 45. Bloque de carga docente se puede observar los dos grupos seleccionados por el profesor, donde se muestra la información: (a) las horas sin cubrir que tiene el grupo (rojo, cuando no se ha cubierto el número de horas, verde cuando es cero, amarillo, cuando se ha excedido), (b) horas seleccionadas por el profesor, (c) las siglas del área de conocimiento y (d) de la titulación de la asignatura, (e) nombre de la asignatura, (f) las horas que tiene el grupo y además (g) el estado que se encuentra la solicitud. En la parte superior se indica el número de horas que puede seleccionar el profesor y el número de horas seleccionadas (tanto en solicitud presentadas como las que está seleccionando en el momento).



	Titulación	Área	Asignatura	Grupo	Tipo	Horas	Horas sin cubrir	H. Seleccionadas	Estado
☐	GRIE	ATC	arquitectura de computadores	1	Teoría	30	30	30	Pendiente
☐	GRIE	ATC	arquitectura de computadores	41	Prácticas Laboratorio	15	0	15	Pendiente
☐	GRIE	ATC	arquitecturas y programación paralela	1	Teoría	30	-15	15	Pendiente

Imagen 45. Bloque de carga docente

El otro bloque de esta página se encuentra la carga docente disponible en el POD, como se ve en la Imagen 46. Bloque de carga disponible.

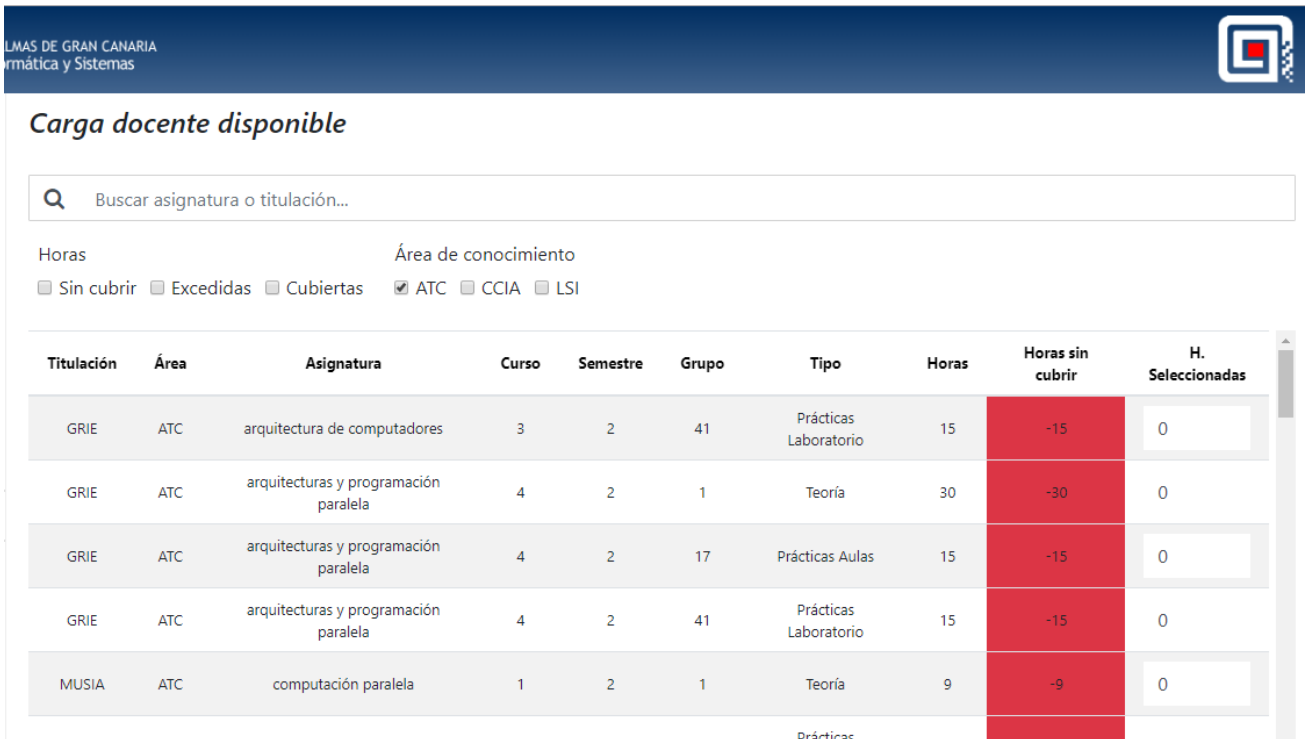


Imagen 46. Bloque de carga disponible

La apariencia del listado de la "*carga docente disponible*" es muy similar al listado de la "carga docente seleccionada". La tabla de la "*carga docente disponible*" cuenta con un buscador para localizar las asignaturas/titulación que el profesor desee impartir y un filtro para seleccionar el tipo de horas y área de conocimiento, donde éste se mantendrá seleccionado por defecto. En la parte superior de este bloque aparecerá dos botones que nos permitirá guardar la edición de las horas de los grupos que hemos seleccionado, editarlo o directamente eliminarlo cuando los seleccionemos en el grupo de *checkbox's* (éste funciona de la misma forma que como se explica en el apartado 8.3.4).

Para editar se puede cambiar directamente sobre la columna de "*H. seleccionadas*" y pulsar el botón de "*Guardar*". No se permitirá que un profesor seleccione una mayor cantidad de horas que las horas disponibles por el grupo; en cuyo caso se colocará el máximo disponible.

Controlador de la interfaz gráfica

El controlador de esta sección se encuentra en el archivo *user-subject.component.ts* que descargará, en la función *loadData(),* tanto los grupos seleccionados por el profesor y toda la demanda docente existente (Imagen 47. Carga de asignaturas del profesorado*).*

De ambos conjuntos de datos se eliminará de la carga docente la que ya ha sido

```
private loadData() {
  forkJoin(
    this.loadGroups(),
    this.loadTeacher()
  )
  .subscribe( next: value => {
      this.teacherGroupsConfirm = value[1];
      this.groups = value[0].filter(teaGrp => !this.teacherGroupsConfirm.find( predicate: val => this.compareGroup(val, teaGrp)) );
      this.loadDeleterIcon = false;
      this.calculatorHoursImpart();
  });
}
```

Imagen 47. Carga de asignaturas del profesorado

seleccionada por el profesor, para evitar que dichos grupos estén repetidos en ambas tablas.

El código que se describe en la Imagen 48. Código de confirmación de la demanda docente se encuentra las dos funciones que son llamadas cuando el usuario presione el

```
private confirmSelection() {
  const sendData = this.groups
    .filter(value => value.impart_hours > 0.1 )
    .map(value => {
      return { area_cod: value.area_cod, subject_cod: value.subject_cod, group_cod: value.group_cod, impart_hours: value.impart_hours };
    });
  this.sendSelectionGroups(sendData)
    .subscribe(
      next: val => this.loadData(),
      error: err => alert('ERROR al confirmar los grupos'),
      complete: () => this.loadSaveIcon = false
    );
}

private updateGroup() {
  const dataForSend = this.teacherGroupsConfirm
    .filter( callbackfn: val => val.impart_hours > 0.1)
    .map( callbackfn: val => {
      return { area_cod: val.area_cod, subject_cod: val.subject_cod, group_cod: val.group_cod, impart_hours: val.impart_hours };
    });

  this.sendUpdateGroups(dataForSend)
    .subscribe(
      next: res => this.loadData(),
      error: err => alert('Error al actualizar los datos: ' + err.message),
      complete: () => this.loadSaveIcon = false
    );
}
```

Imagen 48. Código de confirmación de la demanda docente

botón de "*guardar"*. Dado que ambos conjuntos se encuentran en el mismo array de datos la función *updateGroups* es llamado cuando se desee modificar las horas de un grupo (y es mayor que cero). La función *confirmGroup* es llamado cuando se añada nuevos grupos.

El botón de eliminar debe pulsarse una vez que alguna fila de la "*carga docente disponible*" esté seleccionada. En cuyo caso se ha activado la variable *actividChanger* que tiene como

```
private deleteGroup() {
  this.loadDeleterIcon = true; this.deleteBtnAvailable = false; this.allSelected = false;

  const listDelete = this.teacherGroupsConfirm
    .filter( callbackfn: val => val.actividChanger)
    .map( callbackfn: val => {
      return {
        area_cod: val.area_cod,
        subject_cod: val.subject_cod,
        group_cod: val.group_cod
      };
    });

  this.sendDeleteGroup(listDelete).subscribe( next: res => { this.loadData(); this.loadDeleterIcon = false; });
```

Imagen 49. Código de eliminación de grupos seleccionado

atributo cada objeto de la lista, y éste, posteriormente será invocada la función *delegreGroup* que enviará la solicitud de eliminar al servidor.

Controlador del servidor

Las solicitudes denominadas CRUD (de las siglas en inglés, Create, Read, Update, Delete) descritas anteriormente, son recogidas por el controlador del servidor como se comenta a continuación.

En la Imagen 51. Creación de la demanda docente se muestra el código que se encarga de crear la asignación entre grupo y profesor a través del método POST. El servidor debe de recibir el código de grupo, el código de asignatura y el código de área para poder referenciar a un grupo. Una vez, obtenido el grupo se creará una referencia en la tabla de impartición entre grupo y profesor, donde quedará reflejada en dicha referencia el estado (aceptado o rechazado por el administrador) y la selección de horas asignadas. La referencia es devuelta, en formato JSON, al cliente.

```python
@app.route('/teacher_load', methods=['POST'])
@token_required
def create_teachers_loads(user):
    data = request.get_json()
    if not data:
        return response({'message': 'Data not found'}, 401)
    if not user.teacher:
        return response({'message': 'User is not teacher'}, 401)

    try:
        group = Group.get(data['area_cod'], data['subject_cod'], data['group_cod'])
        if group:
            impart = Impart(group, user.teacher, str(data['impart_hours']))
            if impart.save():
                return response({'message': impart.to_dict()})
        return response({'message': 'Group not found'}, 404)

    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 51. Creación de la demanda docente

En la Imagen 50. Obtención de la demanda docente se puede apreciar el código de obtención de la carga docente de un profesor. Según la arquitectura REST, la solicitud tiene la forma de *"/teacher_load/<dni>",* donde *dni* variará según la carga docente del profesor; solo se devolverá la carga docente de dicho profesor.

```python
@app.route('/teacher_load/<dni>', methods=['GET'])
@token_required
def get_teacher_load(user, dni=None):
    try:
        list_group, teacher = [], Teacher.get(dni)
        if not teacher:
            return response({'message': 'Techer not found'}, 404)

        for impart in teacher.group:
            output = build_dict(teacher=teacher, impart=impart, group=impart.group, subject=impart.group.subject,
                                area=impart.group.subject.knowledgeArea,
                                university=impart.group.subject.university_degree)
            output.update(teacher_cover_hours(impart.group))
            list_group.append(output)
        return response({'teacher_name': teacher.name, 'teacher_surnames': teacher.surnames, 'groups': list_group})
    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 50. Obtención de la demanda docente

El código generará una lista con los grupos que tenga asignado el profesor donde se incluye información como: el profesor, la asignación entre profesor y grupo (tabla de impartición), el grupo, la asignatura, el área de conocimiento y el título universitario. Esta información es devuelta por la función build_dict() que se encuentra en el archivo *resources.py.*

En la Imagen 52. Código de actualización de la carga docente se puede ver el código de actualización de la carga docente del profesor. A este recurso se le puede pasar el *dni* del profesor o bien, (por defecto) el usuario que está autentificado.

Dado que la clave principal de la tabla de impartición es una combinación de claves

```python
@app.route('/teacher_load/<area_cod>/<subject_cod>/<group_cod>/<teacher_dni>', methods=['PUT'])
@app.route('/teacher_load/<area_cod>/<subject_cod>/<group_cod>', methods=['PUT'])
@token_required
def update_teacher_load(user, area_cod=None, subject_cod=None, group_cod=None, teacher_dni=None):
    try:
        data = request.get_json()
        if not area_cod and not subject_cod and not group_cod and not data:
            return response({'message': 'Param not found'}, 404)

        dni = teacher_dni if teacher_dni else user.teacher.dni
        impart = Impart.get(area_cod=area_cod, subject_cod=subject_cod, group_cod=group_cod, teacher_dni=dni)
        impart.update(data)
        if impart.save():
            return response({'message': 'Teacher load update successfully'})
        return response({'message': 'Teacher load update error'}, 404)

    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 52. Código de actualización de la carga docente

foráneas entre las claves (quién asigna la referencia entre profesor y grupo), es necesario pasar como parámetros el código de área, código de asignatura, código de grupo y el dni. El objeto *impart* contiene la función *update* que actualiza los datos de estado de las solicitudes (aceptado, rechazado o pendiente) y el número de horas seleccionados. Si esto tiene éxito se notificará al cliente con un estado de éxito 201, en caso contrario, un error 404.

Por último, dentro de las funciones CRUD, queda la opción *delete* o *remove* que es la encargada de eliminar la referencia entre profesor y grupo. Es por ello, en la Imagen 53. Código de eliminación de carga docente se muestra el código que recoge dicha petición con la *DELETE*. Dicha función o recurso se le debe pasar los códigos que identifica al grupo que

desea eliminar, como es el código de área, código de asignatura y código de grupo y, por defecto, se tomará al usuario que estará registrado en ese momento. En cuyo caso de tener éxito, devolverá un código de estado 201; en caso contrario, un código de estado 404.

```python
@app.route('/teacher_load/<area_cod>/<subject_cod>/<group_cod>', methods=['DELETE'])
@token_required
def delete_teacher_load(user, area_cod=None, subject_cod=None, group_cod=None):
    try:
        if not area_cod or not subject_cod or not group_cod:
            return response({'message': 'Param not found'}, 404)

        impart = Impart.get(group_cod, subject_cod, area_cod, user.teacher.dni)
        if impart.delete():
            return response({'message': 'Teacher load deleted successfully'}, 201)

        return response({'message': 'Teacher load delete error'}, 404)
    except Exception as ex:
        return response({'message': ex}, 500)
```

Imagen 53. Código de eliminación de carga docente

8.3.6 Filtros de búsqueda

El filtro de búsqueda está desarrollado con las denominadas *pipes* que son componentes especiales de *Angular.* Estos componentes realizan transformaciones de una lista en otra siguiendo un criterio. Cada uno de los bloques del filtro es una *pipe*. Por ejemplo, está *área-filter.pipe.ts* para las áreas de conocimiento (Ver **¡Error! No se encuentra el origen de la referencia.**), donde los argumentos recibidos serán los acrónimos de cada área. Así se conoce qué elemento de la lista (que serán grupos o bien profesores) permite añadir a una

```
@Pipe({
  name: 'areaFilter',
  pure: false
})
export class AreaFilterPipe implements PipeTransform {

  transform(value: any, args?: any): any {

    /**
     * ATC: Arquitectura Y Tecnología de Computadores (35)
     * CCIA: Ciencia De La Comp. E Intel. Artificial(75)
     * LSI: Lenguajes Y Sistemas Informáticos (570)
     * */

    if (args == null || (!args[0].checked && !args[1].checked && !args[2].checked) ) { return value; }

    let ATC = []; let CCIA = []; let LSI = [];

    if (args[0].value === 'ATC' && args[0].checked) {
      ATC = value.filter(data => data.area_cod === '35');
    }
    if (args[1].value === 'CCIA' && args[1].checked) {
      CCIA = value.filter(data => data.area_cod === '75');
    }
    if (args[2].value === 'LSI' && args[2].checked) {
      LSI = value.filter(data => data.area_cod === '570');
    }

    value = ATC.concat(CCIA);
    return value.concat(LSI);
  }
```

Imagen 54. Pipe área de conocimiento

lista final y cuáles no.

En el fichero hours-filter.pipes.ts se filtra por horas el conjunto de grupos (ver Imagen 55. FIltro por hora).

```
@Pipe({
  name: 'hoursFilter',
  pure: false
})
export class HoursFilterPipe implements PipeTransform {

  transform(value: any, args?: any): any {

    if (args == null || (!args[0].checked && !args[1].checked && !args[2].checked) ) { return value; }

    let uncovered = []; let cover = []; let exceeded = [];

    if (args[0].value === 'uncovered' && args[0].checked) {
      uncovered = value.filter(data => data.group_cover_hours < 0);
    }
    if (args[1].value === 'exceeded' && args[1].checked) {
      exceeded = value.filter(data => data.group_cover_hours > 0);
    }
    if (args[2].value === 'cover' && args[2].checked) {
      cover = value.filter(data => data.group_cover_hours === 0);
      console.log(cover);
    }

    value = uncovered.concat(exceeded);
    return value.concat(cover);
  }
}
```

Imagen 55. FIltro por hora

En el fichero orden-by-subject.pipe.ts *ordena* la lista de grupos por el nombre de la asignatura (ver Imagen 56. Filtro por asignatura).

```
@Pipe({
  name: 'ordenBySubject'
})
export class OrdenBySubjectPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    if (value == null) {
      return value;
    }
    value.sort((obj1: any, obj2: any) => {
      return obj1.subject_name.localeCompare(obj2.subject_name);
    });
    return value;
  }

}
```

Imagen 56. Filtro por asignatura

Search-filter.pipe.ts realiza búsquedas por titulación o asignatura (ver **¡Error! No se encuentra el origen de la referencia.**). Los argumentos son almacenados en la variable *args* (información introducida por teclado). La función *filter* devolverá sólo los valores que coinciden con el nombre de la asignatura, nombre de la titulación o su acrónimo.

```
@Pipe({
  name: 'searchFilter'
})
export class SearchFilterPipe implements PipeTransform {

  transform(value: any, args?: any): any {

    if (args == null || args === '' || args.length < 3) {
      return value;
    }

    return value.filter(val => {
      return (val.subject_name.toLowerCase().indexOf(args.toLowerCase()) > -1) ||
        (val.university_name.toLowerCase().indexOf(args.toLowerCase()) > -1) ||
        (val.university_acronym.toLowerCase().indexOf(args.toLowerCase()) > -1);
    });
  }
```

Imagen 57. Búsqueda de asignatura y profesor

En *search-teacher.pipe.ts* se realiza la búsqueda por el nombre del profesor (ver Imagen 58. Pipe de búsqueda de profesor).

```
@Pipe({
  name: 'searchTeacher'
})
export class SearchTeacherPipe implements PipeTransform {

  transform(value: any, args?: any): any {

    if (args == null || args === '') {
      return value;
    }
    const resultSubjects = [];
    for (const post of value) {
      if (post.teacher_name.toLowerCase().indexOf(args.toLowerCase()) > -1) {
        resultSubjects.push(post);
      } else if (post.teacher_surnames.toLowerCase().indexOf(args.toLowerCase()) > -1) {
        resultSubjects.push(post);
      }
    }
    return resultSubjects;
  }
```

Imagen 58. Pipe de búsqueda de profesor

Estos filtros que se encuentran en las páginas de *Demanda docente*, *Carga docente, Tutorías y Mi carga docente.* Utiliza estas pipes como filtros de búsqueda y ordenación como se puede ver en las líneas 147 y 148 de la Imagen 59. Código de tabla demanda docente disponible

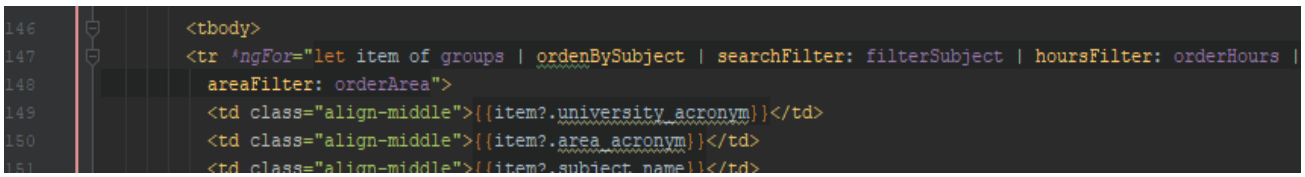


Imagen 59. Código de tabla demanda docente disponible

9. Presupuesto

El tiempo de desarrollo del proyecto es de tres meses. En base a esta estimación se ha elaborado el siguiente presupuesto.

Tabla 6. Tabla de presupuesto

Plataforma	
Portatil	Aprox. 850 € (Vida Útil 5 Años) – 42 € (3 meses)
Pycharm (Professional Developers)	50 € (3 meses)
Webstrom (Ide Javascript)	32 € (3 meses)
Base de Datos (Sqlite)	0 €
Total	124 €
Salario Desarrollador Full-Stack (Según datos del campustraining.es [14.14])	4.250 €
Total	4.374 €

Tabla 6. Tabla de presupuesto se muestra el conjunto de gastos producidos en el desarrollo de la aplicación. En ella no se ha tenido en cuenta una serie de gastos secundarios como son, electricidad, alquiler del puesto de trabajo, etc.

10. Conclusiones y trabajo futuro

Conclusión

Durante los últimos años el Departamento de Informática y Sistemas (DIS) de la ULPGC ha estado utilizando una aplicación informática para la confección y organización de la docencia asignada al departamento denominada POD. Esta aplicación cumple con su función, pero tiene una serie de limitaciones a la hora de automatizar los procesos de importación y exportación de datos por parte del administrador del sistema donde debe volcar los datos de forma manual. Otra de las limitaciones que presenta es la falta de filtros o buscadores para sintetizar o localizar ciertos datos dentro de la aplicación debido a que la actual plataforma visualiza toda la demanda docente en tablas extremadamente grandes provocando, de forma innecesaria, una visión abrumadora de la información.

La app gestión docente que se presenta aquí, pretende ser una evolución de la antigua aplicación teniendo en cuenta según el análisis previo los requisitos, inconvenientes y necesidades que presentaba el POD. Entre las características más relevantes de la nueva aplicación están (i) un nuevo diseño de la interfaz, (ii) filtros de búsqueda para sintetizar la información mostrada al profesorado, (iii) la automatización de la exportación e importación de los datos a un fichero (xlsx) para facilitar las labores de gestión del administrador, (iv) las bajas restricciones y (v) la tecnología que se utiliza en el modelo permite mayor flexibilidad de cambios frente a nuevos requisitos o cambios normativos. También cabe destacar, el sistema REST en el cual se basa la arquitectura de esta aplicación que permite ser escalable, permitiendo así añadir nuevas aplicaciones móviles nativas.

Todos los cambios introducidos hasta el momento en la app gestión docente supone un cambio bastante notable tanto para el usuario final, donde podrá disfrutar de las comodidades de una aplicación intuitiva y de fácil uso frente a su predecesora. También para el administrador, que verá reducido su tiempo de trabajo en la exportación de los datos, al estar este proceso completamente automatizado.

Al nivel personal, todo el proyecto me ha supuesto un reto sin precedentes, debido a que jamás me había enfrentado en analizar, diseñar e implementar un sistema completo de gestión, donde tenga que confeccionar un modelo de base de datos, desconociendo

completamente el sistema interno del DIS. En cuyo caso he tenido que buscar todas aquellas personas responsables de dicho sistema y que tuviera cierta idea de cómo trabajan los distintos componentes que lo forman.

Trabajo futuro

El estado actual de la aplicación permite su uso para la confección de futuros POD del DIS. Sin embargo, estamos hablando de un prototipo funcional que siempre se puede mejorar, añadiendo nuevas funcionalidades. Por lo tanto, estamos hablando de que la app gestión docente no es un producto terminado.

Un ejemplo de ello es la introducción de nuevas ideas como (i) las opciones para que el administrador pueda habilitar, según fecha y hora, la selección de grupos de impartición, horas de tutorías y coordinación del profesorado o (ii) un recordatorio de la selección realizada en los años anteriores, (iii) incorporar a la aplicación la selección de horarios de los grupos. Además, (iv) la tecnología elegida en la base de datos permitiría, de forma simple, emigrar el sistema a otros gestores de base de datos más grandes como es MySQL o MaríaDB para una mayor cantidad de datos. Sin embargo, con el actual gestor de base de datos se puede manejar convenientemente el volumen de datos de la aplicación. (v) La versatilidad y la flexibilidad del sistema donde está montado permitiría montar el servicio en un sistema de servidores distribuido.

11. ANEXO I. Instalación y ejecución de la interfaz gráfica

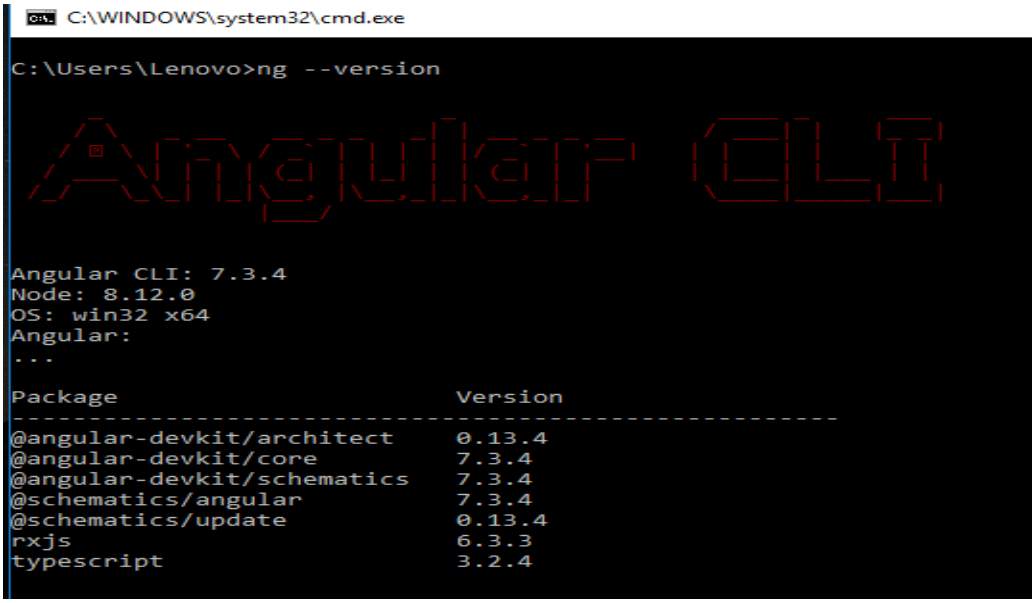
El frontend está desarrollado en *Angular 7.3.4*. Éste trabaja sobre un sistema de gestión de paquetes denominado Node.js, necesario descargar previamente para poder arrancar nuestra aplicación. La instalación y configuración se ha realizado en un entorno de Windows 10.

Paso 1. Instalar Node.js

Nos dirigimos a la página de Node.js [14.3], descargamos e instalamos la última versión de Node.js. Una vez finalizado podemos comprobar la versión que hemos instalado con el comando `npm -v`.

Paso 2. Instalar Angular Cli

Abrimos un terminal como administrador y ejecutamos el comando que encontramos en la web de angular.io [14.5] `npm install -g @angular/cli`.



Comando 1. versión de Ángular CLI

Si deseamos ver qué versión se ha instalado podemos hacerlo a través del comando `ng --version` (imagen Comando 1. versión de Ángular CLI).

Paso 3. Descargar proyecto disManager-client

Para poder usar la interfaz gráfica de la aplicación es necesario previamente descargarla. La aplicación se encuentra disponible desde un repositorio de GitHub [14.1]. Una vez descargado, si lo hemos hecho en formato zip, deberemos de descomprimirlo.

Paso 4. Instalar los paquetes y dependencias

Para instalar los paquetes y dependencias, primero debemos navegar al directorio donde hemos descomprimido nuestro proyecto utilizando un terminal. Una vez en dicho directorio, ejecutamos el siguiente comando: `npm install.` Esto instalará de forma sencilla y automática de todas y cada una de las dependencias gracias al gestor de paquetes.

Paso 5. Ejecutar la aplicación

Una vez finalizado, ya estamos preparado para lanzar nuestra aplicación. Desde el terminal ejecutamos la siguiente orden `ng serve –o`, mostrando la siguiente Imagen 60. Página de acceso.

Si no se abre de forma automática, se puede acceder a través de la dirección localhost:4200. Ésta será la dirección local del servidor de Angular Cli.

Imagen 60. Página de acceso

Con todos estos pasos finalizados la interfaz gráfica estará preparada para ser usada, pero esta no funcionará completamente si no tenemos ejecutándose el backend.

12. ANEXO II. Instalación y ejecución del Servidor

Para obtener los recursos de la base de datos es necesario tener descargado, instalado y ejecutándose el backend de la aplicación. El backend está desarrollado con tecnología Python 3 y Flask, por ello, es necesario instalarlo. El sistema operativo para la cual está explicada esta guía es Windows 10.

Paso 1. Descargar e instalar Python

Desde la página oficial de Python [14.4] podemos descargar el ejecutable de Python 3.7.2. La Imagen 61. Página web de descarga de Python 3.7 puede servir de referencia para localizar la versión, marcada en un cuadro amarillo y el enlace con una flecha negra.

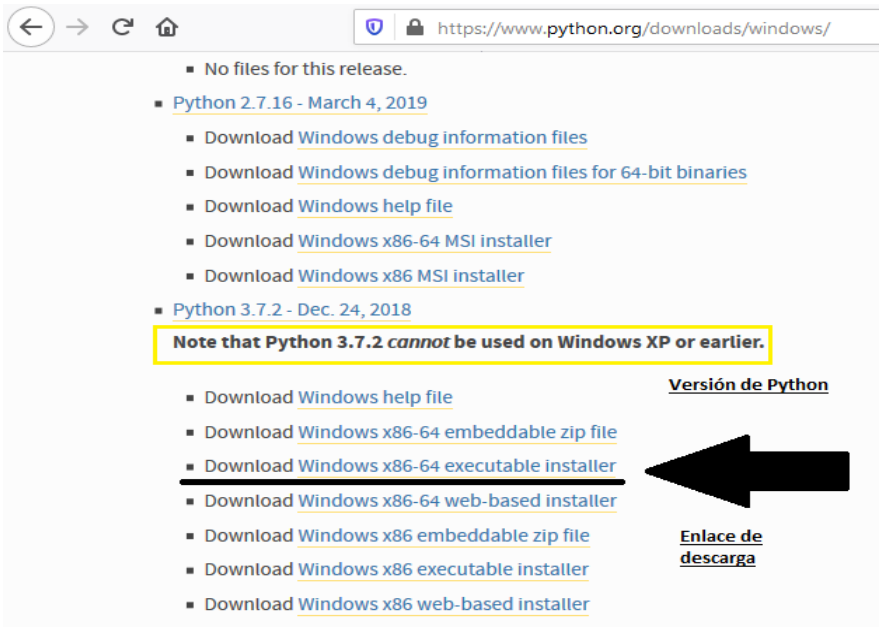


Imagen 61. Página web de descarga de Python 3.7

Una vez descargado, debemos de ejecutarlo como administradores y nos saldrá una imagen como la que se ve en la Imagen 62. Ejecutable de Python 3.7, donde lo primero que habrá que hacer es añadir la ruta del ejecutable (flecha 1) y luego instalar (flecha 2).

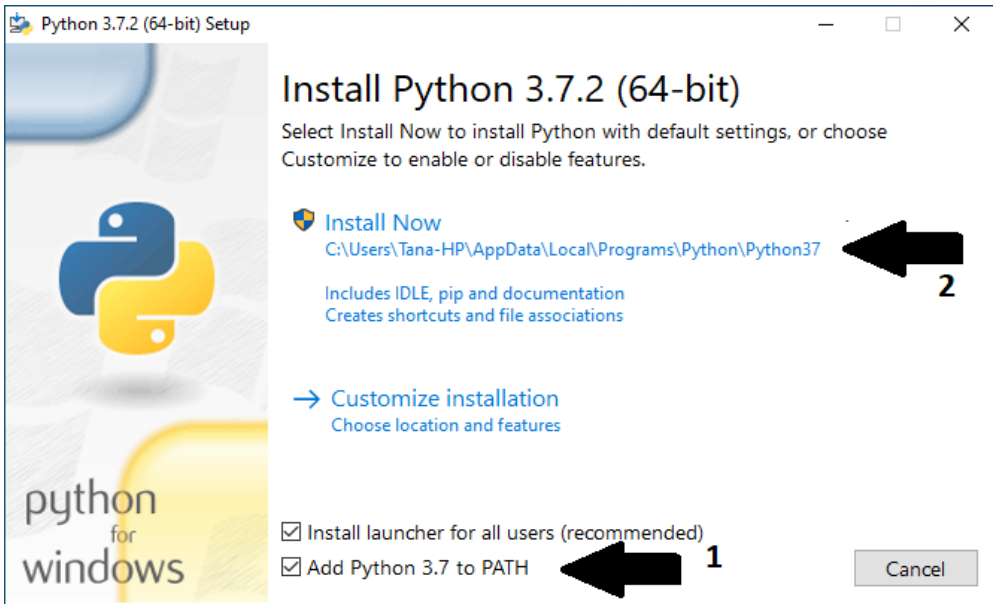


Imagen 62. Ejecutable de Python 3.7

Paso 2. Descargar el proyecto disManager-server

Descargamos el backend que se encuentra disponible desde el repositorio de GitHub [14.2]. Una vez descargado, si lo hemos hecho en formato zip, deberemos de descomprimirlo.

Paso 3. Instalación de Flask

Desde un terminal, navegamos hasta la carpeta que hemos descomprimido anteriormente y comenzamos la instalación de Flask, así como algunas librerías más para poder comenzar a utilizar el backend.

Para realizar la instalación de Flask, sólo es necesario ejecutar la siguiente orden como administrador en el terminal que tenemos abierto, `pip install flask`. Esto apenas tardará algunos segundos.

Paso 4. Instalación de las librerías

Ahora debemos realizar la instalación de los paquetes necesarios de los que depende la aplicación con el siguiente comando `pip install flask-sqlalchemy flask-migrate openpyxl flask-cors Flask-JWT`.

A continuación, se muestra en la Tabla 7. Descripción de dependencias python3 para qué sirve cada paquete.

Tabla 7. Descripción de dependencias python3

Paquete	*Descripción*
Flask-sqlalchemy	Esta librería está enfocada para interactuar con la base de datos relacional a través de modelos, proporcionando así diversas herramientas, la cual permite una interfaz genérica e independiente del gestor de base de datos. Mapeador entre objetos y transacciones (ORM). [14.6]
Flask-migrate	Extensión de Flask para manejar los cambios en el modelo de SQLAlchemy [14.7].
Flask-cors	*"Este paquete es una extensión de Flask que, de forma predeterminada, habilita el soporte CORS en todas las rutas, para todos los orígenes y métodos. Permite la parametrización de todos los encabezados CORS en un nivel por recurso. El paquete también contiene un decorador, para aquellos que prefieren este enfoque"* [14.8].
Flask-JWT	Esta extensión de Flask se utiliza para manejar el token de usuario, como modo de autentificación. (JWT – JSON Web Token) [14.9].
openpyxl	Librería Python para poder leer y escribir archivos Excel.

Paso 5. Ejecutar servidor

Con todos estos pasos realizados sólo quedan dos pasos más. El primero es indicar al sistema operativo con la variable de entorno FLASK_APP donde se encuentra la aplicación. En el caso de utilizar Windows el comando es `set FLASK_APP=dismanagerapp.py`. Si estamos en otro entorno distinto de Windows sustituir 'set' por 'export'. Ya habiendo hecho

esto, queda arrancar el servidor flask con el comando flask run. Éste nos responderá con un mensaje similar al que nos encontramos en la imagen Comando 2. flask run.

```
C:\Users\Tana\Desktop\disManagerApp-server-master>flask run
 * Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
 * Serving Flask app "dismanagerapp.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Comando 2. flask run

13. ANEXO III. Guía de usuario

En el siguiente apartado se presentará una guía de uso para el usuario donde se le enseñará a utilizar la aplicación.

Presentación PowerPoint (.pptx)

Se dispone de una versión extendida de esta guía de usuario con ilustraciones y explicaciones más detalladas en OneDrive, donde se puede acceder a través del siguiente enlace: https://alumnosulpgc-my.sharepoint.com/:p:/g/personal/tanausu_suarez101_alu_ulpgc_es/ESMmGtZtThlHgQzK7rKGRCcBLPa86MXkxzGTEP1KkQ8rNQ?e=QeNXfg

O bien, en versión PDF: https://alumnosulpgc-my.sharepoint.com/:b:/g/personal/tanausu_suarez101_alu_ulpgc_es/EdeBXkBMyW5EsoCpzftD678BuhVz_ykG9HjKEhFO_YLOYQ?e=Cxn8gK

Acceso

Para acceder a la aplicación debemos introducir en nuestro navegador la dirección del nuestro servidor, localhost:4200. Por defecto ésta será la dirección del servidor de Angular (*ver cómo iniciar servidor [11] y cliente [12]*).

Una vez accedamos a la plataforma se mostrará la página de acceso, Imagen 60. Página de acceso. Puntos a tener en cuenta:

1. Se requiere de un usuario y contraseña.
2. El DIS será el encargado de registrar a los usuarios en la plataforma.
3. Si eres administrador, existe por defecto un usuario denominado “admin” y contraseña “admin”.
4. En el caso de ser profesor, el usuario por defecto será el DNI.

Estructura de la plataforma

La plataforma está dividida en tres bloques diferentes:

1. Mi cuenta
2. Administración
3. POD

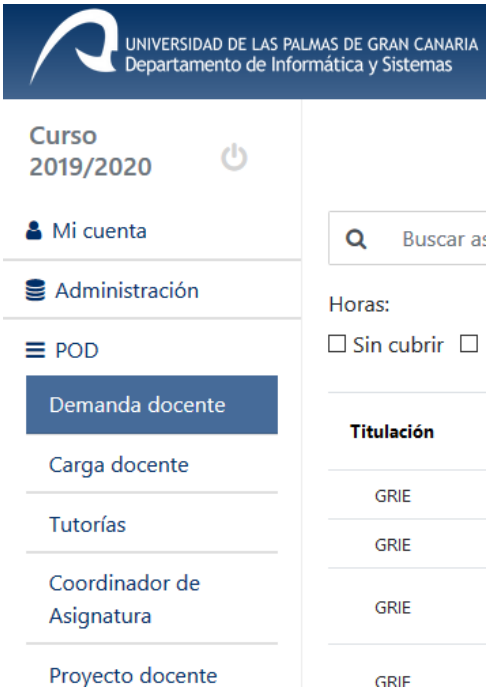


Imagen 63. Bloques del menú de navegación

En el lateral izquierdo se desplegará un menú para poder acceder a los distintos bloques. En el caso que fuera necesario por requisitos de espacio de la pantalla, desaparecerá y en su lugar aparecerá un botón en la cabecera para poder desplegarlo. Para ver más detalles de este comportamiento ir a 8.2.2.

Todas aquellas páginas que contengan filtros de búsqueda por áreas de conocimientos estarán marcadas por defecto si el usuario que ha accedido es un profesor.

Bloque común

El bloque POD es la parte común solamente de lectura y servirá para dar información a todos los usuarios -Ver Imagen 63. Bloques del menú de navegación-. Este bloque cuenta con cinco páginas principales, las cuales son:

1. Demanda docente

En la página *demanda docente* aparecerá el listado de grupos que el profesor tiene disponible, así que, podemos acceder con mayor detalle a la información de un grupo sobre él.

2. Carga docente

La *carga docente* muestra un listado del profesorado registrado en la plataforma, indicando detalles como áreas de conocimientos, potencial y horas cubiertas. Al igual que la demanda docente, cliqueando encima de algún profesor podemos acceder a mayores detalles.

3. Tutorías

En la página de *tutoría* podemos ver el listado de profesores que existe con tutorías seleccionadas. Cliqueando sobre algún profesor para ver su horario de tutorías.

4. Coordinadores de asignatura

En la página *Coordinadores de asignaturas* podemos obtener la información de quién es el profesor coordinador y el responsable de práctica de las asignaturas de la plataforma.

5. Proyectos docentes (PDA)

La página *PDA* muestra un listado de proyectos docentes subidos a la plataforma donde se puede visualizar el estado en el que se encuentran los proyectos docentes de cada una de las asignaturas.

Bloque del administrador

El bloque Administración es la parte exclusiva para el administrador del sistema. Aquí el administrador – ver Imagen 26. Página de administración de BD - podrá realizar gestiones como:

1. Importar archivos Excel (listado del profesorado, esquema de la base de datos y listado de PDA) -ver 8.1.4-.
2. Exportar el POD a un archivo Excel.
3. Eliminar el contenido de la base de datos.
4. Gestionar los usuarios de la plataforma (añadir, eliminar, editar y visualizar).
5. Aceptar, rechazar o visualizar las solicitudes de impartición de docencia.

Puntos importantes a tener en cuenta en este bloque:

1. La importación del fichero debe tener la extensión ***.xlsx***
2. Hay que tener en cuenta que los proyectos docentes y el profesorado dependen de su área de conocimiento y de las asignaturas. Por tanto, los archivos de importación del profesorado, como el del proyecto docente, no se podrán cargar hasta que no se importe el esquema de la base de datos.

Bloque del perfil del usuario

El bloque ***Mi cuenta*** es la sección dedicada al profesor – Ver Imagen 64. Bloque de la cuenta del usuario-, a excepción del ***perfil*** donde todos los usuarios podrán cambiar su contraseña de acceso.

El profesorado podrá realizar labores propias como son:

1. Seleccionar el número de horas que desea impartir en los grupos disponibles, en la sección *mi carga docente*.
2. Seleccionar la coordinación de las asignaturas o responsable de prácticas, en *mi coordinación.*
3. Seleccionar el horario de tutorías en la sección *mis tutorías* a través de una ventaja emergente clicando en editar – ver Imagen 65. Ventana emergente para editar tutorías.-
4. Visualizar la carga docente y su estado en la sección *mi solicitud*.

Imagen 64. Bloque de la cuenta del usuario

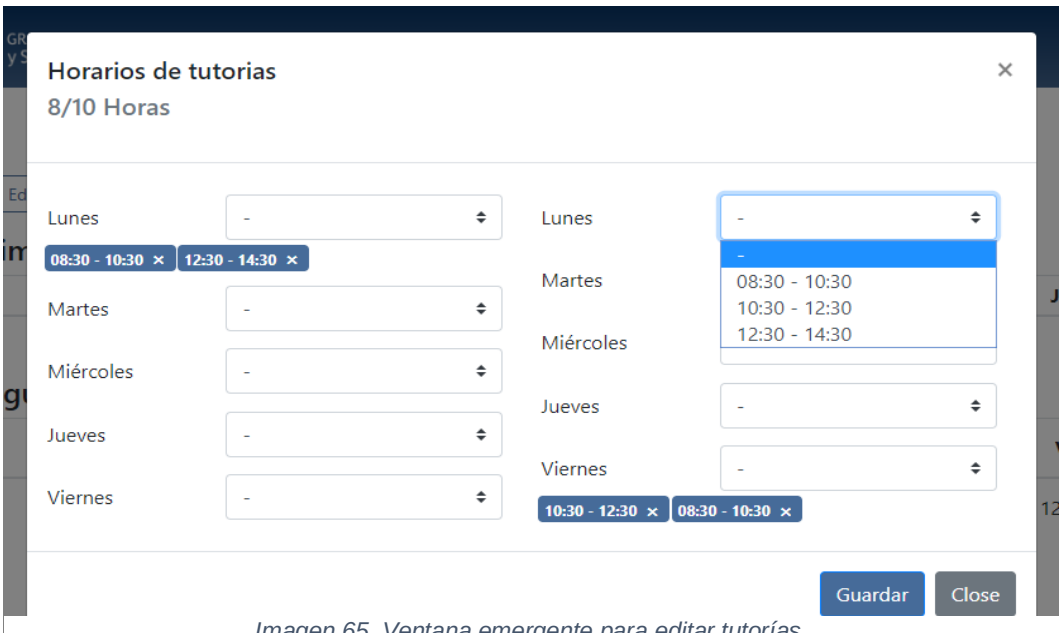


Imagen 65. Ventana emergente para editar tutorías.

14. Referencias

14.1 GitHub - frontend [21/09/19]
https://github.com/tanausuarez101/disManagerApp-client/

14.2 GitHub – backend [21/09/19]
https://github.com/tanausuarez101/disManagerApp-server/

14.3 Node.js [21/09/19]
https://nodejs.org/es/download/

14.4 Python 3.7.2 [21/09/19]
https://python.org/downloads/

14.5 Ángular Cli [22/09/2019]
https://angular.io/guide/setup-local

14.6 SQLAlchemy [22/09/2019]
https://docs.sqlalchemy.org/en/13/

14.7 Flask-migrate [22/09/2019]
https://flask-migrate.readthedocs.io/en/latest/#

14.8 Flask-cors [22/09/2019]
https://flask-cors.readthedocs.io/en/latest/

14.9 Flask-JWT [22/09/2019]
https://pythonhosted.org/Flask-JWT/

14.10 Sistema REST [27/09/2019]
https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html

14.11 Sistema de autentificación [29/09/2019]
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins

14.12 Competencias GII [02/10/2019]
https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf

14.13 Conexión a otras DDBB [08/10/2019]
https://docs.sqlalchemy.org/en/13/core/engines.html#database-urls

14.14 Campusstraining.es, salario desarrollador web [22/11/2019]
https://www.campustraining.es/noticias/sueldo-desarrollador-web/

14.15 Modelo entidad-relación [22/11/2019]
https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n