Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Universidad de Jaén

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN WEB PARA LA CREACIÓN DE LOS HORARIOS DE LAS TITULACIONES DE LA UNIVERSIDAD DE JAÉN

Alumno: Ignacio Beltrán Mata

Tutor: Prof. D. Juan Carlos Cuevas Martínez

Depto.: Ingeniería de Telecomunicación

Junio, 2024

Universidad de Jaén

Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN WEB PARA LA CREACIÓN DE LOS HORARIOS DE LAS TITULACIONES DE LA UNIVERSIDAD DE JAÉN

Alumno: Ignacio Beltrán Mata
Tutor: Prof. D. Juan Carlos Cuevas Martínez
Depto.: Ingeniería de Telecomunicación

Firma del autor

BELTRAN MATA IGNACIO - 77384018N

Firmado digitalmente por BELTRAN MATA IGNACIO - 77384018N
Fecha: 2024.06.22 12:44:46 +02'00'

Firma del tutor

CUEVAS MARTINEZ JUAN CARLOS - 26038239P

Firmado digitalmente por CUEVAS MARTINEZ JUAN CARLOS - 26038239P
Fecha: 2024.06.22 17:02:05 +02'00'

Índice

Indice de Figuras

Indice de Tablas

Resumen

La gestión de turnos u horarios en la mayoría de las instituciones suele ser un tema complejo y que abarca demasiadas casuísticas en su elaboración. Normalmente se requieren una serie de recursos y tiempos para que su consecución se realice de forma correcta, es por ello por lo que surge una necesidad a la hora de facilitar este proceso. Este Trabajo de Final de Grado trata de solventar esta necesidad.

A lo largo de este documento se describe el planteamiento, estudio, análisis y desarrollo para el despliegue de una aplicación web que tenga como cometido la facilitación del proceso de creación horaria. Esta plataforma será objeto de la gestión horaria de la Universidad de Jaén, la cual permitirá tanto la creación como la consulta de los horarios de las diferentes titulaciones que conforman el plan docente de dicha universidad.

La aplicación web permitirá la creación y modificación de los diferentes horarios y turnos que los conforman, con las restricciones propias de tiempo y espacio de los recursos disponibles. Además, gestionará de forma simple y efectiva esos recursos, proporcionando al usuario final una experiencia clara y de fácil accesibilidad para la consecución del objetivo.

Abstract

The schedule management tends to be a complex issue for most institutions. It covers many casuistic in his elaboration. Normally, a series of resources and time are required for its proper implementation, this is the reason why there is a need to facilitate this process. This Final Degree Project aims to address this need.

Throughout this document, the approach, study, analysis, and development for the deployment of an assisted scheduling web application are described. This platform will be used for the scheduling management of the University of Jaén, which will allow both the creation and consultation of the schedules of the different university degrees.

The web application will allow us to create and modify the different schedules with the inherent time and space restrictions of the available resources. Furthermore, it will manage these resources simply and effectively, providing the user with a clear and easily accessible experience to achieve the objective.

1 Introducción

Este Trabajo Fin de Grado (TFG) detalla el desarrollo de una aplicación web que busca satisfacer la necesidad de facilitar el proceso de gestión horaria en la Universidad de Jaén. Esta plataforma permitirá la creación, modificación y consulta de los documentos horarios pertenecientes a las diferentes titulaciones que ofrece la Universidad de Jaén. Se tratará de gestionar de forma eficiente las principales problemáticas de solapamiento y gestión de recursos que surgen al abarcar un proyecto de esta envergadura.

A lo largo de este apartado se explica de forma detallada el contexto, motivación, objetivos y estructuración que seguirá el presente documento.

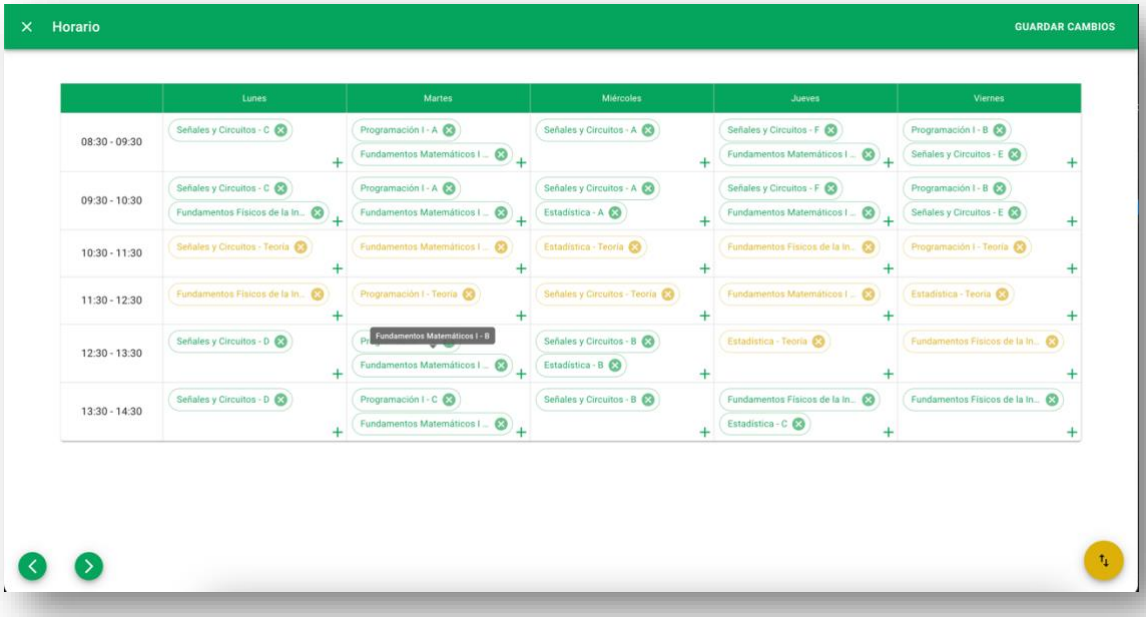


Ilustración 1: Ejemplo de la aplicación de gestión horaria

1.1 Contexto y Motivación

Actualmente, la planificación horaria se realiza de forma manual en la Universidad de Jaén. En cada curso académico se reúnen recursos y efectivos para llevar a cabo esta labor. Este hecho genera una necesidad, puesto que estos esfuerzos podrían verse reducidos con la existencia de una herramienta para este proceso.

La principal motivación en la elaboración de este Trabajo Fin de Grado es la consecución de todos los procesos que conllevan la gestión horaria de manera asistida, teniendo en cuenta todos los recursos y casuísticas que esto conlleva, además de la generación automática de la documentación para su consulta.

De esta manera, el proceso de definición de turnos y horarios en la transición de cada curso académico, que antes era una tarea compleja y prolongada, ahora se produciría de forma más breve, consistente y amigable.

1.2 Objetivos

El presente TFG tiene como objetivo principal el desarrollo de un portal web que permita la gestión de los horarios de clase de las titulaciones de un centro de la Universidad de Jaén. Estará pensado para que los responsables de las titulaciones puedan confeccionar los horarios a partir de los datos del Plan de Organización Docente (POD) de cada curso, controlando la disponibilidad horaria de cada profesor. Los objetivos concretos son los siguientes:

- Desarrollar una aplicación web que permita crear y editar los horarios de clase para las titulaciones de un centro de la Universidad de Jaén.
- La aplicación deberá contar con las siguientes funciones mínimas:
  - Acceso autenticado.
  - Creación de los roles de administrador (gestión de usuarios y publicación de horarios) y editor (creación/edición de horarios).
  - Herramienta de importación de datos del Plan de Organización Docente de la Universidad de Jaén.
  - Capacidad de revisión y modificación de los datos importados.
  - Importación a través de autenticación con SIDUJA (opcional).
  - Creación de horarios a través de una herramienta gráfica que controle si el grupo de la asignatura está ya colocado y de si el responsable del mismo tiene docencia a esa misma hora en alguna otra titulación o curso para ese mismo cuatrimestre y curso.
  - Guardado de los horarios como documentos, con capacidad de edición, renombrado y borrado.
  - Visualización de un horario publicado como HTML a través de un enlace permanente.
  - Exportación a CSV/Excel de un horario concreto.

1.3 Estructura de la Memoria

El presente documento se estructura en nueve secciones más bibliografía y anexos. La sección actual detalla una breve introducción sobre el proyecto, indicando el contexto previo y objetivos a lograr para su finalización. Estado del Arte recoge una visión general de aplicaciones y/o plataformas que hacen una función similar a la aplicación web. La

sección de Metodología detalla la planificación previa y las herramientas y/o tecnologías a usar durante el desarrollo. En Análisis y Diseño se especifica la estructuración de datos, las relaciones entre ellos y la documentación necesaria para comenzar el desarrollo. Implementación recoge el desarrollo tanto *Front-End*[1] como *Back-End*[2] de la aplicación en detalle. Pruebas y validación explica el proceso de pruebas para comprobar el correcto funcionamiento de la plataforma. En el capítulo de Resultados se muestra de forma visual el resultado de todo el trabajo previamente descrito a modo de interfaz de usuario. Presupuesto detalla el desglose horario que se ha utilizado para cada etapa. Por último, en el apartado de Conclusiones y trabajo futuro se hace un resumen general de todo el proceso de desarrollo y lo que se ha aprendido durante éste, además de posibles mejoras que puedan aplicarse a la plataforma.

[1] *Parte del desarrollo que engloba la interfaz y experiencia de usuario que puede observarse en un navegador web, normalmente definido como frontal.*

[2] *Parte del desarrollo que engloba la lógica de la aplicación. Consiste en una serie de acciones que se efectúan en paralelo al frontal ofreciéndole funcionalidades.*

2 Estado del Arte

Es importante conocer plataformas existentes que logren un efecto similar al que se está buscando con la aplicación web, para así poder estudiarlas y que sirva como punto de partida para realizar un buen análisis. Esta plataforma satisface una necesidad directa, por lo que no existen aplicaciones similares con la que compararla dentro del ámbito de la Universidad de Jaén.

Sin embargo, se puede tomar como referencia otro tipo de plataformas conocidas que satisfacen necesidades similares. A continuación, se procede a citar algunas de ellas y ver los puntos que tienen en común.

2.1.Aplicaciones de Calendario

En términos de plataformas de gestión horaria, las aplicaciones de calendario son las más populares que presentan algunas similitudes. Por ejemplo, la plataforma *Calendar* de *Google* es una de las aplicaciones de calendarios más usada, esta agrupa en celdas los diferentes eventos que conforman el día a día. Estas celdas están segmentadas por días de la semana y tramos horarios. Los eventos se van superponiendo en las celdas y conforman una especie de horario, muy similar a lo que se quiere lograr con la aplicación web.

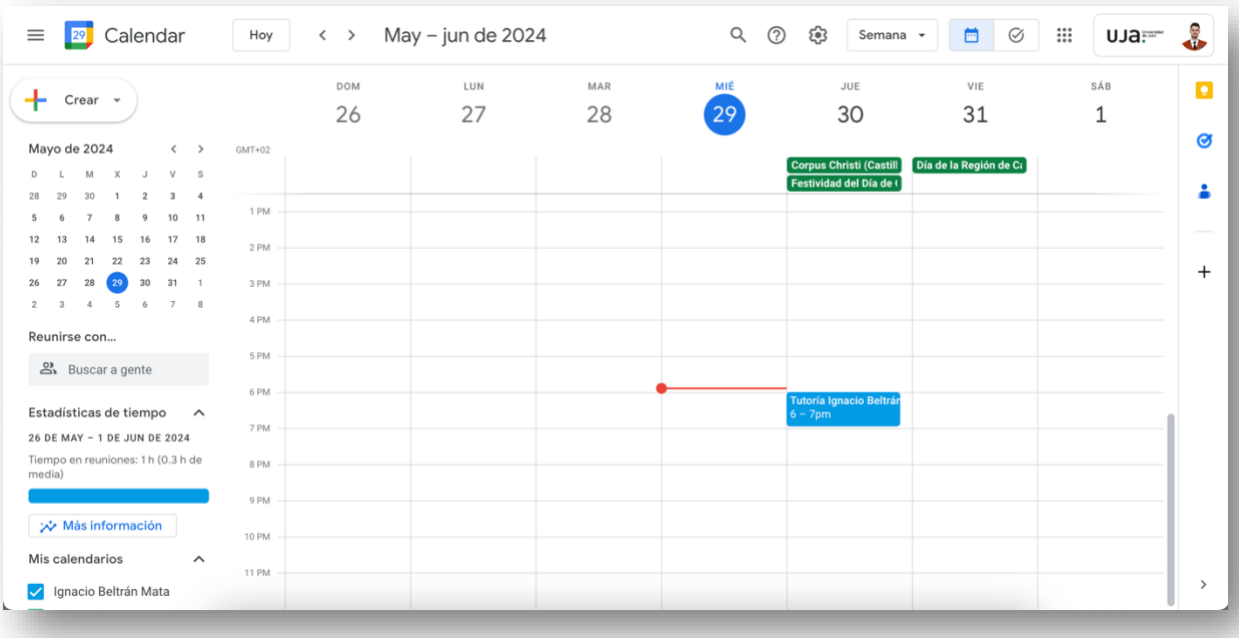


Ilustración 2: Interfaz de Google Calendar

Otra alternativa es la plataforma *Calendar* que ofrece *Microsoft Outlook*. Esta posee un funcionamiento muy similar a *Google Calendar*, pero con una vista más enfocada al mes. Dentro de cada día permite agregar eventos y se habilita una vista dedicada en la parte derecha para visualizar todos estos eventos. En términos generales, son aplicaciones con una

amplia usabilidad que pueden ser compatibles para la gestión horaria, aunque sin contar con funcionalidades específicas que aporta una aplicación web dedicada exclusivamente a este cometido.

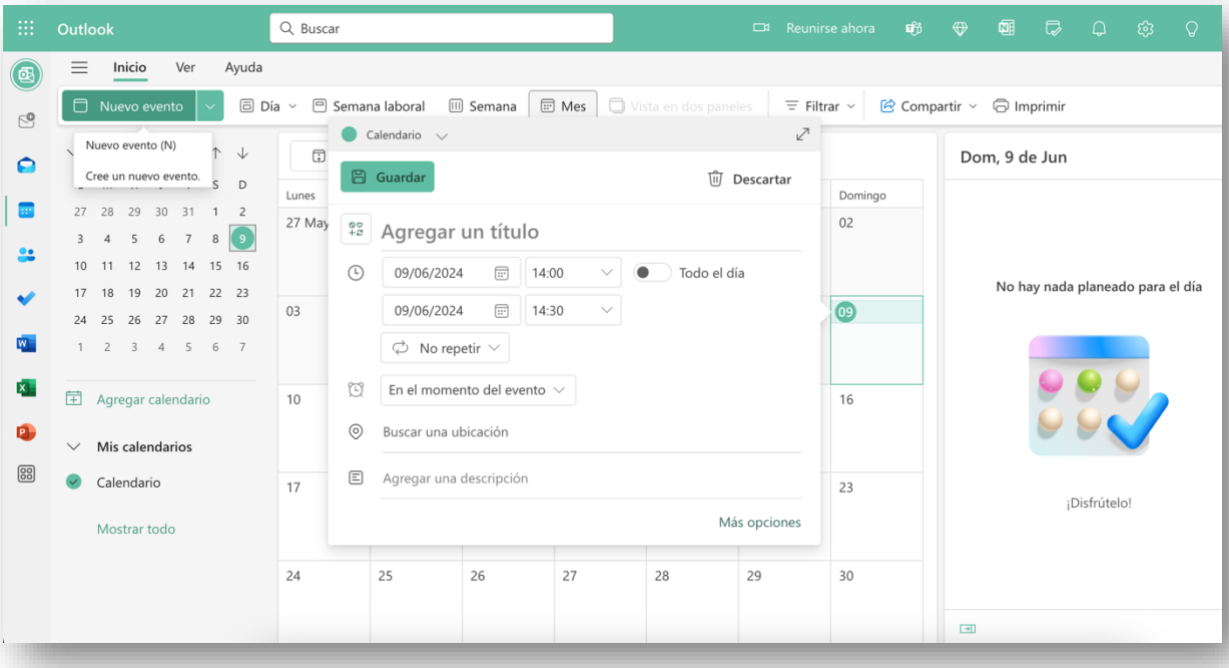


Ilustración 3: Interfaz de Microsoft Outlook Calendar

3 Metodología

Todo desarrollo software debe contar con una metodología que marque los pasos a seguir para alcanzar el objetivo de manera organizada. Existen multitud de metodologías en el desarrollo software, cada una con sus ventajas e inconvenientes, dependiendo de las características, objetivos y envergadura del proyecto. Gracias a la implementación de técnicas y métodos se puede comenzar a trabajar de forma más eficiente y con una serie de pautas a seguir. El hecho de no poseer una metodología clara de trabajo puede desembocar en tiempo adicional en el desarrollo y un proceso mucho más complejo en su totalidad.

En este proyecto se han seguido metodologías **tradicionales** debido a que no se cuenta con un grupo de trabajo y el alcance del proyecto para un único desarrollador debía ser planificado en su totalidad al inicio de éste, para así evitar retrasos en el desarrollo. Se han diferenciado cinco etapas principalmente a la hora del desarrollo, éstas son *Planteamiento, Análisis, Diseño, Programación y Pruebas*. (*Metodologías de Desarrollo de Software: ¿Qué Son?*, 2020)

Ilustración 4: Metodologías tradicionales vs Metodologías ágiles (https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html)

Las últimas cuatro etapas serán explicadas en detalle en secciones posteriores mientras que el planteamiento de este proyecto se presenta a continuación.

3.1 Planteamiento

El primer paso una vez conocidos los objetivos y el alcance del proyecto es **plantear** cómo se va a lograr completar con éxito todos los requerimientos en el plazo fijado. El primer paso es conocer las diferentes tecnologías que se tienen como alternativas a la hora de programar. Será necesario escoger un lenguaje de programación, un entorno de desarrollo o *IDE*[3] *(Integrated Development Environment)* en el que llevar a cabo el desarrollo y un marco de trabajo o *Framework*[4] si fuese necesario. Este planteamiento será necesario tanto para la interfaz de usuario, o *Front-End*, como para el lado del servidor, o *Back-End*. Además, será necesario definir cómo se gestiona el almacenamiento de datos en su correspondiente base de datos.

Las principales tecnologías *Front-End* que se demandan en el mundo laboral son *React* y *Angular*, por lo que se deben poner en contexto pros y contras para la elección de una de ellas. Respecto al *Back-End* se plantean dos de los *Framework* más populares actualmente, los cuales son *Django* y *Flask*.

Por lo que, en esta primera etapa de **Planteamiento**, se basa en investigar todas estas tecnologías para conocer cuál se adapta mejor a las necesidades del presente TFG y tomar una decisión en base a ello.

3.2 Tecnologías y Herramientas

En este apartado se explican las tecnologías y herramientas utilizadas en este proyecto haciendo hincapié en las más relevantes para el análisis y desarrollo de este:

3.2.1 Visual Studio Code

El entorno de desarrollo que se utiliza durante el proceso de codificación de este TFG será Visual Studio Code. Es un *IDE* ligero y versátil, capaz de soportar diferentes lenguajes. La principal razón por la que se elige el uso de este entorno es por la posibilidad de aunar el desarrollo *Front-End* y *Back-End* bajo este mismo *IDE*. También cuenta con una serie de extensiones para multitud de lenguajes, que ayudan y facilitan la labor de programación.

Dentro de las extensiones más utilizadas para este proyecto para *Python*[5] se tienen *Pylance* y *Python Debugger*, las cuales proporcionan soporte avanzado para la codificación

---

[3] *Hace referencia a un entorno de desarrollo, un programa cuya finalidad es llevar a cabo la programación de código con herramientas útiles para el desarrollador.*

[4] *Conjunto de conceptos y prácticas que se aplican para desarrollar software de forma más eficiente.*

[5] *Lenguaje de programación en el que está basado el Back-End.*

en *Python*, como autocompletado, comprobación de errores, verificación en tiempo real, así como herramientas de depuración, puntos de ruptura y ejecución paso a paso, entre otras.

Para la programación en *React* se tienen extensiones como *ESLint*, para la identificación y corrección de errores en el código *JavaScript*[6]. Además de *Prettier*, un formateador de código para hacer uso de buenas prácticas e implementar código limpio y legible. (*Documentation For Visual Studio Code*, s.f.)

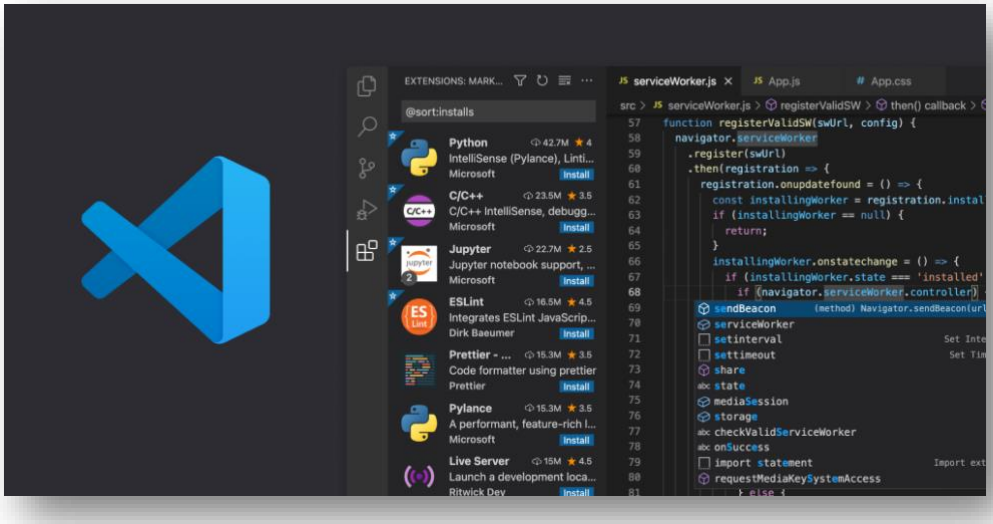


Ilustración 5: Visual Studio Code (https://code.visualstudio.com/)

3.2.2 React

React es una biblioteca de *JavaScript*, creada para simplificar el desarrollo de interfaces de usuario. La parte del *Front-End* de la aplicación web será codificada en esta tecnología, por lo que se basará en el principio **SPA** (*Single Page Application*) que usa *React* para el renderizado, realizando un enfoque sobre una única página que va mutando a través de los cambios y la navegación. Esto es posible gracias a su rápida optimización gracias a su **VDOM** (*Virtual DOM*), un concepto de programación que trata de representar idealmente el *DOM*[7] (*Document Object Model*) antes de aplicar realmente las modificaciones. La programación en esta tecnología está orientada a componentes, que serán reutilizados a lo largo de la aplicación web como podrá verse más adelante y así facilitar el desarrollo. Para el uso de esta librería se debería conocer y dominar las tecnologías web clásicas como

[6] *Lenguaje de programación empleado en el Front-End.*

[7] *Interfaz de programación para HTML y XML que permite la representación estructural de un documento en estas tecnologías.*

HTML[8] *(HyperText Markup Language)*, *CSS*[9] *(Cascading Style Sheets)* y *JavaScript*, ya que *React* hace uso de ellas.

El lenguaje que utiliza esta tecnología se denomina **JSX**, se define como una mezcla entre *HTML* y *JavaScript* puro con alguna peculiaridad. Al utilizar *React* se hace uso de *NodeJS* y *NPM* para el despliegue en desarrollo y la gestión de paquetes y dependencias del proyecto. (*React*, s. f.)

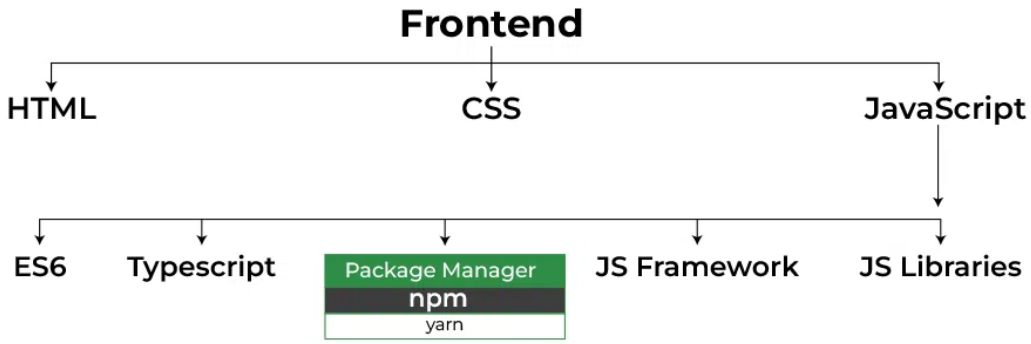


Ilustración 6 - Esquema del Front-End (https://www.geeksforgeeks.org/node-js-npm-node-package-manager/)

Debido a su popularidad en el mercado laboral, se pensaba afrontar el *Front-End* con tecnologías como *React* o *Angular*. La elección finalmente recaló en el uso de *React* en lugar de otras librerías o *Frameworks* debido a una serie de características que se definen a continuación.

- El rendimiento y renderizado es muy rápido gracias al **DOM Virtual** que se ha mencionado anteriormente, esto hace que la optimización usando React sea notablemente superior.
- La reusabilidad de código es un aspecto muy importante, gracias a sus **componentes** facilita un desarrollo sencillo y ahorra tiempos al codificar.
- Su lenguaje **JSX** permite incluir en un mismo archivo sentencias *JavaScript* con *HTML*, esto permite una claridad a la hora de leer e interpretar el código que *Angular* no ofrece.

[8] *Lenguaje basado en marcas que define la sintaxis de las páginas web.*

[9] *Denominadas 'Hojas de Estilo en Cascada' son complementarias a HTML para aplicar estilos sobre páginas web.*

- Al ser una librería y no un *Framework* como tal, *React* permite de forma flexible y con total libertad adaptar el proyecto con las librerías y/o herramientas que más se ajusten a los objetivos o necesidades.
- El flujo de datos que sigue la aplicación en *React* es **unidireccional**, lo que significa que se transfieren en una única dirección, de componentes padre a hijos a través de *props*[10], esto hace que la arquitectura de la aplicación se simplifique respecto a un enfoque bidireccional. (Powell, 2023)
- La complejidad de un *Framework*, como puede ser *Angular*, presenta una curva de aprendizaje más pronunciada, lo cual puede presentar mayores tiempos para obtener un resultado similar.

3.2.3 Django

Django es un *Framework* de alto nivel enfocado al desarrollo web. Este marco de trabajo hace uso del lenguaje *Python* para el desarrollo. La parte del *Back-End* de la aplicación web será desarrollada usando esta tecnología. Un proyecto *Django* engloba una serie de aplicaciones, cada aplicación se basa en una arquitectura **Modelo-Vista-Controlador**[11] (*MVC*). Gracias a este enfoque, *Django* permite definir el modelo que tomarán los datos en la base de datos, la funcionalidad y la vista de usuario de forma totalmente separada y así facilitar el desarrollo. (*Documentación de Django*, s. f.)

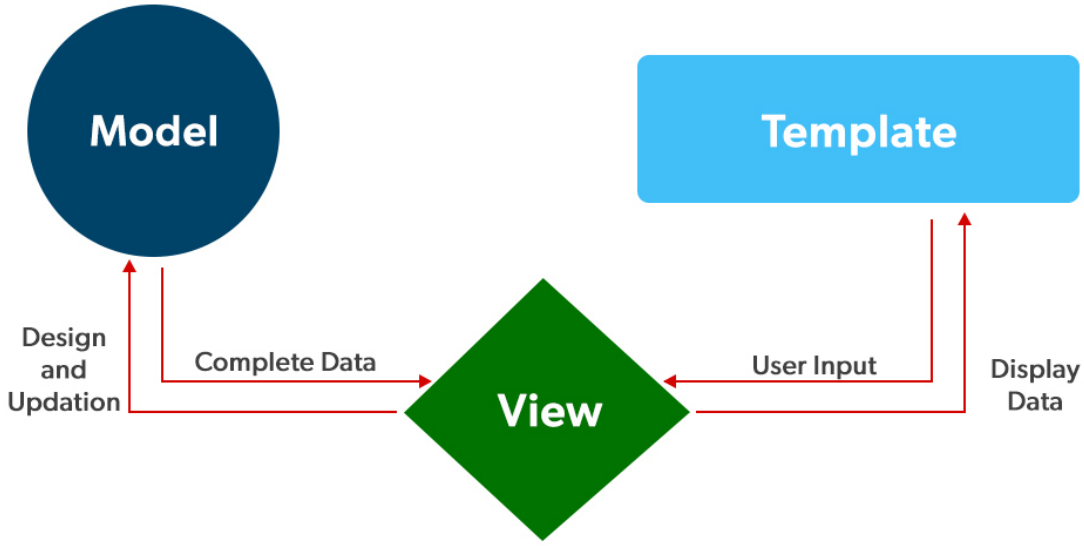


Ilustración 7 - Arquitectura MVC (https://www.geeksforgeeks.org/python-web-development-django/)

[10] *Argumentos que se pasan de componentes padres a hijos en React.*

[11] *Patrón de desarrollo software comúnmente utilizado para definir interfaces, lógica y datos.*

A la hora de elegir un *Framework* para el desarrollo del *Back-End* hay que prestar atención a la escalabilidad que brinda el mismo, en este caso *Django* y *Flask* eran las principales opciones. A continuación, se detallan las principales características que decantaron la utilización de esta tecnología.

- *Django* brinda herramientas **ORM**[12] (*Object-Relational Mapping*), que ayudan en la interacción de la lógica de programación con los datos presentes en la base de datos sin necesidad de manejar sentencias *SQL*.
- A la hora de manejar grandes cantidades de datos es muy importante la visibilidad que se tiene de los mismos al momento del desarrollo, es por eso por lo que *Django* ofrece una vista de administrador que permite monitorizar los datos presentes en todas las aplicaciones del proyecto.
- La seguridad es un aspecto para tener en cuenta en aplicaciones expuestas a la red, es por eso por lo que *Django* ofrece protección ante inyecciones *SQL*[13], ataques *XSS*[14] (Cross-Site Scripting), *CSRF*[15] (*Cross-Site Request Forgery*) y *Clickjacking*[16].
- *Django* es compatible con la utilización de *Django REST Framework* que brinda la serialización de los datos y el acceso a los mismos desde *React* de forma simple a través de una *API RESTful*[17]. (Bahgat, 2023)

[12] *Herramientas en desarrollo de software que permiten el mapeo o transformación de tablas en la base de datos a objetos en el lenguaje de programación que se esté utilizando.*

[13] *Método de infiltración de código malicioso que aprovecha una vulnerabilidad en la validación de entradas en la base de datos.*

[14] *Vulnerabilidad que permite al atacante ejecutar código malicioso en el navegador del usuario o del lado del cliente.*

[15] *Ataque que falsifica la petición de un usuario autenticado en una página web para la realización de acciones maliciosas sin su conocimiento.*

[16] *Técnica que pretende engañar al usuario final para que pulse en un señuelo que contiene acciones maliciosas.*

[17] *Interfaz que dos sistemas utilizan para intercambiar información entre sí a través de Internet por medio de diferentes servicios expuestos denominados Endpoints.*

3.2.4 SQLite

Un tema importante a la hora de configurar el *Back-End* es elegir el **Sistema Gestor de Bases de Datos**[18] *(SGBD)* que se va a utilizar en el proyecto. *Django* brinda acceso a numerosos *SGBD* de forma oficial como son *SQLite*, *PostgreSQL*, *MySQL*, *MariaDB* y *Oracle*, entre otros. Al tener unos requerimientos iniciales muy detallados con dependencias claras entre elementos, una base de datos relacional es una solución efectiva para el desarrollo del presente TFG, por lo que todas las alternativas oficiales son opciones válidas de cara a la aplicación a desarrollar.

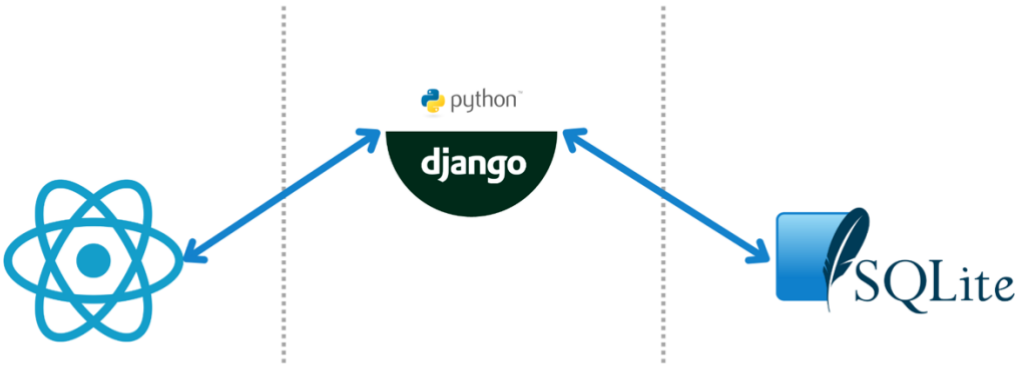


Ilustración 8: Modelo distribución de datos

A continuación, se enumeran las ventajas de *SQLite* para el almacenamiento de datos de la aplicación sobre otras alternativas:

- *SQLite* no necesita de configuración extra para empezar a funcionar, *Django* viene con *SQLite* de forma predeterminada, por lo que no se necesitan instalaciones ni configuraciones adicionales.
- Es perfecta para entornos de desarrollo y pruebas, debido a su eficiencia y facilidad de despliegue, por lo que es ideal para comenzar con el desarrollo.
- *SQLite* almacena toda la información de forma minimalista en un único archivo, por lo que su portabilidad hacia otros sistemas es muy sencilla.
- Está pensada para aplicaciones de una concurrencia razonable, por lo que se ajusta a las especificaciones, siendo extremadamente eficiente en estos casos.

[18] *Software que permite la gestión de una base de datos.*

- Cuenta con transacciones **ACID**, siglas que hacen referencia a las características de *Atomicidad*, *Consistencia*, *Aislamiento y Durabilidad* que aseguran la integridad de los datos que conformen la aplicación. (*SQLite Page*, s. f.-a)

3.2.5 Django REST Framework

Django REST Framework es una biblioteca que brinda apoyo a *Django* para crear **APIs RESTful**. La principal idea con el *Back-End* es que maneje el flujo de información a través de peticiones a varios **endpoints**[19], los cuales brindan funcionalidades a la interfaz de usuario. De este modo se adopta una política de *API RESTful* que brinda una serie de facilidades a la hora de tratar con los datos.

Django REST Framework ofrece serializadores[20] que permiten la conversión de modelos en *Django* (*QuerySet*[21]) en objetos nativos *Python* y *JSON* para el frontal de forma rápida y sencilla. Otra utilidad notable en el uso de esta biblioteca son las llamadas *ViewSets*[22], que ofrecen operaciones **CRUD** (*Crear*, *Leer*, *Actualizar y Eliminar*) de forma automática para el modelo.

También facilita el desarrollo y las pruebas, ya que brinda una interfaz propia para el consumo de los *endpoints* disponibles y la visualización de datos, que junto con el panel de

---

[19] *Hace referencia a un nodo de red en la comunicación expuesto por un comunicante para el consumo, en este caso, de una API RESTful.*

[20] *Componentes que permiten la transformación de los modelos Django a objetos nativos y formato JSON.*

[21] *Tipo de dato que hace referencia a una consulta sobre el modelo de datos en Django.*

[22] *Componentes que permiten la creación automática de métodos CRUD para un modelo Django dado.*

administrador de *Django* resulta una herramienta imprescindible a la hora de codificar. (*Django REST Framework*, s. f.)

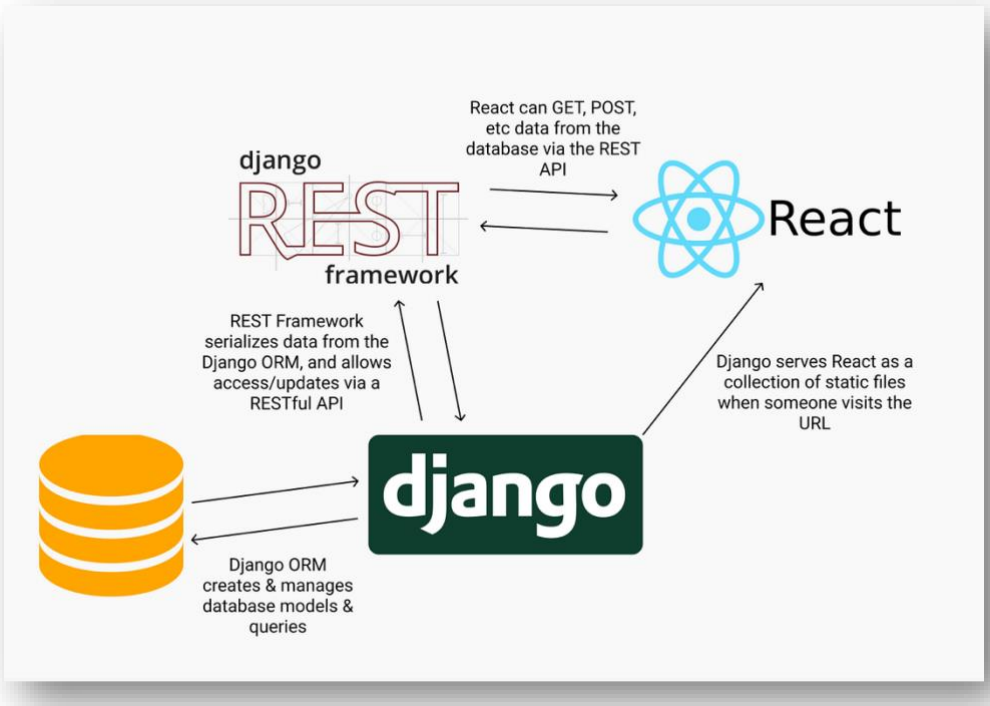


Ilustración 9 - Arquitectura con Django REST Framework (https://bennettgarner.medium.com/react-on-django-getting-started-f30de8d23504)

4 Análisis y Diseño

En este apartado se detalla la fase posterior al planteamiento, donde se define la estructura de los datos y el estudio previo al desarrollo necesario para la correcta consecución de los objetivos propuestos.

4.1 Requerimientos Funcionales

La aplicación debe cumplir con una serie de funcionalidades de cara al usuario final, el conjunto de estos requerimientos funcionales conforma en su totalidad la usabilidad de la aplicación web. Se procede a detallar de cuáles se tratan y en qué consisten.

- El **acceso** a la aplicación debe ser **autenticado** a través de un inicio de sesión donde se comprueban las credenciales de usuario. Además de un acceso **público** específico para la consulta de estos horarios, incorporando filtros específicos para que esta búsqueda sea lo más detallada posible.
- La **gestión** de contenido sensible o **CRUD** de datos como titulaciones, asignaturas, profesorado y horarios entre otros. Al usuario se le permite a través de la interfaz la creación, modificación y borrado de dichos datos.

- La **disposición** de los datos de forma ordenada en tablas para su correcta visualización, de modo que el usuario pueda filtrar por orden alfabético por cada atributo que muestran estas tablas.
- Vista de la **localización** de ubicaciones de diferentes espacios que conforman la plataforma, como *Campus* o *Edificios*.
- La **importación** de horarios de cursos anteriores al actual, para facilitar la creación de nuevos horarios.
- Posibilidad de **desplazar** las cuadrículas horarias hacia derechas y/o izquierdas, de modo que las columnas de un horario pueden desplazarse de un día a otro.
- Descarga y consulta de horarios en formato **PDF**, **HTML** y/o **CSV**. Tanto para la consulta pública como para los administradores del sistema, se puede descargar cada documento horario en formato *PDF*, *HTML* y/o *CSV*.

4.2 Requerimientos No Funcionales

A continuación, se pasará de las funcionalidades que ofrece la aplicación, a los criterios que reúne el sistema. Dichos requerimientos reciben el nombre de no funcionales y son los siguientes.

- La interfaz de usuario debe seguir un diseño **responsivo**, esto quiere decir que permite la adaptación de la aplicación web independientemente del tamaño de pantalla o navegador que se esté usando.
- La aplicación web debe ser compatible y funcional en los principales **navegadores** web como son Chrome, Firefox y Safari.
- La aplicación web debe estar totalmente **documentada**, tanto con el uso de la *API*, como con un manual de usuario y guía para el despliegue del sistema.
- Debe implementar medidas de **seguridad** como el uso de *JWT*[23] en todas las peticiones a la *API*, el uso encriptado con *BCRYPT*[24] para el tratado de contraseñas, además de la protección de seguridad que brinda Django contra ataques más comunes como *XSS*, *CSRF* o Inyección *SQL*.
- El uso de Frameworks como *React* o *Django* permiten que el sistema sea fácilmente **escalable** para futuras modificaciones o nuevas funcionalidades.

[23] *JSON Web Token son identificadores permiten conocer los privilegios para el acceso al sistema.*

[24] *Función de hash para la codificación de contraseñas.*

4.3 Casos de Uso

La aplicación web contempla tres principales actores en los casos de uso previstos para la interfaz, los cuales son **Administrador, Profesor y Alumno.**

Un alumno accederá a la parte pública de la aplicación, enfocado a la consulta de horarios y descarga de los mismos en *PDF* y/o *HTML*.

Un profesor tendrá acceso autenticado al sistema, pudiendo visualizar la totalidad de datos que ofrece la plataforma, con la posibilidad de crear horarios para la titulación en la que es responsable.

Un administrador tendrá igualmente acceso autenticado, con la particularidad de poder crear y modificar los recursos que se muestran en la plataforma, además de gestionar el acceso a nuevos usuarios. Este tipo de actor tendrá control total sobre la plataforma.

A continuación, se muestra una ilustración que ejemplifica los casos de uso de los actores que se han definido previamente.

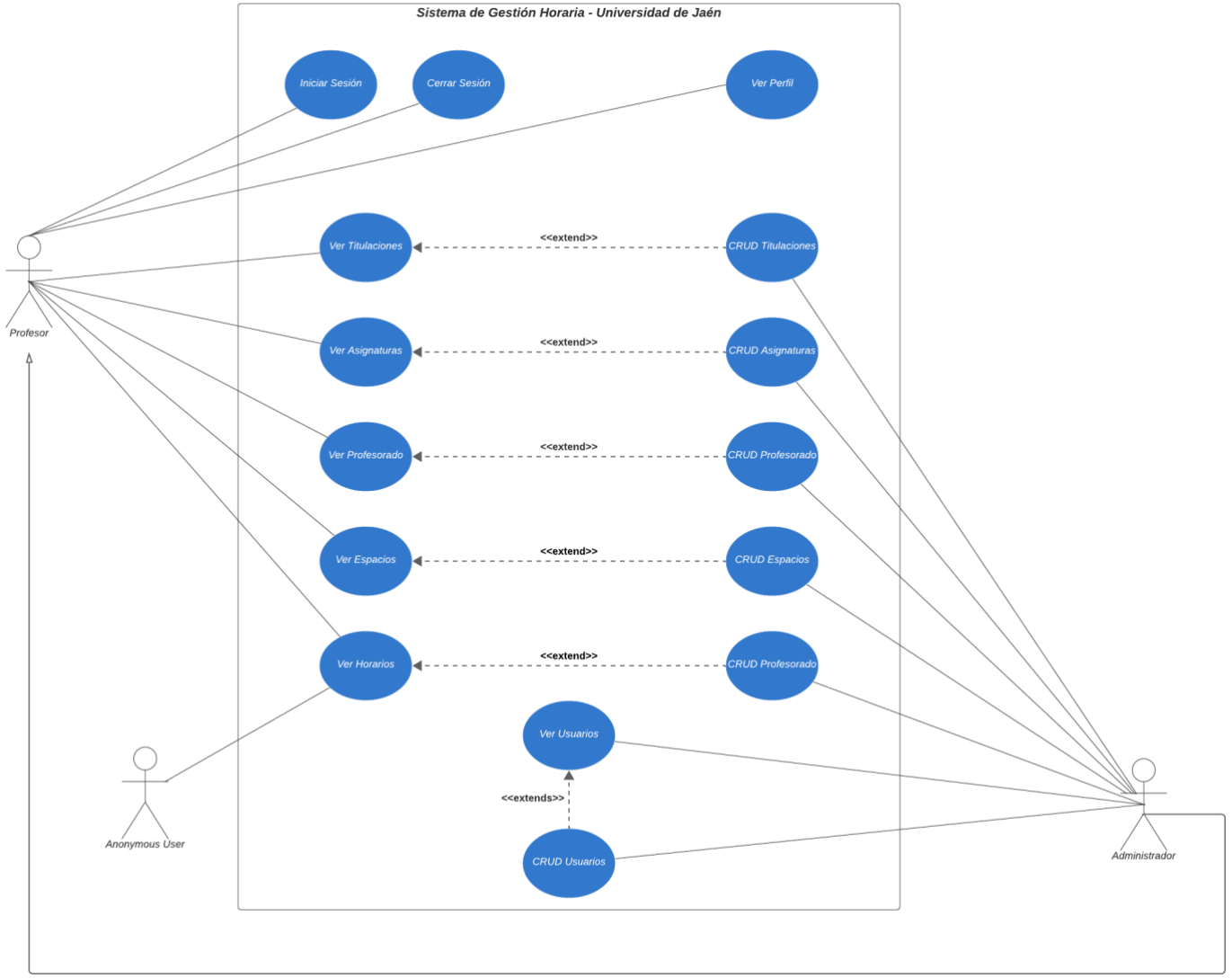


Ilustración 10: Diagrama de Casos de Uso

4.4 Arquitectura del sistema

Los elementos que conforman la arquitectura del sistema se exponían con las principales tecnologías utilizadas en esta aplicación web. Los dos grandes grupos en los que se puede dividir la arquitectura serán el *Front-End* y el *Back-End*, la conexión de ambos conformaría la estructura fundamental del sistema.

El *Back-End* se compone por la base de datos *SQLite* y el *Framework Django*, que con la ayuda de *Django REST Framework* conformarían el núcleo del lado servidor.

Dentro de la base de datos se almacenaría toda la información sensible que se va mostrando y recolectando en la interacción con la aplicación web.

Django proporciona toda la lógica de programación del lado servidor, brindando herramientas *ORM* (*Object-Relational Mapping*) para la comunicación directa con la base de datos de forma automática y definiendo las acciones a seguir para cada *endpoint* que se define en la *API*.

Con *Django REST Framework* se adapta el servidor para que funcione como una *API RESTful*, con una serie de *endpoints* definidos para su llamada. Además, se proporcionan serializadores para adaptar los datos entre *Django* y *React* de forma sencilla.

El *Front-End* se compone de la interfaz de usuario creada con *React*, una librería *JavaScript* para facilitar el desarrollo haciendo uso de tecnologías web clásicas como *HTML*, *CSS* y *JavaScript*. El usuario final accederá a esta interfaz a través de un navegador web.

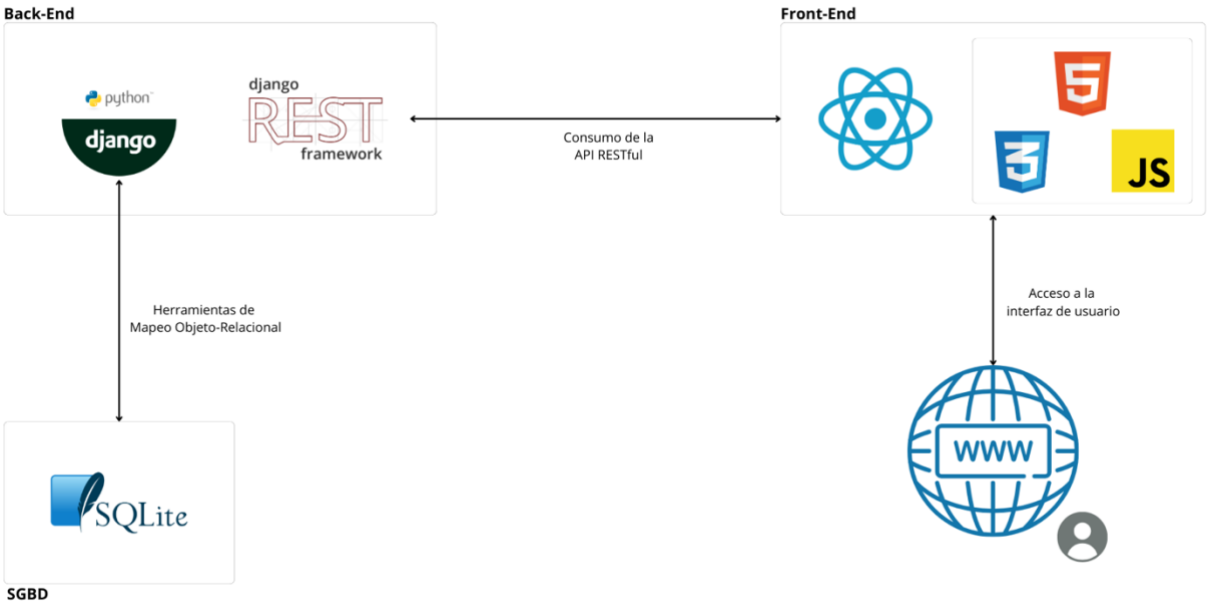


Ilustración 11 - Arquitectura del sistema

4.5 Diseño de la Base de Datos

La parte que engloba la mayor dedicación radica en el diseño de la estructura de la base de datos. Cómo los datos serán almacenados y la relación entre ellos conforman una parte esencial de cara al desarrollo futuro. Un buen análisis previo facilita la codificación y evita inconvenientes a posteriori.

A continuación, se muestra el diagrama **Entidad-Relación** que enumera las diferentes tablas que sigue el diseño de la base de datos, así como las relaciones entre ellas. Cada tabla corresponde con un objeto tradicional en Python, gracias a las herramientas ORM que ofrece Django esta conversión se hace de manera automática.

Se procede a detallar cada tabla y sus principales atributos, así como el significado y papel que toman en la aplicación web.

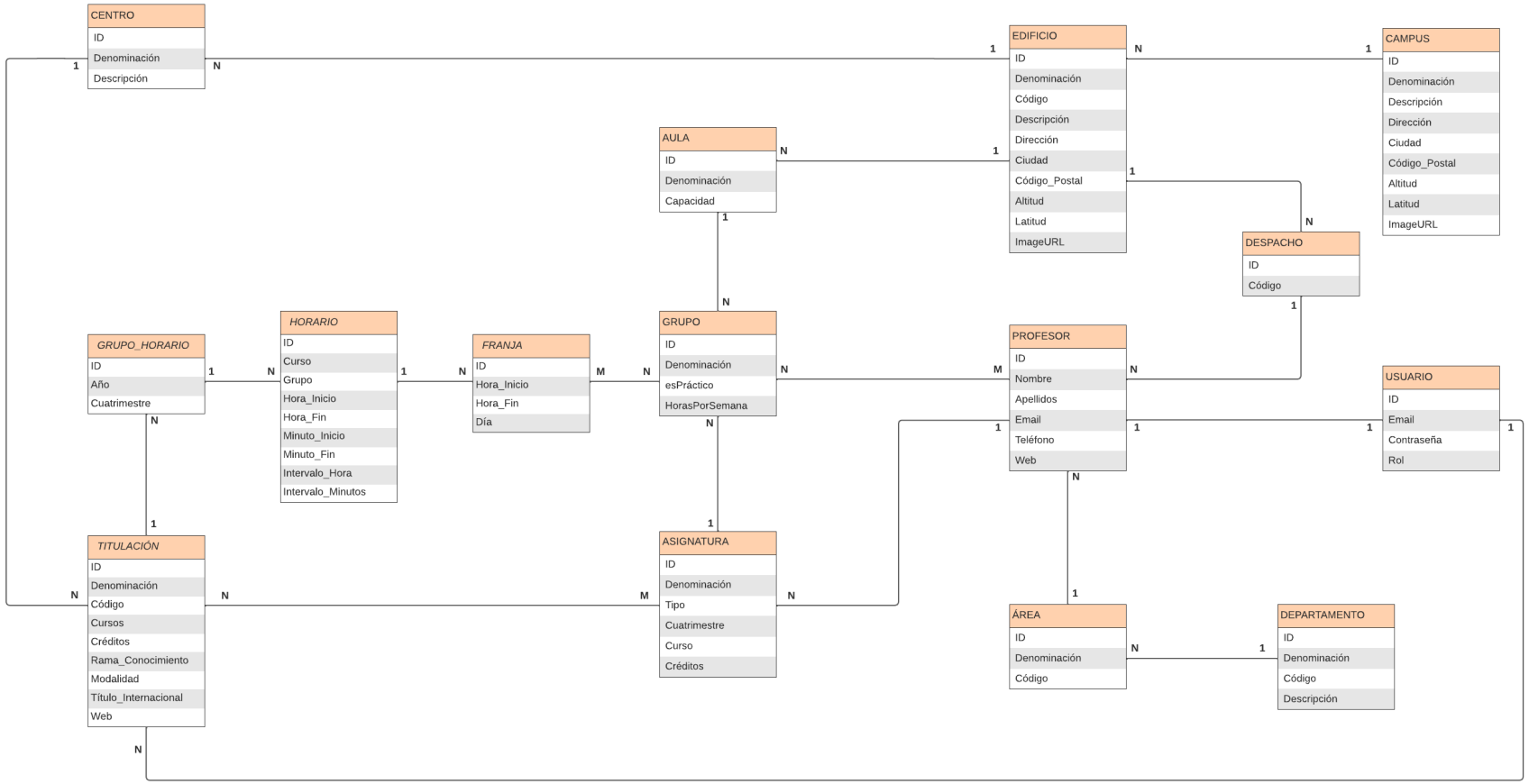


Ilustración 12: Diagrama Entidad-Relación

4.5.1 *Campus*

Tabla que identifica los *Campus* que conforman la Universidad de Jaén, dentro de los cuales pueden existir *Edificios*.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Campus.	-
denominacion	Nombre que recibe el Campus.	-
descripcion	Texto que define el Campus.	-
direccion	Dirección del Campus.	-
ciudad	Ciudad de pertenencia del Campus.	-
codigo_postal	Código Postal del Campus.	-
altitud	Coordenadas de localización.	-
latitud	Coordenadas de localización.	-
imageURL	Imagen de muestra del Campus.	-

Tabla 1: Campus

4.5.2 *Edificio*

Tabla que identifican los *Edificios* independientes y/o pertenecientes a los diferentes *Campus* de la Universidad de Jaén. Si el atributo *campus* queda nulo significa que el *Edificio* es independiente y no pertenece a ningún *Campus*. Existen edificios en la Universidad de Jaén que no se ubican en ningún campus, por lo que es una restricción para tener en cuenta en el sistema.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Edificio.	-
denominacion	Nombre que recibe el Edificio.	-
codigo	Código identificador del Edificio.	-
descripcion	Texto que define el Edificio.	-
direccion	Dirección del Edificio.	-
ciudad	Ciudad de pertenencia del Edificio.	-
codigo_postal	Código Postal del Edificio.	-
altitud	Coordenadas de localización.	-
latitud	Coordenadas de localización.	-
imageURL	Imagen de muestra del Edificio.	-
campus	Si pertenece a un Campus o no.	**Campus** (id)

Tabla 2: Edificio

4.5.3 *Centro*

Tabla que identifica los *Centros* de los que dispone la Universidad de Jaén. Cada *Titulación* debe pertenecer a un *Centro*. Por norma general, cada *Centro* reside en un *Edificio* o *Campus* de la Universidad de Jaén.

Atributo	*Definición*	*Clave Foránea*
id	Identificador del Centro.	-
denominacion	Nombre del Centro.	-
descripcion	Texto que define al Centro.	-
edificio	Hace referencia al Edificio donde se aloja dicho Centro.	**Edificio** (id)

Tabla 3: Centro

4.5.4 *Aula*

Tabla que enumera las diferentes *Aulas* con las que se cuenta en la Universidad de Jaén, esta tabla es esencial para conocer las estancias disponibles para impartir docencia y que marcarán las restricciones espaciales a la hora de confeccionar los horarios.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Aula.	-
denominacion	Nombre que recibe el Aula.	-
capacidad	Capacidad de alumnos.	-
edificio	Edificio al que pertenece el Aula.	**Edificio** (id)

Tabla 4: Aula

4.5.5 *Despacho*

Tabla que enumera los diferentes *Despachos* en los que se aloja el profesorado de la Universidad de Jaén. Los *Despachos* son otro tipo de estancia que tiene un *Edificio* asignado y a los cuales se les asigna un *Profesor*.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Despacho.	-
codigo	Código identificador de Despacho.	-
edificio	Edificio al que pertenece el Despacho.	**Edificio** (id)

Tabla 5: Despacho

4.5.6 *Departamento*

Tabla que enumera los *Departamentos* a los que pertenece el profesorado de la Universidad de Jaén.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Departamento.	-
denominacion	Nombre que recibe el Departamento.	-
codigo	Código identificador de Departamento.	-
descripcion	Texto que define el Departamento.	-

Tabla 6: Departamento

4.5.7 *Área*

Tabla que enumera las *Área* en las que se dividen los diferentes *Departamentos* de la Universidad de Jaén. Contienen el atributo *Departamento* para identificar su procedencia. Tanto *Departamento* como *Área* son datos utilizados para agregar consistencia al profesorado.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Departamento.	-
denominacion	Nombre que recibe el Departamento.	-
codigo	Código identificador de Departamento.	-
departamento	Departamento al que pertenece el Área.	**Área** (id)

Tabla 7: Área

4.5.8 *Profesor*

Tabla fundamental en el sistema, que identifica a los diferentes *Profesores* que imparten docencia en la Universidad de Jaén. Posteriormente, este profesorado impartirá los *Grupos* que conforman las cuadrículas horarias. Cada *Profesor* tiene un *Despacho* y un *Área* asignados.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Profesor.	-
nombre	Nombre del Profesor.	-
apellidos	Apellidos del Profesor.	-
email	Correo del Profesor.	-
telefono	Teléfono del Profesor.	-

web	Web del Profesor.	-
despacho	Despacho al que pertenece el Profesor.	**Despacho** (id)
area	Área al que pertenece el Profesor.	**Área** (id)

Tabla 8: Profesor

4.5.9 Usuario

Esta tabla incluye los usuarios que tienen acceso autenticado a la plataforma. El atributo rol permite clasificar el acceso y brindar funcionalidades dependiendo del tipo de perfil. Si un *Usuario* tiene asignado un *Profesor* podrá ver su información detallada en la sección de Mi Perfil. Además, en caso de tener el rol de *profesor* se puede conocer las titulaciones de las que es responsable y sobre las que puede generar horarios.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Usuario	-
email	Correo asignado al Usuario.	-
contraseña	Contraseña de acceso para el Usuario.	-
rol	Identifica los permisos que tiene el Usuario.	-
profesor	Hace referencia si el Usuario pertenece a un Profesor en concreto.	**Profesor** (id)

Tabla 9: Usuario

4.5.10 Titulación

La tabla de *Titulaciones* es de las más importantes del sistema. En ella se definen todas las titulaciones que imparte la Universidad de Jaén. Una titulación pertenece a un *Centro* en concreto. Además, cada titulación tiene un *Usuario* que será el responsable de ésta, y podrá generar cuadrículas horarias de dicha titulación, aparte de los usuarios con rol de *administrador*.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de la Titulación.	-
denominacion	Nombre de la Titulación.	-
codigo	Código que identifica la Titulación.	-
cursos	Número de cursos que tiene la Titulación.	-
creditos	Número de créditos que conforman la Titulación.	-

rama_conocimiento	Rama de Conocimiento que pertenece la Titulación.	-
modalidad	Tipo de modalidad que sigue la Titulación.	-
internacional	Booleano que indica si la Titulación tiene doble título internacional.	-
web	Web de la Titulación.	-
centro	Centro al que pertenece la Titulación.	**Centro** (id)
usuario_titular	Usuario con rol *profesor* responsable de la Titulación.	**Usuario** (id)

Tabla 10: Titulación

4.5.11 Asignatura

Tabla que enumera las diferentes *Asignaturas* que existen en la Universidad de Jaén. Cada *Asignatura* puede ser común para varias *Titulaciones*, por lo que se generará otra tabla en esta relación. Cada *Asignatura* cuenta con un *Profesor* responsable de ésta.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de la Asignatura.	-
denominacion	Nombre que recibe la Asignatura.	-
tipo	Enumera uno de los cuatro tipos posible de Asignatura.	-
cuatrimestre	Cuatrimestre al que pertenece la Asignatura.	-
curso	Curso al que pertenece la Asignatura.	-
creditos	Número de créditos que tiene la Asignatura.	-
responsable	Profesor responsable de la Asignatura.	**Profesor** (id)

Tabla 11: Asignatura

4.5.12 Titulación_Asignatura

Tabla surgida de una relación M:M[25] entre *Titulaciones* y *Asignaturas*. Debido a que cada *Asignatura* es independiente de la *Titulación* y puede pertenecer a más de una con un código único, se crea esta nueva tabla que identifica las asignaciones entre ambas tablas.

[25] *Relación entre tablas muchos a muchos, esto quiere decir que una titulación posee muchas asignaturas y una asignatura puede estar presente en muchas titulaciones.*

Atributo	*Definición*	*Clave Foránea*
titulacion	Identifica la Titulación en la asignación.	**Titulacion** (id)
asignatura	Identifica la Asignatura en la asignación.	**Asignatura** (id)
codigo	Indica el código de Asignatura.	-

Tabla 12: Titulación_Asignatura

4.5.13 Grupo

Tabla que recoge los diferentes *Grupos* impartidos para una misma *Asignatura*, en esta tabla se recogerían los grupos amplios de teoría y los grupos reducidos de prácticas. Además, hay un atributo aula que hace referencia a una clave foránea de la tabla *Aulas*, debido a que todo grupo debe impartirse en un espacio reservado.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Grupo.	-
denominacion	Nombre que recibe el Grupo.	-
practico	Booleano que indica si es un grupo amplio o un grupo reducido.	-
horasxsemana	Número de veces que se imparte el grupo a la semana (la duración viene definida por el intervalo que se especifica en el horario).	-
aula	Aula donde se imparte el Grupo.	**Aula** (id)
asignatura	Asignatura a la que pertenece el Grupo.	**Asignatura** (id)

Tabla 13: Grupo

4.5.14 Profesor_Grupo

Esta tabla es el resultado de una relación M:M entre *Profesores* y *Grupos*. Como un profesor puede impartir clase en varios grupos de diferentes asignaturas, y un grupo puede llegar a ser impartido por varios profesores, se obtiene como resultado esta tabla. En ella se puede consultar la asignación de grupos impartidos por profesores.

Atributo	*Definición*	*Clave Foránea*
profesor	Identificador del Profesor.	**Profesor** (id)
grupo	Identificador del Grupo.	**Grupo** (id)

Tabla 14: Profesor_Grupo

4.5.15 Grupo_Horario

La tabla de *Grupo Horario* hace referencia al modo de organización que siguen los horarios de las diferentes titulaciones de la Universidad de Jaén. Cada *Titulación* tiene para un curso académico y un cuatrimestre un documento que muestra todas las cuadrículas horarias de los diferentes cursos y grupos que conforman a esa titulación para el curso y cuatrimestre especificado. En esta plataforma, un objeto de esta tabla hará referencia a ese documento que engloba una serie de cuadrículas.

Atributo	*Definición*	*Clave Foránea*
id	Identificador de Grupo Horario.	-
anho	Curso académico referente al Grupo Horario.	-
cuatrimestre	Cuatrimestre referente al Grupo Horario.	-
titulacion	Titulación sobre la que se realiza el Grupo Horario.	**Grupo_Horario** (id)

Tabla 15: Grupo_Horario

4.5.16 Horario

La tabla *Horario* hace referencia a cada cuadrícula de la que se conforma un grupo horario. Cada cuadrícula pertenece a un curso y a un grupo de la titulación, curso académico y cuatrimestre indicados por el grupo horario. Los atributos que definen el rango que marca el inicio y el fin del horario. Además, la duración de cada celda vendrá definida por el atributo de *intervalo*, este deberá coincidir con la duración de los grupos impartidos y con el rango entre hora de inicio y hora de fin de manera equitativa y exacta. La comprobación de tiempos es muy importante antes de la creación de un *Horario* para que no surjan incongruencias al usar el sistema.

Atributo	*Definición*	*Clave Foránea*
id	Identificación del Horario.	-
curso	Curso al que pertenece el Horario.	-
grupo	Grupo al que pertenece el Horario.	-
hora_inicio	Entero que indica la hora de inicio del Horario.	-
hora_fin	Entero que indica la hora de fin del Horario.	-
minuto_inicio	Entero que indica el minuto de inicio del Horario.	-
minuto_fin	Entero que indica el minuto de fin del Horario.	-

intervalo_hora	Entero que indica el intervalo en horas del Horario.	-
intervalo_minutos	Entero que indica el intervalo en minutos del Horario.	-
grupo_horario	Grupo Horario al que pertenece el Horario.	**Grupo_Horario** (id)

Tabla 16: Horario

4.5.17 Franja

La tabla *Franja* hace referencia a cada celda que conforma un *Horario*. Al conocer el rango de horas de un *Horario* y su intervalo, es fácilmente calcular el número de celdas con las que contará dicho *Horario*. Cada *Franja* cuenta con un día, una hora de inicio y una hora de fin.

Atributo	*Definición*	*Clave Foránea*
id	Identificación de la Franja.	-
hora_inicio	Hora de inicio de la Franja.	-
hora_fin	Hora de fin de la Franja.	-
dia	Día de la Franja.	-
horario	Horario al que pertenece la Franja.	**Horario** (id)

Tabla 17: Franja

4.5.18 Franja_Grupo

La tabla *Franja_Grupo* ejemplifica la asignación de grupos que se imparten en una determinada franja. Una franja puede albergar varios grupos, y un grupo puede darse en varias franjas a la semana, por lo que esta tabla es imprescindible para la creación de un horario.

Atributo	*Definición*	*Clave Foránea*
franja	Identificación de Franja.	**Franja** (id)
grupo	Identificación de Grupo.	**Grupo** (id)

Tabla 18: Franja_Grupo

5 Implementación

Una vez realizado el análisis previo y la definición de la estructura de la base de datos, dará comienzo la siguiente etapa de la metodología propuesta. Comienza el proceso de **desarrollo** e implementación. Esta etapa es el reflejo de todo el estudio previo, simplificando el proceso si se ha realizado la etapa de análisis correctamente.

En este apartado se detallará técnicamente el proceso de desarrollo, tanto del *Front-End* como del *Back-End*, con ejemplos de código destacado, muestra de la interfaz de usuario, desafíos y obstáculos encontrados.

5.1 Descripción técnica

Una vez conocidas las tecnologías que se utilizan y contando con el análisis previo de requisitos y especificaciones, se comienza a detallar técnicamente tanto el *Front-End* como el *Back-End* de este proyecto. Se procede a dividir esta sección en dos partes donde se detallará técnicamente cada tecnología y se desarrollará código a modo de ejemplo. Primero se comenzará con el *Back-End* desarrollado con *Python* en *Django*, y después se definirá el *Front-End* desarrollado con *JavaScript* haciendo uso de la librería *React*.

5.1.1 Descripción técnica en el Back-End

Antes de comenzar el desarrollo en *Python* se debe tener preparado una serie de requisitos previos y conocer la estructura que sigue *Django* al crear un proyecto con este *Framework*.

El primer paso es configurar un entorno virtual en *Python* con el que se trabajará de forma aislada y poder gestionar paquetes y dependencias sin afectar a la configuración global del equipo. A partir de este entorno virtual, se deberá instalar *Django* y *Django REST Framework*, y ejecutar una serie de comandos para la creación y configuración de un proyecto Django.

Una vez creado, dentro de un proyecto *Django* se contará con varios archivos con utilidades específicas:

- ***Settings.py***, es un archivo de configuración del proyecto, donde se puede ajustar las necesidades específicas de la aplicación. En este caso, para implementar *Django REST Framework* se debe modificar este archivo e incluirlo en el *array INSTALLED_APPS*. Otra configuración que se debe hacer en este archivo es la gestión del *CORS* y que no se tenga problemas más adelante al realizar peticiones desde el frontal.
- ***Urls.py***, define las rutas a las que se puede acceder desde el servidor *Django*, por defecto se tiene un panel de administración donde se pueden gestionar los

datos de la base de datos en */admin* y por defecto, *Django REST Framework* ofrece una *API* de consulta en la ruta raíz del servidor.

- ***Manage.py***, este archivo define los comandos a ejecutar en el entorno virtual, tanto para la migración de datos en la base de datos, como para arrancar el servidor de desarrollo.

Cada proyecto de *Django* cuenta con una serie de aplicaciones que se pueden añadir de forma modular, estas aplicaciones sirven para diferenciar partes del proyecto en caso de tener una gran envergadura. En este caso se ha hecho uso de una única aplicación que se debe configurar en el proyecto a través de un comando específico que se encuentra en *manage.py*.

La aplicación cuenta con una serie de archivos en los que se comenzará realmente la programación:

- ***Admin.py***, archivo que se encarga de registrar tablas de la base de datos en el panel de gestión de administrador que ofrece *Django*.
- ***Models.py***, en este documento se definen los modelos correspondientes a las tablas que se han obtenido en el análisis de la estructura de la base de datos.
- ***Views.py*** hace referencia a la lógica de programación que se define para cada petición a un *Endpoint*, este archivo constituye el núcleo del *Back-End* y es el que brinda todas las funcionalidades que se definen posteriormente.
- ***Urls.py*** define las rutas y enlaza cada *Endpoint* con la lógica de programación que se encuentra en el archivo *views.py*.
- ***Serializer.py*** define serializadores gracias a *Django REST Framework*, estos serializadores permiten adaptar los datos para enviarlos en el formato correcto.

A partir de este momento da comienzo el desarrollo del *Back-End*. El primer paso en la codificación es implementar los modelos que se obtuvieron del análisis en el archivo correspondiente para ello. A continuación, se muestra un ejemplo de código en el que se crea el modelo *'Titulación'*, este modelo hace referencia a la tabla correspondiente del diagrama entidad-relación. Cada modelo cuenta con una serie de atributos, gracias a la importación de la clase *models* perteneciente a *django.db* se pueden definir tipos de datos preestablecidos, ya sean texto, numérico, enumeración, booleano, entre otros.

```
class Titulacion(models.Model):
    denominacion = models.CharField(max_length=50)
```

```
    codigo = models.CharField(max_length=10)
    cursos = models.IntegerField()
    creditos = models.IntegerField()
    RamaConocimientoTypes = models.TextChoices("RamaConocimientoTypes", "Arte-Humanidades Ciencias
Ciencias-Salud Sociales-Jurídicas Ingeniería-Arquitectura")
    rama_conocimiento = models.CharField(max_length=25, choices=RamaConocimientoTypes)
    ModalidadTypes = models.TextChoices("ModalidadTypes", "Presencial Distancia")
    modalidad = models.CharField(max_length=15, choices=ModalidadTypes)
    internacional = models.BooleanField(default=False)
    web = models.CharField(max_length=200)
    usuario = models.ForeignKey(Usuario, on_delete=models.SET_NULL, default=1, null=True)
    centro = models.ForeignKey(Centro, on_delete=models.SET_NULL, default=1, null=True)
    asignaturas = models.ManyToManyField(Asignatura, related_name='asignaturas',
through='Titulacion_Asignatura')


    def __str__(self):
        return self.denominacion + ' (' + self.codigo + ')'
```

Si un modelo posee un atributo que identifica a otra tabla/modelo del sistema se hace a través de claves foráneas. Para cada clave foránea es importante definir acciones como *on_delete*, *default* o *null* para controlar la respuesta de los datos en caso de eliminación o presencia de datos vacíos en ese campo. La función *__str__* es indispensable en cada modelo, pues imprimirá los atributos más relevantes de un modelo para identificarlo por consola o en entornos como el panel de administración.

Una vez se define el modelo o se haga algún cambio significativo en este, es importante realizar una migración de la base de datos. Para realizar esto se debe ejecutar dos comandos que ofrece *manage.py*, los cuales son *makemigrations* y *migrate*. Para crear una nueva migración se hace uso del primer comando, *makemigrations*, que se almacenará dentro de la aplicación en la carpeta *migrations*. Con *migrate* se aplicará la última migración a la base de datos real.

El siguiente paso en el desarrollo del *Back-End* es la implementación de las vistas en el fichero *views.py* de la aplicación *Django*. Al usar *Django REST Framework*, el *CRUD* de los diferentes modelos se realiza automáticamente gracias a los *ViewSets*. Un *ViewSet* es una clase que agrupa un conjunto de vistas, gracias a esto se simplifican y automatizan los procesos repetitivos como son el *CRUD* de los diferentes modelos del sistema y centrar la

atención en llamadas específicas que brinden funcionalidad real a la aplicación web. Al utilizar esto se implementan medidas *DRY (Do not Repeat Yourself)* que reducen la repetición de código que puede albergar un sistema. Un ejemplo de código de la implementación de un *ViewSet* se muestra a continuación:

```
from rest_framework import viewsets

#CRUD TITULACION
class TitulacionView(viewsets.ModelViewSet):
    serializer_class = TitulacionSerializer
    queryset = Titulacion.objects.all()
```

Para utilizar la funcionalidad que brindan estos *ViewSets* se debe registrar las rutas que realizarán este *CRUD* en el sistema, para ello se implementa un *router* de *Django REST Framework*. Este *router* se encarga de agregar las rutas necesarias de creación, modificación y/o eliminación de forma automática. Un ejemplo de cómo configurar un *router* de *Django REST Framework* se muestra a continuación:

```
from rest_framework import routers

router = routers.DefaultRouter()
router.register(r'titulaciones', views.TitulacionView, 'titulaciones')
```

De esta forma si se accede a *'/titulaciones'* con un método *GET, PUT o DELETE*, automáticamente detectará la ruta y proporcionará la funcionalidad *CRUD* si el resto de los campos de la petición son correctos.

A continuación, se definen las vistas y rutas específicas para funcionalidades clave de la aplicación web, por ejemplo, la autenticación de la plataforma, la consulta de los grupos a los que imparte docencia un profesor o la comprobación de las restricciones de espacio y profesorado al asignar un grupo a una cuadrícula horaria.

Se comienza poniendo un ejemplo básico de vista y se irá desglosando poco a poco para explicar su estructura y cómo funciona realmente. En la siguiente vista se procede a definir la lógica que devuelve toda la información relevante acerca de un profesor, por ejemplo, cuando en la aplicación web se accede a la vista de un profesor se deben pedir sus

datos personales, su área, su departamento, los grupos a los que imparte docencia, entre otras cosas.

La vista comienza con un *@api_view*, esto es una anotación que proporciona *REST Framework* y se utiliza para definir detalles de la llamada, en este caso el método *HTTP* a utilizar que será un *POST*.

```
from rest_framework.decorators import api_view

@api_view(['POST'])
def detailsfromprofesor(request):
    id_profesor = request.data.get('id_profesor')
    if id_profesor is None:
        return Response({"error": "Se necesita proporcionar el ID del Profesor en la solicitud."},
status=status.HTTP_400_BAD_REQUEST)

    try:
        profesor = Profesor.objects.get(pk=id_profesor)
    except Profesor.DoesNotExist:
        return Response({"error": "El Profesor con el ID proporcionado no existe."},
status=status.HTTP_404_NOT_FOUND)
```

El nombre que recibe la vista es *detailsfromprofesor* y posteriormente será llamada para la asignación con la ruta en el archivo *urls.py*. Esta vista siempre recibe un *request* como parámetro, esta es la llamada que se produce desde el frontal e incluirá información importante como el *body* de la petición en caso de que se tratara de un *POST*.

En este ejemplo, *request* incluye parámetros en el *body*, los cuales pueden ser recogidos como *request.data.get* y especificar la denominación del dato. En el código anterior se obtiene el ID de un profesor en concreto.

En todas las vistas se debe ser muy riguroso de cara a la generación de errores, por lo que se debe contemplar las diferentes casuísticas que se presentan a la hora de recibir una petición. Es el caso de que llegara un ID de profesor vacío, el microservicio debe ser capaz de devolver un error *HTTP 400* indicando el motivo del fallo. De esta forma se puede tener controlados los diferentes errores que pueden surgir en la aplicación web para que se adapte correctamente y ofrezca una experiencia de usuario sólida y satisfactoria.

Una vez se tiene el ID de un profesor, se necesita obtener el objeto *Profesor*, para ello, y gracias a las herramientas *ORM* que ofrece *Django*, esto se hace de manera automática. Basta con especificar el modelo, en este caso el de un profesor, y llamar a *Profesor.objects.get*. Este método permitirá obtener de entre todos los profesores que están en la base de datos, el que su clave primaria corresponda con el identificador que se ha proporcionado. En caso de no existir dicho profesor, se debe controlar la devolución del error correspondiente, en este caso *404 Not Found*. Si todo ha ido correctamente se consigue un objeto *Profesor* correctamente almacenado en un objeto *Python*.

```
gruposData = Profesor_Grupo.objects.filter(profesor=profesor.id)
grupos = []
asignaturas = []
asignaturas_agregadas = set()
for profesor_grupo in gruposData:
    grupos.append(profesor_grupo.grupo)
    asignatura_id = profesor_grupo.grupo.asignatura.id
    if asignatura_id not in asignaturas_agregadas:
        asignaturas.append(profesor_grupo.grupo.asignatura)
        asignaturas_agregadas.add(asignatura_id)

serializer_gruposprof = ProfesorGrupoSerializer(gruposData, many=True)
serializer_asignaturas = AsignaturaSerializer(asignaturas, many=True)
serializer_grupos = GrupoSerializer(grupos, many=True)
```

En la tabla *Profesor_Grupo* se almacenaba cada asignación de un profesor con los grupos a los que imparte docencia. El método *filter* permite obtener todos los objetos de dicha tabla que cumplen con los requisitos que se pasan como parámetro. En este caso, se desea encontrar todos los grupos a los que el objeto *profesor* imparte docencia, para ello se hace uso de *Profesor_Grupo.objects.filter*. A continuación, se itera con un bucle todos los resultados del filtro anterior, y se va almacenando en *arrays* los diferentes grupos y asignaturas a los que un profesor imparte docencia. Esto es posible gracias a las relaciones que se han definido en el modelo, y a las herramientas *ORM* de *Django* que permiten acceder fácilmente a la asignatura que pertenece determinado grupo. Como detalle, se utilizarán un conjunto o *set* para identificar de forma única cada asignatura, y así no repetirlas dentro del

array. Una vez almacenados los grupos y asignaturas que imparte un profesor, se pasa al *serializador* para obtener los datos en formato *JSON*.

En las siguientes líneas de código se intenta acceder al despacho, área y departamento del profesor, de no tener uno asignado, se quedará vacío.

```
if profesor.despacho is not None:
    serializer_despacho = DespachoSerializer(profesor.despacho, many=False)
    despacho_data = serializer_despacho.data
else:
    despacho_data = {}

if profesor.area is not None:
    serializer_area = AreaSerializer(profesor.area, many=False)
    area_data = serializer_area.data
    if profesor.area.departamento is not None:
        serializer_departamento = DepartamentoSerializer(profesor.area.departamento, many=False)
        departamento_data = serializer_departamento.data
    else:
        departamento_data = {}
else:
    area_data = {}
    departamento_data = {}
```

Una vez obtenidos todos los *serializadores* se monta el objeto *JSON* de respuesta, para ello se accede a la propiedad *data* de cada *serializador*.

Por último, con *Response* se envía la respuesta a la petición, y así quedaría completado el flujo completo de una petición.

```
response_data = {
    "profesorgrupos": serializer_gruposprof.data,
    "asignaturas": serializer_asignaturas.data,
    "grupos": serializer_grupos.data,
    "despacho": despacho_data,
    "area": area_data,
    "departamento": departamento_data
}

return Response(response_data)
```

Una vez definido la función que se ejecutará al llamar a una ruta determinada, hay que definir el nombre de la ruta y asignarle la función que se acaba de crear. Para ello se debe modificar el archivo *urls.py* que se citaba anteriormente, pero ahora agregando rutas a la variable *urlpatterns*.

```
urlpatterns = [
    path('', include(router.urls)),
    path('fromprofesor/getdetails', views.detailsfromprofesor, name='detailsfromprofesor'),
]
```

Este *array* debe incluir las rutas que ya se definen en el mismo archivo *urls.py* del proyecto *Django* y agregar las nuevas funcionalidades del fichero de vistas.

Si todo se ha configurado correctamente, al llamar a la ruta con el método y parámetros correctos, esto devolverá la información del profesor que se quiere consultar.

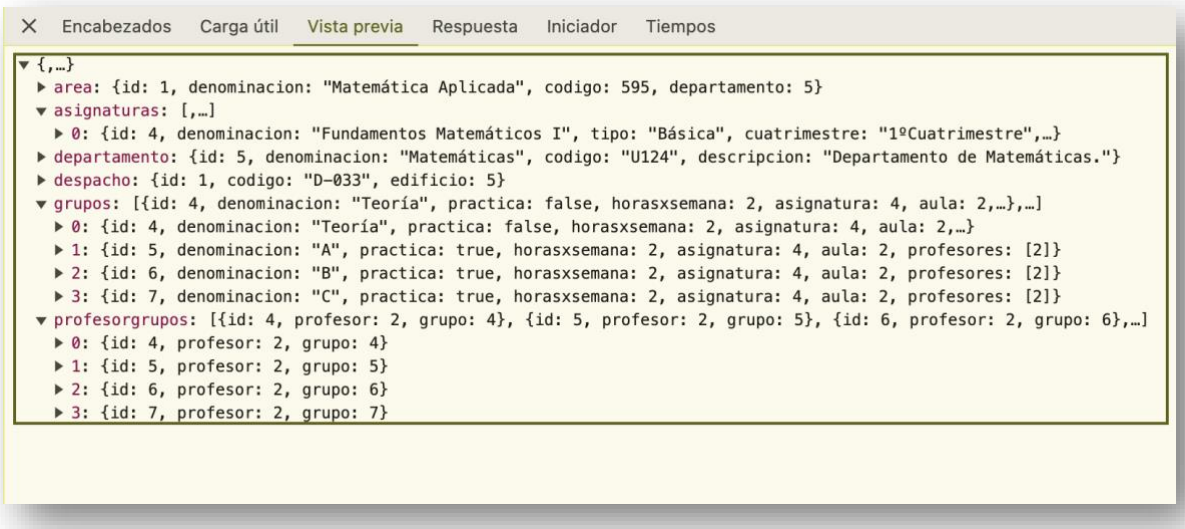


Ilustración 13: Respuesta de datos del API RESTful

Por último, se procede a explicar una de las vistas que más está relacionada con la gestión horaria, pues es el principal objetivo de este TFG. La vista en cuestión se basa en la creación de una cuadrícula horaria con todas sus franjas.

```
@api_view(['POST'])
def addhorario(request):
    id_grupohorario = request.data.get('grupo_horario')
    if id_grupohorario is None:
```

```
        return Response({"error": "Se necesita proporcionar el ID del Grupo Horario en la solicitud."},
status=status.HTTP_400_BAD_REQUEST)

    try:
        grupohorario = GrupoHorario.objects.get(pk=id_grupohorario)
    except GrupoHorario.DoesNotExist:
        return Response({"error": "El Grupo Horario con el ID proporcionado no existe."},
status=status.HTTP_404_NOT_FOUND)

    if Horario.objects.filter(grupo_horario=grupohorario, curso=request.data.get('curso'),
grupo=request.data.get('grupo')).exists():
        return Response({"error": "Ya existe un horario con estos valores."}, status=400)

    days_of_week = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes']
    start_time = request.data.get('hora_inicio') * 60 + request.data.get('minuto_inicio')
    end_time = request.data.get('hora_fin') * 60 + request.data.get('minuto_fin')
    interval = request.data.get('intervalo_hora') * 60 + request.data.get('intervalo_minutos')

    if interval <= 0:
        return Response({"error": "El intervalo proporcionado no es válido."},
status=status.HTTP_400_BAD_REQUEST)

    if end_time <= start_time:
        return Response({"error": "El tiempo final es menor que el tiempo inicial."},
status=status.HTTP_400_BAD_REQUEST)

    if (end_time - start_time) % interval != 0:
        return Response({"error": "El intervalo no coincide exacto con los tiempos de inicio y fin."},
status=status.HTTP_400_BAD_REQUEST)
```

En esta petición se requiere identificar el *Grupo Horario*, que es el documento que finalmente se descarga y que incluye diferentes cuadrículas horarias referentes a una asignatura, un curso académico y un cuatrimestre.

Una vez obtenido correctamente, se debe comprobar si la cuadrícula horaria que se quiere crear no existe ya, por lo que se realiza un filtro por curso y grupo al que pertenece.

Se debe continuar con las comprobaciones, no es posible crear cuadrículas horarias con intervalos negativos, con una hora final menor que la hora inicial o que el intervalo no se ajuste en el rango de horas de la cuadrícula.

```
try:
horario = Horario.objects.create(curso=request.data.get('curso'), grupo=request.data.get('grupo'),
hora_inicio=request.data.get('hora_inicio'), hora_fin=request.data.get('hora_fin'),
minuto_inicio=request.data.get('minuto_inicio'), minuto_fin=request.data.get('minuto_fin'),
intervalo_hora=request.data.get('intervalo_hora'), intervalo_minutos=request.data.get('intervalo_minutos'),
grupo_horario=grupohorario)

for day in days_of_week:
            current_time = start_time
      while current_time < end_time:
            franja = Franja.objects.create(
           hora_inicio = f"{current_time // 60}:{current_time % 60}",
                                 hora_fin = f"{(current_time+interval) // 60}:{(current_time+interval) % 60}",
           dia = day,
           horario = horario
            )
      current_time += interval

except:
            return Response({"error": "Hubo un error al crear el grupo horario."},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Una vez realizadas todas las comprobaciones, se procede a crear la cuadrícula horaria u *Horario* perteneciente a ese *Grupo Horario*. Una vez creado, se debe crear también todas las *Franjas* que contendrá este *Horario*.

```
try:
asignaturas_titulacion = Titulacion_Asignatura.objects.filter(titulacion = grupohorario.titulacion)

for item in asignaturas_titulacion:
   if item.asignatura.cuatrimestre == grupohorario.cuatrimestre and item.asignatura.curso == horario.curso:
asignatura_titulacion = Titulacion_Asignatura.objects.filter(asignatura = item.asignatura)
if asignatura_titulacion.count() > 1:
   grupos_asignatura_repeat = Grupo.objects.filter(asignatura = item.asignatura)
for grupo_repeat in grupos_asignatura_repeat:
```

```
    franjas_grupo_horarios = Franja_Grupo.objects.filter(
        grupo = grupo_repeat, franja__horario__grupo_horario__anho = grupohorario.anho,
franja__horario__grupo_horario__cuatrimestre = grupohorario.cuatrimestre
    )
if franjas_grupo_horarios:
    for franja_grupo_horario in franjas_grupo_horarios:
        franja = franja_grupo_horario.franja
if franja.horario.hora_inicio == horario.hora_inicio and \
franja.horario.hora_fin == horario.hora_fin and \
franja.horario.minuto_inicio == horario.minuto_inicio and \
franja.horario.minuto_fin == horario.minuto_fin and \
franja.horario.intervalo_hora == horario.intervalo_hora and \
franja.horario.intervalo_minutos == horario.intervalo_minutos:
franja_equivalente = Franja.objects.get(
    horario = horario,
    dia = franja.dia,
    hora_inicio = franja.hora_inicio,
    hora_fin = franja.hora_fin
)
if franja_equivalente:
    if not Franja_Grupo.objects.filter(franja = franja_equivalente, grupo = grupo_repeat).exists():
Franja_Grupo.objects.create(
    franja = franja_equivalente,
    grupo = grupo_repeat
)
except Exception as e:
print(f"Error inesperado: {e}")

return Response('OK')
```

Por último, si en este *Horario* existen asignaturas que son comunes para otras titulaciones, y por lo tanto estarán presentes en otros horarios, se deben recuperar y añadir a este *Horario* que se acaba de crear. Este paso es muy importante para mantener la integridad y la lógica de los datos que confirman cada *Horario*.

5.1.2 Descripción técnica en el Front-End

Para comenzar con el desarrollo del *Front-End* se hace uso de una herramienta de construcción de proyectos. Existen varias alternativas, pero para usar cualquiera de ellas se

debe instalar *Node.js*[26] para hacer uso de **npm**[27] (*Node Package Manager*). Entre las principales herramientas de construcción rápida en React se encuentran **CRA**[28] (*Create React App*) o **Vite**[29]. Los motivos de la elección de esta última herramienta son los siguientes:

- Mayor velocidad de construcción y arranque, Vite hace uso de *ESBuild*[30], notablemente más rápido en tiempos de carga e inicio que *Webpack*[31] de *CRA*.
- En caso de querer generar *bundles* para sacar a producción la aplicación web, *Vite* realiza el *build* del proyecto de forma más rápida.
- Ambas ofrecen **HMR**[32] (*Hot Module Replacement*) que permite el refresco automático en cambios de estilo *CSS* o componentes, pero *Vite* lo incorpora de forma más eficiente que *CRA*.
- Además, *Vite* ofrece soporte para la construcción de más tecnologías, no está únicamente ligada a *React*.

Por lo general, *Vite* es una opción más eficiente en etapas de desarrollo que *CRA*, aunque ambas son las herramientas más populares para la creación de proyectos en React, se ha optado por *Vite* para la construcción del proyecto.

---

[26] Entorno de programación basado en JavaScript del lado del servidor, implementa funcionalidades como npm para el desarrollo del Front-End.

[27] Sistema de gestión de paquetes por defecto en Node.JS para controlar el manejo de dependencias en el Front-End.

[28] Interfaz de línea de comandos para la creación rápida y sin configuraciones adicionales de un proyecto React.

[29] Herramienta de creación de proyectos de múltiples tecnologías como React.

[30] Empaquetador y minificador moderno utilizado por Vite.

[31] Empaquetador de módulos altamente configurable utilizado por CRA.

[32] Funcionalidad que permite la actualización de módulos en tiempo de ejecución sin recarga del navegador.

Para crear un proyecto con *Vite* basta con ejecutar el siguiente comando en la ubicación que se desee: *npm create vite@latest*. Comenzarán una serie de preguntas para configurar el proyecto junto con el lenguaje que se quiera usar (*JavaScript* o *TypeScript*), en este caso se desarrollará en *React* con *JavaScript*. Uno de los principales temas para tener en cuenta cuando se comienza un desarrollo es la jerarquía y estructura de los archivos.

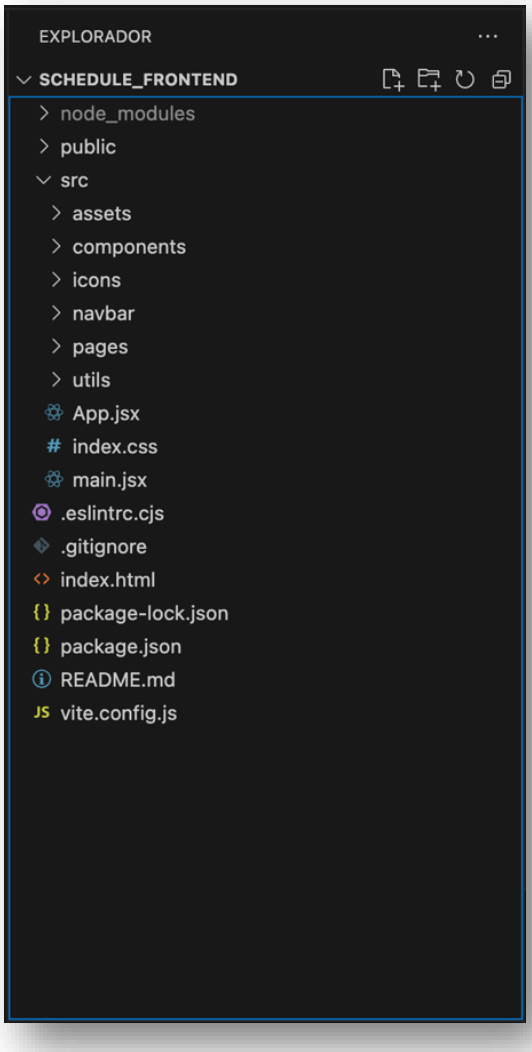


Ilustración 14: Organización del proyecto Front-End

Por defecto, *Vite* genera una carpeta *node_modules* con todos los módulos y librerías necesarias para que el proyecto se ejecute correctamente, esta carpeta es la más pesada y no es necesaria incluirla en ningún repositorio.

Los archivos *package.json* son de suma importancia, en ellos se definen las librerías utilizadas junto a sus respectivas versiones, además de los script a ejecutar mediante comandos rápidos. Este archivo es fundamental para el despliegue en otros sistemas.

El archivo *index.html* es la raíz *HTML* del proyecto, en él se renderizarán los diferentes componentes de *React* siguiente el patrón *SPA* (*Single Page Application*).

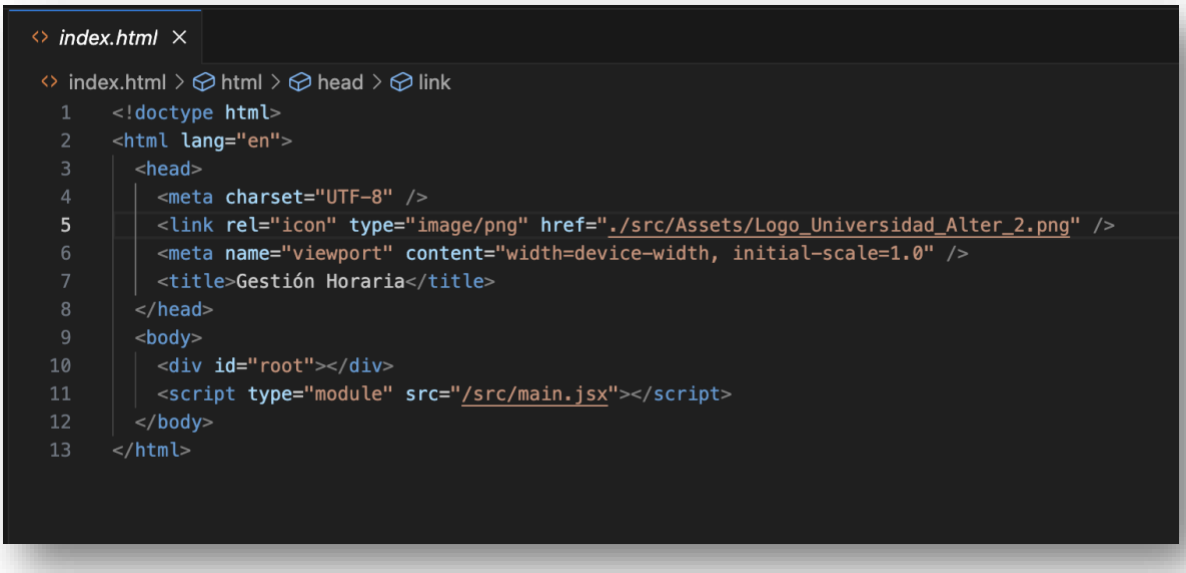


Ilustración 15: Elemento raíz HTML

README.md incluye información detallada del proyecto en Markdown[33], *vite.config.js* incluye configuraciones de *Vite* y *gitignore* especifica lo que se excluye en la subida al repositorio.

En la carpeta *src* se encontrará el núcleo del frontal, la distribución en esta carpeta queda pendiente del enfoque estructural que se quiera dar. En este caso se ha optado por una estructura por funcionalidad o modular, en la que se dividirá el código dependiendo de la función que realice en el proyecto.

Todas las pantallas que aparecen en la aplicación web se incluirán en la carpeta de *pages*. Cada pantalla cuenta con una serie de componentes, los más reutilizados se incluirán a su vez en la carpeta *components*. Existen archivos de utilidad como la definición de rutas para peticiones al *Back-End*, la creación de rutas privadas que requieran de autenticación o englobar las llamadas *Axios*[34] en un mismo componente, para todos ellos existe la carpeta *utils*.

[33] *Lenguaje de marcado ligero basado en la máxima legibilidad.*

[34] *Dependencia que ofrece un cliente HTTP basado en promesas para tecnologías web.*

El resto de las carpetas se usan para almacenar elementos gráficos de la aplicación como pueden ser iconos en *SVG* o imágenes.

Una vez definida la estructura del proyecto en *React*, se hace uso de librerías importantes en el desarrollo web que brindan soporte a *React* y aportan funcionalidades esenciales, a continuación, se detallan cada una de ellas:

- *MUI* o **Material UI** es una biblioteca que brinda componentes predeterminados en *React*. Inspirada en un estilo minimalista, *MUI* ofrece componentes reutilizables que siguen un diseño común altamente personalizable a las necesidades específicas del proyecto. Esto dará un aspecto más moderno y en concordancia con los estilos más actuales. Aún así, *Material UI* permite el uso de *CSS* para personalizar cada componente de forma específica. (*Material UI: React Components That Implement Material Design*, s. f.)

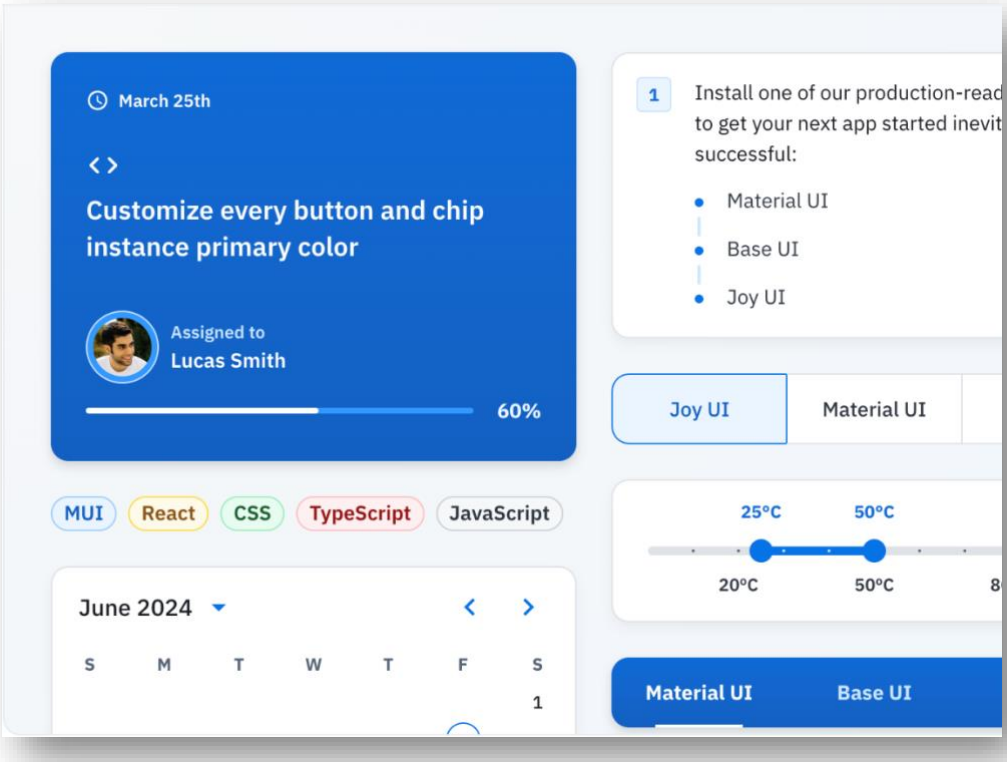


Ilustración 16: Componentes ejemplo de Material UI (https://mui.com/)

- **React Router DOM** es una librería que permite manejar la navegación entre pantallas de la aplicación web. Con esta biblioteca se podrá definir las diferentes rutas con las que cuenta un dominio y manejar esta jerarquía de enrutamiento de forma sencilla. (*Home v6.23.1*, s. f.)

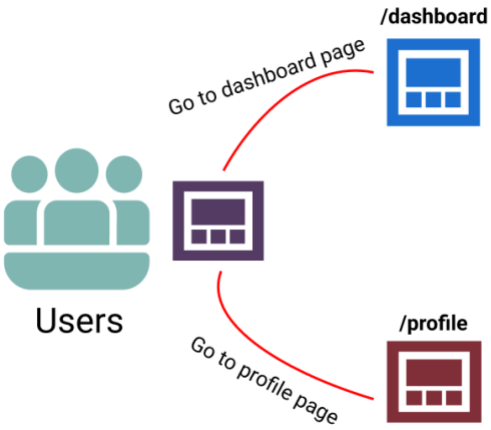


Ilustración 17: Routing en React (https://medium.com/nerd-for-tech/what-is-the-difference-between-react-router-and-conventional-routing-9b11159d92a4)

- **Axios** es la librería por excelencia para realizar peticiones *HTTP*. El *Back-End* está trabajando como una *API RESTful*, es indispensable que en el frontal se cuente con un método para realizar peticiones. Existen varios métodos para realizar peticiones desde el *Front-End*, otra alternativa era *Fetch*, pero debido a que la sintaxis de *Axios* es más limpia y sencilla, que además transforma automáticamente la información que se obtiene como respuesta y que soporta datos binarios como *blobs* para la descarga de *PDF*, *HTML* y *CSV*, se ha optado por esta opción. (*Axios Docs*, s. f.)

Ilustración 18: Petición utilizando Axios

- **Leaflet** es una biblioteca que permite el despliegue de mapas de forma sencilla si se proporcionan unas coordenadas. Como en la aplicación se hace referencia a diferentes espacios de la Universidad de Jaén como campus o

edificios, esta funcionalidad extra es perfecta para incluirla en el proyecto. (*Leaflet — An Open-source JavaScript Library For Interactive Maps*, s. f.)

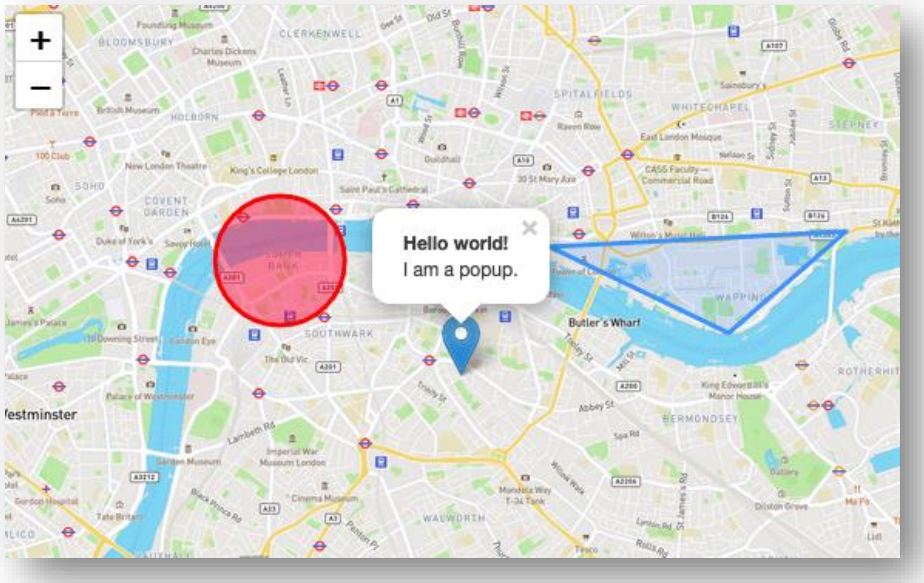


Ilustración 19: Mapa usando Leaflet (https://leafletjs.com/)

Dentro de la carpeta *src* existen dos archivos importantes que vienen por defecto en la creación de proyecto con Vite. *Main.jsx* es el encargado de coger el *div* raíz del índice *HTML* y renderizar el primer componente en React, el cual será *App.jsx*. En este primer fichero *main.jsx* se agregan los temas que tomará la aplicación web. *MUI* permite generar temas de forma sencilla definiendo paletas de colores, tipografías, tamaño de la fuente, entre otras. Para definir un tema se hará uso de *createTheme*, este se aplicará al componente *App* gracias al *wrapper* llamado *ThemeProvider*. *CSSBaseline* unifica estilos para diferentes navegadores e intenta que los estilos sean lo más parecidos posible entre entornos.

```
const theme = createTheme({
  palette: {
    type: 'light',
    primary: {
      main: '#00a65d',
    },
    secondary: {
      main: '#ddb10a',
    },
  },
  typography: {
```

```
    button: {
      fontSize: '1.75vmin',
      fontWeight: '600'
    },
  },
});

ReactDOM.createRoot(document.getElementById('root')).render(
  <>
    <CssBaseline />
    <ThemeProvider theme={theme}>
      <App />
    </ThemeProvider>
  </>
)
```

El primer componente *App* se encarga de todo el enrutamiento que ofrece *React Router DOM* para la navegación y el anidamiento de rutas. Para comenzar a usar esta librería se necesita definir un *router*, básicamente un *router* es un elemento que anida rutas y les asigna una pantalla o página de la aplicación web. También ofrece métodos específicos para el manejo de rutas. En este caso se empleará *CreateBrowserRouter* por su limpieza en la declaración y un anidamiento más claro y robusto.

```
  const router = createBrowserRouter([
    {
      path: "/",
      element: <LandingPage />,
    },
    {
      path: "/login",
      element: <LoginPage setUser={setUser} />,
      async action() {
        setUser(false);
      },
    },
    {
      path: "/schedules",
      element: <MainHorariosPage />,
```

```
    },
    {
      path: "/home",
      element:
        <PrivateRoute user={user}>
          <HomePage user={user} />
        </PrivateRoute>,
      children: [
        {
          path: "/home",
          element: <MainPage />,
        },
        {
          path: "/home/titulaciones",
          element: <TitulacionesPage />,
        },
      ]
    },
]);

return (
  <RouterProvider router={router} />
);
```

Finalmente, se incluye un *wrapper* con el componente *RouterProvider* proporcionando el *router* que se acaba de crear para dar vida a la aplicación web, esta será

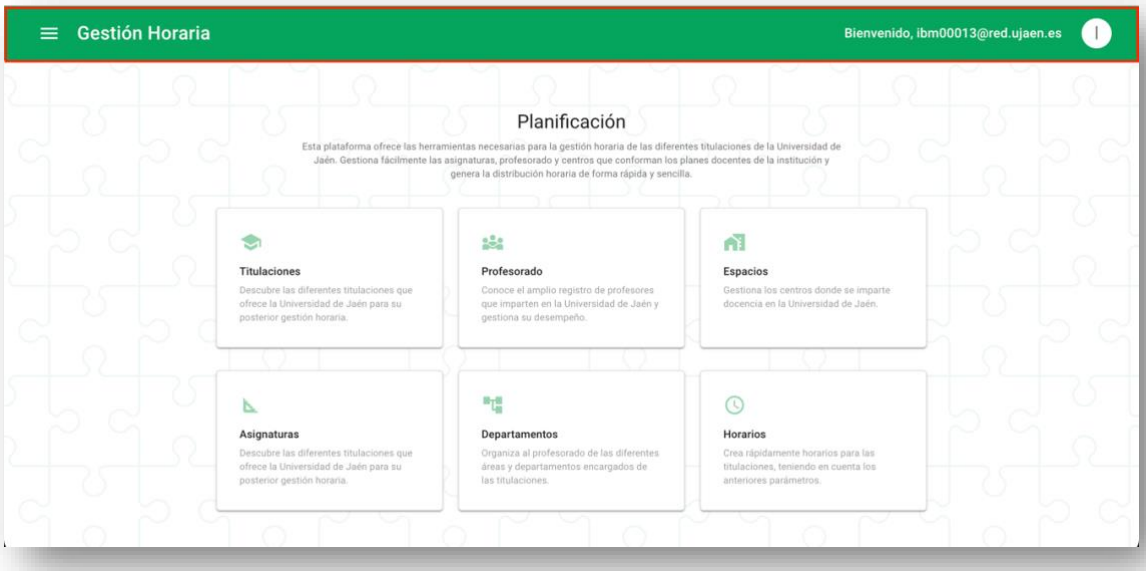


Ilustración 20: NavBar principal de la plataforma

la base principal del proyecto. Cabe destacar que el acceso autorizado a ciertas pantallas queda protegido con el componente *PrivateRoute*, que limita el acceso únicamente si se ha iniciado sesión correctamente.

Una vez se inicie sesión en el sistema y se esté autenticado, se mantendrá de manera fija una barra de navegación superior que despliega un menú en el lado izquierdo de la pantalla, en el espacio sobrante irán renderizándose las diferentes páginas de la aplicación.

En el menú lateral se observan las diferentes opciones de navegación y pantallas que conforman la funcionalidad de la aplicación.

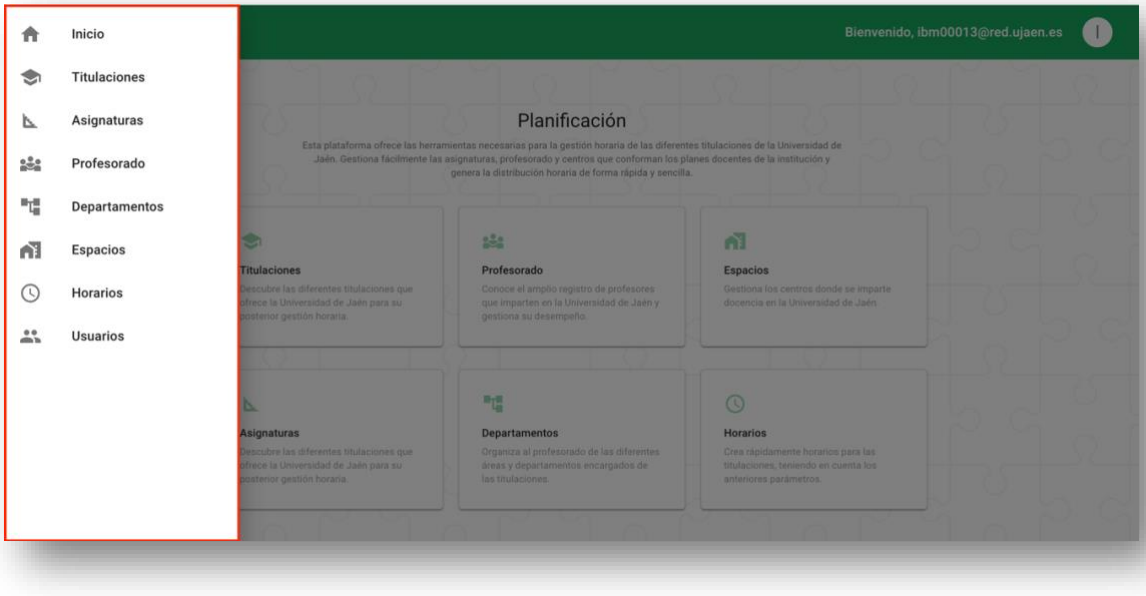


Ilustración 21: Menú principal de la plataforma

A partir de ahora, se procederán a definir las pantallas que componen la aplicación, por lo que se van a mostrar algunos ejemplos de código, con los componentes más utilizados de *Material UI* y la interfaz de usuario resultante.

- **Container** es un componente de *MUI* muy utilizado para ser el elemento raíz en el *return* de un componente funcional mayor y agrupar todo el contenido de este. Para cada componente de *Material UI* se puede agregar una clase y modificar el estilo directamente en un archivo *CSS* o *SASS*. La opción que brinda *MUI* es que, si no existen atributos específicos de ese componente para su modificación, se haga uso de *sx* y se defina dentro todo el estilo en *CSS* de dicho componente.

```
<Container maxWidth='false'
        className="backgroundDetails"
```

```
        sx={{
                width: '100%',
                height: '100%',
                display: 'flex',
                justifyContent: 'center',
                alignItems: 'flex-start',
                paddingTop: '4vmin',
}}>
```

- **Grid** y **Stack** hacen referencia a bloques con la propiedad *display* con valor *Grid* o con valor *Flex*. Estos dos tipos de posicionamiento son esenciales en desarrollo web y todo elemento que se quiera colocar dentro de una vista debería estar posicionada acorde con estos métodos.

```
<Grid item xs={2}>
        <Stack direction={'row'} justifyContent={'center'} alignItems={'center'} sx={{
                width: '12vmin',
                height: '16vmin'
        }}>
{…}
        </Stack>
</Grid>
```

- **Box** es otro componente clave en *Material UI*, hace referencia a un *div* en *HTML*. Estas etiquetas son fundamentales en desarrollo web, sirven de contenedores para otro tipo de elementos y hacen la función de maquetación de la vista. Además, al no incluir estilo se personalizan fácilmente con *CSS*.

```
<Box sx={{
        width: '100%',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        gap: '1vmin',
        overflowY: 'scroll',
        p: '1vmin'
}}>
```

```
{…}
</Box>
```

- Gracias a **Typography** se pueden incluir diferentes tipografías en la plataforma, ya sean cabeceras, subtítulos o texto común. Son altamente personalizables gracias a los atributos que brinda *MUI* y a la propiedad *sx*.

```
<Typography variant="h6" sx={{
        width: '100%',
        fontSize: '4vmin',
        textAlign: 'center',
        alignSelf: 'center',
        marginRight: '8vmin',
}}>Horarios</Typography>
```

- **FAB** es un botón de posicionamiento absoluto que se suele colocar en las esquinas inferiores derecha o izquierda y habilitan una funcionalidad. Con la propiedad *onClick* se puede controlar la llamada a una función cada vez que se pulse sobre este componente botón. Se puede tanto abrir diálogos, como navegar a otras páginas o hacer llamadas a la *API*.

```
<Fab color="primary" aria-label="add"
        sx={{ position: 'absolute', bottom: 20, left: 20 }}
        onClick={() => {
                setOpenAddAsignatura(!openAddAsignatura);
        }}>
                <AddIcon />
</Fab>
```

- **SnackBar** y **Alert** son avisos que se muestran en la interfaz para avisar al usuario final de un suceso. Cuando la persona que interactúa con la aplicación web llega a un error debido a que ha introducido unos datos de forma errónea o ha accedido a una vista a la que no tiene permisos, se muestran estas pequeñas alertas que aportan información al usuario final, enriqueciendo la interfaz y experiencia de usuario. MUI brinda atributos que se pueden agregar al componente para indicar si se desea que se oculte automáticamente después

de cierto valor de tiempo, si se quiere incluir una animación cuando aparezca la alerta o el posicionamiento de esta, entre otros.

```
<Snackbar
        open={alert.open}
        autoHideDuration={1000}
        onClose={handleCloseAlert}
        anchorOrigin={{ vertical: 'top', horizontal: 'right' }}
        TransitionComponent={SlideTransition}
>
        <Alert onClose={handleCloseAlert} severity={alert.severity}>
                {alert.text}
        </Alert>
</Snackbar>
```

A continuación, se procede a visualizar una página completa de la plataforma. Esta vista contiene una tabla formada por datos de asignaturas que se imparten en la Universidad de Jaén. Dicha tabla la ofrece *Material UI* a modo de plantilla, simplemente se deben recoger los datos directamente de la *API RESTful* y modificar la estructura de la tabla para que se ajuste con la cabecera y campos. Para la comunicación con el *Back-End* se deben definir llamadas o peticiones que se irán reutilizando a lo largo de la aplicación web cada vez que se necesite una funcionalidad específica.

```
import axios from "axios";
import config from "./RouteConfig";

const caller = axios.create({
   baseURL: config.apiUrl,
   timeout: 10000,
   headers: {
      'Content-Type': 'application/json',
   },
});

export const getCall = async (url) => {
   try {
      const response = await caller.get(url);
```

```
    console.log(response);
    return response.data;
  } catch (error) {
    console.error(error);
    throw error;
  }
};
```

Aquí se muestra un ejemplo de definición y uso de una petición de tipo *GET*, la cual no requiere de datos adicionales en un *body*, como pasaría con un tipo *POST*. Al definir esta llamada con *Axios* en un componente reutilizable, se puede hacer uso de ella desde cualquier parte de la aplicación web y comunicarse así con la *API RESTful*.

```
async function getAsignaturas() {
  try {
    const asignaturasData = await getCall(config.asignatura);
    setAsignaturas(asignaturasData);
  } catch (error) {
    console.error('Error al obtener Asignaturas:', error);
    handleErrorAlert();
  }
}
```

Al recuperar datos de la *API RESTful* se pueden almacenar en estados de *React*, estos estados guardan información en el ciclo de vida de un componente, se denominan *hooks*. Cada vez que un *hook* de tipo *useState* cambia de estado en *React*, se produce una renderización en el *DOM Virtual* que ofrece *React.* Por lo que sólo aquellas partes que se modifiquen realmente en un renderizado nuevo pasarán a modificarse en el *DOM* original.

```
return (
        <TableRow
                hover
                onClick={(event) => handleClick(event, row.id)}
                role="checkbox"
                aria-checked={isItemSelected}
                tabIndex={-1}
                key={row.id}
```

```
                selected={isItemSelected}
                sx={{ cursor: 'pointer' }}
            >
                <TableCell padding="checkbox">
                    <Checkbox
                        color="primary"                                    checked={isItemSelected}
                        inputProps={{'aria-labelledby': labelId,}}
                    />
                </TableCell>
                <TableCell
                    component="th"
                    id={labelId}
                    scope="row"
                    padding="none"
                >
                    {row.denominacion}
                </TableCell>
                <TableCell align="left">{row.tipo}</TableCell>
                <TableCell align="left">{row.creditos + ' ECTS'}</TableCell>
                <TableCell align="left">{row.curso + 'º'}</TableCell>
                <TableCell align="left">{row.cuatrimestre}</TableCell>
            </TableRow>
    );
```

Finalmente, esta es una ejemplificación de una pantalla finalizada usando todos los componentes explicados previamente.

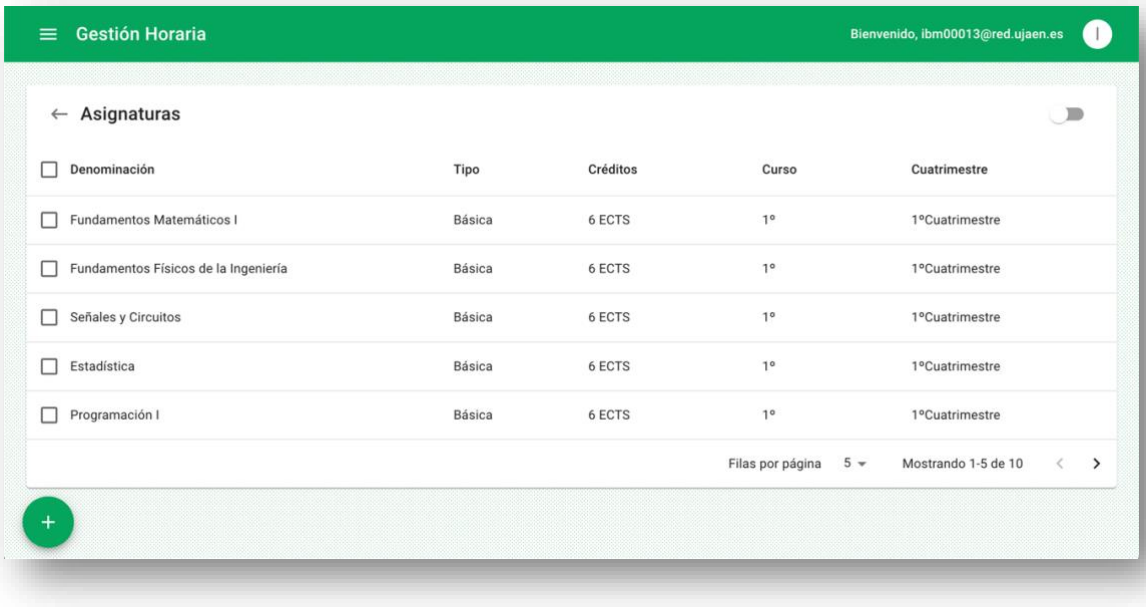


Ilustración 22: Tabla para mostrar información

5.2 Interfaz y Experiencia de Usuario

La interfaz y experiencia de un usuario final ante una aplicación web es de suma importancia, es fundamental que la persona que utilice la plataforma lo haga de manera satisfactoria. Lograr un desarrollo sobresaliente en aspectos *UI/UX*[35] *(User Interface/User Experience)* es lo que hace que una plataforma sea atractiva y no existe una mejor manera que captar la atención del usuario final a través de la presentación visual. Para ello, se han buscado una serie de características y claves a seguir para conseguir este objetivo:

- **Minimalismo**. Las interfaces claras, con estilos minimalistas y bien definidos hacen que una aplicación web se sienta limpia y facilita la familiaridad del usuario con esta.
- **Intuitividad**. Viene ligado con el apartado anterior, una interfaz simple pero efectiva proporciona al usuario una sensación de conocer el funcionamiento sin haberla usado anteriormente. Un ejemplo de esto es la pantalla de inicio (*Landing)* de la aplicación, es una primera toma de contacto del usuario final con la interfaz y se entiende de forma rápida la utilidad de esta y los pasos a seguir.

[35] *Siglas que hacen referencia a la interfaz de usuario y a la experiencia de usuario en términos de desarrollo web.*

Ilustración 23: Página de inicio de la aplicación web

- **Accesibilidad**. Una aplicación web debe ser accesible para todos los usuarios que interactúen con ella. Es por ello por lo que se incluye el uso de diferentes técnicas como la implementación de texto alternativo para las imágenes, la navegación por teclado, el correcto contraste de colores en todas las pantallas de la plataforma, entre otros. Estas medidas permiten que todo usuario se beneficie de la aplicación web sin ninguna barrera.
- **Rendimiento**. Es importante que el flujo de la plataforma sea fluido, sin altas demoras en tiempos de carga ni situaciones en las que la interfaz quede congelada o sin respuesta. Esto proporciona una sensación de robustez al usuario final y brinda calidad a la plataforma.

- **Retroalimentación**. Es muy importante que la interfaz responda a las acciones del usuario e informe sobre ellas, sobre todo cuando se comete un fallo durante el uso de la plataforma.

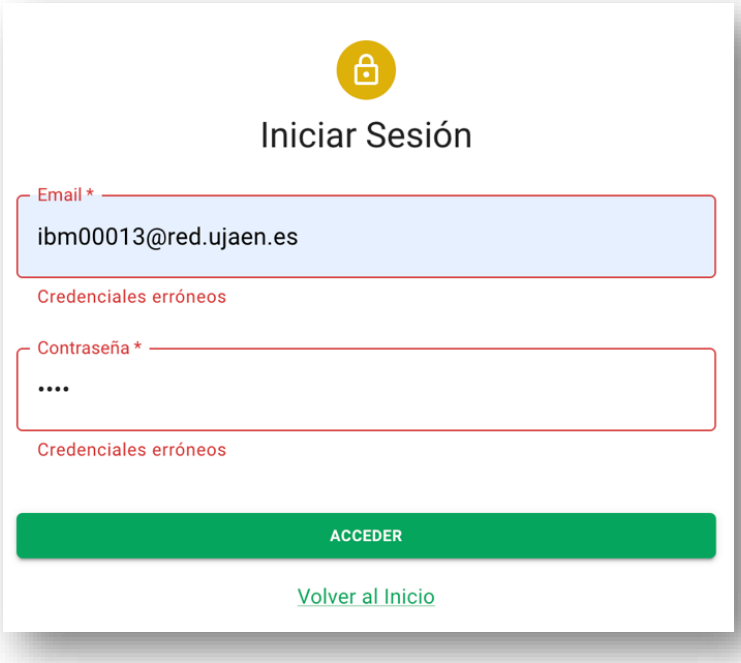


Ilustración 24: Retroalimentación en la pantalla de inicio de sesión

Si por ejemplo se comete un error o se realiza satisfactoriamente una acción, es importante que el sistema alerte de ello, para aclarar al usuario lo que está sucediendo y cómo proseguir.

Esto se ejemplifica claramente con un acceso erróneo en la autenticación de la aplicación, o con la creación de algún elemento con datos faltantes como se muestra en la Ilustración 25: Retroalimentación con alertas.

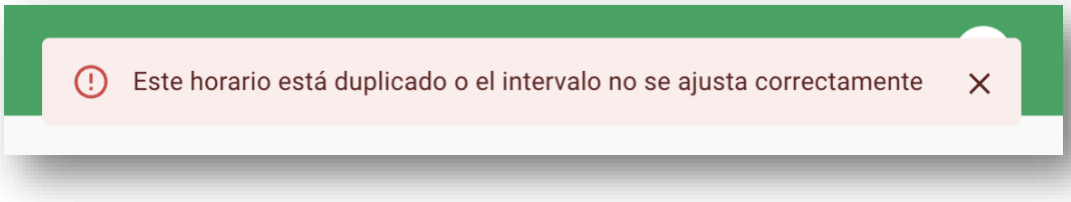


Ilustración 25: Retroalimentación con alertas

- **Diseño Responsivo**. A la hora de agregar estilos en la interfaz de usuario, se busca utilizar medidas como *porcentajes* (*%*) o *vmin/vmax*. Así se asegura que dependiendo del tipo de pantalla dónde se visualice la interfaz lo haga de forma similar. Por ejemplo, en la pantalla principal (*Home)* de la aplicación, si la ventana es demasiado estrecha, los elementos pasan a listarse uno encima de otro.

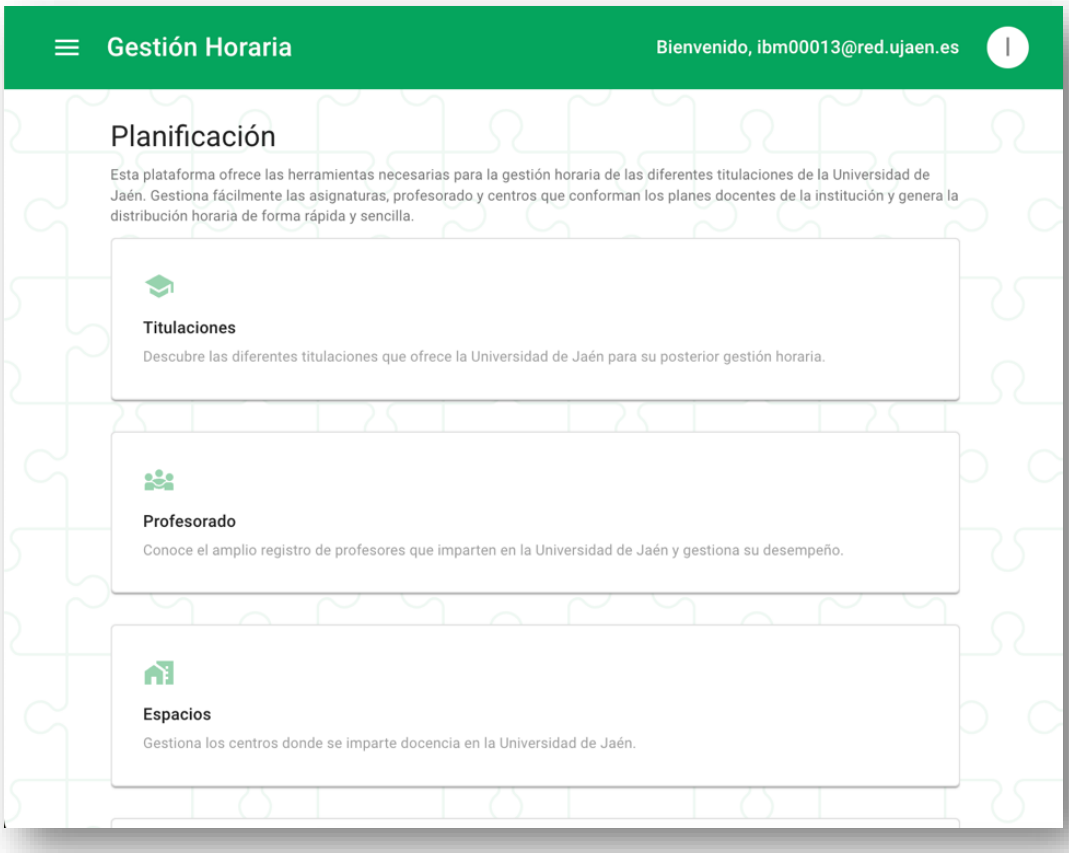


Ilustración 26: Diseño responsive en la pantalla principal

- **Menús**. Toda interfaz web necesita de una buena navegación, es por eso por lo que la maquetación inicial que se explicaba en el apartado de detalles técnicos del Front-End se agregaba un menú desplegable que facilita esta característica.
- ***Cross Browser Testing***. Es importante corroborar que la aplicación se visualice correctamente independientemente del navegador que utilice el usuario final.

- **Elementos Interactivos**. Toda aplicación web debe contener alguna animación que alivie la sensación de una interfaz estática, en este caso se agregan componentes interactivos como *Tooltips* para dar una experiencia dinámica al usuario final.

Ilustración 27: Cuadrícula horaria con Tooltips

5.3 Desafíos y Soluciones

A lo largo del desarrollo software pueden surgir problemáticas que no se contemplaron en etapas iniciales de planificación y análisis. Por lo que este apartado trata de definir los diferentes desafíos que han surgido a lo largo de la etapa de codificación.

- Uno de los principales desafíos con los que se cuenta cuando se inicia un proyecto de gestión horaria es el **solapamiento** tanto de espacio como unipersonal. No es posible que, en un mismo habitáculo, o aula en este caso, se impartan dos grupos al mismo tiempo. Este impedimento se solventó posteriormente a la creación de cuadrículas horarias, al agregar un *Horario* dentro de un *Grupo Horario* no se comprobaba si el *Aula* de ese *Grupo* o si el *Profesor* que impartía docencia estaban ocupados a esa misma hora. Por lo que se tuvo que modificar el microservicio de agregación horaria. A partir de este punto, cada vez que se agregaba un *Horario* se comprobaban todas las *Aulas* y *Profesores* de ese mismo curso académico y cuatrimestre.
- La **simultaneidad** de *Asignaturas* en varias *Titulaciones* es una problemática a la hora de la gestión horaria. Es importante controlar la unificación de los diferentes horarios que cuentan asignaturas en común. Si una *Asignatura* es común en varias *Titulaciones*, en los *Horarios* de dichas titulaciones deberá aparecen el mismo día y a la misma hora. De igual forma, si se produce una

modificación en alguna de estas asignaturas comunes, deberá verse reflejado en todos los *Horarios* de las diferentes *Titulaciones*.

- La **importación** horaria era un requisito inicial que se implementó en las últimas instancias del desarrollo. Esta utilidad era fundamental para obtener la cuadrícula de cursos pasados y desplegarla de forma rápida. Además, cada cuadrícula puede **desplazarse** lateralmente a la izquierda y/o derecha para modificar de forma rápida horarios, debido a que muchas titulaciones reaprovechan horarios de cursos pasados y los desplazan lateralmente en cada curso académico. Este requerimiento supuso la agregación de nueva funcionalidad y rutas a la *API RESTful.*
- La **redundancia** de datos fue corregida en últimas instancias del desarrollo. Es importante que no existan horarios duplicados para una titulación de un mismo curso académico y cuatrimestre. Este error no estaba contemplado y suponía una falla en algunas funcionalidades del *Back-End*. Gracias a esta corrección se asegura la integridad de la aplicación.

6 Pruebas y validación

Una vez finalizada la etapa de desarrollo y codificación, queda la última fase para la correcta finalización del proyecto. Se deben realizar pruebas y validar el correcto funcionamiento de la aplicación web. Esta fase es de suma importancia, pues agrega fallas con las que no se contaba en un principio. La correcta depuración del código por medio de pruebas y validaciones le agregan solidez y robustez a la plataforma. A continuación, se detallan una serie de **pruebas** o *tests* que se realizaron a la aplicación web en busca de posibles errores o fallas. (*La Importante de las Pruebas de Software*, 2022)

- Como se especificaba en el apartado Interfaz y Experiencia de Usuario, es crucial probar la aplicación en diferentes navegadores puesto que no se conoce de antemano cual usará el usuario final. La aplicación deberá responder con semejanza, independientemente del navegador que se utilice, logrando todas sus funcionalidades sin problemas. A esta prueba se la conoce con el nombre de *Cross Browser Testing*.
- Otro de los errores típicos de *UI/UX* es la redimensión de pantalla. El contenido de la interfaz debe amoldarse a cualquier dimensión de pantalla o ventana que el usuario final establezca. Esto otorga gran solidez a la experiencia de usuario como ya se comentaba en el apartado de *UI/UX*. El diseño responde satisfactoriamente al redimensionamiento de ventana, gracias a la utilización de medidas como *porcentajes (%)* o *vmin/vmax.* Gracias a esto, se puede decir que el diseño de la aplicación web es *Responsive*.
- Se han realizado pruebas unitarias en cada función de la *API RESTful* con diferentes datos para corroborar el correcto funcionamiento de cada Endpoint por separado. Estas pruebas se realizan haciendo uso de herramientas como *Postman* y extensiones de *Visual Studio Code* como *Thunder Client.*
- Las pruebas de integración se realizan haciendo uso de la aplicación web, aquí se integran las diferentes llamadas a Endpoints y se verifica el correcto funcionamiento cuando se integran varias funcionalidades en un mismo entorno.
- Una vez completado el desarrollo, hay que verificar que los requisitos iniciales se cumplen, y que el sistema resultante cumple con los objetivos que se marcaron previamente. Estas pruebas de sistema se detallarán en el

siguiente apartado, y verifican la entrega de un producto final de acuerdo con unos requerimientos.

- Ante la presencia de desafíos como los de solapamiento, simultaneidad, importación y desplazamiento, se deben realizar pruebas de regresión para comprobar que las nuevas funcionalidades que se añadieron a la aplicación web no ocasionan fallas en partes que funcionaban con normalidad. Al implementar estos cambios, la plataforma se volvió a someter a un proceso de prueba para corroborar este apartado.
- Por último, se comprueba el rendimiento de la aplicación ante un uso exhaustivo de la misma. Gracias a esto, se asegura que no se produce ninguna carga de trabajo elevada ni tiempos de demora excesivos. Esto asegura un correcto rendimiento de la aplicación web.

7 Resultados

Al comenzar este TFG se definieron una serie de objetivos y requerimientos que deben cumplirse para la correcta finalización de este proyecto. El desarrollo concluye con una serie de funcionalidades que se traducen en logros, estos logros deben compararse con los requerimientos iniciales y suplir la necesidad que se definía en un comienzo.

- El objetivo principal es desarrollar una aplicación web capaz de crear y editar horarios de las diferentes titulaciones de la Universidad de Jaén. La plataforma logra gestionar horarios de forma idéntica a los que se publican en la web de la Universidad de Jaén, habilitando la creación y la edición de forma satisfactoria.
- El primer requerimiento era proporcionar un acceso autenticado a la plataforma. Se logra, no solo el despliegue de un acceso autenticado a la plataforma de gestión horaria, sino también un acceso público de consulta de los horarios ya creados.
- El segundo requerimiento citaba la presencia de roles en los diferentes usuarios que accedieran de forma autenticada a la plataforma. Se logra identificar cada usuario por un atributo rol, el cual puede ser *administrador* o *profesor*. Un *administrador* es capaz de gestionar el *CRUD* de todos los elementos que engloban la aplicación (titulaciones, asignaturas, profesores, etc.), gestionar los usuarios que acceden a la plataforma y asignarles un rol, y gestionar los diferentes horarios. Un *profesor* solo es capaz de gestionar horarios de las titulaciones en las que figure como responsable.
- Otro requerimiento que se logra es la importación de horarios de otros cursos académicos al actual. Con la posibilidad de desplazar lateralmente las columnas de cada horario.
- Se necesitaba una herramienta gráfica para la construcción de horarios, en la que se controlara el solapamiento tanto espacial como de personal docente. Se logra satisfactoriamente este requerimiento, proporcionando una cuadrícula gráfica y elementos de tipo *chip* que corresponden con los grupos que se imparten en una determinada celda. Antes de crear una cuadrícula, se comprueba este solapamiento.
- Se logra la definición de documentos que engloban los horarios de una titulación para un curso académico y cuatrimestre especificados. Estos

documentos se pueden acceder tanto de forma autenticada como pública. Adicionalmente, se habilita la descargar del documento horario en formato *PDF*, similar a la web de la Universidad de Jaén, en formato *HTML* para su integración en otras páginas web, o incluso en formato *CSV*.

Con estos logros se solventa el núcleo de objetivos y requerimientos que se especificaron en la documentación de partida, haciendo esta plataforma muy similar a la que se planteaba inicialmente.

8 Presupuesto

En este apartado se procede a desglosar el trabajo realizado en la aplicación web a través de las diferentes etapas que se han explicado en este documento. Cada etapa se divide en una serie de actividades principales que engloban la elaboración del proyecto.

Etapa	*Actividad*	*Horas Dedicadas*	*Base Imponible*	*IVA*	*Total*
Planteamiento	Planificación	*10 horas*	*250.0€*	*52.5€*	*302.5€*
-	Revisión de literatura y estado del arte	*15 horas*	*375.0€*	*78.75€*	*453.75€*
-	Estudio de tecnologías y herramientas	*35 horas*	*875.0€*	*183.75€*	*1058.75€*
Subtotal	-	*60 horas*	*1500.0€*	*315.0€*	*1815.0€*
Análisis	Estudio de requisitos funcionales y no funcionales	*12 horas*	*300.0€*	*63.0€*	*363.0€*
-	Estudio de la arquitectura del sistema	*15 horas*	*375.0€*	*78.75€*	*453.75€*
-	Estudio de casos de uso	*5 horas*	*125.0€*	*26.25€*	*151.25€*
Subtotal	-	*32 horas*	*800.0€*	*168.0€*	*968.0€*
Diseño	Diseño de la base de datos	*30 horas*	*750.0€*	*157.5€*	*907.5€*
-	Prototipado de la interfaz de usuario	*25 horas*	*625.0€*	*131.25€*	*756.25€*
Subtotal	-	*55 horas*	*1375.0€*	*288.75€*	*1663.75€*
Implementación	Implementación de la	*8 horas*	*200.0€*	*42.0€*	*242.0€*

	arquitectura del sistema				
-	Desarrollo Front-End	*55 horas*	*1375.0€*	*288.75€*	*1663.75€*
-	Implementación de la base de datos	*8 horas*	*200.0€*	*42.0€*	*242.0€*
-	Desarrollo Back-End	*57 horas*	*1425.0€*	*299.25€*	*1724.25€*
Subtotal	-	*128 horas*	*3200.0€*	*672.0€*	*3872.0€*
Pruebas	Pruebas de integración	*10 horas*	*250.0€*	*52.5€*	*302.5€*
-	Pruebas de usuario	*15 horas*	*375.0€*	*78.75€*	*453.75€*
Subtotal	-	*25 horas*	*625.0€*	*131.25€*	*756.25€*
Total	-	*300 horas*	*7500.0€*	*1575.0€*	*9075.0€*

El presupuesto de ejecución del presente Trabajo Fin de Grado asciende a la cantidad total de nueve mil setenta y cinco euros (9075.0 €).

9 Conclusiones y trabajo futuro

Este último apartado recoge una serie de conclusiones a las que se han llegado al recorrer todas las etapas definidas en el presente documento. Además, se define el trabajo futuro que haría más versátil, potente y configurable esta versión de la aplicación web.

9.1 Conclusiones

En primer lugar, se debe hacer una revisión global de objetivos y requisitos con los resultados obtenidos una vez se ha finalizado el desarrollo de la plataforma, como se indica en el apartado Resultados. La conclusión general a la que se llega es que se han cumplido satisfactoriamente los objetivos planteados al comienzo de este proyecto. La implementación de esta aplicación ha permitido una administración eficiente y efectiva de los horarios, proporcionando una herramienta robusta y funcional que responde a las necesidades y expectativas detalladas al inicio del proyecto.

Cabe destacar que el mayor índice de éxito a la hora del desarrollo de software recae en una buena planificación y un buen análisis para lograr los objetivos marcados. Si no se dedica el suficiente tiempo a un estudio previo para el sistema, este se verá comprometido una vez se comience con el desarrollo. Si se analiza correctamente cada parte del desarrollo, se lograrán resultados más eficientes con mayor facilidad.

Si se remarcara una parte indispensable del análisis, esta sería la estructuración de la base de datos sin lugar a duda. Más aún en aplicaciones que involucran elementos complejos como cuadrículas horarias. Si se define correctamente la implicación de cada objeto o tabla, y que éstos sean congruentes con el resultado que se persigue, se tendrá todo el trabajo hecho a posteriori.

En este proyecto se hacen uso de diferentes tecnologías, cada una de ellas se centran en aspectos diferentes del desarrollo de software. Es importante dedicar el tiempo suficiente para superar la curva de aprendizaje de cada tecnología, esto ahorrará tiempo en el futuro, aunque requiera de mayor pausa al comienzo de la codificación. Por ejemplo, *Material UI* requiere de una curva de aprendizaje para familiarizarse con todos sus componentes y los atributos individualizados que usa cada uno. Puede parecer más sencillo definir directamente todos tus componentes y las hojas de estilo para cada uno con tal de comenzar el desarrollo cuanto antes. Pero a la larga, una correcta utilización de una biblioteca o un Framework supone una ventaja notable, ya que esa es su funcionalidad y enfoque.

Al final, en el desarrollo de software profesional no se trabaja de forma aislada. La mayoría de las veces se formará parte de un equipo y se desarrollará en base a metodologías ágiles. En este caso, se sigue una metodología más tradicional, pero no se debe perder el

enfoque de lo que las metodologías más utilizadas aportan y seguir una perspectiva cíclica en el desarrollo permitirá agilidad a la hora de implementar nueva funcionalidad.

La gestión horaria es compleja y muy variada, dependiendo de la titulación a la que se enfoque dentro de la Universidad de Jaén, presentará sus particularidades específicas. La plataforma de gestión horaria se ha desarrollado siguiendo como ejemplo multitud de horarios publicados en la web de la Universidad de Jaén, y se adapta a la mayoría de las casuísticas que se han encontrado durante ese análisis. Bien es cierto, que cada vez son más las titulaciones que engloba la Universidad de Jaén, y pueden surgir requerimientos nuevos a la hora de gestionar horarios para ciertas titulaciones.

9.2 Trabajo futuro

El desarrollo de esta aplicación web no es un proceso cerrado, es objeto de mejora e implementación continua para mejorar su usabilidad y corregir errores. Es por eso por lo que se pueden citar claves o pasos a seguir en un futuro para lograr esto.

De cara al trabajo futuro que puede desarrollarse para esta aplicación web, de acuerdo con las necesidades que podría tener la Universidad de Jaén, está la mejora de la gestión horaria para que pueda abarcar, no solo titulaciones, sino másteres cuyo horario se asigna en intervalos semanales. Esta aplicación no está preparada para lograr cambios tan dinámicos en la planificación horaria, por lo que puede ser un foco de trabajo futuro ideal para implementar en la aplicación web y que continúe sirviendo de utilidad a la Universidad de Jaén.

Además, la codificación de esta aplicación se hace bajo una etapa de *desarrollo*, por lo que si se desea desplegar la plataforma de manera oficial se debe pasar a una etapa de *producción*. Para ello, habría que obtener un dominio para el despliegue del frontal y permitir la comunicación con la *API RESTful* a través de internet. Esto requiere especial atención a la seguridad tanto del *Front-End* como del *Back-End*, ya que al abandonar el entorno local de desarrollo son susceptibles a posibles ataques o accesos indeseados.

Por último, se debería monitorear el rendimiento de la aplicación e intentar lograr siempre la mínima latencia y demora en la misma. Una aplicación web no abarca nunca un desarrollo cerrado, y siempre puede estar sujeta a mejoras de interfaz, rendimiento, accesibilidad o nueva funcionalidad.

Bibliografía

[1] *Axios Docs*. (s. f.). Axios. https://axios-http.com/es/docs/intro

[2] Bahgat, A. (2023, 25 agosto). *Flask vs Django: Elijamos Tu Próximo Framework Python*. Kinsta. https://kinsta.com/es/blog/flask-vs-django/

[3] *Documentation for Visual Studio Code*. (s.f.). Visual Studio Code. https://code.visualstudio.com/docs

[4] *Documentación de Django*. (s. f.). Django Project. https://docs.djangoproject.com/es/5.0/

[5] *Django REST Framework*. (s. f.). Django REST Framework. https://www.django-rest-framework.org/

[6] *Home v6.23.1*. (s. f.). React Router. https://reactrouter.com/en/main

[7] *La importancia de las pruebas de software*. (2022, 26 agosto). UNIR. https://www.unir.net/ingenieria/revista/pruebas-software/

[8] Leaflet — an open-source JavaScript library for interactive maps. (s. f.). Leaflet. https://leafletjs.com/

[9] *Material UI: React components that implement Material Design.* (s. f.). Material UI. https://mui.com/material-ui/

[10] *Metodologías de desarrollo de software: ¿qué son?* (2020, 21 diciembre). Santander Open Academy. https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html

[11] Powell, Z. (2023, 20 agosto). *Angular vs React: Una Comparación En Profundidad*. Kinsta. https://kinsta.com/es/blog/angular-vs-react/

[12] *React*. (s. f.). React. https://es.react.dev/

[13] *SQLite Page*. (s. f.). SQLite. https://sqlite.org/

Anexo I: Manual de Usuario

Introducción

A continuación, se detalla una guía informativa para la correcta utilización y mantenimiento de la *plataforma* de **gestión horaria** de la Universidad de Jaén. El objetivo de este manual de usuario es brindar información básica sobre el funcionamiento de la interfaz, así como consejos y claves para facilitar su usabilidad en el menor tiempo posible.

Este documento va enfocado a los tres actores principales para los que se centra el uso de la aplicación. Tanto si se trata de un *Administrador*, como de un *Profesor* o *Alumno*, esta guía hará un recorrido sobre las principales funciones que habilita su rol.

Configuración Inicial

Para el acceso a la aplicación web se requiere de una conexión a *Internet*, además de un navegador web a su elección (Chrome, Safari, Mozilla, Edge, etc.). A través del navegador se accederá a la URL donde se encuentre desplegada la plataforma.

Para el acceso autenticado al sistema se requiere un usuario con rol de *Administración* o *Profesorado*. Si su intención es consultar algún horario puede acceder desde la parte pública de consultas de la aplicación. Para obtener un usuario autenticado debe contactar con la administración de la plataforma para gestionar un nuevo acceso según sus necesidades.

Acceso al Sistema

Existen dos modalidades de acceso a la plataforma de gestión horaria:

- Acceso público para la consulta
- Acceso autenticado para la gestión horaria

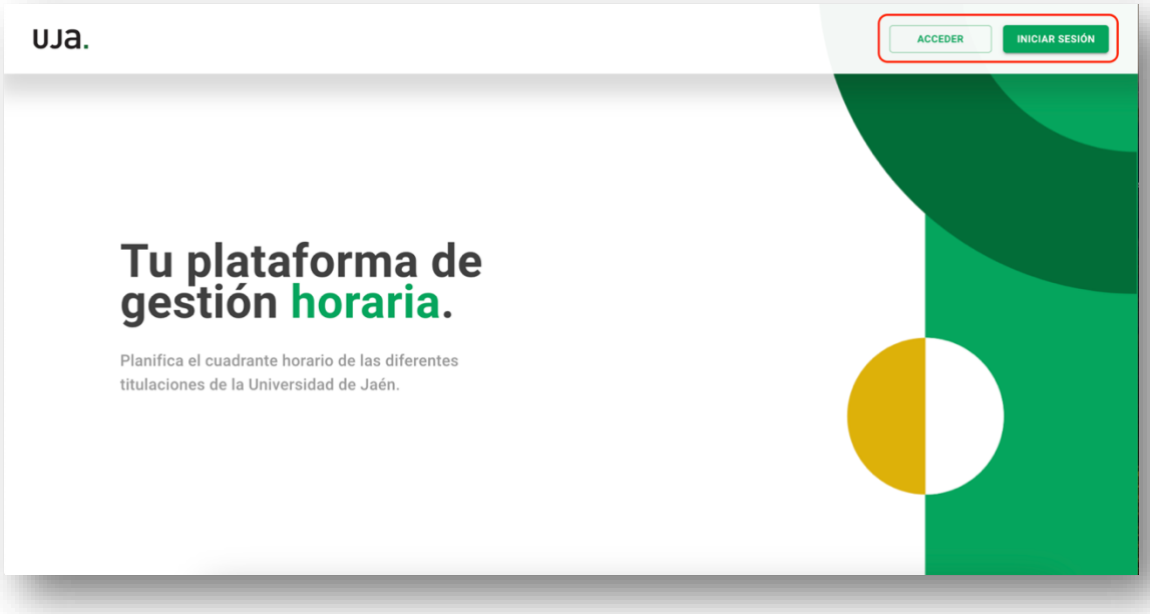


Ilustración 28: Página de inicio de la plataforma

Para la consulta libre de horarios se debe pulsar en *'Acceder'* desde la vista de inicio de la plataforma. Se producirá una redirección automática hacia un buscador específico para realizar la consulta del horario el cual se esté interesado. Para el acceso autenticado se debe *'Iniciar Sesión'*, se producirá una redirección hacia la vista de inicio de sesión de la aplicación, donde se deberá indicar las credenciales de usuario para acceder al sistema.

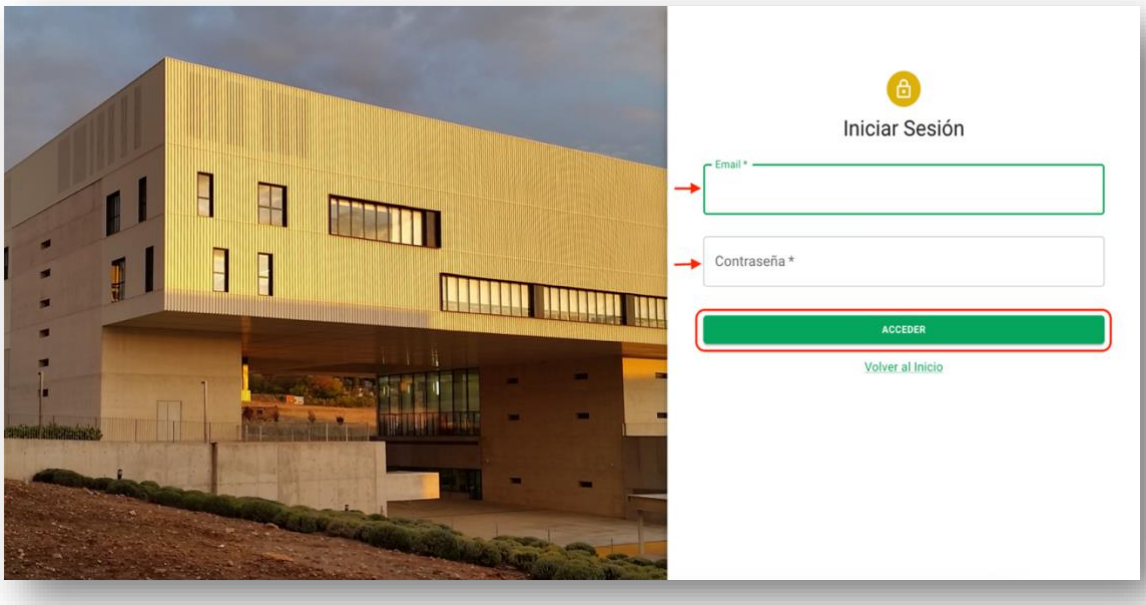


Ilustración 29: Página de inicio de sesión

Consulta Pública de Horarios

Si se accede al sistema por un método que no requiere autenticación se podrán consultar los diferentes horarios publicados en la plataforma.

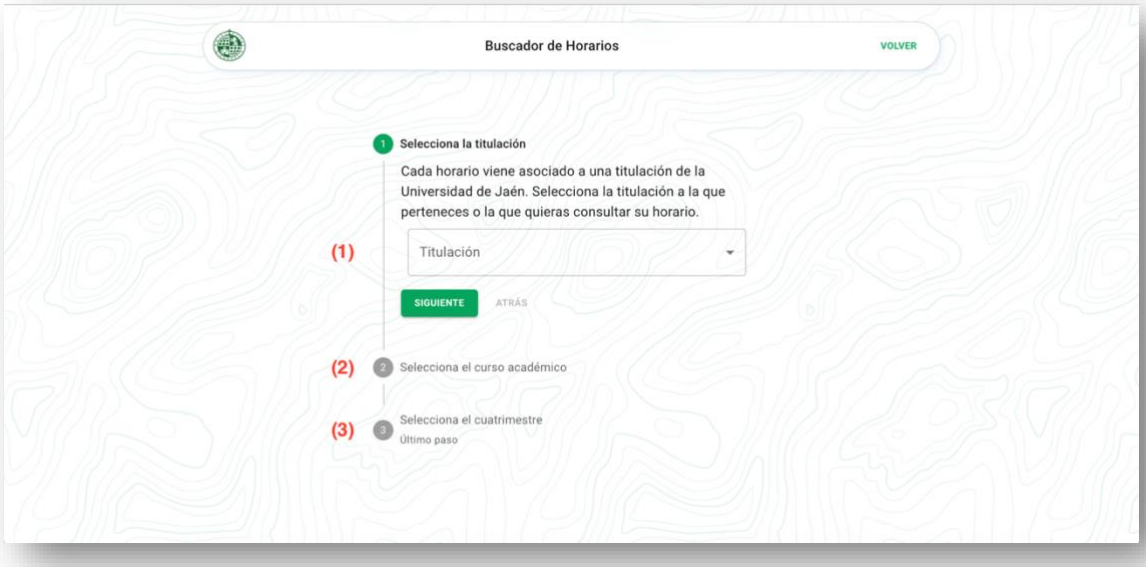


Ilustración 30: Filtros de consulta horaria

Al pulsar en *'Acceder'* se redirigirá a la página que se muestra en la Ilustración 30: Filtros de consulta horaria, en ella se deberá completar tres campos para iniciar la búsqueda de horarios de acuerdo con los filtros.

1. Se indica mediante un selector la Titulación de la que se quiere consultar los documentos horarios. Si no se indica ninguna se filtrarán todas las titulaciones.
2. Se indica mediante un selector el curso académico por el que se desean filtrar los documentos horarios.
3. Se indica mediante un selector el cuatrimestre por el que se desean filtrar los documentos horarios. Se puede filtrar por ambos cuatrimestres.

El resultado es un listado de documentos horarios que hayan coincidido con la búsqueda.

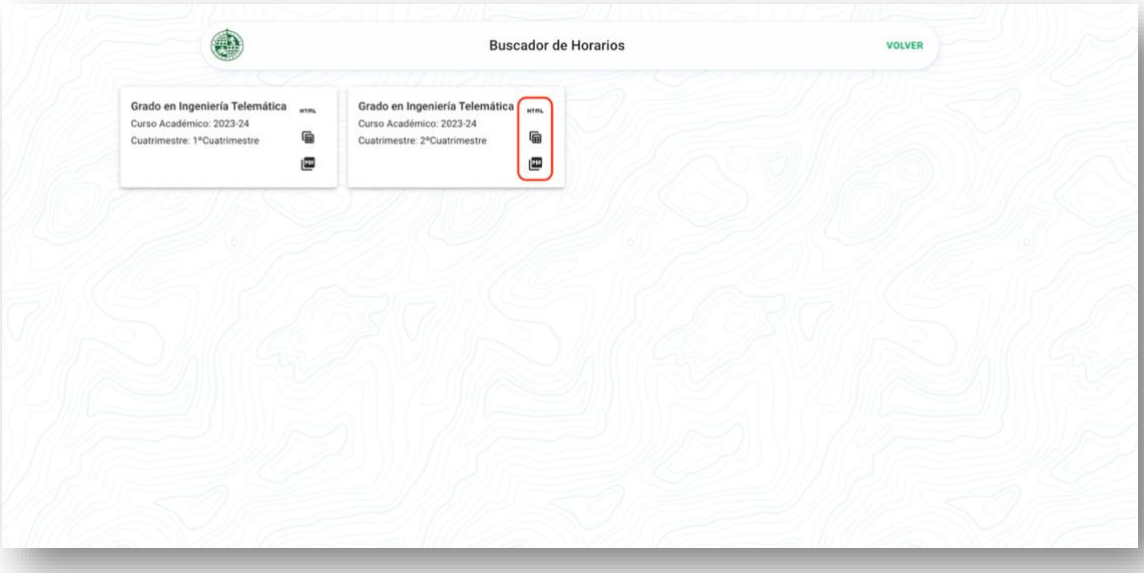
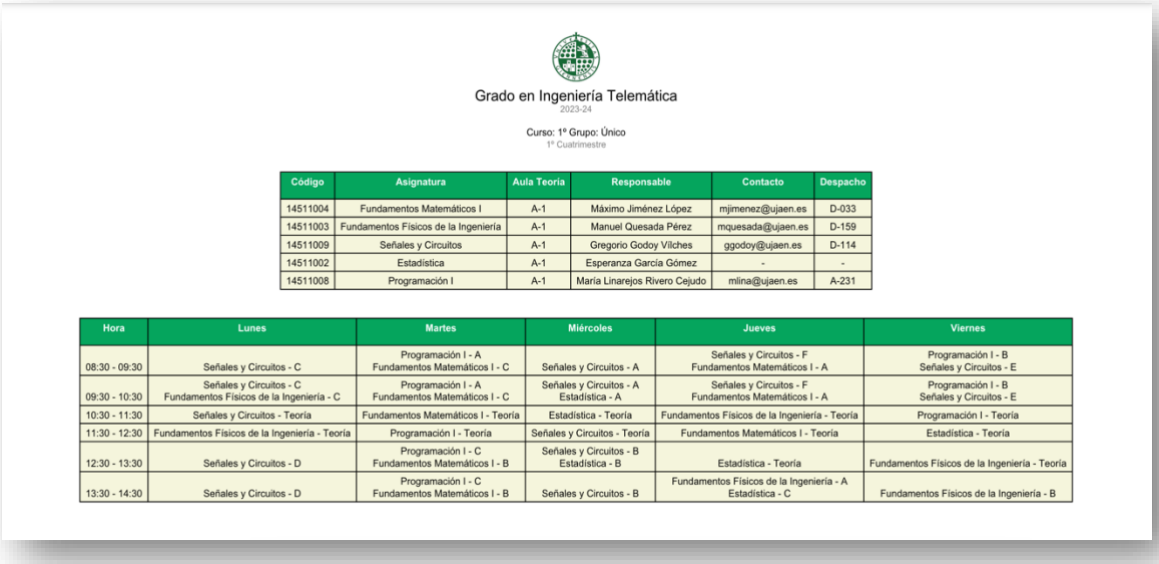


Ilustración 31: Página de consulta horaria

Para cada documento horario se podrá acceder a visualizar su contenido a través de un fichero *HTML*, una descarga *CSV* o una descarga *PDF*. A continuación, se muestra un ejemplo de visualización para cada tipo.



Grado en Ingeniería Telemática

2023-24

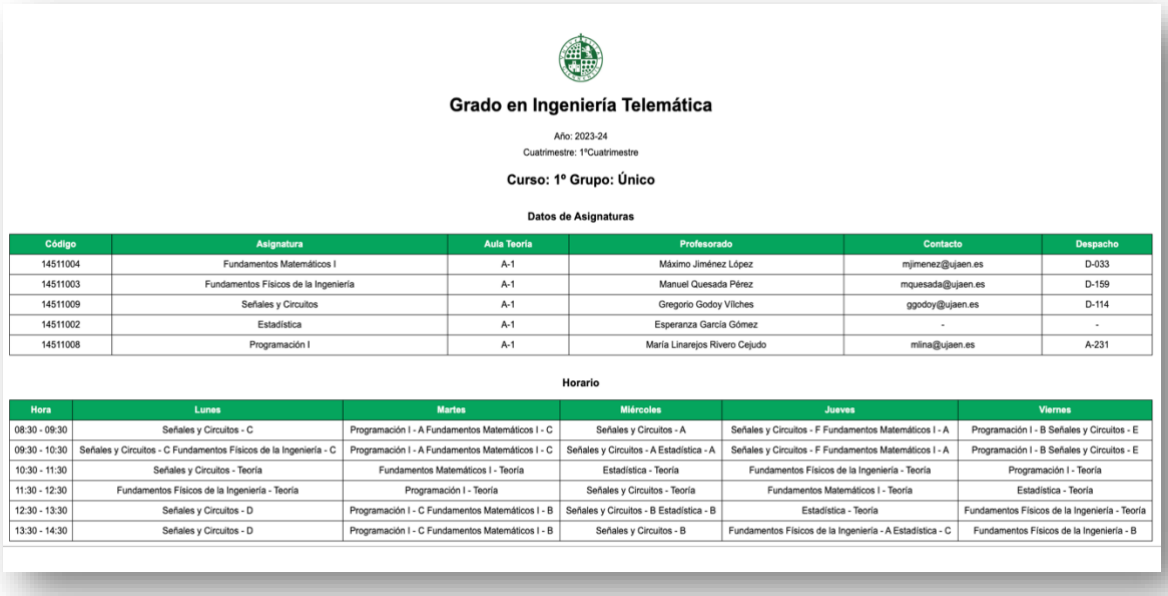
Curso: 1º Grupo: Único

1º Cuatrimestre

Código	Asignatura	Aula Teoría	Responsable	Contacto	Despacho
14511004	Fundamentos Matemáticos I	A-1	Máximo Jiménez López	mjimenez@ujaen.es	D-033
14511003	Fundamentos Físicos de la Ingeniería	A-1	Manuel Quesada Pérez	mquesada@ujaen.es	D-159
14511009	Señales y Circuitos	A-1	Gregorio Godoy Vilches	ggodoy@ujaen.es	D-114
14511002	Estadística	A-1	Esperanza García Gómez	-	-
14511008	Programación I	A-1	María Linarejos Rivero Cejudo	mlina@ujaen.es	A-231

Hora	Lunes	Martes	Miércoles	Jueves	Viernes
08:30 - 09:30	Señales y Circuitos - C	Programación I - A Fundamentos Matemáticos I - C	Señales y Circuitos - A	Señales y Circuitos - F Fundamentos Matemáticos I - A	Programación I - B Señales y Circuitos - E
09:30 - 10:30	Señales y Circuitos - C Fundamentos Físicos de la Ingeniería - C	Programación I - A Fundamentos Matemáticos I - C	Señales y Circuitos - A Estadística - A	Señales y Circuitos - F Fundamentos Matemáticos I - A	Programación I - B Señales y Circuitos - E
10:30 - 11:30	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría
11:30 - 12:30	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría
12:30 - 13:30	Señales y Circuitos - D	Programación I - C Fundamentos Matemáticos I - B	Señales y Circuitos - B Estadística - B	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría
13:30 - 14:30	Señales y Circuitos - D	Programación I - C Fundamentos Matemáticos I - B	Señales y Circuitos - B	Fundamentos Físicos de la Ingeniería - A Estadística - C	Fundamentos Físicos de la Ingeniería - B

Ilustración 32: Visualización PDF de un horario



Grado en Ingeniería Telemática

Año: 2023-24

Cuatrimestre: 1ºCuatrimestre

Curso: 1º Grupo: Único

Datos de Asignaturas

Código	Asignatura	Aula Teoría	Profesorado	Contacto	Despacho
14511004	Fundamentos Matemáticos I	A-1	Máximo Jiménez López	mjimenez@ujaen.es	D-033
14511003	Fundamentos Físicos de la Ingeniería	A-1	Manuel Quesada Pérez	mquesada@ujaen.es	D-159
14511009	Señales y Circuitos	A-1	Gregorio Godoy Vilches	ggodoy@ujaen.es	D-114
14511002	Estadística	A-1	Esperanza García Gómez	-	-
14511008	Programación I	A-1	María Linarejos Rivero Cejudo	mlina@ujaen.es	A-231

Horario

Hora	Lunes	Martes	Miércoles	Jueves	Viernes
08:30 - 09:30	Señales y Circuitos - C	Programación I - A Fundamentos Matemáticos I - C	Señales y Circuitos - A	Señales y Circuitos - F Fundamentos Matemáticos I - A	Programación I - B Señales y Circuitos - E
09:30 - 10:30	Señales y Circuitos - C Fundamentos Físicos de la Ingeniería - C	Programación I - A Fundamentos Matemáticos I - C	Señales y Circuitos - A Estadística - A	Señales y Circuitos - F Fundamentos Matemáticos I - A	Programación I - B Señales y Circuitos - E
10:30 - 11:30	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría
11:30 - 12:30	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría
12:30 - 13:30	Señales y Circuitos - D	Programación I - C Fundamentos Matemáticos I - B	Señales y Circuitos - B Estadística - B	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría
13:30 - 14:30	Señales y Circuitos - D	Programación I - C Fundamentos Matemáticos I - B	Señales y Circuitos - B	Fundamentos Físicos de la Ingeniería - A Estadística - C	Fundamentos Físicos de la Ingeniería - B

Ilustración 33: Visualización HTML de un horario

Hora	Lunes	Martes	Miércoles	Jueves	Viernes
08:30 - 09:30	Señales y Circuitos - C	Programación I - A, Fundamentos Matemáticos I - C	Señales y Circuitos - A	Señales y Circuitos - F, Fundamentos Matemáticos I - A	Programación I - B, Señales y Circuitos - E
09:30 - 10:30	Señales y Circuitos - C, Fundamentos Físicos de la Ingeniería - C	Programación I - A, Fundamentos Matemáticos I - C	Señales y Circuitos - A, Estadística - A	Señales y Circuitos - F, Fundamentos Matemáticos I - A	Programación I - B, Señales y Circuitos - E
10:30 - 11:30	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría
11:30 - 12:30	Fundamentos Físicos de la Ingeniería - Teoría	Programación I - Teoría	Señales y Circuitos - Teoría	Fundamentos Matemáticos I - Teoría	Estadística - Teoría
12:30 - 13:30	Señales y Circuitos - D	Programación I - C, Fundamentos Matemáticos I - B	Señales y Circuitos - B, Estadística - B	Estadística - Teoría	Fundamentos Físicos de la Ingeniería - Teoría
13:30 - 14:30	Señales y Circuitos - D	Programación I - C, Fundamentos Matemáticos I - B	Señales y Circuitos - B	Fundamentos Físicos de la Ingeniería - A, Estadística - C	Fundamentos Físicos de la Ingeniería - B

Ilustración 34: Visualización CSV de un horario

Ventana General del Sistema

Una vez se inicie sesión con las credenciales se accederá a la pantalla principal de la aplicación.

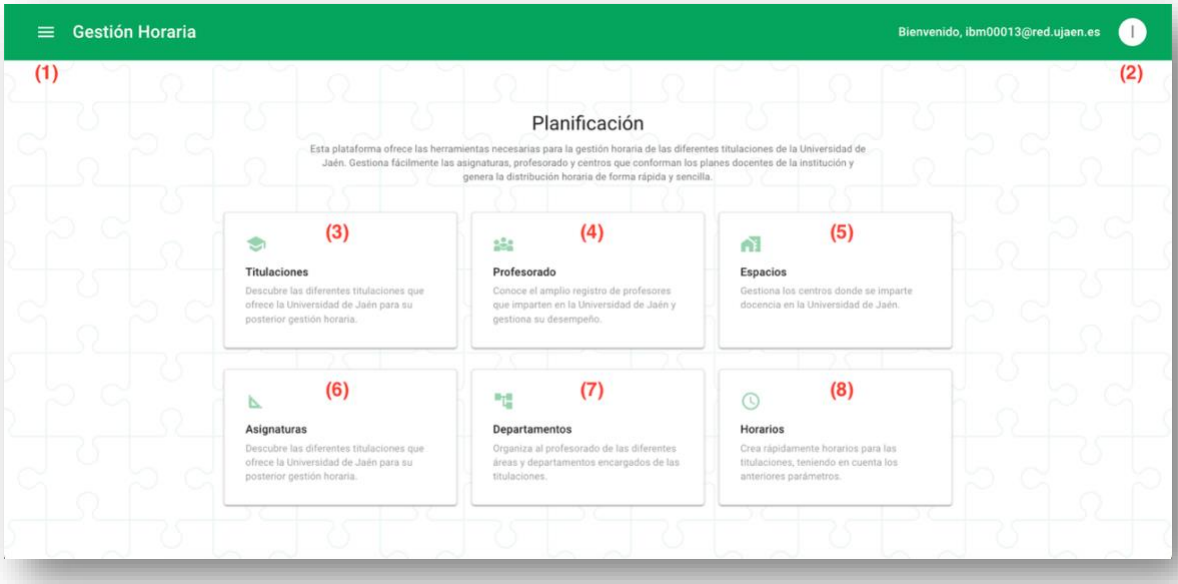


Ilustración 35: Vista principal de la plataforma

Esta pantalla cuenta con una barra de navegación principal y varios accesos rápidos a diferentes elementos de la plataforma.

1. En el extremo izquierdo de la barra de navegación existe un botón desplegable que abre y cierra el menú de navegación lateral.

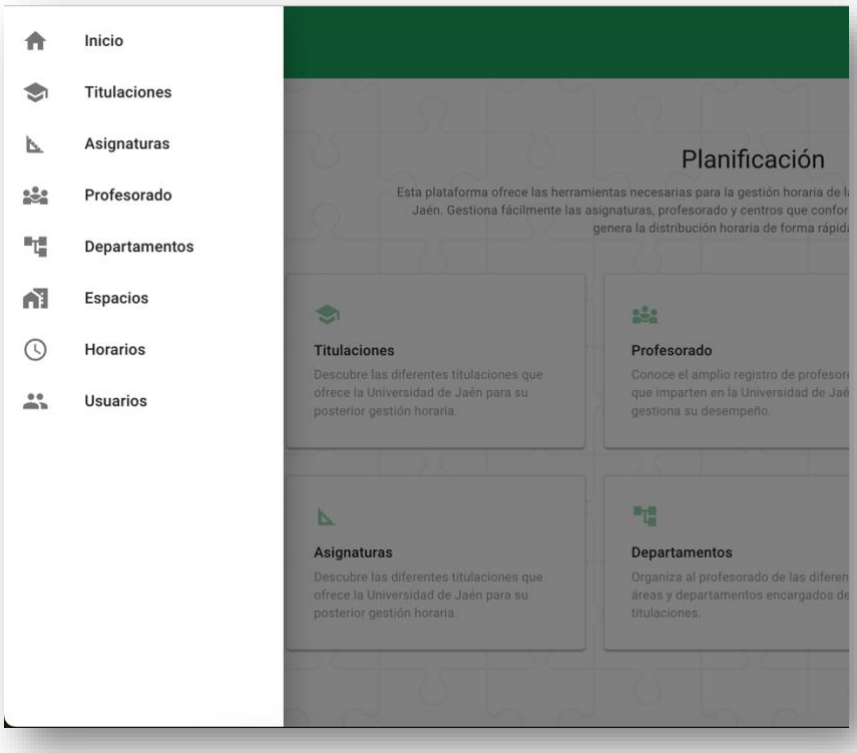


Ilustración 36: Menú desplegable lateral

2. En el extremo derecho de la barra de navegación existe un botón en forma de avatar que abre y cierra un desplegable con las opciones de visualización de perfil, un apartado de acerca de y cerrar sesión.

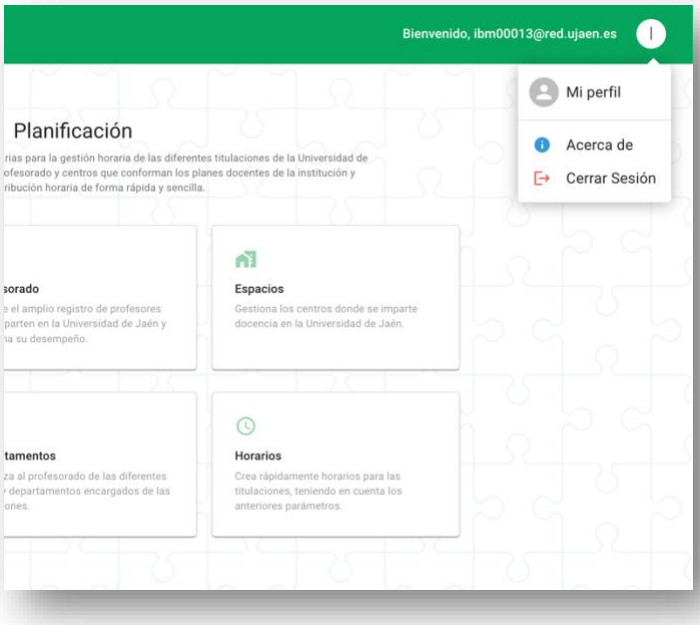


Ilustración 37: Desplegable de acceso al Perfil

3. Acceso rápido al apartado de Titulaciones de la plataforma.
4. Acceso rápido al apartado de Profesorado de la plataforma.
5. Acceso rápido al apartado de Espacios (Campus, Edificios, Aulas, etc.) de la plataforma.
6. Acceso rápido al apartado de Asignaturas de la plataforma.
7. Acceso rápido al apartado de Departamentos de la plataforma.
8. Acceso rápido al apartado de Horarios de la plataforma.

Gestión de Recursos

La gestión de recursos sólo estará disponible para *Administradores*. En esta vista se permite la visualización de todos los recursos de un mismo tipo que están almacenadas en el sistema a través de una tabla. La vista de gestión de *Titulaciones* será el ejemplo para seguir en este apartado, pero es común para *Asignaturas*, *Profesorado*, *Departamentos*, *Usuarios* y algunos *Espacios* (como *Aulas*, *Despachos* y *Centros*).

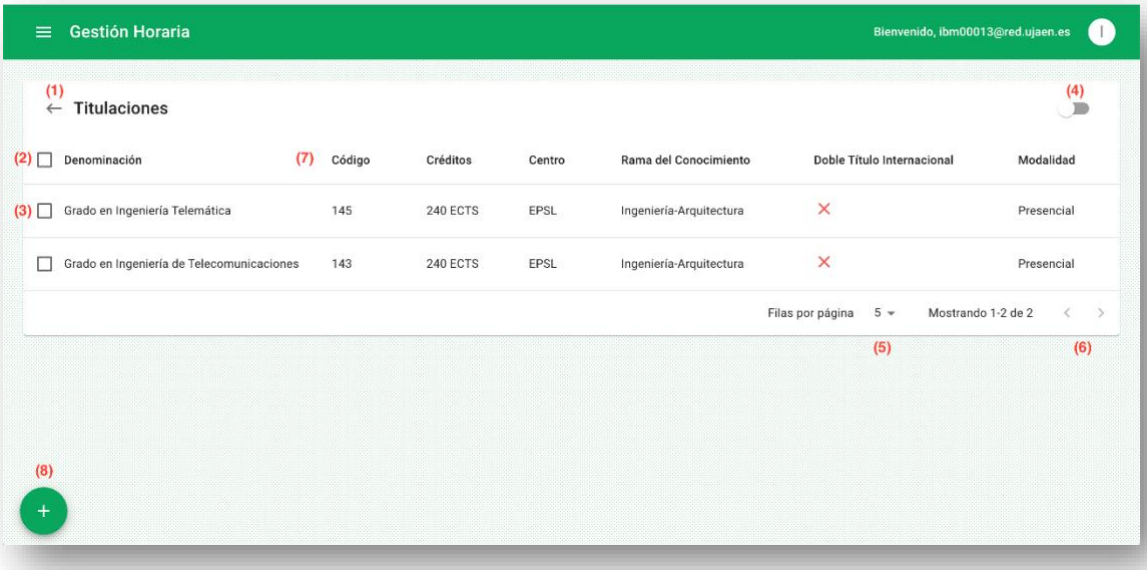


Ilustración 38: Vista de listado de Titulaciones

El funcionamiento de las tablas de gestión de recursos se ilustra en la imagen y se divide en los siguientes puntos.

1. Botón de retroceso, permite volver a la pantalla anterior. En este caso se devolvería a la pantalla principal de la plataforma.

2. Selector global, selecciona todos los elementos de la tabla, indicando el número de elementos totales seleccionados y la opción de borrado masivo.

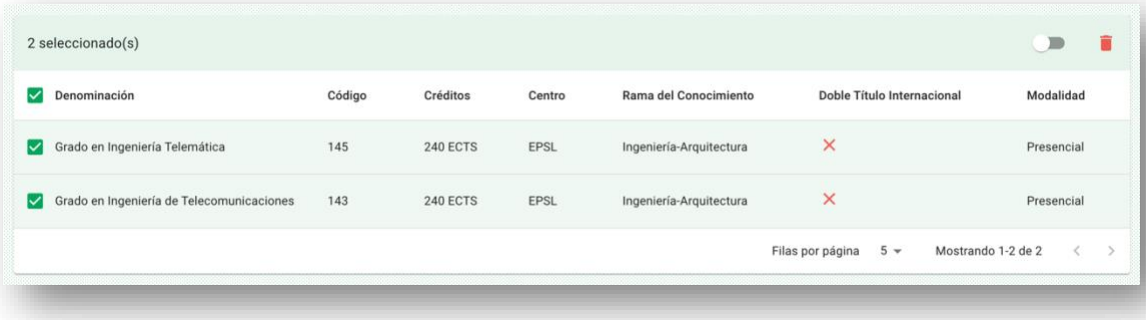


Ilustración 39: Selector global de recursos

3. Selector individual, permite seleccionar un único elemento de la tabla. Al seleccionar un recurso se habilitan diferentes opciones en la esquina superior derecha de la tabla. En ella se podrá obtener más información del elemento, editarlo o eliminarlo.

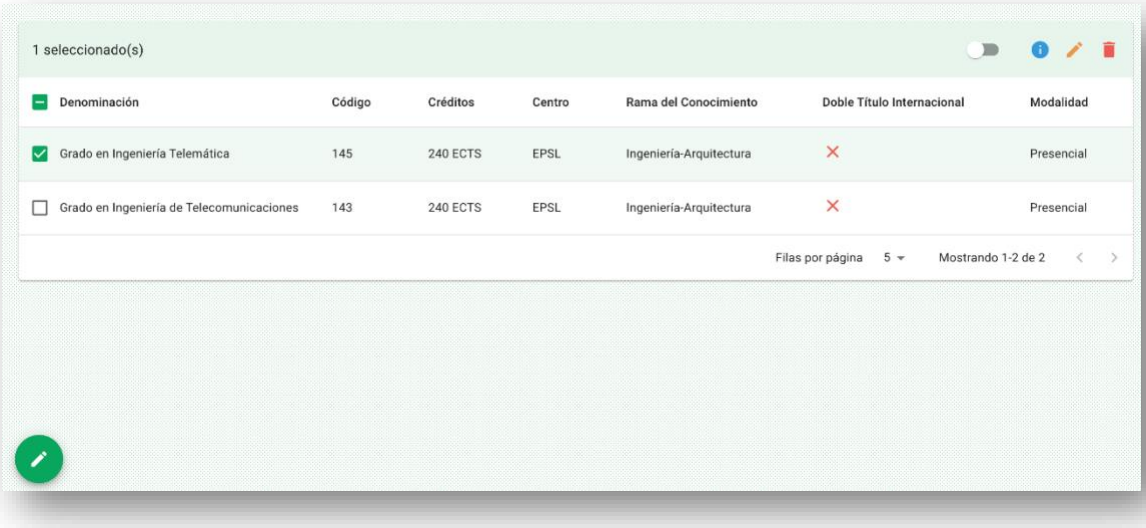


Ilustración 40: Selector individual de recursos

4. Interruptor que habilita o deshabilita la vista compacta de la tabla.

Ilustración 41: Vista de listado compacta

5. Selector para elegir el número máximo de recursos que se mostrarán por página. Puede tomar los valores de 5, 10 y 25. Por defecto su valor será 5.
6. Desplazamiento entre diferentes páginas de la paginación que incluye la tabla. En todo momento un texto indicará la numeración de los recursos que se están mostrando y el número total de ellos.
7. Filtros de ordenación por columnas. Cada columna cuenta con una opción para ordenar por orden alfabético (si fueran caracteres alfanuméricos) o de mayor a menor (en caso de caracteres numéricos) y viceversa.

Ilustración 42: Flechas para la ordenación de campos

8. Botón flotante que permite añadir nuevos recursos a la tabla. En caso de tener seleccionado un recurso con los selectores individuales, el botón flotante pasará a ser de edición para ese recurso seleccionado. Al añadir o editar un recurso se habilita un diálogo con una serie de campos a completar.

   Cada diálogo de creación o modificación tiene una guía superior con los pasos que hay que seguir para completar toda la información que se pide. Al llegar al último paso del diálogo se habilitará un botón para crear o modificar el recurso.

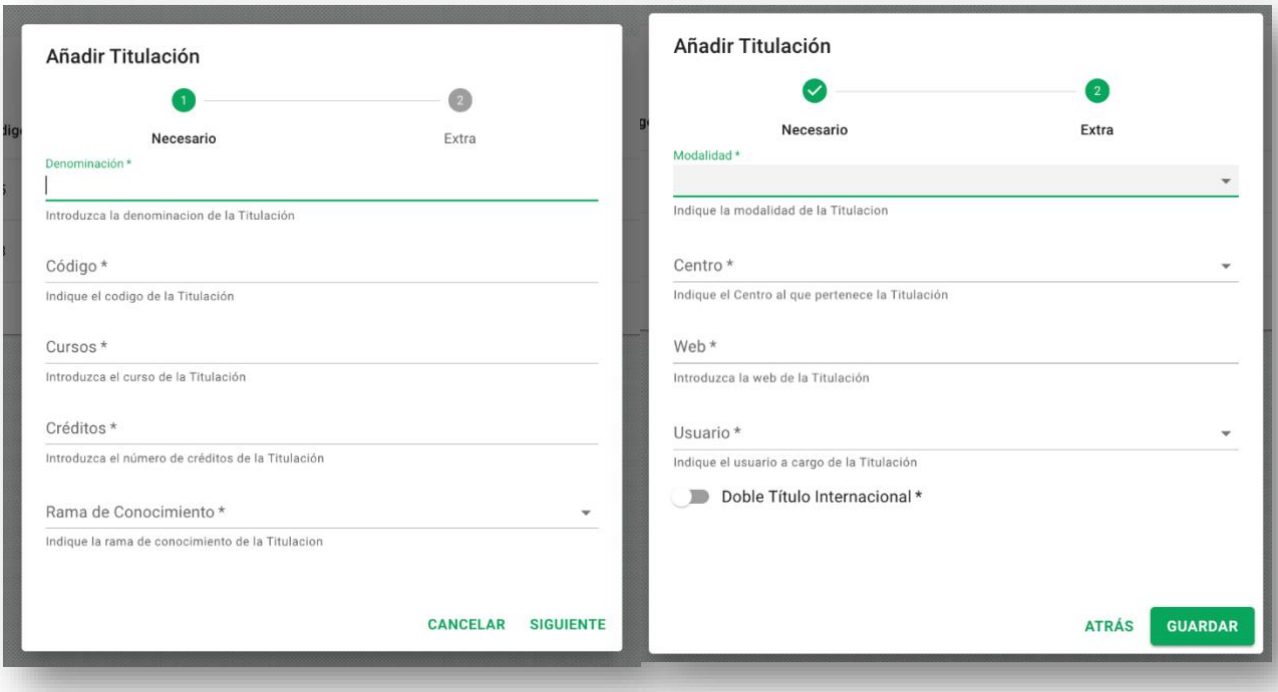


Ilustración 43: Diálogo para añadir Titulación

En el caso de la edición, el título del diálogo indicaría que se está modificando un recurso y los campos del diálogo aparecerían rellenos con la información del recurso que se ha seleccionado previamente.

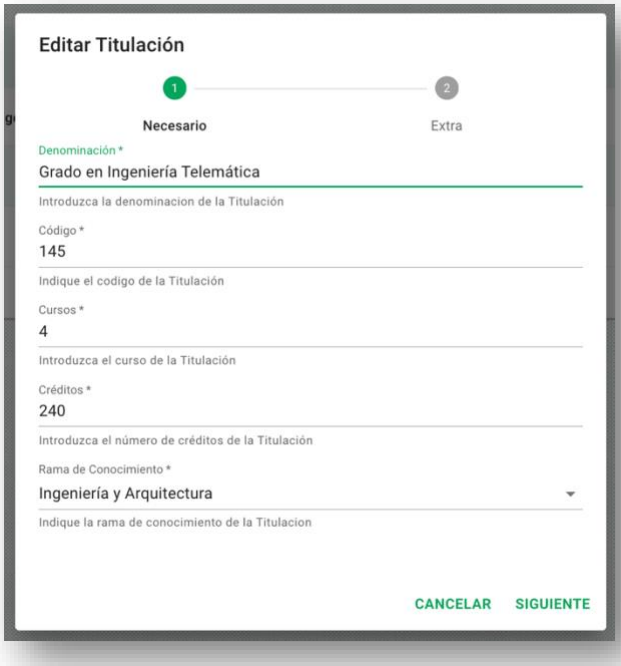


Ilustración 44: Diálogo para editar Titulación

Información sobre un Recurso

La información de recursos sólo estará disponible para *Administradores*. Dentro de la vista de gestión de recursos se podrá acceder a información de un recurso en concreto. Para ello se debe seleccionar individualmente un recurso y acceder al icono de *más información* en la esquina superior derecha de la tabla.

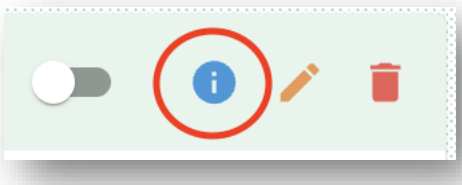


Ilustración 45: Opciones de recurso (Titulación)

Conducirá a una nueva pantalla en la que se muestra más en detalle información acerca de ese recurso. Además de funcionalidades extra propias de cada tipo de recurso.

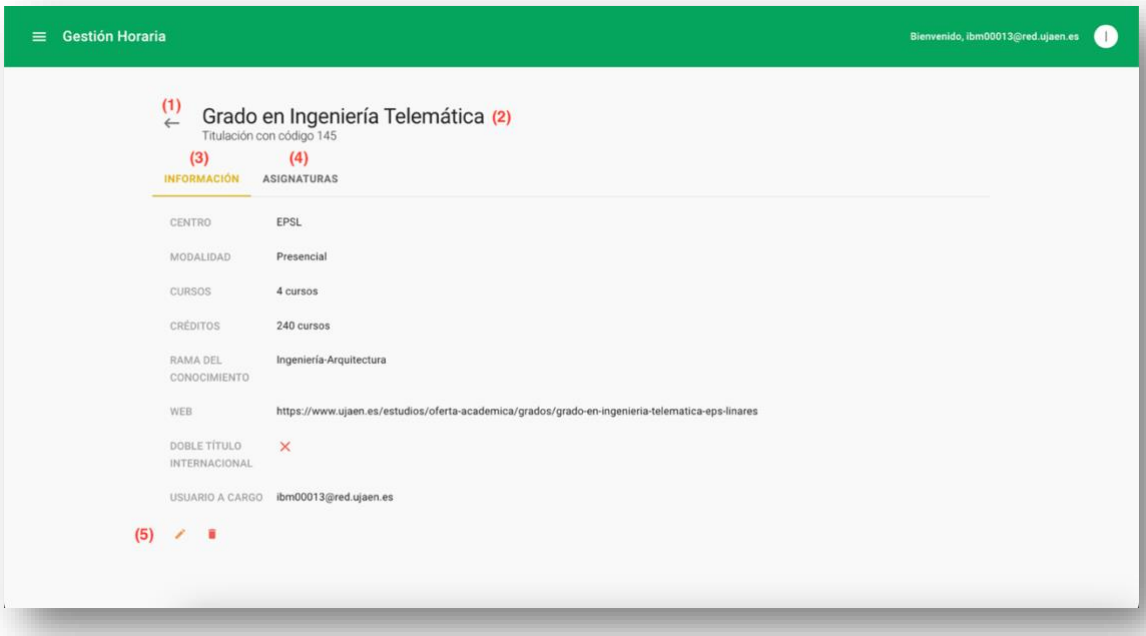


Ilustración 46: Vista de detalle de una Titulación

Se procede a desglosar las vistas de información sobre un recurso en diferentes puntos:

1. Botón de retroceso, devuelve a la vista de gestión de recursos donde se mostraba la tabla con todos los recursos ordenados.
2. Título del recurso, normalmente la denominación que identifica a la titulación, asignatura, centro, etc. Además, se incluye un subtítulo que aporta más información como puede ser el código de la titulación, el tipo de asignatura, los apellidos del profesor, etc.
3. Ventana de información, esta ventana es común para los diferentes tipos de recursos. En ella se muestran diferentes campos que definen un recurso y se disponen verticalmente para su consulta.
4. Ventanas extra, son opcionales dependiendo del tipo de recurso que se esté consultando. En el caso de una titulación, existe la ventana *asignaturas* que indica todas las asignaturas que incluye esta titulación, pero dependiendo del recurso pueden variar y aportar nuevas funcionalidades. Más adelante se definirán las principales ventanas extra para los diferentes recursos del sistema.
5. Controles de edición y borrado, permiten desplegar el diálogo de edición o borrado del recurso directamente desde esta vista.

Espacios

Los diferentes *Espacios* que se almacenan en la aplicación siguen un método de agrupación diferente. En la pantalla de *Espacios* se agrupan cinco recursos diferentes:

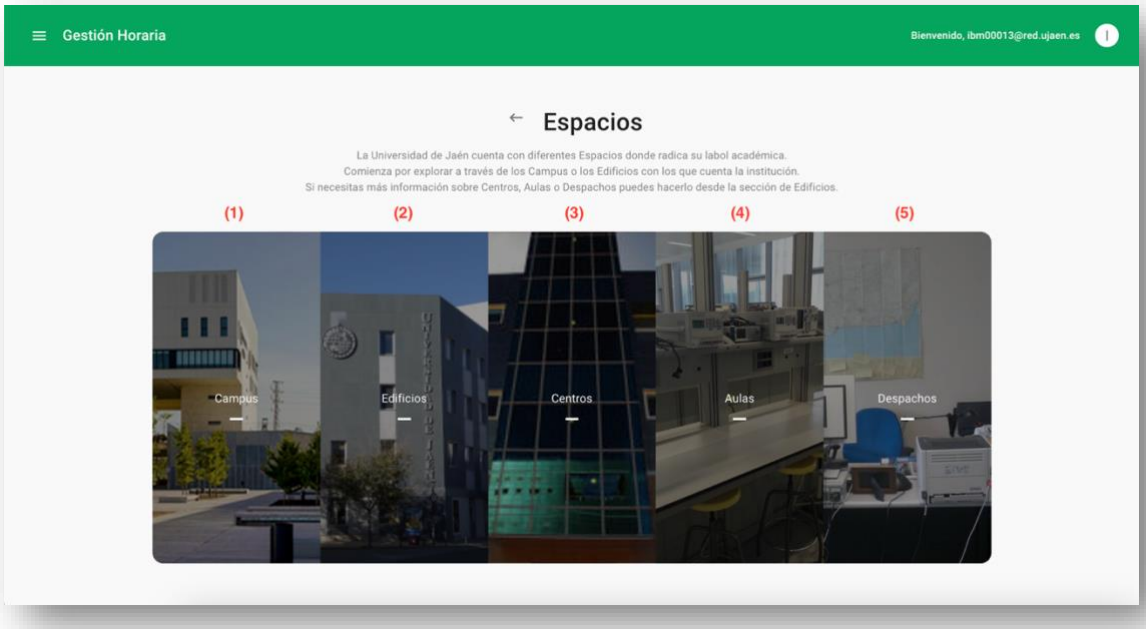


Ilustración 47: Vista de Espacios

1. El primer bloque habilita la gestión de *Campus*. Si se accede a dicho bloque se navega hacia el listado de *Campus*.
2. El segundo bloque habilita la gestión de *Edificios*. Si se accede a dicho bloque se navega hacia el listado de *Edificios*
3. El tercer bloque habilita la gestión de *Centros*. Si se accede a dicho bloque se navega hacia el listado de *Centros*.
4. El cuarto bloque habilita la gestión de *Aulas*. Si se accede a dicho bloque se navega hacia el listado de *Aulas*.
5. El quinto bloque habilita la gestión de *Despachos*. Si se accede a dicho bloque se navega hacia el listado de *Despachos*.

El bloque de *Campus* muestra de forma visual los diferentes *Campus* que conforman la Universidad de Jaén. Esta vista no implementa una tabla como el resto de los recursos.

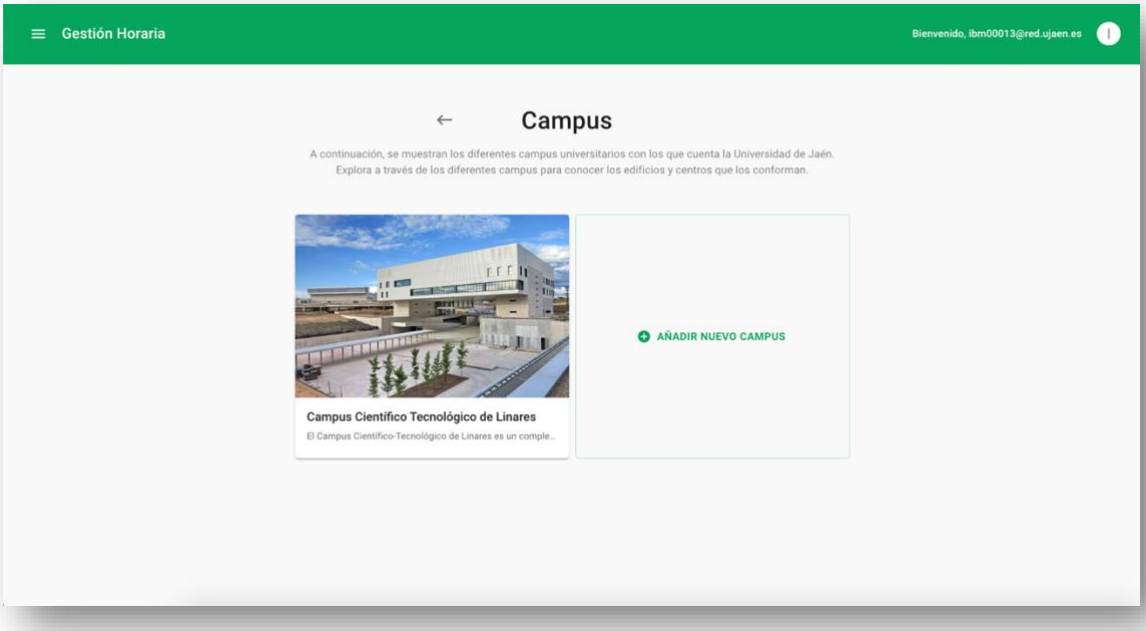


Ilustración 48: Vista de listado de Campus

El botón de *'añadir nuevo campus'* da la opción de crear un nuevo *Campus* rellenando todos los campos del diálogo que se mostrará. Entre los campos se encuentran la denominación, descripción, dirección, ciudad, código postal, coordenadas e imagen del *Campus*.

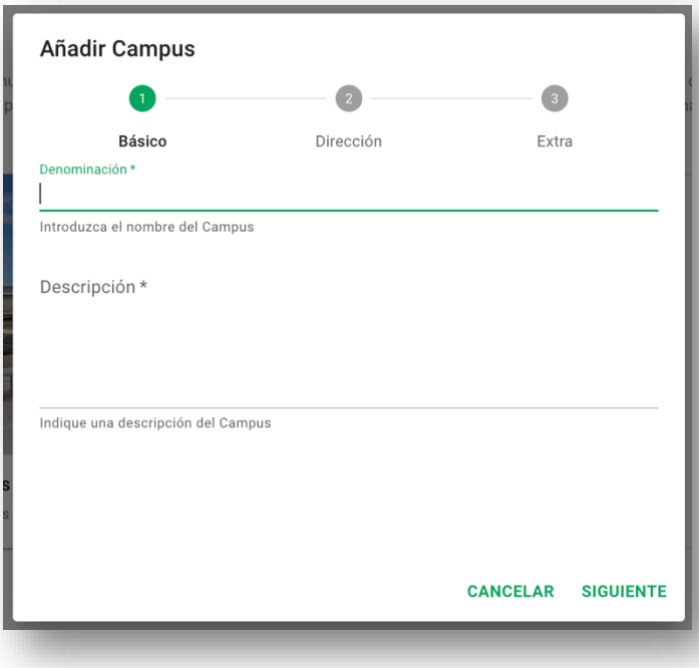


Ilustración 49: Diálogo para añadir Campus

Otra de las vistas que no sigue los patrones de gestión de recursos en tablas es la de *Edificios*. En esta vista se dispone visualmente los diferentes *Edificios* que componen la Universidad de Jaén.

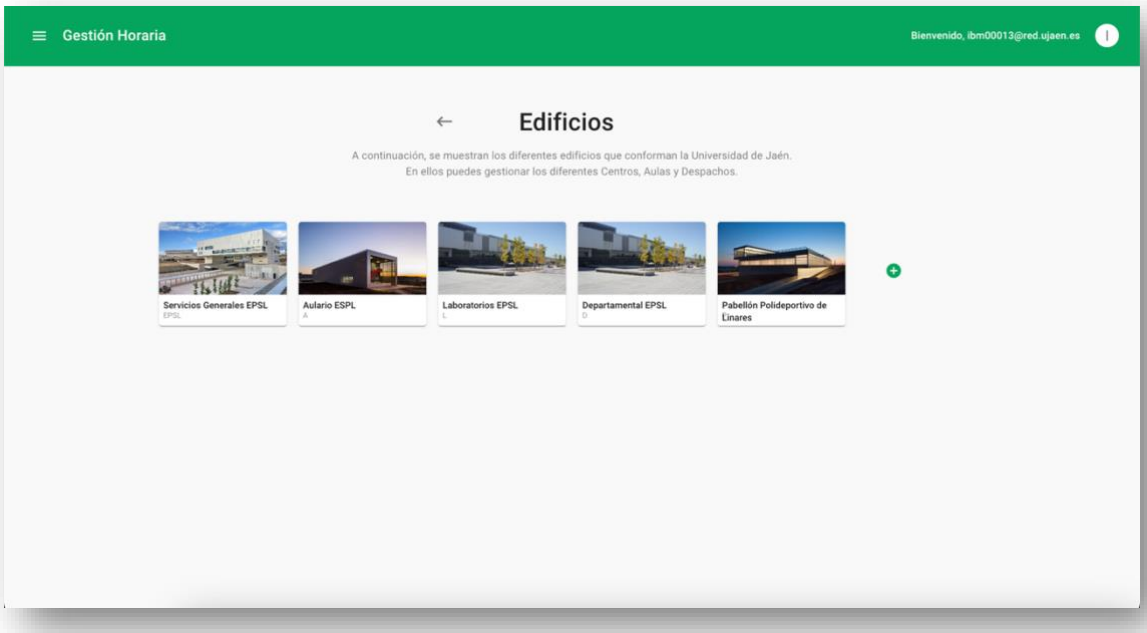


Ilustración 50: Vista de listado de Edificios

El botón de añadir da la opción de crear un nuevo *Edificio* rellenando todos los campos del diálogo que se mostrará. Entre los campos se encuentran la denominación, código, descripción, dirección, ciudad, código postal, coordenadas, imagen del *Edificio* y *Campus* al que pertenece (si perteneciera a alguno).

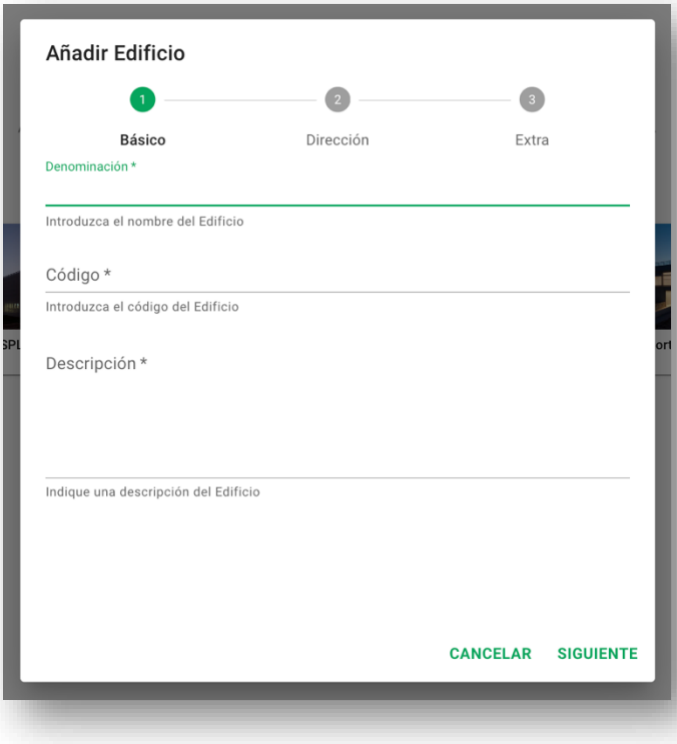


Ilustración 51: Diálogo de añadir Edificio

El resto de los recursos como *Centros*, *Aulas* y *Despachos* siguen el mismo enfoque de gestión que el resto de los recursos.

Ventanas Específicas

Dependiendo del tipo de recurso al que se acceda para ver más información existen ventanas personalizadas de este tipo de recurso. Por ejemplo, dentro de una *Titulación* existen las *Asignaturas* que incluye en su plan docente, pero dentro de un *Edificio* se encuentran las *Aulas* y *Despachos* que incluye en su interior.

Se procede a repasar las principales ventanas extra de cada recurso:

- **Titulación**

Dentro de una titulación existe la ventana *Asignaturas*, esta ventana muestra todas las asignaturas que conforman el plan docente de la *Titulación*.

Cada *Asignatura* tendrá un código identificativo único de su asignación con esta *Titulación*. Esto es debido a que es posible que una *Asignatura* puede pertenecer a varias *Titulaciones*.

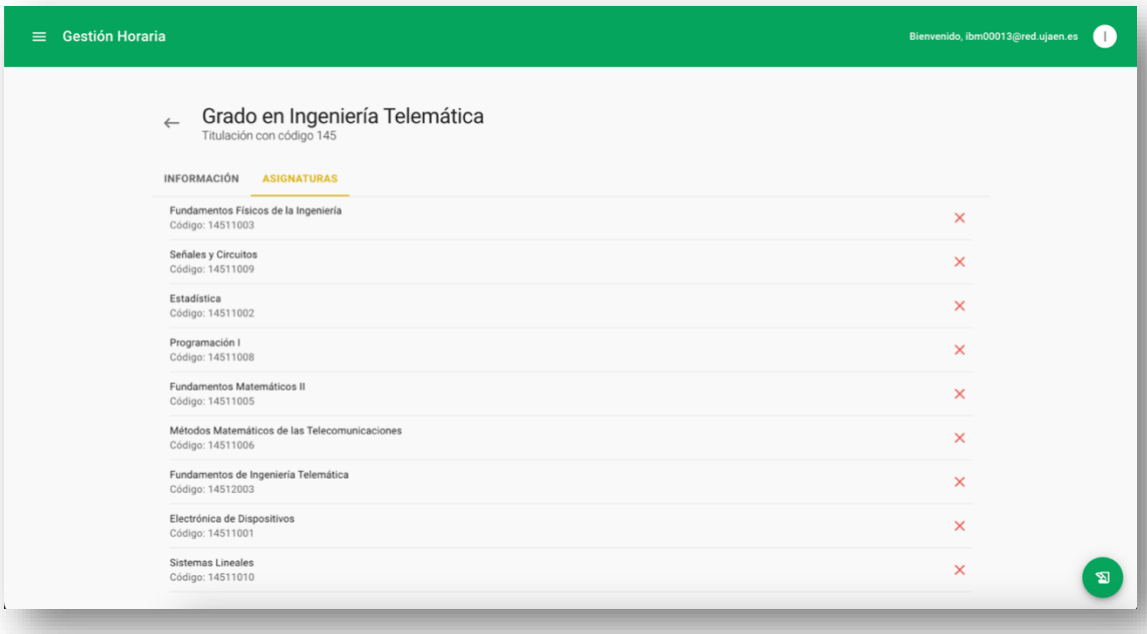


Ilustración 52: Asignaturas asignadas a una Titulación

Se dispone de un botón en forma de equis para quitar una *Asignatura* de una *Titulación*. Aparecerá un diálogo de confirmación para que permitir la operación.

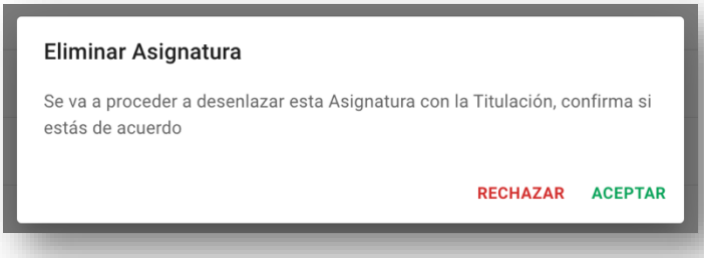


Ilustración 53: Diálogo para desenlazar Asignatura de una Titulación

En la esquina inferior derecha se dispone de un botón flotante que sirve para la asignación de nuevas *Asignaturas* a la *Titulación*. Si se pulsa aparecerá el siguiente diálogo donde se podrá realizar una nueva asignación.

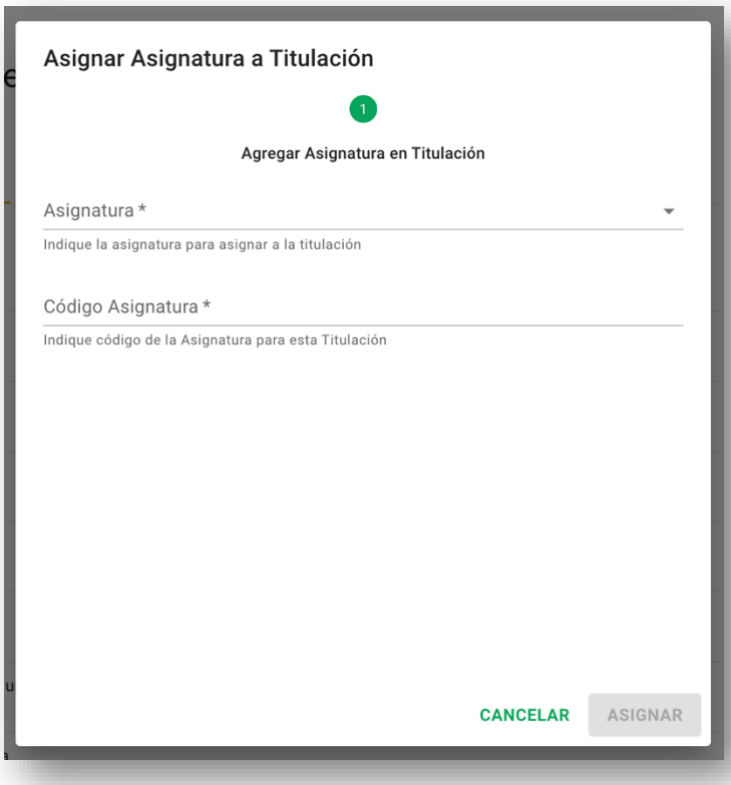


Ilustración 54: Diálogo para asignar una Asignatura a una Titulación

- **Asignatura**

Dentro de la información del recurso *Asignatura* existen dos ventanas extra, la primera es *Titulaciones* y la segunda *Grupos*.

Ilustración 55: Pestañas específicas del detalle de una Asignatura

Dentro de *Titulaciones*, el funcionamiento es similar, para cada *Asignatura* se listan las diferentes *Titulaciones* a las que pertenece, incluyendo su código identificativo para cada una de ellas.

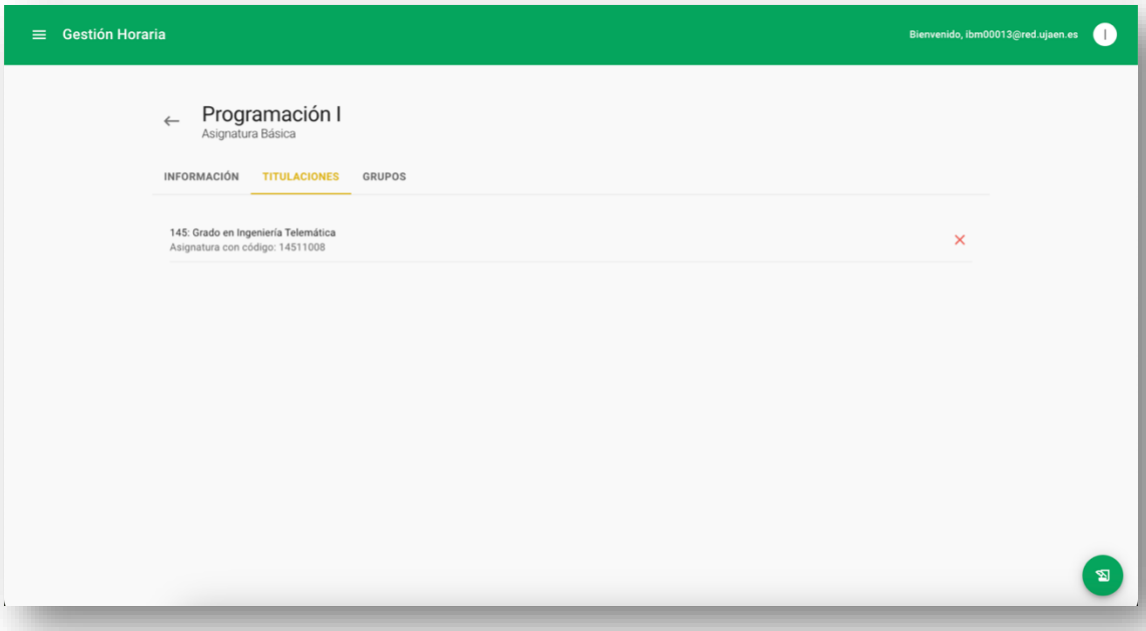


Ilustración 56: Vista de Titulaciones a las que pertenece una Asignatura

El funcionamiento es análogo a la eliminación de *Asignaturas* en la vista de *Titulaciones*, se puede eliminar la pertenencia a una *Titulación* pulsando sobre el botón en forma de equis. Aparecerá el diálogo de confirmación para que se permita la operación.

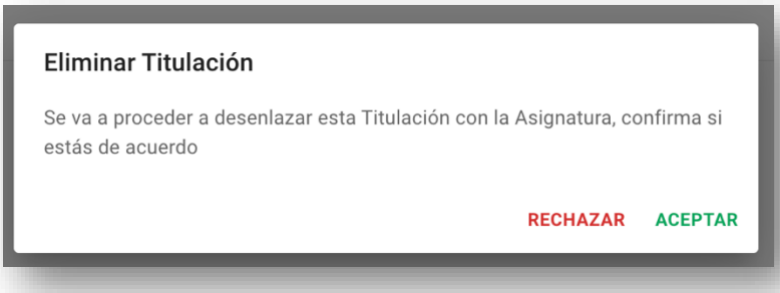


Ilustración 57: Diálogo para desenlazar una Titulación de una Asignatura

El botón flotante de la esquina inferior derecha habilita la asignación de nuevas *Titulaciones* para la *Asignatura* que se está mostrando. Se habilita un diálogo que se debe completar con la *Titulación* y el código que tendrá la *Asignatura* una vez asignada.

Ilustración 58: Diálogo de asignación Titulación-Asignatura

Para el caso de la ventana *Grupos* se pueden ver todos los grupos que conforman la *Asignatura.*

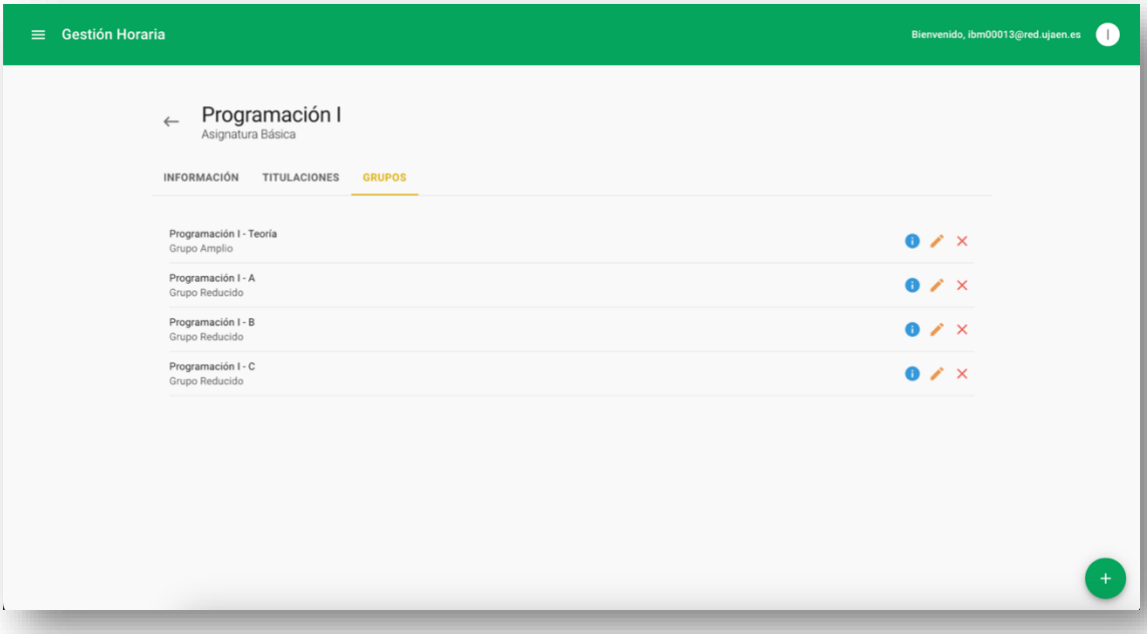


Ilustración 59: Grupos que conforman una Asignatura

Para cada *Grupo* se puede ver su tipo (Amplio o Reducido), el nombre de la *Asignatura* y la denominación del *Grupo*. Además, se habilitan tres acciones disponibles para cada *Grupo*:

Ilustración 60: Opciones de recurso (Grupo)

- Obtener información del *Grupo* en cuestión. Se desplegará un diálogo con las características del *Grupo*, así como la frecuencia y el tipo.

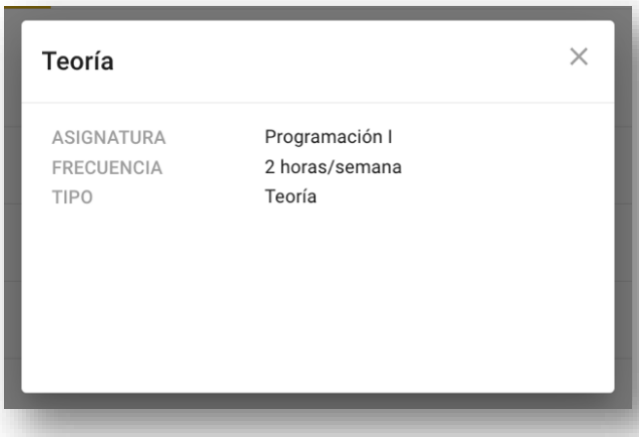


Ilustración 61: Información detallada de un Grupo

- Editar el *Grupo*. Se desplegará un diálogo con diferentes campos para modificar, así como el *Aula* donde se imparte la docencia de ese *Grupo*.

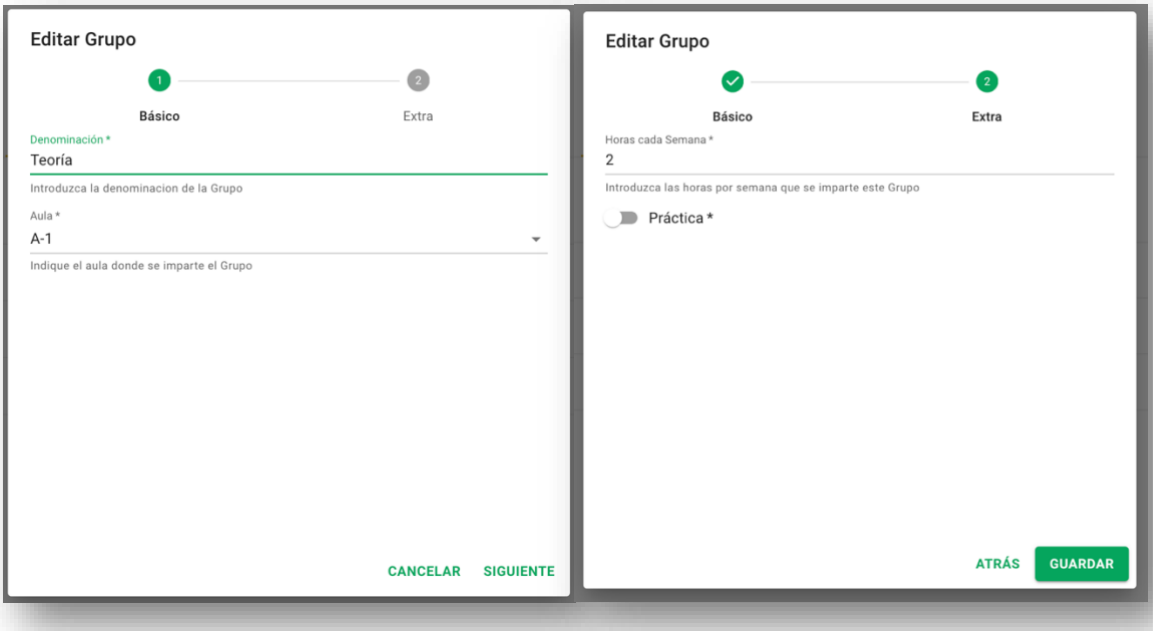


Ilustración 62: Diálogo de edición de Grupo

Cada *Grupo* tiene unas *horas por semana* que impartir, las cuales se verán reflejadas en los correspondientes *Horarios*. Además de un indicador para marcar el *Grupo* como Amplio o Reducido.

- Eliminar el *Grupo.* Se desplegará un diálogo de confirmación para asegurar la ejecución de la operación.

Ilustración 63: Diálogo de eliminación de Grupo

El botón flotante de la parte inferior derecha permite añadir un nuevo *Grupo*. Se desplegará un diálogo semejante al de edición, pero con los campos vacíos, que deberán rellenarse correctamente para añadir un nuevo *Grupo.* Deberán especificarse denominación, aula, horas por semana y tipo de grupo.

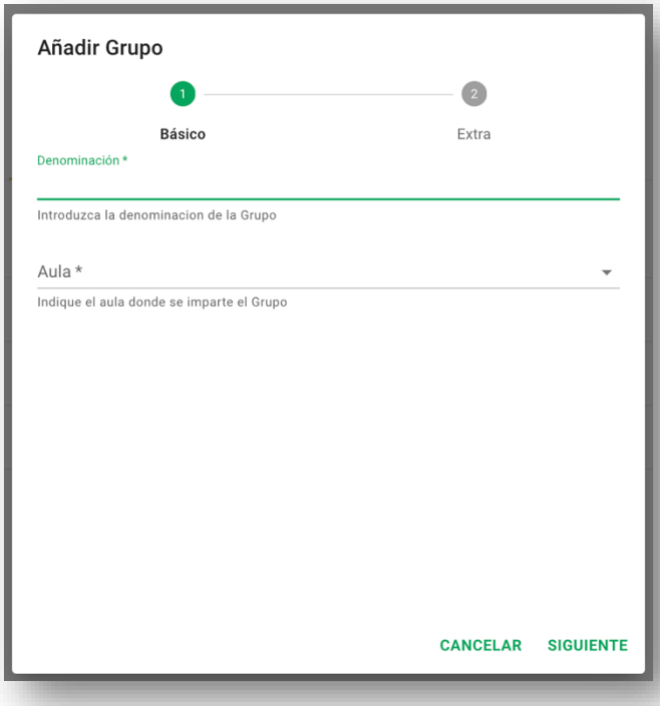


Ilustración 64: Diálogo para añadir Grupo

- **Profesorado**

Dentro de la vista en detalle de un *Profesor* existe la ventana *Asignaturas*. Esta ventana indica los *Grupos* a los que imparte docencia un *Profesor*. En cada fila de esta ventana se indica la denominación del *Grupo* y la *Asignatura* a la que pertenece dicho *Grupo.*

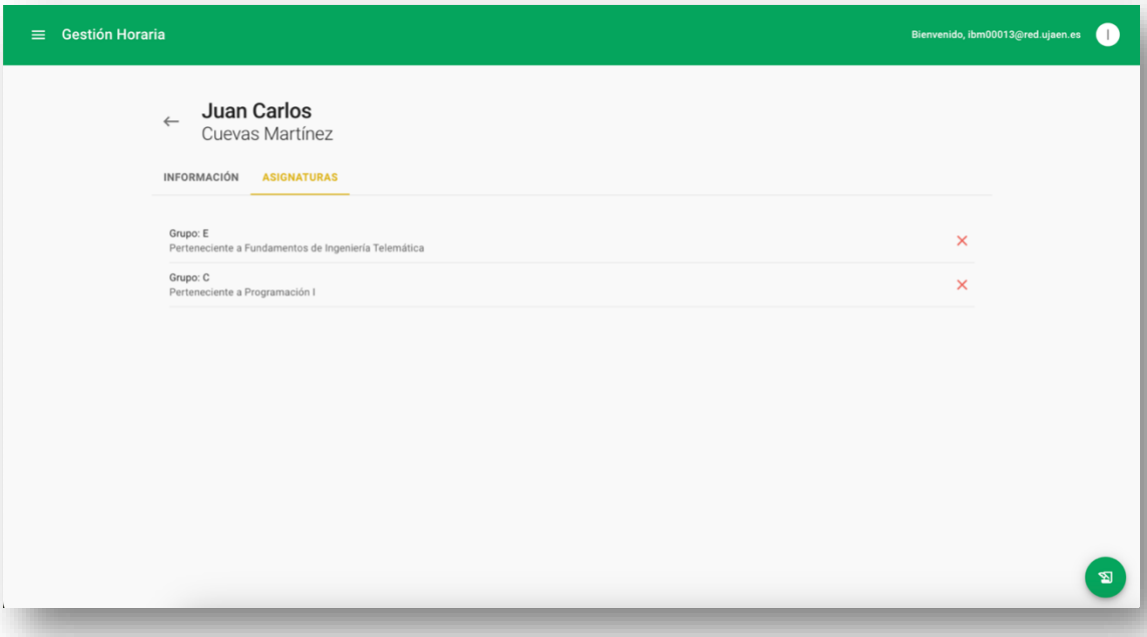


Ilustración 65: Vista en detalle de un Profesor

Para eliminar una asignación de un *Profesor* con un *Grupo* basta con pinchar sobre el icono con forma de equis. Se habilitará un diálogo de confirmación para que se permita la operación.

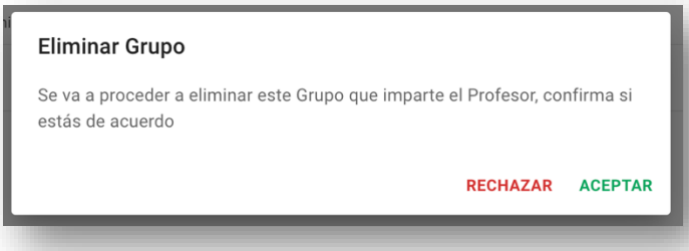


Ilustración 66: Diálogo para eliminar Grupo impartido por Profesor

Si se accede al botón flotante de la esquina inferior derecha, se podrá añadir un nuevo *Grupo* para que el *Profesor* imparta docencia. Esto se hará a través del diálogo de selección que aparecerá.

Ilustración 67: Diálogo de asignación de un Grupo impartido por un Profesor

- **Departamento**

Dentro de la vista de información sobre un departamento, existe la ventana *Áreas*. En esta ventana se listan las diferentes *Áreas* que componen un *Departamento*.

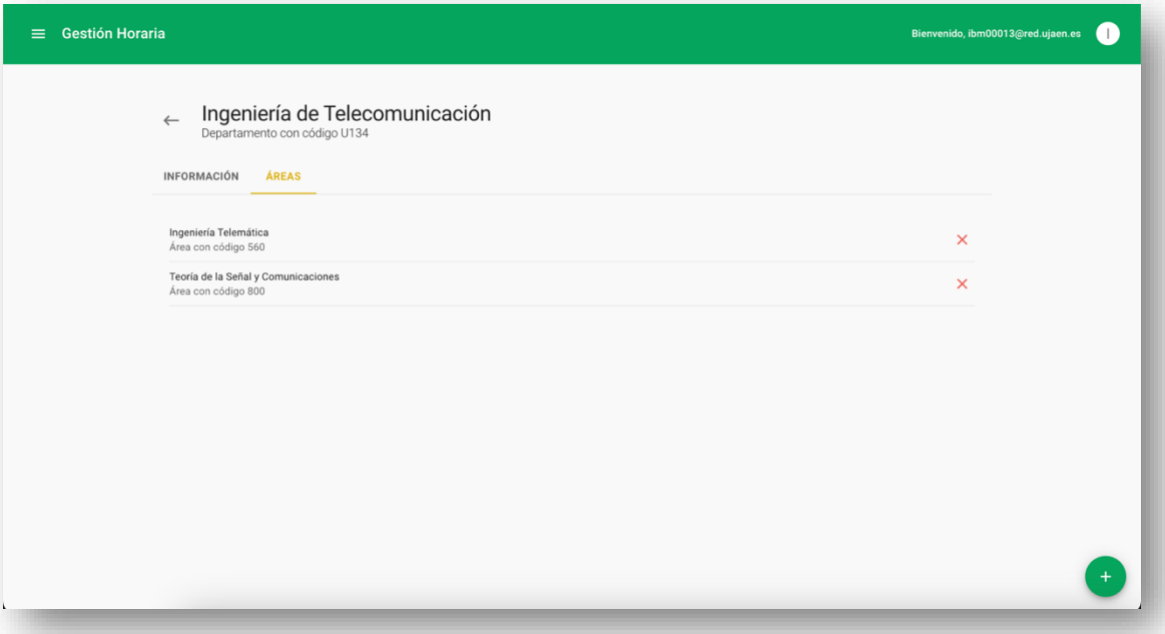


Ilustración 68: Vista de Áreas que conforman un Departamento

Cada *Área* aparece con su denominación y código identificativo. Se puede eliminar un *Área* en el botón con forma de equis. Se desplegará un diálogo de confirmación para permitir la acción.

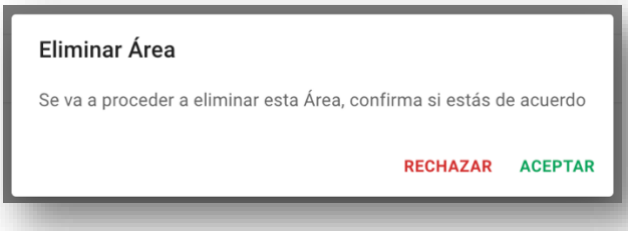


Ilustración 69: Diálogo de eliminación de Área

En el botón flotante de la equina inferior derecha se habilita un diálogo que permite agregar una nueva *Área* al *Departamento*. Deberá indicarse una denominación y un código para la agregación.

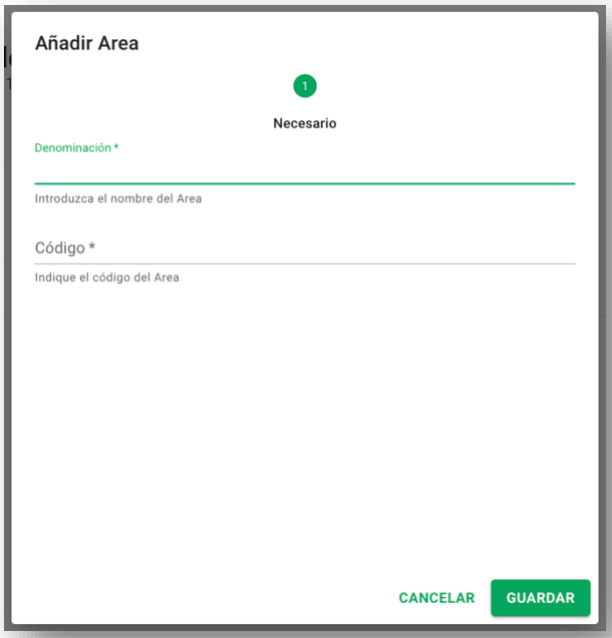


Ilustración 70: Diálogo para añadir Área

- **Campus**

La vista informativa de un Campus contiene dos ventanas específicas sobre *Edificios* y Localización.

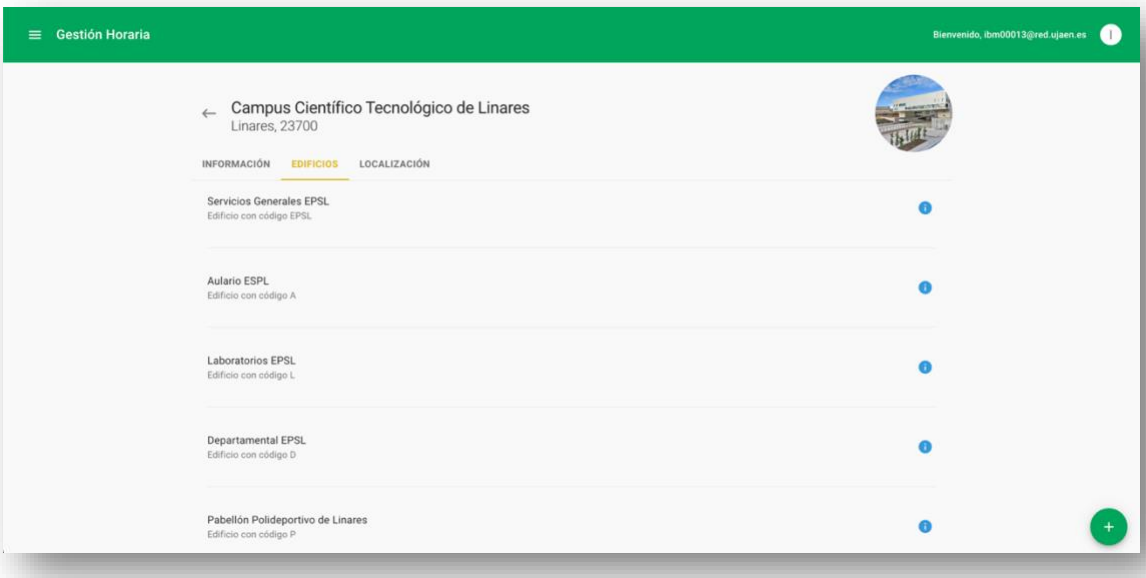


Ilustración 71: Vista de Edificios que incluye un Campus

En la ventana de *Edificios* se pueden ver todos los *Edificios* que forman parte de un *Campus*. Si se desea navegar hacia la información de algún *Edificio* que aparezca en esa lista, basta con pulsar el botón de *'más información'* en el extremo de cada listado. También se podrá añadir algún *Edificio* automáticamente desde la vista informativa de un *Campus* con el botón flotante de la esquina inferior derecha. Se desplegará un diálogo que se debe rellenar para añadir un nuevo *Edificio* al *Campus*.

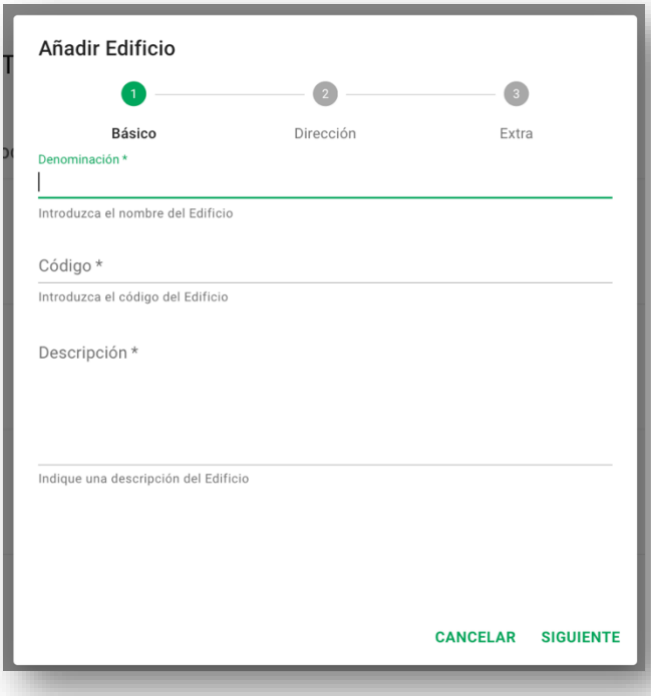


Ilustración 72: Diálogo para añadir Edificio

La particularidad de este diálogo, frente al de *'Añadir Edificio'* que ya se observó, es que no es necesario introducir el *Campus*, pues se coge por defecto el *Campus* de la vista en la que se está visualizando el listado.

En la ventana Localización se muestra la ubicación del *Espacio* en un pequeño mapa dentro de la ventana.

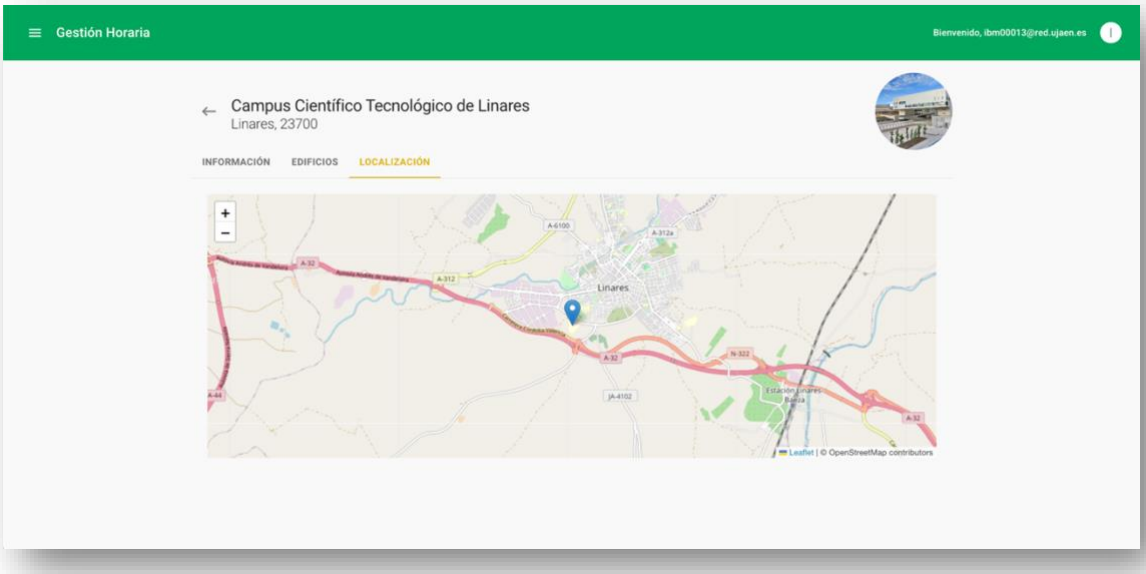


Ilustración 73: Vista de localización de un Espacio

- **Edificio**

Dentro de la vista sobre la información de un *Edificio* existen tres nuevas ventanas referentes a *Aulas*, *Despachos* y *Centros* que pueden estar ubicados o asociados a un *Edificio*.

En primer lugar, existe la ventana de *Aulas*, en ella se muestran todas las *Aulas* que conforman un *Edificio*.

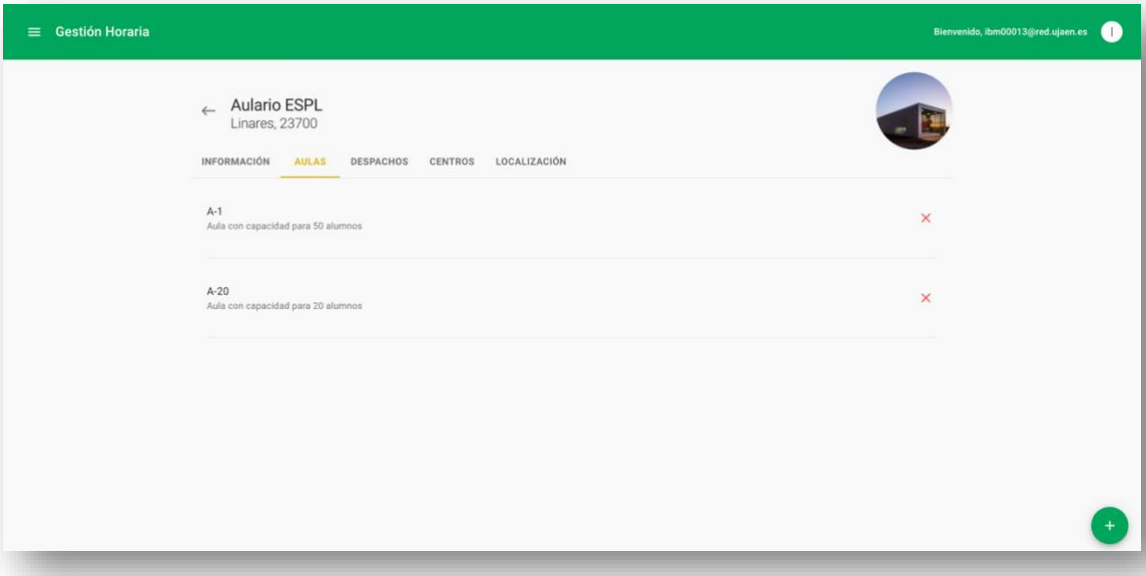


Ilustración 74: Vista de Aulas presentes en un Edificio

En el listado se puede ver la denominación del *Aula* y su capacidad. Para eliminar un *Aula* basta con pulsar sobre el botón con forma de equis del extremo del listado. Se desplegará un diálogo de confirmación para permitir la operación.

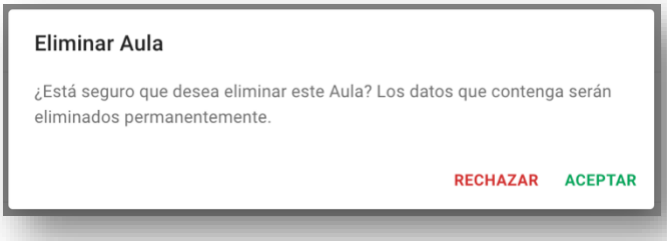


Ilustración 75: Diálogo para eliminar Aula de un Edificio

Se puede agregar automáticamente *Aulas* a este *Edificio* a través del botón flotante de la esquina inferior derecha. Se desplegará un diálogo en el que se debe completar todos los campos.

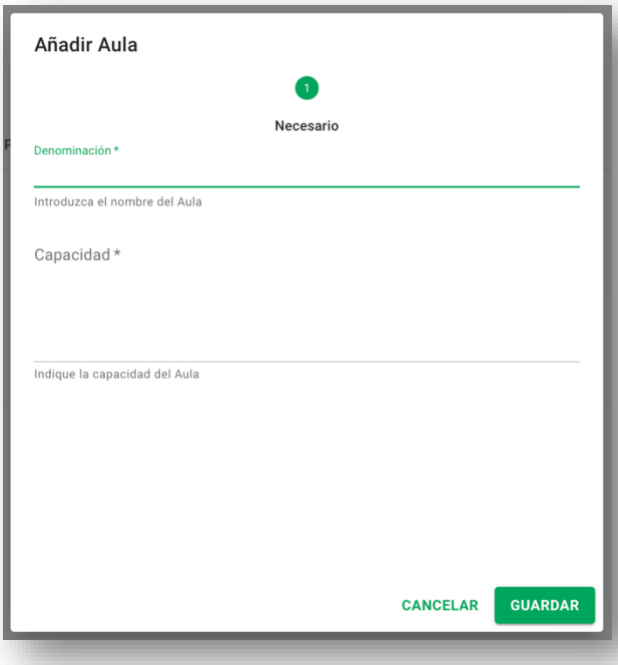


Ilustración 76: Diálogo para añadir Aula a un Edificio

Al completarlo, se agregará automáticamente una nueva *Aula* en el *Edificio* que se encontraba anteriormente.

Gestión Horaria

Una vez completos todos los recursos necesarios para la gestión horaria, se puede proceder con la creación y modificación de documentos horarios. Para ello basta con acceder al apartado *Horarios* desde el menú desplegable o mediante el acceso rápido de la página principal.

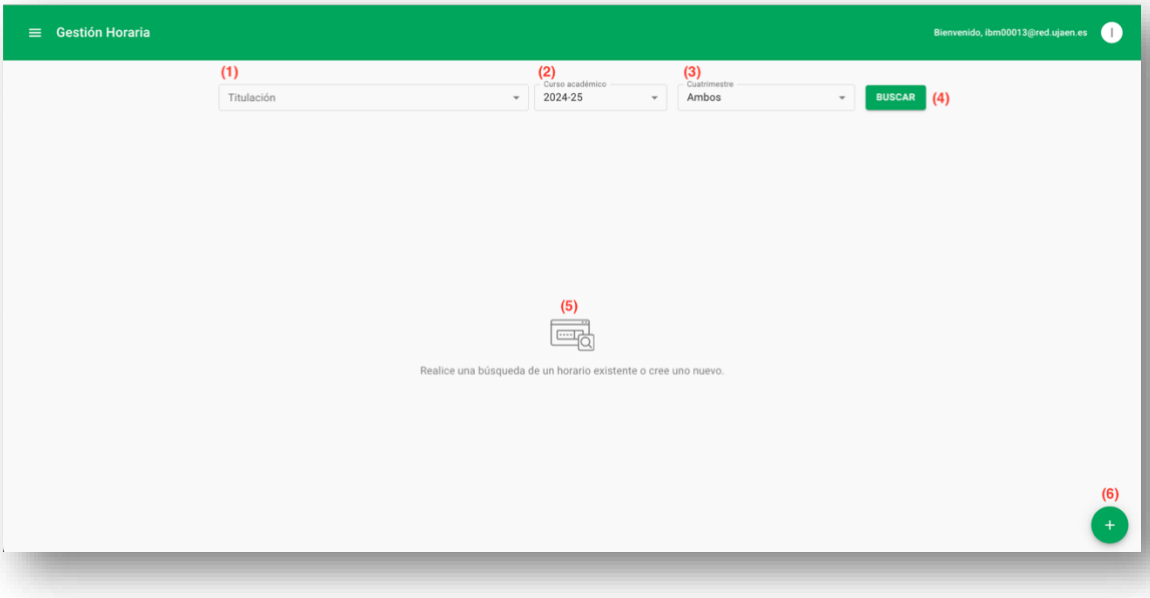


Ilustración 77: Vista de búsqueda de documentos horarios

Una vez se acceda a la pantalla *Horarios* se muestra una vista como la que se refleja en la anterior imagen. Existen una serie de elementos clave en la interfaz:

1. Selector de *Titulación*. Sirve para buscar la *Titulación* la cual se quiera consultar sus documentos horarios. Si no se especifica ninguna *Titulación* por defecto se realiza una búsqueda en todas las *Titulaciones* del sistema.
2. Selector de curso académico. Especifica el curso académico del cual se desean obtener los documentos horarios, este selector es obligatorio y por defecto selecciona el curso académico actual.
3. Selector de cuatrimestre. Se puede elegir filtrar por documentos horarios del primer cuatrimestre, del segundo cuatrimestre, como de ambos.
4. Botón de búsqueda. Una vez completados los tres selectores se hace un filtrado de documentos horarios que cumplan con los requisitos especificados.
5. En caso de filtrar satisfactoriamente por documentos horarios, estos aparecerán en esta franja de la interfaz para que se seleccionen. De no

encontrar ningún documento horario de acuerdo con los parámetros dados se mostrará un mensaje de error que indicará que la búsqueda no ha arrojado resultados.

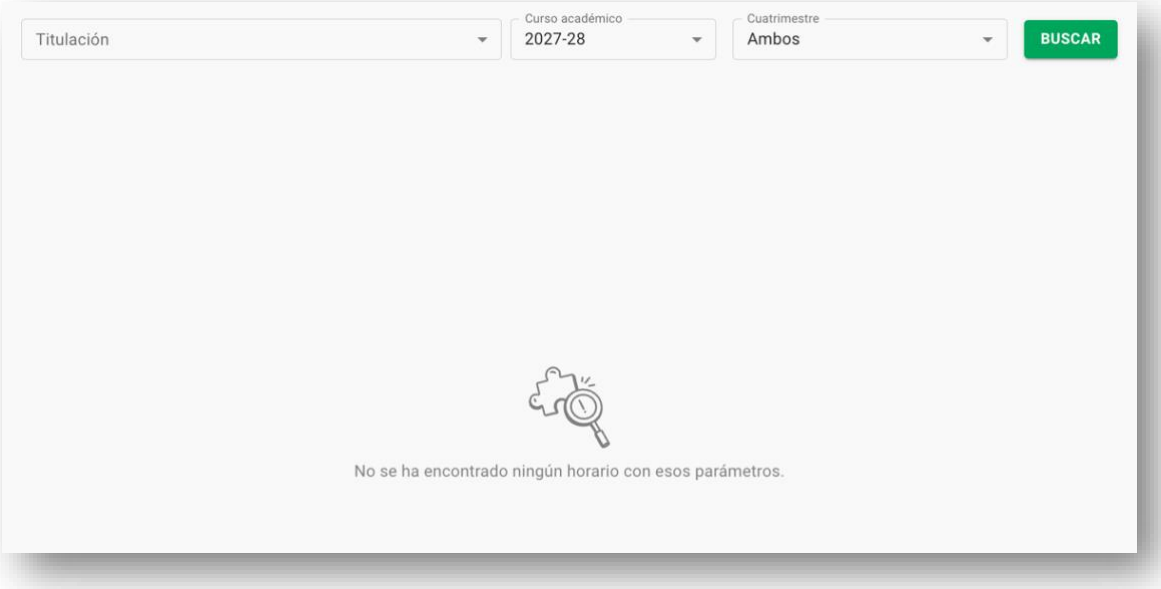


Ilustración 78: Parámetros no devuelven ningún documento horario

Con el botón flotante de la esquina inferior derecha se puede crear un nuevo documento horario.

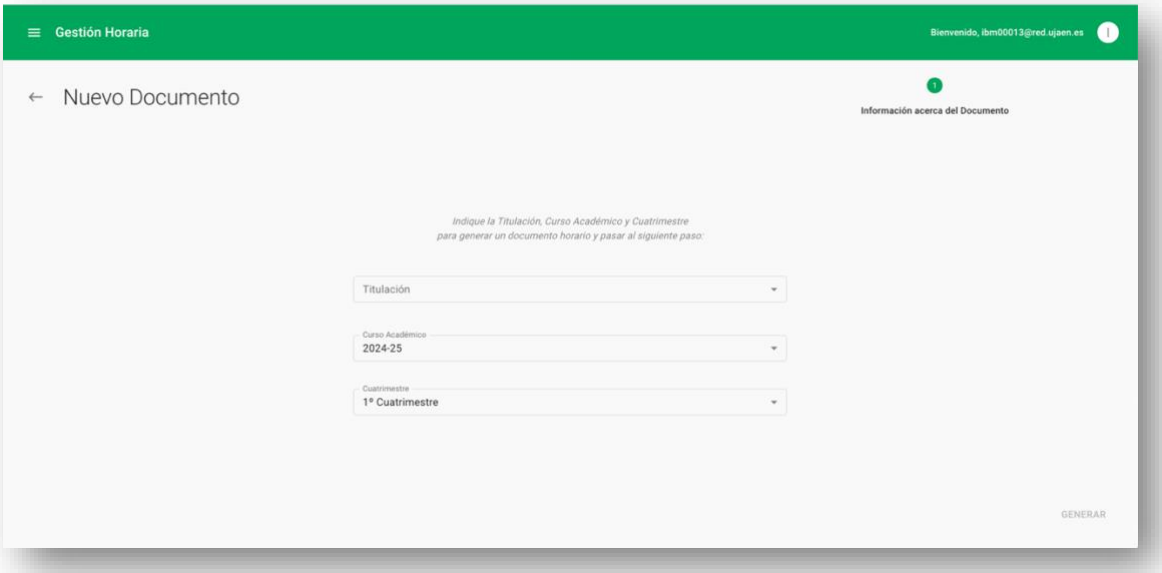


Ilustración 79: Vista para añadir un documento horario

Se habilitará una nueva pantalla para la creación de un nuevo documento, para esto es indispensable introducir tres datos, la titulación a la que pertenece el documento horario, el curso académico objeto del documento y el cuatrimestre en el que se encuentra. Una vez completos los tres selectores se habilitará el botón de *'Generar'* en la esquina inferior derecha para crear ese nuevo documento horario en blanco.

Si se realiza una nueva búsqueda, el documento horario figurará ahora como un documento horario creado.

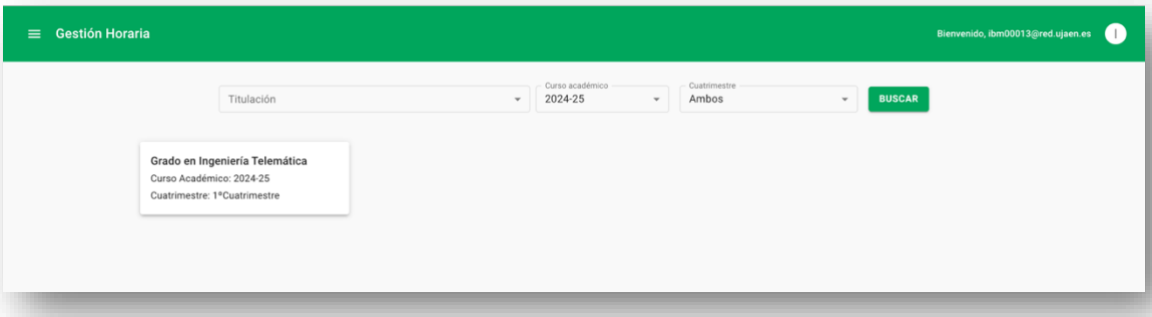


Ilustración 80: Búsqueda devuelve un documento horario

Si se accede a un documento horario se muestran todas las cuadrículas que lo conforman. Un documento horario está formado por las cuadrículas de los diferentes cursos y grupos de una titulación para un curso académico y un cuatrimestre.

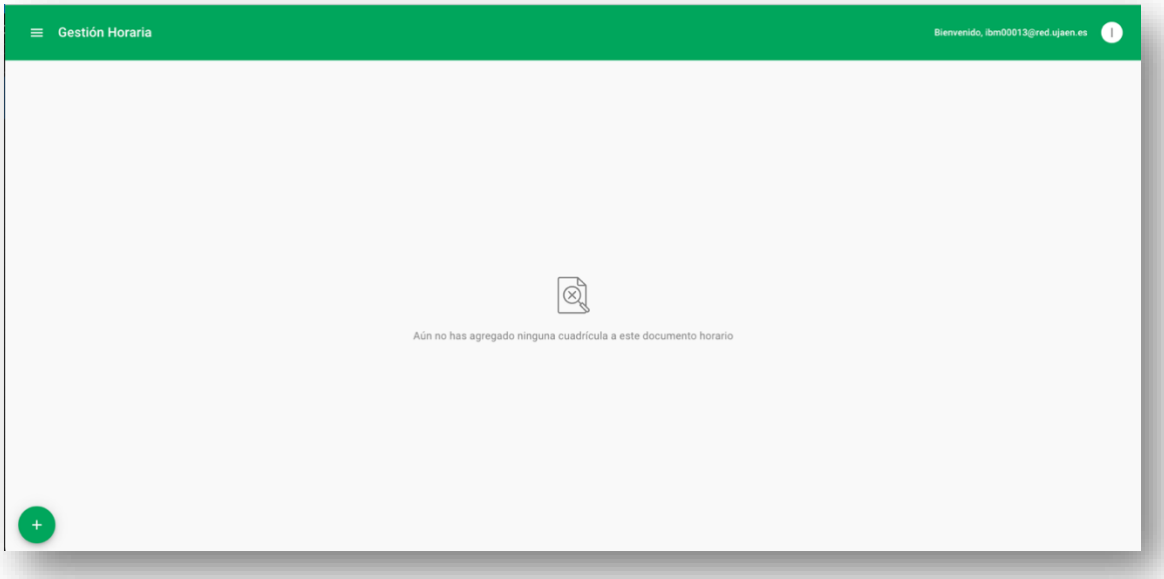


Ilustración 81: Documento horario sin cuadrículas añadidas

Si se accede al documento horario recién creado se verá que no tiene ninguna cuadrícula. Para agregar una nueva cuadrícula se debe acceder al botón flotante de la esquina inferior izquierda. Se desplegará un diálogo completo que pedirá curso, grupo, horas de inicio y fin e intervalo de la cuadrícula que se quiera crear.

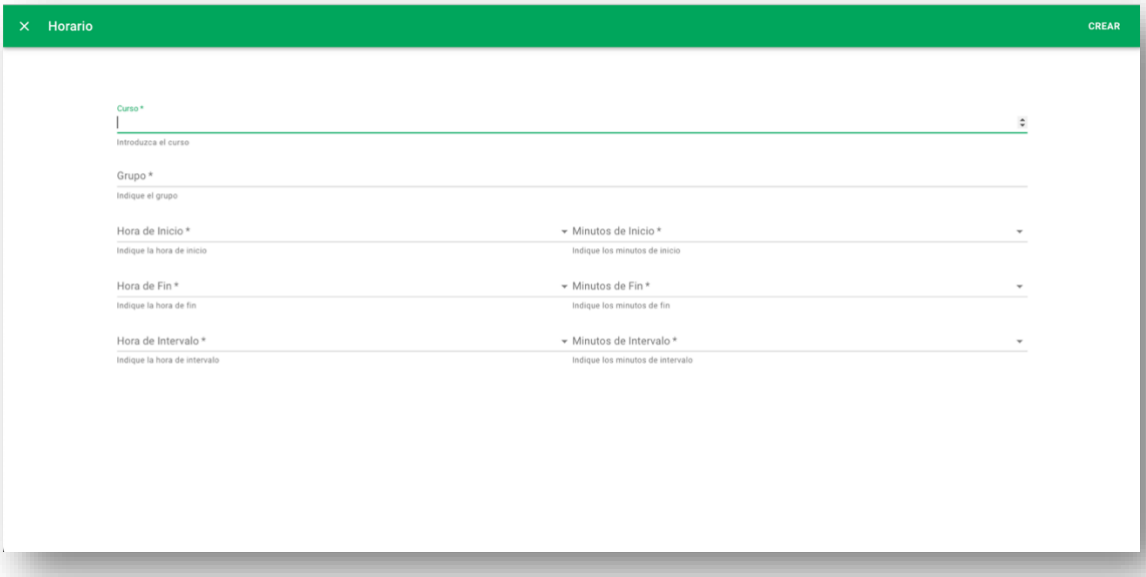


Ilustración 82: Diálogo para añadir una cuadrícula a un documento horario

Una vez se rellenen los campos, se debe pulsar sobre *'Crear'* en la esquina superior derecha para crear e incluir la cuadrícula al documento horario. Si los campos introducidos estuvieran duplicados o no fueran válidos el sistema alertaría. Si todo ha sido introducido correctamente se cerrará el diálogo y aparecerá la nueva cuadrícula como parte del documento horario.

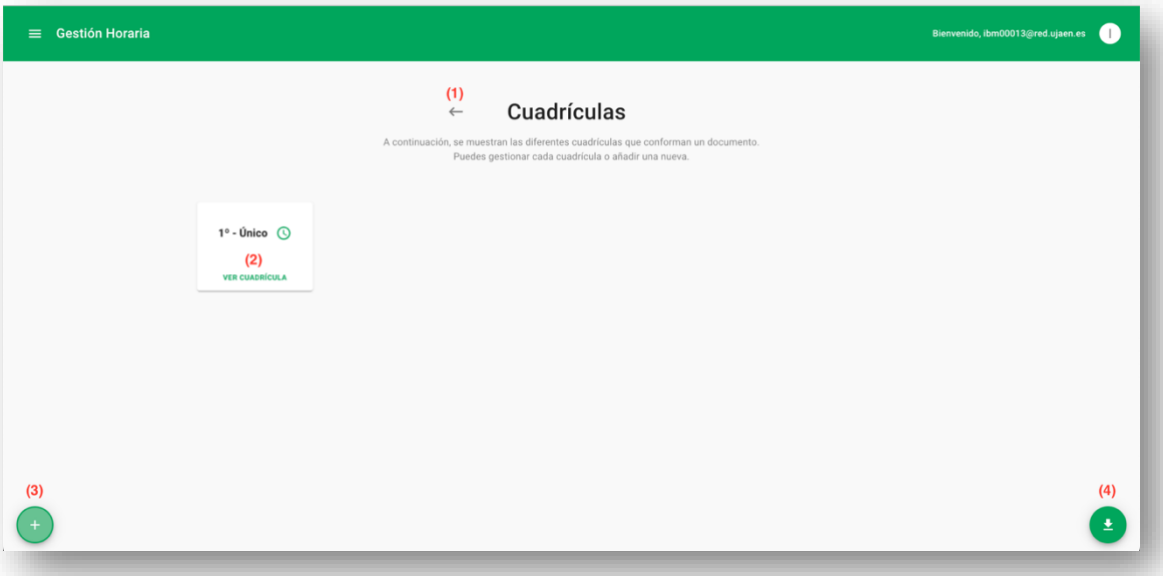


Ilustración 83: Vista de documento horario con cuadrículas añadidas

Las cuadrículas se irán apilando dentro de la vista de documento horario, que ahora tiene nuevas opciones:

1. Botón de retroceso a la vista de *Horarios*, para buscar o crear un nuevo documento horario.
2. Se puede acceder dentro de cada cuadrícula para modificar su contenido.
3. Se puede crear nuevas cuadrículas con el botón flotante.
4. Se puede descargar el documento horario en PDF, HTML o EXCEL. Como ya se observó en la pantalla de acceso público para la consulta de horarios.

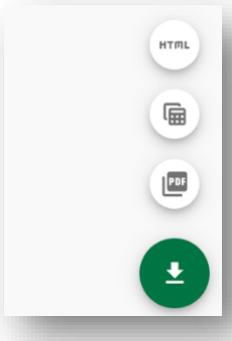


Ilustración 84: Botón flotante de descarga

Si se pulsa en *'Ver Cuadrícula'* se podrá acceder a la cuadrícula de un documento horario para su modificación. Esta vista se divide en varias funcionalidades:

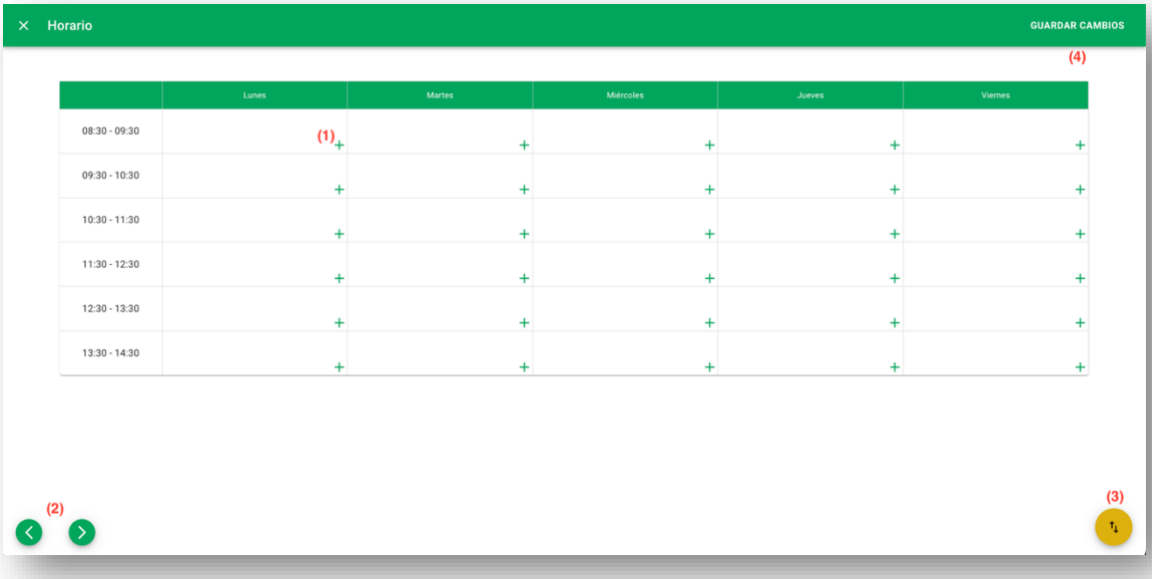


Ilustración 85: Cuadrícula horaria

1. Con el símbolo *'+'* de cada franja se desplegará un diálogo para añadir un grupo a esa franja. Una vez asignado aparecerá automáticamente en la franja, para quitarlo basta con acceder a la equis que aparece junto a cada grupo.

Ilustración 86: Resultado de asignación grupo a franja

2. Los dos botones flotantes con flechas permiten el desplazamiento lateral de la cuadrícula. Las columnas se desplazan a izquierdas o derechas.

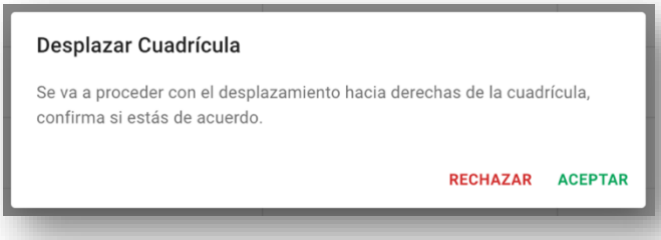


Ilustración 87: Diálogo para desplazar cuadrícula lateralmente

3. Con el botón flotante de la esquina inferior derecha se permite la importación de la cuadrícula con el mismo curso y grupo del curso académico anterior.

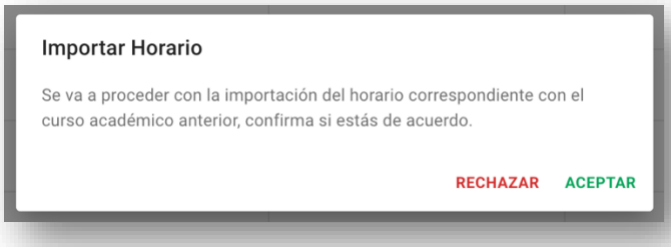


Ilustración 88: Diálogo para importar anterior curso

4. Cuando se deseen guardar los cambios hechos en la cuadrícula horaria se deberá pulsar sobre *'Guardar Cambios'*. La cuadrícula guardará el estado automáticamente.

Con los pasos detallados en este manual de usuario se conoce la usabilidad completa de la plataforma, permitiendo la consulta, creación y modificación de horarios, además de cualquier gestión de recursos que involucre la gestión horaria, tales como titulaciones, asignaturas o profesorado, entre otros.

Anexo II: Guía de implementación

Introducción

Para poder probar la plataforma se necesita implementarla de manera local en el sistema o desplegarla para su acceso público a través de Internet. El presente anexo detalla los pasos a seguir para la correcta puesta en marcha y posterior despliegue de la aplicación web independientemente del sistema en el que se quiera ejecutar.

Requisitos previos

Se necesitan cumplir una serie de requisitos previos para comenzar la implementación local. Tanto disponer de las tecnologías y herramientas necesarias, como tener acceso a los repositorios del proyecto es fundamental para comenzar con la implementación.

Entre las tecnologías y herramientas con las que debe contar el sistema para la ejecución del proyecto se encuentran las siguientes:

- Node.JS y npm
- Python y pip
- Git

El acceso a los repositorios se realiza a través de la plataforma GitHub. Existen dos repositorios, uno para el Front-End y otro para el Back-End.

- https://github.com/ibm00013/schedule_frontend
- https://github.com/ibm00013/schedule_backend

Preparación del Entorno de Desarrollo

Se deben seguir los siguientes pasos desde un terminal o consola, o desde un entorno de desarrollo que permite comandos por consola, como puede ser *Visual Studio Code*.

El primer paso es clonar los repositorios en la ubicación que se desee en el sistema. Para ello se debe ejecutar *git clone* <*repositorio*> en un directorio conocido del dispositivo.

Una vez se clonan ambos repositorios se procederá con la instalación de dependencias. Para el Front-End se debe localizar el directorio donde se ha clonado el repositorio del frontal con *cd frontend* y ejecutar el comando *npm install*. Automáticamente se construirá el proyecto Front-End con todas las dependencias incluidas en el archivo *package.json* correspondiente. Para el Back-End se debe localizar el directorio correcto de igual forma que se hizo para el Front-End con *cd backend*, y ejecutar el comando *pip install -r requirements.txt*. Este comando instalará las dependencias necesarias que se encuentran en el archivo *requirements.txt*.

Si se han seguido correctamente los pasos anteriores, se deberá tener correctamente configurado el entorno de desarrollo tanto del Front-End como del Back-End.

Configuración del Back-End

Se deben modificar una serie de archivos para adecuar la arquitectura del proyecto al nuevo dispositivo en el que se quiere probar. Para ello se debe abrir el archivo *settings.py*, el cual contiene la configuración principal del proyecto Django.

En este archivo se podrá modificar la base de datos, por defecto se usa SQLite en desarrollo y pruebas, por lo que no es necesario modificar nada para probar la aplicación web, pero se recomienda la modificación a otro sistema gestor más robusto en caso de que la aplicación web se despliegue en producción. Esto puede hacerse desde la variable con denominación *DATABASES*. Un ejemplo de configuración de base de datos en PostgreSQL sería la siguiente:

```
DATABASES = {
   "default": {
      "ENGINE": "django.db.backends.postgresql",
      "OPTIONS": {
         "service": "my_service",
         "passfile": ".my_pgpass",
      },
   }
}
```

La variable con denominación *ALLOWED_HOSTS* especifica los dominios que se permiten en el Back-End. En caso de probar la aplicación no es necesario modificar el valor ` *[]* ` por defecto. Si la aplicación se despliega en producción hay que incluir el dominio correspondiente del frontal para que acepte peticiones.

De igual modo, la variable con denominación *CORS_ALLOWED_ORIGINS* debe incluir el dominio tanto local como de producción para que no existan problemáticas *CORS* en las peticiones.

```
CORS_ALLOWED_ORIGINS = [
   'http://localhost:5173'
]
```

Una vez se configura el archivo *settings.py* se deben aplicar las migraciones pertinentes en la base de datos para que esté lista para su funcionamiento. Para ello habrá que ejecutar los comandos *python manage.py makemigrations* y posteriormente *python manage.py migrate*.

El siguiente paso será configurar un *superuser* para el panel de administración que ofrece *Django*. Para ello se debe ejecutar el comando *python manage.py createsuperuser*. La consola pedirá nombre de usuario, correo electrónico y contraseña. Una vez creado se podrá acceder al panel de administración y gestión de base de datos de *Django* correctamente.

Para ejecutar el servidor de desarrollo basta con ejecutar *python manage.py runserver* y se iniciará un servidor *Django* de pruebas para admitir las peticiones del Front-End.

Configuración del Front-End

El primer paso en la configuración del Front-End es modificar el dominio de la *API RESTful*, se deberá modificar el archivo *RouteConfig.jsx* almacenado en la carpeta *utils* del proyecto frontal. En este archivo se debe indicar el dominio del Back-End en la variable *apiUrl*.

```
const config = {
  apiUrl: "http://localhost:8000/",
};
```

Se puede construir el proyecto con *npm run build* para comprobar que no existen fallos o advertencias a la hora de querer desplegar el proyecto en producción.

Con la ejecución del comando *npm run dev* se despliega de forma local el frontal para probar la aplicación web.

Despliegue en producción

Para desplegar el Back-End a producción y permitir su acceso de forma global a través de Internet se necesita configurar un servidor web (como *Apache HTTP Server* o *Nginx*) y un servidor de aplicaciones (como *Gunicorn*). *El* servidor web recibirá las peticiones de los usuarios y las pasará a *Gunicorn*, encargado de ejecutar la aplicación *Django.*

Para desplegar el Front-End a producción se necesita configurar un servidor web (como *Apache HTTP Server* o *Nginx*) que almacene el proyecto que se ha construido anteriormente (*build*).

Se pueden aplicar métodos de Integración Continua y Despliegue Continua (*CI/CD*) como *Github Actions* aprovechando que los repositorios se encuentran en esta plataforma. Esto permite la automatización de la construcción y despliegue, gracias a la creación de flujos de trabajo (*workflows*) que se ejecutan de forma automática ante acciones como *pushes* al repositorio.